ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA

CORSO DI LAUREA IN INGEGNERIA
ELETTRONICA, INFORMATICA E TELECOMUNICAZIONI

TITOLO DELL' ELABORATO

SVILUPPO DI UNA PIATTAFORMA SOFTWARE PER ACQUISIZIONE DATI DA UN SISTEMA DI MISURA DI IMPEDENZE

Tesi in:
ELETTRONICA DEI SISTEMI DIGITALI

Relatore:
Prof. Aldo Romani

Candidato:
Luca Gessi

Correlatori:
Dr. Marco Crescentini

Sessione I
Anno accademico 2014-5

alla mia famiglia

Indice

Introduzione

Il progetto è il proseguimento di una tesi di laurea[1] in cui si è studiato un metodo di analisi della salute della pelle non invasivo. A tale scopo si è approfondito il tema della spettroscopia d'impedenza ed è stato realizzato un sistema per la loro misura. Questo sistema prevede l'utilizzo di una parte analogica formata da un generatore di segnale sinusoidale a frequenza variabile e la circuiteria per estrarre i valori efficaci di tensione e corrente e il valore di fase. La parte digitale, invece, condiziona il segnale del blocco analogico agendo su trimmer digitali, acquisisce i dati e li trasmette tramite la UART.
Lo strumento effettuava le misurazioni ed inviava continuamente i dati grezzi al computer, tramite un convertitore UART/USB, risultando poco versatile.
L'obiettivo del progetto è realizzare una piattaforma software che comunichi con l'hardware, permettendo la configurazione dello strumento e la manipolazione dei dati grezzi ricevuti, svincolando quindi l'utente dai problemi di basso livello.
Si è studiato un protocollo di comunicazione che permette la trasmissione di maggiore informazione e sono stati scelti dei comandi mnemonici che lo strumento possa facilmente interpretare.
Il progetto prevede quindi una prima fase di modifica del vecchio firmware, in modo che il microcontrollore possa leggere e comprendere i dati ricevuti tramite la UART.
Nella seconda fase si è sviluppato il software utilizzando il linguaggio di programmazione Java. Lo sviluppo comprende lo studio delle librerie grafiche offerte da JavaFX (soprattutto per la rappresentazione dei dati grezzi in grafici a due assi), di un metodo di gestione e salvataggio su disco delle impostazioni del sistema, della comunicazione seriale e infine del sistema software nella sua completezza.
Alcune prove sperimentali sono infine state svolte per verificare la funzionalità dei due sistemi, firmware e software.

1 Vedi tesi di Giulia Luciani “Strumento per la diagnosi di carcinoma BCC basato su spettroscopia empedenziometrica” 2013-2014

Capitolo 1: il firmware

1.1 L'obiettivo

Il sistema di misura di impedenze
L'obiettivo del progetto originale è lo studio di un metodo non invasivo per l'analisi dello stato di salute della pelle. A tale scopo è stata approfondita la spettroscopia d'impedenza.
Il progetto comprende una parte analogica e una digitale.

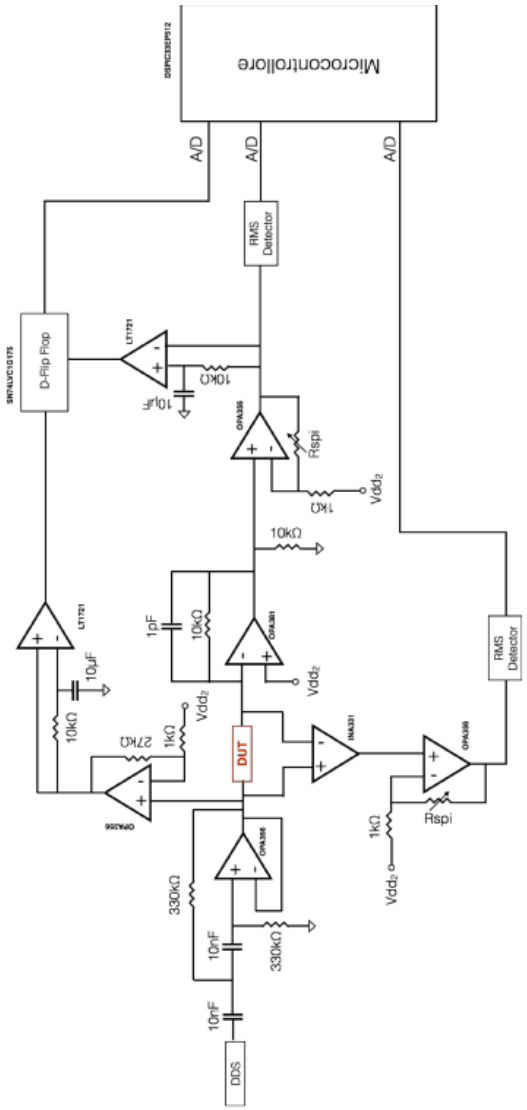


Figura 1[2]. Schema sistema di misura di impedenze

Lo strumento misura valore efficace di tensione e corrente e sfasamento fra i due segnali, in modo da ricostruire il modulo e la fase dell'impedenza.
La figura 1 mostra lo schematico del sistema di misura. La catena analogica si ripete due volte in modo da poter effettuare misurazioni doppie e confrontare i valori di pelle sana e pelle a rischio.
Per generare il segnale sinusoidale si utilizza un DDS. Questo è programmato dal microcontrollore e genera sinusoidi a frequenze comprese fra 1Khz e 1Mhz. Un filtro passa banda è posto in cascata al DDS per pulire il segnale. Il segnale infine cade sul DUT (device under test), cioè l'impedenza da misurare.
L'intervallo di impedenze misurabili varia fra 1Kohm e 1Mohm. Questo determina un segnale all'ingresso degli RMS detector variabile, quindi, per mantenerlo entro un limite che permetta una corretta lettura, si utilizzano stadi amplificatori con trimmer digitali opportunamente programmati dal microcontrollore.
La misura della corrente è possibile grazie ad un convertitore corrente-tensione. La tensione ai capi del DUT invece è prelevata tramite un amplificatore da strumentazione. Infine in entrambi i casi (corrente e tensione) i segnali ottenuti entrano nella cascata di amplificatore a guadagno variabile e RMS detector.
La misura di fase utilizza invece due comparatori e un flip flop. I comparatori permettono di generare, partendo dai segnali sinusoidali, onde quadre alla stessa frequenza. Il flip flop, di tipo D, presenta un ingresso D, un'uscita Q, un CLOCK e un CLEAR. Quando CLEAR è basso il dato è trasferito dall'ingresso all'uscita allo scandire del clock. Ponendo D a valore logico alto fisso e collegando le uscite dei comparatori ai pin CLOCK e CLEAR, il segnale all'uscita Q si presenta alto quando lo sono contemporaneamente CLOCK e CLEAR. Il duty cycle dell'onda quadra ottenuta mi quantifica lo sfasamento. Un filtro passa basso estrae poi il valore continuo.
Infine i tre rami presentano in uscita valori continui di tensione convertibili dagli ADC del microcontrollore.
Per il corretto calcolo dei valori bisogna considerare i guadagni variabili degli stadi amplificatori.

Il firmware

Il progetto attuale comprende una modifica al firmware dell'impedenziometro.

2 Figura della tesi di Giulia Luciani "Strumento per la diagnosi di carcinoma BCC basato su spettroscopia empedenziometrica" 2013-2014

Lo strumento esegue misurazioni su tre decadi di frequenze (1Khz → 10Khz, 10Khz → 100Khz, 100Khz → 1Mhz). Il firmware originale acquisiva lo stesso numero di campioni per decade e inviava continuamente tramite la UART successive misurazioni separate dai caratteri "**".
Un esempio di misurazione poteva essere:

```
23474 23455
23434 23465
23453 23435
23454 23456
24354 23655
23354 23355
**
```

Ogni campione è lungo 5 caratteri decimali, il primo rappresenta il valore efficace di corrente, il secondo di tensione e ogni riga è separata dai caratteri line feed e carriage return (rispettivamente 0xA e 0xD nel codice ASCII). I campioni sono da leggere in mV. Questo blocco si ripeteva continuamente risultando un sistema poco versatile.
L'obiettivo della modifica è rendere il sistema in grado di eseguire comandi ricevuti tramite UART.

1.2 La realizzazione

1.2.1 Il protocollo di comunicazione

Il protocollo di comunicazione scelto prevede una estensione del blocco dati per trasmettere il valore di fase, valore efficace di tensione e corrente, valore di un trimmer digitale, frequenza e dati relativi ad una seconda misurazione. Un generico blocco dati potrebbe presentarsi:

```
11111 22222 33333 44444 55555 #  22222 33333 44444 55555
11111 22222 33333 44444 55555 #  22222 33333 44444 55555
11111 22222 33333 44444 55555 #  22222 33333 44444 55555
11111 22222 33333 44444 55555 #  22222 33333 44444 55555
11111 22222 33333 44444 55555 #  22222 33333 44444 55555
11111 22222 33333 44444 55555 #  22222 33333 44444 55555
**
```

Ogni riga è separata dai caratteri finali “\r\n” (rispettivamente line feed e carriage return) e rappresenta i campioni misurati alla specifica frequenza, indicata dalle prime 5 cifre (nell'esempio la sequenza di 1).
Le sequenze di 2 e 3 rappresentano le misure rispettivamente di valore efficace di corrente e di tensione. Le sequenze di 4 rappresentano il valore di fase mentre quelle di 5 il valore del trimmer digitale. Il carattere '#', insieme ovviamente agli spazi, delimita il confine fra i campioni relativi alla prima e seconda misura (rispettivamente sinistra e destra). Il discorso fatto sui valori della prima misura vale anche per la seconda ad eccezione della frequenza che, essendo comune, viene trasmessa una sola volta. Infine i caratteri “**” indicano la fine della misurazione.

