
# Standard Firmata per Arduino

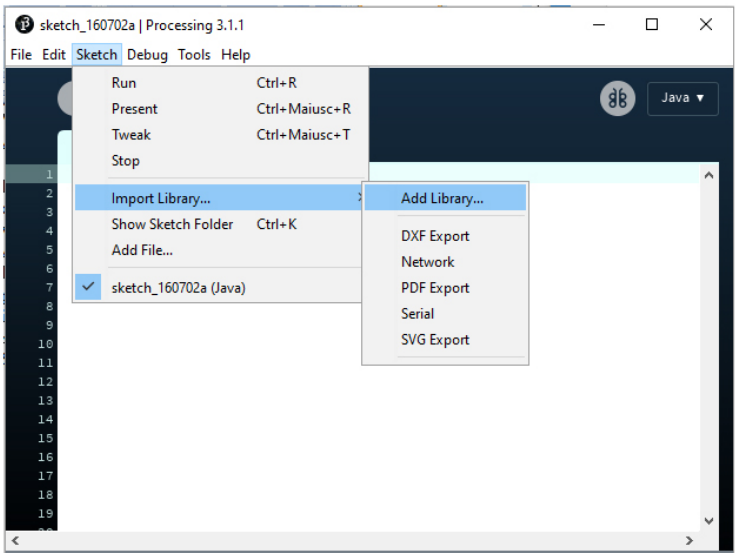
**Obiettivo**
Sperimentare la gestione di Arduino da remoto tramite l'utilizzo di un protocollo standard.

**Conoscenze preliminari**
Per controllare Arduino da remoto, anziché fare tutto da soli – definendo un insieme di stringhe di comando, predisponendo uno sketch su Arduino che attende i comandi in arrivo sulla linea seriale e li esegue e attivando un programma su PC, che invia i comandi e visualizza le risposte – si può utilizzare un protocollo già collaudato, corredato di apposite librerie.
Lo standard Firmata è un protocollo seriale di comunicazione e controllo per Arduino già incluso in alcuni ambienti di programmazione per Windows.

**Fasi operative**

**1) Sketch per Arduino**
Lo sketch per la scheda Arduino, in grado di interpretare i comandi Firmata provenienti dalla linea seriale, è reperibile in *Esempi* → *Firmata* → *StandardFirmata*.
Una volta aperto lo sketch, controllare che in *setup ( )* la velocità seriale sia settata a 57600 bps (*"Firmata.begin(57600);"*), compilare il programma e caricarlo su Arduino. Annotare qual è la porta COM utilizzata da Arduino (per esempio "COM3"), perché tale informazione servirà successivamente, e chiudere l'ambiente.

**2) Programma sul PC in ambiente Processing**
**A)** Predisporre il programma su PC. Per farlo, si può utilizzare l'ambiente di programmazione *Processing* per Windows, reperibile gratuitamente sul sito http://processing.org e da installare in *C:/Programmi(x86)*, che dispone già delle librerie per il protocollo Firmata nel file *processing2-arduino.zip*, scaricabile dal sito http://playground.arduino.cc/Interfacing/Processing.
**B)** Scompattare il file e copiare la cartella Arduino in *C:/Programmi(x86)/processing-3.1.1/lib/*.
**C)** Avviare l'ambiente di sviluppo "*i³ processing*" e aggiungere la libreria Arduino aprendo *Sketch* → *Import Library* → *Add library…* e selezionando *Arduino (Firmata)* dall'elenco che appare.

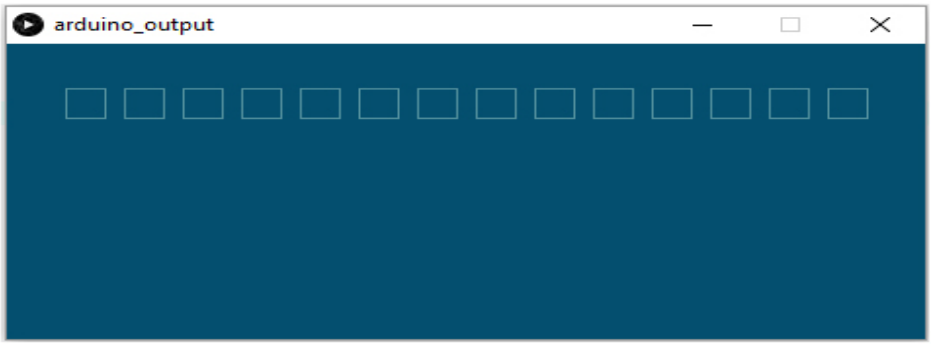



*Inserimento della libreria* Arduino (Firmata) *nell'ambiente di sviluppo* Processing.

**D)** Come primo programma per il PC, si può aprire l'esempio *File* → *Examples…* → *Contributed Libraries* → *Arduino (Firmata)* → *arduino_output*, che può attivare il LED L presente sulla scheda Arduino, connesso con il pin 13.

```
import processing.serial.*;
import cc.arduino.*;
Arduino arduino;

color off = color(4, 79, 111);
color on = color(84, 145, 158);

int[] values = { Arduino.LOW, Arduino.LOW, Arduino.LOW, Arduino.LOW,
 Arduino.LOW, Arduino.LOW, Arduino.LOW, Arduino.LOW, Arduino.LOW,
 Arduino.LOW, Arduino.LOW, Arduino.LOW, Arduino.LOW, Arduino.LOW };

void setup() {
  size(470, 200);
  // Prints out the available serial ports.
  println(Arduino.list());
  arduino = new Arduino(this, Arduino.list()[0], 57600);
   // Set the Arduino digital pins as outputs.
  for (int i = 0; i <= 13; i++)
    arduino.pinMode(i, Arduino.OUTPUT);
}

void draw() {
  background(off);
  stroke(on);

  for (int i = 0; i <= 13; i++) {
    if (values[i] == Arduino.HIGH)
      fill(on);
    else
      fill(off);

    rect(420 - i * 30, 30, 20, 20);
  }
}

void mousePressed()
{
  int pin = (450 - mouseX) / 30;

  // Toggle the pin corresponding to the clicked square.
  if (values[pin] == Arduino.LOW) {
    arduino.digitalWrite(pin, Arduino.HIGH);
    values[pin] = Arduino.HIGH;
  } else {
    arduino.digitalWrite(pin, Arduino.LOW);
    values[pin] = Arduino.LOW;
  }
}
```

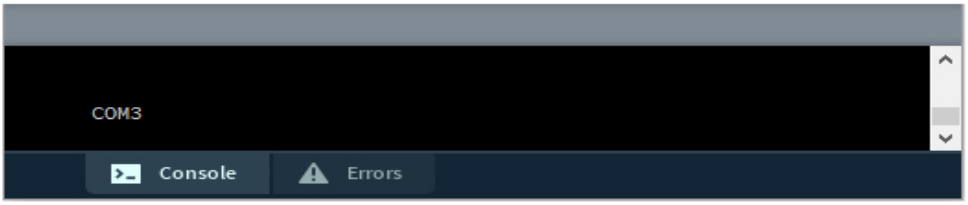
Il programma è scritto in linguaggio Java, non molto dissimile dal C, ed è composto da 3 funzioni.
La funzione *setup ()*, tra le altre cose, definisce l'ampiezza della finestra di interazione che appare sullo schermo del PC (*size(470, 200);*) e comanda Arduino a predisporre tutti i pin come uscite (*arduino.pinMode(i, Arduino.OUTPUT);*).
La funzione *draw ()*, sempre attiva, disegna e aggiorna continuamente il colore dei 13 quadrati interni alla maschera di interazione.




*Finestra su PC per il controllo delle uscite di Arduino.*

La funzione *mousePressed()* gestisce gli eventi scatenati dal click del mouse (solo se di coordinata corretta), modificando il colore del rettangolo interessato e inviando il comando ad Arduino per la modifica del livello logico dell'uscita interessata.

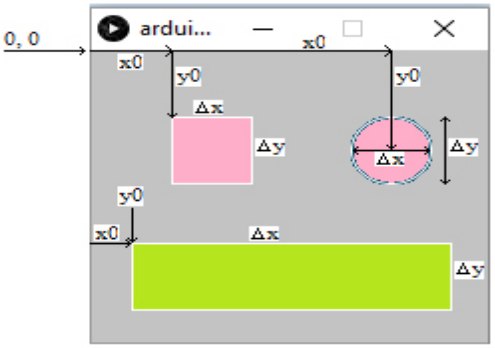
**E)** Lanciare il programma e osservare, nell'area dei messaggi riportata nella zona inferiore della finestra, la posizione della porta COM impegnata da Arduino all'interno della lista che appare, considerando "0" l'indice di posizione della prima.




*Lista delle porte COM.*

Nell'esempio riportato in figura, la porta COM3 è l'unica individuata e poiché corrisponde a quella effettivamente utilizzata per la connessione con la scheda Arduino, ha indice 0. In caso contrario, modificare di conseguenza l'indice 0 dell'istruzione *Arduino.list()[0]*, presente *nel setup ()*.

**F)** Lanciare nuovamente il programma e cliccare con il mouse sul quadrato più a sinistra della finestra che appare. Ad ogni attivazione si rovescia il livello logico del pin 13 sulla scheda Arduino, accendendo e spegnendo il LED L.




*Coordinate grafiche di* Processing.

La funzione grafica *rect (x0, y0, Δx, Δy)*, utilizzata da *draw()*, traccia un rettangolo (o un quadrato) partendo dalle coordinate x0, y0, espresse in pixel, riferite all'origine della maschera posta in alto a sinistra.
La figura tracciata è riempita con il colore presente nell'ultima fill (R, G, B), secondo le tre componenti ciascuna di valore 0 – 255.
Per tracciare un'ellisse (o un cerchio), si utilizza la funzione *ellipse (x0, y0, Δx, Δy)*, indicando le coordinate del centro e le dimensioni dei due semiassi.


Le principali istruzioni *Processing* che inviano comandi o testano gli ingressi di Arduino sono:

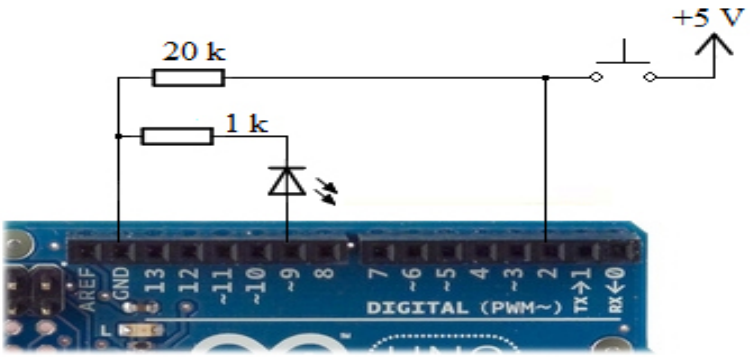
```
arduino.pinMode(pin, Arduino.OUTPUT);
arduino.digitalWrite(pin, Arduino.HIGH);
arduino.digitalWrite(pin, Arduino.LOW);
arduino.pinMode(pin, Arduino.INPUT);
if (arduino.digitalRead(pin) == Arduino.HIGH) …
arduino.analogWrite(pin, valore);
```

*Processing* è in grado di gestire anche la grafica in 3D e le animazioni. In rete si trovano numerosi esempi e tutorial. Le applicazioni possono essere esportate in formato eseguibile (*.exe*), per i diversi sistemi operativi, o come applet Java.

