Davide Oldani (Galaxi93)

IMPARIAMO CON IL PIERIN - LA USART

12 July 2013

Premesse

In questo articolo verrà spiegato cos'è e come configurare correttamente la periferica USART presente sul PIC della scheda PIERIN PIC18. L'obiettivo di questo articolo è quello di stabilire una comunicazione seriale tra la scheda e un PC, inviando e ricevendo dati.

Cos'è l'USART?

L'USART (Universal Syncronous Asynchronous Receiver/Transmitter) è una periferica di trasmissioni dati seriale che utilizza solo due cavi per la comunicazione. E' una delle primissime interfaccedi comunicazione anche se ultimamente è stata declassata da altri tipi di comunicazioni seriali (USB). Esistono essenzialmente due modalità di trasmissione:

-Modalità asincrona (full-duplex)

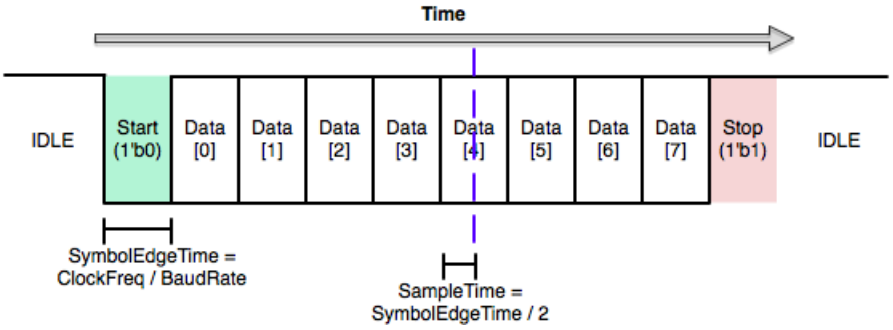
-Modalità sincrona (half-duplex)

La prima modalità (l'unica trattata in questo articolo), è una comunicazione asincrona, ovvero il trasmettitore e il ricevente non sono sincronizzati e posso inviare e ricevere dati in contemporanea(full-duplex). La velocità di trasmissione deve essere la medesima in entrambi i dispositivi ed è detta Baud Rate, espressa in bit al secondo. Per la comunicazione sono necessari due pin: uno trasmittente (TX) e uno ricevente (RX),

Nella modalità sincrona la trasmissione e la ricezione sono sincronizzate da un clock di sistema fornito da un master e uno o più slave che ricevono e inviano dati alla frequenza stabilita dal master. Dato che la linea dati è unica sia in ricezione che in trasmissione, la comunicazione è di tipo half-duplex: trasmissione e ricezione tra dispositivi non può avvenire contemporaneamente. Sono necessari due pin: uno per i dati (DT) e uno per il clock (CK).

La prima modalità è meno efficiente della prima, perché oltre ad inviare dati, deve inviare anche un bit di start e uno di stop. Ne deriva quindi che l'80% dei bit inviati rappresenta un'informazione utile al ricevente, al contrario di quella sincrona in cui ogni bit rappresenta un'informazione utile. Nonostante ciò, la modalità asincrona è quella che maggiormente si è diffusa.

Protocollo di trasmissione



Trasmissione

I dati transitano da e per la periferica USART utilizzando il seguente schema: un bit di start, 8 bit dati (1 byte) e uno o due bit di stop. Se si devono inviare più byte, la procedura si ripete (nuovo bit di start e così via). Solitamente viene inviato prima il bit meno significativo e lo stato di riposo (ovvero ciò che si trova tra due trasmissioni separate) si trova allo stato logico alto. Tutto ciò è comunque impostabile mediante i registri del PIC.

I registi di configurazione

Per la configurazione della periferica USART sono presenti 3 registri: TXSTAx , RCSTAx e BAUDCONx.

REGISTER 21-1: TXSTAx: TRANSMIT STATUS AND CONTROL REGISTER (1, ACCESS FADh; 2, FA8h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN(1)	SYNC	SENDB	BRGH	TRMT	TX9D
bit 7							bit 0

TXSTAx

Bit	Funzione
CSRC	Nella modalità asincrona non regola nulla mentre nella modalità sincrona regola se il dispositivo è master o slave
TX9	Permette di stabilire una trasmissione a 9 bit
TXEN	Abilita il trasmettitore

SYNC Scelta tra modalità sincrono o asincrona

SENDB Se settato invia un sync break che serve per il “risveglio” dalla modalità sleep.

