Gestione di un oscilloscopio da porta seriale

Attilio Andreazza
Università degli Studi di Milano

7 maggio 2003

1 Introduzione

Una delle attività tipiche di laboratorio è l'acquisizione di dati da strumenti e la loro analisi successiva per estrarre delle informazioni su quantità fisiche con una precisione maggiore di quella che ci permette la semplice lettura dello strumento. Per questo molti strumenti di misura sono dotati di interfacce che ne permettono il controllo da computer.

L'interfaccia seriale, anche se un po' obsoleta, rimane comunque una delle più diffuse. Essa viene cosí chiamata perché permette di trasferire dati un bit alla volta usando un unico conduttore.

Il protocollo di comunicazione, noto anche come RS-232 (diventato poi EIA/TIA 232) permette di trasferire dati fino ad una velocità massima di 115200 baud (bit per secondo), anche se recenti implementazioni possono raggiungere velocità di trasferimento più elevate [1]. La comunicazione di dati ed istruzioni tra PC e strumenti solitamente avviene trasferendo stringhe di caratteri ASCII di 8-bit.

Per svolgere questo esercizio (come per svolgere in generale attività di questo genere in laboratorio), non è necessario sapere nulla dei dettagli della connessione fisica (struttura dei cavi e connettori per la porta seriale, livelli e specifiche del segnale), in quanto sarà possibile utilizzare librerie di livello più alto.

In questo esercizio simuleremo l'acquisizione di dati di un circuito RLC da un oscilloscopio simile a quelli disponibili nei laboratori del terzo anno. Impulsando il circuito con una funzione sinusoidale, si dovrà acquisire la funzione d'onda di risposta del circuito e, con una minimizzazione di χ^2 cotse ne determinerà altezza e fase. Variando la frequenza si ricostruirà la curva di risonanza del circuito e si determineranno i valori dei suoi parametri.

Nel seguito verranno indicati degli obiettiva da raggiungere. **Essi saranno indicati in grassetto.** In più saranno indicati alcuni aspetti opzionali. *Essi saranno indicati in corsivo.* Infine la sezione 6 di [2] contiene una descrizione di come effetuare una minimizzazione di χ^2 nel caso più generale di funzioni non lineare. L'implementazione di questo metodo costituisce una parte opzionale che sarà utile per risolvere il problema della sezione 7.

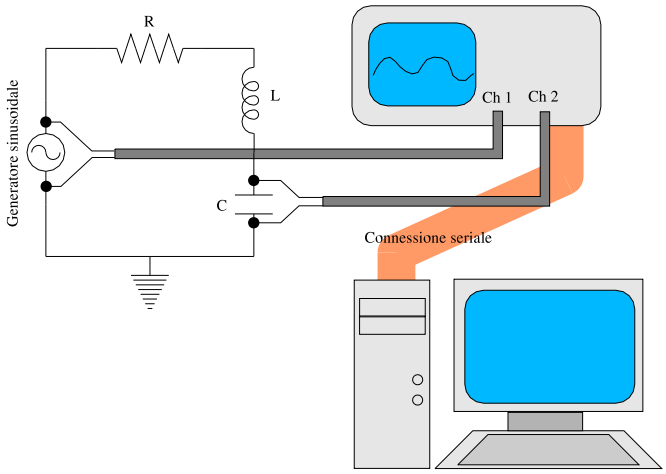

Figure 1: Schema dell'apparato sperimentale.

2 Descrizione del circuito

Il circuito simulato è descritto in figura 1. Un circuito RLC è guidato da un generatore d'onda sinusoidale con ampiezza di 1 V e frequenza regolabile. Un oscilloscopio è collegato al circuito in due punti: il canale 1 vede l'uscita del generatore d'onda, mentre il canale 2 misura la tensione ai capi del condensatore C.

L'oscilloscopio è collegato ad un computer che ne controlla l'acquisizione e viene usato per memorizare ed analizzare i dati.

