Curva caratteristica del diodo con Arduino

fuso@df.unipi.it; http://www.df.unipi.it/∼fuso/dida

(Dated: version 3 - Lara Palla e Francesco Fuso, 4 marzo 2015)

Questa nota discute alcuni aspetti di interesse per l'esperienza di registrazione della curva caratteristica I-V di un diodo bipolare a giunzione p-n condotta in laboratorio usando Arduino.

I. INTRODUZIONE

Anche questa esperienza è particolarmente semplice dal punto di vista concettuale. Arduino è usato come scheda di input/output in una modalità che permette di automatizzare la presa dati e di raccogliere un numero sufficiente di punti per ricostruire con buon dettaglio una curva sperimentale.

La curva in questione è la cosiddetta *curva caratteristica I-V* di un diodo a giunzione bipolare in silicio. Come (sarà) noto, un diodo a giunzione si comporta come un componente che ha una risposta decisamente *non ohmica*. Infatti la corrente (I) che attraversa la giunzione in condizioni di polarizzazione diretta non è linearmente proporzionale alla differenza di potenziale (ΔV) applicata ai terminali del componente. Secondo il modello di Shockley, la legge che descrive il comportamento è

$$I = I_0(\exp(\frac{\Delta V}{\eta V_T}) - 1) \, , \qquad (1)$$

con I_0 corrente di saturazione inversa (del valore tipico dell'ordine di 1-10 nA per i diodi usati in laboratorio, dunque molto piccola), η parametro costruttivo di valore tipico $\eta \simeq 1.5 - 2$ per gli ordinari diodi al silicio, V_T differenza di potenziale legata alla temperatura di operazione T, alla carica elementare e e alla costante di Boltzmann k_B attraverso la relazione $eV_T = k_BT$; poiché $k_BT \simeq 1/40$ eV a temperatura ambiente, si ha $V_T \simeq 26$ mV.

Tracciare la curva in questione richiede, in una semplice configurazione sperimentale, di poter disporre di un generatore di d.d.p. variabile, di misurare il valore effettivo ΔV applicato e di misurare la corrispondente intensità di corrente I che fluisce nel diodo. Tale semplice configurazione sperimentale è descritta schematicamente in Fig. 1(a) in cui si suppone implicitamente di poter trascurare gli effetti di tutte le resistenze interne (quella del generatore e del misuratore di corrente, ritenute trascurabili, e quella del misuratore di d.d.p., ritenuta così grande da non sottrarre corrente al resto del circuito).

In laboratorio non disponiamo di un generatore di d.d.p. variabile: potremmo realizzarne facilmente uno, per esempio costruendo un partitore di tensione con una resistenza variabile (potenziometro). Tuttavia per avere una ricostruzione significativa della curva $I - V$ [rappresentata per esempio in Fig. 1(b)] occorre registrare i dati su un numero relativamente elevato di punti corrispondenti a piccole variazioni del valore di ΔV, cosa non semplice dal punto di vista pratico.

La scheda Arduino, opportunamente programmata e con l'aggiunta di pochi elementi circuitali esterni, può permettere un'acquisizione *automatizzata* via computer, cioè ottenere un file contenente un numero di punti sperimentali sufficiente per le ulteriori analisi (grafico, best-fit). Gli ingredienti necessari sono:

1. ottenere una d.d.p. variabile, ovvero una rampa di tensione che evolve nel tempo con tanti piccoli gradini;
2. registrare il valore della d.d.p. applicata al diodo per ogni gradino della rampa in modo automatico;
3. registrare il valore della corrispondente intensità di corrente che circola nel diodo; poiché le porte analogiche di Arduino consentono solo misure di d.d.p., questo punto richiede di "convertire" questa intensità di corrente in una opportuna tensione, cosa che può facilmente essere realizzata a posteriori, cioè lavorando sui dati grezzi acquisiti, sfruttando la legge di Ohm.

II. LA MODALITÀ PWM DI ARDUINO

Abbiamo più volte fatto riferimento ad Arduino come a una scheda I/O (di input/output). Nell'esperienza che vogliamo svolgere ci sono sicuramente delle grandezze analogiche *in ingresso* (input) a Arduino che vogliamo leggere, cioè digitalizzare e acquisire con il computer. Queste grandezze analogiche sono quelle rappresentative di ΔV e I. Ci piacerebbe molto avere anche una grandezza analogica *in uscita* (output), cioè la d.d.p. che deve essere applicata al diodo.

Purtroppo, dal punto di vista tecnologico ottenere una grandezza analogica in uscita da una scheda I/O è più complicato che non eseguire la digitalizzazione di una grandezza analogica in ingresso. Il microcontroller di Arduino, infatti, non ha la possibilità di creare una d.d.p. di valore determinato a partire da un'istruzione digitale.

Arduino ha invece la possibilità di accendere o spegnere delle porte digitali in uscita, cioè di porle a potenziale nullo (entro l'incertezza) o massimo (entro l'incertezza), dove il massimo, come discusso altrove, si riferisce a un valore legato, nel nostro caso, alla tensione di alimentazione della scheda.

