Chapter 9: Universal Synchronous Asynchronous Receiver Transceiver

The PIC18 Microcontroller

Han-Way Huang

Minnesota State University, Mankato
Why Serial Communication?

- **Parallel** I/O uses many signal pins
- Synchronization problem over longer distance
- Cost of cable
- Many applications do not need high data rate

Types of Serial I/O

1. USART (universal synchronous asynchronous receiver and transceiver)
2. SPI (synchronous peripheral interface)
3. I²C (inter-integrated circuit)
4. CAN bus (controller area network bus)
5. LIN (local interconnect network) – won’t be covered
The EIA232 Standard

- Developed in 1960 by electronic industry association.
- The latest revision is EIA232E.
- EIA232 is also referred to as RS232 for historical reason.
- Both computers and terminals are called **data terminal equipment** (DTE).
- Modems, bridges, and routers are called **data communication equipment** (DCE).
- There are four aspects to the EIA232:

  1. Electrical specifications
  2. Functional aspects—the function of each signal
  3. Mechanical specifications
  4. Procedural specifications
**Electrical Specifications**

- Data rates: EIA232 is applicable to data rate no more than 20 Kbps. The most commonly used data rates are 300, 1200, 2400, 9600, and 19200 baud.

- Signal state voltage assignments:
  1. voltages between -3 V to -25 V are considered as logic 1
  2. voltages between +3 V to 25 V are considered as logic 0

- Signal transfer distance: Signal should be able to transferred correctly up to 15 meters.
Functional Specifications

- Twenty-two signals are defined.
- Signals are divided into six groups:

  1. Signal ground and shield
  2. Primary communication channel
  3. Secondary communication channel
  4. Modem status and control signals
  5. Transmitter and receiver timing signals
  6. Channel test signals

- Secondary communication channel consists of the same signals as in the primary channel but rarely used.
- Timing signals are not used in asynchronous mode.
- Channel test signals are not used in normal communications.
Primary Communication Channel Signals

- Pin 2: transmit data
- Pin 3: receive data
- Pin 4: request to send signal (RTS)—asserted when in logic 0
- Pin 5: clear to send signal (CTS)—asserted when in logic 0

Modem Status and Control Signals

- Pin 6: data set ready (DSR)—asserted when in logic 0
- Pin 20: DTE ready (DTR)—asserted when in logic 0
- Pin 8: received line signal detector (CD)—also called carrier detect, asserted in logic 0
- Pin 22: ring indicator (RI)—asserted when in logic 0
- Pin 23: data signal rate selector—asserted when in logic 0
Mechanical Specification
- Specifies a 25-pin connector
- A 9-pin connector is used most often in PC but not specified in the standard
### Signal Direction and Signal Name

<table>
<thead>
<tr>
<th>Signal Direction</th>
<th>Signal Name</th>
<th>Signal Name</th>
<th>Signal Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>to DCE</td>
<td>Secondary transmitted data</td>
<td>14</td>
<td>Protective ground</td>
</tr>
<tr>
<td>to DTE</td>
<td>Transmit clock</td>
<td>15</td>
<td>Transmitted data</td>
</tr>
<tr>
<td>to DTE</td>
<td>Secondary received data</td>
<td>16</td>
<td>Received data</td>
</tr>
<tr>
<td>to DTE</td>
<td>Receiver clock</td>
<td>17</td>
<td>Request to send</td>
</tr>
<tr>
<td>to DTE</td>
<td>Unassigned</td>
<td>18</td>
<td>Clear to send</td>
</tr>
<tr>
<td>to DCE</td>
<td>Secondary request to send</td>
<td>19</td>
<td>Data set ready</td>
</tr>
<tr>
<td>to DCE</td>
<td>Data terminal ready</td>
<td>20</td>
<td>Signal ground</td>
</tr>
<tr>
<td>to DTE</td>
<td>Signal quality detect</td>
<td>21</td>
<td>Carrier detect</td>
</tr>
<tr>
<td>to DTE</td>
<td>Ring indicator</td>
<td>22</td>
<td>Reserved</td>
</tr>
<tr>
<td>Both</td>
<td>Data rate select</td>
<td>23</td>
<td>Reserved</td>
</tr>
<tr>
<td>to DCE</td>
<td>Transmit clock</td>
<td>24</td>
<td>Unassigned</td>
</tr>
<tr>
<td>to DCE</td>
<td>Unassigned</td>
<td>25</td>
<td>Secondary carrier detect</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Secondary clear to send</td>
</tr>
</tbody>
</table>

