LaurTec

ATmega 2560

Corso di programmazione per microcontrollori Atmel

Autore : *Marcello Pinna*

ID: UT0005-IT

Copyright © 2020 Mauro Laurenti

INFORMATIVA

Come prescritto dall'art. 1, comma 1, della legge 21 maggio 2004 n.128, l'autore avvisa di aver assolto, per la seguente opera dell'ingegno, a tutti gli obblighi della legge 22 Aprile del 1941 n. 633, sulla tutela del diritto d'autore.
Tutti i diritti di questa opera sono riservati. Ogni riproduzione ed ogni altra forma di diffusione al pubblico dell'opera, o parte di essa, senza un'autorizzazione scritta dell'autore, rappresenta una violazione della legge che tutela il diritto d'autore, in particolare non ne è consentito un utilizzo per trarne profitto.
La mancata osservanza della legge 22 Aprile del 1941 n. 633 è perseguibile con la reclusione o sanzione pecuniaria, come descritto al Titolo III, Capo III, Sezione II.
A norma dell'art. 70 è comunque consentito, per scopi di critica o discussione, il riassunto e la citazione, accompagnati dalla menzione del titolo dell'opera e dal nome dell'autore.

AVVERTENZE

I progetti presentati non hanno la marcatura CE, quindi non possono essere utilizzati per scopi commerciali nella Comunità Economica Europea.

Chiunque decida di far uso delle nozioni riportate nella seguente opera o decida di realizzare i circuiti proposti, è tenuto pertanto a prestare la massima attenzione in osservanza alle normative in vigore sulla sicurezza.
L'autore declina ogni responsabilità per eventuali danni causati a persone, animali o cose derivante dall'utilizzo diretto o indiretto del materiale, dei dispositivi o del software presentati nella seguente opera.
Si fa inoltre presente che quanto riportato viene fornito così com'è, a solo scopo didattico e formativo, senza garanzia alcuna della sua correttezza.

L'autore ringrazia anticipatamente per la segnalazione di ogni errore.

Tutti i marchi citati in quest'opera sono dei rispettivi proprietari.

Indice

Capitolo XI....4
Il modulo USART....4
Generatore di clock....5
Formato del frame....6
Impostazione dei parametri di comunicazione....7
Flag di interrupt in trasmissione....10
Registri di configurazione....11
Programmi di esempio....14
History....20

Capitolo XI

Il modulo USART

Il modulo USART (*Universal Synchronous and Asynchronous Receiver-Transmitter*) è una periferica che supporta una comunicazione seriale e può essere programmata per comunicare con altri dispositivi in modo sia sincrono che asincrono. A bordo dell'ATmega 2560 sono presenti ben quattro USART uguali tra loro, chiamate USART0, USART1, USART2 e USART3. Le caratteristiche principali sono:

- Supporto modalità full duplex.
- Supporto modalità sincrona o asincrona.
- Generazione del baud rate ad alta risoluzione.
- Supporto di frame seriali con 5, 6, 7, 8, 9 data bit e 1 o 2 stop bit.
- Generazione del bit di parità.
- Rilevamento di errori nel frame.

In Figura 1 è riportato il relativo diagramma di blocchi.

