

# GUIDA ARDUINO DIGISPARK

## Potenziometro + Led RGB



Mattia Ruberto & Matteo Ghilardini

# SOMMARIO

<i>Sommario</i> .....	2
<i>Scopo</i> .....	3
Arduino Digispark.....	3
Potenziometro .....	4
Led RGB .....	4
<i>Schema Elettrico</i> .....	5
<i>Librerie</i> .....	6
Libreria Led RGB .....	6
Libreria Potenziometro.....	Errore. Il segnalibro non è definito.
<i>Utilizzo</i> .....	7
<i>Hardware</i> .....	7
<i>Software</i> .....	8

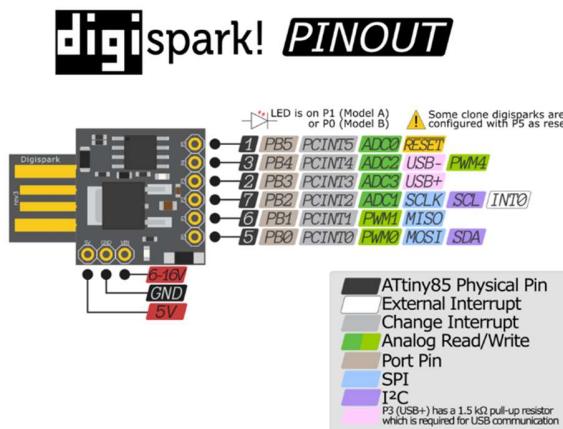
# Scopo

Lo scopo di questa guida è illustrare il funzionamento del circuito in modo che sia facilmente comprensibile anche agli utenti più inesperti. Illustreremo perciò ogni componente utilizzato e il funzionamento di essi singolarmente, così anche per il prodotto globale.

## Arduino Digispark

Arduino Digispark, così come tutti gli altri componenti della famiglia Arduino, è una scheda elettronica dotata di un microcontrollore. La funzionalità principale di Arduino è quella di realizzare in maniera pressoché semplice dei dispositivi di controllo oppure degli automatismi (specialmente nel caso di Arduino Digispark). Uno dei punti di forza di Arduino è la sua convenienza economica dal momento che le schede programmabili hanno prezzi veramente bassi (per Digispark meno di 5 CHF) e inoltre il software e il linguaggio di programmazione utilizzato sono Open Source (ossia gratis).

Per collegare elementi esterni alle schede si utilizzando dei pin che possono venir saldati sulle apposite interfacce. L'alimentazione (ossia il +) è indicata da "5V", mentre la terra (ossia il -) è indicata da "GND", mentre gli altri pin (da P0 a P5) possono assumere diverse funzionalità seguendo il seguente modello:



Per poter utilizzare il software di Arduino col Digispark sono necessari alcuni accorgimenti, per poter installare le schede è necessaria una connessione a internet (preferibilmente senza proxy):

- Nelle impostazioni di arduino (File→Impostazioni), nel campo “URL aggiuntive per il Gestore schede:” inserire l’URL [http://digistump.com/package\\_digistump\\_index.json](http://digistump.com/package_digistump_index.json);
- Riavviare il software (se procedendo qualcosa non va, riavviare il pc);
- Cliccando “Gestore schede” (Strumenti → Scheda) verrà aperta una schermata nella quale è presente una barra di ricerca, scriveteci “Digistump” e verrà mostrata una possibilità come quella da immagine:

#### Digistump AVR Boards by Digistump versione 1.6.7 INSTALLED

Schede incluse in questo pacchetto:

Digispark (Default - 16.5mhz), Digispark Pro (Default 16 Mhz), Digispark Pro (16 Mhz) (32 byte buffer), Digispark Pro (16 Mhz) (64 byte buffer),  
Digispark (16mhz - No USB), Digispark (8mhz - No USB), Digispark (1mhz - No USB).

[Online help](#)

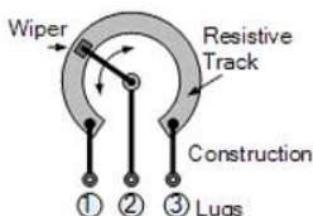
[More info](#)

nell'angolo in basso a destra di questa sarà presente il pulsante Installa (premerlo);

- Riavviare il software (se procedendo qualcosa non va, riavviare il pc);
- Nella selezione delle schede cercare e selezionare “Digispark (Default – 16.5 MHz)”;
- Per quanto riguarda la selezione della porta (COM...) dipende dal vostro computer e da quale porta usb utilizzerete per inserire il digispark.

## Potenziometro

Il potenziometro è una sorta di resistenza che però può essere modificata, ossia può essere gestita la sua resistenza elettrica. Nell'elettronica “semplice” viene utilizzato per modificare delle frequenze o per modificare la luminosità dei led, mentre nell'elettronica di Arduino i suoi utilizzi aumentano a dismisura grazie ad un semplice comando utilizzato molto spesso: `map(value, fromLow, fromHigh, toLow, toHigh)`. Tramite un `analogRead()` il potenziometro ritorna un valore fra 0 e 1023 perciò molte volte può essere utile magari rimappare l'intervallo fra magari 0 e 100 per ricevere la percentuale di rotazione (se utilizziamo un potenziometro rotativo), in questo caso usiamo il comando `map(analogRead(pinPotenziometro),0,1023,0,100)` e otterremo il valore percentuale di quanto è stato rotato il potenziometro (sempre se utilizziamo un potenziometro rotativo).

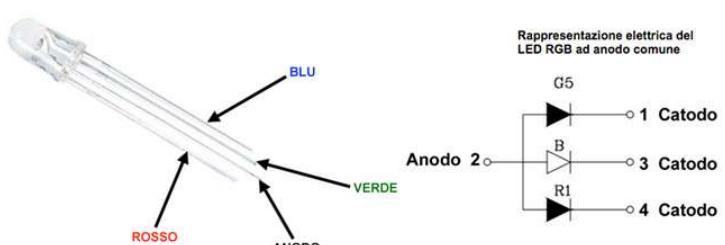


I potenziometri hanno un multiplo di 3 pin (il più comune e quello che usiamo anche noi che ne ha infatti 3) dove i 2 laterali vengono collegati al “+” e al “-”, mentre quello centrale ritorna il valore desiderato. Come nelle resistenze normali, anche i potenziometri non hanno polarità perciò è indifferente quale dei pin esterni inseriamo nel “+” o rispettivamente nel “-”, ma se non ritorna il valore che ci aspettiamo dovremo invertirne il senso.

