SAPIENZA
UNIVERSITÀ DI ROMA

ESERCITAZIONE 8: FAMILIARIZZAZIONE CON ARDUINO

G. Galbato Muscio L. Gravina L. Graziotto

11 Dicembre 2018

GRUPPO 11

Abstract

Si studia il tempo impiegato dal microcontrollore Arduino UNO a svolgere determinate istruzioni, da operazioni aritmetiche, a calcolo di funzioni, a operazioni di Input/Output. Si studia il comportamento dei pin dotati di *pulse width modulation*. Si realizza, mediante la funzione di lettura analogica, un ADC, e se ne calcolano i valori di calibrazione. Si compie infine un campionamento digitale di un segnale analogico periodico.

Indice

1 Comunicazione seriale

Si realizza un programma per Arduino (Listato 1) finalizzato a misurare i tempi impiegati per lo svolgimento di alcune istruzioni, e la dipendenza di tali tempi dall'ordine di grandezza dei numeri da manipolare. La funzione `micros()` permette di registrare i tempi di esecuzione, che vengono riportati in Tabella 1: in questa prima fase, i numeri sono di tipo `float` con una cifra dopo la virgola, e il tempo di esecuzione è misurato escludendo quello impiegato per la scrittura su schermo del risultato.

```
float a, b, c;
long unsigned t0, t1;
void setup() {
  Serial.begin(9600);
  a = 2.68;
  b = 11.9;
}

void loop() {
  t0 = micros();
    c = a+b; //somma
    //c = a*b; //prodotto
    //c = a/b; //divisione
    //c = sqrt(a); //radice quadra
    //c = sin(a); //seno
    //c = max(a,b); //massimo
    //c = a*a; //quadrato
    //c = pow(a,2); //quadrato con
    funzione pow
  t1 = micros();
  Serial.print(t0);
  Serial.print(" ");
  Serial.print(t1);
  Serial.print(" ");
  Serial.println(t1-t0);
}
```

Listing 1: Codice per la misura dei tempi di esecuzione di operazioni elementari

Tabella 1: Misure per la stima dei tempi di esecuzione

Operazione	t_0 [µs]	t_1 [µs]	Δt [µs]
$a+b$	312	324	12
$a \cdot b$	312	328	16
a/b	336	372	36
$\sqrt{a}$	796	834	38
$sin(a)$	500	612	112
$max\{a,b\}$	264	272	8
a^2	1208	1220	12
$pow(a,2)$	1208	1220	12

Per ottenere una stima più precisa, si ripete ogni operazione inserendola all'interno di un ciclo `for` che la iteri per $N = 1000$ volte; quindi si ottiene il tempo medio di esecuzione come $(t_1 - t_0)/N$. Si riportano i risultati in Tabella 2; anche in questo caso non sono inclusi i tempi necessari alla scrittura a schermo. Si osserva che in questo secondo caso i tempi sono maggiori fino al 25% (nel caso della somma), poiché è incluso il tempo di esecuzione del ciclo `for`.

Tabella 2: Misure per la stima dei tempi di esecuzione, con $N = 1000$ iterazioni

Operazione	t_0 [µs]	t_1 [µs]	$\Delta t/N$ [µs]
$a+b$	17148	32956	15.8
$a \cdot b$	18908	34692	17.6
a/b	80832	119840	39.0
$\sqrt{a}$	158988	197184	38.2
$sin(a)$	235864	352236	116.4
$max\{a,b\}$	63096	71048	8.0
a^2	18088	34852	16.8
$pow(a,2)$	18088	34852	16.8

Per l'operazione di moltiplicazione, si studia la dipendenza del tempo di esecuzione dal numero di cifre: con il ciclo `for` si varia il numero a, in questo caso intero, da moltiplicare per 10, da 0 a 10^5 a passi di 1, e si misura volta per volta il tempo di esecuzione (si veda il Listato 2). Si osserva in questo caso che il tempo rimane circa costante.

```
#define MAX 10000

int p, q;
long unsigned t0, t1;

void setup() {
  Serial.begin(9600);
  p=10;
}

void loop() {
  Serial.print(p);
  t0 = micros();
  q = p*10;
  t1 = micros();
  p = p+1;
  Serial.print(p);
  Serial.print(" ");
  Serial.println(t1-t0); //stampa il
    tempo
  while (p > MAX) { }; //il programma si
     arresta quando il numero supera MAX
```

```
  }
```