### Figure 9.1a EIA232E DB25 Connector and Pin Assignment

- Ring Indicator: 9
- Clear to Send: 8
- Request to send: 7
- DCE Ready: 6
- Received Line Signal Detect: 2
- Received Data: 2
- Transmitted Data: 3
- DTE Ready: 4
- Ground: 5

### Figure 9.1b EIA232E DB9 Connector and Signal Assignment

- Protective ground: 1
- Transmitted data: 2
- Received data: 3
- Request to send: 4
- Clear to send: 5
- Data set ready: 6
- Signal ground: 7
- Carrier detect: 8
- Reserved: 9
- Reserved: 10
- Unassigned: 11
- Secondary carrier detect: 12
- Secondary clear to send: 13
- Both: 14
- to DCE: 15
- to DTE: 16
- Both: 17

---

Copyright @ 2005 Thomson Delmar Learning

H. Huang Transparency No.9-8
EIA232 Procedural Specification

Case 1: Point-to-point asynchronous connection

![Diagram of point-to-point asynchronous connection]

Tx: transmit data  CTS: clear to send
Rx: receive data   RTS: request to send
CD: data carrier detect   DSR: data set ready

Figure 9.2 Point-to-point asynchronous connection
Sequence of events occurred during data transmission over dedicated link

Local

1. DCE asserts DSR

2. DTE asserts RTS

3. DCE asserts CTS

4. DTE starts to send data (to local DCE)

5. DCE sends out a carrier and then the modulated data

Remote

6. DCE asserts DCD

7. DTE waits for arrival of data

8. DCE sends out demodulated received data

9. DEC receives demodulated data
Case 2: two DTEs exchange data over public phone lines

- Two additional signals are needed: DTR and RI
- There are three phases in the data transmission:
  1. Establishing the connection
  2. Data transmission
  3. Disconnection

Figure 9.3 Asynchronous connection over public phone line
Connection establishment phase

1. DTE asserts DTR
2. DCE dials the phone number
3. DCE detects the ring and asserts RING
4. DTE asserts DTR to accept the call
5. DCE sends out a carrier and asserts DSR
6. DCE asserts DSR and DCD and also sends out a carrier for full duplex operation
7. DCE asserts DCD (full duplex operation)
Sequence of events occur during data transmission (continued)

<table>
<thead>
<tr>
<th>Local (transmission side)</th>
<th>Remote (receiving side)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data transmission phase</strong></td>
<td><strong>Disconnection phase</strong></td>
</tr>
<tr>
<td>1. DTE asserts RTS</td>
<td>1. DTE drops RTS</td>
</tr>
<tr>
<td>2. DCE asserts CTS</td>
<td>2. DCE drops CTS</td>
</tr>
<tr>
<td>3. DTE sends out data to DCE</td>
<td>3. DCE deasserts DCD &amp; DSR</td>
</tr>
<tr>
<td>4. DCE modulates data and sends it out</td>
<td>4. DTE deasserts DTR</td>
</tr>
<tr>
<td>5. DCE demodulates data and forwards the data to DTE</td>
<td>6. DTE receives data</td>
</tr>
</tbody>
</table>
Data Format

- Data is transmitted character by character.
- Each character is preceded by a start bit, followed by seven to nine data bits, and terminated by one to two stop bits.
- The receiver uses a clock signal with a frequency that is a multiple (usually 16) of the data rate to sample the incoming data in order to detect the arrival of start bit and determine the bit values.
- A majority circuit takes three samples (bit 7, 8, and 9) in each bit time to detect the arrival of the start bit and determine the logic value of each data bit.
- In older designs, a character can be terminated by one, one and a half, two stop bits.
- In new designs, only one stop bit is used.

Figure 9.4 The format of a character
Example 9.1 Sketch the output of the letter K when it is transmitted using the format of one start bit, eight data bits, and one stop bit.

Solution: Letters are represented in ASCII code. The ASCII code of letter K is $4B (= 01001011)$. The format of the output of letter K is shown in Figure 9.6.

![Output waveform diagrams](attachment:output_waveform.png)

Figure 9.6 Data format for letter K
**Data Transmission Errors**

1. Framing error
   - May occur due to clock synchronization problem
   - Can be detected by the missing stop bit
2. Receiver overrun
   - May occur when the CPU did not read the received data for a while
3. Parity errors
   - Occur due to odd number of bits change values
**Null Modem Connection**
- Used when two DTEs are located side by side and use the EIA232 interface to exchange data.
- Null modem connection connects signals in such a way to full two DTEs to think that they connected through a modem.
- In Figure 9.7, the signals on the same row of DTE1 and DTE2 are connected together.
- The cost of two modems are saved with Null modem connection.

