Laboratorio di Modellazione

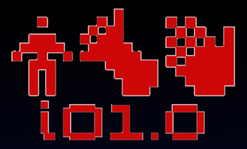
Stefano Sanna

Laboratorio del corso "Atelier di Disegno Industriale 2" - Prof. Lorenzo Imbesi
Prima Facoltà di Architettura - Università La Sapienza - Roma

Anno Accademico 2007/2008

interactiveobject1.0

Versione 1.7 - 30 Giugno 2008



Panoramica

- Presentazioni

- Obiettivi del laboratorio
- Risorse

- Programma del laboratorio

- START! :-)

Obiettivi del laboratorio

- Scopo principale di questo modulo è introdurre l'adozione della piattaforma **Arduino** per la realizzazione di prototipi di oggetti **interattivi** attraverso l'uso di dispositivi elettronici **programmabili**, un insieme di **sensori** e **attuatori**, sistemi di **comunicazione** wireless

Risorse

- Le slide degli argomenti trattati in aula saranno temporaneamente disponibili all'indirizzo

  http://www.gerdavax.it/seminari

  e saranno costantemente aggiornate

- Per informazioni, approfondimenti, suggerimenti potete scrivermi all'indirizzo: **gerdavax@tiscali.it**

Licenza

- Queste slide sono distribuite con licenza Creative Commons - ShareAlike - Non Commercial. E' possibile trovare copia della licenza in italiano all'indirizzo: http://creativecommons.org/licenses/by-nc-sa/2.5/it/

- Arduino, il software e i progetti hardware/software ad esso correlati sono distribuiti con licenza **Creative Commons - ShareAlike** e sono utilizzati in queste slide a scopo didattico

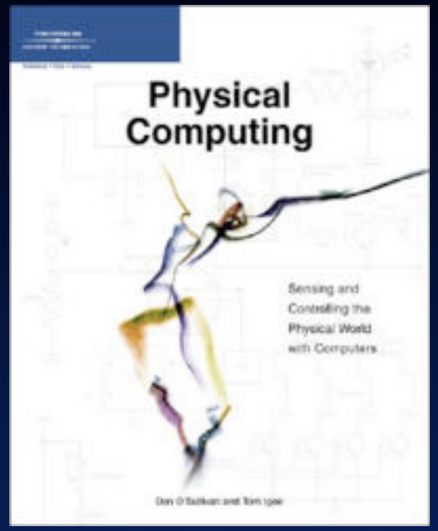
Riconoscimenti

- Accanto a contenuti originali, queste slide si ispirano al lavoro della community di utilizzatori di Arduino

- Per la realizzazione di questo materiale è stato preso spunto dai siti web:
  - Arduino Official Site: **http://www.arduino.cc**
  - Bionic Arduino: **http://todbot.com/blog/bionicarduino/**
  - Limor Ladyada(?) web site: **http://www.ladyada.net/**
  - Wikipedia: **http://www.wikipedia.org**

Approfondimenti

- Physical Computing:
  Sensing and Controlling
  the Physical World with Computers
  - Dan O'Sullivan, Tom Igoe
  - Course Technology, 2004



- Making Things Talk
  - Tom Igoe
  - Make Books (O'Reilly), 2007

Programma

- Parte 1
  - introduzione ai sistemi per Interaction Design, dall'idea al prototipo
  - introduzione ad Arduino: hardware e software, primi esperimenti
- Parte 2
  - input e output analogico e digitale, sensori e attuatori
- Parte 3
  - comunicare con Arduino (segue seminario: "Comunicazione Pervasiva")
- Le tre parti cooperano allo svolgimento del laboratorio e alla realizzazione dei prototipi interattivi

Dall'idea di interazione...

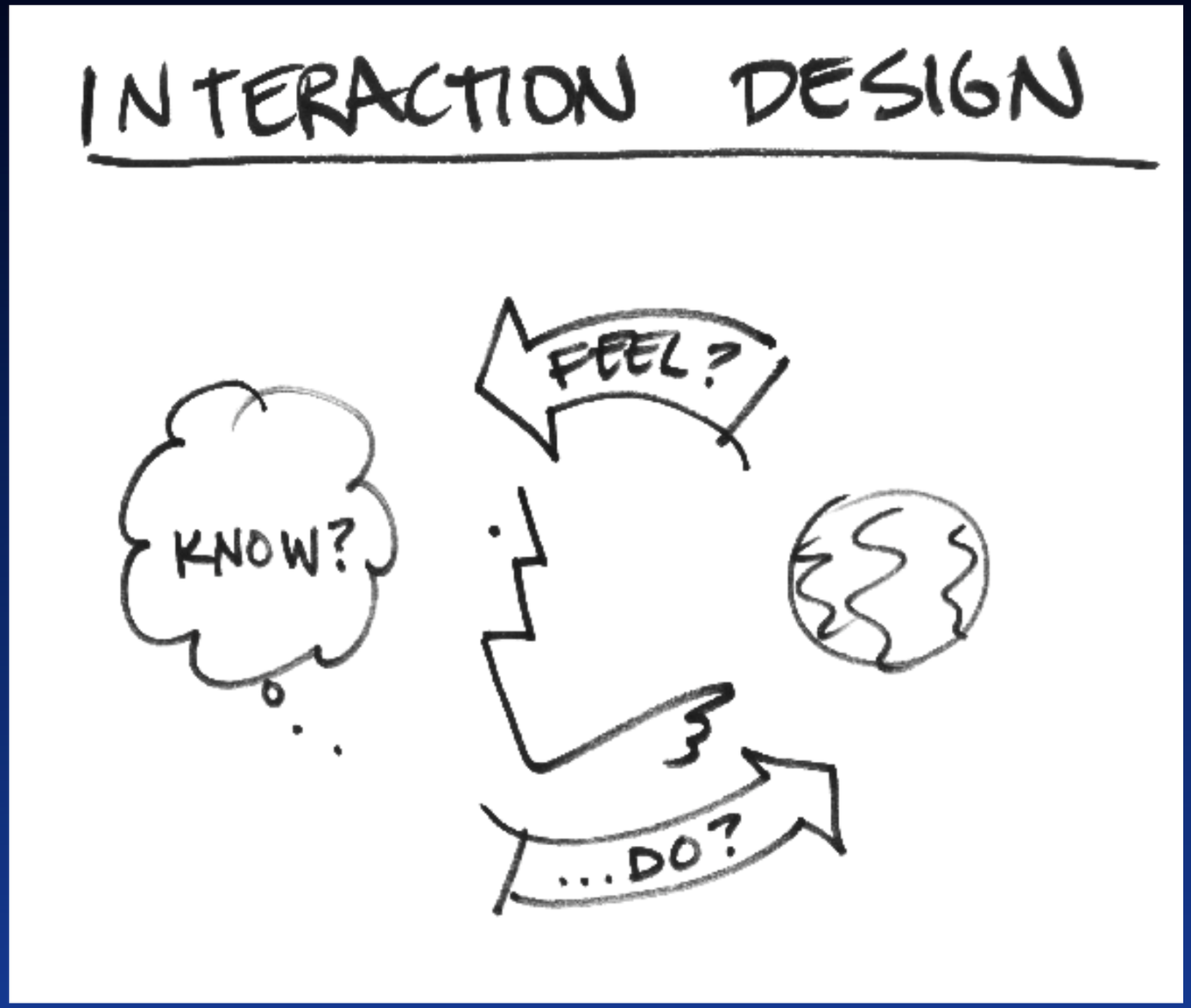


Image by Bill Verplank

... all'oggetto interattivo!

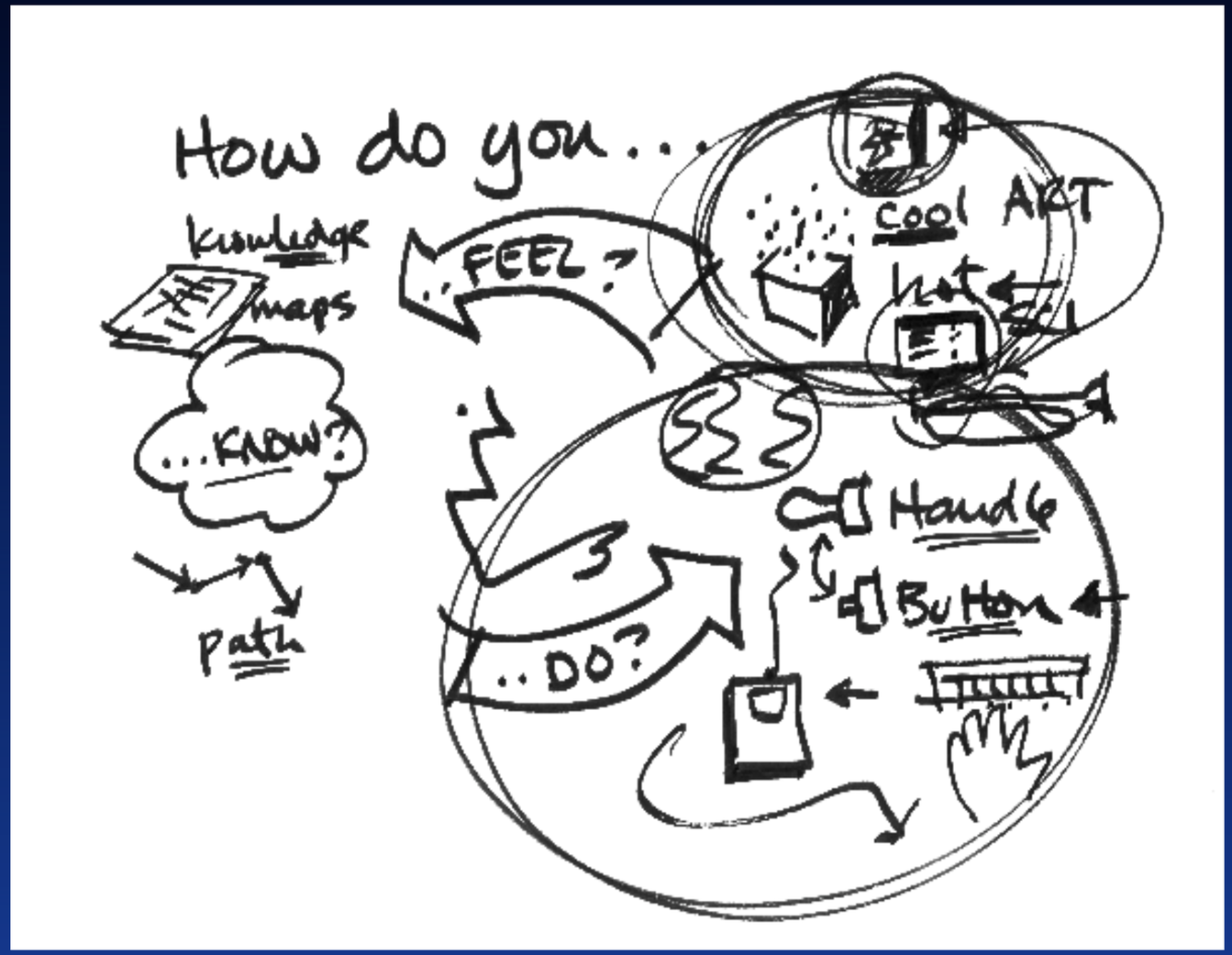


Image by Bill Verplank

Hardware & Software

- L'interattività degli oggetti si ottiene attraverso la realizzazione di componenti fisici (hardware) opportunamente programmati (software)

- Occorre tenere presente:
  - cosa è fisicamente fattibile e come farlo (meccanica/fisica)
  - quali componenti realizzano la funzione richiesta e come si collegano (elettronica)
  - quale logica descrive il comportamento richiesto e come programmarla (informatica)

Input, elaborazione, output

- **Input**: dati provenienti da un insieme di sensori, ciascuno in grado di trasformare la grandezza fisica di interesse in una tensione (sensore analogico) o in una sequenza di bit (sensore digitale)
- **Elaborazione**: insieme di algoritmi che valutano i dati in ingresso e producono dei risultati in uscita
- **Output**: cambiamenti nell'ambiente circostante il sistema, eseguiti da un insieme di attuatori, ciascuno in grado di trasformare il risultato dell'elaborazione in azioni fisiche (una luce accesa, una scritta visualizzata su un display, un motore in rotazione, un dato inviato ad un elaboratore remoto)

Modellazione

- Modellare l'**input**: quali sensori rilevano la grandezza di interesse e come questa può essere espressa all'interno del programma

- Modellare l'**output**: quali attuatori producono l'effetto desiderato e quali istruzioni occorre fornire affinché eseguano le azioni

- Modellare la **logica**: interpretare i dati di input, valutare la correlazione e la sequenza tra eventi, predisporre dati di output

Sistemi embedded

- Arduino: http://www.arduino.cc
- Phidgets: http://www.phidgets.com
- MULN: http://www.robot-italy.com
- Basic STAMP: http://www.parallax.com
- MAKE Controller: http://www.makezine.com/controller/
- LEGO Mindstorms NXT: http://mindstorms.lego.com
- BUGS: http://www.buglabs.net
- Sun SPOT: http://www.sunspotworld.com

Requisiti principali

- Affinché possano essere utilizzati per la realizzazione di sistemi interattivi, i moduli devono possedere queste caratteristiche:

  - programmabilità
  - numero adeguato di porte di input e output (I/O)
  - buona dotazione di interfacce di comunicazione di alto livello
  - facile interfacciabilità con altri sistemi
  - piccole dimensioni e ridotto consumo energetico

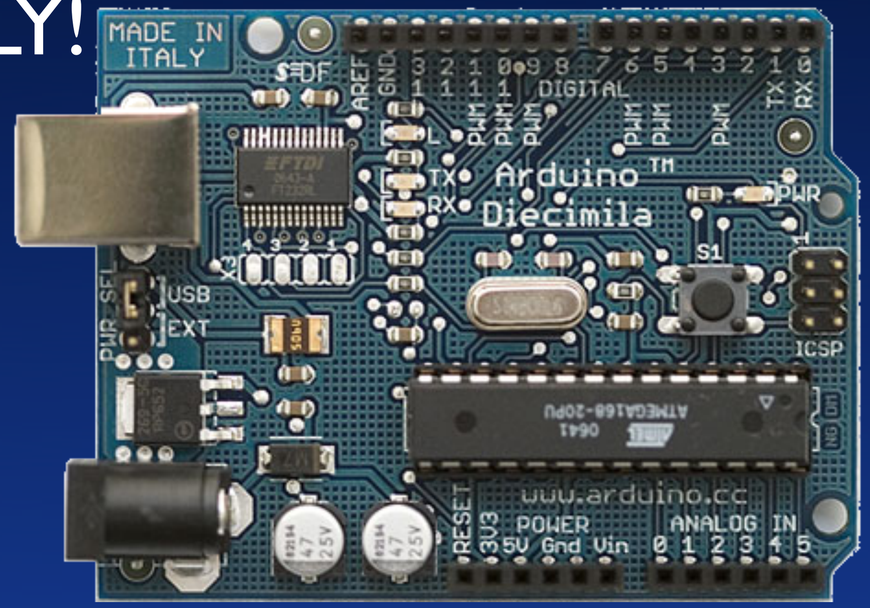
  - **community**

Introduzione ad Arduino

Caratteristiche hardware e software

Arduino

- E’ un sistema economico, “open”, flessibile, semplice da utilizzare, supportato da una vasta comunità di sviluppatori
- Il linguaggio di programmazione deriva dal C
- Arduino è orgogliosamente MADE IN ITALY!
- È sufficiente effettuare una ricerca su Google, Youtube, Flickr per scoprire un ricchissimo patrimonio di idee, progetti, esperimenti, prodotti basati su Arduino



What is Arduino?

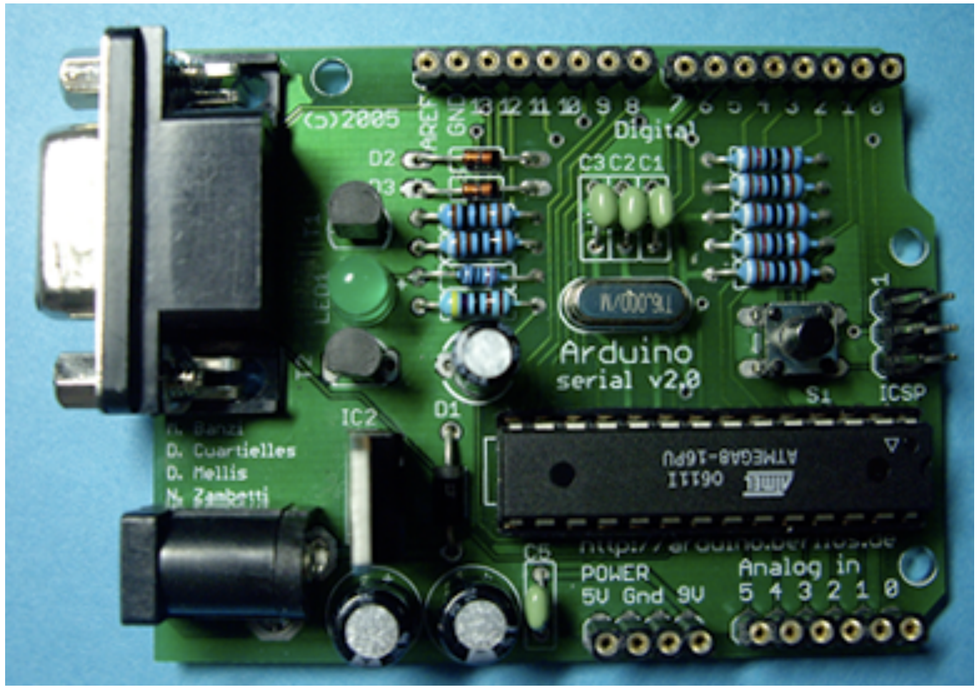
- Dal sito ufficiale:

  Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments.

  Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators.

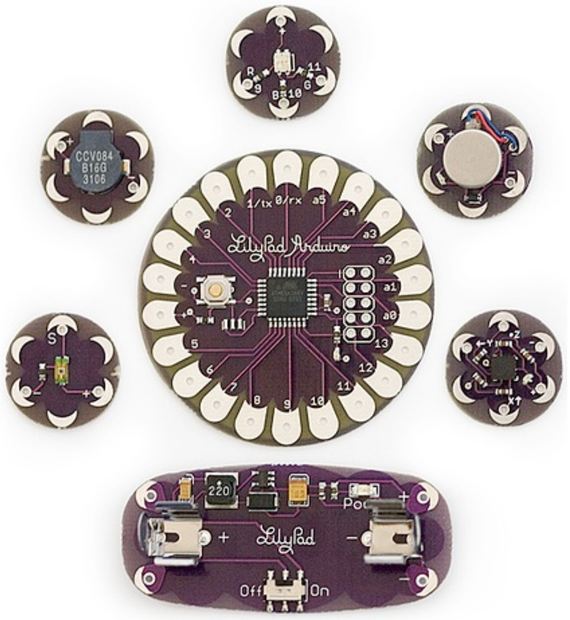
Genesi

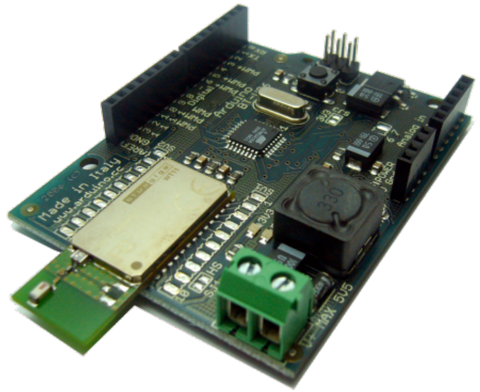
- Arduino è ideato da Massimo Banzi e sviluppato insieme a David Cuartielles, Tom Igoe, Gianluca Martino e David A. Mellis;
- Arduino (linguaggio e piattaforma hardware) deriva dal progetto Wiring, mentre l'ambiente di sviluppo deriva da Processing, un sistema di programmazione designer e creativi
- In breve tempo (tre anni!) Arduino è diventato una degli strumenti di sperimentazione di interaction design più utilizzati al mondo!

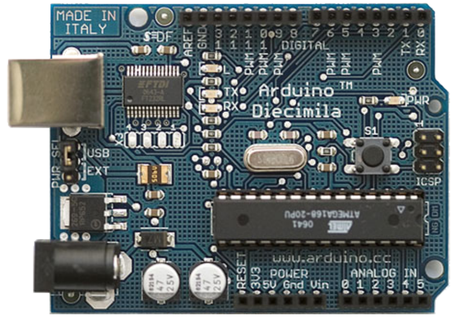


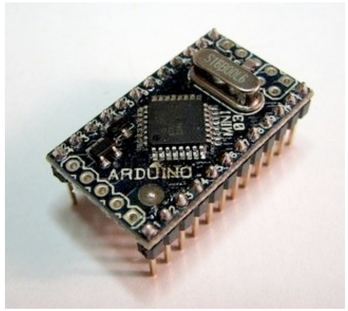
Varianti di Arduino

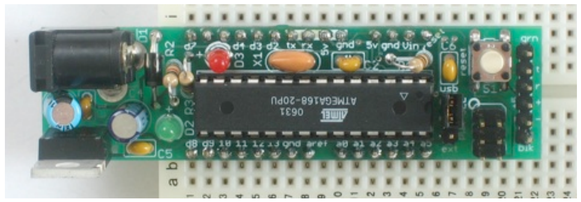
- Prime versioni
  - Arduino, Arduino USB, Arduino NG
- Versioni attuali:
  - **Arduino Diecimila**
  - Arduino Mini
  - **Arduino Bluetooth**
  - Lilypad
  - Boarduino
  - Freeduino





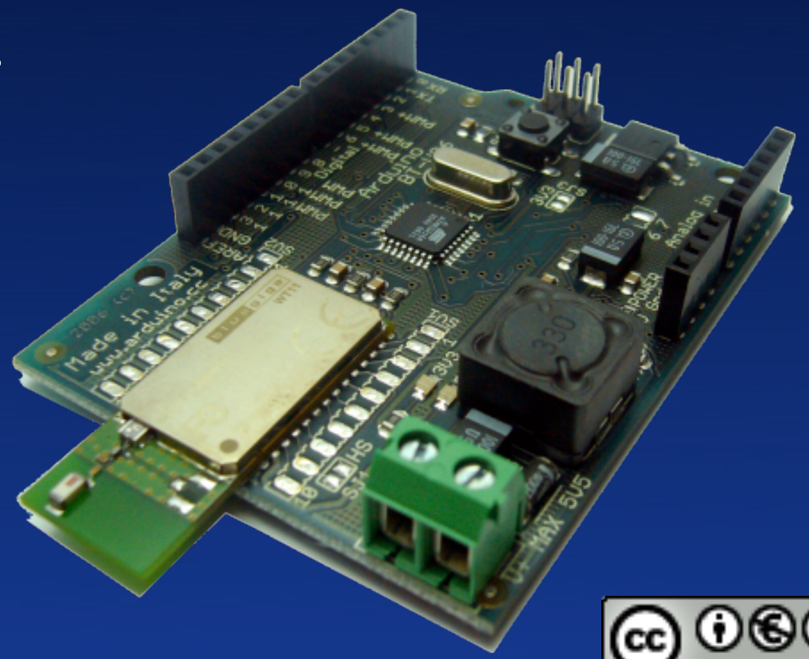


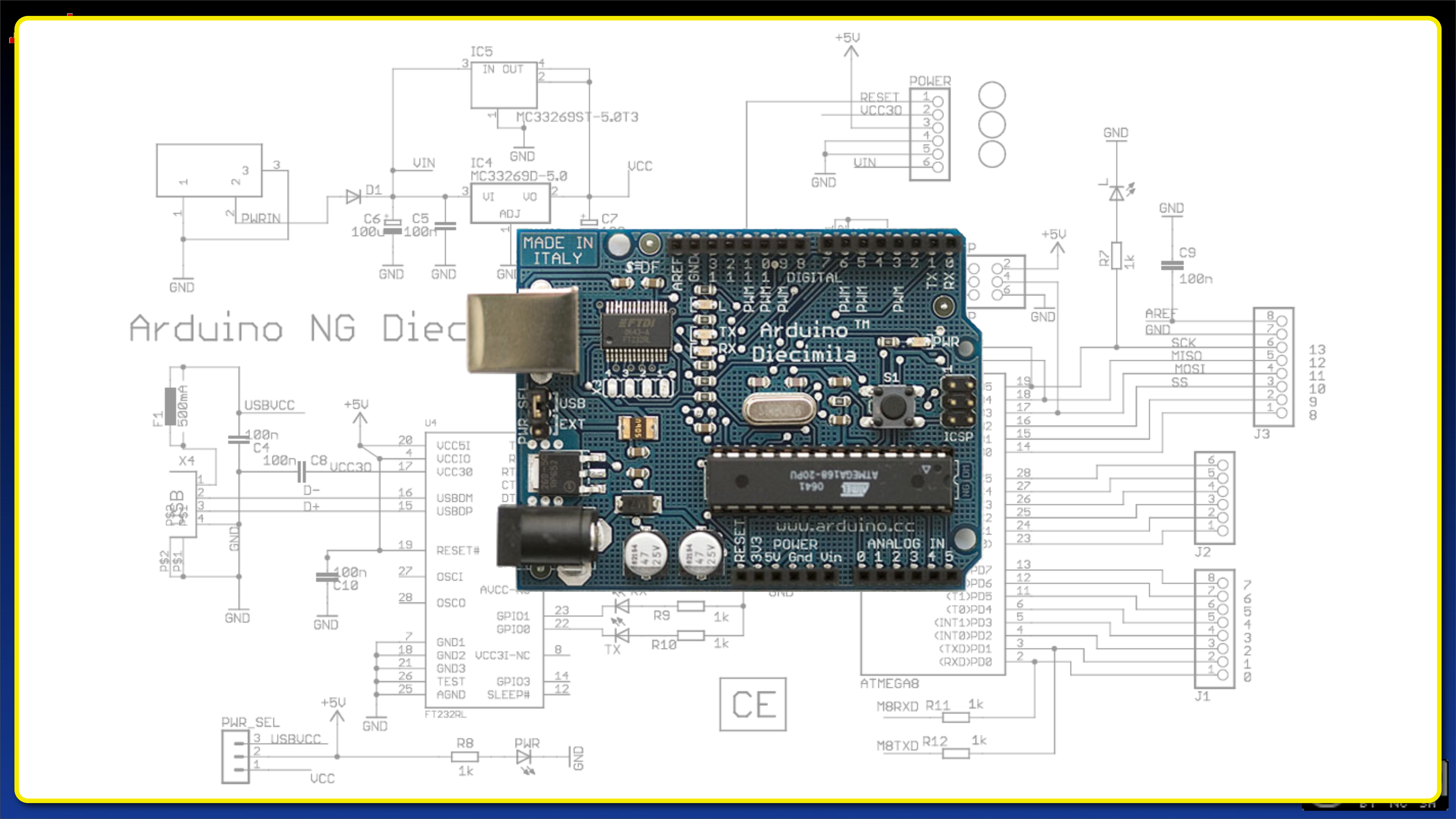




Arduino Bluetooth

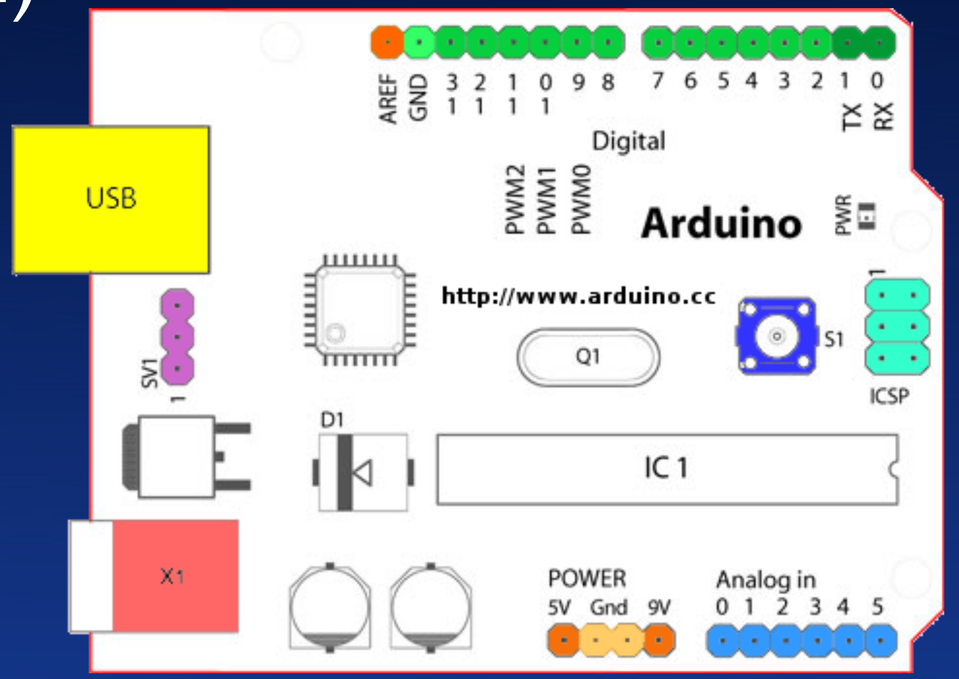
- È una versione di Arduino equipaggiata di un modulo Bluetooth che sostituisce l'interfaccia USB
- La connessione wireless può essere utilizzata per la programmazione del modulo e per comunicare con periferiche Bluetooth quali computer, cellulari, palmari...
- Sarà utilizzata a fine corso per il controllo remoto di Arduino attraverso un telefono cellulare




Arduino NG Diec
MADE IN ITALY
Arduino
Diecimila
www.arduino.cc
POWER
ANALOG IN
DIGITAL
ICSP
IC5
MC33269ST-5.0T3
IC4
MC33269D-5.0
VIN
VCC
GND
PWRIN
D1
C5
C6
C7
C9
100n
100u
R7
1k
+5V
RESET
VCC30
J1
J2
J3
S1
PWR
USB
EXT
PWR_SEL
USBVCC
F1
500mA
C4
C8
C10
X4
D-
D+
U4
FT232RL
VCC5I
VCCIO
VCC30
USBDM
USBDP
RESET#
OSCI
OSCO
GPIO1
GPIO0
GND1
GND2
GND3
TEST
AGND
VCC3I-NC
GPIO3
SLEEP#
R8
R9
R10
R11
R12
M8RXD
M8TXD
ATMEGA8
SCK
MISO
MOSI
SS
CE

Hardware

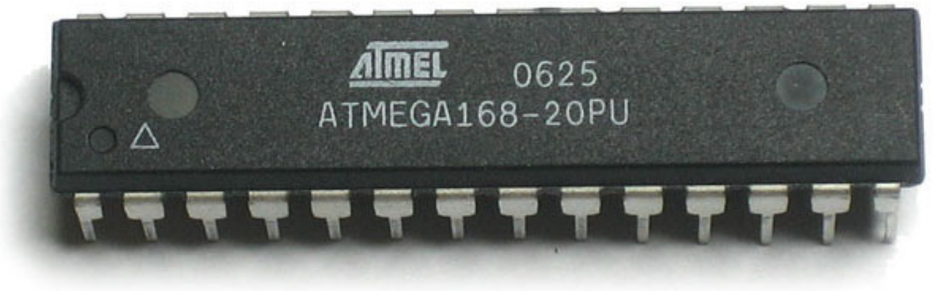
- Microcontroller ATMEL ATmega168
- Alimentazione: 6-20V (consigliati 7-12V) o via USB
- Ingressi/uscite digitali: 14 (6 output PWM)
- Ingressi analogici: 6
- Porta seriale TTL 5V
- Programmazione da PC via USB
- Dimensioni: 7 x 5.5 cm



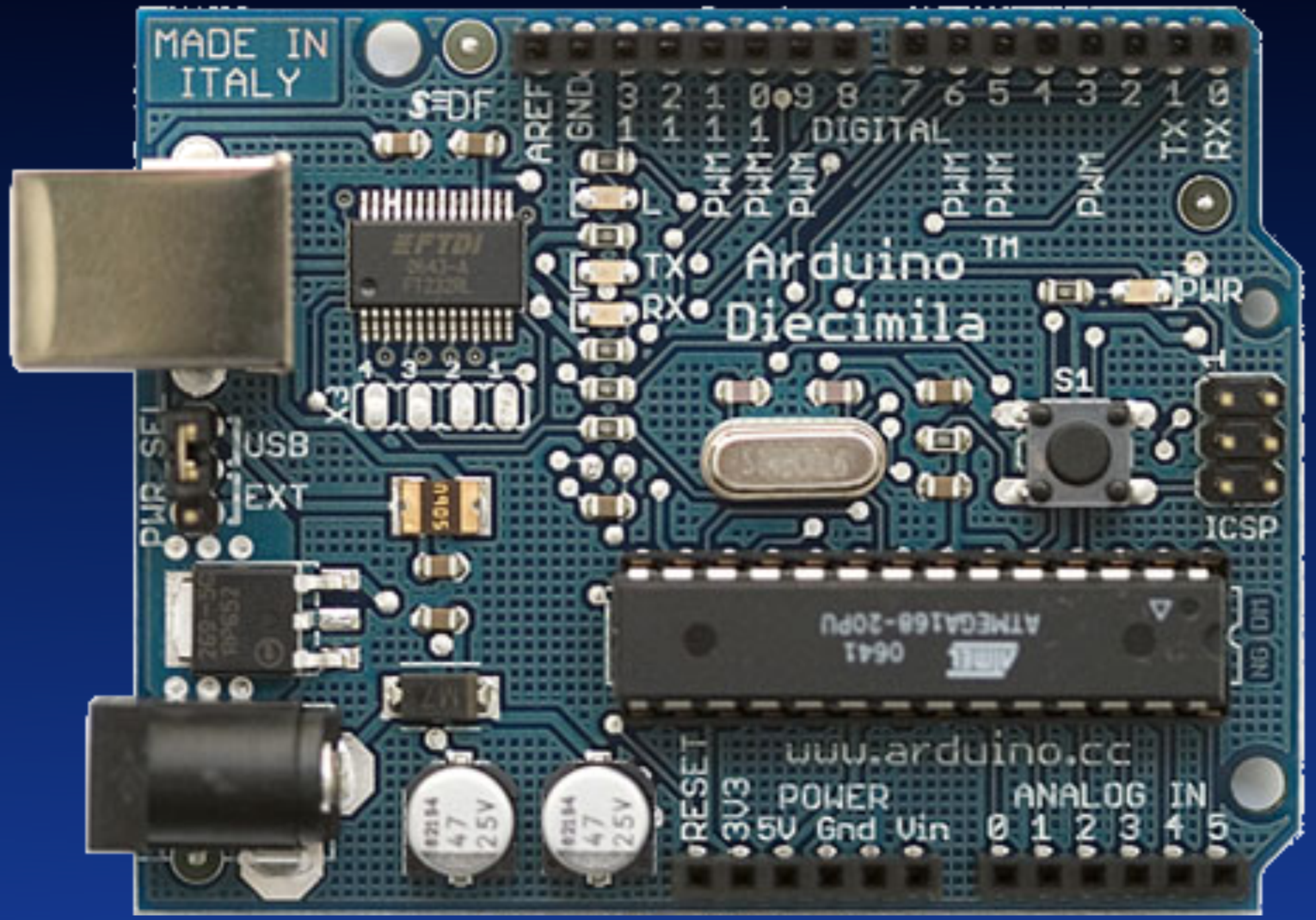
Software

- Clock: 16Mhz
- Memoria flash: 16KB (di cui 14KB per i programmi utente)
- RAM: 1KB
- EEPROM: 512B

- Impossibile fare confronti con il PC di casa!



Arduino Diecimila



Interfacce

Porta USB

Input/Output digitali

Interfaccia di programmazione microcontrollore

Porte di alimentazione per dispositivi esterni

Selettore alimentazione (USB - esterna)

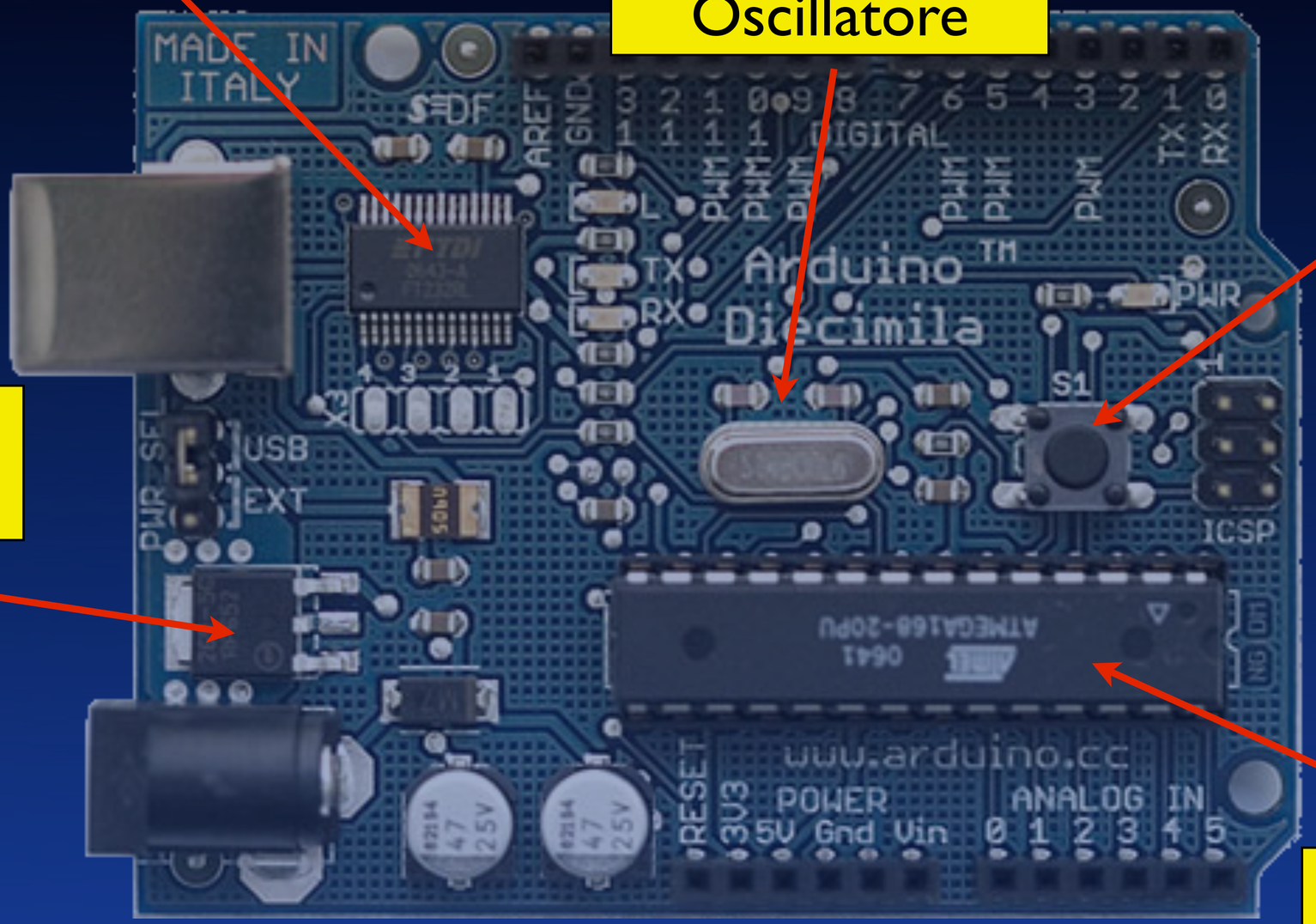
Ingressi analogici

Connettore di alimentazione

Interfacce

- Porta USB:
  - consente la comunicazione con il PC, sia durante la programmazione (caricamento nuovi sketch) che l'esecuzione dei programmi. La porta USB provvede anche ad alimentare il circuito.
- Connettori di alimentazione:
  - consentono di alimentare Arduino quando non connesso al PC. È necessaria una tensione continua compresa tra 7 e 12V
- Selettore alimentazione:
  - seleziona l'alimentazione via USB o attraverso alimentatore esterno.