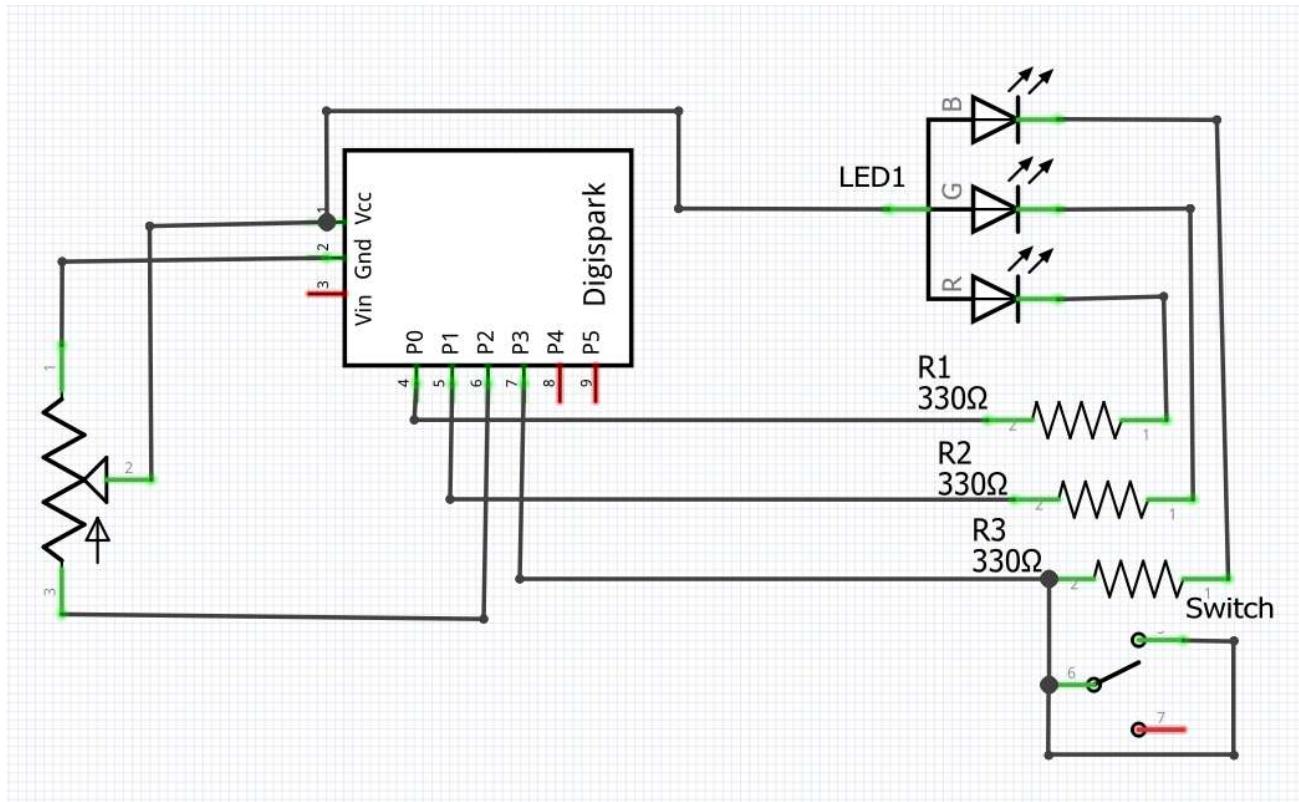
## Led RGB

Un led RGB è un solo led con al suo interno 3 led rispettivamente di colore rosso, verde e blu. I led (sia quelli RGB che quelli semplici) non possono essere collegati direttamente al polo positivo (o negativo, dipende se anodo o catodo comune. Vedi sotto) perché subirebbero un voltaggio troppo alto rispetto a quello supportato, per questo dobbiamo utilizzare delle resistenze. Il minimo per il led che utilizziamo noi è una resistenza da  $330\ \Omega$ .

Esistono globalmente 2 tipi di led RGB, quelli con l'anodo (“+”) comune o con catodo (“-”) comune. Nel nostro caso utilizziamo un led anodo comune.



# Schema Elettrico



# Librerie

Tutte le librerie realizzate per questo progetto sono state realizzate nel linguaggio di C++, come d'altronde anche il software di Arduino.

Per includere una libreria (o una cartella di librerie) dobbiamo spostarci nella cartella ".\Arduino\libraries" (se non la trovate, premete tasto destro sull'icona dell'editor di Arduino, quindi "Apri percorso File"), all'interno di questa cartella creiamo a sua volta una cartella chiamata come la libreria o come il componente al quale fa riferimento, all'interno di questa cartella, copiamo sia l'header che la libreria stessa.

Quando apriamo l'editor di Arduino, selezionare "Sketch", quindi "#include libreria", e ora scegliere la libreria desiderata (si chiamerà come la cartella che avete creato in precedenza).

Tutte le nostre librerie sono composte da un'interfaccia (chiamata nel linguaggio specifico di "C" Header, ed è un file con estensione ".h") e la libreria in sé (con estensione ".cpp") che estende l'interfaccia.

Sia l'Header, che la libreria devono includere l'interfaccia "Arduino.h".

## Libreria Led RGB

L'Header contiene:

- ◆ 3 attributi: uno per ogni pin corrispondente ad un colore diverso;
- ◆ 5 metodi:
  - `setLedPin(int myRedPin, int myGreenPin, int myBluePin);`
  - `setColor(int red, int green, int blue);`
  - `setRed(int red);`
  - `setGreen(int green);`
  - `setBlue(int blue);`

La libreria contiene tutti i metodi dell'Header dichiarandoli in questo modo:

```
void <NomeHeader>::<metodo>(<tipoParametro> <parametro>){};
```

Ecco cosa fa nello specifico ogni metodo:

- ◆ **setLedPin**: in base ai parametri che riceve, attribuisce tali valori agli attributi che indicano i pin dei colori;
- ◆ **setColor**: rappresenta il colore sui vari pin a seconda dei valori che riceve per ogni colore;
- ◆ **setRed, setGreen, setBlue**: modificano semplicemente solo il loro valore e lo rappresentano.

## Libreria Potenziometro

L'Header contiene:

- ◆ 2 metodi:
  - `int setRange(int valuePotentiometer, int valueMinPotentiometer, int valueMaxPotentiometer, int valueMin, int valueMax);`
  - `int getValue(int port);`

La libreria contiene tutti i metodi dell'Header dichiarandoli in questo modo:

```
void <NomeHeader>::<metodo>(<tipoParametro> <parametro>){};
```

Ecco cosa fa nello specifico ogni metodo:

- ◆ **getMappedValue**: in base ai parametri che riceve, effettua una mappatura del valore del potenziometro (`valuePotentiometer`) dai valori originari (`valueMinPotentiometer` e `valueMaxPotentiometer`) ai nuovi valori scelti (`valueMin` e `valueMax`) e ritorna il valore mappato.  
Se il potenziometro è collegato direttamente al circuito (senza resistenze nel mezzo), i valori per `valueMinPotentiometer` e `valueMaxPotentiometer` sono rispettivamente 0 e 1023;
- ◆ **getValue**: riceve la porta analogica dalla quale leggere il valore del potenziometro e ritorna tale valore.

# Utilizzo

## Hardware

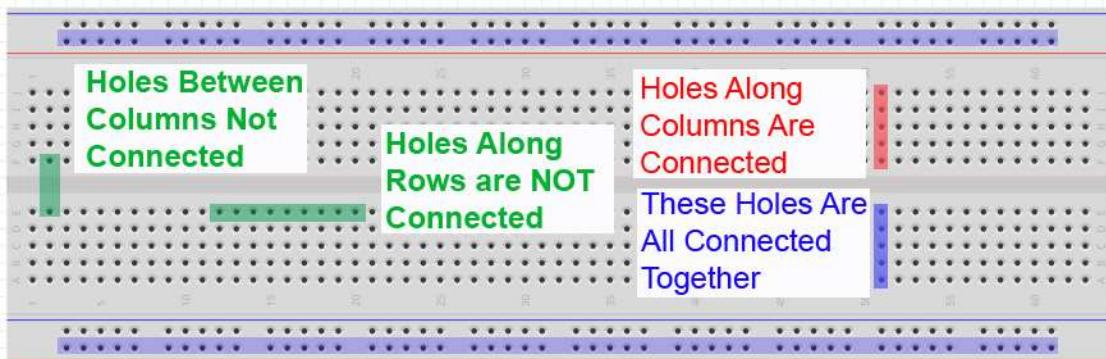
I componenti da utilizzare per questo progetto sono i 3 citati più e più volte all'interno di questa guida:

- Arduino Digispark;
- Potenziometro (rotativo);
- Led RGB anodo comune;

Per costruire il circuito dobbiamo fissare il potenziometro e il led su di una breadboard (circuito provvisorio) o su una veroboard (circuito definitivo), non è necessario metterli in uno schema preciso a patto che abbia un senso.

⚠ Fare attenzione alle piste delle board per evitare cortocircuiti ⚠

(In caso di dubbio, eccoti un'immagine che mostra com'è fatta una breadboard di Arduino al suo interno)



Collegare i pin esterni del potenziometro ad alimentazione e GND (è indifferente quale ad uno o all'altro, nel caso in cui i valori non corrispondano a quelli desiderati è sufficiente invertirli), mentre il pin centrale del potenziometro deve essere collegato alla porta P2 del Digispark.

Collegare al pin più lungo del led RGB l'alimentazione (anodo in comune). A tutti gli altri pin collegiamo delle resistenze da  $330\ \Omega$  e in serie anche i connettori per portare il segnale alle corrispettive porte del Digispark, nello specifico:

- Il rosso alla porta P0;
- Il verde alla porta P1;
- Il blu alla porta P4;

In caso di dubbio su quale sia il pin corrispondente ad un determinato colore, consulta la documentazione del led RGB a pagina 4.

## Software

- Il primo esempio di codice che abbiamo realizzato con questi componenti si occupa di far scorrere una gamma di colori a seconda del valore del potenziometro, ossia, a seconda del valore del potenziometro, viene mostrato un colore diverso partendo dal rosso arrivando al viola passando da tutti i colori primari e secondari (rosso <-> giallo <-> verde <-> azzurro <-> blu <-> viola).