BRGH Scelta tra alta o bassa velocità

TRMT Bit di sola lettura che indica se il registro TSR(quello di trasmissione) è pieno o meno

TX9D E’ il nono bit nella trasmissione a 9 bit

REGISTER 21-2: RCSTAx: RECEIVE STATUS AND CONTROL REGISTER (1, ACCESS FACh; 2, FC9h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

RCSTAx

Bit	Funzione
SPEN	Setta i pin RC6 e RC7 come porte seriali
RX9	Attiva la ricezione a 9bit
SREN	Ha utilità solo nella modalità sincrona master.
CREN	Attiva il modulo ricevente
ADDEN	Nella modalità a 9bit, serve per scegliere se quello ricevuto è un indirizzo o un byte con bit di parità
FERR	Bit di sola lettura che segnala un errore nella struttura del pacchetto ricevuto
OERR	Bit di sola lettura che segnala che un byte è arrivato e il registro RCxSTA non è ancora stato letto
RX9D	E’ il nono bit nella trasmissione a 9 bit

REGISTER 21-3: BAUDCONx: BAUD RATE CONTROL REGISTER (ACCESS F7Eh, F7Ch)

R/W-0	R-1	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
ABDOVF	RCIDL	RXDTP	TXCKP	BRG16	—	WUE	ABDEN
bit 7							bit 0

BAUDCONx

Bit	Funzione
ABDOVF	Attiva la modalità di auto rilevazione del baud rate.
RCIDL	Bit di sola lettura che indica se la linea dati RXè occupata o meno
RXDTP	Inverte lo stato logico della comunicazione(attivo-alto o attivo-basso)
TXCKO	Scelta tra stato logico alto o basso per lo stato di riposo
BRG16	Scelta tra baud rate generator a 8 o 16 bit

WUE	Attiva il “risveglio” del micro nel caso in cui ci sia un passaggio logico da alto a basso sulla linea RX
ABDEN	Attiva la rilevazione del baud rate dal prossimo dato

Poi ci sono i registri TXREGx, RCREGx, SPBRGx e SPBRGHx. Ilprimo serve per inviare un byte via seriale, il secondo invece serve per leggere un byte appena ricevuto. Gli ultimi due servono per stabilire il baud rate che viene calcolato secondo queste formule in base ai settaggi precedenti.

TABLE 21-1: BAUD RATE FORMULAS

Configuration Bits			BRG/EUSART Mode	Baud Rate Formula
SYNC	BRG16	BRGH		
0	0	0	8-bit/Asynchronous	Fosc/[64 (n + 1)]
0	0	1	8-bit/Asynchronous	Fosc/[16 (n + 1)]
0	1	0	16-bit/Asynchronous	
0	1	1	16-bit/Asynchronous	Fosc/[4 (n + 1)]
1	0	x	8-bit/Synchronous	
1	1	x	16-bit/Synchronous	

Legend: x = Don't care, n = value of SPBRGHx:SPBRGx register pair

Tabella baud rate

Stabiliamo la comunicazione

Iniziamo preparando l’ambiente di lavoro in MPLABX. Se avete problemi potete consultare questo articolo. Ad ogni modo, il mio programma è strutturato in 4 file: main.c, funzioni.c, header.h econfiguration_bits.c. Il primo contiene il main e le funzioni delle interrupt; funzioni.c contiene tutte le altre funzioni mentre l’header.h contiene tutte le definizioni e costanti del programma, le variabili globali e i prototipi di funzione. Comunque ognuno è libero di organizzare il proprio lavoro come è più comodo.

Per prima cosa dobbiamo configurare la periferica:

```
//Pin TX e RX del modulo UART
TRISC6=0;
TRISC7=1;

//Configurazione modulo UART
//Trasmissione asincrona a 8bit, alta velocità e generatore del baud-rate a 8bit
//Invio ericezione dati non invertita e idle state ad alto livello
TXSTA1=0b00100100;
```

