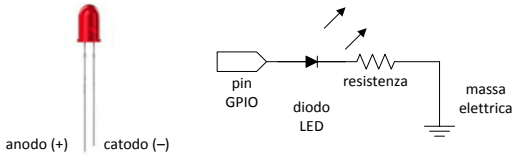
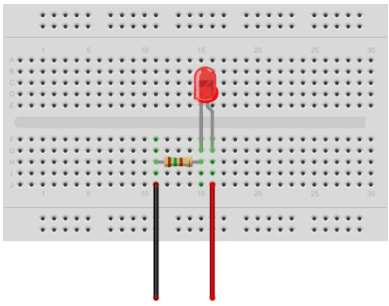
LABORATORIO DIDATTICO 10

Controllo di un LED con Raspberry Pi e Python

Il classico esempio per verificare la possibilità di utilizzare un pin del connettore GPIO come output digitale è quello di connettervi un LED (*Light Emitting Diode*) e di farlo lampeggiare accendendolo e spegnendolo periodicamente.
Un LED è un componente elettronico avente le caratteristiche di un diodo: come tale può essere attraversato dalla corrente elettrica in una sola direzione, dall'"anodo" (il terminale a potenziale elettrico positivo, più lungo) verso il "catodo" (il terminale a potenziale elettrico negativo, più corto) che deve quindi essere montato verso la massa elettrica:



La resistenza deve essere scelta in modo che la differenza di tensione ai capi del diodo LED acceso sia circa 2V e che la corrente che lo attraversa sia circa 0,01A[1] (10mA, che è anche la corrente massima che un singolo pin del connettore GPIO può generare). Dato che quando è configurato come output digitale ed impostato al valore logico "1" il pin del connettore GPIO ha una tensione rispetto alla massa elettrica di 3,3V, la differenza di tensione ai capi della resistenza dovrà essere di 3,3V – 2V = 1,3V; per avere una corrente di 0,01A la resistenza si calcola applicando la legge di Ohm: $1{,}3_{[V]} : 0{,}01_{[A]} = 130\ \Omega$ (resistenze più alte diminuiscono la corrente e riducono la luminosità emessa dal LED, resistenze più basse aumentano la corrente e possono danneggiare permanentemente il pin del connettore GPIO del dispositivo Raspberry Pi).
Anche se questo semplice circuito elettronico può essere realizzato "in aria"
è preferibile montarlo utilizzando una *breadboard*:



Il cavetto nero connesso al catodo del LED mediante la resistenza deve essere connesso ad una massa elettrica del connettore GPIO del Raspberry Pi, ad esempio il pin 6; il cavetto rosso connesso direttamente all'anodo del LED deve invece essere connesso ad un pin configurabile come input/output del connettore GPIO del Raspberry Pi, ad esempio il pin 7.
Nell'ipotesi di questo cablaggio, il seguente programma in linguaggio Python fa lampeggiare per 30 volte il LED con un periodo di 2s:

[1] queste sono le caratteristiche elettriche dei comuni LED rossi, gialli e verdi

```
import time
# importazione del modulo RPi.GPIO con il nome "GPIO"
import RPi.GPIO as GPIO

# impostazione della numerazione dei pin del connettore GPIO
GPIO.setmode(GPIO.BOARD)
# configurazione del pin 7 del connettore GPIO come output digitale
GPIO.setup(7, GPIO.OUT)

# ciclo di operazioni ripetute per 30 volte
for n in range(30):
    # impostazione del pin 7 del connettore GPIO al valore logico "1"
    GPIO.output(7, GPIO.HIGH)
    # attesa di 1s
    time.sleep(1)
    # impostazione del pin 7 del connettore GPIO al valore logico "0"
    GPIO.output(7, GPIO.LOW)
    # attesa di 1s
    time.sleep(1)

# ripristino della configurazione predefinita del connettore GPIO
GPIO.cleanup()
```

L'esecuzione di questo programma dalla Python shell non produce nessun effetto perché deve essere eseguito con i privilegi dell'utente amministratore del sistema operativo Raspbian; un modo è quello di eseguirlo utilizzando l'interfaccia testuale digitando il seguente comando:

```
# sudo python LED.py
```

nell'ipotesi che il codice sia contenuto nel file "LED.py".
Per controllare l'intensità luminosa di un LED è possibile utilizzare la tecnica PWM per modulare un segnale elettrico di alimentazione periodico con una frequenza non visibile dall'occhio umano: essa sarà alta per valori del *duty-cycle* prossimi a 100, bassa per valori prossimi a 0 fino allo spegnimento completo che si ottiene impostando il valore 0.
Il seguente codice Python, se eseguito con i privilegi dell'utente amministratore, genera sul pin 7 del connettore GPIO un segnale elettrico avente una frequenza di 100Hz (corrispondente a un periodo di 10ms) e un *duty-cycle* iniziale del 50%; l'utente può inserire nuovi valori per il *duty-cycle* che modificano dinamicamente la generazione del segnale (valori negativi o maggiori di 100 causano l'interruzione dell'esecuzione del programma):