Le prime linee di codice servono ad includere le librerie necessarie al programma (solo quelle che abbiamo creato noi perché quella base “Arduino.h” viene inclusa automaticamente):

```
#include <LibraryLedRGB.h>
#include <LibraryPotentiometer.h>
```

Le linee successive le utilizziamo per dichiarare le variabili che ci serviranno più tardi tra queste anche l'istanza delle nostre librerie :

```
LibraryPotentiometer libraryPotentiometer;
LibraryLedRGB libraryLedRGB;

int valuePotentiometer;
int rangeValuePotentiometer;
```

Nel metodo setup richiamiamo il metodo che abbiamo creato nella libreria che si occupa di attribuire ad ogni colore del led RGB un pin di Digispark:

```
void setup() {
    libraryLedRGB.setLedPin(0, 1, 4);
}
```

Adesso entriamo nella parte più sostanziosa del programma: il metodo loop.

La prima cosa che facciamo nel loop è leggere il valore del potenziometro, memorizzarlo in una variabile e poi rimappare il valore di questa variabile passando da valori 0-1023, a valori 0-6:

```
valuePotentiometer = libraryPotentiometer.getValue(1);
rangeValuePotentiometer = libraryPotentiometer.getMappedValue(
    valuePotentiometer, 0, 1023, 1, 6
);
```



L'ultima parte di codice è una serie di IF che a seconda del valore del potenziometro, richiama il metodo setColor() della nostra libreria passandogli dei valori diversi a seconda del colore che dovrà essere rappresentato:

```
if(rangeValuePotentiometer > 0 && rangeValuePotentiometer <= 1){  
    libraryLedRGB.setColor(255,0,0);  
}else if(rangeValuePotentiometer > 1 && rangeValuePotentiometer <= 2){  
    libraryLedRGB.setColor(255,255,0);  
}else if(rangeValuePotentiometer > 2 && rangeValuePotentiometer <= 3){  
    libraryLedRGB.setColor(0,255,0);  
}else if(rangeValuePotentiometer > 3 && rangeValuePotentiometer <= 4){  
    libraryLedRGB.setColor(0,255,255);  
}else if(rangeValuePotentiometer > 4 && rangeValuePotentiometer <= 5){  
    libraryLedRGB.setColor(0,0,255);  
}else if(rangeValuePotentiometer > 5 && rangeValuePotentiometer <= 6){  
    libraryLedRGB.setColor(255,0,255);  
}
```

- Il secondo esempio che abbiamo pensato, scorre tutte le tonalità di ogni colore separatamente a seconda del valore del potenziometro. Ogni colore viene rappresentato per la durata di 1/3 di giro del potenziometro, all'interno di questo 1/3 vengono passati 255 valori che può assumere il colore.

Le prime linee di codice servono ad includere le librerie necessarie al programma (solo quelle che abbiamo creato noi perché quella base "Arduino.h" viene inclusa automaticamente):

```
#include <LibraryLedRGB.h>  
#include <LibraryPotentiometer.h>
```

Le linee successive le utilizziamo per dichiarare le variabili che ci serviranno più tardi, tra queste anche l'istanza delle nostre librerie:

```
LibraryPotentiometer libraryPotentiometer;  
LibraryLedRGB libraryLedRGB;  
  
int valuePotentiometer;  
int rangeValuePotentiometer;
```

Nel metodo setup richiamiamo il metodo che abbiamo creato nella libreria che si occupa di attribuire ad ogni colore del led RGB un pin di Digispark:

```
void setup() {  
    libraryLedRGB.setLedPin(0,1,4);  
}
```

Adesso entriamo nella parte più sostanziosa del programma: il metodo loop.

La prima cosa che facciamo nel loop è leggere il valore del potenziometro:

```
valuePotentiometer = libraryPotentiometer.getValue(1);
```

L'ultima parte di codice è una serie di IF che a seconda del valore del potenziometro, mostra un colore diverso in tutte le sue tonalità (0 a 255 per ogni colore):

```
if(rangeValuePotentiometer < 256){
    libraryLedRGB.setRed(rangeValuePotentiometer);
    libraryLedRGB.setGreen(0);
    libraryLedRGB.setBlue(0);
}else if(rangeValuePotentiometer < 511){
    libraryLedRGB.setRed(0);
    libraryLedRGB.setGreen(rangeValuePotentiometer-255);
    libraryLedRGB.setBlue(0);
}else {
    libraryLedRGB.setRed(0);
    libraryLedRGB.setGreen(0);
    libraryLedRGB.setBlue(rangeValuePotentiometer-510);
}
```

- Il terzo esempio che abbiamo pensato, scorre tutte le tonalità di ogni colore separatamente a seconda del valore del potenziometro. Ogni volta che il potenziometro torna a 0 (o 255 a seconda della polarità) viene cambiato il colore da scorrere.

Le prime linee di codice servono ad includere le librerie necessarie al programma (solo quelle che abbiamo creato noi perché quella base “Arduino.h” viene inclusa automaticamente):

```
#include <LibraryLedRGB.h>
#include <LibraryPotentiometer.h>
```

Le linee successive le utilizziamo per dichiarare le variabili che ci serviranno più tardi, tra queste anche l'istanza delle nostre librerie:

```
LibraryPotentiometer libraryPotentiometer;
LibraryLedRGB libraryLedRGB;

int valuePotentiometer;
int rangeValuePotentiometer;
int counter = 0;
bool ceck = true;
```

Nel metodo setup richiamiamo il metodo che abbiamo creato nella libreria che si occupa di attribuire ad ogni colore del led RGB un pin di Digispark:

```
void setup() {
    libraryLedRGB.setLedPin(0,1,4);
}
```

Adesso entriamo nella parte più sostanziosa del programma: il metodo loop.  
La prima cosa che facciamo nel loop è leggere il valore del potenziometro:

```
valuePotentiometer = libraryPotentiometer.getValue(1);
```

A questo punto verifichiamo che il valore del potenziometro sia “valido” (il valore è rappresentabile), in caso positivo settiamo la variabile check a true:

```
if(valuePotentiometer > 50){  
    ceck = true;  
}
```

Se check è true e se il valore del potenziometro è inferiore a 30 (quello che consideriamo 0 per evitare eventuali problemi di hardware), passiamo al colore successivo. Se sono già stati passati tutti i colori, ricomincia il ciclo:

```
if(ceck){  
    if(valuePotentiometer < 30){  
        counter++;  
        if(counter == 3){  
            counter = 0;  
        }  
        ceck = false;  
    }  
}
```

A seconda del valore del contatore (corrispondente ad ogni colore), mostra il colore indicato con tonalità definita dal valore del potenziometro:

```
if(counter == 0){  
    libraryLedRGB.setColor(valuePotentiometer,0,0);  
}else if(counter == 1){  
    libraryLedRGB.setColor(0,valuePotentiometer,0);  
}else if(counter == 2){  
    libraryLedRGB.setColor(0,0,valuePotentiometer);  
}
```

# GUIDA ARDUINO DIGISPARK

## Pulsante + Led



Mattia Ruberto & Matteo Ghilardini

# SOMMARIO

<i>Sommario</i> .....	2
<i>Scopo</i> .....	3
<i>Componenti</i> .....	3
Arduino Digispark.....	3
Pulsante.....	4
Led RGB .....	4
<i>Schema Elettrico</i> .....	5
<i>Librerie</i> .....	6
Libreria Led .....	6
<i>Utilizzo</i> .....	6
<i>Hardware</i> .....	8
<i>Software (ogni codice dovrà essere mostrato)</i> .....	<i>Errore. Il segnalibro non è definito.</i>

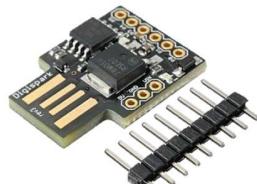
# Scopo

Lo scopo di questa guida è illustrare il funzionamento del circuito in modo che sia facilmente comprensibile anche agli utenti più inesperti. Illustreremo perciò ogni componente utilizzato e il funzionamento di essi singolarmente, così anche per il prodotto globale.

