

$$V_{\text{SHUNT}} = IR_s \Rightarrow U_p - U_1 = IR_s ; I = \frac{5V - U_1}{R_s}$$

Essendo i potenziometri tutti in serie, la corrente I è uguale per tutti \Rightarrow

$$V_N = U_1 - U_2 = IR_N \Rightarrow R_N = \frac{U_1 - U_2}{5V - U_1} R_s$$

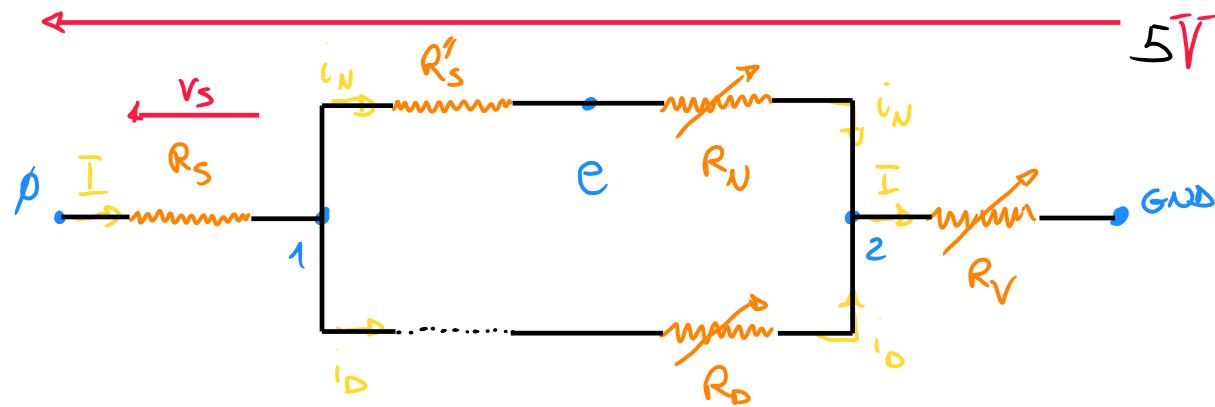
$$V_D = U_2 - U_3 = IR_D \Rightarrow R_D = \frac{U_2 - U_3}{5V - U_1} R_s$$

$$V_V = U_3 - U_{\text{GND}} = IR_V \Rightarrow R_V = \frac{U_3 - U_{\text{GND}}}{5V - U_1} R_s = 0$$

Così sarei in grado di misurare le resistenze in modo indipendente, però 3 potenziometri in serie potrebbero richiedere troppo spazio.

SOLUZIONE DAI ESPORRERÉ

Altra possibile soluzione è collegare due dei potenziometri in parallelo e uno in serie al parallelo dei due precedenti (ad esempio $(R_N \parallel R_D) + R_V$)



$$LKC \quad \textcircled{2} \quad I = i_N + i_D \quad \left(\begin{array}{l} \text{mi permette di risparmiare una} \\ \text{resistenza di SHUNT} \end{array} \right)$$

$$U_{GND} = \emptyset \quad V = \Rightarrow U_o = 5V ; \quad V_2 = i_D R_D ; \quad R_D = \frac{V_2}{I - i_N}$$

$$V_S = U_o - U_1 = IR_S \Rightarrow I = \frac{5V - U_1}{R_S}$$

Portatore di corrente:

$$R_{12} = (R''_S + R_N) \parallel R_D = \frac{R''_S R_D + R_N R_D}{R''_S + R_N + R_D}$$

$$* i_N = I \frac{R_D}{R''_S + R_N + R_D} ; \quad i_N = \frac{U_1 - U_e}{R''_S} ;$$

$$i_N \frac{R''_S R_N}{R''_S + R_N} = i_D R_D$$

* Espongo e metto i valori noti dello stesso lato dell'uguale

$$i_N R''_S + i_N R_N + i_N R_D = I R_N = \Rightarrow$$

$$\Rightarrow (i_N - I) R_N = -i_N R''_S - i_N R_D = \Rightarrow R_N = \frac{i_N (R''_S + R_D)}{I - i_N}$$

$$\Rightarrow R_N = \frac{i_N (R''_S - R_D)}{I - i_N} = \frac{i_N}{I - i_N} \left(R''_S - \frac{U_1 - U_2}{I - i_N} \right)$$

Così calcolo R_N con R_D già nota, e so inoltre che la resistenza fra 1 e 2 è sempre minore della più piccola $\Rightarrow R_{12} \leq 10 \text{ k}\Omega$