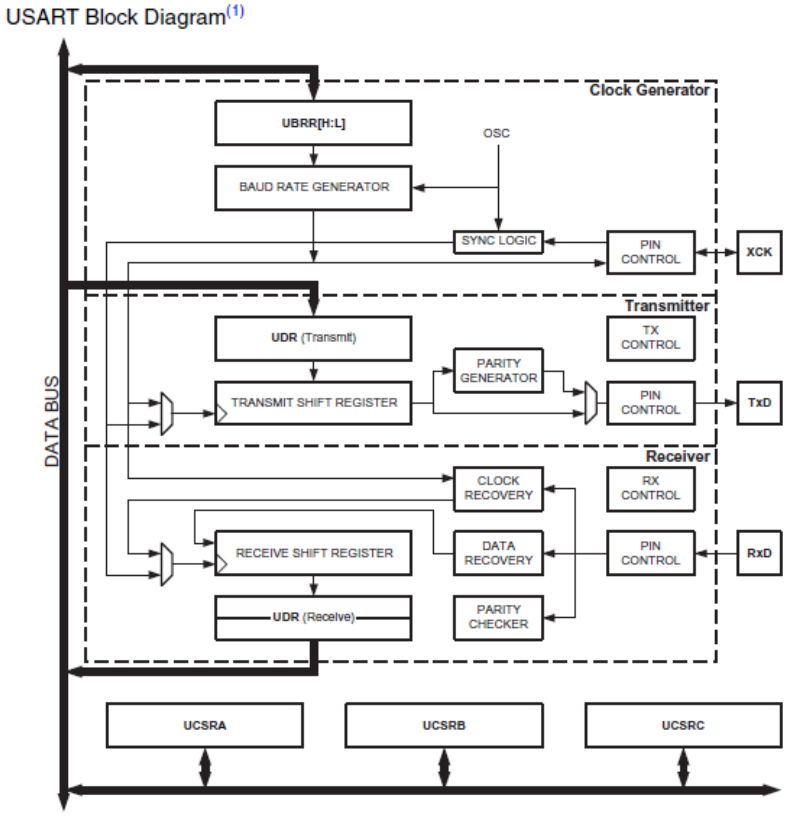
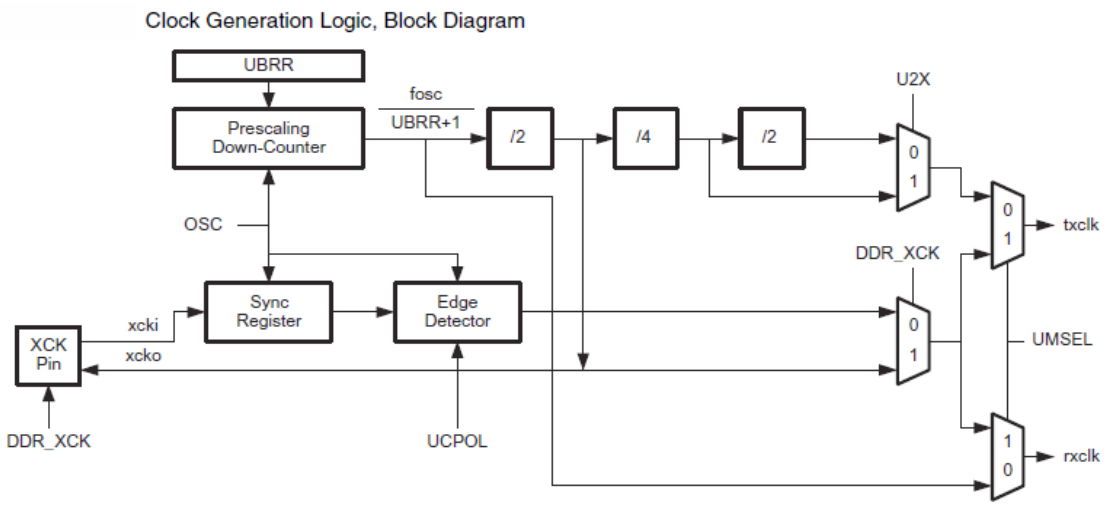


Figura 1: *Diagramma a blocchi del modulo USART.*

Come si può notare, la periferica è divisa in tre unità principali che sono il generatore di clock, il trasmettitore ed il ricevitore, mentre i registri di controllo sono condivisi da tutte le unità. La logica di generazione del clock è composta da una logica di sincronizzazione, usata per sorgenti di clock esterne in operazioni sincrone con lo slave e dal generatore di *baud rate*. Il pin XKCn è usato solo in modo sincrono. Il trasmettitore è composto da un singolo buffer in scrittura, che permette un trasferimento continuo dei dati provenienti dal *data register* UDR, uno *shift register* seriale, il generatore di *parity* che calcola il bit di parità e da una logica di controllo che elabora i diversi formati del frame. Il ricevitore è composto dal *clock recovery*, dal *data recovery*, usato per la ricezione dati in operazioni asincrone, dal *parity checker* che si occupa di verificare il bit di parità e da uno *sfift register* che riceve i dati e li memorizza nel *data register* UDR

Generatore di clock

Il generatore di clock crea la base del tempo per la trasmissione e ricezione dei dati. Ogni USART dell'ATmega supporta quattro modi per gestire il segnale di clock: velocità normale asincrona, velocità doppia asincrona, modo sincrono sul master, modo sincrono sullo slave. In Figura 2 è riportato il diagramma di blocchi della logica del modulo per la generazione del clock.