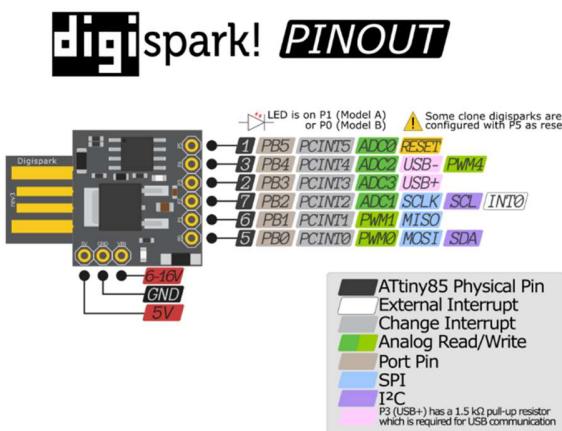
# Componenti

## Arduino Digispark

Arduino Digispark, così come tutti gli altri componenti della famiglia Arduino, è una scheda elettronica dotata di un microcontrollore. La funzionalità principale di Arduino è quella di realizzare in maniera pressoché semplice dei dispositivi di controllo oppure degli automatismi (specialmente nel caso di Arduino Digispark). Uno dei punti di forza di Arduino è la sua convenienza economica dal momento che le schede programmabili hanno prezzi veramente bassi (per Digispark meno di 5 CHF) e inoltre il software e il linguaggio di programmazione utilizzato sono Open Source (ossia gratis).



Per collegare elementi esterni alle schede si utilizzando dei pin che possono venir saldati sulle apposite interfacce. L'alimentazione (ossia il +) è indicata da "5V", mentre la terra (ossia il -) è indicata da "GND", mentre gli altri pin (da P0 a P5) possono assumere diverse funzionalità seguendo il seguente modello:



Per poter utilizzare il software di Arduino col Digispark sono necessari alcuni accorgimenti, per poter installare le schede è necessaria una connessione a internet (preferibilmente senza proxy):

- Nelle impostazioni di arduino (File→Impostazioni), nel campo "URL aggiuntive per il Gestore schede:" inserire l'URL [http://digistump.com/package\\_digistump\\_index.json](http://digistump.com/package_digistump_index.json);
- Riavviare il software (se procedendo qualcosa non va, riavviare il pc);
- Cliccando "Gestore schede" (Strumenti → Scheda) verrà aperta una schermata nella quale è presente una barra di ricerca, scriveteci "Digistump" e verrà mostrata una possibilità come quella da immagine:

#### Digistump AVR Boards by Digistump versione 1.6.7 INSTALLED

Schede incluse in questo pacchetto:

Digispark (Default - 16.5Mhz), Digispark Pro (Default 16 Mhz), Digispark Pro (16 Mhz) (32 byte buffer), Digispark Pro (16 Mhz) (64 byte buffer),  
Digispark (16mhz - No USB), Digispark (8mhz - No USB), Digispark (1mhz - No USB).

[Online help](#)

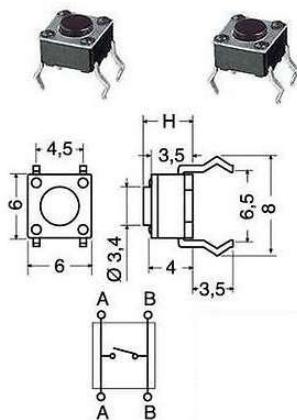
[More info](#)

nell'angolo in basso a destra di questa sarà presente il pulsante Installa (premerlo);

- Riavviare il software (se procedendo qualcosa non va, riavviare il pc);
- Nella selezione delle schede cercare e selezionare “Digispark (Default – 16.5 MHz)”;
- Per quanto riguarda la selezione della porta (COM...) dipende dal vostro computer e da quale porta usb utilizzerete per inserire il digispark.

## Pulsante

Un pulsante si comporta come se fosse un cavo che viene collegato e scollegato. La funzione corrispondente al fatto che è collegato, sarebbe quando viene premuto il pulsante, mentre quando viene rilasciato il circuito viene aperto (e quindi scollegato).



Esistono numerosi tipi diversi di pulsanti, ma quelli più comuni e più utilizzati sono quelli a 4 pin come quello mostrato nelle foto. I pin sono collegati a coppie e perciò per collegarli in modo da rilevare la pressione del pulsante bisogna seguire lo schema a sinistra (collegare un polo A, con un polo B).

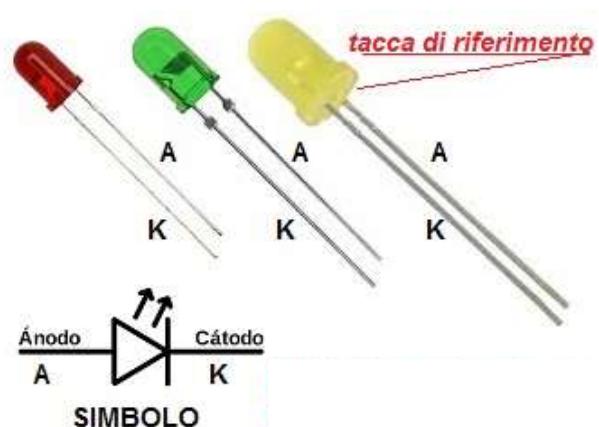


I pulsanti non hanno nessun circuito sensibile al loro interno, quindi non è necessario prestare attenzione ai voltaggi che gli vengono impressi o alla loro polarità. Questo perché, come già detto, i pulsanti sono esattamente come se fossero due cavi che vengono collegati e scollegati a seconda del fatto che sia stato premuto o meno il pulsante.

## Led

Le LED non possono essere collegati direttamente al polo positivo o negativo della corrente perché subirebbero un voltaggio troppo alto rispetto a quello supportato, per questo dobbiamo utilizzare delle resistenze. Il minimo per il led che utilizziamo noi è una resistenza da  $330\ \Omega$ .

Per distinguere il pin positivo e quello negativo è sufficiente guardare la lunghezza del suddetto pin e la posizione della tacca di riferimento (Vedi immagine a fianco). Il pin più lungo rappresenta il polo positivo, quindi il più corto quello negativo. Il polo positivo è identificabile anche dalla presenza della tacca.



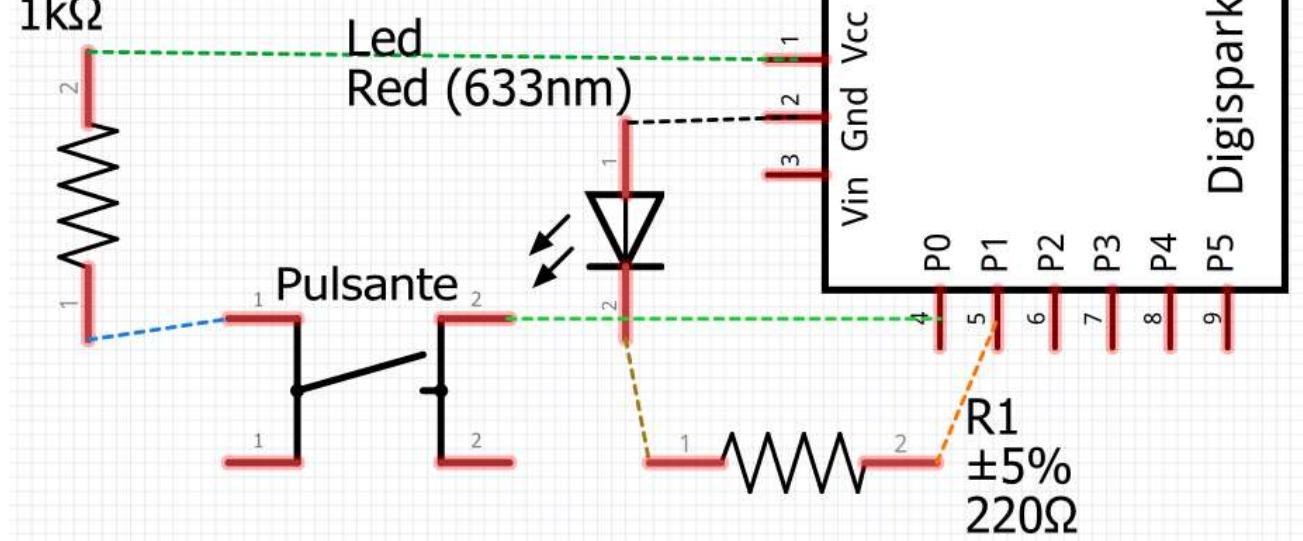
DIODI LED diam. 5 mm

# Schema Elettrico

Resistenza

$\pm 5\%$

$1k\Omega$



# Librerie

Tutte le librerie realizzate per questo progetto sono state realizzate nel linguaggio di C++, come d'altronde anche il software di Arduino.

Per includere una libreria (o una cartella di librerie) dobbiamo spostarci nella cartella ".\Arduino\libraries" (se non la trovate, premete tasto destro sull'icona dell'editor di Arduino, quindi "Apri percorso File"), all'interno di questa cartella creiamo a sua volta una cartella chiamata come la libreria o come il componente al quale fa riferimento, all'interno di questa cartella, copiamo sia l'header che la libreria stessa.

