

1

Introduzione ad Arduino

1.1 Che cos'è Arduino?

Arduino è una scheda elettronica basata su un microcontrollore (un computer realizzato su un unico circuito integrato), al quale sono stati aggiunti i componenti necessari per renderlo economico e facilmente utilizzabile anche da chi non è un professionista del settore elettronico-informatico.

A questo dispositivo hardware i progettisti hanno aggiunto un ambiente di sviluppo integrato (IDE), scaricabile gratuitamente dal sito www.arduino.cc, con cui si può programmare e caricare nella memoria interna di Arduino il software (sketch) che ne definisce il funzionamento in base alle proprie esigenze.

La **figura 1.1** rappresenta i due principali elementi con cui avremo a che fare: (a) la scheda Arduino Uno; (b) l'ambiente di sviluppo (IDE) per la scrittura del software.

Il nome Arduino è ispirato a quello del bar di Ivrea frequentato dal gruppo di progettisti dell'Interaction Design Institute che ha messo a punto il sistema nel 2005; il nome del bar richiama a sua volta quello di Arduino d'Ivrea, Re d'Italia dal 1002 al 1014.



Questo sistema dal 2005 si è diffuso in tutto il mondo, non solo in ambito didattico e hobbyistico, ma anche in quello scientifico e industriale; oltre alla

Figura 1.1 a) Scheda Arduino Uno;
b) ambiente di sviluppo (IDE).

scheda Arduino Uno sono state prodotte altre schede con diverse caratteristiche, per adattarsi alle differenti esigenze (Arduino Mega, Leonardo, LilyPad, Yun, ...).

Tutte le schede Arduino sono *open source* (gli schemi sono pubblici e non coperti da copyright) e consentono così agli utenti di costruirle in modo indipendente (cloni) e adattarle alle loro particolari esigenze; anche il software è *open source* e sta crescendo grazie ai contributi di programmatori di tutto il mondo.

1.2 Che cosa può fare Arduino?

Uno dei principali campi d'impiego di Arduino è il *Physical Computing*. (figura 1.2): la scheda Arduino acquisisce e misura alcune grandezze fisiche dal mondo reale (come temperatura, pressione, posizione) attraverso dei trasduttori collegati agli ingressi e, dopo aver elaborato queste informazioni, interviene sul mondo reale mediante **attuatori** (motori, lampade, display, relè) connessi alle uscite.

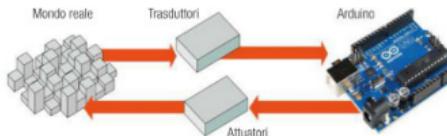


Figura 1.2 Schematizzazione del Physical Computing.

Un esempio pratico di *Physical Computing* è il **termostato d'ambiente**, che comanda l'accensione della caldaia di un impianto di riscaldamento in modo che la temperatura dell'ambiente rimanga costante:

- un trasduttore di temperatura invia al sistema elettronico (per esempio Arduino) una grandezza elettrica (per esempio una tensione) il cui valore dipende dalla temperatura dell'ambiente;
- il sistema acquisisce tale grandezza (in pratica la converte in un numero binario e la memorizza), la confronta con il valore di riferimento impostato dall'utente e stabilisce se accendere la caldaia oppure no, inviando un opportuno segnale elettrico;
- la caldaia e i termostoni costituiscono l'attuatore, che agisce sull'ambiente per modificarne la temperatura in base al risultato dell'elaborazione.

La figura 1.3 rappresenta più in dettaglio gli elementi che compongono un sistema basato su Arduino:

- i **trasduttori** convertono una grandezza fisica (temperatura, luminosità, suono, velocità, pressione, posizione, ...) rilevata nel mondo reale, in una grandezza elettrica (tensione, resistenza, corrente, ...); anche un pulsante, impiegato dall'utente per inviare informazioni binarie ad Arduino, può essere pensato come trasduttore;

» i circuiti di condizionamento convertono le uscite di ogni trasduttore in segnali che possono essere interpretati dagli ingressi di Arduino:

- input digitali: tensioni alte (5 V) o basse (0 V);
- input analogici: tensioni variabili tra 0 V e 5 V.

» Il **computer**, collegato ad Arduino mediante cavo USB, ha le seguenti funzioni:

- fornire la tensione di alimentazione alla scheda;
- redigere (*editing*) il programma (*sketch*) mediante la finestra dell'ambiente di sviluppo (*IDE*) (figura 1.1b);
- caricare (*loading*) nella memoria di Arduino il programma, convertito nel linguaggio macchina del microcontrollore (*compilazione*);
- ricevere informazioni sull'esecuzione del programma, mediante l'uso del Monitor Seriale, per correggerne eventuali errori (*debugging*).

Una volta messo a punto il programma e caricato definitivamente su Arduino, il computer potrà essere rimosso (funzionamento *stand alone*) e l'alimentazione dovrà essere fornita da un alimentatore esterno, mediante l'apposito connettore.

» Attraverso le uscite (*output*) di Arduino si inviano i segnali di comando agli attuatori, che agiranno sull'ambiente come specificato dal programma in esecuzione.

» Molti attuatori richiedono in ingresso segnali potenti (tensioni e correnti elevate) che Arduino non è in grado di fornire direttamente: quindi tra le uscite di Arduino e gli attuatori è necessario interporre circuiti di potenza (*azionamenti*), basati su *transistor* o circuiti integrati, in grado di amplificare i segnali di uscita.

Ogni pin **digitale** di Arduino può essere programmato come *input* o come *output*, mediante opportune istruzioni inserite nello sketch. Al contrario, i sei pin analogici hanno solo funzioni di *input*.

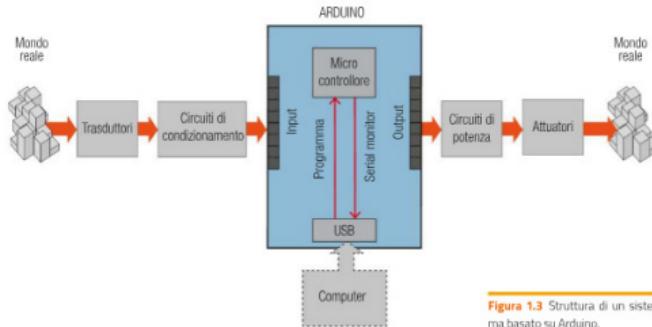


Figura 1.3 Struttura di un sistema basato su Arduino.

1.3 I vantaggi di Arduino

Le caratteristiche principali di Arduino, che hanno contribuito alla sua diffusione a livello mondiale, sono le seguenti:

- l'economicità dell'hardware (il prezzo è variabile in base al modello, ma è nell'ordine di qualche decina di euro) e la gratuità del software, scaricabile dal sito web di Arduino www.arduino.cc;
- la scheda contiene già tutti i componenti necessari per la programmazione e un utilizzo di base;
- la semplicità di collegamento della scheda, sia verso il computer (USB), sia verso gli altri circuiti elettronici (mediante pin messi a disposizione sulla scheda, senza necessità di saldature);
- la relativa semplicità della programmazione in un linguaggio ad alto livello, derivato dal C/C++, che non richiede la conoscenza del linguaggio assembler del microcontrollore;
- l'ampia disponibilità di librerie di funzioni che semplificano il pilotaggio di display LCD, motori, trasduttori, servomotori;
- l'ampia disponibilità di schede aggiuntive (**shield**), realizzate da vari produttori, per interfacciare la scheda Arduino a motori, a carichi di potenza, a trasmettitori wireless, alla rete internet;
- la filosofia open source dell'hardware e del software.

1.4 L'hardware: la scheda Arduino Uno

Le principali caratteristiche della scheda Arduino Uno (figura 1.4), a cui si farà riferimento nel presente testo, sono le seguenti:

- **microcontrollore:** ATmega328P (Atmel);
- **tensione consigliata per l'ingresso di alimentazione:** 7V - 12V (il + è il contact interno del connettore); per esempio, Arduino può essere alimentato con una batteria da 9 V. Tale tensione può essere fornita anche tra i pin Vin e GND;
- **pin di ingresso/uscita digitali:** 14 (di cui 6 programmabili come uscite PWM);
- **pin d'ingresso analogici:** 6;
- **livelli di tensione per ingressi/uscite digitali:** 0 V e 5 V
- **range di tensione per ingressi analogici:** da 0 V a 5 V (modificabile con il pin AREF);
- **correnti massime per i pin di ingresso/uscita digitali:** 20 mA (oltre i 40 mA il microcontrollore si danneggia);
- **Flash Memory:** 32 KB (memoria non volatile, dove viene memorizzato il programma);

➢ **SRAM:** 2 KB (memoria volatile, dove sono memorizzate le variabili durante l'elaborazione);

➢ **EPPROM:** 1 KB (memoria non volatile, dove si possono immagazzinare dati mediante la libreria di istruzioni `<EEPROM.h>`);

➢ **Frequenza di clock:** 16 MHz.

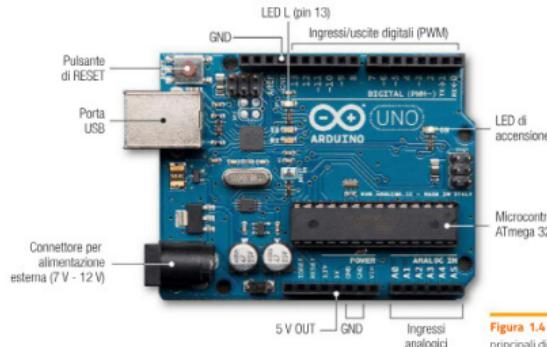


Figura 1.4 I pin e i componenti principali di Arduino Uno.

Le funzioni dei pin di Arduino Uno di uso comune sono le seguenti:

➢ **Digital input/output (0-13):** ognuno dei 14 digital pins può essere impostato come input o output, mediante l'istruzione `pinMode()`; per la lettura di un bit in ingresso si utilizza l'istruzione `digitalRead()`, mentre per la scrittura in uscita `digitalWrite()`. I digital pin hanno un resistore di pull-up interno (descritto nel capitolo 2) disconnesso per default; per attivarlo si usa l'istruzione `pinMode(pin, INPUT_PULLUP)`.

Alcuni digital pin hanno delle funzionalità speciali:

- **PWM:** i pin 3, 5, 6, 9, 10, 11, se impostati come uscite, possono fornire segnali PWM mediante l'istruzione `analogWrite()`, come descritto nei capitoli 2 e 5, per pilotare motori, servomotori, lampade, ecc.
- **LED (L):** il LED sulla scheda è connesso al digital pin 13.
- **Serial:** i pin 0 (RX) e 1 (TX) sono usati anche per scambiare dati seriali a livelli TTL.
- **External Interrupts:** i pin 2 e 3 possono ricevere un segnale di interrupt, come descritto nel paragrafo 4 del capitolo 4.

- Analog inputs (pin A0 - A5): le tensioni analogiche sui pin, comprese tra 0 V e 5 V, sono lette mediante l'istruzione `analogRead()` e convertite proporzionalmente in numeri binari di 10 bit (1024 possibili valori binari differenti, che corrispondono ai numeri decimali da 0 a 1023). Possono anche funzionare come pin di input/output digitale, riferendosi agli indirizzi da 14 a 19.
- Il pin **Reset**: portato BASSO resetta il microcontrollore, facendo ripartire da zero l'esecuzione dello sketch; ciò equivale alla pressione del pulsante **Reset**.
- Pin di alimentazione (5 V e GND): forniscono una tensione stabilizzata di 5 V utilizzabile per alimentare circuiti esterni (corrente max. 200 mA).

1.5 Il software per la programmazione di Arduino

Il software Arduino (IDE, Integrated Development Environment o Ambiente di Sviluppo Integrato) rende semplice la scrittura (*editing*) del programma (sketch), la ricerca degli errori di programmazione (*debugging*) e il trasferimento del programma sulla scheda (*loading*).

Il software è scritto in Java e basato su **Processing** e altri software *open source*; esso può essere utilizzato con qualsiasi scheda Arduino e funziona su Windows, Mac OS X e Linux.

Il linguaggio di programmazione è il **Wiring**, derivato dal C/C++.

■ Per iniziare ad utilizzare Arduino Uno:

1. Entrare nel sito internet <https://www.arduino.cc> e scaricare l'ultima versione del software Arduino (IDE), adatta al proprio sistema operativo.
2. Installare il software sul computer.
3. Collegare la scheda Arduino Uno a una presa USB del computer, per alimentarla e programmarla: si accende il LED verde ON sulla scheda. Una volta programmato, Arduino potrà funzionare scollegato dal computer (*stand alone*), fornendo alla scheda la tensione di alimentazione da 7 V a 12 V sull'apposito connettore, mediante un alimentatore esterno o una batteria da 9 V (il + è il contatto interno).
4. Lanciare il software Arduino.
5. Tra i menuù a tendina seguire il percorso: **Strumenti > Scheda** e selezionare la scheda (board) Arduino Uno.
6. Tra i menuù a tendina seguire il percorso: **Strumenti > Porta** e selezionare la porta seriale a cui è collegato Arduino.

■ Per verificare la connessione Arduino-computer

Per verificare rapidamente che la comunicazione tra il computer e Arduino sia corretta, seguire i seguenti passi:

1. Aprire, tra gli esempi, lo sketch «Blink» che fa lampeggiare il LED L sulla

scheda: menuù File (oppure pulsante > Esempi > 01.Basics > Blink; nella finestra di Editor appaiono le istruzioni che costituiscono lo sketch.

2. Premere il pulsante per compilare e caricare (upload) su Arduino lo sketch «Blink». Se la comunicazione è regolare, durante l'upload lampeggiano i LED RX e TX su Arduino.
3. Una volta finito l'uploading, se il trasferimento dello sketch si è svolto senza problemi, in basso nella finestra appare il messaggio "Caricamento completato" e inizia automaticamente l'esecuzione dello sketch: il LED L, posto vicino al pin 13 e collegato a esso, lampeggia in modo regolare.

■ Elementi principali della finestra di programmazione (IDE)

Barra del menù

Contiene i menù (a tendina) che consentono di accedere a tutte le funzioni necessarie per gestire Arduino (aprire e salvare file, modificare lo sketch, verificare la sintassi, compilare e caricare lo sketch su Arduino, selezionare il tipo di scheda e la porta USB, ecc.).

Barra verde scuro

(sotto quella dei menù) Contiene i pulsanti che rappresentano delle scorciatoie per funzioni di uso frequente:



Verifica

Verifica la correttezza della sintassi dello sketch e ne esegue la compilazione (cioè la conversione dal linguaggio **sorgente** di Arduino a quello oggetto del microcontrollore).



Carica

Trasferisce lo sketch nella scheda Arduino, dopo averlo verificato e compilato.



Nuovo

Apre una nuova finestra per la creazione di un nuovo sketch.



Apri

Apre uno sketch esistente.



Salva

Salva sul computer lo sketch attivo.



Monitor seriale

Apre una finestra mediante la quale è possibile ricevere o inviare informazioni (per esempio il valore di una variabile) al programma in esecuzione; è utile per la ricerca di errori di programmazione in fase di messa a punto dello sketch (*debugging*).

Nella parte inferiore della finestra l'IDE fornisce informazioni come gli eventuali errori di sintassi rilevati, l'esito della compilazione, l'esito del caricamento del programma, la memoria occupata dallo sketch.

1.6 La programmazione di uno sketch

Vediamo ora la struttura del codice di un programma per Arduino (sketch), prendendo come esempio lo sketch «Blink» nella figura 1.5.

Lo sketch è sostanzialmente costituito da due sezioni:

- La funzione `setup()`: viene eseguita una sola volta, all'accensione o dopo il caricamento dello sketch, oppure dopo la pressione del pulsante **Reset** di Arduino; contiene tra le parentesi graffe le istruzioni `pinMode()` per impostare lo stato (`input` o `output`) dei pin, le inizializzazioni delle librerie utilizzate, l'apertura della porta seriale per il Serial Monitor, ...
- La funzione `loop()`: contiene tra le parentesi graffe il corpo del programma che viene eseguito ciclicamente, specificando quali azioni deve compiere Arduino.

Prima della funzione `setup()` si pongono:

- le dichiarazioni delle **variabili globali**, cioè quelle variabili definite in tutto il programma e non solo all'interno di una funzione;
- le direttive `#include <nome-libreria>`, per includere le librerie utilizzate nello sketch (le librerie sono gruppi di istruzioni che semplificano l'utilizzo di alcuni dispositivi, come servomotori, display LCD, ecc.). Nell'esemplo in figura 1.5 non ci sono dichiarazioni o direttive prima di `setup()`.

Figura 1.5 Codice dello sketch «Blink».

```
/*
Blink
Accende il LED interno (collegato al pin 13) per 1 s
e poi lo spegne per 1 s, ripetutivamente.
*/

// tra le graffe della funzione setup inseriamo le istruzioni
// che devono essere eseguite solo una volta all'accensione o al reset

void setup(){
    pinMode(13, OUTPUT); // inizializza il pin 13 come uscita
}

// tra le graffe della funzione loop inseriamo il programma che sarà
// eseguito ciclicamente all'infinito, fino al prossimo reset

void loop(){
    digitalWrite(13, HIGH); // pone a livello ALTO il pin 13 (LED ON)
    delay(1000); // attende 1 s
    digitalWrite(13, LOW); // pone a livello BASSO il pin 13 (LED OFF)
    delay(1000); // attende 1 s
}
```

• I commenti: tutto ciò che viene scritto tra i simboli `/* ... */` (su più righe), oppure a destra del simbolo `//` (su una riga), è un commento del programmatore e non viene preso in considerazione dal compilatore: quindi di fatto non fa parte del programma.

È buona pratica porre un commento all'inizio dello sketch, per descriverne la funzione, e altri a fianco delle istruzioni chiave, per evidenziare la logica dell'algoritmo; questo semplificherà le future modifiche al programma, soprattutto se effettuate da programmatori diversi dall'autore originale.

• Tutte le istruzioni (tranne le direttive `#define` e `#include`) terminano con il simbolo `;` (punto e virgola).

Dal **capitolo 3** in poi si affrontano esempi e progetti in cui si descrive la sintassi delle principali istruzioni di Arduino, raccolte nell'appendice in fondo al testo.

Nella home page del sito www.arduino.cc, seguendo il percorso **Learning > Reference**, si ha il quadro completo del set di istruzioni del linguaggio di programmazione di Arduino; in alternativa, si seleziono nell'IDE: **Aiuto > Guida di riferimento**.

■ Analisi dello sketch «Blink»

La figura 1.6 ripropone lo sketch «Blink», già visto nella figura 1.5, che fa lampeggiare il LED a bordo di Arduino.



```
Il codice dello sketch «Blink»
void setup() {
    pinMode(13, OUTPUT);
}

void loop() {
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
}
```

Figura 1.6 Lo sketch «Blink» di figura 1.5 senza i commenti.

Nella funzione `setup()`, con l'istruzione `pinMode(13, OUTPUT)`, si definisce `OUTPUT` il pin 13, a cui è connesso il LED.

Si noti come l'IDE colora le parole che vengono riconosciute appartenenti al lessico del linguaggio.

Nella funzione `loop()`, le cui istruzioni tra le graffe sono eseguite ciclicamente, l'istruzione `digitalWrite(13, HIGH)` pone un livello alto (5 V) sul pin 13, provocando l'accensione del LED di bordo, collegando un LED esterno tra il pin 13 e il pin GND, anch'esso si accende.

L'istruzione `delay(1000)` impone una pausa di 1000 ms (cioè di 1 s) all'esecuzione del programma, per cui il LED rimane acceso per 1 s, prima di essere spento dall'istruzione successiva `digitalWrite(13, LOW)`.

Il risultato è che il LED viene acceso per 1 s e spento per 1 s, ciclicamente.

1.1 Esercizio

Modifica lo sketch «Blink» in modo che la frequenza di accensione del LED risulti 5 Hz.

Suggerimento Il LED deve lampeggiare 5 volte al secondo, quindi deve rimanere acceso per 100 ms e spento per 100 ms.

1.7 Costanti, variabili e il ciclo FOR

Una costante è un dato (associato a un nome) situato in una porzione di memoria (una o più locazioni di memoria) che non può essere modificato nel corso dell'esecuzione del programma.

Nella sketch in figura 1.7 le costanti `timer1` e `timer2` sono dichiarate intere (occupano 2 byte) e poste allo stesso indirizzo (200 e a 2000; quando vengono richiamate nello sketch, per esempio come parametro di `delay`, al loro nome viene sostituito il valore ad esse assegnato).

Nell'appendice *La sintassi delle principali istruzioni per Arduino* sono elencati i principali tipi di variabili e costanti (Int, Char, Long, ecc.) con la relativa occupazione in byte di memoria.

1.2 Esercizio svolto

Tre lampeggi e pausa

Ottieni una sequenza di 3 impulsi veloci seguiti da una pausa di 2 s, ripetuta ciclicamente.

Soluzione 1

Prendendo spunto dallo sketch «Blink», la prima idea che viene in mente è quella di ripetere per tre volte l'accensione e lo spegnimento del LED, intervallati da pause di 200 ms, per poi concludere il loop con una pausa di 2 s, come si può vedere nello sketch che segue.

L'uso delle costanti `timer1` e `timer2` consente di variare i tempi semplicemente modificando il valore a esse assegnato, senza correggere a una a una le istruzioni in cui esse compaiono.

Figura 1.7

Il codice dello sketch «Tre lampeggi e pausa» (senza il ciclo FOR)

```
const int timer1 = 200; // tempo di ritardo 200 ms
const int timer2 = 2000; // tempo di ritardo 2 s
```

```
void setup() {
  pinMode(13, OUTPUT); // inizializza il pin 13 come uscita
}
```



```
void loop() {
  // primo lampeggio
  digitalWrite(13, HIGH); // pone a livello ALTO il pin 13 (LED ON)
  delay(timer1); // attende 200 ms
  digitalWrite(13, LOW); // pone a livello BASSO il pin 13 (LED OFF)
  delay(timer1); // attende 200 ms

  // secondo lampeggio
  digitalWrite(13, HIGH);
  delay(timer1);
  digitalWrite(13, LOW);
  delay(timer1);

  // terzo lampeggio
  digitalWrite(13, HIGH);
  delay(timer1);
  digitalWrite(13, LOW);

  // pausa di 2 s dopo i tre lampeggi
  delay(timer2);
}
```

■ Il ciclo FOR

Il limite della soluzione precedente è che si ripete lo stesso blocco di istruzioni per tre volte; se volessimo produrre una sequenza di 100 impulsi, lo sketch diventerebbe lunghissimo.

Per ripetere n volte un blocco di istruzioni, si può utilizzare l'istruzione FOR, la cui sintassi è:

```
for (inizializzazione; condizione; incremento) {
  // sequenza di istruzioni (ciclo)
}
```

Il FOR ripete la sequenza di istruzioni tra le graffe finché la variabile di controllo (per esempio `x`), definita e inizializzata in `inizializzazione` e che a ogni ciclo viene incrementata della quantità `incremento`, soddisfa la condizione (per esempio `x <= n`).

Quando la variabile non soddisfa più la condizione si esce dal FOR e l'esecuzione del programma riprende dall'istruzione posta dopo la parentesi graffa di chiusura del ciclo.

Consideriamo come esempio lo sketch che somma i primi 100 numeri interi utilizzando il ciclo FOR. In questo sketch, `somma` e `x` sono variabili.

Una **variabile** è un dato che può essere modificato nel corso dell'esecuzione del programma; la dichiarazione di una variabile specifica il tipo (int, bool, char) senza essere preceduta da `const`.

Quindi all'uscita dal ciclo la variabile `somma` contiene la somma dei primi 100 numeri interi.

! Il ciclo FOR ripete n volte un gruppo di istruzioni.

Il codice del ciclo FOR che somma i primi 100 numeri interi



```
int somma = 0; // la variabile somma è dichiarata
                // intera e inizializzata a 0
for (int x = 1; x <= 100; x++) { // per x che va da 1 a 100,
    // a incrementi di 1 (x++)
    somma = somma + x; // al valore di somma viene
    // aggiunto il valore di x
}
```

Soluzione 2 (con il ciclo FOR)

! Nel linguaggio C++, per sommare 1 a una variabile x, si scrive `x++`, che equivale a `x=x+1`.
Per decrementare: `x--`.

Ritornando allo sketch «Tre lampaggi e pausa», vediamo come si può modificare il programma facendo uso del ciclo FOR: il blocco di istruzioni che accende e spegne il LED è scritto una sola volta, ma viene eseguito per tre volte, che corrispondono al numero di incrementi della variabile cont per arrivare da 1 a `nLamp=3`. Si noti che, modificando il valore della costante `nLamp`, si può variare il numero dei lampaggi.

Il codice dello sketch «Tre lampaggi e pausa» (con il ciclo FOR)



```
const int nLamp=3; // n° di lampaggi = 3
const int timer1 = 200; // tempo di ritardo 200 ms
const int timer2 = 2000; // tempo di ritardo 2 s

void setup() {
    pinMode(13, OUTPUT); // inizializza il pin 13 come uscita
}

void loop() {
    // Ciclo FOR: le istruzioni tra le graffe sono eseguite nLamp=3 volte,
    // cioè finché la variabile "cont", inizializzata a 1 e incrementata
    // di 1 ad ogni ciclo, risulta minore o uguale a nLamp-3

    for(int cont = 1; cont <= nLamp; cont++){
        digitalWrite(13, HIGH); // poni a livello ALTO il pin 13 (LED ON)
        delay(timer1); // attendi 200 ms
        digitalWrite(13, LOW); // poni a livello BASSO il pin 13 (LED OFF)
        delay(timer1); // attendi 200 ms
    }
    delay(timer2); // pausa di 2 s dopo i tre lampaggi
}
```

Progettare con ARDUINO

1.A Lampaggio complesso

Fai lampeggiare il LED sulla scheda Arduino, con una sequenza ciclica di 1, 2, 3 lampaggi veloci, intervallati da pause di 2 s.

Soluzione

Per contare il numero dei cicli e il numero dei lampaggi crescenti all'interno di ogni ciclo, devi impiegare due FOR uno dentro l'altro (nidificati):

- nel FOR esterno la variabile `ciclo` si incrementa da 1 a `nLampMax`, inizializzato al valore 3, quindi `ciclo` conta i 3 cicli di lampaggi;
- nel FOR interno la variabile `cont` si incrementa da 1 al valore corrente di `ciclo`, quindi conta il numero di lampaggi da effettuare in quel ciclo.

Modificando il valore di `nLampMax` (numero massimo di lampaggi) si può variare il ciclo dei lampaggi; ad esempio ponendo `nLampMax=5`, si ottiene una sequenza di 1, 2, 3, 4, 5 lampaggi.

Il codice dello sketch «Lampaggio complesso» con due FOR nidificati

```
const int nLampMax = 3; // n° massimo di lampaggi
const int timer1 = 200; // tempo tra le commutazioni degli impulsi (200 ms)
const int timer2 = 2000; // pausa tra ogni serie di impulsi (2 s)

void setup() {
    pinMode(13, OUTPUT); // inizializza il pin 13 come uscita
}

void loop() {
    /*
    Due FOR nidificati:
    - nel FOR esterno, la variabile "ciclo", variando da 1 a nLampMax=3, contiene il
    numero del ciclo, corrispondente al numero di lampaggi da effettuare in quel ciclo
    - nel FOR interno, la variabile "cont" conta i lampaggi (crescenti da 1 a nLampMax=3)
    all'interno del ciclo
    - al termine di ogni ciclo di lampaggi (1, 2, 3) c'è una pausa di 2 s
    */

    for (int ciclo = 1; ciclo <= nLampMax ; ciclo++) { // FOR esterno
        for (int cont = 1; cont <= ciclo ; cont++) { // FOR interno
            digitalWrite(13, HIGH); // poni a livello ALTO il pin 13 (LED ON)
            delay(timer1); // attendi 200 ms
            digitalWrite(13, LOW); // poni a livello BASSO il pin 13 (LED OFF)
            delay(timer1); // attendi 200 ms
        }
        delay(timer2); // pausa di 2 s al termine di ogni ciclo
    }
}
```

1.3 Esercizio**Codice Morse**

Pilota il LED L sulla scheda Arduino Uno, generando ciclicamente la sequenza del codice Morse corrispondente alla richiesta di soccorso SOS:

.....

dove:

S: tre punti (tre impulsi luminosi brevi, scegliere ad esempio $t_p = 200$ ms);

O: tre linee (tre impulsi luminosi lunghi il triplo del punto, $t_l = 600$ ms).

Suggerimento Lo spazio tra un simbolo e l'altro (punto o linea) dura come il punto

$$t_s = t_p = 200 \text{ ms}$$

mentre quello tra una lettera e l'altra dura come la linea ($t_{lw} = 600$ ms).

Quando avrai appreso l'uso del buzzer e delle relative istruzioni ([capitolo 3, paragrafo 3](#)), lo potrai collegare a un pin digitale per udire il segnale Morse.

**Capitolo****2**

I componenti elettronici

Per lavorare con Arduino servono componenti elettronici aggiuntivi, come trasduttori e attuatori per interagire con l'ambiente circostante (Physical Computing).

La maggior parte dei componenti elettronici presentati in questo capitolo sono contenuti nell'[Arduino Starter Kit](#): non approfondiremo la teoria che sta alla base del loro funzionamento, ma spiegheremo le loro funzioni per poterli utilizzare con consapevolezza negli esempi e nei progetti dei prossimi capitoli.

2.1 Breadboard e jumper

Normalmente i componenti elettronici che realizzano un circuito sono connessi tra loro mediante le piste di un circuito stampato; la scheda madre di un computer ne è un esempio.

La realizzazione di un circuito stampato è piuttosto complessa e non economica. Per costruire prototipi è molto più pratico e conveniente l'uso della **breadboard**, che consente di connettere i terminali dei componenti e i fili di collegamento, detti **jumper** (ponticelli), semplicemente incastrandoli nei fori, senza necessità di saldature.

I fori della breadboard sono uniti internamente da collegamenti metallici con la seguente logica ([figura 2.1](#)):

- tutti i fori disposti su ogni linea orizzontale (le due linee in alto e le due linee in basso) sono collegati insieme internamente; in genere queste linee servono a distribuire la tensione di alimentazione e la massa (V_{cc} e GND) a tutte le parti del circuito.
- I fori disposti su ogni linea verticale sono collegati internamente a gruppi di 5 adiacenti.

Di conseguenza, sia i fili rossi sia i fili verdi della [figura 2.1](#) hanno i due estremi connessi tra loro grazie ai contatti della breadboard.

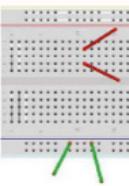


Figura 2.1 Ogni linea nera attraversa fori che sono connessi tra loro. I fili rossi hanno dunque i due estremi connessi tra loro, così come i fili verdi.

La **figura 2.2** mostra l'impiego della breadboard per collegare un LED e un resistore in serie (cioè in modo che la corrente li attraversi entrambi), tra il pin 13 di Arduino e il pin GND.

La **figura 2.2a**, come molte rappresentazioni di circuiti su breadboard nelle prossime pagine, è stata realizzata con il software *Fritzing*, scaricabile gratuitamente dal sito <http://fritzing.org/download/>.

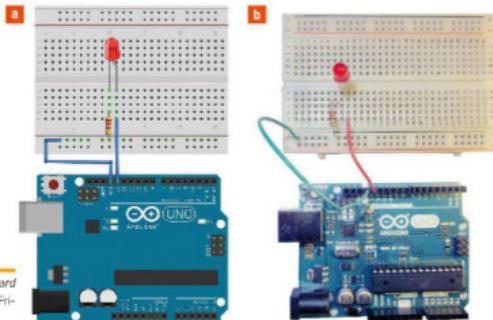


Figura 2.2 Uso della breadboard con Arduino: a) simulazione con Fritzing; b) circuito reale.

2.2 Componenti elettronici di uso frequente

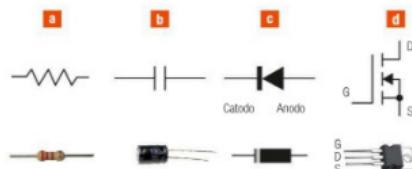


Figura 2.3 Simboli e contenitori di: a) resistore; b) condensatore; c) diodo al silicio; d) transistor MOSFET IRF520.

Il resistore

Il resistore (**figura 2.3a**) è un componente a due terminali (bipolo) che presenta una resistenza al passaggio della corrente elettrica. In genere utilizziamo un resistore in serie a un LED, per limitarne la corrente (come in **figura 2.2**), oppure in serie a un interruttore per realizzare il *pull-up*, come vedremo nel **paragrafo 3**.

Il parametro caratteristico di un resistore è la resistenza R , espressa in

ohm (Ω), che rappresenta il valore del rapporto tra la tensione applicata ai capi e la corrente che scorre: $R = V/I$ (legge di Ohm).

