Using the UART in Microchip PIC18F Microcontrollers

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it

L.A.P. 1 Course
UART = **Universal Asynchronous Receiver/Transmitter**

- It is commonly referred as **serial port**
- It is a peripheral for point-to-point communication between two devices
- Communication occurs **in serial**, i.e. one bit at time
- Two communication PINs: **RX** and **TX**

![Diagram of UART communication between two devices](image-url)
UART transmission basics

- When no transmission, the line is set to **Logical “1”**
- Then the software triggers the transmission of a byte (e.g. “C”, hexcode **43**, binary **0100 0011**)
- First a **Logical “0”** is transmitted, called **start bit**
- Then the byte is transmitted **LSB first**
- An additional **parity bit** may follow (not in the example); it used for error checking
- One or two **stop bits (Logical “1”)** ends the transmission

![Data Byte Diagram](image)
The following parameters must be set in the UART hardware:

- **transmission speed**, in bps = Bit Per Second or baud
- number of bits per character, usually 8
- presence/absence of parity bit, usually absent
- number of stop bits, usually 1

A setting **19200,8,N,1** means:

- speed = 19200 bit-per-second;
- bits = 8;
- parity = None;
- stop bits = 1.
UART in the PIC18F25K22

From the datasheet:
- Two different UART
- The TX line of the UART1 is shared with RC6
- The RX line of the UART1 is shared with RC7
- The TX line of the UART2 is shared with RB6
- The RX line of the UART2 is shared with RB7

Basic registers per UARTx ("x" is the UART number):
- **TXSTAx**, configuration of the transmitter
- **RCSTAx**, configuration of the receiver
- **BAUDCONx**, configuration of the baud rate (speed) generator
- **SPBRGx**, baud rate (speed) generator value
The Baud Rate Generator

It is **divisor** driven from system clock frequency (FOSC).

It is controlled by the following bits/registers:

- **SPBRGHx**: the **divisor value** to be applied to FOSC to obtain the desired baud rate
- **BAUDCONxbits.BRG16**, a bit which indicates whether the divisor value uses 8 or 16 bits
- **TXSTAxbits.BRGH**, a bit which indicates if we are using a **high-speed** baud rate

A formula exists to compute the divisor value, but... Microchip provides very useful tables!
### TABLE 16-5: BAUD RATES FOR ASYNCHRONOUS MODES

<table>
<thead>
<tr>
<th>BAUD RATE</th>
<th>Actual Rate</th>
<th>% Error</th>
<th>SPBRGxG value (decimal)</th>
<th>Actual Rate</th>
<th>% Error</th>
<th>SPBRGxG value (decimal)</th>
<th>Actual Rate</th>
<th>% Error</th>
<th>SPBRGxG value (decimal)</th>
<th>Actual Rate</th>
<th>% Error</th>
<th>SPBRGxG value (decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1200</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1202</td>
<td>0.19</td>
<td>207</td>
<td>1200</td>
<td>0.00</td>
<td>143</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>2400</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>2404</td>
<td>0.16</td>
<td>103</td>
<td>2400</td>
<td>0.00</td>
<td>71</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>9600</td>
<td>9615</td>
<td>0.16</td>
<td>103</td>
<td>9615</td>
<td>0.16</td>
<td>25</td>
<td>9600</td>
<td>0.00</td>
<td>17</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>10417</td>
<td>10417</td>
<td>0.00</td>
<td>95</td>
<td>10417</td>
<td>0.00</td>
<td>23</td>
<td>10165</td>
<td>-2.42</td>
<td>16</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>19.2k</td>
<td>19.23k</td>
<td>0.16</td>
<td>51</td>
<td>19.23k</td>
<td>0.16</td>
<td>12</td>
<td>19.20k</td>
<td>0.00</td>
<td>8</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>57.6k</td>
<td>58.82k</td>
<td>2.12</td>
<td>16</td>
<td>57.60k</td>
<td>0.00</td>
<td>7</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>57.50k</td>
<td>0.00</td>
<td>2</td>
</tr>
<tr>
<td>115.2k</td>
<td>111.11k</td>
<td>-3.55</td>
<td>8</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

