SERIALE RS232 CON ARDUINO

La comunicazione seriale asincrona RS232 è stata già descritta nel seguente documento, considerando il microcontrollore PIC:

https://danielepostacchini.it/wp-content/uploads/2020/02/PIC-E-RS232.pdf

Vediamo in questo caso come gestirla con la scheda Arduino. Esistono 3 modalità di gestire la comunicazione con Arduino (come anche con il microcontrollore PIC) e più precisamente si può:

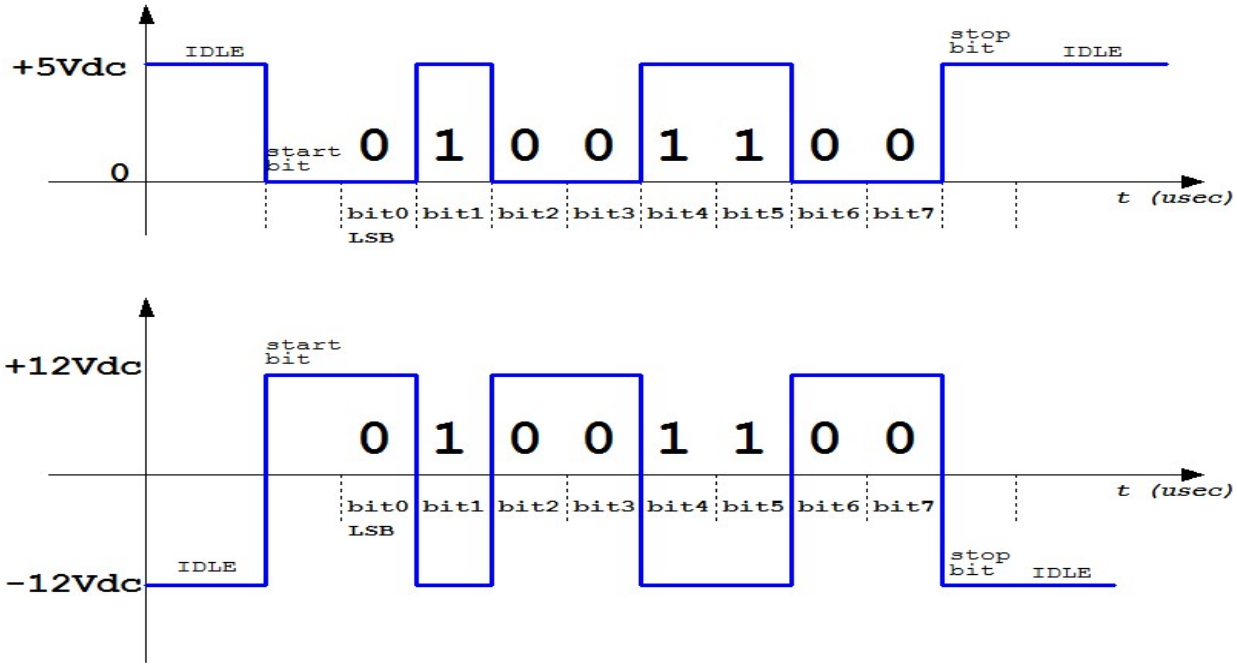
1. Gestire la seriale tramite l'hardware implementato nel microcontrollore presente sulla scheda.
2. Gestire la seriale tramite le librerie software rese disponibili sull'I.D.E. di Arduino (I.D.E. indegrated Development Environment, ambiente di sviluppo integrato).
3. Realizzare delle funzioni per inviare e ricevere i dati.

Sulla scheda Arduino, l'hardware interno viene utilizzato per la comunicazione USB con il PC, come ad esempio quanto utilizziamo il monitor seriale.

Pertanto può capitare, di non poter utilizzare i due piedini, collegati all'hardware interno che sono D0 e D1. Per questo motivo le modalità 2 e 3 sono forse le più utilizzate.