(Nel caso in cui lo scarto di tensione fosse troppo posso mettere in parallelo un resistore da $5.1 \text{ k}\Omega$)

Adesso calcolo il valore di resistenza di R_V :

$$V_{2,\text{GND}} = U_2 - U_{\text{GND}} = U_2 = IR_V \Rightarrow R_V = \frac{U_2}{5V - U_1} R_S$$

C'è comunque il rischio che R_V influenzi troppo / venga troppo influenzata dai valori di R_N ed R_D \Rightarrow

- Possibile SOLUZIONE: al posto di fare $R_N // R_D$ faccio $R_N // R_V$ (scambiando semplicemente di posto i potenziometri) in quanto R_N ed R_V sarebbero i due a controllare più valori.

Sostituisco ad R_D uno o più BOTTONI che controllino i valori del denominatore della mia frazione.

I valori al denominatore che mi servono sono 2 4 6 8, posso scrivere uno script che "mette in loop" questa sequenza di numeri e faccio un passo avanti ogni volta che viene premuto un pulsante

I bottoni però hanno solamente status ON/OFF \Rightarrow

\Rightarrow Siccome mi serve verificare fra i numeri posso prendere 2 bottoni e leggere le 2^2 possibilità di status totale, assegnando loro un numero tra $\{1, 2, 3, 4\}$:

BOTONE 1	BOTONE 2	NUMERO ASSEGNATO
0 OFF	0 OFF	2
0 OFF	1 ON	4
1 ON	0 OFF	6
1 ON	1 ON	8

In questo modo dovrebbe essere abbastanza semplice verificare fra i valori del Denominatore (possibile ~~ELEMENTO DI NOVITÀ~~)

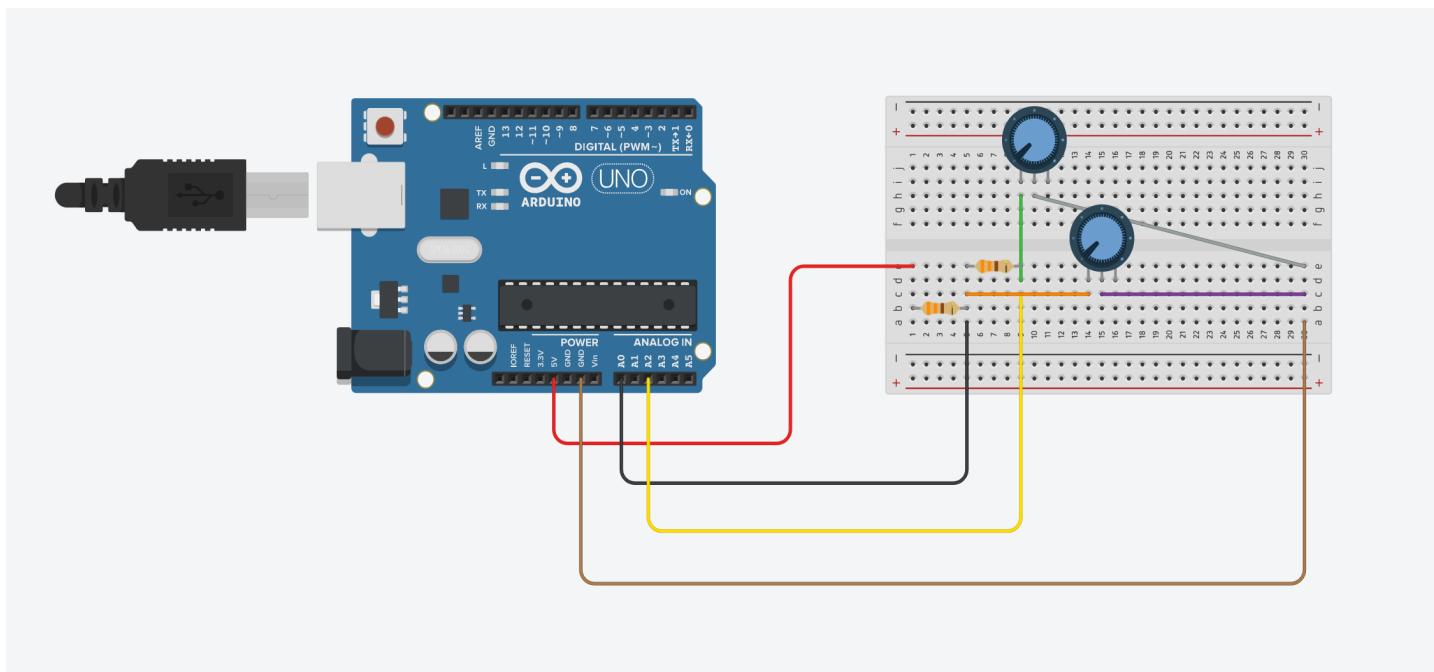
11/09

PROBLEMA: il dover misurare due diverse correnti non solo è molto ingombro ma rischia anche di non essere per niente preciso