C'è un'interessantissima ulteriore opzione: alcune delle porte digitali di Arduino, quelle marcate sulla scheda con un simbolo tilde, possono operare in modalità *Pulse*

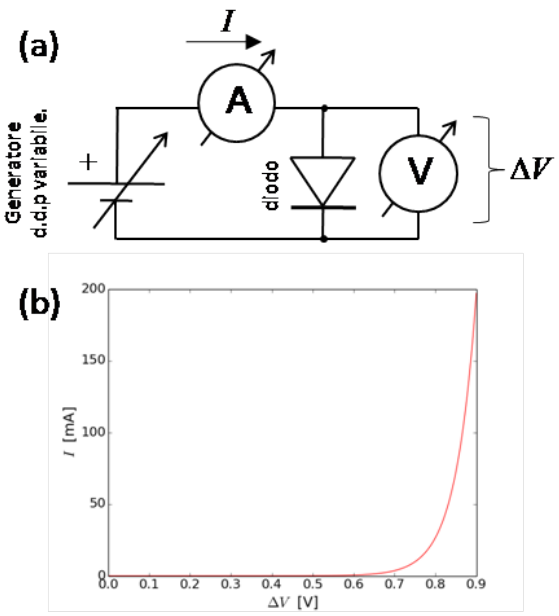

Figura 1. Schema concettuale di un'esperienza di di ricostruzione della curva caratteristica $I-V$ di un diodo (a) e curva calcolata secondo Eq. 1, supponendo $\eta = 2$, $V_T = 26$ mV e $I_0 = 3$ nA: per chiarezza è mostrato il solo ramo corrispondente a valori positivi della d.d.p. applicata, nel solo intervallo $\Delta V = 0 - 0.9$ V.

Width Modulation (PWM). Questo significa che in uscita si trova un'*onda quadra con duty cycle variabile* da zero (onda quadra "spenta", cioè nessun segnale in uscita) al massimo (onda quadra "sempre accesa", cioè segnale continuo pari al massimo, come per le altre porte digitali quando vengono accese). Il duty cycle è aggiustabile in maniera digitale agendo su un carattere, cioè su un byte: dunque sono possibili $2^8 = 256$ livelli diversi di duty cycle che possono essere scelti via software, attraverso un'opportuna istruzione dello sketch. Più propriamente, questa onda quadra si chiama *treno di impulsi*.

A. Implementazione software

Un aspetto molto importante di Arduino è che il treno di impulsi prodotto è gestito "direttamente" dal microcontroller, cioè esso non risulta da cicli inseriti nel programma dello sketch. Dunque le sue caratteristiche non risentono di eventuali latenze del microcontroller e, una volta definite attraverso l'istruzione relativa, rimangono nominalmente costanti nel tempo (entro l'incertezza).

Come controparte, la frequenza del treno di impulsi è determinata internamente e non può essere variata facilmente. Infatti essa è agganciata alla frequenza del contatore, cioè dell'orologio interno al microcontroller. Inoltre, tale frequenza è, purtroppo, piuttosto bassa, in analogia con la generale "lentezza" del microcontroller usato in Arduino. Infatti essa vale nominalmente $f = 976$ Hz per le porte digitali PWM ∼`5` e ∼`6`, e 488 Hz per le porte digitali PWM ∼`3`, ∼`9`, ∼`10`. Spulciando tra le specifiche, si vede come in realtà la frequenza possa essere modificata, in particolare aumentata per multipli di 2, ma di questa possibilità, che implica di ritoccare in qualche modo l'orologio interno del microcontroller, non faremo uso.

L'istruzione software da mettere nello sketch per creare un treno di impulsi con un certo duty cycle è molto semplice e auto-esplicativa. Chiamata `RampPin` la variabile che punta al numero della porta digitale PWM da impiegare, che è la ∼`5` nel nostro caso (e infatti nello sketch comparirà la dichiarazione `const unsigned int RampPin = 5;`), si dovrà inizializzare questa porta come uscita attraverso il comando, da mettere nel `void setup` dello sketch, `pinMode(RampPin, OUTPUT);`. A questo punto, un'onda con duty cycle corrispondente al livello `i` (con `i` intero compreso tra 0 e 255) sarà ottenuta con il semplice comando `analogWrite(RampPin, i);`.

B. Integratore

Abbiamo dunque capito come creare un treno di impulsi, cioè un'onda quadra con un certo duty cycle. Siamo ancora lontani dall'avere una d.d.p. continua variabile, però abbiamo disponibile tra le nostre conoscenze un metodo che permette di ottenere una tensione *quasi continua* a partire dal treno di impulsi. È infatti evidente che il *valore medio* di un'onda quadra con duty cycle variabile dipende dal duty cycle stesso. Sappiamo poi che l'operazione di media è (a meno di coefficienti) equivalente all'*integrazione temporale* e ci è ben noto come costruire un integratore, per esempio facendo uso di un filtro passa-basso RC.

Naturalmente il circuito RC che ci proponiamo di realizzare dovrà avere una costante tempo RC sufficientemente alta, ovvero una frequenza di taglio $f_T = 1/(2\pi RC)$ sufficientemente bassa, da permettere un'efficace integrazione temporale. In particolare ci aspettiamo che debba essere $f_T << f$; negli esempi di seguito supporremo di avere $f_T = 10$ Hz, cioè $f/f_T \sim 10^2$.