Siccome non è possibile avere un apparato sperimentale completo per tutti, tutta la parte di strumentazione (generatore d'onda, circuito RLC, oscilloscopio) sono simulate da un programmino scaricabile dal sito web del Laboratorio di Calcolo 2 `http://www.mi.infn.it/~andreazz/Laboratorio2/Oscilloscopio.tar`. Le istruzioni su come compilare ed utilizzare il programma sono nel file di README.

Il programma può simulare circuiti LC, RC e RLC impulsati sia da un'onda sinusoidale che quadra di frequenza arbitraria. La scelta del tipo di circuito, funzione d'onda e frequenza viene fatta attraverso parametri dalla linea di comando che fa eseguire il programma.

Una volta in esecuzione, il programma rimane in ascolto sulla porta seriale ttyS1 (la COM2 del DOS) aspettando comandi di configurazione e rispondendo alle richieste di informazioni sul suo stato. Quando viene richiesto di acquisire una funzione d'onda, il programma fa partire un'integrazione delle equazioni del circuito con un metodo di Runge-Kutte [3] e calcola il valore delle tensioni sui canali 1 e 2 ad ogni istante. Se ad un certo punto viene soddisfatta la condizione di trigger dell'oscilloscopio, il programma memorizza 2500 punti attorno al momento del trigger e li comunica attraverso la porta seriale.

I comandi che l'emulatore comprende sono un sottoinsieme di quelli di un oscilloscopio modello TDS 1002 della Tektronix e sono elencati in sezione 3.

Sui vostri computer sono stati montati dei cavi seriali "incrociati" che collegano la porta seriale ttyS0 (COM1 del DOS) e la ttyS1, per cui, una volta che il programma di emulazione dell'apparato sperimentale è in esecuzione, la situazione che si presenta è completamente analoga ad avere un oscilloscopio esterno connesso alla porta ttyS0 del PC.

Scopo dell'esercizio è scrivere un programma che comunichi con l'oscilloscopio, consenta di configurarne i parametri e di acquisire i dati delle forme d'onda.

Una volta realizzata la funzionalità di input-output, bisognerà realizzare un programma od una macro di ROOT in grado di ricavare dai dati ottenuti i parametri di ampiezza e fase di una sinusoide acquisita dall'oscilloscopio virtuale.

Infine, variando la frequenza del generatore d'onda si ricostruirà la curva di risonanza del circuito.

3 Comandi dell'emulatore

I comandi dati all'emulatore sono delle stringhe costituite da un istruzione (spesso consistente di diversi "token" separati da ":") seguita eventualmente da un argomento. Qui di seguito sono indicati i comandi che serviranno per svolgere l'esercizio.

Se i comandi vogliono un argomento, nel caso questo sia una stringa, i valori possibili sono elencati di seguito, in lettere maiuscole e separati da una "/". Se l'argomento è un numero, questo è indicato in lettere minuscole.

Tutte le istruzioni che ammettono argomenti, hanno anche una forma interrogativa senza argomenti, consistente nella stessa parola chiave seguita da un "?" (senza nessuno spazio in mezzo), alla quale l'emulatore risponde restituendo il valore del parametro (o di un insieme dei parametri) richiesto.

ID?

Questo comando esiste solo nella forma interrogativa. Restituisce una stringa che identifica lo strumento.

HEADER ON/OFF

Questo comando mette/toglie un'intestazione alle risposte che l'emulatore restituisce. L'intestazione è utile se si vuole interrogare l'emulatore manualmente, ma conviene eliminarla se si vuole analizzare la risposta con un programma.

CH#:SCALE valore

dove # può essere 1 o 2, legge (nella forma interrogativa) o definisce la scala verticale in V/div del canale selezionato. Siccome il visore dell'oscilloscopio ha 10 divisioni, l'intervallo di accettanza dello strumento è $\pm 5 \times$ **valore**. Segnali oltre tali limiti saranno troncati. Il range possibile per **valore** è tra 2 mV/div e 5 V/div, altri valori verranno arrotondati a quelli ammissibili più vicini.