INOLTRE, facendo dei test con una configurazione e un codice di questo tipo (e prossima pagina), per come è calcolata i_N , se $R_V = 0 \Omega$ allora anche $i_N = 0 A$, e diventa quindi impossibile calcolare R_N .

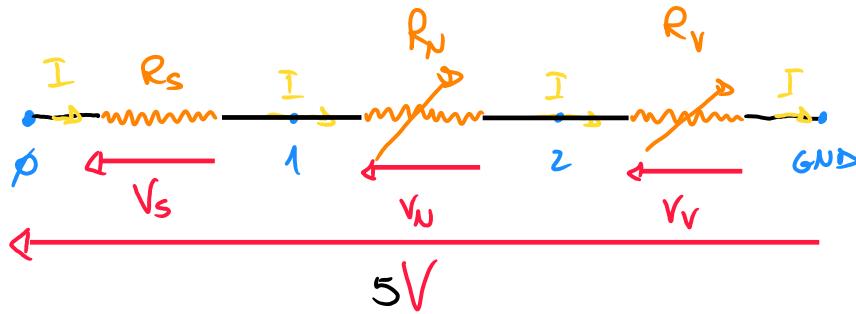
Per ovviare ad entrambi i problemi si potrebbe ritornare ai due potenziometri in serie e fare dei test.

```
1 void setup() {
2     // put your setup code here, to run once:
3     Serial.begin(9600);
4 }
5
6 void loop() {
7     // put your main code here, to run repeatedly:
8
9     // Ottengo le misure di tutto quello che mi serve tutte insieme, dopodiché le vado a stampare in modo da capirci qualcosa :
10    float u1 = analogRead(A0) * (5.0 / 1024.0);
11    float I = (5.0 - u1) / 330.0;
12
13    float ue = analogRead(A2) * (5.0 / 1024.0);
14    float i_n = (u1 - ue) / 330.0;
15
16    float velRes = u1 / (I - i_n);
17    float numRes = (i_n / (I - i_n)) * (330 - (u1 / (I - i_n)));
18
19    delay(1000);
20    Serial.print("Tensione di Shunt collegata ad I: ");
21    Serial.print(5.0 - u1);
22    Serial.println(" V");
23
24    Serial.print("Tensione di Shunt collegata ad i_n: ");
25    Serial.print(u1 - ue);
26    Serial.println(" V");
27
28    Serial.print("Resistenza di Numeratore: ");
29    Serial.print(numRes);
30    Serial.println(" Ohm");
31
32    Serial.print("Resistenza di Velocità: ");
33    Serial.print(velRes);
34    Serial.println(" Ohm");
35 }
```



{ Codice e topologie del circuito usati per effettuare i primi test che hanno evidenziato il problema }

È seguito i prossimi test utilizzando un circuito del genere



Calcolo I tramite la resistenza di Shunt R_s

$$I = \frac{V_s}{R_s} = \frac{5V - U_1}{R_s}$$

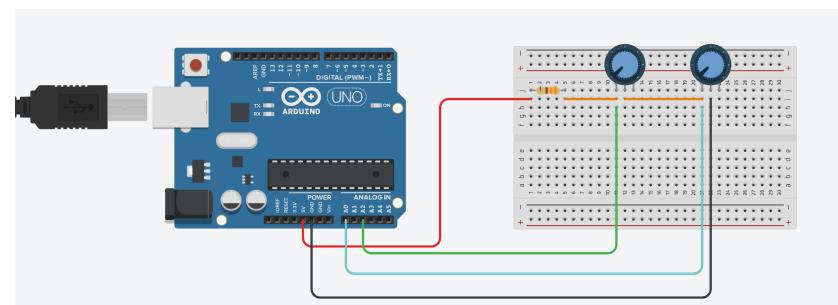
Una volta calcolato I posso trovare facilmente R_N e R_V tramite le loro equazioni costitutive