### TABLE 16-5: BAUD RATES FOR ASYNCHRONOUS MODES (CONTINUED)

<table>
<thead>
<tr>
<th>BAUD RATE</th>
<th>Actual Rate</th>
<th>% Error</th>
<th>SPBRGxG value (decimal)</th>
<th>Actual Rate</th>
<th>% Error</th>
<th>SPBRGxG value (decimal)</th>
<th>Actual Rate</th>
<th>% Error</th>
<th>SPBRGxG value (decimal)</th>
<th>Actual Rate</th>
<th>% Error</th>
<th>SPBRGxG value (decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>1200</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>2400</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>9600</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>10417</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>19.2k</td>
<td>19.23k</td>
<td>0.16</td>
<td>207</td>
<td>19.20k</td>
<td>0.00</td>
<td>59</td>
<td>19.23k</td>
<td>0.16</td>
<td>51</td>
<td>19.20k</td>
<td>0.00</td>
<td>35</td>
</tr>
<tr>
<td>57.6k</td>
<td>57.97k</td>
<td>0.64</td>
<td>68</td>
<td>57.60k</td>
<td>0.00</td>
<td>19</td>
<td>58.82k</td>
<td>2.12</td>
<td>16</td>
<td>57.60k</td>
<td>0.00</td>
<td>11</td>
</tr>
<tr>
<td>115.2k</td>
<td>114.29k</td>
<td>-0.79</td>
<td>34</td>
<td>115.2k</td>
<td>0.00</td>
<td>9</td>
<td>111.1k</td>
<td>-3.55</td>
<td>8</td>
<td>115.2k</td>
<td>0.00</td>
<td>5</td>
</tr>
</tbody>
</table>

Corrado Santoro  Using the UART in PIC18F Family
Setting the UART

UART Setup

```c
// setup UART
TRISCbits.TRISC6 = 0; // TX as output
TRISCbits.TRISC7 = 1; // RX as input

TXSTA1bits.SYNC = 0; // Async operation
TXSTA1bits.TX9 = 0;  // No tx of 9th bit
TXSTA1bits.TXEN = 1; // Enable transmitter

RCSTA1bits.RX9 = 0;  // No rx of 9th bit
RCSTA1bits.CREN = 1; // Enable receiver
RCSTA1bits.SPEN = 1; // Enable serial port

// Setting for 19200 BPS
BAUDCON1bits.BRG16 = 0; // Divisor at 8 bit
TXSTA1bits.BRGH = 0;  // No high-speed baudrate
SPBRG1 = 51;  // divisor value for 19200
```
Transmission is triggered by loading the byte into TXREGx;

The byte is loaded into the shift-register and the TXxIF is set;

The byte is “shifted-out” (transmitted) and, when transmission is completed TXSTAxbits.TRMT is set;

A new byte can be loaded into TXREG for a new transmission.
Transmitting a byte

Basic transmission

```c
// wait the end of transmission
while (TXSTA1bits.TRMT == 0) {};
TXREG1 = new data; // start sending the new byte
```

... or, you can use directly the `printf` function, provided that you implement a proper `putch(char c)` function, that is then called by printf for each character to send.

Using printf

```c
void putch(char c) {
    // wait the end of transmission
    while (TXSTA1bits.TRMT == 0) {};
    TXREG1 = c; // send the new byte
}
...
printf("Hello world!!");
...```
The UART Receiver

When a byte is received it is automatically loaded into \texttt{RCx\textregistered};

When reception is completed \texttt{RCxIF} is set, which can be tested (using polling or interrupt);

Reading \texttt{RCREGx} causes \texttt{RCxIF} to be automatically reset;

But some errors may occur and must be handled:

- **Overrun error**, \texttt{RCSTA\textregistered}bits.OERR==1, a new data is received but RCREGx has not previously read;

- **Framing error**, \texttt{RCSTA\textregistered}bits.FERR==1, the data is not properly “framed”, i.e. the stop bit(s) are not valid.
Reading a character

UART Char Reading

```c
char read_char(void) {
    while (PIR1bits.RC1IF == 0) { // wait for char
        if (RCSTA1bits.OERR == 1) {
            RCSTA1bits.OERR = 0; // clear overrun if it occurs
            RCSTA1bits.CREN = 0;
            RCSTA1bits.CREN = 1;
        }
    }
    return RCREG1;
}
```
Using the UART in Microchip PIC18F Microcontrollers

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it

L.A.P. 1 Course