PWM

Con i microcontrollori

Introduzione

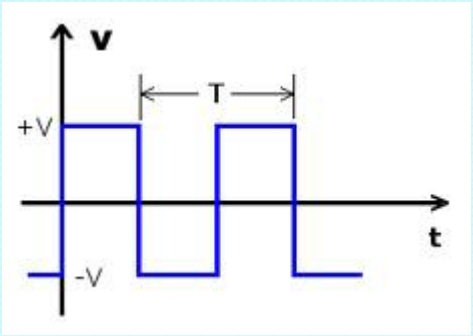
- **PWM= Pulse Width Modulation**. Modulazione digitale che permette di ottenere una tensione media variabile dipendente dal rapporto tra la durata dell'impulso positivo e quello negativo.
- Nella trasmissione tramite modulazione PWM si usa come supporto un segnale a onda quadra, di frequenza e ampiezza fissate, nel quale la larghezza, ovvero la durata degli impulsi, è proporzionale al livello del segnale modulante.
- Si utilizza nei protocolli di comunicazione in cui l'informazione è codificata sottoforma di durata nel tempo di ciascun impulso
- In elettrotecnica viene utilizzata per variare la potenza elettrica inviata ad un carico

PWM in elettronica

- Nei controlli automatici si ha l'esigenza di controllare grandezze analogiche tramite microcontrollori. I micro sono dotati di un convertitore analogico/digitale
- Se si ha l'esigenza di ottenere grandezze analogiche dai microcontrollori, si utilizza la PWM

PWM e onda quadra

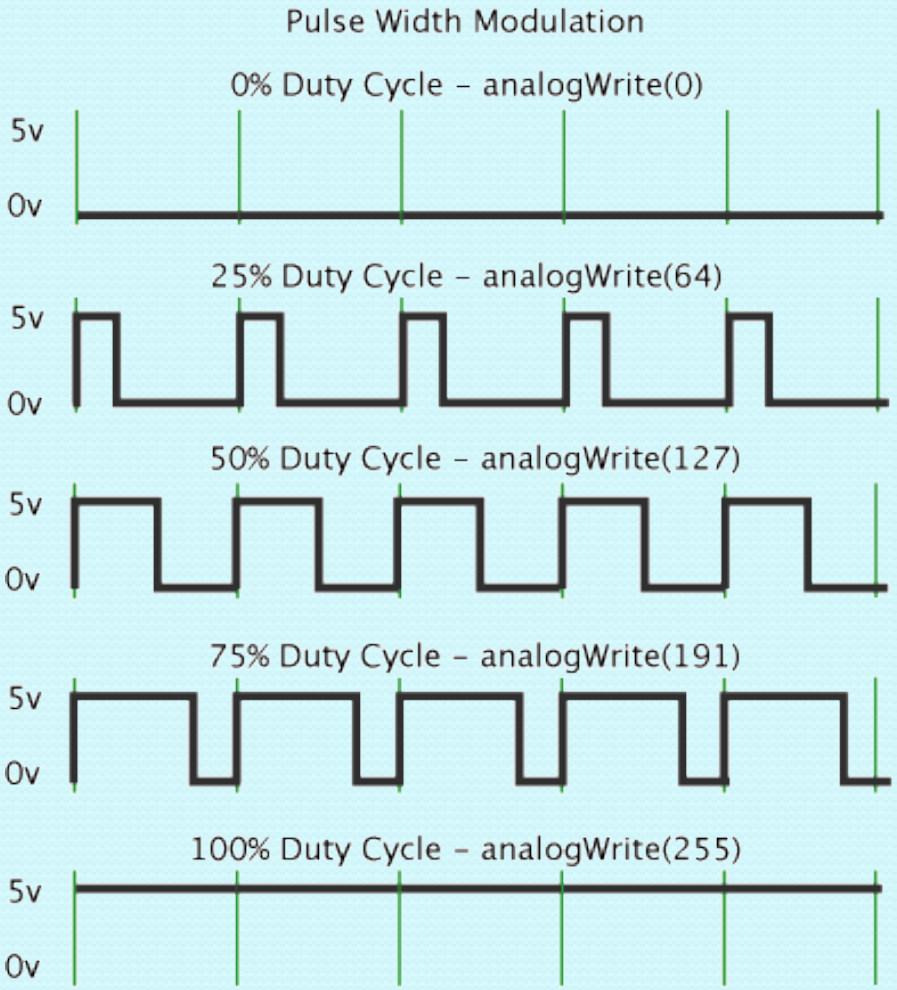
- Un segnale PWM è un'onda quadra con la larghezza variabile, duty cycle variabile. Questa variazione permette di controllare l'assorbimento di potenza di un carico elettrico.
- Un'onda quadra si presenta nel seguente modo



