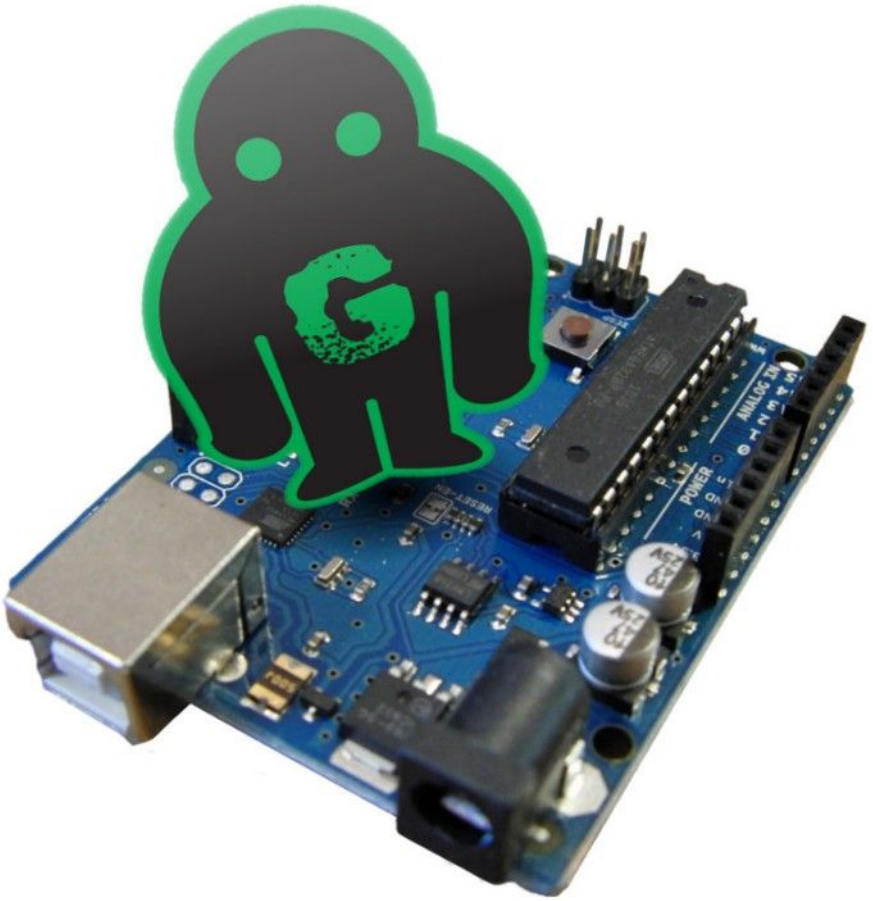
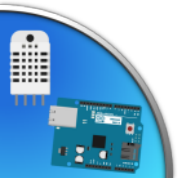
CORSO ARDUINO

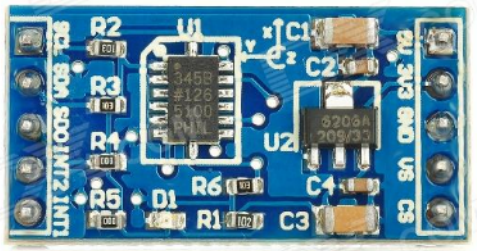


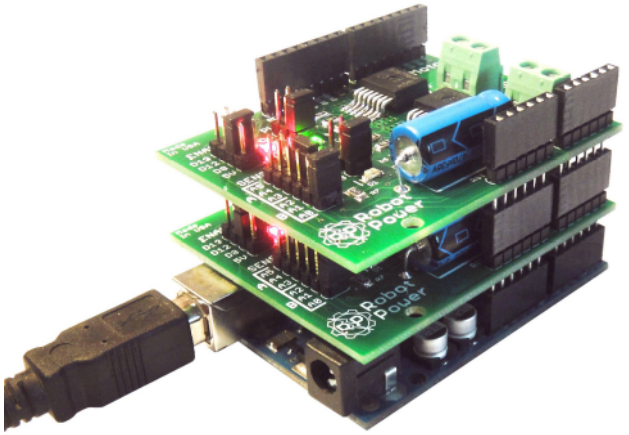
Jacopo Belli
Giulio Fieramosca
Luca Mattii
GOLEM 2016



Di cosa parliamo oggi...

- **Generazione di suoni**
- **Sensoristica complessa**: i protocolli di comunicazione;
- **Shield per Arduino**: circuiti avanzati pronti all'uso.





Generazone di suoni: buzzer

Il modo più semplice per generare un suono: onda quadra (PWM con duty cycle 50% e frequenza variabile!)