In entrambi i casi la gestione della seriale viene fatta tramite software, ma nel terzo caso (il primo che affronteremo) dobbiamo fare attenzione alle tempistiche utilizzate, in quanto per rispettare il bitrate stabilito, occorre considerare il tempo impiegato dalle istruzioni presenti nelle varie funzioni come vedremo più avanti.

Inoltre occorre ricordare che i segnali generati dalla scheda Arduino, sono segnali 0-5Volt o in determinate schede 0-3,3V. Pertanto occorre traslare con dei circuiti il segnale al livello previsto dallo stantard RS232 e cioè +12V e -12V.



Routine software per la TRASMISSIONE di un dato via seriale asincrona on ARDUINO.

METODO 1

```
//funzione per inviare un byte, 8 bit nessuna parita, 9600bps
//la variabile dato contiene il dato da inviare
void TRASMETTI(char dato){
     char cont=8;      //conteggio degli 8 bit inviati
     char dato_temp;   //variabile temporanea di appoggio del dato
     char mask=1;      //maschera per isolare il bit LSB

     //bit di start
     digitalWrite(TX,LOW);
     delayMicroseconds(104);

     //loop per invio degli 8 bit
     while(cont>0){
        dato_temp=dato & mask; //maschero 7 bit e lascio il bit LSB
        if(dato_temp==0)  digitalWrite(TX,LOW);
        else              digitalWrite(TX,HIGH);
        dato=dato>>1;
        cont--;
        delayMicroseconds(104);
     }//while

     //bit di stop
     digitalWrite(TX,HIGH);
     delayMicroseconds(104);
}//TRASMETTI
```

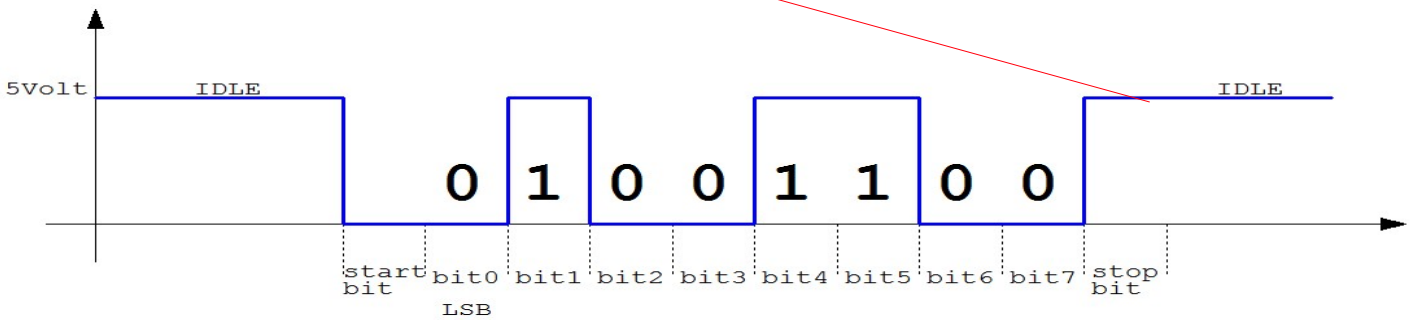
Il dato viene passato alla funzione quando viene richiamata nel programma es. TRASMETTI('A');

La variabile dato conterra in questo caso il codice ASCII del carattere A, la variabile è temporanea, e viene utilizzata solo all'interno della funzione.

*Con questo metodo, si utilizza una variabile per conteggiare gli 8 bit da inviare **cont**, una variabile che contiene il dato dopo l'operazione di mascheratura **dato_temp** ed una variabile che contiene il valore della maschera **mask**.*