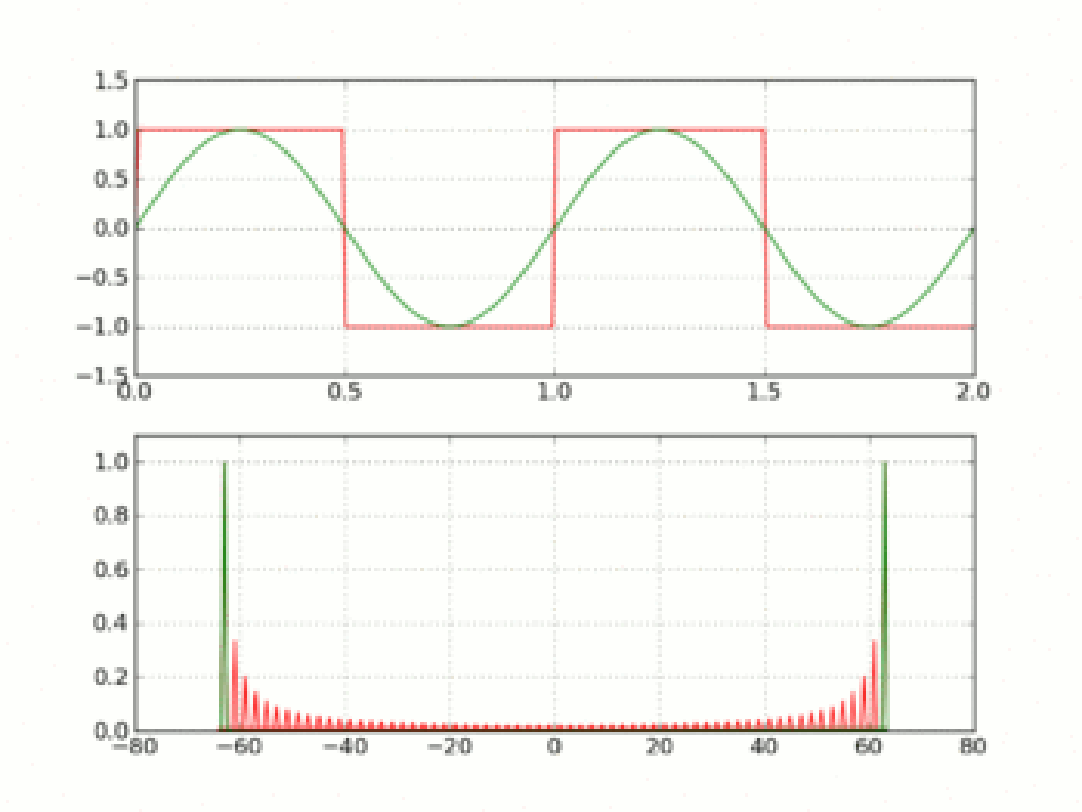
PWM e onda quadra

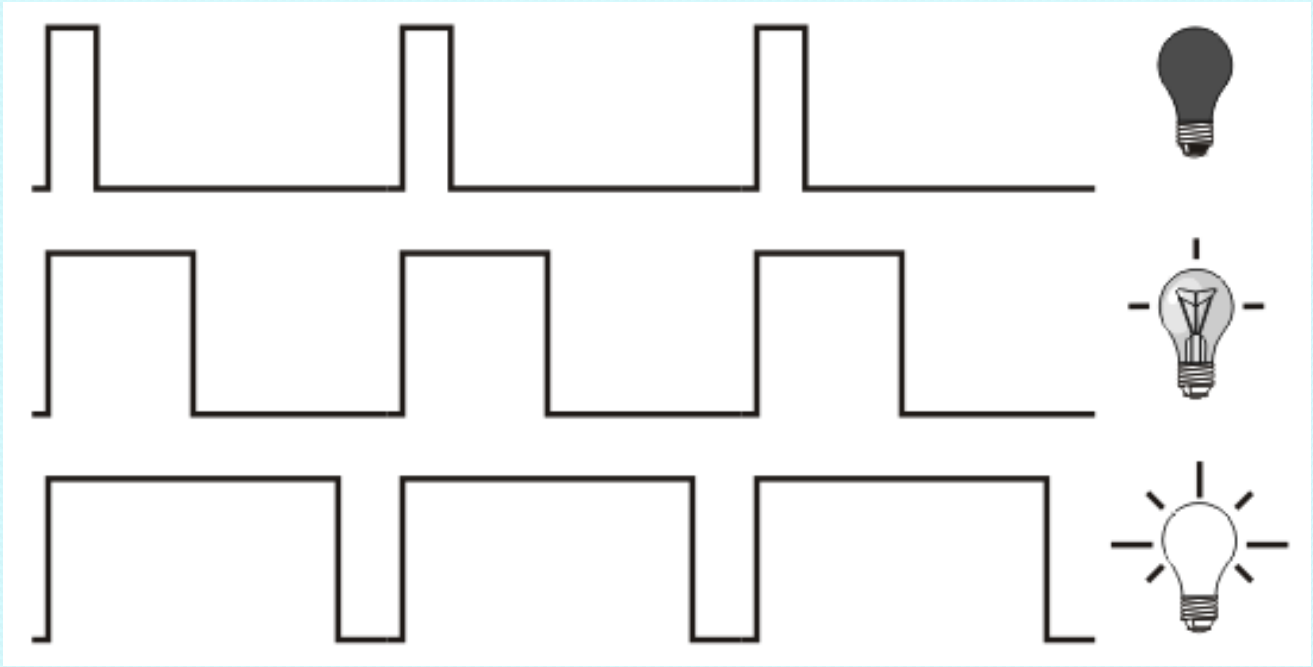
- L'onda quadra è un segnale periodico, cioè un segnale che si ripete allo stesso modo ad intervalli di tempo uguale detto periodo T.
- Le caratteristiche di un segnale periodico sono: periodo e ampiezza.
- Le caratteristiche di un'onda quadra sono oltre al periodo e all'ampiezza, anche il duty cycle
- Il duty cycle è il rapporto tra il tempo t_h in cui l'onda assume valore alto e l'intero periodo T dell'onda stessa: t_h / T
  - Se il tempo in cui il segnale è alto è uguale a quello in cui il segnale è basso, il duty cycle è del 50% ;
  - se il segnale è sempre alto, il duty cycle è 100%;
  - Se il segnale è sempre basso, il duty cicle è 0%

Duty Cycle


Pulse Width Modulation
0% Duty Cycle – analogWrite(0)
5v
0v
25% Duty Cycle – analogWrite(64)
5v
0v
50% Duty Cycle – analogWrite(127)
5v
0v
75% Duty Cycle – analogWrite(191)
5v
0v
100% Duty Cycle – analogWrite(255)
5v
0v

Onda quadra come sovrapposizione di armoniche





Specifiche tecniche

Nei microcontrollori e nei microprocessori la PWM può essere realizzata generando un segnale periodico fatto nel seguente modo:

-Durante un intervallo di tempo il segnale è alto

-Nell'intervallo successivo il segnale è basso

Gli intervalli temporali devono essere piccolissimi, dellordine del millisecondo

Tuttavia, esiste un metodo molto semplice perché alcuni registri e i relativi pin sono dedicati alla PWM. Esiste una frequenza base per il segnale che può essere cambiata attraverso dei fattori detti di pescaler

PWM con Arduino al 10%

```
void setup() { pinMode(13, OUTPUT); }
void loop() { digitalWrite(13, HIGH);
   delayMicroseconds(100);
// circa il 10% del duty cycle con f= 1KHz
digitalWrite(13, LOW);
delayMicroseconds(900); }
```