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>DTE 1</th>
<th>DTE 2</th>
<th>Signal Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DB25 pin</td>
<td>DB9 pin</td>
<td>DB9 pin</td>
</tr>
<tr>
<td>FG (frame ground)</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TD (transmit data)</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>RD (receive data)</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>RTS (request to send)</td>
<td>4</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>CTS (clear to send)</td>
<td>5</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>SG (signal ground)</td>
<td>7</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>DSR (data set ready)</td>
<td>6</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>CD (carrier detect)</td>
<td>8</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>DTR (data terminal ready)</td>
<td>20</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>DTR (data terminal ready)</td>
<td>20</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

Figure 9.7 Null Modem connection
The PIC18 Serial Communication Interface

- A serial communication interface can be called a USART or UART.
- A USART supports both synchronous and asynchronous modes of operation.
- A PIC18 device supports either one or two identical USARTs.
- The USART port can be used to communicate with a CRT terminal, a personal computer, A/D converter, D/A converter, and serial EEPROMs.
- The USART can operate in three modes:
  
  1. Asynchronous mode (full duplex)
  2. Synchronous—master (half duplex)
  3. Synchronous—slave (half duplex)

**USART-Related Pins**

- RC6/TX1/CK1 and RC7/RX1/DT1 (USART1)
- RG1/TX2/CK2 and RG2/TX2/DT2 (USART2)

**USART-Related Registers**

- Transmit status register (TXSTA) - Transmit register (TXREG)
- Receive status register (RCSTA) - Receive register (RCREG)
- Baud rate generate register (SPBRG)
### The PIC18 Microcontroller

**Value after reset**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSRC</td>
<td>TX9</td>
<td>TXEN</td>
<td>SYNC</td>
<td>--</td>
<td>BRGH</td>
<td>TRMT</td>
<td>TX9D</td>
</tr>
</tbody>
</table>

- **CSRC**: Clock Source Select bit
  - Asynchronous mode: (don't care)
  - Synchronous mode:
    - 0 = Slave mode (clock from external source)
    - 1 = Master mode (clock generated internally from BRG)
- **TX9**: 9-bit Transmit Enable bit
  - 0 = selects 8-bit transmission
  - 1 = selects 9-bit transmission
- **TXEN**: Transmit Enable Bit
  - 0 = Transmit disabled
  - 1 = Transmit enabled
- **SYNC**: USART Mode Select Bit
  - 0 = Asynchronous mode
  - 1 = Synchronous mode
- **BRGH**: High Baud Rate Select Bit
  - Asynchronous mode:
    - 0 = low speed
    - 1 = high speed
  - Synchronous mode (unused)
- **TRMT**: Transmit Shift Register Status Bit
  - 0 = TSR full
  - 1 = TSR empty
- **TX9D**: 9th bit of transmit data
  - Can be Address/Data bit or a parity bit

---

**Figure 9.8 The TXSTA Register** (redraw with permission of Microchip)
The PIC18 Microcontroller

<table>
<thead>
<tr>
<th>Value after reset</th>
<th>SPEN</th>
<th>RX9</th>
<th>SREN</th>
<th>CREN</th>
<th>ADDEN</th>
<th>FERR</th>
<th>OERR</th>
<th>RX9D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

SPEN: Serial Port Enable bit
- 0 = Serial port disabled
- 1 = Serial port enabled

RX9: 9-bit Receive Enable bit
- 0 = Selects 8-bit reception
- 1 = Selects 9-bit reception

SREN: Single Receive Enable bit
- Asynchronous mode (don't care)
- Synchronous mode - Master (This bit is cleared after reception is complete):
  - 0 = Disables single receive
  - 1 = Enables single receive
- Synchronous mode - Slave: (don't care)

CREN: Continuous Receive Enable bit
- Asynchronous mode:
  - 0 = Disables receiver
  - 1 = Enables receiver
- Synchronous mode:
  - 0 = Disables continuous receive
  - 1 = Enables continuous receive (CREN overrides SREN)

ADDEN: Address Detect Enable bit
- Asynchronous mode 9-bit (RX9 = 1):
  - 0 = Disables address detection, all bytes are received, and 9th bit can be used as parity bit.
  - 1 = Enables address detection, enable interrupt and load of the receive buffer when RSR<8> is set
- Framing Error bit
  - 0 = no framing error
  - 1 = Framing error

OERR: Overrun Error bit
- 0 = No overrun error
- 1 = Overrun error (can be cleared by clearing bit CREN)

RX9D: 9th bit of Received Data
- This can be Address/Data bit or a parity bit, and must be calculated by firmware.

Figure 9.9 The RXSTA Register (redraw with permission of Microchip)
- The shift clock for the USART is generated by the **baud rate generator (SPBRG)**.
- In asynchronous mode, the BRGH bit of the TXSTA register also involves in the baud rate computation.
- The baud rate is computed using the formula shown in Table 9.2.