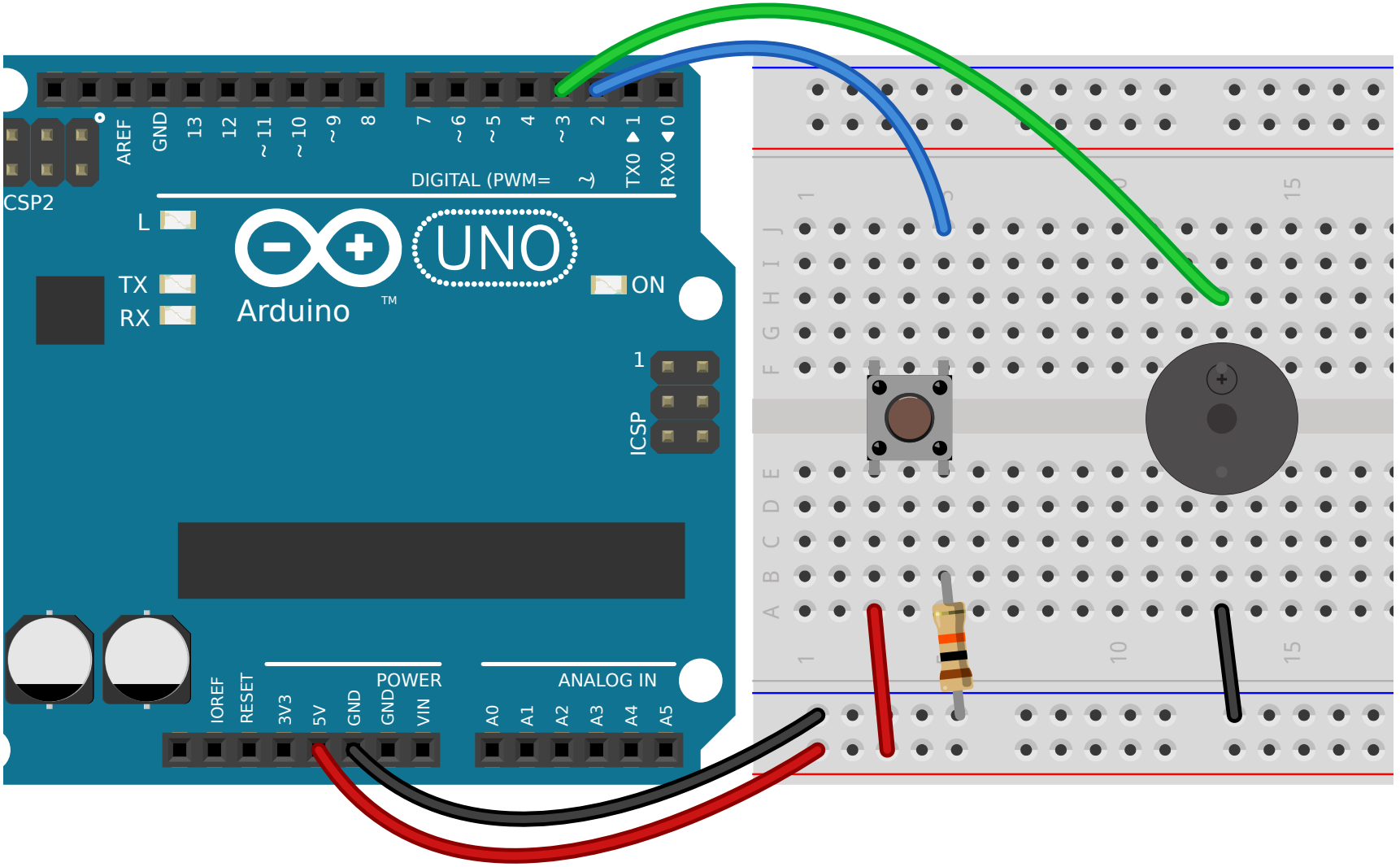
Buzzer passivo

elemento piezoelettrico: varia la sua forma in modo proporzionale alla tensione

Buzzer attivo

integra un oscillatore: se alimentato *suona* ad una frequenza fissa

Buzzer Attivo

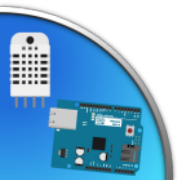


Buzzer Attivo

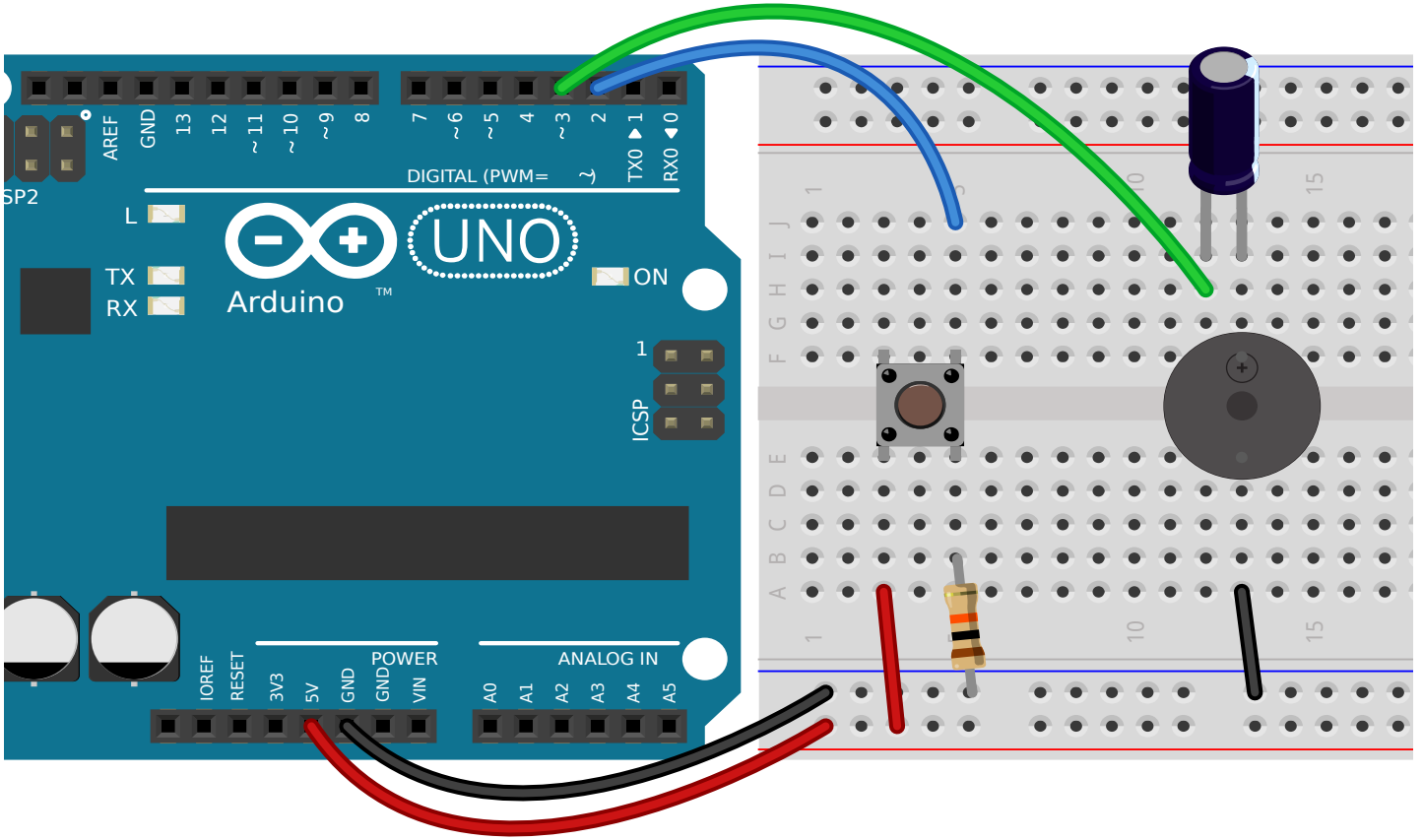
```
const byte PIN_PULSANTE = 2;
const byte PIN_BUZZER   = 3;

void setup() {
  pinMode(PIN_PULSANTE, INPUT);
}

void loop() {
    bool lettura = digitalRead(PIN_PULSANTE);
    if (lettura == HIGH) {
        digitalWrite(PIN_BUZZER, HIGH);
    }
    else {
        digitalWrite(PIN_BUZZER, LOW);
    }
    delay(10);
}
```