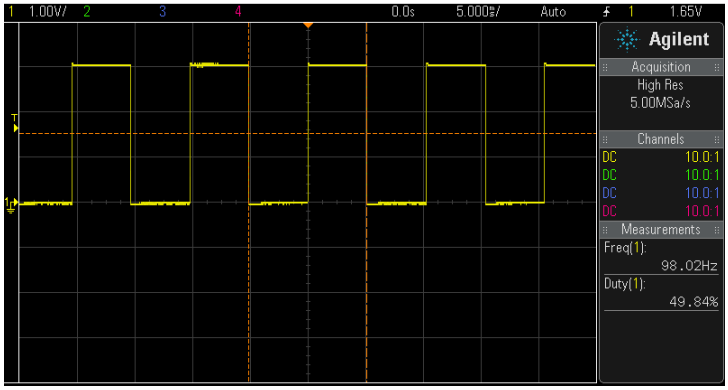
```
# importazione del modulo RPi.GPIO con il nome "GPIO"
import RPi.GPIO as GPIO

# impostazione della numerazione dei pin del connettore GPIO
GPIO.setmode(GPIO.BOARD)
# configurazione del pin 7 del connettore GPIO come output digitale
GPIO.setup(7, GPIO.OUT)
# generazione sul pin 7 di un segnale PWM con frequenza di 100Hz
pwm = GPIO.PWM(7, 100)
# avvio generazione segnale PWM con duty-cycle 50%
pwm.start(50)
```

```
# ciclo di richiesta di nuovi valori per il duty-cycle
while True:
      s = input("[0-100] ")
      dc = int(s)
      # uscita dal ciclo per valori non corretti
      if dc<0 or dc>100:
            break
      # variazione del valore del duty-cycle del segnale PWM
      pwm.ChangeDutyCycle(dc)

# arresto generazione segnale PWM
pwm.stop()
# ripristino della configurazione predefinita del connettore GPIO
GPIO.cleanup()
```

Utilizzando l'oscilloscopio è possibile visualizzare le caratteristiche del segnale generato sul pin del connettore GPIO del microcomputer Raspberry Pi:

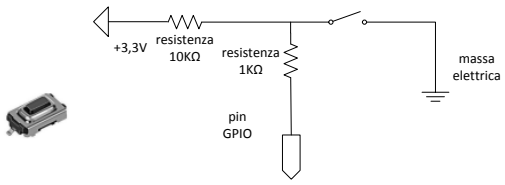


Segnale generato con *duty-cycle* 50%

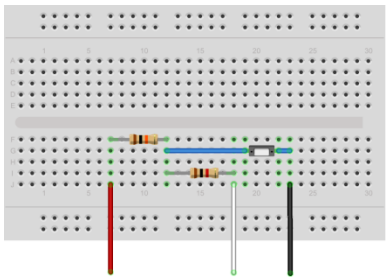
LABORATORIO DIDATTICO 11

Acquisizione dello stato di un pulsante con Raspberry Pi e Python

Il classico esempio per verificare la possibilità di utilizzare un pin del connettore GPIO come input digitale è quello di connettervi un pulsante e di acquisirne lo stato premuto/non-premuto da un programma in linguaggio Python. I due pin del pulsante devono permette di chiudere/aprire un circuito elettrico realizzato tra la tensione di riferimento del connettore GPIO (3,3V) e la massa elettrica, ma per evitare un cortocircuito è necessario inserire nel circuito una resistenza piuttosto elevata:



La seconda resistenza non è invece strettamente necessaria, ma permette di evitare un corto circuito potenzialmente dannoso se il pin del connettore GPIO viene configurato erroneamente o temporaneamente come output anziché come input. Anche in questo caso il semplice circuito elettronico può essere montato utilizzando una *breadboard*:



Il cavetto nero connesso ad un estremo del pulsante deve essere connesso ad una massa elettrica del connettore GPIO del Raspberry Pi, ad esempio il pin 6; il cavetto rosso connesso mediante la resistenza da 10 kΩ all'altro estremo del pulsante deve essere connesso ad una sorgente di alimentazione di 3,3V, ad esempio il pin 1 del connettore GPIO del Raspberry Pi; infine il cavetto bianco connesso mediante la resistenza da 1 kΩ all'estremo del pulsante deve essere connesso ad un pin configurabile come input/output del connettore GPIO del Raspberry Pi, ad esempio il pin 7.
Nell'ipotesi di questo cablaggio, il seguente programma in linguaggio Python visualizza ogni secondo lo stato premuto/non-premuto del pulsante per un periodo di 60s:

```
import time
# importazione del modulo RPi.GPIO con il nome "GPIO"
import RPi.GPIO as GPIO

# impostazione della numerazione dei pin del connettore GPIO
GPIO.setmode(GPIO.BOARD)
# configurazione del pin 7 del connettore GPIO come input digitale
GPIO.setup(7, GPIO.IN)

# acquisizione del tempo di sistema iniziale
```

```
start = time.time()
# ciclo di operazioni ripetute per 60 secondi
while (time.time() - start) < 60.0:
    # acquisizione dello stato dell'input digitale
    if gpio.input(7):
        print("NON premuto.")
    else:
        print("Premuto.")
    # attesa di 1s
    time.sleep(1)

# ripristino della configurazione predefinita del connettore GPIO
GPIO.cleanup()
```

Quando il pulsante non è premuto infatti, il circuito elettrico è aperto e non scorrendo corrente nella resistenze il pin del connettore GPIO ha la stessa tensione della tensione di riferimento di 3,3V che viene interpretata come un "1" logico, corrispondente al valore *True*; quando il pulsante è premuto attraverso la resistenza che connette la tensione di riferimento con la massa elettrica scorre una corrente che determina una caduta della tensione: in questo caso il pin del connettore GPIO ha praticamente la stessa tensione di 0V della massa elettrica che viene interpretata come uno "0" logico, corrispondente al valore *False*.
Un circuito di input di questo tipo è denominato *pull-up* in quanto allo stato "di riposo" del pulsante si ha una tensione sul pin di input che corrisponde al valore logico "1".

Il ciclo del programma utilizzato nel laboratorio didattico precedente realizza una ripetuta acquisizione del valore logico di un pin del connettore GPIO del Raspberry Pi: questa tecnica è denominata ***polling*** e anche se viene diminuita o eliminata l'attesa tra una ripetizione e l'altra del ciclo presenta diversi inconvenienti. Infatti, come il programma dell'esempio precedente non rileva una sequenza di pressione/rilascio del pulsante se essa è più breve di un secondo, se nel ciclo sono eseguite computazioni di una certa durata è possibile che alcune attivazioni del pulsante non siano rilevate.
Dato che in generale interessa non tanto lo stato attuale di un pulsante (cioè se è premuto o non premuto), ma il cambiamento del suo stato (il fatto cioè che viene premuto o rilasciato); il modulo Python RPi.GPIO rende disponibili alcune costanti e funzioni per ottimizzare l'acquisizione delle **transizioni di stato** di un input digitale del connettore GPIO del Raspberry Pi:

Costante/funzione	Descrizione
`RISING`	identifica una transizione dal valore logico "0" al valore logico "1"
`FALLING`	identifica una transizione dal valore logico "1" al valore logico "0"
`BOTH`	identifica una transizione qualsiasi da un valore logico all'altro
`wait_for_edge()`	sospende l'esecuzione del programma in attesa che avvenga una transizione di valore logico su uno specifico input digitale
`add_event_detect()`	registra un evento (una transizione di valore logico su uno specifico input digitale) da rilevare
`remove_event_detect()`	deregistra un evento da rilevare registrato in precedenza
`event_detected()`	restituisce *True* o *False* a seconda che l'evento registrato sia stato o meno rilevato

LABORATORIO DIDATTICO 12

Rilevazione delle transizioni dello stato di un pulsante con Raspberry Pi e Python

Nell'ipotesi del cablaggio del laboratorio didattico precedente, il seguente programma in linguaggio Python visualizza ogni singola pressione del pulsante per 10 volte:

```
import time
# importazione del modulo RPi.GPIO con il nome "GPIO"
import RPi.GPIO as GPIO

# impostazione della numerazione dei pin del connettore GPIO
GPIO.setmode(GPIO.BOARD)
# configurazione del pin 7 del connettore GPIO come input digitale
GPIO.setup(7, GPIO.IN)

# ciclo di operazioni ripetute per 10 volte
for n in range(10):
    # attesa di una transizione 1 -> 0 dell'input digitale
    GPIO.wait_for_edge(7, gpio.FALLING)
    print("Premuto.")

# ripristino della configurazione predefinita del connettore GPIO
GPIO.cleanup()
```