D'altra parte è ovvio che, aumentando il tempo di integrazione, cioè diminuendo la *banda passante* del sistema, dovremo introdurre degli opportuni tempi di attesa nell'operazione di Arduino, necessari affinché, dopo aver cambiato il duty cycle del treno di impulsi, il segnale in uscita dall'integratore possa raggiungere una nuova condizione stazionaria. Dunque avremo un'acquisizione automatizzata che, però, richiederà un po' di tempo per essere compiuta.

C. Serie di Fourier per il treno di impulsi

È sicuramente interessante "simulare" il comportamento di un integratore al cui ingresso abbiamo un treno di impulsi, ovvero un'onda quadra con un certo duty cycle. Possediamo già lo strumento concettuale che consente di

eseguire la simulazione: è sufficiente esprimere il treno di impulsi in serie di Fourier, cioè conoscerne i coefficienti dell'espansione di Fourier, e quindi applicare alle varie componenti, cioè alle varie armoniche, la funzione di trasferimento [attenuazione $A(f)$ e sfasamento $\Delta\phi$] del passa-basso RC.

Due osservazioni preliminari:

- nella simulazione considereremo una situazione "ideale", per cui trascureremo le resistenze interne, per esempio quella del generatore (la resistenza della porta digitale di Arduino) e di tutto il resto, cioè del circuito contenente il diodo e il collegamento alle porte analogiche di Arduino necessarie per la misura;

- il treno di impulsi in ingresso all'integratore non sarà mai alternato: infatti, anche per un'onda quadra simmetrica (duty cycle pari al 50%), la d.d.p. prodotta oscilla tra zero e il valore massimo, cioè è sempre positiva, per cui la sua media non è mai nulla.

Supponendo per semplicità un treno di impulsi rappresentato da una funzione $g(t)$ *pari* nel tempo (cioè l'istante $t = 0$ si trova a metà strada della parte alta di un impulso) e di ampiezza unitaria (valore minimo 0, valore massimo 1, in unità arbitrarie), si ha che $g(t)$ può essere rappresentata dalla seguente serie di *coseni*:

$$g(t) = \delta + \Sigma_{k=1}^{n} \frac{2}{k\pi} \sin(k\pi\delta) \cos(\omega_k t) \ , \qquad (2)$$

dove $\delta = \tau/T$ rappresenta il duty cycle variabile tra 0 e 1 (τ è la durata della parte alta dell'impulso e T è il periodo del treno di impulsi) e $\omega_k = k\omega$ è la frequenza angolare dell'armonica k-esima. Poiché talvolta il duty cycle si esprime come percentuale D (la percentuale di tempo in cui l'impulso si trova a livello alto rispetto al periodo), scriviamo l'ovvia conversione $D = \delta \times 100$. Infine, tenendo conto del fatto che in Arduino il duty cycle può essere impostato via software attraverso l'intero `i` variabile tra 0 e 255 (vedi sopra), l'altrettanto ovvia conversione che lega δ a `i` recita $\delta =$ `i`$/256$.

Come già affermato, per determinare la funzione $g_{out}(t)$ che rappresenta l'uscita dell'integratore le varie armoniche di frequenza $f = \omega_k/(2\pi)$ vanno moltiplicate per l'ampiezza del passa-basso, $A(f) = 1/\sqrt{1 + (f/f_T)^2}$, e sfasate di $\Delta\phi = \arctan(-f/f_T)$, esattamente come si fece per l'esercizio sulla "pinna di squalo". La Fig. 2 riporta un esempio dei risultati per vari valori del duty cycle (si riporta per chiarezza proprio l'istruzione da usare nello sketch di Arduino): i grafici mostrano in blu il treno di impulsi simulato e in rosso l'uscita simulata dell'integratore. Come specificato sopra, il treno di impulsi ha una frequenza $f = 976$ Hz, mentre la frequenza di taglio dell'integratore è supposta $f_T = 10$ Hz.

Il risultato simulato è in accordo con le attese: effettivamente all'uscita dell'integratore si ritrova un livello *quasi-continuo* che dipende linearmente dal duty cycle impostato. Il carattere quasi-continuo è dovuto alla frequenza di taglio dell'integratore che abbiamo supposto finita, cioè diversa da zero: esso dà luogo a una sorta di *ripple* (piccola modulazione) che non riesce a essere integrata in maniera completa. Per gli scopi della nostra esperienza questo ripple è tollerabile, specie se si tiene conto che entrambi le d.d.p. rilevanti per la misura potranno essere acquisite "in contemporanea" (nei limiti dei tempi di risposta della scheda). Tuttavia è chiaro che esso potrà contribuire in qualche forma all'incertezza delle misure.

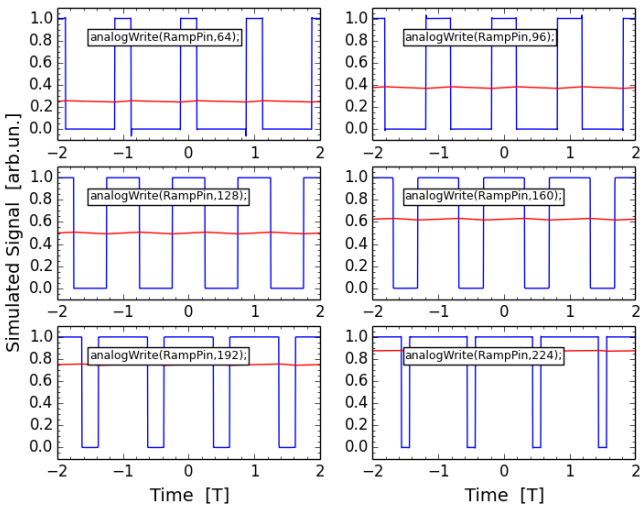

Figura 2. Esempi di simulazione di treni di impulsi con duty cycle variabile (curve blu) e di uscita dall'integratore (curve rosse). I vari grafici si riferiscono a diverse scelte del duty cycle: in figura è riportata l'istruzione software usata nello sketch di Arduino. Per il calcolo si è supposto un integratore RC con frequenza di taglio $f_T = 10$ Hz; il periodo T con cui si misurano i tempi vale $T = 1/f$, con $f = 976$ Hz.