T=(100+900)microsecondi=1000 microsecondi=$1000*10^{-6}=10^{-3}$ secondi=1 ms

f=1/T=10^3 Hz=1kHz

dc=(100 microsecondi)/(1000microsecondi)*100=10%

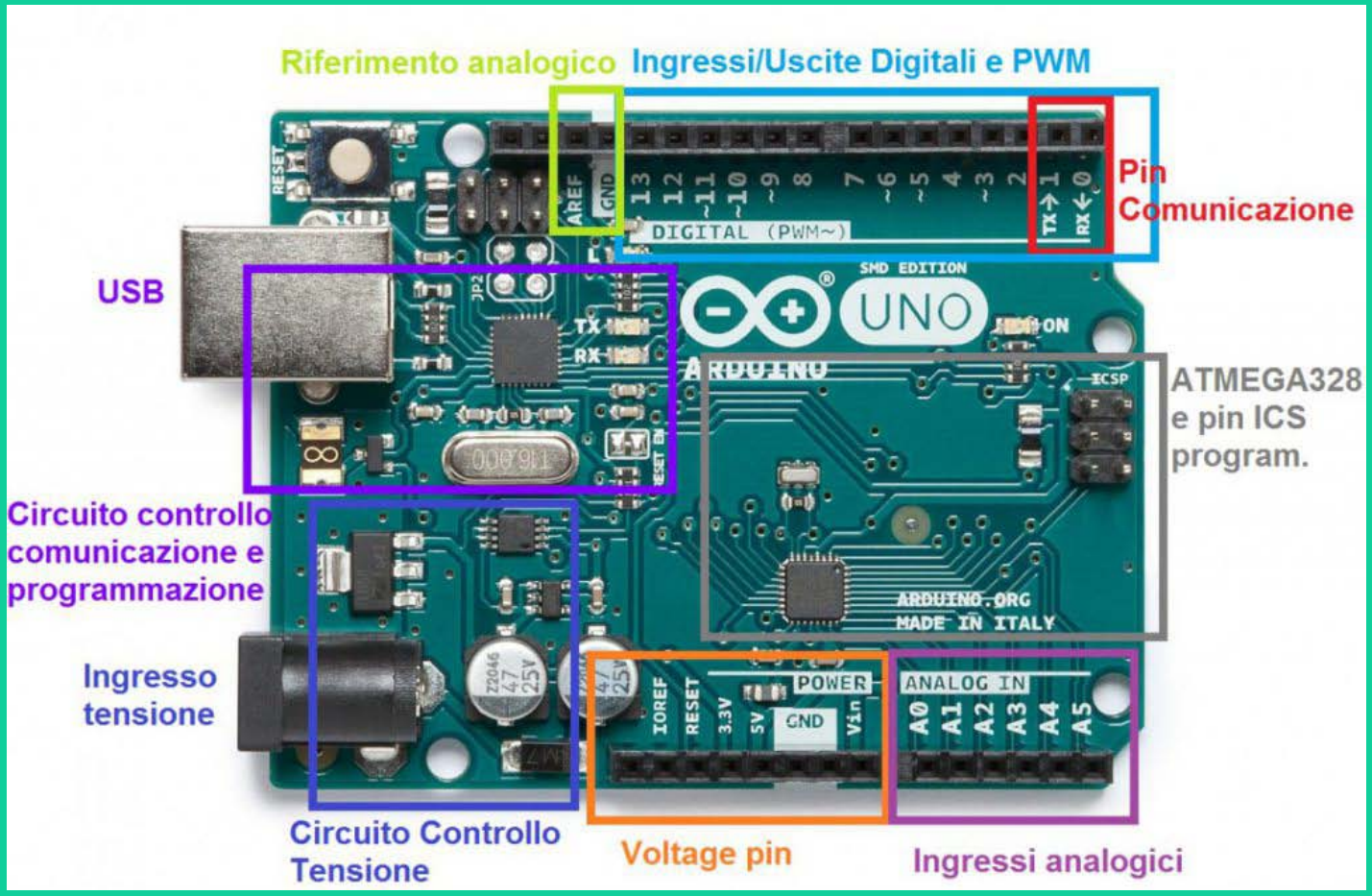
PWM con Arduino all'80%

```
void setup()
{ pinMode(13, OUTPUT); }
void loop() { digitalWrite(13, HIGH);
   delayMicroseconds(800);
// circa il 80% del duty cycle con f= 1KHz
digitalWrite(13, LOW);
delayMicroseconds(200); }
//T=(800+200)microsecondi=1000 microcsecondi
 //dc=(800/1000)*100=80%
```