Quando apriamo l'editor di Arduino, selezionare "Sketch", quindi "#include libreria", e ora scegliere la libreria desiderata (si chiamerà come la cartella che avete creato in precedenza).

Tutte le nostre librerie sono composte da un'interfaccia (chiamata nel linguaggio specifico di "C" Header, ed è un file con estensione ".h") e la libreria in sé (con estensione ".cpp") che estende l'interfaccia.

Sia l'Header, che la libreria devono includere l'interfaccia "Arduino.h".

## Libreria Led

L'Header contiene:

- ◆ 2 attributi:
  - led: indica il pin del led;
  - state\_led: indica lo stato del led (acceso / spento);
- ◆ 5 metodi:
  - setLedPin(**int** ledPort);
  - powerOn();
  - powerOff();
  - setLed(**bool** stato\_led);
  - blink(**int** frequency);

La libreria contiene tutti i metodi dell'Header dichiarandoli in questo modo:

```
void <NomeHeader>::<metodo>(<tipoParametro> <parametro>){};
```

Ecco cosa fa nello specifico ogni metodo:

- ◆ **setLedPin**: in base al parametro che riceve, attribuisce il valore all'attributo che indica il pin del led;
- ◆ **powerOn**: setta il valore di state\_led ad HIGH (che corrisponde a "1" o a "true"), a seconda della polarità del led, lo accende o lo spegne (se il contatto è al polo positivo, lo accende);
- ◆ **powerOff**: setta il valore di state\_led a LOW (che corrisponde a "0" o a "false"), a seconda della polarità del led, lo accende o lo spegne (se il contatto è al polo positivo, lo spegne);
- ◆ **setLed**: rappresenta lo stato del led ricevuto come parametro (acceso o spento);
- ◆ **blink**: accende e spegne il led con un intervallo definito dal parametro "frequency";

## Libreria Bottone (Pulsante)

L'Header contiene:

- ◆ 6 attributi:
  - `button`: indica il pin del bottone;
  - `state_button`: indica lo stato del bottone (premuto o meno);
  - `lastButtonState`: indica l'ultimo stato del bottone (necessario per l'anti-rimbalzo);
  - `lastDebounceTime`: memorizza i millisecondi da quando il bottone è stato premuto;
  - `debounceDelay`: indica il tempo per garantire l'anti-rimbalzo del bottone;
  - `ledState`: indica lo stato del led che potrebbe venir "toggleato";
- ◆ 5 metodi:
  - `setButtonPin(int buttonPort)`;
  - `boolean getStateButton()`;
  - `boolean toggle()`;

La libreria contiene tutti i metodi dell'Header dichiarandoli in questo modo:

```
void <NomeHeader>::<metodo>(<tipoParametro> <parametro>){};
```

Ecco cosa fa nello specifico ogni metodo:

- ◆ **setButtonPin**: in base al parametro che riceve, attribuisce il valore all'attributo che indica il pin del bottone;
- ◆ **getStateButton**: ritorna il valore del bottone. A seconda della polarità cambia, ma un valore viene ritornato quando il bottone è premuto, mentre l'opposto quando non lo è;
- ◆ **toggle**: ritorna il valore di ledState invertito, il metodo contiene anche un controllo per l'anti-rimbalzo;

# Utilizzo

## Hardware

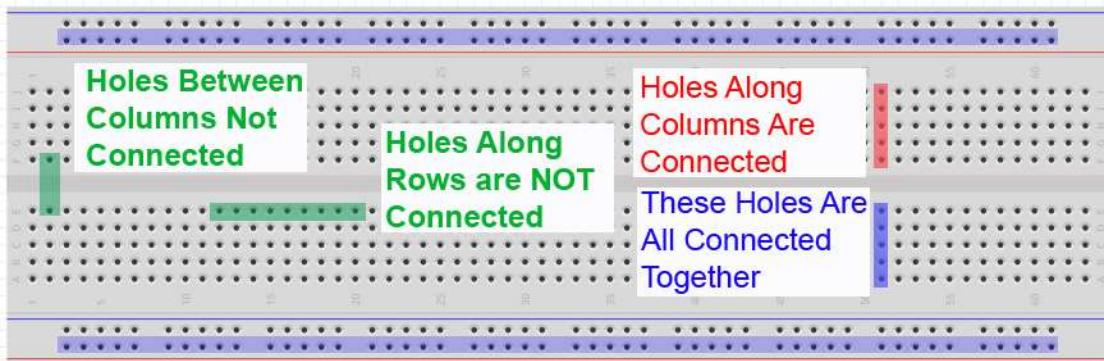
I componenti da utilizzare per questo progetto sono i 3 citati più e più volte all'interno di questa guida:

- Arduino Digispark;
- Pulsante;
- Led (di un qualunque colore);

Per costruire il circuito dobbiamo fissare il pulsante e il led su di una breadboard (circuito provvisorio) o su una veroboard (circuito definitivo), non è necessario metterli in uno schema preciso a patto che abbia un senso.

**⚠ Fare attenzione alle piste delle board per evitare cortocircuiti ⚠**

*(In caso di dubbio, eccoti un'immagine che mostra com'è fatta una breadboard di Arduino al suo interno)*



Per costruire il circuito dobbiamo fissare il pulsante in modo che le 2 coppie di pin non siano in contatto fra loro. Il led allo stesso modo può essere montato in qualunque modo a patto che i 2 pin non siano connessi.

Per gestire la corrente nel circuito abbiamo bisogno di 2 resistenze

- $220\Omega$  per il led: deve essere collegata in serie fra il collegamento al Digispark (P1) e il polo positivo del led (quello più lungo).
- $10K\Omega$  per il pull-Up o pull-Down del pulsante: deve essere collegata fra il pin del pulsante diagonalmente opposto a quello al quale è collegato il pin di lettura del Digispark (P0) e, a seconda del fatto che viene utilizzata per fare pull-Up o pull-Down, rispettivamente al +5V o al GND.

In caso di dubbio su come collegare il pulsante, consulta la documentazione del pulsante a pagina 4.

## Software

- Il primo esempio di codice che abbiamo realizzato con questi componenti si occupa di far lampeggiare il led quando il bottone rimane premuto.

Le prime linee di codice servono ad includere le librerie necessarie al programma (solo quelle che abbiamo creato noi perché quella base “Arduino.h” viene inclusa automaticamente):

```
#include <LibraryLed.h>
#include <LibraryButton.h>
```

Le linee successive le utilizziamo per dichiarare le istanze delle nostre librerie:

```
LibraryLed libraryLed;
LibraryButton libraryButton;
```

Nel metodo setup richiamiamo i metodi che abbiamo creato nelle librerie che si occupano di attribuire al pin di scrittura del led e a quello di lettura del bottone un pin di Digispark:

```
void setup() {
    libraryLed.setLedPin(1);
    libraryButton.setButtonPin(0);
}
```

Adesso entriamo nella parte più sostanziosa del programma: il metodo loop.

La prima cosa che facciamo nel loop è leggere lo stato del bottone memorizzarlo in una variabile:

```
boolean stato_bot = libraryButton.getStateButton();
```

L’ultima parte di codice controlla se il bottone è premuto o meno, in caso positivo il led lampeggerà, altrimenti rimarrà spento:

```
if (stato_bot == HIGH) {
    libraryLed.blink(600);
} else {
    libraryLed.powerOff();
}
```

Il secondo esempio che abbiamo pensato, inverte lo stato del led ogni qualvolta che il bottone viene premuto.

Le prime linee di codice servono ad includere le librerie necessarie al programma (solo quelle che abbiamo creato noi perché quella base “Arduino.h” viene inclusa automaticamente):

```
#include <LibraryLed.h>
#include <LibraryButton.h>
```

Le linee successive le utilizziamo per dichiarare le istanze delle nostre librerie:

```
LibraryLed libraryLed;
LibraryButton libraryButton;
```

Nel metodo setup richiamiamo i metodi che abbiamo creato nelle librerie che si occupano di attribuire al pin di scrittura del led e a quello di lettura del bottone un pin di Digispark:

```
void setup() {  
    libraryLed.setLedPin(1);  
    libraryButton.setButtonPin(0);  
}
```

Adesso entriamo nella parte più sostanziosa del programma: il metodo loop.