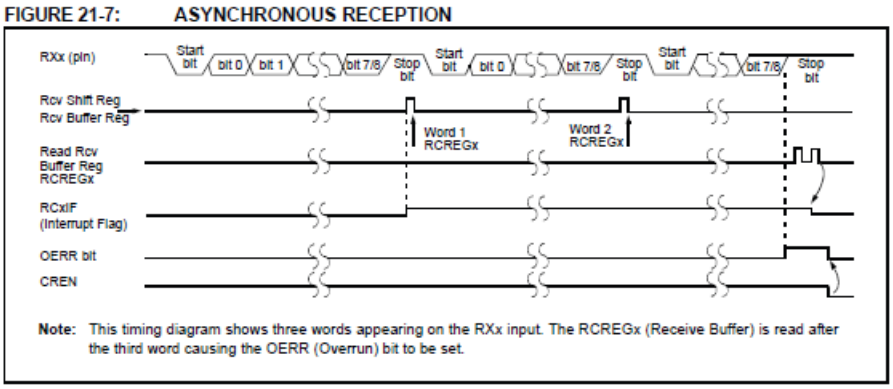
```
    RCSTA1=0b10010000;
    BAUDCON1=0;
    SPBRG1=155;          //Baud=19200(Fout=Fosc/(16*(SPBRG1+1))
```

Se avete letto con attenzione la spiegazione dei vari registri non avrete problemi a capire queste poche righe.

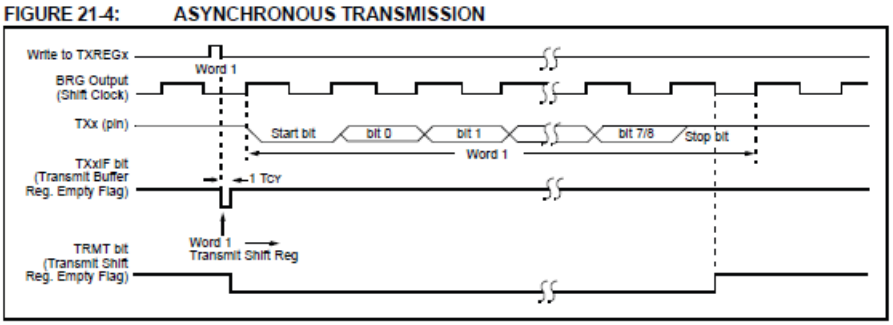
N.B: la maggior parte delle volte è impossibile trovare un valore per il registro del baud rate tale che la frequenza di trasmissione venga perfettamente il valore prefissato. Nel caso corrente, per esempio, ne risulta un baud rate di 19230, quindi l'errore è del 0,15%. (per il calcolo dell'errore si veda pag. 350 del datasheet). E' consigliato mantenere l'errore inferiore al 6% per evitare errori.

```
//Invia questo testo al PC solo all'inizio
    stampaTesto("Ciao Programmatore!!! ");
    stampaTesto("Invia un carattere e il pierin te lo rimandera'! \n");
    stampaTesto("\n");
    //Ciclo infinito
    while(1)
    {
        while(!RC1IF);
        RD6=1;              //Segnalo con il led LD1 che il PIC ha ricevuto un dato
        dato=RCREG1;        //Salvo il carattere nella variabile dato
        stampaTesto("Il Pierin ha ricevuto: ");     //Invio al PC una stringa
        while(!TX1IF);      //Controllo e attendo che il PIC abbia finito di trasmetter
        TXREG1=dato;        //e invio il carattere appena ricevuto
        stampaTesto("\n");  //Vado a capo
        RD6=0;              //Spengo il led
    }
```

Questo codice contenuto nel main, si occupa di ricevere ed inviare i byte al computer. Le prime istruzioni inviano una stringa di testo utilizzando una funzione che vedremo in seguito. Dentro al ciclo while, il micro attende che un byte venga inviato. Infatti come è possibile vedere nell'immagine qua sotto, quando un byte viene ricevuto, il bit RC1IF del registro PIR1 viene settato. Per resettare il bit basta leggere il registro RCREG1, cosa che nel listato avviene e il valore che contiene viene salvato in una variabile.



