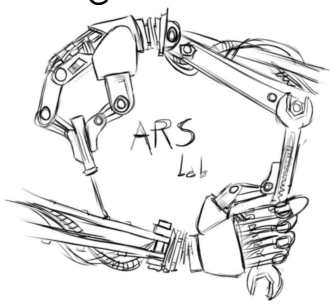
Using the UART in Microchip PIC18F Microcontrollers

Corrado Santoro

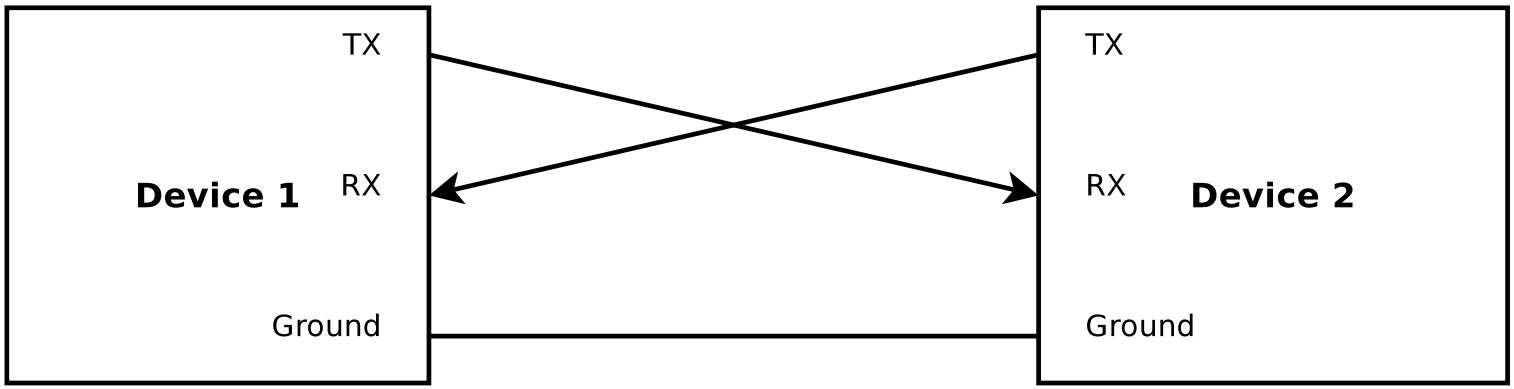
ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it



L.A.P. 1 Course

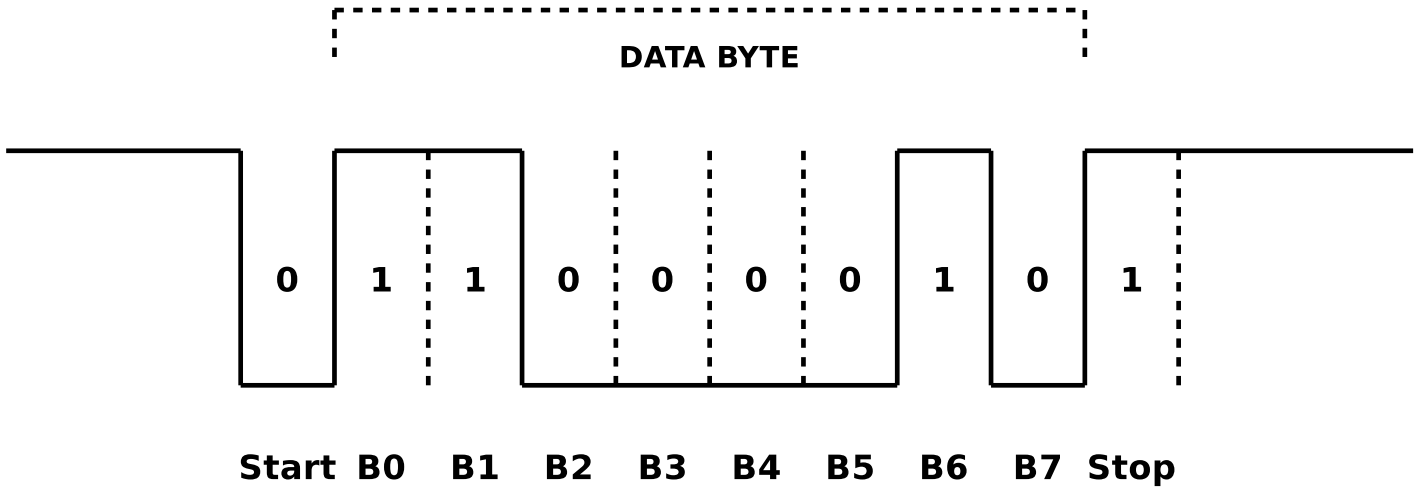
UART in MCUs

- UART = **Universal Asynchronous Receiver/Trasmitter**
- 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**



UART trasmission basics

- When no transmission, the line is set to **Logical “1”**
- Then the software triggers the trasmission 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



UART trasmission parameters

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 partity 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:SPBRGx**: 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 date

A formula exists to compute the divisor value, but... Microchip provides very usefull tables!