### Table 9.2 Formula for baud rate

<table>
<thead>
<tr>
<th>SYNC bit</th>
<th>BRGH = 0 (low speed)</th>
<th>BRGH = 1 (high speed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(Asynchronous) Baud rate = ( \frac{F_{OSC}}{64 (X+1)} )</td>
<td>Baud Rate = ( \frac{F_{OSC}}{16(X+1)} ) N/A</td>
</tr>
<tr>
<td>1</td>
<td>(Synchronous) Baud Rate = ( \frac{F_{OSC}}{4(X+1)} )</td>
<td></td>
</tr>
</tbody>
</table>

Note. X is the content of the SPBRG register

In asynchronous mode:
- When BRGH = 1, SPBRG = \( \left( \frac{F_{OSC}}{16 \times \text{baud rate}} \right) - 1 \)
- When BRGH = 0, SPBRG = \( \left( \frac{F_{OSC}}{64 \times \text{baud rate}} \right) - 1 \)

In synchronous mode:
- SPBRG = \( \left( \frac{F_{OSC}}{4 \times \text{baud rate}} \right) - 1 \)
**Example 9.2** Compute the value to be written into the SPBRG register to generate 9600 baud for asynchronous mode high-speed transmission assuming the frequency of the crystal oscillator is 20 MHz.

**Solution:** The value (for BRGH = 1) to be written into the SPBRG register is

\[
SPBRG = 20 \times 10^6 \div (16 \times 9600) - 1 = 130 - 1 = 129
\]

The actual baud rate is

\[
20,000,000 \div (16 \times 130) = 9615.4
\]

The resultant error rate is \((9615.4 - 9600) \div 9600 \times 100\% = 0.16\%\).

The same baud rate can also be achieved by using low speed (BRGH = 0) approach in which

\[
SPBRG = 20,000,000 \div (64 \times 9600) - 1 = 31
\]

The actual baud rate is

\[
20000000 \div (64 \times 32) = 9765.6
\]

The resultant error rate is \((9765.6 - 9600) \div 9600 \times 100\% = 1.7\%\).
**USART Asynchronous Mode**
- Data format is one start bit, eight or nine data bits, and one stop bit.
- The least significant bit is transmitted first.
- The transmitter and receiver are independent but use the same data format and baud rate.
- Asynchronous mode is stopped in sleep mode.
- The diagram of the transmitter block is shown in Figure 9.10.

**Figure 9.10 USART Transmitter block diagram (redraw with permission of Microchip)**

*Note.* $y = 1$ or $2$
Example 9.3 Write a subroutine to configure the USART1 transmitter to transmit data in asynchronous mode using 8-bit data format, disable interrupt, set baud rate to 9600. Assume the frequency of the crystal oscillator is 16 MHz.

Solution:
- Write the value of 0x24 into the TXSTA1 register
- Configure TX1 pin for output, RX1 pin for input:

```assembly
movlw 0x24
movwf TXSTA1
movlw D’103’ ; set baud rate to 9600
movwf SPBRG1 ; “
bsf TRISC,RC7 ; configure RX1 pin for input
bsf TRISC,RC6 ; configure TX1 pin for output
bcf PIE1, TXIE ; disable transmit interrupt
bsf RCSTA1, SPEN ; enable USART1
return
```

In C language,

```c
void usart1_open(void)
{
    TXSTA1 = 0x24;
    SPBRG1 = 103;
    TRISCbits.RC7 = 1; /* configure RX1 pin for input */
    TRISCbits.RC6 = 0; /* configure TX1 pin for output */
    PIE1bits.TXIE = 0; /* disable transmit interrupt */
    RCSTA1bits.SPEN = 1; /* enable USART port */
}
```
**Example 9.4** Write a subroutine to output the character in WREG to USART1 using the polling method.

**Solution:**

- Data can be sent to the transmitter only when the transmit register is empty.

```assembly
putc_usart1 btfss PIR1,TXIF,A ; wait until TXIF flag is set before output
bra putc_usart1
movwf TXREG1
return
```

In C language,

```c
void putc_usart1 (char xc);
{
    while (!PIR1bits.TX1IF);
    TXREG1 = xc;
}
```
Example 9.5 Write a subroutine to output a string (in program memory) pointed to by TBLPTR and terminated by a NULL character from USART1.

Solution:

; The following subroutine outputs the string pointed to by TBLPTR. It is called with fast save enabled.

; The following subroutine outputs the string pointed to by TBLPTR. It is called with fast save enabled.