Il valore del resistore è espresso mediante un codice colore: due valori che utilizzeremo di frequente sono 220 Ω (rosso, rosso, marrone), in serie ad LED, e 10 k Ω (marrone, nero, arancio), per il *pull-up* degli interruttori. La fascia color oro indica una tolleranza del 5%.

Il condensatore

Il condensatore (**figura 2.3b**) è un bipolo che ha la funzione di accumulare carica elettrica; viene impiegato per esempio per eliminare le oscillazioni spurious di una tensione continua (*filtro*), o per realizzare oscillatori. Il suo parametro caratteristico è la capacità, che si misura in farad (F). I condensatori ceramici e poliestere, di piccola capacità (dell'ordine dei pF o dei nF), non sono polarizzati, mentre quelli elettrolitici (di capacità superiore a 1 μF) devono essere collegati correttamente: con il terminale **-** (meno) posto al potenziale inferiore.

Il diodo al silicio

Il diodo al silicio (**figura 2.3c**) è un bipolo asimmetrico: il terminale in corrispondenza della fascetta è detto catodo, l'altro anodo. Ha la proprietà di far scorrere la corrente solo in un senso: dall'anodo al catodo.

È utilizzato per esempio per realizzare il raddrizzatore (blocco fondamentale dell'alimentatore, che ricava la tensione continua dall'alternata) oppure è posto in parallelo alle bobine dei motori e dei relè, per proteggere i transistor durante le commutazioni.

Il transistor MOSFET

Il transistor MOSFET (**figura 2.3d**) è un componente al silicio con tre terminali: G (Gate), D (Drain) e S (Source). Per i nostri scopi, è assimilabile a un interruttore

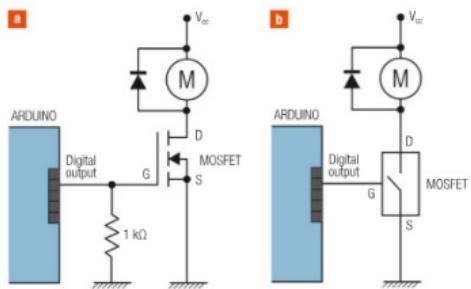


Figura 2.4 Azionamento di potenza per un motore con MOSFET: a) schema elettrico; b) MOSFET schematizzato come un interruttore comandato.

comandato elettricamente (**Figura 2.3d**): fornendo tensione sul *Gate*, la corrente può scorrere tra il *Drain* e il *Source*. La piedinatura di **Figura 2.3d** si riferisce al MOSFET **IRF520**, in dotazione allo Starter Kit di Arduino.

Il MOSFET è utilizzato come amplificatore negli azionamenti di potenza, quando il carico richiede una corrente troppo elevata per l'uscita del circuito di comando. Nello schema in **Figura 2.4a** si vede un'uscita digitale di Arduino che pilota un motore mediante un MOSFET che funge da interfaccia; il motore necessita di una corrente (qualche centinaio di mA) più elevata di quella che può fornire l'uscita di Arduino (20 mA; sopra i 40 mA si danneggia).

Il MOSFET richiede da Arduino una corrente bassa sul *Gate*, ma può invece pilotare la corrente elevata del motore tra *Drain* e *Source*; nella **Figura 2.4b** il MOSFET è schematizzato come un interruttore comandato dall'uscita di Arduino.

Si noti il diodo di ricircolo in parallelo al motore (bobina): quando il MOSFET è ON (interruttore chiuso) la corrente passa per il motore e non per il diodo; nella commutazione ON → OFF, che produrrebbe una sovratensione ai capi della bobina danneggiando il transistor, il diodo crea un percorso ad anello per la corrente, che dissipata in modo «dolce» l'energia accumulata dalla bobina del motore.

2.3 I trasduttori e i circuiti di condizionamento

Il trasduttore (detto anche sensore) è un dispositivo che converte una grandezza fisica in una elettrica, che viene poi acquisita da un sistema elettronico; se la grandezza elettrica non è quella accettata dal sistema di acquisizione (Arduino accetta tensioni comprese tra 0 V e 5 V) è necessario interporre un circuito di condizionamento (**Figura 2.5**).

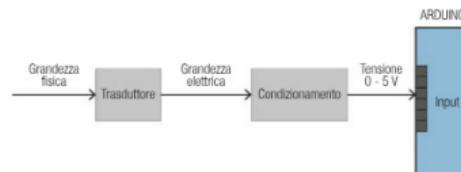


Figura 2.5 Trasduttore e condizionamento per l'acquisizione di una grandezza fisica.

In commercio sono disponibili trasduttori per qualunque grandezza fisica, in alcuni casi con il circuito di condizionamento già integrato.

L'uscita del trasduttore può essere un segnale:

- **digitale:** la grandezza elettrica ha solo due valori significativi (in genere associati ai simboli 0 e 1), corrispondenti a due campi di valori della grandezza fisica;
- **analogo:** ad ogni valore della grandezza fisica corrisponde un valore significativo di quella elettrica, spesso la relazione è di proporzionalità.

Trasduttori digitali ON-OFF

I trasduttori digitali ON-OFF associano ai due stati significativi della grandezza fisica l'apertura e la chiusura di un contatto.

Pulsante (**Figura 2.6a**) e finecorsa (**Figura 2.6b**).

Il pulsante è utilizzato come dispositivo di INPUT (per inserire comandi da parte dell'utente), quindi a rigore non dovrebbe essere incluso tra i trasduttori, ma l'impiego circitale è identico.

Il finecorsa è analogo a un pulsante, ma è realizzato in modo da essere premuto da un oggetto in movimento quando giunge al termine della sua corsa. I pulsanti che usiamo nel testo sono del tipo a contatti normalmente aperti (NO, *Normally Open*): premendo il pulsante si chiude il contatto.

Si noti che il pulsante in **Figura 2.6a** ha quattro pin: l'interruzione del contatto avviene tra i pin sullo stesso lato, che sono connessi a due a due con quelli sul lato opposto.

▪ **Sensore di tilt (sensore di inclinazione)** (**Figura 2.6c**): se posto in verticale, il contatto è chiuso, mentre oltre una certa inclinazione il contatto si apre. In pratica è costituito da un tubo contenente una pallina metallica che, in posizione verticale, chiude il contatto tra i terminali. Il sensore **RBS040100**, contenuto nello Starter Kit, presenta 4 pin: i due allineati con la scanalatura sono collegati insieme, così come quelli opposti.

▪ **Ampolla Reed** (**Figura 2.6d**): è un sensore di campo magnetico costituito da due lamelle affacciate; avvicinando un magnete, al di sotto di una certa distanza il contatto tra i terminali si chiude.



Condizionamento dei trasduttori ON-OFF

Il condizionamento per tutti i trasduttori ON-OFF, assimilabili a un contatto che si apre e si chiude, avviene ponendo il contatto in serie a un resistore R (pull-up) tra massa e +5 V (**Figura 2.7**).

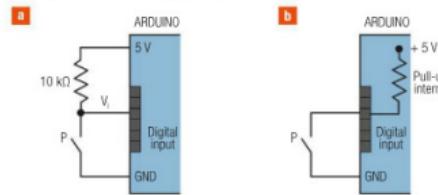


Figura 2.7 Condizionamento di un trasduttore ON-OFF, con resistore R di pull-up **a)** esterno; **b)** interno.

La tensione V_o , prelevata tra resistore e contatto, viene portata a un pin digital input di Arduino:

- se il contatto è aperto: V_o è (HIGH, 1 logico) a causa del resistore R (**pull-up**) collegato a 5 V;
- se il contatto è chiuso: V_o è BASSA (LOW, 0 logico) perché il pin d'ingresso è connesso a GND; il resistore da 10 kΩ limita la corrente tra V_{cc} e GND al valore $I = V_{cc}/R = 0,5$ mA.

Arduino, in particolare il microcontrollore al suo interno, semplifica il condizionamento, in quanto mette a disposizione un **pull-up** interno attivabile a scelta: per esempio, se vogliamo attivare il resistore di pull-up interno definendo il digital pin come INPUT_PULLUP nell'interno della funzione `setup()` scriviamo l'istruzione `pinMode(10, INPUT_PULLUP)`. In questo modo il resistore R di pull-up esterno non è più necessario, e possiamo collegare direttamente il pulsante (o il trasduttore) tra il pin d'ingresso e GND (figura 2.7b).

Trasduttori analogici

I trasduttori analogici sono trasduttori in cui la grandezza elettrica d'uscita segue con continuità le variazioni della grandezza fisica in ingresso.

Il circuito di condizionamento per questi trasduttori ha il compito di convertire la grandezza elettrica in tensione (se non lo è già in origine), facendola variare in un range prefissato, che per Arduino è compreso tra 0 V e 5 V.

- Il potenziometro (figura 2.8a) normalmente viene usato come dispositivo di regolazione analogica (per esempio per modificare il volume di un amplificatore), ma può essere pensato anche come **trasduttore di posizione angolare**. Presenta tre terminali e un alberino rotante; se non ha l'alberino, il componente è detto trimmer (figura 2.8b) e va regolato con un cacciavite.

Tra i due terminali esterni si misura una resistenza R fissa, il cui valore è stampato sul contenitore; il trimmer contenuto nello Starter Kit è da 10 kΩ. Il terminale centrale è collegato a un cursore, mosso dall'alberino, che suddivide la resistenza R in due parti, R_s e R_o (variabili con la posizione dell'alberino rotante) la cui somma vale $R = R_s + R_o$. Di conseguenza esiste una relazione tra la posizione angolare dell'alberino e i valori di R_s e R_o ; tale relazione può essere lineare (come nel caso del potenziometro dello Starter Kit) o logaritmica (per i potenziometri destinati ad applicazioni audio). I due modi più comuni per usare un potenziometro sono schematizzati nella figura 2.9:

1. collegandosi tra il terminale centrale e uno dei due esterni, si ottiene un resistore variabile, in questo caso da 0 Ω a 10 kΩ (figura 2.9a);

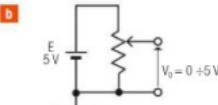
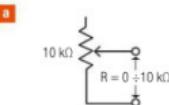


Figura 2.8 a) Potenziometro; b) trimmer; c) simbolo elettronico.

Figura 2.9 Impieghi del potenziometro: a) resistore variabile; b) partitore di tensione variabile.

2. applicando una tensione continua (per esempio 5 V) tra i terminali esterni si ottiene, tra il centrale e un esterno, una tensione d'uscita V_o variabile da 0 V a 5 V (figura 2.9b). Questa tensione può essere applicata a un pin d'ingresso analogico di Arduino e acquisita con l'istruzione `analogRead(pin)`, come si vedrà nel capitolo 5.

- Il fotoresistore (figura 2.10a) è un trasduttore di luminosità, la cui resistenza tra i terminali diminuisce al crescere dell'intensità luminosa incidente (circa 1 MΩ al buio, circa 10 kΩ alla luce). Il suo condizionamento (conversione resistenza → tensione) può essere realizzato come in figura 2.10b: all'aumentare della luminosità, la resistenza R_o diminuisce e, di conseguenza, la tensione V_o aumenta; la relazione tra luminosità e V_o non è lineare.

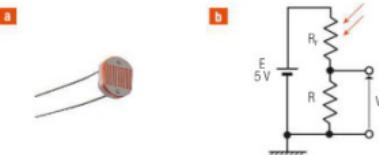


Figura 2.10 a) Fotoresistore;
b) circuito di condizionamento.

- Il trasduttore di temperatura TMP36 (figura 2.11a) converte la temperatura compresa in un range da -40°C a +125°C in una tensione (che cresce di 10 mV per ogni grado centigrado) compresa nel range tra 0,1 V e 1,75 V. Tra i pin +V_{cc} e GND si fornisce un'alimentazione compresa tra +2,7 V e 5,5 V (per esempio i 5 V forniti da Arduino), mentre la tensione di uscita si preleva tra V_{out} e GND.

La relazione tra la temperatura T e tensione V_{out} è data dalla formula:

$$T=100 \cdot (V_{out} - 0,5)$$

Il TMP36 non necessita di alcuna conversione, poiché la grandezza di uscita è già una tensione. Per sfruttare al meglio la risoluzione di Arduino dobbiamo però adattare il campo di variabilità del segnale V_{out} (da 0,1 V a 1,75 V) con quello accettato da Arduino, che per default è compreso tra 0 V e 5 V; potremmo amplificare il segnale, ma è più semplice modificare il valore di fondo scalare di Arduino, ponendo tale tensione sul pin AREF e inserendo nella funzione `setup()` l'istruzione `analogReference(EXTERNAL)`, come vedremo nel capitolo 5.

- Il modulo misuratore di distanza a ultrasuoni (figura 2.11b) consente di calcolare la distanza di un oggetto che vi è posto di fronte, misurando il tempo di andata e ritorno di un impulso acustico a frequenza non udibile.



Figura 2.11 a) Transduttore di temperatura TMP36; b) modulo misuratore di distanza a ultrasuoni HC-SR04.

2.4 Gli attuatori e gli azionamenti di potenza

Gli attuatori sono quei dispositivi che, pilotati dalle uscite digitali di Arduino, agiscono sull'ambiente circostante: visualizzano informazioni (LED e display), provocano spostamenti (motori e servomotori), generano calore (riscaldatore resistivo), emettono suoni (buzzer e altoparlanti) o luce (lampe).

Dovendo cedere una certa potenza all'ambiente, l'attuatore assorbe una corrente che Arduino potrebbe non essere in grado di fornirgli; in questo caso è necessario interporre un **azionamento di potenza**, in genere costituito da un amplificatore a transistor o a circuito integrato, per aumentare la corrente e la tensione (figura 2.12).



Figura 2.12 Schema per il pilottaggio di un attuatore.

Alcuni degli attuatori presenti nello Starter Kit, come i LED, il servomotore e il *buzzer*, sono modelli di piccola potenza, quindi non necessitano del circuito di azionamento e possono essere collegati direttamente alle uscite di Arduino.

PWM: regolazione della potenza di un attuatore

Per comandare un attuatore, come un LED, una lampada o un motore, con solo due stati, acceso e spento, possiamo collegarlo a un'uscita digitale di Arduino: un valore ALTO su un dato pin, ottenuto con l'istruzione `digitalWrite(pin, HIGH)`, provocherà l'accensione del LED collegato a quel pin.

Se invece vogliamo variare la luminosità di un LED o di una lampada o la velocità di un motore, dobbiamo usare la tecnica PWM (Pulse Width Modulator, Modulazione di Larghezza d'impulso): collegiamo un LED a un'uscita digitale di Arduino. Tali uscite, con il numero preceduto dal simbolo `v` e cioè 3, 5, 6, 9, 10, 11, possono produrre un segnale a impulsi (490 impulsi al secondo) di cui si può variare il valore medio, agendo sul rapporto tra il tempo dello stato ALTO e il tempo complessivo dell'impulso (ALTO + BASSO); tale rapporto è detto **Duty Cycle**.

Con l'istruzione `analogWrite(pin, valore)`, variando il parametro `valore` da 0 a 255 si modifica il Duty Cycle della tensione in uscita sul pin, da 0% (valore medio nullo) a 100% (valore medio massimo = 5 V), come schematizzato nella figura 2.13. In questo modo la luminosità del LED varia tra zero e il valore massimo.

L'istruzione `analogWrite` è così chiamata nonostante il segnale sia prelevato da un pin **digital**: questo perché, anche se la tensione varia solo tra due livelli (0 V e 5 V), l'informazione associata al segnale, in particolare al valore del Duty Cycle, è di tipo analogico.

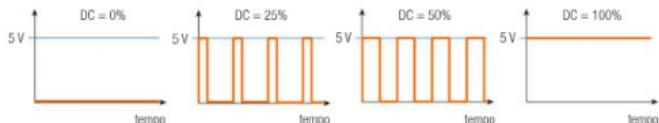


Figura 2.13 Segnale PWM per diversi valori del Duty Cycle.

I LED

Il diodo LED (Light Emitting Diode) (figura 2.14a) è utilizzato per visualizzare lo stato di una variabile binaria. Esso emette luce quando è attraversato da una corrente compresa tra 10 mA e 20 mA diretta da **anodo** (il terminale più lungo) a catodo; la tensione ai capi vale circa 2 V. Il colore della luce dipende dal semiconduttore con cui il diodo è realizzato.

Il LED RGB (Red, Green, Blue) è composto da tre LED dei colori fondamentali (rosso, verde, blu), posti in un unico contenitore. Dosando l'intensità dei tre colori fondamentali, mediante la tecnica PWM (descritta al punto precedente) si possono ottenere tutti gli altri colori. Come si vede nella figura 2.14b, i catodi dei tre diodi sono collegati insieme e vanno connessi al terminale GND di Arduino (esiste anche la versione ad anodo comune).

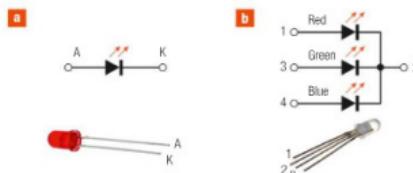


Figura 2.14 a) LED: simbolo e componente; b) LED RGB a catodo comune: collegamenti interni e componente.

I collegamenti ad Arduino del LED e del LED RGB sono rappresentati nella figura 2.15: ogni diodo ha in serie un resistore (in genere 220 Ω) che limita la corrente a un valore compreso tra 10 mA e 20 mA. Si noti che i catodi sono sempre collegati verso massa (GND).

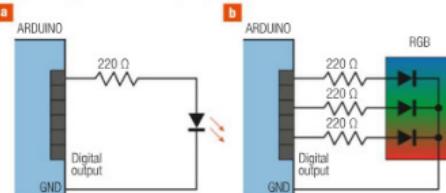


Figura 2.15 Collegamento ad Arduino: a) di un LED; b) di un LED RGB a catodo comune.



Figura 2.16 Display LCD Hitachi HD44780 (2x16).



Figura 2.17 Buzzer.

■ Il display LCD

Il display a cristalli liquidi LCD (Liquid Crystal Display) visualizza caratteri alfabetici (lettere e numeri) per consentire ad Arduino di comunicare all'esterno, per esempio, messaggi di testo o valori di variabili.

Il display LCD HD44780 (figura 2.16), contenuto nello Starter Kit di Arduino, ha 16 pin ed ha i caratteri organizzati in 2 righe e 16 colonne.

■ Il buzzer

Il **buzzer** (figura 2.17) è un piccolo altoparlante, realizzato con un cristallo che si deforma quando gli viene applicata una tensione (effetto piezoelettrico); affinché la deformazione produca una nota udibile deve essere pilotato con tensioni a onda quadra di frequenza corrispondente a quella della nota emessa. Data la ridotta corrente assorbita, non necessita di un circuito di azionamento e può essere pilotato direttamente da un'uscita digitale di Arduino.

Per generare una nota si usa la funzione `tone(pin, frequenza, durata)`, dove i parametri sono: il numero del pin d'uscita, la frequenza della nota, la durata in millisecondi (facoltativo); con `noTone(pin)` la nota si interrompe.

■ I motori

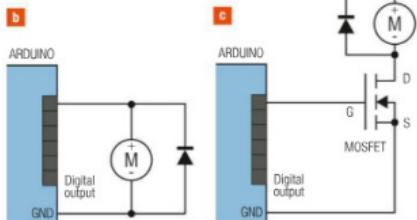
I motori sono dispositivi basati sull'interazione tra i campi magnetici prodotti da bobine percorse da corrente e i magneti permanenti; vengono utilizzati nell'automazione per produrre spostamenti. Tra i più comuni si descrivono il **motore in continua**, il **servomotore** (presenti nello Starter Kit) e il **motore passo-passo**.

1. Il motore in continua (figura 2.18a)

È un motore elettrico alimentato con una tensione continua, dal cui valore dipende la velocità di rotazione; invertendo la polarità della tensione si invierte il senso di rotazione.



Figura 2.18 a) Motore in continua; **b)** pilotaggio diretto con Arduino (potenza molto piccola); **c)** pilotaggio tramite MOSFET (potenza medio-alta).



► Pilotaggio ON-OFF

Un'uscita digitale di Arduino può pilotare direttamente un motore da 5 V di

potenza molto piccola (corrente max. 20 mA) (figura 2.18b); per potenze superiori, come nel caso del motore dello Starter Kit, è necessario interporre un circuito di potenza a MOSFET (figura 2.18c).

Possiamo notare la presenza del diodo di ricircolo, che protegge il circuito di comando dalle sovratensioni prodotte dalla bobina del motore.

Con l'istruzione `digitalWrite(pin, val)` il motore, collegato al pin inizializzato come **Output**, si avvia con `val=HIGH` e si arresta con `val=LOW`.

► Variazione della velocità con la tecnica PWM

Per variare la velocità di un motore in continua dobbiamo collegarlo a un **pin digitale** abilitato all'uscita PWM (quelli contrassegnati con \sim cioè 3, 5, 6, 9, 10, 11) inizializzate come **Output**. Lo schema è quello di figura 2.18b o di figura 2.18c, in base alla potenza del motore.

Come già detto, con l'istruzione `analogWrite(pin, valore)` modifichiamo il Duty Cycle del segnale d'uscita e quindi anche la velocità del motore, da zero (`valore=0`) al massimo (`valore=255`).

► Inversione del verso di rotazione col ponte H

Per invertire il verso di rotazione del motore dobbiamo invertire la polarità della sua tensione di alimentazione; ciò si realizza con un circuito a 4 MOSFET, detto **ponte H** a causa della sua forma (figura 2.19), schematizzabile come 4 interruttori comandati dai segnali $S_1 \sim S_4$. I quattro diodi di protezione i MOSFET, consentono il ricircolo della corrente quando i transistor passano da ON a OFF.

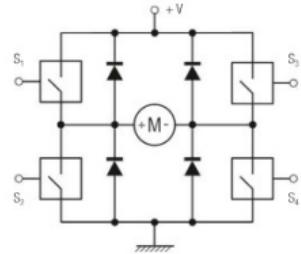


Figura 2.19 Ponte H per l'inversione del verso di rotazione del motore.

Il funzionamento del ponte H è il seguente:

- chiudendo gli interruttori S_1 e S_4 il motore ruota in un verso;
- chiudendo gli interruttori S_2 e S_3 la tensione sul motore s'inverte e quindi il motore ruota nel verso opposto;
- apendo tutti gli interruttori (oppure le coppie $S_1 \sim S_2$ o $S_3 \sim S_4$) il motore è fermo.

Bisogna evitare la chiusura contemporanea delle coppie $S_1 \sim S_2$ e $S_3 \sim S_4$, perché ciò comporterebbe un cortocircuito tra $+V$ e GND.

L'integrato L293D, il cui pin-out è in [figura 2.20a](#), contiene due ponti H e i relativi diodi di ricircolo: il primo ponte si realizza come in [figura 2.20b](#), il secondo in modo analogo collegandosi ai pin 9, 10, 11, 14, 15. I pin 4, 5, 12, 13, connessi insieme internamente, sono la massa del circuito (GND).

I pin di alimentazione sono il pin 16 e il pin 8:

- al pin 16 (V_{cc}) viene fornita una tensione di 5 V direttamente dai pin di Arduino;
- al pin 8 (V_s) può essere fornita una tensione esterna (per esempio da una batteria) da 4,5 V a 36 V, che verrà utilizzata dal motore.

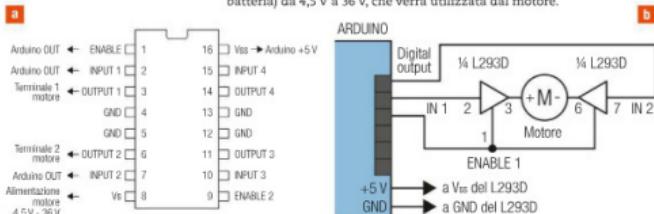


Figura 2.20 a) Pin-out dell'integrato L293D (due ponti H); b) pilotaggio di un ponte H del L293D con Arduino.

Il funzionamento dell'integrato L293D è il seguente ([figura 2.20b](#)):

- con $ENABLE1=LOW$ i pin 3 e 6 sono in alta impedenza (in pratica tutti gli interruttori del ponte H sono aperti) e il motore è fermo;
- con $ENABLE1=HIGH$, $IN1=HIGH$ e $IN2=LOW$, il motore gira in un verso;
- con $ENABLE1=HIGH$, $IN1=LOW$ e $IN2=HIGH$, il motore gira nel verso opposto.

2. Il servomotore

Il servomotore ([figura 2.21a](#)) (detto anche **servo**) è un motore elettrico in grado di ruotare l'albero di un angolo compreso tra 0° e 180°, rimanendo fermo nella posizione raggiunta in modo stabile; il segnale di controllo è di tipo PWM.

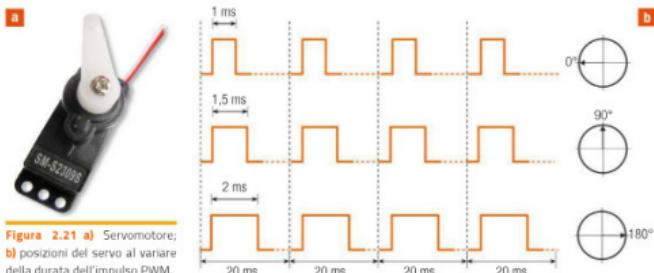


Figura 2.21 a) Servomotore; b) posizioni del servo al variare della durata dell'impulso PWM.

Nei servomotori la modulazione a larghezza d'impulso (PWM) è applicata nel modo seguente ([figura 2.21b](#)):

- la frequenza degli impulsi è sempre 490 Hz, con un periodo di circa 20 ms;
- l'angolo del servo dipende dalla larghezza dell'impulso a livello ALTO, che varia da 1 ms a 2 ms: 1 ms corrisponde a 0°, 1,5 ms a 90°, 2 ms a 180°.

Per comandare i servomotori in modo semplice è stata creata la libreria **servo.h**, le cui istruzioni consentono di assegnare l'angolo del servo senza preoccuparsi della durata degli impulsi corrispondente (come vedremo nel [capitolo 5](#)).

I fili che pilotano il servo compreso nello Starter Kit hanno le seguenti funzioni:

- rosso: alimentazione (+5 V fornito da Arduino)
- bianco: controllo della posizione (uscita PWM di Arduino)
- nero: massa (GND)

Un servomotore di piccola potenza può essere collegato direttamente a un pin digitale di Arduino, tra quelli abilitati all'uscita PWM (contrassegnati con ~: 3, 5, 6, 9, 10, 11) ([figura 2.22](#)).

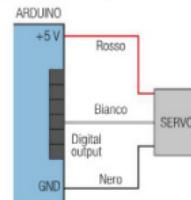


Figura 2.22 Collegamento di un servomotore ad Arduino.

3. Il motore passo-passo

Questo motore ha la caratteristica di poter essere comandato a impulsi, tramite un'opportuna interfaccia di potenza (per esempio l'integrato L293D) e per ogni impulso compie un passo di rotazione; per esempio, sono comuni i motori che suddividono un giro completo in 200 passi. È quindi sufficiente fornire un certo numero di impulsi e la direzione dello spostamento, per portare il motore in una posizione precisa, che viene mantenuta stabilmente; di conseguenza non serve un trasduttore per rivelare la posizione raggiunta dall'albero motore.

Questo motore è utilizzato nei casi in cui è richiesta una buona precisione nel movimento, per esempio nel posizionamento del carrello delle stampanti. Useremo il motore passo-passo nel progetto «Clown giocoliere» del [capitolo 5](#).

■ Relè ed elettromagneti

Il relè è un dispositivo elettromeccanico che contiene un elettromagnete (cioè una bobina avvolta su nucleo di ferro) che, se alimentato, fa commutare uno o più contatti mediante un sistema di leve. Per esempio, nel relè in [figura 2.23a](#), in assenza di alimentazione della bobina i contatti del deviatore sono aperto a sinistra (NA) e chiuso a destra (NC); alimentando la bobina, la situazione si inverte.

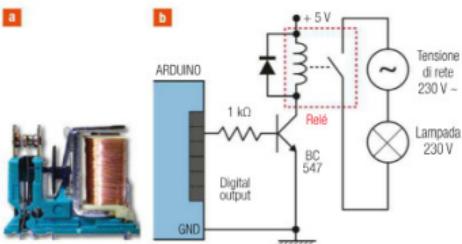


Figura 2.23 a) Relé; b) pilotaggio di una lampada a 230 V mediante transistor BJT e relé con bobina da 5 V.

La bobina di un relè in genere assorbe una corrente > 20 mA (corrente massima di una *digital output* di Arduino); quindi, anche se la tensione di funzionamento della bobina fosse 5 V, essa non potrebbe essere collegata direttamente all'uscita. È dunque necessaria un'interfaccia a transistor, come il MOSFET in dotazione allo Starter Kit o un transistor BJT come il BC547 o il 2N2222 (**figura 2.23b**).

Ora Arduino, attraverso i contatti del relè, può pilotare un carico con tensione e corrente elevate, in continua o alternata, come una lampada a 230 V alternati; i valori limiti di corrente e tensione tra i contatti sono stampati sull'involucro del relè.

Come al solito, per proteggere il transistor durante la commutazione ON→OFF dobbiamo inserire un diodo di ricircolo in parallelo alla bobina.

2.5 Le shield

Esistono diversi moduli di espansione per Arduino, detti **shield**, che svolgono varie funzioni e semplificano il collegamento verso dispositivi esterni, come trasduttori, attuatori, rete WiFi, dispositivi USB, rete cellulare GSM.

Le **shield** sono schede che si sovrappongono ad Arduino, sfruttando i connettori sui connettori e replicandoli nella parte superiore, per renderli disponibili a ulteriori impieghi. Nella **figura 2.24** possiamo vedere:



Figura 2.24 Shield per Arduino:
a) Robot shield; b) Relay shield.

- **Robot shield**

Le **robot shield** sono schede dotate di sensori a ultrasuoni, che utilizzano per evitare ostacoli. Possono pilotare motori e servomotori per creare robot e mezzi in grado di muoversi con una certa autonomia.

- **Relay shield**

Le **relay shield** sono in grado di pilotare 4 relè, per collegare Arduino con attuatori che necessitano di elevate tensioni e correnti.

Progettare con ARDUINO

2.A Controllo del movimento di un carrello

Utilizzando Arduino e i dispositivi studiati in questo capitolo, progetta la parte hardware di un sistema di automazione che sposti un carrello da un estremo all'altro della sua corsa lineare, con la logica seguente (**figura 2.25**):

1. alla pressione di un pulsante P il carrello si sposta dall'estremo A della sua corsa e raggiunge l'altro estremo B;
2. una volta che il carrello è arrivato in B, si accende un LED e un segnale digitale S_{tx} viene inviato a una macchina, che inizia a compiere delle lavorazioni sull'oggetto posto sul carrello;
3. finite le lavorazioni, la macchina invia un segnale S_{rx} , in corrispondenza del quale il carrello torna verso l'estremo A;
4. una volta che il carrello è giunto in A, viene emesso un suono. Poi il ciclo ricomincia dal punto 1.

Soluzione

La struttura meccanica potrebbe essere quella schematizzata nella **figura 2.25**:

1. il moto rotatorio del motore diventa lineare grazie a una puleggia e a una cinghia collegata al carrello;
2. agli estremi A e B della corsa ci sono due finecorsa (FcA e FcB) per informare Arduino che il carrello è arrivato e quindi deve fermare il motore; si possono usare anche due ampolle Reed che sentono l'avvicinarsi di un magnete montato sul carrello.

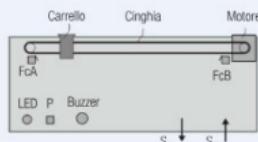


Figura 2.25 Automazione con Arduino: struttura meccanica.

Lo schema elettrico (**figura 2.26**) evidenzia i dispositivi e i segnali d'ingresso a sinistra (il pulsante, i due finecorsa e il segnale S_{rx}) e quelli d'uscita a destra (il LED, il buzzer, il segnale S_{tx} e il motore interfacciato con l'integrato L293D per il pilotaggio a quattro quadranti).

I contatti P, FcA e FcB sono privi di resistore di pull-up; nello sketch si attiverà il pull-up interno.

Nello schema di montaggio di **figura 2.27**:

1. per provare il prototipo su breadboard possiamo usare due pulsanti FcA e FcB al posto dei finecorsa;
2. i segnali S_{rx} e S_{tx} sono stati portati a un connettore a due pin;
3. un altro connettore a due pin è utilizzato per collegare un alimentatore esterno in modo da fornire al motore, tramite l'integrato L293D, una tensione di alimentazione e una corrente superiori a quelle che è in grado di fornire Arduino (batteria da 9 V per il motore dello Starter Kit);

Per quanto riguarda il software, nello sketch i digital pin 6, 11, 12, 13 dovranno essere definiti come ingressi (con pull-up interno, tranne il 6), mentre i pin 2, 3, 4, 5, 7, 10 dovranno essere definiti come uscite.