Componenti



Driver USB

Oscillatore

Reset

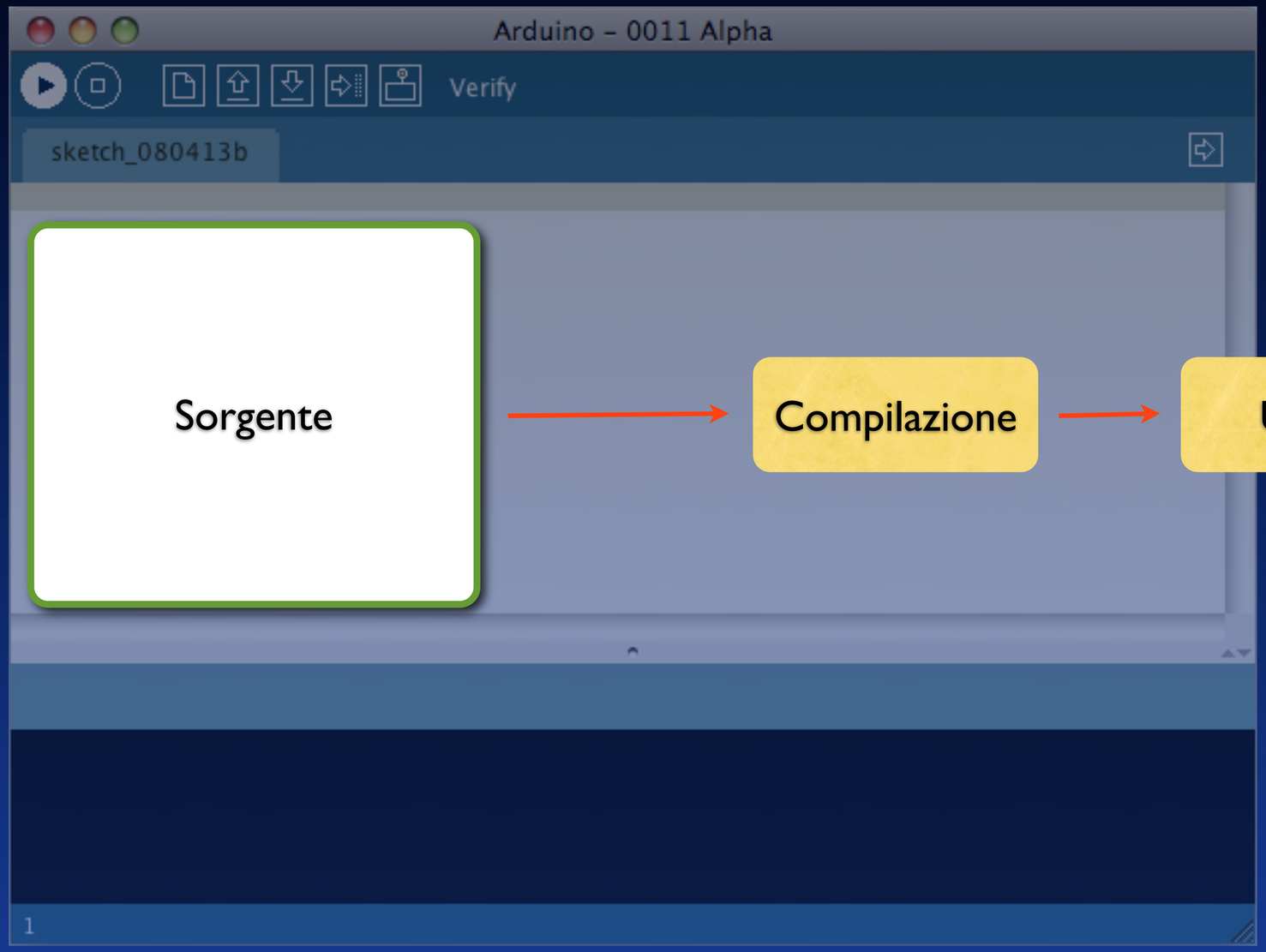
Regolatore di tensione

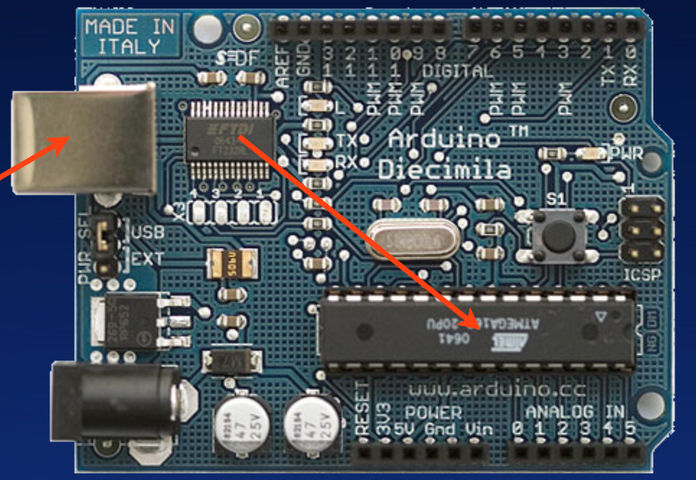
Microcontroller

Componenti

- Microcontroller:
  - esegue i programmi, acquisisce dati in ingresso e invia i dati in uscita.
- Driver USB:
  - realizza l'interfaccia tra la seriale dell'ATmega e la posta USB del PC
- Oscillatore:
  - è "l'orologio interno" di Arduino, con il quale il microcontrollore esegue i programma e gestisce i flussi dati digitali
- Regolatore di tensione:
  - provvede a mantenere costante la tensione di alimentazione per tutti i circuiti

Processo di sviluppo



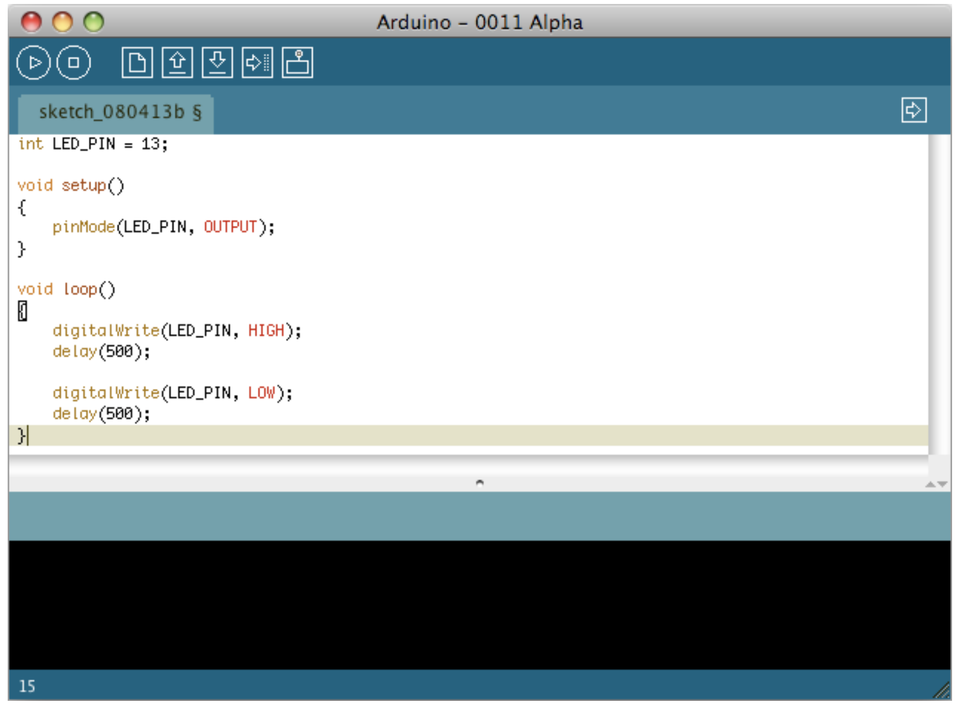


Ambiente di sviluppo

- Per scrivere programmi (chiamati sketch) per Arduino è necessario utilizzare un software open source scaricabile gratuitamente all'indirizzo:

  http://www.arduino.cc

- L'ambiente di sviluppo è disponibile per Windows, Mac OS X e Linux

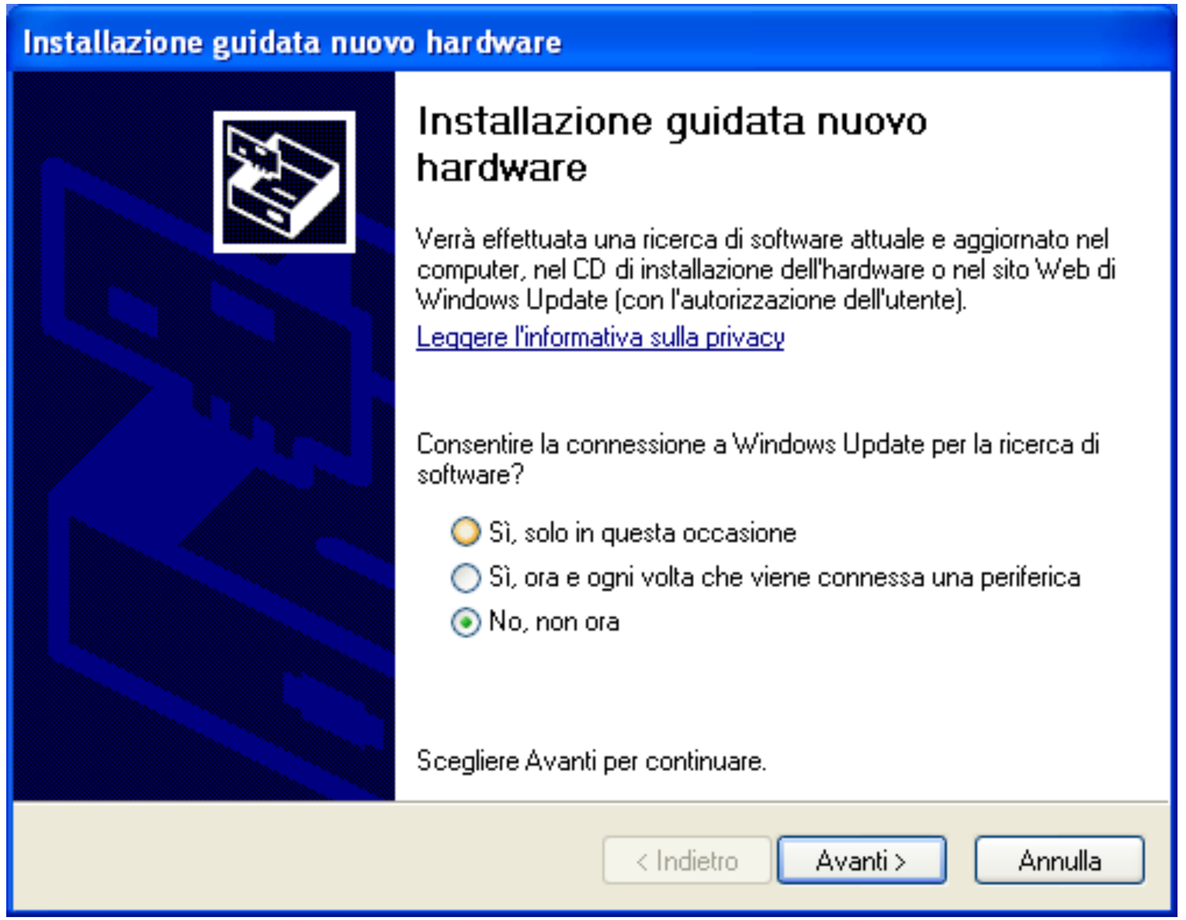


Installazione

- Windows
  - Scaricare il file **http://www.arduino.cc/files/arduino-0011-win.zip** e decomprimerlo in una cartella
  - L'installazione del driver per l'interfaccia USB-FTDI per Arduino è richiesta automaticamente dal sistema operativo connettendo Arduino alla porta USB: indicare il driver contenuto nella cartella **drivers/FTDI USB Drivers**
- Mac OS X (Tiger e Leopard)
  - Scaricare il file **http://www.arduino.cc/files/arduino-0011-mac.zip** e decomprimerlo in una cartella
  - Installare il driver per l'interfaccia USB-FTDI per Arduino, contenuto nella cartella drivers (**FTDIUSBSerialDriver_v2_1_9.dmg** per PowerPC e **FTDIUSBSerialDriver_v2_2_9_Intel.dmg** per Intel)

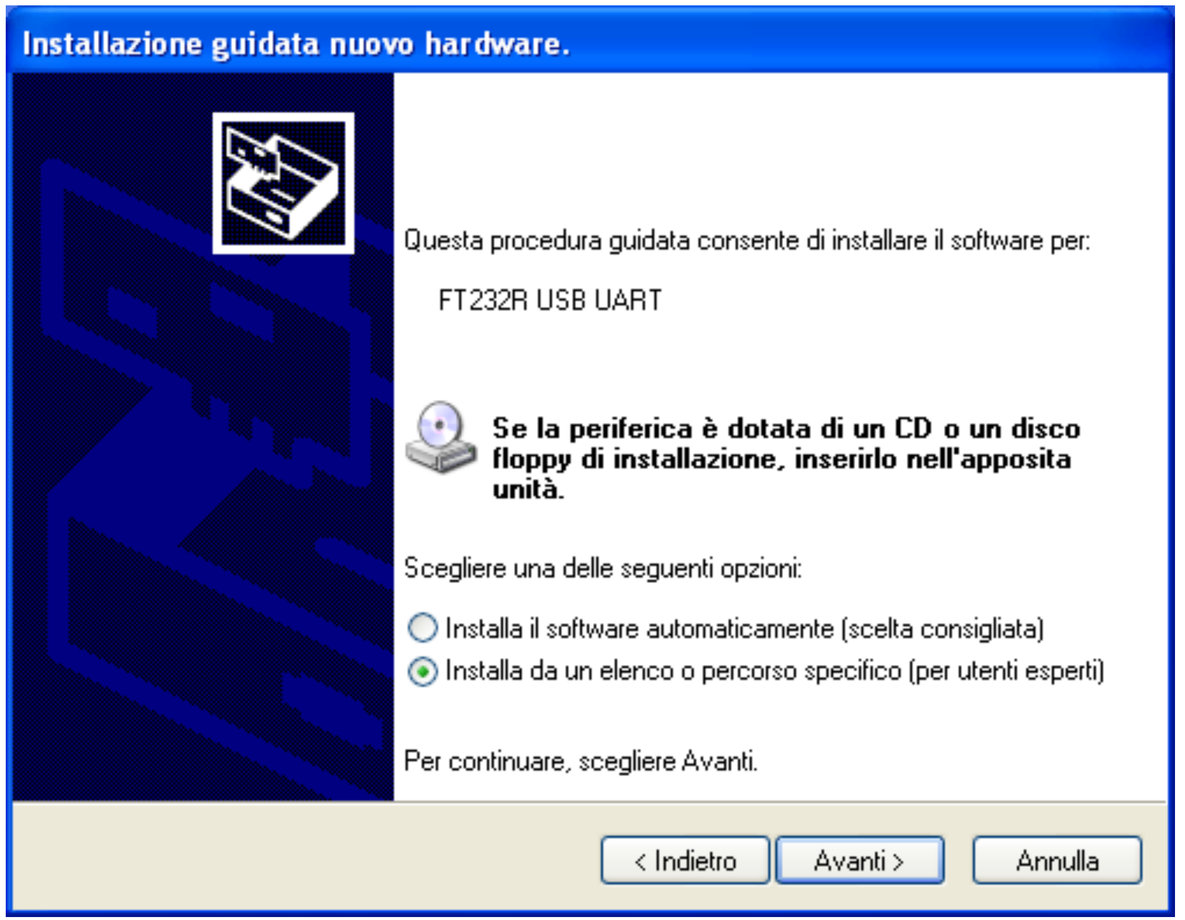
Installazione Windows [1]

- Dopo aver scaricato e decompresso l'archivio di Arduino, connettere la scheda ad una USB libera
- Il sistema avvisa che è stato trovato un nuovo hardware
- Non è necessario effettuare la connessione al servizio Windows Update



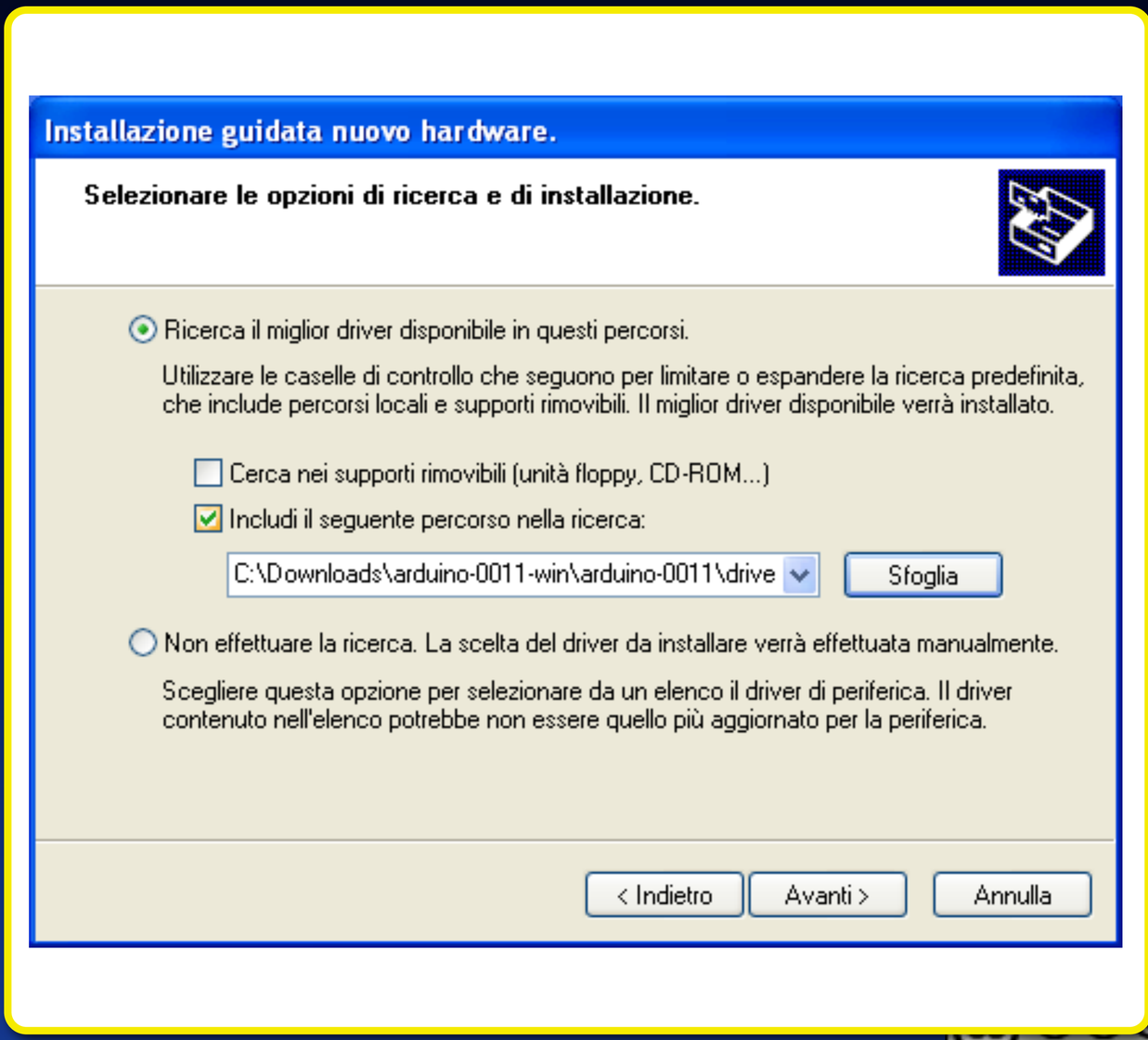
Installazione Windows [2]

- Il primo driver da installare è per il device **FT232R USB UART**
- Selezionare l'installazione da percorso specifico (non selezionare la modalità automatica)



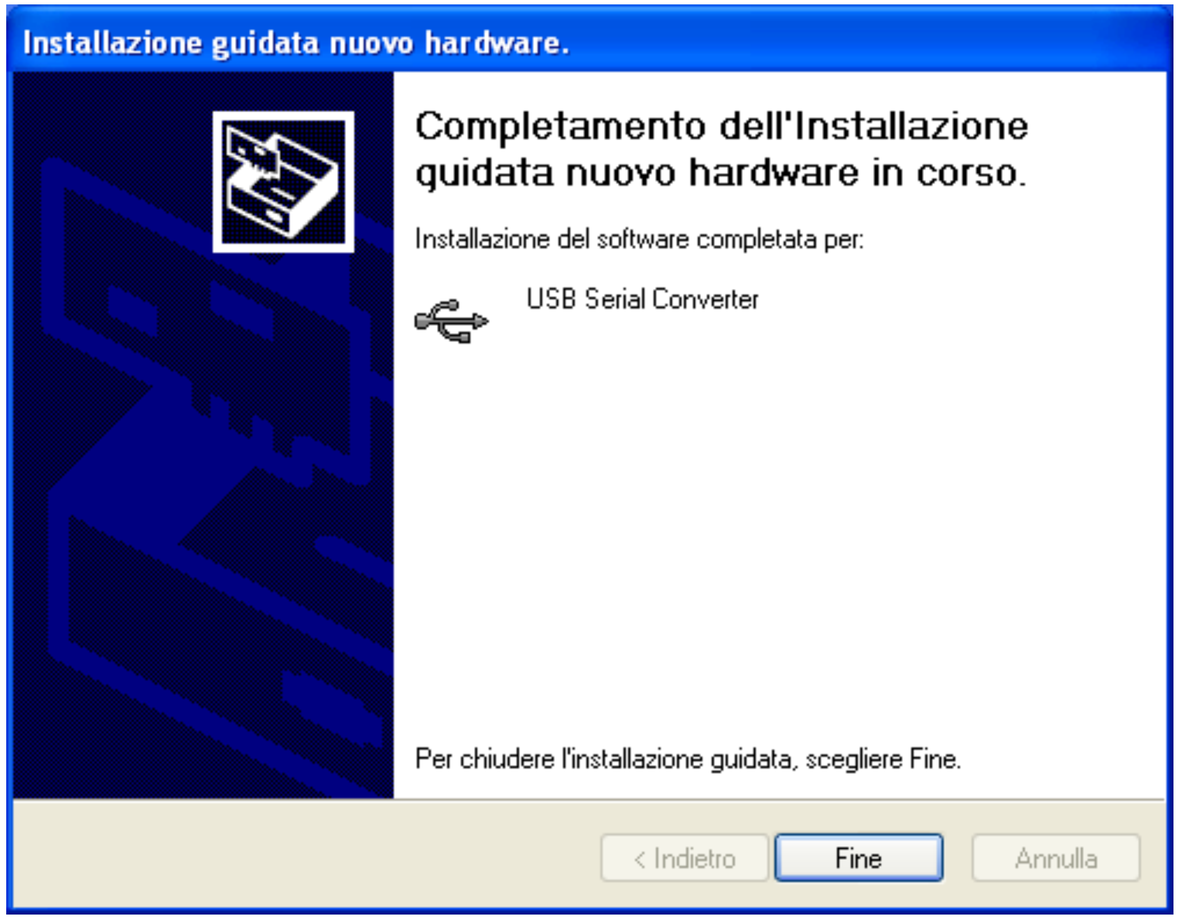
Installazione Windows [3]

- Indicare il percorso di ricerca dei driver nella cartella **driver** del folder dove è stato decompresso l'archivio di Arduino



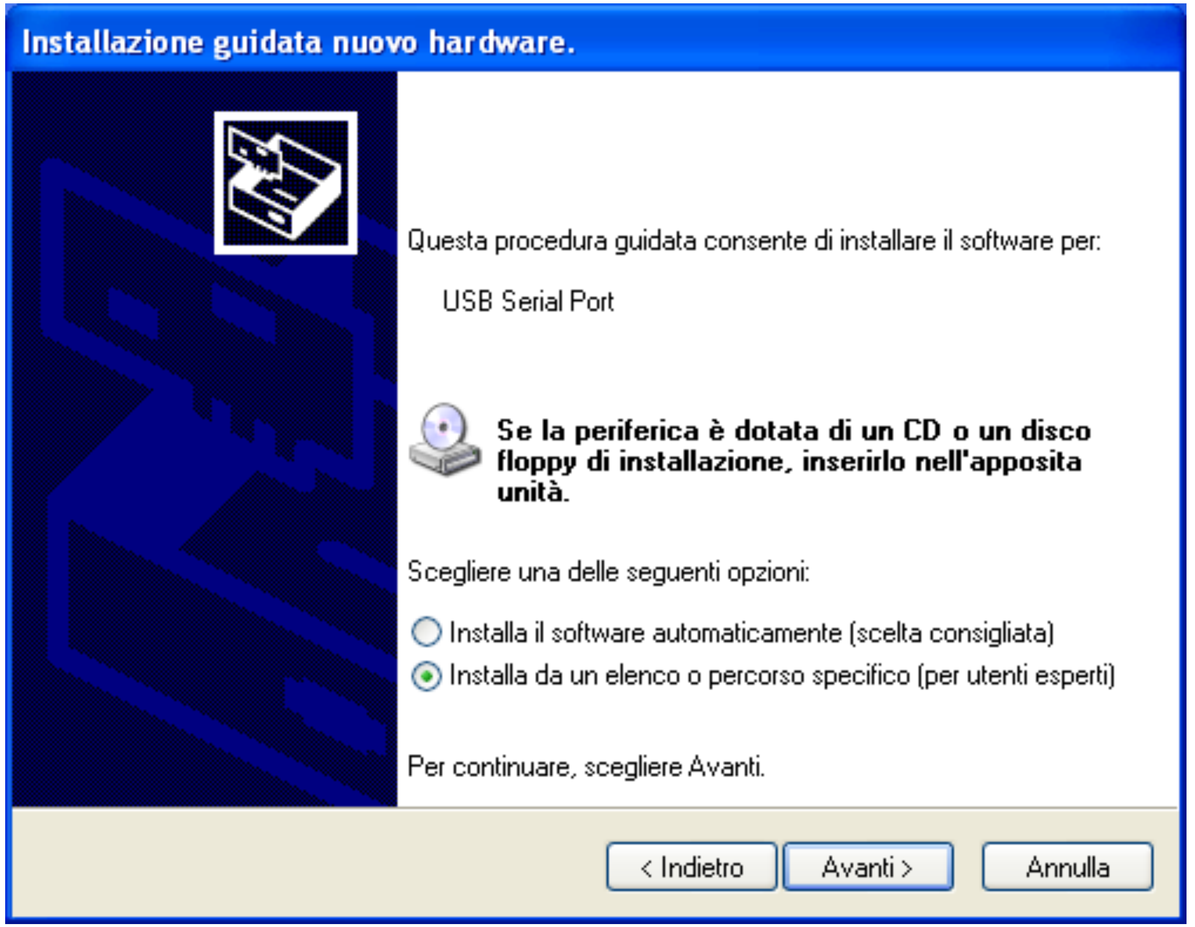
Installazione Windows [4]

- Dopo alcuni secondi, il primo driver è installato



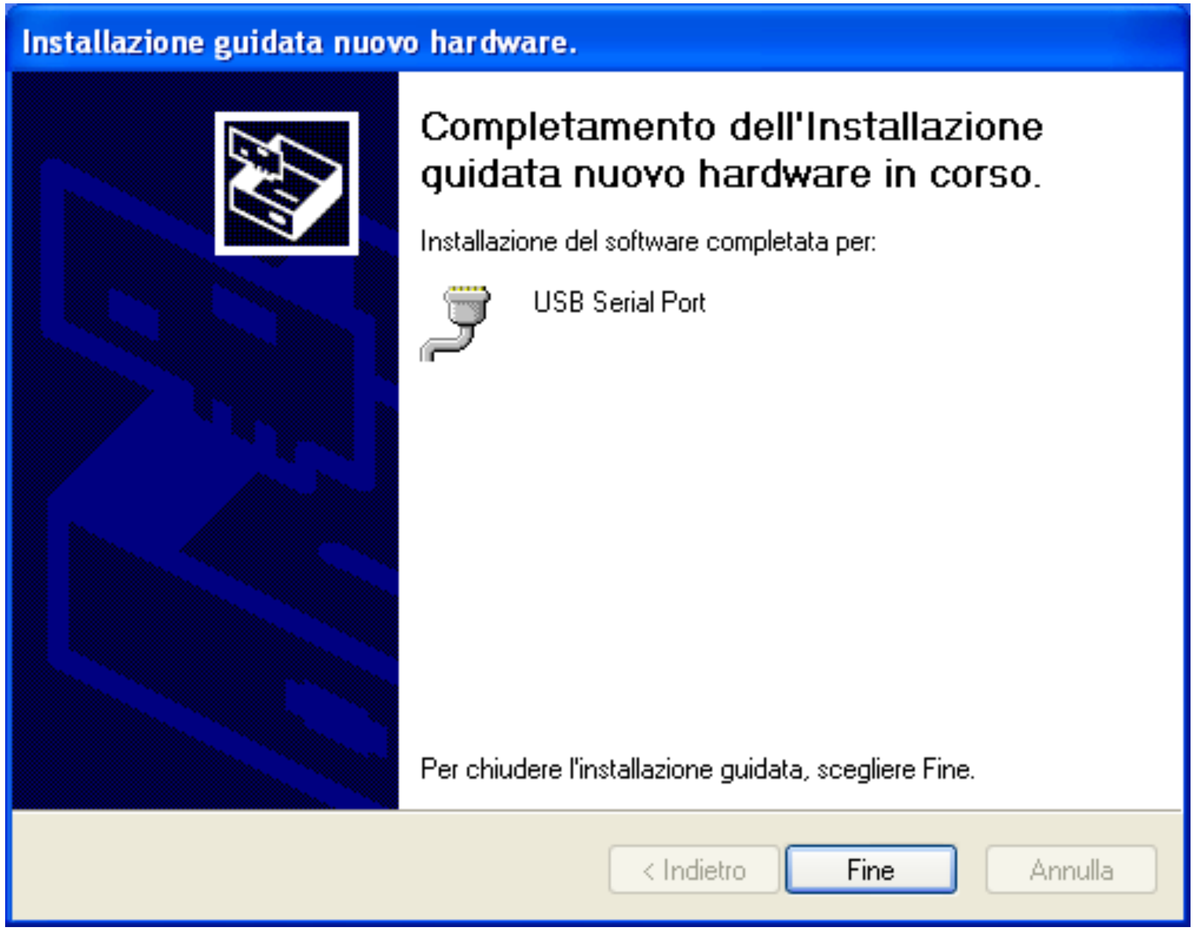
Installazione Windows [5]

- È ora necessario installare il driver per il dispositivo **USB Serial Port**
- Valgono le stesse selezioni viste nelle slide precedenti



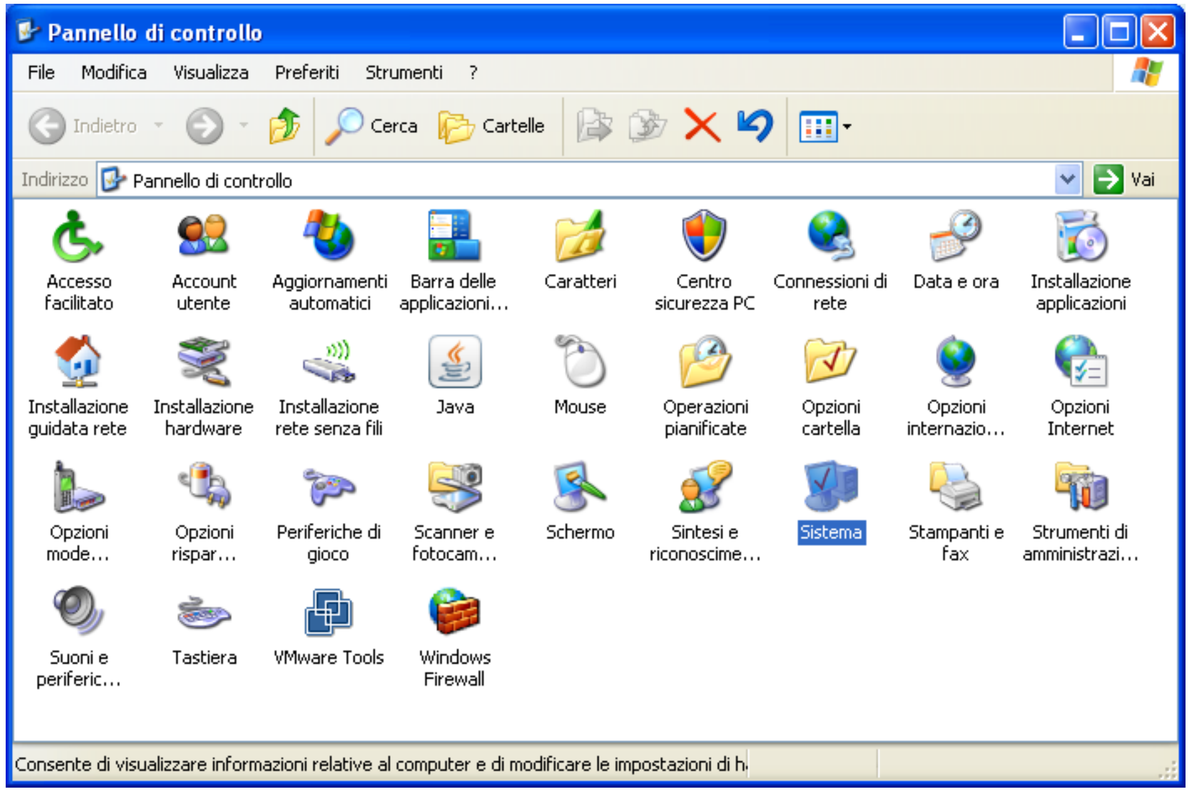
Installazione Windows [6]

- Dopo alcuni secondi, il secondo driver è installato



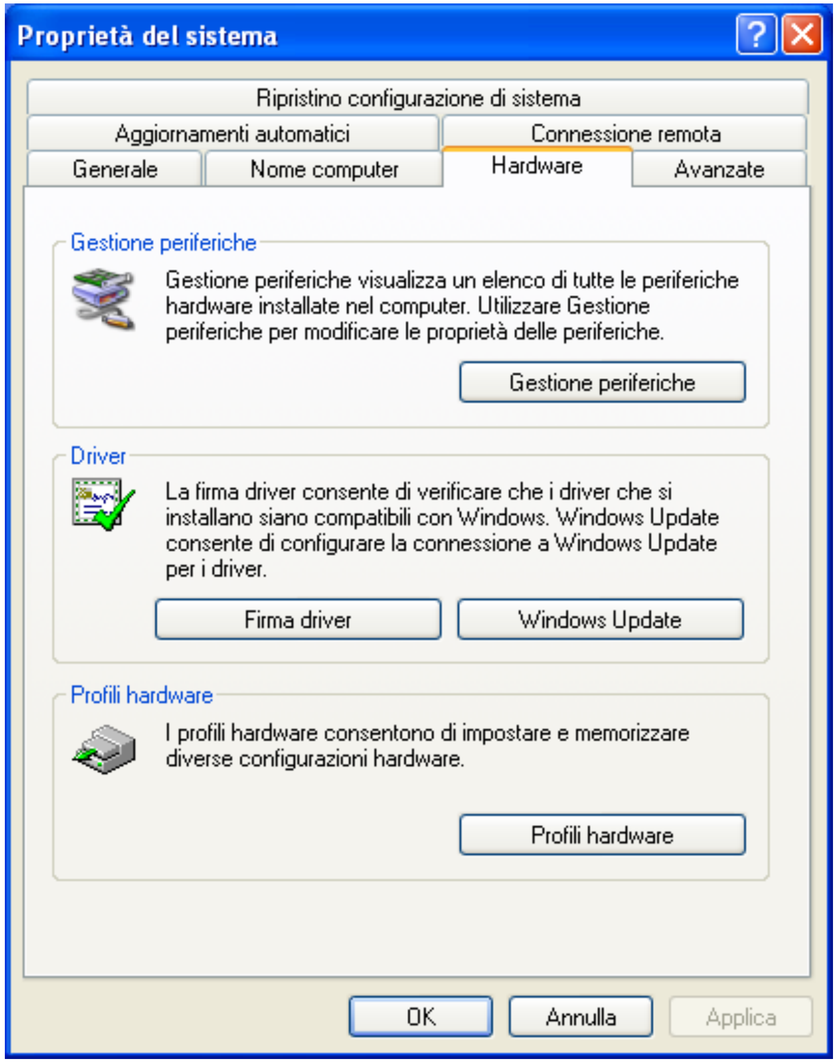
Installazione Windows [7]

- È ora necessario individuare la porta seriale virtuale che è stata assegnata al driver FTDI associato ad Arduino
- Aprire il pannello di controllo e selezionare l'icona **Sistema**



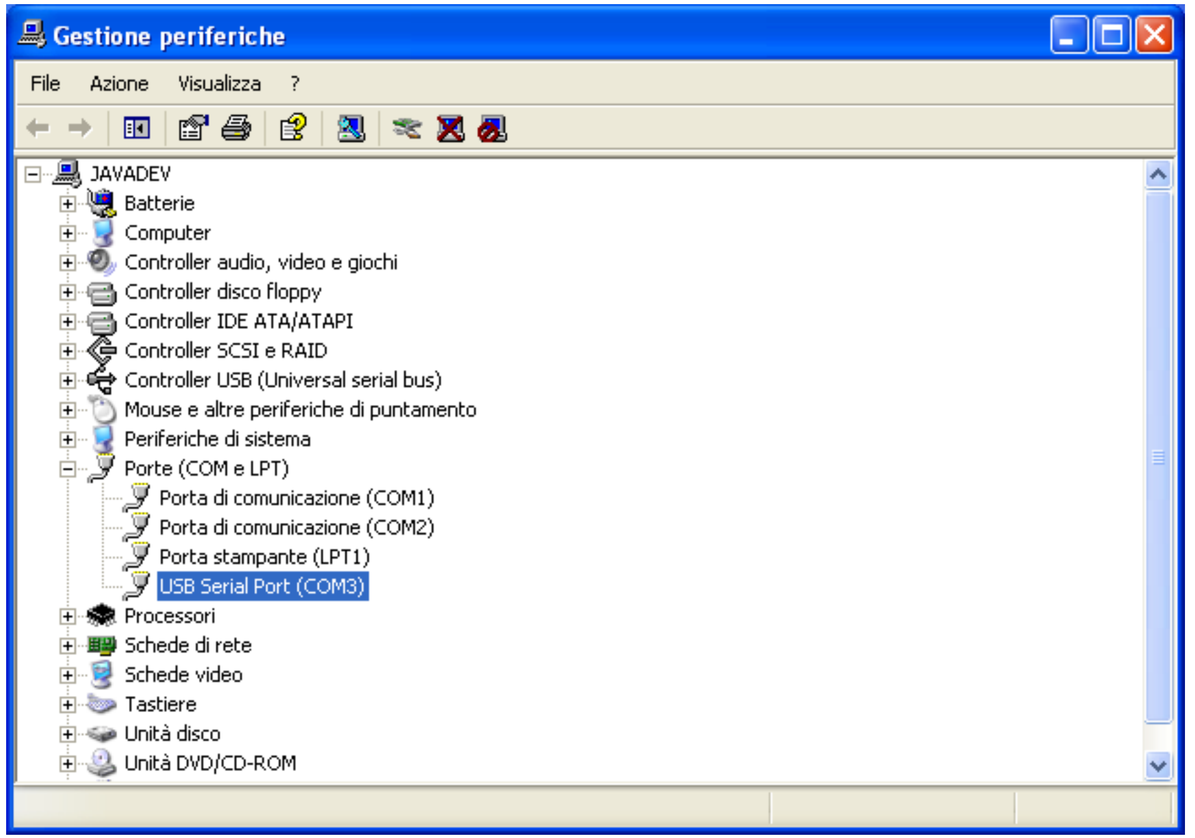
Installazione Windows [8]

- Selezionare il pannello **Hardware** e il pulsante **Gestione periferiche**



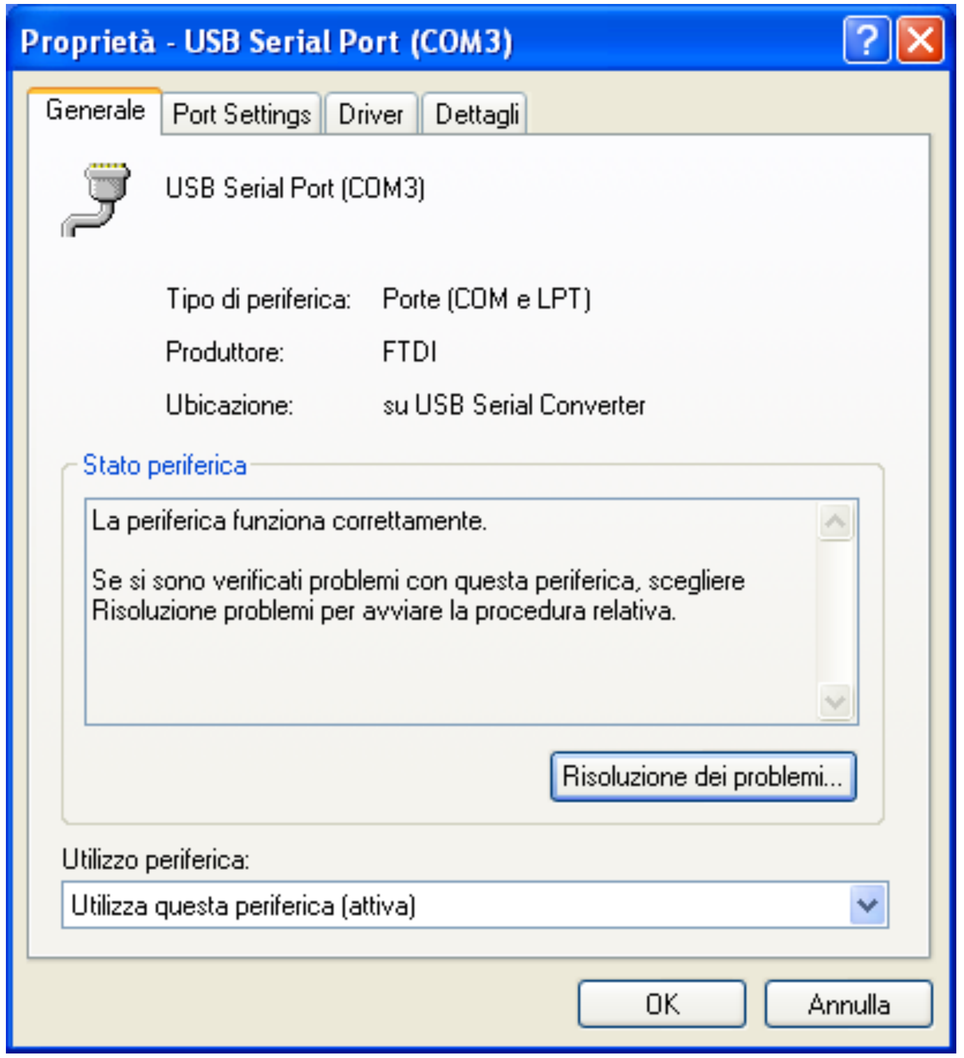
Installazione Windows [9]

- Nella finestra Gestione periferiche, selezionare il gruppo Porte (COM e LPT)
- Tra le porte seriali e parallele elencate, dovrebbe esserne presente una denominata **USB Serial Port**: tra parentesi è indicato il nome assegnato alla porta (in questo caso, **COM3**)



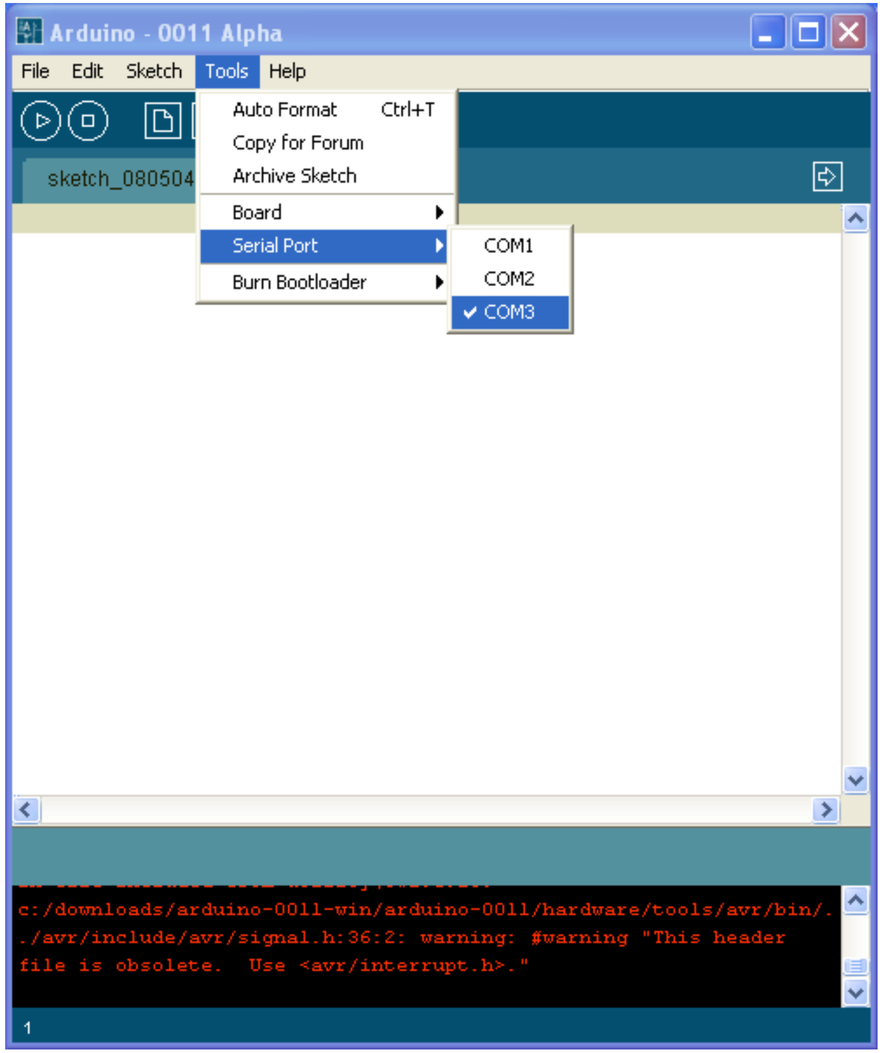
Installazione Windows [10]

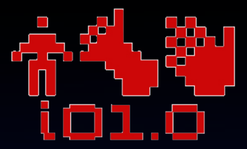
- Se fossero presenti più dispositivi simili, è possibile, con un doppio click sull'icona, aprire la finestra dei dettagli che mostra il dispositivo associato alla porta
- Verificare che si tratti di un dispositivo del produttore **FTDI**



Installazione Windows [11]

- Avviando l'ambiente di sviluppo di Arduino, è ora possibile indicare la porta seriale associata alla scheda attraverso il menu **Serial Port** del gruppo **Tools**



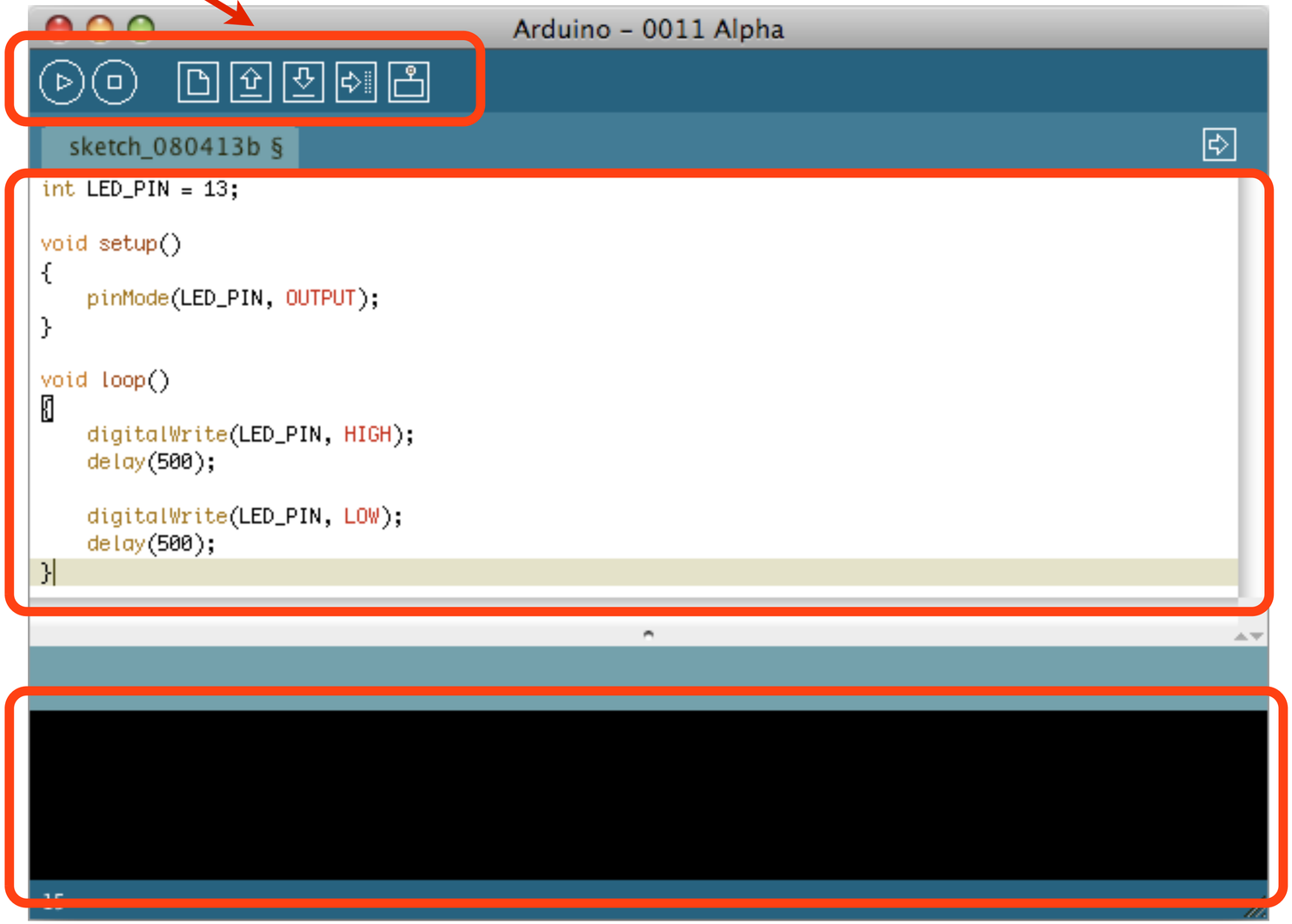


Installazione su Linux

- Work in progress (sorry!)