Setting the Baudrate generator

TABLE 16-5: BAUD RATES FOR ASYNCHRONOUS MODES

BAUD RATE	SYNC = 0, BRGH = 0, BRG16 = 0											
	Fosc = 64.000 MHz			Fosc = 18.432 MHz			Fosc = 16.000 MHz			Fosc = 11.0592 MHz		
	Actual Rate	% Error	SPBRxG value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)
300	—	—	—	—	—	—	—	—	—	—	—	—
1200	—	—	—	1200	0.00	239	1202	0.16	207	1200	0.00	143
2400	—	—	—	2400	0.00	119	2404	0.16	103	2400	0.00	71
9600	9615	0.16	103	9600	0.00	29	9615	0.16	25	9600	0.00	17
10417	10417	0.00	95	10286	-1.26	27	10417	0.00	23	10165	-2.42	16
19.2k	19.23k	0.16	51	19.20k	0.00	14	19.23k	0.16	12	19.20k	0.00	8
57.6k	58.82k	2.12	16	57.60k	0.00	7	—	—	—	57.60k	0.00	2
115.2k	111.11k	-3.55	8	—	—	—	—	—	—	—	—	—

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

BAUD RATE	SYNC = 0, BRGH = 1, BRG16 = 0											
	Fosc = 64.000 MHz			Fosc = 18.432 MHz			Fosc = 16.000 MHz			Fosc = 11.0592 MHz		
	Actual Rate	% Error	SPBRGx value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)
300	—	—	—	—	—	—	—	—	—	—	—	—
1200	—	—	—	—	—	—	—	—	—	—	—	—
2400	—	—	—	—	—	—	—	—	—	—	—	—
9600	—	—	—	9600	0.00	119	9615	0.16	103	9600	0.00	71
10417	—	—	—	10378	-0.37	110	10417	0.00	95	10473	0.53	65
19.2k	19.23k	0.16	207	19.20k	0.00	59	19.23k	0.16	51	19.20k	0.00	35
57.6k	57.97k	0.64	68	57.60k	0.00	19	58.82k	2.12	16	57.60k	0.00	11
115.2k	114.29k	-0.79	34	115.2k	0.00	9	111.1k	-3.55	8	115.2k	0.00	5