III. LIMITAZIONE DELLA CORRENTE E SUA MISURA

Come si può facilmente intuire dall'Eq. 1 e dalla sua rappresentazione di Fig. 1(b), il diodo usato in polarizzazione diretta è in grado di far passare una corrente molto intensa, ovvero la sua resistenza effettiva (o "dinamica", nel senso che sarà chiarito in un'altra esperienza pratica), può diventare trascurabile. Ci sono diversi motivi che suggeriscono di *limitare* il valore massimo ammissibile per questa intensità di corrente: i principali sono che il diodo, al di sopra di un certo valore di corrente (~ 100 mA, per i diodi in uso in laboratorio) si distrugge per surriscaldamento; inoltre, e più importante, Arduino non è progettato per generare correnti superiori a poche decine, di mA. In altre parole, utilizzato come generatore di

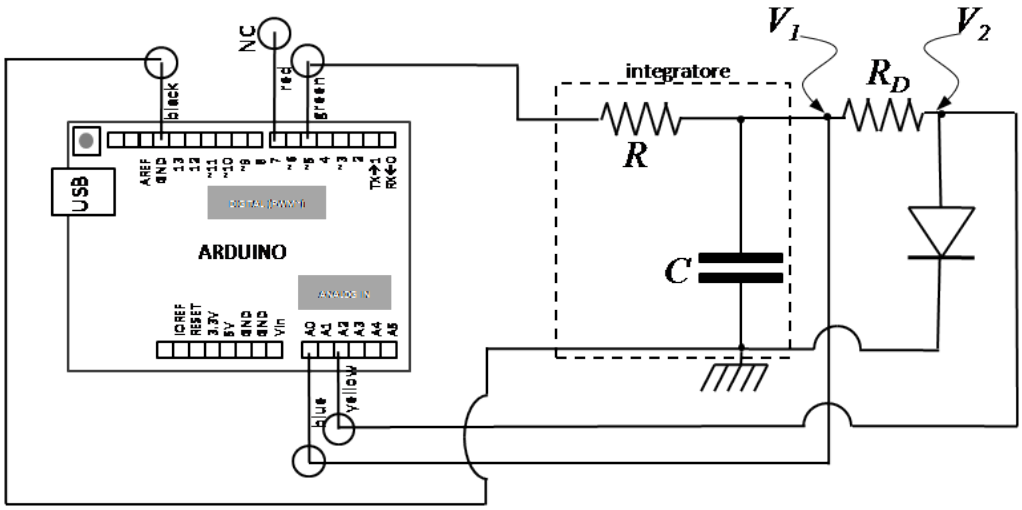

Figura 3. Schema circuitale dell'esperienza con indicate le quattro connessioni da effettuare alla scheda Arduino (NC significa non collegato).

d.d.p., Arduino ha una resistenza interna tutt'altro che trascurabile.

Per limitare la corrente massima che può fluire nel diodo è sufficiente inserire un resistore R_D in serie al diodo stesso. Supponendo trascurabile la resistenza interna del diodo nelle condizioni "peggiori" che stiamo esaminando, la corrente massima sarà limitata al valore $\Delta V/R_D$. Se vogliamo che questa corrente non superi, poniamo precuazionalmente, alcuni mA, e se teniamo conto che al massimo ΔV può valere circa 5 V nel nostro esperimento, possiamo dimensionare R_D nell'ordine dei kohm.

Tornando allo schema di Fig. 1(a), sappiamo che dobbiamo ancora occuparci della misura dell'intensità di corrente I, che va necessariamente "convertita" in una d.d.p. affinché sia digeribile per Arduino. Possiamo allo scopo sfruttare ancora R_D, cioè il resistore in serie al diodo: infatti per la legge di Ohm si ha semplicemente $I = \Delta V_{RD}/R_D$, dove ΔV_{RD} indica la caduta di potenziale sulla resistenza R_D.

Dal punto di vista pratico, dato che le porte analogiche di Arduino misurano d.d.p. relative alla linea di terra, la misura ΔV_{RD} deve essere eseguita per differenza, $\Delta V_{RD} = V_1 - V_2$, tra le d.d.p. (riferite a terra) che si trovano all'"ingresso" (V_1) e all'"uscita" (V_2) di R_D. La d.d.p. V_2 è inoltre quella che di fatto si trova ai capi del diodo, cioè corrisponde a ΔV nella simbologia di Eq. 1.

In definitiva, lo schema circuitale adottato è quello di Fig. 3: in esso si prevede l'impiego della porta digitale PWM ∼`5` (boccola verde) e le porte analogiche `A0` e `A2` (boccole blu e gialla), rispettivamente per la misura delle d.d.p. V_1 e V_2.