Riga equivalente e più compatta

```
digitalWrite(PIN_BUZZER, digitalRead(PIN_PULSANTE));
```

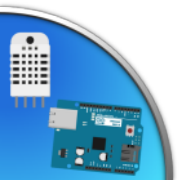


Buzzer **Passivo**: funzione tone



Buzzer **Passivo**: funzione tone

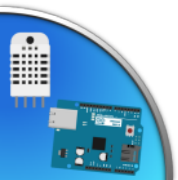
```
const byte PIN_PULSANTE = 2;
const byte PIN_BUZZER   = 3;

void setup() {
  pinMode(PIN_PULSANTE, INPUT);
}

void loop() {
    bool lettura = digitalRead(PIN_PULSANTE);
    if (lettura == HIGH) {
        tone(PIN_BUZZER,440);
    }
    else {
        noTone(PIN_BUZZER);
    }
    delay(10);
}
```



Buzzer **Passivo**: funzione tone

Variante: bip quando premi il pulsante

```
const byte PIN_PULSANTE = 2;
const byte PIN_BUZZER   = 3;
bool lettura_precedente;

void setup() {
  pinMode(PIN_PULSANTE, INPUT);
  lettura_precedente = digitalRead(PIN_PULSANTE);
}

void loop() {
    bool lettura = digitalRead(PIN_PULSANTE);
    if (lettura == HIGH && lettura_precedente == LOW) {
        tone(PIN_BUZZER, 440, 100);
    }

    lettura_precedente = lettura;
    delay(10);
}
```



Note sulle funzioni di tono

```
tone(pin, frequenza, durata);
```

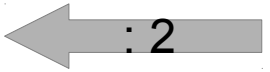
Viene attivata la generazione della frequenza, il programma prosegue normalmente e l'onda quadra “termina” automaticamente

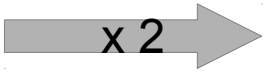
```
tone(pin, frequenza);
```

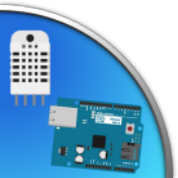
Viene attivata la generazione della frequenza “senza fine”. Il suono termina con la funzione...

```
noTone();
```

Do	Re	Mi	Fa	Sol	La	Si
261	294	330	349	392	440	494







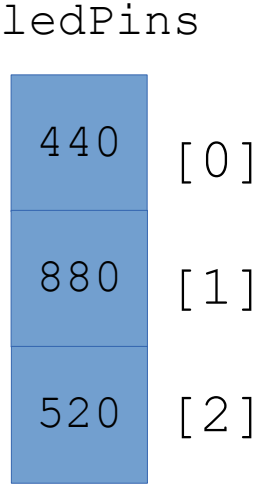
Array

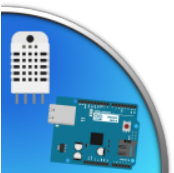
- Più informazioni *dello stesso tipo* possono essere raggruppate in un **array**;
- Nella dichiarazione specifico quanti "cassetti" dovrà avere l'array, e successivamente vi posso accedere **contando da 0!**

```
int melodia[3];
melodia[0] = 440;
melodia[1] = 880;
melodia[2] = 520;
```

oppure

```
int melodia[3] = {440, 880, 520};
```





Applicazione degli array al ciclo for

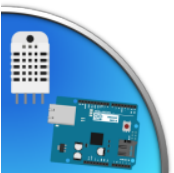
- Posso ripetere un blocco di codice tenendo il conto di quanti giri vado a fare.
- **Esempio:** dato un array di pin a cui sono connessi dei LED, li imposto tutti in OUTPUT e li spengo:

```
const byte PINLEDS[5] = {2, 4, 5, 10, 11};

void setup () {
    for (byte c = 0; c < 5; c++) {
        pinMode (PINLEDS[c], OUTPUT);
        digitalWrite (PINLEDS[c], LOW);
    }
}
```

Inizializzo una o più variabili da usare come contatore

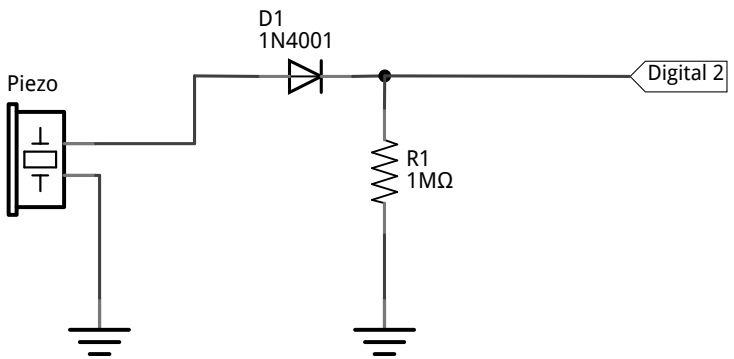
Controllo ad ogni ciclo se la condizione è vera