La prima cosa che facciamo nel loop è invertire una variabile chiamata stato\_led tramite il metodo toggle:

```
boolean stato_led = libraryButton.toggle();
```

In fine rappresentiamo lo stato di tale variabile:

```
libraryLed.setLed(stato_led);
```

- Il terzo esempio che abbiamo pensato, quando il bottone viene premuto, accende il led per 1 secondo, se al termine di questo secondo il bottone è ancora premuto, il led comincia a lampeggiare con una frequenza di 20 millisecondi per 1500 millisecondi.

Le prime linee di codice servono ad includere le librerie necessarie al programma (solo quelle che abbiamo creato noi perché quella base “Arduino.h” viene inclusa automaticamente):

```
#include <LibraryLed.h>  
#include <LibraryButton.h>
```

Le linee successive le utilizziamo per dichiarare le istanze delle nostre librerie:

```
LibraryLed libraryLed;  
LibraryButton libraryButton;
```

Nel metodo setup richiamiamo i metodi che abbiamo creato nelle librerie che si occupano di attribuire al pin di scrittura del led e a quello di lettura del bottone un pin di Digispark:

```
void setup() {  
    libraryLed.setLedPin(1);  
    libraryButton.setButtonPin(0);  
}
```

Adesso entriamo nella parte più sostanziosa del programma: il metodo loop.  
La prima cosa che facciamo nel loop è leggere lo stato del bottone:

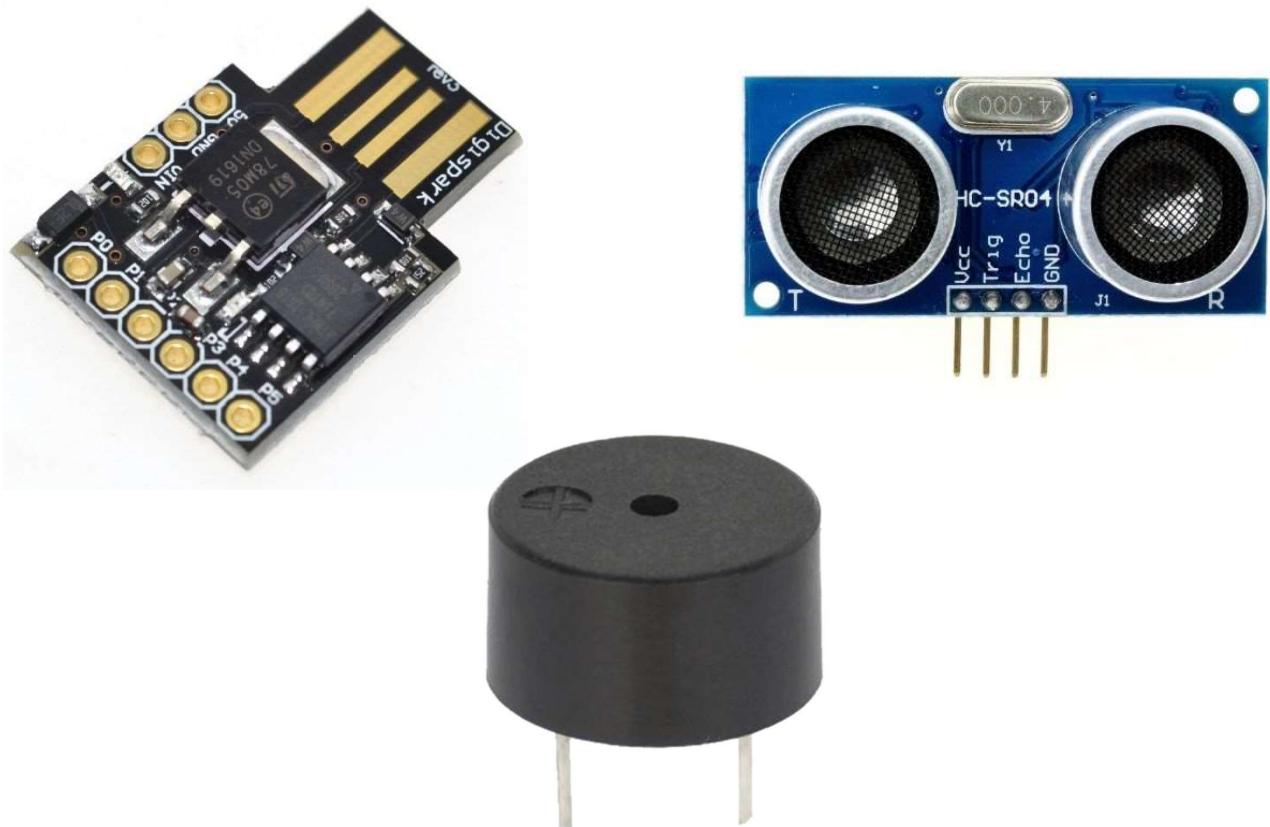
```
boolean stato_bot = libraryButton.getStateButton();
```

In fine, se il bottone viene premuto, accendiamo il led per 1 secondo. Se al termine di questo secondo il bottone è ancora premuto, il led comincerà a lampeggiare per 1500 millisecondi con una frequenza di 20 millisecondi. Se invece non è premuto, il led si spegne:

```
if(stato_bot == HIGH){
    libraryLed.powerOn();
    delay(1000);
    stato_bot = libraryButton.getStateButton();
    if(stato_bot == HIGH){
        unsigned long currentMilles = millis();
        while((millis() - currentMilles) < 1500){
            libraryLed.blink(20);
        }
    }
}else{
    libraryLed.powerOff();
}
```

# GUIDA ARDUINO DIGISPARK

## Ultrasuoni + Cicalino



Mattia Ruberto & Matteo Ghilardini

# SOMMARIO

<i>Sommario</i> .....	2
<i>Scopo</i> .....	3
<i>Componenti</i> .....	3
Arduino Digispark.....	3
Sensore a Ultrasuoni .....	4
Cicalino .....	4
<i>Schema Elettrico</i> .....	5
<i>Librerie</i> .....	6
Libreria Buzzer (Cicalino) .....	6
Libreria UltraSound (Sensore a Ultrasuoni).....	6
<i>Utilizzo</i> .....	7
<i>Hardware</i> .....	7
<i>Software</i> .....	9

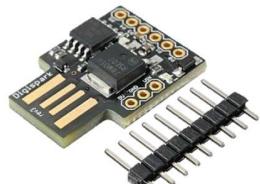
# Scopo

Lo scopo di questa guida è illustrare il funzionamento del circuito in modo che sia facilmente comprensibile anche agli utenti più inesperti. Illustreremo perciò ogni componente utilizzato e il funzionamento di essi singolarmente, così anche per il prodotto globale.

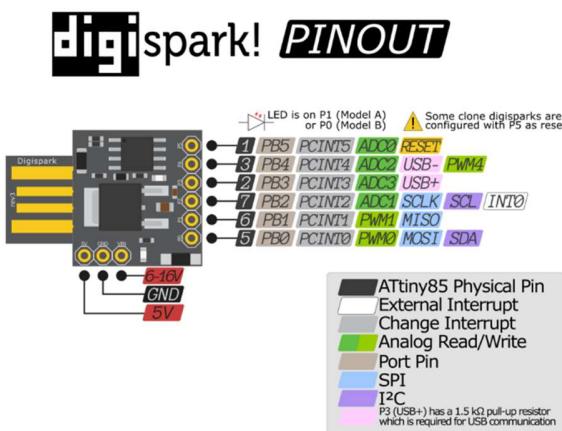
# Componenti

## Arduino Digispark

Arduino Digispark, così come tutti gli altri componenti della famiglia Arduino, è una scheda elettronica dotata di un microcontrollore. La funzionalità principale di Arduino è quella di realizzare in maniera pressoché semplice dei dispositivi di controllo oppure degli automatismi (specialmente nel caso di Arduino Digispark). Uno dei punti di forza di Arduino è la sua convenienza economica dal momento che le schede programmabili hanno prezzi veramente bassi (per Digispark meno di 5 CHF) e inoltre il software e il linguaggio di programmazione utilizzato sono Open Source (ossia gratis).