*Il funzionamento della mascheratura è il seguente. Supponiamo di dover controllare il valore di un solo bit, in questo caso il **bit 0** (LSB) di una variabile ad 8 bit (dato). Per fare questo, si fa l'operazione di AND bit a bit tra la variabile ad 8 bit ed una variabile contenente il valore **00000001** chiamata **maschera**. Il risultato conterrà **0** se il bit LSB della variabile **dato** vale 0, o 1 se il bit LSB dell variabile **dato** vale **1**. Successivamente testando la variabile **dato_temp**, si decide se mettere TX ad 0 o ad 1. Dopo ogni bit inviato sul pin TX, occorre shiftare la variabile dato ed attendere un tempo di 104us (9600bps 1bit=104us).*

dato	B7	B6	B5	B4	B3	B2	B1	B0
				AND				
mask	0	0	0	0	0	0	0	1
				=				
dato_temp	0	0	0	0	0	0	0	B0



shift a destra di una posizione
dato=dato>>1

dato	B7	B6	B5	B4	B3	B2	B1	B0

pin TX

Il ciclo si ripete 8 volte in un ciclo while.

METODO 2

```
//funzione per inviare un byte, 8 bit nessuna parita, 9600bps
//la variabile dato contiene il dato da inviare
void TRASMETTI(char dato){
     char cont=8;     //conteggio degli 8 bit inviati

     //bit di start
     digitalWrite(TX,LOW);
     delayMicroseconds(104);

     //loop per invio degli 8 bit
     while(cont>0){
        if(bitRead(dato,0)) digitalWrite(TX,LOW);
        else                digitalWrite(TX,HIGH);
        dato=dato>>1;
        cont--;
        delayMicroseconds(104);
     }//while

     //bit di stop
     digitalWrite(TX,HIGH);
     delayMicroseconds(104);
}//TRASMETTI
```

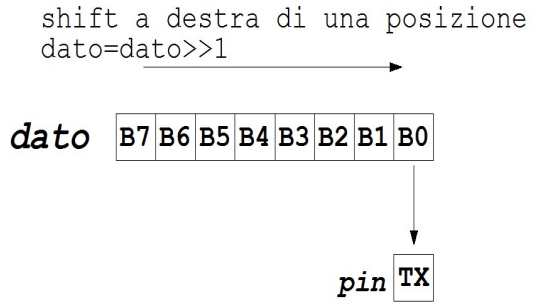
In questo caso la funzione si differenzia dalla pecedente nella parte interna al ciclo while, dove si testa il bit LSB del dato da inviare.
*In questo caso si utilizza la possibilità che l'I.D.E. di Arduino offre per il test di un singolo bit di una variabile, e cioè la funzione **bitRead**.*

Alla funzione bitRead va passata la variabile ed il numero del bit da testare, e restituisce il valore del bit testato.

*Scrivendo ad esempio **risultato=bitTest(dato,0)** si testa il **bit 0** della variabile **dato** ed il valore del bit viene caricato nella variabile **risultato**.*

*Così nel nostro caso scrivere **if(bitRead(dato,o))** significa testare il **bit 0** della variabile **dato**, ed in base al risultato si decide se mettere TX a livello basso o alto.*

*Per il resto la funzione è identica alla precedente, in quanto viene testato il bit 0 della variabile **dato** che viene shiftata di una posizione a destra per 8 volte.*