Per la implementazione dei comandi da inviare allo strumento sono state scelte parole mnemoniche, facili da ricordare e che permettano la verifica del sistema attraverso una semplice finestra seriale. I comandi scelti sono:

1. “DECADES=101\r\n”: comando per abilitare/disabilitare la misura delle singole decadi. Le 3 cifre che seguono l'uguale indicano se la relativa decade sia da attivare (1) o no (0);
2. “SAMPLES=0213\r\n”: comando per impostare il numero di campioni da acquisire per decade. Il numero composto da 4 cifre decimali indica i campioni;
3. “START\r\n”: comando usato per iniziare la misurazione.
4. “DOUBLEMEASURE=0\r\n”: comando utilizzato per abilitare/disabilitare la misurazione doppia.

Attualmente solamente DOUBLEMEASURE non è implementato dal firmware.

1.2.2 L'abilitazione delle periferiche

Lo strumento monta il microcontrollore dsPIC33EP512GM604 della Microchip e si è utilizzato l'IDE MPLAB della stessa casa insieme al programmatore Pickit3.
Il microcontrollore era già programmato per utilizzare la UART lato trasmettitore quindi per l' attivazione del lato ricevitore è stato sufficiente

assicurarsi che i bit *UARTEN* del registro *U1MODE* fossero a valore 0[3] .
Inoltre è tornato utile attivare un timer di sistema (timer4) per realizzare una funzione di time out. Brevemente una routine di time out serve per verificare se sia trascorso un preciso intervallo di tempo. L'abilitazione del timer consiste nel programmare il registro *T4CON*, abilitare la relativa interrupt nel registro *IEC1* e attivare le interrupt generali ponendo a 1 il bit *GIE* (general interrupt enable) . In questo modo il timer 4 è programmato per[4]:

1. utilizzare il clock interno F_P come sorgente, il quale lavora a 50Mhz[5];
2. lavorare in modalità 16 bit;
3. imporre un fattore di divisione unitario.

Il periodo del timer risulta quindi:

$$T = \frac{1}{F_P} \cdot 2^{16} = T_P \cdot 2^{16} = 1.31\,ms$$

La interrupt del timer viene chiamata quindi ogni T secondi ed incrementa di 1 la variabile volalile long chiamata *sysTick*. Questa variabile permette il confronto di istanti temporali diversi in termini di *Tick* (battito) e la implementazione di routine di attesa non bloccanti a differenza di *_delay_ms* . In questo modo è semplice realizzare funzioni di time out. Ad esempio si potrebbe salvare in una variabile in valore corrente di sysTick sommato ad una costante che quantifica un intervallo di tempo (in battiti). Quando il valore corrente supererà quello memorizzato, sarà "scaduto il time out", cioè sarà trascorso un tempo maggiore di quello indicato dalla costante additiva.

1.2.3 Le modifiche al codice C

All'interno del ciclo *while(1)* della funzione *main*() è inserita solamente la funzione principale:

void checkAndRunCommand(CommHelper Help)*

Questa si occupa della ricezione dei caratteri, verifica dei comandi e

3 Vedi pagina 291 del datasheet della famiglia di microcontrollori dsPIC33EPXXXGM3XX/6XX/7XX
4 Vedi pagina 216 del datasheet della famiglia di microcontrollori dsPIC33EPXXXGM3XX/6XX/7XX
5 Vedi pagina 143 del datasheet della famiglia di microcontrollori dsPIC33EPXXXGM3XX/6XX/7XX

eventualmente della loro esecuzione.
La funzione prende in ingresso un puntatore alla struct C *CommHelper*.
La struct contiene un array di *char receivedText* e un byte senza segno *responseLength*.

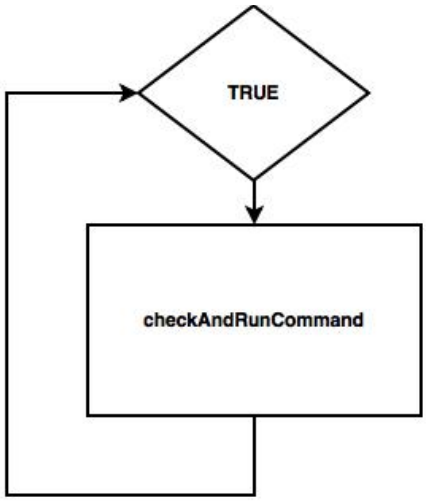


Figura 2. Logica del firmware

La figura 2 descrive la relazione fra la funzione e il firmware. Questa è chiamata dentro un loop infinito e rappresenta il cuore del sistema.

Ricezione di un comando
Per appurare la ricezione di una generica stringa di caratteri viene chiamata la routine *receiveData(CommHelper* Help)* la quale ritorna un valore positivo (memorizzato nella variabile *responseLength,* indicante la lunghezza del messaggio ricevuto) se è stata ricevuta una stringa, 0 altrimenti.
ReceiveData esegue *receiveChar(char* carattere)* fino a quando quest'ultima non ritorna 0. La funzione *receiveChar* prende in ingresso un puntatore a carattere e ci memorizza il valore ricevuto eventualmente dalla UART. In particolare:

1. Aspetta che sia trascorso un tempo maggiore del time out (vedi paragrafo 1.2.2) oppure che sia stato ricevuto un carattere dalla UART (in tal caso il bit *URXDA* del registro *U1STA* vale 1);
2. Se è stato ricevuto un carattere lo salva nel puntatore passato come argomento e ritorna 1, altrimenti 0 e il puntatore a carattere è invariato.

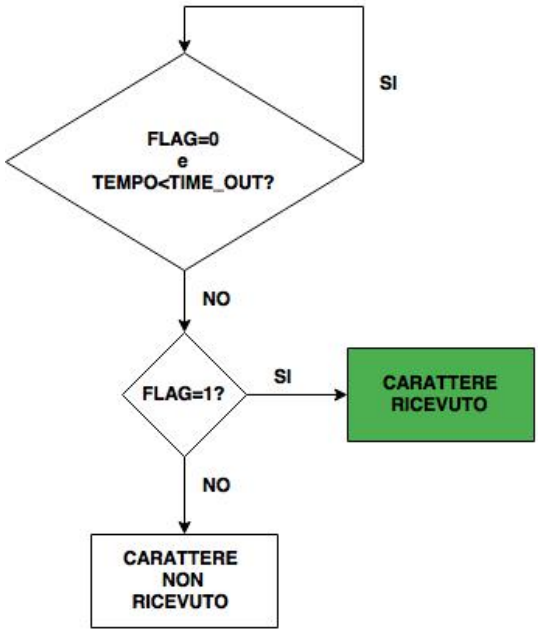


Figura 3. Logica della funzione receiveChar

La figura 3 riassume il funzionamento di *receiveChar*. Con TEMPO si intende il tempo trascorso dentro la funzione mentre FLAG indica il bit indicante se è stato ricevuto un carattere tramite la UART.
La routine *receiveData* chiama *receiveChar* passandogli il puntatore ad un elemento dell'array *receivedText,* la cui posizione è definita da *responseLength*. Se l'esito di *receivedChar* è positivo, viene incrementata *responseLength* di un valore. In questo modo l'array è riempito progressivamente con i caratteri ricevuti dalla UART.
Ovviamente il ciclo si ferma quando trascorre un tempo maggiore del time out. Se *responseLength* vale 0, significa che nessun carattere, quindi messaggio, è stato ricevuto.

Verifica del comando ricevuto

Nell'ipotesi che sia stata ricevuta una stringa, questa viene esaminata per decidere se si tratta di un comando valido.
Il confronto fra stringhe, in C puntatori a char, è facilitato dalla funzione *strncmp* che verifica l'ugualianza fra due stringhe.
Per prima cosa si verifica che le ultime due cifre corrispondano a "\r\n" e che la lunghezza in caratteri sia maggiore di 6 (il comando con lunghezza minore conta 7 caratteri).
Si esegue poi uno *switch case* sulla lunghezza della stringa (indicata dalla

variabile *responseLength* di *CommHelper*). Si distinguono quindi 3 casi:

1. Lunghezza pari a 14: il comando potrebbe essere SAMPLES. Si verifica che i primi 8 caratteri corrispondano a “SAMPLES=” e che i successivi 4 siano valori decimali (cioè che il valore del char sia compreso fra 47 e 58[6]). Si esclude anche il comando “SAMPLES=0000\r\n”.
2. Lunghezza pari a 13: il comando potrebbe essere DECADES. Si verifica che i primi 8 caratteri corrispondano a “DECADES=” e che i successivi 3 siano '0' o '1'. Si esclude anche il comando “DECADES=000\r\n”.
3. Lunghezza pari a 7: il comando potrebbe essere START. Si verifica che i primi 5 caratteri corrispondano a “START”.

Se la catena si ferma prima di riconoscere un comando viene inviata la stringa “ERROR\r\n” e il ciclo rincomincia da capo.
Se invece il comando ricevuto dal microcontrollore è valido viene trasmessa la stringa “OK\r\n” e si esegue la opportuna istruzione (memorizzando la nuova configurazione delle decadi o dei campioni per decade nelle relative variabili). Se il comando è START allora viene chiamata anche la funzione *startMeasure* che inizia la misurazione.