IDE

Toolbar



Editor

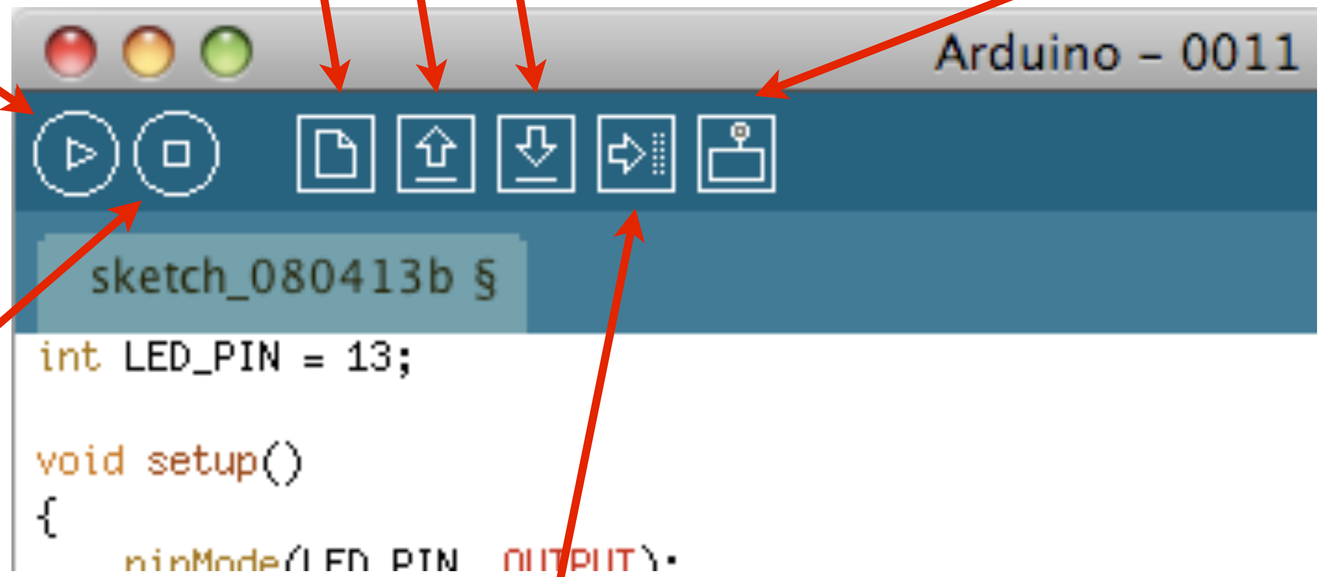
Console

IDE Toolbar

Gestione file:
nuovo, apri, salva

Compilazione

Attivazione console seriale



Chiusura
console seriale

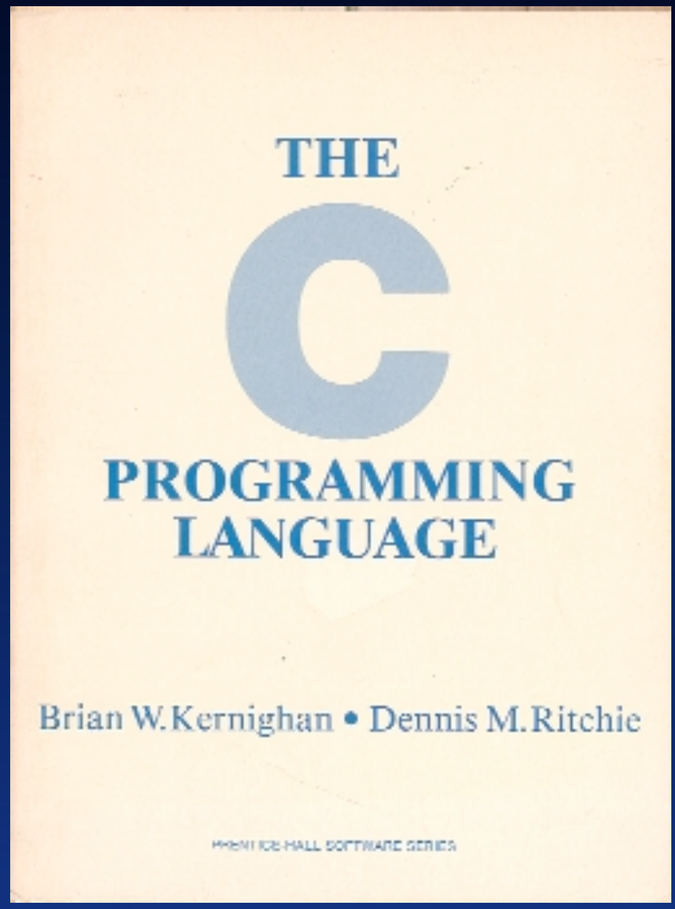
Compilazione e caricamento
programma su Arduino

IDE Menu

DEMO

Hello World!

- Nel 1978 Brian Kernigham e Dennis Ritchie pubblicano un libro storico:
  “The C Programming Language”
- Il linguaggio C è universalmente utilizzato su tutte le piattaforme hardware e software
- Il libro di Kernigham e Ritchie ha introdotto la curiosa usanza di “insegnare” ogni nuovo linguaggio di programmazione con l’esempio più semplice possibile: un programma che dice “Hello World”, “Ciao mondo”.



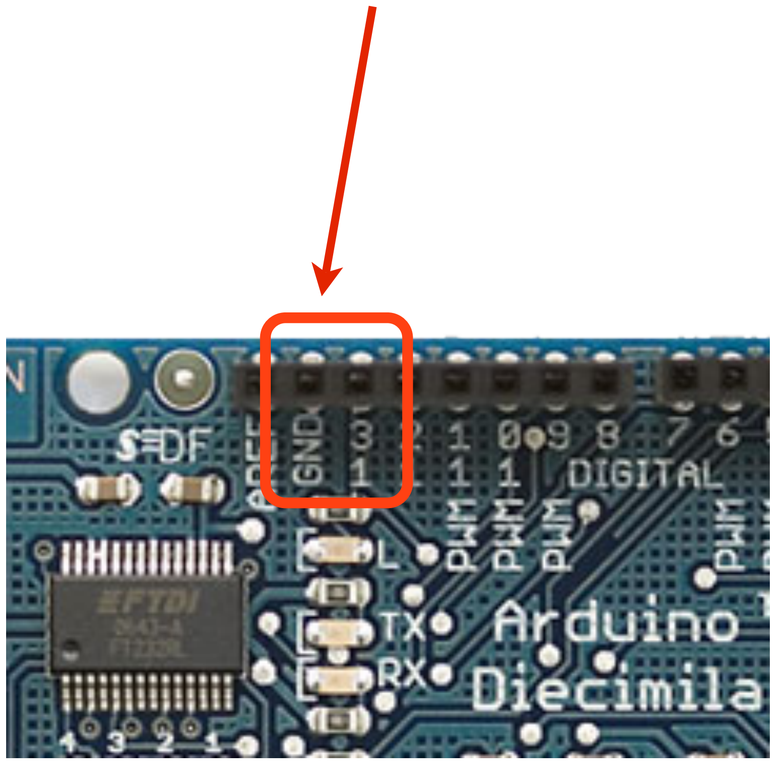
Hello Arduino!

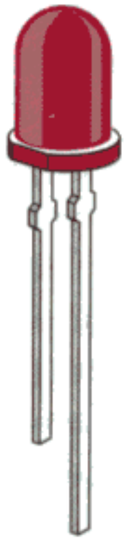
- Nella programmazione di sistemi embedded, il programma più semplice che è possibile scrivere svolga una sola funzione essenziale: accende e spegne un LED ad intervalli regolari

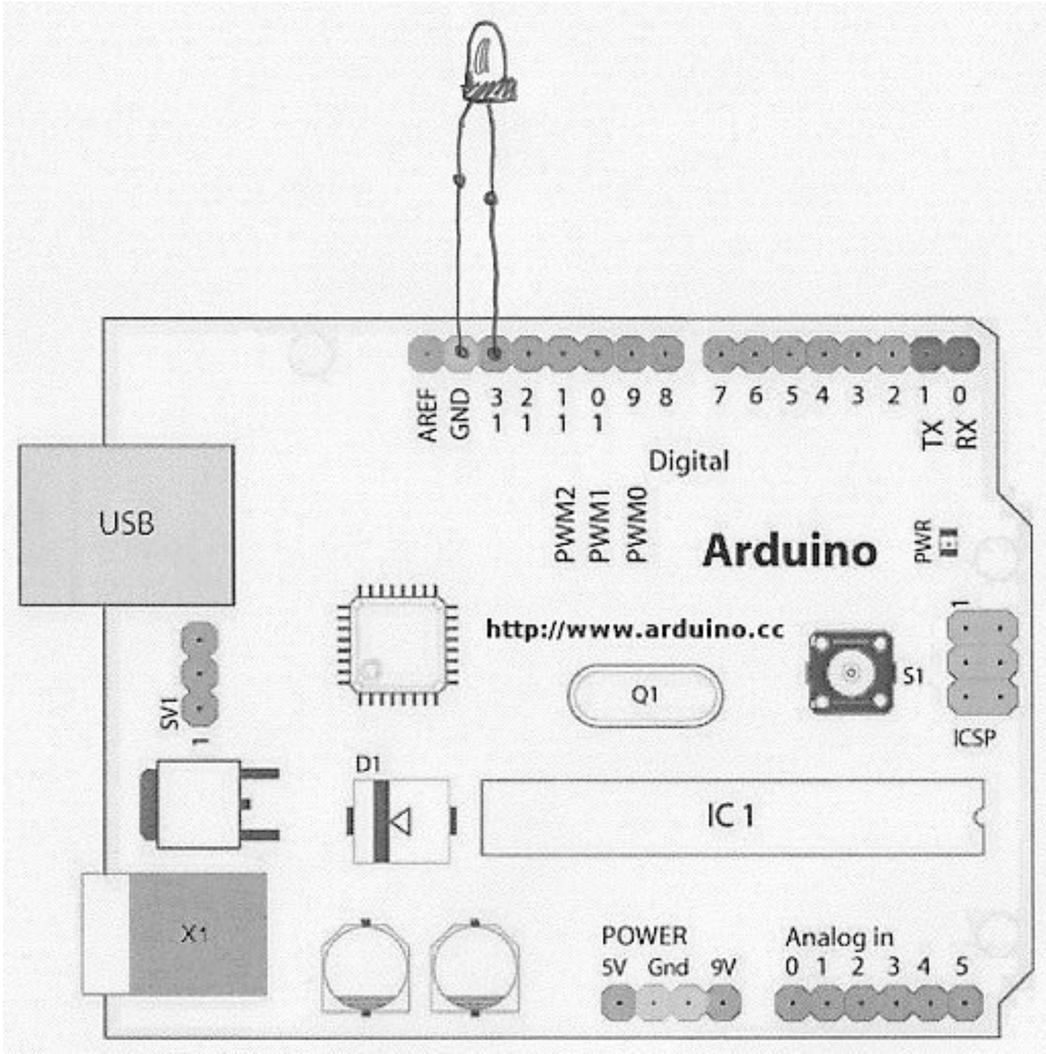
- Pur nella sua semplicità, questo esempio consente di conoscere il processo di sviluppo e di cablaggio di un circuito!

Hello Arduino!

PIN 13 e
PIN GND (ground, “massa”)







Hello Arduino!

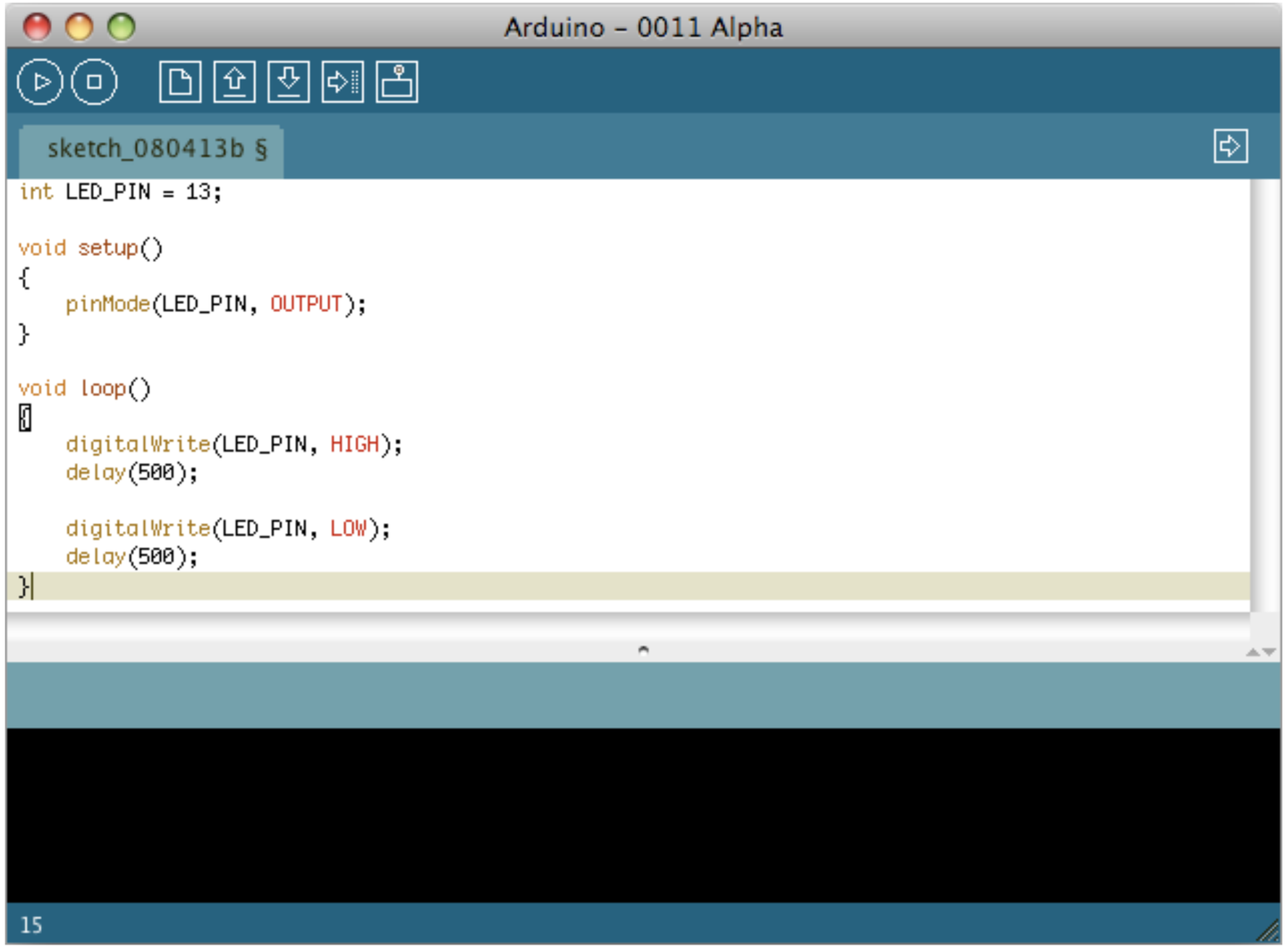
- Individuato il componente per la funzionalità richiesta (il LED) e stabilito il circuito (collegamento al PIN 13) è opportuno predisporre la logica

- Proviamo a descriverne il comportamento in un metalinguaggio...

```
0) Assegnare al PIN 13 una linea di
output

1) Accendere il LED (inviando un
segnale sul PIN 13)

2) Attendere mezzo secondo

3) Spegnere il LED (annullando il
segnale sul PIN 13)

4) Attendere mezzo secondo

5) Ripartire dal punto (1)
```

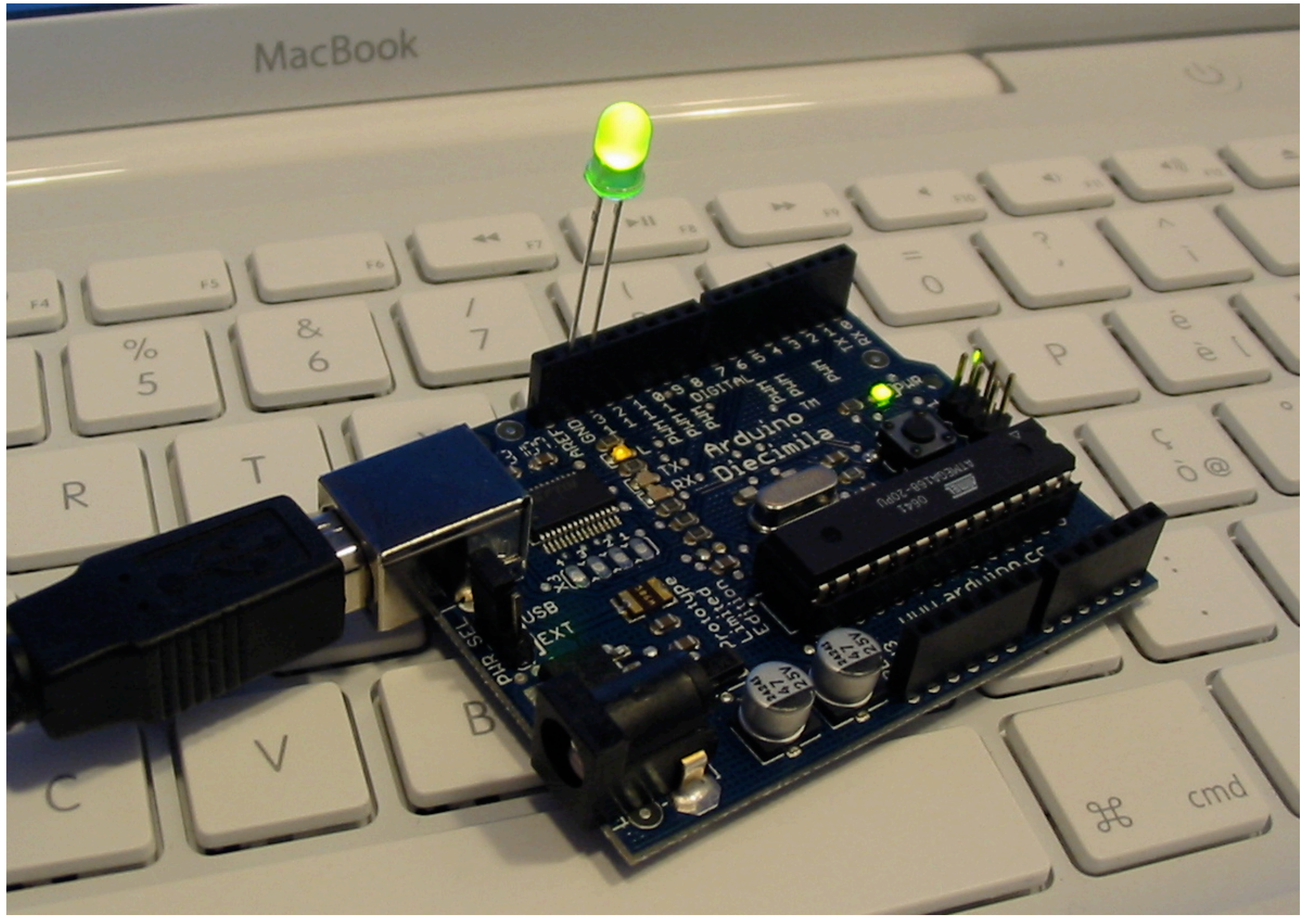
Hello Arduino!

```
int LED_PIN = 13;

void setup()
{
    pinMode(LED_PIN, OUTPUT);
}

void loop()
{
    digitalWrite(LED_PIN, HIGH);
    delay(500);

    digitalWrite(LED_PIN, LOW);
    delay(500);
}
```



Hello Arduino!

```
int LED_PIN = 13;

void setup()
{
    pinMode(LED_PIN, OUTPUT);
}

void loop()
{
    digitalWrite(LED_PIN, HIGH);
    delay(500);

    digitalWrite(LED_PIN, LOW);
    delay(500);
}
```



Struttura di un sketch

```
int LED_PIN = 13;
```

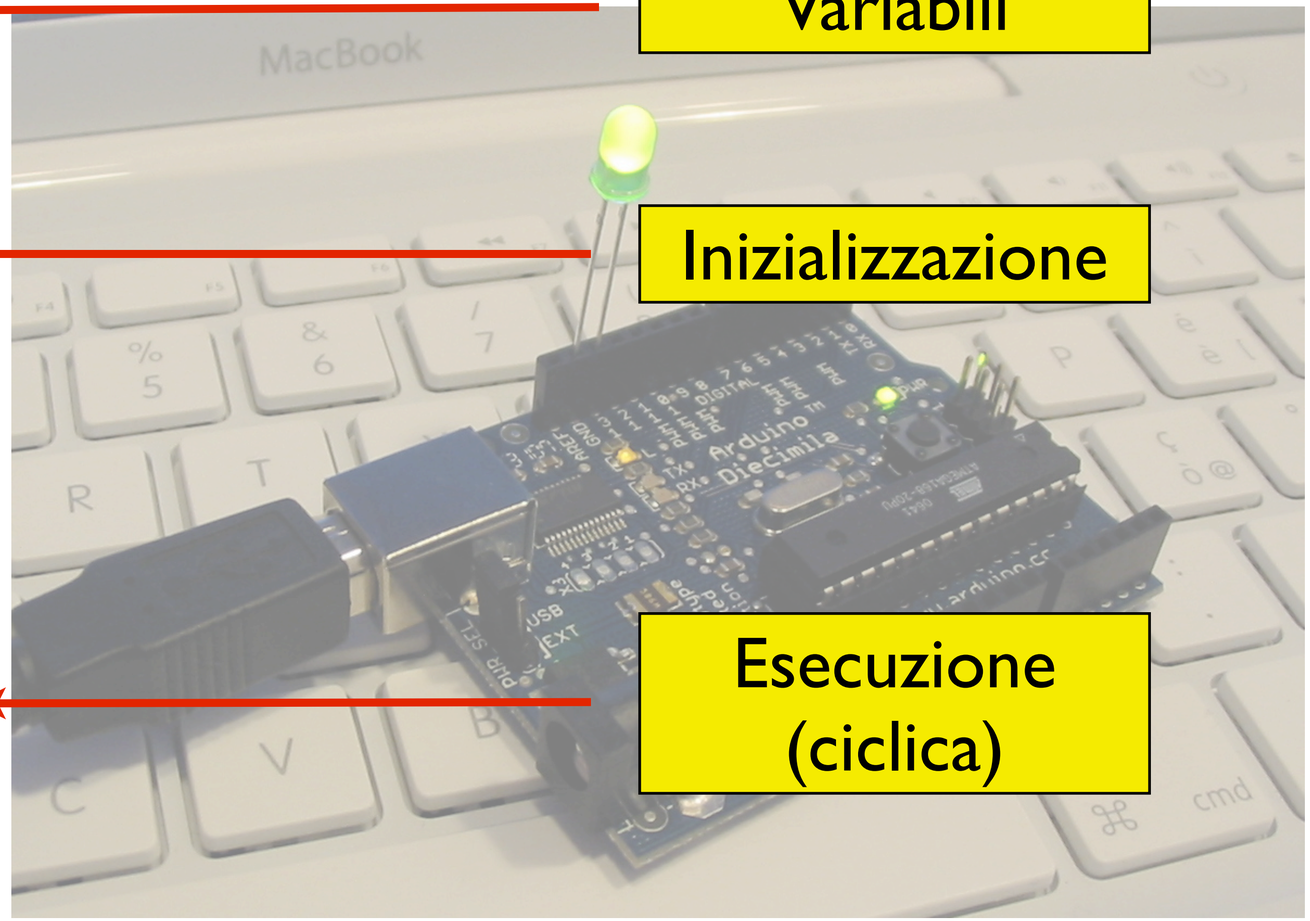
```
void setup()
{
    pinMode(LED_PIN, OUTPUT);
}
```

```
void loop()
{
    digitalWrite(LED_PIN, HIGH);
    delay(500);

    digitalWrite(LED_PIN, LOW);
    delay(500);
}
```

Variabili

Inizializzazione

Esecuzione (ciclica)



Commenti

- È possibile inserire commenti (annotazioni) nel sorgente per facilitare la lettura del codice o evidenziare una criticità

- I commenti sono **ignorati** dal compilatore ma sono molto utili ai programmatori per documentare il codice scritto

```
// commento su una sola riga

/*
commento su piu' righe
tutte ignorate dal compilatore
*/
```

Variabili

- Le variabili sono aree di memoria caratterizzate da un nome e da un tipo:
  - il nome ne consente l'utilizzo all'interno del programma
  - il tipo determina quali dati (numerici, logici, testuali) sono memorizzati
- Prima di essere utilizzata, una variabile è "**dichiarata**", ovvero espressa in nome e tipo

```
int temperatura;

float PI = 3.14;
```

Valore iniziale

Dichiarazione di una variabile:

tipo nome_variabile;

Il simbolo "punto e virgola" chiude la dichiarazione

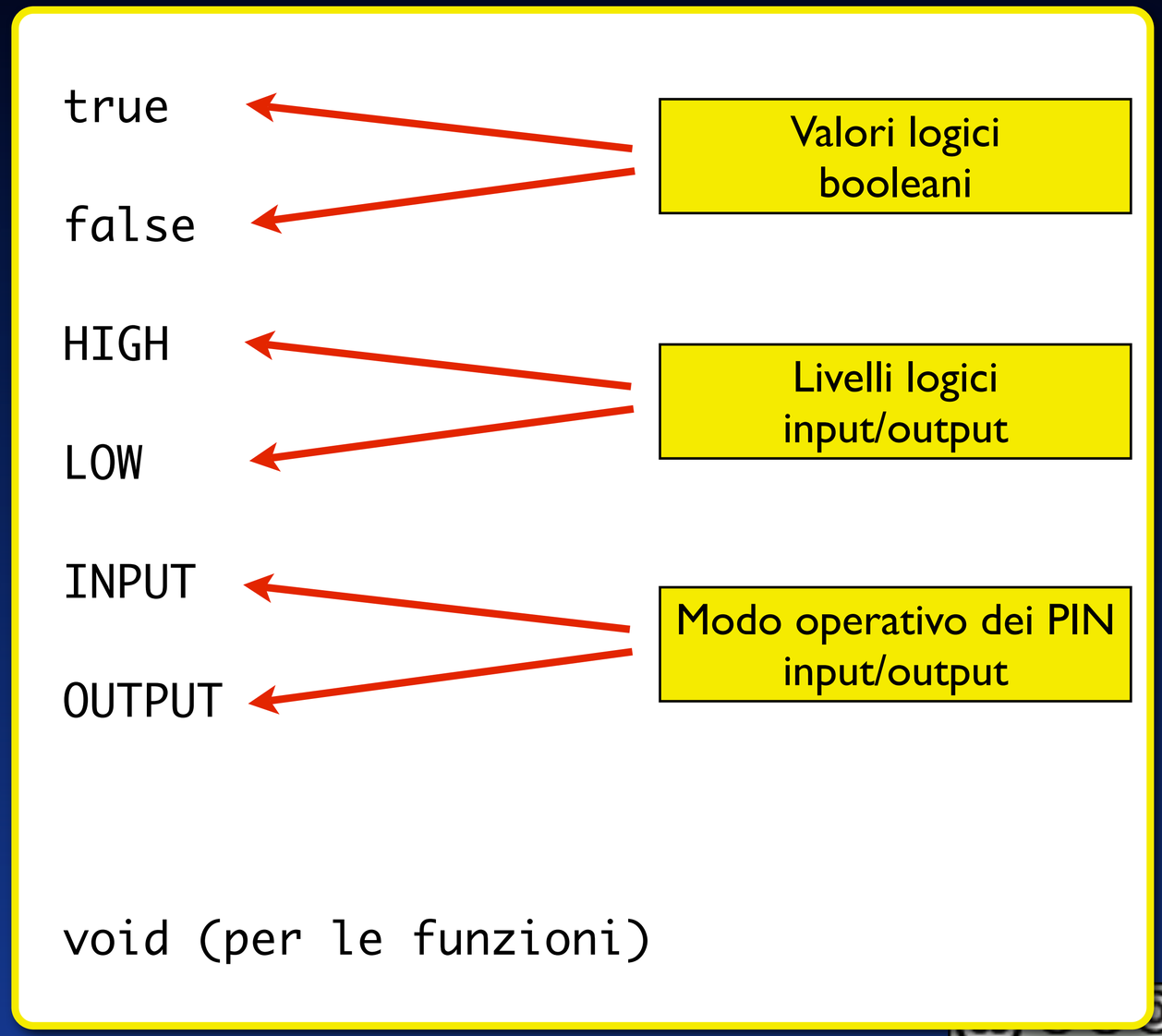
Tipi di variabili

- Il linguaggio di Arduino consente di dichiarare diversi tipi di variabili:
  - numerici (interi e virgola mobile)
  - logici (boolean, “true” e “false)
  - caratteri
  - vettori

```
int i = 5;

float PI = 3.14;

boolean ready = false;

char key = ‘x’;

char name[6] = “Robert”;
```

Costanti

- Il linguaggio di Arduino definisce alcuni costanti che esprimono:
  - le condizioni true/false dei booleani
  - lo stato HIGH/LOW di un PIN digitale
  - la modalità di lavoro di un PIN (ingresso o uscita)
  - l'assenza di valore di ritorno di una funzione



Tipi interi

- byte
  - 8 bit, 0:255
- int
  - 16 bit -32768:32767
- unsigned int
  - 16 bit, 0:65535
- long
  - 32 bit, -2147483468 : 2147483647
- unsigned long
  - 32 bit, 0 : 4294967295

```
unsigned int age = 25;

byte floor = 12;

long millis = 1208296770;
```

Variabili in virgola mobile

- float
  - 32 bit, -3.4028235E+38 e 3.4028235E +38
- double
  - 64 bit, massimo numero rappresentabile circa 2 x 10^308

```
float angle = 1.57;
```

Istruzioni

- Le istruzioni sono la più semplice porzione di un programma:
  - assegnazione: attribuzione di un valore ad una variabile
  - chiamata a funzione: esecuzione di una sequenza di istruzioni individuata da un nome
  - verifica condizioni sulle variabili
- Le istruzioni terminano con “;” (punto e virgola)

```
// assegnazione
int temperatura = 18;

// chiamata a funzione
avviaTermometro();

// chiamata e assegnazione
temperatura = leggiTemperatura();
```

Operatori

- Gli operatori agiscono sulle variabili producendo un risultato:
  - operatori matematici (somma, sottrazione...)
  - operatori di confronto
  - operatori logici
  -

```
// l'operatore matematico somma
a = b + c;

// l'operatore di confronto "maggiore"
a > b

// l'operatore logico AND
italian && young
```

Operatori aritmetici

- Sono disponibili 5 operatori per le operazioni aritmetiche:
  - somma: +
  - differenza: -
  - prodotto: *
  - quoziente: /
  - resto: %
- Il quoziente intero prevede arrotondamento

```
int a = 10;
int b = 3;
int c = 0;

c = a + b;     // c = 13

c = b - a;     // c = -7

c = a * b;     // c = 30

c = a / b;     // c = 3

c = a % b;     // c = 1

c = b % a;     // c = 3
```

Altri operatori

- Il linguaggio C offre degli operatori che effettuano contemporaneamente assegnazione e una operazione utilizzando un operando come variabile per il risultato

```
// equivale a: i = i + x;
i += x;

// equivale a: i = i - 1;
i -= x;

// equivale a: i = i * x;
i *= x;

// equivale a: i = i / x;
i /= x;
```

Incremento

- È possibile incrementare o descrementare di una unità una variabile utilizzando gli operatori **++** e **--**
- L'operatore prefisso incrementa il valore della variabile **prima** che sia usata
- L'operatore postfisso incrementa la variabile dopo che è stata utilizzata

```
int a;
int i = 5;

// incrementa "i" di 1 ("i" varrà 6)
i++;

// prima incrementa "i"
// poi assegna ad "a", che varrà 6
a = ++i;

// prima assegna ad "a", che varrà 5
// poi incrementa "i" (che varrà 6)
a = i++;
```

Operatori di confronto

- Consentono di stabilire la relazione d'ordine e uguaglianza tra valori numerici
- Il risultato è un valore logico true o false

```
int x;
int y;
boolean b;

b = (x == y);     // uguaglianza
b = (x != y);     // disuguaglianza

b = (x > y);      // maggiore
b = (x >= y);     // maggiore o uguale

b = (x < y);      // minore
b = (x <= y);     // minore o uguale
```

Operatori logici

- Consentono di effettuare operazioni sui valori booleani:
  - **AND**: restituisce **true** se entrambi gli operandi sono **true**
  - **OR**: restituisce **true** se almeno uno degli operandi è **true**
  - **NOT**: cambia valore logico ad una variabile boolean (se true diventa false, se false diventa true)

```
boolean a;
boolean t = true;
boolean t2 = true;
boolean f = false;

a = t && t2;     // a = true;
a = t && f;      // a = false;
a = t || f;      // a = true;

a = !t;          // a = false;
a = !f;          // a = true;
```

Blocchi

- Un blocco è una sequenza di istruzioni racchiusa tra una coppia di parentesi graffe { }
- I blocchi sono utilizzati per raggruppare le istruzioni da eseguire al verificarsi di una condizione o all'interno di un ciclo

```
{
    a = 12;

    b = c + d;

    leggiLivelloSegnale();

    avviaMotore();

    delay(1000);

    spegniMotor();
}
```

Funzioni

- Le funzioni sono **blocchi** di codice **individuati da un nome**
- Le funzioni possono avere uno o più parametri (argomenti) e possono restituire opzionalmente un valore

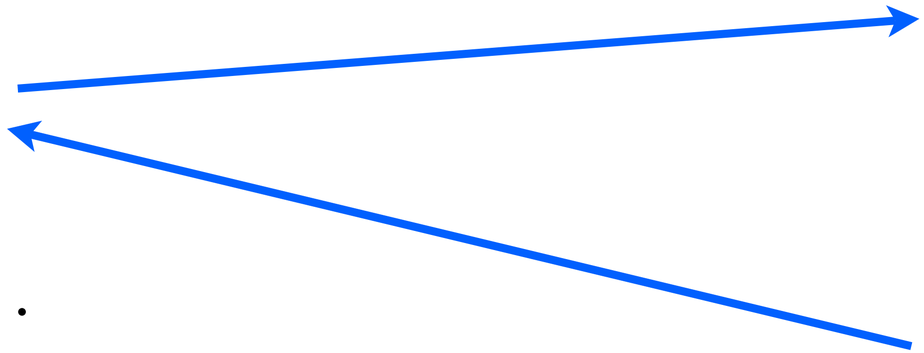
```
int leggiSegnale()
{
    // istruzioni

    return segnale;
}
```

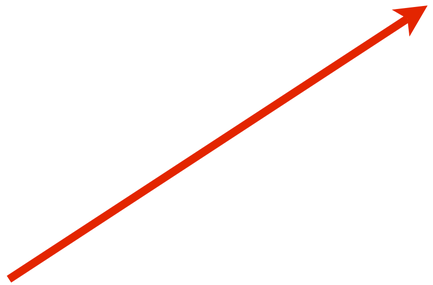
```
void avviaMotore()
{
    // istruzioni

    // nessun valore restituito
}
```

Chiamata a funzione

```
// istruzioni...

avviaMotore();

// istruzioni...
```

```
void avviaMotore()
{
    // istruzioni...
    // per l'avvio del motore
}
```





Quando nel codice vi è il riferimento ad una funzione, il programma "salta" al codice della funzione stessa, eseguendone il blocco corrispondente

Funzioni con parametri

- Una funzione può prevedere uno o più parametri, che sono visti all'interno del blocco come variabili
- I parametri, individuati da **tipo** e **nome**, consentono al programma di variare il comportamento della funzione dello stato corrente dell'esecuzione

```
void avviaMotore(int velocita)
{
    /*
    il parametro consente
    alla funzione di scegliere la
    velocita' desiderata dal programma
    */
}
```

Valore di ritorno

- Una funzione può terminare l'elaborazione senza un risultato in memoria: in tal caso, il nome è preceduto dalla keyword **void**

```
void avviaMotore()
{

    // istruzioni

}
```

Valore di ritorno

- Se la funzione restituisce un valore (come se fosse un operatore...), il suo tipo è indicato **prima** del nome della funzione (come se fosse una variabile) e assegnato attraverso **return**

```
int leggiSegnale()
{
    int segnaleSensore;

    // istruzioni che attribuiscono
    // un valore a “segnale”

   return segnaleSensore;
}
```

Funzioni matematiche

- Vanno parte delle funzionalità predefinite di Arduino
- Permettono di calcolare:
  - massimo e minimo
  - valore assoluto
  - estrazione della radice quadrata
  - elevazione a potenza
  - vincolo
  - range

```
int a = 0; int b = 16; int c = -3;

a = min(b, c);     // a = -3
a = max(b, c);     // a = 16
a = abs(c);        // a = 3

a = sqrt(b);       // a = 4

a = pow(c, 2);     // a = 9

a = constraint(b, 0, 10);     // a = 10
a = constraint(b, 0, 20);     // a = 16

a = map(b, 0, 20, 0, 10);     // a = 8
```

Funzioni di libreria

- Arduino dispone di funzioni predefinite che si occupano di:
  - effettuare le impostazioni sull'hardware (I/O, memoria, porta di comunicazione)
  - leggere i dati in ingresso
  - inviare dati in uscita
  - facilitare l'uso di funzionalità comuni (accesso a display, controllo di motori...)

```
//  I/O digitale
pinMode(pin, mode);
digitalWrite(pin, value);
int digitalRead(pin);

//  I/O analogico
int analogRead(pin);
analogWrite(pin, valore);

//  Temporizzazione
unsigned long millis();
delay(ms);
delayMicroseconds(us);
```

Controllo

- È possibile controllare il flusso di esecuzione delle istruzioni, verificando delle condizioni ed eseguendo i blocchi opportuni
- I controlli possono essere concatenati (if... else if...) o annidati

```
int temp;

temp = leggiTemperatura();

if (temp > 35 && temp < 70)
{
    avviaMotore();
}
else
{
    spegniMotore();
}
```

switch

- Talvolta è necessario effettuare un controllo su una variabile che può assumere valori costanti noti
- Il costrutto switch() consente di gestire i casi in maniera ordinata all'interno di blocchi **case** terminati da **break**

```
int numeroRuote;

switch (numeroRuote)
{
    case 1:
        //monociclo
        break;
    case 2:
        //biciclo
        break;
    case 3:
        //triciclo
        break;

}
```

Cicli

- Consentono di ripetere in blocco di codice per un numero predefinito di volte o finché una certa condizione è vera
- Il ciclo for() qui accanto esegue dieci volte il blocco di codice tra graffe

```
int i;

for (i = 0; i < 10; i++)
{
    powerLedOn();
    doSound();

    delay(1000);

    powerLedOff();
    stopSound();

    delay(1000);

}
```

Ciclo while()

- Esegue il blocco di codice finché la condizione in argomento è true

```
while (temp > 12)
{
    if (motorOff)
    {
        powerOnMotor();
    }
}
```

continue, break

- All’interno di un ciclo è possibile forzare l’avanzamento all’interazione successiva o interrompere il blocco

- continue: considera conclusa interazione corrente e salta all successiva
- break: esce dal ciclo

```
int i;
for (i = 0; i < 10; i++)
{
    // istruzioni che leggono “x”
    if (x > 100)
    {
        // salta all’i successivo
        continue;
    }

    if (x < 0)
    {
        break;
    }
}
```

Rivediamo Hello W

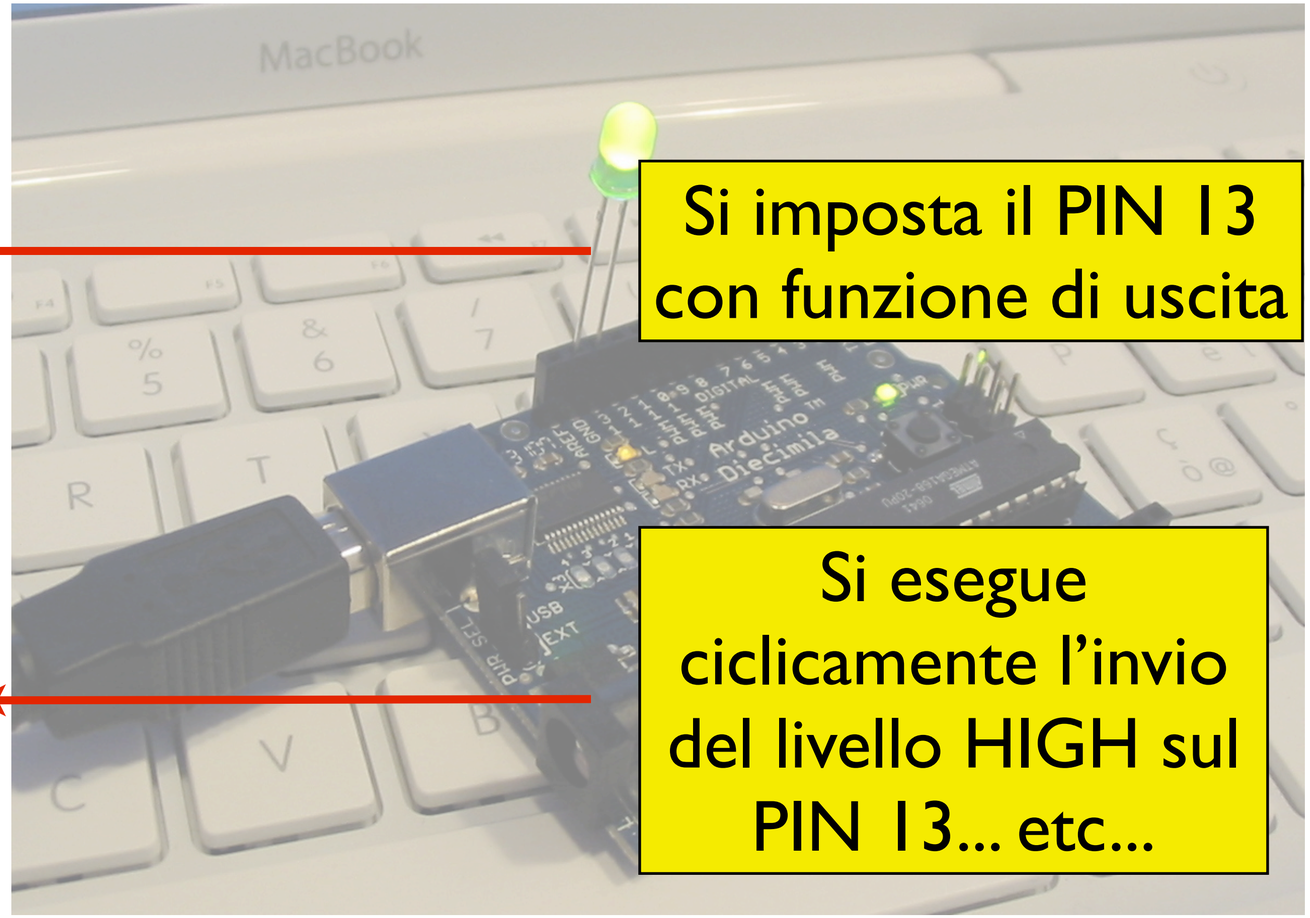
```
int LED_PIN = 13;
```

Si dichiara una variabile intera LED_PIN che ha valore 13

```
void setup()
{
    pinMode(LED_PIN, OUTPUT);
}
```

Si imposta il PIN 13 con funzione di uscita

```
void loop()
{
    digitalWrite(LED_PIN, HIGH);
    delay(500);

    digitalWrite(LED_PIN, LOW);
    delay(500);
}
```



Si esegue ciclicamente l'invio del livello HIGH sul PIN 13... etc...

setup( )

- setup() è la funzione eseguita **una sola volta** all'avvio del programma
- È responsabile delle impostazioni iniziali di Arduino:
  - gestione dei PIN di ingresso/uscita
  - configurazione della porta seriale
  - inizializzazione librerie

```
void setup()
{
   // PIN 13 di uscita
   pinMode(13, OUTPUT);

   // PIN 2 di ingresso
   pinMode(2, INPUT)
}
```

loop()

- È il “cuore” di uno sketch: è la funzione eseguita ciclicamente finché Arduino è alimentato (equivale ad un ciclo while() infinito)

```
void loop()
{
    readSensorData();

    if (temp > 100)
    {
        powerOnFan();
    }
    else if (temp < 40)
    {
        powerOffFan();
    }

    // riparte da capo...
}
```

Prima di cominciare...