METODO 3

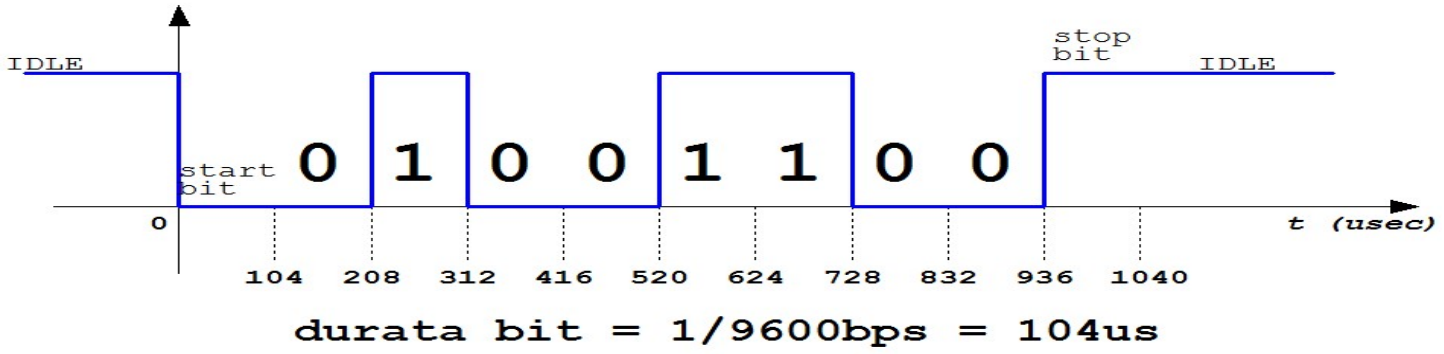
```
//funzione per inviare un byte, 8 bit nessuna parita, 9600bps
//la variabile dato contiene il dato da inviare
void TRASMETTI(char dato){
     char num_bit=0;     //numero del bit letto

     //bit di start
     digitalWrite(TX,LOW);
     delayMicroseconds(104);

     //loop per invio degli 8 bit
     while(num_bit<8){
        if(bitRead(dato,num_bit)) digitalWrite(TX,LOW);
        else                      digitalWrite(TX,HIGH);
        num_bit++;
        delayMicroseconds(104);
     }//while

     //bit di stop
     digitalWrite(TX,HIGH);
     delayMicroseconds(104);
}//TRASMETTI
```

*In questo caso si utiizza sempre la funzione **bitRead**, ma anziché shiftare la variabile **dato** si modifica il bit da testare, indicato in questo caso con una variabile chiamata **num_bit.***

*La variabile inizializzata al valore di 0, consentirà perciò la lettura dei 7 bit della variabile **dato**.*

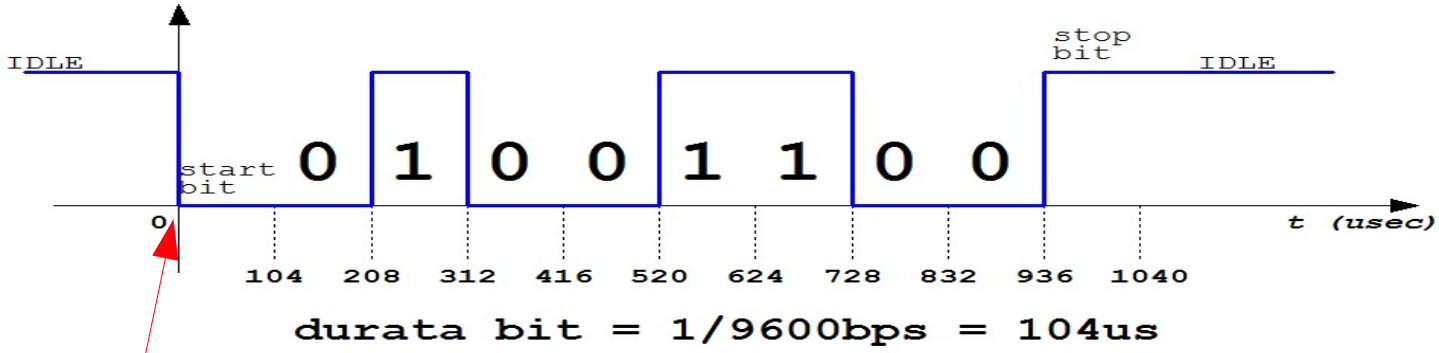
Per il resto anche in questo caso il programma risulta identico ai precedenti.

Nei 3 casi precedenti, non abbiamo considerato il tempo necessario per l'esecuzione delle istruzioni. Per rispettare la tempistica richiesta dalla velocità di 9600 bit per secondo, occorrerebbe calcolare il tempo impiegato per le varie istruzioni, ed eseguire un delay non di 104 us, ma di un valore leggermente inferiore.
In alternativa il problema si potrebbe risolvere utilizzando le funzioni messe a disposizione nella libreria software serial, come vedremo più avanti.