HORIZONTAL:MAIN:SCALE valore

legge o definisce la scala orizzontale dell'oscilloscopio in s/div. I dati raccolti coprono l'intervallo di tempo di $\pm 5 \times$**valore** attorno al momento del trigger. La scala dei tempi può essere selezionata tra 5 ns/div e 5 s/div a valori di 1, 2.5 e 5. Valori diversi vengono arrotondati a quelli più vicini.

TRIGGER:MAIN:EDGE:SOURCE CH1/CH2

definisce quale canale usare per il trigger.

TRIGGER:MAIN:LEVEL valore

definisce la soglia del trigger in V.

TRIGGER:MAIN:EDGE:SLOPE RISE/FALL

definisce se il trigger scatta quando il segnale attraversa la soglia in salita (RISE) o discesa (FALL).

DATA:SOURCE CH1/CH2

definisce di quale canale devono essere acquisiti i dati della funzione d'onda. Si ricordi che la sorgente del trigger e quella dei dati possono essere diverse.

DATA:WIDTH 1/2

definisce il numero di byte utilizzati per la trasmissione dei dati. Con un byte a disposizione un canale può assumere 256 valori, da -128 a +127. Con due byte a disposizione, il numero di valori possibili aumenta: da -32768 a +32767.

CURVE?

chiede all'oscilloscopio di restituire la funzione d'onda relativa al canale sorgente dei dati. La funzione d'onda è costituita da 2500 interi y_i nel range descritto nella sezione precedente e separati da ",". Il punto i-esimo della funzione d'onda ha valori di tempo e tensione, dati dalla relazione:

$$\begin{aligned} t_i &= X_0 + X_{\text{incr}} \times i, \\ V_i &= Y_0 + Y_{\text{mult}} \times \left(y_i - Y_{\text{off}}\right). \end{aligned} \qquad i = 1, 2, \ldots, 2500 \qquad (1)$$

Per definizione il trigger è avvenuto al tempo $t = 0$.

WFMPRE:XZERO? WFMPRE:XINCR? WFMPRE:YZERO? WFMPRE:YMULT? WFMPRE:YOFF?

sono i comandi interrogativi che permettoni di ricavare le costanti necessarie in (1).

EXIT

Questo non è un vero comando dell'oscilloscopio, ma può essere utile per il nostro emulatore: ordina al programma di terminare l'esecuzione.

4 Parlare con l'emulatore

Dopo aver compilato e fatto partire l'emulatore, per verificare che funzioni correttamente bisogna riuscire a comunicare attraverso l'interfaccia seriale.

Qualunque ambiente di sviluppo di codice per acquisizione dati fornisce delle interfacce di alto livello per la comunicazione con i protocolli più comuni. Questo laboratorio non fa eccezione ed insieme ai file distribuiti con l'emulatore c'è un file `serial.c` (ed il relativo `serial.h`) che contiene le funzioni necessarie per dialogare utilizzando la porta seriale: viene definita una struttura `serialport` che contiene tutti i parametri necessari per accedere ad una porta ed inizializzarla correttamente e poi vengono implementate delle funzioni che operano su puntatori `serialport*` per aprire, chiudere, leggere da e scrivere su porte seriali.

Tali funzioni sono:

- `serialport* OpenSerial(const char* deviceName)`
  che inizializza una porta seriale. Come argomento vuole una stringa con il nome del file (in Linux tutte le device sono dei file) corrispondente alla porta seriale. "`/dev/ttyS0`" e "`/dev/ttyS1`" sono i nomi possibili. Il valore di ritorno è un puntatore ad una struttura `serialport` con i dati della porta cosí aperta. Se non si riesce ad aprire la porta, il puntatore restituito è nullo.

- `void CloseSerial(serialport* port)`
  chiude correttamente la porta.