Per collegare elementi esterni alle schede si utilizzando dei pin che possono venir saldati sulle apposite interfacce. L'alimentazione (ossia il +) è indicata da "5V", mentre la terra (ossia il -) è indicata da "GND", mentre gli altri pin (da P0 a P5) possono assumere diverse funzionalità seguendo il seguente modello:



Per poter utilizzare il software di Arduino col Digispark sono necessari alcuni accorgimenti, per poter installare le schede è necessaria una connessione a internet (preferibilmente senza proxy):

- Nelle impostazioni di arduino (File→Impostazioni), nel campo "URL aggiuntive per il Gestore schede:" inserire l'URL [http://digistump.com/package\\_digistump\\_index.json](http://digistump.com/package_digistump_index.json);
- Riavviare il software (se procedendo qualcosa non va, riavviare il pc);
- Cliccando "Gestore schede" (Strumenti → Scheda) verrà aperta una schermata nella quale è presente una barra di ricerca, scriveteci "Digistump" e verrà mostrata una possibilità come quella da immagine:

#### Digistump AVR Boards by Digistump versione 1.6.7 INSTALLED

Schede incluse in questo pacchetto:

Digispark (Default - 16.5Mhz), Digispark Pro (Default 16 Mhz), Digispark Pro (16 Mhz) (32 byte buffer), Digispark Pro (16 Mhz) (64 byte buffer),  
Digispark (16mhz - No USB), Digispark (8mhz - No USB), Digispark (1mhz - No USB).

[Online help](#)

[More info](#)

nell'angolo in basso a destra di questa sarà presente il pulsante Installa (premerlo);

- Riavviare il software (se procedendo qualcosa non va, riavviare il pc);
- Nella selezione delle schede cercare e selezionare “Digispark (Default – 16.5 MHz)”;
- Per quanto riguarda la selezione della porta (COM...) dipende dal vostro computer e da quale porta usb utilizzerete per inserire il digispark.

## Sensore a Ultrasuoni

Un sensore che lavora con gli ultrasuoni sfrutta le onde emesse da qualunque rumore per percepire la distanza dell'oggetto che ha fatto rimbalzare tale impulso.

Esistono diversi tipi di sensori ad ultrasuoni che si differenziano gli uni dagli altri da potenza, frequenza o portata del segnale, ma concettualmente sono tutti pressoché identici nel funzionamento e nella struttura.

Quelli più basilari, come anche il nostro, sono composti da 2 “antenne” (un'emittente e una ricevente), 2 pin per l'elettricità (VCC e GND) e 2 pin per la lettura dei dati.



Il funzionamento di tale apparecchio si basa in realtà sul tempo che impiega il segnale a tornare alla sorgente, in seguito a questo, utilizzando la semplice formula fisica  $S = V * T$  (conoscendo la velocità del suono che è di circa  $343 \frac{m}{s}$  alla temperatura di  $20^\circ C$ ) è in grado di definire la distanza dall'oggetto con il calcolo  $S = 343 * T$  (tenendo conto che il tempo dovrà essere in secondi per mantenere valida questa formula).

## Cicalino

Concettualmente il cicalino è un apparecchio estremamente banale:

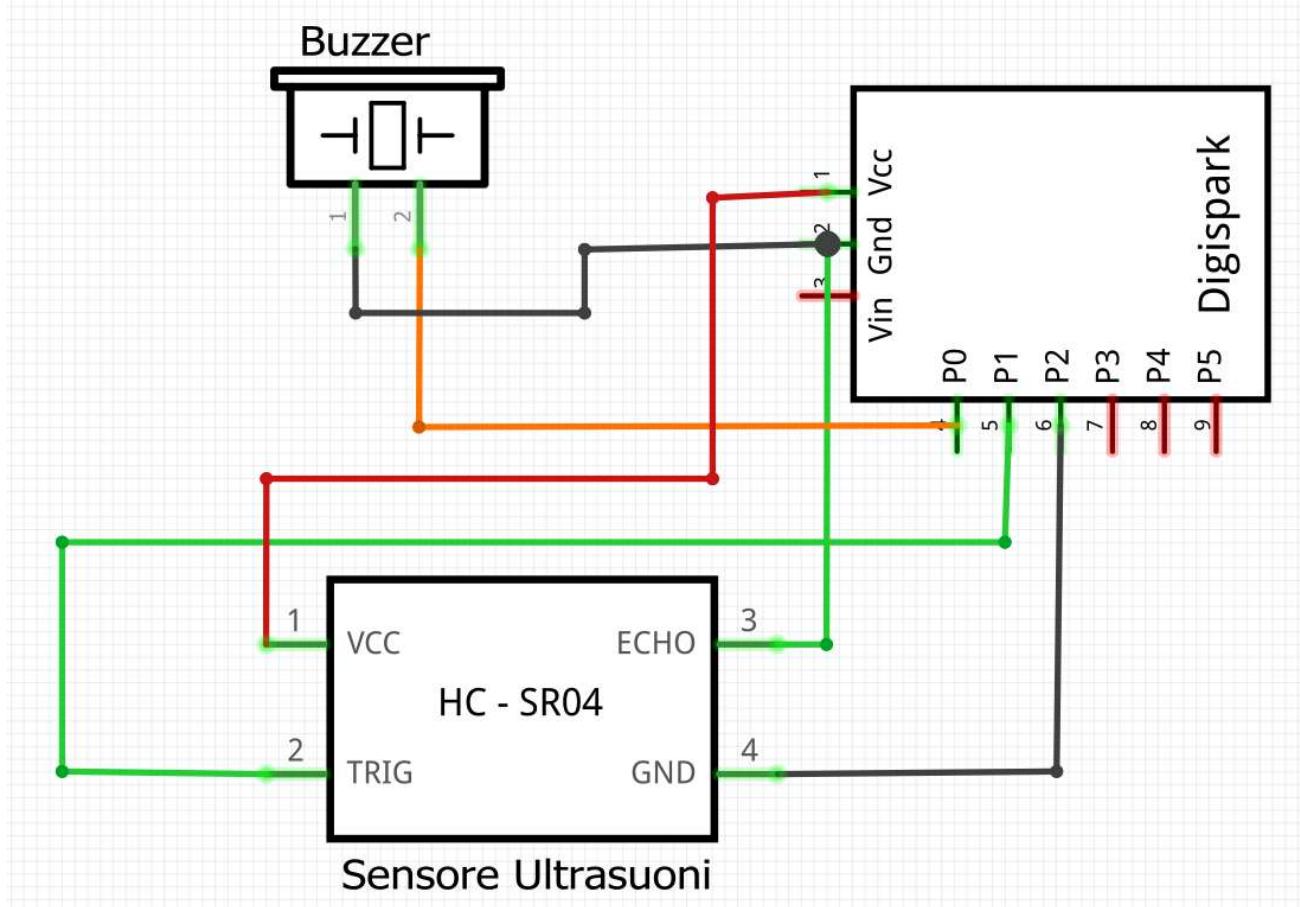
al suo interno troviamo una membrana che viene mossa da degli impulsi elettromagnetici e questi movimenti generano il suono. Per modificare il suono si procede col impostare una frequenza diversa nel segnale che viene mandato al circuito.

Non può essere perciò modificato il volume sonoro, ma come già detto, solo la l'intervallo fra un impulso e il successivo (ossia la frequenza).

Alcuni cicalini vengono illustrati con un polo rosso e uno nero, ma in realtà (quando si tratta di cicalini come il nostro, ossia con 2 pin) non c'è una polarità.



# Schema Elettrico



# Librerie

Tutte le librerie realizzate per questo progetto sono state realizzate nel linguaggio di C++, come d'altronde anche il software di Arduino.

Per includere una libreria (o una cartella di librerie) dobbiamo spostarci nella cartella ".\Arduino\libraries" (se non la trovate, premete tasto destro sull'icona dell'editor di Arduino, quindi "Apri percorso File"), all'interno di questa cartella creiamo a sua volta una cartella chiamata come la libreria o come il componente al quale fa riferimento, all'interno di questa cartella, copiamo sia l'header che la libreria stessa.

Quando apriamo l'editor di Arduino, selezionare "Sketch", quindi "#include libreria", e ora scegliere la libreria desiderata (si chiamerà come la cartella che avete creato in precedenza).

Tutte le nostre librerie sono composte da un'interfaccia (chiamata nel linguaggio specifico di "C" Header, ed è un file con estensione ".h") e la libreria in sé (con estensione ".cpp") che estende l'interfaccia.

Sia l'Header, che la libreria devono includere l'interfaccia "Arduino.h".

## Libreria Buzzer (Cicalino)

L'Header contiene:

- ◆ 1 attributo: corrisponde al pin al quale è collegato il cicalino;
- ◆ 3 metodi:
  - `setPinBuzzer(int buzzerPort);`
  - `setTone(int frequency);`
  - `powerOff();`

La libreria contiene tutti i metodi dell'Header dichiarandoli in questo modo:

```
void <NomeHeader>::<metodo>(<tipoParametro> <parametro>){};
```

Ecco cosa fa nello specifico ogni metodo:

- ◆ **setPinBuzzer**: in base al parametro che riceve, attribuisce tale valore all'attributo che indica il pin del cicalino;
- ◆ **setTone**: fa emettere al cicalino dei suoni a seconda della frequenza ricevuta come parametro;
- ◆ **powerOff**: spegne il cicalino.