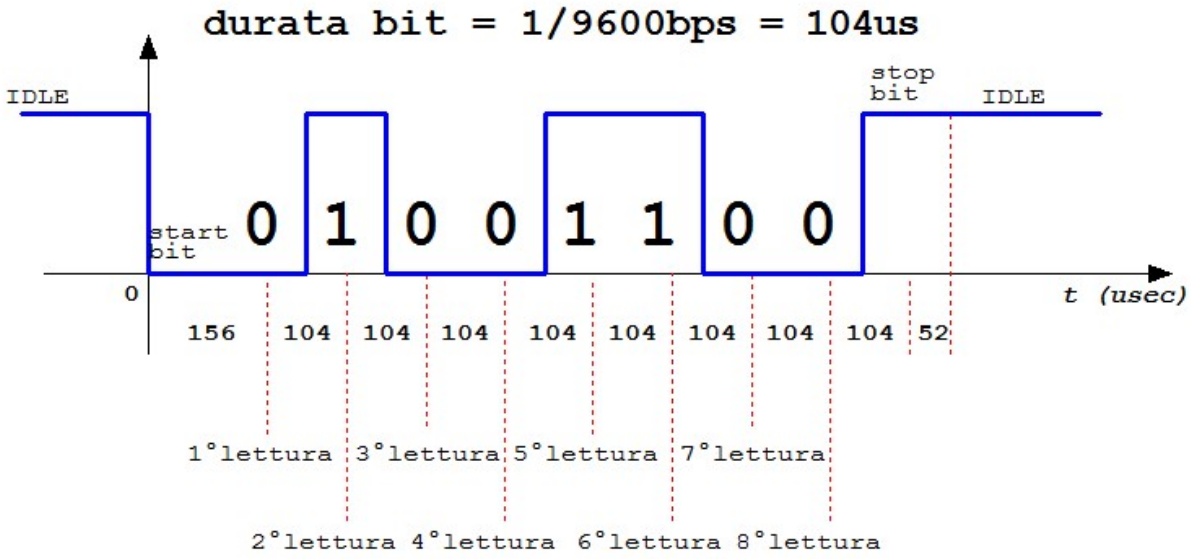
Routine software per la RICEZIONE di un dato via seriale asincrona con ARDUINO

In questo caso valutiamo due metodi, a differenza della trasmissione evitiamo il metodo della mascheratura perché troppo macchinoso.
La funzione che chiameremo RICEVI, verrà chiamata quando durante il programma principale, si rileva un valore di RX=0, e cioè presumibilmente corrispondente allo start bit.




```
void loop() {

  if(digitalRead(RX)==0) valore=RICEVI();

}
```

Nel ciclo principale, si testa il bit RX, e se il bit vale 0 si richiama la funzione RICEVI, che restituirà il valore ricevuto sulla seriale nella variabile "valore". La variabile dovrà essere ovviamente dichiarata.



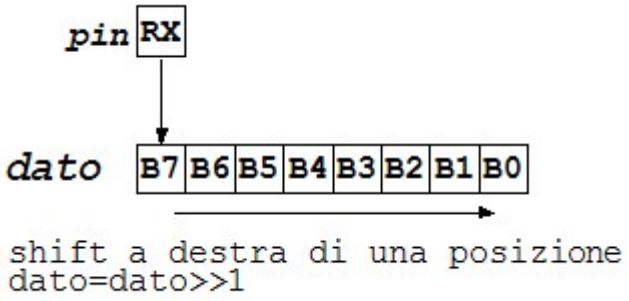
All'interno della funzione di lettura, ci dovrà essere un'attesa iniziale di 156 us, per posizionarsi al centro del primo bit.

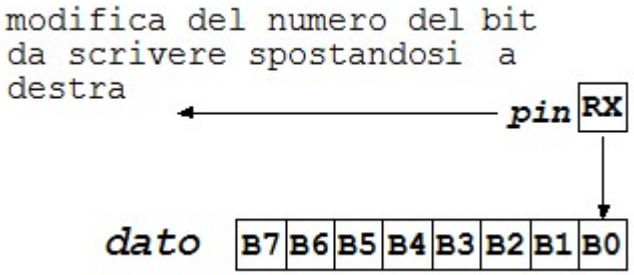
Successivamente si leggerà lo stato della linea RX ogni 104us, trovandosi sempre in questo modo, al centro del bit.