$$R_N = \frac{U_1 - U_2}{I}$$

$$R_V = \frac{U_2 - U_{GND}}{I}$$

```

1 void setup() {
2   // put your setup code here, to run once:
3   Serial.begin(9600);
4 }
5
6 void loop() {
7   // put your main code here, to run repeatedly:
8   // Ottengo le misure di tutto quello che mi serve tutte insieme, dopodiché le vado a stampare in modo da capirci qualcosa :
9   float u1 = analogRead(A2) * (5.0 / 1024.0);
10  float I = (5.0 - u1) / 330.0;
11
12  float u2 = analogRead(A0) * (5.0 / 1024.0);
13
14  float numer = (u1 - u2) * I;
15  float veloc = (u2 / I);
16
17
18  delay(1000);
19
20  Serial.print("Tensione di Shunt: ");
21  Serial.print(u1 - u2);
22  Serial.println(" V");
23
24  Serial.print("Intensità di Corrente " I );
25  Serial.print(" * 1000");
26  Serial.println(" mA");
27
28
29  Serial.print("Tensione di Numeratore: ");
30  Serial.print(u1);
31  Serial.println(" V");
32
33  Serial.print("Tensione di Velocità: ");
34  Serial.print(u2);
35  Serial.println(" V");
36
37  Serial.print("Resistenza di Numeratore: ");
38  Serial.print(u1Res);
39  Serial.println(" Ohm");
40
41  Serial.print("Resistenza di Velocità: ");
42  Serial.print(u2Res);
43  Serial.println(" Ohm");
44
45
46 }
```



È seguendo dei test vedo che riesco a misurare le due resistenze indipendentemente, e che nonostante i valori di tensione variabili le misurazioni continuano ad essere effettuate correttamente al variare alternato dei valori dei potenziometri.

Dra che l'acquisizione dei valori di resistenza funziona in modo abbastanza preciso posso alla progettazione della seconda parte del circuito, cioè i due **BOTTONI** e il **BUZZER**, gli elementi che mi permetteranno di decidere il **denominatore** e poi di emettere l'effettivo **suono del metronomo**.

Dopo qualche test non riesco a far funzionare i bottoni come mi serve: posso semplificare un po' il circuito e lasciare il denominatore sempre su 4, tanto in un metronomo cambia poco.

Mi concentro solo sul buzzzer e sulla scrittura dell'algoritmo (con possibilità future di implementare degli switch e controllo del denominatore).

17/04

Sfruttando l'esempio proposto da Arduino IDE "tone Melody" mi rendo conto che l'utilizzo di un singolo **buzzer passivo** può essere la soluzione migliore per emettere il suono dei battiti del metronomo, devo solo scrivere il giusto script.

Attraverso i codici trovati su questo sito web

<https://www.filiconnesso.it/comunicazione-arduino-python/>

sono riuscito a fare in modo che le melodie contenute di `default` in "toneMelody" parta in loop solamente quando, attraverso il mio script di Python, do una stringa in input contenente "ACCESO", e si ferma solamente quando do una seconda stringa in input contenente qualunque cosa di diverso dalle parole "ACCESO".

Nelle pagine seguenti i due script (la Arduino ha le connessioni già utilizzate nell'esempio "tone Melody", quindi non le riporto in quanto già ben spiegato sul sito di Arduino)

```

serial_write.py > ...
1 import serial
2 import time
3
4 Arduino = serial.Serial('COM3', 9600)
5 time.sleep(1)
6
7 while True:
8     comando = input("Acceso o spento? ").encode()
9     Arduino.write(comando)
10
11

```

Ferendo prima partire lo script delle IDE Arduino e dopo quello di Python i due riusciremo a comunicare perfettamente

```

1 #include "pitches.h"
2
3 int melody[] = {
4     NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4
5 };
6
7 int noteDurations[] = {
8     4, 8, 8, 4, 4, 4, 4, 4
9 };
10
11 String msg;
12 char ch;
13
14 void setup() {
15     Serial.begin(9600);
16     msg = "";
17 }
18
19 void loop() {
20     while(Serial.available()){
21         delay(50);
22         ch = Serial.read();
23         msg += ch;
24     }
25
26     msg.toLowerCase();
27     if (msg.length() > 0){
28         if (msg == "acceso"){
29             for (int thisNote = 0; thisNote < 8; thisNote++) {
30                 int noteDuration = 1000 / noteDurations[thisNote];
31                 tone(8, melody[thisNote], noteDuration);
32
33                 int pauseBetweenNotes = noteDuration * 1.30;
34                 delay(pauseBetweenNotes);
35
36                 noTone(8);
37             }
38         }
39         delay(1000);
40     }
41 }

```

Adesso so quindi che passare dati da Python ad Arduino è fattibile, quindi dovrei essere in grado (conoscendo già il modo per usare Python per prendere delle informazioni da Arduino) di dividere il funzionamento del metronomo in 3 distinte parti:

I. ARDUINO → PYTHON

Calcolo i valori di resistenza dei potenziometri e li faccio leggere a Python

II. PYTHON

Utilizzo Python per fare tutte le analisi dei dati che mi servono per trasformare dei valori di resistenza in valori discreti di "NUMERATORE" e "VELOCITÀ".