; *******************************************************************
; *******************************************************************

puts_usart1

TBLRD*+

puts_usart1

TBLRD*+

movf TABLAT,W,A

movf TABLAT,W,A

bz done

bz done

call putc_usart1

call putc_usart1

bra puts_usart1

bra puts_usart1

done

done

return fast

return fast

In C language,

void puts_usart1 (unsigned rom char *cptr)
{
    while(*cptr)
        putc_usart1 (*cptr++);
}

Copyright @ 2005 Thomson Delmar Learning

H. Huang Transparency No.9-27
The PIC18 Microcontroller

Figure 9.11 USART receive block diagram (redraw with permission of Microchip)
Example 9.6 Write an instruction sequence to configure the USART1 to receive data in asynchronous mode using 8-bit data format, disable interrupt, set baud rate to 9600. Assume that the frequency of the crystal oscillator is 16 MHz.

Solution:

```
movlw 0x90 ; enable USART1 and receiver
movwf RCSTA1,A
movlw D'103'
movwf SPBRG1,A ; set up baud rate to 9600
bsf TRISC,RX1,A ; configure RX1 pin for input
bcf TRISC,TX1,A ; configure TX1 pin for input
```

In C language,

```
RCSTA1 = 0x90;
SPBRG  = 103;
TRISC |= 0x80;  /* configure RC7/RX1 pin for input */
TRISC &= 0xBF  /* configure RC6/TX1 pin for output */
```
Example 9.7 Write a subroutine to read a character from USART1 and return the character in WREG using the polling method. Ignore any errors.

Solution: A new character is received if the RCIF flag of the PIR1 register is set to 1.

```
getc_usart1 btfss PIR1,RCIF ; make sure a new character has been received
bra getc_usart1
movf RCREG1,W ; read the character
return FAST
```

In C language,

```
unsigned char getc_usart1 (void)
{
    while (!PIR1.bit.RCIF);
    return RCREG1;
}
```
**Example 9.8** Write a subroutine to read a string from the USART1 and store the string in a buffer pointed to by FSR0.

**Solution:**
The string from the USART port is terminated by a carriage return character.

```assembly
CR equ 0x0D
gets_usart1 call getc_usart1
movwf INDF0 ; save the character in buffer
sublw CR
bz done
clrf PREINC0,F ; move the pointer
goto gets_usart1

done clrf INDF0 ; terminate the string with a NULL character
return
```

Copyright @ 2005 Thomson Delmar Learning
In C language,

```c
#define CR 0x0D
void gets_usart1 (char *ptr)
{
    char xx;
    while (1)
    {
        xx = getc_usart1(); /* read a character */
        if (xx == CR) { /* is it a carriage return? */
            *ptr = '\0'; /* terminate the string with a NULL */
            return;
        }
        ptr++ = xx; /* store the received character in the buffer */
    }
}
```
Flow Control of USART in Asynchronous Mode

- In some circumstances, the software cannot read the received data and needs to inform the transmitter to stop.
- In some other situation, the transmitter may need to be told to suspend transmission because the receiver is too busy to read data.
- Both situations are handled by flow control.
- There are two flow control methods: hardware and XON/XOFF.
- XON and XOFF are two standard ASCII characters.
- The ASCII code for XON and XOFF are 0x11 and 0x13, respectively.
- Whenever a microcontroller cannot handle the incoming data, it sends the XOFF to the transmitter.
- When the microcontroller can handle incoming characters, it sends out XON character.
## C Library Functions for USART

**Table 9.3a Library functions for devices with only one USART**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusyUSART</td>
<td>Is the USART transmitting?</td>
</tr>
<tr>
<td>CloseUSART</td>
<td>Disable the USART</td>
</tr>
<tr>
<td>DataRdyUSART</td>
<td>Is data available in the USART read buffer?</td>
</tr>
<tr>
<td>getcUSART</td>
<td>Read a byte from USART</td>
</tr>
<tr>
<td>getsUSART</td>
<td>Read a string from USART</td>
</tr>
<tr>
<td>OpenUSART</td>
<td>Configure the USART</td>
</tr>
<tr>
<td>putcUSART</td>
<td>Write a byte to the USART</td>
</tr>
<tr>
<td>putsUSART</td>
<td>Write a string from data memory to the USART</td>
</tr>
<tr>
<td>putrsUSART</td>
<td>Write a string from program memory to the USART</td>
</tr>
<tr>
<td>ReadUSART</td>
<td>Read a byte from the USART</td>
</tr>
<tr>
<td>WriteUSART</td>
<td>Write a byte to the USART</td>
</tr>
</tbody>
</table>