Listing 2: Codice per la misura dei tempi di esecuzione della moltiplicazione

Per stimare il tempo impiegato a scrivere su schermo si realizza una stringa di caratteri di lunghezza via via maggiore, che viene quindi stampata a video, come da Listato 3. Si riporta in Figura ?? il grafico del tempo di esecuzione dell'istruzione di output a video in funzione del numero di caratteri della stringa.

```
#define LMAX 100 //Lunghezza stringa

int i=0;
long unsigned t0, t1;
String string;

void setup() {
  Serial.begin(9600);
}

void loop() {
    string += "a";
    t0 = micros();
    Serial.print(string);
    t1 = micros();

    i = i+1;
    Serial.print(" ");
    Serial.print(i);
    Serial.print(" ");
    Serial.println(t1-t0);

    while(i>LMAX) { }
}
```

Listing 3: Codice per la misura dei tempi di scrittura a video

2 Operazioni di Input Analogico

Si utilizzano i pin digitali in modalità *pulse width modulation* (PWM), che permette di ottenere un'onda quadra con *duty cycle* variabile scegliendo un valore compreso tra 0 e 255, si veda il Listato 4. La frequenza è fissata: per il pin `9` è di (490 ± 14) Hz, mentre per il pin `6` di (976 ± 30) Hz, misurate entrambe con l'oscilloscopio.

[1]La frequenza è sufficientemente alta da non distinguere tra livello logico alto e basso, ossia non si percepisce la luminosità del LED come oscillante.

Si connette innanzitutto al pin dotato di PWM un LED protetto verso massa da una resistenza $R = (0.998 \pm 0.005)\,\mathrm{k\Omega}$, misurata con il multimetro: si osserva l'aumento di luminosità del LED al crescere del *duty cycle*, poiché il tempo per periodo in cui il livello logico è alto aumenta[1]. Successivamente si connette direttamente il pin `9` al canale `CH1` dell'oscilloscopio, e si riportano le istantanee per valori del *duty cycle* del 20% e del 78% rispettivamente nelle Figure 1 e 2.

```
int ledPin = 9;
int val = 128;
void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
  Serial.println("Ready");
  Serial.println(" ");
}

void loop() {
  if (Serial.available() > 0) {
    val=Serial.parseInt();
  // check value in
     if (val>=0 && val <=255) {
      analogWrite(ledPin, val);
      Serial.print("Scritto valore :");
      Serial.println(val);
     }
     else {
      Serial.println("Valore fuori range
     (0:255)!!!");
     }
  }
}
```

Listing 4: Codice per la scrittura PWM

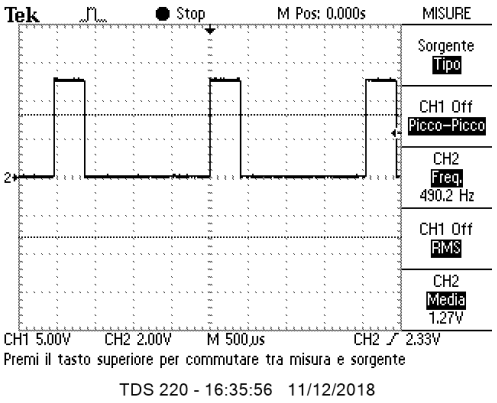

Figura 1: Istantanea dell'oscilloscopio per il pin 9 in PWM: *duty cycle* del 20%

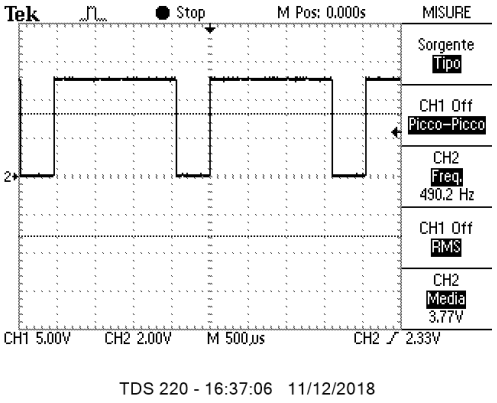

Figura 2: Istantanea dell'oscilloscopio per il pin 9 in PWM: *duty cycle* del 78%

Si ripete l'operazione di scrittura analogica con la PWM utilizzando un'onda triangolare per modulare la durata del *duty cycle*, come da Listato 5; per ottenere un'onda triangolare visibile sull'oscilloscopio a partire da quella triangolare di modulazione PWM, si realizza un filtro RC integratore. Si sceglie

- $R = (0.328 \pm 0.002)\,\mathrm{k\Omega}$;
- $C = (83.7 \pm 0.4)\,\mu\mathrm{F}$,

misurati con multimetro e ponte, in modo da avere una bassa frequenza di taglio, molto inferiore alla frequenza dell'onda modulata (che è di 200 Hz):

$$f_T = \frac{1}{RC} = (36.4 \pm 0.4)\,\mathrm{Hz}.$$

Si riporta in Figura 3 lo screenshot dell'oscilloscopio per il segnale in uscita.

```
int ledPin = 9;
int val,i;
int millisecondi = 5;

void setup() {
  i = val = 0;
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
  Serial.println("Ready");
}

void loop() {
  for(i=0;i<255;i++){
    val = i;
    analogWrite(ledPin,val);
    delay(millisecondi);
  }
    for(i=0;i<255;i++){
    val = 255-i;
    analogWrite(ledPin,val);
    delay(millisecondi);
  }
}
```