Qualunque altro tipo di analisi aggiuntiva può essere inserita in Python per evitare di sprecare memoria in Arduino

III. PYTHON → ARDUINO

Dal Python prendo soltanto i valori strettamente necessari che servono ad Arduino per capire quante volte far suonare il **buzzer** e con quale frequenza ad ogni battuta (idealmente 2 valori, "Num" e "Vec", e ad ogni singolo loop corrisponde una singola battuta).

Sviluppando quindi gli algoritmi necessari per questi 3 diversi momenti singolarmente e poi unendoli in un unico algoritmo Python ed un unico algoritmo per Arduino potrai essere in grado di realizzare un **METRONOMO** perfettamente funzionante.

Potrebbe inoltre essere utile, in previsione futura, creare un qualche lettore da utilizzare come lettore d'uscita da un loop: visto che, durante l'uso, i due lettori facilmente modificabili sono i valori di resistenza dei **potenziometri**, posso pensare ad un lettore di uscita del tipo (`bool(numRes + velRes < 8)`), in modo che abbassare al minimo entrambe le manovelle dei potenziometri possa rappresentare una condizione di uscita, non si sa mai potrebbe servire.

Valori al numeratore che mi servono (e rispettive fasce): *(x 1/10)

$$2 \quad 0 \leq \text{numRes}^* < 100$$

$$3 \quad 100 \leq \text{numRes}^* < 200$$

$$4 \quad 200 \leq \text{numRes}^* < 300$$

$$5 \quad 300 \leq \text{numRes}^* < 400$$

$$6 \quad 400 \leq \text{numRes}^* < 500$$

$$7 \quad 500 \leq \text{numRes}^* < 600$$

$$8 \quad 600 \leq \text{numRes}^* < 700$$

$$9 \quad 700 \leq \text{numRes}^* < 800$$

$$10 \quad 800 \leq \text{numRes}^* < 900$$

$$11 \quad 900 \leq \text{numRes}^* \leq 1000$$

Valori di tempo (velocità) e rispettive fasce: $*(\times \frac{1}{10})$

(Prendo solo i valori più comuni per evitare di dover creare troppe fasce ad intervalli troppo piccoli, visto anche che la misura di resistenza non è precisissima)

d Semiminima	FASCE	d Semiminima	FASCE
64 (Grave)	$0 \leq \text{VelRes}^* < 100$	108 (Moderato)	$500 \leq \text{VelRes}^* < 600$
60 (Adagio)	$100 \leq \text{VelRes}^* < 200$	121 (Allegro)	$600 \leq \text{VelRes}^* < 700$
72 (Maestoso)	$200 \leq \text{VelRes}^* < 300$	132 (Allegro)	$400 \leq \text{VelRes}^* < 500$
81 (Andante)	$300 \leq \text{VelRes}^* < 400$	144 (Vivace)	$800 \leq \text{VelRes}^* < 900$
92 (Animato)	$400 \leq \text{VelRes}^* < 500$	162 (Presto)	$900 \leq \text{VelRes}^* < 1000$

$\text{d} = x \text{ bpm} : x \text{ battiti al minuto} \Rightarrow \frac{x}{60} \text{ battiti per secondo}$

{ Es. $\text{d} = 120 \text{ bpm} \rightarrow \frac{120}{60} \text{ bps} \rightarrow 2 \text{ bps}$
 $\rightarrow 0,5 \text{ secondi per battito}$ }

$y \text{ secondi per battito} = \frac{60}{x} \text{ battiti per secondo}$

18/09

Dall'esempio "toneMelody" capisco che, all'interno del valore y devono stare l'intera durata della nota più il tempo di "silenzio" tra una nota e l'altra per renderle distinguibili.