IV. SCRIPT DI PYTHON E SKETCH DI ARDUINO

Come per l'esperienza della carica/scarica del condensatore, anche in questo caso ci serviamo di uno script di Python per gestire la partenza delle operazioni di Arduino e per consentire il trasferimento dei dati da questo al computer tramite porta seriale USB.

Lo script sfrutta tutte le particolarità già discusse a proposito dell'esperienza sulla carica/scarica del condensatore; esso si trova in rete con il nome di `diodo_v1.py` e, ovviamente, sarà già caricato sui computer di laboratorio. Anche in questo caso lo script invia un'informazione ad Arduino attraverso un carattere (byte) che rappresenta il ritardo tra una coppia di misure e la successiva, necessario per permettere all'integratore di raggiungere condizioni stazionarie. Il carattere, che può essere impostato tra 1 e 9, rappresenta il ritardo in unità di 10 ms, e per default è regolato a 50 ms.

Il testo dello script è il seguente:

```
import serial # libreria per gestione porta seriale (USB)
import time   # libreria per temporizzazione

print('Apertura della porta seriale\n') # scrive sulla console (terminale)
ard=serial.Serial('/dev/ttyACM0',9600)  # apre la porta seriale
```

```
time.sleep(2)   # aspetta due secondi

ard.write(b'5')#intervallo (ritardo) in unita' di 10 ms <<<< questo si puo' cambiare (default messo a 50 ms

print('Start!\n') # scrive sulla console (terminale)
Directory='D:/..../....'   # nome directory dove salvare i file dati
FileName=(Directory+'.....txt') # nomina il file dati <<<< DA CAMBIARE SECONDO GUSTO

outputFile = open(FileName, "w+" ) # apre file dati in scrittura


# loop lettura dati da seriale (sono 256 righe, eventualmente da aggiustare)
for i in range (0,256):
    data = ard.readline().decode() # legge il dato e lo decodifica
    if data:
            outputFile.write(data) # scrive i dati nel file

outputFile.close() # chiude il file dei dati

ard.close() # chiude la comunicazione seriale con Arduino

print('end') # scrive sulla console (terminale)
```

Anche lo sketch di Arduino, che si può trovare in rete con il nome `diod.ino`, è molto semplice e pressoché auto-esplicativo, oltre che concettualmente e strutturalmente molto simile a quello usato per l'esperienza di carica/scarica del condensatore. Il testo è il seguente:

```
const unsigned int RampPin = 5;  //pin 5 uscita pwm per generare la rampa.
                                 // Questo pin viene collegato all'integratore
const unsigned int analogPin_uno=0;  //pin A0 per lettura V1
const unsigned int analogPin_due=2;  //pin A2 per lettura V2
unsigned int i=0;  //variabile che conta gli step durante la salita della rampa
int V1[256];  //array per memorizzare V1 (d.d.p, letta da analogPin_uno)
int V2[256];  //array per memorizzare V2 (d.d.p, letta da analogPin_due)
int delay_ms;  //variabile che contiene il ritardo tra due step successivi (in unita' di 10 ms)
int start=0;  //flag per dare inizio alla misura

//Inizializzazione
void setup()
{
  pinMode(RampPin, OUTPUT);  //pin pwm RampPin configurato come uscita
  Serial.begin(9600);  //inizializzazione della porta seriale
  Serial.flush(); // svuota il buffer della porta seriale
}

//Ciclo di istruzioni del programma
void loop()
{
    if (Serial.available() >0) // Controlla se il buffer seriale ha qualcosa
      {
        delay_ms = (Serial.read()-'0')*10; // Legge il byte e lo interpreta come ritardo (unita' 10 ms)
        Serial.flush(); // Svuota il buffer della seriale
start=1; // Pone il flag start a uno
      }
  if(!start) return // solo se il flag è a uno parte l'acquisizione
    delay(2000); // attende 2s
```

```
  for(i=0;i<256;i++)  //il valore che definisce il duty cycle dell'onda quadra è scrivibile su 8 bit
                       //cioè assume valori da 0 a 256
    {
     analogWrite(RampPin, i); //incrementa il duty cycle di uno step
     V1[i]=analogRead(analogPin_uno);   //legge il pin analogPin_uno
     V2[i]=analogRead(analogPin_due);   //legge il pin analogPin_due
     delay(delay_ms);   //aspetta il tempo impostato
    }
   for(i=0;i<256;i++)  //nuovo ciclo che scorre gli array di dati e li scrive sulla seriale
   {
     Serial.print(V1[i]);
     Serial.print(" ");
     Serial.println(V2[i]);
   }
  start=0; // Annulla il flag
  Serial.flush(); // svuota il buffer della porta seriale
}
```

V. SENSIBILITÀ E INCERTEZZE

Come per ogni esperimento in cui si adotta una procedura automatizzata di presa dati, è necessario discutere alcuni aspetti relativi alla sensibilità della misura e all'incertezza da attribuire ai dati.

Innanzitutto notiamo che la d.d.p. prodotta da Arduino seguito dall'integratore è necessariamente discreta, dato che discreta (digitale) è l'impostazione del duty cycle del treno di impulsi. Supponendo di avere un valore massimo di tensione attorno a 5 V, e tenendo conto che l'impostazione avviene con una dinamica di 8 bits, cioè sono possibili $2^8 = 256$ livelli diversi, il "quanto" della discretizzazione vale 5 V/256 $\simeq$ 20 mV.