PWM Arduino

- Per arduino uno solo alcuni pin sono dedicati alla PWM:3,5, 6, 9,10,11 e sono anche a frequenza differenti
- La differenza di frequenza è dovuta al fatto che i pin sono collegati su memorie di 8 bit e di 16 bit
- I pin 3, 9,10 e 11 sono a 16 bit, gli altri a 8 bit
- Il valore di prescaler è quello con divisore 8
- Sui pin 5 e 6 si trova una PWM a frequenza pari a 976 Hz mentre sui pin 3,9,10,11 la frequenza è 490 Hz
- Sulla scheda Arduino i pin con la PWM sono indicati con il simbolo ~

Scheda Arduino UNO



Fattore di prescaler

T_{ck}= 62.5 ns f_{ck}=16 MHz

k	Ts	fs
1	16 µs	62,5 kHz
8	128 µs	7,81 kHz
32	512 µs	1,95 kHz
64	1,024 ms	976,5 Hz
128	2.048 ms	488,8 Hz
256	4,096 ms	244,1 Hz
1024	16,38 4ms	61 Hz

k fattore di prescaler
Ts periodo del segnale PWM
fs frequenza del segnale PWM (1/Ts)

Fattori di prescaler

Table 10-2. Watchdog Timer Prescale Select

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 5.0V
0	0	0	0	2K (2048) cycles	16 ms
0	0	0	1	4K (4096) cycles	32 ms
0	0	1	0	8K (8192) cycles	64 ms
0	0	1	1	16K (16384) cycles	0.125 s
0	1	0	0	32K (32768) cycles	0.25 s
0	1	0	1	64K (65536) cycles	0.5 s
0	1	1	0	128K (131072) cycles	1.0 s
0	1	1	1	256K (262144) cycles	2.0 s
1	0	0	0	512K (524288) cycles	4.0 s
1	0	0	1	1024K (1048576) cycles	8.0 s
1	0	1	0	Reserved	
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

Esempio PWM Arduino

```
int led =3;
 void setup(){
 pinMode(led,OUTPUT);}
 void loop(){
 analogWrite(led, 125);//dc al 50%
delay(1000);
analogWrite(led, 255); //dc al 100%
 delay(2000);
 analogWrite(led, 90);dc al 35%
 delay(3000);
}
```

Per far variare il valore di duty cycle, si scrive *analogWrite(nomepin, valore);*
Il valore varia da 0 255 perché stiamo lavorando a 8 bit

Un terzo programma PWM per Arduino

```
#define LED 11
 int valoreFade = 0;
void setup()  {
  pinMode(LED, OUTPUT);     }
void loop()
   {
for (valoreFade = 0 ; valoreFade < 255; valoreFade++) {
 analogWrite(LED, valoreFade);
 delay(10);                              }
for(valoreFade = 255 ; valoreFade > 0; valoreFade--
    ) {      analogWrite(LED, valoreFade);
 delay(10);                 }   }
```

PWM, frequenza e relativi registri di Arduino UNO

Timer	Frequenza Base	Output Compare		Pin Arduino	Frequenza Default
TCCR0	62500Hz	OC0	OC0A	6	976Hz
	62500Hz		OC0B	5	976Hz
TCCR1	31250Hz	OC1	OC1A	9	488Hz
	31250Hz		OC0B	10	488Hz
TCCR2	31250Hz	OC2	OC1A	11	488Hz
	31250Hz		OC2B	3	488Hz