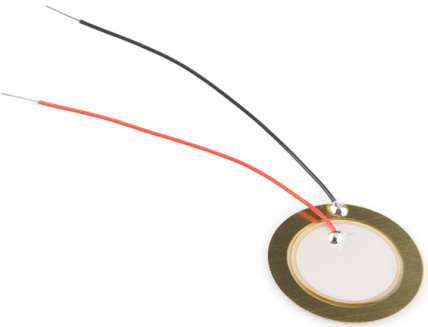
Incremento il contatore alla fine di ogni ciclo

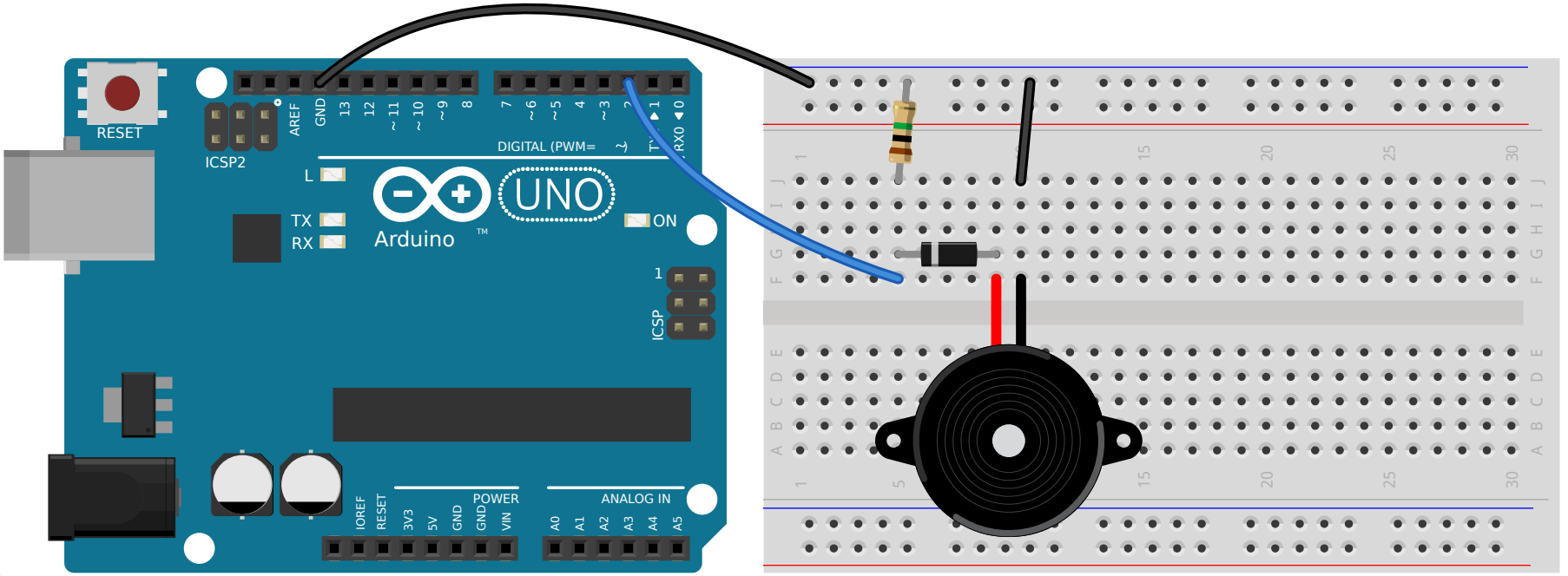


Stefano Panichi e Giulio Fieramosca

Buzzer “Nudo”

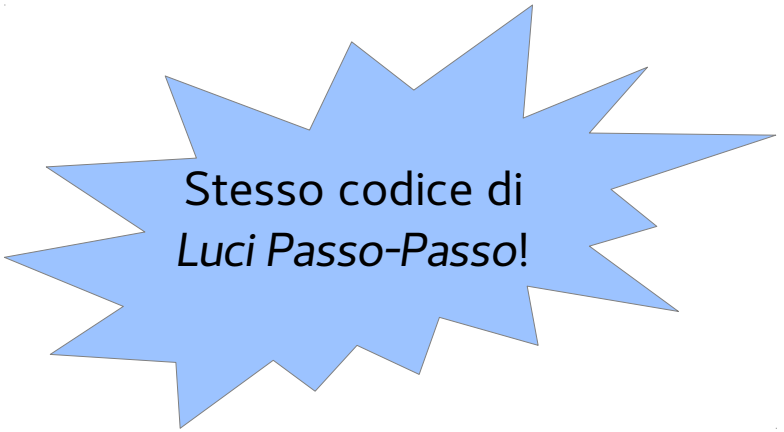


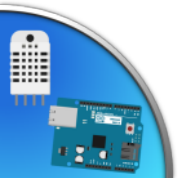




Buzzer “Nudo”

```
const byte PIN_LED      = 13;
const byte PIN_PIEZO    = 2;
bool lettura_precedente = LOW;

void setup() {
  pinMode(PIN_PIEZO, INPUT);
  pinMode(PIN_LED, OUTPUT);
}

void loop() {
  bool lettura = digitalRead(PIN_PIEZO);
  if (lettura == HIGH && lettura_precedente == LOW)
    digitalWrite(PIN_LED, !digitalRead(PIN_LED));

  lettura_precedente = lettura;
  delay(10);
}
```