Osserviamo poi che la ricostruzione dell'intensità di corrente I che fluisce nel diodo richiede di eseguire un calcolo in cui compare una differenza:

$$I = \frac{V_1 - V_2}{R_D} \,, \tag{3}$$

dove tutte le grandezze sono già state definite in precedenza. Dato che la dinamica delle porte analogiche è 10 bits, cioè sono possibili $2^{10} = 1024$ livelli, la lettura di V_1 e V_2 ha una sensibilità, sempre supponendo che la tensione di riferimento effettiva sia (circa) 5 V, di 5 V/1024 $\sim$ 5 mV. Ponendo per esempio $R_D = 3.3$ kohm nominali, la sensibilità nella misura di I è approssimativamente 5 mV/3.3 kohm $\simeq 2$ μA.

Aumentare la sensibilità richiede, in linea di principio, di aumentare il valore di R_D. Tuttavia, ciò comporta anche una maggiore caduta di tensione sulla resistenza stessa e, poiché il diodo è a monte, una riduzione della d.d.p. applicata al diodo. Sempre in linea di principio, c'è anche un'altra possibilità, che consiste nell'usare una V_{ref} più piccola e costruita ad hoc (è sufficiente che essa sia maggiore di 1.1 V perché Arduino possa funzionare). A questo scopo si potrebbe usare il piedino `AREF` della scheda e configurare in modo opportuno questa funzionalità nello sketch.

Discutiamo ora le incertezze delle nostre misure. Al solito, esse sono in generale causate da motivi statistici e da errori sistematici, per esempio dovuti alla calibrazione. Come ben sappiamo, Arduino restituisce dei numeri interi per le letture delle porte analogiche, per le quali è opportuno considerare come errore statistico un'unità arbitraria di digitalizzazione[1]. Se si vuole convertire l'unità arbitraria in una grandezza fisica, cioè se si vuole usare l'unità di misura fisica V, occorre tenere conto anche dell'errore sistematico (di calibrazione) nella determinazione del valore di riferimento V_{ref}. Nell'esperienza della carica/scarica del condensatore questo non era richiesto, poiché l'informazione rilevante, il tempo caratteristico, poteva essere determinata lavorando, cioè facendo un best-fit, sulle misure in unità arbitrarie. Qui, invece, vogliamo produrre un grafico dell'intensità di corrente in funzione della d.d.p. e inoltre vogliamo eseguirne un best-fit in cui i parametri, per essere confrontati con le attese, devono essere espressi in unità fisiche di corrente e tensione. Allora è necessario convertire le letture in unità fisiche e tenere in debito conto l'errore sistematico di calibrazione.

A. Procedura di calibrazione e determinazione dell'errore

Per la conversione occorre conoscere il valore della tensione di riferimento V_{ref} usata internamente dal digitalizzatore di Arduino. Questa tensione dipende dall'alimentazione, che avviene tramite porta USB. Grossolanamante, cioè trascurando altre, più sottili, possibili cause di errore, essa equivale al valore massimo di tensione che può essere fornito da una porta digitale.

Nello sketch di Arduino si può esservare come, alla fine delle operazioni, la porta digitale PWM $\sim$`5` venga a tro-

varsi al valore di duty cycle corrispondente al 100%, dato che alla fine del ciclo di misura l'indice `i` che stabilisce il duty cycle ha raggiunto il suo valore massimo (255). In queste condizioni la porta fornisce un valore *continuo* che corrisponde al valore massimo possibile, e quindi grossolanamente a V_{ref}. La misura di V_{ref} può allora essere condotta in maniera molto semplice: è sufficiente collegare il multimetro digitale tra il punto indicato del circuito con V_1 e la terra alla fine delle operazioni per poter leggere il suo valore. Si suggerisce di eseguire la misura "a vuoto", cioè dopo aver scollegato il diodo, in modo da trascurare l'eventuale caduta di tensione provocata da integratore e diodo. Infatti, supponendo come è ragionevole che le porte analogiche `A0` e `A2` abbiano resistenza di ingresso molto grande, paragonabile a quella del multimetro digitale usato come voltmetro, scollegare il diodo equivale a non far passare alcuna corrente (ovvero, una corrente trascurabile) nel resto del circuito.

Si sottolinea che *scollegare significa scollegare*: almeno una delle connessioni al diodo va rimossa e non va fatta alcuna altra modifica al circuito, in particolare non si deve rimpiazzare il diodo con alcunché (men che meno con un cortocircuito). In queste condizioni la lettura corrisponde grossolanamente al valore V_{ref} usato internamente dal microcontroller come riferimento per la digitalizzazione.

Siete anche invitati a eseguire la stessa misura con il diodo collegato, e anche a verificare quanto vale la tensione, in queste condizioni, in altri punti del circuito (per esempio quello indicato con V_2, oppure direttamente alla boccola verde connessa alla porta di uscita della scheda). Probabilmente misurerete valori più bassi di quello misurato "a vuoto": questo è molto semplicemente dovuto alla caduta di tensione sulle resistenze R e R_D in presenza di una corrente (quella che passa nel diodo). Questa caduta di tensione avviene "fuori" da Arduino, che quindi non se ne accorge e, di conseguenza, non modifica la propria tensione di riferimento. È superfluo sottolineare che l'ultima affermazione è corretta solo se possiamo supporre ideale il generatore di d.d.p. costituito da Arduino, come è lecito fare, stando alle specifiche, se la corrente richiesta è inferiore almeno a una decina di mA (tali condizioni dovrebbero essere ben verificate nell'esperimento).