- I circuiti di seguito descritti sono semplici e alimentati a bassa tensione. Questo riduce i rischi di danni alle persone e agli oggetti, ma è necessario adottare opportune cautele:
  - i componenti elettronici dispongono di estremità metalliche accuminate che possono facilmente lacerare la cute se premuti con forza o provocare abrasioni se inavvertitamente proiettati verso gli occhi
  - Arduino dispone di un circuito di protezione della porta USB: è comunque consigliato, prima di effettuare modifiche al prototipo, scollegare la scheda dal computer e da qualsiasi altra fonte di alimentazione, verificando la nuova configurazione prima di riavviare la scheda
  - se si utilizza un saldatore, attenersi scrupolosamente al manuale d'uso, avendo cura di tenere lontana la punta calda dai cavi di alimentazione

Serie e parallelo

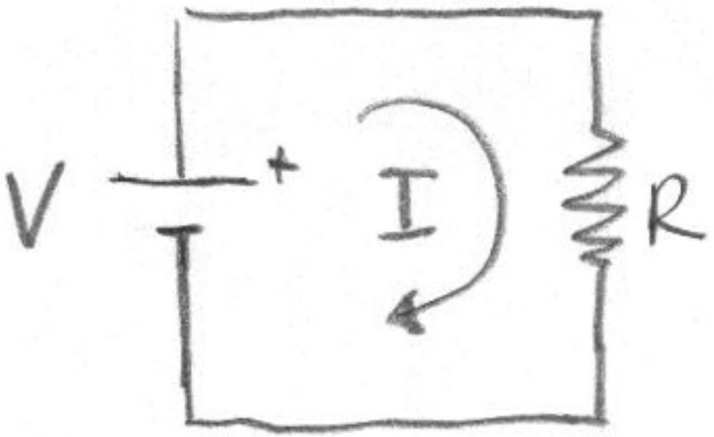
- Due componenti bipolari (con due soli terminali) si dicono connessi in **serie** quando l'uscita del primo è connessa all'ingresso del secondo, ovvero quando sono attraversati dalla stessa corrente. Due componenti in serie hanno un solo terminale in comune.

- Due componenti bipolari si dicono connessi in **parallelo** quando sono soggetti alla stessa tensione e hanno in comune ingressi e uscite.

Resistore

- È un componente passivo che provoca una caduta di tensione ai suoi capi

- La corrente circolante nel circuito è proporziale alla tensione e inversamente proporzionale alla resistenza (misurata in Ohm)



$$I = \frac{V}{R}$$

Codici colori delle resistenze

- Il valore della resistenza di un resistore è inciso sul corpo del componente attraverso una sequenza di bande colorate
- Ad ogni colore è assegnato una cifra, mentre ad ogni banda è assegnato un significato

Colore	1° Anello	2° Anello	3°Anello	4° Anello
	Cifra 1	Cifra2	Moltiplicatore	Tolleranze
-	-	-	-	± 20%
argento	-	-	10^{-2}	± 10%
oro	-	-	10^{-1}	± 5%
nero	0	0	10^{0}	-
marrone	1	1	10^{1}	± 1%
rosso	2	2	10^{2}	± 2%
arancio	3	3	10^{3}	-
giallo	4	4	10^{4}	-
verde	5	5	10^{5}	± 0,5%
blu	6	6	10^{6}	± 0,25%
viola	7	7	10^{7}	± 0,1%
grigio	8	8	10^{8}	± 0,05%
bianco	9	9	10^{9}	-

LED

- Un **LED** (light-emitting diode) è un dispositivo a semiconduttore che emette **luce** quando attraversato da una corrente continua
- Il LED conduce ("fa passare corrente", dunque si illumina) solo se collegato correttamente: anodo verso il polo positivo, catodo verso quello negativo
- Una corrente eccessiva può **danneggiarlo**: è consigliato alimentarlo attraverso una resistenza che limiti la corrente circolante a 20mA

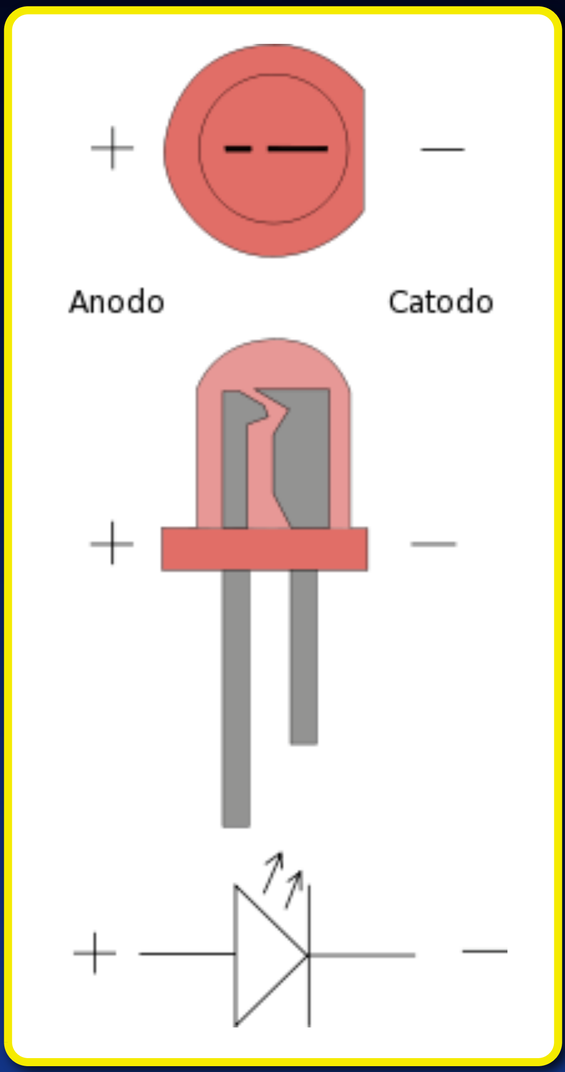
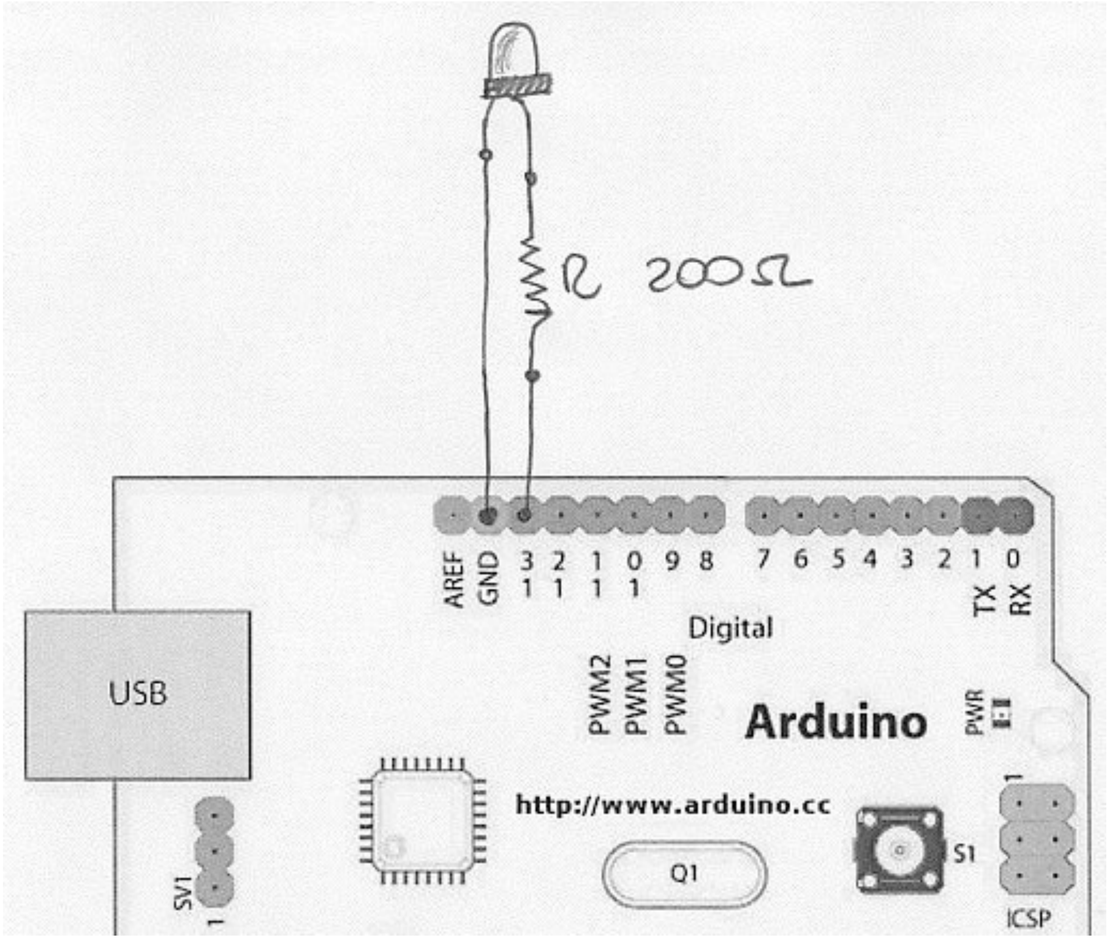


Immagine tratta da Wikipedia

Rivediamo Hello Arduino!

- Per limitare la corrente che attraversa il LED, è necessario porre una resistenza in serie
- Essendo le uscite a 5V, è sufficiente porre una resistenza di 220 ohm

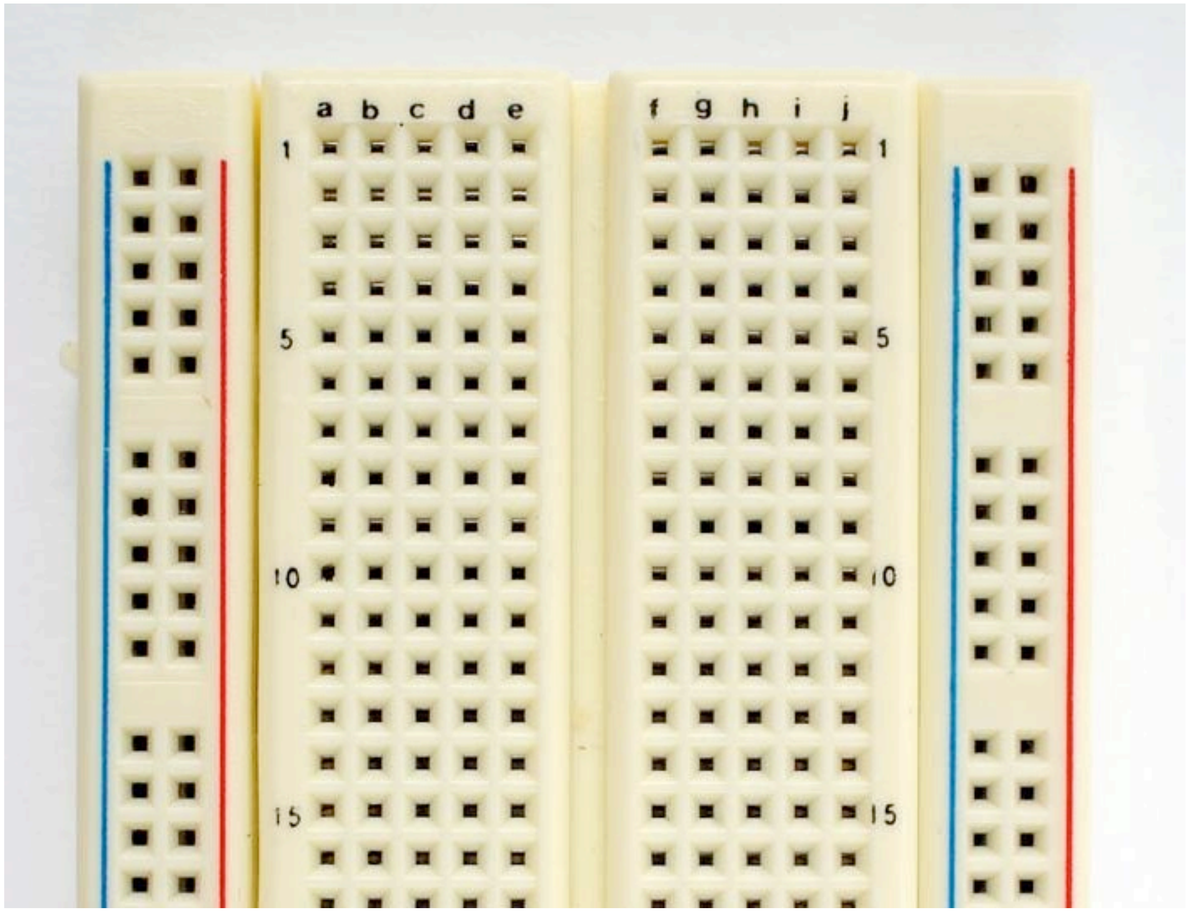


Cablaggio dei circuiti

- La realizzazione dei circuiti richiede il collegamento elettrico tra i componenti ed un adeguato **isolamento** delle connessioni dal resto dell'ambiente
- Nei casi più semplici è possibile usare "collegamenti volanti" con morsettiere o piccole saldature
- Al crescere della complessità del circuito e volendolo modificare spesso con la certezza di riciclare i componenti, è consigliabile utilizzare schede sperimentali (**breadboard**)

Breadboard

- Una scheda sperimentale consente la realizzazione di circuiti elettrici senza necessità di saldature o modifiche ai componenti
- Il cablaggio avviene collocando i componenti ed effettuando i collegamenti con piccoli spezzoni di filo metallico

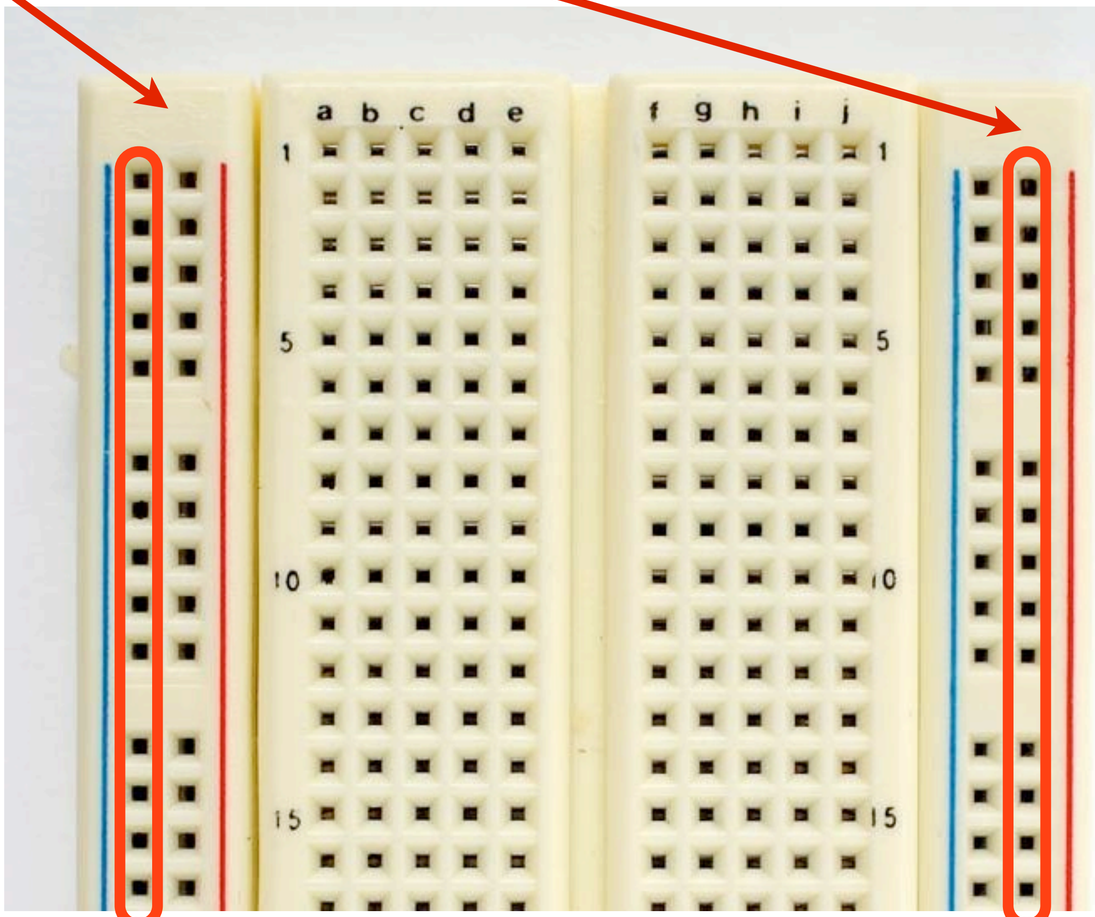


Breadboard

Linee di alimentazione

- Le linee di **alimentazione** sono continue verticalmente e consentono di distribuire massa e tensione positiva lungo tutto il circuito

- Lungo tutta la breadboard è possibile “prelevare” corrente laddove occorre

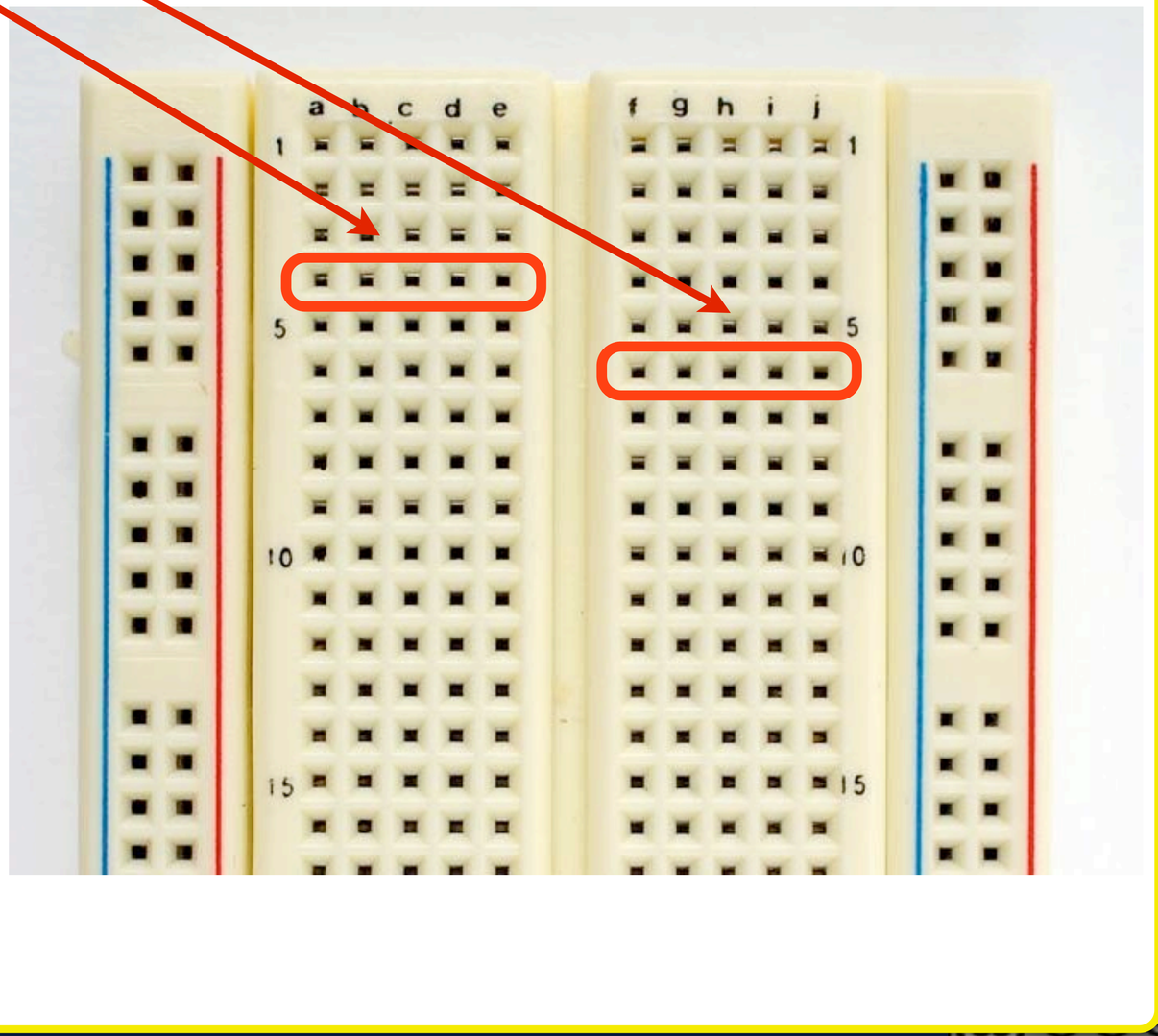


Breadboard

Bus dei componenti

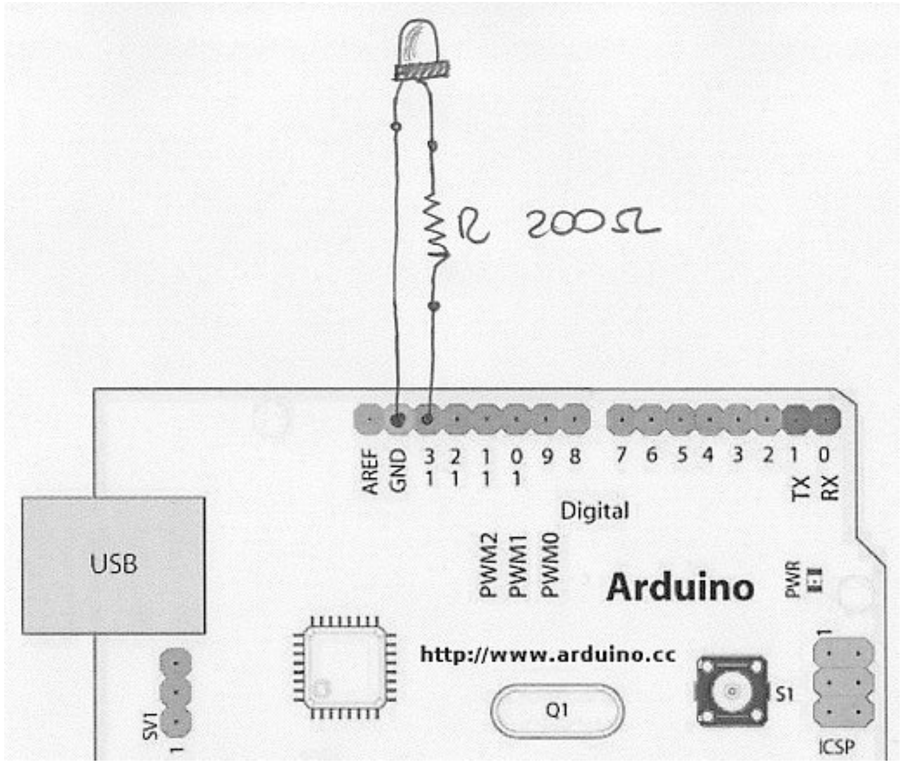
- Il **bus** dei componenti si estende sulle righe orizzontali di ciascuna sezione

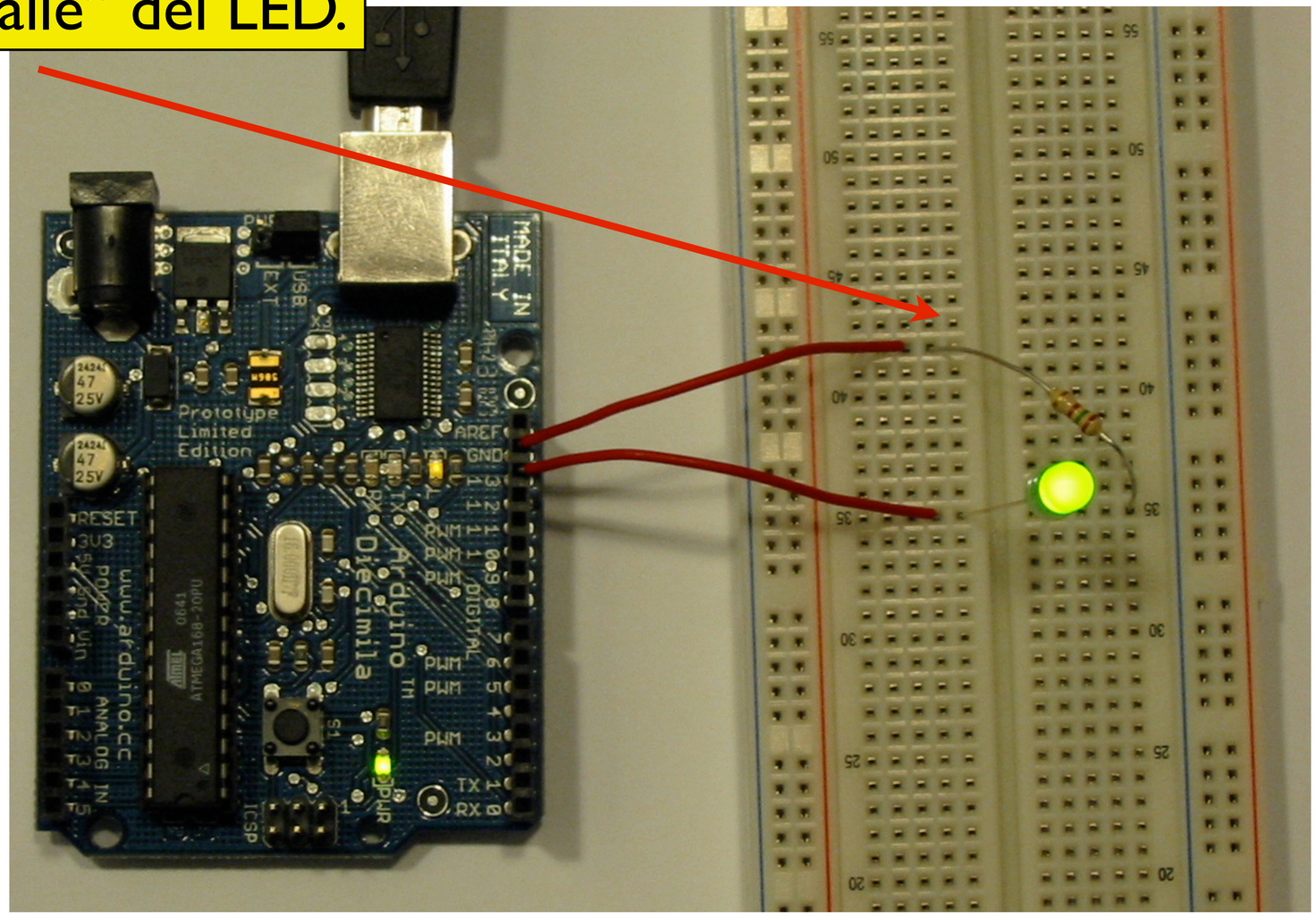
- Un componente può essere collocato “a cavallo” tra due sezioni per sfruttare due file di pin paralleli



Hello Arduino su breadboard

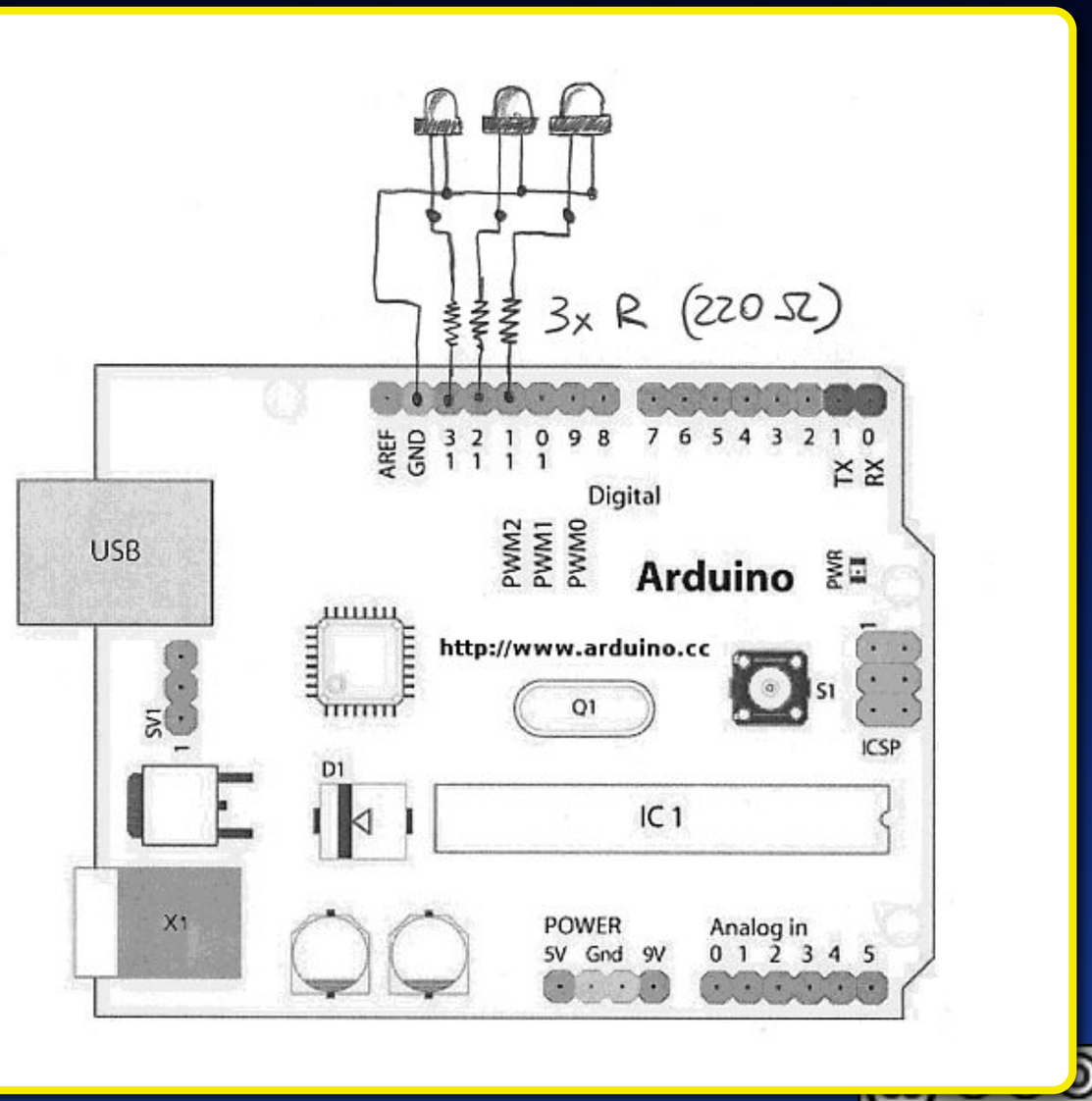
Nota: la resistenza è stata posta “a valle” del LED.





Un semplice semaforo

- L'esempio appena visto può essere esteso per realizzare un semplice semaforo
- Tre LED (verde, giallo, rosso) sono accesi in sequenza:
  - verde acceso, gli altri spenti
  - dopo 2 secondi, si accende il giallo
  - dopo due secondi, si spengono verde e giallo, si accende il rotto
  - dopo due secondi, riparti da capo



Sketch [1]

- Il primo passo consiste nel definire tre variabili corrispondenti ai PIN che si vogliono utilizzare e nell'impostarli come uscite digitali

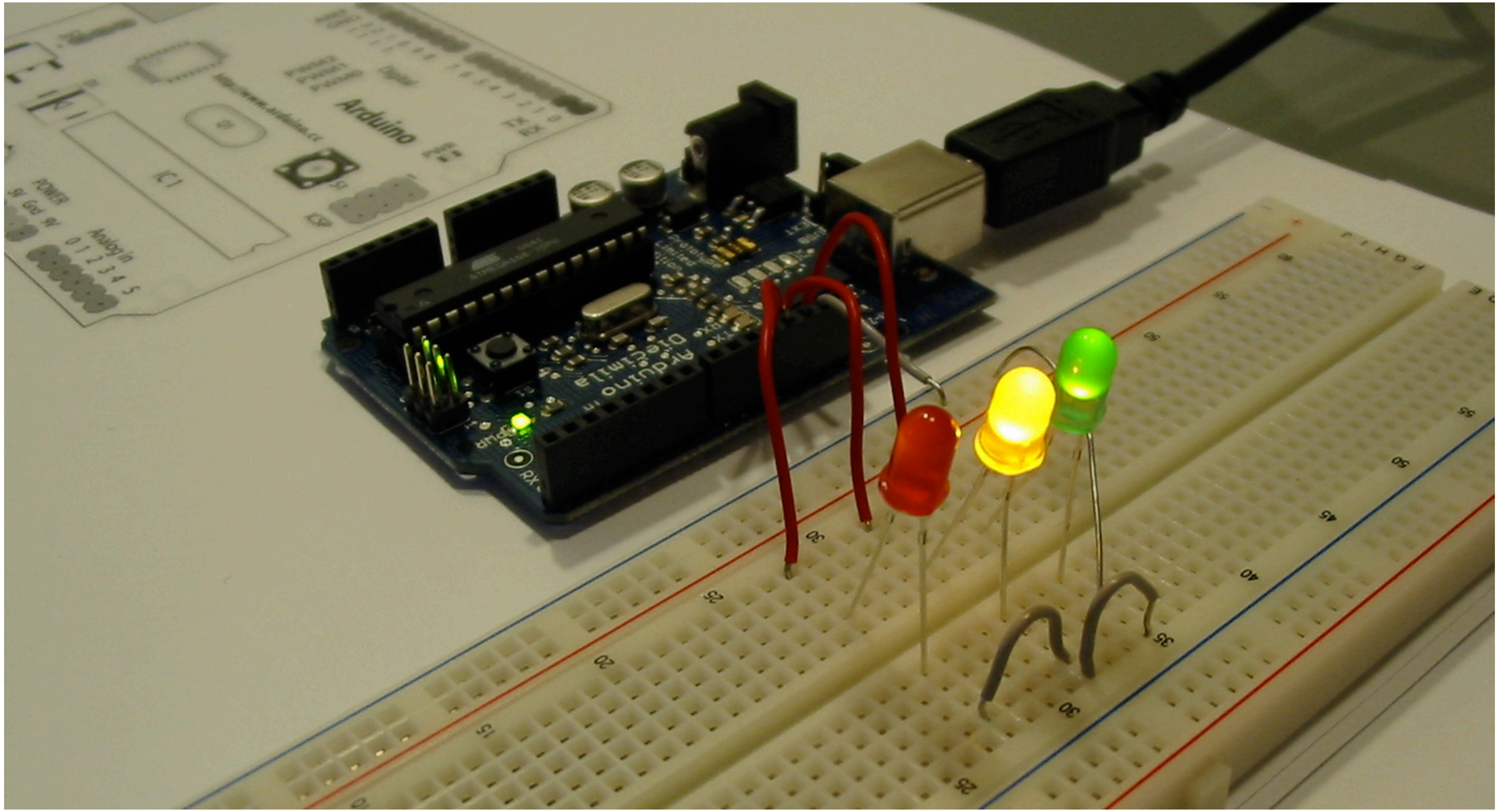
```
int GREEN_LED = 13;
int YELLOW_LED = 12;
int RED_LED = 11;

void setup()
{
    pinMode(GREEN_LED, OUTPUT);
    pinMode(YELLOW_LED, OUTPUT);
    pinMode(RED_LED, OUTPUT);
}
```

Sketch [2]

- Il loop() del programma è costituito dai seguenti passi:
  - si spengono rosso e giallo, si accende il verde; si aspetta due secondi
  - si accende il giallo (verde e rosso non cambiano stato, restando rispettivamente acceso e spento);si aspetta altri due secondi
  - si accende il rosso, si spengono gli altri; si aspetta due secondi
- Il semaforo è pronto!

```
void loop()
{
    digitalWrite(RED_LED, LOW);
    digitalWrite(YELLOW_LED, LOW);
    digitalWrite(GREEN_LED, HIGH);
    delay(2000);

    digitalWrite(YELLOW_LED, HIGH);
    delay(2000);

    digitalWrite(RED_LED, HIGH);
    digitalWrite(YELLOW_LED, LOW);
    digitalWrite(GREEN_LED, LOW);
    delay(2000);
}
```

Semaforo su breadboard



Analogico vs Digitale

- I segnali in ingresso e in uscita possono essere analogici o digitali
- Un segnale **analogico** può assumere qualsiasi valore (all'interno di un range noto). Con notevole semplificazione, si può pensare che siano "analogiche" tutte le grandezze fisiche misurabili nell'ambiente
- Un segnale **digitale** può assumere due soli stati (HIGH - LOW), corrispondenti a due livelli di tensione convenzionali (ad esempio, 0-5V). Con simile semplificazione, si può pensare che siano "digitali" tutte le informazioni scambiate tra componenti logici (microprocessori, memorie, interfacce di rete, display...)