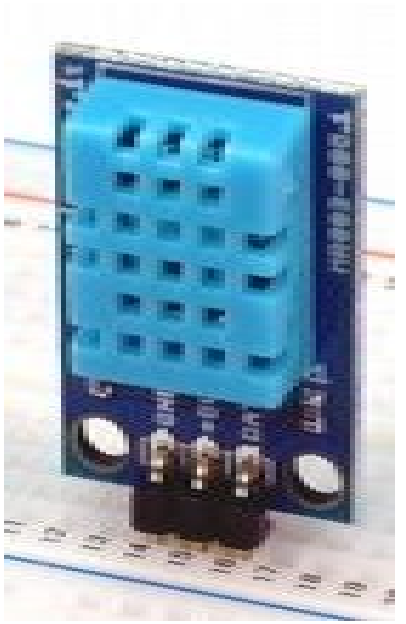




Protocolli di comunicazione

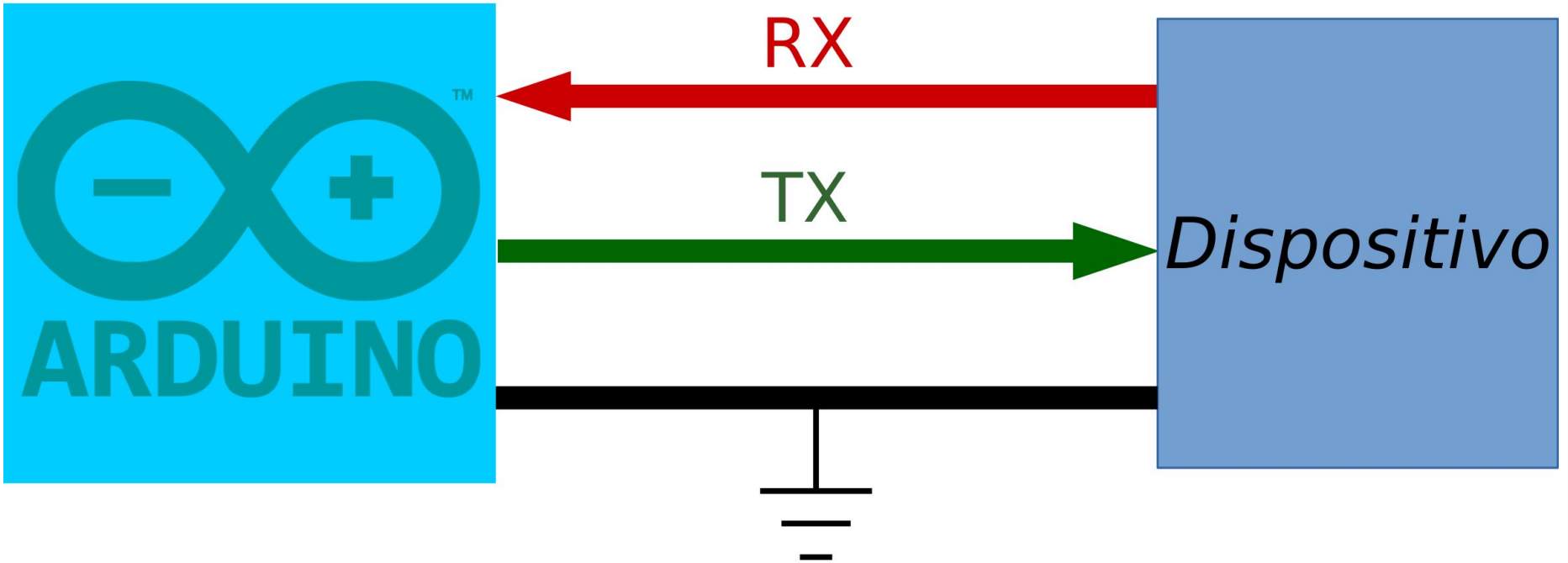
I microcontrollori dispongono di alcune periferiche in grado di gestire dei protocolli di comunicazione complessi:

- **Seriale**: TX/RX
- **I²C**: SDA/SCL
- **SPI**: MISO/MOSI/SCK/SS
- **OneWire**: DAT



Seriale (asincrona)

Parole chiave: TTL Serial, UART, USART,

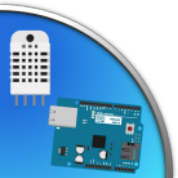


⚠ RX e TX sono dal punto di vista di Arduino!

Alcune funzioni utili

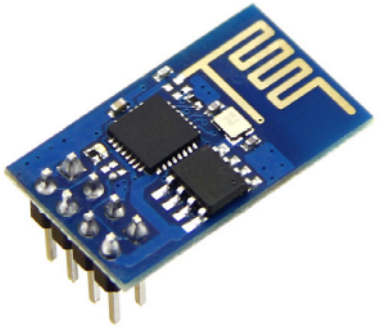
- Leggere dei dati inviati verso arduino:
  - Un byte per volta: `char carattere = Serial.read()`
  - Un numero intero: `int numero = Serial.parseInt()`
  - Oppure un float: `float numero = Serial.parseFloat()`
  - La `parseInt` e `parseFloat` attendono un po' di tempo, impostabile con `Serial.setTimeout(millisecondi)`
- Inviare dati al computer: `Serial.print` e `Serial.println`

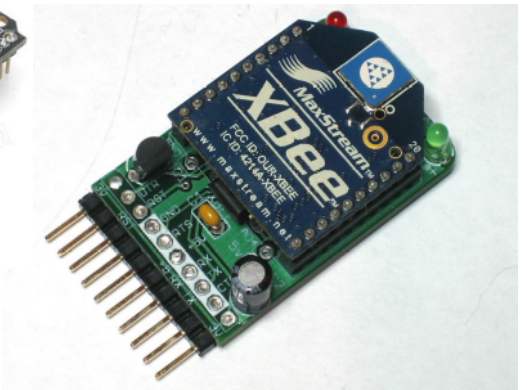
  Con `Serial.available` si controlla se il buffer ha dei dati