Noto V_{ref} potrà essere determinato il fattore di conversione ξ (in V/arb.un.) che permette di passare dalle unità arbitrarie di digitalizzazione a quelle fisiche; esso infatti è semplicemente dato da $\xi = V_{ref}/2^{10}$, dove $2^{10} = 1024$ è il numero di livelli distinti di digitalizzazione[2]. Dato che V_{ref} è stata misurata, e quindi il valore ha un'incertezza (l'errore del multimetro), anche ξ dovrà avere un'incertezza che possiamo intepretare in maniera diretta come *incertezza di calibrazione*.

Alla fine, l'incertezza sulle misure di d.d.p. potrà essere ottenuta sommando in quadratura l'errore statistico dovuto alla digitalizzazione e quello sistematico attribuito alla calibrazione: state attenti al fatto che, di norma, l'errore sistematico prevale su quello statistico solo quando la lettura assume valori grandi (se la lettura è una unità arbitraria, l'errore statistico, che vale una unità, è pari al 100%, quindi molto superiore all'errore di calibrazione). In nessun caso l'incertezza sulle misure di d.d.p. potrà essere nulla, anche quando è nulla la corrispondente lettura.

Si sottolinea ancora che la procedura di determinazione dell'incertezza appena descritta trascura altre possibili cause di errore, per esempio quelle legate alla non linearità del digitalizzatore, ad eventuali cadute di tensione interne al microcontroller, alla scarsa stabilità di V_{ref}, all circostanza che la misura di V_1 non avviene realmente in contemporanea con quella di V_2 (e, nel frattempo, la d.d.p. in uscita dall'integratore potrebbe cambiare a causa del ripple, vedi Fig. 2).

Infine, per quanto riguarda la misura di I, l'incertezza va valutata attraverso propagazione dell'errore sull'Eq. 3. Ovviamente occorre in questo caso tenere conto anche dell'incertezza con la quale si determina il valore di R_D, misurata tipicamente con il multimetro digitale. È probabile che il valore di I risulti nullo (sotto la soglia di sensibilità) per alcune misure: anche in questo caso, l'incertezza associata non potrà in alcun modo essere nulla.

VI. ESEMPIO DI MISURA E COMMENTI

L'esperimento è stato compiuto usando un integratore realizzato con $R = 6.8$ kohm (nominali, tolleranza 5%) e $C = 2.2$ μF (nominali, tolleranza 10%). La frequenza di taglio nominale è quindi $f_T = 10$ Hz, con tolleranza dominata da quella sulla capacità. La resistenza in serie al diodo è stata misurata $R_D = (3280 \pm 36)$ ohm, dove l'incertezza tiene conto dell'errore di calibrazione del multimetro nella scala utilizzata ($\pm 0.8\%$) e del suo proprio errore di lettura (± 1 digit). La tensione di riferimento, misurata con la procedura descritta sopra, è risultata $V_{ref} = (4.89 \pm 0.03)$ V, e si è osservata una diminuzione di 10 mV (minore dell'errore) collegando il circuito costituito da integratore e diodo. La somma delle resistenze $R+R_D$ è relativamente grande, oltre 10 kohm, e quindi la richiesta di corrente alla porta di Arduino è relativamente piccola, dell'ordine di 0.5 mA; in simili condizioni la porta digitale di Arduino approssima bene un generatore di d.d.p. ideale.

Come intervallo di tempo tra un'acquisizione e la successiva è stato scelto il valore di 90 ms (la scelta comunque non è critica). La Fig. 4 riporta un esempio delle misure: nella determinazione delle barre di errore sono state seguite le istruzioni specificate nella sezione precedente. La figura mostra anche, con una linea continua rossa, il risultato di un best-fit dei dati secondo l'Eq. 1, eseguito lasciando come parametri liberi del fit I_0 e ηV_T.

Facciamo subito alcune osservazioni sui dati: intanto si nota che la spaziatura dei valori sull'asse orizzontale, che in principio ci aspetteremmo sempre pari a circa 20 mV, secondo quanto stabilito sopra, non è affatto uniforme. Per bassi valori di ΔV possiamo attribuire la circo-

stanza all'incertezza con la quale viene generata la d.d.p. e al tempo di risposta finito dell'integratore, ma certamente l'addensamento dei punti di misura a partire da circa 0.45-0.5 V richiede una spiegazione. Inoltre è anche evidente che il range spazzato nella misura è molto inferiore rispetto alle aspettative (all'uscita dell'integratore, come si può verificare facilmente usando per esempio un oscilloscopio e scollegando il diodo, si ritrova una d.d.p. costante, o quasi-costante, che arriva fino a diversi V).