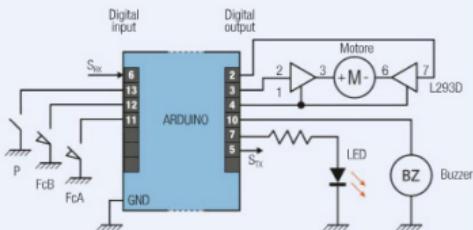


Figura 2.26 Controllo del movimento di un carrello: schema elettronico.

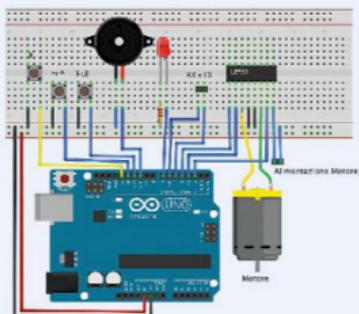
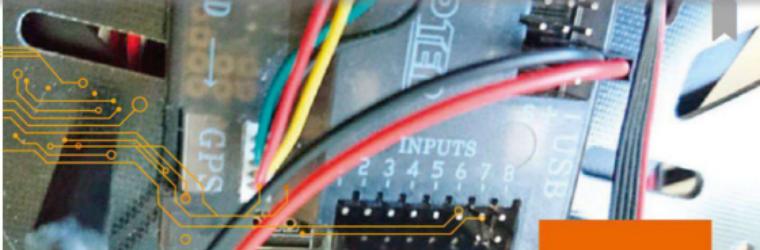


Figura 2.27 Controllo del movimento di un carrello: schema di montaggio.



Capitolo

3

Input e output digitali

3.1 Le istruzioni per input e output digitali

Studiamo ora la sintassi delle istruzioni che consentono la lettura di un bit da un ingresso digitale (per esempio lo stato di un pulsante) e la scrittura di un bit su un'uscita digitale (per esempio per pilotare un LED).

Negli sketch dei vari esempi e progetti del capitolo useremo varie istruzioni del linguaggio Wiring (C/C++); l'appendice *La sintassi di Arduino* raccoglie una sintesi di tali istruzioni, mentre il set di istruzioni completo è consultabile sul sito internet di Arduino.

Arduino Uno presenta 14 pin di ingresso/uscita digitali (figura 3.1a).



La definizione del tipo di pin (ingresso o uscita) si effettua nella funzione `setup()`:

```
pinMode(pin, OUTPUT); // pin definito come uscita  
pinMode(pin, INPUT); // pin definito come ingresso
```

A un ingresso digitale si può collegare:

- un segnale digitale proveniente, per esempio, da una porta logica;

Figura 3.1 a) I 14 pin digitali di Arduino Uno; b) Pull-up interno attivato su un ingresso digitale.

- b. un pulsante o un dispositivo, come l'ampolla Reed o il sensore di Tilt, che apre e chiude un contatto.

Ricorda che per attivare il resistore di pull-up interno in un ingresso a cui si è collegato un pulsante ([figura 3.1b](#)) si deve scrivere:

`pinMode(pin, INPUT_PULLUP)`

In questo modo quando l'interruttore P è aperto l'ingresso è HIGH (1 logico), mentre quando P è chiuso l'ingresso è LOW (0 logico).

A un'uscita digitale si possono collegare attuatori ([capitolo 2](#)) con due stati di funzionamento (acceso/spento) nei seguenti modi:

- collegamento diretto (tensione 5 V e corrente < 20 mA): LED, Buzzer, motori in continua e servomotori di piccola potenza;
- collegamento tramite interfaccia a transistor o circuito integrato: motori, servomotori, lampade, ecc. alimentati in continua, anche con tensione diversa da 5 V;
- con interfaccia a transistor + relè: attuatori potenti alimentati in tensione continua o alternata.

Ricorda che 6 digital pin di Arduino Uno (3, 5, 6, 9, 10, 11) possono generare segnali PWM per regolare la luminosità di lampade, la velocità di motori in continua, la posizione di servomotori; vedremo alcuni esempi e progetti nel [capitolo 5](#).

Le istruzioni fondamentali per l'input/output sono:

```
digitalRead(pin); // legge il bit presente sul pin (da 0 a 13)
digitalWrite(pin, val); // scrive sul pin il bit val (LOW o HIGH)
```

! Anche i sei pin analogici (A0 - A5) possono essere usati come digitali, facendo riferimento agli indirizzi 14 - 19.

3.1 Esercizio svolto

Leggi un pulsante in input e pilota un LED in output

Progetta un sistema per accendere e spegnere un LED in base alla pressione di un pulsante:

- pulsante rilasciato → LED ON;
- pulsante premuto → LED OFF.

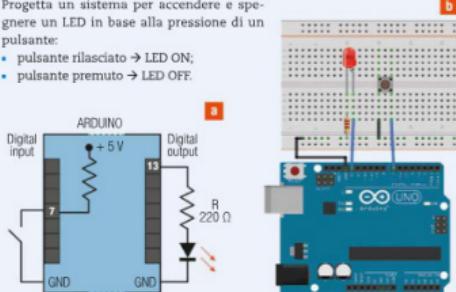


Figura 3.2 a) Schema elettrico; b) schema di montaggio.

Soluzione

Collega un pulsante (normalmente aperto) tra il digital pin 7 e GND (attivando il pull-up interno), e un LED, con un resistore da 220 Ω in serie, tra il digital pin 13 e GND ([figura 3.2](#)).

Il codice dello sketch

```
bool puls; // la variabile booleana puls memorizza lo stato del pulsante

void setup() {
  pinMode(7, INPUT_PULLUP); // imposta il pin digitale 7 come input (pull-up interno)
  pinMode(13, OUTPUT); // imposta il pin digitale 13 come output
}

void loop() {
  puls = digitalRead(7); // leggi il bit sul pin 7, collegato al pulsante
  // e memorizzalo nella variabile puls
  digitalWrite(13, puls); // scrivi sul pin 13, collegato al LED, il valore di puls
}
```

Nello sketch puoi notare che:

- la variabile `puls`, destinata a memorizzare un bit, è definita come `bool` (boolean); sarebbe stato accettato anche il tipo `int` (integer), ma avrebbe occupato due byte di memoria invece che uno;
- il loop ripete all'infinito la lettura del bit sul pin 7 e la scrittura di tale bit sul pin 13; si sarebbe potuto compattare anche in una istruzione unica: `digitalWrite (13, digitalRead (7));`
- dato che il pin 13 è collegato anche al LED L sulla scheda, i due LED si accendono e spengono contemporaneamente.

3.2 Esercizio

Modifica l'[esercizio svolto 3.1](#) in modo che il funzionamento sia invertito: pulsante premuto LED ON, rilasciato LED OFF.

Suggerimento Il simbolo ! equivale alla negazione logica (complementazione del bit) quindi l'istruzione `puls = !puls;` memorizza in `puls` il vecchio valore di `puls` complementato.

3.3 Esercizio

Collega due LED di colori diversi a due uscite e, modificando l'[esercizio svolto 3.1](#), fai in modo che con il pulsante rilasciato siano uno ON e l'altro OFF, e il contrario con il pulsante premuto.

3.4 Esercizio svolto**L'istruzione IF...ELSE per scegliere tra due alternative**

Con un pulsante e due LED, progetta un sistema in cui:

- con il pulsante rilasciato si accende solo il LED rosso;
- premendo il pulsante si accende solo il LED verde.

Soluzione

Collega il pulsante al pin 7 (con pull-up interno) e i LED rosso al pin 12 e verde al pin 13 ([figura 3.3](#)).

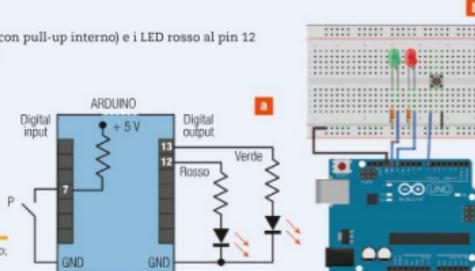


Figura 3.3 a) Schema elettrico; b) schema di montaggio.

Il codice dello sketch

```
bool puls; // la variabile booleana puls memorizza lo stato del pulsante
const int pinPuls=7; // ugualia la costante pinPuls a 7 (pin del pulsante)
const int pinRosso=12; // ugualia la costante pinRosso a 12 (pin del LED rosso)
const int pinVerde=13; // ugualia la costante pinVerde a 13 (pin del LED verde)

void setup(){
  pinMode(pinPuls, INPUT_PULLUP); // imposta il pin digitale 7 come input (con pull-up interno)
  pinMode(pinRosso, OUTPUT); // imposta il pin digitale 12 come output
  pinMode(pinVerde, OUTPUT); // imposta il pin digitale 13 come output
}

void loop(){
  puls = digitalRead(pinPuls); // leggi il bit sul pin 7, collegato al pulsante
  // e memorizzalo nella variabile puls

  if (puls==1){ // se puls=1 (pulsante rilasciato)
    digitalWrite(pinRosso, HIGH); // LED rosso ON (pin 12)
    digitalWrite(pinVerde, LOW); // LED verde OFF (pin 13)
  }
  else { // altrimenti se puls=0 (pulsante premuto)
    digitalWrite(pinRosso, LOW); // LED rosso OFF (pin 12)
    digitalWrite(pinVerde, HIGH); // LED verde ON (pin 13)
  }
}
```

Nello sketch puoi notare che:

- sono state definite le costanti intere pinPuls=7, pinRosso=12 e pinVerde =13, che memorizzano i pin a cui sono collegati il pulsante e i due LED; queste sono poi richiamate nello sketch. In questo modo il programma è più leggibile e, nel caso in futuro tu volessi modificare i pin a cui sono collegati i componenti, sarà sufficiente variare il contenuto delle costanti all'inizio dello sketch.
- È stata usata l'istruzione IF...ELSE per compiere due azioni differenti secondo il valore della variabile puls; se è soddisfatta la condizione tra parentesi (puls==1), sono eseguite le istruzioni del blocco tra le graffe successive, altrimenti quelle del blocco dopo l'**else**.
- Il simbolo **=** è usato per assegnare un valore a una variabile/costante, ad esempio puls=0.
- Il simbolo **==** è usato nelle espressioni per effettuare un confronto (il risultato è booleano, vero o falso); ad esempio nella istruzione **if(puls==1)** l'espressione tra parentesi è VERA se puls vale 1. È equivalente scrivere **if(puls)** ma, per maggiore chiarezza, evidenziamo la condizione completa.

La sintassi dell'istruzione IF...ELSE è (flowchart nella [figura 3.4a](#)):

```
if (espressione) {
  // blocco di istruzioni A
}
else {
  // blocco di istruzioni B
}
```

Se espressione è VERA si esegue il blocco di istruzioni A, altrimenti viene eseguito il blocco B.

L'**else** è facoltativo e va eliminato se non esiste alternativa ad A (flowchart nella [figura 3.4b](#)):

```
if (espressione) {
  // blocco di istruzioni A
}
```

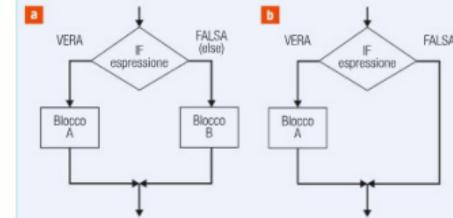


Figura 3.4 Diagrammi di flusso dell'istruzione IF. a) con **else**; b) senza **else**.

3.5 Esercizio svolto

L'istruzione WHILE

L'istruzione WHILE serve a creare un ciclo che si ripete finché è verificata una certa condizione.

Con un pulsante e un LED, progetta un sistema in cui:

- con il pulsante rilasciato il LED è OFF;
- con il pulsante premuto il LED lampeggia alla frequenza di 2 Hz (ON per 0,25 s e OFF per 0,25 s).

Soluzione

Il codice dello sketch

```
bool puls; // puls memorizza lo stato del pulsante
const int pinPuls=7; // ugualia la costante pinPuls a 7 (pin del pulsante)
const int pinLed=13; // ugualia la costante pinLed a 13 (pin del LED)

void setup(){
  pinMode(pinPuls, INPUT_PULLUP); // pin digitale 7 come input (pull-up interno)
  pinMode(pinLed, OUTPUT); // pin digitale 13 come output
}

void loop(){
  digitalWrite(pinLed, LOW); // spegni il LED
  puls = digitalRead(pinPuls); // leggi il bit sul pin 7, collegato al pulsante
  // e memorizzalo nella variabile puls

  while(puls==0){ // finché puls==0 (pulsante premuto) il LED lampeggia
    digitalWrite(pinLed, HIGH); // LED ON
    delay(250); // attendi 250 ms = 0,25 s
    digitalWrite(pinLed, LOW); // LED OFF
    delay(250); // attendi 250 ms
    puls = digitalRead(pinPuls); // aggiorna il valore di puls
  }
}
```



! Il ciclo WHILE è ripetuto finché l'espressione è vera.

Dal punto di vista hardware il circuito è identico a quello in [figura 3.3a](#).

Nello sketch puoi notare che:

- è utilizzata l'istruzione WHILE che ha la seguente sintassi:

```
while(espressione){
  // blocco di istruzioni
}
```

dove il blocco di istruzioni viene ripetuto finché espressione (booleana) è VERA; appena diventa FALSA, il programma procede con le istruzioni dopo la graffa di chiusura del **while**.

L'espressione viene testata prima dell'esecuzione del blocco di istruzioni.

- Nell'[esercizio svolto 3.5](#), il blocco di istruzioni del **while** accende e spegne il LED due volte al secondo.

- È fondamentale che nel blocco di istruzioni del **while** sia prevista la possibilità di uscire dal ciclo: in questo caso l'ultima istruzione del blocco

puls = digitalRead(pinPuls)

effettua la lettura del pin del pulsante (**pinPuls**) e aggiorna la variabile **puls**, in modo che, se il pulsante viene rilasciato, al prossimo test l'espressione del **while** risulta FALSA e si esce dal ciclo.

Esiste anche l'istruzione DO...WHILE, la cui sintassi

```
do
{
  // blocco di istruzioni
} while (espressione);
```

! Il ciclo DO ... WHILE è ripetuto almeno una volta, perché l'espressione viene testata a fine ciclo.

è analoga a quella del WHILE, ma l'espressione è testata dopo aver eseguito il blocco di istruzioni, quindi il blocco viene eseguito almeno una volta.

3.6 Esercizio

Testa o croce

Realizza un gioco «Testa o croce», con un pulsante e un LED:

- finché il pulsante è premuto, il LED lampeggia velocemente (20 Hz)
- rilasciando il pulsante, il LED si ferma casualmente in uno dei due stati.

Suggerimento Modificando l'[esercizio svolto 3.5](#) puoi far sì che la casualezza dell'uscita (LED acceso o spento) sia legata alla frequenza più alta del lampeggio (**delay(25)**) e alla variabilità del tempo di pressione del pulsante. Definisci e inizializza a 0 una variabile **bool statoLed** (memorizza lo stato del LED) che viene invertita (**statoLed!=statoLed**) ad ogni ciclo **while**, in modo che all'uscita dal ciclo (pulsante rilasciato) il valore di **statoLed** non sia prevedibile; a ogni ciclo dovrà anche inviare al LED il valore di **statoLed**.

3.7 Esercizio

Il pulsante Toggle

Realizza un circuito con un pulsante e un LED in cui:

- all'avvio il LED è OFF
- a ogni pressione del pulsante, lo stato del LED si inverte (funzionamento **toggle**).

Suggerimento Anche in questo caso è necessaria una variabile **statoLed**, per memorizzare lo stato del LED, che si deve invertire ogni volta che il pulsante viene premuto e rilasciato; si possono usare due cicli WHILE, uno in attesa che il pulsante sia premuto, l'altro in attesa che il pulsante sia rilasciato.

In alternativa si possono usare due variabili (**oldPuls** e **newPuls**) per memorizzare il vecchio stato e il nuovo stato del pulsante: quando il vecchio è «non premuto» e il nuovo è «premuto» si inverte lo stato del LED.

! I rimbalzi dei contatti
Il contatto di un pulsante in genere si chiude con una serie di rimbalzi che si esauriscono in 10 ms. Nei casi, come il toggle, in cui questo provoca malfunzionamenti, è bene introdurre una pausa di 10 ms dopo la prima rilevazione di chiusura.

3.2 Monitor Seriale: la messa a punto dello sketch

Durante la scrittura di uno sketch, l'IDE fornisce informazioni sulla correttezza delle parole chiave, colorando in modo differente quelle che riconosce appartenenti al lessico del linguaggio di programmazione; inoltre, in fase di compilazione, l'IDE ci invia messaggi nel caso in cui individui degli errori di sintassi nell'uso delle istruzioni.

Una volta che la compilazione e il caricamento su Arduino sono andati a buon fine, non siamo ancora sicuri che il programma esegua esattamente quello che ci eravamo prefissati: possono esserci degli errori di programmazione, difficilmente individuabili con la sola osservazione del comportamento del circuito.

Durante l'esecuzione di uno sketch, per effettuare la ricerca di eventuali errori di programmazione (**debugging**), si può interagire con Arduino mediante il **Monitor Seriale**: premendo l'icona in alto a destra nell'IDE (figura 3.5a) si apre la finestra rappresentata nella figura 3.5b.



Figura 3.5 a) Icona per l'apertura della finestra del Monitor Seriale; b) finestra del Monitor Seriale.

Gli elementi principali della finestra del Monitor Seriale sono:

- Trasmissione da PC verso Arduino:** la linea in alto, dove si inseriscono i dati da mandare ad Arduino premendo il pulsante Invia;
- Trasmissione da Arduino verso PC:** il campo sottostante, in cui sono visualizzati i dati ricevuti da Arduino.

Le principali istruzioni per l'impiego del Monitor Seriale sono:

```
Serial.begin(9600); //imposta in setup() la velocità di comunicazione  
//a 9600 bps (bit per secondo)
```

PC → Arduino (TX)

```
Serial.read(); //restituisce il valore trasmesso dal PC via  
//monitor seriale  
Serial.available(); //restituisce il numero di byte arrivati dal PC  
//sulla porta seriale e disponibili per la lettura  
flush(); //cancella la coda dei dati arrivati dal PC sulla  
//sulla porta seriale
```

```
Arduino → PC (RX)  
Serial.println(val); //stampa sul monitor il valore di val  
//(variabile o costante) e va a capo;  
//il posto di val può esserci una stringa di  
//caratteri compresa tra apici: ad esempio  
// "La velocità è:"  
Serial.print(val); //È come Serial.println(val) ma non va a capo  
//dopo la stampa
```

L'esempio seguente è utile per sperimentare le principali istruzioni per la trasmissione e la ricezione mediante Monitor Seriale.

3.8 Esercizio svolto

Scambio di informazioni tra PC e Arduino, con il Monitor Seriale

Invia un carattere dal monitor seriale; Arduino lo riceve e poi lo ritrasmette al PC scrivendo sul monitor «Ho ricevuto:< seguito dal codice ASCII (in decimal) del carattere.

Per esempio, se scrivi il carattere a e premi invio nella linea in alto della finestra, nel campo di ricezione del monitor seriale appare: **Ho ricevuto: 97** (il codice ASCII decimale corrispondente alla lettera a minuscola è 97).

Soluzione

Il codice dello sketch

```
int byteRicevuto = 0; // variabile per il byte inviato dal PC tramite Monitor Seriale  
  
void setup(){  
  Serial.begin(9600); // apri la porta seriale alla velocità di 9600 bps  
}  
void loop(){  
  
  if (Serial.available() > 0) { // se ci sono dati sulla porta seriale  
    byteRicevuto = Serial.read(); // leggi il carattere inviato dal PC tramite Monitor Seriale e  
    // caricalo nella variabile byteRicevuto  
    Serial.print("Ho ricevuto:"); // scrivi sul monitor la frase "Ho ricevuto:", senza andare a capo  
    Serial.println(byteRicevuto); // scrivi sul monitor il codice ASCII (in decimale) del carattere ricevuto  
  }  
}
```

3.9 Esercizio

Realizza uno sketch che accende un LED quando dal Monitor Seriale viene inviato il carattere X.

Suggerimento Modifica l'**esercizio svolto 3.8**, tenendo conto che la variabile che memorizza il carattere ricevuto deve essere di tipo char.

Una volta effettuata la messa a punto del programma, le istruzioni relative al Monitor Seriale possono essere rimosse, anche perché Arduino in

genere è destinato a funzionare **stand alone**, senza computer con cui scambiare dati sul Monitor Seriale.

Oltre che per il debugging, il Monitor Seriale può essere utilizzato per sostituire, in fase di sviluppo, un display non ancora disponibile su cui visualizzare dei dati, come nell'[esercizio svolto 3.10](#).

3.10 Esercizio svolto

Visualizza un conteggio con il Monitor Seriale

Conta il numero di pressioni di un pulsante (collegato al *digital pin* 7 di Arduino) e visualizza il risultato sul Monitor Seriale.

Soluzione

Il codice dello sketch



```
bool puls;           // la variabile puls memorizza lo stato del pulsante
const int pinPuls=7; // pulsante sul pin 7
int cont=0;          // inizializza a 0 il contatore

void setup(){
  pinMode(pinPuls, INPUT_PULLUP); // pin digitale 7 come input (pull-up interno)
  Serial.begin(9600);           // apri la porta seriale alla velocità di 9600 bps
}

void loop(){
  do{
    puls = digitalRead(pinPuls); // leggi il pulsante
    }while(puls==1);            // finché puls=1 (pulsante rilasciato)

  cont++;                   // incrementa il contatore cont, quando il pulsante viene premuto
  Serial.print("N' pressioni:");
  Serial.println(cont);      // scrivi sul monitor la frase "N' pressioni:" e il valore del contatore
  delay(200);                // antbounce: ritardo per far esaurire i rimbalzi dei contatti

  do{
    puls = digitalRead(pinPuls); // leggi il pulsante
    }while(puls==0);            // finché puls=0 (pulsante premuto)
  delay(200);                // antbounce per eventuali rimbalzi in apertura
}
```



3.11 Esercizio

Aggiungi agli esercizi dal [3.1](#) al [3.7](#) le istruzioni Serial Monitor utili per rilevare eventuali errori di programmazione.

Suggerimento: Stampa sul serial monitor il nome e i valori delle variabili significative per comprendere l'evoluzione dello sketch, utilizzando le istruzioni `Serial.print()` e `Serial.println()`.

■ Approfondimento

La trasmissione seriale

All'interno di un computer o di un microprocessore/microcontrollore la trasmissione dei dati avviene su più linee contemporaneamente (**trasmissione parallela** sul bus dati). In questo modo ad ogni clock si possono inviare più bit in uno stesso istante, rendendo veloce la comunicazione.

All'esterno del computer, nella comunicazione con le periferiche, si preferisce inviare un bit alla volta (**trasmissione seriale**), uno dei motivi è ridurre il numero di fili nei cavi, rendendoli più maneggevoli ed economici.

Uno standard diffuso per la trasmissione seriale è l'USB, acronimo di Universal Serial Bus (Bus Universale Seriale).

Il Monitor Seriale usa la trasmissione seriale per la comunicazione con il computer via USB. Tale comunicazione è replicata sui digital pin 0 (RX dal computer) e pin 1 (TX verso il computer), per cui, se si usa il Monitor Seriale, i digital pin 0 e 1 non devono essere utilizzati per l'input/output.

In realtà, se si impiegano solo istruzioni `Serial.print()` per la comunicazione di dati verso il computer, in caso di necessità il pin 0 (ricezione dal computer) può essere utilizzato senza creare conflitto, come vedrai nel progetto [3.8 «Gara di riflessi»](#).

3.3 Il Buzzer: suonare con Arduino

Dal [capitolo 2](#) sappiamo che il **buzzer** è un attuatore acustico di piccola potenza, basato sull'effetto piezoelettrico (per questo è detto anche **piezo**), usato per emettere suoni ([figura 3.6](#)).

L'effetto piezoelettrico è la proprietà che hanno alcuni cristalli, tra cui il quarzo:

- **se sottoposti a tensione si deformano:** se la tensione è variabile nel tempo, ad esempio un'onda quadrata, la deformazione produce un'onda sonora; questo viene sfruttato per realizzare buzzer o altoparlanti per alte frequenze (tweeter);

- **se sottoposti a compressione generano una tensione:** ciò consente di realizzare microfoni, sensori di pressione o accendini piezoelettrici.

! Se usi il Monitor Seriale, non utilizzare i digital pin 0 e 1 per l'input/output.



Figura 3.6 Il buzzer.

Il buzzer (**piezo**) contenuto nello Starter Kit può quindi essere usato come emettitore di suoni, collegandolo direttamente a un'uscita digitale di Arduino su cui generare un'onda quadrata (tensione alternativamente HIGH e LOW) a una certa frequenza. In alternativa, può essere collegato a un ingresso analogico per rivelare vibrazioni acustiche.

Arduino mette a disposizione la funzione `tone` per semplificare la generazione delle note:

`tone (pin, frequenza, durata)`

dove i parametri sono:

- **pin:** il numero del pin d'uscita a cui è collegato il buzzer;
- **frequenza:** la frequenza della nota;

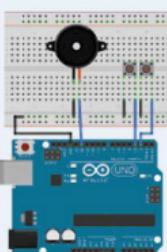
! L'istruzione `tone` per generare note musicali

L'altezza di un suono (nome della nota) dipende dalla sua frequenza: ad esempio la nota LA del diapason oscilla a 440 Hz, cioè 440 volte al secondo.

- durata (facoltativo): la durata in millisecondi della nota.

Con l'istruzione `notone(p)` la nota sul pin s'interruppe.

In commercio esistono anche buzzer che generano una nota fissa se alimentati con una tensione continua, ad esempio 5 V; sono usati come segnalatori acustici.



3.12 Esercizio svolto

Suona due note

Suona le note DO e RE con il buzzer, premendo due pulsanti.

Soluzione

Il cuore della funzione `loop()` è realizzato con due cicli `while`, uno per pulsante: finché un pulsante è premuto il programma rimane nel ciclo corrispondente, insensibile alla pressione dell'altro. All'uscita del `while` (pulsante rilasciato) la nota sul buzzer viene interrotta, in attesa della pressione di uno dei due pulsanti.

Figura 3.7 Circuito per l'esercizio svolto 3.12

«Suona due note».

Il codice dello sketch

```
const int pinPulsDo=3; // pulsante del Do sul pin 3
const int pinPulsRe=2; // pulsante del Re sul pin 2
const int pinBuzzer=12; // Buzzer sul pin 12
bool notaDo; // la variabile notaDo memorizza lo stato del pulsante Do
bool notaRe; // la variabile notaRe memorizza lo stato del pulsante Re

void setup(){
  pinMode(pinPulsDo, INPUT_PULLUP); // pin digitale 3 come input (pull-up interno)
  pinMode(pinPulsRe, INPUT_PULLUP); // pin digitale 2 come input (pull-up interno)
  pinMode(pinBuzzer, OUTPUT); // pin digitale 12 come output (buzzer)
}

void loop(){
  notaDo = digitalRead(pinPulsDo); // leggi il pulsante Do
  notaRe = digitalRead(pinPulsRe); // leggi il pulsante Re

  while(notaDo==0){ // finché il pulsante Do è premuto
    tone(pinBuzzer, 262); // suona la nota Do (262 Hz)
    notaDo = digitalRead(pinPulsDo); // leggi il pulsante Do
  }
  noTone(pinBuzzer); // dopo il rilascio del Do, spegni la nota

  while(notaRe==0){ // finché il pulsante Re è premuto
    tone(pinBuzzer, 294); // suona la nota Re (294 Hz)
    notaRe = digitalRead(pinPulsRe); // leggi il pulsante Re
  }
  noTone(pinBuzzer); // dopo il rilascio del Re, spegni la nota
}
```



3.13 Esercizio

Realizza una tastiera con 5 note, le cui frequenze sono: Do=262 Hz, Re=294 Hz, Mi=330 Hz, Fa=349 Hz, Sol=392 Hz.

Suggerimento Espandi l'esercizio svolto 3.12.

3.14 Esercizio

Progetta un allarme sonoro a sirena, modificando l'esercizio svolto 3.12: quando si preme un pulsante, il buzzer alterna le note Do (262 Hz) e Fa (349 Hz).

Suggerimento Per ogni nota, inserisci nell'istruzione `tone` una durata di 500 ms.

3.15 Esercizio svolto

Suona il brano «Fra Martino campanaro»

Premendo un pulsante, il buzzer suona le prime quattro battute del brano, come indicato nel rigo musicale in figura 3.8. Il tempo del metronomo è 240 semiminime (le note con la testa piena) al minuto.

Le frequenze delle note utilizzate sono: Do=262 Hz, Re=294 Hz, Mi=330 Hz, Fa=349 Hz, Sol=392 Hz.



Figura 3.8 Rigo musicale con la melodia di «Fra Martino campanaro».

Soluzione

Lo schema hardware è analogo a quello in figura 3.7, con solo un pulsante collegato al pin 3.

Il metronomo a 240 significa che devi suonare 240/60 = 4 semiminime al secondo; quindi ognuna dura 250 ms.

Nello spartito leggi due Sol che durano il doppio delle altre note (minime, quelle con la testa vuota); per semplificare lo sketch, al posto di ogni minima si suonano due semiminime (quindi due volte il Sol). Di conseguenza suonerai una sequenza di 16 note, 4 per ogni battuta.

```

void setup(){
  pinMode(pinPuls, INPUT_PULLUP); // pin digitale 3 input (pulsante, con pull-up interno)
  pinMode(pinBuzzer, OUTPUT); // pin digitale 12 output (Buzzer)
}

void loop(){
  puls = digitalRead(pinPuls); // leggi lo stato del pulsante

  if(puls==0){ // se il pulsante è premuto
    for(int nota = 0; nota < 16; nota++){
      tone(pinBuzzer, melodia[nota], 250); // esegui il ciclo FOR per suonare le 16 note
      delay(250); // suona ogni nota di melodia[ ] per 250 ms
      delay(250); // attendi 250 ms
    }
  }
}

```

Gli array sono usati per gestire gruppi di dati omogenei.

Nello sketch è dichiarato l'array `melodia[]`, che contiene la sequenza delle 16 note della melodia:

- L'array è un tipo di dato costituito da un insieme di n elementi omogenei, numerati da 0 a $n-1$, in base alla posizione occupata nell'elenco;
- La dichiarazione:

```

int melodia [ ] = {
  DO, RE, MI, DO, DO, RE, MI, DO,
  MI, FA, SOL, SOL, MI, FA, SOL, SOL
};

```

definisce l'array di nome `melodia` contenente 16 valori interi (int). Tra le graffe sono elencati gli elementi dell'insieme numerati da 0 a 15, che in questo caso corrispondono alle frequenze delle 16 note associate alle costanti Do, Re, ecc.

Tra le quadre si può indicare il numero complessivo degli elementi dell'array [16], ma è facoltativo.

- La posizione dell'elemento nell'insieme individua il suo numero: per esempio, per richiamare il sesto elemento dell'array `melodia` si scrive `melodia[5]` (ricorda che in un array la numerazione degli elementi parte da 0).

- Quindi nel ciclo `for` dello sketch:

```

for (int nota = 0; nota < 16; nota++){
  tone (pinBuzzer, melodia[nota], 250);
  delay(250);
}

```

mentre la variabile `nota` s'incrementa da 0 a 15 con il ciclo `for`, nel campo **frequenza** dell'istruzione `tone` si avvicedono le 16 frequenze delle 16 note del brano.

Progettare con ARDUINO

3.A Gara di riflessi

Progetta un circuito per gestire una gara di riflessi tra due concorrenti A e B, utilizzando i seguenti elementi:

- un pulsante e un LED rosso per ogni concorrente;
- un pulsante di start, un LED verde, un buzzer.

Lo svolgimento di una prova è il seguente:

- durante la fase di attesa il LED verde è acceso;
- quando si preme start il LED verde si spegne per un tempo di durata casuale (da 3 s a 8 s);
- al termine del tempo suona il buzzer e si riaccende il LED verde;
- il primo concorrente che preme il proprio pulsante vince: il suo LED lampeggia e il buzzer suona sincrono col LED per 5 s;
- se alla fine del tempo casuale (al suono del buzzer) uno dei due concorrenti sta già premendo il pulsante, vince l'altro, se ha premuto correttamente.