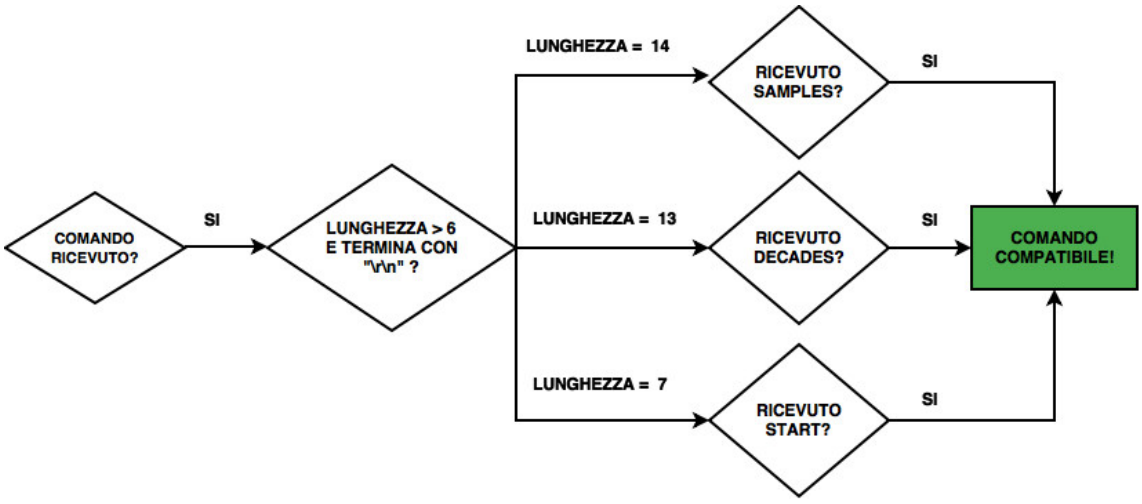


Figura 4. Logica verifica comandi ricevuti

La figura 4 mostra brevemente la logica utilizzata per la verifica di un

6 Vedi caratteri ASCII

generico comando.

La misurazione

La misurazione è gestita dalla funzione *startMeasure* che utilizza diverse parti di codice del vecchio firmware. *StartMeasure* esegue un loop per ogni decade abilitata, dentro il quale:

1. Viene calcolato il passo di frequenza secondo la regola:

$$Passo = \frac{Intervallo}{NumeroDiCampioniPerDecade}$$

Intervallo = 9000, 90000 e 900000 rispettivamente per la prima, seconda e terza decade.

2. Viene calcolata la frequenza corrente usando l'indice i del ciclo for (valore iniziale 1) e il valore ottenuto sopra.

$$Decade + (Passo \cdot (i - 1))$$

Decade = 1000, 10000 e 100000 rispettivamente per la prima, seconda e terza decade.

3. Il risultato è passato alla funzione *ChangeFreq1* che configura il generatore di funzione per funzionare alla frequenza specifica.
4. Vengono letti i valori degli ADC e inviati tramite *LeggiADC* e *writeCharUART*.

Le funzioni *ChangeFreq1, LeggiADC, writeCharUART* erano già presenti nel firmware.

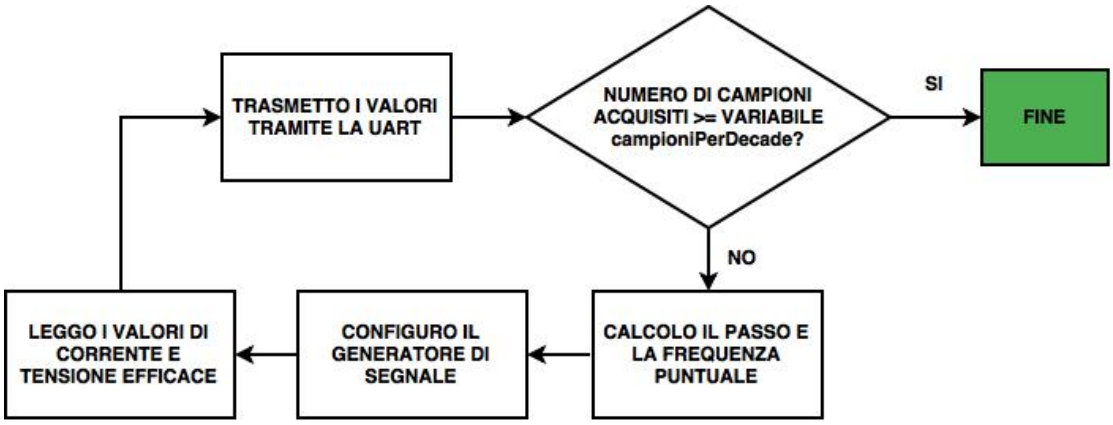


Figura 5. Logica acquisizione e invio campioni di una decade

La figura 5 descrive intuitivamente le azioni svolte per l'acquisizione e l'invio dei campioni di una decade. Lo stesso processo è eseguito per tutte e tre le decadi.

Dettagli maggiori sulle sperimentazioni si trovano nel capitolo 3.

Capitolo 2: il software

2.1 L'obiettivo

Lo sviluppo del software Java è la parte del progetto complementare allo sviluppo del firmware. L'obiettivo è creare una applicazione attraverso cui l'utente possa comunicare con l'impedenziometro: configurarlo, eseguire misurazioni e manipolare i dati ricevuti.

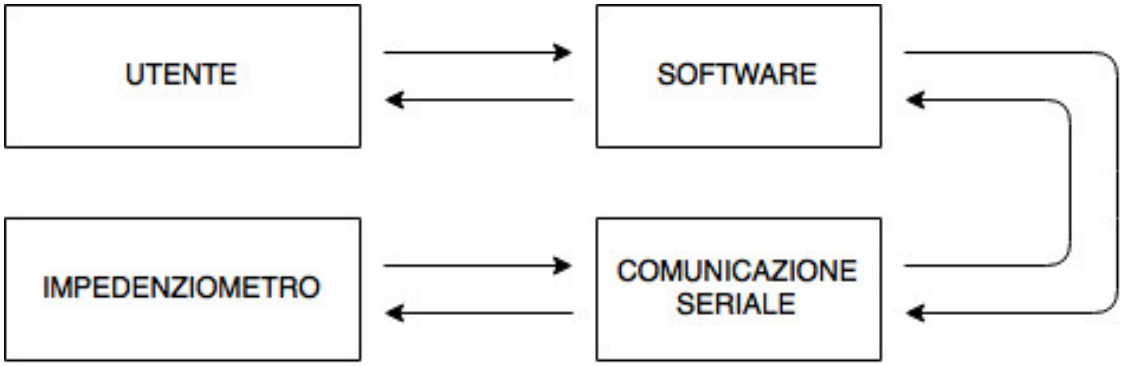


Figura 6. Schema astratto di funzionamento del sistema completo

La figura 6 mostra uno schema astratto dell'intero funzionamento del sistema composto da impedenziometro e software.
L'applicazione deve svincolare l'utente dai problemi di basso livello, come l'invio dei comandi allo strumento e la gestione dei dati ricevuti. Per tale ragione offre una interfaccia grafica che rende la gestione dell'hardware quanto più possibile intuitiva e immediata.

Le funzionalità richieste dal software sono:

1. Offrire una interfaccia attraverso cui comunicare con lo strumento in maniera immediata;
2. Possibilità di disporre in un grafico i dati acquisiti durante la misurazione;
3. Esportare i dati in un file con estensione CSV in modo da poter successivamente rielaborare il materiale con un foglio di calcolo qualsiasi.

2.2 La realizzazione

2.2.1 La scelta di linguaggio di programmazione e delle librerie

JavaFX
Il linguaggio di programmazione Java è stato scelto per la sua vasta offerta di librerie e versatilità. Il software, progettato sotto un sistema UNIX, si è dimostrato funzionare anche sotto un sistema Windows.
La libreria grafica scelta è JavaFX che permette la creazione di ricche applicazioni seguendo una logica vicina al Web, in cui ogni elemento della grafica è un nodo di un grande grafo ad albero. JavaFX mette a disposizione inoltre delle librerie per la creazione di grafici.
L'ambiente di sviluppo utilizzato è Netbeans, che in un unico IDE dispone di tutti gli strumenti necessari per progettare applicazioni JavaFX.

JSSC
JSSC (Java simple serial communication) è una libreria cross-platfom per lavorare con le porte seriali usando il linguaggio Java.

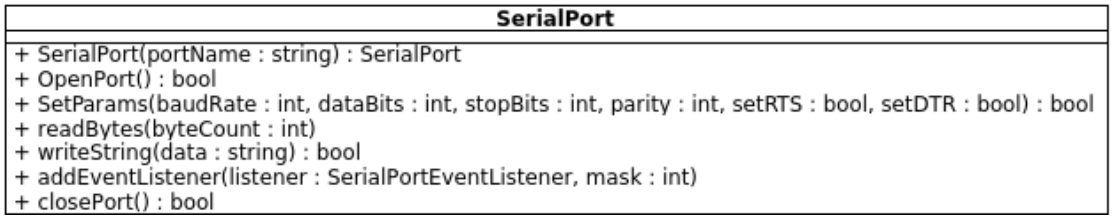


Figura 7. La classe SerialPort

La figura 7 descrive la classe fondamentale per l'utilizzo della libreria.
La connessione inizia con la creazione di un oggetto *SerialPort*, il cui costruttore necessita della la stringa indicante la porta seriale a cui connettersi. Tale stringa nei sistemi Windows è del tipo "COM" seguito da un valore numerico (“COM1”, “COM2”), mentre nei sistemi
UNIX è spesso del tipo "tty" seguito da caratteri alfanumerici (“ttyUSB0”, “ttyACM1”).
Per conoscere le porte virtuali disponibili si utilizza il metodo statico *SerialPortList.GetPortNames()* che ritorna un array di stringhe indicanti le

porte a cui connettersi.