- `int WriteSerial(serialport* port, const char* buffer)`
  invia una stringa alla porta seriale. il valore di ritorno è la dimensione della stringa scritta sulla porta seriale.

- `int ReadSerial(serialport* port, char* buffer)`
  legge dalla porta seriale e mette il risultato nella stringa `buffer`. Tale stringa deve essere già dimensionata nel programma che usa la funzione e deve essere di dimensione `MAXBUFFERSIZE`. Il valore di ritorno è la dimensione della stringa letta, o 0 se non viene letto niente dalla porta seriale durante `SERIAL_TIMEOUT` secondi. Entrambi i valori di `MAXBUFFERSIZE` e `SERIAL_TIMEOUT` sono definiti in `serial.h`.

Per poter usare queste funzioni non è necessario vederne l'implementazione in `serial.c`, ma dovrebbe essere sufficiente l'informazione presente nell'header file.

Realizzare un programma che permetta di comunicare interattivamente con l'emulatore, seguendo lo schema:

1. Aprire la porta COM1 (`/dev/ttyS0`).
2. Leggere una riga di comando da terminale.
3. Inviare la stringa alla porta seriale.
4. Provare a leggere indietro una risposta dalla porta seriale: se si ottiene una risposta dallo strumento, stamparla su terminale, altrimenti mandare un avviso che non si è osservato niente durante il perido di attesa `SERIAL_TIMEOUT`.
5. ripetere i passi da 2 a 4 fino a quando non viene digitato un `Ctrl-d`.
6. Chiudere la porta seriale ed uscire dal programma.

Una complicazione rispetto agli altri esercizi che abbiamo fatto è che non sappiamo a priori come è fatta la linea da leggere: potrebbere essere un comando solo, comando più argomento, o addirittura più comandi concatenati. Quindi, anziché usare uno scanf, conviene usare la funzione delle standard libraries
`int getline(char **line, int *n, FILE *input)`
che appunto legge un'intera linea e la mette in una stringa allocata dinamicamente con un `malloc`. All'uscita della chiamata, `*line` è un puntatore all'inizio della zona di memoria allocata con `malloc`, `n` contiene la dimensione di questa zona di memoria ed il valore di ritorno è il numero di caratteri che effettivamente compongono la stringa letta, o -1 in caso di errore. Bisogna poi anche ricordare che lo standard input non è altro che un `FILE*` un po' speciale: si chiama `stdin` ed è dichiarato in `stdio.h`.

Quello che segue è un programma di esempio che usa getline per leggere una riga e ristamparla sullo schermo: dovrebbe essere sufficiente adattarlo un po' per aggiungere la parte di comunicazione attraverso le funzioni in `serial.h`:

```
#include <stdlib.h>
#include <stdio.h>
int main() {
  char *riga=0;
  int dim=0;
  int letti;
  while ( (letti=getline(&riga,&dim,stdin))!=-1 )
    printf("Letti %d caratteri:\n==%s==\n",letti,riga);
  return 0;
}
```

Utilizzare il programma cosí costruito per verificare che si è in grado di ricevere l'informazione dall'oscilloscopio e verificare il formato con cui vengono restituiti i dati. Quando si devono leggere i dati di una forma d'onda bisogna avere un attimo di pazienza: **quanto tempo ci vuole per leggere una stringa di 10000 caratteri a 9600 baud (bit per secondo), che è la velocità di trasmissione dell'oscilloscopio?**

Siccome la risposta del circuito ad una forzante è calcolata tramite l'integrazione di un'equazione differenziale di una funzione d'onda, se si chiede di integrare su

un tempo troppo lungo, potrebbe essere che il vostro programma vada in time out mentre l'emulatore sta facendo i suoi calcoli. Siccome la velocità di calcolo dipende molto dalla CPU e dalla configurazione del PC, **provate a vedere qual è la lunghezza di tempo massima per cui si riesce ad integrare** (per acquisire una funzione d'onda, l'emulatore deve integrare per almeno 10 volte il valore di s/div dato con `HORIZONTAL:MAIN:SCALE`, quindi basterà variare questo parametro).