## 3) Possibile continuazione

Predisporre attorno ad Arduino l'hardware indicato in figura e comporre uno sketch *Processing* che dialoghi con Arduino (su cui è caricato *StandardFirmata.pde*), in modo da presentare sullo schermo del PC un quadrato, un cerchio e un rettangolo, con le seguenti funzioni:

- con il mouse sul quadrato si attiva o disattiva l'uscita 13 (LED L);
- il cerchio deve cambiare colore in base al livello logico presente sull'ingresso 2;
- scorrendo il mouse sul rettangolo si regola la luminosità del LED connesso all'uscita 9.




*Arduino con tasto di ingresso e LED di uscita.*


C. Ferrari

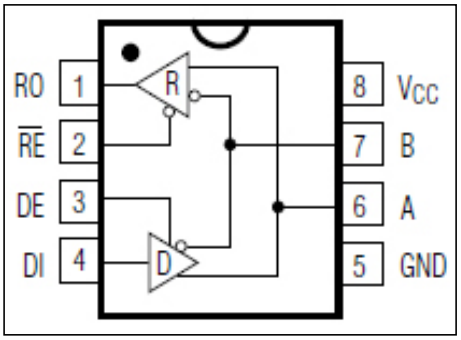
# Prove su RS 485

**Obiettivo**

Sperimentare il livello elettrico delle connessioni differenziali in RS 485, partendo sia dalla porta RS 232 di un PC, sia dalla USB.

**Conoscenze preliminari**

Un driver per RS 485, quale il driver Maxim Max 485, funziona con livelli 0-5 V.
Al suo interno, l'integrato contiene un ricetrasmettitore con abilitazioni indipendenti e di livello complementare per il trasmettitore e il ricevitore.
Se il trasmettitore non è abilitato (DE = 0), le linee A e B sono poste in alta impedenza.
Abilitando in modo permanente sia il ricevitore (RE = 0) sia il trasmettitore (DE = 1), le uscite A e B assumono un livello logico attivo complementare e, inoltre, si ha l'eco dell'informazione eventualmente trasmessa.




*Max 485.*

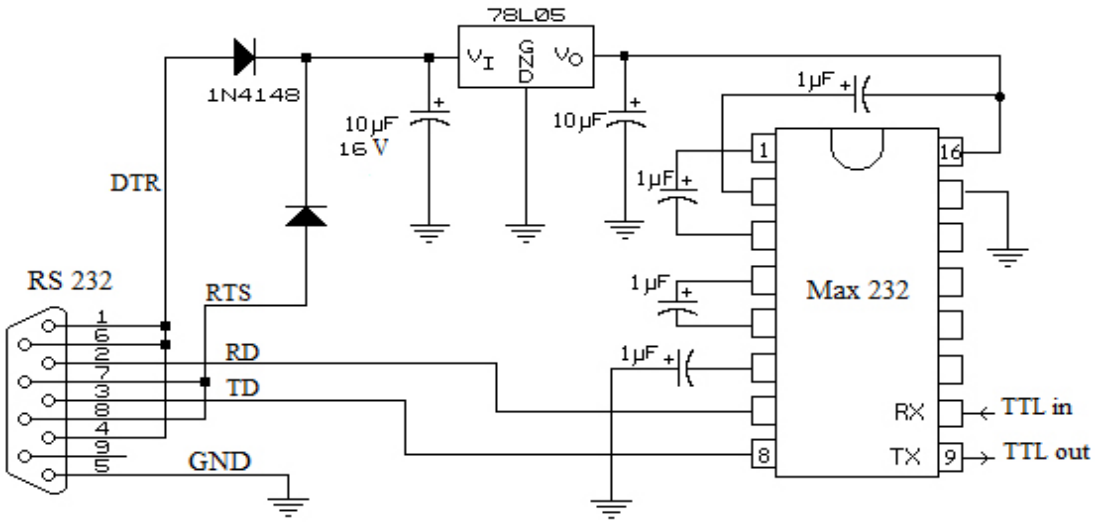
**Fasi operative**

**1) Da RS 232 a RS 485**

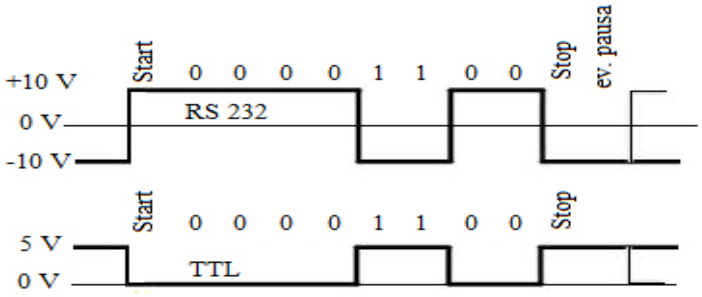
Poiché il driver Max 485 funziona con livelli 0-5 V, mentre RS 232 lavora con ±10 V, bisogna prima riportare i livelli RS 232 a 0-5 V e con questi azionare il driver 485.

**A)** Da RS 232 a 5 V

Per portare i livelli ±10 V di RS 232 a livelli 0-5 V, si utilizza l'integrato convertitore MAX 232 (Maxim), alimentato a 5 V, che autocostruisce le due tensioni bilanciate (+10 V e −10 V), a partire dal +5 V.
Si procede dunque a:
- montare il circuito in figura;
- trasmettere con Hyperterminal il carattere "0" e verificare con l'oscilloscopio sul pin 9 del Max 232 i livelli TTL corrispondenti, con lo stato di mark a +5 V.




*Convertitore di livello da RS 232 a TTL.*

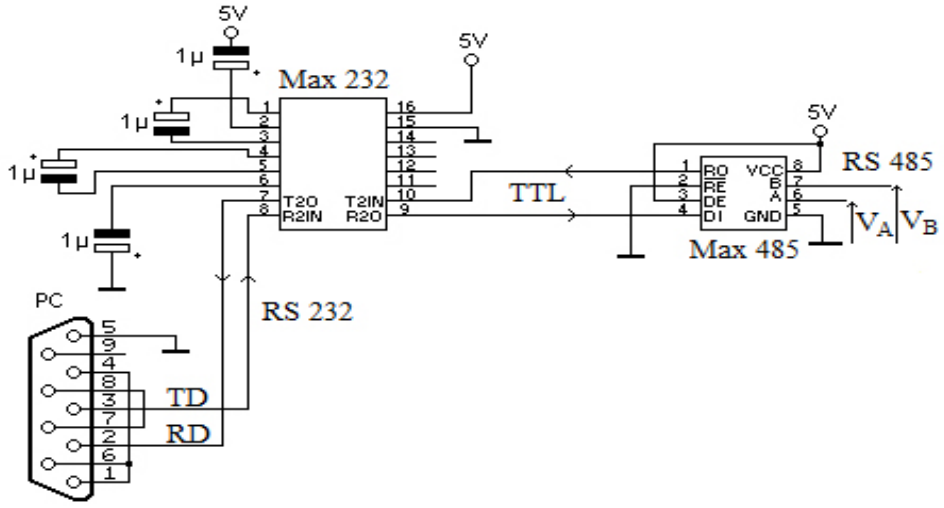



*Livelli RS 232 e TTL.*

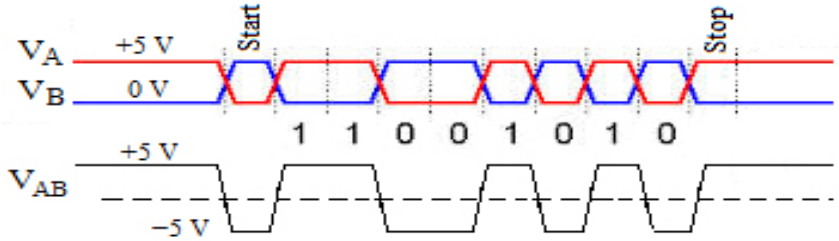
**B)** Da 5 V a RS 485

Per passare dai livelli single ended TTL al segnale differenziale RS 485 è necessario:
- completare il circuito in figura, derivando il 5 V necessario per l'alimentazione del Max 485 dai segnali DTR e RTS del connettore RS 232;
- rilevare, mediante un oscilloscopio, con una sonda sul segnale A (pin 6) e una sul segnale B (pin 7) dell'integrato Max 485, la differenza di tensione tra le due uscite, utilizzando la funzione A-B.

Poiché il trasmettitore è sempre attivo, la linea a riposo risulta polarizzata in stato di mark con $V_{AB}$ = +5 V, senza bisogno d'altro.




*Convertitore RS 232 - RS 485.*

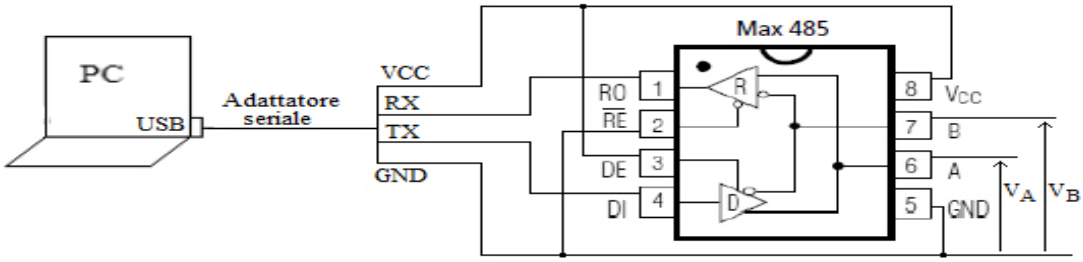



*Segnali per RS 485.*

**2) Da USB a RS 485**

Per passare da USB a RS 485, bisogna prima riportare il formato USB in seriale 0-5 V e con questo azionare il driver 485.


**A)** Utilizzare un cavetto USB to serial, o un adattatore USB-TTL, per convertire da USB a seriale.
**B)** Completare il circuito in figura con il driver Max 485, prelevando il 5 V di alimentazione dell'integrato dall'apposita uscita del convertitore utilizzato.
**C)** Trasmettere con Hyperterminal il carattere "0" e verificare con l'oscilloscopio sul terminale TX dell'adattatore USB i livelli 0-5 V corrispondenti, con lo stato di mark a +5 V.
**D)** Rilevare, mediante un oscilloscopio, con una sonda sul segnale A (pin 6) e una sul segnale B (pin 7) dell'integrato Max 485, la differenza di tensione tra le due uscite, utilizzando la funzione A-B.