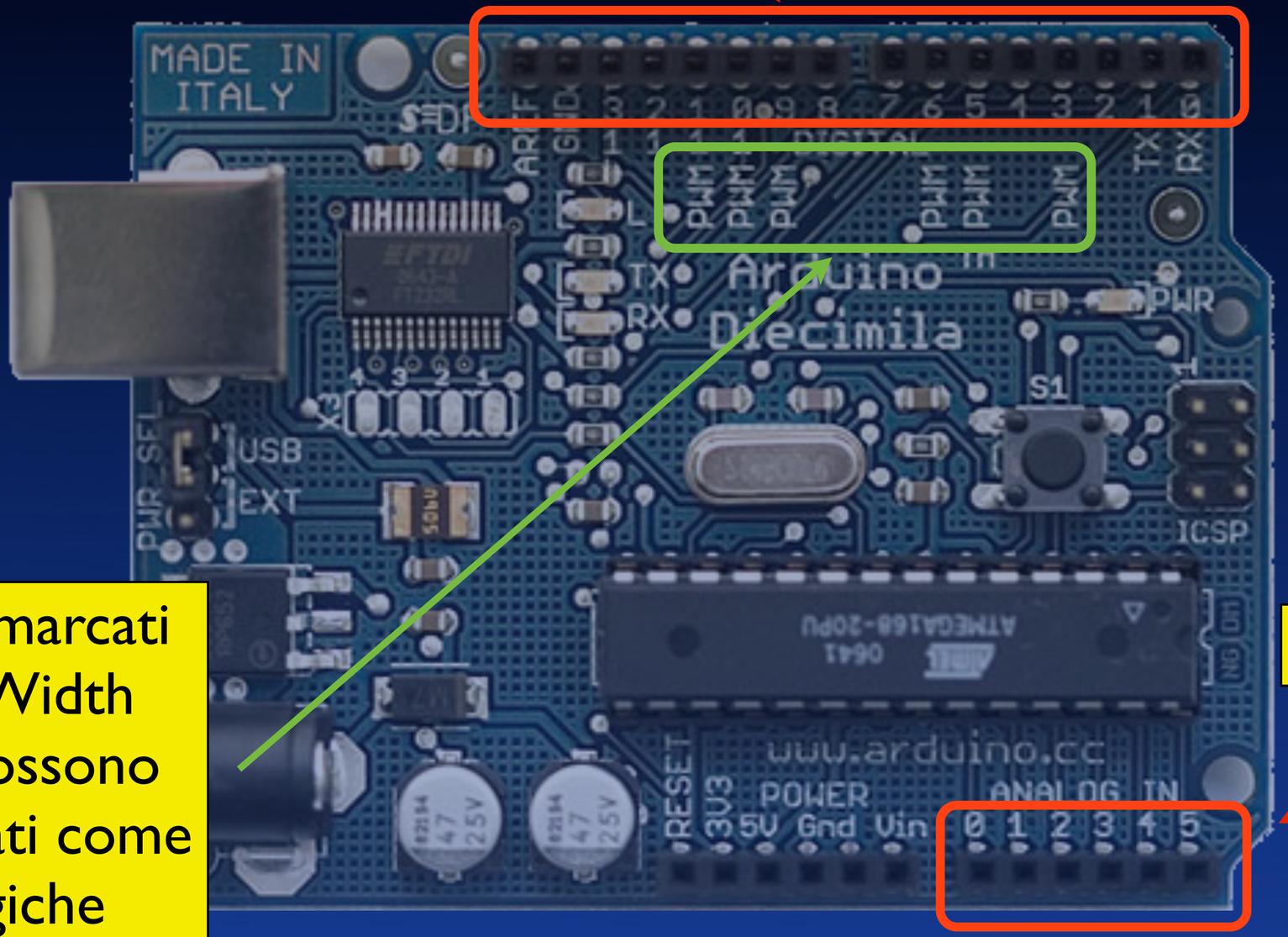
Analogico vs Digitale

- Affinché possa essere letto ed elaborato da Arduino, il segnale analogico deve essere **campionato**, ovvero **convertito** in in una sequenza di bit che ne esprime l'ampiezza

- Un segnale digitale è immediatamente "leggibile" non appena ne è stato discriminato il livello (HIGH - LOW).

Interfacce

Input/Output digitali



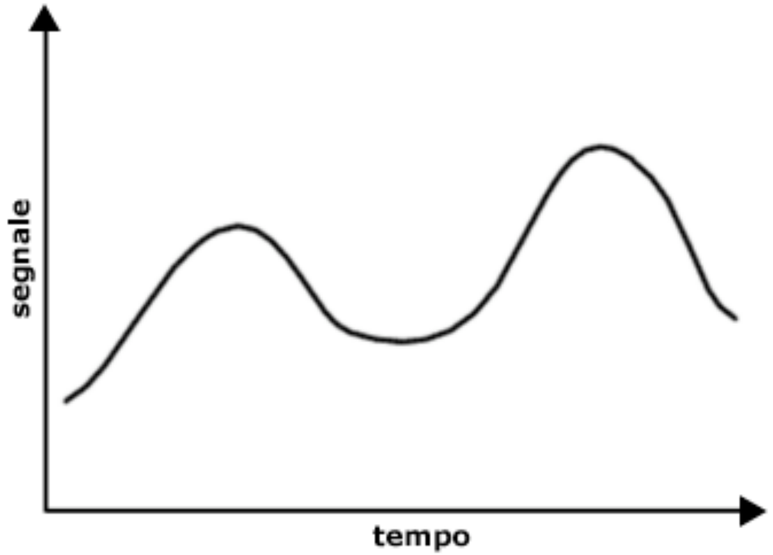
I 6 PIN digitali marcati PWM (Pulse Width Modulation) possono essere configurati come uscite analogiche

Ingressi analogici

Analogico vs Digitale

- Del segnale analogico interessa leggere il valore istantaneo, opportunamente campionato

- Del segnale digitale occorre sapere solo lo stato alto o basso



Analogico vs Digitale

- I PIN digitali devono essere preventivamente impostati in modalità input o output
- Lettura e scrittura avvengono attraverso funzioni dedicate:
  - digitale: digitalWrite() e digitalRead()
  - analogico: analogWrite() e analogRead()

```
pinMode(13, OUTPUT);

pinMode(2, INPUT);


digitalWrite(13, HIGH)

int button = 0;
button = digitalRead(2);

int temp;
temp = analogRead(0);
```

Input

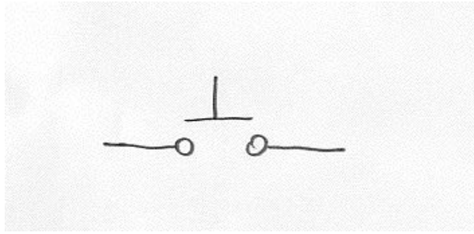
- Arduino dispone di linee di ingresso analogiche e digitali per l'interfacciamento a sensori
- Esempi di componenti di input:
  - pulsante (apertura/chiusura di un circuito)
  - potenziometro (variazione di resistenza in funzione di rotazione/traslazione)
  - fotoresistenza: (resistenza in funzione dell'intensità luminosa)
  - sensore di temperatura
  - accelerometro
  - lettore RFID

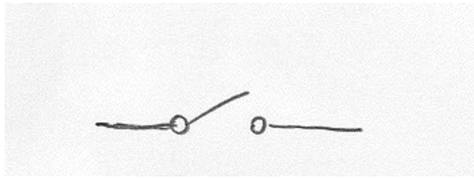
Output

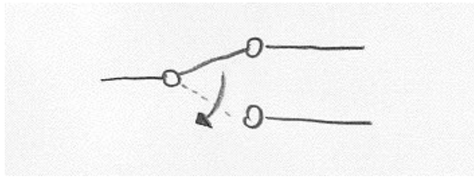
- Arduino dispone di linee di uscita digitali (livello alto/basso) e analogiche (PWM), per il controllo di attuatori
- Esempi di componenti di output:
  - LED (luce)
  - display LCD (testo e grafica)
  - buzzer (suono)
  - motori (direct/servo/stepper, movimento rotatorio/traslatorio)
  - qualsiasi dispositivo controllabile con un driver dedicato

Interruttori

- Pulsante
  - chiude il circuito finché premuto, per poi riaprirlo al rilascio
- Interruttore
  - chiude o apre il circuito in maniera permanente
- Commutatore (deviatore)
  - devia la corrente tra due rami di un circuito; l'azione è permanente, fino alla successiva commutazione

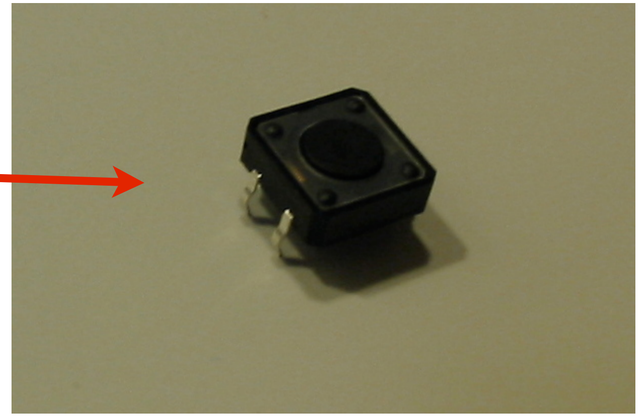


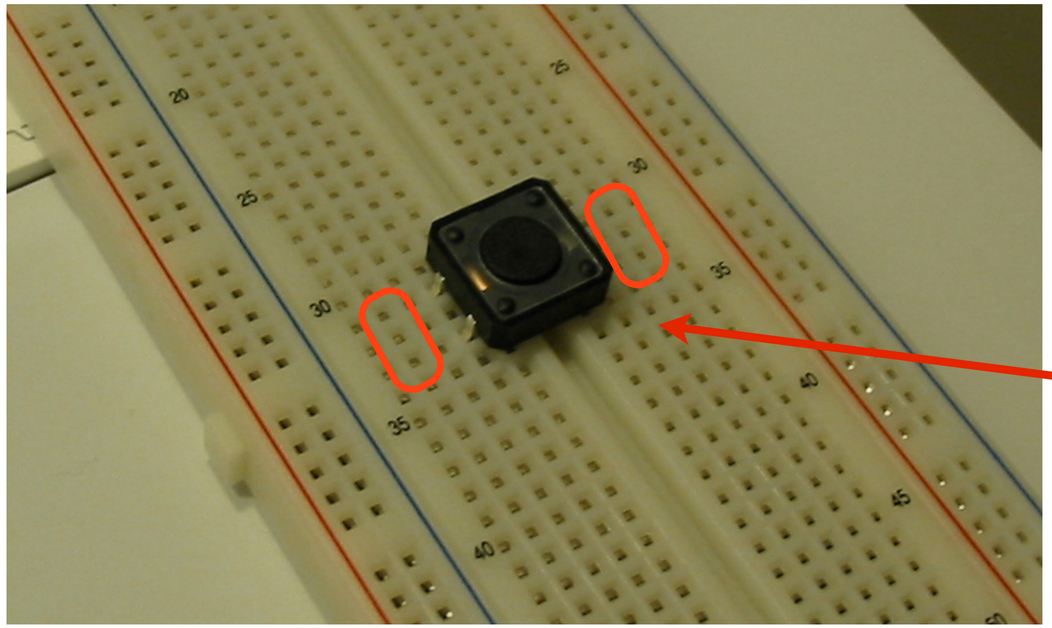




Pulsante per breadboard

Il pulsante chiude il circuito ai capi delle coppie di PIN più vicine

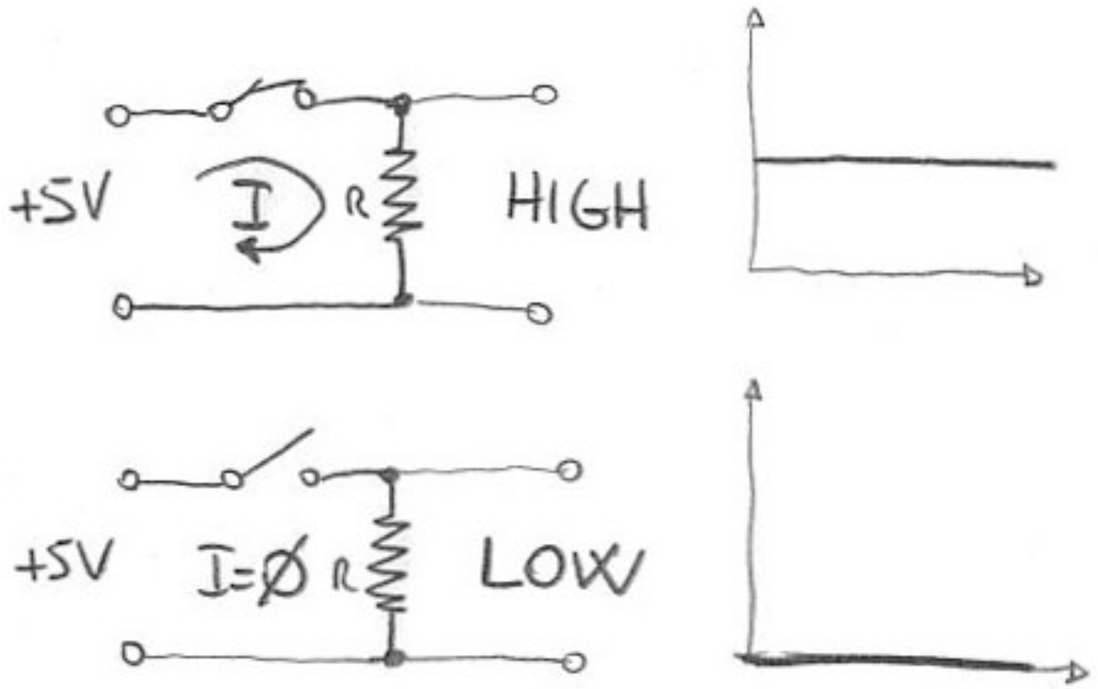




Il pulsante si pone a cavallo dei due bus. Alla pressione del pulsante, su entrambi i bus sono cortocicuitati i PIN

Inviare i livelli HIGH e LOW

- Utilizzando una resistenza ed un interruttore è possibile realizzare un semplice circuito che invia i livelli HIGH e LOW
  - HIGH: il circuito è chiuso, la corrente circola all'interno della resistenza ai cui capi è presente la differenza di potenziale corrispondente al livello alto
  - LOW: il circuito è aperto, non vi è alcune ddp ai capi della resistenza

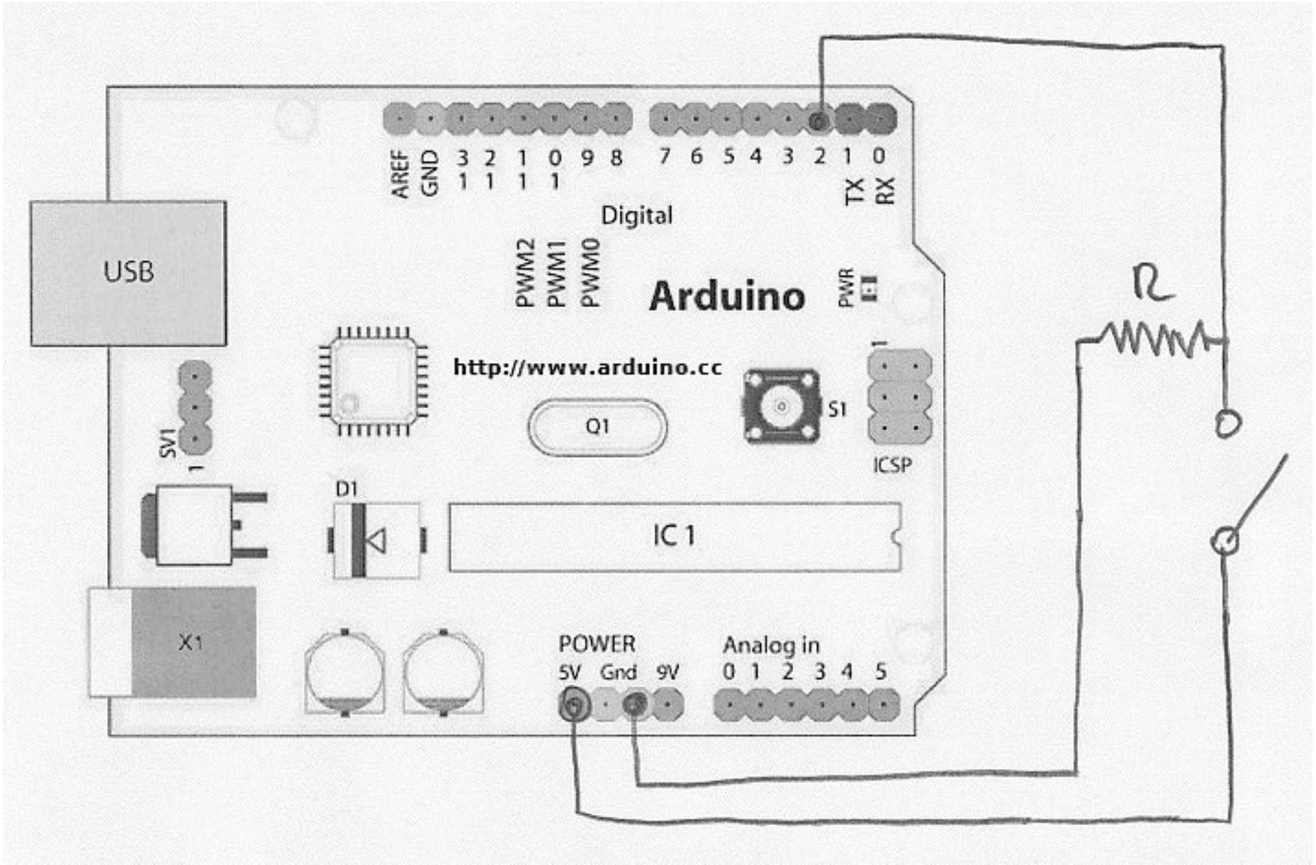


Alimentare il circuito

- Arduino dispone di due PIN a 5V e 3.3V che possono essere utilizzati per alimentare piccoli componenti del circuito realizzato sulla breadboard

Uscita 5V

Il circuito

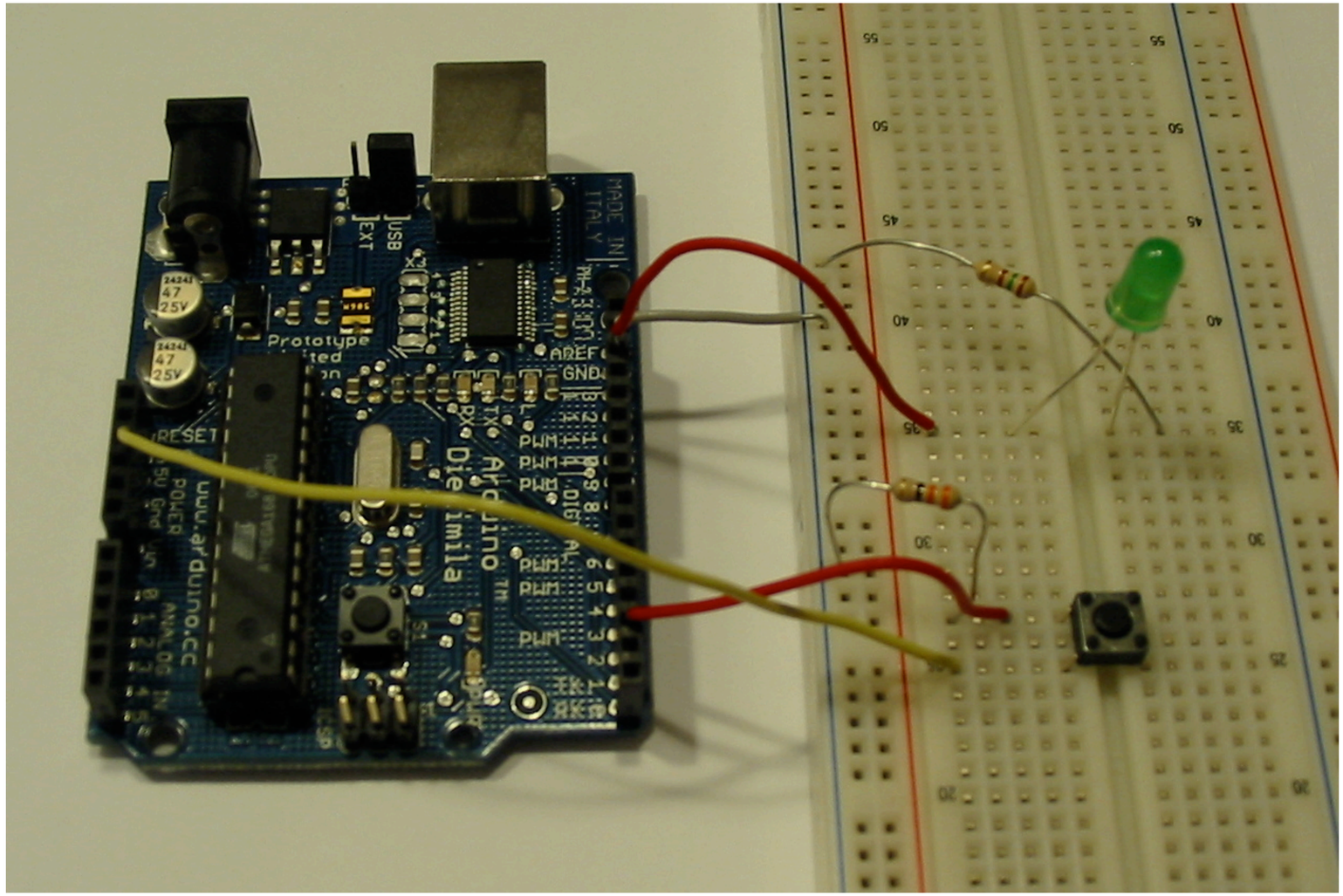

AREF
GND
3 2 1 1 1
0 9 8
7 6 5 4 3 2 1 0
TX
RX
Digital
PWM2
PWM1
PWM0
Arduino
PWR
USB
http://www.arduino.cc
SV1
Q1
S1
ICSP
D1
IC 1
X1
POWER
5V Gnd 9V
Analog in
0 1 2 3 4 5
R

Input digitale

- Un ingresso digitale può assumere i due valori LOW (basso) o HIGH (alto)
- I due valori corrispondono rispettivamente ad una tensione nulla (GND) o positiva (5V nominali)

```
int BTN_PIN = 2;
int LED_PIN = 13;
int state = LOW;

void setup()
{
    pinMode(BTN_PIN, INPUT);
    pinMode(LED_PIN, OUTPUT);
}

void loop()
{
    int state = digitalRead(BTN_PIN);
    digitalWrite(LED_PIN, state);
}
```

Il prototipo



Variante (I)

- Il LED lampeggia con un intervallo di mezzo secondo
- Alla pressione del pulsante, il LED lampeggia più velocemente

```
int BTN_PIN = 2;
int LED_PIN = 13;
int state = LOW;
int ledDelay = 500;

void setup()
{
    pinMode(BTN_PIN, INPUT);
    pinMode(LED_PIN, OUTPUT);
}
```

Variante (II)

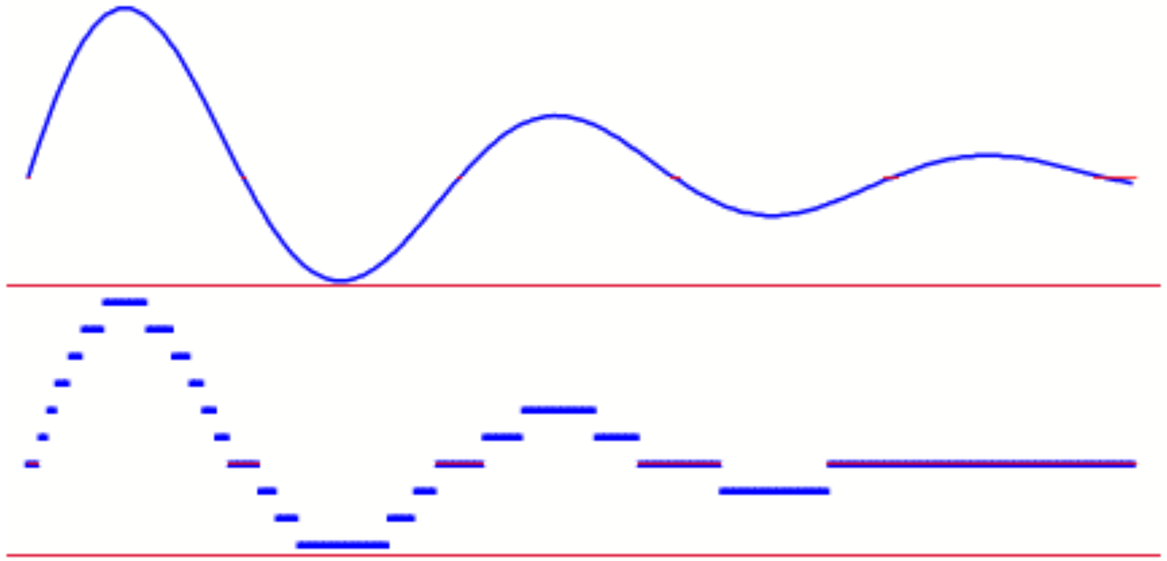
- Il LED lampeggia con un intervallo di mezzo secondo
- Alla pressione del pulsante, il LED lampeggia più velocemente

```
void loop()
{
    int state = digitalRead(BTN_PIN);
    if (state == HIGH)
    {
        ledDelay = 200;
    } else
    {
        ledDelay = 500;
    }
    digitalWrite(LED_PIN, HIGH);
    delay(ledDelay);
    digitalWrite(LED_PIN, LOW);
    delay(ledDelay);
}
```

Input analogico

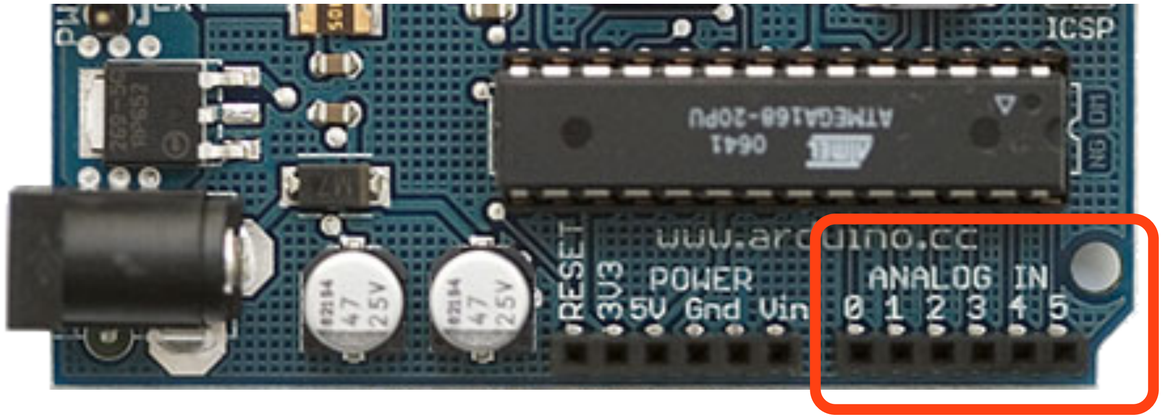
- Attraverso gli ingressi analogici è possibile leggere tensioni comprese tra 0 e 5V
- La funzione analogRead() riceve come parametro il numero del PIN da leggere e restituisce un intero compreso tra 0 e 1023 (ogni unità corrisponde dunque a 4.9 mV)

```
int temp;

temp = analogRead(0);
```



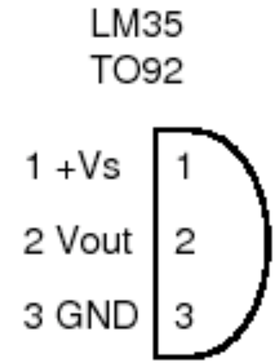
Input analogico

- Qualsiasi grandezza opportunamente trasformata in una tensione può essere letta da Arduino:
  - temperatura: sensore di temperatura
  - rotazione: potenziometro
  - intensità luminosa: fotoresistenza
  - distanza: sensore ad infrarossi
  - inclinazione: accelerometro



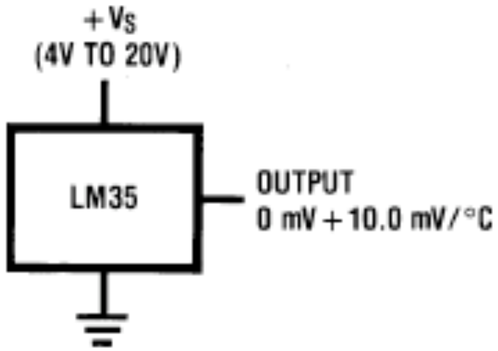
Sensore di temperatura

- I sensori analogici attivi (quelli ai quali è necessario fornire alimentazione) forniscono un segnale leggibile direttamente da Arduino
- Il sensore LM35DZ restituisce una tensione d'uscita proporzionale alla temperatura ambietale: 10 mV per ogni grado centigrado



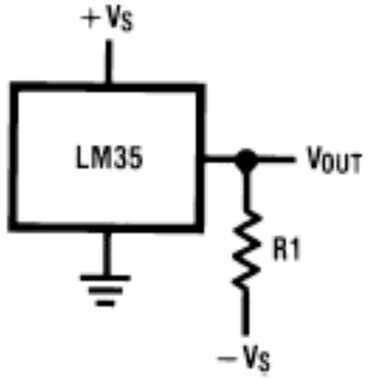
Piedinatura vista da sopra

Configurazioni LM35DZ



DS005516-3

FIGURE 1. Basic Centigrade Temperature Sensor (+2˚C to +150˚C)



DS005516-4

Choose $R_1 = -V_S/50\ \mu A$

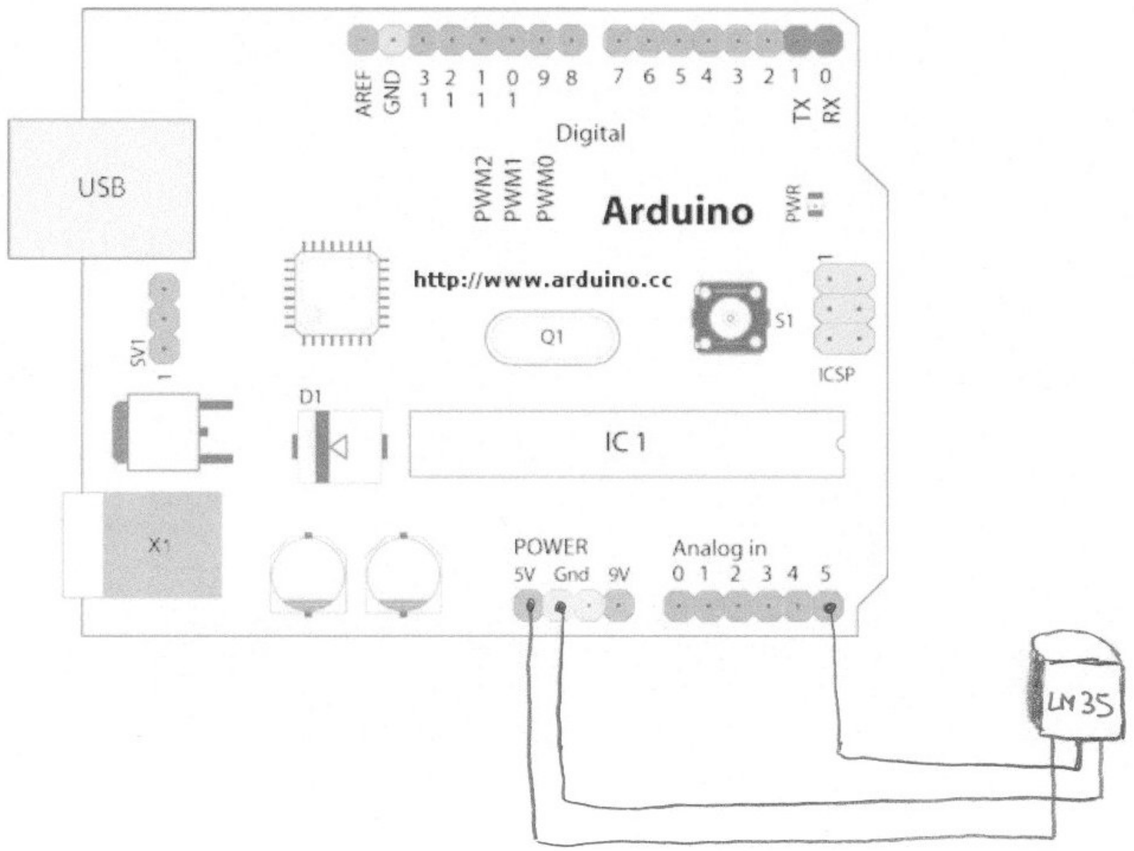
V_{OUT} = +1,500 mV at +150˚C
= +250 mV at +25˚C
= –550 mV at –55˚C

FIGURE 2. Full-Range Centigrade Temperature Sensor

http://www.national.com/ds/LM/LM35.pdf

Il circuito

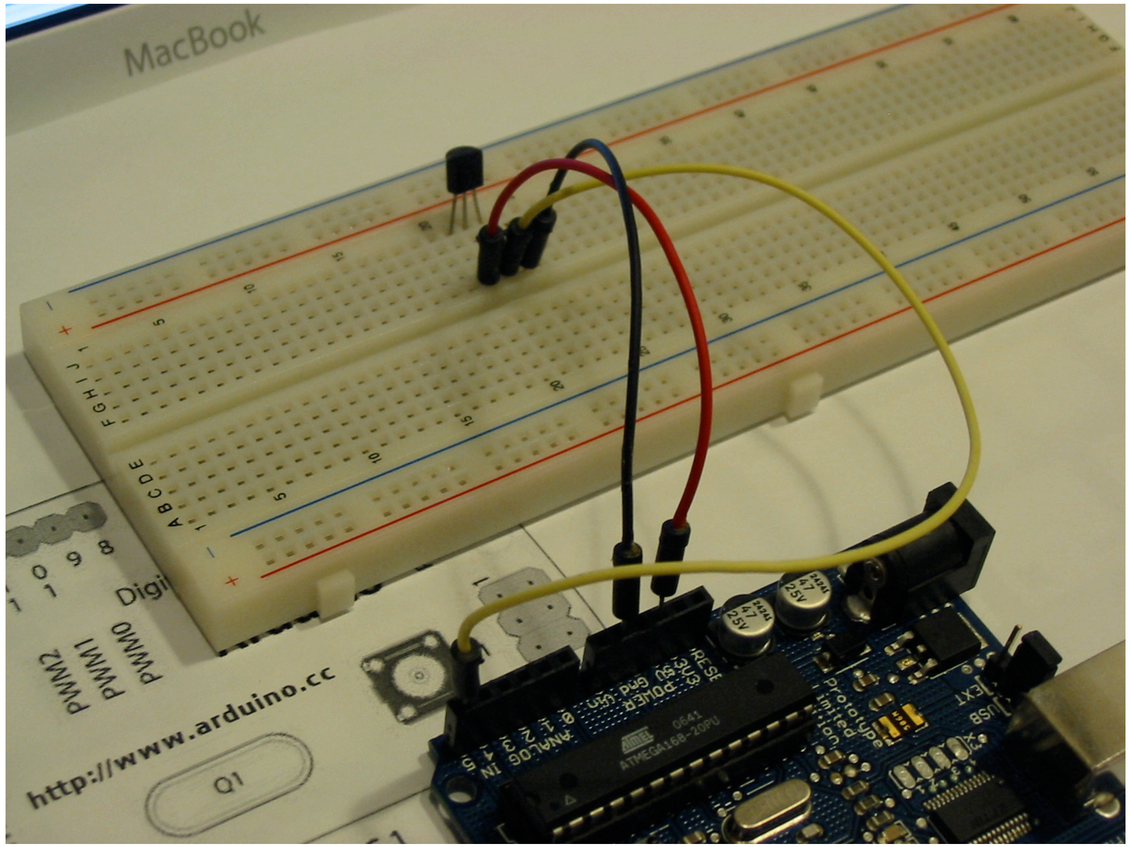
- Arduino può alimentare direttamente il sensore di temperatura e leggerne l'output attraverso un ingresso analogico



Il prototipo

```
int TEMP_PIN = 5;
int temp;

void setup()
{
}

void loop()
{
    temp = analogRead(TEMP_PIN) * 0.5;

    delay(100);
}
```

Ogni step di analogRead() corrisponde a 5mv (circa); un grado corrisponde ad un intervallo di 10mV, dunque:
temp = analogRead(TEMP_PIN) * 5 / 10;

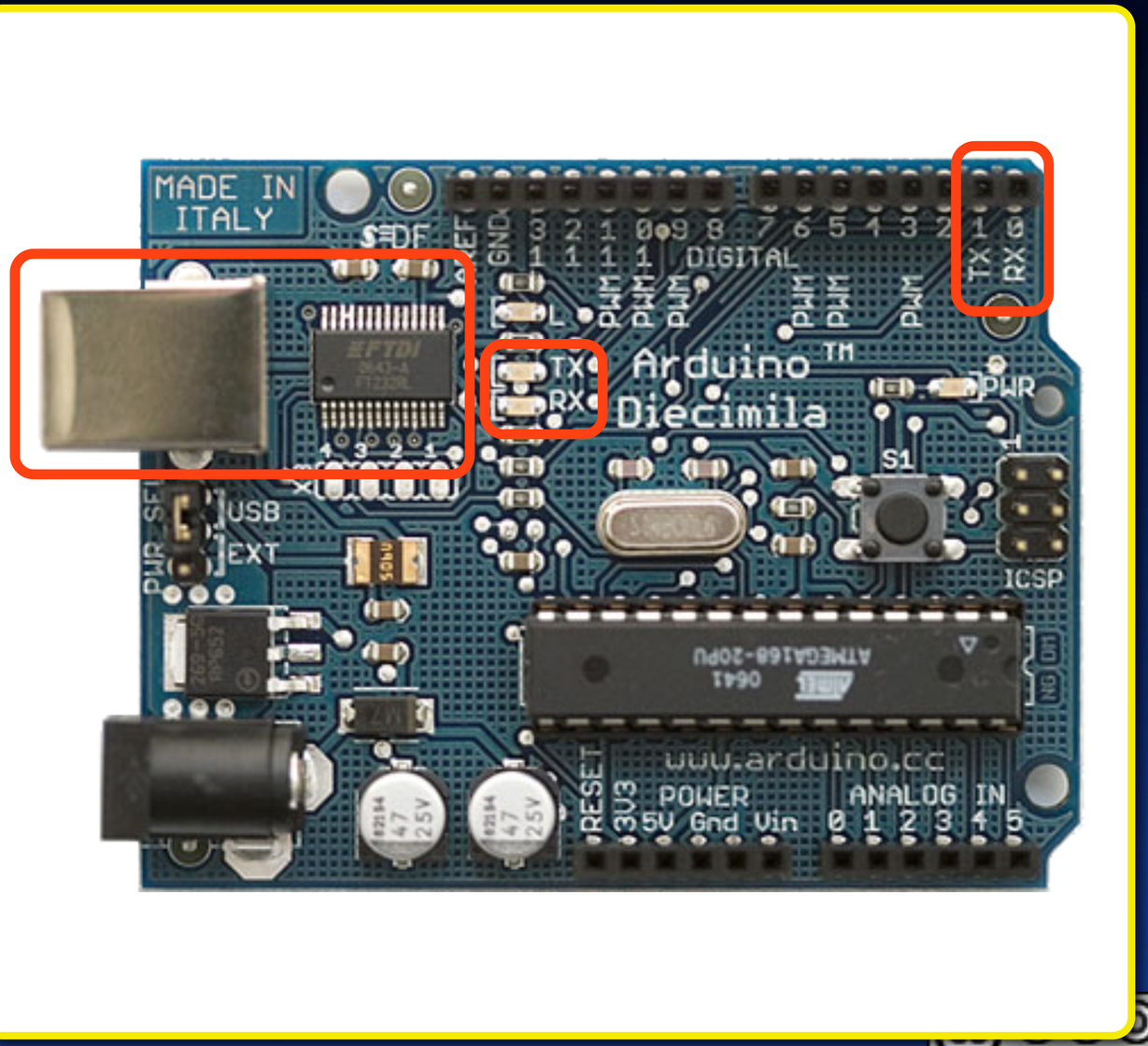


Cosa sta leggendo Arduino?

- La pressione di un pulsante (input digitale) può essere immediatamente verificata attraverso un output semplice: un LED acceso o spento
- Nel caso di un input analogico, il valore in ingresso appartiene ad un range ampio (1024 possibili valori) ed è dunque necessario predisporre un tipo di output diverso
- Sfruttiamo la porta seriale...