Signal description:

txclk	Transmitter clock (Internal Signal).
rxclk	Receiver base clock (Internal Signal).
xcki	Input from XCK pin (internal Signal). Used for synchronous slave operation.
xcko	Clock output to XCK pin (Internal Signal). Used for synchronous master operation.
f_{osc}	XTAL pin frequency (System Clock).

Figura 2: *Diagramma a blocchi della logica per la generazione del clock.*

Da notare che il clock interno Fosc è usato solo in modalità master, sia in modo sincrono che asincrono. Il registro UBRR (*Usart Baud Rate Register*) ed il *down-counter* connesso ad esso, funzionano come un prescaler o come un generatore di *baud rate*. Il *down-counter* utilizza il clock di sistema Fosc, è caricato con il valore contenuto in UBRR ogni volta che raggiunge lo zero (oppure quando UBRR viene scritto) ed allo stesso tempo genera un

nuovo clock in uscita. Questo nuovo clock viene diviso secondo i fattori di 2, 8 o 16 per essere usato direttamente dall'unità ricevente. In Figura 3 è riportata la tabella contenente le equazioni per il calcolo del *baud rate* da impostare nel registro UBRR.

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal mode (U2Xn = 0)	$BAUD = \frac{f_{OSC}}{16(UBRRn+1)}$	$UBRRn = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed mode (U2Xn = 1)	$BAUD = \frac{f_{OSC}}{8(UBRRn+1)}$	$UBRRn = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master mode	$BAUD = \frac{f_{OSC}}{2(UBRRn+1)}$	$UBRRn = \frac{f_{OSC}}{2BAUD} - 1$

Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps).

BAUD Baud rate (in bits per second, bps).

f_{osc} System Oscillator clock frequency.

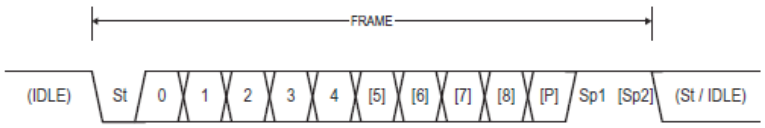
UBRRn Contents of the UBRRHn and UBRRLn Registers, (0-4095).

Figura 3: *Equazioni per il calcolo del baud rate.*

Formato del frame

Un frame seriale, o pacchetto dati, è definito come un carattere, composto da un minimo di 5 ad un massimo di 9 bit, a cui sono associati dei bit di sincronizzazione per l'inizio e fine trasmissione e dei bit di parità per la verifica di eventuali errori in trasmissione o in ricezione. In Figura 4 è riportato un esempio di formato del frame.

Frame Formats



St Start bit, always low.

(n) Data bits (0 to 8).

P Parity bit. Can be odd or even.

Sp Stop bit, always high.

IDLE No transfers on the communication line (RxDn or TxDn). An IDLE line must be high.

Figura 4: *Formato del frame.*

Impostazione dei parametri di comunicazione

Come riferimento per la scrittura di questo paragrafo, valido per tutte le USART a bordo dell'ATmega, prenderò in considerazione l'USART0. I parametri di comunicazione scelti per la trasmissione e la ricezione sono i seguenti:

- 9600 baud
- 8-bit
- un bit di stop
- nessuna parità
- modo asincrono

Il valore del *baud rate* da assegnare al registro UBRR0, espresso in bit/s, viene calcolato in fase di scrittura del software utilizzando l'equazione riportata in Figura 3.

Gli **8-bit** vengono impostati scrivendo ad 1 i bit UCSZ01 e UCSZ00 (*Usart Character Size*) nel registro UCSR0C (*Usart Control Status Register*), come riportato in Figura 5.

UCSZn Bits Settings

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Figura 5: *Impostazioni dei bit UCSZn.*

Un bit di stop (**Stop Bit**) viene impostato lasciando a 0 il bit USBS0 (*Usart Stop Bit Select*) nel registro UCSR0C, come mostrato in Figura 6.

USBS Bit Settings

USBSn	Stop Bit(s)
0	1-bit
1	2-bit

Figura 6: *Impostazione del bit USBSn.*

La **Parità** viene impostata lasciando a 0 i bit UPM01 e UPM00 (*Usart Parity Mode*) nel registro UCSR0C, come mostrato in Figura 7.

UPMn Bits Settings

UPMn1	UPMn0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

Figura 7: *Impostazioni dei bit UPMn.*

Il **Modo Asincrono** viene impostato lasciando a 0 i bit UMSEL01 e UMSEL00 (*Usart Mode Selection*) nel registro UCSR0C, come mostrato in Figura 8.