Dopo 8 letture, dovremo aspettare 52us, per attendere la fine dello stop bit.

Anche in questo caso si dovrebbe tener conto del tempo impiegato dalle istruzioni del programma, e ridurre in base a questo, la durata delle varie attese.

*La differenza tra le due seguenti funzioni, sta solo nel fatto che la prima legge nel ciclo il bit RX e mette ad uno il bit 7 della variabile **dato**, shiftandola ogni volta a destra, mentre la seconda mette a 0 o ad 1, i bit della variabile **dato** senza shiftarla, ma modificando in ogni ciclo il numero del bit.*





Dopo la lettura dell'ultimo bit ricevuto non devo shiftare.

```
//funzione per ricevere un byte, 8 bit nessuna parita, 9600bps
//restituisce il valore ricevuto
char RICEVI(){
  char cont=8;
  char dato=0;

  delayMicroseconds(156);  //attendo il bit di start
                           //per posizionarmi al centro del primo bit
  while(cont>0){
    if(digitalRead(RX)==0)   bitClear(dato,7);  //metto a zero il bit più significativo
    else                     bitSet(dato,7);    //metto ad uno il bit più significativo
    if(cont>1) dato=dato>>1;           //eseguo lo shift a destra tranne l'ultimo bit
    cont--;                            //decremento il contatore dei bit
    delayMicroseconds(104);            //attendo 104usec (9600 bps)
  }//while
    delayMicroseconds(52);            //attendo la fine del bit di stop
    return(dato);                      //restituisco il valore letto
}//RICEVI
```

```
//funzione per ricevere un byte, 8 bit nessuna parita, 9600bps
//restituisce il valore ricevuto
char RICEVI(){
  char num_bit=0;
  char dato=0;

  delayMicroseconds(156);  //attendo il bit di start
                           //per posizionarmi al centro del primo bit
  while(num_bit<8){
    if(digitalRead(RX)==0)   bitClear(dato,num_bit); //metto a zero il bit
    else                     bitSet(dato,num_bit);   //metto ad uno il bit
    num_bit++;                                       //incemento il bit
    delayMicroseconds(104);                          //attendo 104usec (9600 bps)
  }//while

    delayMicroseconds(52);                    //attendo la fine del bit di stop

    return(dato);                             //restituisco il valore letto
}//RICEVI
```

SERIALE SOFTWARE

SoftwareSerial Library

The Arduino hardware has built-in support for serial communication on pins 0 and 1 (which also goes to the computer via the USB connection). The native serial support happens via a piece of hardware (built into the chip) called a UART. This hardware allows the Atmega chip to receive serial communication even while working on other tasks, as long as there room in the 64 byte serial buffer.

The SoftwareSerial library has been developed to allow serial communication on other digital pins of the Arduino, using software to replicate the functionality (hence the name "SoftwareSerial"). It is possible to have multiple software serial ports with speeds up to 115200 bps. A parameter enables inverted signaling for devices which require that protocol.

The version of SoftwareSerial included in 1.0 and later is based on the NewSoftSerial library by Mikal Hart.

Functions

- SoftwareSerial()
- available()
- begin()
- isListening()
- overflow()
- peek()
- read()
- print()
- println()
- listen()
- write()

Nella guida di Arduino trovate la libreria per gestire una seriale in maniera software senza utilizzare I due piedini 0 ed 1 di Arduino, utilizzati per la comunicazione seriale con il PC.

Le funzioni che utilizzeremo sono quelle evidenziate.