Inoltre, al termine di ogni prova, deve apparire sul Monitor Seriale:

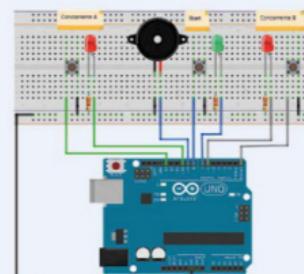
- la scritta: «Ha vinto» seguita dal nome del concorrente vincente (A o B);
- nella riga sotto, il punteggio dei due concorrenti, da quando lo sketch è stato caricato o dall'ultimo reset.

Considera la possibilità di aggiungere in futuro un display a cristalli liquidi per visualizzare, oltre al vincitore della prova e ai punteggi dei concorrenti, anche il tempo del vincitore e il record.

Soluzione

Facciamo le seguenti scelte hardware (figura 3.9):

- pulsanti:
 - PA (pulsante concorrente A) → pin 13;
 - PB → pin 0;
 - Pstart → pin 8
- LED:
 - LA (LED concorrente A) → pin 10;
 - LB → pin 6;
 - Lverde → pin 7
- Buzzer → pin 9



La scelta dei pin tiene conto di una possibile espansione con un display LCD, per il quale si riservano i pin (12, 11, 5, 4, 3, 2), e del fatto che utilizzando il Monitor Seriale solo in TX (pin 1) da Arduino al computer, il pin 0 (RX seriale) può essere collegato al pulsante PB senza conflitto.

Analizziamo lo sketch dividendolo in tre parti.

! Usiamo l'istruzione `random`, il Monitor Seriale, le function e gli operatori OR e AND.

Figura 3.9 Circuito per il progetto «Gara di riflessi».

Il codice della sketch (parte 1)

```

const int pinPA=13; // pin 13 pulsante concorrente A
const int pinPB=0; // pin 0 pulsante concorrente B
const int pinPstart=8; // pin 8 pulsante start
const int pinBuzzer=9; // pin 9 Buzzer
const int pinLA=10; // pin 10 LED concorrente A
const int pinLB=6; // pin 6 LED concorrente B
const int pinVerde=7; // pin 7 LED verde

bool PA; // stato del pulsante A
bool PB; // stato del pulsante B
bool Pstart; // stato del pulsante Start
int totVittA=0; // contatore vittorie concorrente A
int totVittB=0; // contatore vittorie concorrente B
int nota; // frequenza della nota del vincitore

void setup(){
    pinMode(pinPA, INPUT_PULLUP); // pin digitale 13 input (pull-up interno)
    pinMode(pinPB, INPUT_PULLUP); // pin digitale 0 input (pull-up interno)
    pinMode(pinPstart, INPUT_PULLUP); // pin digitale 8 input (pull-up interno)
    pinMode(pinBuzzer, OUTPUT); // pin digitale 9 output (Buzzer)
    pinMode(pinLA, OUTPUT); // pin digitale 10 output
    pinMode(pinLB, OUTPUT); // pin digitale 6 output
    pinMode(pinVerde, OUTPUT); // pin digitale 7 output

    randomSeed(analogRead(0)); // inizializza la funzione random, leggendo del rumore
    // sul pin scollegato analog input 0

    Serial.begin(9600); // apri la porta seriale alla velocità di 9600 bps
}

```

Nella prima parte dello sketch puoi notare, oltre alle varie definizioni e inizializzazioni, l'istruzione:

```
randomSeed(analogRead(0));
```

Che serve ad inizializzare la funzione `random`, usata nel `loop()` per generare un tempo casuale tra 3 s e 8 s: `analogRead(0)` legge la tensione presente sul pin `analog input 0`, che, essendo scollegato, sarà un valore casuale dovuto al rumore captato sul pin 0.

Il codice dello sketch (parte 2)

```

void loop(){
    digitalWrite(pinVerde, HIGH); // accendi il LED verde
}

```



↓

```

do{
    Pstart = digitalRead(pinPstart); // leggi il pulsante Start
} while (Pstart==1); // finché il pulsante Start non è premuto

digitalWrite(pinVerde, LOW); // spegni il LED verde
delay(Crandom(3000, 8000)); // attendi un tempo casuale tra 3 s e 8 s
PA = digitalRead(pinPA); // leggi il pulsante PA
PB = digitalRead(pinPB); // leggi il pulsante PB
tone (pinBuzzer, 262); // suona la nota D (262 Hz)
digitalWrite(pinVerde, HIGH); // riaccendi il LED verde

if(PA==0 || PB==0) { // se uno dei due concorrenti ha premuto in anticipo
    if(PA==0 && PB==1) vincitore(pinLA); // se PA è già premuto vince B (chiama la funzione "vincitore")
    if(PA==1 && PB==0) vincitore(pinLB); // se PB è già premuto vince A (chiama la funzione "vincitore")
    if(PA==0 && PB==0) Serial.println("Gara non valida"); // se entrambi hanno premuto in anticipo
}
else {
    // se è tutto regolare aspetta la prima pressione
    do {
        PA = digitalRead(pinPA); // leggi il pulsante PA
        PB = digitalRead(pinPB); // leggi il pulsante PB
        } while(PA && PB); // finché non viene premuto un pulsante

    if(PA==0) vincitore(pinLA); // se P1 è premuto vince A (chiama la funzione "vincitore")
    if(PB==0) vincitore(pinLB); // se P2 è premuto vince B (chiama la funzione "vincitore")
}
noTone (pinBuzzer); // spegni il buzzer
}

```

La seconda parte dello sketch è la funzione `loop()`:

- i commenti descrivono il flusso del programma, che evolve come richiesto dalle specifiche;
- puoi notare l'uso della funzione `random`, come parametro dell'istruzione `delay`, per ottenere un ritardo casuale, compreso tra 3000 ms e 8000 ms; `random(min, max)` va usata abbinate a `randomSeed(analogRead(0))`, contenuta in `setup()`, per ottenere un numero casuale tra i valori `min` e `max`;
- nell'istruzione `if (PA==0 || PB==0)`, il simbolo `||` realizza la funzione logica OR tra le due espressioni: il risultato è VERO se almeno una delle due è VERA, cioè se almeno un concorrente ha premuto (A o B);
- nell'istruzione `if (PA==0 && PB==1)`, il simbolo `&&` realizza la funzione logica AND tra le due espressioni; il risultato è VERO se le espressioni sono entrambe VERE, cioè se A ha premuto durante il suono mentre B non premeva (quindi vince B);
- nota che in entrambe le istruzioni `if` le graffe non sono necessarie perché il blocco è di una sola riga;
- in vari punti dello sketch si nomina `vincitore()`: questa è una chiamata alla funzione `vincitore`, definita nella terza parte dello sketch, alla quale si comunica (con il parametro tra parentesi) il pin del LED del vincitore (`pinLA` o `pinLB`). La funzione farà lampeggiare tale LED e altre operazioni relative al vincitore della sfida.

Il codice dello sketch (parte 3)

```

void vincitore (int winner){ // FUNCTION che esegue le operazioni relative al vincitore;
    // il pin del LED è passato e associato al parametro winner
    digitalWrite(pinVerde, LOW); // spegni il LED verde

    if(winner==pinLA){ // se ha vinto A
        Serial.println("Ha vinto A"); // scrivi sul serial Monitor "Ha vinto A"
        totWinA++; // incrementa i punti di A
        nota=440; // assegna alla variabile nota il valore 440 (IA)
    }

    if(winner==pinLB){ // se ha vinto B
        Serial.println("Ha vinto B"); // scrivi sul serial Monitor "Ha vinto B"
        totWinB++; // incrementa i punti di B
        nota=587; // assegna alla variabile nota il valore 587 (RE)
    }

    // scrivi sul serial monitor i punti totali di A e di B
    Serial.print("Punti totali A:");
    Serial.print(totWinA);
    Serial.print(" B:");
    Serial.println(totWinB);

    // fai lampeggiare il LED e suona la nota del vincitore, per 5 volte
    for (int cont =1; cont<=5; cont++) { // per 5 volte
        tone (pinBuzzer, nota); // suona la nota del vincitore
        digitalWrite(winner, HIGH); // accendi il LED del vincitore
        delay(500); // aspetta 500 ms
        noTone (pinBuzzer); // interrompi la nota
        digitalWrite(winner, LOW); // spegni il LED
        delay(500); // aspetta 500 ms
    }
}

```

La terza porzione di sketch è costituita dalla function `vincitore`, scritta fuori dalla funzione `loop()`.

La function `vincitore`, quando viene nominata nel `loop()`, esegue le seguenti operazioni:

1. riceve il pin del LED del vincitore (`pinLA` o `pinLB`) e lo associa alla variabile `winner`;
2. scrive sul Monitor (**figura 3.10**) chi ha vinto e poi incrementa il contatore dei punti del vincitore (ad esempio `totWinA++` significa somma 1 a `totWinA`);
3. scrive sul Monitor i punti totali di A e B;
4. fa suonare il buzzer (con note diverse per ciascun concorrente) e fa lampeggiare il LED del vincitore in modo sincrono per 5 volte.

Poi l'esecuzione del programma ritorna al `loop()` e prosegue dal punto successivo a quello in cui è stata chiamata la function. Nota l'uso delle istruzioni `Serial.print()` e `Serial.println()`, per comporre sul Monitor Seriale le scritte richieste.

```

Ha vinto A
Punti totali A:1 B: 0
Ha vinto A
Punti totali A:2 B: 0
Ha vinto B
Punti totali A:2 B: 1

```

Figura 3.10 Le scritte sul Monitor Seriale, dopo 2 vittorie di A e una vittoria di B.

3.16 Esercizio

Modifica il [progetto 3.A «Gara di riflessi»](#) in modo che l'azzeramento dei contatori delle vittorie dei concorrenti si ottenga tenendo premuto start per più di due secondi, senza dover premere reset su Arduino; ciò deve essere segnalato da un breve lampeggio del LED verde. Scrivi anche sul Monitor Seriale una frase che segnali l'inizio di una nuova gara con i contatori a zero.

Suggerimento All'inizio del loop, dopo aver rivelato la pressione di start e aver spento il LED verde, aspetta 2 s e rileggi start; se è ancora premuto, azzerà i contatori `totWin`, fai lampeggiare il LED verde e scrivi sul monitor la frase «Nuova partita», seguita dai punteggi che ora risultano a zero. Avendo già aspettato 2 s, il tempo di attesa nella funzione `random` dovrà essere ridotto tra 1000 ms e 6000 ms.

3.17 Esercizio

Espandi il [progetto 3.A «Gara di riflessi»](#) a 3 concorrenti.

Suggerimento Per ogni concorrente devi aggiungere un LED e un pulsante.

■ Approfondimento**Le function**

Una function (detta anche subroutine) è una porzione di sketch che svolge una funzione specifica, che viene effettuata più volte nel programma principale (`loop`).

La function va codificata fuori dal ciclo `loop`.

L'uso della function consente di scrivere il codice una sola volta, associandolo a un nome (nel nostro caso `vincitore`), passando gli opportuni parametri tra parentesi: nel nostro caso il pin (`pinLA` o `pinLB`) del LED del vincitore.

Quando si nomina la function nel `loop`, ad esempio `vincitore(pinLA)`, viene eseguita la corrispondente sequenza di istruzioni (scritte fuori dal `loop`), sostituendo al parametro (`winner`) il valore di `pinLA`.

La function può restituire un risultato che viene usato nella prosecuzione del `loop`: nel nostro caso non c'è nessun valore restituito e il nome della function `vincitore` è preceduto da `void`.

Qui di seguito riportiamo un esempio della sintassi di una function che restituisce un valore: quando la function `moltiplica` (dichiarata fuori dal ciclo `loop`) è richiamata nel `loop()`, esegue il prodotto tra due interi `a` e `b` e restituisce (`return`) il risultato.

```

void setup() {
}

void loop() {
    int a = 2;
    int b = 3;
    int p = moltiplica(a, b); // p assume il valore a*b, restituito
                             // dalla function "moltiplica" (ora p=6)
}

```

! Le function consentono di semplificare lo sketch: nel caso di operazioni ripetute più volte, il codice viene scritto solo una volta e richiamato quando serve.

```
int multiplica(int x, int y){ // codice della function "multiplica"
    // scritto fuori dal loop
    int prodotto = x*y;
    return prodotto; // la function restituisce la variabile "prodotto"
}
```

Il risultato di questo sketch può essere visualizzato sul Monitor Serie, inserendo opportunamente le istruzioni `Serial.begin(9600)` e `Serial.println(p)`, come descritto nel [paragrafo 3.2](#), oltre a un `delay` per rallentare la scrittura dei valori sul Monitor.

L'esempio che segue contiene la function `print` che stampa sul Monitor Serie la frase «Hello world!» senza passare parametri (parentesi vuote). La function non restituisce nulla e, in questo caso, nella dichiarazione il nome va preceduto da `void`.

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

void loop(){
    print(); // chiama la function "print" senza passare parametri
    delay(500);
}

void print(){
    // codice della function "print",
    // non restituisce valori (void)
    Serial.println("Hello world!"); // scrivi "Hello world!" sul monitor
    // non serve "return" perché non ci sono
    // valori restituiti
}
```

Progettare con ARDUINO

3.B Heart beating, un cuore che pulsia

! In questo progetto faremo uso dei cicli FOR nidificati, della funzione `random` e di un array.

Figura 3.11 Progetto «Heart beating»: la disposizione dei LED sul supporto.

Progetta un circuito con un pulsante, un buzzer e 10 LED rossi, disposti a forma di cuore, con questo funzionamento:

- con il pulsante rilasciato: il buzzer è silenzioso e i LED si accendono uno alla volta in modo casuale;
- con il pulsante premuto: il buzzer suona una melodia e i LED lampeggiano a ritmo, tutti contemporaneamente, evidenziando la forma del cuore.

Soluzione

I 10 LED devono essere montati su un supporto morbido da appendere al collo come una collana, sotto una maglia chiara, in modo che i LED siano a contatto con il tessuto e visibili dall'esterno quando sono accesi ([Figura 3.11](#)).



È più pratico inserire Arduino Uno, il buzzer e la batteria da 9 V in un contenitore da tenere all'altezza della vita, collegato con il supporto dei LED mediante un cavo a 11 fili e un connettore. Il pulsante dovrà essere montato sulla scatola e accessibile facilmente.

Se impieghi il piccolo Arduino LilyPad, questo può essere fissato direttamente sul supporto dei LED. Alla pressione del pulsante, il buzzer suonerà la melodia e i LED lampeggeranno tutti insieme.

Nello schema elettrico in [figura 3.12](#) gli anodi dei dieci LED sono collegati ai digital pin da 1 a 10: questo semplificherà alcune istruzioni dello sketch.

La corrente massima complessiva che Arduino Uno può fornire sulle uscite vale 150 mA; poiché ogni LED (con $220\ \Omega$ in serie) assorbe poco più di 10 mA, quando sono tutti accesi, compreso il buzzer, il limite massimo di corrente non viene superato. Per diminuire il consumo della batteria puoi portare il valore dei resistori a $330\ \Omega$, con un leggero calo di luminosità.

Analizziamo lo sketch dividendolo in due parti.

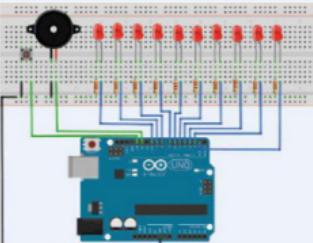


Figura 3.12 Progetto «Heart beating»: il circuito su breadboard.

Il codice dello sketch (parte 1)

```
bool puls; // puls memorizza lo stato del pulsante
const int pinPuls=13; // pulsante sul pin 13
const int pinBuzzer=12; // buzzer sul pin 12
const int melodia[] = {220, 247, 262, 294, 330, 349, 349, 349, 349, 349};
// note: La, Si, Do, Re, Mi, Fa, Fa, Fa, Fa
```

```
void setup(){
    pinMode(pinPuls, INPUT_PULLUP); // pulsante sul pin 13, input con pull-up interno
    pinMode(pinBuzzer, OUTPUT); // Buzzer sul pin 12, output

    for (int pin=1; pin<=10; pin++) { // i dieci LED sono collegati ai pin da 1 a 10
        pinMode(pin, OUTPUT); // e inizializzati output con un ciclo FOR
    }

    randomSeed(analogRead(0)); // inizializza la funzione random, leggendo il rumore
    // sul pin scollegato analog input 0
}
```

Prima di `setup()` puoi notare la definizione dell'array `melodia[]`, che contiene la sequenza delle frequenze delle note della melodia eseguita nel `loop()`. Per ridurre il numero delle righe del codice, l'inizializzazione delle dieci uscite collegate ai LED è realizzata con un ciclo `for`.

Al termine del `setup()` è stata posta `randomSeed(analogRead(0))` per inizializzare la funzione `random`, usata nel `loop()`, in modo che la sequenza prodotta sia veramente casuale.

Il codice dello sketch (parte 2)

```

void loopO(){
    puls = digitalRead(pinPuls); // leggi il pulsante

    if(puls==1){ // se il pulsante è rilasciato
        int ledRN = random(1,10); // scegli un LED a caso e memorizzalo in ledRN
        digitalWrite(ledRN, HIGH); // accendi il LED
        delay (300); // attendi 300 ms
        digitalWrite(ledRN, LOW); // spegni il LED
    }

    else{ // se invece il pulsante è premuto
        for(int nota = 0; nota<9; nota++){ // seleziona ciclicamente le 9 note
            for(int ledPin = 1; ledPin<=10; ledPin+=1){ // accendi tutti i LED
                digitalWrite(ledPin, HIGH);
            }
            tone(pinBuzzer, melodia[nota], 150); // suona la nota dell'array melodia[]
            delay (200); // mantieni i LED accesi per 200 ms
            for(int ledPin = 1; ledPin<=10; ledPin+=1){ // spegni tutti i LED
                digitalWrite(ledPin, LOW);
            }
            delay (100); // mantieni i LED spenti per 100 ms
        }
        delay (500); // fa una pausa tra una melodia e l'altra
    }
}

```

Nel `loopO()`, se il pulsante è rilasciato, i LED sono accesi e spenti in modo casuale (`random()`). Se invece (`else`) il pulsante è premuto (`puls==0`), viene eseguito un ciclo `for` (selezione ciclicamente le 9 note della melodia) che contiene altri due `for` interni (nidificati):

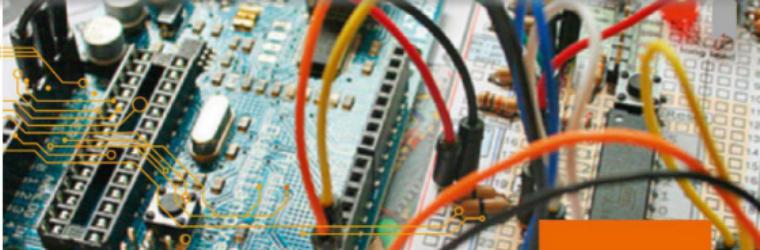
1. il primo dei `for` interni accende tutti i LED;
2. dopo di che viene suonata la nota corrente mediante l'istruzione `tone()`;
3. poi il secondo `for` spegne tutti i LED;
4. dopo una pausa di 500 ms si ricomincia il `loop`.

La nota dura 150 ms, mentre il delay che segue è un po' più lungo (200 ms); in questo modo i suoni risultano staccati uno dall'altro.

3.18 Esercizio

Nel progetto 3.8 «Heart beating» scegli e programma un'altra melodia; quella attuale è l'inizio del brano di Luigi Tenco «Mi sono innamorato di te». Prova ad aggiungere al circuito un secondo pulsante, per poter scegliere tra due melodie differenti.

Suggerimento Sul sito Arduino, nella pagina [Reference > note](#), trovi l'elenco delle note e delle corrispondenti frequenze. Se il brano che hai scelto contiene note con durate differenti tra loro, nell'istruzione `tone()` devi specificare, oltre alle frequenze delle note lette nell'array `melodia[]`, anche le loro durate in millisecondi, che inserirai in un altro array chiamato, per esempio, `durata[]`.



Capitolo

4

Il display LCD e le misure di tempo

4.1 Il display LCD

I display sono dispositivi di output che visualizzano informazioni, in generale caratteri alfabetici (lettere e numeri), simboli o figure. Le tecnologie principali per realizzare un display sono:

- **LED (Light Emitting Diode):** gli elementi che formano un display sono diodi LED, che si illuminano quando sono alimentati; sono visibili al buio ma hanno un consumo di energia non trascurabile ([figura 4.1a](#)).
- **LCD (Liquid Crystal Display):** gli schermi di questo tipo sono basati su particolari cristalli, liquidi a temperatura ambiente, che posti tra due vetri si orientano quando sono sottoposti a un campo elettrico. Il campo viene creato da tanti elettrodi trasparenti con la forma di un elemento d'immagine (pixel o altra figura), a cui si porta tensione; il campo elettrico fa sì che il cristallo da trasparente diventi opaco, mostrando la forma dell'elettrodo. Il display LCD non emette luce, quindi per essere visto al buio necessita di retroilluminazione (a LED); il consumo è molto ridotto, quindi è adatto anche in apparecchi alimentati a batteria.



Figura 4.1 a) display LED a 7 segmenti; b) display LCD 16 colonne x 2 righe.

Il display LCD compreso nello starter kit ([figura 4.1b](#)), che useremo per i prossimi esempi, ha le seguenti caratteristiche:

- può visualizzare caratteri alfabetici organizzati in 16 colonne (da 0 a 15) e 2 righe (0 e 1). Ogni carattere è composto da una matrice di punti (5 x 7); è anche possibile creare caratteri personalizzati fino a 5 x 8 punti..

- Il circuito integrato che controlla il funzionamento del display è l'Hitachi HD44780, diventato uno standard di riferimento.
- Il display comunica attraverso 16 pin, 12 dei quali vanno collegati come specificato nella **tavola 4.1** e nello schema in **figura 4.2**.

Pin Display LCD	Pin Arduino	Descrizione
1 (VSS)	GND	
2 (VDD)	+5 V	
3 (VO)		Regolazione del contrasto mediante trimmer (0 V - 5 V)
4 (RS)	12 (Digital)	Selezione del registro interno su cui leggere/scrivere
5 (R/W)	GND	Seleziona la modalità: lettura (5 V) / scrittura (GND)
6 (Enable)	11 (Digital)	Abilita la scrittura nei registri interni
11 (D4)	5 (Digital)	Linea dati collegata a un registro interno
12 (D5)	4 (Digital)	" " " "
13 (D6)	3 (Digital)	" " " "
14 (D7)	2 (Digital)	" " " "
15 (A)	+4,2 V	Anodo del LED per retroilluminazione
16 (K)	GND	Catodo del LED per retroilluminazione

Tavola 4.1 Collegamenti tra i pin del display LCD e i pin di Arduino.

Per fornire al pin 15 i 4,2 V necessari per la retroilluminazione del display, si usa la V_{cc} di Arduino (5 V); in serie alla V_{cc} si inserisce una resistenza di $220\ \Omega$; la caduta sul resistore farà abbassare i 5 V ad un valore accettabile dal display.

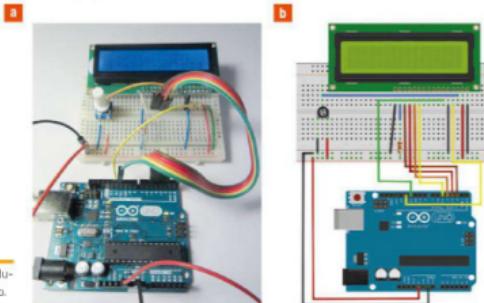


Figura 4.2 Collegamento tra Arduino e display: a) reale; b) simulato.

4.2 La gestione del display

Per gestire il display LCD si usano le istruzioni della libreria `LiquidCrystal.h`. Quelle fondamentali sono le seguenti:

```
#include <LiquidCrystal.h> // include la libreria LiquidCrystal.h
LiquidCrystal lcd(12,11,5,4,3,2); // inizializza la libreria con i numeri dei pin di
// interfaccia di Arduino (12,11,5,4,3,2)
lcd.begin(16, 2); // imposta il numero di colonne e righe del display
lcd.setCursor(colonna, riga); // porta il cursore nella posizione del display
// specificata da (colonna 0-15, riga 0-1)
lcd.print(dato); // scrive il dato (di tipo: char, byte, int, long, string)
// sul display; ad esempio lcd.print(cont), scrive
// il valore della variabile cont nella posizione del
// cursore; lcd.print("Ciao!"), scrive la stringa
// "Ciao!" nella posizione del cursore.
```

4.1 Esercizio svolto

Scritta «W Arduino!» lampeggiante

Scrivi uno sketch per far lampeggiare sul display la scritta «W Arduino!», con la «W» al centro della prima riga (riga 0) e «Arduino!» al centro della seconda riga (riga 1).

Soluzione

Tieni presente che le 16 colonne sono numerate da 0 a 15, mentre le due righe da 0 a 1. Qui di seguito lo sketch commentato.

```
Il codice dello sketch «W Arduino!»
#include <LiquidCrystal.h> // include la libreria LiquidCrystal.h
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // inizializza la libreria con i numeri
// dei pin di interfaccia

void setup() {
    lcd.begin(16, 2); // imposta il numero di colonne e righe del display
    lcd.setCursor(7, 0); // porta il cursore su colonna 7 e riga 0
    lcd.print("W"); // scrivi "W" al centro della prima riga
    lcd.setCursor(4, 1); // porta il cursore su colonna 4 e riga 1
    lcd.print("Arduino!"); // scrivi "Arduino!" al centro della seconda riga
}

void loop() {
    // lampeggio del display
    lcd.noDisplay(); // spegni il display
    delay(500); // pausa di 500 ms
    lcd.display(); // accendi il display
    delay(500); // pausa di 500 ms
}
```

Altre istruzioni della libreria `LiquidCrystal.h` sono:

```
lcd.noDisplay(); // spegne il display
lcd.display(); // accende il display
lcd.clear(); // cancella il contenuto del display
lcd.home(); // porta il cursore in alto a sinistra
lcd.setCursor(); // mostra il cursore sul display
lcd.noCursor(); // nasconde il cursore sul display
lcd.autoscroll(); // attiva l'autoscroll: i caratteri si aggiungono
// a destra traslando verso sinistra
lcd.noAutoscroll(); // disattiva l'autoscroll
```

Collegare il display a pin differenti di Arduino

È possibile collegare il display a pin diversi dai sei sopra elencati (12, 11, 5, 4, 3, 2), tenendo conto del significato dei parametri della inizializzazione riportati nella [tavola 4.1](#):

`LiquidCrystal lcd(RS, Enable, D4, D5, D6, D7)`

Quindi, per esempio, se si vogliono liberare i pin 3 e 2 per dedicarli a segnali di interrupt esterni (vedi [paragrafo 4.4](#)) è sufficiente spostare i fili su altri pin liberi, per esempio 7 e 6, e dichiarare

`LiquidCrystal lcd(12, 11, 5, 4, 7, 6)`

Si ricorda che anche i sei pin analogici (A0 - A5) possono essere usati come digitali e quindi essere collegati al display; i loro indirizzi, come pin digitali, vanno dal 14 al 19.

4.2 Esercizio

Scritta lampeggiante con numero

Modifica l'[esercizio svolto 4.1](#) in modo che, oltre alla scritta lampeggiante, in basso a destra nel display compaia il numero dei lampaggi effettuati; una volta arrivato a 15 il numero ricomincia da 1.

Suggerimento Conta i 15 lampaggi con un ciclo `for`, scrivendo il valore della variabile contatore nella posizione (14, 1); fino al 9 il numero occuperà un carattere, dal 10 in poi due (colonne 14 e 15). Prima di iniziare di nuovo il ciclo, cancella tutto lo schermo con l'istruzione `lcd.clear()`, in modo che non rimangano residui del conteggio precedente nella posizione (15,1), che a inizio conteggio non viene occupata.

Fai attenzione al fatto che, se hai scritto i caratteri nel `setup()`, una volta cancellati devono essere riscritti.

4.3 Esercizio

Fiocchi di neve sul display

Fai apparire sul display tanti asterischi (fiocchi di neve) disposti a scacchiera,

dopo 500 ms gli asterischi e gli spazi si invertono per altri 500 ms, e così via. Inoltre, per collegare il display ad Arduino con un unico connettore, utilizza i pin da 8 a 13.

Suggerimento Anche in questo caso è necessaria l'istruzione `lcd.clear()` per pulire lo schermo, altrimenti dopo un ciclo, il display si riempie di asterischi in ogni posizione.

Per variare i pin, dichiara `LiquidCrystal lcd(13, 12, 11, 10, 9, 8)`, spontaneamente i fili nell'ordine: 12 → 13, 11 → 12, ecc. fino a 2 → 8.



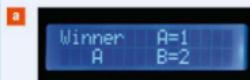
Figura 4.3 Disposizione delle scritte sul display LCD: a) nell'esercizio 4.1; b) nell'esercizio 4.2.

Progettare con ARDUINO

4.A «Gara di riflessi» con display

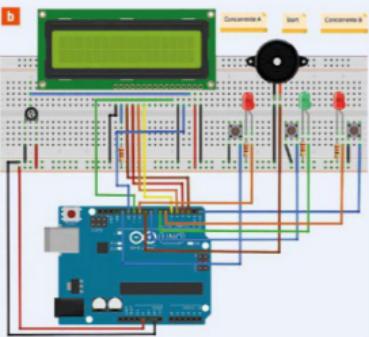
Modifica la «Gara di riflessi» ([progetto 3.A del capitolo 3](#)), in modo che i risultati in uscita appaiano su un display LCD invece che sul Serial Monitor. Inoltre, quando uno dei due concorrenti raggiunge i 5 punti, fai lampeggiare 10 volte il display, in contemporanea ai LED e al suono del buzzer (che alterna due note), per segnalare la fine della gara e indicare il vincitore e il punteggio finale. La [figura 4.4a](#) mostra una delle soluzioni possibili per la designazione del vincitore e l'indicazione dei punteggi ottenuti sul display.

Soluzione



Il circuito di [figura 4.4b](#) è l'unione degli schemi del [progetto 3.A «Gara di riflessi»](#) e quello nella [figura 4.2b](#) di questo capitolo, relativa al collegamento tra Arduino e il display. Vista la sua lunghezza, suddividiamo lo sketch in quattro parti, evidenziando le modifiche apportate a quello scritto nel [capitolo 3](#).

[Figura 4.4 «Gara di riflessi» con display: a\) disposizione delle scritte sul display LCD; b\) circuito completo di display.](#)



- Prima del setup includiamo e inizializziamo la libreria `LiquidCrystal.h`, con le direttive:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

- Definiamo anche la costante `puntMax`, che rappresenta il punteggio da raggiungere per vincere la gara.

Il codice dello sketch (parte prima)

```
#include <LiquidCrystal.h> // include la libreria LiquidCrystal.h
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // inizializza la libreria con i numeri dei pin di interfaccia

const int pinPA=13; // pin 13 pulsante concorrente A
const int pinPB=0; // pin 0 pulsante concorrente B
const int pinPstart=8; // pin 8 pulsante start
const int pinBuzzer=9; // pin 9 Buzzer
const int pinLA=10; // pin 10 LED concorrente A
const int pinLB=6; // pin 6 LED concorrente B
const int pinL Verde=7; // pin 7 LED verde
const int puntMax=5; // punti da raggiungere per la vittoria della gara
bool PA; // stato del pulsante A
bool PB; // stato del pulsante B
bool Pstart; // stato del pulsante Start
int tottotoLA=0; // contatore vittorie concorrente A
int tottotoLB=0; // contatore vittorie concorrente B
int nota; // frequenza della nota del vincitore
```



Nella seconda parte dello sketch (setup):

- `Serial.begin(9600)` va sostituito con `lcd.begin(16, 2)`;
- scriviamo sul display i caratteri che rimangono fissi durante il loop:

```
Winner
A=
B=
```

Il codice dello sketch (parte seconda)

```
void setup(){
  lcd.begin(16, 2); // imposta il numero di colonne e righe del display
  pinMode(pinPA, INPUT_PULLUP); // pin digitale 13 input (pull-up interno)
  pinMode(pinPB, INPUT_PULLUP); // pin digitale 0 input (pull-up interno)
  pinMode(pinPstart, INPUT_PULLUP); // pin digitale 8 input (pull-up interno)
  pinMode(pinBuzzer, OUTPUT); // pin digitale 9 output (Buzzer)
  pinMode(pinLA, OUTPUT); // pin digitale 10 output
  pinMode(pinLB, OUTPUT); // pin digitale 6 output
  pinMode(pinL Verde, OUTPUT); // pin digitale 7 output
```