Listing 5: Codice per la scrittura PWM, modulazione triangolare

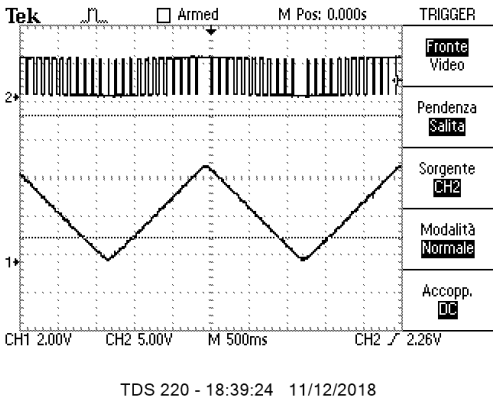

Figura 3: Istantanea dell'oscilloscopio per il pin 9 in PWM, onda triangolare

Si ripete la stessa identica procedura per un segnale modulato con una PWM sinusoidale. Il programma è riportato nel Listato 6, e l'istantanea dell'oscilloscopio per questa configurazione in Figura 4.

```arduino
int ledPin = 9;
int val;
float x;
float dx = 0.01;
int millisecondi = 1;

void setup() {
  val = 0;
  x=0.;
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
  Serial.println("Ready");
}

void loop() {
  val = (int)(127.5*sin(x)+127.5);
  Serial.println(val);
  analogWrite(ledPin,(int)(val));
  x += dx;
  delay(millisecondi);
}
```

Listing 6: Codice per la scrittura PWM, modulazione sinusoidale

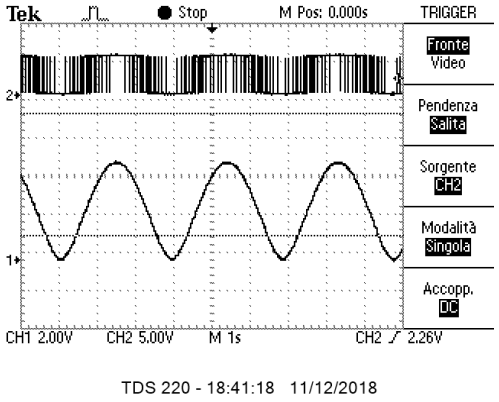

Figura 4: Istantanea dell'oscilloscopio per il pin 9 in PWM, onda sinusoidale

In entrambi gli screenshot è riportata in alto l'uscita del pin 9 di Arduino, che rappresenta l'onda quadra *pulse width modulated*: si notano gli addensamenti e le rarefazioni di linee in corrispondenza dei massimi e dei minimi delle funzioni triangolare e sinusoidale.

3 Calibrazione ADC

Si utilizza il pin A3 di ingresso analogico, al quale è inviata una tensione continua compresa tra 0 e 5V mediante il generatore di tensione. Mediante il Listato 7, si converte il valore analogico della tensione V_{in} in un valore digitale Val_V a 10 bit (dunque compreso tra 0 e $2^{10}-1 = 1023$), proporzionale al primo. La tensione in ingresso V_{in} è misurata con il multimetro; nel caso ideale la tensione corrispondente al valore digitale acquisito da Arduino dovrebbe essere

$$V_x = \frac{\text{Val}_V}{1023} \cdot V_{\text{ref}},$$

dove $V_{\text{ref}} = 5.0\,\text{V}$ è la tensione di riferimento qui impostata. Si confronterà successivamente l'aderenza dei dati sperimentali a questa formula. Si riportano in Tabella 3 le misure per la calibrazione dell'ADC. Il programma scritto è riportato nel Listato 7.

```arduino
int analogPin = 3;
int RdVal = 0;
float Vx = 0.0;
void setup() {
  analogReference(DEFAULT); // 5 V
  Serial.begin(9600);
  Serial.println("Arduino Voltage meter"
    );
  Serial.println(" ");
}

void loop() {
  RdVal = analogRead(analogPin);
  Vx = float(5.0)*(float(RdVal)/float
    (1023));
  delay(2000);
  // Serial.print(RdVal);
  Serial.print(" Tensione letta: ");
  Serial.println(Vx);
}
```