Istanziato l'oggetto è necessario:

1. Aprire la porta chiamando il metodo *openPort()*;
2. Specificare i parametri per la comunicazione seriale, come baudrate, stopbit, databits e paritybit;
3. Aggiungere un *EventListener* in modo da gestire dinamicamente gli eventi seriali, filtrati tramite opportune maschere di bit specificate nell'argomento "*mask*" del metodo *addEventListener*;
4. Se necessario, inviare alla seriale stringhe di dati tramite il comando *sendString*;
5. Chiudere la connessione chiamando il metodo *closePort*, il quale rimuove anche l'EventListener.

JSON-simple

Il sito web json.org descrive JSON (JavaScript Object Notation) come "un semplice formato per lo scambio di dati. Per le persone è facile da leggere e scrivere, mentre per le macchine risulta facile da generare e analizzarne la sintassi"[7]. JSON supporta diversi tipi di dato ma per nel progetto si utilizza principalmente il tipo stringa.
JSON-simple è una libreria Java che permette la codifica e decodifica di file JSON.
La libreria è stata utilizzata per salvare in un file esterno, in modo permanente, le configurazioni di seriale, dispositivo e applicazione. Questo è servito principalmente per garantire la compatibilità del software con le future versioni dell'impedenziometro. Infatti, cambiando manualmente i valori nel file JSON, la applicazione sa se lo strumento è in grado di inviare valori di fase, trimmer, frequenza e doppia misura.
In particolare tali valori sono caricati all'avvio e salvati alla chiusura.

2.2.2 La GUI

La applicazione è formata di una finestra principale, il cui contenuto cambia dinamicamente, più due finestre secondarie utilizzate per modificare le impostazioni.
La classe *Tesi*, estensione la classe *Application,* è il "punto d'ingresso" di della applicazione JavaFX.

7 Da *http://json.org/json-it.html*

Nel metodo *start* della classe *Tesi* vengono istanziati tutti i nodi e creato il grafo ad albero. Alcuni nodi risultano visibili solo in opportune condizioni così anche il testo di alcune *Label* si adatta agli eventi informando l'utente dell'accaduto.
L'intera GUI è il risultato di combinazioni di molteplici nodi *Parent,* i quali possono contenere nodi figli fungendo così da contenitori di nodi.

I *Parent* più utilizzati nel progetto sono:

1. *HBox*: contenitore che dispone i nodi figli su una unica riga;
2. *VBox:* contenitore che dispone i nodi figli su una unica colonna;

Figura 8. Finestra principale all'avvio della applicazione

La figura 8 mostra la finestra all'avvio dell'applicazione. In alto è posizionato il menù orizzontale. In JavaFX questo è costituito dall'oggetto *MenuBar* e i suoi figli sono oggetti *Menu* (*File, Modifica, About*). Gli

oggetti *Menu* a loro volta contengono *MenuItem,* le voci nella finestra che si apre al passaggio del mouse (*Impostazioni seriale, Impostazioni dispositivo*). Per gestire il click sopra i *MenuItem* è sufficiente aggiungere un *EventHandler* attraverso il metodo *setOnAction*. Cliccando le due voci presenti in figura si aprono le relative finestre per configurare la seriale e il device.
Il centro della finestra è occupato da un *Button* nominato *start,* il cui stile è chiaramente personalizzato, avendo colore di background verde, dimensione del font maggiore e bordi arrotondati.
Sotto si trova invece una semplice *Label*, un contenitore di testo, il cui contenuto è modificato dinamicamente, tramite il metodo *setText.*

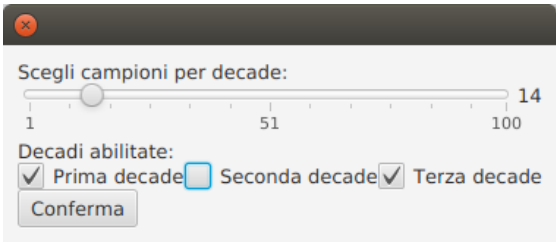


Figura 9. Configurazione dello strumento

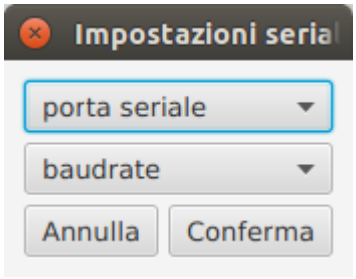


Figura 10. Configurazione seriale

Il grafo ad albero, che riflette la disposizione della GUI, può essere cambiato a runtime, così un nodo, o un insieme di nodi, può essere aggiunto o rimosso, realizzando effetti dinamici. Questo è possibile attraverso i metodi (ovviamente presenti solo nei nodi *Parent*) *getChildren().add* e *getChildren().remove,* dove *getChildren* ritorna una lista contenente tutti i nodi figli. Bisogna sottolineare che se un nodo viene aggiunto ad un *Parent* mentre è già figlio di un altro, viene automaticamente rimosso da quest'ultimo.

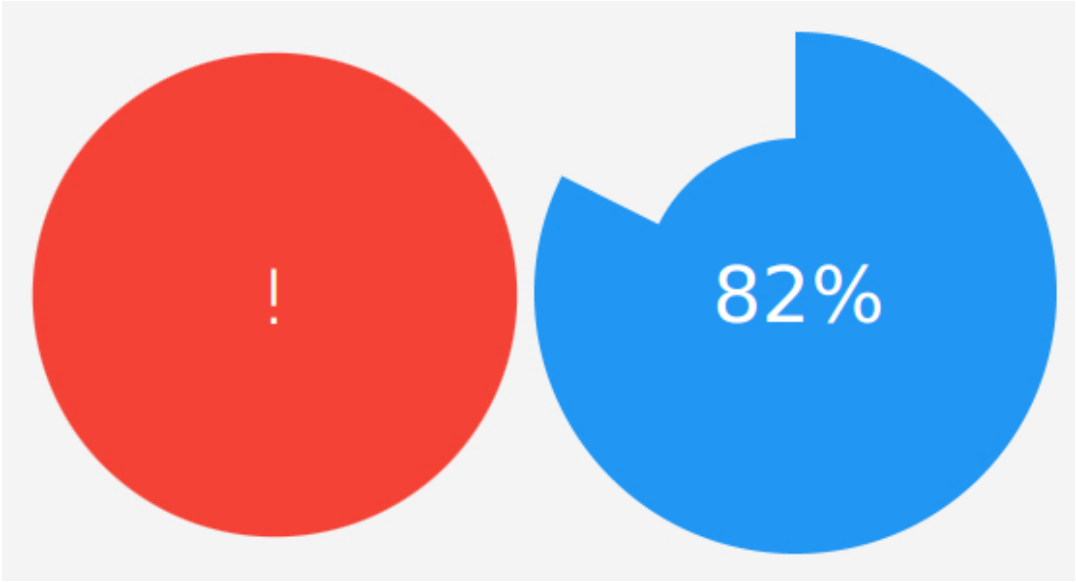


Figura 11. Nodi alternativi utilizzati nella applicazione

La figura 11 mostra due nodi, alternativi a quello di figura 8, utilizzati nella applicazione. Il primo si mostra quando, durante la ricezione dei campioni, si presenta un errore. Il secondo mostra invece la percentuale di dati ricevuti e funge da indicatore di progresso ed è un nodo personalizzato, chiamato *CustomProgressIndicator,* estensione del *Parent StackPane.*

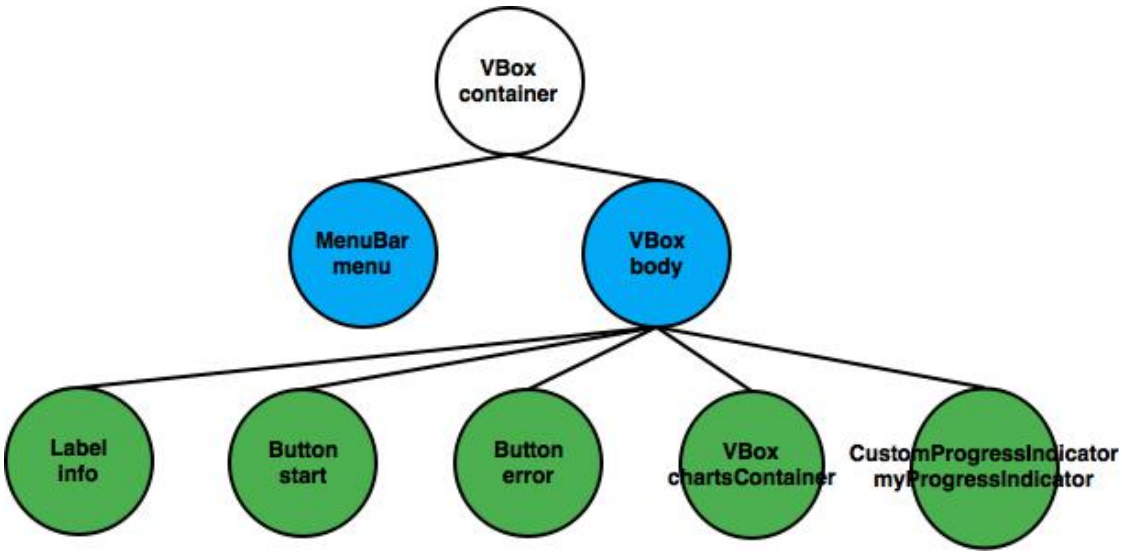


Figura 12. Grafo ad albero della GUI

La figura 12 mostra come sono disposti i principali nodi che compongono il grafo della scena principale. In cima vi è il nodo root, rappresentato dal

nodo bianco. I nodi blu sono fissi mentre quelli verdi sono aggiunti o rimossi dinamicamente. La *Label info* informa l'utente sul procedere delle operazioni.