UMSELn Bits Settings

UMSELn1	UMSELn0	Mode
0	0	Asynchronous USART
0	1	Synchronous USART
1	0	(Reserved)
1	1	Master SPI (MSPIM)(1)

Figura 8: *Impostazioni dei bit UMSELn.*

Si fa presente che tutti i bit posti inizialmente a 0 si possono tralasciare nella scrittura del programma, ma è sempre buona norma impostarli come richiesto dall'applicazione.

Abilitazione dell'USART alla trasmissione

La trasmissione viene abilitata portando ad 1 il bit TXEN (*Transmitter Enable*) nel registro UCSR0B.

```
UCSR0B |= (1 << TXEN0);
```

Abilitazione dell'USART alla ricezione

La ricezione viene abilitata portando ad 1 il bit RXEN (*Receiver Enable*) nel registro UCSR0B.

```
UCSR0B |= (1 << RXEN0);
```

Funzione per l'inizializzazione dell'USART in trasmissione e ricezione

La funzione di inizializzazione finale per l'usart0, comprensiva dei `define` necessari e di tutti i parametri scelti in precedenza, risulta la seguente:

```
// Definisco la frequenza della CPU
#define F_CPU 16000000UL

// Definisco il valore della velocità in baud
#define BAUD 9600
```

```
// Definisco il valore del prescaler
#define BAUD_PRESCALER ((F_CPU)/(BAUD*16UL)-1)

// Funzione per inizializzare l'usart
void usart_init (void){

 // Imposto il prescaler
 UBRR0H = (BAUD_PRESCALER >> 8);
 UBRR0L = BAUD_PRESCALER;

 // Abilito la trasmissione e la ricezione
 UCSR0B |= (1 << TXEN0) | (1 << RXEN0);

 // Imposto l'usart a 8-bit, nessuna parità, 1 bit di stop,  modo
asincrono
 UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00);
}
```

Funzione per la trasmissione di un frame da 5 a 8 data bit

Affinché la trasmissione abbia luogo non basta solo abilitarla, bisogna portare ad 1 anche il bit UDRE0 (*Usart Data Register Enable*) nel registro UCSR0A. Essa inizia caricando il dato da trasmettere nel registro UDR0 (*Usart Data Register*). Questo dato viene trasferito nello *shift register* appena questo è pronto per l'invio di nuovi dati, ovvero quando non ci sono trasmissioni in corso o subito dopo il bit di stop dell'ultimo frame trasmesso. Il dato viene trasmesso secondo la velocità stabilita in fase di inizializzazione. La funzione per l'invio dei dati è la seguente:

```
// Funzione per inviare il dato
void usart_transmit_Char (unsigned char chr){

 // Attendo che si svuoti il buffer di trasmissione
 while (! (UCSR0A & (1 << UDRE0)));

 // Carico il dato nel registro e lo trasmetto
 UDR0 = chr;
}
```

La verifica che il dato è stato trasmesso ed il buffer di trasmissione è vuoto, quindi pronto a ricevere nuovi dati, viene fatta controllando che il bit UDRE0 sia impostato ad 1. In questo caso viene caricato un nuovo valore nel registro UDR0.

Flag di interrupt in trasmissione

I bit che possono essere usati in trasmissione come flag di interrupt sono i bit UDRE0 e TXC0 (*Transmit Complete*). Nel caso del bit UDRE0, l'evento d'interrupt viene abilitato portando ad 1 il bit UDRIE0 (*Data Register Empty Interrupt Enable*) nel registro UCSR0B ed abilitando l'interrupt globale. Quando si verifica l'evento d'interrupt, UDRE0 viene impostato ad 1, indicando così che il buffer è vuoto e pronto per ricevere nuovi dati. Alla ricezione dei nuovi dati viene ripulito. Nel caso del bit TXC0, l'evento interrupt viene abilitato portando ad 1 il bit TXCIE0 (*Data Register Empty Interrupt Enable*) nel registro UCSR0B ed abilitando l'interrupt globale. TXC0 viene impostato ad 1 quando tutto il frame viene svuotato dallo *shift register* e non sono presenti nuovi dati nel buffer di trasmissione UDR. Viene automaticamente ripulito dall'hardware al verificarsi dell'evento interrupt, ovvero quando la trasmissione è completata