**Table 9.3b Library functions for devices with multiple USARTs**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusyxUSART</td>
<td>Is the USART x transmitting?</td>
</tr>
<tr>
<td>ClosexUSART</td>
<td>Disable the USART x</td>
</tr>
<tr>
<td>DataRdyxUSART</td>
<td>Is data available in the USART x read buffer?</td>
</tr>
<tr>
<td>getcxUSART</td>
<td>Read a byte from USART x</td>
</tr>
<tr>
<td>getsxUSART</td>
<td>Read a string from USART x</td>
</tr>
<tr>
<td>OpenxUSART</td>
<td>Configure the USART x</td>
</tr>
<tr>
<td>putcxUSART</td>
<td>Write a byte to the USART x</td>
</tr>
<tr>
<td>putsxUSART</td>
<td>Write a string from data memory to the USART x</td>
</tr>
<tr>
<td>putrsxUSART</td>
<td>Write a string from program memory to the USART x</td>
</tr>
<tr>
<td>ReadxUSART</td>
<td>Read a byte from the USART x</td>
</tr>
<tr>
<td>WritexUSART</td>
<td>Write a byte to the USART x</td>
</tr>
</tbody>
</table>

**Note.** x = 1 or 2
The PIC18 Microcontroller

The following functions return a “1” if the transmitted is busy:

- char **BusyUSART** (void); -- used on devices with single USART
- char **Busy1USART** (void); -- used on devices with two USARTs
- char **Busy2USART** (void); -- used on devices with two USARTs

The following functions disable the transmitter and receiver:

- void **CloseUSART** (void); -- used on devices with single USART
- void **Close1USART** (void); -- used on devices with two USARTs
- void **Close2USART** (void); -- used on devices with two USARTs

The following functions return a “1” if the RCIF flag is set:

- char **DataRdyUSART** (void); -- used on devices with single USART
- char **DataRdy1USART** (void); -- used on devices with two USARTs
- char **DataRdy2USART** (void); -- used on devices with two USARTs
Any one of the following functions read a byte from the receive buffer including the 9th bit:

- `char getcUSART (void);` -- used on devices with single USART
- `char getc1USART (void);` -- used on devices with two USARTs
- `char getc2USART (void);` -- used on devices with two USARTs
- `char ReadUSART (void);` -- used on devices with single USART
- `char Read1USART (void);` -- used on devices with two USARTs
- `char Read2USART (void);` -- used on devices with two USARTs

The following functions read the specified number of bytes from the USART module and save the string in a buffer:

- `void getsUSART (char *buffer, unsigned len);` -- for devices with single USART
- `char gets1USART (char *buffer, unsigned len);` -- for devices with two USARTs
- `char gets2USART (char *buffer, unsigned len);` -- for devices with two USARTs
The status bit and the 9th data bit are saved in a union with the following declaration:

```c
union USART
{
    unsigned char val;
    struct
    {
        unsigned RX_NINE: 1;
        unsigned TX_NINE: 1;
        unsigned FRAME_ERROR: 1;
        unsigned OVERRUN_ERROR: 1;
        unsigned fill: 4;
    };
};
```
The PIC18 Microcontroller

The following functions write one character to the transmit buffer:

char **putc**USART (char data);  -- used on devices with single USART
char **putc**1USART (char data);  -- used on devices with two USARTs
char **putc**2USART (char data);  -- used on devices with two USARTs
char **Write**USART (char data);  -- used on devices with single USART
char **Write**1USART (char data);  -- used on devices with two USARTs
char **Write**2USART (char data);  -- used on devices with two USARTs

The following functions output a string in program memory to the USART module:

void **putrs**USART (const rom char *data);  -- used on devices with single USART
void **putrs**1USART (const rom char *data);  -- used on devices with two USARTs
void **putrs**2USART (const rom char *data);  -- used on devices with two USARTs

The following functions output a string in data memory to the USART module:

void **puts**USART (char *data);  -- used on devices with single USART
void **puts**1USART (char *data);  -- used on devices with two USARTs
void **puts**2USART (char *data);  -- used on devices with two USARTs
The following functions configured the specified USART module:

```c
void OpenUSART(unsigned char config,
               char spbrg);          -- used on devices with single USART

void Open1USART(unsigned char config,
               char spbrg);         -- used on devices with two USARTs

void Open2USART(unsigned char config,
               char spbrg);         -- used on devices with two USARTs
```

The `config` parameter is defined by ANDing the following parameters:

**Interrupt on Transmission:**
- `USART_TX_INT_ON` Transmit interrupt ON
- `USART_TX_INT_OFF` Transmit interrupt OFF

**Interrupt on Reception:**
- `USART_RX_INT_ON` Receive interrupt ON
- `USART_RX_INT_OFF` Receive interrupt OFF

**USART Mode:**
- `USART_ASYNCH_MODE` Asynchronous mode
- `USART_SYNCH_MODE` Synchronous mode
Transmission Width:

- **USART_EIGHT_BIT**: 8-bit transmit/receive
- **USART_NINE_BIT**: 9-bit transmit/receive