Serial port

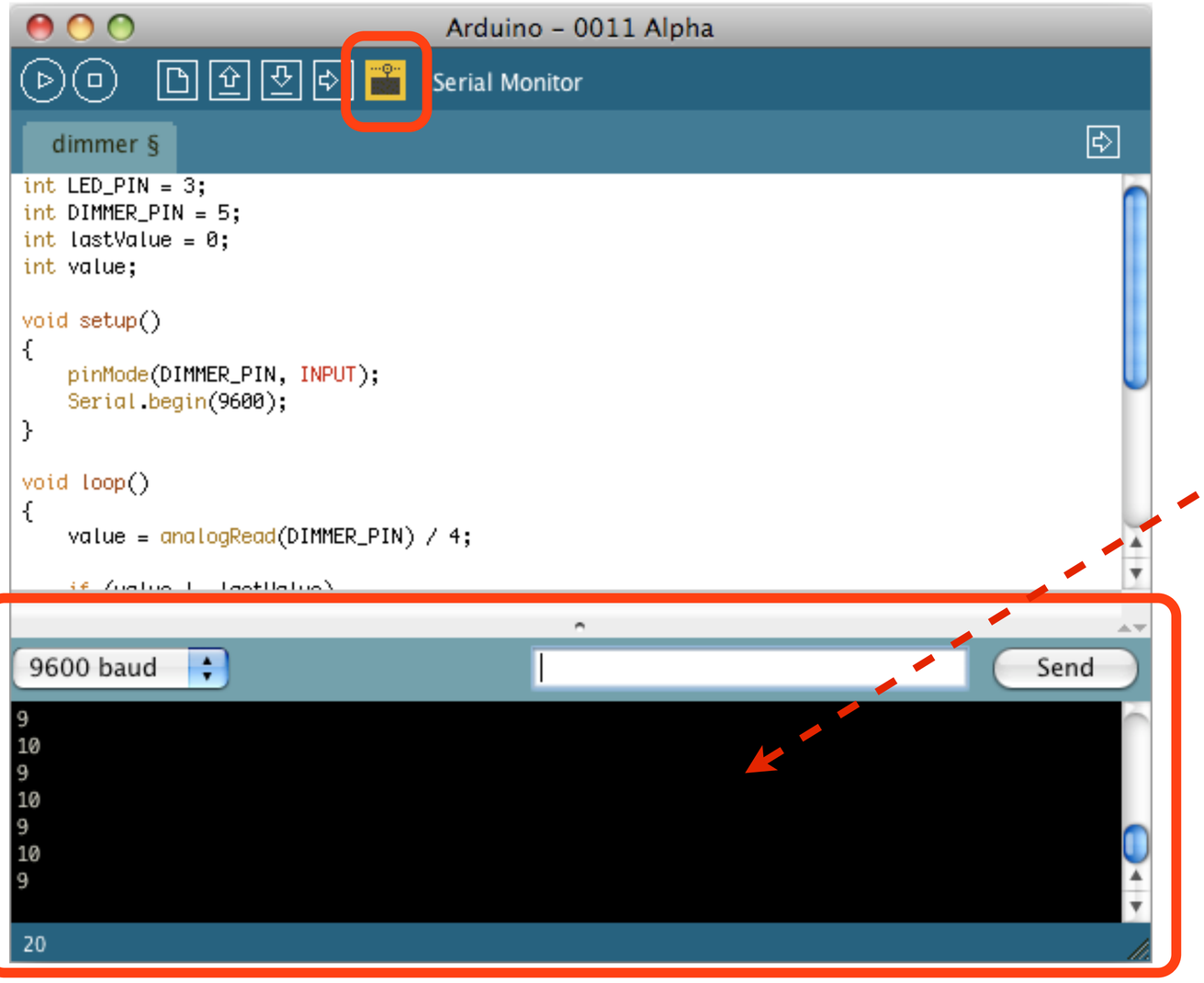
- Il microcontroller Atmel dispone di una porta seriale TTL, accessibile attraverso i PIN 1 e 2
- Il transito di dati in ingresso e uscita è segnalato da due LED TX ed RX sulla board
- La stessa seriale è accessibile attraverso l'interfaccia USB (o Bluetooth)

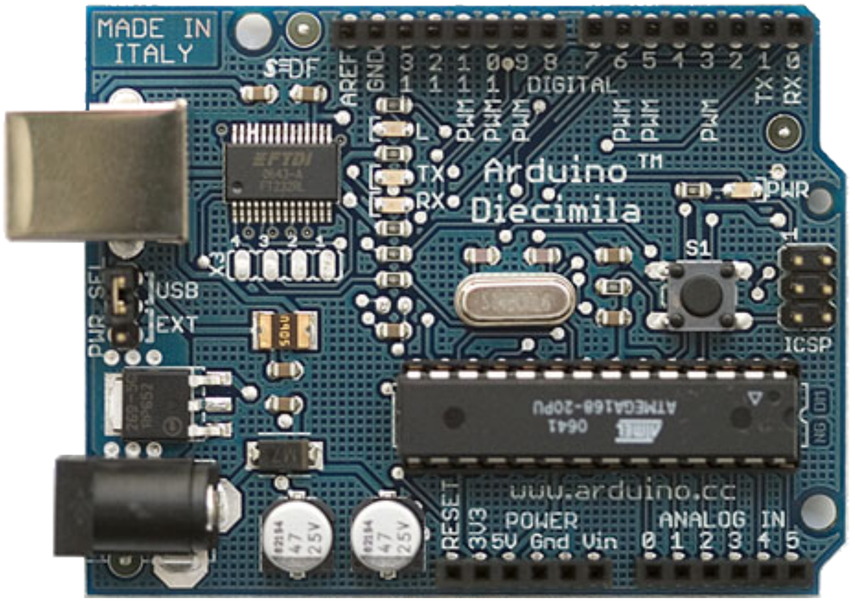


Serial port

- La connessione seriale (USB o Bluetooth) utilizzata per il caricamento degli sketch può essere utilizzara per la comunicazione con il PC **durante l'esecuzione** di un programma
- possibile dunque ricevere comandi (read) dal PC o inviare i dati provenienti dai sensori (write) al PC
- Attivando la **console seriale** nell'ambiente di sviluppo è possibile controllare in tempo reale l'esecuzione del programma inserendo opportune istruzioni in scrittura

Serial port


Arduino - 0011 Alpha
Serial Monitor
dimmer §
int LED_PIN = 3;
int DIMMER_PIN = 5;
int lastValue = 0;
int value;
void setup()
{
pinMode(DIMMER_PIN, INPUT);
Serial.begin(9600);
}
void loop()
{
value = analogRead(DIMMER_PIN) / 4;
9600 baud
Send
9
10
9
10
9
10
9
20


MADE IN ITALY
Arduino
Diecimila
www.arduino.cc
POWER
ANALOG IN
DIGITAL
USB
EXT
PWR
ICSP
RESET

Serial port

- La funzione Serial.begin() apre la porta seriale specificando la velocità di comunicazione
- La funzione Serial.print() scrive un valore che sarà ricevuto dal PC
- La funzione Serial.read() legge un valore proveniente dal PC

```
// imposta la seriale alla velocità
// SPEED
Serial.begin(9600);


// stampa il valore sulla seriale
Serial.print(VALORE);


// legge un valore dalla seriale
int value = 0;
value = Serial.read();
```

Serial port

- Il codice qui accanto invia un messaggio di avviso nel caso in cui la temperatura superi i 30 gradi

```
void setup()
{
    Serial.begin(9600);
}

void loop()
{
    // codice...

    if (temp > 30)
    {
        Serial.println(“WARNING!”);
    }

    // codice...
}
```

Un termometro per PC

- Utilizzando il sensore LM35DZ visto in precedenza è possibile realizzare un semplice ma efficace termometro per PC

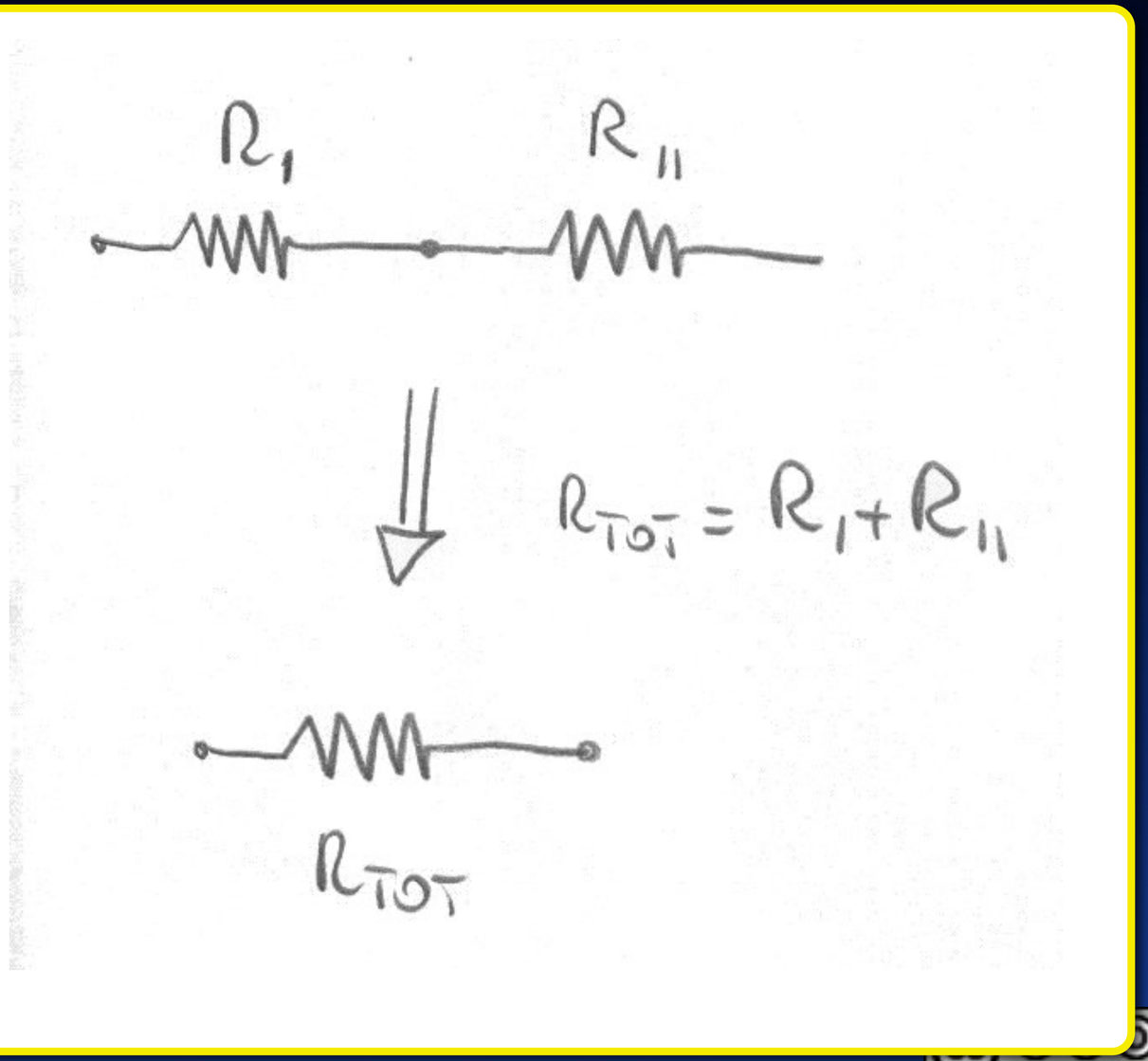
```
void setup()
{
    Serial.begin(9600);
}

void loop()
{
    temp = analogRead(TEMP_PIN) * 0.5;

    Serial.println(temp);

    delay(100);
}
```

Resistenze variabili

- I sensori più semplici sono i resistori variabili, componenti passivi che modificano la propria resistenza in funzione di una azione meccanica, termica o in funzione dell'esposizione alla luce
- Qui accanto una fotoresistenza, con il suo simbolo circuitale

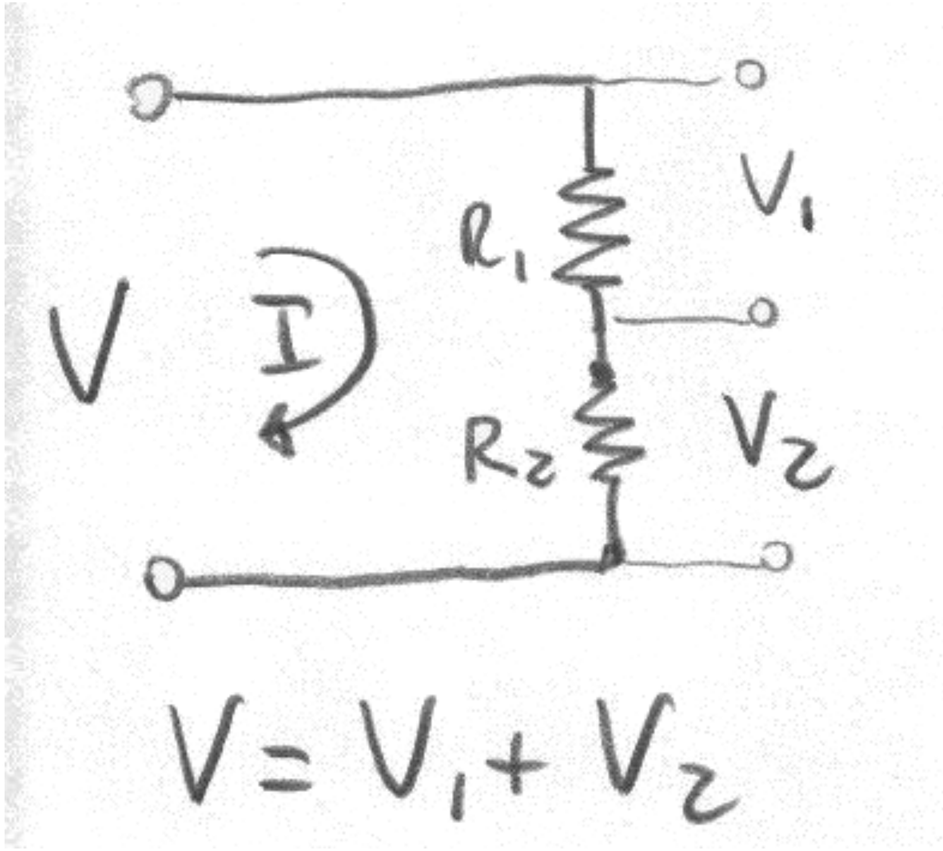
Resistori in serie

- Collegando due o più resistori **in serie**, la resistenza equivalente è pari alla **somma delle singole resistenze**
- La corrente risultante è inferiore a quella che si avrebbe in presenza di uno solo dei resistori



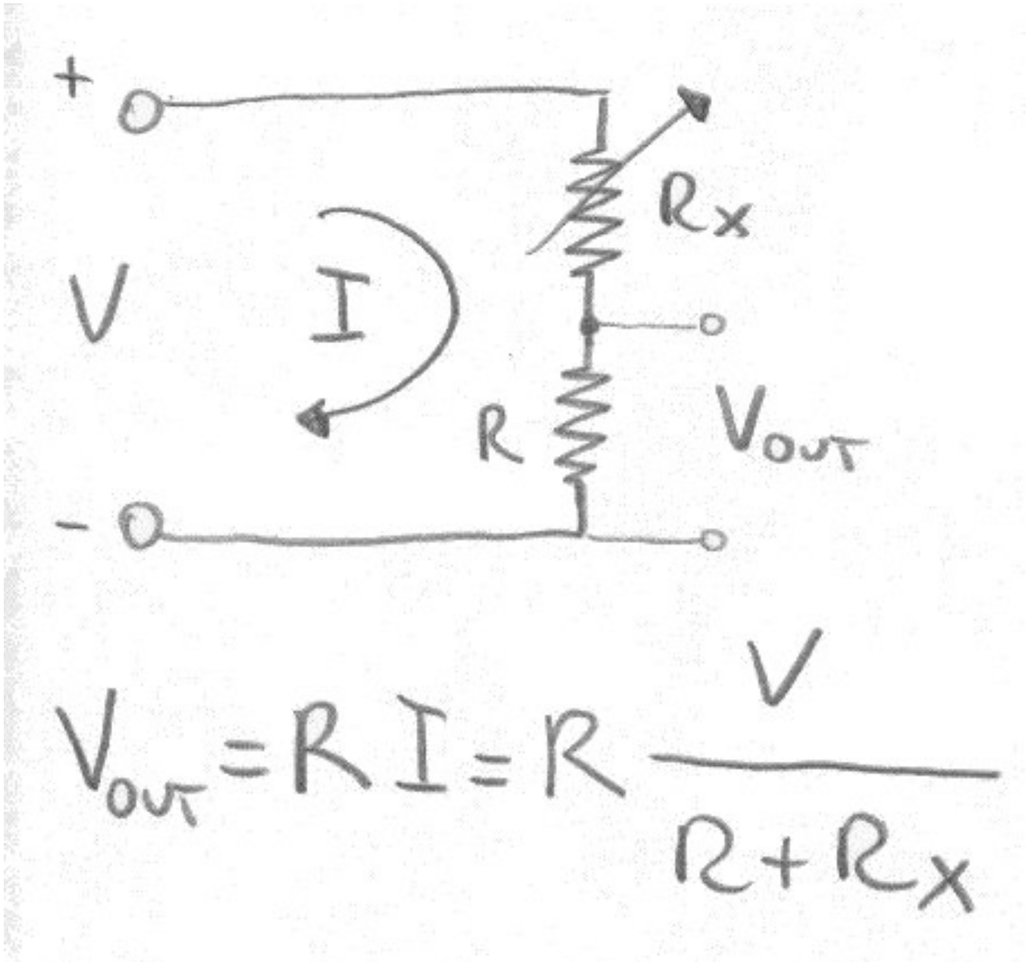
Partitore

- Due resistenze in serie costituiscono un partitore di tensione: la tensione di alimentazione ai capi del circuito è pari alla somma delle cadute di tensione ai capi di ciascuna resistenza
- La tensione ai capi di ciascun resistore è proporzionale alla resistenza dello stesso



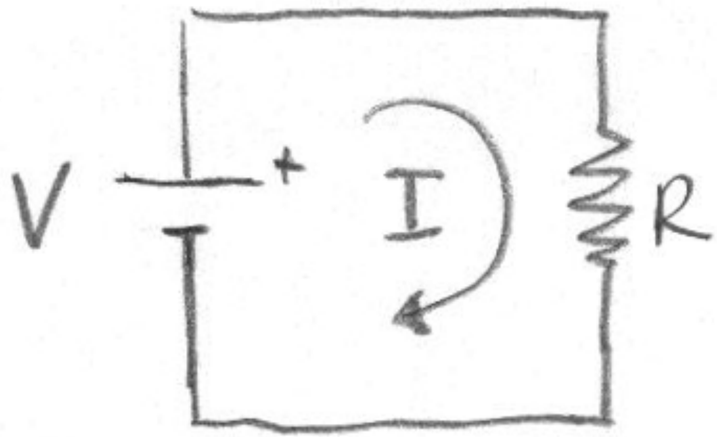
Partitore

- All'interno di un partitore, la una **resistenza variabile** provoca una **variazione di corrente** che, a sua volta, produce una **variazione di tensione** ai capi delle resistenze: tale variazione è ciò che si dovrà leggere come input di Arduino



Perché occorre un partitore?

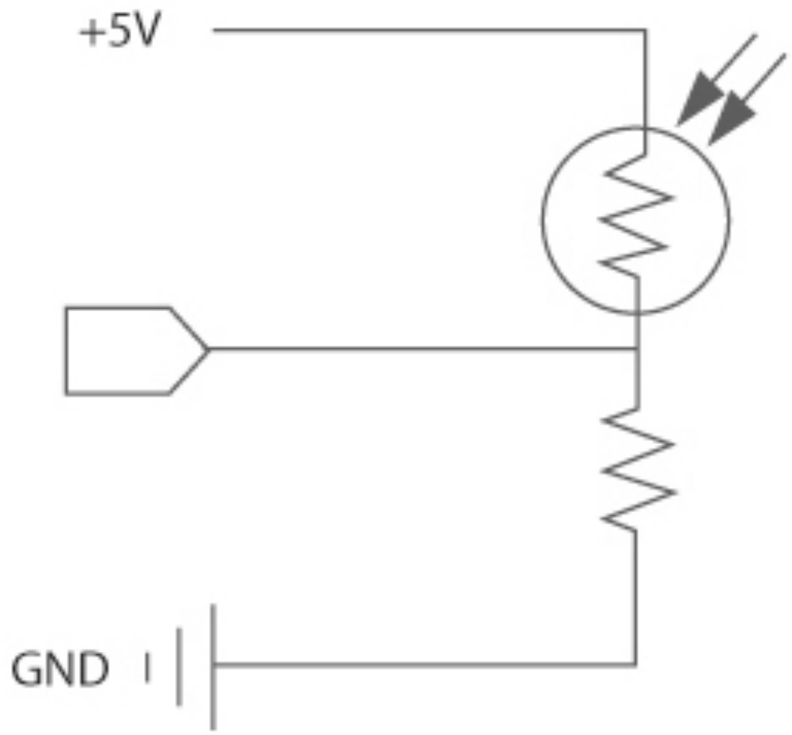
- L'uso di una sola resistenza variabile **non** permette di misurare una variazione di tensione: malgrado la corrente nel circuito vari in funzione della resistenza, la tensione resta pressoché costante, ovvero pari al valore fornito dal generatore



$$I = \frac{V}{R}$$

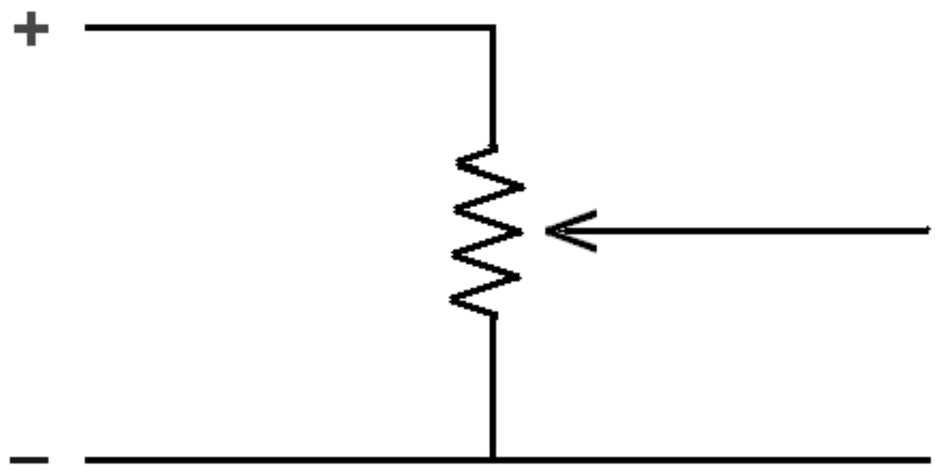
Fotoresistore

- Al variare dell'intensità luminosa varierà la resistenza del fotoresistore e, dunque, la tensione misurata ai capi della resistenza costante



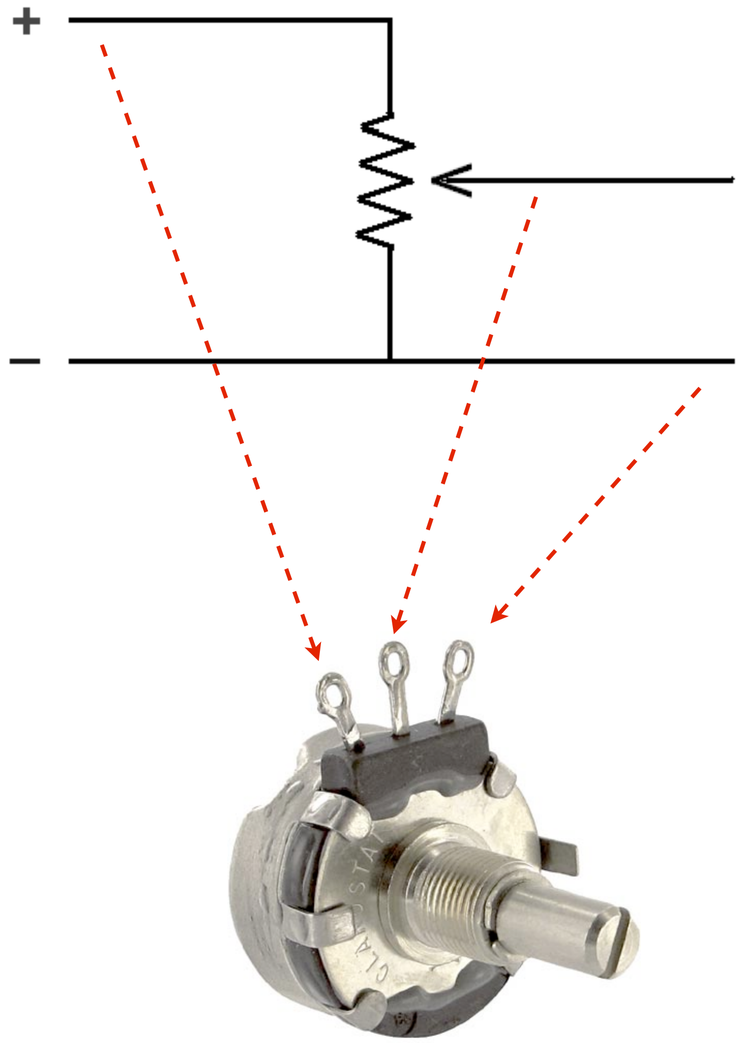
Potenziometro

- È un resistore la cui resistenza varia in base allo spostamento meccanico (rotazione, scorrimento, flessione)
- La lettura della tensione in uscita avviene tra il PIN centrale e la massa



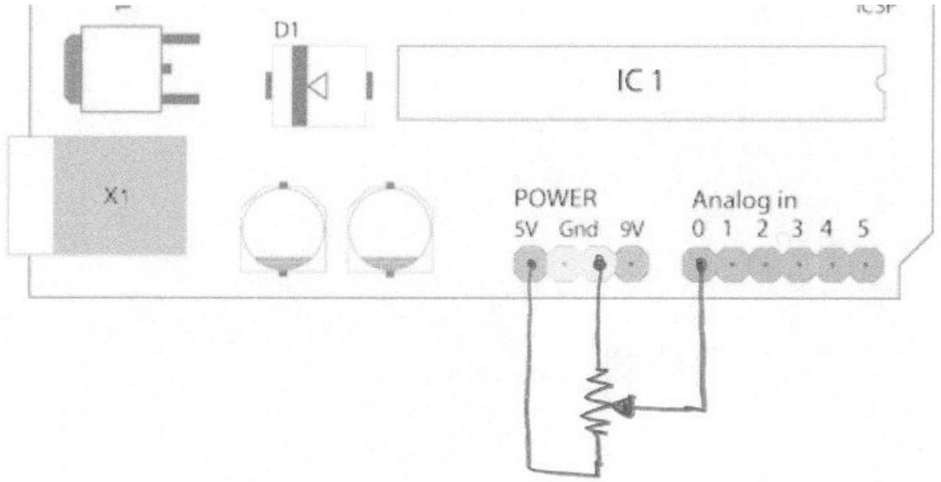
Potenziometro

- Ruotando in senso antiorario il cursore si sposta verso massa, riducendo il valore del secondo ramo resistivo e, dunque, della tensione
- Ruotando in senso orario il cursore si sposta verso la tensione di alimentazione, aumentando la resistenza del secondo ramo



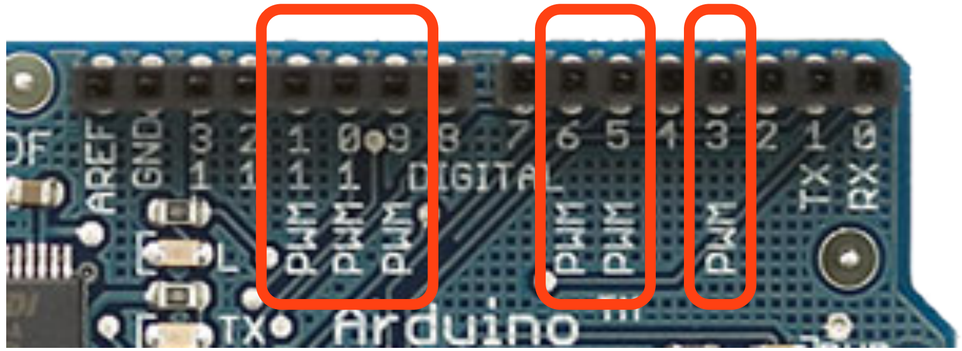
Potenziometro

- I contatti agli estremi saranno collegati alla tensione di servizio a 5V e alla massa
- Il contatto centrale sarà connesso ad un ingresso analogico
- La lettura del valore di tensione avviene attraverso la funzione analogRead()



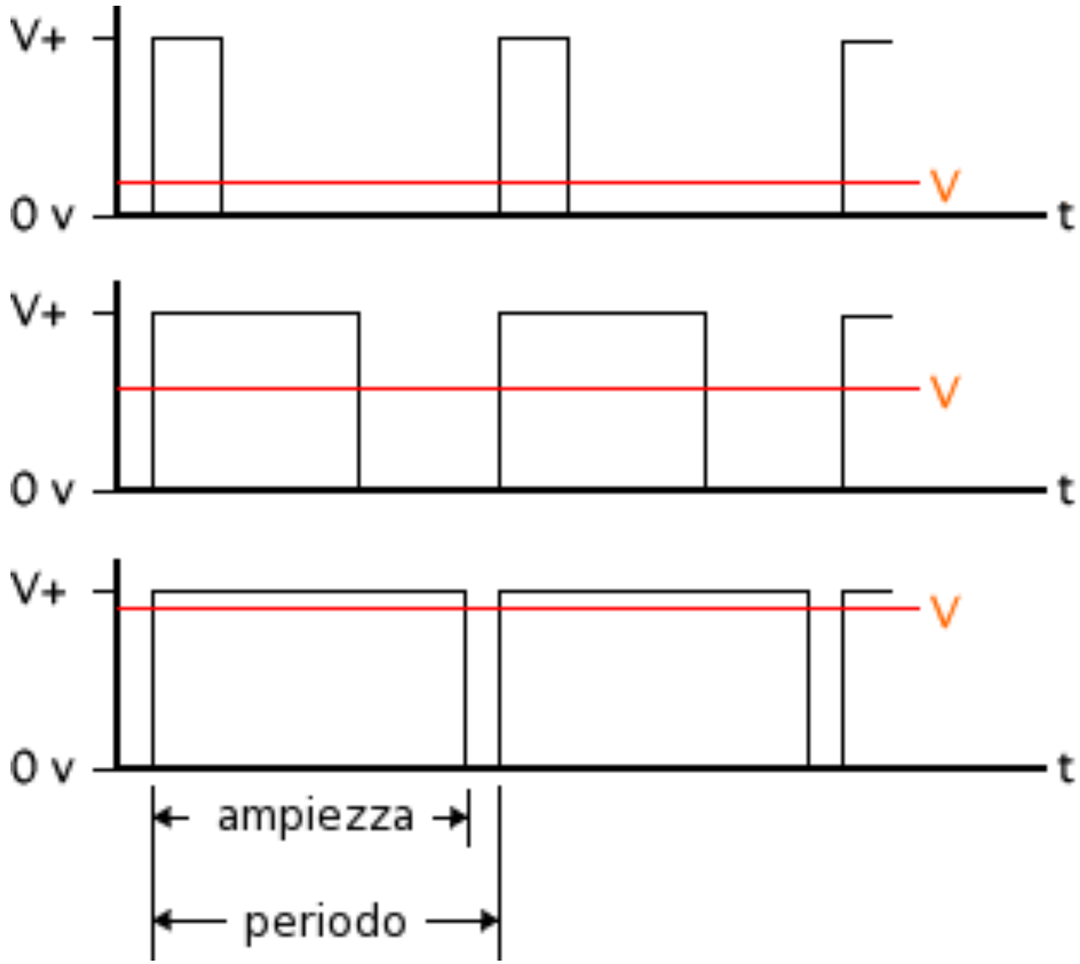
Output analogico (PWM)

- È possibile attivare un output analogico sotto forma di treno di impulsi di ampiezza variabile (Pulse Width Modulation, PWM)
- La funzione analogWrite() riceve come parametri il PIN di uscita e l'ampiezza temporale dell'impulso nel range 0-255
- L'output PWM è disponibile solamente sui PIN 3, 5 ,6 9, 10 e 11



Output analogico (PWM)

- La frequenza degli impulsi è circa 490 Hz
- Maggiore è l'ampiezza dell'impulso e maggiore sarà la tensione media sul periodo (5V continui al limite)
- Minore è l'ampiezza dell'impulso e minore sarà la tensione media sul periodo (0V costanti al limite)



Fade in & out [1]

- Utilizzando un PIN PWM è possibile variare l'intensità (media sul periodo) dell'uscita analogica per controllare la luminosità di una luce, l'intensità di un suono o la velocità di un motore
- Due cicli for in sequenza permettono di realizzare un effetto *fade in & out*

```
int LED_PIN = 3;
int w = 0;

void setup()
{
  /*
  non e' necessario impostare
  il pinMode in OUTPUT, pero'...
  */
}
```

Fade in & out [2]

```
void loop()
{
    for(w = 0 ; w <= 255; w+=5)
    {
        analogWrite(LED_PIN, w);
        delay(50);
    }

    for(w = 255; w >=0; w-=5)
    {
        analogWrite(LED_PIN, w);
        delay(50);
    }
}
```

Aumenta l'ampiezza dell'impulso finché non raggiunge il massimo. Ad ogni ciclo attende 50 ms prima dell'incremento successivo

Decrementa l'ampiezza dell'impulso finché non è nulla. Ad ogni ciclo attende 50 ms prima dell'iterazione successiva

Dimmer [1]

- Il dimmer è un circuito che consente di regolare la l'intensità di una sorgente luminosa
- Utilizzando un potenziometro (input) e un LED (output) è possibile realizzare un semplice dimmer... per la casa delle bambole!

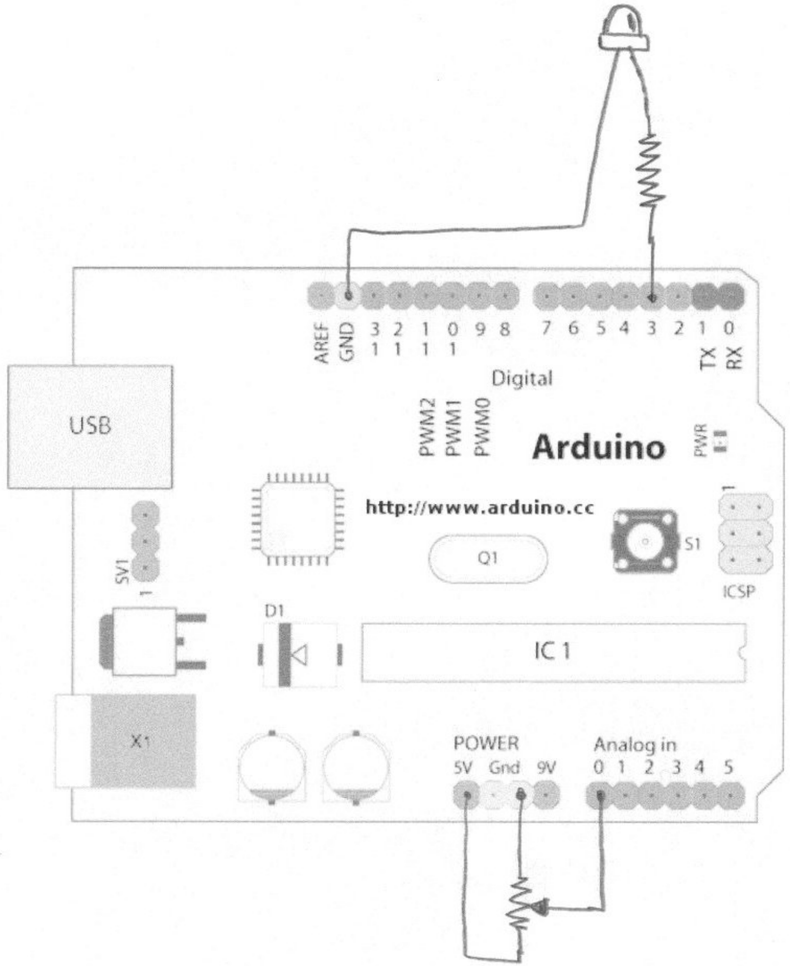
```
int LED_PIN = 3;
int DIMMER_PIN =0;

// memorizza il valore precedente
int lastValue = 0;

// memorizza l'ultimo valore letto
int value;

void setup()
{
}
```

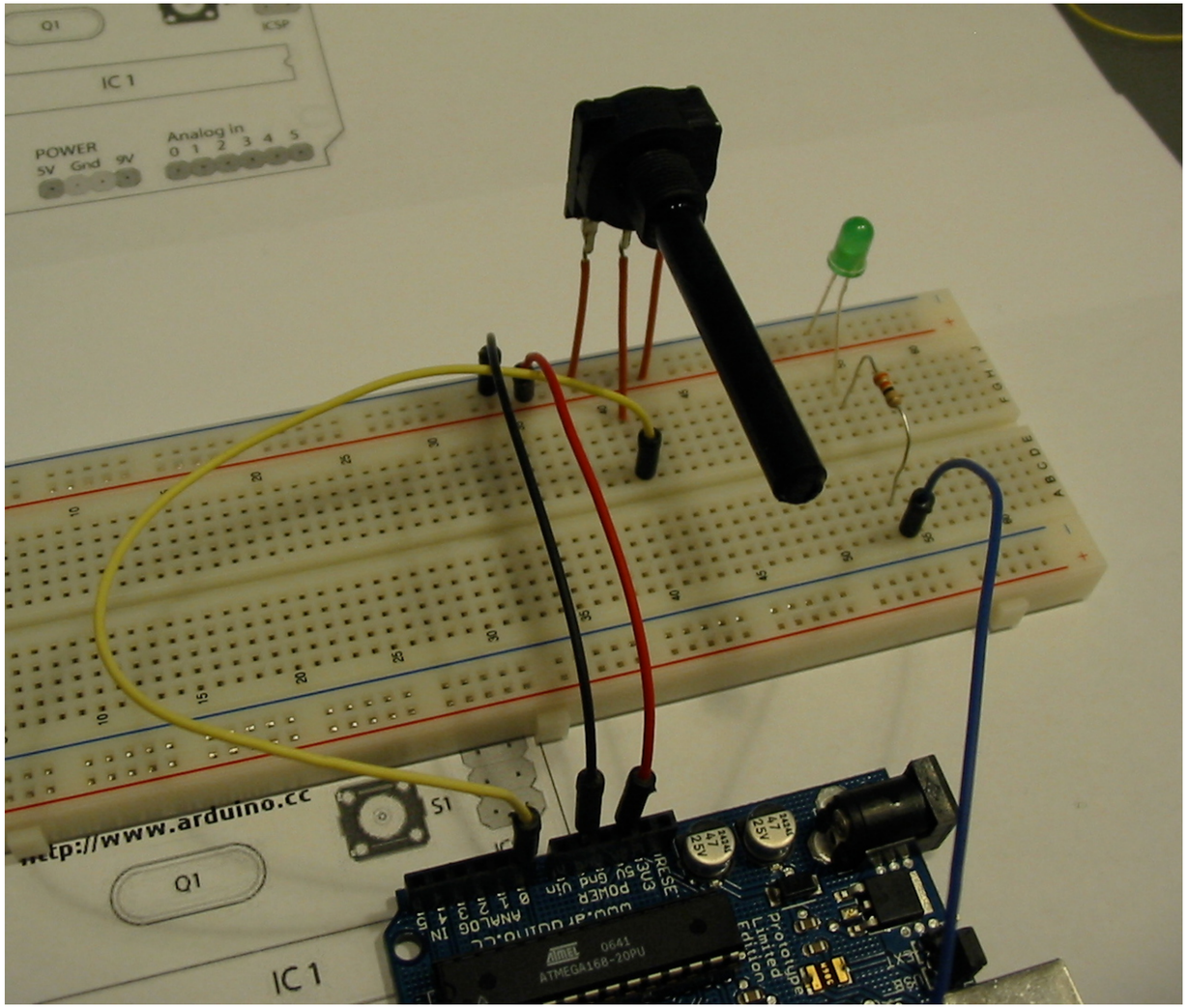
Dimmer [2]

- Il dimmer è un circuito che consente di regolare la luminosità di una luce
- Utilizzando un potenziometro (input) e un LED (output) è possibile realizzare un semplice dimmer... per la casa delle bambole!

```
void loop()
{
  value = analogRead(DIMMER_PIN) / 4;

  if (value != lastValue)
  {
    lastValue = value;

    analogWrite(LED_PIN, value);
    }
}
```

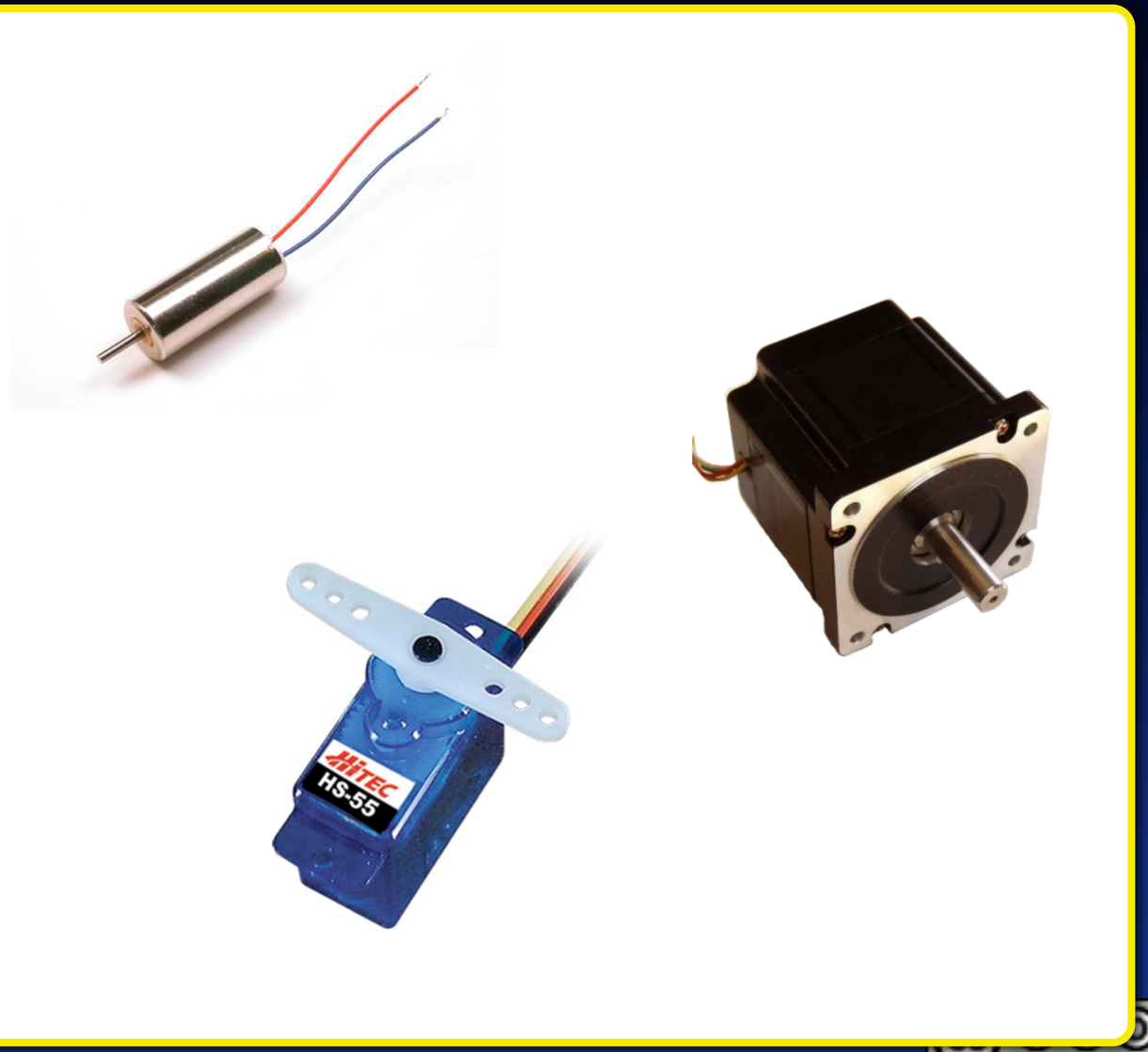
Dimmer [3]


AREF
GND
3 2 1 0 9 8
7 6 5 4 3 2 1 0
TX
RX
Digital
PWM2
PWM1
PWM0
Arduino
PWR
USB
http://www.arduino.cc
Q1
S1
ICSP
SV1
D1
IC 1
X1
POWER
5V Gnd 9V
Analog in
0 1 2 3 4 5


Q1
ICSP
IC 1
POWER
5V Gnd 9V
Analog in
0 1 2 3 4 5
http://www.arduino.cc
S1
Q1
IC 1

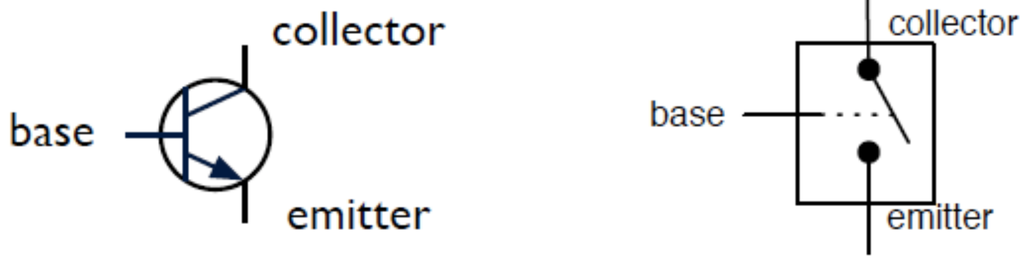
Motori

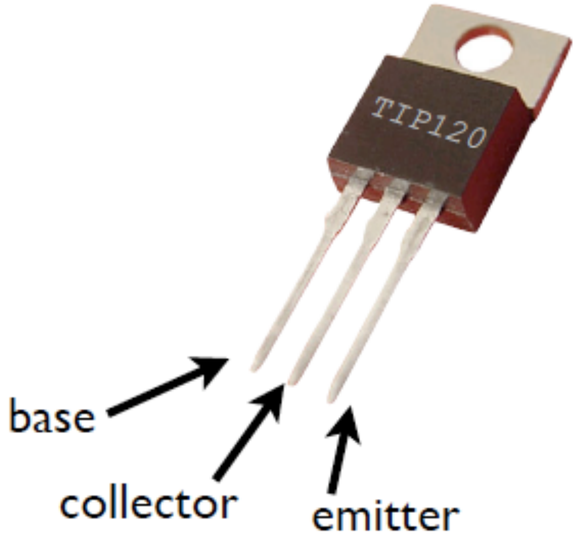
- Arduino consente il controllo dei tre tipi principali di motori elettrici:
  - motori diretti
  - servomotori
  - motori passo-passo
- Arduino **non alimenta** direttamente i motori ma ne controlla il movimento attraverso appositi circuiti