Listing 7: Codice per la conversione analogico-digitale

Tabella 3: Misure per la calibrazione dell'ADC

V_{in} [V] ($\pm 0.5\%$)	Val_V	V_x [V]
5.00	1022	5.00
4.50	921	4.50
4.00	818	4.00
3.50	716	3.50
3.01	615	3.00
2.50	511	2.50
2.00	408	1.99
1.50	306	1.50
1.00	203	0.99
0.500	100	0.49
0.0410	6	0.03

Nel grafico di Figura 5 sono riportati i punti sperimentali e la retta $y = mx + q$ che meglio li interpola, di coefficiente angolare $m = (205.0 \pm 0.1)\,\text{V}^{-1}$ e intercetta $q = -2.0 \pm 0.3$; li si confronta con i valori previsti $m^{(\text{atteso})} = 1023/5.0\,\text{V} = 204.6\,\text{V}^{-1}$ e $q^{(\text{atteso})} = 0$. Il chi quadro è, inoltre, $\chi^2 = 2.4$, che si confronta con il numero di gradi di libertà (9), da cui si ipotizza che le incertezze sul valore di Val_V, supposte di 1 bit, siano sottostimate.

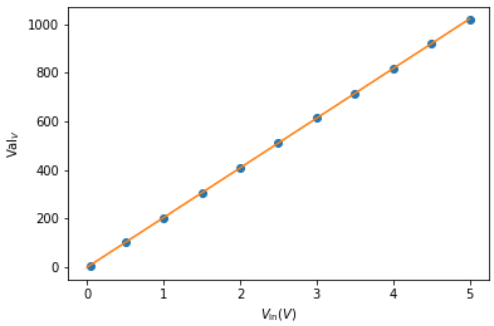

Figura 5: Val_V in funzione di V_{in}

Invertendo la relazione per il coefficiente angolare, si ottiene la costante di calibrazione k dell'ADC, per cui si avrà, nel seguito

$$V_x = \text{Val}_V \cdot k = \text{Val}_V \cdot (4.878 \pm 0.002) \times 10^{-3}\,\text{V}$$

[2]Questo è fatto per non far incidere il tempo di trasmissione seriale sulla velocità di campionamento. In ogni caso, si ha cura di non acquisire troppi punti data la memoria limitata di Arduino. Per campionamenti più numerosi, si potrebbe usare una scrittura volta per volta aumentando la velocità di trasmissione oltre i 9600 Baud.

4 Acquisizione dati dall'ADC

Si esegue un campionamento digitale di un segnale analogico, utilizzando l'ADC studiato nella sezione precedente. La frequenza del campionamento è variabile fino ad un massimo imposto dalle caratteristiche di Arduino di circa 9 kHz, dunque si sceglierà una frequenza inferiore, come da Listato seguente.

Si invia pertanto, mediante il generatore di segnali, un segnale sinusoidale di ampiezza picco-picco $(4.24 \pm 0.01)\,\text{V}$ e frequenza $(69 \pm 2)\,\text{Hz}$; si agisce inoltre sull'offset affinché l'onda abbia tensione sempre positiva. Il codice scritto per Arduino, riportato nel Listato 8, esegue il campionamento salvando i punti in un buffer, che viene scritto su schermo solo in un secondo momento[2]; quindi esso viene analizzato. Si riporta in Figura **??** l'acquisizione del segnale sinusoidale, con V_x calcolata a partire dal valore digitale mediante la costante di calibrazione ricavata in precedenza. Si riporta inoltre in Figura **??** un'istantanea dell'oscilloscopio per questa configurazione.

```
int data[100];
int analogPin = 3;
int NSamples = 100;
float FullScale=5.;
float f = 80;
int T = int (1000000./(float)(f*NSamples
    ));
int i = 0;
long unsigned t0,t1,t;

void setup() {
Serial.begin(9600);
}
void loop() {
  t0 = micros();
  for (i=0;i<NSamples;i++){
    t = micros();
    data[i] = analogRead(analogPin);
    delayMicroseconds(T);
  }
  t1 = micros();
  for(i=0;i<NSamples;i++){
```

```
    Serial.print(i);
    Serial.print(" ");
    Serial.print(t);
    Serial.print(" ");
    Serial.print(data[i]);
    Serial.print(" ");
    //Serial.print(" V = ");
    float temp = float (data[i])/1023*
    FullScale;
    Serial.println(temp);
  }
  delay(1000);
  while(1){ }
}
```

Listing 8: Codice per l'acquisizione dall'ADC

Si ripete il campionamento per un segnale triangolare, di ampiezza picco-picco (4.16 ± 0.12) V e frequenza (68 ± 2) Hz, e si riportano in Figura **??** il grafico di V_x acquisita in funzione del tempo e in Figura **??** l'istantanea dell'oscilloscopio per questa configurazione.