Slave/Master Select:

- **USART_SYNC_SLAVE**: Synchronous slave mode
- **USART_SYNC_MASTER**: Synchronous master mode

Reception mode:

- **USART_SINGLE_RX**: Single reception
- **USART_CONT_RX**: Continuous reception

Baud rate: (applied to asynchronous mode only)

- **USART_BRGH_HIGH**: High baud rate
- **USART_BRGH_LOW**: Low baud rate

The `spbrg` is the value to be written into the SPBRG register to set the baud rate.

An example,

```
Open1USART (USART_TX_INT_OFF & USART_RX_INT_OFF &
            USARTASYNCH_MODE & USART_EIGHT_BIT &
            USART_BRGH_HIGH, 103);
```
Interface Asynchronous Mode USART with EIA232

- The USART in asynchronous mode is mainly used with the EIA232 interface.
- The USART module uses 0 and 5V to represent logic 0 and 1 and hence cannot be connected to the EIA232 circuit directly.
- A circuit called EIA232 transceiver is needed to translate the voltage levels of the USART to and from the voltage levels of the EIA232 interface.
- EIA232 transceivers are available from many vendors.
- LT1080/1081 from Linear Technology, ST232 from SGS Thompson, the ICL232 from Intersil, the MAX232 from MAXIM, and the DS14C232 from National Semiconductor are examples of the EIA232 transceiver. These EIA232 transceivers operate with a 5-V power supply.
- The pin assignment of the MAX232 is shown in Figure 9.12.
- The circuit connection of the MAX232 with the PIC18 is shown in Figure 9.13.
Figure 9.12 Pin assignments and connections of the MAX232A
The PIC18 Microcontroller

Figure 9.13 Diagram of USART and EIA232 DB9 connector wiring in SSE452, SSE8680, and SSE8720 demo boards

Note: Both CTS and RTS are jumpered to an I/O pin in case hardware handshake is needed.
**USART Synchronous Master Mode**

- This mode is entered by setting bits SYNC, SPEN and CSRC.
- Data is transmitted in half-duplex mode.
- The clock signal is driven out from the CK pin.

**Synchronous Master Transmission**

- Setting the TXEN bit enables the USART module, which will start the SPBRG circuit to generate the shift clock.
- The actual data transmission does not start until a byte is written into the TXREG register.
- The data bits are shifted out on the rising edge of CK and becomes stable on the falling edge of the same clock period.

**Synchronous Master Reception**

- Reception is enabled by setting the SREN or CREN bit.
- Data is sampled on the falling edge of the CK clock.
- If only the SREN bit is set, only one character will be received.
- If the CREN bit is set, the reception will continue until the CREN bit is cleared.
Figure 9.14 USART transmission in synchronous master mode (redraw with permission of Microchip)
Figure 9.15 USART reception in synchronous master mode (SREN=1)  
(redraw with permission of Microchip)
**USART Synchronous Slave Mode**
- Shift clock is provided by external device and is input to the CK pin.
- Slave mode is entered by setting both the SYNC and SPEN bits and clearing the CSRC bit.

**Synchronous Slave Transmission**
- The shift clock signal is an input from the CK pin.
- CREN bit must be cleared.
- Transmission is enabled by setting the TXEN bit.
- Transmission is started by loading data into the TXREG register.

**Synchronous Slave Reception**
- Clock source is an input from the CK pin.
- When the RCIF flag is set, read the RCREG register to get the lower 8 bits.
- If 9-bit reception is enabled, then read the 9th bit from the RCSTA register.
- If there is any error, then clear the error by clearing the CREN bit.
Applications of USART Synchronous Mode

- Interfacing with shift registers to expand the number of I/O ports.
- Exchange data with other PIC18 microcontrollers

The 74LS165 Shift Register
- Has parallel inputs P7…P0 to be loaded into 74LS165 when $\overline{PL}$ is low.
- Data is shifted in from the DS pin when the $\overline{PL}$ input is high, one clock input is high, and the second clock input has a rising edge.
- Data is shifted out from Q7, which allows multiple 74LS165 to be concatenated.

![74LS165 pin assignment](image)

Figure 9.16 74LS165 pin assignment
Example 9.9 Show the circuit connection for using one 74LS165 to add an input port to the PIC18F8720 using the USART2 in synchronous master mode. Write a subroutine to configure the USART2 properly and a subroutine to shift one byte of data assuming that the PIC18F8720 is running with a 16-MHz crystal oscillator. Assume that the user uses eight switches to set the value to be input and uses INT1 interrupt to inform the arrival of data.