Ricezione



Trasmissione

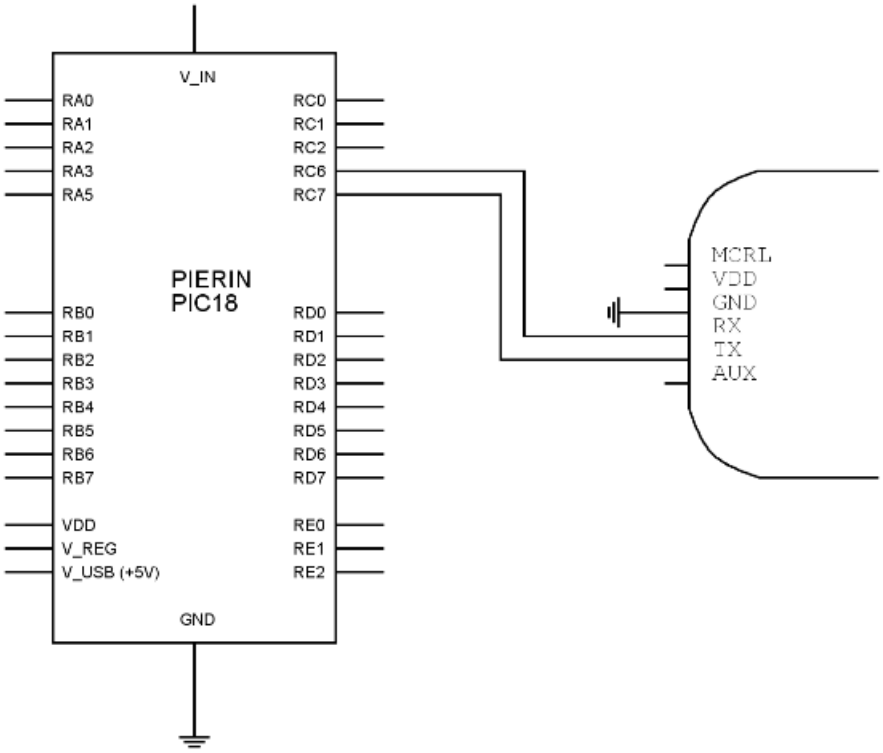
Poi controlla se il registro di trasmissione è pieno o meno,e se è vuoto procede con l'invio del byte via seriale. Tutto questo procedimento viene segnalato da il led LD1 che si accende al momento della ricezione del byte e si spegne a ritrasmissione compiuta.

```
//Dato in ingresso una stringa, invia un carattere per volta alla seriale.
void stampaTesto(char *t)
{
    while (*t)
    {
        while(!TX1IF);
        TXREG1=*t;
        t++;
    }
}
```

Questa funzione non fa altro che inviare un carattere per volta fino a che arriva alla fine della stringa.