5 Acquisizione e grafico dei dati

Una volta verificato che si riesce a parlare con l'emulatore, bisogna provare a leggere e fare un grafico della funzione d'onda raccolta. Siccome l'oscilloscopio restituisce i dati come interi, per fare un grafico questi devono essere convertiti secondo l'equazione (1). Per fare questa operazione possiamo usare il seguente approccio:

1. fare un programma che configuri l'oscilloscopio, acquisisca la funzione d'onda e scriva su terminale le costanti di conversione ed i valori della funzione d'onda;
2. usare la ridirezione dell'output per scrivere questi dati su di un file;
3. scrivere una macro di ROOT che definisca una funzione in grado di leggere tale file e restituire un grafico della funzione d'onda che poi possiamo disegnare.

Realizzare un programma che configuri l'oscilloscopio ed acquisisca una funzione d'onda secondo questo schema:

1. aprire una porta seriale per comunicare con l'oscilloscopio;
2. definire il trigger: livello, salita o discesa, e canale sorgente;
3. definire la scale dei tempi, quella verticale ed il canale da osservare;
4. interrogare l'oscilloscopio per recuperare i valori di X_0, X_{incr}, Y_0, Y_{mult} and Y_{off};
5. scrivere queste costanti su terminale;
6. acquisire una funzione d'onda;
7. scrivere il risultato su terminale.

Siccome bisognerà misurare la fase relativa tra la forzante e la risposta del circuito, quasi sempre sarà conveniente triggerare sul canale 1 al passaggio dello zero in salita (fissando la fase della forzante a zero) e leggere il canale 2. Spesso invece si cambieranno i valori delle scale quindi sarebbe comodo passare da linea di comando i valori delle scale dei tempi e delle tensioni.

Scrivere il programma in modo che accetti i valori delle scale dei tempi e delle tensioni da riga di comando. Per fare questo sarà necessario dichiarare la funzione `main` *nella sua forma completa:*
`int main(int argc, char* argv[])`
ed a questo punto gli argomenti da riga di comando saranno contenuti nelle stringhe `argv[1]` *e* `argv[2]`*, che possono essere lette tramite* `sscanf`*. Bisognerà poi usare* `sprintf` *per inserirlo nella stringa da inviare alla porta seriale tramite* `WriteSerial`*.*

Realizzare una macro di ROOT che definisca una funzione
`TGraph* LeggiOnda( char* inputfile, char* outputfile)`
che legga dal file `inputfile` **le costanti di conversione, ricostruisca la funzione d'onda, stampi i punti cosí calcolati nel file** `outputfile` **e ne faccia il grafico.**

La parte concettualmente complicata di questo esercizio è il fatto che non è possibile per una funzione restituire un oggetto `TGraph`, ma può solo restituire un `TGraph*`. È quindi necessario per la funzione allocare dinamicamente l'oggetto. Questa operazione deve essere fatta utilizzando la sintassi del C++ che è diversa da quella del C: la funzione `malloc` viene sostituita dall'operatore `new` e `LeggiOnda` dovrà quindi contenere i seguenti frammenti di codice:

```
TGraph* LeggiOnda( char* inputfile, char* outputfile) {
/* Dichiarazione di un puntatore ad oggetti TGraph */
TGraph* grafico;
⋮
/* Allocazione dinamica dell'oggetto TGraph */
grafico = new TGraph;
⋮
/* lettura del file di input, inserimento dei punti
nel grafico, scrittura sul file di output */
⋮
/* Al termine della macro restituiamo un puntatore all'oggetto costruito
*/
return grafico;
}
```

In teoria il grafico che viene ricostruito dovrebbe essere un'onda sinusoidale.