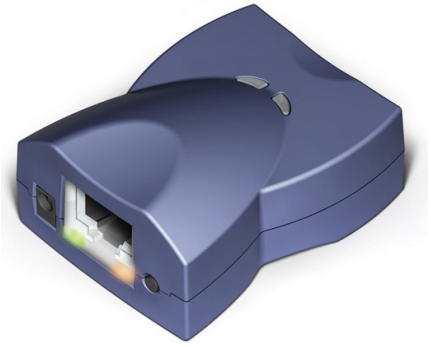

*Convertitore USB – RS 485.*

C. Ferrari


# Serial device server

**Obiettivo**
Collaudare l'attivazione di un socket TCP/IP utilizzando un convertitore Ethernet seriale.

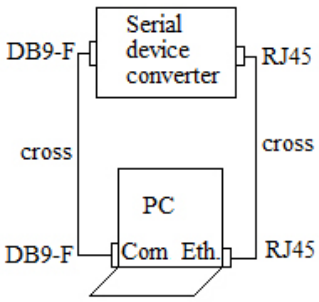
**Conoscenze preliminari**
Un convertitore Ethernet seriale è un dispositivo con funzionalità server che presenta una porta Ethernet su un lato e una seriale asincrona sull'altro. Nel caso del *serial device server* Tibbo DS203, la porta seriale è standard RS 232.
Il dispositivo va alimentato dall'esterno con una tensione continua compresa tra 9 e 25 V.
Per poter essere utilizzato, il dispositivo va riconosciuto e configurato, ponendolo in comunicazione diretta con un applicativo sul PC, mediante un cavo incrociato su uno dei port, e utilizzando il software a corredo, che comprende un insieme di comandi di monitor attivabili da seriale al reset.



*Serial device server Tibbo DS203.*

**Fasi operative**

**1) Configurazione del dispositivo**
Collegare il dispositivo al PC da entrambi i lati, utilizzando due cavi incrociati (cross), sia per la RS 232, sia per Ethernet. Un cavo Ethernet cross incrocia le coppie 1-2 con 3-6, cioè il doppino di trasmissione con quello di ricezione, ed è da utilizzare solo per i collegamenti diretti tra due dispositivi finali, senza interposizione di hub o altro.




*Connessioni cross.*

**A)** Configurazione dalla porta RS232
Se si tratta di una prima esperienza e non si conosce l'indirizzo IP attuale del dispositivo, si può operare dalla seriale RS 232, nel modo seguente:
- aprire la COM del PC mediante una finestra Hyperterminal con i parametri indicati nel manuale del dispositivo (38400, 8, N, 1, Hw);
- dopo aver premuto il tasto di reset del dispositivo, inviare il comando di richiesta dell'IP attuale (GET IP), digitando la stringa di 5 caratteri ASCII

Ctrl-B G I P $C_R$

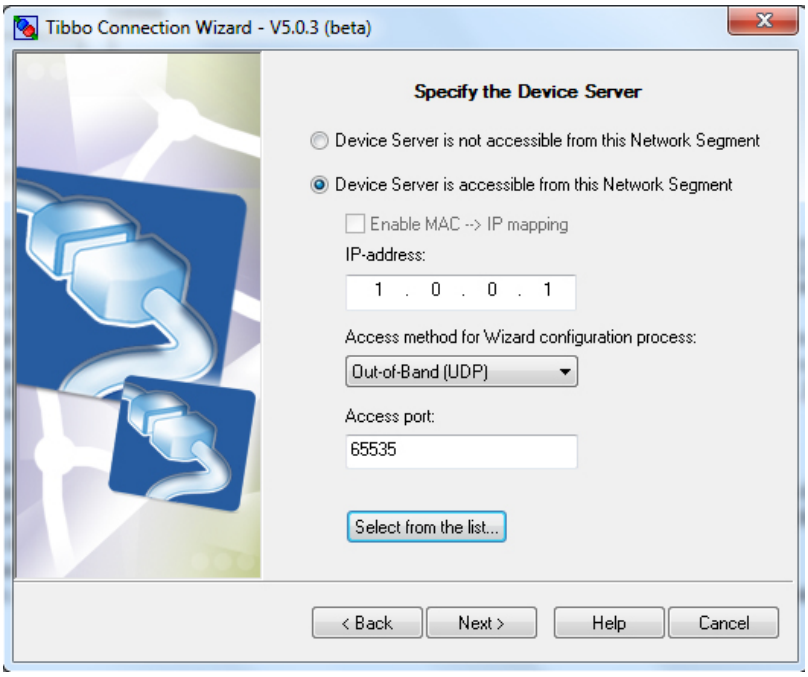
terminata con il carattere di invio ($C_R$ = torna a capo = 0DH), in risposta alla quale il sistema fornisce il proprio indirizzo, per esempio 1.0.0.1;
- per uscire dalla modalità comandi, inviare

Ctrl-B O $C_R$

oppure spegnere e riaccendere il dispositivo.

**B)** Configurazione dalla porta Ethernet
Per configurare il dispositivo, utilizzando il software da Ethernet, è necessario:
- predisporre l'indirizzo della scheda locale del PC per il medesimo segmento di rete del dispositivo, entrando nella configurazione delle proprietà relative al protocollo TCP/IPv4 della scheda (in *Apri Centro connessioni di rete* → *Modifica impostazioni scheda* → *Connessione alla rete locale LAN* → *Proprietà* → *Rete* → *Protocollo Internet versione 4*) e impostando un indirizzo fisso compatibile, quale, per esempio 1.0.0.2, con maschera 255.255.0.0;


- richiamare il software di configurazione in dotazione *Tibbo Connection Wizard* e cliccare su *Select from the list*, iniziando la ricerca dei dispositivi collegati che individua il serial server in esame;
- una volta selezionato il dispositivo, procedere indicando altri parametri, quali chi è il primo a inviare i dati (l'applicazione sul PC) e il protocollo utilizzato (TCP/IP sul port 1001).

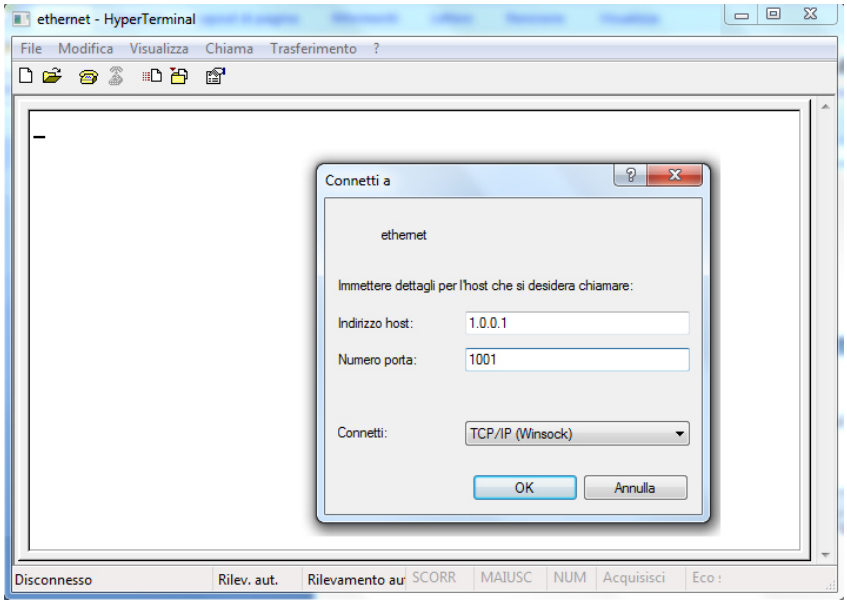



*Tibbo Connection Wizard.*

## 2) Collaudo della connessione

**A)** Utilizzando una seconda finestra Hyperterminal, aprire un canale di collegamento (socket) con il dispositivo server, indicandone i parametri: l'indirizzo IP 1.0.0.1, il numero del port 1001, e il protocollo TCP/IP (Winsock).

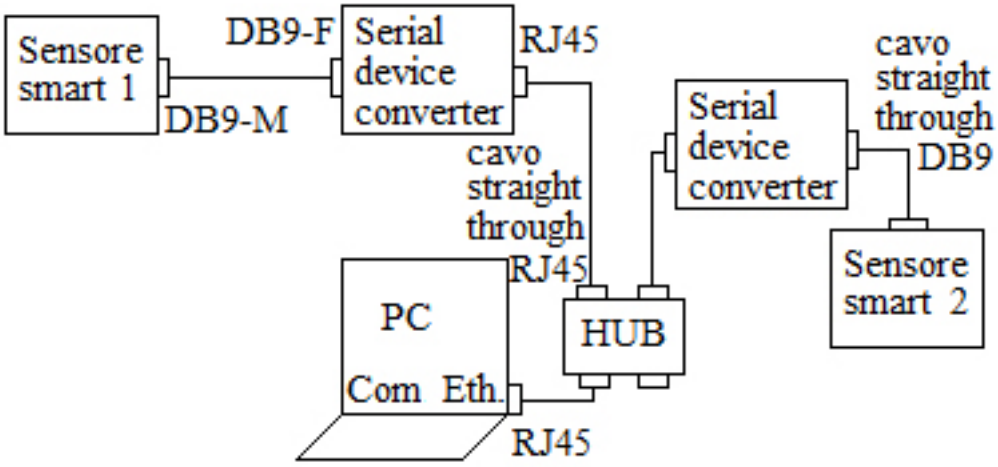
**B)** Con entrambe le finestre attive, se il dispositivo è funzionante, ciò che si scrive in una finestra deve comparire sull'altra (e viceversa).




*Seconda finestra Hyperterminal.*


### 3) Rete di sensori

Assegnando a diversi serial converter indirizzi IP differenti, appartenenti al medesimo segmento di rete del PC, è possibile comporre una rete di sensori in RS 232 utilizzando hub o switch di diramazione.
I cavi da utilizzare sono tutti diritti (*straight through*), sia i seriali, con connettori maschio-femmina, sia gli Ethernet.
Per attivare la rete, l'applicazione sul PC deve prima aprire un canale TCP/IP (socket) con ciascuno dei serial server, indicando per ognuno l'indirizzo IP (univoco) e il numero del port.

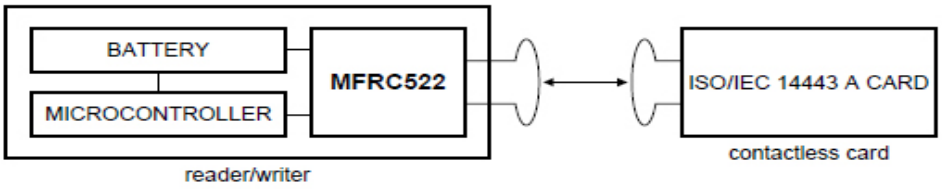



*Rete di sensori smart in RS 232.*


# Lettore RFID MFRC522 con Arduino

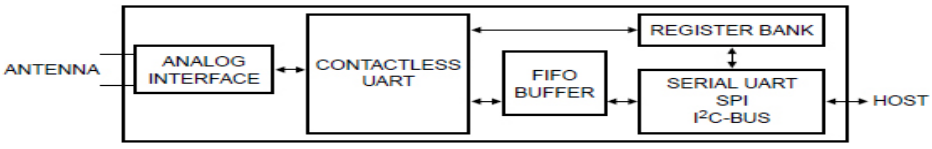
**Obiettivo**
Sperimentare il funzionamento di un dispositivo di identificazione in tecnologia RFID a 13,56 MHz.