CustomProgressIndicator

L'idea di base è creare un nodo che mostri la percentuale di caricamento e che fornisca un semplice metodo (*setProgress*) attraverso cui aggiornare l'avanzamento[8].

Il risultato è stato ottenuto sovrapponendo 3 componenti principali, un *Label,* un cerchio centrale (*Circle*), e un oggetto *Arc* di raggio maggiore.

L'oggetto *Arc* fornisce il metodo *setLength* con cui definire l'apertura, in gradi, dell'arco. La chiamata del metodo *setProgress* aggiorna quindi l'apertura dell'arco e il testo della *Label.* Siccome le dimensioni dell'arco cambiano in relazione alla sua apertura, è stato necessario creare un *group*[9] composto da un rettangolo, di lato uguale al diametro dell'arco, e dall'arco stesso. In questo modo, al variare dell'apertura l'arco rimane centrato. Brevemente, un oggetto *group* permette di propagare, le trasformazioni effettuate su di esso, a tutti i nodi ad esso collegato. Se ad esempio un gruppo viene traslato tutti i nodi collegati traslano della stessa quantità.

I grafici

La applicazione deve essere in grado di elaborare i dati ricevuti e stamparli in grafici logaritmici. Attualmente lo strumento invia solo il valore efficace di corrente e tensione ed è quindi possibile graficare solo il modulo dell'impedenza. Si è studiato però anche il protocollo che permetta la trasmissione del valore della fase e di altre informazioni utili. La classe utilizzata per creare i diagrammi di ampiezza e di fase è comunque la stessa.

JavaFX dispone di librerie per la creazione di diversi tipi di grafico. Il tipo da noi utilizzato è il LineChart, un grafico a due assi, x e y, estensione della classe *XYChart.*

I componenti fondamentali di un grafico a due assi sono:

1. La classe *LineChart* che si preoccupa di rappresentare i dati nel grafico a due assi;
2. Gli assi stessi del grafico. Purtroppo JavaFX non fornisce assi logaritmici quindi è stato necessario estendere direttamente la classe

8 Spunti tecnici sono stati presi dalla domanda posta su stackoverflow.com http://stackoverflow.com/questions/27418189/javafx-center-an-resizable-arc-in-a-borderpane

9 Maggiori informazioni al sito https://docs.oracle.com/javase/8/javafx/api/javafx/scene/Group.html

ValueAxis.

3. L'insieme di dati da rappresentare, contenuti nell'oggetto *XYChart.Series,* il quale contiene una lista di oggetti di tipo *XYChart.Data* che costituiscono la coppia di valori x e y dei due assi, cioè i singoli campioni.

La classe *LogarithmicAxis* implementa un asse di tipo logaritmico, si basa su un progetto trovato su Github[10] ed implementa tutti i metodi astratti della classe *ValueAxis.*
L'istanza di un oggetto *LogarithmicAxis* necessita solamente dei limiti inferiore e superiore dell'asse passati all'argomento del costruttore.

La classe *LineChart,* se non diversamente specificato attraverso il metodo s*etCreateSymbols,* crea automaticamente dei nodi di default (cerchi) in corrispondenza dei campioni della serie. Questi nodi sono stati resi tutti non visibili usando il metodo *setVisible* in modo che si possa evidenziare solamente il nodo eventualmente cliccato.
Sono stati aggiunti *EventHandler* per gestire gli eventi di input:

1. Alla pressione del tasto sinistro del mouse sull'oggetto *XYChart.Series* viene determinata la coordinata x del punto premuto usando il metodo *getValueForDisplay* della classe *LogarithmicAxis.* Ora, per individuare il campione da visualizzare, è sufficiente riconoscere quale della lista appartenente a *XYChart.Series* ha coordinata x più vicina al punto premuto. Trovato *XYChart.Data* viene reso visibile e i valori degli assi x e y vengono stampati sotto il grafico;
2. Alla pressione delle frecce destra o sinistra, viene evidenziato il campione rispettivamente successivo o precedente del punto selezionato. Essendo la lista ordinata basta trovare il primo valore con frequenza maggiore della serie oppure quello con frequenza minore che più si avvicina al campione visibile. Trovato il nuovo punto, viene reso visibile quest'ultimo e nascosto il precedente.

Tutte le funzionalità precedentemente discusse sono state riassunte nella classe *CharNode,* estensione di *VBox.* Questa, basandosi sul parametro *type* passatogli nel costruttore, sa se dovrà creare un asse verticale logaritmico (grafico del modulo) o lineare (grafico della fase). Infine, nel caso l'asse

10 Per maggiori informazioni visitare https://gist.github.com/kevinsdooapp/3227204

verticale sia logaritmico, vengono calcolati i valori di massimo e minimo del modulo in modo da poter passare al costruttore di *LogarithmicAxis* i valori estremi dell'asse.
I valori utilizzati per la creazione della list di *XYChart.Data,* cioè del grafico stesso, sono passati direttamente al costruttore di *ChartNode.* Vengono utilizzati una *List<Number>* per l'asse verticale e un array di int per l'asse delle frequenze.

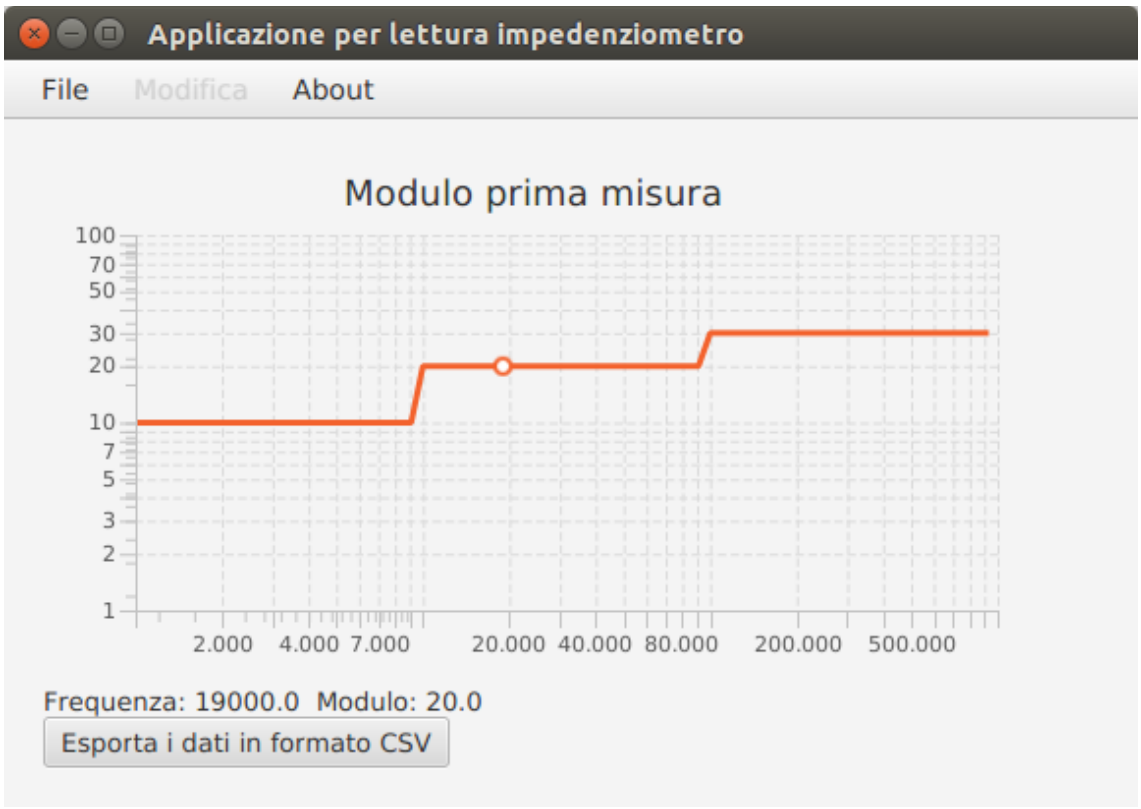


Figura 13. Esempio di grafico

La figura 13 mostra un esempio di grafico logaritmico.

2.2.3 La gestione delle impostazioni

Le impostazioni sono gestite dalla classe *SettingsManager*. Questa classe è del tipo *singleton,* cioè una unica istanza è condivisa da tutta la applicazione. La realizzazione è possibile attraverso la dichiarazione di un costruttore privato e di un metodo statico pubblico che ritorna la istanza esistente.
SettingsManager contiene tre campi privati rappresentanti tre classi distinte che fungono da contenitore per le configurazioni di applicazione, seriale, strumento.

1. La classe *SerialConfig* gestisce le impostazioni della seriale. Contiene variabili da passare al metodo *SetParams* della classe *SerialPort*;
2. La classe *DeviceConfig* gestisce le impostazioni dello strumento hardware. Contiene le variabili da appendere ai comandi da inviare allo strumento (numero di campioni per decade, decadi abilitate);
3. La classe *ApplicationConfig* gestisce le impostazioni della applicazione. Contiene variabili booleane che dichiarano se lo strumento è in grado o meno di gestire un particolare tipo di misura. Giocando con queste variabili il software si aspetta un blocco dati più o meno denso di informazioni, ad esempio potrà supportare l'invio di fase e/o frequenza. In questo modo si è cercato di mantenere la compatibilità con le future versioni dello strumento.