Solution:
1. Use the synchronous master mode to interface with the 74LS165.
2. Configure RG2/DT2 and RG1/CK2 for input and output, respectively.
3. Set baud rate to 1 Mbps
4. Enable INT1 interrupt
5. Circuit connection is shown in Figure 9.17.
Figure 9.17 Circuit connection between the PIC18F8720 and 74LS165
Procedure for inputting data from 74LS165

1. Set up a new value to P0 to P7 using the DIP switches.
2. Press the debounced switch to request an interrupt using the INT1 pin.
3. The INT1 interrupt service routine reads in the data from the RCREG2.

```assembly
#include <p18F8720.inc

rcv_buf set 0 ; buffer to hold the received byte
org 0x00
goto start
org 0x08
goto hi_ISR
org 0x18
retfie

start bcf TRISG,RG1,A ; configure CK2 pin for output
bsf TRISG,RG2,A ; configure DT2 pin for input
call open_usart2
call open_INT1
forever nop
bra forever
```
; The following function configures USART2 to operate in
; synchronous master mode with baud rate set to 10**6.
;******************************************************************************
open_usart2 movlw 0x03
          movwf SPBRG2,A ; set the shift rate to 1MHz
movlw 0x90 ; set synchronous master mode
movwf TXSTA2,A ;
                 ;
movlw 0x80 ; enable USART2 and disable
movwf RCSTA2,A ; reception
return
;******************************************************************************
; The following subroutine configures INT1 to high priority and
; then enable its interrupt.
;******************************************************************************
open_INT1 bsf RCON,IPEN,A ; enable priority interrupt
           movlw 0x48 ; set INT1 interrupt to high priority,
movwf INTCON3,A ; and then enable INT1 interrupt
movlw 0xC0 ; enable global interrupt
movwf INTCON ;
return
The high priority interrupt service routine makes sure that the interrupt is caused by usart2 INT1IF and then enable reception by setting SPEN bit. Wait until a byte is received.

```assembly
hi_ISR
  btfss INTCON3,INT1IF,A ; is interrupt caused by INT1?
  retfie ; interrupt is not caused by INT1
  bcf INTCON3,INT1IF,A ; clear INT1IF
  bcf PORTB,RB4,A ; load data into 74LS165
  nop ; "n
  bsf PORTB,RB4,A ; disable new data into 74LS165
  bcf PIR3,RC2IF,A ;
  bsf RCSTA2,SREN,A ; enable single reception from USART2

wait
  btfss PIR3,RC2IF,A ; wait for the arrival of a byte
  bra wait
  movff RCREG2,rcv_buf
  retfie
end
```
The 74LS164 Shift Register
- Has parallel outputs and two serial inputs
- Pin assignment and truth table are shown in Figure 9.18 and Table 9.5, respectively.

![74LS164 Diagram](image)

Table 9.5 Truth table of 74LS164

<table>
<thead>
<tr>
<th>Operating Mode</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MR</td>
<td>A</td>
</tr>
<tr>
<td>Reset (clear)</td>
<td>L</td>
<td>X</td>
</tr>
<tr>
<td>Shift</td>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td></td>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td></td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td>H</td>
<td>H</td>
</tr>
</tbody>
</table>

Copyright @ 2005 Thomson Delmar Learning

H. Huang Transparency No.9-54
Example 9.10 Show the circuit connection if you are to use one 74LS164 to add an output port to the PIC18F8720 using the USART2 module in synchronous master mode. Write a subroutine to shift out one byte of data assuming that the PIC18F8720 is running with a 16-MHz crystal oscillator. The user may use Q7..Q0 to drive LEDs.

Solution: The circuit connection is shown in Figure 9.19. Q7…Q0 should be used as the least significant bit to the most significant bit.
The following subroutine outputs the character in WREG to the USART2:

```
outc_usart2 bsf RCSTA2,SPEN ; enable USART2 port pins
poll2 btfss PIR3, TX2IF, A ; is TX2IF flag set?
bra poll2
movwf TXREG2, A
return
```

### Using the USART Synchronous Mode to Exchange Data with other PIC18 MCUs

![Diagram of two PIC18s exchanging data using USART synchronous mode]

**Figure 9.20** Two PIC18s exchange data using USART synchronous mode
Enhanced USART

- Newer PIC18 devices add some enhancements to the USART port:

  1. Automatic baud rate detection and calibration
  2. Sync break reception
  3. 12-bit break character transmit
  4. A baud rate control register (BAUDCONx, x = 1, or 2) is added for additional baud control

- These additional features can be used to support the LIN protocol