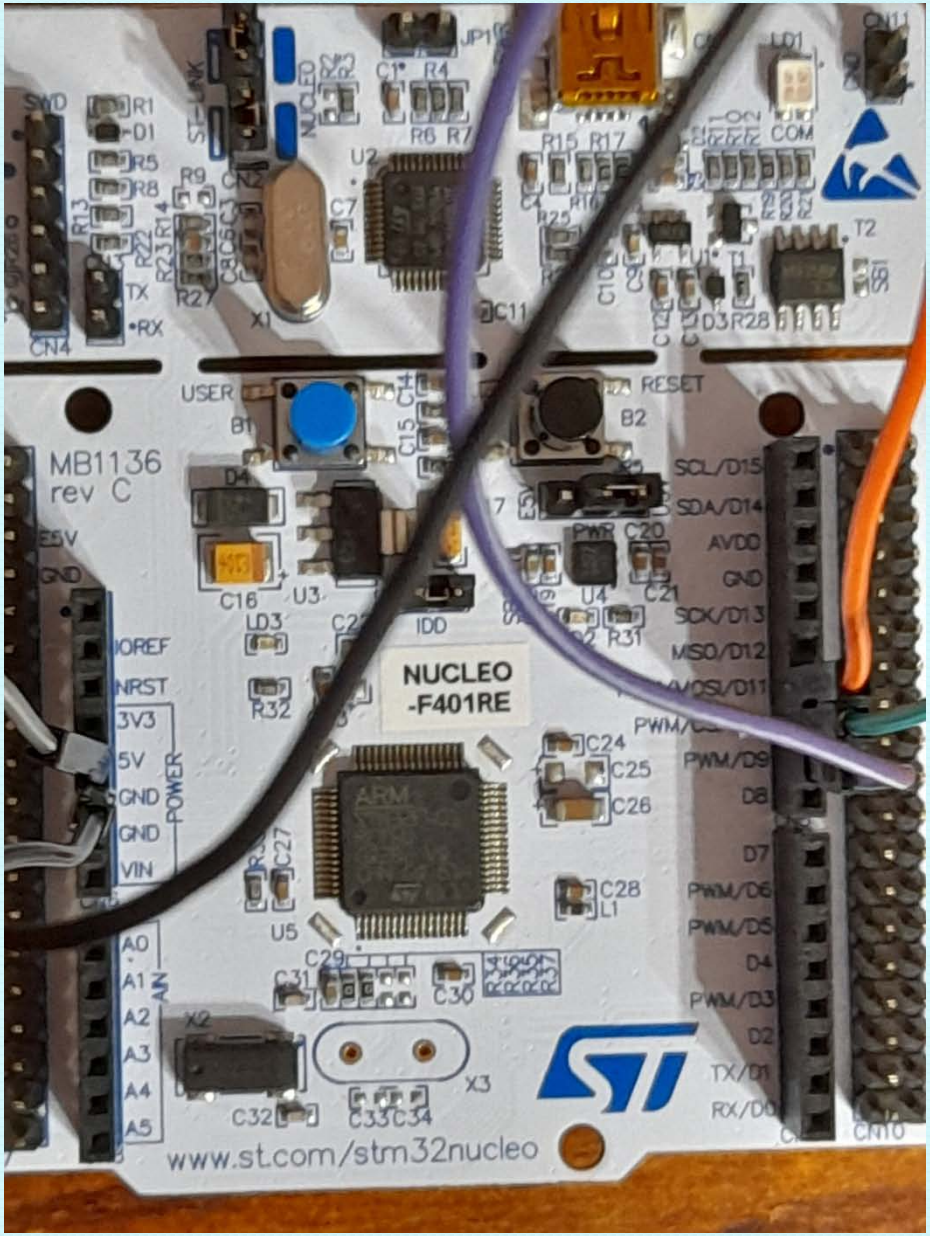
Cambiare la frequenza di PWM

```
void setPwmFrequency(int pin, int divisor) {
  byte mode;
  if(pin == 5 || pin == 6 || pin == 9 || pin == 10) {
    switch(divisor) {
      case 1: mode = 0x01; break;
      case 8: mode = 0x02; break;
      case 64: mode = 0x03; break;
      case 256: mode = 0x04; break;
      case 1024: mode = 0x05; break;
      default: return;
    }
    if(pin == 5 || pin == 6) {
      TCCR0B = TCCR0B & 0b11111000 | mode;
    } else {
      TCCR1B = TCCR1B & 0b11111000 | mode;
    }
  }
  else if(pin == 3 || pin == 11) {
    switch(divisor) {
      case 1: mode = 0x01; break;
      case 8: mode = 0x02; break;
      case 32: mode = 0x03; break;
      case 64: mode = 0x04; break;
      case 128: mode = 0x05; break;
      case 256: mode = 0x06; break;
      case 1024: mode = 0x7; break;
      default: return;
    }
    TCCR2B = TCCR2B & 0b11111000 | mode;
  }
}
```