All'avvio della applicazione, dentro il metodo *start* della classe *Tesi,* viene invocato il metodo statico *SettingsManager.getInstance* ed, essendo la prima invocazione, richiama il costruttore privato. Il costruttore carica le impostazioni salvate utilizzando il metodo *loadSettings*.
Il metodo *loadSettings* apre il file salvato su disco "*settings.json*" usando un *BufferedReader*[11] generando una stringa con il contenuto del file. Questa stringa è poi decodificata usando il metodo *parse* della classe *JSONParser*[12].
Il metodo genera una java *map* da cui si ricavano i valori salvati e si generano i 3 oggetti contenitori di informazioni descritti sopra.
Durante la chiusura del programma si effettua l'operazione duale. Basandosi sui valori correnti dei campi *SerialConfig, ApplicationConfig,*

11 Per maggiori informazioni visitare http://docs.oracle.com/javase/7/docs/api/java/io/BufferedReader.html
12 Per maggiori informazioni visitare https://code.google.com/p/json-simple/

DeviceConfig di *SettingsManager* il metodo pubblico *saveSettings* sovrascrive il file "*settings.json"* con le nuove impostazioni.

Figura 14. Esempio di file settings.json

La figura 14 mostra un esempio di configurazione. Se si volessero cambiare le impostazioni della applicazione sarebbe sufficiente modificare il valore booleano ("*true*" oppure "*false*") in corrispondenza della rispettiva variabile. Ovviamente il tipo delle variabili è stringa ma si intende possano assumere solo valori "*true*" o "*false*".
Le istanze di *SerialConfig* e *DeviceConfig,* campi di *SettingsManager,* sono passate agli *stage* che permettono la modifica dei valori (vedi figure 5 e 6), così da cambiare la configurazione corrente.

2.2.4 La comunicazione seriale

La comunicazione seriale è gestita dalla classe *DeviceManager,* anche questa *singleton*. La classe contiene una istanza di *SerialPort* e utilizza i dati gestiti da *SettingsManager* per connettersi alla porta seriale scelta e inviare i comandi allo strumento. La gestione della connessione seriale avviene utilizzando la libreria JSSC come spiegato precedentemente.

DeviceManager possiede un campo privato di tipo S*tringBuffer*[13] chiamato *receivedData* che è il contenitore della stringa ricevuta. Il tipo è stato scelto perché, a differenza del tipo stringa, è passato alle funzioni come puntatore e offre metodi "thread-safe", cioè sincronizzati.
Per la ricezione dei caratteri è stato implementato un *SerialPortEventListener,* in cui, ogni carattere ricevuto, è appeso a *receivedData.*

L'invio dei comandi allo strumento
L'invio di comandi al dispositivo si basa sull'interfaccia *Command,* che presenta il metodo *"public boolean Execute()"*.
Due comandi utili sono:

1. *MultiExecutionCommand:* al costruttore si passa un oggetto di tipo *Command* e la chiamata di *Execute* esegue il metodo *Execute* del comando passatogli un numero di volte indicato dalla costante *NUMBER_OF_EXECUTION,* a una distanza di tempo specificata dalla costante *SLEEP_TIME*, fino a che non ritorna *true.* Se la esecuzione del comando non ritorna mai *true* allora *Execute* di *MultiExecutionCommand* ritorna *false*;
2. *WaitResponseCommand:* comando utilizzato per verificare se la stringa inviata allo strumento sia stata interpretata correttamente o no, cioè se si riceva rispettivamente *"OK\r\n"* oppure *"ERROR\r\n"*. Al costruttore si passa direttamente *receiveData.* Il metodo *Execute* analizza il contenuto di *receiveData* usando i metodi forniti dalla classe *stringBuffer* (*subString, delete*) e comuni agli aggetti *String* (*equal*). Se riconosce la stringa *"OK\r\n"* ritorna *true.* Se si riconosce *"ERROR\r\n"* ritorna *false.* In entrambi i casi la stringa ricevuta viene rimossa da *receiveData*. Infine se trascorre un tempo in millisecondi maggiore del valore specificato dalla costante *TIME_OUT_RECEIVING* (funzionalità realizzata utilizzando *System.currentTimeMillis()* e paragonando il tempo all'inizio della esecuzione con quello corrente) o se la stringa analizzata differisce totalmente, il metodo ritorna *false.*

Le classi *SetDecadesCommand, SetDoubleMeasureCommand, SetSamplesCommand, StartMeasureCommand* inviano i relativi comandi allo strumento attraverso la seriale. Al loro costruttore è necessario passare

13 Per maggiori informazioni visitare http://docs.oracle.com/javase/7/docs/api/java/lang/StringBuffer.html

sia l'istanza di *SerialPort* sia *receivedData*.
In particolare:

1. Generano la stringa da inviare combinando il comando stesso ("DECADES=", "SAMPLES=") e i valori di configurazione ottenuti tramite *SettingsManager;*
2. Inviano la stringa ottenuta (ad esempio "DECADES=001\r\n", "SAMPLES=0004\r\n" o semplicemente "START\r\n") chiamando il metodo *writeString* di *SerialPort;*
3. Eseguono il comando *WaitResponseCommand;*

2.2.5 La misurazione completa

Scelte le configurazioni di strumento, seriale e applicazione il bottone di start diventa attivo ed è possibile iniziare la misurazione.

L'aggiornamento della GUI
Alla pressione del pulsante viene chiamato il metodo start di *DeviceManager.* L'aggiornamento della GUI è facilitato dalla lettura della variabile *state* di tipo *DeviceManagerState* (*enum)*.
I valori assunti di *state* sono:

- *DeviceManagerState.idle:* valore di default. La applicazione è in attesa di una richiesta di inizio misurazione;
- *DeviceManagerState.connecting:* valore assunto dopo essersi connessi alla porta seriale ma si stanno inviando i comandi di configurazione all'impedenziometro;
- *DeviceManagerState.waitingData:* la comunicazione con lo strumento è avvenuta e si sta ricevendo il blocco dati utile;
- *DeviceManagerState.elaboratingData:* l'intero blocco dati è stato ricevuto. Si stanno processando i dati;
- *DeviceManagerState.finished:* la misurazione è finita. I dati sono contenuti nell'oggetto *DataContainer*[14].
- *DeviceManagerState.errorOccured:* valore assunto qualora si presentasse un errore a qualsiasi livello della misurazione, anche se la ricezione dei dati impiegasse troppo tempo.

Dopo la chiamata del metodo start di *DeviceManager,* viene lanciato un

14 Vedi paragrafo successivo

thread che osserva la dinamica della misurazione chiamando il metodo *getState* di *DeviceManager.* Viene utilizzata la funzione *Platform.runLater()*[15] qualora sia necessario aggiornare la GUI. Nello specifico:

1. Si attende che lo stato di *DeviceManager* sia diverso da *DeviceManagerState.idle;*
2. Si sostituisce il *button start* con il *CustomProgressIndicator* e si visualizza la scritta "*Connessione in corso...*" editando il testo della *Label*;
3. Se lo stato è *DeviceManagerState.waitingData* si cancella il testo della *Label* e si attende che cambi valore. Ogni 500 millisecondi si aggiorna la percentuale del *CustomProgressIndicator* chiamando il metodo *setProgress* e passandogli il rapporto fra la lunghezza della stringa ricevuta e la lunghezza del blocco dati totale stimato (vedi il prossimo paragrafo);
4. Se lo stato è *DeviceManagerState.finished* la misurazione è conclusa. Vengono rimossi i nodi non più utilizzati e viene chiamata la funzione *populateChartsContainer* che prende in ingresso l'oggetto *DataContainer* e lo *stage* corrente, il quale serve per realizzare la funzione di esportazione file CSV. Infine viene chiamato il metodo *clearState* di *DeviceManager* che riporta lo stato a *DeviceManagerState.idle* e resetta la variabile *receivedData*.

Qualora si verificassero problemi e lo stato diventasse *DeviceManagerState.errorOccured* verrebbe stampato a video il bottone rosso (vedi figura 7) con la relativa segnalazione.

Il metodo start della classe *DeviceManager*

Il metodo start lancia un nuovo thread che gestisce la comunicazione seriale, manipola i dati ricevuti e cambia, quando necessario, lo stato di *DeviceManager.*

In particolare:

1. Tenta la connessione alla porta seriale scelta;
2. Stabilita la connessione prova a configurare l'impedenziometro. Ogni comando è passato ad una istanza di *MultiExecutionCommand* e ne viene eseguito il relativo Execute. Se l'operazione ha esito positivo

15 Per maggiori informazioni visitare https://docs.oracle.com/javase/8/javafx/api/javafx/application/Platform.html

viene inviato il comando di inizio della misurazione “START\r\n”.

3. Viene chiamato il metodo *estimateLength.* Questo, basandosi sulle configurazioni correnti di seriale, strumento e applicazione, calcola la lunghezza totale del blocco dati che si aspetta di ricevere, la lunghezza di ogni riga (delimitata dai caratteri finali “\r\n”), e stima un tempo (pessimistico) di ricezione dell'intero blocco. Il tempo di ricezione, in millisecondi, è calcolato moltiplicando il numero di righe attese per il tempo stimato di invio di una singola riga specificato dalla costante *TICK_PER_ROW.*
4. Si attende che l'intero blocco sia ricevuto. L'evento si ritiene verificato quando la lunghezza della stringa ricevuta (contenuta ricordiamo nella variabile *receivedData*) è maggiore o uguale alla lunghezza stimata.
5. Si chiude la connessione con la porta seriale e si chiama il metodo decode*Data.* Tale metodo, basandosi sulle configurazioni correnti di seriale, strumento e applicazione, calcola a quale posizione, in ogni riga, si trova ogni singolo campione. Utilizzando i valori trovati scansiona l'intera stringa ricevuta salvando i campioni nei relativi contenitori. Per i campioni di valore efficace di tensione e corrente, fase, fase calcolata, modulo e trimmer di entrambi le misurazioni si utilizzano *List<Number>* per la loro versatilità mentre per i campioni di frequenza si usa un array di int. Se lo strumento non supporta l'invio del valore di frequenza allora viene calcolato usando un metodo analogo al microcontrollore[16]. Inoltre calcola i moduli e le fasi delle misure. Ottenuti tutti i valori, viene creato un oggetto *DataContainer* a cui si passano tutti i contenitori (liste e array) detti sopra. Questa semplice classe presenta i metodi che ritornano la misura di interesse. Ad esempio *getI1rms* ritorna la lista contenente i valori efficaci della prima misurazione mentre *getFrequency* l'array con i valori di frequenza.