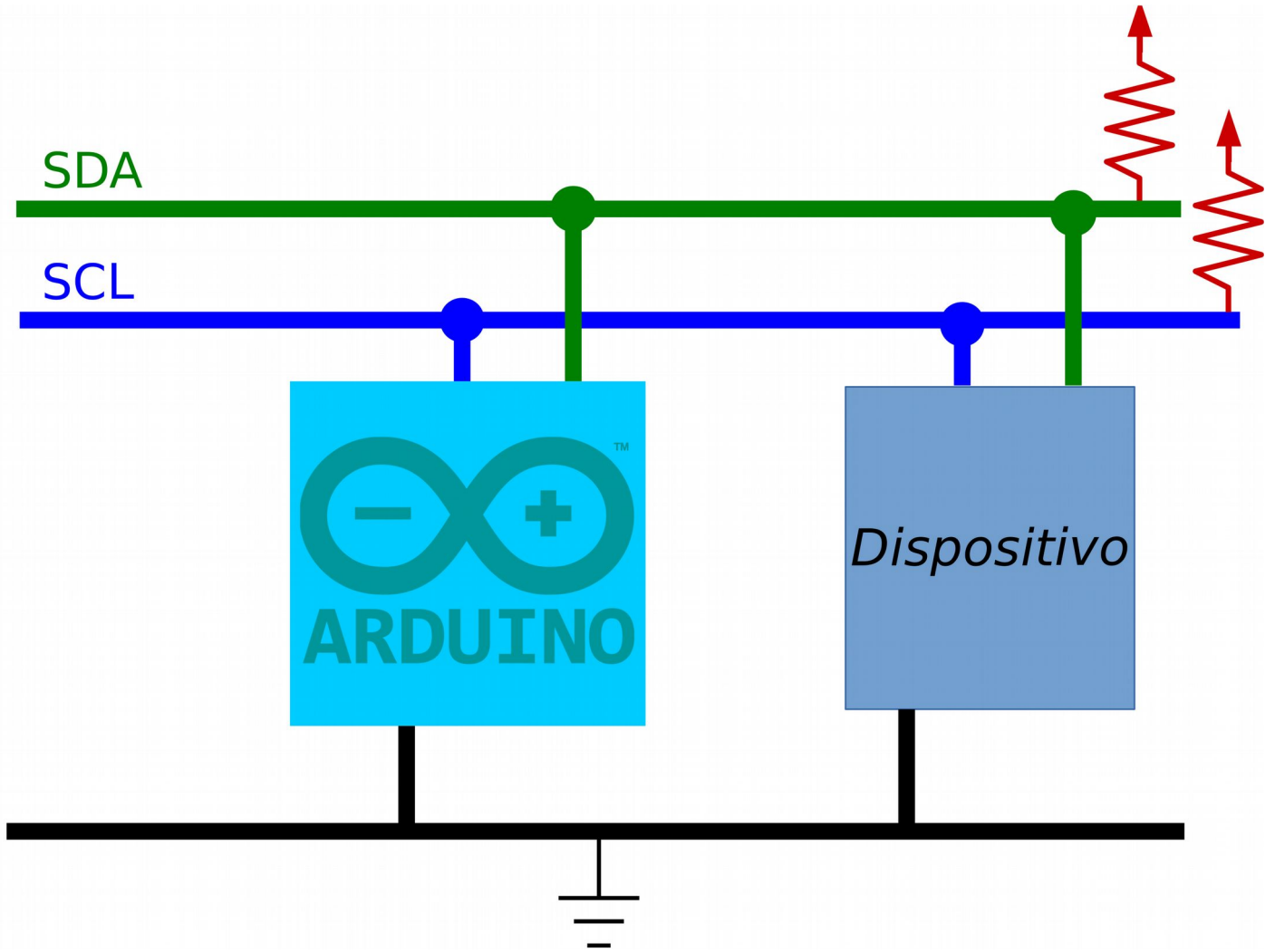
Altre applicazioni della seriale

- Comunicazione fra due Arduini
- Schede wireless basate su ESP8266
- Schede bluetooth
- Dispositivi radio wireless (XBEE, GSM, GPS)



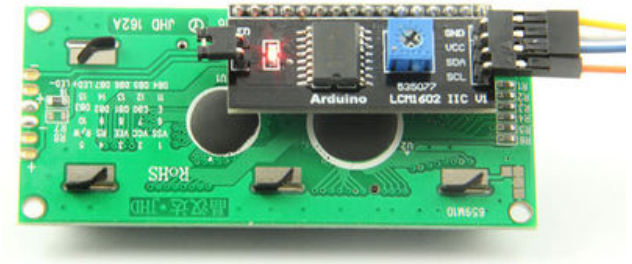


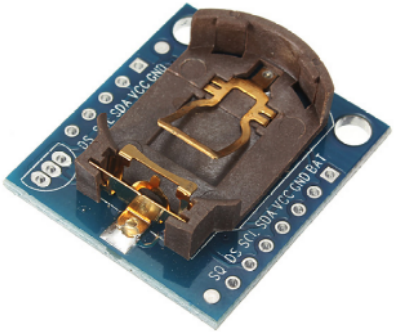
I²C

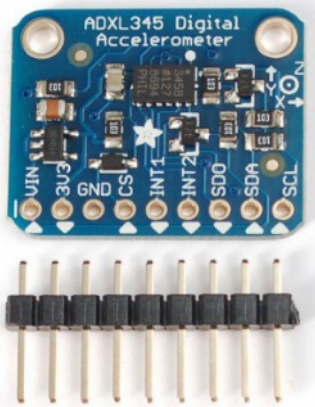


Applicazioni dell'I²C

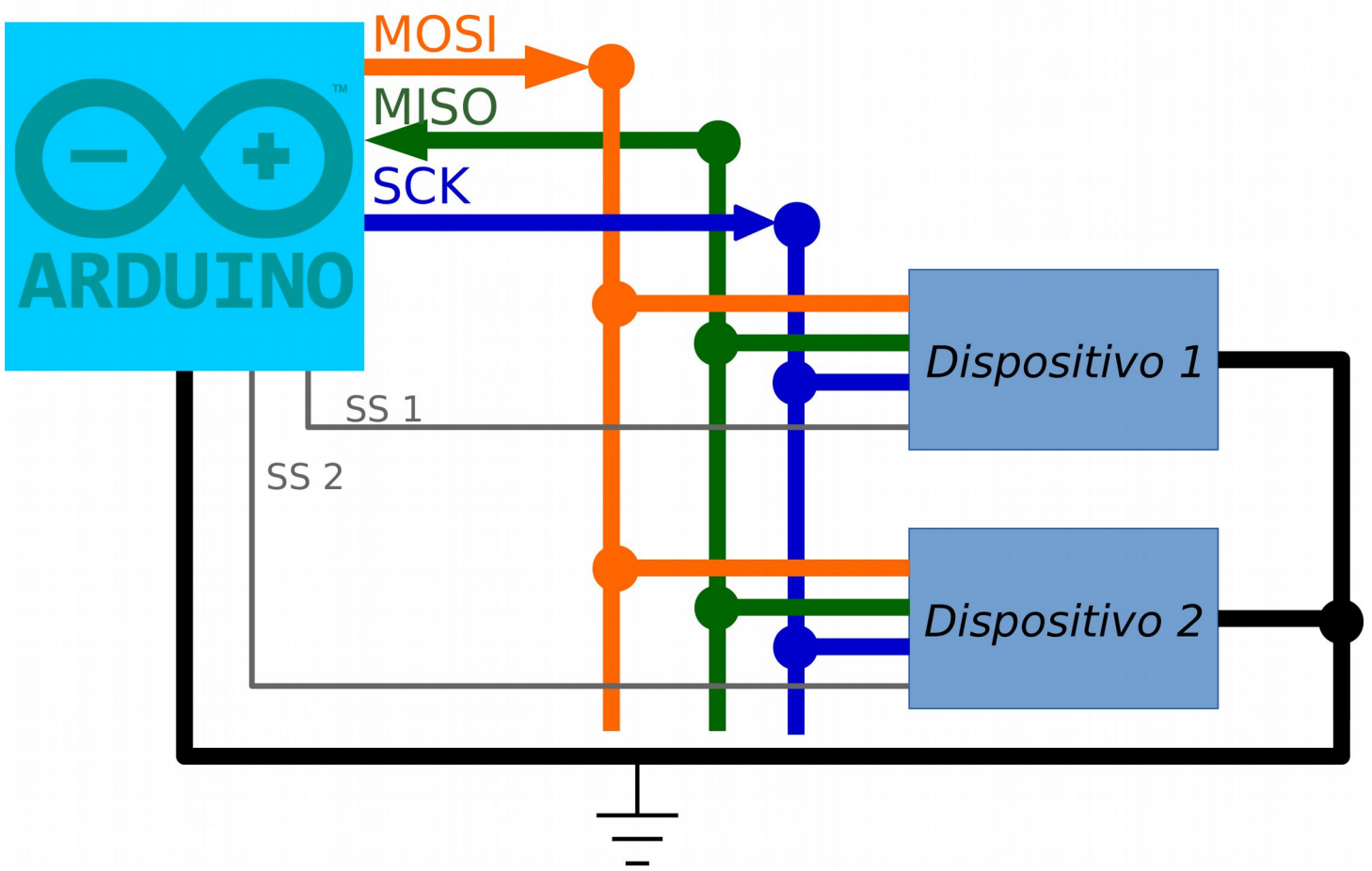
- I/O Expander: di solito usato per controllare un display via I²C
- Modulo RTC: temporizzatore che memorizza data e ora anche quando Arduino è spento
- Accelerometri, memorie esterne,