Transistor

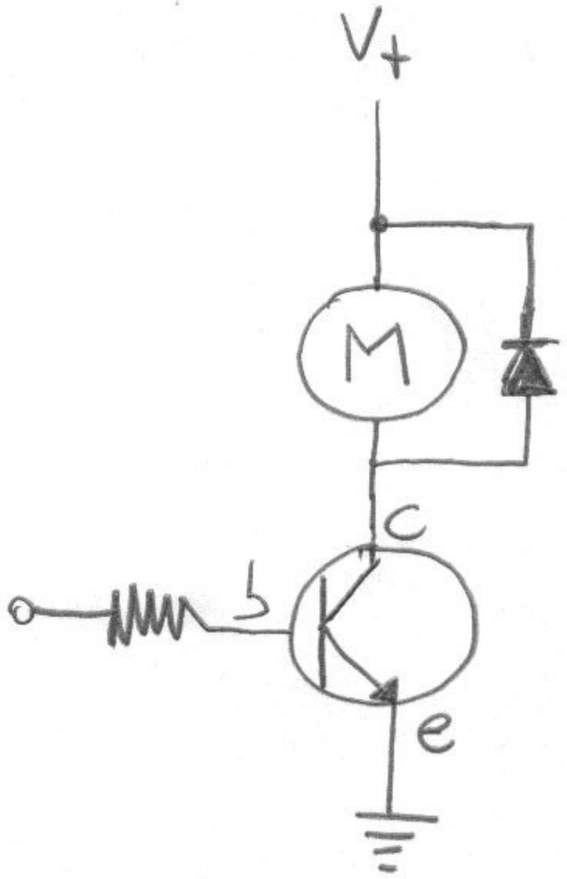
- Il transistor è un componente a semiconduttore utilizzato come interruttore controllato, amplificatore
- Polarizzando la **base**, **collettore** ed **emettitore** conducono corrente, comportandosi come un circuito chiuso





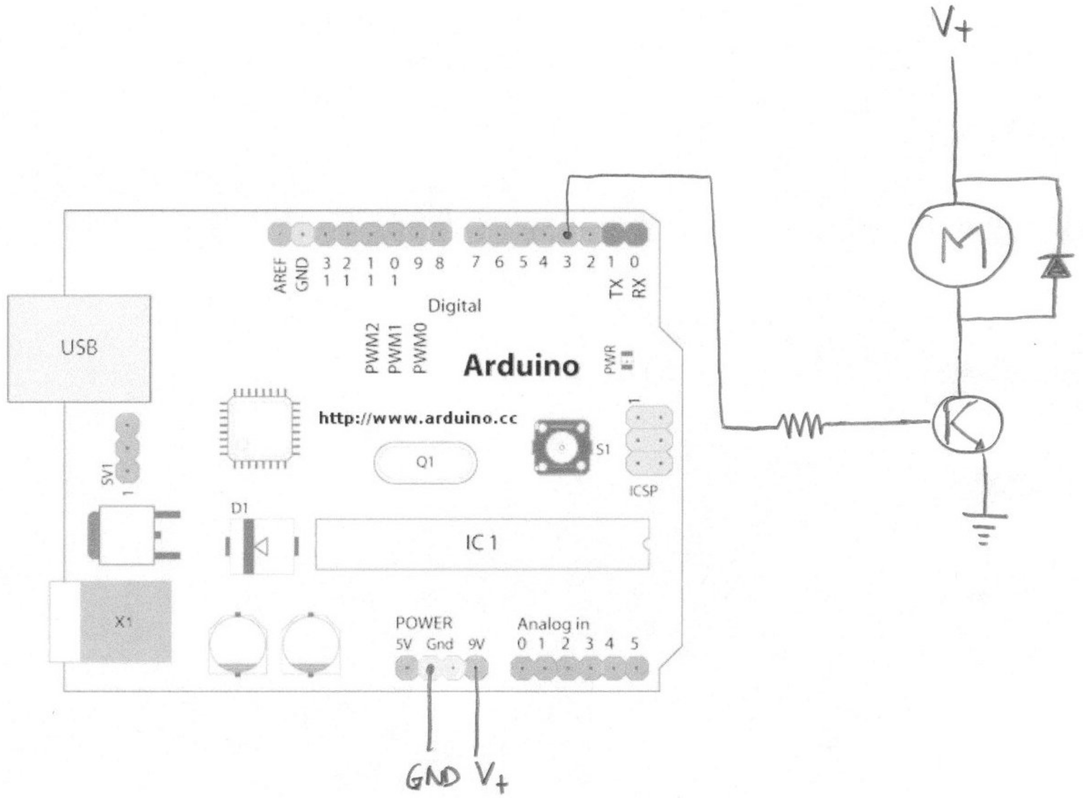
Controllare un motore

- L'azionamento di un motore con Arduino richiede l'uso di un transistor, che agisce nel circuito come un interruttore controllato
- Portando a livello alto la base, il transistor conduce e alimenta il motore
- Il diodo in parallelo elimina la tensione indotta dal motore

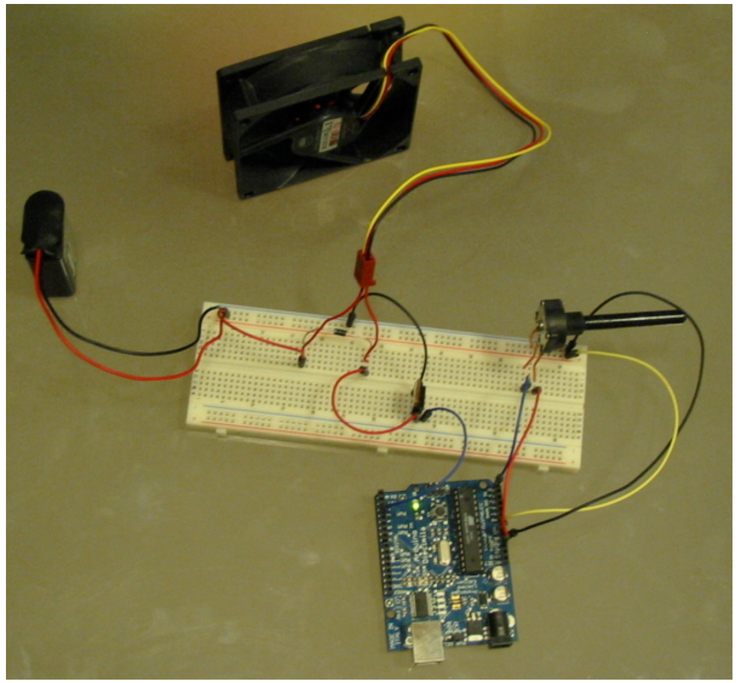


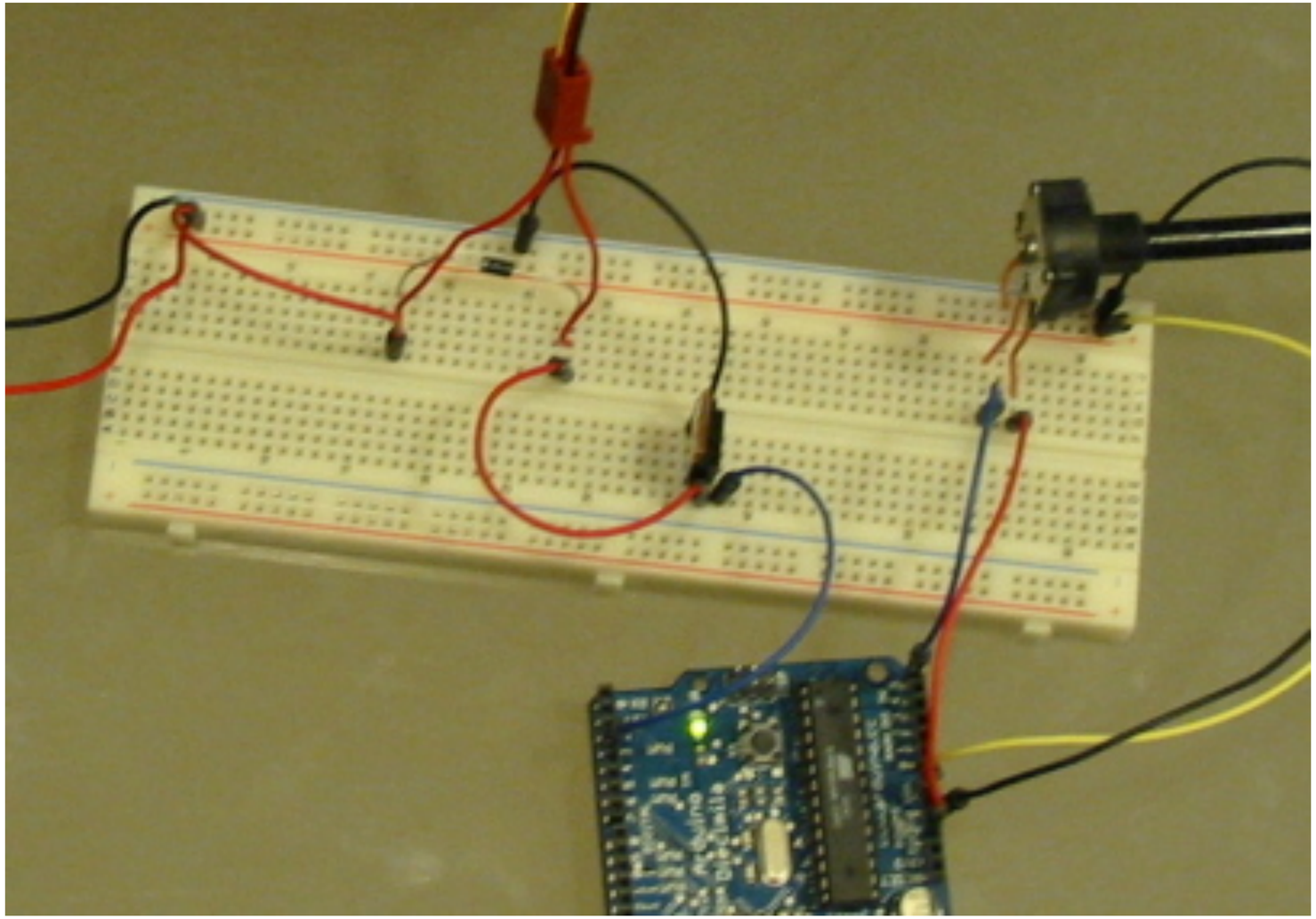
Controllo PWM

- Utilizzando una uscita analogica è possibile variare la velocità di rotazione del motore modificando l'ampiezza dell'impulso



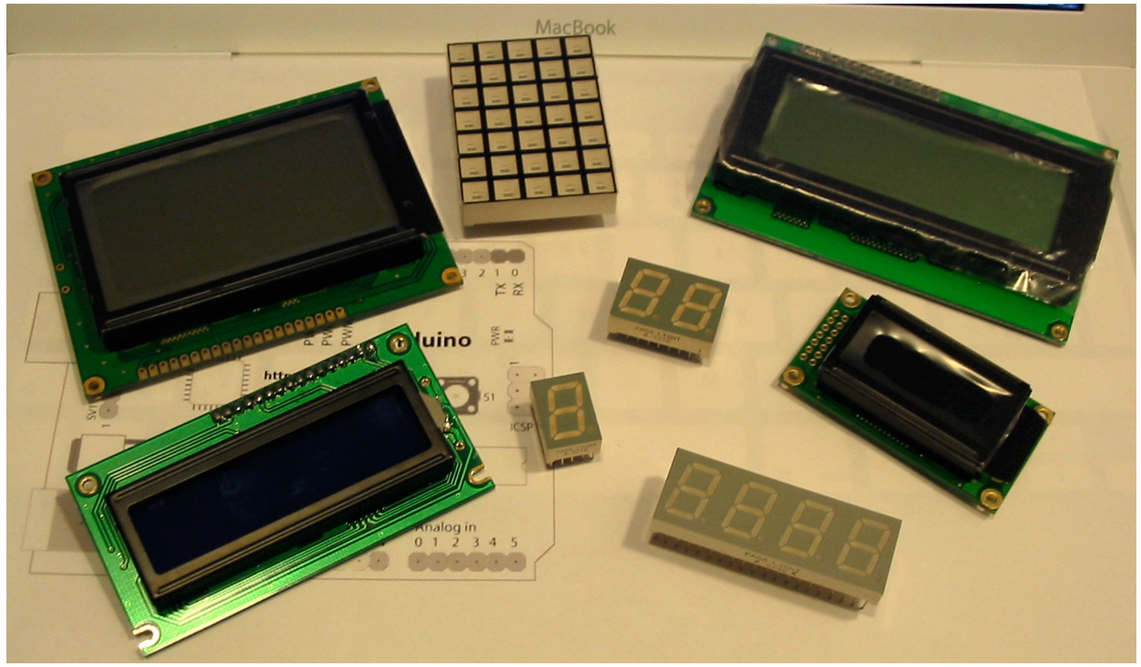
Un motore controllato





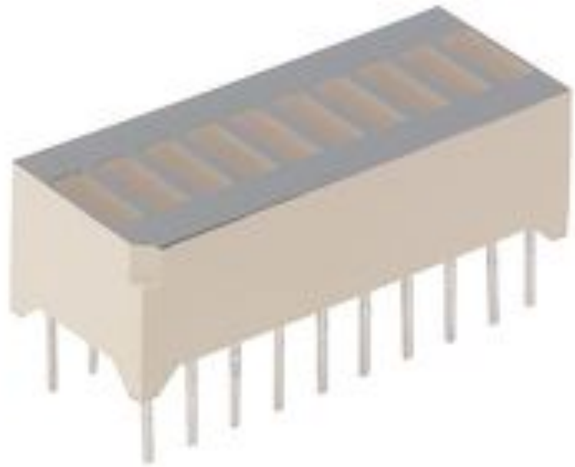
Display

- I display consentono ad Arduino di visualizzare informazioni complesse: intensità, numeri, testi e grafica:
  - BAR graph
  - cifre LED a 7 segmenti
  - LCD alfanumerici
  - LCD grafici



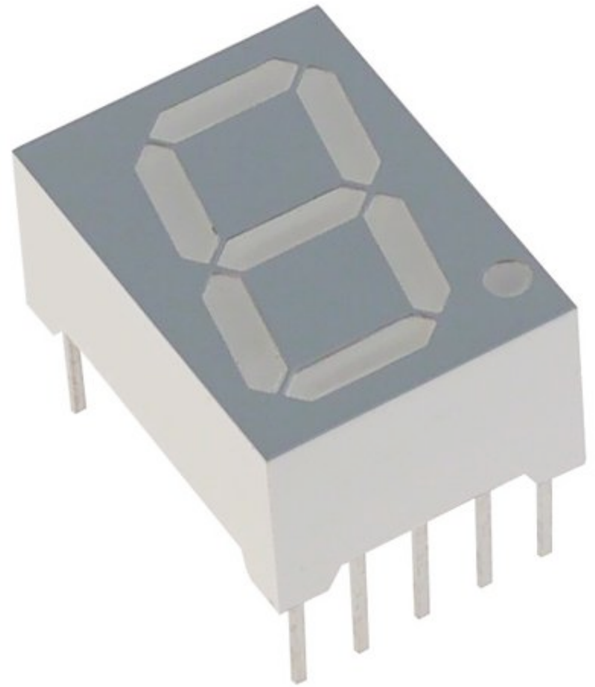
BAR Graph

- È il display più semplice: è costituito da una sequenza ordinata di LED che possono essere accesi in successione per indicare un livello
- Solitamente ogni LED è controllato individualmente e valgono le indicazioni già viste sulle limitazioni in tensione e corrente



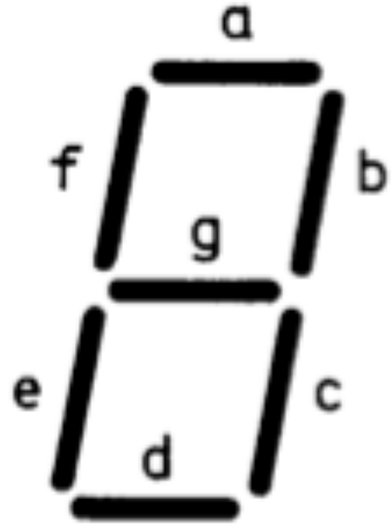
Display LED 7 segmenti

- Sono costituiti da un insieme di LED piatti disposti per formare le **cifre decimali**
- Sono disponibili in due configurazioni:
  - Anodo comune: un unico PIN è connesso a V+, il controllo avviene sul catodo (LOW) di ciascun LED
  - Catodo comune: un unico PIN è connesso a GND, il controllo avviene sull'anodo (HIGH) di ciascun LED



Display LED 7 segmenti

- Questo tipo di display richiede 7 PIN digitali, uno per ciascun LED
- È possibile ridurre gli output a 4 utilizzando un display driver e la codifica BCD



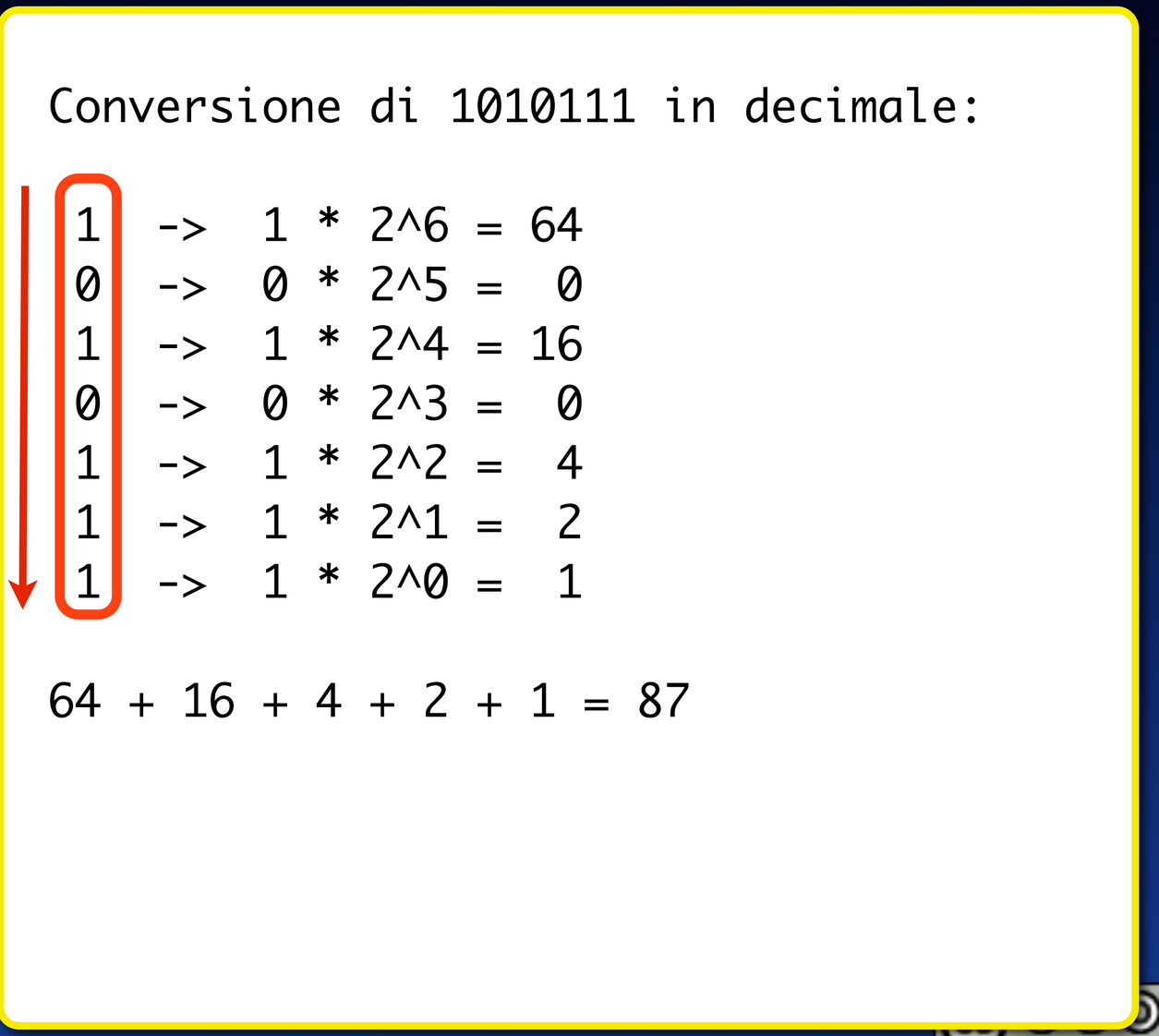
Da decimale a binario

- La conversione di un numero da base decimale a base binaria avviene attraverso una sequenza di divisioni:
  - si divide il numero per 2 ed il resto rappresenta il bit meno significativo del numero decimale
  - il quoziente è diviso nuovamente per 2, e il resto sarà la cifra decimale successiva
  - si continua con finché il quoziente è nullo

```
Conversione di 87 in binario:

87 / 2 = 43  ->   87 % 2 = 1
43 / 2 = 21  ->   43 % 2 = 1
21 / 2 = 10  ->   21 % 2 = 1
10 / 2 =  5  ->   10 % 2 = 0
 5 / 2 =  2  ->    5 % 2 = 1
 2 / 2 =  1  ->    2 % 2 = 0
 1 / 2 =  0  ->    1 % 2 = 1

87 in binario diventa: 1010111
```

Da binario a decimale

- La conversione di un numero da base binaria a base decimale si effettua attraverso la sommatoria del prodotto di ciascuna cifra per 2 elevato il peso (la "posizione", da 0 a N-1) della cifra stessa



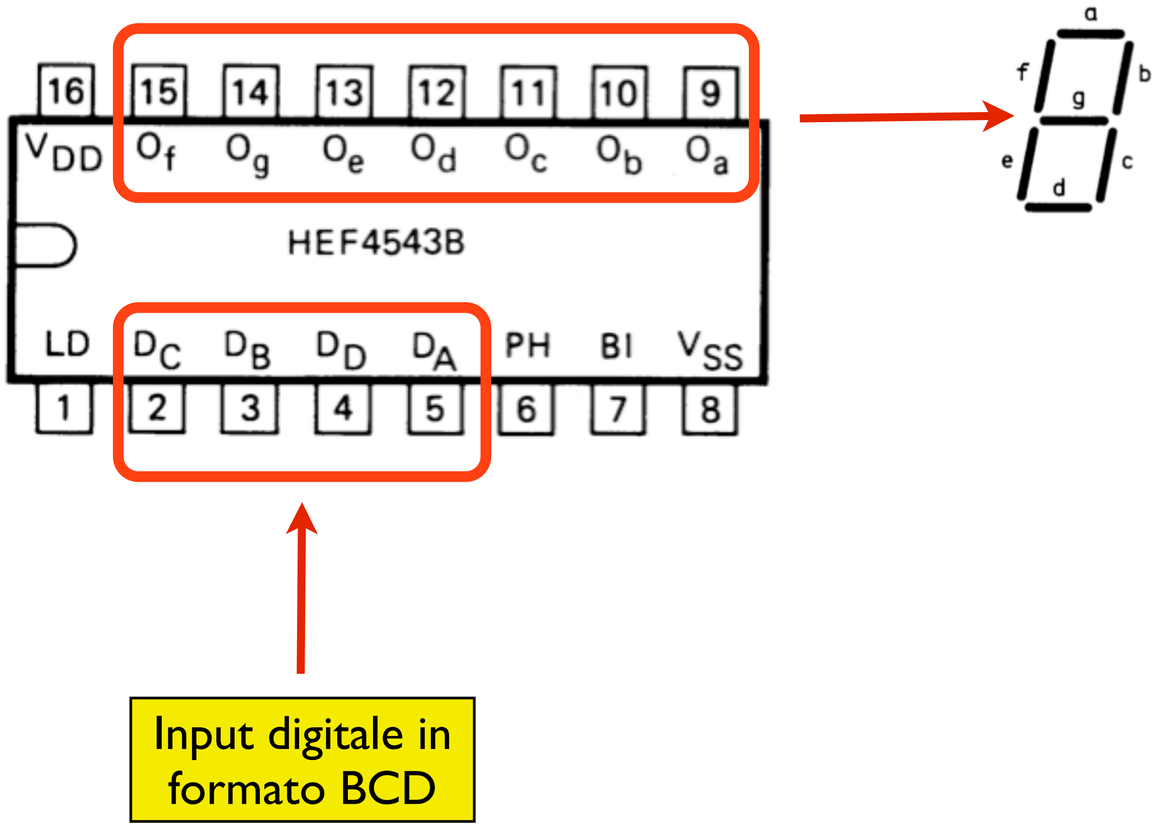
Binary-Coded Decimal

- Il Binary-Coded Decimal è un formato di codifica di numeri decimali in binario, la cui trasformazioni avviene in maniera indipedente per ciascuna cifra
- Il numero risultante deve essere letto a gruppi di 4 bit per volta

```
Codifica binaria:

12     ->    1100
123    ->    01111011


Codifica BCD:

12     ->    1 2        ->   0001 0010
123    ->    1 2 3      ->   0001 0010 0011
```

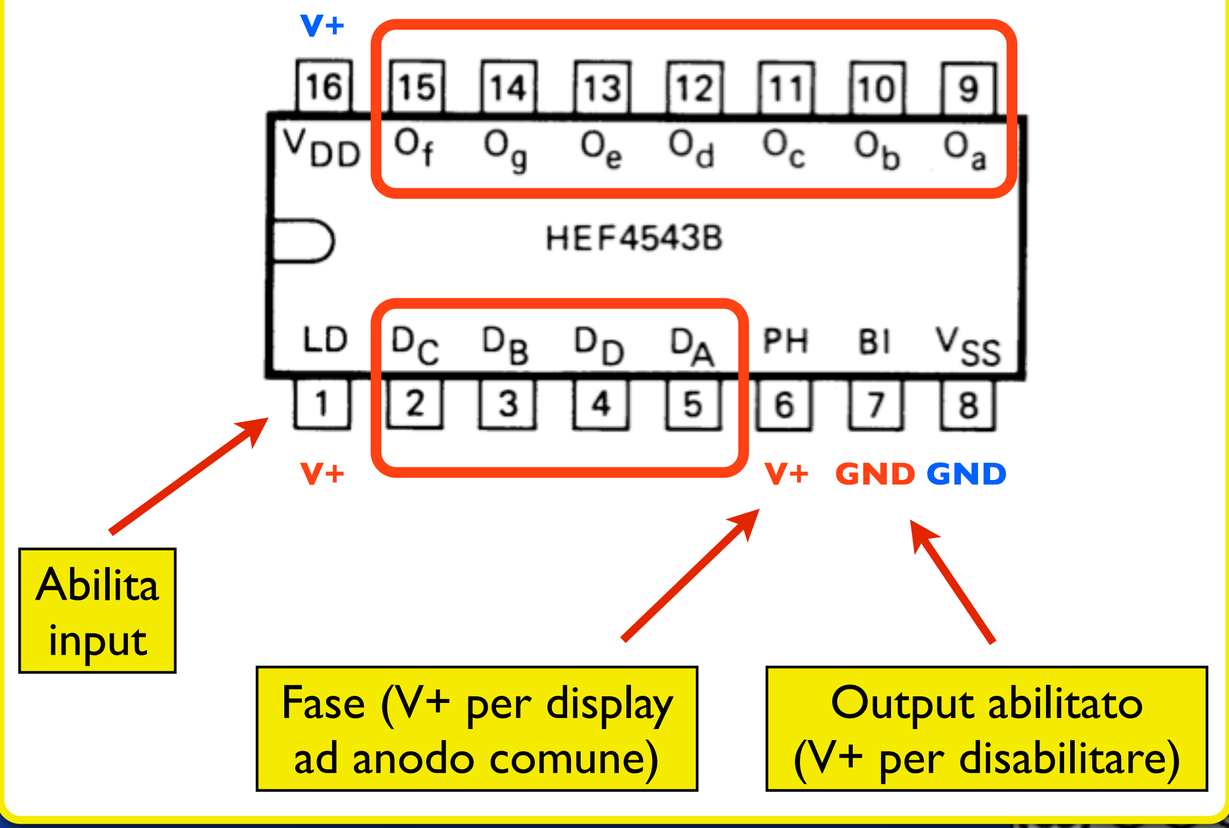
Display Driver HEF 4543

- Il driver HEF4543 è alimentato a 3V+
- I PIN DA, DB, DC, DD ricevolo l'input della cifra decimale in formato BCD
- I PIN Oa - Og controllano direttamente i segmenti del display a LED



Display Driver HEF 4543

- Configurazione:
  - Alimentazione: 2V-6V
  - **LD**: se impostato LOW, disabilita l'input e "congela" l'output sull'ultimo input
  - **PH**: impostato LOW consente di pilotare display a **catodo comune**; impostato a livello alto consente di pilotare display ad **anodo comune**
  - **BI**: impostato HIGH disabilita l'output (il display si spegne)



Un semplice contatore [1]

```
int D[] = {5, 4, 3, 2};

int L[] = { 0, 0, 0, 0,
            0, 0, 0, 1,
            0, 0, 1, 0,
            0, 0, 1, 1,
            0, 1, 0, 0,
            0, 1, 0, 1,
            0, 1, 1, 0,
            0, 1, 1, 1,
            1, 0, 0, 0,
            1, 0, 0, 1 };

int j;
int i;
```

PIN Output

0000 = 0
0001 = 1
0010 = 2
0011 = 3
0100 = 4
0101 = 5
0110 = 6
0111 = 7
1000 = 8
1001 = 9

Un semplice contatore [2]

```
void setup()
{
  for (i = 0; i < 4; i++)
  {
    pinMode(D[i], OUTPUT);
  }
}
```

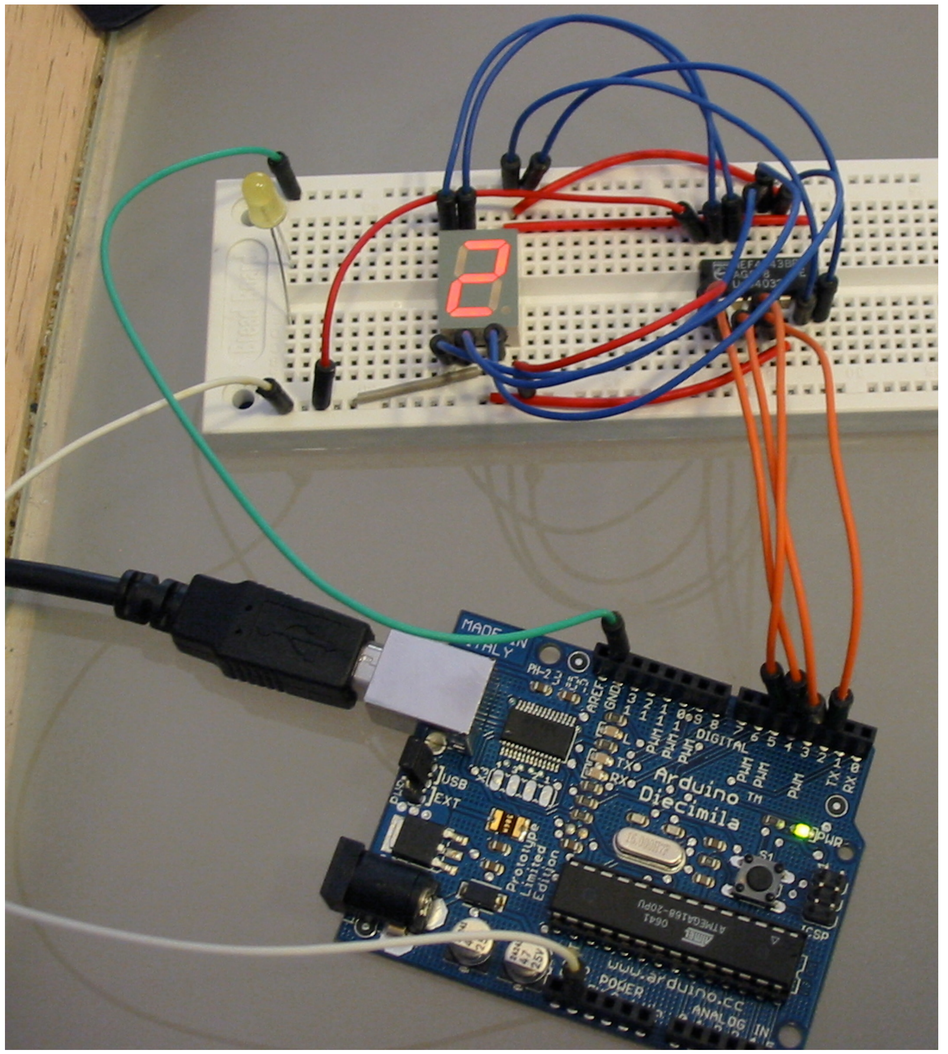
Un semplice contatore [3]

- Il ciclo esterno scorre le cifre decimali (le righe del vettore L)
- Il ciclo interno scrive l'output digitale della codifica BCD di ciascuna cifra decimale

```
void loop()
{
  for (i = 0; i <  10; i++)
  {
    for (j = 0; j < 4; j++)
    {
      digitalWrite(5-j, L[j+4*i]);
    }

    delay(1000);
  }
}
```

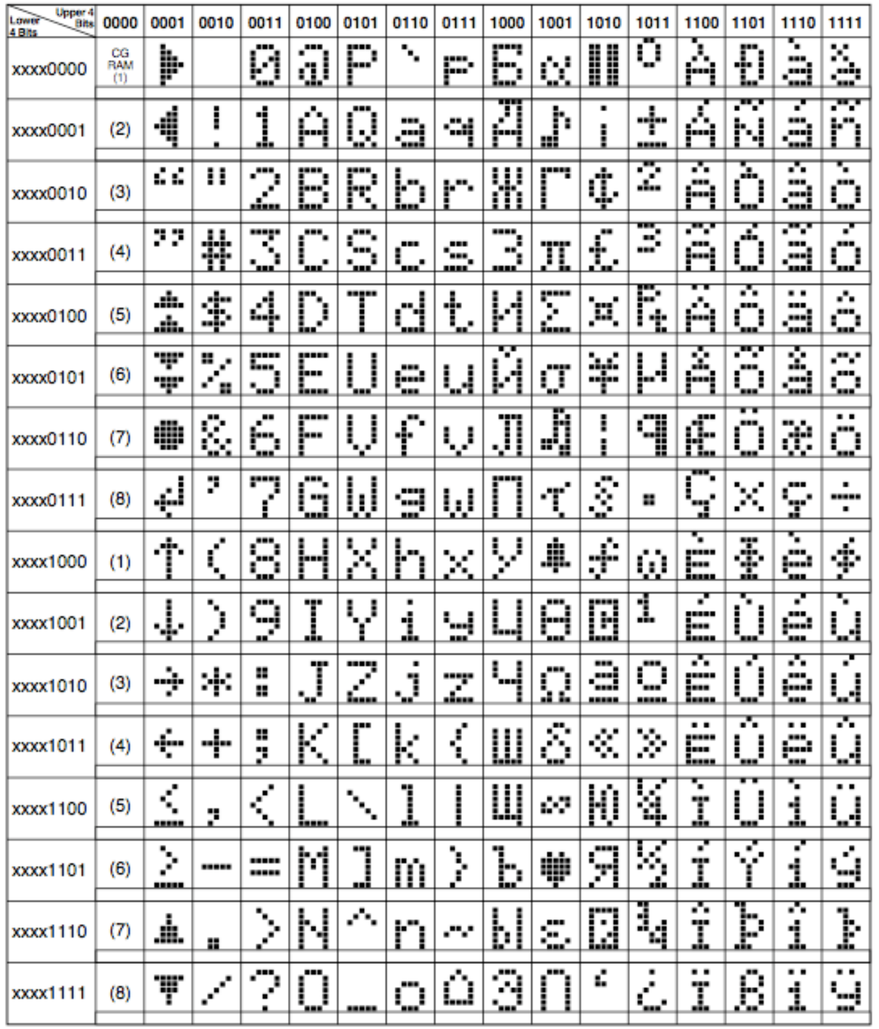
Prototipo

- La breadboard è piuttosto affollata attorno al display driver, ma il collegamento ad Arduino è ordinato e soprattutto molti PIN sono liberi per rivere input o controllare altri dispositivi



Display LCD

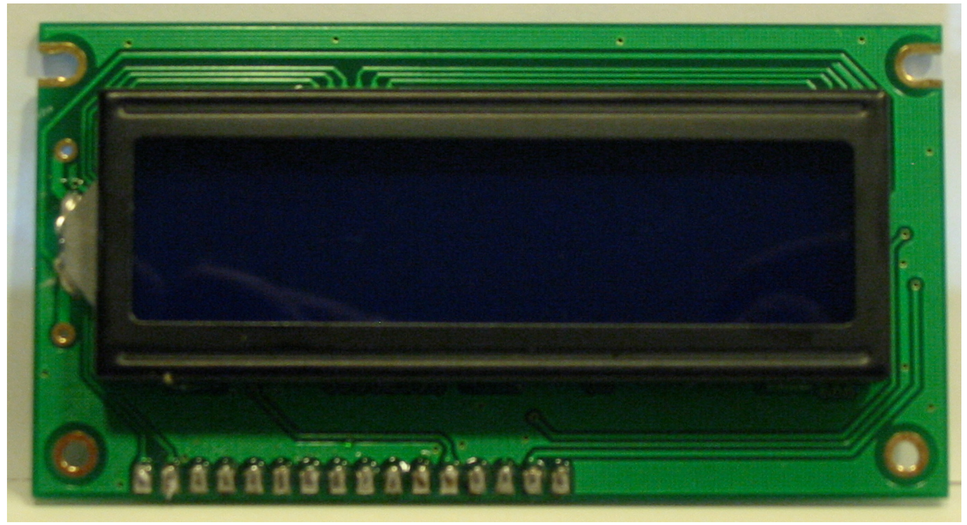
- I display LCD consentono di visualizzare testo su una o più linee, oppure grafica
- Un apposito microprocessore provvede ad intepretare i comandi provenienti dagli input digitali e a pilotare il display vero e proprio

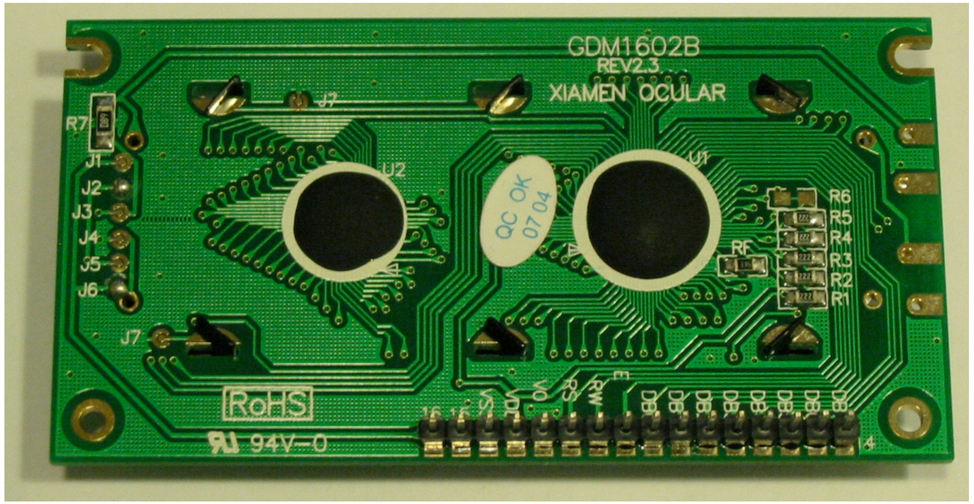


Upper 4 Bits / Lower 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			0	@	P	`	p								
xxxx0001	(2)		!	1	A	Q	a	q								
xxxx0010	(3)		"	2	B	R	b	r								
xxxx0011	(4)		#	3	C	S	c	s								
xxxx0100	(5)		$	4	D	T	d	t								
xxxx0101	(6)		%	5	E	U	e	u								
xxxx0110	(7)		&	6	F	V	f	v								
xxxx0111	(8)		'	7	G	W	g	w								
xxxx1000	(1)		(	8	H	X	h	x								
xxxx1001	(2)		)	9	I	Y	i	y								
xxxx1010	(3)		*	:	J	Z	j	z								
xxxx1011	(4)		+	;	K	[	k	{								
xxxx1100	(5)		,	<	L	\	l	|								
xxxx1101	(6)		-	=	M	]	m	}								
xxxx1110	(7)		.	>	N	^	n	~								
xxxx1111	(8)		/	?	O	_	o									

Display LCD

- I display più diffusi (e più economici!) sono basati sul chip Hitachi HD44780, in grado di controllare modelli di diverse dimensioni (1-4 righe, 8-40 colonne)
- Grazie alla loro diffusione, sono largamente supportati

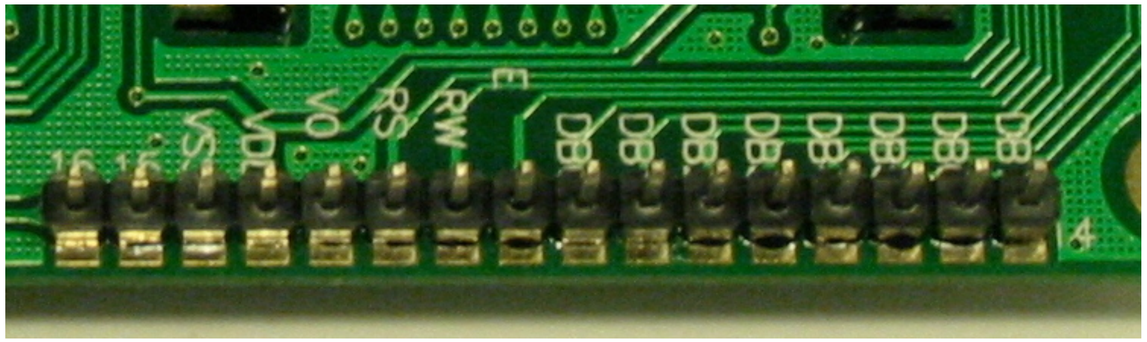




Display LCD: cablaggio

- La disposizione dei PIN pu cambiare a seconda del modello, ma la serigrafia sul circuito stampato consente di effettuare rapidamente il cablaggio
- Occorono:
  - alimentazione 5V+
  - 11 PIN digitali
  - potenziometro (contrasto)

DISPLAY	ARDUINO
DB0 - DB7	2 - 9
E	10
RW	11
RS	12
V0	potenziometro
VDD	5V+
VSS	GND
15-16	LED illuminazione



LCD Library

- Il modo più semplice per controllare i display LCD è utilizzare la libreria **LCD Library**, che contiene le funzioni per inizializzare il display e inviare stringhe di testo
- La versione disponibile su **http://www.arduino.cc/en/Tutorial/LCDLibrary** non è compatibile con la versione corrente di Arduino e non supporta display con 2 linee di testo
- All'indirizzo **http://www.gerdavax.it/data/LCDLibrary_2x40.zip** è disponibile una versione modificata dell libreria