Il seguente programma in linguaggio Python invece visualizza ogni singola pressione del pulsante per un periodo di 60 secondi:

```
import time
# importazione del modulo RPi.GPIO con il nome "GPIO"
import RPi.GPIO as GPIO

# impostazione della numerazione dei pin del connettore GPIO
GPIO.setmode(GPIO.BOARD)
# configurazione del pin 7 del connettore GPIO come input digitale
GPIO.setup(7, GPIO.IN)
# registrazione della transizione 1 -> 0 sul pin 7 del connettore GPIO
GPIO.add_event_detect(7, GPIO.FALLING)

# acquisizione del tempo di sistema iniziale
start = time.time()
# ciclo di operazioni ripetute per 60 secondi
while (time.time() - start) < 60.0:
    # verifica di rilevazione dell'evento registrato
    if GPIO.event_detected(7):
        print("Premuto.")

# deregistrazione dell'evento
GPIO.remove_event_detect(7)
# ripristino della configurazione predefinita del connettore GPIO
GPIO.cleanup()
```

I due programmi producono lo stesso risultato, ma la tecnica di programmazione utilizzata nel secondo permette di eseguire nel ciclo anche altre operazioni, senza il rischio di non rilevare la transizione del segnale di input registrata come evento.
Il programma che segue invece mostra come delegare ad una funzione le operazioni da eseguire quando viene rilevato un evento costituito da una transizione del segnale di

input in modo indipendente dal flusso di esecuzione principale che, in questo caso, visualizza un "conto alla rovescia":

```
import time
# importazione del modulo RPi.GPIO con il nome "GPIO"
import RPi.GPIO as GPIO

# funzione delegata alla gestione dell'evento registrato
def pushed(channel):
    print("Premuto.")

# impostazione della numerazione dei pin del connettore GPIO
GPIO.setmode(GPIO.BOARD)
# configurazione del pin 7 del connettore GPIO come input digitale
GPIO.setup(7, GPIO.IN)
# registrazione della transizione 1 -> 0 sul pin 7 del connettore GPIO
# viene delegata alla gestione dell'evento la funzione "pushed"
GPIO.add_event_detect(7, gpio.FALLING, callback=pushed)

# ciclo di operazioni ripetute per 60 volte
for n in range(60):
    # visualizzazione "conto alla rovescia"
    print(str(60 - n))
    time.sleep(1)

# deregistrazione dell'evento
GPIO.remove_event_detect(7)
# ripristino della configurazione predefinita del connettore GPIO
GPIO.cleanup()
```

Eseguendo i programmi precedenti, in tutti i casi può accadere che alcune singole transizioni siano rilevate due o più volte: questo è dovuto al fatto che la pressione del pulsante può causare alcuni rimbalzi meccanici che effettivamente hanno come conseguenza transizioni multiple del valore di tensione sul pin del connettore GPIO del Raspberry Pi configurato come input digitale.
Allo scopo di evitare le conseguenze del fenomeno del "rimbalzo" di un pulsante la funzione `add_event_detect()` del modulo Python RPi.GPIO prevede un parametro opzionale denominato *bouncetime* che consente di specificare un tempo espresso in millisecondi di attesa per la stabilizzazione del segnale.
Il codice del programma del laboratorio didattico precedente può essere modificato per evitare che eventuali transizioni spurie del livello di tensione generate dal pulsante si riflettano nella rilevazione di eventi indesiderati da parte del programma:

```
import time
# importazione del modulo RPi.GPIO con il nome "GPIO"
import RPi.GPIO as GPIO

# registrazione del tempo di inizio dell'esecuzione del programma
start = time.time()

# funzione delegata alla gestione dell'evento registrato: visualizza
# il tempo trascorso dall'inizio dell'esecuzione del programma
def pushed(channel):
    print(str(time.time() - start))

# impostazione della numerazione dei pin del connettore GPIO
GPIO.setmode(GPIO.BOARD)
```

```
# configurazione del pin 7 del connettore GPIO come input digitale
GPIO.setup(7, GPIO.IN)
# registrazione della transizione 1 -> 0 sul pin 7 del connettore GPIO
# viene delegata alla gestione dell'evento la funzione "pushed"
# il tempo di filtraggio dei "rimbalzi" è impostato a 1ms
GPIO.add_event_detect(7, GPIO.FALLING, callback=pushed, bouncetime=1)

# attesa di 60 secondi
time.sleep(60)

# deregistrazione dell'evento
GPIO.remove_event_detect(7)
# ripristino della configurazione predefinita del connettore GPIO
GPIO.cleanup()
```

Rispetto al programma originale non viene visualizzato il conto alla rovescia, ma ogni volta che si preme il pulsante viene visualizzato il numero di secondi trascorso dall'inizio dell'esecuzione.