PWM STM32

Nella scheda STM32 F401 i pin dedicati alla PWM sono gli stessi della scheda Arduino: D3, D5, D6, D9, D10

Il periodo di onda quadra è più gestibile rispetto alla scheda Arduino in quanto viene settato ogni volta da programma.
Il limite inferiore di periodo è 1 us- 2us



PWM STM32

```
#include "mbed.h"
PwmOut rled(D9);
int main() {
    rled = gled = bled = 1;
    while(1) {
        rled.period_ms(25);
        rled=0.0001;
        }
        }
```

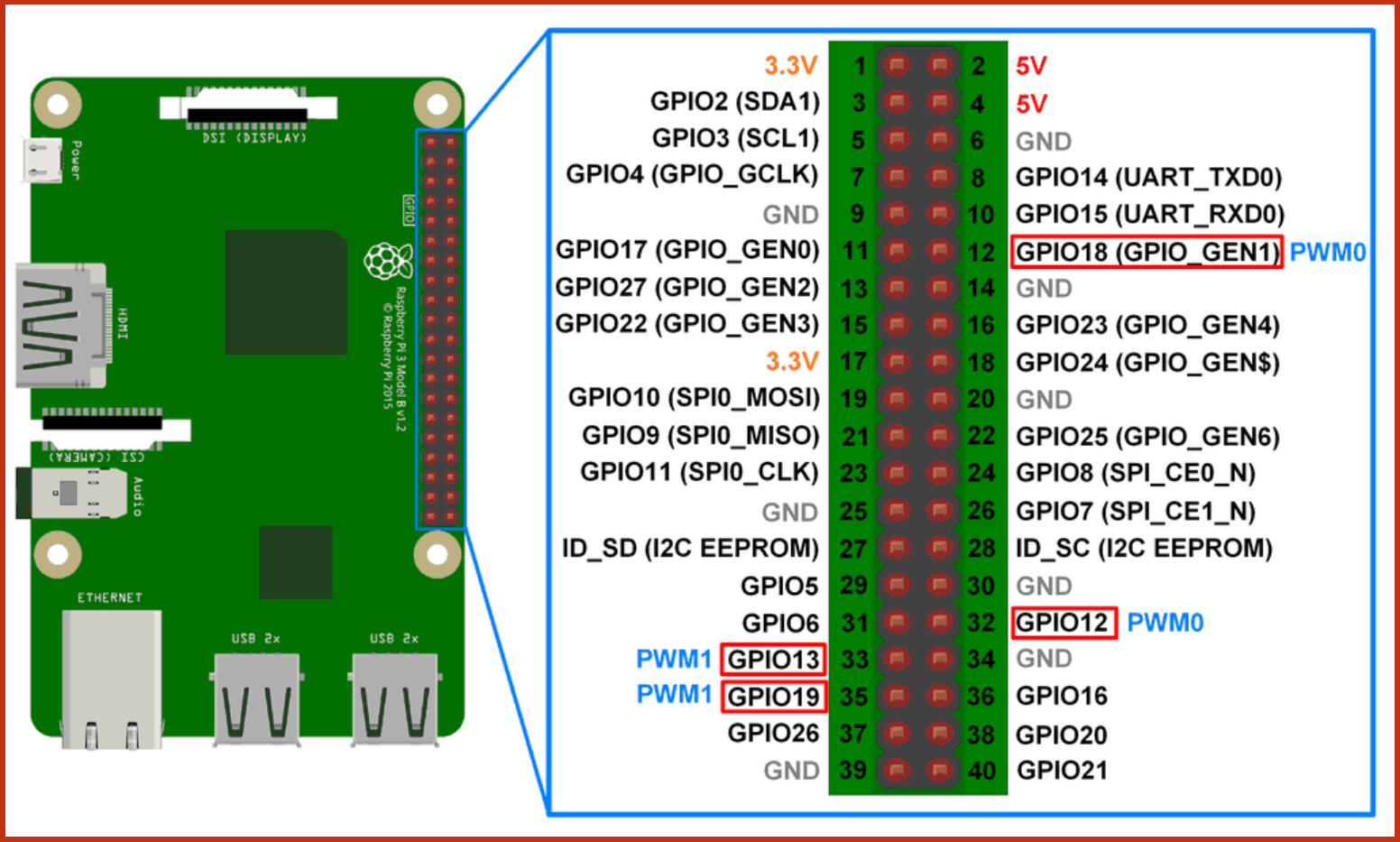
PWM STM32 con RGB

```
#include "mbed.h"
PwmOut rled(D9);
PwmOut gled(D10);
PwmOut bled(D11);
int main() {
    rled = gled = bled = 1;
    while(1) {
        for(float i=0.0; i<=1.0; i+=0.005) {
            rled = i;
            wait(0.01);
        }
        for(float i=0.0; i<=1.0; i+=0.005) {
            gled = i;
            wait(0.01);
        }
        for(float i=0.0; i<=1.0; i+=0.005) {
            bled = i;
            wait(0.01);
        }
    }
}
```