**Conoscenze preliminari**
Il sistema di identificazione utilizzato in questa esercitazione si basa sul dispositivo MFRC522, un integrato RFID (*Radio Frequency IDentification*) specifico per la realizzazione di una stazione base di lettura/scrittura di transponder passivi a 13,56 MHz.
La stazione base emette un campo a 13,56 MHz alla ricerca della presenza di transponder nello spazio circostante e il dispositivo presente risponde modulando il campo in base al proprio codice interno.



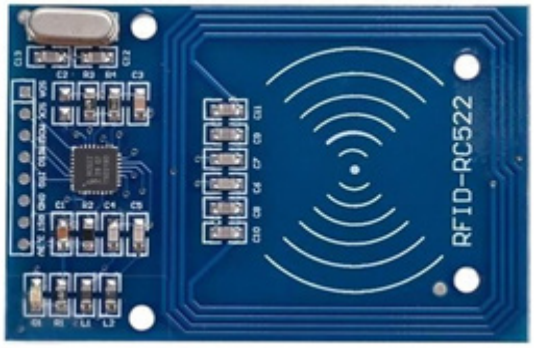

*Sistema di lettura/scrittura RFID con MFRC522.*

Il dispositivo MFRC522 contiene sia l'elettronica di trasmissione, con i pin per la connessione diretta dell'antenna, sia la sezione di ricezione, con demodulazione e decodifica dello standard ISO/IEC 14443 A, compreso il rilevamento di eventuali errori (di parità ecc.). Internamente dispone di un buffer FIFO di 64 byte RAM per i messaggi e comunica esternamente mediante una delle tre interfacce seriali disponibili: UART, SPI, I2C.




*Schema a blocchi semplificato dell'integrato MFRC522.*

Sono reperibili in commercio moduli MFRC522 completi dell'antenna su circuito stampato, con un connettore a 8 pin per l'alimentazione (3,3 V, 100 mA) e la comunicazione.
Come connessione seriale il modulo utilizza lo standard SPI (*Serial Peripheral Interface*) che impegna una linea per il clock (SCK), due linee per i dati (MOSI e MISO) e una linea per l'abilitazione (SDA).




*Modulo MFRC522.*

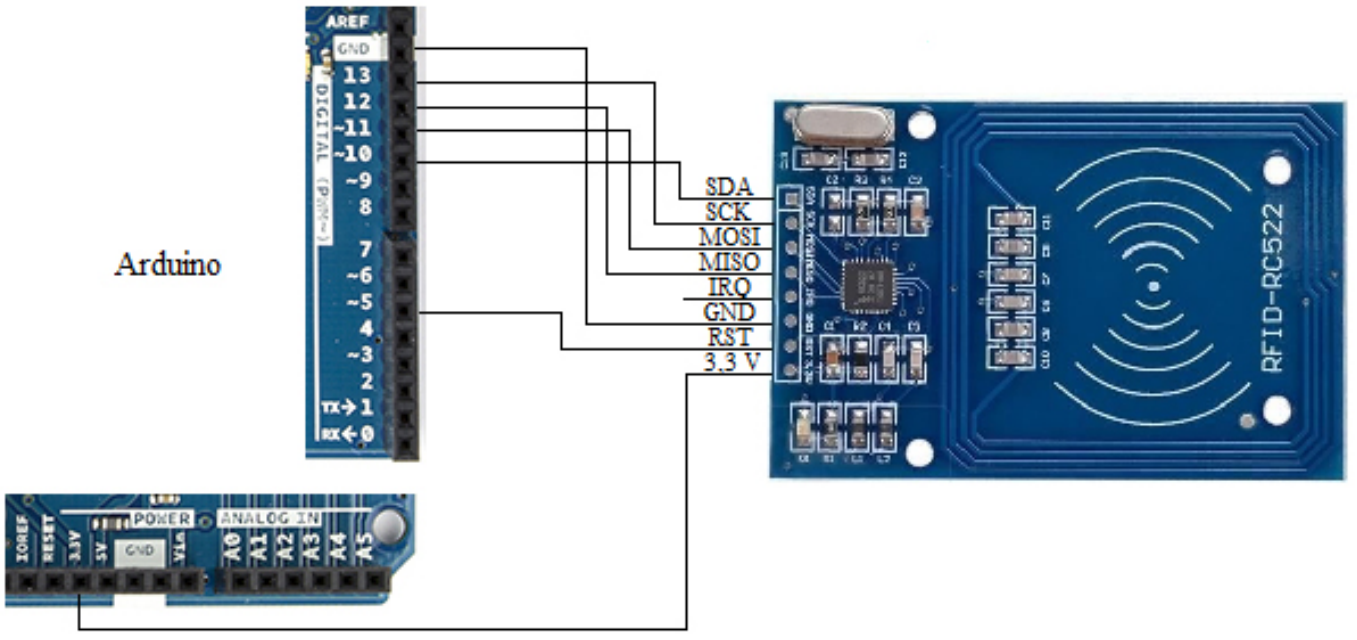

**Fasi operative**

Per la comunicazione del modulo MFRC522 con Arduino, può bastare la libreria SPI.h disponibile all'interno dell'ambiente di programmazione, ma, considerata la complessità dei frame da elaborare e scambiare, conviene iniziare includendo nel progetto una delle librerie complete di decodifica disponibili nel web.
Il programma che segue, nello specifico, include la libreria AddicoreRFID.h.

**1)** Reperire la libreria AddicoreRFID.h sul sito www.addicore.com e includerla nella cartella delle librerie di Arduino (C:/Programmi (x86)/Arduino/libraries), tramite il comando *Sketch* → *Inclusione librerie* → *Aggiungi una libreria da file.ZIP*.
**2)** Collegare il modulo con la scheda Arduino come indicato in figura e in tabella 1. I pin 11, 12 e 13 sono vincolati all'interfaccia SPI di Arduino, mentre i restanti 10 (*Enable*) e 5 (RST) sono quelli di default della libreria Addicore.

**Tab. 1 – CONNESSIONI TRA MFRC522 E ARDUINO**

| Modulo MFRC522 | Arduino |
|---|---|
| 1 - SDA | Digital 10 |
| 2 - SCK | Digital 13 |
| 3 - MOSI | Digital 11 |
| 4 - MISO | Digital 12 |
| 5 - IRQ | Non connesso |
| 6 - GND | GND |
| 7 - RST | Digital 5 |
| 8 - 3,3 V | 3.3 V |



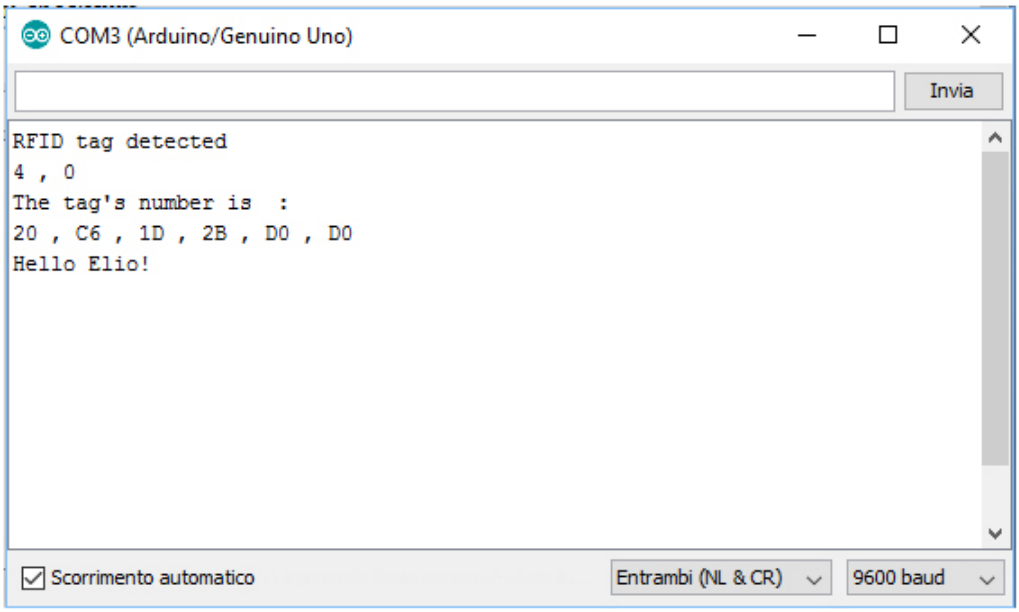

*Connessione del modulo MFRC522 con Arduino.*


**3)** Caricare e collaudare il programma che segue, avvicinando uno dei transponder passivi in dotazione.

```
#include <AddicoreRFID.h>
#include <SPI.h>
AddicoreRFID myRFID; // struttura per il controllo del modulo RFID

#define Enable  10  //SDA = Enable SPI (default)
#define RST   5     //default

void setup() {
   Serial.begin(9600);
   SPI.begin();
   pinMode(Enable,OUTPUT);
   digitalWrite(Enable, LOW); // Attiva il lettore RC522
   pinMode(RST,OUTPUT);
   digitalWrite(RST, HIGH);
   myRFID.AddicoreRFID_Init();
}

void loop()
{
 unsigned char status, checksum1,str[5]; // serial number + checksum
 unsigned int i;

  //Ricerca dei tag, ne ritorna il tipo
  status = myRFID.AddicoreRFID_Request(PICC_REQIDL, str);
  if (status == MI_OK)
  {
     Serial.println("RFID tag detected");
     Serial.print(str[0],HEX);
     Serial.print(" , ");
     Serial.print(str[1],HEX);
     Serial.println(" ");
  }

  //Anti-collision, ritorna il serial number su 4 byte
  status = myRFID.AddicoreRFID_Anticoll(str);
  if (status == MI_OK)
  {
     checksum1 = str[0] ^ str[1] ^ str[2] ^ str[3];
     Serial.println("The tag's number is  : ");
     for(i=0;i<5;i++){
          Serial.print(str[i], HEX);
          Serial.print(" , ");
     }
     Serial.println(checksum1,HEX);
     // Per semplicità controllo solo il primo byte
     if(str[0] == 0x15) Serial.print("Hello Carlo!\n");
     //da cambiare in base ai tag disponibili
      else if(str[0] == 0x20) Serial.print("Hello Elio!\n");
     Serial.println();
     delay(1000);
  }
}
```

Avvicinando un transponder, il monitor seriale ne riporta il tipo, su 2 byte, e successivamente i 4 byte del codice letto (in esadecimale), seguiti dal byte di checksum acquisito e dal checksum calcolato dal programma.

```
checksum1 = str[0] ^ str[1] ^ str[2] ^ str[3];
```




*Monitor seriale.*

Il checksum vale il risultato dell'operazione di exclusive-or bit a bit effettuata tra i 4 byte del codice.
Il riconoscimento dei 4 byte può essere utilizzato, per esempio, per azionare un relè che apre una elettroserratura.