Posso quindi impostare una equazione

$$Y = \frac{60}{X}, \quad Y = \alpha + \frac{13}{10} \alpha,$$

↳ secondi per battito ↳ tempo di silenzio in s
↳ Tempo di durata delle note in s

$$\frac{23}{10} \alpha = \frac{60}{X}, \quad \alpha' = \frac{600}{23X} \times 1000 \text{ [ms]}$$

Essendo "x" un valore che dipende da "velRes", attraverso queste formule posso già in Python determinare i millisecondi di funzionamento del **buzzer**, cioè α'

21/04

Negli scorsi giorni, nel poco tempo avuto a disposizione da impiegare nel progetto del metronomo, mi sono impegnato nella scrittura dei due algoritmi, uno in C++ per Arduino e uno in Python, trovando più di qualche difficoltà:

- Ho studiato un po' di Python, ma non ho mai fatto C, C# o C++, quindi usare le sintassi del C++ ha reso lo sviluppo un po' più lento. Inoltre non conoscevo (ne tanto meno li conosco ora) le funzioni che avevo a disposizione in C++, quindi c'è anche un po' di lavoro di ricerca dietro che non ho riportato.

- Essendo l'algoritmo Arduino e quello Python strettamente connessi fra di loro non potevo inviare uno senza anche inviare l'altro, e questo è stato un grosso problema perché l'unico modo che avevo per controllare l'output delle funzioni in Arduino (comprese e soprattutto le funzioni `Serial.available()` e `Serial.read()`, che leggono dati da Python) era usare le funzioni `Serial.print()` e `Serial.println()`, che però richiedono l'utilizzo del `Serial Monitor`, il quale non può essere attivato (funzionante) se alla stessa porta seriale sta già eseguendo Python.

- Dove passare due valori differenti in una volta solo da Python all'Arduino è stato una sfida osérei dire di "creatività", perché serviva un modo ingegnoso ma non troppo lungo e complicato che mi permettesse di estrarre due valori da uno solo senza alcuna possibilità di errore.

Lo chiedevo di volta è stato il `.toFloat()`:

1. mi ha permesso di lavorare con delle stringhe facilmente trasmutabili in numeri floating point;
2. ritorna in output \emptyset se la stringa non inizia con una cifra: mi ha permesso di creare una semplice "condizione uscita" che dipende solamente dal valore delle resistenze dei **potenziometri**, e quindi facilmente modificabili.

- Un altro grande problema è stato trovare un modo facile per accendere il circuito e comandarlo, una sorta di pulsante di accensione: utilizzare Python era fuori discussione in quanto lo scambio di informazioni fra la scheda Arduino e Python era già abbastanza complesso.

Fecendo molti test mi sono accorto che Python inizia a leggere i dati "emessi" da Arduino dopo che lo script è stato avviato (ovviamente), ma in particolare modo che Python non legge i dati "emessi" da Arduino se questi sono stati emessi PRIMA dell'invio dello script:

questo mi ha fatto pensare che posso far fare all'Arduino UNA SINGOLA MISURA DI RESISTENZA, farla leggere da Python e solo una volta ricevuto un risponso da Python far partire l'effettivo algoritmo del metronomo.

Queste due considerazioni, insieme, mi hanno fatto ragionare su due questioni:

1. Per mettere in moto l'intero algoritmo, in questo modo, dovrei inviare lo script Python PRIMA di "avviare" l'Arduino (cioè prima di collegarla al pc e alla porta seriale)
2. Non posso inviare lo script Python prima di aver connesso l'Arduino alla porta seriale

Effettivamente, però, e me non serve inviare la scheda prima di inviare lo script Python, mi serve solo che la PRIMA MISURAZIONE avvenga dopo l'invio dello script Python: posso utilizzare il TASTO DI RESET.

A fine di giorno 21/04 sono uscito e far funzionare l'algoritmo come volevo io, facendo come uniche modifiche quelle di eliminare "12" come possibile valore di Numeratore (per qualche motivo o me ignoto manda in tilt il **buzzer**) e "162" come possibile valore di bpm (in quanto tra l'ultimo beat in levare e il beat in battere c'erano istanti di pausa extra che mandavano tutto fuori tempo, forse dovuti ai tempi impiegati da Python per fare le operazioni in virgola mobile, o se non per questo non so bene per quale motivo)

Essendo Python fondamentale in questo algoritmo la scheda Arduino deve essere collegata ad un PC, tuttavia sarebbe possibile rendere il tutto indipendente integrando lo script Python in quello C++ e usando un componente come uno **switch (ON/OFF)** come "condizione d'entrata"/"condizione d'usata".

Lo script lo lascio allegato insieme a questo file, lascio invece qui di seguito la topologia:

