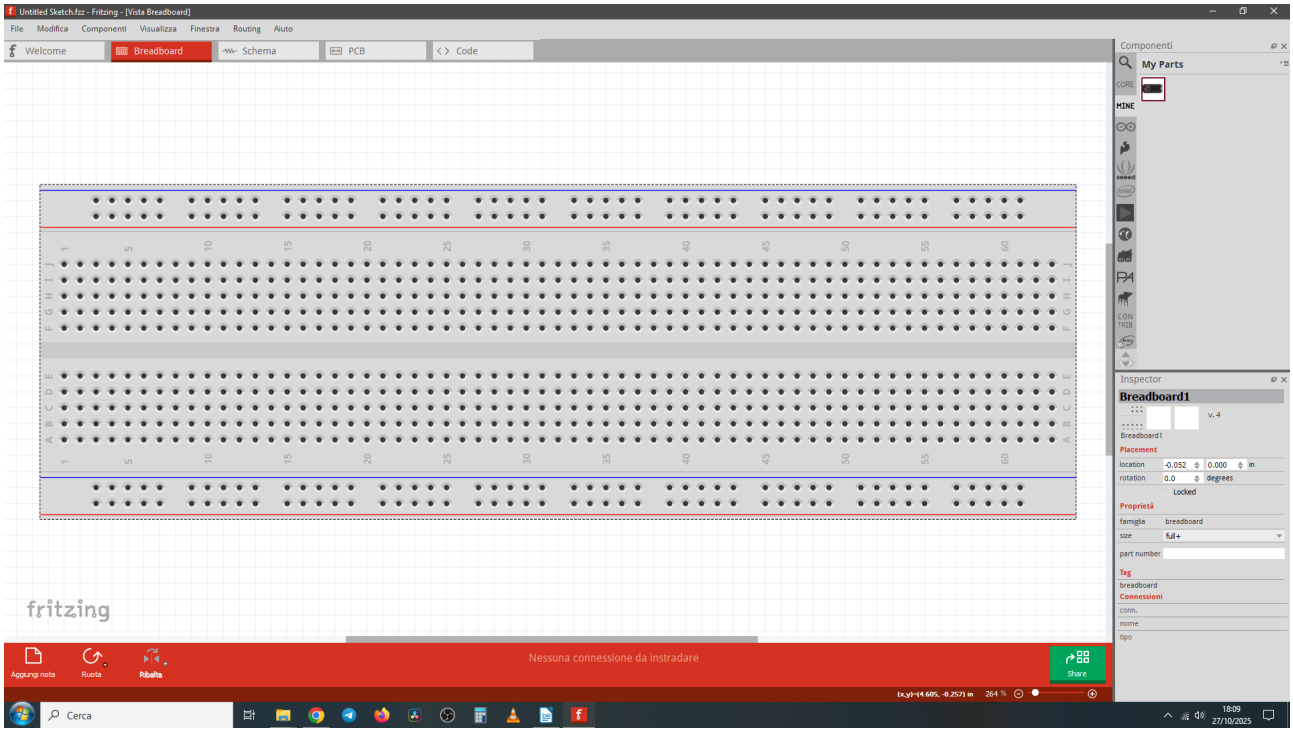
Corso “ESP32 – da Zero a Robot”

Modulo 1 – Dispensa alla Lezione 2

La breadboard

Tutte le volte che nei video verrà visualizzato il microcontrollore ESP32, questo sarà su di una **breadboard**. Una breadboard è una scheda per la prototipazione rapida e sicura, che ci permette di realizzare dei semplici circuiti senza necessità di saldare.



Nella raccolta delle componenti hardware utilizzati in questo corso (che potete trovare seguendo questo link) ne potete trovare una dotata di 830 punti ed un piccolo alimentatore con due uscite regolabili a 3,3V od a 5V.

Quella mostrata in figura, invece, è la breadboard virtuale contenuta in Fritzing, un software open source per disegnare circuiti elettrici, che potete trovare sul sito di riferimento (https://fritzing.org/download/) oppure nel repository del progetto su GitHub (https://github.com/fritzing/fritzing-app/releases).

Raccomando caldamente l’uso di una breadboard per gli esperimenti, perché ci evita di saldare e dissaldare più volte componenti al microcontrollore. Ci sarà tempo di realizzare delle PCB quando i nostri progetti saranno definitivi e collaudati.

Una breadboard è costituita da una ‘scatola’ di plastica che su di un lato presenta numerosi fori di diametro fisso (0.9 mm), ad una distanza standard (un ‘passo’ di 2.54 mm, 1/10 di pollice). Al suo interno ci sono delle piste metalliche affiancate a creare dei connettori.

Lo schema di connessione delle piste è diviso in linee di alimentazione (le quattro file di fori più esterne, due su di un lato e due sull’altro, di solito affiancate da linee rosse e blu che indicano + e -) e delle linee perpendicolari a quelle di alimentazione, destinate alle componenti.

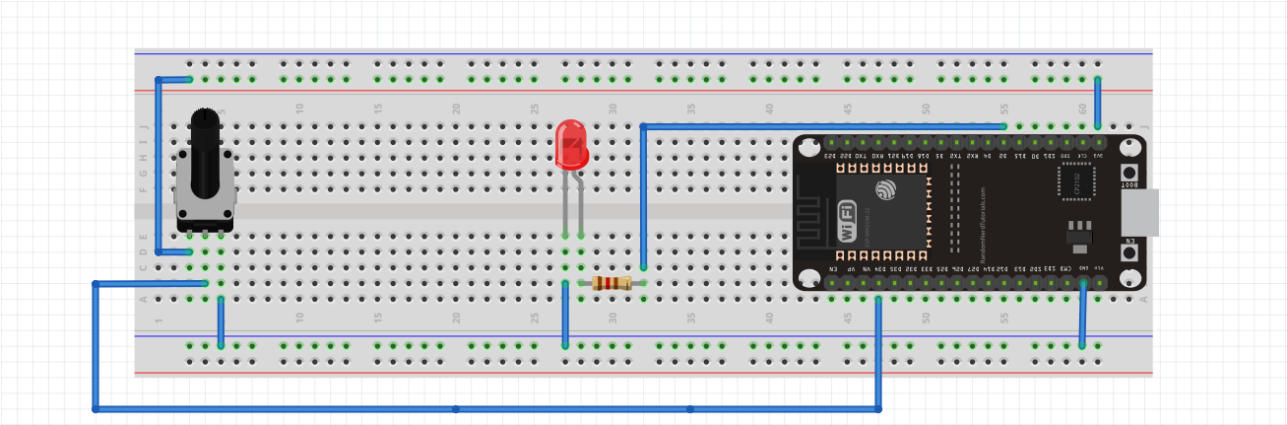
Attenzione: le due sezioni centrali (in alto e in basso) sono separate da un canale; non sono connesse. L'ESP32 deve essere posizionato in modo che il canale centrale separi i suoi pin, evitando cortocircuiti tra le file opposte.

Tutti gli schemi elettrici relativi agli esperimenti proposti sono disegnati con Fritzing, dando per scontato l’uso di una breadboard, per una maggior chiarezza.

Il primo circuito

Nella prima lezione ci si limitava ad accendere o spegnere il LED a bordo della scheda, nella seconda lezione vengono aggiunti alcuni componenti: un LED, una resistenza ed un potenziometro nei primi due esperimenti, un sensore BME280 nel terzo esperimento. Nel secondo esperimento la configurazione elettrica rimane la stessa del primo e cambia solo il codice che viene eseguito.

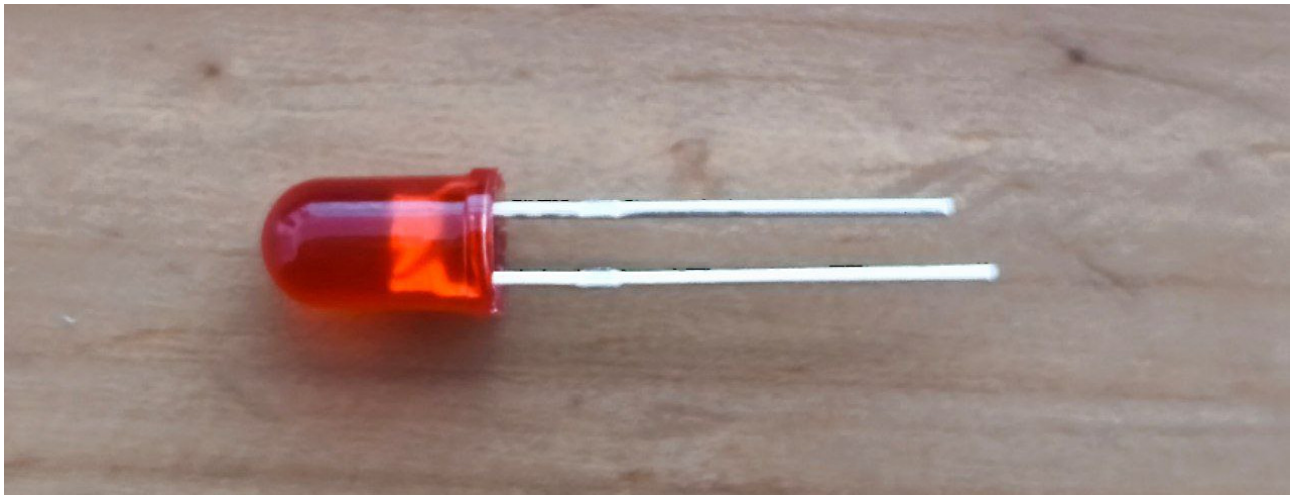
Schema di collegamento di primo e secondo esperimento:



Il collegamento dei componenti dei primi due esperimenti è il seguente:

Componente	**Pin/Valore**	**Collegamento all'ESP32**
Potenziometro *(10 kΩ)*	*Pin esterno 1*	*3.3V*
	Pin esterno 3	*GND*
	Pin centrale (Wiper)	**GPIO 34** *(Pin solo input ADC sull'ESP32)*
LED	*Anodo (+)*	*Resistenza in serie*
	Catodo (-)	*GND*
Resistenza	*100Ω* o *120Ω*	*In serie tra Anodo LED e* **GPIO 2**

Tip: l’anodo del LED



Per identificare l’anodo del LED ci sono diversi sistemi. Anzitutto, quando sono nuovi, il piedino più lungo è l’anodo. Se fossero stati tagliati (come nelle componenti di recupero), sul corpo, nella parte inferiore, dovrebbe esserci una tacca od una smussatura che indicano il catodo. Se anche queste non fossero identificabili, guardando il LED in trasparenza, l’anodo è generalmente più robusto del catodo.

Altrimenti, avrete il 50% di possibilità d’indovinare. Oppure usate un tester!

Codice del primo esperimento (frequenza di lampeggio):

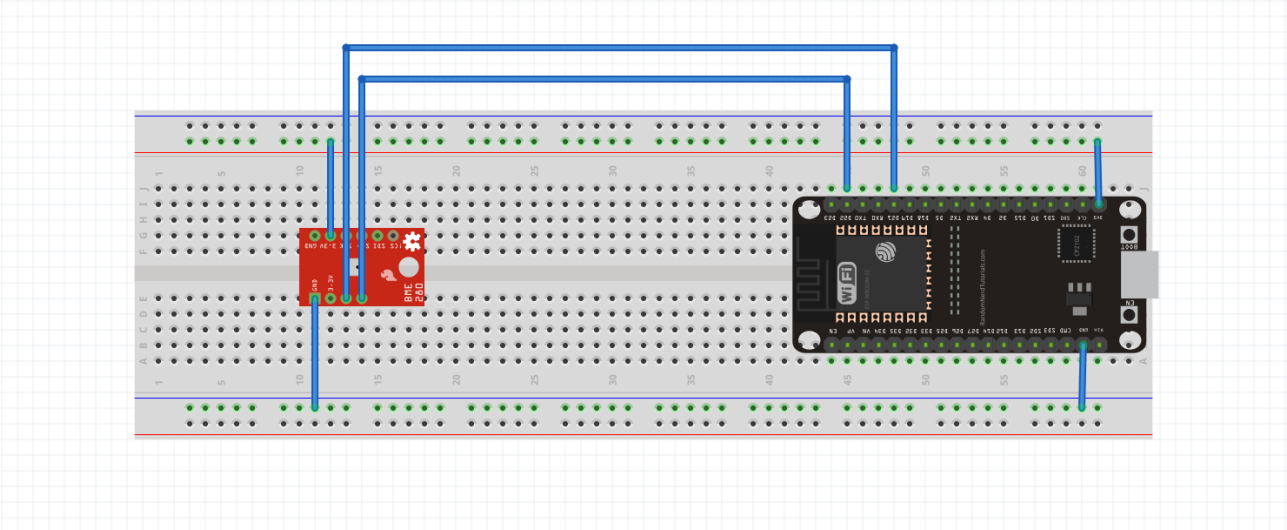
```
from machine import Pin, ADC
import time

# Useremo GPIO 34, che è solo input (ADC1_CH6) e molto stabile.
adc_pin = Pin(34)
adc = ADC(adc_pin)

# Setta la risoluzione a 12 bit (0-4095)
adc.width(ADC.WIDTH_12BIT)
# Setta l'attenuazione a 11dB (copre tutto il range 0-3.3V)
adc.atten(ADC.ATTN_11DB)

# Per variare la FREQUENZA usiamo il controllo digitale base
led_pin = Pin(2, Pin.OUT) # GPIO 2 ha un LED integrato su molte schede

def map_value(x, in_min, in_max, out_min, out_max):
    # Mappa un valore da un intervallo a un altro.
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

print("Il potenziometro controlla frequenza LED. (CTRL+C per uscire)")
while True:
    # 1. Legge il valore ADC (0 a 4095)
    adc_value = adc.read()

    # 2. Mappa il valore ADC a un intervallo tra 50ms e 1000mS
    delay_ms = map_value(adc_value, 0, 4095, 50, 1000)

    # 3. Controlla il LED con il ritardo calcolato
    led_pin.value(1) # Accende
    time.sleep_ms(int(delay_ms))

    led_pin.value(0) # Spegne
    time.sleep_ms(int(delay_ms))

    # Stampa i valori per debug nella REPL di Thonny
    print(f"ADC: {adc_value} -> Ritardo: {delay_ms:.0f} ms")
```

Codice del secondo esperimento (PWM e luminosità):

```
from machine import Pin, ADC, PWM
import time

# Useremo GPIO 34 (Pin solo input)
adc_pin = Pin(34)
adc = ADC(adc_pin)

# Risoluzione a 12 bit ed  Attenuazione a 11dB
adc.width(ADC.WIDTH_12BIT)
adc.atten(ADC.ATTN_11DB)

# Usiamo GPIO 2, con resistenza in serie.
pwm_pin = Pin(2)
led_pwm = PWM(pwm_pin)

# Frequenza del PWM (es. 5000 Hz è invisibile all'occhio umano)
led_pwm.freq(5000)

# La risoluzione del duty cycle di default è 10 bit (0-1023)
MAX_DUTY = 1023
MAX_ADC = 4095

print("Il potenziometro controlla la luminosità. (CTRL+C per uscire)")

while True:
    # Legge il valore ADC dal potenziometro (0 a 4095)
    adc_value = adc.read()

    # Mappa l’ADC al Duty Cycle PWM (0 a 1023)
    duty_cycle = int((adc_value / MAX_ADC) * MAX_DUTY)

    # Imposta il Duty Cycle PWM
    led_pwm.duty(duty_cycle)

    # Stampa i valori per debug nella REPL
    print(f"ADC Value: {adc_value} -> PWM Duty Cycle: {duty_cycle}")

    # Breve pausa per non sovraccaricare la REPL
    time.sleep_ms(50)
```

Esperimento 3: il BME280

Il terzo esperimento prevede il collegamento di un sensore BME280. Questo sensore, che rileva tempera, pressione ed umidità dell'aria, usa l'interfaccia standard I2C e prevede quindi 2 soli fili per la comunicazione col microcontrollore e 2 fili per l'alimentazione.



N.B.: quello raffigurato nello schema ha un fattore di forma differente da quello usato nel video, ma non importa: a noi interessano solo i piedini di alimentazione (3,3V e GND) ed i due di comunicazione (SDA ed SCL), che sono comuni a tutte le versioni.

Collegamenti elettrici:

Componente	**Pin/Valore**	**Collegamento all'ESP32**
Modulo BME280	*Pin VCC*	*3.3V*
	Pin GND	*GND*
	Pin SDA (Data)	**GPIO 21**
	Pin SCL (Clock)	**GPIO 22**

Libreria bme280.py

La libreria bme280.py contiene i driver che permettono la lettura dei dati trasmessi da questo sensore. È possibile installarla con pip o scaricarla dal web, nel caso la versione generica non funzionasse correttamente.

```
pip install bme280
```

La libreria si trova e si può scaricare in diverse versioni da GitHub, per esempio a questo indirizzo:

https://github.com/robert-hh/BME280

In rarissimi casi potrebbe essere necessario cercarne una versione specifica.

La libreria sarà poi da copiare sul dispositivo, aprendola con Thonny e usando “salva come”. Se l’avrete scaricata, allora sarà dove l’avete salvata sul vostro computer, se l’avete installata con pip, allora sarà nelle librerie di Python.

Nel mio caso, che sto usando un’installazione di default su Windows, si trova in:

```
C:\Python311\Lib\site-packages\bme280
```

Se non sapete dove è salvata sul vostro sistema, potete usare il comando:

```
pip show bme280
```

Questo vi mostrerà i dettagli della libreria, incluso il percorso in cui è installata.

Materiali di riferimento e dotazione hardware

La pagina di riferimento a questo corso si trova a questo indirizzo:
https://www.capirciqualcosa.it/esp32-da-zero-a-robot-presentazione-del-corso

La pagina di riferimento a questo modulo si trova a questo indirizzo:
https://www.capirciqualcosa.it/esp32-da-zero-a-robot-modulo-1

Le lezioni video si trovano anche a questo indirizzo: https://bit.ly/ESP32-DaZeroARobot

Tutto il materiale hardware usato nel corso si può trovare a questo indirizzo:
https://www.amazon.it/shop/capirciqualcosa