## Libreria UltraSound (Sensore a Ultrasuoni)

L'Header contiene:

- ◆ 2 attributi: corrispondenti ai pin di emissione e ricezione del sensore;
- ◆ 3 metodi:
  - `setUltraSoundPin(int pinTrigPort, int pinEchoPort);`
  - `getDistance();`

La libreria contiene tutti i metodi dell'Header dichiarandoli in questo modo:

```
void <NomeHeader>::<metodo>(<tipoParametro> <parametro>){};
```

Ecco cosa fa nello specifico ogni metodo:

- ◆ **setUltraSoundPin**: in base ai parametri che riceve, attribuisce tali valori agli attributi che indicano i pin di emissione e ricezione del sensore;
- ◆ **getDistance**: ritorna la distanza misurata dal sensore in cm;

# Utilizzo

## Hardware

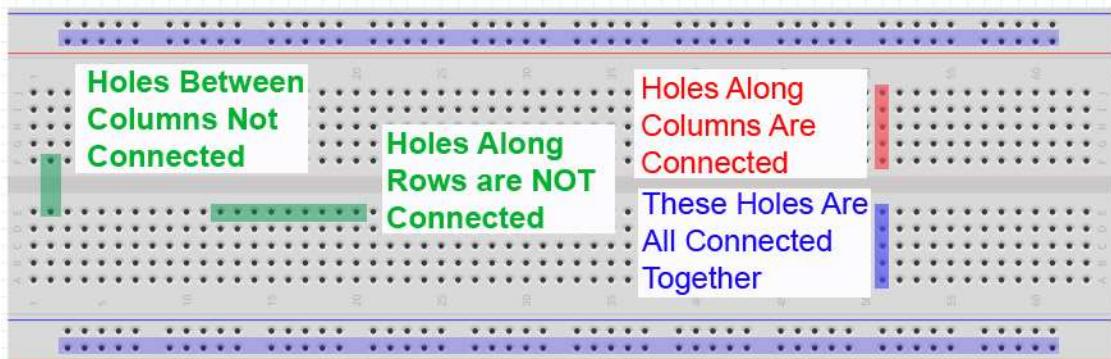
I componenti da utilizzare per questo progetto sono i 3 citati più e più volte all'interno di questa guida:

- Arduino Digispark;
- Sensore a Ultrasuoni;
- Cicalino (o Buzzer);

Per costruire il circuito dobbiamo fissare il sensore e il cicalino su di una breadboard (circuito provvisorio) o su una veroboard (circuito definitivo), non è necessario metterli in uno schema preciso a patto che abbia un senso.

**⚠ Fare attenzione alle piste delle board per evitare cortocircuiti ⚠**

*(In caso di dubbio, eccoti un'immagine che mostra com'è fatta una breadboard di Arduino al suo interno)*



Per costruire il circuito dobbiamo fissare il sensore a ultrasuoni in modo che tutti i 4 pin non siano in contatto fra loro.

Il buzzer deve essere collegato con un pin al GND dell'arduino, mentre con l'altro ad una porta del Digispark (noi usiamo P0). Il buzzer non ha polarità quindi è indifferente quale pin viene collegato a cosa.

Il Sensore a ultrasuoni ha i 2 pin esterni che corrispondono a VCC (polo positivo) e GND (polo negativo) che vanno collegati rispettivamente a VCC e ad una porta del digispark (per noi P2). I 2 pin più interni che sono "trig" e "echo" si occupano del segnale; echo va collegato al GND dell'arduino, mentre trig ad un pin di lettura (per noi P1).



## Software

- Il primo esempio di codice che abbiamo realizzato con questi componenti si occupa di suonare sempre più velocemente il cicalino man mano che la distanza registrata diminuisce.

Le prime linee di codice servono ad includere le librerie necessarie al programma (solo quelle che abbiamo creato noi perché quella base “Arduino.h” viene inclusa automaticamente):

```
#include <LibraryUltraSound.h>
#include <LibraryBuzzer.h>
```

Le linee successive le utilizziamo per dichiarare le istanze delle nostre librerie:

```
LibraryUltraSound libraryUltraSound;
LibraryBuzzer libraryBuzzer;
```

Nel metodo setup richiamiamo i metodi che abbiamo creato nelle librerie che si occupano di attribuire ai pin di emissione e ricezione del sensore a ultrasuoni e al pin del Buzzer un pin di Digispark:

```
void setup() {
    libraryBuzzer.setPinBuzzer(0);
    libraryUltraSound.setUltraSoundPin(1, 2);
}
```

Adesso entriamo nella parte più sostanziosa del programma: il metodo loop.

La prima cosa che facciamo nel loop è memorizzare la distanza registrata dal sensore:

```
int distance = libraryUltraSound.getDistance();
```

A questo punto, dopo aver controllato che la distanza registrata sia minore di 1 metro, mappiamo tale distanza in modo da ottenere l’intervallo fra un suono e l’altro del cicalino. Al termine di tutto impostiamo un delay di 50 millisecondi per evitare interferenze:

```
if(distance <= 100){
    int risultato = map(distance, 3, 100, 20, 500);
    libraryBuzzer.setTone(100);
    delay(risultato);
    libraryBuzzer.powerOff();
}
delay(50);
```

- Il nostro secondo esempio di codice invece utilizza la frequenza, ossia la distanza letta dal sensore e la frequenza del cicalino sono direttamente proporzionali.

Le prime linee di codice servono ad includere le librerie necessarie al programma (solo quelle che abbiamo creato noi perché quella base “Arduino.h” viene inclusa automaticamente):

```
#include <LibraryUltraSound.h>
#include <LibraryBuzzer.h>
```

Le linee successive le utilizziamo per dichiarare le istanze delle nostre librerie:

```
LibraryUltraSound libraryUltraSound;  
LibraryBuzzer libraryBuzzer;
```

Nel metodo setup richiamiamo i metodi che abbiamo creato nelle librerie che si occupano di attribuire ai pin di emissione e ricezione del sensore a ultrasuoni e al pin del Buzzer un pin di Digispark:

```
void setup() {  
    libraryBuzzer.setPinBuzzer(0);  
    libraryUltraSound.setUltraSoundPin(1, 2);  
}
```

Adesso entriamo nella parte più sostanziosa del programma: il metodo loop.

La prima cosa che facciamo nel loop è memorizzare la distanza registrata dal sensore:

```
int distance = libraryUltraSound.getDistance();
```

A questo punto mappiamo tale distanza in modo da ottenere la frequenza dei suoni con un delay di 100 millisecondi fra un suono e l'altro:

```
int distance = libraryUltraSound.getDistance();  
int frequenza = map(distance, 0, 100, 0, 1000);  
libraryBuzzer.setTone(frequenza);  
delay(100);
```

- Il terzo esempio che abbiamo pensato fa in modo che il cicalino cominci a suonare a partire da una certa distanza.

Le prime linee di codice servono ad includere le librerie necessarie al programma (solo quelle che abbiamo creato noi perché quella base "Arduino.h" viene inclusa automaticamente):

```
#include <LibraryUltraSound.h>  
#include <LibraryBuzzer.h>
```

Le linee successive le utilizziamo per dichiarare le istanze delle nostre librerie:

```
LibraryUltraSound libraryUltraSound;  
LibraryBuzzer libraryBuzzer;
```

Nel metodo setup richiamiamo i metodi che abbiamo creato nelle librerie che si occupano di attribuire ai pin di emissione e ricezione del sensore a ultrasuoni e al pin del Buzzer un pin di Digispark:

```
void setup() {  
    libraryBuzzer.setPinBuzzer(0);  
    libraryUltraSound.setUltraSoundPin(1, 2);  
}
```

Adesso entriamo nella parte più sostanziosa del programma: il metodo loop.

La prima cosa che facciamo nel loop è memorizzare la distanza registrata dal sensore:

```
int distance = libraryUltraSound.getDistance();
```

Se la distanza registrata è inferiore (o pari) a 20, il buzzer comincia a suonare pressoché ininterrottamente (intervallo di 10 millisecondi). In caso contrario, si spegne:

```
if(distance <= 20){  
    libraryBuzzer.setTone(100);  
    delay(10);  
}else{  
    libraryBuzzer.powerOff();  
}
```