**3)** Per emettere un saluto di benvenuto, il programma proposto confronta solo il primo byte del numero seriale. Adattare quindi il programma ai badge a disposizione, modificando il valore di confronto secondo i codici identificativi dei transponder disponibili.

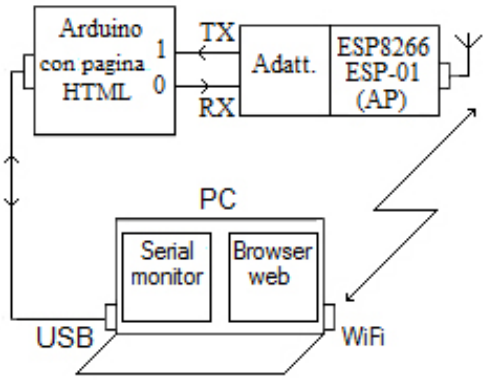
C. Ferrari


# Arduino web server WiFi

**Obiettivo**
Realizzare su Arduino un piccolo web server raggiungibile da PC mediante connessione WiFi.

**Conoscenze preliminari**
Per realizzare e testare un web server WiFi sono richieste due cose:
- un server dotato di access point WiFi, in grado di restituire una pagina in linguaggio HTML;
- un client con browser, che attiva una ricerca *http://www...*, chiedendo una connessione temporanea con il server, con protocollo http, sul port 80.

In questa esperienza il server è demandato a Arduino, mentre il client è un PC.
Prima di procedere con questa esperienza si consiglia di effettuare le *Arduino web server* (pag. 295 del libro) e *WiFi con Arduino* (pag. 311 del libro), della quale può essere considerata una continuazione.




*Modulo connesso al serial monitor.*

**Fasi operative**

**1) Predisposizione dell'access point su Arduino**
Con le modalità specificate nell'esperienza *WiFi con Arduino*, seguire i passi indicati.
**A)** Realizzare le connessioni elettriche tra ESP-01 e Arduino.
**B)** Porre in alta impedenza i due pin dell'UART di Arduino, caricando lo sketch fornito in precedenza.
**C)** Configurare il monitor seriale per 115.200 baud e l'inserimento automatico di CR e NL.
**D)** Connettere il modulo ESP8266 con i pin 0 e 1 di Arduino.
**E)** Configurare il modulo ESP8266 come AP-Server sul port 80, con i comandi

```
AT+CWMODE=2
AT+CIPMUX=1
AT+CIPSERVER=1,80
```

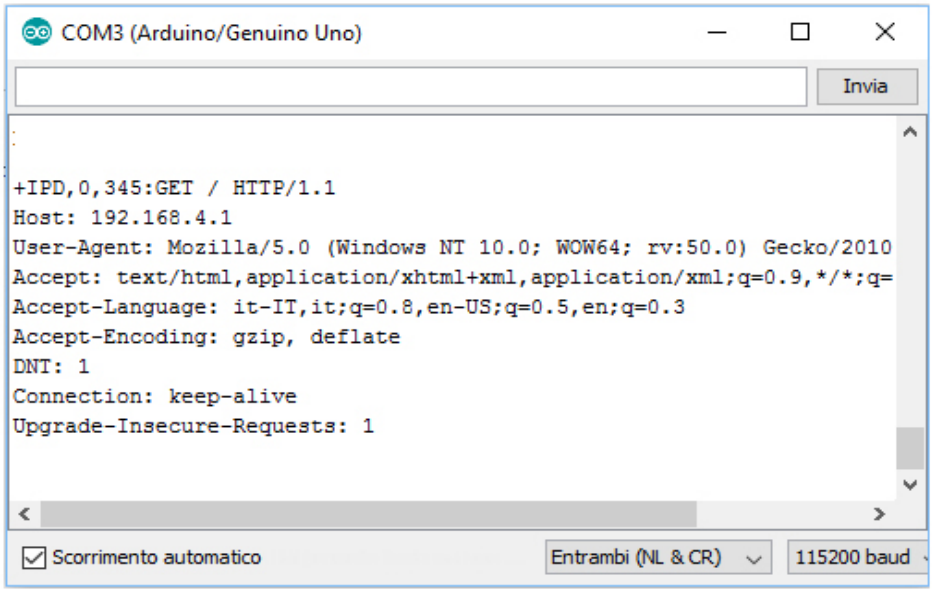
**F)** Verificare che il modulo abbia assunto l'indirizzo di default della modalità AP server 192.168.4.1, mediante il comando

```
AT+CIFSR
```

**G)** Da PC, analizzare l'elenco delle reti wireless captate dal PC e selezionare l'access point Al-Thinker.
**H)** Richiedere e ottenere la connessione.

**2) Analisi dei messaggi scambiati con il browser web**
Prima di procedere alla composizione del web server, è bene sperimentare, operando da PC, la struttura dei messaggi scambiati tra il browser del client e il server durante una connessione.
**A)** Aprire, mediante un browser web, una connessione http con l'indirizzo 192.168.4.1. Sul monitor di Arduino appaiono, così, alcune interessanti informazioni, come mostrato in figura. Dopo i 5 caratteri "`+IPD,`", il numero della connessione ("`0`"), il numero dei


caratteri trasferiti (“345”) e il carattere “:”, si ritrova il solito messaggio di apertura che inizia con i 5 caratteri “GET/” e altre informazioni, quali nome e versione del browser, il sistema operativo utilizzato, il tipo di file atteso, il linguaggio preferito, ecc.

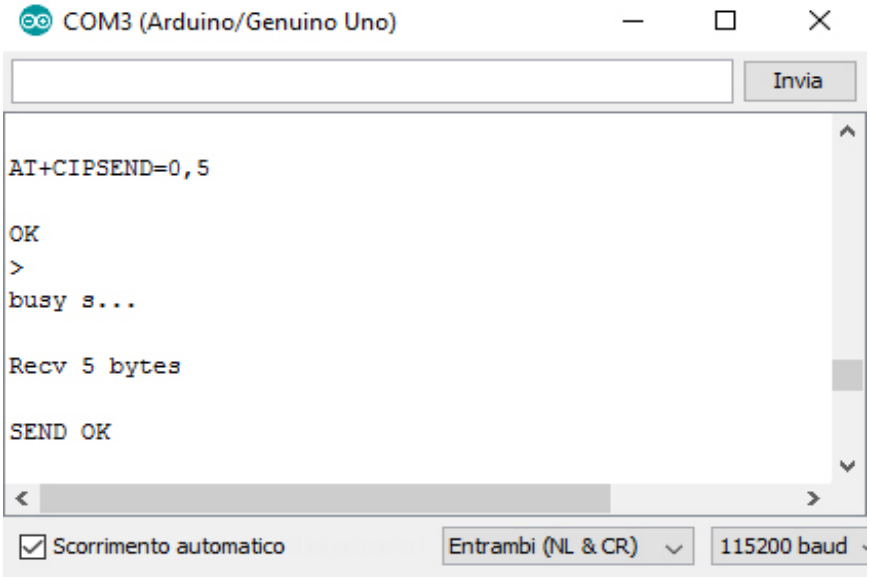



*Informazioni relative al browser.*

**B)** Inviare dal monitor seriale al browser del PC la scritta “ciaoo” di 5 caratteri, utilizzando prima il comando

```
AT+CIPSEND=0,5
```

e, all’apparire del carattere >, la stringa stessa *“ciaoo”*.

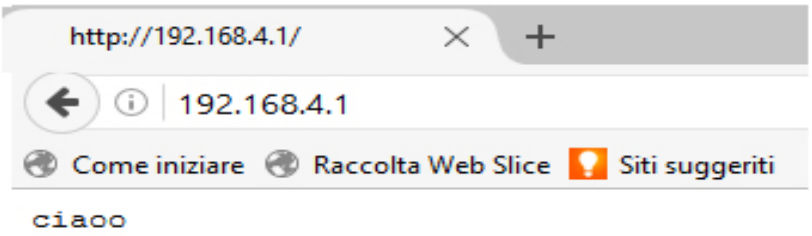



*Invio al browser di 5 caratteri.*

**C)** Sebbene il monitor avvisi che l’invio è andato a buon fine, sulla finestra del browser non appare nulla, finché (come richiesto dal protocollo) non si chiude il canale con il comando

```
AT+CIPCLOSE=0
```

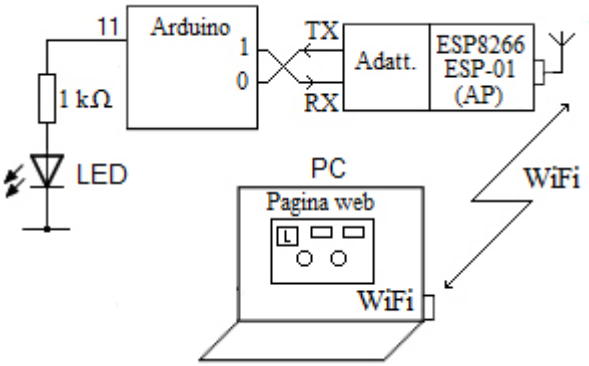
dove “0” indica appunto il numero del canale da chiudere.


*Messaggio visualizzato al termine nella finestra del browser web.*

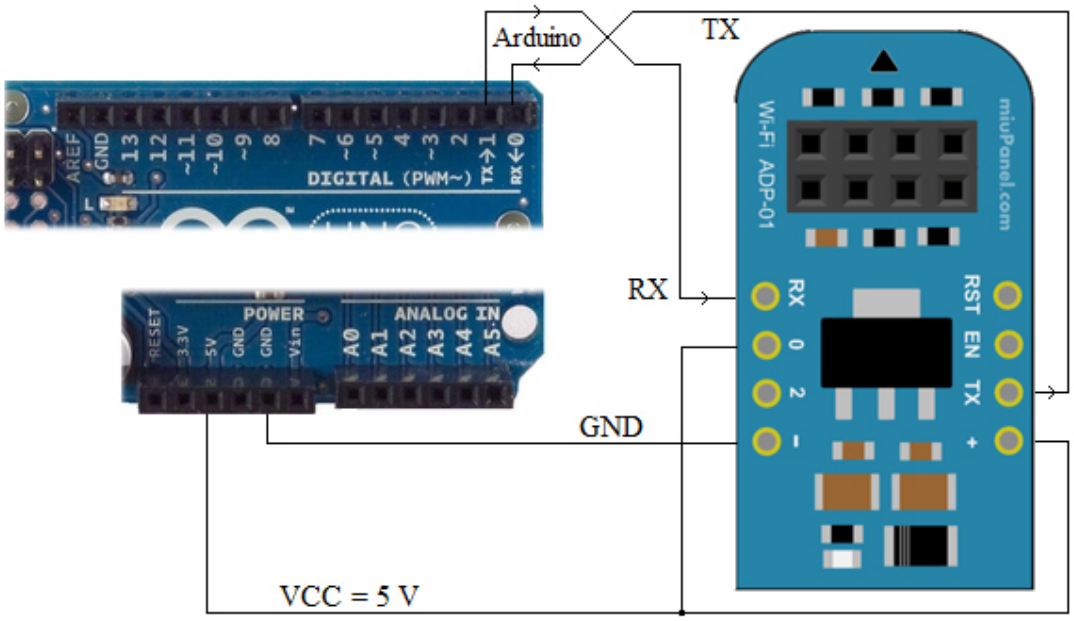
### 3) Invio comandi da una pagina HTML residente sul PC

Per controllare Arduino mediante una pagina web con connessione WiFi, oltre a configurare il modulo ESP8266 in modalità operativa AP e avviare il servizio server sul port 80, serve un file HTML che, interpretato dal browser del PC, disegni sullo schermo una pagina grafica nella quale, per semplificare, premendo un bottone, si modifica lo stato di un'uscita di Arduino.




*Comandi da pagina web.*

**A)** Programmare su Arduino.
Questa volta, anziché il monitor seriale del PC, è il programma su Arduino che deve interagire in tempo reale con il modulo WiFi.
- Incrociare i segnali di comunicazione tra modulo e Arduino, come indicato in figura.

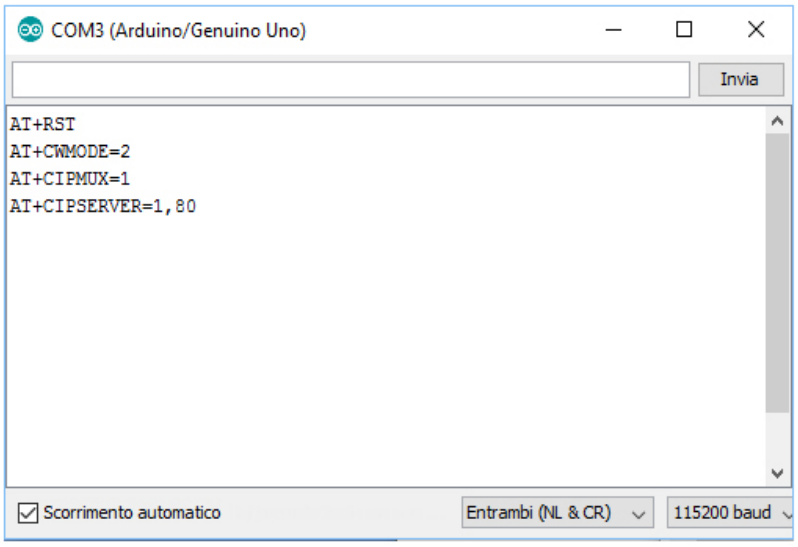



*Connessioni seriali per funzionamento web server.*

- Caricare in Arduino il programma che segue, sollevando il filo TX del modulo durante il caricamento, per non confliggere con la TXD del PC. Lo sketch attende di ricevere dal client comandi contenenti la scritta "pin=11", per rovesciare lo stato logico dell'uscita digitale 11.


```
/* WiFi_html */
void setup()
{
  digitalWrite(11,LOW);pinMode(11,OUTPUT);
  Serial.begin(115200); // comunicazione con ESP8266
  Serial.print("AT+RST\r\n");
  delay(1000); // resetta modulo
  Serial.print("AT+CWMODE=2\r\n");
  delay(500); // configura come access point
  Serial.print("AT+CIPMUX=1\r\n");
  delay(500); // configura per connessioni multiple
  Serial.print("AT+CIPSERVER=1,80\r\n");
  delay(500); // avvia server sulla porta 80
}
void loop()
{
  if(Serial.available()) // controlla se il modulo sta comunicando
  {
    if(Serial.find("+IPD,")) //Cerca nella stringa il testo "+IPD," inviato
                             premendo il bottone nella pagina html
    {
      delay(20);
      int connectionId = Serial.read()-48;
      //al connection id va sottratto 48 ("0" ASCII)
      Serial.find("pin="); // adesso cerca "pin="
      int pinNumber = (Serial.read()-48)*10;
      //il primo numero sono le decine
      pinNumber += (Serial.read()-48);
      //il secondo numero (unità) va sommato al primo
      digitalWrite(pinNumber, !digitalRead(pinNumber));
      // rovescia lo stato del pin
      // compone il comando di chiusura da inviare al modulo
      String closeCommand = "AT+CIPCLOSE=";
      closeCommand+=connectionId; // attacca il connection id
      closeCommand+="\r\n";
      Serial.print(closeCommand);
      delay(20);// attesa invio del comando di chiusura
    }
  }
}
```

- Una volta caricato il programma su Arduino (tenendo staccato il TX del modulo), ricollegare TX e aprire la finestra di monitor. Resettando la scheda, Arduino invia i comandi di configurazione al modulo WiFi e resta in attesa.




*Comandi che Arduino invia al modulo WiFi.*


**B)** File HTML per il PC

Il file HTML che, interpretato dal browser, comanda da PC (via WiFi) l'uscita 11 di Arduino è riportato di seguito.

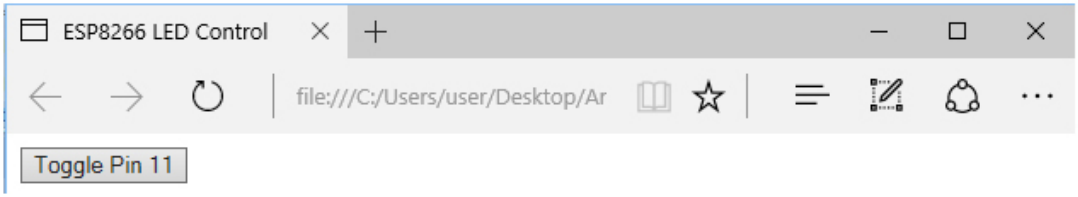
```
<html>
        <head>
                <title>ESP8266 LED Control</title>
        </head>
        <body>
<!-- in the <button> tags below the ID attribute is the value sent to the
arduino -->
        <button id="11" class="led">Toggle Pin 11</button>
<!-- button for pin 11 -->
        <script src="jquery.min.js"></script>
        <script type="text/javascript">
                $(document).ready(function(){
                        $(".led").click(function(){
                                var p = $(this).attr('id');
// get id value (i.e. pin13, pin12, or pin11)
// invia la richiesta HTTP GET all'IP address con il parametro "pin" di
valore "p"
                                $.get("http://192.168.4.1:80/", {pin:p});
// execute get request
                        });
                });
        </script>
        </body>
</html>
```

Il codice HTML utilizzato usa difatti la libreria Javascript JQuery, richiamata con l'istruzione:

```
<script src="jquery.min.js"></script>
```

Occorre quindi seguire i passi indicati.

- Copiare il codice riportato in un documento, mediante Notepad o qualsiasi altro text editor, e salvarlo in formato testo in una cartella (per esempio, *control.txt*).
- Rinominare il file, modificandone l'estensione in ".html" (per esempio, *control.html*).
- Importare nella medesima cartella il file Java *jquery.min*, copiandolo da http://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js
- Richiedere e ottenere dalla rete wireless del PC la connessione con l'access point Al-Thinker.
- Fare doppio click sul file *control.html*. Compare la finestra mostrata in figura.




*Pagina web visualizzata.*

- Cliccando sul bottone "Toggle Pin 11", il programma invia una GET request al modulo ESP8266, mediante l'istruzione

```
$.get("http://192.168.4.1:80/", {pin:p});
```

indicandone l'indirizzo IP e il port (80), con il numero 11 all'interno della variabile *p*.


- Tramite la seriale, Arduino riceve una stringa di comunicazione del tipo

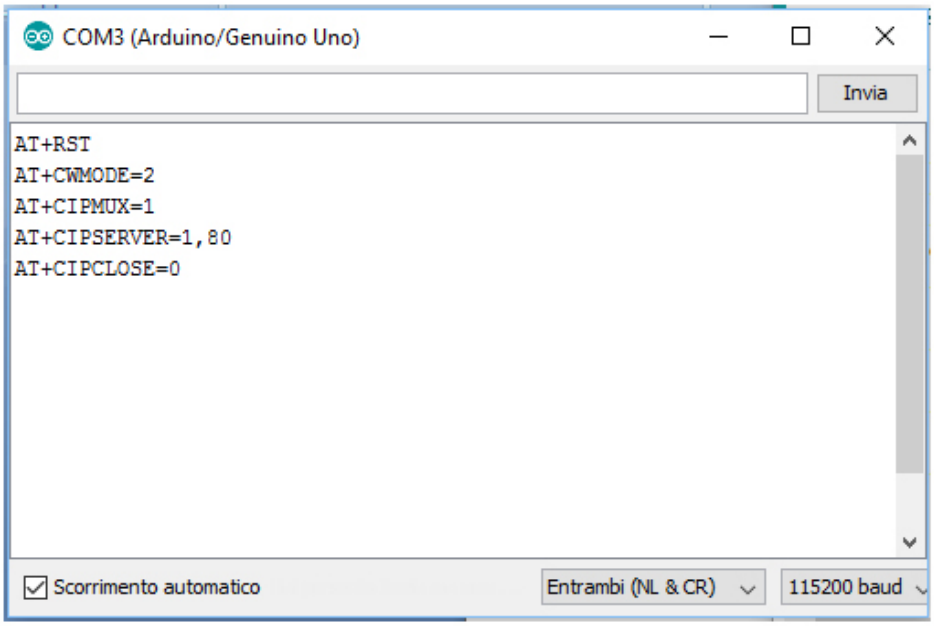
```
+IPD,0,345:GET /?pin=11 HTTP/1.1
Host: 192.168.4.1
Connection: keep-alive
Accept:text/html,…
```

**C)** Funzionamento del programma su Arduino
Al sopraggiungere di messaggi dalla linea seriale [istruzione *if(Serial.available())*], Arduino identifica i comandi da elaborare controllando la presenza dello spezzone "+IPD," nel buffer ricevuto, mediante la funzione *Serial.find* [istruzione *if(Serial.find("+IPD,"))*].
Se ciò avviene, il codice legge il carattere successivo, corrispondente all'ID della connessione ("0" nell'esempio riportato), necessario per la corretta chiusura successiva della connessione stessa (richieste simultanee hanno ID differenti) [istruzione *int connectionId = Serial.read()-48;*].
Successivamente il codice scorre la stringa alla ricerca dello spezzone "pin=" e acquisisce il numero del pin interessato, prima le decine e poi le unità.

```
Serial.find("pin=");
int pinNumber = (Serial.read()-48)*10;
    pinNumber += (Serial.read()-48);
```

Al termine, rovescia il livello logico del pin indicato [istruzione *digitalWrite(pinNumber, !digitalRead(pinNumber));*] e chiude la connessione.

```
String closeCommand = "AT+CIPCLOSE=";
      closeCommand+=connectionId; // attacca il connection id
      closeCommand+="\r\n";
      Serial.print(closeCommand
```




*Invio del comando di chiusura.*

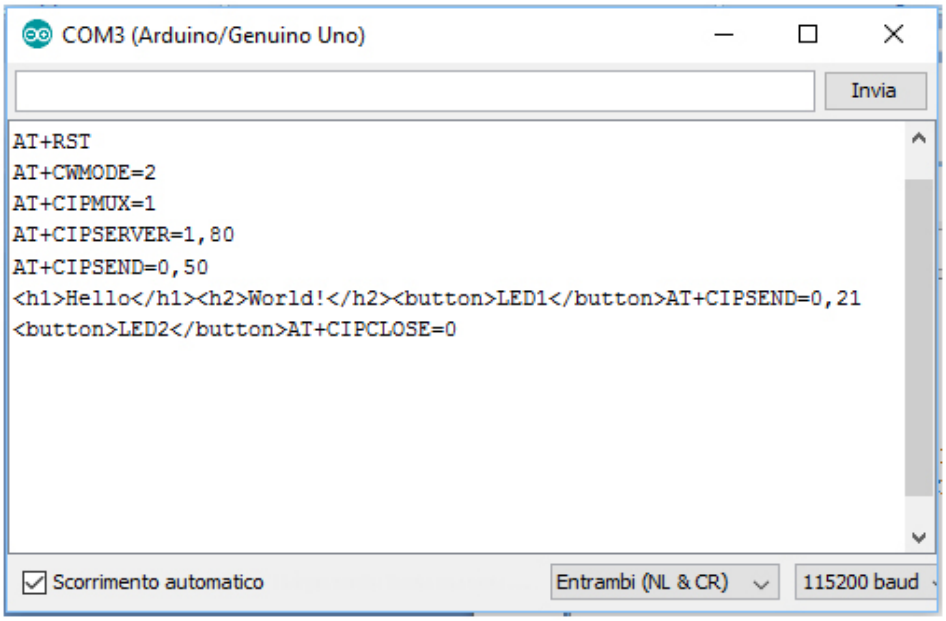
**D)** Modificare lo sketch di Arduino e aggiungere altri due bottoni nella pagina HTML, in modo da controllare anche le uscite 12 e 13.

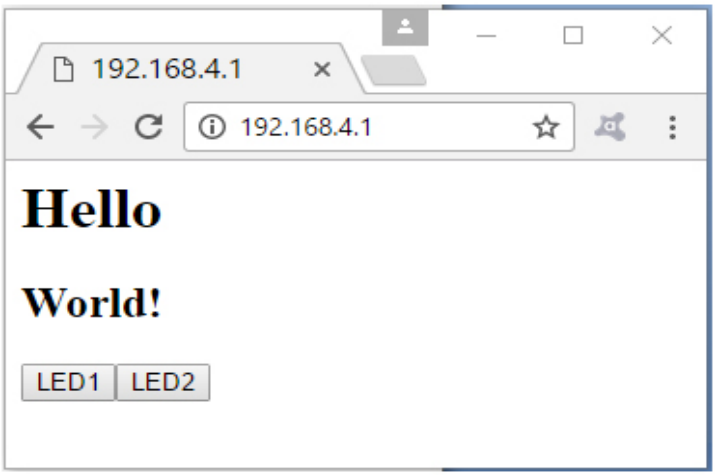

## 4) Pagina HTML residente su Arduino

Un server web è un dispositivo sempre in attesa di una richiesta da parte di un client, la serve e la chiude ogni volta. Solitamente, è il dispositivo web server (Arduino) a possedere una pagina HTML, da spedire all'indietro al client che chiede la connessione http, con all'interno le possibilità di interazione offerte. Poiché in questo caso è interposta una connessione WiFi, Arduino deve gestire l'invio della pagina rispettando l'ordine previsto dal protocollo seriale del modulo ESP8266, ovvero invia il comando AT+CIPSEND=c,n con "c" l'identificativo del canale aperto e "n" il numero dei caratteri della pagina da trasferire. Quando appare il simbolo ">" (o con opportuno ritardo) inviare la pagina stessa. Nello sketch che segue, l'invio di una pagina è delegato alla funzione *Send_webPage ()*, utilizzando il "connectionId" letto all'interno del messaggio ricevuto dal browser subito dopo la scritta "+IPD,".

```
/* Arduino WiFi server web */
int connectionId;
String webPage;

void setup() {
  Serial.begin(115200); // comunicazione con ESP8266
  Serial.print("AT+RST\r\n");
  delay(1000); // resetta modulo
  Serial.print("AT+CWMODE=2\r\n");
  delay(500); // configura come access point
  Serial.print("AT+CIPMUX=1\r\n");
  delay(500); // configura per connessioni multiple
  Serial.print("AT+CIPSERVER=1,80\r\n");
  delay(500); // avvia server sulla porta 80 }

void loop() {
  if(Serial.available()) //controlla se il modulo sta comunicando
  {
    if(Serial.find("+IPD,"))
    {
      delay(20);
      connectionId = Serial.read()-48;
      webPage = "<h1>Hello</h1><h2>World!</h2><button>LED1</button>";
      Send_webPage ();
      webPage="<button>LED2</button>";
      Send_webPage ();
      Send_closeCommand();
    } } }

void Send_webPage (){
  String sendCommand = "AT+CIPSEND=";
  sendCommand += connectionId;
  sendCommand += ",";
  sendCommand +=webPage.length();
  sendCommand +="\r\n";
  Serial.print(sendCommand);
  delay(500);
  Serial.print(webPage);
  delay(500);    }

void Send_closeCommand (){
  String closeCommand = "AT+CIPCLOSE=";
  closeCommand+=connectionId;
  closeCommand+="\r\n";
  Serial.print(closeCommand);
  delay(500);    }
```


**A)** Caricare il programma su Arduino (tenendo staccato il TX del modulo).
**B)** Ricollegare TX e aprire la finestra di monitor sul PC.
**C)** Resettare la scheda. Arduino invia, così, i comandi di configurazione al modulo WiFi e resta in attesa.
**D)** Attivare la connessione WiFi tra il PC e Al-Thinker e aprire un browser all'indirizzo 192.168.4.1. Sul monitor seriale compare la sequenza di invio della pagina di risposta e sul browser la sua interpretazione grafica.



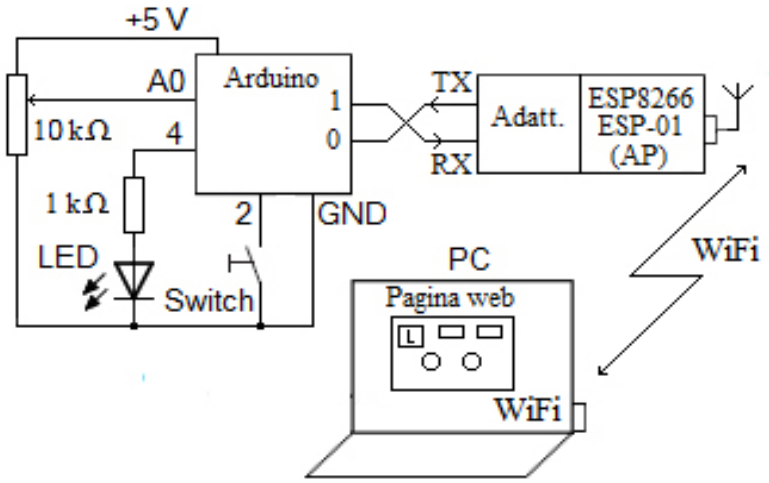




*Sopra: Comandi inviati da Arduino.*

*Sinistra: Pagina visualizzata dal browser.*

## 5) Pagina web WiFi interattiva

Per ottenere una connessione che permetta il controllo del processo su Arduino, i bottoni vanno resi capaci di scatenare una richiesta *Get* e il browser del client (PC) va istruito in modo da inviare una richiesta http con periodicità fissa e in modo automatico, per l'aggiornamento in tempo reale dello stato degli ingressi residenti sul server (Arduino), includendo nella pagina HTML l'istruzione *Refresh*.
Il programma che segue, per esempio, gestisce le richieste provenienti da due bottoni per il governo di un LED posto sul pin 4 di Arduino, e aggiorna ogni 5 secondi (Refresh: 5) il client sullo stato dell'ingresso digitale 2 e dell'ingresso analogico A0.


*Web server WiFi.*

```
/* Arduino WiFi server web_2 */
int connectionId, i;
String webPage;
char str[100];
char* StatoLed4;

void setup(){
  Serial.begin(115200); // comunicazione con ESP8266
  Serial.print("AT+RST\r\n");
  delay(1000); // resetta modulo
  Serial.print("AT+CWMODE=2\r\n");
  delay(500); // configura come access point
  Serial.print("AT+CIPMUX=1\r\n");
  delay(500); // configura per connessioni multiple
  Serial.print("AT+CIPSERVER=1,80\r\n");
  delay(500); // avvia server sulla porta 80
  pinMode(2, INPUT_PULLUP); //pin 2 input con pull-up
  //digitalWrite(2, HIGH);
  pinMode(4, OUTPUT);
  digitalWrite(4, LOW); }

void loop() {
  if(Serial.available()) //controlla se il modulo sta comunicando
  {
    if(Serial.find("+IPD,"))
    {
      delay(20);
      connectionId = Serial.read()-48;
      if(Serial.find("GET /?")) {
        for(i=0;i<7;i++)str[i]=Serial.read();
        str[i]=0;
        if(strstr(str,"pin4=on"))digitalWrite(4, HIGH);
        else if(strstr(str,"pin4=of"))digitalWrite(4, LOW);
      }
      if(digitalRead(4))StatoLed4="on";
        else StatoLed4="off";
      homePage();// prepara la pagina
      Send_webPage ();
      Send_closeCommand();
    }
  }
}
```


```
void Send_webPage (){
  String sendCommand = "AT+CIPSEND=";
  sendCommand += connectionId;
  sendCommand += ",";
  sendCommand +=webPage.length();
  sendCommand +="\r\n";
  Serial.print(sendCommand);
  while (Serial.read ()!= '>');
  Serial.print(webPage);
  delay(200);
}
void Send_closeCommand (){
  String closeCommand = "AT+CIPCLOSE=";
  closeCommand+=connectionId;
  closeCommand+="\r\n";
  Serial.print(closeCommand);
  delay(300);
}
static word homePage() {
 long t = millis() / 1000;
 word h = t / 3600;
 byte m = (t / 60) % 60;
 byte s = t % 60;
 webPage="HTTP/1.0 200 OK\r\n"
      "Content-Type: text/html\r\nPragma: no-cache\r\nRefresh: 5\r\n\r\n"
      "<html><head><title>ENC28J60</title></head>"
      "<body>"
      "<div style='text-align:center;'>"
      "<h1>ENC28J60 web server</h1>"
      "<h4>Durata della connessione :";
 sprintf(str,"%d%d:%d%d:%d%d</h4>",h/10, h%10, m/10, m%10, s/10, s%10);
 webPage.concat(str);
 sprintf(str,"<br />Input 2 = %d<br />",digitalRead(2));
 webPage.concat(str);
 sprintf(str,"<br /><br />Stato del LED4: %s<br />",StatoLed4);
 webPage.concat(str);
 webPage+="<a href='/?pin4=on'><input type='button' value='OUT 4 ON'></a>"
      "<a href='/?pin4=off'><input type='button' value='OUT 4 OFF'></a>"
      "<br /><br />";
 sprintf(str,"<br /><br />Potenziometro: %d (su 1024)",analogRead(0));
 webPage.concat(str);
 webPage+="<br /><br /></body></html>";}
```

**A)** Caricare lo sketch.
**B)** Aprire la connessione WiFi.
**C)** Attivando il browser web all'indirizzo 192.168.4.1. Comparirà la schermata riportata in figura, con il tempo di connessione, lo stato logico dell'ingresso digitale, la possibilità di modificare lo stato del LED e il livello rilevato su A0.


Ogni 5 secondi o premendo uno dei pulsanti, il PC attiva una richiesta di connessione e invia un comando.
Se nella stringa inviata dal client (di seguito ne è proposto un esempio), lo sketch rileva lo spezzone "*+IPD,*", salva il numero della connessione, gestisce l'eventuale richiesta *GET/?* (se presente), invia una pagina HTML con i valori aggiornati e chiude la comunicazione.

```
+IPD,1,437:GET /?pin4=off HTTP/1.1
Host: 192.168.4.1 …
```

**D)** Aggiungere un ulteriore switch attivo basso sul pin 3, un ulteriore LED sul pin 5 e modificare il programma in modo da renderli entrambi interattivi come già fatto per i precedenti.

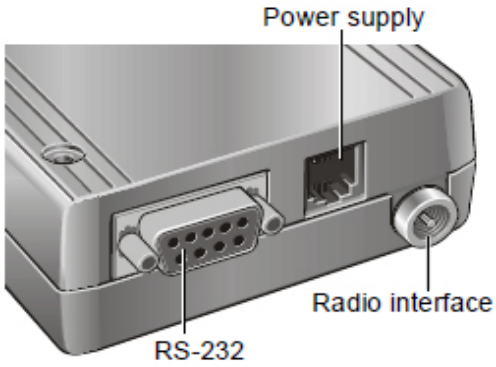

# Comandi AT per GSM

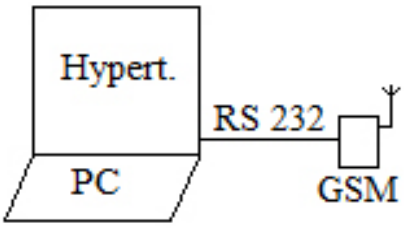
**Obiettivo**
Sperimentare i comandi AT necessari per leggere e inviare SMS mediante un GSM, utilizzando una porta RS 232.

**Conoscenze preliminari**
Per inviare SMS in un applicativo per automazione, si può utilizzare un telefono GSM con cavetto seriale RS 232 oppure un GSM industriale, quale, per esempio, MC35i Siemens.
I dispositivi GSM possono essere governati inviando appositi comandi, detti comandi AT (ATtention), mediante una linea seriale asincrona.
I comandi sono stringhe di caratteri ASCII che iniziano con le due lettere 'A' e 'T' (maiuscole o minuscole) e terminano con il carattere di invio ($C_R$ = torna a capo = 0DH), mentre le risposte risultano incapsulate tra due coppie di caratteri $C_RL_F$ .
I comandi possibili sono numerosissimi, ma per le funzioni di automazione bastano i pochi sviluppati in questa esercitazione.

**Fasi operative**

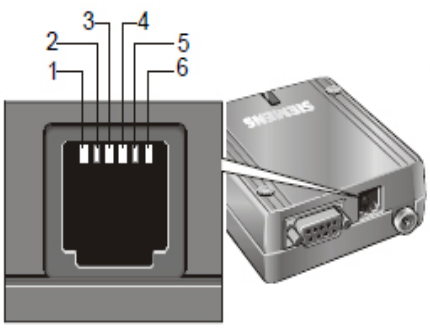
**1) Connessioni elettriche**
**A)** Connettere il telefonino GSM al PC mediante il cavetto seriale in dotazione.
Se si dispone del GSM industriale MC35i Siemens, la connessione impiega un cavo DTE – DCE diritto, con connettori femmina-maschio.



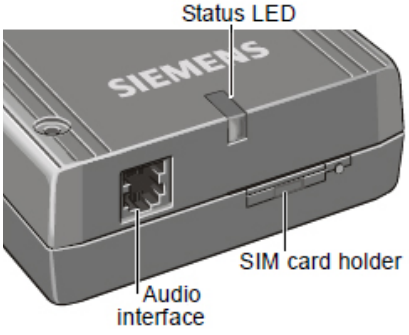

*Sopra: connessione con un GSM.*
*A sinistra: Interfacce per GSM industriale.*

**B)** Connettere al GSM anche l'antenna a frusta in dotazione e riporre la SIM nell'apposita tasca estraibile.
**C)** Oltre ai due conduttori di alimentazione, il GSM industriale dispone solitamente di ulteriori ingressi per ridurre il consumo durante i tempi di inutilizzo (*Power Down*) o per attivare il dispositivo all'accensione (*Ignition*). In tal caso, questi ingressi vanno collegati opportunamente a uno dei terminali di alimentazione, in modo da porre in funzione il dispositivo (PD_IN = GND; IGT_IN = + V).




Pin assignment:

| | |
|---|---|
| 1 | + 8 ÷ 30 V |
| 2 | free |
| 3 | PD_IN |
| 4 | IGT_IN |
| 5 | free |
| 6 | GND |




*A sinistra: Jack Western a 6 poli per l'alimentazione del GSM MC35i (Siemens)*
*Qui a fianco: tasca per la SIM.*


## 2) Comandi di base

**A)** Aprire una finestra di comunicazione Hyperterminal sul PC e attivare l'eco dei caratteri trasmessi, in modo da vedere ciò che si invia.
**B)** Digitando il comando AtCR (tab. 1), formato dai tre caratteri "A", "t" e invio, se il GSM è alimentato, sul video compare *AtAt*, con *OK* su una seconda riga. Questo perché, di default, il GSM esegue al suo interno l'eco dei comandi ricevuti e conferma l'esecuzione del comando con una stringa da sei caratteri

$C_RL_F$ $OKC_RL_F$ ($L_F$ = salta riga = 0AH)

- Per eliminare l'eco del GSM, digitare $A_TE0C_R$.
- Digitando un comando errato, si ottiene come risposta la stringa ERROR.

**Tab. 1 – COMANDI DI BASE**

| Comando | Risposta | |
|---|---|---|
| | 1° riga | 2° riga |
| $AtC_R$ | AtAt | OK |
| $ATE0C_R$ | AtAtee00 | OK |
| $AtciaoC_R$ | ERROR | |
| At+cpin="2222"$C_R$ | OK | |
| At+clip=1$C_R$ | OK | |
| $ATH0C_R$ | OK | |
| At+CSQ$C_R$ | +CSQ:21.99 | |

**C)** Se abilitato per accedere ai servizi della SIM, inserire il codice pin, con il comando *at+cpin* seguito dal codice. Da questo momento, se esiste una chiamata per la SIM inserita, sul PC compare periodicamente la scritta RING.
- Per abilitare la visualizzazione del numero del chiamante, digitare *at+clip=1*$C_R$.
- Per rifiutare la chiamata, il comando è *ATH0*$C_R$.

**D)** Se si vuole conoscere il livello del campo in ricezione, inviare *at+csq*$C_R$. La risposta è un numero compreso tra 0 (–113 dBm o minore) e 31 (–51 dBm o maggiore).
*At+cgmi*$C_R$ restituisce il costruttore del terminale, mentre *at+cgmm*$C_R$ fornisce il modello del terminale. Per altri comandi consultare il manuale del terminale.

## 3) Lettura dei messaggi

All'interno di un dispositivo GSM, i messaggi possono essere trattati in formato PDU a 7 bit o in formato ASCII. Il secondo è il formato più comodo per la lettura da PC.
**A)** Attivare il formato ASCII inviando il comando *at+cmgf=1*$C_R$ (con 'f' da "format").
**B)** Inviare *at+cmgl="ALL"*$C_R$ (con 'l' da "list"), per visualizzare tutti i messaggi presenti nel dispositivo.
**C)** Ripetere il comando precedente utilizzando la stringa "REC_READ" per leggere solo quelli già letti, oppure "REC_UNREAD" per ottenere solo eventuali messaggi nuovi, mai aperti.

La risposta del GSM è composta da più righe.
Supponendo, per esempio, che in memoria esista un solo messaggio (numero 1) già letto, la prima riga è del tipo:

```
+CMGL: 1,"REC_READ","+393409876543","14/01/05,17:18:58+04"
```

L'informazione:
- indica che al messaggio è stato assegnato il numero 1;
- dice che è già stato aperto;
- fornisce il numero del mittente (+39 per l'Italia);
- specifica l'istante di ricezione: anno (2014), mese (01), giorno (05), ore (17), minuti (18), secondi (58) e il meridiano di riferimento (+4 quarti d'ora rispetto al meridiano zero di Greenwich).

Una seconda riga contiene il testo in chiaro del messaggio e una terza riga chiude con la stringa OK.

**D)** Per cancellare un messaggio dalla memoria del GSM, si ricorre al comando *at+cmgd=* seguito dal numero del messaggio da eliminare e chiuso con il carattere $C_R$ di invio.

### 4) Invio di un messaggio

**A)** Comporre il comando *at+cmgs=* seguito dal numero del destinatario e chiuso con $C_R$ (tab. 2). Il modem GSM risponde su una nuova riga con il simbolo "**>**" seguito da uno spazio in attesa del testo del messaggio.

**B)** Comporre il testo del messaggio e chiuderlo con il carattere *Ctrl-Z (1AH)*, tipicamente entro un tempo massimo di 180 s (premere *ESC* se si vuole uscire prima dal comando).

Una volta inviato il messaggio, il GSM risponde con l'eco del comando seguito dal numero progressivo assegnato al messaggio nella memoria e termina con l'OK di conferma.

**Tab. 2 – INVIO DI UN MESSAGGIO DI TESTO**

| Operatore | GSM |
|---|---|
| at+cmgs=340987654$C_R$ | |
| | > |
| valvola 32 attiva Ctrl-Z | |
| | $C_RL_F$ +CMGS:26$C_RL_F$ |
| | $C_RL_F$ OK$C_RL_F$ |