```
randomSeed(analogRead(0)); // inizializza la funzione random, leggendo del rumore
// sul pin collegato analog input 0
// predisponi le scritte fisse sul display
lcd.setCursor(0, 0); // imposta il cursore su colonna 0, riga 0
lcd.print("Winner"); // scrivi "Winner" all'inizio della prima riga
lcd.setCursor(9, 0); // imposta il cursore su colonna 9, riga 0
lcd.print("A="); // scrivi "A="
lcd.setCursor(9, 1); // imposta il cursore su colonna 9, riga 1
lcd.print("B="); // scrivi "B="
```

Nella terza parte dello sketch:

- il loop, che realizza il ciclo di ogni singola prova, non subisce modifiche; in vari punti si chiama la funzione `vincitore` passandole, tra parentesi, il pin del LED del vincitore (`pinLA` o `pinLB`);
- per chiarezza si è scritto `while(Pstart==1)`, ma sarebbe stato sufficiente scrivere `while(Pstart)`.

Il codice dello sketch (parte terza)

```
void loop(){
  digitalWrite(pinL Verde, HIGH); // accendi il LED verde

  do {
    Pstart = digitalRead(pinPstart); // leggi il pulsante Start
  } while(Pstart==1); // finché Start non è premuto

  digitalWrite(pinL Verde, LOW); // spegni il LED verde
  delay(3000, 8000); // attendi un tempo casuale tra 3 s e 8 s
  PA = digitalRead(pinPA); // leggi il pulsante PA
  PB = digitalRead(pinPB); // leggi il pulsante PB
  tone(pinBuzzer, 262); // suona la nota Do (262 Hz)
  digitalWrite(pinL Verde, HIGH); // riaccendi il LED verde

  if(PA==0 || PB==0){ // se uno dei due concorrenti ha premuto in anticipo
    if(PA==0 && PB==0) vincitore(pinLB); // se PA è già premuto vince B (chiama la function "vincitore")
    if(PA==1 && PB==0) vincitore(pinLA); // se PA è già premuto vince A (chiama la function "vincitore")
    if(PA==0 && PB==0){ // se entrambi hanno premuto in anticipo
      lcd.setCursor(2, 1); // imposta il cursore su colonna 2, riga 1
      lcd.print("NULL"); // scrivi che la gara è nulla
      delay(4000); // aspetta 4 secondi
      lcd.setCursor(2, 1); // imposta il cursore su colonna 2, riga 1
      lcd.print(" "); // cancella NULL
    }
  }
  else { // se è tutto regolare aspetta la prima pressione
    do {
      PA = digitalRead(pinPA); // leggi il pulsante PA
      PB = digitalRead(pinPB); // leggi il pulsante PB
    } while(PA && PB); // finché non viene premuto un pulsante

    if(PA==0) vincitore(pinLA); // se PA è premuto vince A (chiama la function "vincitore")
    if(PB==0) vincitore(pinLB); // se PB è premuto vince B (chiama la function "vincitore")
    noTone(pinBuzzer); // spegni il buzzer
  }
}
```

Nella quarta parte dello sketch:

- la function vincitore riceve il valore del pin del concorrente vincitore (pinLA o pinLB) e l'associa alla variabile locale winner;
- con `lcd.setCursor()` posiziona il cursore e con `lcd.print()` scrive il vincitore (A o B) e i punteggi (`totWinA` e `totWinB`); poi la function verifica se uno dei due ha raggiunto i 10 punti e, nel caso, esegue 10 lampaggi del LED del display, alterna le note RE e LA e poi ripristina il display per una nuova gara.

Il codice dello sketch (parte quarta)

```
void vincitore (int winner){ // FUNCTION che esegue le operazioni relative al
                           // vincitore e gestisce il display LCD e il buzzer
    digitalWrite(pinVerde, LOW); // spegni il LED verde

    if(winner==pinLA){          // se ha vinto A
        lcd.setCursor(3, 1);    // imposta il cursore su colonna 3, riga 1
        lcd.print("A");         // scrivi "A" (vincitore)
        totWinA++;              // incrementa i punti di A
        nota=446;               // assegna alla variabile nota il valore 446 (LA)
    }
    if(winner==pinLB){          // se ha vinto B
        lcd.setCursor(3, 1);    // imposta il cursore su colonna 3, riga 1
        lcd.print("B");         // scrivi "B" (vincitore)
        totWinB++;              // incrementa i punti di B
        nota=587;               // assegna alla variabile nota il valore 587 (RE)
    }

    //scrivi sul display LCD i punti totali di A e di B
    lcd.setCursor(11, 0);       // imposta il cursore su colonna 11, riga 0
    lcd.print(totWinA);         // scrivi i punti di A
    lcd.setCursor(11, 1);       // imposta il cursore su colonna 11, riga 1
    lcd.print(totWinB);         // scrivi i punti di B

    if(totWinA==puntMax || totWinB==puntMax){ // se uno dei due ha vinto la gara
                                                // (ha raggiunto il puntMax=5)
        for(int cont=1; cont<=10; cont++){ // per 10 volte
            tone (pinBuzzer, 587);        // suona la nota RE
            digitalWrite(winner, HIGH);   // accendi il LED di A
            lcd.noDisplay();             // spegni il display
            delay(200);                 // aspetta 200 ms
            tone (pinBuzzer, 446);        // suona la nota LA
            digitalWrite(winner, LOW);   // spegni il LED di B
            lcd.display();              // accendi il display
            delay(200);                 // aspetta 200 ms
        }

        // predisponi il display per una nuova gara
        lcd.setCursor(3, 1);           // imposta il cursore su colonna 3, riga 1
        lcd.print(" ");               // cancella il vincitore
        lcd.setCursor(11, 0);          // imposta il cursore su colonna 11, riga 0
    }
}
```



```
lcd.print("0");           // azzerza i punti di A
lcd.setCursor(11, 1);     // imposta il cursore su colonna 11, riga 1
lcd.print("0");           // azzerza i punti di B
// azzerza i contatori dei punti di A e di B
totWinA=0;
totWinB=0;
}

else{                     // altrimenti, se nessuno è ancora arrivato a puntMax
    for(int cont =1; cont<=5; cont++){ // per 5 volte
        tone (pinBuzzer, nota);      // suona la nota del vincitore
                                         // della prova (LA o RE)
        digitalWrite(winner, HIGH);   // accendi il LED del vincitore
        delay(500);                 // aspetta 500 ms
        noTone (pinBuzzer);          // interroghi la nota
        digitalWrite(winner, LOW);   // spegni il LED del vincitore
        delay(500);                 // aspetta 500 ms
    }
}
}
```

Con l'aggiunta delle nuove specifiche, lo sketch «Gara di riflessi» ha superato le 130 righe di codice; nonostante ciò, mantenendo un certo ordine e scrivendo commenti esplicativi, non è difficile seguire la logica dell'algoritmo.

Lo sketch non è ancora finito: nel [progetto 4.8](#) a fine capitolo aggiungeremo le misure dei tempi di reazione e la memorizzazione del record.

4.3 Le istruzioni per misurare il tempo

Fino ad ora abbiamo visto solo un'istruzione di Arduino che ha a che fare con il tempo:

```
delay(ms);           // l'esecuzione dello sketch viene posta
                     // in pausa per ms millisecondi
```

Per pause più brevi si può usare:

```
delayMicroseconds(us); // pausa di us microsecondi.
```

Per misurare il tempo è utile la funzione:

```
millis();           // restituisce il numero di millisecondi
                     // trascorsi da quando lo sketch è stato
                     // caricato su Arduino. Il conteggio
                     // riparte da a zero dopo circa 50 giorni.
```

Per misurare l'intervallo di tempo tra due eventi si può quindi leggere `millis()` in due tempi successivi e calcolare la differenza:

```
timel = millis(); // tempo dell'evento 1
...
time2 = millis(); // tempo dell'evento 2
```

```
deltaT = time2 - time1; // intervallo tra i due eventi
dove time1, time2 e deltaT devono essere dichiarate variabili di tipo
unsigned long (occupano 4 byte di memoria).
```

```
micros(); // è analoga a millis() ma restituisce
// i microsecondi
```

Per misurare la durata di un impulso ([figura 4.5](#)) si usa la funzione:

```
pulseIn(pin, valore); // restituisce la durata dello stato
// HIGH (se valore=HIGH)
// LOW (se valore=LOW)
// di un impulso sul digital input pin.
```

Il valore restituito è del tipo unsigned long.

Se un impulso non si completa nel tempo di 1 s (timeout di default), la funzione restituisce 0; è possibile specificare un differente timeout (in microsecondi) come terzo parametro. La funzione pulseIn() lavora con impulsi da 10 µs a 3 minuti.



Figura 4.5 Misura della durata di un impulso: a) a livello HIGH; b) a livello LOW.

4.4 Esercizio svolto

Misura di un intervallo di tempo con millis()

Misura l'intervallo di tempo che intercorre tra la pressione di due pulsanti (A e B) e rappresenta il valore in millisecondi sul Serial Monitor.

Soluzione

Le variabili TA, TB e deltaT sono dichiarate del tipo unsigned long (4 byte), perché destinate a memorizzare ed elaborare il valore fornito dalla funzione millis().

Il codice dello sketch (setup)

```
const int pinA=12; // pin del pulsante A
const int pinB=13; // pin del pulsante B
bool statoA; // stato del pulsante A
bool statoB; // stato del pulsante B
unsigned long TA; // tempo di A
unsigned long TB; // tempo di B
unsigned long deltaT; // intervallo tra A e B
```



void setup(){

```
pinMode(pinA, INPUT_PULLUP);
pinMode(pinB, INPUT_PULLUP);
Serial.begin(9600); // apri la porta seriale alla velocità di 9600 bps
}
```

Nel loop dello sketch (riportato di seguito) attendiamo, con un ciclo `do... while`, che venga premuto A e poi registriamo il tempo del timer interno con `TA=millis()`; lo stesso vale per B, poi calcoliamo la differenza dei tempi (`deltaT = TB-TA`).

Dopo aver scritto il risultato sul Serial Monitor, attendiamo che entrambi i pulsanti siano rilasciati per ricominciare il ciclo con una nuova prova.

Il codice dello sketch (loop)

```
void loop(){
  do {
    statoPA=digitalRead(pinA); // continua a leggere A
  } while(statoPA==1); // finché A non viene premuto

  TA=millis(); // memorizza il tempo TA

  do {
    statoPB=digitalRead(pinB); // continua a leggere B
  } while(statoPB==1); // finché B non viene premuto

  TB=millis(); // memorizza il tempo TB
  deltaT = TB-TA; // calcola l'intervallo di tempo tra A e B

  Serial.print("Intervallo di tempo:"); // scrivi sul Serial Monitor
  Serial.print(deltaT);
  Serial.println(" ms");

  while(!statoA||!statoB){ // continua a leggere lo stato dei pulsanti
    statoPA=digitalRead(pinA); // finché entrambi non sono rilasciati (HIGH)
    statoPB=digitalRead(pinB);
  }
}
```

4.5 Esercizio

Modifica l'[esercizio svolto 4.4](#) in modo da scrivere «Prova non valida» sul Serial Monitor se, alla pressione di A, il pulsante B è già premuto. Sostituisci poi il Serial Monitor con il display LCD.

4.6 Esercizio

Autovelox

Realizza un autovelox per misurare la velocità di un'automobilina, sul cui tetto è stato fissato un magnete (figura 4.6).

Suggerimento Puoi rivelare il passaggio del magnete e quindi dell'automobilina con due ampolle Reed, poste lungo il percorso. I contatti delle ampolle sostituiscono i pulsanti dell'esercizio svolto 4.4 e, con uno sketch analogo, si misura l'intervallo di tempo tra gli impulsi dei sensori; nota la distanza tra i sensori, si calcola la velocità dell'auto.

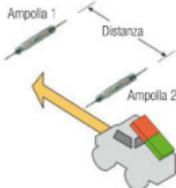


Figura 4.6 Disposizione del magnete e dei sensori per l'autovelox.

4.7 Esercizio svolto

Conta il numero di eventi in un intervallo di tempo, con millis()

Conta il numero di pressioni di un pulsante nell'intervallo di tempo di 5 secondi e scrivi il risultato sul Serial Monitor. L'intervallo di tempo del conteggio inizia alla prima pressione del pulsante ed è segnalato dall'accensione di un LED.

Soluzione

Il codice dello sketch



```
const int pinPuls=0; // pin del pulsante
const int pinLed=13; // pin del LED
const int interv=5; // durata dell'intervallo di conteggio in secondi
bool newPuls; // nuovo stato del pulsante
bool oldPuls=1; // vecchio stato del pulsante
unsigned long Tstart; // tempo di inizio intervallo di conteggio
int cont=0; // inizializza il conteggio a zero

void setup(){
    pinMode(pinPuls, INPUT_PULLUP);
    pinMode(pinLed, OUTPUT);
    Serial.begin(9600); // apri la porta seriale alla velocità di 9600 bps
}

void loop(){
    do {
        newPuls=digitalRead(pinPuls); // leggi lo stato del pulsante
        while(newPuls==1); // finché non viene premuto
    } while(newPuls==1); // leggi lo stato del pulsante
    if (newPuls==0 && oldPuls==1){ // se il pulsante viene premuto (passaggio da 1 a 0)
        cont++; // incrementa il contatore
    }
    oldPuls=newPuls; // aggiorna il valore di oldPuls
}

void loop(){
    if (cont>0) { // se il contatore è maggiore di zero
        digitalWrite(pinLed, HIGH); // accendi il LED
        delay(1000); // attendi 1000 ms (1 secondo)
        digitalWrite(pinLed, LOW); // spegni il LED
        Serial.print("Numero di pressioni in "); // scrivi sul Serial Monitor
        Serial.print(interv);
        Serial.print(" secondi: ");
        Serial.println(cont);
    }
}
```

```
Tstart=millis(); // alla prima pressione memorizza il timer in Tstart
digitalWrite(pinLed, HIGH); // accendi il LED

while(millis()-Tstart <= (interv*1000)){ // finché non è terminato l'intervallo di conteggio
    newPuls=digitalRead(pinPuls); // leggi lo stato del pulsante
    delay(10);
    if(newPuls==0 && oldPuls==1){ // se il pulsante viene premuto (passaggio da 1 a 0)
        cont++; // incrementa il contatore
    }
    oldPuls=newPuls; // aggiorna il valore di oldPuls
}

digitalWrite(pinLed, LOW); // spegni il LED
Serial.print("Numero di pressioni in "); // scrivi sul Serial Monitor
Serial.print(interv);
Serial.print(" secondi: ");
Serial.println(cont);

do {
    newPuls=digitalRead(pinPuls); // leggi lo stato del pulsante
} while(newPuls==0); // esci dal loop quando il pulsante è rilasciato

cont=0; // azzera il contatore
oldPuls=1; // inizializza oldPuls
delay(2000); // attendi 2 secondi per l'inizio di una nuova prova
}
```

Nello sketch (riportato sopra) il loop ha il seguente funzionamento:

- dopo il ciclo `do...while`, che attende la prima pressione del pulsante, viene letto il `timer` con l'istruzione `millis()`, e viene memorizzato in `Tstart`, che rappresenta il tempo d'inizio dell'intervallo; si accende anche il LED;
- il ciclo `while` che segue ha una durata pari a `interv*1000=5000 ms`; infatti il ciclo viene eseguito fino a quando la lettura del timer non eccede `Tstart di 5 s`;
- nel ciclo `while` si legge lo stato del pulsante e lo si memorizza in `newPuls`; il ritardo di 10 ms che segue la lettura serve a fare esaurire eventuali rimbalzi dei contatti, che potrebbero falsare il conteggio;
- il contatore `cont` viene incrementato solo se (if) la nuova lettura del pulsante è «premuto» (`newPuls==0`) e (`&&` = AND) la vecchia lettura era «rilasciato» (`oldPuls == 1`), cioè se siamo nella transizione tra rilasciato e premuto;
- alla fine del ciclo l'ultima lettura del pulsante viene caricata su `oldPuls` (`oldPuls=newPuls`) e si ricomincia con una nuova lettura del pulsante.

La chiusura di un contatto e, in genere, seguita da una serie di rimbalzi (bounce) che si esauriscono in circa 10 ms. Dopo la prima chiusura del contatto, si ferma per 10 ms l'esecuzione dello sketch con l'istruzione `delay(10)`, per evitare di contare anche i rimbalzi.

Si noti che la precisione della durata dell'intervallo di conteggio dipende dal tempo di esecuzione del ciclo `while`; nel nostro caso, dovendoci confrontare

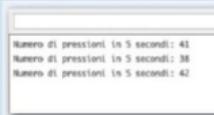


Figura 4.7 Alcuni risultati riportati sul Serial Monitor.

con tempi «umani» questo non compromette il risultato della misura. Una soluzione che garantisce una risposta migliore è quella dell'**interrupt**, che verrà presentata nel [paragrafo 4.4](#) e applicata nel [progetto 4.C](#).

Nella [figura 4.7](#) si vedono i risultati di tre prove visualizzati dal Serial Monitor.

4.8 Esercizio

Misura della velocità di un trillo

Per valutare la velocità di esecuzione di un trillo al pianoforte, misura il numero di pressioni alternate di due pulsanti (A e B) nell'intervallo di tempo di 4 s e scrivi il risultato sul Serial Monitor.

Suggerimento Modifica l'[esercizio svolto 4.7](#), incrementando il contatore solo se viene eseguita la sequenza corretta: A premuto - B rilasciato, A rilasciato - B rilasciato, B premuto - A rilasciato, A rilasciato - B rilasciato.

4.4 La gestione degli interrupt

Pensate se, mentre siamo in casa, a intervalli regolari di tempo dovessimo andare ad aprire la porta per verificare se fuori c'è qualcuno che ha bisogno di noi; nella maggior parte dei casi sarebbe tempo perso.

Questo però è simile a ciò che abbiamo fatto fino ad ora, con strutture tipo:

```
do{
    statoPuls = digitalRead(pinPuls); // continua a leggere il pulsante
} while(statoPuls); // finché non viene premuto
```

In pratica continuiamo a leggere un ingresso, o più ingressi, finché questi non cambiano stato; questa gestione delle periferiche in informatica è detta **polling**.

Se il microcontrollore non ha tanti eventi da gestire contemporaneamente, come nella maggior parte degli esempi e progetti del testo, la gestione con il **polling** non comporta problemi, soprattutto se i tempi sono riferiti a quelli umani o a sistemi lenti e dotati di inerzia, come motori, riscaldatori, ecc.

In alcuni casi però potrebbe succedere che, mentre il microcontrollore è occupato a eseguire alcune istruzioni, un breve impulso su un pin venga perso perché in quell'intervallo di tempo non lo si è letto.

Tornando alla nostra metafora iniziale, è come se, tra un controllo della porta e l'altro, qualcuno fosse arrivato e poi ripartito perché aveva trovato la porta chiusa; è molto più efficiente mettere un campanello, rimanere occupati nelle nostre faccende e andare ad aprire la porta solo quando sentiamo suonare il campanello.

Una gestione di questo tipo in informatica è detta **interrupt**:

- all'arrivo dell'**interrupt**, l'esecuzione dello sketch è interrotta;
- viene eseguita la funzione (**function**) che specifica cosa fare in quel caso; tale funzione è detta **ISR (Interrupt Service Routine, o routine di servizio dell'interrupt)**. Una volta eseguita la **function**, l'esecuzione dello sketch viene ripresa dal punto in cui era stata interrotta.

Per Arduino Uno l'**interrupt** può nascere all'interno del microcontrollore o provenire dall'esterno:

➤ **Interrupt interni**: tra le varie sorgenti di interrupt interno ci sono i tre timer (Timer 0, Timer 1 e Timer 3) contenuti nel microcontrollore. Un timer di Arduino ha un funzionamento simile a quello di un timer usato in cucina per i tempi di cottura: viene programmato impostando un certo intervallo di tempo e poi avvisa il programma in esecuzione, con un interrupt, che il tempo programmato è esaurito. Le funzioni **delay()** e **millis()** usano il Timer0, la libreria **Servo** il Timer1, la funzione **tone()** il Timer2.

➤ **Interrupt esterni**: sono segnali applicati ai digital pin 2 o 3 di Arduino, provenienti da dispositivi esterni. L'istruzione principale per gestire gli interrupt esterni su Arduino Uno è:

attachInterrupt(interrupt, function, mode)

dove:

- **interrupt**: numero dell'**interrupt** (0 per il digital pin 2, 1 per il digital pin 3);
- **function**: nome della funzione da richiamare quando si verifica l'**interrupt** (nessun parametro viene fornito o restituito);
- **mode**: definisce la modalità di riconoscimento:
LOW: quando il pin è LOW;
CHANGE: quando il pin cambia valore;
RISING: sul fronte di salita (LOW→HIGH);
FALLING: sul fronte di discesa (HIGH→LOW).

Altre istruzioni per gli **interrupt** sono:

- **noInterrupts()**: disabilita tutti gli **interrupt**; si usa nei punti dello sketch che non devono essere interrotti.
- **detachInterrupt(interrupt)**: disabilita l'**interrupt** 0 o 1.
- **interrupts()**: riabilita tutti gli **interrupt** disabilitati; all'avvio dello sketch gli **interrupt** sono abilitati per default.

Le variabili modificate all'interno di una function di servizio di un **interrupt** devono essere dichiarate di tipo **volatile**, per esempio dichiarando:

volatile int cont = 0;

La variabile intera **cont**, inizializzata a zero, potrà essere modificata dentro

una function richiamata da un interrupt.

Quando viene eseguita la function di servizio di un interrupt, gli altri interrupt vengono disabilitati; quindi è bene che tale function sia breve e veloce.

La disabilitazione degli interrupt comporta che le funzioni che utilizzano gli interrupt interni, come `delay()`, `millis()` e `micros()`, non possano essere utilizzate all'interno di una function di servizio; fa eccezione `delayMicroseconds()`, che non usa interrupt.

Da ciò si deduce che le istruzioni:

```
noInterrupts()
detachInterrupt(interrupt)
interrupts()
```

vanno usate in modo consapevole, per evitare malfunzionamenti imprevisti.

4.9 Esercizio svolto

Gestione di un interrupt

Conta il numero delle pressioni di un pulsante e scrivi il risultato sul Serial Monitor.

Soluzione

Il codice dello sketch

```
volatile int cont=0; // cont è la variabile contatore del numero di pressioni;
// è dichiarata "volatile" perché modificata all'interno
// della function conta(), al servizio a un interrupt

void setup(){
pinMode(2, INPUT_PULLUP); // digital pin 2 INPUT, con pull-up interno
attachInterrupt(0, conta, FALLING); // usiamo l'interrupt 0 che è associato al digital pin 2
// il passaggio HIGH->LOW sul pin 2 fa eseguire la function "conta"
Serial.begin(9600);
}

void loop(){ // il loop è vuoto, si attendono gli interrupt sul pin 2
}

void conta(){
cont++; // incrementa il conteggio
Serial.print("Conteggio: "); // scrivi il valore di cont sul Serial Monitor
Serial.println(cont);
}
```

Nello sketch di questo esempio puoi osservare che:

- la funzione `conta`, che incrementa il contatore `cont`, viene eseguita quando l'ingresso digitale 2 (interrupt 0), collegato al pulsante, passa da HIGH a LOW (FALLING);

- una volta incrementato `cont`, il suo valore è scritto sul Serial Monitor;
- la variabile `cont` è dichiarata `volatile` perché modificata dentro la function `conta` dell'servizio dell'interrupt.
- La funzione `loop()` è vuota: è solo un ciclo in attesa della pressione del pulsante (interrupt), che provoca l'esecuzione di `conta`.

Progettare con ARDUINO

4.8 «Gara di riflessi» completa

Completiamo finalmente il progetto «Gara di riflessi», iniziato nel [capitolo 3](#) e arricchito con il display nel [progetto 4.A](#) di questo capitolo, aggiungendo le misure dei tempi; sul display devono apparire:

- il vincitore e il tempo della prova in millisecondi;
- i punteggi dei due concorrenti;
- il record di tempo (in ms) da quando lo sketch è stato avviato.

Il display in [figura 4.8](#), per esempio, segnala che ha vinto l'ultima prova il concorrente A con un tempo di 140 ms, i punteggi sono A=6 e B=4, il record di tempo è 128 ms.

Soluzione

L'hardware del sistema rimane quello nella [figura 4.6b](#) del [progetto 4.A](#).

Vista la lunghezza raggiunta dallo sketch, elenchiamo solo le modifiche apportate per effettuare la misura del tempo, la memorizzazione del record e la visualizzazione sul display.

Prima del `setup()` dichiariamo tre nuove variabili, destinate a memorizzare:

- `Tstart`: il valore del timer nell'istante in cui si accende il LED verde e suona il buzzer;
- `RitVinc`: il tempo di ritardo, rispetto a `Tstart`, con cui il vincitore preme il pulsante;
- `RitRecord`: il ritardo minimo registrato nelle prove e nelle gare precedenti (record).

Poiché tali tempi sono valori restituiti dalla funzione `millis()`, le variabili sono dichiarate di tipo `unsigned long` (4 byte).

Il codice dello sketch (dichiarazioni)

```
unsigned long Tstart; // tempo di start
unsigned long RitVinc; // ritardo del vincitore
unsigned long RitRecord=1000; // ritardo record
```

Nel `setup()` cambia la disposizione delle scritte fisse sul display ([figura 4.9](#)).

Il codice dello sketch (setup)

```
// predisporre le scritte fisse sul display
lcd.setCursor(0, 0); // imposta il cursore su colonna 0, riga 0
```



Figura 4.8 Disposizione delle scritte sul display per la Gara di riflessi.

```

lcd.print("Winner");
lcd.setCursor(7, 0); // scrivi "Winner" all'inizio della prima riga
lcd.print("A");
lcd.setCursor(7, 1); // imposta il cursore su colonna 7, riga 1
lcd.print("B");
lcd.setCursor(12, 0); // imposta il cursore su colonna 12, riga 0
lcd.print("Rec");
// scrivi "Rec"

```

Nel `loop()` completo, le righe contrassegnate con `****` contengono le nuove istruzioni, rispetto al [progetto 4.A](#), inserite per la misura del tempo.

Il codice dello sketch (loop completo)

```

void loop() {
    digitalWrite(pinverde, HIGH); // accendi il LED verde
    do {
        Pstart = digitalRead(pinPstart); // leggi il pulsante Start
        } while (!Pstart); // finché Start non è premuto

    digitalWrite(pinverde, LOW); // spegni il LED verde
    delay(5000, 8000); // attendi un tempo casuale tra 3 s e 8 s
    Tstart=millis(); // salva il timer nella variabile Tstart ****
    PA = digitalRead(pinPA); // leggi il pulsante PA
    PB = digitalRead(pinPB); // leggi il pulsante PB
    tone (pinbuzzer, 262); // suona la nota Do (262 Hz)
    digitalWrite(pinverde, HIGH); // riaccendi il LED verde

    if(PA==0 || PB==0) { // se uno dei due concorrenti ha premuto in anticipo
        RitVincitore(); // azzerza il tempo del vincitore per segnalare l'irregolarità ****
        if(PA==0 && PB==1) vincitore(pinA); // se PA è già premuto vince B (chiama la function "vincitore")
        if(PA==1 && PB==0) vincitore(pinA); // se PB è già premuto vince A (chiama la function "vincitore")
        if(PA==0 && PB==0){ // se entrambi hanno premuto in anticipo la gara è nulla
            lcd.setCursor(2, 1); // imposta il cursore su colonna 2, riga 1
            lcd.print("NULL"); // scrivi che la gara è nulla
            delay(4000); // aspetta 4 secondi
            lcd.setCursor(2, 1); // imposta il cursore su colonna 2, riga 1
            lcd.print(" "); // cancella NULL
        }
    } else { // se è tutto regolare aspetta la prima pressione
        do {
            PA = digitalRead(pinPA); // leggi il pulsante PA
            PB = digitalRead(pinPB); // leggi il pulsante PB
            } while(PA && PB); // finché non viene premuto un pulsante

        RitVincitore();-Tstart; // calcola il tempo di ritardo del vincitore ****
        if(RitVincitore<RitRecord) RitRecord=RitVinc; // se il record è battuto sostituisce ****
        if(PA==0) vincitore(pinA); // se P1 è premuto vince A
        if(PB==0) vincitore(pinB); // se P2 è premuto vince B
    }
    noTone (pinbuzzer);
}

```

Analizziamo ora in dettaglio il codice di questo sketch:

- dopo la pressione del pulsante `Pstart` e lo spegnimento del LED verde, cancelliamo il risultato della prova precedente scrivendo 6 spazi all'inizio della seconda riga, con `lcd.print(" ")`;
- dopo il delay casuale da 3 a 8 secondi, contemporaneamente all'accensione del LED verde e del suono del buzzer, registriamo il timer nella variabile `Tstart`, con l'istruzione `Tstart=millis();`;
- appena si rileva una pressione regolare di un pulsante, viene memorizzato il suo ritardo rispetto a `Tstart` con l'istruzione `RitVinc=millis()-Tstart;`
- nella riga successiva, se `RitVinc` è inferiore al record (`RitRecord`), questo è rimpiazzato dal nuovo tempo.

Rispetto alla funzione `vincitore` del [progetto 4.A](#), le modifiche riguardano solo la scrittura sul display:

- variano leggermente le posizioni di scrittura della lettera (A o B) del vincitore e dei due punteggi;
- i valori di `RitVinc` e `RitRecord` vanno scritti nelle posizioni specificate dalla [figura 4.8](#).

Lo sketch completo è disponibile online, sul sito del libro.

Progettare con ARDUINO

4.C Fidget Spinner Game

Progetta un sistema per realizzare sfide di Fidget Spinner con le regole seguenti:

- entrambi i concorrenti avviano i propri Fidget Spinner e li pongono contemporaneamente in mezzo a due forcille ottiche, per misurare la velocità ([figura 4.9](#)); i passaggi delle eliche sono rivelati mediante l'interruzione del fascio di luce emesso da un LED e ricevuto da un fototransistor;

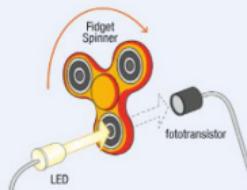


Figura 4.9 «Fidget Spinner Game»: la soluzione costruttiva.

- quando i Fidget Spinner sono in posizione si preme il pulsante `Start` per l'avvio della misura; ciò provoca anche l'accensione del LED verde;
- dopo 4 secondi, effettuata la misura delle velocità di rotazione (in giri/minuto), il LED verde si spegne e il LED rosso del vincitore lampeggia, contemporaneamente al suono di un buzzer;

Record da batteria= 995 giri/minuto
Velocità A= 650 giri/minuto
Velocità B= 0 giri/minuto
MA VINTO A
Punti A: 3
Punti B: 0

Record da batteria= 995 giri/minuto
Velocità A= 988 giri/minuto
Velocità B= 0 giri/minuto
MA VINTO A
Punti A: 4
Punti B: 0

Record da batteria= 995 giri/minuto
Velocità A= 1130 giri/minuto
Velocità B= 0 giri/minuto
MA VINTO A
NUOVO RECORD!
Punti A: 5
Punti B: 0

Vincitore della gara: A
NUOVA GARA

Figura 4.10 «Fidget Spinner Game»: i risultati di una gara visualizzati su Serial Monitor. Per provare il prototipo si è utilizzato solo il pin d'ingresso 2 (concorrente A).

- i risultati sono visualizzati sul Serial Monitor (**figura 4.10**): le velocità dei concorrenti in giri al minuto, il vincitore (A o B), il numero totale delle vittorie dei due concorrenti nella gara e il record di velocità da quando lo sketch è stato lanciato;
- la gara termina quando uno dei due raggiunge il punteggio 5; in questo caso il display visualizza la scritta «Vincitore della gara» seguita dalla lettera del vincitore, mentre il buzzer alterna due note per 10 volte;
- prima di iniziare una nuova gara, si visualizza la scritta «NUOVA GARA».

Soluzione

Analizzando le specifiche del progetto noterai varie analogie con la «Gara di riflessi» (**progetto 4.A**): due concorrenti con due LED, due segnali d'ingresso da misurare, il pulsante Start con il LED verde, il buzzer che suona sincrono con il lampeggio dei LED, i dati da visualizzare (vincitore, punti e velocità).

Per ora la visualizzazione avviene sul Serial Monitor: nell'**esercizio 4.10** occorrerà visualizzare i dati su un display LCD.

L'hardware

- Dal punto di vista circuituale, il sistema è praticamente identico a quello del **progetto 4.A -Gara di riflessi-**; l'unica differenza è che i segnali d'ingresso sono prelevati da due forcille ottiche invece che dai pulsanti dei concorrenti; ogni forcilla invia ad Arduino un impulso a ogni passaggio di un'elica del Fidget Spinner.
- È necessario chiudere i fori delle eliche, per evitare che il loro passaggio falsi la misura.
- La forcilla ottica può essere realizzata affacciando un LED sempre acceso a un fototransistor (**figura 4.11**): il fototransistor ha tre pin, ma la base (B) non si utilizza perché la sua funzione è sostituita dalla luce che colpisce la lente d'ingresso.

Quando il fototransistor è illuminato (assenza dell'elica) la V_o è circa zero, perché il circuito tra C e D è chiuso, e quindi il pin 2 di Arduino

Figura 4.11 Forcella ottica con fototransistor: a) schema elettrico; b) pinout del fototransistor 2BS0; c) montaggio del prototipo su breadboard.



risulta LOW; quando l'elica mette in ombra il fototransistor, la V_o va verso i 5 V, perché tra C ed E il circuito è aperto, e il pin 2 di Arduino risulta HIGH. Naturalmente sul pin 3 è collegato il circuito identico del concorrente B. Per avere la migliore precisione di lettura è necessario che il fascio di luce e il fototransistor siano perfettamente allineati.

Il software

A differenza degli sketch precedenti, in cui la lettura degli ingressi veniva effettuata con la tecnica del polling, mediante `digitalRead()`, in questo progetto sceglieremo di usare l'interrupt, vista la maggiore velocità con cui si presentano gli impulsi e il fatto che arrivano contemporaneamente, ma in modo asincrono, su due ingressi.

Quando le forcille ottiche **Sensa A** e **Sensa B** (collegate ai pin 2 e 3 abilitati agli interrupt) rivelano il passaggio di un'elica, l'interrupt che ne deriva provoca l'avvio di una function (**contaA** o **contaB**) che incrementa il contatore corrispondente (**contA** o **contB**).

Prima del setup, tra le dichiarazioni iniziali spiccano quelle dei due contatori (**contA** e **contB**), inizializzati a zero, che sono dichiarati **volatile** in quanto modificati all'interno delle function di servizio all'interrupt.

Il codice dello sketch (dichiarazioni e setup)