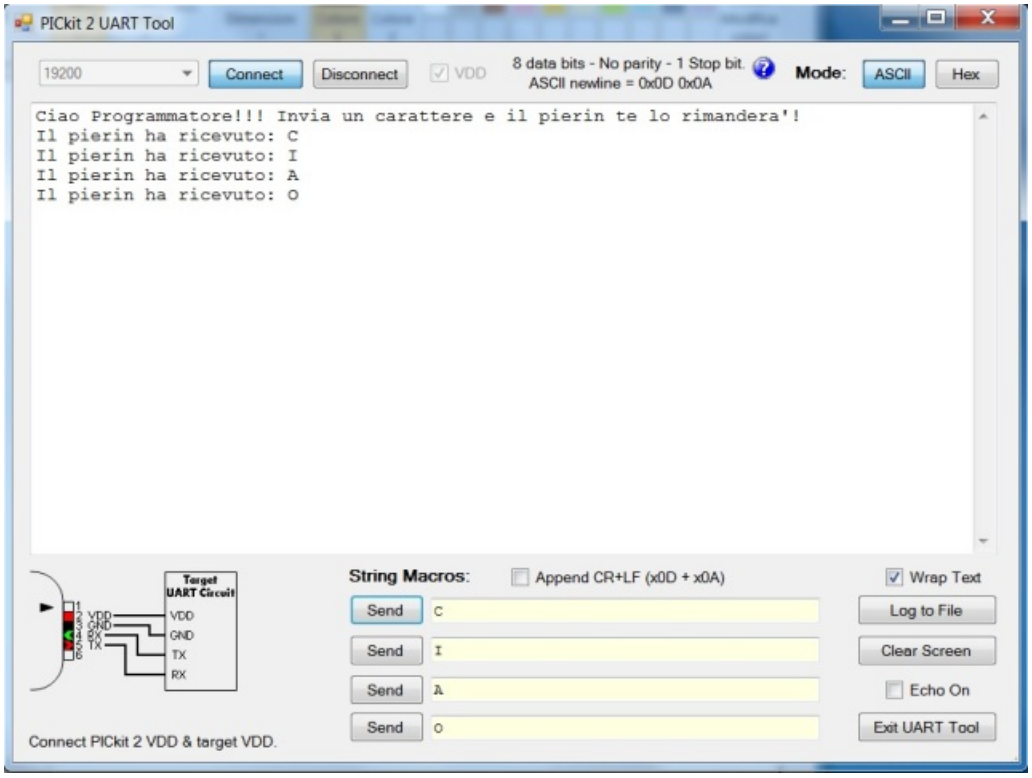
Collegamenti

Per simulare una porta seriale che i computer moderni non hanno più, si deve usare un qualche altro dispositivo esterno. Io ho usato ilPickit 2 che ha anche la funzione di porta seriale. Ecco lo schema di collegamento:



Schema collegamento

Per stabilire la comunicazione, dovete avviare il pickit2,nel menù Tool selezionare UART tool. In seguito settate il baud rate a 19200 es puntate la casella VDD. Premete su connect e collegate al PIERIN PIC18 il cavoUSB per dargli alimentazione. Vi dovrebbe comparire una scritta iniziale e poi ad ogni carattere che gli inviate, la scheda dovrebbe rispondervi come in figura:



Uart tool

Migliorie al codice

In un programma reale, è improbabile che il PIC svolga solo il compito di gestire i dati della seriale. Probabilmente dovrà gestire qualcos'altro,tipo un lcd, un motore un altro dispositivo seriale ecc. Utilizzando il codice precedente, si "paralizza" in un certo senso il micro, dato che rimane inattesa di un dato in arrivo, e se nel frattempo si deve svolgere qualcos'altro(che non sia gestito via interrupt) si deve aspettare. E' quindi una buona cosa gestire il flusso dati della UART mediante gli interrupt.

Il modulo USART del micro è in grado di scatenare un interrupt quando un dato è stato ricevuto. Per attivarlo si deve agire sul bit RC1iE del registro PIE1 e settare la priorità con il bit RC1IP del registro IPR1.

Le nuove configurazioni della periferica diventano:

```
//Pin TX e RX del modulo UART
    TRISC6=0;
    TRISC7=1;
```

```
    //Configurazione modulo UART
    //Trasmissione asincrona a 8bit, alta velocità e generatore del baud-rate a 8bit
    //Invio e ricezione dati non invertita e idle state ad alto livello
    TXSTA1=0b00100100;
    RCSTA1=0b10010000;
    BAUDCON1=0;
    SPBRG1=155;         //Baud=19200 (Fout=Fosc/(16*(SPBRG1+1))

    //Interrupt UART
    RC1IE=1;            //Attivo l'interrupt in ricezione
    RC1IP=1;            //Alta priorità

    //Abilitazione interrupt generale
    GIE=1;
    PEIE=1;
```

Così facendo, ogni volta che viene ricevuto un dato, si scatena un interrupt che pone al livello logico 1 il bit RC1IF. La gestione dell'interrupt sarà quindi la seguente:

```
//Alta Priorità
void interrupt high_isr(void)
{
    //Se il registro RCREG1 è pieno, si scatena un interrupt
    if (RC1IF)
    {
        RD6=1;              //Segnalo con il led LD1 che il PIC ha ricevuto un dato
        dato=RCREG1;        //Salvo il carattere nella variabile dato
        stampaTesto("Il Pierin ha ricevuto: ");     //Invio al PC una stringa
        while(!TX1IF);      //Controllo e attendo che il PIC abbia finito di trasmetter
        TXREG1=dato;        //e invio il carattere appena ricevuto
        stampaTesto("\n");  //Vado a capo
        RD6=0;              //Spengo il led
    }
}
```

Molto simile al codice precedente, solo che ora non è presente il while che imponeva l'attesa di un dato in ingresso. Ora il micro può svolgere qualsiasi altra cosa e se arriva un byte dalla seriale, interrompe tutto e lo legge, per poi riprendere da dove si era interrotto.

Conclusioni

Qui potete scaricare il primo programma.

Qui potete scaricare il secondo.

Se avete domande o ho commesso qualche errore, non esitate a commentare l'articolo. Buona sperimentazione!

Estratto da "http://www.electroyou.it/mediawiki/index.php?title=UsersPages:Galaxi93:impariamo-con-il-pierin-la-usart-parte-1"