PWM Raspberry

```
import RPi.GPIO  as GPIO
import time
 GPIO.setmode(GPIO.BOARD)
 GPIO.setup(12, GPIO.OUT)
p=GPIO.PWM(12, 70)
 p.start(0)
 p.ChangeDutyCycle(30)
time.sleep(2)
p.ChangeDutyCycle(0)
p.stop()
GPIO.cleanup()
```

Pin Raspberry



- Il pin PWM è il 12 secondo la numerazione BOARD che coincide con il 18 secondo la numerazione BCM.
- Si può generare un segnale PWM sui pin 12, 32,33,35 ONBOARD di Raspberry, che corrispondono come BCM a GPIO18, GPIO12, GPIO13, GPIO19
- Comandi tipici della PWM
  - p=GPIO.PWM(12,70) si da un nome alla inizializzazione della PWM sul pin 12 a 70 Hz, si crea una istanza
  - p.start(0) il duty cycle di partenza è zero
  - p.ChangeDutyCycle(30) porta il duty cycle al 30%
  - p.stop() termina la pwm

Cambiare la PWM da tastiera

```
import RPi.GPIO as GPIO
 GPIO.setmode(GPIO.BCM)
 GPIO.setup(18, GPIO.OUT)
 lum= GPIO.PWM(18, 500)
 lum.start(100)
 while True:
       duty_s = raw_input(“Inserisci la luminosità (0      to 100):")
     duty = int(duty_s)
       lum.ChangeDutyCycle(duty)
```

Specifiche del programma precedente

- raw_input() acquisisce un dato da tastiera e lo chiama duty_s
- int(duty_s) trasforma un dato che viene visto come carattere, in intero e lo chiama duty

- Per poter inserire il dato da tastiera, bisogna eseguire il programma dalla shell di ubuntu con il codice nomefile.py
- Il programma attende un comando e poi esegue le istruzioni

Luminosità variabile

```
import RPi.GPIO as GPIO
 import sleep ledpin = 12
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD) GPIO.setup(ledpin,GPIO.OUT)
pi_pwm = GPIO.PWM(ledpin,1000)
 pi_pwm.start(0)
while True:
    for duty in range(0,101,1):   #il led si accende piano piano da 0% a 100%
          pi_pwm.ChangeDutyCycle(duty)
          sleep(0.01)
    sleep(0.5)
    for duty in range(100,-1,-1): #il led si spegne piano piano da 100% a 0%
          pi_pwm.ChangeDutyCycle(duty)
         sleep(0.01)
    sleep(0.5
```

Frequenza di pwm

-

Frequenza richiesta	Frequenza misurata
50 Hz	50 Hz
100 Hz	98.7 Hz
200 Hz	195 Hz
500 Hz	470 Hz
1 kHz	980 Hz
10 kHz	4.4 Hz

Importante notare che all'aumentare della frequenza, diminuisce la stabilità