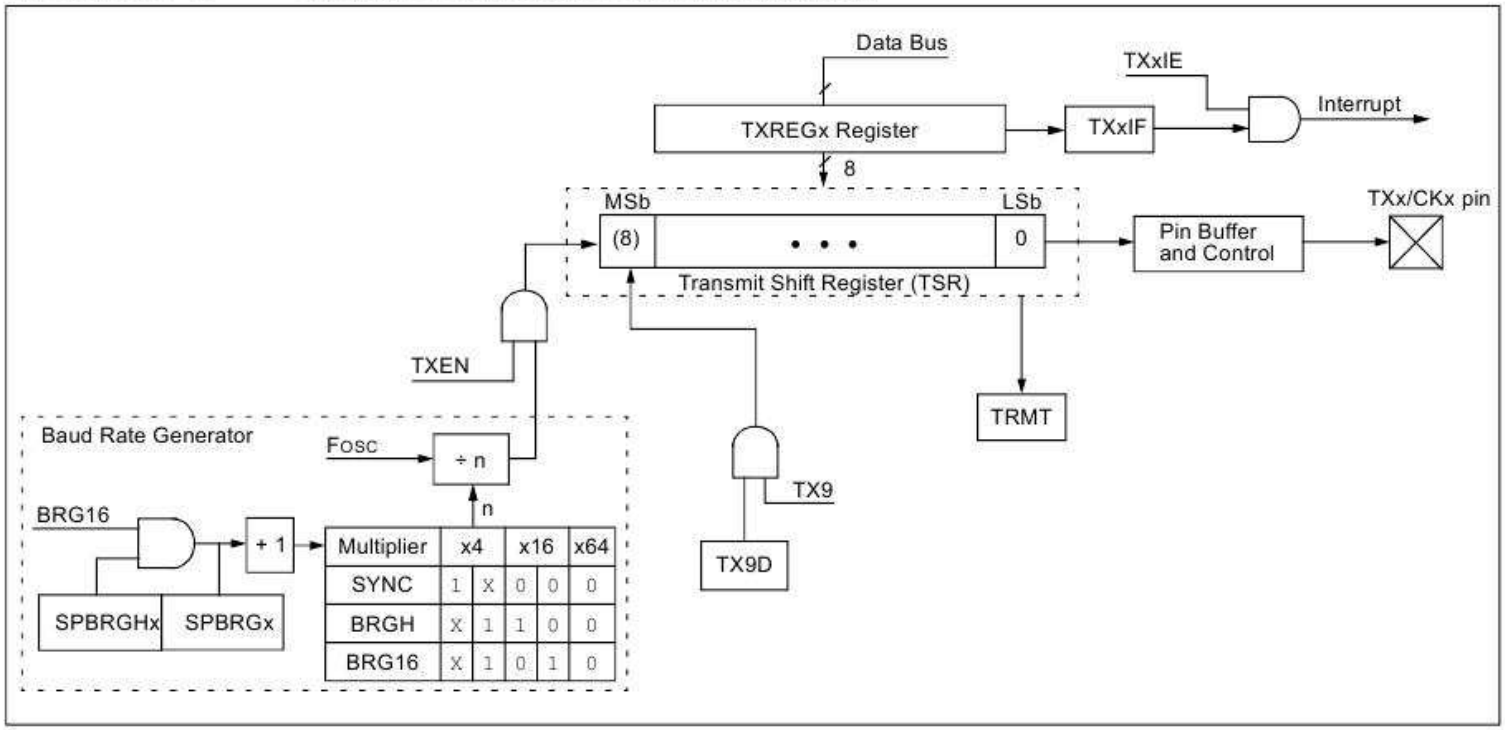
Setting the UART

UART Setup

```
...
// 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
```

The UART Transmitter

FIGURE 16-1: EUSART TRANSMIT BLOCK DIAGRAM



- 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

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

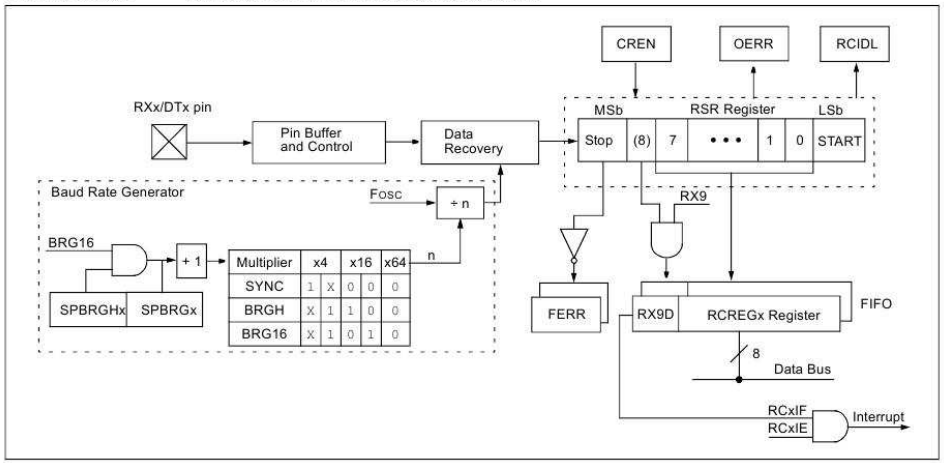
... or, you can use directly the **printf** function, provided that you implement a proper **putch(char c)** function, that is then called by printf for each character to send.

Using printf

```
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

FIGURE 16-2: EUSART RECEIVE BLOCK DIAGRAM



- When a byte is received it is automatically loaded into **RCREGx**;
- When reception is completed **RCxIF** is set, which can be tested (using polling or interrupt);
- Reading **RCREGx** causes **RCxIF** to be automatically reset;
- But some errors may occur and must be handled:
  - **Overrun error**, **RCSTAxbits.OERR==1**, a new data is received but RCREGx has not previously read;
  - **Framing error**, **RCSTAxbits.FERR==1**, the data is not properly “framed”, i.e. the stop bit(s) are not valid.

Reading a character

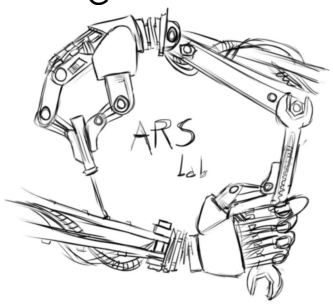
UART Char Reading

```
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