Siccome lo strumento non supporta ancora l'invio dei valori dei trimmer, i colcoli di modulo e fase non sono precisi. A questo scopo sono state create due funzioni private di *DeviceManager*, *calculateModule* e *calculatePhase,* chiamate da *decodeData,* in cui è possibile modificare la espressione e considerare anche il valore di *trimmer.* Attualmente infatti eseguono solo il rapporto fra valore efficace di tensione e corrente per il modulo mentre per la fase ritornano il valore di fase stesso.

16 Vedi paragrafo 1.2.2

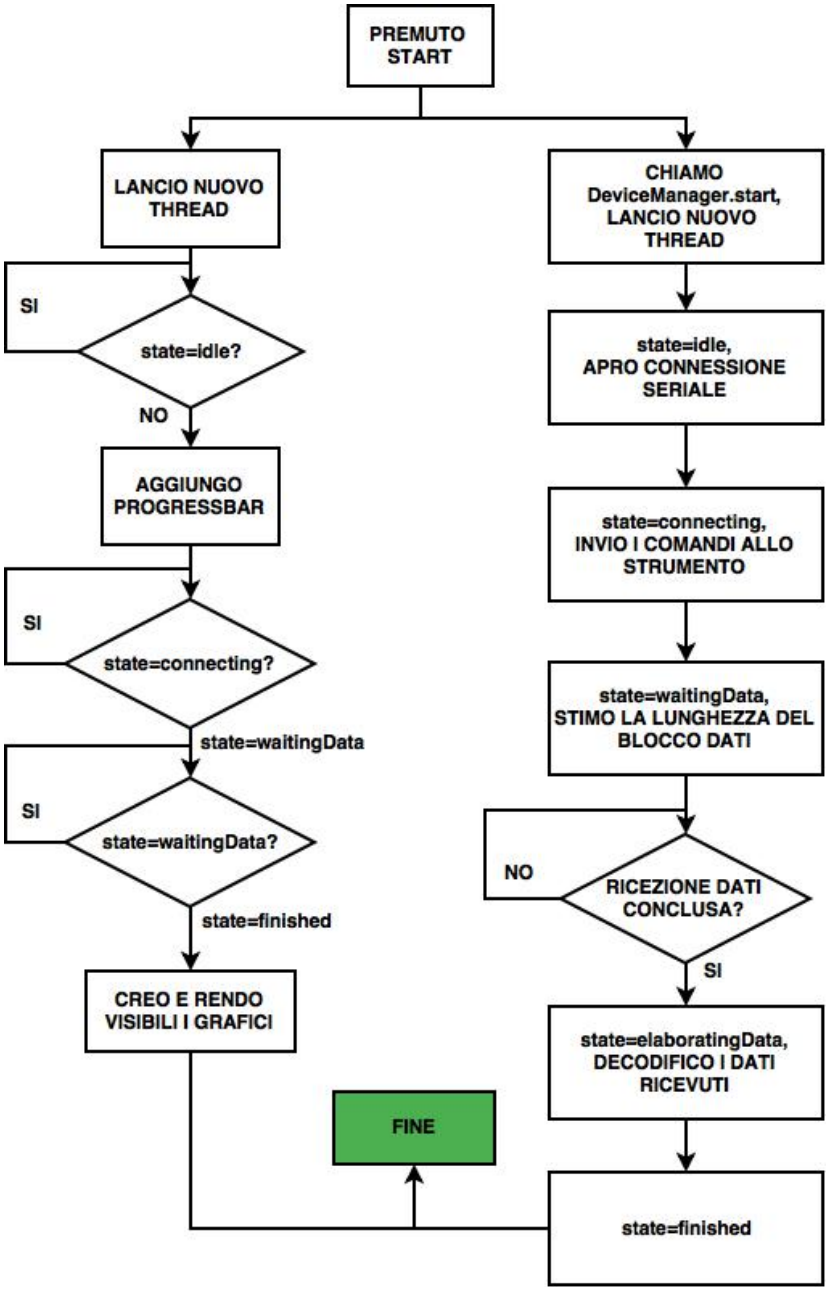


Figura 15. Diagramma di flusso della misurazione

La figura 15 mostra il diagramma di flusso relativo alla gestione dell'intero processo di misurazione. Non sono riportati i casi di errore per comodità di rappresentazione.

Il metodo *populateChartsContainer*

Il metodo *populateChartsContainer* aggiunge ad un oggetto *VBox* chiamato *chartsContainer* istanze di *ChartNode,* nodo che rapprenseta un singolo grafico (vedi il paragrafo sulla GUI) , e lo appende nella finestra principale. Basandosi sulle solite configurazioni ottenute tramite la classe *SettingsManager* il metodo capisce quali grafici stampare. Se lo strumento è abiliatto all'invio della fase, stampa, oltre al grafico del modulo, anche il diagramma di fase. Se supporta anche la doppia misura, ripete lo stesso procedimento per quest'ultima.
Infine viene appeso anche il bottone *export* che permette il salvataggio dei dati in formato CSV.

Il salvataggio dei dati in formato CSV

Un file CSV è un file di testo, con una precisa ma semplice formattazione, che un generico foglio di calcolo può interpretare.
In particolare ogni riga della tabella è delimitata dal carattere '\n' mentre ogni colonna è separata indifferentemente dallo spazio o dalla virgola ','.
Alla pressione del tasto *export:*

1. Una istanza della classe *FileChooser* è creata ed eseguito il suo metodo *showSaveDialog* passandogli lo stage corrente. Il metodo apre la classica finestra per la selezione di un percorso nel file system. Un oggetto *File,* indicante il percorso scelto, è riportato dal metodo;
2. Viene creata una stringa che riflette le regole del formato CSV. La prima riga indica il titolo della relativa colonna (modulo1, fase1, V1rms, frequenza ecc.) le successive rappresentano i campioni alla specifica frequenza. Per riempire la stringa si scandiscono con un unico loop tutti i contenitori forniti da *DataContaniner* riga per riga filtrati in base alle impostazioni di *SettingsManager;*
3. Utilizzando la classe *FileWriter* viene infine salvato il contenuto della stringa al percorso specificato dall'oggetto *File.*

Capitolo 3: le prove sperimentali

3.1 Test del firmware

Per testare il firmware si è utilizzato uno strumento incluso nell' IDE di arduino[17] che facilita la comunicazione con le porte seriali e permette di inviare i comandi tramite tastiera. In questo modo si ha avuto la possibilità di testare manualmente la funzionalità del sistema.
A parte i primi inceppi iniziali il firmware si è dimostrato funzionante.

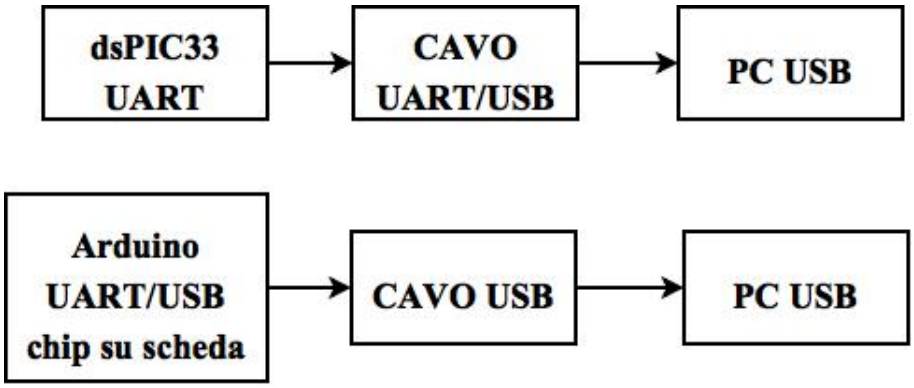


Figura 16. Collegamenti di arduino e impedenziometro con PC

La figura 16 mostra come i dispositivi sono stati collegati al computer.
Nel caso dell'impedenziometro è stato necessario utilizzare un cavo con integrato un convertitore UART/USB, mentre nel caso di arduino la scheda comprende già un chip analogo quindi è sufficiente un semplice cavo USB.