Provate a far partire l'emulatore del circuito RLC con una frequenza di 10 kHz ed acquisire una funzione d'onda con scala orizzontale 0.1 ms/div. Ottenete davvero un'onda sinusoidale? Se no, perché? Che soluzione possiamo adottare?

6 Determinazione di ampiezza e fase della risposta del circuito

Il motivo per cui abbiamo fatto in modo che LeggiOnda scrivesse su file i risultati è che vogliamo poi processare ulteriormente questi dati per determinare

l'ampiezza e la fase della sinusoide di risposta. Infatti, applicando il metodo delle impedenze complesse, sappiamo che ad una forzante

$$V_{\text{in}} = V_0 \sin \omega t \tag{2}$$

il circuito in figura 1 risponde con un segnale sinusoidale

$$V_{\text{out}} = A(\omega) V_0 \sin\left(\omega t + \varphi(\omega)\right) \tag{3}$$

dove

$$\begin{array}{rcl} A(\omega) & = & \frac{1}{\sqrt{(1-\omega^2 LC)^2 + (\omega RC)^2}}, \\ \tan\varphi(\omega) & = & -\frac{\omega RC}{1-\omega^2 LC}. \end{array} \tag{4}$$

Facciamo a questo punto due brevi osservazioni: non bisogna confondere la pulsazione ω con la frequenza $f = \omega/2\pi$. Inoltre l'equazione (4) per $\phi(\omega)$ di fatto riduce la fase nell'intervallo $(-\pi/2, \pi/2)$, mentre il suo vero intervallo di validità è su tutto 2π. Quindi utilizzare le relazioni (4) con attenzione, per tenere conto del quadrante esatto in cui si trova φ. Per fortuna che in C, come in FORTRAN, esiste la funzione `atan2(y,x)` che resituisce l'angolo polare del punto (x, y).

Siccome il valore di $V_{\text{out}}(t)$ misurato dall'oscilloscopio sarà affetto da errori di misura, possiamo dare la migliore stima di A e φ con una minimizzazione di χ^2, ovvero trovando la coppia di parametri che rende minima la somma dei quadrati delle differenze tra le quantità misurate e la funzione attesa, pesando ogni punto con il suo errore:

$$\chi^2 = \sum_i \frac{\left(V_i - A V_0 \sin\left(\omega t_i + \varphi\right)\right)^2}{\sigma_V^2}.$$

Scritto in questo modo il χ^2non è lineare nei parametri da determinare, ma può essere linearizzato osservando che l'eq. (3) può anche essere scritta

$$V_{\text{out}}(t) = V_0 \left(A \cos\varphi \sin\omega t + A \sin\varphi \cos\omega t\right) \equiv V_0 \left(a_1 \sin\omega t + a_2 \cos\omega t\right). \tag{5}$$

E quindi il χ^2 si può riscrivere come

$$\chi^2 = \sum_i \frac{\left(V_i - V_0 a_1 \sin\left(\omega t_i\right) - V_0 a_2 \cos\left(\omega t_i\right)\right)^2}{\sigma_V^2}$$

che si può esprimere in forma matriciale come

$$\chi^2 = (\vec{V} - S\vec{\alpha})^T W (\vec{V} - S\vec{\alpha})$$

dove

$$\vec{V} = \begin{pmatrix} V_1 \\ V_2 \\ \vdots \\ V_N \end{pmatrix}, \; \vec{\alpha} = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}, \; S = V_0 \begin{pmatrix} \sin\omega t_1 & \cos\omega t_1 \\ \sin\omega t_2 & \cos\omega t_2 \\ \vdots & \vdots \\ \sin\omega t_N & \cos\omega t_N \end{pmatrix},$$

$$W = \begin{pmatrix} 1/\sigma_V^2 & 0 & \cdots & 0 \\ 0 & 1/\sigma_V^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1/\sigma_V^2 \end{pmatrix}.$$

Si deriva facilmente che il minimo si ha per il vettore di parametri

$$\vec{a} = \left(S^T W S\right)^{-1} S^T W \vec{V}$$

L'altra proprietà interessante è che la matrice di covarianza dei parametri è data da:

$$C_{a_1,a_2} = \begin{pmatrix} \sigma_{a_1}^2 & \rho\sigma_{a_1}\sigma_{a_2} \\ \rho\sigma_{a_1}\sigma_{a_2} & \sigma_{a_2}^2 \end{pmatrix} = \left(S^T W S\right)^{-1}$$

dove ρ è il coefficiente di correlazione tra i parametri a_1 e a_2.