- ***SoftwareSerial*** *crea l'oggetto che conterrà i metodi (le funzioni) per gestire la seriale.*
- *nome_oggetto.**available*** *è la funzione che consente di verificare l'arrivo del dato da leggere*
- *nome_oggetto.**begin*** *consente di inizializzare l'oggetto e stabilire la velocità in bps*
  - *nome_oggetto.**read*** *consente di leggere il dato dalla seriale*
  - *nome_oggetto.**print*** *consente di inviare il dato sulla seriale*

```
#include <SoftwareSerial.h>

SoftwareSerial mySerial(13,12);   //pin13=RX pin12=TX

char lettura;  //variabie per leggere dalla seriale

void setup() {
  mySerial.begin(9600);  //9600 bps
}

void loop() {
  //scrittura sulla seriale
  myserial.print("A");  //invia il carattere A

  //lettura dalla seriale
  if(mySerial.available()){
     lettura=mySerial.read();
   }
}
```

*Nel programma di esempio che vediamo qui vicino, troviamo l'inclusione della libreria, e la successiva definizione dei due piedini TX e RX nel momento in cui si crea l'oggetto **mySerial***

Definizione della velocità (bps).

E nel ciclo principale possiamo utilizzare le 3 funzioni principali;

myserial.print *- scrive il valore sulla seriale*

myserial.available *- verifica se c'è un dato sulla seriale*

myserial.read *- legge il dato presente sulla seriale*

SERIALE HARDWARE

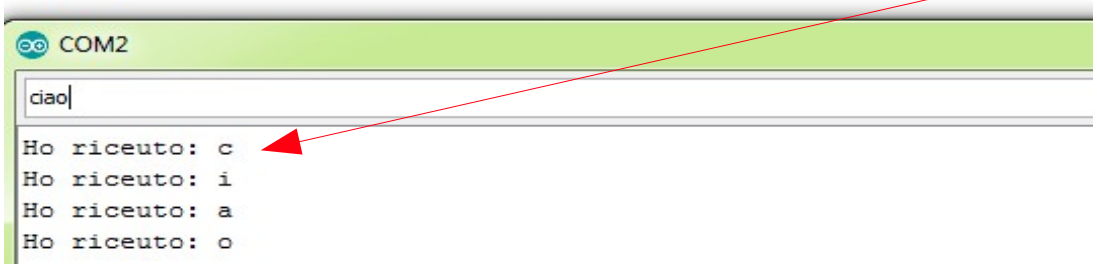
Come detto precedentemente la seriale hardware nella scheda Arduino, utilizza i pin D0 e D1. La seriale viene utilizzata anche per comunicare con il PC ed inviare i dati al monitor seriale.

```
char dato_letto;

void setup() {
  Serial.begin(9600);     //abilito la seriale per comunicare
}

void loop() {
if (Serial.available() > 0) {      //verifico la presenza del dato
     dato_letto = Serial.read();  //leggo il byte
     Serial.print("Ho riceuto: "); //scrivo una stringa
     Serial.println(dato_letto);
  }//if
}//loop
```

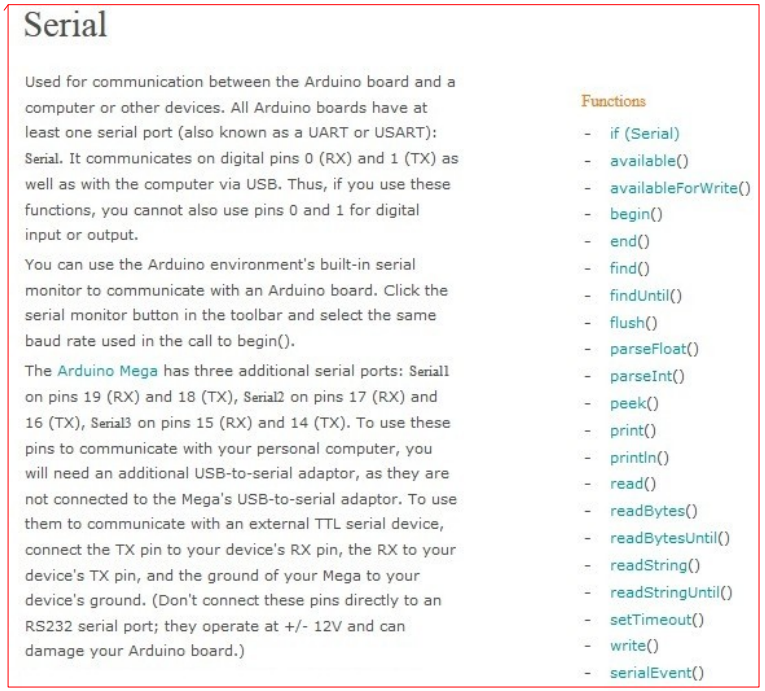
In questo esempio viene letto uno o più caratteri ricevuto sulla seriale hardware.