Funzione per la ricezione di un frame da 5 a 8 data bit

La ricezione inizia quando al ricevitore arriva un bit valido di inizio comunicazione (start bit). Ogni bit che lo segue viene campionato e trasferito nello *shift register* fino al primo bit di stop. Quando tutto il frame è stato ricevuto viene trasferito nel registro UDR0 per la lettura. La funzione per la ricezione dei dati è la seguente:

```
// Funzione per ricevere il dato
unsigned char uart_receive_Char (void){

 // Attendo che il dato sia ricevuto
 while (! (UCSR0A & (1 << RXC0)));

 // Leggo il dato ricevuto
 return UDR0;
}
```

La funzione controlla se sono presenti dati non ancora letti nel buffer di ricezione UDR0, verificando lo stato del flag RXC0 (*Receive Complete*). Se sono presenti dati non ancora letti questo bit è impostato ad 1, se tutti i dati sono stati letti questo bit è impostato a 0.

Flag di interrupt in ricezione

L'unico bit che può essere usato in ricezione come flag di interrupt è il bit RXC0. Se il ricevitore è disabilitato (RXEN0 = 0) tutti i dati contenuti nel registro UDR vengono scaricati (*flush*) e questo bit assume il valore 0. L'evento interrupt viene abilitato portando ad 1 il bit RXCIE0 (*Receive Complete Interrupt Enable*) nel registro UCSR0B ed abilitando l'interrupt globale. Se nel buffer di ricezione sono presenti dati non ancora letti RXC0 viene impostato ad 1. E' necessario comunque leggere i dati ricevuti dal registro UDR per permettere la pulizia di questo flag.

Funzione per lo svuotamento del buffer di ricezione

Nel caso fosse necessario svuotare il registro UDR0 a causa di errori in ricezione, la funzione che esegue questo è la seguente:

```
void USART_Flush( void ){

 unsigned char dummy;

 while ( UCSRnA & (1<<RXCn) ) dummy = UDRn;
}
```

Registri di configurazione

UDR0: Usart I/O Data Register

Questo registro è diviso in due buffer da 8 bit ciascuno, uno ricevente relativo alla lettura dei dati ricevuti (RXB) ed uno trasmittente relativo alla scrittura dei dati da trasmettere. In Figura 9 è mostrato il dettaglio del registro.

UDRn – USART I/O Data Register n

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDRn (Read)
	TXB[7:0]								UDRn (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 9: *USART I/O Data Register.*

UCSR0A: Usart Control Status Register A

Questo è il primo registro di controllo di stato. In Figura 10 è mostrato il dettaglio del registro.

UCSRnA – USART Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

Figura 10: *USART Control Status Register A.*

Il bit 7 **RXC0** (*Receive Complete*) verifica che la ricezione dei dati sia completa. Viene impostato ad 1 quando sono ancora presenti dati da leggere ed impostato a 0 quando il buffer in ricezione è vuoto. Può essere usato per generare un interrupt.

Il bit 6 **TXC0** (*Transmit Complete*) verifica che la trasmissione dei dati sia completa. Viene impostato ad 1 quando l'intero frame è stato trasferito nello *shift register* e non ci sono dati nel registro UDR. Viene ripulito automaticamente in caso di un evento d'interrupt oppure può essere ripulito scrivendo un 1 logico nella sua locazione. Può essere usato per generare un interrupt.

Il bit 5 **UDRE0** (*Data Register Empty*) indica se il buffer in trasmissione nel registro UDR è pronto per ricevere nuovi dati. Se impostato ad 1 indica che il buffer è vuoto. Può essere

usato per generare un interrupt.

Il bit 4 **FE0** (*Frame Error*) verifica il primo bit di stop del frame memorizzato nel buffer di ricezione. Se il bit di stop è stato correttamente letto FE0 viene impostato a 0, in caso di errore viene impostato ad 1. Questo bit deve essere sempre scritto a 0 quando si deve scrivere sul registro UCSR0A

Il bit 3 **DOR0** (*Data OverRun*) indica una perdita di dati dovuta alla ricezione di un nuovo frame, prima che il precedente sia letto. Viene impostato ad 1 quando vengono persi dei frame tra due letture del registro UDR e viene ripulito quando il frame ricevuto è stato trasferito con successo dallo *shift register* al buffer di ricezione. Questo bit deve essere sempre scritto a 0 quando si deve scrivere sul registro UCSR0A.