Siccome in C non è comodo passare matrici, si consiglia di effettuare direttamente il calcolo del vettore di 2 elementi $S^T W \vec{V}$ e della matrice $S^T W S$ che, essendo 2×2, è facilmente invertibile, anziché calcolare separatamente tutte le matrici e fare le moltiplicazioni alla fine.

Una volta ricavati i valori dei parametri, questi si possono convertire nelle quantità fisiche che ci interessanto A e φ usando le relazioni

$$\begin{aligned} A &= \sqrt{a_1^2 + a_2^2}, \\ \varphi &= \arctan \tfrac{a_1}{a_2}, \end{aligned}$$

$$C_{A,\varphi} = \begin{pmatrix} \frac{\partial A}{\partial a_1} & \frac{\partial A}{\partial a_2} \\ \frac{\partial \varphi}{\partial a_1} & \frac{\partial \varphi}{\partial a_2} \end{pmatrix} C_{a_1,a_2} \begin{pmatrix} \frac{\partial A}{\partial a_1} & \frac{\partial A}{\partial a_2} \\ \frac{\partial \varphi}{\partial a_1} & \frac{\partial \varphi}{\partial a_2} \end{pmatrix}^T.$$

Passiamo ora a discutere praticamente come arrivare a queste determinazioni.

Determinare l'errore di misura dell'oscilloscopio confrontando i valori di una funzione d'onda misurata con una funzione d'onda nota (ad esempio, siccome sappiamo esattamente come è fatta la forzante, possiamo usare la lettura del canale 1 per fare questo confronto).

Implementare la minimizzazione di χ^2. Verificare che funzioni con una forma d'onda nota (anche in questo caso si può utilizzare la forzante) **e che gli errori calcolati siano ragionevoli**. Per farlo si suggerisce di definire una funzione di ROOT che abbia come argomenti il nome di un file da leggere e la frequenza da usare.

In particolare si faccia attenzione alla fase: siccome l'oscilloscopio ha frequenze di campionamento più o meno fisse, il momento del trigger, che corrisponde a $t = 0$ usando la (1) in realtà è noto con una precisione non superiore a $1/\sqrt{12} f_{\text{sample}}$, dove f_{sample} è la frequenza di campionamento. Dato che si accumulano 2500 punti in 10 divisioni, f_{sample} è uguale a 250 diviso la larghezza di una divisione. **Come si traduce questo errore sul tempo del trigger in un errore sulla fase?**

Inoltre si ricorda che in (1) i va da 1 a 2500 e non, come sembrerebbe ovvio in C, da 0 a 2499.

Alternativamente alla procedura in due passi con χ^2 lineare descritta in questa sezione, si può implemetare un χ^2 non lineare, seguendo quando descritto nella sezione 6 della simulazione dello spettrometro a prisma[2].

Fare un grafico della sinusoide determinata e sovrapporla alla funzione d'onda misurata dall'oscilloscopio, controllare la qualità della minimizzazione facendo un istogramma delle differenze tra i valori di tensione misurati e quelli della sinusoide ricostruita.

7 Determinazione della frequenza di risonanza

Una volta che il meccanismo di analisi dati è stato impostato correttamente, si possono prendere dati per diversi valori della frequenza della forzante e verificare che la risposta segua la curva di risonanza data in (4).