```
const int pinSensA=2; // pin del sensore A (interrupt 0)
const int pinSensB=3; // pin del sensore B (interrupt 1)
const int pinLedA=13; // pin del LED A
const int pinLedB=12; // pin del LED B
const int pinLedVerde=11; // pin del LED Verde (misura in atto)
const int pinStart=8; // pin del pulsante Start
const int pinBuzzer=9; // pin del buzzer
const int tempoMis=4000; // durata del tempo di misura in millisecondi (4 secondi)
const int puntNess=5; // punti da raggiungere per la vittoria della gara
bool Pstart; // stato del pulsante Start
int totVittA=0; // contatore vittorie concorrente A
int totVittB=0; // contatore vittorie concorrente B
int Record=0; // variabile per il record
unsigned long Tstart; // tempo di inizio intervallo di conteggio
volatile int contaA=0; // contatore passaggi sensore A (modificato nella function "conta")
volatile int contaB=0; // contatore passaggi sensore B (modificato nella function "conta")
```

void setup()

```
pinMode(pinSensA, INPUT);
pinMode(pinSensB, INPUT);
pinMode(pinStart, INPUT_PULLUP);
pinMode(pinLedA, OUTPUT);
pinMode(pinLedB, OUTPUT);
pinMode(pinLedVerde, OUTPUT);
pinMode(pinBuzzer, OUTPUT);
```

```

attachInterrupt(0, contaA, FALLING); // interrupt 0 associato al digital pin 2; il passaggio
// HIGH->LOW sul pin 2 chiama la function "contaA"
attachInterrupt(1, contaB, FALLING); // interrupt 1 associato al digital pin 3; il passaggio
// HIGH->LOW sul pin 3 chiama la function "contaB"

Serial.begin(9600); // apri la porta seriale alla velocità di 9600 bps
}

```

Nel `setup()`, le istruzioni `attachInterrupt()` specificano le modalità con cui vengono riconosciuti e serviti i due interrupt.

Nel `loop()`:

- dopo aver premuto Start, viene memorizzato il tempo con `Tstart=millis()`; poi inizia un ciclo `while` che dura 4 secondi, in cui si ricevono gli interrupt dai sensori, ognuno dei quali provoca, nella funzione di servizio, l'incremento del relativo contatore (`contA` o `contB`);
- terminato il tempo di misura poniamo `Pstart=1`, con funzione di flag: per evitare che vengano contati anche i successivi passaggi delle eliche, all'interno delle funzioni di servizio agli interrupt incrementiamo i contatori solo se `Pstart=0`. In alternativa si potrebbero disabilitare gli interrupt, facendo attenzione a non mettere fuori uso alcune funzioni come `tone` o `delay`, che utilizziamo più avanti.
- stampiamo poi i risultati: in particolare il numero di giri al minuto si ricava moltiplicando per 5 il numero delle eliche contate in 4 secondi (`contA` o `contB`): infatti `contA/3` dà il numero di giri di A in 4 secondi; moltiplicando per 15 ho il numero di giri al minuto. Quindi

$$(contA/3) \cdot 15 = contA \cdot 5$$

dà il numero di giri di A al minuto.

- Dal confronto tra `contA` e `contB` si deduce il vincitore, di cui viene incrementato il contatore dei punti (`totWin`).
- La nota del buzzer dipende dal vincitore (440 Hz o 587 Hz).
- Confrontando i contatori dei punti (`totWin`) con la costante `puntMax`, capiamo se uno dei due ha vinto la gara, nel qual caso si scrive «Vincitore della gara» seguito dalla lettera del vincitore, poi un ciclo `for` suona un buzzer per 10 volte e fa lampeggiare il LED rosso del vincitore.
- Alla fine azzeriamo i contatori dei punti, scriviamo «NUOVA GARA» e il loop ricomincia per una nuova gara.

Il codice dello sketch (loop)

```

void loop(){
  digitalWrite(pinLedVerde, LOW); // spegni il LED verde

  do {
    Pstart=digitalRead(pinStart); // leggi il pulsante start (Pstart è usata anche come flag)
  } while(Pstart); // finché non viene premuto
}

```



```

Tstart=millis(); // memorizza il timer in Tstart
digitalWrite(pinLedVerde, HIGH); // accendi il LED verde

while(millis()-Tstart < tempoMs){ // finché non è terminato il tempo di misura, attendi
  // gli impulsi di A e di B sui pin 2 e 3 (interrupt)

  Pstart=1; // poni a 1 la flag Pstart che segnala alle fuction che il tempo di misura è finito
  digitalWrite(pinLedVerde, LOW); // spegni il LED verde

  Serial.print("Record da battere= "); // scrivi il record da battere
  Serial.print("Record");
  Serial.println(" giri/minuto");

  // scrivi sul Serial Monitor le velocità di A e di B
  Serial.print("Velocità A= "); // contaA = numero di passaggi delle eliche di A in 4 secondi
  Serial.print(contA*5); // n° giri al minuto di A = (contA/3)*15=contA*5
  Serial.println(" giri/minuto");

  Serial.print("Velocità B= "); // contaB = numero di passaggi delle eliche di B in 4 secondi
  Serial.print(contB*5); // n° giri al minuto di B
  Serial.println(" giri/minuto");

  if(contA>contB){ // se ha vinto A
    Serial.println("HA VINTO A"); // scrivi che ha vinto A
    digitalWrite(pinLedA, HIGH); // accendi il LED A
    totWin++; // incrementa i punti di A
    if((contA*5) > Record){ // se il record è battuto
      Serial.println("NUOVO RECORD!"); // scrivi "Nuovo Record!" sul S. M.
      Record = (contA*5); // aggiorna il record
    }
    tone(pinBuzzer, 440, 5000); // suona la nota La sul buzzer per 5 secondi
    delay(5000);
    digitalWrite(pinLedA, LOW); // spegni il LED A
  }
  else{ // altrimenti, se ha vinto B
    Serial.println("HA VINTO B"); // scrivi che ha vinto B
    digitalWrite(pinLedB, HIGH); // accendi il LED B
    totWin++; // incrementa i punti di B
    if((contB*5) > Record){ // se il record è battuto
      Serial.println("NUOVO RECORD!"); // scrivi "Nuovo Record!" sul S. M.
      Record = (contB*5); // aggiorna il record
    }
    tone(pinBuzzer, 587, 5000); // suona la nota Re sul buzzer per 5 secondi
    delay(5000);
    digitalWrite(pinLedB, LOW); // spegni il LED B
  }

  // scrivi sul Serial Monitor i punti totali
  Serial.print("Punti A: ");
  Serial.println(totWinA);
  Serial.print("Punti B: ");
}

```



```

Serial.println(totWinA);
Serial.println(" ");
Serial.println(" ");

// azzera i contatori dei passaggi sotto i sensori

contA=0;
contB=0;

if(totWinA==puntMax || totWinB==puntMax){ // se uno dei due ha vinto la gara
    Serial.print("Vincitore della gara: ");
    // (ha raggiunto il puntMax)
    if(totWinA==puntMax){
        // se A ha vinto la gara
        Serial.println("A");
        for(int cont=1; cont<=10; cont++){ // per 10 volte
            tone (pinBuzzer, 440); // suona la nota La
            digitalWrite(pinLedA, HIGH); // accendi il LED A
            delay(300); // aspetta 300 ms
            noTone (pinBuzzer); // spegni la nota
            digitalWrite(pinLedA, LOW); // spegni il LED A
            delay(200); // aspetta 200 ms
        }
    }

    if(totWinB==puntMax){ // se B ha vinto la gara
        Serial.println("B");
        for(int cont=1; cont<=10; cont++){ // per 10 volte
            tone (pinBuzzer, 587); // suona la nota Re
            digitalWrite(pinLedB, HIGH); // accendi il LED B
            delay(300); // aspetta 300 ms
            noTone (pinBuzzer); // spegni la nota
            digitalWrite(pinLedB, LOW); // spegni il LED B
            delay(200); // aspetta 200 ms
        }
    }
}

// inizializza per una nuova gara

totWinA=0;
totWinB=0; // azzera i contatori dei punti di A e di B
Serial.println(" ");
Serial.println(" ");
Serial.println("NUOVA GARA"); // scrivi "nuova gara" sul S. M.
Serial.println(" ");
}

```

Le due function di servizio degli interrupt, contaA e contaB, devono solo incrementare i relativi contatori dei passaggi (contA e contB) previo controllo, tramite la flag pstart, che il tempo di misura non sia terminato.

Il codice dello sketch (function di servizio degli interrupt)

```

void contaA(){ // function lanciata dall'interrupt 0 (impulso da A sul pin 2)
    if(Pstart==0){ // se siamo nell'intervallo di misura (flag Pstart=0)
        contaA++; // incrementa contaA
    }
}
void contaB(){ // function lanciata dall'interrupt 1 (impulso da B sul pin 3)
    if(Pstart==0){ // se siamo nell'intervallo di misura (flag Pstart=0)
        contaB++; // incrementa contaB
    }
}

```

4.10 Esercizio

-Fidget Spinner Game» con display

Per un utilizzo stand-alone (senza computer) del Fidget Spinner Game, aggiungi al circuito un display LCD, evitando la scrittura su Serial Monitor. Organizza i dati sul display (vincitore, giri/minuto, punteggi e record) come specificato nella [figura 4.12](#).

w	i	n	n	e	r	A	=	4		R	e	c
A	1	0	2	5		B	=	2		1	2	1

Figura 4.12 La disposizione dei dati sul display.

Suggerimento Per prima cosa devi decidere a quali pin collegare il display, visto che i pin 2 e 3 di Arduino sono indisponibili in quanto occupati obbligatoriamente dagli interrupt; nel [paragrafo 4.2](#) trovi come fare a modificare i pin dedicati al display, rispetto ai soliti (12, 11, 5, 4, 3, 2). A questo punto, per non cambiare l'assegnazione anche degli altri pin, una buona idea potrebbe essere quella di collegare il display ai sei pin analogici A0 - A5 (non utilizzati), indirizzandoli come digitali (14-19). Poi sostituisci le istruzioni Serial con quelle Liquid Crystal.

Capitolo

5

Input e output analogici

5.1 L'input analogico

Fino a questo punto del testo abbiamo posto all'ingresso di Arduino segnali digitali, cioè segnali che presentano solo due stati significativi (LOW = 0 e HIGH = 1). Questo significa che applicando tensione su un pin digital INPUT, tutti i valori al di sotto di una certa soglia (1,5 V) sono associati allo stesso valore logico LOW; sopra un'altra soglia (3,0 V) tutti i valori sono associati al valore logico HIGH.

Quando invece vogliamo dare significato a ogni valore assunto da una grandezza fisica e quindi alla tensione che la rappresenta, il segnale è analogico e quindi deve essere posto su uno dei sei ingressi analog in (da A0 ad A5).

La **figura 5.1** mostra come collegare un potenziometro, che produce una tensione variabile tra 0 V e 5 V, all'ingresso A0 di Arduino.

Il convertitore A/D del microcontrollore ha una risoluzione di 10 bit e consente di distinguere $2^{10} = 1024$ valori di tensione differenti.

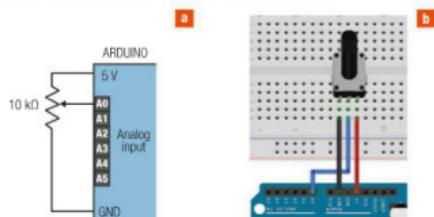


Figura 5.1 Collegamento di un potenziometro all'ingresso analogico A0 di Arduino: a) schema elettrico; b) schema di montaggio.

Con l'istruzione:

`analogRead(analogPin);`

Arduino converte la tensione d'ingresso compresa tra 0 V e 5 V (valore di default) presente su uno dei 6 pin di ingresso analogico (A0-A5) in un numero intero compreso tra 0 e 1023 (conversione analogico-digitale a 10 bit).

5.1 Esercizio svolto

Voltmetro digitale

Realizza il circuito della **figura 5.1** e programma uno sketch per scrivere sul Serial Monitor, ogni due secondi, il valore digitale acquisito dall'ingresso analogico a cui è connesso il potenziometro. Inoltre scrivi a fianco il valore della tensione V_p , sapendo che mentre la tensione del potenziometro varia da 0 V a 5 V, il corrispondente valore digitale varia da 0 a 1023. In questo modo hai realizzato un voltmetro digitale.

Soluzione

Il codice dello sketch

```
int pinPot=A0; // pin analogico di ingresso
int Vp0digit; // valore digitale della tensione Vp (0-1023)
float VpVolt; // valore della tensione Vp in volt (da 0.00 V a 5.00 V) (float = con virgola)

void setup(){
    Serial.begin(9600); // apri la porta seriale alla velocità di 9600 bps
}

void loop(){
    Vp0digit = analogRead(pinPot); // leggi la tensione sul pin A0 e carica su Vp0digit il valore convertito in digitale
    VpVolt=map(Vp0digit, 0, 1023, 0, 500); // mappa Vp0digit nell'intervallo 0-500, in proporzione
    VpVolt=VpVolt/100; // dividì VpVolt per 100, così la scala è 0.00 V - 5.00 V

    Serial.print("Valore digitale:"); // scrivi sul S. M. il valore digitale (numero da 0 a 1023)
    Serial.print(Vp0digit);
    Serial.print(" ");
    Serial.print("Tensione:"); // scrivi sul S. M. la tensione Vp (da 0.00 V a 5.00 V)
    Serial.print(VpVolt);
    Serial.println(" V");
    delay(2000); // ritardo di 2 s tra una misura e l'altra
}
```

La prima istruzione del `loop` effettua la lettura del valore analogico di tensione presente sul pin 0 e restituisce un numero, `VpDigit`, compreso tra 0 e 1023, proporzionale alla tensione.

La tensione `Vpvolt` si calcola da `VpDigit` con la proporzione:

$$Vpvolt = \frac{VpDigit \cdot 5}{1023}$$

Per calcolare la proporzione è utile l'istruzione

`vpvolt=map(vpDigit, 0, 1023, 0, 500)`

La funzione `map` converte il valore di `vpDigit`, compreso nell'intervallo tra

Valore digitale:0	Tensione:0,00 V
Valore digitale:11	Tensione:0,05 V
Valore digitale:26	Tensione:0,13 V
Valore digitale:70	Tensione:0,34 V
Valore digitale:250	Tensione:1,22 V
Valore digitale:393	Tensione:1,92 V
Valore digitale:496	Tensione:2,42 V
Valore digitale:640	Tensione:3,13 V
Valore digitale:724	Tensione:3,54 V
Valore digitale:985	Tensione:4,42 V
Valore digitale:1821	Tensione:4,99 V

Figura 5.2 Valori ottenuti sul Monitor Seriale ruotando il potenziometro da un estremo all'altro.

5.2 Esercizio

Voltmetro digitale con display LCD

Modifica l'esercizio svolto 5.1 inserendo nello schema un display LCD per visualizzare il valore della tensione.

■ Modifica della tensione di fondo scala

E' possibile modificare la tensione di fondo scala di Arduino, ponendo tale tensione (compresa tra 0 V e 5 V) sul pin AREF e inserendo l'istruzione `analogReference(EXTERNAL)`

nel blocco `setup()`.

Se, per esempio, il segnale d'ingresso varia tra 0 V e 2 V, si può porre sul pin AREF una tensione di 2 V, ottenendo il massimo della risoluzione della misura; per evitare di inserire nel circuito un generatore di tensione di riferimento da 2 V, conviene collegare ad AREF la tensione di 3,3 V presente su un pin di Arduino (vedi il progetto 5.6), ottenendo comunque un miglioramento della risoluzione rispetto al valore di default di fondo scala (5 V).

Progettare con ARDUINO

5.A Termostato programmabile

Progetta un termostato per comandare un riscaldatore elettrico (lampada a incandescenza) in modo da mantenere costante la temperatura all'interno di un contenitore. La temperatura deve essere programmata mediante due pulsanti (Up e Down), a passi di un grado centigrado. Un display LCD deve mostrare la temperatura programmata e quella reale; un LED verde si deve accendere quando il riscaldatore è attivato.

O e 1023, in un nuovo valore proporzionale, compreso tra 0 e 500; in questo modo, dividendo per 100 nella successiva istruzione, si ottiene una tensione vpvolt che varia tra 0 V e 5 V, con due cifre decimali. Per poter esprimere vpvolt con due cifre decimali, la variabile è stata dichiarata di tipo float.
Scriviamo poi i valori di vpdigit e vpvolt sul Monitor Seriale (figura 5.2).
Naturalmente sull'ingresso del voltmetro (pin A0) possiamo collegare qualunque tensione, compresa tra 0 V e 5 V; il potenziometro serve solo per testare il circuito.

Soluzione

Utilizziamo il sensore di temperatura TMP36 (figura 5.3a) in dotazione allo Starter Kit di Arduino, le cui caratteristiche sono:

- sensibilità: 10 mV/°C
- campo di lavoro: da -40°C a +125°C
- tensione di alimentazione V_t: da 2,7 V a 5,5 V

La relazione tra la tensione sul pin d'uscita V_t e la temperatura T è:

$$T = (V_t - 0,5) \cdot 100$$

oppure, esplicitando V_t:

$$V_t = T/100 + 0,5$$

quindi:

- per T = 0°C → V_t = 0,5 V
- per T = 20°C → V_t = 0,7 V
- per T = 100°C → V_t = 1,5 V.



Figura 5.3 La piedinatura a) del sensore di temperatura TMP36; b) del transistor BC547B.

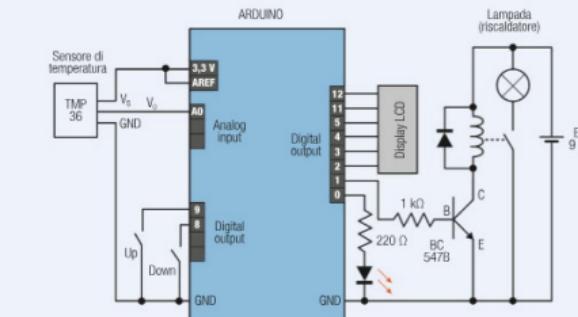


Figura 5.4 Termostato programmabile: lo schema elettrico

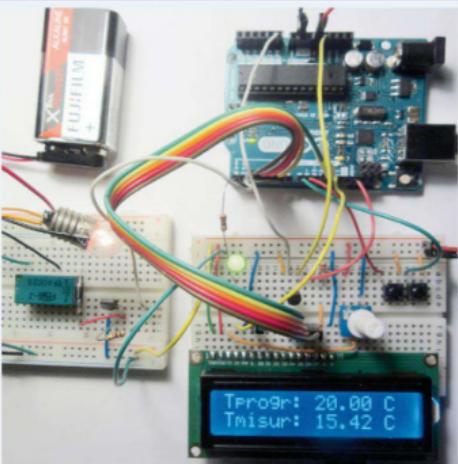


Figura 5.5 Termostato programmabile: il circuito montato.

L'hardware

Nel circuito puoi notare il transistor BC547B che riceve sulla base il segnale dal digital pin 1 di Arduino; quando questo viene posto HIGH il transistor «chiude» (va in saturazione) il circuito tra collettore ed emettitore alimentando la bobina del relè, che a sua volta chiude il contatto che alimenta la lampada.

Avendo usato una batteria (o un alimentatore) da 9 V, anche il relè e la lampada devono essere da 9 V; è possibile comunque alimentare relè e lampada in modo separato, con tensioni differenti.

Per il collegamento del display LCD fai riferimento al [paragrafo 4.1](#).

Il codice dello sketch (prima parte)

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12,11,5,4,3,2);

const int pinSens=A0; // sensore TMP36 collegato al pin analog in A0
const int pinUp=9; // pulsante Up al pin 9
const int pinDown=8; // pulsante Down al pin 8
const int pinRel=1; // pin del relè che comanda la lampada (HIGH = ON)
const int pinLed=0; // pin del LED
```



```
float tempProg=20; // valore iniziale della temperatura programmata
float tempMis; // valore della temperatura misurata
float lettura; // lettura del sensore dall'analog pin A0
float Vo; // lettura del sensore convertita in tensione
```

```
void setup(){
  pinMode(pinUp, INPUT_PULLUP);
  pinMode(pinDown, INPUT_PULLUP);
  pinMode(pinRel, OUTPUT);
  pinMode(pinLed, OUTPUT);
```

```
analogReference(EXTERNAL); // il pin AREF è collegato alla tensione di riferimento 3,3 V
lcd.begin(16, 2);
```

```
lcd.print("Tprog: ");
lcd.print(" ");
lcd.setCursor(0, 1);
lcd.print("Tmisur: ");
lcd.print(" ");
}
```

Il software

- Nella prima parte, le variabili che trattano i valori derivati dalla lettura del sensore sul pin analogico sono dichiarate `float` (4 byte) per avere il massimo della precisione e rappresentare i valori con due decimali sul display.

- Nel `setup()`, con `analogReference(EXTERNAL)` si comunica che il fondo scala della conversione non è più 5 V, ma è il valore collegato al pin AREF, cioè 3,3 V; in questo modo migliora la risoluzione della misura, considerando che la tensione sul pin A0 non supera 1,5 V.

Il codice dello sketch (seconda parte)

```
void loop(){
  if(digitalRead(pinUp)==0) tempProg++; // se è premuto il pulsante Up, incrementa di 1 grado
  if(digitalRead(pinDown)==0) tempProg--; // se è premuto il pulsante Down, cala di 1 grado
  lcd.setCursor(8, 0);
  lcd.print(tempProg); // scrivi sul display la temperatura programmata

  lettura = analogRead(pinSens); // leggi il sensore di temperatura

  Vo = lettura * (3.3/1024); // converti la lettura in volt (riferimento esterno di tensione 3,3 V)
  tempMis = (Vo - 0.5) * 100; // calcola la temperatura

  lcd.setCursor(8, 1); // scrivi sul display la temperatura misurata
  lcd.print(tempMis);
```



```

if (tempMis>tempProg-0.5) { // se è freddo (margini di 0,5 °C) accendi il riscaldatore e il LED
  digitalWrite(pinRilev, HIGH);
  digitalWrite(pinLed, HIGH);
}
if (tempMis<tempProg+0.5) { // se è caldo (margini di 0,5 °C) spegni il riscaldatore e il LED
  digitalWrite(pinRilev, LOW);
  digitalWrite(pinLed, LOW);
}
delay(2000); // aspetta 2 s

```

La prima parte del `loop()` è dedicata alla regolazione e visualizzazione della temperatura programmata (`tempProg`), cioè quella temperatura che vogliamo che raggiunga l'ambiente.

In seguito viene letto il sensore e ricavata la temperatura attuale dell'ambiente (`tempMis`).

Dal confronto tra `tempProg` e `tempMis` si capisce se accendere il riscaldatore e il LED verde oppure no.

5.2 L'output analogico

Come si è visto, l'input analogico converte la tensione posta su uno dei pin A0 - A5 in un numero a essa proporzionale (conversione analogico-digitale). Ci si potrebbe aspettare che l'output analogico di Arduino faccia l'opposto, cioè che generi una tensione su un pin di valore proporzionale a un numero (conversione digitale - analogica).

In realtà per Arduino e in generale per i microcontrollori, non è così: il segnale analogico in uscita è di tipo PWM e lo si trova su un pin digital.

Il segnale PWM

La modulazione a larghezza d'impulso (Pulse Width Modulation), a cui abbiamo già accennato nel [capitolo 2](#), associa a un numero un segnale impulsivo in cui la larghezza dell'impulso dipende dal numero; in particolare c'è una proporzionalità tra il valore numerico e il Duty Cycle (rapporto tra il tempo dello stato ALTO T_H e il tempo complessivo dell'impulso $T_H + T_L$).

Per generare segnali PWM con Arduino Uno si compiono i tre passi seguenti:

1. si sceglie un pin tra quelli digitali abilitati al PWM, contrassegnati dal simbolo \sim , cioè 3, 5, 6, 9, 10, 11 ([figura 5.6](#));
2. si dichiara il pin PWM come uscita con:

`pinMode(pin, OUTPUT)`

3. per generare il segnale PWM si usa l'istruzione `analogWrite(pin, val)`

dove:

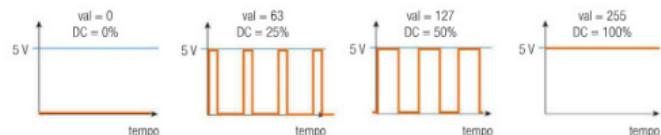
`pin`: è il numero di uno dei pin PWM (3, 5, 6, 9, 10, 11)

`val`: è un numero tra 0 e 255 (corrispondente a un numero binario a 8 bit)



Figura 5.6 Pin digitali abilitati al PWM: 3, 5, 6, 9, 10, 11

In un segnale PWM, il Duty Cycle varia da 0% a 100% in proporzione al parametro `val` ([figura 5.7](#)); la frequenza degli impulsi in uscita è fissa (490 Hz).



Gli impieghi più frequenti del segnale PWM sono:

regolazione della potenza di lampade, motori, riscaldatori.

Tutti questi attuatori hanno in comune l'inerzia della risposta: le lampade a incandescenza hanno inerzia termica (e comunque la persistenza delle immagini sulla retina crea un'inerzia anche per la luce «fredda» dei LED), i motori hanno inerzia meccanica, i riscaldatori hanno inerzia termica. Di conseguenza l'effetto ottenuto non risente del carattere impulsivo del segnale, ma solo del suo valor medio, proporzionale al Duty Cycle. In pratica l'inerzia si comporta come un filtro passa-basso rispetto alla luminosità di una lampada o di un LED ([figura 5.8](#)), alla velocità di un motore ([figura 5.9](#)) e alla temperatura di un riscaldatore: l'effetto deriva solo dalle variazioni lente (basse frequenze) e non risente degli impulsi (alte frequenze).

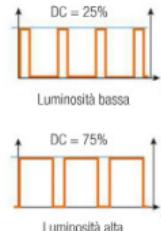
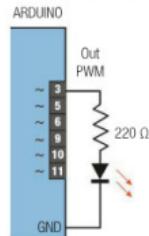


Figura 5.8 Segnale PWM per regolare la luminosità di un LED, variando il Duty Cycle.

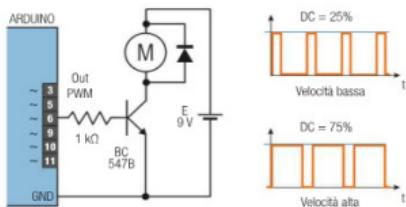


Figura 5.9 Segnale PWM per regolare la velocità di un motore in continua variando il Duty Cycle.

Conversione digitale - analogica

Per realizzare la conversione digitale - analogica mediante un segnale PWM, si può procedere in questo modo: si genera attraverso l'istruzione `analogWrite(pin, val)` un segnale PWM il cui valore medio è proporzionale al numero `val` che si vuole convertire; poi si ricava una tensione continua pari al valore medio del segnale, tramite un filtro passa-basso RC (**figura 5.10**) opportunamente dimensionato.

Il filtro passa-basso è un circuito analogico che lascia passare solo le componenti del segnale d'ingresso con frequenza al di sotto di un dato valore (detto frequenza di taglio, f_t), mentre elimina quelle a frequenza maggiore di f_t . Il filtro passa-basso RC nella **figura 5.10**, realizzato con i componenti dello Starter Kit ($R=1\text{ k}\Omega$, $C=100\text{ }\mu\text{F}$), ha una frequenza di taglio data dalla formula

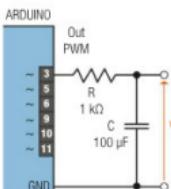
$$f_t = 1/(2\pi RC) \approx 2\text{ Hz}$$

Questo filtro lascia quindi passare solo la componente continua che, avendo frequenza 0 Hz, è l'unica inferiore a 2 Hz; elimina invece la componente a 490 Hz (e multipli) del segnale PWM. Il risultato si vede nelle forme d'onda della **figura 5.10**: la tensione V_o all'uscita del filtro è proporzionale al Duty Cycle e quindi al numero `val`.

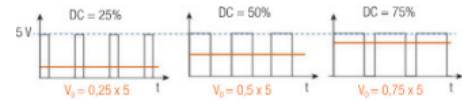
Per esempio, scrivendo

`analogWrite(3, 127)`

Figura 5.10 Conversione digitale - analogica con segnale PWM e filtro passa-basso RC.



generiamo sul pin 3 un segnale PWM con Duty Cycle DC=50% (perché 127 è il valore approssimato della metà di 255); all'uscita del filtro otteniamo un segnale continuo di ampiezza pari al 50% di 5 V, cioè 2,5 V. Variando `val` da 0 a 255 possiamo quindi ottenere, all'uscita del filtro, i corrispondenti valori di tensione continua da 0 V a 5 V.



Posizionamento di servomotori

Come si è visto nel [capitolo 2](#), il servomotore può ruotare l'albero di un angolo compreso tra 0° a 180° in base a un segnale di controllo PWM (Pulse Width Modulation) di frequenza 490 Hz (periodo circa 20 ms) con la seguente logica (**figura 5.11a**):

- larghezza dell'impulso a livello ALTO di 1,0 ms → angolo 0°;
- larghezza dell'impulso 1,5 ms → angolo 90°;
- larghezza dell'impulso 2,0 ms → angolo 180°.

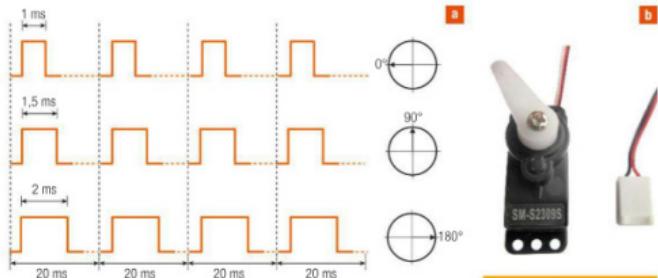


Figura 5.11 a) Posizioni del servo al variare della durata dell'impulso PWM; **b)** servomotore.

I tre fili del servo hanno la seguente funzione (**figura 5.11b**):

- rosso: alimentazione (+5 V fornita da Arduino)
- bianco: controllo della posizione (collegato a uno dei pin PWM di Arduino)
- nero: massa (GND)

Mediante la libreria `Servo.h`, è possibile assegnare l'angolo del servo senza preoccuparsi della durata corrispondente degli impulsi; le principali istruzioni sono:

- prima di `setup()`:
`#include <Servo.h>` include la libreria `Servo.h`;
- dentro a `servo1:`
`Servo servo1;` nome assegnato al servomotore;
- dentro a `setup()`:
`servo1.attach(pin);` specifica il pin a cui è collegato `servo1`;
- dentro al `loop()`:
`servo1.write(val);` invia a `servo1` l'angolo `val` (0° - 180°) da raggiungere.

Il nome `servo1` è stato assegnato arbitrariamente: in caso di più servomotori si potranno chiamare gli altri `servo2`, `servo3`, ... oppure assegnare nomi che specificano la funzione svolta.

5.3 Pilotaggio di attuatori con il PWM

Gli esempi che seguono sono applicazioni dei concetti appresi nei primi due paragrafi.