Installazione LCD Library

- Per poter utilizzare la LCD Library è necessario installarla all'interno dell'ambiente di sviluppo di Arduino
- Occorre decomprimere il file **LCDLibrary_2x40.zip**, che contiene la cartella LiquidCrystal. Occorre copiare questa cartella all'interno della directory **hardware/libraries** contenuta nella installazione di Arduino
- Tale procedura è compiuta una sola volta e non è necessario ripeterla per ogni programma

Display LCD: codice

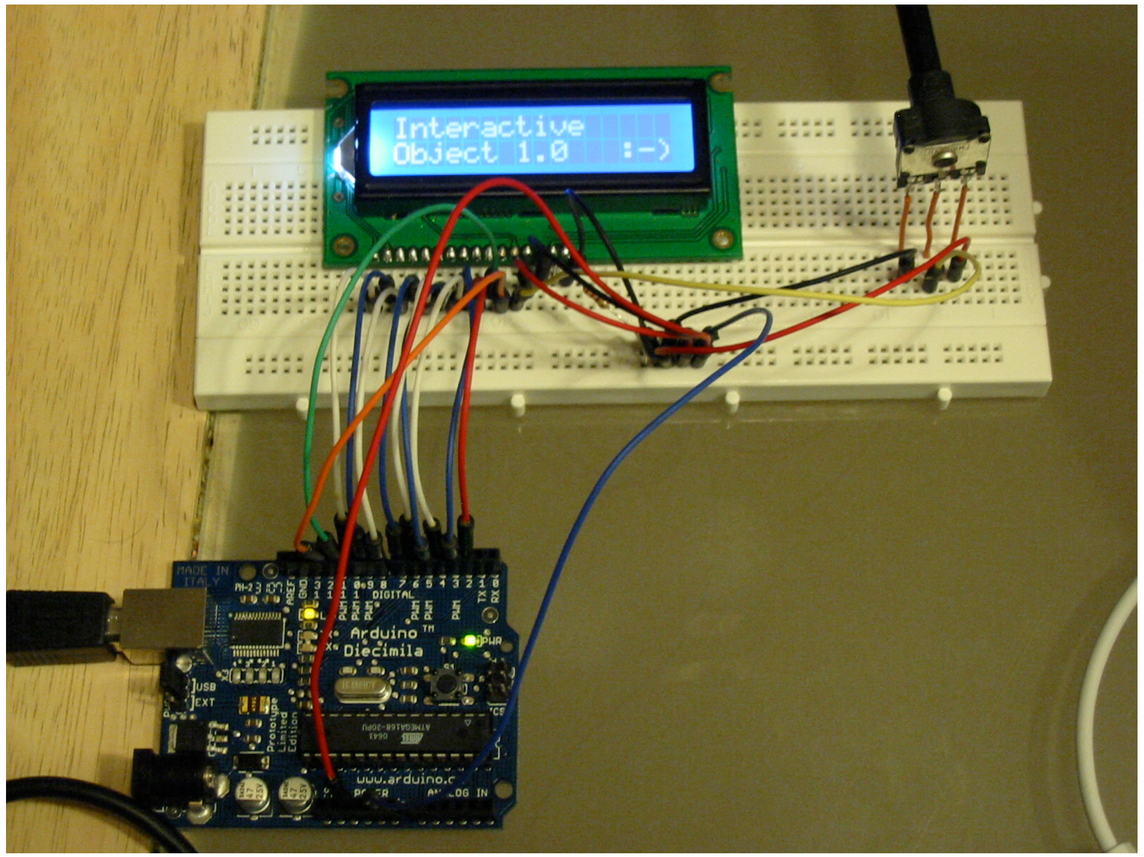
- Per poter utilizzare la LCDLibrary è necessario “importarla” (grazie alla direttiva #include) all’interno del programma
- Il tipo LiquidCrystal fornisce le funzioni per inizializzare e ripulire il display

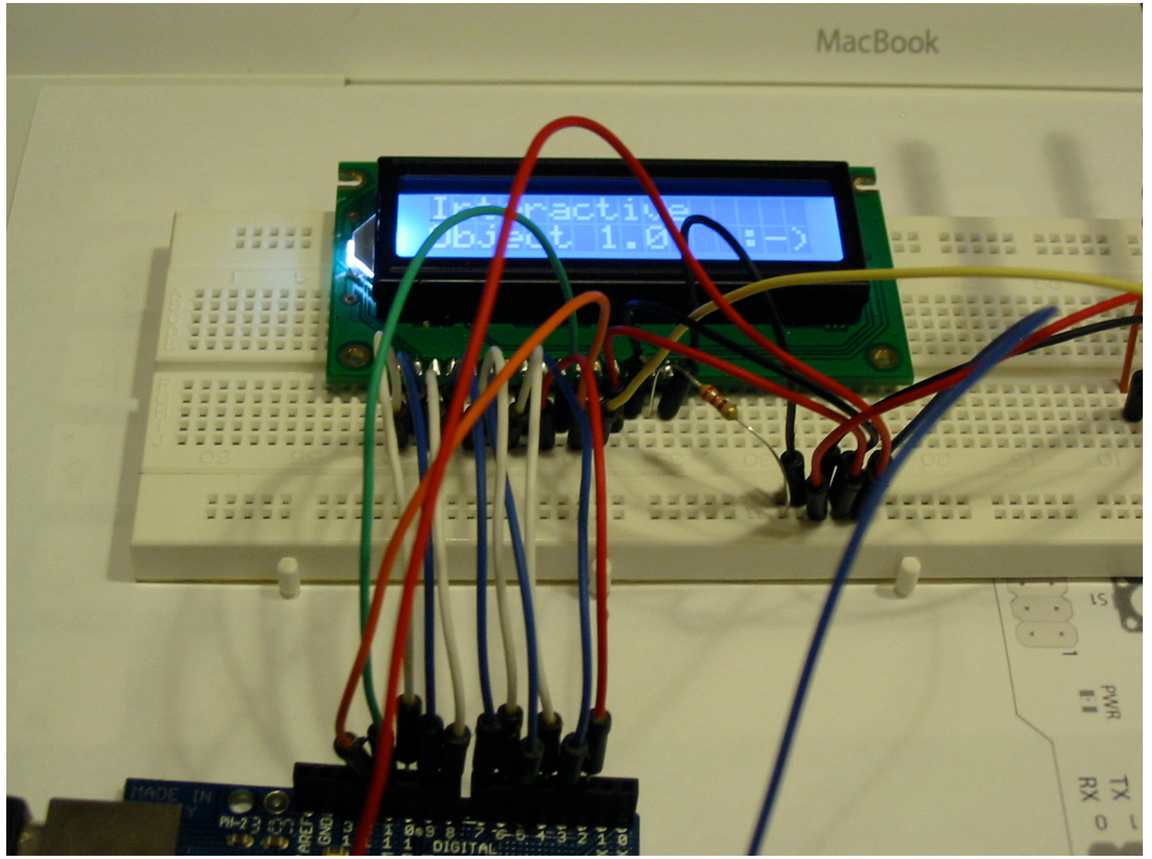
```
#include <LiquidCrystal.h>

LiquidCrystal lcd = LiquidCrystal();

void setup(void)
{
    lcd.init();
    lcd.clear();
}
```

Display LCD: codice

```
void loop(void)
{
    lcd.commandWrite(128);                  // va all'inizio della prima riga

    lcd.printIn("Arduino rocks!");          // scrive nella prima riga

    lcd.commandWrite(168);                  // va all'inizio della seconda riga

    lcd.printIn("LCD 2x14 display!");       // scrive nella seconda riga

    delay(1000);
}
```

Display LCD: il prototipo



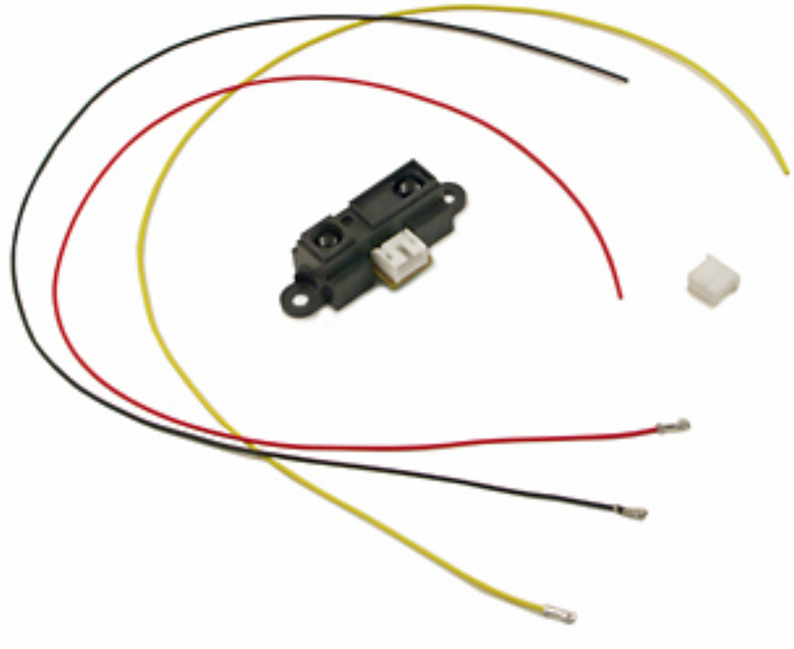


Misurare distanze

- I sistemi più diffusi per la valutazione delle distanze si basano su ultrasuoni o infrarossi
- Come un piccolo radar, il gruppo di misurazione è costituito da un emettitore e da un sensore che misura il ritardo (ultrasuoni) o l'angolo di riflessione (infrarossi) del segnale riflesso dall'oggetto verso il quale è puntato il sensore
- Il range di misura varia da pochi cm ad oltre 1m. Esistono sensori analogici e digitali

Sensore ad infrarossi

- Il modulo Sharp GP2D12 è un sensore ad infrarossi in grado di misurare distanze dai 10cm agli 80cm
- Dispone di una **uscita analogica** che restituisce una tensione di circa 2V alla minima distanza e 0.4V alla massima distanza



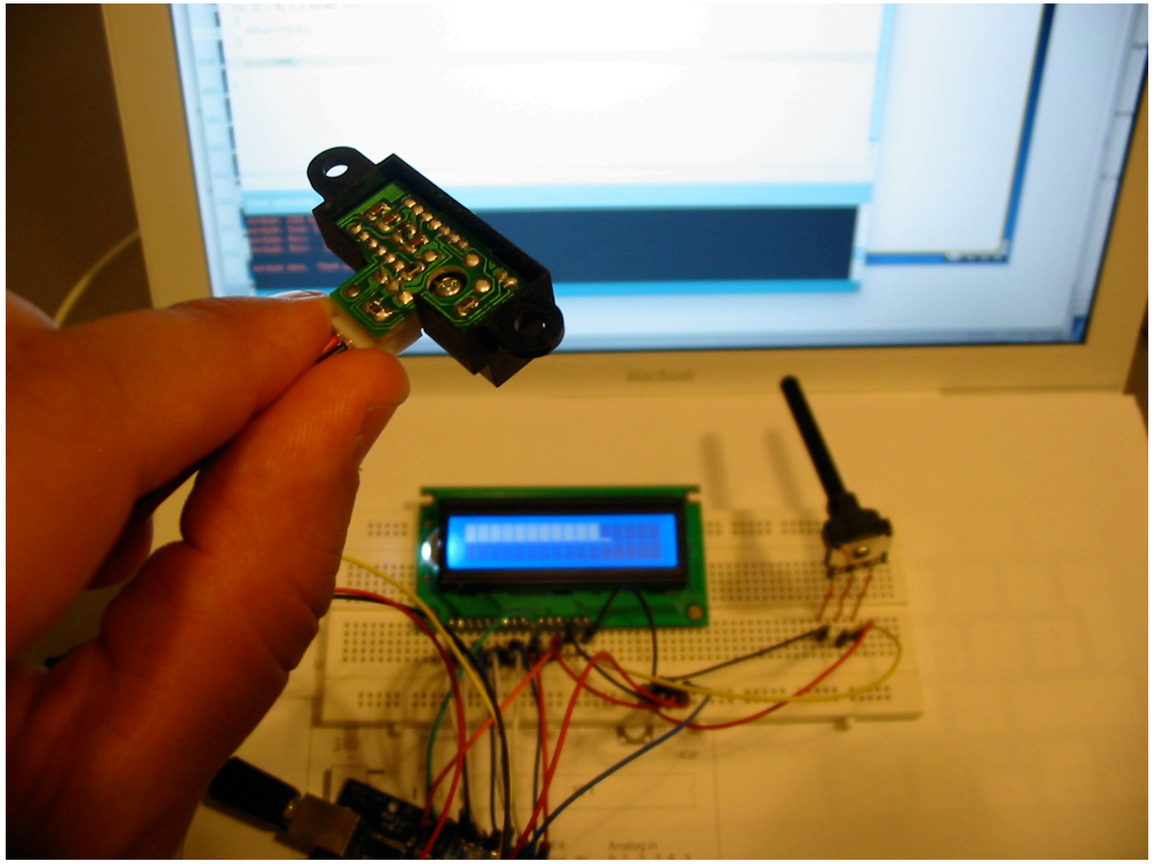
PIN 1 (yellow): signal
PIN 2 (black): GND
PIN 3 (red): +5V

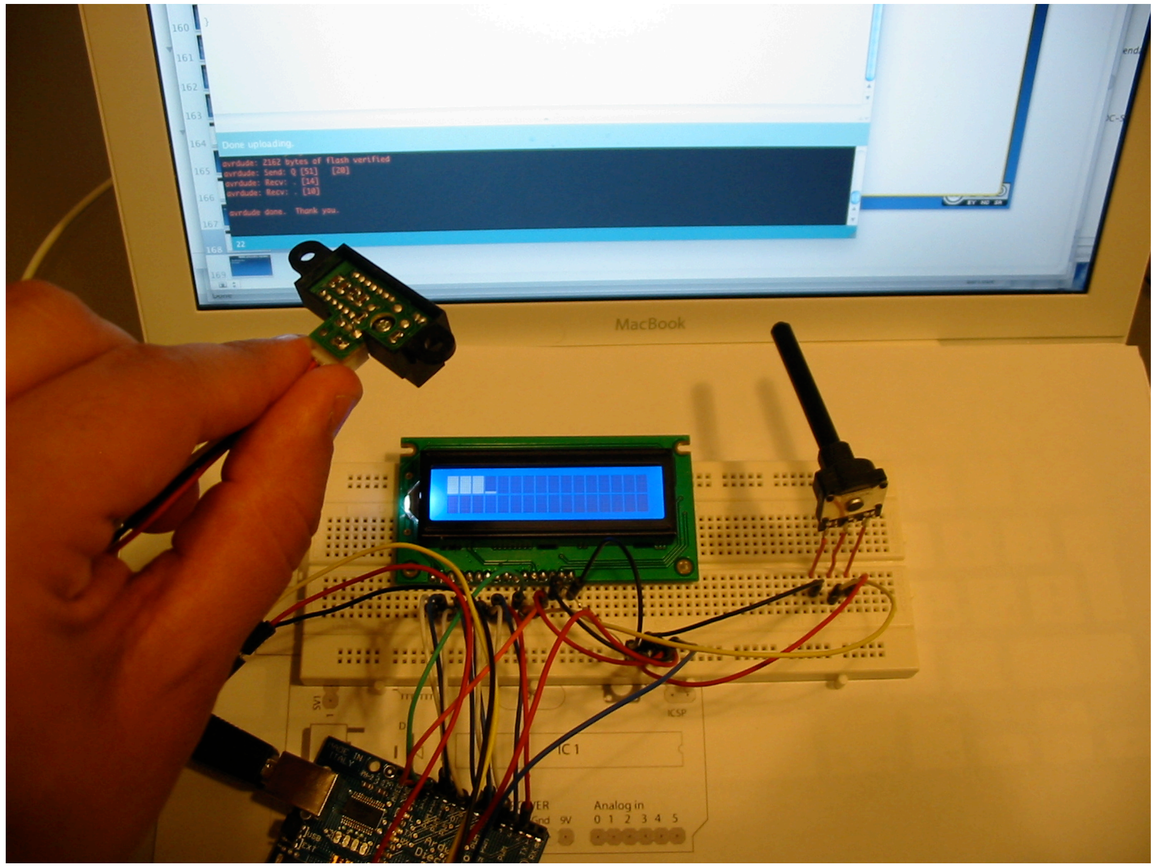
Sensore e display LCD!

- Utilizzando il display LCD ed il sensore ad infrarosso è possibile realizzare un semplice (ma non troppo preciso...!) misuratore di distanze
- La tensione letta è convertita in mV, riportata al range intero 6-20 e cambiata di segno

```
void loop(void)
{
    lcd.clear();

    value = 22 -
          (analogRead(0) * 5 ) / 100;

    for (i = 0; i < value; i++)
    {
        lcd.print(-1);
    }

    delay(100);

}
```

Sensore e display LCD!



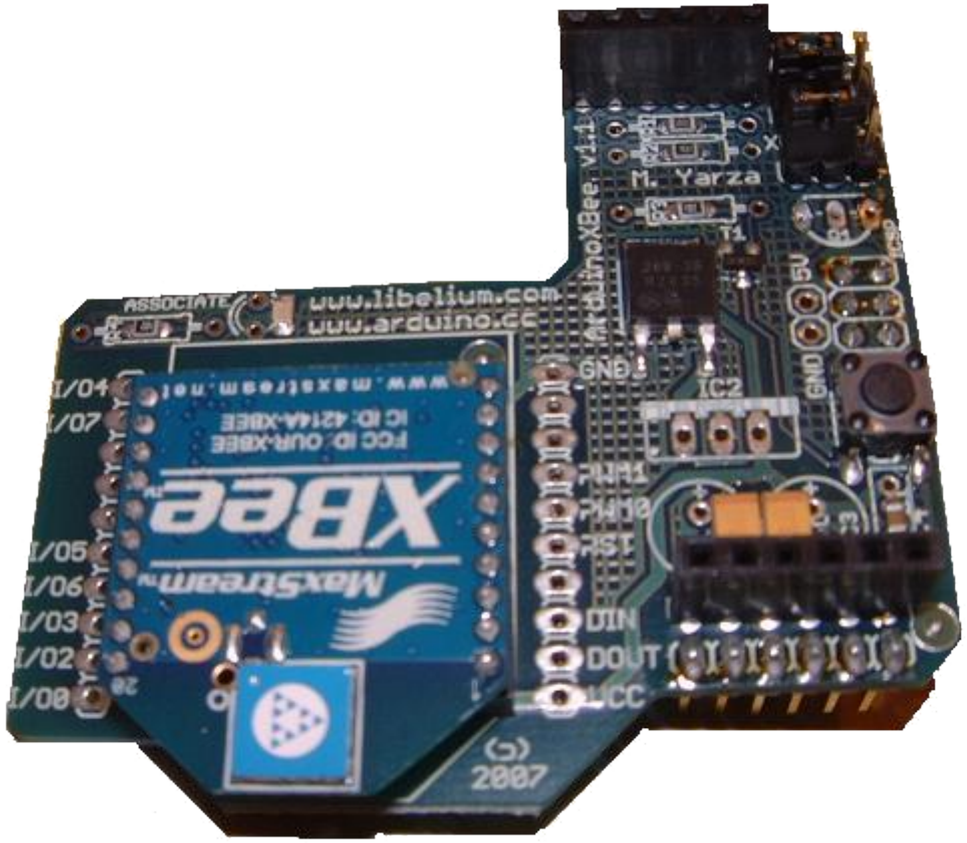


Comunicazione wireless

Arduino XBee e Bluetooth

Arduino XBee

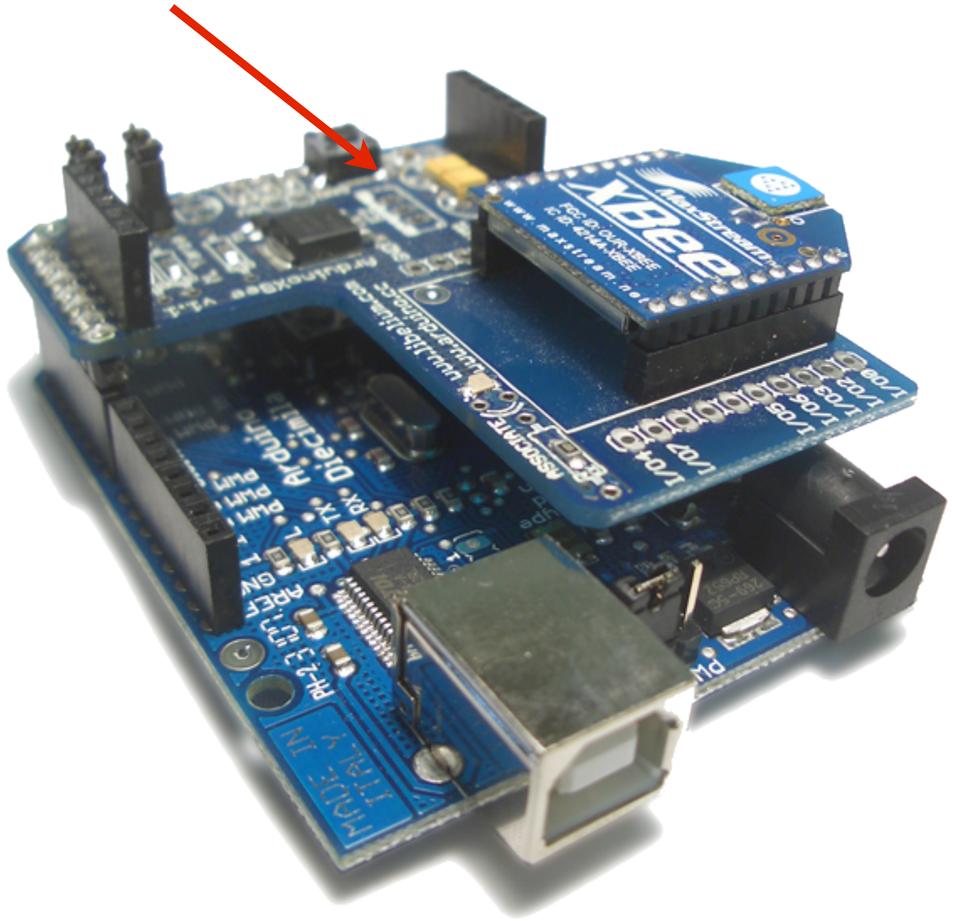
- L'Arduino XBee Shield è un modulo aggiuntivo per Arduino che consente la comunicazione seriale attraverso il protocollo ZigBee



Arduino XBee: note

- Per programmare Arduino è necessario rimuovere il modulo XBee, che altrimenti intercetta i comandi provenienti dal computer impedendo che siano ricevuti dal microcontrollore
- Il modulo potrà essere rimontato a programmazione avvenuta

Rimuovere durante la programmazione di Arduino

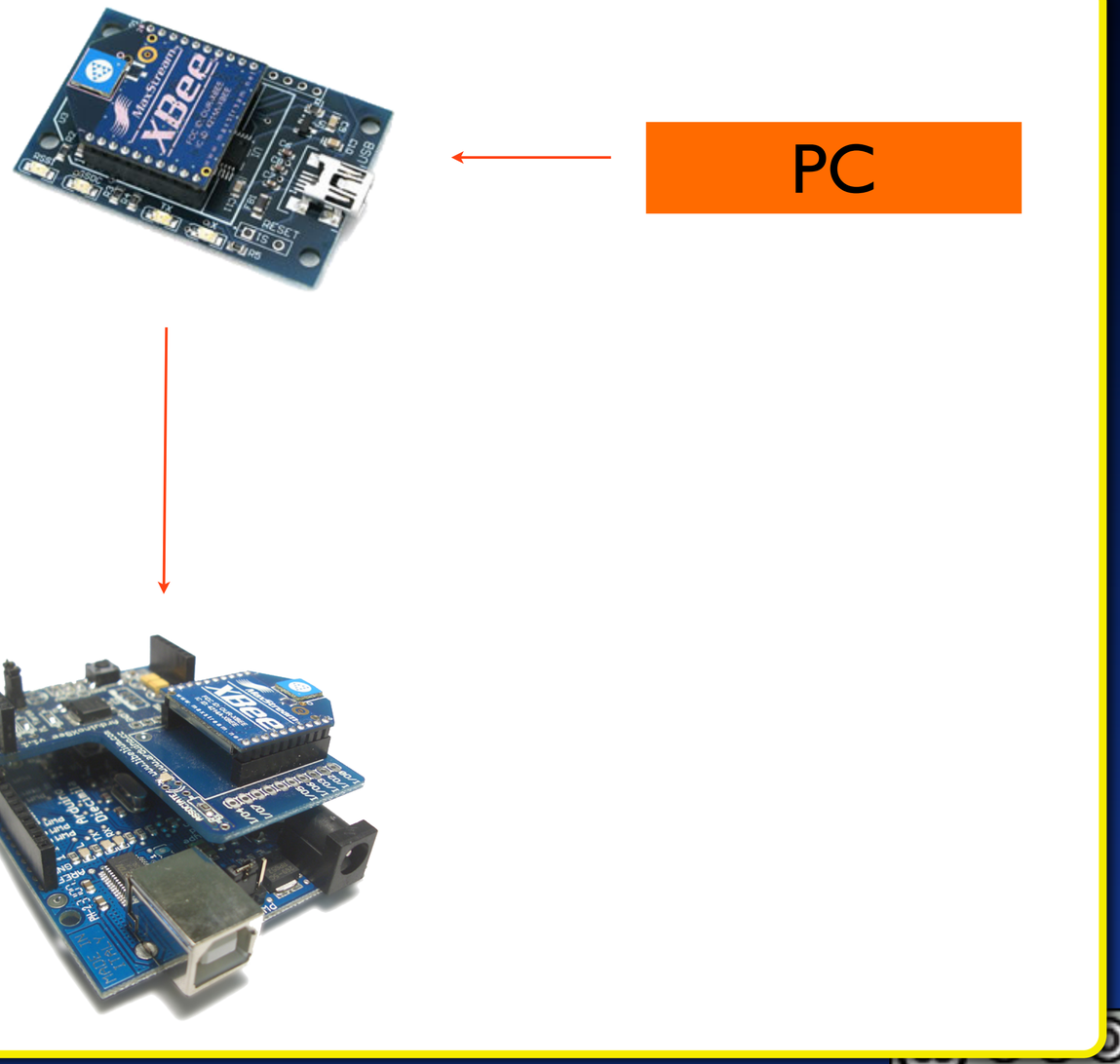


XBee USB Adapter

- È un adattatore che consente di connettere un modulo XBee (analogo a quello installato sull'Arduino XBee Shield) ad un computer attraverso porta USB
- Il modulo XBee è accessibile attraverso la porta seriale (virtuale) associata al modulo

XBee Beacon

- Utilizzando XBee è possibile realizzare un semplice radiofaro (**beacon**) eseguito su PC, che invia continuamente dei dati che, quando ricevuti da Arduino, attivano determinate funzioni
- XBee consente, dunque, di realizzare un semplice sensore di prossimità



XBee Beacon: lato PC

- Per attivare il Beacon su PC è necessario scaricare il pacchetto SerialBeacon disponibile all'indirizzo:

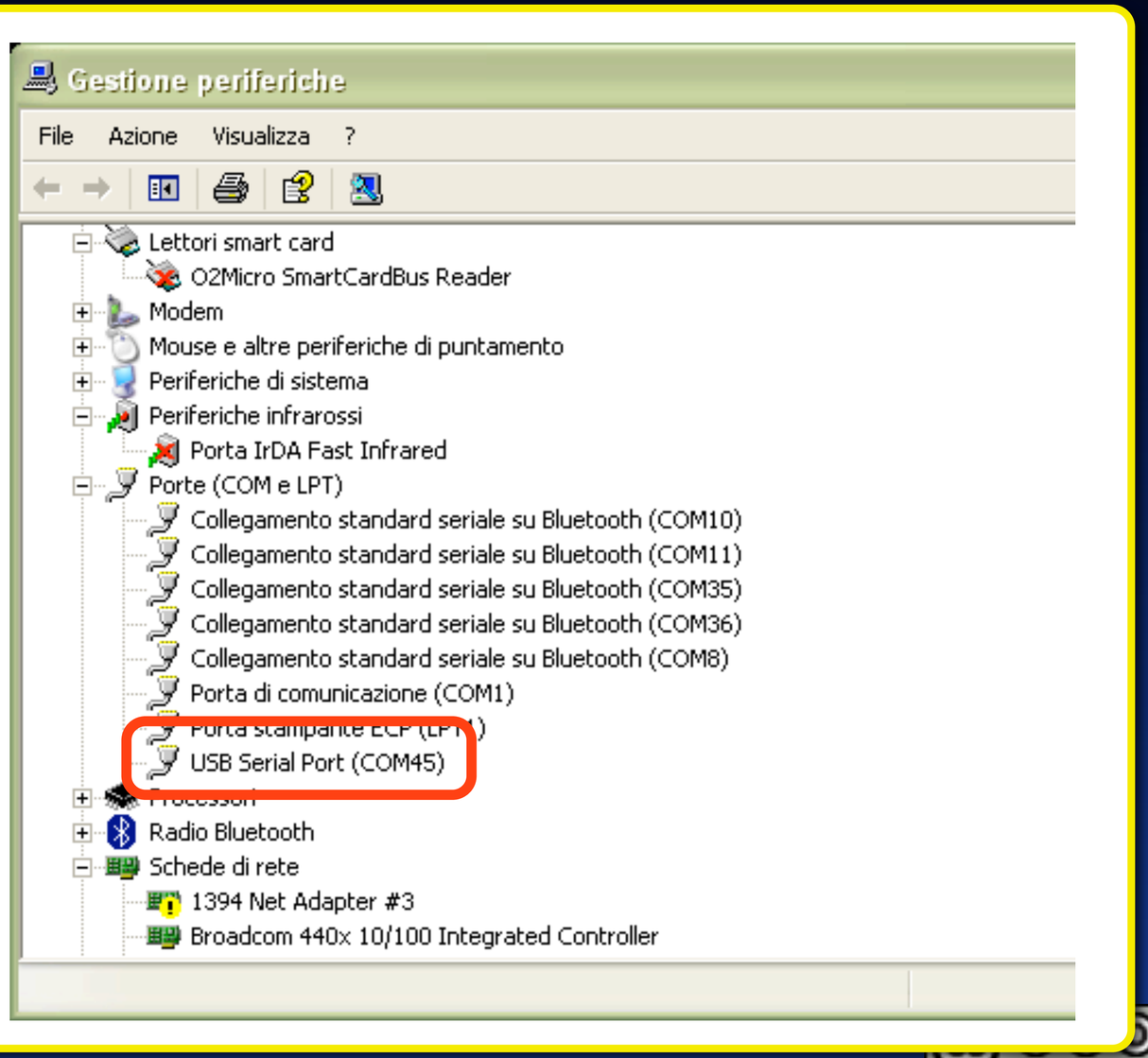
  http://www.gerdavax.it/data/SerialBeacon.zip

  e decomprimere l'archivio in una cartella

- Il file beacon.conf contiene i parametri di configurazione:
  - port=COMxx, dove "xx" è la porta seriale assegnata al modulo XBee
  - signal=x, dove "x" è il simbolo (lettera o numero) inviato via ZigBee
  - delay=x, dove "x" è l'intervallo in secondi tra un invio e il successivo

XBee USB Adapter

- Per determinare la porta seriale assegnata all'adattatore XBee USB, è necessario aprire il pannello **Gestione Periferiche** del gruppo **Sistema** all'interno del **Pannello di Controllo** di Windows

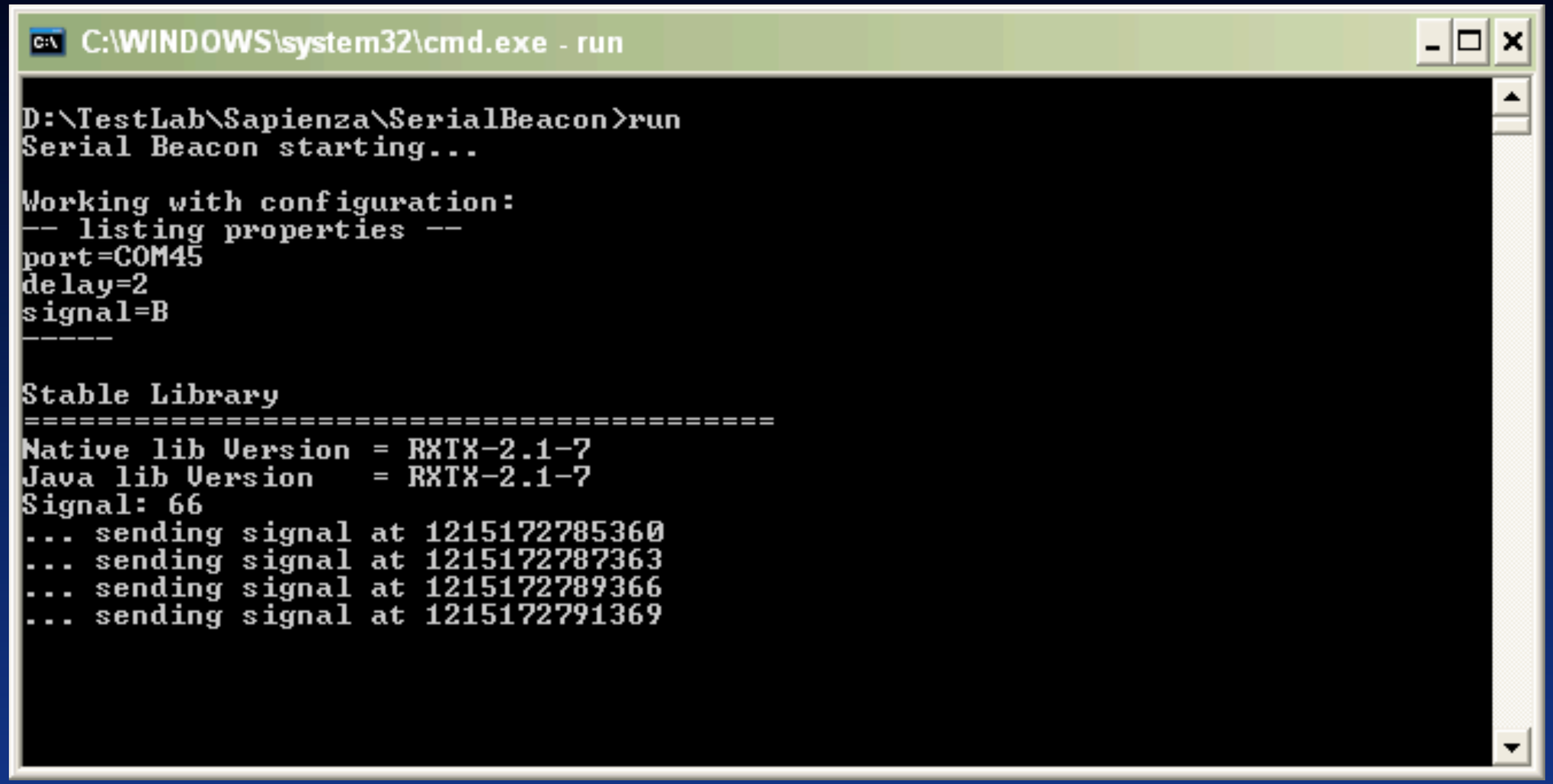


XBee Beacon

- Il beacon, attualmente disponibile sono per Windows, si attiva lanciando il script **run.cmd**

- Dopo alcuni secondi, durante i quali l'applicazione provvede ad inizializzare la porta seriale del computer, il beacon inizia a trasmettere il carattere indicato nel file di configurazione ad intervalli regolari

XBee Beacon in esecuzione



XBee Beacon: lato Arduino

- Per ricevere il segnale del beacon su Arduino è sufficiente:
  - aprire la connessione seriale a 9600bps: `Serial.begin(9600)`
  - se ci sono dati disponibili: `Serial.available() > 0`
  - leggere un singolo carattere: `Serial.read()`
  - attivare le funzioni in base al carattere letto

- La lettura può essere fatta ciclicamente, ma occorre **sempre verificare** che ci siano effettivamente dati disponibili

LED di prossimità

- Questo programma di esempio accede il LED sul PIN 13 quando il modulo XBee riceve il carattere “A” (corrispondente al carattere ASCII 65)
- Se il segnale che acceso il LED non è più rilevato per dieci secondi, il LED viene spento automaticamente

```
int read;
long time;

void setup()
{
  Serial.begin(9600);
}
```

LED di prossimità

```
void loop()
{
  if (Serial.available() > 0)
  {
    read = Serial.read();

    if (read == 65)
    {
      digitalWrite(13, HIGH);
      time = millis();
    } else
    {
      digitalWrite(13, LOW);
    }
```

Se ci sono dati in ingresso e il carattere letto è “A”, il LED viene acceso e viene memorizzato l’istante di ricezione.

Se il carattere non è “A”, il LED si spegne immediatamente

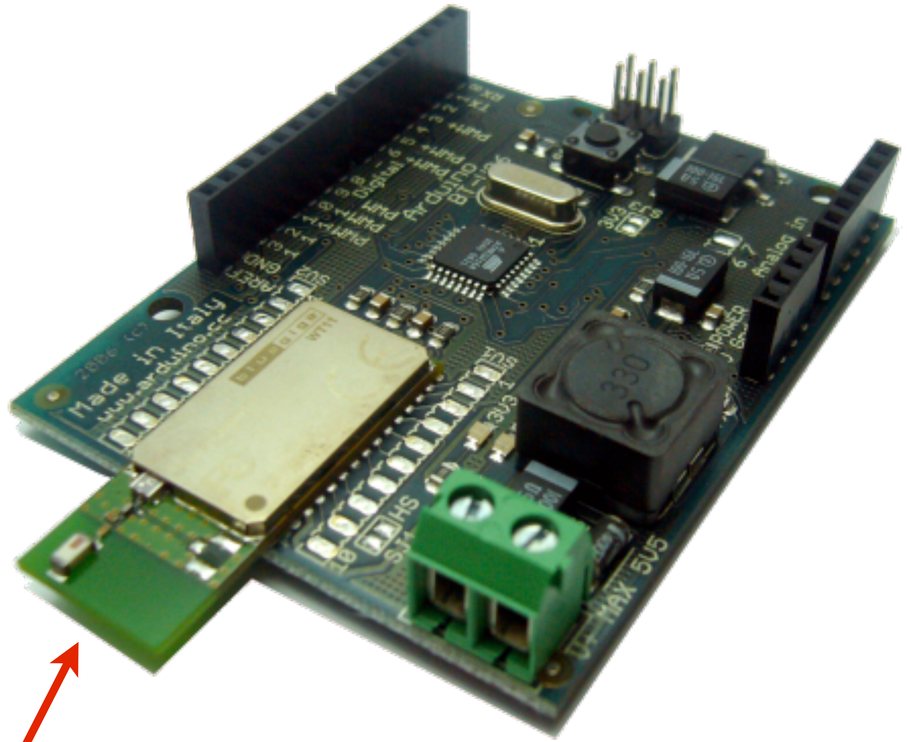
LED di prossimità

```
    } else
    {
      long now = millis();
      if ((now - time) > 10000)
      {
        digitalWrite(13, LOW);
      }
      delay(100);
    }
}
```

Se ci **non** sono dati in ingresso e sono trascorsi più di dieci secondi dall'ultima ricezione del carattere "A", il LED viene spento. L'applicazione attende 100 ms prima di effettuare una nuova lettura

Arduino Bluetooth

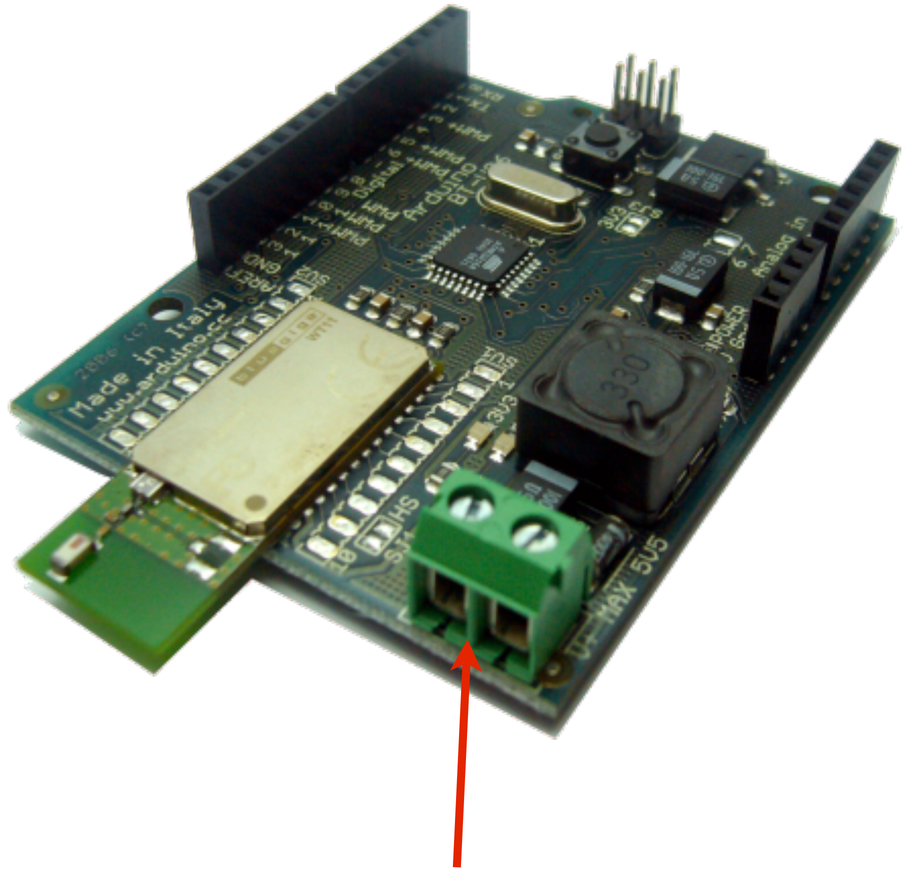
- È una versione di Arduino equipaggiata con un modulo Bluetooth e priva di porta USB
- Programmazione e comunicazione seriale avvengono esclusivamente attraverso connessione wireless Bluetooth



Modulo Bluetooth

Alimentazione

- Non essendo presente la porta USB, è necessario alimentare Arduino Bluetooth attraverso batterie o un alimentatore
- La tensione massima di alimentazione è 5V attraverso i morsetti posti accanto al modulo Bluetooth



Punto di alimentazione

Interfacciamento a PC

PC controller

- È possibile utilizzare Arduino per attivare funzioni sul PC attraverso una connessione USB

- Sul computer è necessario disporre di un opportuno software che legga i dati attraverso la porta seriale (virtuale) ed esegua l'operazione richiesta; è possibile utilizzare qualsiasi linguaggio in grado di accedere alle porte seriali e leggere caratteri

Licenza Creative Commons

- Attribuzione-Non commerciale-Condividi allo stesso modo 2.5IT
- E’ consentito:
  - riprodurre, distribuire, presentare in pubblico quest’opera
  - modificare quest’opera
- Alle seguenti condizioni
  - Attribuzione: le opere derivate devono attribuire la paternità a questa
  - Non commerciale: non è possibile utilizzare quest’opera per fini commerciali
  - Condividi allo stesso modo: opere derivate da questa devono essere distribuite con la stessa licenza