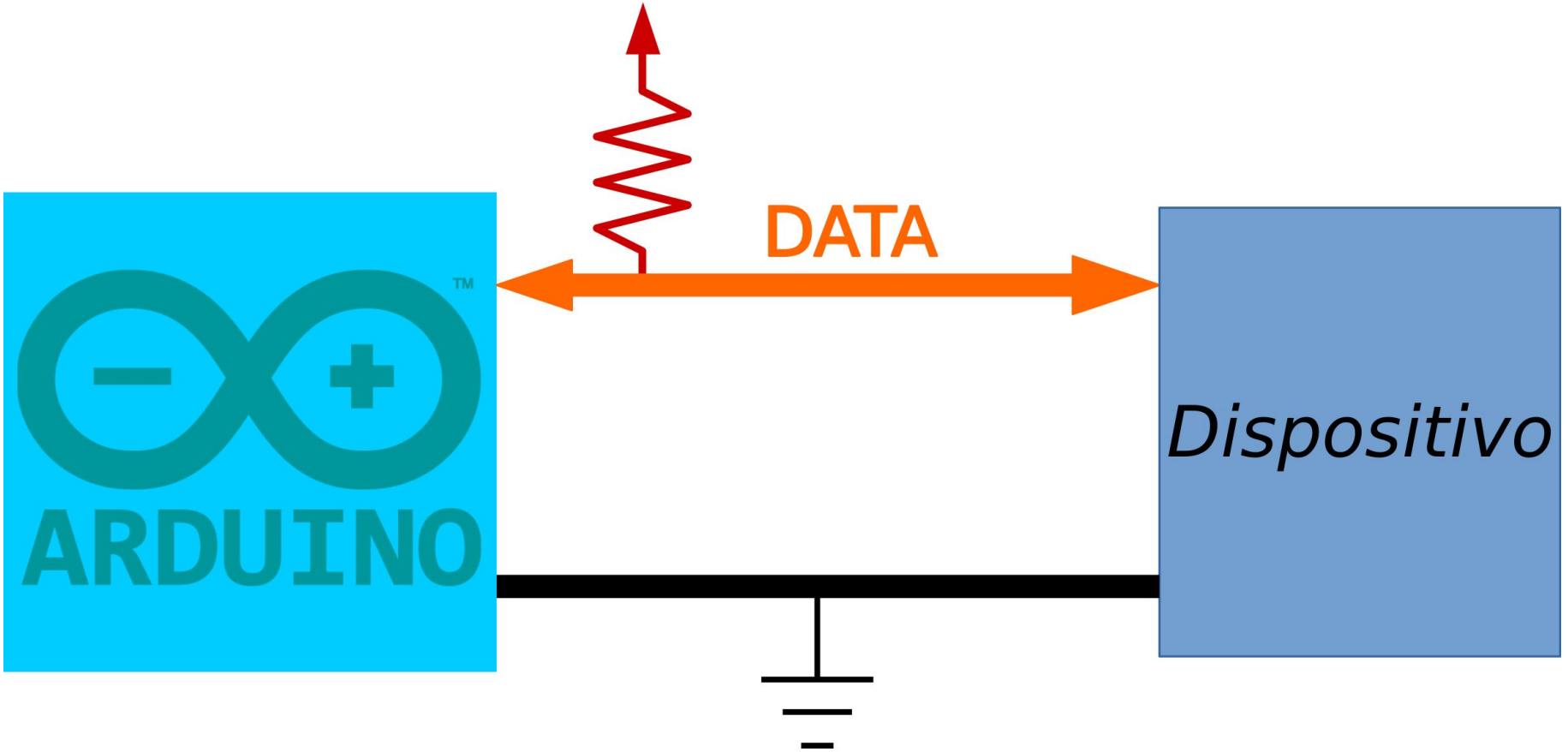
SPI



Applicazioni dell'SPI

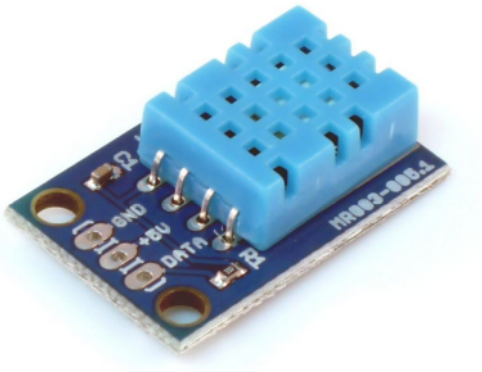
- Controllo di schede SD e microSD
- Comunicazione con Ethernet Shield o WiFi Shield
- Ricevitori RFID ed NFC
- Lo Shift Register sfrutta “una specie di SPI”

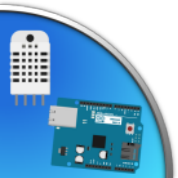
One Wire



Applciazioni dell'One Wire

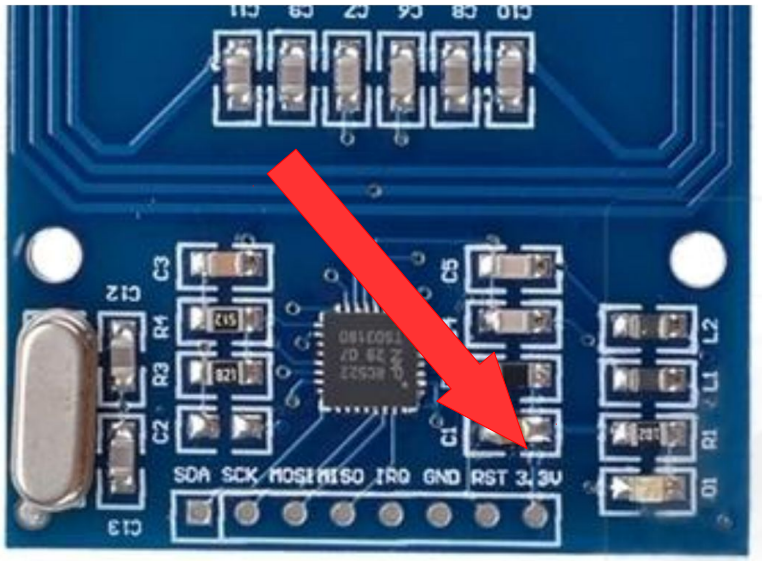
- Sensore temperatura e umidità DHT11
- Sensore per telecomandi IR

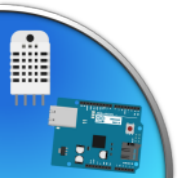




Attenzione!

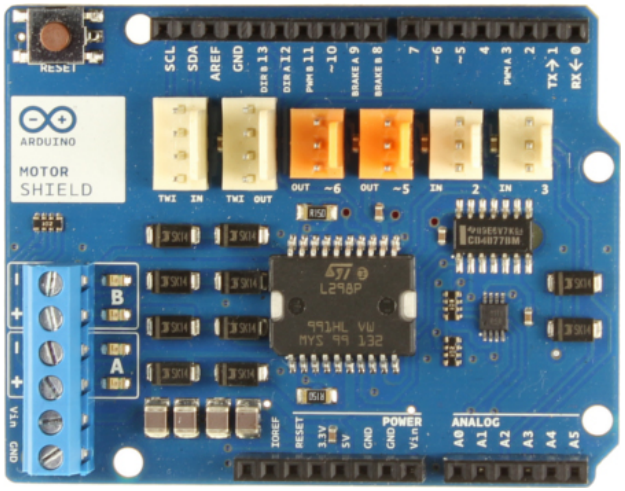
Non tutti i dispositivi funzionano alla stessa tensione di Arduino. Spesso si trovano oggetti che devono essere alimentati a 3.3V e dialogano a quella tensione!



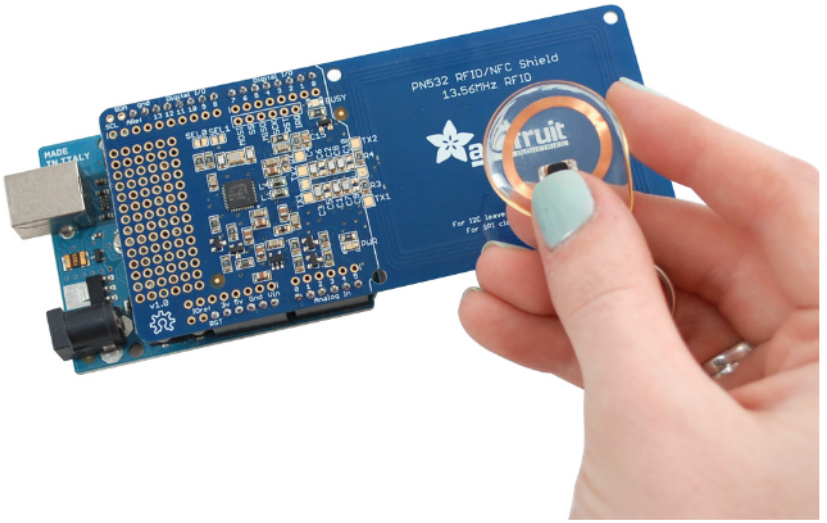


Altri Shields interessanti...

Motor Shield



RFID Shield



Wave Shield