Sul monitor seriale posso scrivere ad esempio “ciao”, in questo modo verranno inviati i 4 caratteri componenti la parola sottoforma di codice ASCII sulla seriale hardware.

Il microcontrollore presente sulla scheda Arduino, leggera e risponderà con la scritta “ho ricevuto:” e l’eco dei caratteri letti.



Tutte le informazioni relative alle varie funzioni, sono disponibili nella guida online, nella sezione Serial.



CONCLUSIONI ED OSSERVAZIONI

In tutti i casi visti sopra, è stata considerata una comunicazione a 9600bps, con 8 bit, un bit di stop e nessun bit di parità.
Ciò significa che non è stato utilizzato alcun controllo di errore. Nei metodi che utilizzano una funzione software ad esempio, il bit di start viene considerato presente con una sola lettura del livello basso sul pin RX. Questo potrebbe causare dei problemi in casi di disturbi sulla linea, pertanto volendo realizzare una comunicazione seriale RS232, è sempre preferibile adottare dei metodi per controllare gli errori che potrebbero essere i seguenti:

- *Controllo dello start bit e stop bit.*
- *Utilizzo del bit di parità.*
- *Utilizzo del CRC (Cyclic Redundancy Check) applicandolo sull'intero messaggio composto da più byte nel seguente modo:*

  - *caricare un registro a 16 bit con il valore 0xFFFF (tutti 1)*
  - *eseguire l'OR esclusivo del registro con il primo byte, il risultato va nel registro stesso*
  - *spostare a destra di un bit il registro per 8 volte*
  - *se dopo ogni spostamento a destra, il bit fuoriuscito è 1 fare l'OR esclusivo del registro, con il byte 0xA001*
  - *eseguire l'OR esclusivo del byte successivo con il registro*
  - *ripetere i punti da 3 a 5 per tutti i byte del messaggio.*

In questo algoritmo di esempio, viene calcolato il CRC sui 6 byte contenuti dal vettore ***messaggio****.*
Il polinomio generatore utilizzato è ***A001*** *hex,utilizzato nel protocollo MODBUS.*
La spiegazione dettagliata del funzionamento del CRC, richiede un maggiore approfondimento, per ora limitiamoci a considerare questo algoritmo come funzionante.

```
char messaggio[6];     //vettore che contiene il messaggio

void calcola_CRC(char num_byte){
    unsigned int crc=0xFFFF;  //inizializzo un registro a 16 bit con tutti 1
    unsigned int carry;
    char cont=8;      //inizializzo i contatori
    char cont_byte=0;
    char lo_crc,hi_crc;

    while(cont_byte<num_byte){  //eseguo il ciclo per ogni byte partendo dal primo
      crc=crc^messaggio[cont_byte];  //OR esclusivo tra registro crc e byte del messaggio
      while(cont){
        carry=crc&0x0001;
        crc=crc>>1;                  //shift a destra di un bit
        if(carry) crc=crc^0xA001;  //se il bit uscito a destra è 1, OR escusivo con 0xA001
        cont--;                    //decremento per fare 8 volte la stessa operazione
       }//whie
      cont_byte++;  //incremento il contatore dei byte
      cont=8;
    }//while
}//calcola_CRC

 void loop(){
  messaggio[0]=0x0F;
  messaggio[1]=0x01;
  messaggio[2]=0x00;
  messaggio[3]=0x03;
  messaggio[4]=0x00;
  messaggio[5]=0x14;
  calcola_CRC(6);
 }
```