Il bit 2 **UPE0** (*Parity Error*) indica che il frame ricevuto ha un errore di parità. Se il *parity checker* non è abilitato questo bit viene sempre letto come 0. Questo bit deve essere sempre scritto a 0 quando si deve scrivere sul registro UCSR0A.

Il bit 1 **U2X0** (*Double Transmission Speed*) se impostato a 0 ha effetto solo su operazioni asincrone, se impostato ad 1 lavora solo con operazioni sincrone, riduce il divisore del baud rate da 16 a 8 e raddoppia la velocità di trasferimento dei dati.

Il bit 0 **MPCM0** (*Multi Processor Communication Mode*) se impostato ad 1 abilita questa modalità.

UCSR0B: Usart Control Status Register B

Questo è il secondo registro di controllo di stato. In Figura 11 è mostrato il dettaglio del registro.

UCSRnB – USART Control and Status Register n B

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 11: *USART Control Status Register B.*

Il bit 7 **RXCIE0** (*RX Complete Interrupt Enable*), se impostato ad 1, abilita l'interrupt sul bit RXC0. L'interrupt è generato solo se anche il bit RXC0 e l'interrupt globale sono entrambi impostati ad 1.

Il bit 6 **TXCIE0** (*TX Complete Interrupt Enable*), se impostato ad 1, abilita l'interrupt sul bit TXC0. L'interrupt è generato solo se anche il bit TXC0 e l'interrupt globale sono entrambi impostati ad 1.

Il bit 5 **UDRIE** (*Data Register Interrupt Enable*), se impostato ad 1, abilita l'interrupt sul bit UDRE0. L'interrupt è generato solo se anche il bit UDRE0 e l'interrupt globale sono entrambi impostati ad 1.

Il bit 4 **RXEN0** (*Receiver Enable*), se impostato ad 1, abilita la ricezione dei dati. La sua disabilitazione provoca lo svuotamento del buffer di ricezione.

Il bit 3 **TXEN0** (*Transmitter Enable*), se impostato ad 1, abilita la trasmissione dei dati. Qualora una trasmissione sia in corso, la trasmissione viene disabilitata solo dopo che tutti i bit sono trasmessi.

Il bit 2 **UCSZ02** (*Character Size*), in combinazione con i bit UCSZ01 e UCSz00, imposta il numero dei data bit.

Il bit 1 **RXB80** (*Receive Data Bit 8*) è il nono bit del carattere ricevuto quando si lavora con un frame a 9 bit.

Il bit 0 **TXB80** (*Transmit Data Bit 8*) è il nono bit del carattere trasmesso quando si lavora con un frame a 9 bit.

UCSR0C: Usart Control Status Register C

Questo è l'ultimo registro di controllo di stato. In Figura 12 è mostrato il dettaglio del registro.

UCSRnC – USART Control and Status Register n C

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

Figura 12: *USART Control Status Register C.*

I bit 7-6 **UMSEL01 - UMSEL00** selezionano la modalità operativa, come riportato nella Figura 8.

I bit 5-4 **UPM01 – UPM00** selezionano il modo parità, come riportato in Figura 7.

Il bit 3 **USBS0** seleziona il numero di bit di stop, come riportato in Figura 6.

I bit 2-1 **UCSZ01 - UCSZ00** (*Character Size*), in combinazione con il bit UCSZ02, impostano il numero dei data bit, ovvero la grandezza del frame.

UBRR0L - UBRR0H: Usart Baud Rate Register

Il doppio registro permette di memorizzare i 12 bit che rappresentano il valore del *baud rate* calcolato con l'equazione riportata in Figura 3. UBRRH contiene i quattro bit più significativi mentre UBRRL contiene i restanti otto bit meno significativi di questo valore. In Figura 13 è mostrato il dettaglio del registro.

UBRRnL and UBRRnH – USART Baud Rate Registers

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRR[11:8]				UBRRHn
	UBRR[7:0]								UBRRLn
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Figura 13: *Registro UBRR0.*

Programmi di esempio

Gli esempi sono stati testati utilizzando una demoboard Atmega ed una scheda miniCOM USB LaurTec come porta seriale virtuale, collegata al PC tramite cavo USB (Figura 14).