5.3 Esercizio svolto

Dissolvenza luminosa

Varia la luminosità di un LED in modo ciclico da zero al massimo e viceversa; ogni ciclo deve durare 2 secondi.

Soluzione

Collega un LED tra il pin 3 e GND, con in serie un resistore da 220 Ω. Lo sketch è il seguente:

Il codice dello sketch

```
int pinLed = 3; // LED collegato al digital pin 3
void setup() {
}
void loop() {
    for (int val=0 ; val<255 ; val+=5){ // incrementa val dal min (0) al max (255) a passi di 5
        analogWrite(pinLed, val); // accendi il LED con la luminosità specificata da val
        delay(20); // aspetta 20 ms
    }
    for (int val=255 ; val>=0 ; val-=5){ // decrementa val dal max (255) al min (0) a passi di 5
        analogWrite(pinLed, val); // accendi il LED con la luminosità specificata da val
        delay(20); // aspetta 20 ms
    }
}
```

Il `loop` è costituito da due cicli `for`:

- il primo incrementa `val` e quindi la luminosità del LED a passi di 5 (scrivere `val+=5` equivale a `val=val+5`) da 0 a 255; quindi con 51 passi si compie l'intera escursione. L'istruzione

```
analogWrite(pinLed, val)
```

produce sul pin 3 un segnale impulsivo a 490 Hz, con Duty Cycle proporzionale a `val`.

- Il secondo decremente `val` e quindi la luminosità del LED da 0 a 255 a passi di 5, in altri 51 passi.

Poiché ad ogni incremento/decremento si attendono 20 ms, l'intero ciclo di 102 passi dura circa 2 secondi.

5.4 Esercizio svolto

Regolazione della luminosità di un LED e della velocità di un motore con un potenziometro

Regola contemporaneamente la luminosità di un LED e la velocità di un motore in continua, mediante un potenziometro.

Soluzione

L'hardware è costituito da un potenziometro collegato all'ingresso analogico A0 di Arduino come nella [figura 5.1](#), un LED collegato al pin digital output 3 (PWM) come in [figura 5.8](#), e un motore in continua, pilotato da un transistor BJT BC547B (pin: CBE) la cui base è collegata al pin digital output 6 (PWM), come in [figura 5.9](#). Lo schema di montaggio è nella [figura 5.12](#).

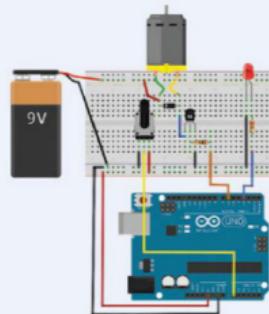


Figura 5.12 Schema di montaggio per la regolazione della luminosità di un LED e della velocità di un motore.

Il codice dello sketch

```
const int pinLed=3; // LED collegato al pin PWM 3
const int pinMot=6; // motore collegato al pin PWM 6
const int pinPot=A0; // potenziometro collegato all'analog pin A0
int val; // variabile per leggere il valore del pin analogico

void setup(){
    pinMode(pinLed, OUTPUT);
    pinMode(pinMot, OUTPUT);
}

void loop(){
    val = analogRead(pinPot); // leggi la tensione del potenziometro
                            // (restituisce in val un valore tra 0 e 1023)

    val = map(val, 0, 1023, 0, 255); // map converte il valore di val da 0 a 1023 (10 bit)
                                      // in una scala compresa tra 0 e 255 (8 bit)
    analogWrite(pinLed, val); // regola la luminosità del LED in base a val
    analogWrite(pinMot, val); // regola la velocità del motore in base a val
    delay(15); // aspetta 15 ms
}
```

Nello sketch:

- l'istruzione

```
val = analogRead(pinPot)
```

legge la tensione prodotta dal potenziometro sul pin A0 e la converte in un valore compreso tra 0 e 1023, che viene assegnato alla variabile val.

- l'istruzione

```
val = map(val, 0, 1023, 0, 255)
```

effettua una proporzione e converte il numero compreso tra 0 e 1023 (vecchio valore di val) in un numero compreso tra 0 e 255, assegnandolo di nuovo a val. Questo è necessario perché `analogRead()` acquisisce un valore a 10 bit (0-1023) mentre `analogWrite()` richiede un valore a 8 bit (0-255).

- Con le due istruzioni `analogWrite()` si invia al LED e al motore il valore che, trasformato in un segnale PWM, ne varia la luminosità e la velocità.

5.5 Esercizio

Modifica l'[esercizio svolto 5.4](#) in modo da regolare la luminosità del LED e la velocità del motore con due potenziometri distinti.

5.6 Esercizio svolto

Pilotaggio di un servomotore con un potenziometro

Utilizza un potenziometro per variare la posizione di un servomotore da 0° a 180°.

Soluzione

Lo schema di montaggio è illustrato in [figura 5.13](#). Nello sketch sono utilizzate le istruzioni della libreria `Servo.h`, descritte alla fine del [paragrafo 5.2](#).

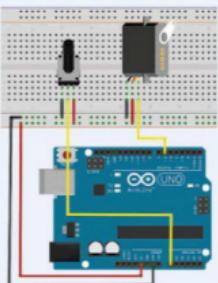


Figura 5.13 Schema di montaggio per il pilotaggio di un servomotore con un potenziometro.

Il codice dello sketch

```
#include <Servo.h> // include la libreria Servo.h
Servo servol; // crea l'oggetto servol

const int pinPot=A0; // potenziometro collegato all'analog pin A0
int val; // variabile per leggere il valore del pin analogico

void setup(){
  servol.attach(6); // servomotore collegato al digital pin 6 (PWM)
}

void loop(){
  val = analogRead(pinPot); // leggi la tensione sul potenziometro (valore tra 0 e 1023)

  val = map(val, 0, 1023, 0, 180); // map converte il valore di val (0-1023) in una scala // compresa tra 0 e 180 (angolo del servo in gradi)
  servol.write(val); // invia al servo l'angolo val da raggiungere
  delay(15); // aspetta per 15 ms che il servo arrivi a destinazione
}
```

Progettare con ARDUINO

5.8 Pista per pallina

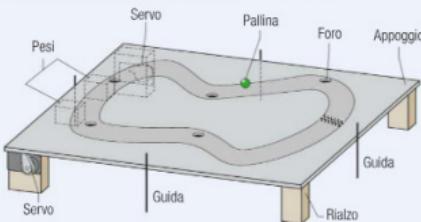
Con due servomotori e due potenziometri, realizza una pista costituita da un piano mobile che si possa inclinare in tutte le direzioni; i due servomotori, comandati dai potenziometri collegati a due leve mosse dal giocatore, cambiano l'inclinazione del piano in modo che la pallina segua la pista, delimitata da un bordo, senza cadere nei fori.

Completa il gioco con un display LCD che mostra il tempo di percorrenza di un giro (e il record), rilevato mediante una forcella ottica posta sul traguardo.

Suggerimento Per la realizzazione della pista seguì il progetto di massima della [figura 5.14](#): la pista è delimitata da un bordo che indirizza il moto della pallina; il

piano della pista, inclinabile in tutte le direzioni grazie a due servomotori che ne sollevano o abbassano due angoli, è mantenuto in posizione da quattro guide (perni) poste sui quattro lati. Il comando dei servomotori è identico a quello dell'[esercizio 5.6](#). Per il collegamento e la programmazione del display LCD vedi il [paragrafo 4.1 del capitolo 4](#), mentre il circuito della forcella ottica, da porre sul traguardo, è identico a quello nel progetto [Fidget Spinner Game](#), del [capitolo 4](#).

Figura 5.14 Progetto di massima della pista per pallina.



5.7 Esercizio

Fai muovere un servomotore ciclicamente da 0° a 180° e viceversa.

Suggerimento Come nell'esercizio svolto 5.3 «**Dissolvenza luminosa**», usa due cicli FOR, per portare il servo da 0° a 180° e poi da 180° a 0°, fissando a piacere l'incremento e un ritardo per aspettare il movimento del servo dopo ogni incremento.

5.8 Esercizio

Utilizza un servomotore per far muovere le pupille degli occhi di una maschera (figura 5.15).

Suggerimento Nella figura 5.15 vedi come realizzare la parte meccanica, per trasferire il moto del servomotore alla piastra su cui sono dipinte le pupille su sfondo bianco. Lo sketch può limitarsi a un movimento circolare, come nell'esercizio 5.7, oppure puoi programmare una sequenza di movimenti più varia.

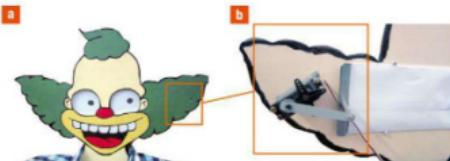


Figura 5.15 Movimento degli occhi: a) maschera; b) realizzazione meccanica.

5.9 Esercizio svolto

Inversione di marcia e regolazione di velocità di un motore in continua

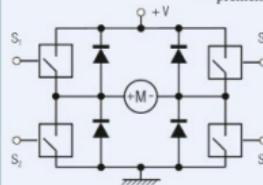
Pilota a quattro quadranti (Avanti, Stop, Indietro) un motore in continua mediante tre pulsanti (A, S, I) e un potenziometro, con la logica seguente:

- il potenziometro regola la velocità del motore;
- premendo A il motore va in avanti (il moto continua anche dopo il rilascio del pulsante);
- premendo S il motore si ferma (stop);
- premendo I il motore va in indietro.

Soluzione

Per invertire la direzione di marcia di un motore in continua devi utilizzare la configurazione a ponte (figura 5.16) descritta nel paragrafo 2.4 del capitolo 2, ottenendo il funzionamento a quattro quadranti: con S1 e S4 chiusi il motore ruota in un senso; con S2 e S3 chiusi ruota nell'altro senso, perché la tensione sul motore si inverte di segno; con tutti gli interruttori aperti il motore è fermo.

Figura 5.16 Ponte H per l'inversione del verso di rotazione di un motore.



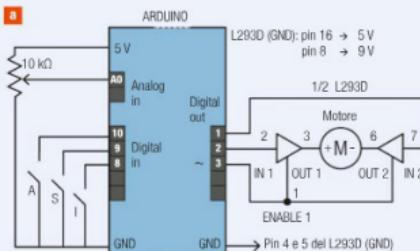
Utilizziamo uno dei due ponti H contenuti nell'integrato L293D (pin-out in figura 5.17), già completo di diodi di ricircolo:

- pin 4, 5, 12, 13: GND;
- pin 16: alimentazione della logica interna ($V_{SS} = 5\text{ V}$);
- pin 8: alimentazione del motore (V_M da 4,5 V a 36 V).

Per regolare la velocità del motore pon un segnale PWM sul pin ENABLE1, che abilita il funzionamento del ponte collegato ai pin 2, 3, 6, 7. Lo schema elettrico e quello di montaggio del circuito sono rappresentati nella figura 5.18.

Il pilotaggio si esegue nel modo seguente:

- ENABLE1=LOW: il motore è fermo;
- ENABLE1: impulsi con DC=0% + 100% (regolazione della velocità);
- IN1=HIGH e IN2=LOW, il motore gira in un verso;
- IN1=LOW e IN2=HIGH, il motore gira nel verso opposto;



Il potenziometro regola la velocità: il valore di tensione acquisito dall'ingresso analogico A0 modifica il Duty Cycle dell'uscita digitale 3 (PWM), collegata al pin ENABLE1 dell'integrato L293D.

Il loop dello sketch:

- legge e memorizza i tre pulsanti (A, S, I);
- legge la tensione del potenziometro e la converte (Pot) da 0 a 255;
- se è stato premuto S (Stop): disabilita L293D (motore fermo);
- se è stato premuto A (Avanti): IN1=LOW e IN2=HIGH (motore avanti), la velocità dipende dal potenziometro (varia il Duty Cycle su ENABLE1);
- se è stato premuto I (Indietro): IN1=HIGH e IN2=LOW (motore indietro), la velocità dipende dal potenziometro;
- mantenendo premuto A o I, le variazioni del potenziometro si ripercuotono immediatamente sulla velocità del motore.

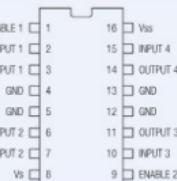


Figura 5.18 Inversione di marcia di un motore in continua e regolazione di velocità: a) schema elettrico; b) schemi di montaggio.

Il codice dello sketch

```
const int pinAv=10; // pulsante Avanti
const int pinS=9; // pulsante Stop
const int pinI=8; // pulsante Indietro
const int pinPot=40; // potenziometro regolazione velocità
const int pinIn1=2; // In1 del L293D
const int pinIn2=3; // In2 del L293D
const int pinEn1=5; // Enable1 del L293D
```



```
bool A;
bool S;
bool I;
int Pot;

void setup(){
    pinMode(pinAv, INPUT_PULLUP);
    pinMode(pinS, INPUT_PULLUP);
    pinMode(pinI, INPUT_PULLUP);
    pinMode(pinIn1, OUTPUT);
    pinMode(pinIn2, OUTPUT);
    pinMode(pinEn1, OUTPUT);

    digitalWrite(pinEn1, LOW); // inizializza Enable1 a zero
}
```

```
void loop(){
    A=digitalRead(pinAv); // leggi il pulsante Avanti
    S=digitalRead(pinS); // leggi il pulsante Stop
    I=digitalRead(pinI); // leggi il pulsante Indietro
    Pot=analogRead(pinPot); // leggi la tensione sul Potenziometro
    Pot=map(Pot, 0, 1023, 0, 255); // map converte il valore di Pot da 0 a 1023 (10 bit)
    // in una scala compresa tra 0 e 255 (8 bit)

    if(S==0){ // se è premuto Stop
        digitalWrite(pinEn1, LOW); // disabilita il ponte L293D (motore fermo)
    }
    else{ // altrimenti (se non è premuto Stop)
        analogWrite(pinEn1, Pot); // imposta il duty cycle sul pin Enable1 del L293D
        if(A==0 & I==1){ // se è premuto Avanti (e non Indietro)
            digitalWrite(pinIn1, LOW); // fa girare il motore avanti
            digitalWrite(pinIn2, HIGH);
        }
        if(A==1 & I==0){ // se è premuto Indietro (e non Avanti)
            digitalWrite(pinIn1, HIGH); // fa girare il motore indietro
            digitalWrite(pinIn2, LOW);
        }
    }
}
```

Progettare con ARDUINO

5.C Drum-ino, il robot batterista

Progetta un robot batterista che suoni un ritmo su piatti e tamburi (timpano, rullante, charleston, piatto), muovendo le bacchette mediante servomotori ([figura 5.19](#)). Il ritmo è costituito da una battuta di 16 tempi (semicrometri) ripetuta all'infinito; il lampeggio di un LED (metronomo) segnala il primo tempo della battuta.

Per avviare e fermare il ritmo, utilizza un interruttore a due posizioni (On/Off).



Figura 5.19 «Drum-ino»:

a) visione d'insieme; b) particolare della doppia bacchetta sul rullante (servomotori con versi opposti).

Soluzione

Facciamo alcune considerazioni preliminari.

- Per riuscire a eseguire sequenze veloci sul charleston e sul rullante, ti conviene utilizzare due bacchette con due servomotori per ognuno, che si muovono alternate ([figura 5.19b](#)); quindi in totale avrai sei bacchette da gestire mediante sei servomotori.
- Per aumentare la potenza del colpo sui tamburi, conviene colpire con la parte della bacchetta opposta alla punta, che ha massa maggiore; non servirà invece aumentare l'escursione del colpo, perché il servomotore si muove a velocità costante.

Schema elettrico

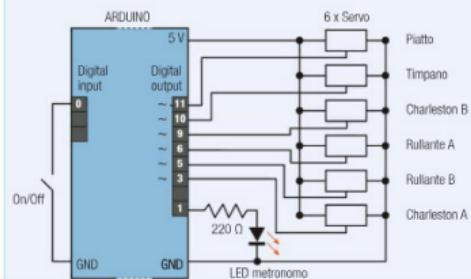


Figura 5.20 «Drum-ino», schema elettrico.

Nello schema elettrico (figura 5.20):

- l'unico ingresso è lo switch (On-Off);
- i sei servomotori sono pilotati (fili bianchi) dalle sei uscite PWM, i digital pin 3, 5, 6, 9, 10, 11; i fili neri del servomotore, connessi insieme, sono collegati a GND mentre quelli rossi alla tensione 5 V;
- per evitare di sovraccaricare l'alimentatore di Arduino, conviene collegare tutti i fili rossi dei servomotori a un alimentatore da 5 V esterno.

Nello sketch, prima del **setup**:

- la costante `metronomo` contiene il tempo del brano (numero di semiminime per minuto o bpm), che servirà nel `loop` per definire il `delay` tra un sedicesimo e l'altro;
- i quattro array (`timpano`, `piatto`, `rullante`, `charleston`) di 16 elementi l'uno, contengono la programmazione del ritmo del brano (in questo caso «Every breath you take» dei The Police).

Il codice dello sketch (dichiarazioni)

```
#include <Servo.h>           // include la libreria Servo.h
Servo servoTimpano;          // definisci i sei oggetti servo
Servo servoRullanteA;
Servo servoRullanteB;
Servo servoCharlestonA;
Servo servoCharlestonB;
Servo servoPiatto;

// DICHIAZIONI delle COSTANTI e VARIABILI
const int pinIntern=8;        // pin interruttore On-Off
const int pinLed=1;            // pin LED metronomo
bool inter;                   // variabile Interruttore ON OFF
double sedicesimo;            // durata di un sedicesimo in ms
bool ch;                      // flag charleston (per gestire l'alternanza delle bacchette)
bool ru;                      // flag rullante (per gestire l'alternanza delle bacchette)

const int metronomo=120;       // metronomo a 120 battiti al minuto (bpm)

// PROGRAMMAZIONE del RITMO (Every breath you take - Sting)
// L'array di ogni strumento contiene 16 elementi (sedicesimi): 1=suona, 0=pausa
const bool timpano [16] = {1,0,0,0,0,0,1,0,1,0,0,0,0,0,1,0};
const bool piatto [16] = {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
const bool rullante [16] = {0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0};
const bool charleston [16] = {1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0};
```



Nel **setup**:

- dalla costante `metronomo` (numero di quarti al minuto) si ricava la durata di un sedicesimo in millisecondi;
- nell'inizializzazione dei servomotori puoi notare che `rullanteB` e `CharlestonB`, a causa del montaggio ribaltato, partono da 180°, mentre gli altri servomotori partono da 0°.

Il codice dello sketch (**setup**)

```
void setup(){
    sedicesimo = 15000/metronomo; // calcolo durata di un sedicesimo in millisecondi
                                    // sedicesimo=(60 X 1000)/(4 X metronomo)
                                    // dove metronomo = n° di quarti al minuto

    // impiego dei pin di Arduino
    pinMode(pinLed, OUTPUT);      // LED metronomo
    pinMode(pinIntern, INPUT_PULLUP); // interruttore ON OFF
    servoTimpano.attach(10);       // pin servomotori
    servoRullanteA.attach(6);
    servoRullanteB.attach(5);
    servoCharlestonA.attach(3);
    servoCharlestonB.attach(9);
    servoPiatto.attach(11);

    // inizializzazioni SERVO
    servoTimpano.write(0);        // servo timpano a riposo
    servoRullanteA.write(0);      // servo rullanteA a riposo
    servoRullanteB.write(180);    // servo rullanteB a riposo (180 perché ribaltato)
    servoCharlestonA.write(0);    // servo charlestonA a riposo
    servoCharlestonB.write(180);  // servo charlestonA a riposo (180 perché ribaltato)
    servoPiatto.write(0);         // servo piatto a riposo
    digitalWrite(pinled, 0);      // LED metronomo spento

    ch=0;                         // inizializzazione FLAG charleston e rullante
    ru=0;                          // 1: suona la bacchetta A; 0: suona la bacchetta B
}
```

Nel **loop**:

- quando l'interruttore è ON, viene eseguito il ciclo `for` che analizza i 16 elementi (sedicesimi) contenuti negli array dei quattro strumenti; un valore 1 nell'array provoca il movimento di 20° del servomotore relativo (suono), un valore zero è interpretato come pausa;
- le variabili `ru` e `ch` sono `flag`: memorizzano la bacchetta del rullante e del charleston che ha suonato per ultima, in modo da far suonare l'altra nel colpo successivo;
- alla fine del `loop` i servomotori tornano in posizione di riposo;
- il `delay` tra un sedicesimo e l'altro è diviso in due parti, per dare il tempo ai servomotori di raggiungere la posizione del colpo e di tornare a riposo.

Il codice dello sketch (**loop**)

```
void loop(){
    inter = digitalRead(pinIntern); // leggi l'interruttore On-Off (On=1, Off=0)
    while(inter){                  // finché è On, suona il ritmo
```



```

for (int sedic=0; sedic<16; sedic++) { // ciclo FOR esecuzione ritmo (16 tempi)

    if (sedic == 0){ digitalWrite(pinled, HIGH);}// LED metronomo ON al 1° tempo
    else { digitalWrite(pinled, 0);} // LED metronomo OFF negli altri tempi

    if (timpano[sedic]==1){ // se nel sedicesimo c'è un 1 nell'array
        servoTimpano.write(20); // timpano, suona il timpano (da 0° a 20°)
    }

    if (rullante[sedic] == 1){ // se nel sedicesimo c'è 1 nell'array
        if (ru==1){ // rullante, se flag ru=1
            servoRullanteA.write(20); // --> suona rullanteA (da 0° a 20°)
        }
        else{
            servoRullanteB.write(180); // se ru=0, suona rullanteB (180°-->160°)
            // (servomotore B è ribaltato rispetto A)
            ru=~ru; // nega la flag ru (rullante),
            // così la prossima volta cambia bacchetta
        }
    }

    if (charleston[sedic] == 1){ // se nel sedicesimo c'è 1 nell'array
        if (ch==1){ // charleston, se flag ch=1
            servoCharlestonA.write(20); // --> suona charlestonA (da 0° a 20 °)
        }
        else{
            servoCharlestonB.write(180); // se ch=0, suona charlestonB (180°-->160°)
            // (servomotore B è ribaltato rispetto A)
            ch=~ch; // nega la flag ch (charleston)
            // così la prossima volta cambia bacchetta
        }
    }

    if (piatto[sedic] == 1){ // se nel sedicesimo c'è un 1 nell'array
        servoPiatto.write(20); // piatto, suona il piatto (da 0° a 20 °)
        delay(sedicesimo/2); // aspetta metà tempo di sedicesimo

        // riporta i sei servomotori a riposo
        servoTimpano.write(0);
        servoRullanteA.write(0);
        servoRullanteB.write(180);
        servoCharlestonA.write(0);
        servoCharlestonB.write(180);
        servoPiatto.write(0);

        delay(sedicesimo/2); // aspetta l'altra metà tempo di sedicesimo
    }

    inter = digitalRead(pinInterr); // leggi l'interruttore On-Off
}

```

Nella **figura 5.21a** puoi vedere il robot batterista ampliato con le seguenti funzioni:

- quattro percussioni colpite mediante perni, mossi da quattro elettromagneti;
- la basetta stampata in basso a sinistra contiene le interfacce a transistor per consentire alle uscite di Arduino di pilotare gli elettromagneti che, quando attivati, assorbono circa 1 A;
- sulla stessa basetta sono presenti dei pulsanti per aggiungere, al ritmo programmato, colpi di percussione in tempo reale;
- in basso a destra vedi la basetta con l'interruttore On-Off e il LED metronomo, a cui sono stati aggiunti altri due switch, per selezionare un ritmo tra quattro, e tre pulsanti per introdurre delle rullate in tempo reale su timpano e rullante, e colpi di piatto non programmati.



Figura 5.21 a) «Drum-ing» ampliato con quattro percussioni e una pulsantiera di comando; **b)** particolare di Arduino con la shield connettori.

La **figura 5.21b** mostra il particolare di Arduino con la shield connettori autostruttura, che è opportuna dato l'alto numero di connessioni verso i servomotori, gli elettromagneti delle percussioni e la pulsantiera di comando.

Progettare con ARDUINO

5.D Clown giocoliere

Progetta un sistema costituito dalla sagoma di un clown che afferra al volo, con un cestino, una pallina lanciata lungo una guida posta sopra la sua testa (**figura 5.22**).

Una volta presa al volo la pallina caduta dalla guida, dopo qualche secondo il cestino deve tornare vicino al corpo del clown, pronto per il prossimo lancio.



Figura 5.22 Il clown giocoliere.

Soluzione

Alcune considerazioni progettuali:

- per misurare la velocità v della pallina e quindi calcolare la distanza d a cui deve portarsi il cestino, nota l'altezza h della guida rispetto al cestino, si pongono lungo la guida due forcille ottiche (LED-fototransistor), analoghe a quelle utilizzate nel progetto «Fidget Spinner Game» ([capitolo 4](#)). Al passaggio della pallina, ogni forcilla invia un impulso ad Arduino: misurando l'intervallo di tempo tra gli impulsi e, nota la distanza tra le forcille (20 cm), si calcola la velocità della pallina.

La distanza d a cui cade la pallina si calcola con la formula seguente:

$$d = \sqrt{\frac{2h}{g}} = v \cdot 0,45152\sqrt{h}$$

dove v è la velocità della pallina rilevata dalle forcille ottiche lungo la guida, h è l'altezza della guida sul cestino, $g = 9,81 \text{ m/s}^2$ è l'accelerazione di gravità.

- Per evitare di costruire la struttura meccanica del sistema che muove il cestino, puoi riciclare una vecchia stampante, meglio se di formato A3, rimuovendo tutta l'elettronica tranne il motore, che deve essere di tipo passo-passo ([capitolo 2, paragrafo 2.4](#)). Procurati un'interfaccia, che puoi facilmente trovare in commercio, per pilotare gli avvolgimenti interni del motore passo-passo; in questo modo Arduino dovrà fornire all'interfaccia solo tre segnali:

Enable: il valore HIGH abilita il motore a funzionare;

Verso: il valore LOW corrisponde all'andata del carrello, HIGH corrisponde al ritorno;

Impulsi: portato HIGH e poi LOW, fornisce l'impulso che fa compiere un passo al motore.

- Per tarare il sistema, devi misurare a quanti millimetri (o frazioni di millimetro) di spostamento del carrello corrisponde un impulso inviato al motore: per esempio manda 1000 impulsi al motore, con un ciclo FOR, e misura lo spostamento corrispondente del carrello.

La stampante usata nel prototipo ha le seguenti caratteristiche:

motore passo-passo:	200 impulsi/giro
spostamento del carrello per ogni impulso:	0,21 mm
escursione massima del carrello (formato A3):	38,7 cm
numero di impulsi per compiere tutta l'escursione:	1842

- Devi porre un fincorsa ([capitolo 2](#)) nel punto di zero del carrello, sotto la verticale della fine della guida; in questo modo Arduino ha un riferimento per conoscere la posizione del carrello. Nel caso in cui il motore fosse troppo lento e, nei lanci lunghi e veloci, il cestino avesse dei problemi ad anticipare la pallina, puoi porre il punto di zero dalla parte opposta, modificando il calcolo degli impulsi da fornire.

- Lo schema elettronico del sistema è rappresentato nella [figura 5.23](#), mentre la [figura 5.24](#) mostra la disposizione dei componenti all'interno della stampante.

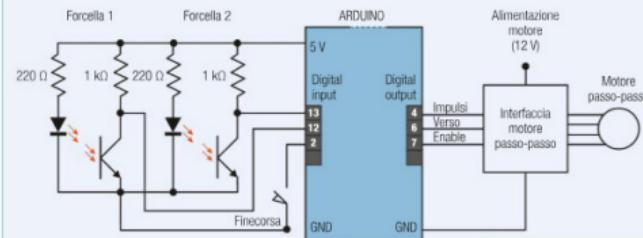


Figura 5.23 Schema elettronico del clown giocattolo.

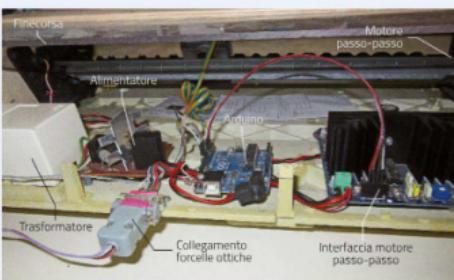


Figura 5.24 Hardware del clown giocattolo.

L'algoritmo di massima che Arduino deve eseguire è il seguente:

- all'accensione invia impulsi al motore con **Verso=indietro**, finché il carrello raggiunge il fincorsa;
- lanciata la pallina, calcola l'intervallo di tempo tra gli impulsi inviati dalle forcille ottiche;
- nota la distanza tra le forcille (20 cm), calcola la velocità v e quindi la distanza d dal punto di caduta della pallina;
- sapendo quanti millimetri di spostamento del carrello corrispondono a un impulso dato al motore, calcola il numero di impulsi N_{imp} da fornire al motore per coprire la distanza d ;
- invia N_{imp} impulsi al motore con **Verso=avanti**;
- la pallina dovrebbe cadere nel cestino;
- dopo 5 secondi ricomincia da capo.

```

Il codice dello sketch 
```

```

const double h=0.6;           // altezza della guida rispetto al cestino, in metri
const double deltaT=0.00021;  // ad ogni impulso il carrello avanza di 0,21 mm
const int pinSens1=12;        // pin prima forcella ottica (HIGH: palla in transito)
const int pinSens2=13;        // pin seconda forcella ottica (HIGH: palla in transito)
const int pinFC=2;            // pin finecorsa (di tipo NA->LOW: carrello al finecorsa)
const int pinVerso=6;          // pin Verso motore (HIGH: indietro)
const int pinImpulsi=4;        // pin Impulsi al motore (HIGH - LOW)
const int pinEnable=7;         // pin abilitazione motore (HIGH: abilitato)

bool valFC;                  // valore del finecorsa (LOW: carrello al finecorsa)
double deltaT;                // intervallo di tempo tra le due forcelle ottiche
bool valSens;                 // valore letto dalle forcelle ottiche
double v;                      // velocità della pallina sulla guida
double d;                      // distanza di caduta
float Nimp;                   // numero di impulsi da fornire al motore

void setup(){
    pinMode(pinSens1, INPUT);
    pinMode(pinSens2, INPUT);
    pinMode(pinFC, INPUT_PULLUP);
    pinMode(pinVerso, OUTPUT);
    pinMode(pinImpulsi, OUTPUT);
    pinMode(pinEnable, OUTPUT);
}

void loop(){
    // porta il carrello al finecorsa
    digitalWrite(pinEnable,HIGH); // abilita il motore
    digitalWrite(pinVerso,HIGH); // verso=indietro
    do{
        digitalWrite(pinImpulsi,HIGH); // manda impulsi al motore ...
        delayMicroseconds(700);
        digitalWrite(pinImpulsi,LOW);
        delayMicroseconds(700);
    }while(digitalRead(pinFC)); // ...finché il carrello non arriva al finecorsa
    digitalWrite(pinEnable,LOW); // arrivato al finecorsa, disabilita il motore

    // misura la velocità della palla e calcola il n° di impulsi da dare al motore
    do{
        valSens = digitalRead(pinSens1); // leggi il sensore 1 finché non passa la palla
    }while(valSens==0);
    double mis1=millis(); // memorizza il tempo 1
    do{
        valSens = digitalRead(pinSens2); // leggi il sensore 2 finché non passa la palla
    }while(valSens==0);
    double mis2=millis(); // memorizza il tempo 2
    double deltaT=mis2-mis1;
}

```



```

v=(28/1000)/(deltaT/1000); // calcola velocità in m/s (distanza sensori 28 cm)
d=v*0,45152*sqrt(h); // calcola la distanza di caduta d, in metri
Nimp= d/deltaT; // calcola il numero di impulsi da inviare al motore

// sposta il carrello nel punto previsto per la caduta della palla
digitalWrite(pinEnable,HIGH); // abilita il motore
digitalWrite(pinVerso,LOW); // verso=avanti
for(int i=0;i<Nimp;i++){
    digitalWrite(pinImpulsi,HIGH);
    delayMicroseconds(700);
    digitalWrite(pinImpulsi,LOW);
    delayMicroseconds(700);
}
digitalWrite(pinEnable,LOW); // disabilita il motore
delay(5000); // aspetta 5 secondi
}

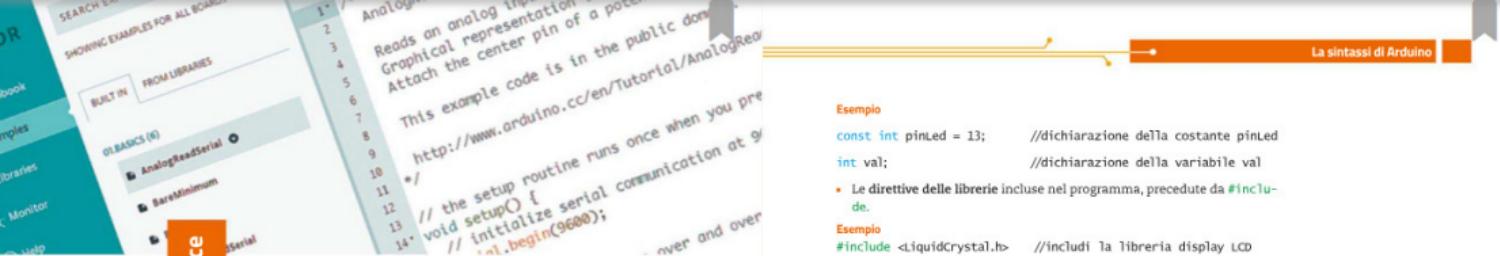
```

- Nel **loop**, la lettura dei due sensori per rilevare la velocità della pallina è stata fatta con la tecnica del **polling** (Arduino continua a leggere il sensore finché non cambia stato); potete anche collegare i sensori ai **pin digital 2 e 3** e sfruttare l'**interrupt** ([capitolo 4, paragrafo 3](#)) ma, poiché nell'attesa della palla Arduino non deve fare nient'altro, le due soluzioni sono equivalenti.