Per automatizzare questa procedura, si può sfruttare il fatto che ROOT è in grado di passare comandi al sistema operativo tramite il metodo `Exec(char * command)` applicato all'oggetto `gSystem` che è sempre definito e rappresenta il sistema operativo sotto cui ROOT sta girando.

In pratica si tratterà di modificare il programma della sezione 5 in modo che faccia terminare l'emulatore al termine dell'acquisizione dell'onda ed effettuare un ciclo in cui per ogni frequenza si usa `gSystem->Exec(...)` per far partire l'emulatore ed un altro per far partire l'acquisizione dati. Si tratta principalmente di un lavoro di `sprintf` per scrivere le stringhe di comandi correttamente.

In tal caso, a causa dei vari time out nella lettura/scrittura da seriale, nel programma può valere la pena di introdurre a mando dei ritardi tra l'invio del comando e la risposta, per dare al tempo all'emulatore di terminare correttamente le operazioni. Per sospendere per un tempo determinato l'esecuzione di un programma si può usare la funzione
`unsigned int sleep(unsigned int secondi)`
nelle librerie `unistd.h`.

Determinare i parametri che governano la risposta del circuito: la frequanza di oscillazione intrinseca $f_0 = 1/2\pi\sqrt{LC}$ e il tempo caratteristico $\tau = RC$.

Si noti che, a meno di *fare un fit non lineare all'equazione per* $A(\omega)$, risulta molto più semplice utilizzare il grafico di $\varphi(\omega)$, in quanto τ può venire ricavato dalla pendenza della fase a basse frequenze, mentre f_0 si può determinare dal punto in cui $\varphi(\omega)$ attraversa $\pi/2$.

In qualunque modo si determinino questi parametri, si cerchi di dare una stima degli errori ad essi associati.

Provare a vedere la risposta del circuito ad un'onda quadra e verificare se i parametri determinati sono effettivamente corretti.

8 Conclusione

L'utilizzo di un oscilloscopio è una delle operazioni più frequenti che un fisico sperimentale si trova a fare. Oltre all'osservazione diretta delle grandezze sullo schermo, la capacità di poter automatizzare l'acquisizione e l'analisi di funzioni d'onda permette di ottenere informazioni quantitative e di ridurre il tempo impiegato per operazioni ripetitive.

La maggior parte dei protocolli di comunicazione con strumenti (i cui più diffusi sono quelle seriale RS-232 e quello GPIB) si basano sulla trasmissione di comandi e dati tramite stringhe. In questo esercizio vengono affrontate in un sistema simulato le problematiche di queste operazione, offrendo la possibilità

di utilizzare metodologie di analisi dati generiche (minimizzazione di χ^2) che vengono ampiamente sfruttate in tutti i campi della fisica.

Si è voluto inoltre mettere in mostra la procedura che si effettua in qualunque laboratorio nella messa a punto del software: dapprima analisi interattiva, poi controllo del funzionamento e delle prestazioni dello strumento ed infine l'acquisizione di grosse quantità di dati ed il loro utilizzo per la misura di proprietà importanti dell'oggetto di studio.

Lo svolgimento dell'esercizio dovrebbe aver illustrato efficacemente come si possono integrare tra loro diversi strumenti (strumenti di misura, programmi in C, tool per l'analisi grafica dei dati) per svolgere un compito complesso.

References

[1] Per dettagli sul funzionamento della porta seriale, si veda D.S. Lawyer, *Serial-HOWTO*, accessibile da `http://www.linuxdoc.org`.

[2] A. Andreazza, *Simulazione dell'esperienza dello spettrometro a prisma*, Università di Milano, 15 novembre 2003, accessibile da `http://www.mi.infn.it/~andreazz/Laboratorio2/Simulazione`.

[3] L'algoritmo utilizzato è stato implementato nel file RungeKutta.c ed è descritto in Numerical Recipes in C, cap 16.3

[4] Le funzioni sono adattamenti degli esempi per I/O non canonico di Gary Frerking, *Serial Programming HOWTO*, accessibile da `http://www.linuxdoc.org`.