Figura 14: *Scheda MiniCom USB.*

La lettura dei dati trasmessi e ricevuti è stata effettuata utilizzando come terminale un interfaccia grafica dedicata, come RS232 Terminal scaricabile gratuitamente dal sito LaurTec. Nell'Atmega2560 i pin seriali relativi alla trasmissione e ricezione dei dati sono i seguenti:

- TXD PE1
- RXD PE0

mentre gli stessi pin sul connettore di espansione SV2 a bordo della miniCOM sono i seguenti:

- TX 1
- RX 2

Prima di procedere bisogna collegare le masse delle due schede tra loro, successivamente

il pin PE1 al pin 2 di SV2 ed il pin PE0 al pin 1. Inoltre, una volta collegata la miniCOM al computer, bisogna aprire il terminale ed impostarlo con gli stessi parametri di trasmissione descritti sopra.

Poiché, come già detto, tutto questo è valido anche per le altre USART, chi per esempio volesse utilizzare l'USART1 non deve fare altro che sostituire nei vari registri il numero 0 con il numero 1 in tutti i registri e bit interessati. Ad esempio, UBRR0H diventa UBRR1H, UBRR0L diventa UBRR1L, i registri UCS0 diventano UCS1 e così via. Consultare eventualmente il datasheet per i relativi nomi dei registri e pin usati dalle relative USART.

Esempio 1

Il primo esempio trasmette al terminale la frase "Hello wordl!". Ogni carattere ricevuto viene affiancato al precedente formando così la stringa che viene inviata al terminale.

```
#pragma config BODLEVEL = BODLEVEL_2V7
#pragma config OCDEN = CLEAR
#pragma config JTAGEN = SET
#pragma config SPIEN = SET
#pragma config WDTON = CLEAR
#pragma config EESAVE = CLEAR
#pragma config BOOTSZ = BOOTSZ_4096W_1F000
#pragma config BOOTRST = SET
#pragma config CKDIV8 = CLEAR
#pragma config CKOUT = CLEAR
#pragma config SUT_CKSEL = SUT_CKSEL_EXTXOSC_8MHZ_XX_16KCK_65MS


// Definisco la frequenza della CPU
#define F_CPU   16000000UL

// Includo i file di testata per la gestione I/O, dei ritardi, della
libreria standard
#include <avr/io.h>
#include <util/delay.h>
#include <stdlib.h>

// Definisco il valore della velocità in baud
#define BAUD 9600
// Definisco il valore del prescaler
#define BAUD_PRESCALER ((F_CPU)/(BAUD*16UL)-1)

char String[] = "Hello world!!";

//Prototipi delle funzioni
void usart_init (void);
void usart_transmit_Char (unsigned char chr);
void usart_transmit_String (char* String);


int main(void){

    // Inizializzo l'usart
    usart_init ();

 while(1){
```

```
    // Invio la stringa al terminale
    usart_transmit_String (String);
    _delay_ms (1000);
 }
}


// Funzione per inizializzare l'usart
void usart_init (void){
    // Imposto il prescaler
    UBRR0H = (BAUD_PRESCALER >> 8);
    UBRR0L = BAUD_PRESCALER;

    // Abilito la trasmissione e la ricezione
    UCSR0B |= (1 << TXEN0) | (1 << RXEN0);

    // Imposto l'usart a 8-bit, nessuna parità, 1 bit di stop,  modo
   //asincrono

    UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00);
}


// Funzione per inviare il dato
void usart_transmit_Char (unsigned char chr){
    // Attendo che si svuoti il buffer di trasmissione
    while (! (UCSR0A & (1 << UDRE0)));

    // Carico il dato nel registro e lo trasmetto
    UDR0 = chr;
}

// Funzione per inviare una stringa
void usart_transmit_String (char* String){
    // Invia la stringa, un carattere per volta, all'usart
    while (*String != 0x00){
       usart_transmit_Char (*String);
       String ++;
    }

}
```