- Con l'istruzione **delayMicroseconds(700)** si attendono 700 microsecondi tra una commutazione e l'altra degli impulsi al motore, riducendo questo tempo il motore si muove più velocemente ma, sotto un certo valore, non riesce a seguire gli impulsi e si blocca. Facendo delle prove puoi individuare il **delay** minimo che garantisce comunque il funzionamento corretto.

Ti suggerisco alcune modifiche per aggiungere altre funzioni al clown:

- fai muovere gli occhi del clown con un servomotore, come proposto nell'[esercizio 5.8](#).
- Inserisci un circuito voice recorder, opportunamente amplificato, per far pronunciare al clown delle frasi: «Lanciami la pallina!», «Bel lancio!», ...
- Per creare un po' di suspense, inserisci un tempo di attesa nella partenza del cestino, in funzione della velocità del lancio, in modo che il cestino arrivi a destinazione solo con un piccolo anticipo rispetto alla pallina.



Appendice

La sintassi di Arduino

1 Informazioni generali

Ricordiamo alcune informazioni generali sulla programmazione di Arduino.

- Per accedere alla sintassi completa del linguaggio di programmazione di Arduino, seguire nell'IDE il percorso: Aiuto > Guida di riferimento.
- Per verificare la correttezza sintattica del programma scritto preme re (verifica) in alto a sinistra nell'IDE.
- Tutte le istruzioni (tranne `#define` e `#include <nome-libreria>`) terminano con ; (punto e virgola);
- I commenti si inseriscono in questo modo:
`// testo del commento (commento su linea singola)`
`/* testo del commento */ (commento su linee multiple)`
- Per scrivere le parentesi graffe{} digitare:
 su Windows: Alt+123 e Alt+125 oppure Alt+Gr+Shift+[e Alt+Gr+Shift+]
 su Mac: Shift+alt+[e Shift+alt+]

2 Struttura dello sketch

Uno sketch è costituito dalle seguenti sezioni:

- Titolo e descrizione (facoltativi ma consigliati). Sono preceduti da /* e seguiti da */.

Esempio

```
/*
Titolo e descrizione
*/
```

- Dichiarazioni direttive

- Dichiarazioni di costanti e variabili globali.

Esempio

```
const int pinLed = 13; // dichiarazione della costante pinLed
```

```
int val; // dichiarazione della variabile val
```

- Le direttive delle librerie incluse nel programma, precedute da `#include`.

Esempio

```
#include <LiquidCrystal.h> // include la libreria display LCD
```

- Il codice di SETUP (che sarà eseguito una sola volta), preceduto da `void setup()`.

Esempio

```
void setup() {
  Serial.begin(9600); //inizializza il Monitor Seriale
  pinMode(13, OUTPUT); //inizializza il pin 13 come uscita
  lcd.begin(16, 2); //imposta il numero di colonne
  //e di righe del display
}
```

- Il codice del programma principale (che sarà eseguito ripetutamente finché la scheda non viene spenta, resettata o riprogrammata), preceduto da `void loop()`.

Esempio

```
void loop(){
  ....;
}
```

- Le eventuali FUNCTION richiamate nel programma.

Esempio

```
void vincitore (int winner){ //function che non restituisce
  //valori (void)
  ....; //codice della function
}
```

```
int moltiplica (int x, int y){ //function che restituisce un valore
  //intero (Int)
  int prodotto = x*y; //codice della function
  return prodotto; //valore restituito (prodotto)
}
```

3 Operatori

■ Operatori aritmetici

=	assegnazione (x=10)
+	somma

-	differenza
++	incrementa x
--	decrementa x
*	prodotto
/	quoziente
%	resto della divisione tra interi

Esempio

```
x = 12 % 5 → x vale 2
```

■ Operatori logici

&&	AND
	OR
!	NOT

■ Operatori di comparazione

x==y	x uguale a y
x!=y	x diverso da y
x<y	x minore di y
x>y	x maggiore di y
x<=y	x minore o uguale a y
x>=y	x maggiore o uguale a y

4 Tipi di dati**const** costante.**char** carattere ASCII, usa 1 byte.**Esempio**

```
char myChar = 'A'
```

bool variabile logica (true=HIGH=1, false=LOW=0). Usa 1 byte.**int** intero, valori da -32768 a +32767. Usa 2 byte.**long** intero, valori da -2147483648 a +2147483647. Usa 4 byte.**float** numero con virgola, valori da -3.402823E+38 a +3.402823E+38. Usa 4 byte.**Esempio**

```
const float Pi = 3.14
```

array insieme di elementi omogeni, individuati da un indice numerico.**Esempio**

```
int myArray[10]={19, 32, 2, 41, 38, 22, 7, 87, 9, 11} dichiara  
un array di 10 elementi interi, numerati da 0 a 9, elencati tra le gra-  
fe; per esempio, richiamando myArray[9] si ottiene il numero 11.
```

volatile vanno dichiarate volatili le variabili modificate all'interno di una routine di servizio a un *interrupt*.

5 Monitor seriale

Per effettuare il *debug* del programma, durante l'esecuzione si possono inviare dati da Arduino al PC e viceversa, mediante le istruzioni *Serial*. Per attivare il Monitor Seriale premere l'icona in alto a destra nell'IDE.

Serial.begin(9600) //nel setup) imposta la velocità di
//comunicazione a 9600 bps

Comunicazione Arduino →PC

Serial.println(val) //scrive sul Monitor Seriale
//il valore della variabile val
//oppure caratteri tra apici
//come «Hello world» e va a capo

Serial.print(val) //scrive sul Monitor Seriale
//senza andare a capo

Comunicazione PC →Arduino

Serial.available() //restituisce il numero di byte
//arrivati dal PC e disponibili
//per la lettura

Serial.read() //legge e restituisce il valore inviato
//invia dal PC mediante
//Monitor seriale

flush() //cancella la coda dei dati arrivati
//sulla porta seriale

6 Pin digitali e analogici**■ Pin digitali (0-13): lettura e scrittura**

Nel setup
pinMode(pin, OUTPUT) //imposta il pin come output

pinMode(pin, INPUT) //imposta il pin come input

pinMode(pin, INPUT_PULLUP) //imposta il pin come input
//attivando il pull-up interno

Istruzioni di lettura e scrittura

```
digitalRead(pin)           //legge il valore logico (HIGH o LOW)
                           //dal pin di input digitale (0-13)

digitalWrite(pin, val)     //scrive sul pin di output digitale
                           //il valore val (HIGH o LOW)
```

In caso di necessità si possono usare come digitali anche i pin analogici, facendo riferimento agli indirizzi da 14 a 19.

Esempio

Scrivi sul pin digitale 13 il valore letto da un pulsante collegato al pin digitale 7, dopo averlo negato.

```
void setup(){
  pinMode(13, OUTPUT);    //imposta il pin digitale 13 come output
  pinMode(7, INPUT_PULLUP); //imposta il pin digitale 7 come input (pull-up interno)
}

void loop(){
  bool val = digitalRead(7); //leggi il valore sul pin di input digitale 7 e associalo a val
  val= !val;               //nega il valore di val
  digitalWrite(13, val);   //scrivi sul pin 13 il valore val (HIGH o LOW)
}
```

Tutto il loop si può compattare in una riga unica equivalente:

```
digitalWrite(13, !digitalRead(7));
```

■ Pin analogici (A0-A5): lettura

```
analogRead(analogPin)      //legge la tensione su un pin analogico
                           // (A0-A5) e converte il valore compreso
                           // tra 0 V e 5 V in un numero intero
                           // compreso tra 0 e 1023
```

```
analogReference(EXTERNAL)  //nel setup modifica la tensione
                           //di fondo scala (5 V) portandola al
                           //valore della tensione vref
                           //collegata al pin AREF
```

Esempio

Leggi ogni 2 secondi il valore di tensione sul pin A0 (fornita mediante un potenziometro) e scrivilo sul Monitor Seriale.

```
int analogPin = A0;          //pin A0 collegato al potenziometro
int val;                     //val: variabile in cui memorizzare il valore letto su A0

void setup(){
  Serial.begin(9600);        //inizializza la comunicazione seriale con il computer a 9600 bps
```



```
void loop(){
  val = analogRead(analogPin); //associa a val il valore digitale (0-1023) corrispondente
                           //alla tensione letta sul pin A0
  Serial.println(val);      //scrivi il valore di val sul Monitor Seriale
}
```

■ Uscita analogica (PWM) sui pin digitali (3, 5, 6, 9, 10, 11)

```
analogWrite(pin, val)       //su uno dei 6 pin digitali
                           //PWM (3, 5, 6, 9, 10, 11)
                           //converte il valore val (0-255)
                           //in un segnale PWM (frequenza 490 Hz)
                           //con duty-cycle (0% - 100%)
                           //proporzionale a val
```

Esempio

Modifica la luminosità di un LED (collegato al digital pin PWM 9), mediante un potenziometro (collegato all'analog pin 0).

```
int pinled = 9;             //LED connesso al digital pin 9 (PWM)
int analogIn = A0;           //potenziometro connesso all'analog pin 0
int val = 0;                 //variabile dove memorizzare il valore letto dal pin analogico
```

```
void setup(){
  pinMode(pinled, OUTPUT); //imposta il pin 9 come output
}

void loop(){
  val = analogRead(analogIn); //legge la tensione sull'analog pin 0 e associa a val
                           //il corrispondente valore digitale (0 - 1023)
  val= map(val, 0, 1023, 0, 255); //map converte il valore di val (0 - 1023) in un nuovo valore
                           //compreso tra 0 e 255 (per l'istruzione analogWrite che segue)

  analogWrite(pinled, val); //genera un segnale PWM sul pin 9, con duty-cycle
                           //proporzionale a val (0 - 255);
                           //quindi la luminosità del LED aumenta all'aumentare di val
}
```

7 Strutture di controllo del flusso di programma**■ Istruzione FOR**

```
for (inizializzazione; condizione; incremento) {
  .....; //blocco di istruzioni (ciclo)
}
```

Il FOR ripete il blocco di istruzioni tra le graffe finché la variabile di controllo (per esempio x), definita e inizializzata in inizializzazione e che a ogni ciclo viene incrementata della quantità incremento, soddisfa

la condizione (per esempio `x <= 10`). Quando la variabile non soddisfa più la condizione, si esce dal FOR e l'esecuzione del programma riprende dall'istruzione posta dopo la parentesi graffa di chiusura del ciclo.

Esempio 1

Somma i primi 100 numeri interi.

```
int somma = 0;
for (int x = 1; x <= 100; x++) {
    // la variabile somma è inizializzata a 0
    // per x che va da 1 a 100, a incrementi di 1 (x++)
    somma = somma + x;
}
// al valore di somma viene aggiunto il valore di x
// a fine ciclo somma è uguale alla somma
// dei primi 100 numeri interi
```

Esempio 2

Varia la luminosità di un LED dal minimo al massimo

```
for (int i=0; i<=255; i++){
    // per i che va da 0 a 255, a incrementi di 1 (i++)
    analogWrite(pinLed, i); // invia al LED un segnale PWM di valore i
    delay(10);
}
```

Istruzione IF

```
if (espressione){
    .....; // blocco di istruzioni
}
```

Se l'espressione (booleana) è VERA, esegue il blocco di istruzioni; altrimenti passa oltre la parentesi graffa di chiusura del ciclo.

Esempio

Accendi il LED se `x > 120`

```
if (x > 120) {
    // se x>120
    digitalWrite(pinLed, HIGH); // accendi il LED
}
```

Istruzione IF...ELSE

```
if (espressione){
    .....; // blocco di istruzioni A
} else{
    .....; // blocco di istruzioni B
}
```

Se l'espressione è VERA, esegue il blocco di istruzioni A; altrimenti il blocco B.

Esempio

`Se x<100 incrementa x e y, altrimenti (se x >= 100) decrementa x e y.`

```
if (x<100){ // se x<100
    x++; // incrementa x e y
    y++;
}
else{ // altrimenti (x>=100)
    x--; // decrementa x e y
    y--;
}
```

Istruzione SWITCH...CASE

```
switch (var) {
    case 1:
        .....; // blocco 1 di istruzioni (quando var = 1)
        break;
    case 2:
        .....; // blocco 1 di istruzioni (quando var = 2)
        break;
    .....; // eventuali altri case
    break;
    default:
        //default è opzionale
        .....; // se var non corrisponde ai casi elencati
        //esegui questo blocco di istruzioni
        break;
}
```

Esegue blocchi differenti di istruzioni, in base al valore (1, 2, 3, ...) della variabile var.

Istruzione WHILE

```
while (espressione){
    .....; // blocco di istruzioni
}
```

Ripete il blocco di istruzioni finché l'espressione (booleana) è VERA; quando diventa FALSA l'esecuzione procede con le istruzioni dopo la parentesi graffa di chiusura. L'espressione viene testata prima dell'esecuzione del blocco di istruzioni.

Esempio

Ripeti un blocco di istruzioni finché non viene premuto un pulsante.

```
Puls = digitalRead(pinPuls); // leggi lo stato del pulsante
while(Puls){
    ...
    // blocco di istruzioni, ripetuto finché Puls non diventa FALSO
    ...
    Puls = digitalRead(pinPuls); // aggiorna Puls
}
```

■ Istruzione DO...WHILE

```
do{
    .....; //blocco di istruzioni
} while (espressione);
```

DO...WHILE è analoga a WHILE, ma l'espressione è testata dopo aver eseguito il blocco di istruzioni; quindi il blocco viene eseguito almeno una volta.

Esempio

Leggi il pin 7, collegato al pulsante PA, finché questo non viene premuto.

```
do{
    PA = digitalRead(7);
}while(PA); // leggi il pulsante PA
            // finché PA=1 (non premuto)
```

■ Istruzione BREAK

BREAK è usata per uscire da cicli FOR, WHILE e DO...WHILE, bypassando la normale condizione del ciclo.

Nell'esempio che segue, il servomotore viene portato da 0° a 180° a passi di 1°, con un ciclo FOR; quando si preme il pulsante durante il ciclo, il servomotore torna a 0° e si esce dal FOR, a causa dell'istruzione break.

```
for (pos = 0; pos <= 180; pos++){
    servol.write(pos); // porta il servol a pos (gradi)
    delay(15);
    puls=digitalRead(pinPuls); // leggi lo stato del pulsante
    if(puls==0){ // se il pulsante è premuto
        pos = 0; // azzerà pos
        servol.write(pos); // riporta il servol a zero
        break; // esce dal ciclo FOR
    }
}
```

■ Funzioni matematiche e trigonometriche

min(x, y)	// restituisce il minimo tra x e y
max(x, y)	// restituisce il massimo tra x e y
val = constrain(val, 10, 150)	// limita il range della variabile val // tra 10 e 150
abs(x)	// restituisce il valore assoluto di x
pow(base, exponent)	// eleva la base all'esponente
sin(rad)	// seno dell'argomento rad in radianti
cos(rad)	// coseno dell'argomento rad in radianti

tan(rad)

//tangente dell'argomento rad in radianti

memcpy(array1, array2, 16)

//copia tutto l'array2 sull'array1
//(16 elementi)

sqrt(x)

//calcola la radice quadrata di x

Esempio

Uguaglia y alla radice quadrata di x.

```
x=4;
y= sqrt(x); // ora si ha: y=2
```

■ Altre funzioni

tone(pin, frequenza, durata)

//genera una nota musicale su un pin
//con una certa frequenza e durata
//in ms (durata è facoltativa)

noTone(pin)

//ferma la nota sul pin

millis()

//restituisce i millisecondi
//dall'avvio dello sketch
//o dall'ultimo reset

pulseIn(pin, value, timeout)

//restituisce la durata in μ s
//(da 10 μ s a 3 min) di un impulso
//sul pin specificato; se value=HIGH
//l'impulso inizia quando il pin va HIGH
//se LOW è il contrario;timeout
//(opzionale): n' di μ s (di default 1 s)
//entro cui deve iniziare l'impulso
//o la funzione restituisce il valore 0

random(min, max)

//genera un numero casuale compreso
//tra min e max; perché sia veramente
//casuale la funzione deve essere
//inizializzata nel setup() con l'istruzione
//randomSeed(analogRead(0))

val= map(val, 0, 1023, 0, 255)

//map converte il valore di val (0 - 1023)
//in un nuovo valore compreso tra 0 e 255

8 Function

Le function sono porzioni di codice che, quando richiamate, eseguono una funzione specifica (per esempio, il calcolo della media di quattro valori) e restituiscono il risultato associato al nome della function.

Due function standard sono **setup()** e **loop()**, ma il programmatore può creare altre funzioni esternamente alle graffe di queste due, per esempio dopo **loop()**.

Se la funzione non restituisce alcun valore, il nome della function è preceduto da **void** e manca l'istruzione **return**.

Esempio 1

La function **moltiplica** (dichiarata fuori dal ciclo **loop**) esegue il prodotto di due interi; riceve i due fattori (**a** e **b**) che associa ai due parametri **x** e **y**, e restituisce un intero (prodotto) associato al nome della function **moltiplica**. Quindi il risultato è: $p=a \cdot b=6$.

```
void loop() {
    int a = 2;
    int b = 3;
    int p = moltiplica(a, b); // 1) chiama la function "moltiplica", passando i valori
                            // dei parametri a, b;
                            // 2) la function restituisce la variabile
                            // prodotto = x * y = 6;
                            // 3) p assume quindi il valore 6
}

int moltiplica(int x, int y){ // codice della function "moltiplica",
    // x, y assumono i valori dei parametri a, b, passati
    // nella chiamata
    int prodotto = x * y;
    // esegui il prodotto
    // la function "moltiplica" restituisce
    // la variabile "prodotto"
}
```

Esempio 2

Quando viene chiamata, la function **print()** scrive «Hello world!» sul Monitor Seriele; non passa parametri (parentesi vuote) e non restituisce valori (dichiarazione preceduta da **void** e senza **return**).

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

void loop(){
    print();
    delay(2000);
}

void print(){ // codice della function "print"
    // che non restituisce valori (void)
    Serial.println("Hello world!"); // scrivi "Hello world!" sul Monitor Seriele;
                                    // non c'è "return" perché non ci sono valori restituiti
}
```

9 Interrupt (esterni)

Quando si verifica un interrupt esterno (sul digital pin 2 o 3), viene interrotta l'esecuzione dello sketch e lanciata la relativa function di servizio. L'istruzione principale per la gestione degli interrupt è:

attachInterrupt(interrupt, function, mode)

dove:

interrupt	// numero dell'interrupt (0 su digital pin 2 // 1 su digital pin 3);
function	// nome della funzione da richiamare quando // si verifica l'interrupt (nessun parametro // fornito o restituito);
mode	// modalità di interrupt: LOW quando il pin è Low // CHANGE quando il pin cambia valore // RISING sul fronte di salita // FALLING sul fronte di discesa.

Le variabili modificate all'interno della function di servizio dell'interrupt devono essere dichiarate di tipo volatile.

Esempio

Ogni secondo scrivi sul digital pin 13 (LED a bordo della scheda) il valore della variabile **state**, che si inverte ogni volta che arriva un interrupt sul digital pin 2 (fronte di discesa: HIGH → LOW)

```
volatile int state = LOW; // "state" (volatile) viene modificata dentro la function "blink"
                          // di servizio all'interrupt

void setup(){
    pinMode(13, OUTPUT);
    attachInterrupt(0, blink, FALLING); // interrupt 0: quando il pin 2 passa da HIGH a LOW,
                                      // chiama la function "blink"
}

void loop(){
    digitalWrite(13, state); // scrivi il valore di "state"
    delay(1000);           // ogni secondo
}

void blink(){ // function "blink" Richiamata dall'interrupt sul pin 2
    // non riceve parametri e non restituisce valori
    state = !state;        // inverti il valore di "state"
}
```

10 Librerie standard

Le librerie mettono a disposizione delle istruzioni che consentono di eseguire in modo semplice alcune funzionalità, come il pilotaggio di display LCD, di servomotori e di motori passo-passo. Per includere una libreria nello sketch, prima del setup si scrive la direttiva:

```
#include<nome_libreria>
```

■ <LiquidCrystal.h>

<LiquidCrystal.h> è la libreria per il pilotaggio di display LCD.

- Prima di setup():


```
#include <LiquidCrystal.h> //Include la libreria LiquidCrystal
LiquidCrystal lcd(12,11,5,4,3,2) //inizializza la libreria
//con i numeri dei 6 pin di interfaccia.
//per usare altri pin, basta dichiararli
//nello stesso ordine tra le parentesi.
```
- Nel setup():


```
lcd.begin(16, 2) //imposta il numero di colonne (16)
//e righe (2) del display
```
- Istruzioni principali:


```
lcd.setCursor(colonna, riga) //porta il cursore nella posizione
//del display specificata da
//(colonna 0-15, riga 0-1)

lcd.print(dato) //scrive il dato (di tipo char, byte,
//int, long, string) sul display;
//per esempio: lcd.print(cont);
//scrive nella posizione del cursore
//il valore della variabile cont,
//mentre cd.print("Ciao!")
//scrive la stringa "Ciao!"

lcd.clear() //cancella tutto il display

lcd.noDisplay() //spegne il display

lcd.display() //riaccende il display
```
- Altre istruzioni della libreria:


```
lcd.home() //porta il cursore in alto a sinistra

lcd.cursor() //mostra il cursore sul display
lcd.noCursor() //nasconde il cursore sul display
lcd.scrollDisplayLeft() //trasla il contenuto a sinistra
//di 1 posizione;

lcd.scrollDisplayRight() //trasla il contenuto a destra
//di 1 posizione;
```

```
lcd.autoscroll() //attiva l'autoscroll: i caratteri si
//aggiungono a destra traslando
//verso sinistra

lcd.noAutoscroll() //disattiva l'autoscroll

lcd.createchar(num, data) //crea un carattere personalizzato
//(glyph) per l'LCD (esempio 2)

lcd.write(data) //visualizza un carattere personalizzato
//sul display LCD
```

Esempio 1

Scrivi sull'LCD «W Arduino» e il numero di secondi dall'avvio.

```
#include <LiquidCrystal.h> // include la libreria LiquidCrystal
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // inizializza la libreria con i numeri
// dei pin di interfaccia

void setup() {
  lcd.begin(16, 2); // imposta il numero di colonne e righe del display

  lcd.setCursor(7, 0); // porta il cursore sulla colonna 3 e riga 0
  lcd.print("W"); // scrivi "W"

  lcd.setCursor(0, 1); // porta il cursore sulla colonna 0 e riga 1
  lcd.print("Arduino!"); // scrivi "Arduino!" all'inizio della seconda riga
}

void loop() {
  lcd.setCursor(0, 1); // imposta il cursore su colonna 0, riga 1
  lcd.print(millis()/1000); // scrivi il numero dei secondi dal reset
}
```

Esempio 2

Crea un carattere personalizzato (faccina sorridente, 7x5 pixel) e scrivilo sul display LCD.

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

byte smiley[8] = { // definisci l'array di 8 elementi di tipo byte (8 righe di pixel)
  // che specifica i pixel del carattere; in ogni byte si definiscono
  // solo 5 bit (3 colonne di pixel), corrispondenti ai pixel di ogni
  // riga: 0 pixel spento, 1 pixel acceso
  B00000,
  B10001,
  B00000,
  B00000,
  B10001,
  B01100,
  B00000,
```

```

    // l'ottava riga è tutta spenta (carattere di 7x5 pixel)
};

void setup() {
  lcd.createChar(1, smiley); // il carattere smiley è il numero 1
  lcd.begin(16, 2);
  lcd.write(1);           // scrivi sul display LCD il carattere personalizzato numero
}
void loop() {}           // loop vuoto

```

■ <Servo.h>

<Servo.h> è la libreria per pilotaggio di servomotori.

- Prima di `setup()`:


```
#include <Servo.h>      //include la libreria Servo.h
```

`Servo servol;` //servol: nome assegnato al servomotore
- Nel `setup()`:


```
servol.attach(pin);    //specifica il pin a cui è collegato servol
```
- Nella `loop()`:


```
servol.write(val);    //invia a servol l'angolo da raggiungere
                            // (in gradi)
```

Esempio

Controlla la posizione di un servomotore con un potenziometro.

```

#include <Servo.h>      // includi la libreria Servo.h
Servo servol;            // crea l'oggetto servol

const int pinPot=A0;     // il potenziometro è collegato all'analog pin A0
int val;                 // variabile che memorizza il valore del pin analogico

void setup(){
  servol.attach(6);       // servomotore collegato al digital pin 6 (PWM)
}

void loop(){
  val = analogRead(pinPot);        // leggi la tensione del potenziometro
                                  // (valore tra 0 e 1023)
  val = map(val, 0, 1023, 0, 180); // map converte il valore di val (0-1023) in una scala
                                  // compresa tra 0 e 180 (posizione del servo in gradi)
  servol.write(val);             // invia al servo l'angolo val da raggiungere
  delay(15);                   // aspetta per 15 ms che il servo arrivi a destinazione
}

```

■ Altre librerie standard

Elenchiamo infine alcune librerie standard.

Stepper	pilota motori passo-passo unipolari e bipolarì.
SD	legge e scrive le SD card.
XBee	gestisce comunicazioni wireless con il modulo XBee.
SimpleTimer()	consente l'esecuzione di una porzione di codice ogni <i>n</i> milisecondi, senza l'impiego di interrupt e del timer interno.
SoftwareSerial	consente la comunicazione su porta seriale con ogni piedino digitale, (velocità fino a 115200 bps).
EEPROM	scrive e legge nella memoria non volatile (EEPROM) del microcontrollore.
Ethernet	gestisce il collegamento a internet mediante la Ethernet Shield.
WiFi	gestisce il collegamento a internet mediante la WiFi Shield.
GSM	gestisce il collegamento alla rete GSM mediante la GSM Shield.

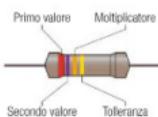
Codice colore dei resistori

Nei resistori di piccola potenza (0,25 W, 0,5 W, 1 W) il valore è stampato secondo un codice a fasce di colore. Per potenze superiori il valore di solito è indicato sul contenitore del resistore.

I comuni resistori con tolleranza al 5% sono caratterizzati da 4 fasce colorate, che indicano i due valori, il moltiplicatore (cioè la potenza di 10 che moltiplica le prime due cifre) e la tolleranza.

Per esempio, il resistore nella figura qui a fianco è dotato di fasce di colore rosso (2), viola (7), arancio (3), oro ($\pm 5\%$); ha quindi un valore di $27 \cdot 10^3 \Omega = 27 \text{ k}\Omega$ con tolleranza $\pm 5\%$.

I codici colore utilizzati e i dati corrispondenti sono riportati nella tabella qui sotto.



Colore	Valore	Moltiplicatore	Tolleranza (%)
Nero	0	0	-
Marrone	1	1	± 1
Rosso	2	2	± 2
Arancio	3	3	$\pm 0,05$
Giallo	4	4	-
Verde	5	5	$\pm 0,5$
Blu	6	6	$\pm 0,25$
Violetto	7	7	$\pm 0,1$
Grigio	8	8	-
Bianco	9	9	-
Oro	-	-1	± 5
Argento	-	-2	± 10
Niente	-	-	± 20

Copyright © 2018 Zanichelli editore S.p.A., Bologna [82054]
www.zanichelli.it

I diritti di elaborazione in qualsiasi forma o opera, di memorizzazione anche digitale su supporti di qualsiasi tipo (flessi magnetici e ottici), di riproduzione e di adattamento totale o parziale con qualsiasi mezzo (compresi i microfilm e le copie fotostatiche), i diritti di noleggio, di prestito e di trasmissione sono riservati per tutti i paesi. L'acquisto della presente copia dell'opera non implica il trasferimento dei suddetti diritti né l'esclusiva.

Le fotocopie per uso personale (cioè privato e individuale, con esclusione qualsiasi di strumenti di uso collettivo) possono essere effettuate con autorizzazione di Zanichelli editore S.p.A. e di Zanichelli editore S.p.A. e di S.I.A.E. alla legge 22 aprile 1941 n. 633. Le fotocopie possono essere effettuate negli esercizi commerciali convenzionati S.I.A.E. o con altre modalità indicate da S.I.A.E.

Per le ristampe ad uso non personale (ad esempio: professionale, economico, commerciale, strumenti di studio collettivi, ecc.) è necessaria la scrittura a mano dell'autore o concedere a pagamento l'autorizzazione a riprodurre un numero di pagine non superiore al 15% delle pagine del volume. Le richieste vanno inviate a:

CLEAP/Edi-Uniti
Città di Porta Romana, n. 108
20122 Milano
e-mail: autorizzazioni@edieuniti.org o sito web: www.edieuniti.org

L'utente, per quanto di propria specifica, constaterà con le proprie fidei del proprio catalogo editoriale.
Le foto presenti nel volume sono state inserite nelle bibliografie e commentate, oltre ai limiti del 15%, non essendo concorrente all'opera. Non possono comunque ritenere le opere di cui esiste, nel catalogo dell'editore, una successiva edizione.
Le opere presenti in cataloghi di altri editori o le opere analogiche. Nei contratti di cessione è esclusa, per titolarità, ogni ristampa, anche se si tratta di opere di pubblico dominio, o di opere di autore o di titolo d'autore. Per permesso di riproduzione, anche digitali, diverse fotocopie devono essere richieste a CLEAP/Edi-Uniti/Zanichelli.

Realizzazione editoriale:

- Coordinamento redazionale: Matteo Forneri, Anna Vivaldo
- Composizione, impiagnatura e disegni: Antonio Libero Morra
- Progetto grafico: Studio BvO, Bologna

Contributi:

- Rilettura critica: Roberto Borchia

Marchi registrati:

- Arduino è un marchio registrato di Arduino AG

Copertina:

- Progetto grafico: Miquel Sal & C., Bologna
- Ideazione: Studio BvO, Bologna
- Realizzazione: Roberto Marchetti e Francesca Ponti
- Immagine di copertina: knif_makariv/Shutterstock

Prima edizione: marzo 2018

Ristampa: prima tiratura

5 4 3 2 1 2018 2019 2020 2021 2022

 Zanichelli garantisce che le risorse digitali di questo volume sotto il suo controllo saranno accessibili, a partire dall'acquisto dell'esemplare nuovo, per tutta la durata della normale utilizzazione didattica dell'opera. Passato questo periodo, alcune o tutte le risorse potrebbero non essere più accessibili o disponibili; per maggiori informazioni, leggi my.zanichelli.it/uricatalogo

File per sintesi vocale

L'edizione mette a disposizione degli studenti non vedenti, ipovedenti, disabili motori o con disturbi specifici di apprendimento i file pdf in cui sono memorizzate le pagine di questo libro. Il formato del file permette l'ingrandimento dei caratteri del testo e la lettura mediante software adeguiti per questo. Le informazioni sui come scaricare i file sono sul sito <http://www.zanichelli.it/uricatalogo-educativi-speciali>

Grazie a chi segnala gli errori

Segnalate gli errori e le proposte di correzione su www.zanichelli.it/correzioni.

Controllate e inseriamo le eventuali correzioni nelle ristampe del libro.

Nello stesso sito trovate anche l'errata corrigé con l'elenco degli errori e delle correzioni.

Zanichelli editore S.p.A. opera con sistema qualità certificato CertiCarGraf n. 4/7 secondo la norma UNI EN ISO 9001:2015



Questo libro è stampato su carta che rispetta le foreste.
www.zanichelli.it/chi-siamo/sostenibilita

Stampa: Grafica Rigno
Via Lombardia 25, 40064 Tolara di Sotto, Ozzano Emilia (Bologna)
per conto di Zanichelli editore S.p.A.
Via Imero 34, 40126 Bologna