Il motivo può essere facilmente intuito ricordando le caratteristiche di funzionamento del diodo bipolare a giunzione. Come evidenziato dall'andamento della curva caratteristica I-V, vedi Fig. 1(b), al di sopra di una certa V_{thr}, detta *tensione di soglia*, che è proprio tipicamente collocata tra 0.4 e 0.6 V per un ordinario diodo al silicio, la caduta di potenziale sul diodo si stabilizza grosso modo a questo valore, o appena al di sopra. Infatti in queste condizioni la resistenza della giunzione diventa praticamente trascurabile, a parte un (piccolo) contributo dovuto alla resistenza dei materiali che costituiscono l'anodo e il catodo, cioè i due elementi della giunzione stessa.

Nel nostro esperimento il diodo è alimentato da Arduino (attraverso la porta digitale PWM utilizzata) seguito da integratore e resistenza R_D. Considerato tutto assieme, il "generatore" che stiamo impiegando ha una resistenza interna tutt'altro che trascurabile. Impiegando l'approccio di Thévenin e trascurando la resistenza interna della porta di Arduino, la resistenza interna del nostro generatore è almeno pari a quella della serie $R+R_D$, cioè, nel caso analizzato, dell'ordine della decina di kohm. Per $\Delta V \gtrsim V_{thr}$ essa è dunque molto maggiore della resistenza effettiva del diodo, per cui c'è una forte caduta di potenziale sulla serie delle resistenze, mentre quella ai capi del diodo si attesta su valori prossimi, o poco superiori, a V_{thr}.

Chiarito questo importante aspetto, notiamo che il fit riproduce in maniera non troppo accurata i dati sperimentali: per alcuni punti, la linea del best-fit passa al di fuori dell'incertezza attribuita ai dati. In parte ciò può essere spiegato dalla possibilità che ci siano degli artefatti, legati tipicamente a impulsi (*spikes*) presenti nella scheda di Arduino e registrati erroneamente dalle porte analogiche.

Per tenere conto dell'incertezza non trascurabile nella "variabile indipendente" (cioè le misure di ΔV), nella procedura di best-fit si è determinata l'incertezza dei punti sperimentali propagando l'errore di lettura di ΔV su I, secondo quanto già fatto in diverse altre occasioni. Prima di eseguire il best-fit, è stato attentamente controllato che gli array contenenti le incertezze non avessero elementi nulli: la presenza di incertezze nulle impedisce a ogni software di calcolo del minimo $1chi^2$ di poter convergere.

I risultati del best-fit sono i seguenti:

$$I_0 = (3.7 \pm 0.5)\ \text{nA} \tag{4}$$

$$\eta V_T = (48.2 \pm 0.06)\ \text{mV} \tag{5}$$

$$\chi^2/\text{ndof} = 5600/254 \tag{6}$$

$$\text{Norm. cov.} = 0.97\ . \tag{7}$$

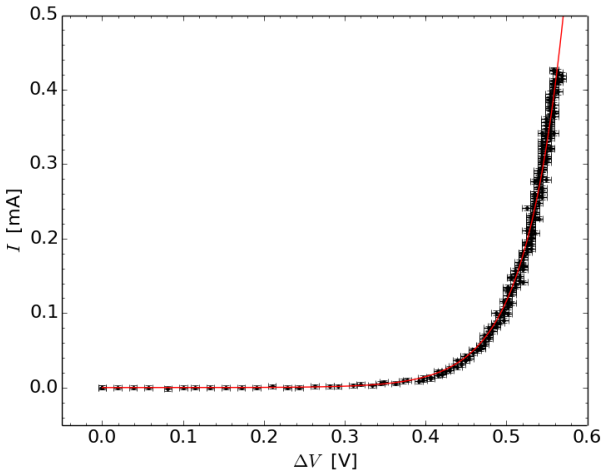

Figura 4. Esempio di misure ottenute con la scelta dei parametri e dei componenti circuitali indicati nel testo; la linea rossa continua rappresenta il best-fit condotto secondo l'Eq. 1 (commenti e risultati nel testo).

La scarso accordo tra dati e best-fit è evidenziato dal valore del $\chi^2_{rid} > 20$; questo valore si riduce sensibilmente eliminando come artefatti alcuni punti sperimentali. Inoltre è evidente che il modello, Eq. 1, fornisce una forte covarianza normalizzata tra i parametri, dovuta alla circostanza che, per molti dei punti sperimentali analizzati, l'esponenziale prevale nettamente rispetto al termine -1. In queste condizioni le variazioni di I_0 e ηV_T sono fortemente correlate tra loro (quando uno diminuisce l'altro aumenta, da cui il segno positivo della covarianza). Tuttavia i valori ottenuti dal best-fit per I_0 e ηV_T sono in ragionevole accordo con le attese.

---

[1] L'incertezza di digitalizzazione è effettivamente pari a uno solo se le misure corrispondono a conteggi diversi da circa 256, circa 512, circa 768, dove, invece, l'incertezza cresce sensibilimente. Di questa limitazione tecnica potremo eventualmente tenere conto in fase di analisi dei dati, eliminando come artefatti le misure corrispondenti.

[2] Oltre a diversi altri meccanismi interni al funzionamento del microcontroller che potrebbero influire nella determinazione di ξ, vale la pena di individuarne uno che è particolarmente semplice da intuire: per la presenza del livello zero, a cui corrisponde una d.d.p. misurata nulla, la divisione dovrebbe essere fatta per 1023, e non per 1024. Tuttavia l'errore che si compie è abbondantemente trascurabile.