Esempio 2

Nel secondo esempio il carattere stampato sul terminale viene inviato per la lettura su un display LCD. Il display è collegato alla PORTD come per l'esempio mostrato nel Capitolo sul display LCD.

```
#pragma config BODLEVEL = BODLEVEL_2V7
#pragma config OCDEN = CLEAR
#pragma config JTAGEN = SET
#pragma config SPIEN = SET
#pragma config WDTON = CLEAR
#pragma config EESAVE = CLEAR
#pragma config BOOTSZ = BOOTSZ_4096W_1F000
#pragma config BOOTRST = SET
#pragma config CKDIV8 = CLEAR
#pragma config CKOUT = CLEAR
```

```
#pragma config SUT_CKSEL = SUT_CKSEL_EXTXOSC_8MHZ_XX_16KCK_65MS


// Definisco la frequenza della CPU
#define F_CPU   16000000UL

// Includo i file di testata per la gestione I/O, dei ritardi, della
libreria standard e dell'lcd
#include <avr/io.h>
#include <util/delay.h>
#include <stdlib.h>
#include "lcd.h"
#include "lcd.c"

// Definisco il valore della velocità in baud
#define BAUD 9600

// Definisco il valore del prescaler
#define BAUD_PRESCALER ((F_CPU)/(BAUD*16UL)-1)

//Prototipi di funzione
void usart_init (void);
unsigned char usart_receive_Char (void);


int main (void){
    // Creo le variabili che conterranno il dato
    unsigned char a;

    // Inizializzo l'usart
    usart_init ();

    // Inizializzo l'lcd
    lcd_init (LCD_DISP_ON_CURSOR);
    lcd_home ();
    _delay_ms (50);

 while(1){
     // Salvo il dato ricevuto nella variabile 'a'
    a = usart_receive_Char ();

    // Posiziono il cursore del display
    //lcd_clrscr ();
    //lcd_home ();

    // Visualizzo il dato ricevuto come CHAR
    lcd_putc (a);
    _delay_ms (100);
 }

 return 0;
}

// Funzione per inizializzare l'usart
void usart_init (void){
    // Imposto il prescaler
    UBRR0H = (BAUD_PRESCALER >> 8);
    UBRR0L = BAUD_PRESCALER;

    // Abilito la trasmissione e la ricezione
    UCSR0B |= (1 << TXEN0) | (1 << RXEN0);
```

```
    // Imposto l'usart a 8-bit, nessuna parità, 1 bit di stop,  modo
     // asincrono
    UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00);
}

    // Funzione per ricevere il dato
    unsigned char usart_receive_Char (void){
    // Attendo che il dato sia ricevuto
    while (! (UCSR0A & (1 << RXC0)));

    // Dato ricevuto
    return UDR0;
}
```

Indice Alfabetico

C
Character Size....13
D
Data OverRun....12
Data Register Empty....11
Data Register Empty Interrupt Enable....10
Data Register Interrupt Enable....12
DOR0....12
Double Transmission Speed....12
F
FE0....12
flush....10
Fosc....5
Frame Error....12
L
LaurTec....14
LCD....16
M
miniCOM....14
Modo Asincrono....8
MPCM0....12
Multi Processor Communication Mode....12
P
Parità....7
parity....5
Parity Error....12
R
Receive Complete....10 e seg.
Receive Data Bit 8....13
Receiver Enable....8, 13
RS232 Terminal....14
RX....14
RX Complete Interrupt Enable....12
RXB....11
RXB80....13
RXC0....10 e seg.
RXCIE0....10, 12
RXD....14
RXEN....8
RXEN0....10, 13
S
shift register....9
shift register....5
Stop Bit....7
T
Transmit Complete....10 e seg.
Transmit Data Bit 8....13
Transmitter Enable....8, 13
TX....14
TX Complete Interrupt Enable....12
TXB80....13
TXC0....10 e seg.
TXCIE0....10, 12
TXD....14
TXEN....8
TXEN0....13
U
U2X0....12
UBRR....5
UBRR0....7
UBRR0H....14
UBRR0L....14
UBRRH....14
UBRRL....14
UCSR0A....9, 11
UCSR0B....8, 10
UCSR0C....7
UCSZ00....7, 13
UCSZ01....7, 13
UCSZ02....13
UDR....5
UDR0....9 e segg.
UDRE0....9 e segg.
UDRIE....12
UDRIE0....10
UMSEL00....8, 13
UMSEL01....8, 13
UPE0....12
UPM00....7, 13
UPM01....7, 13
Usart Character Size....7
Usart Control Status Register....7
Usart Data Register....9
Usart Data Register Enable....9
Usart Mode Selection....8
Usart Parity Mode....7
Usart Stop Bit Select....7
USART0....4
USART1....4
USART2....4
USART3....4
USB....14
USBS0....7, 13
X
XKCn....5

History

Data	Versione	Autore	Revisione	Descrizione Cambiamento
21/11/20	1.0	Marcello Pinna	Mauro Laurenti	Versione Originale.