17 Per maggiori informazioni visitare https://www.arduino.cc/en/Main/Software

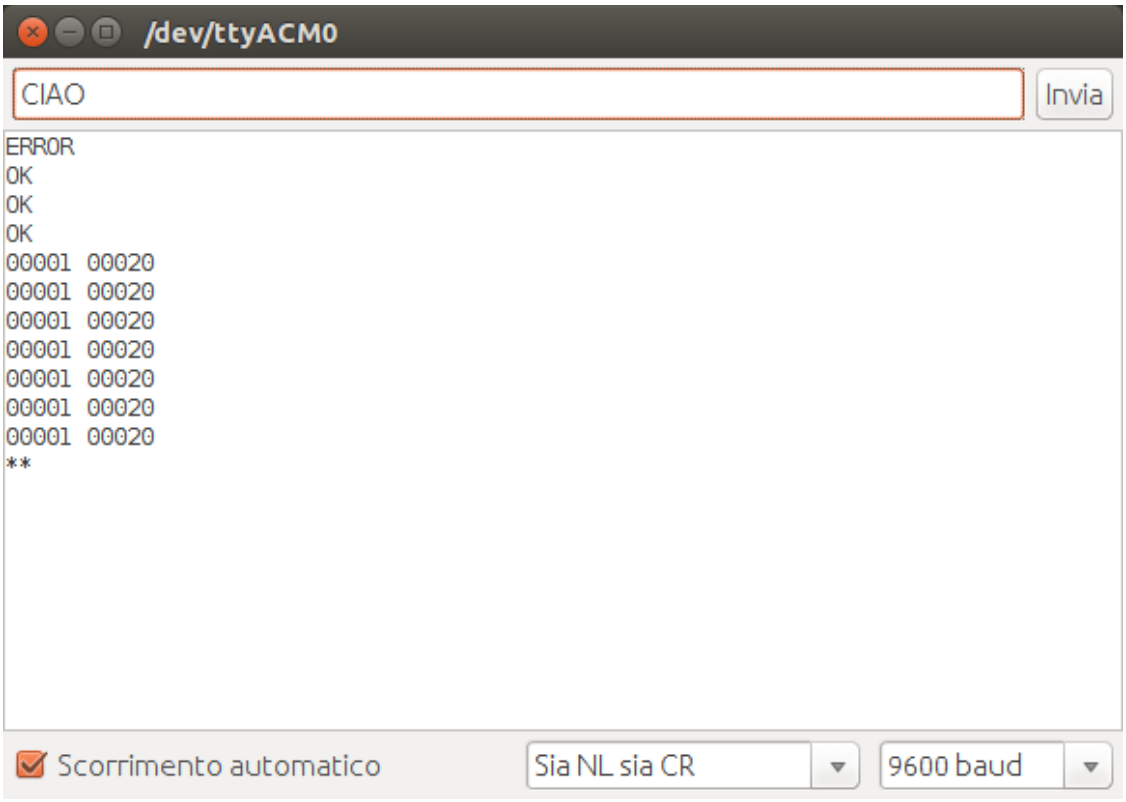


Figura 17. Tool fornito dall'IDE di arduino

La figura 17 mostra come si presenta la schermata del tool utilizzato. Si possono leggere alcune risposte inviate da arduino (vedi dopo).

3.2 Test del software

In maniera duale al test del firmware, è stato programmato arduino in modo da simulare il comportamento dell'impedenziometro. In questa maniera si è potuto verificare anche la compatibilità con gli sviluppi futuri dell'hardware, già simulati da arduino.
Per rendere arduino in grado di simulare le varie fasi evolutive dell'hardware, sono stati programmati alcuni suoi pin digitali, in modo che, muovendo semplicemente dei contatti, sapesse quali dati inviare dipendentemente dal loro valore logico.
Prima di ogni test è stato necessario configurare allo stesso modo arduino e la applicazione (modificando il file *settings.json*).
I dati di valore efficace di tensione e corrente e il valori della fase sono stati

scelti in modo che rappresentassero delle spezzate nel dominio delle frequenze.

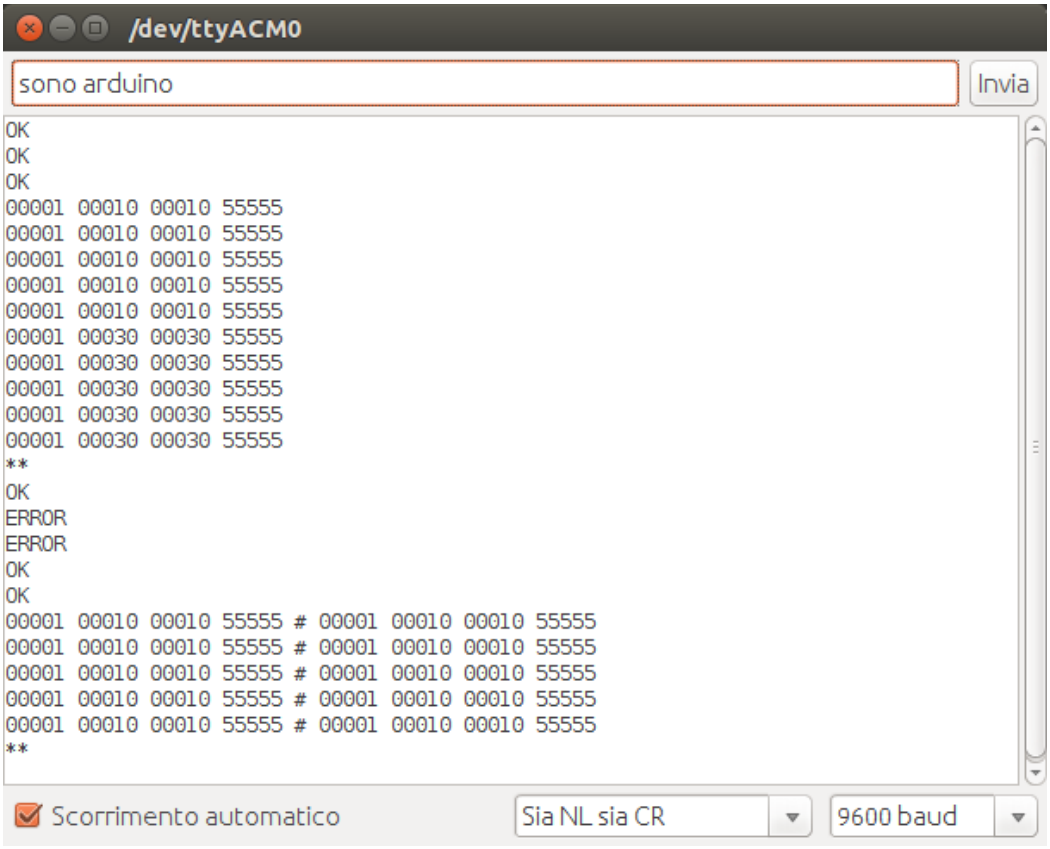


Figura 18. Arduino simula l'impedenziometro

La figura 18 mostra come arduino si comporti all'invio dei comandi. Il primo blocco dati rappresenta l'invio di una singola misura in cui sono stati appesi anche i valori della fase e del trimmer. Il secondo blocco invece coincide con la trasmissione di una misura doppia con fase e trimmer abilitati.

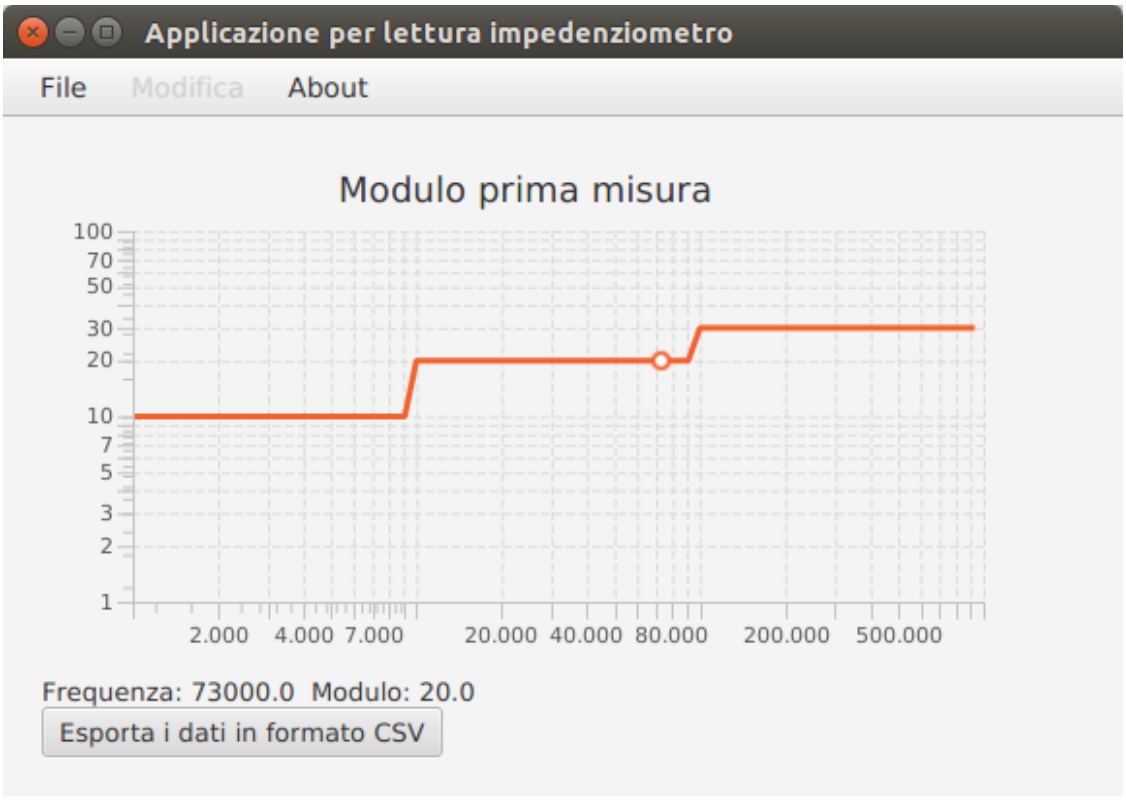


Figura 19. Grafico del solo modulo. Singola misura.

La figura 19 mostra il grafico del solo modulo della prima (e sola) misurazione. Tutte tre le decadi sono abilitate. Come accennato precedentemente viene disegnata una spezzata, costante nelle frequenze relative alle 3 decadi.

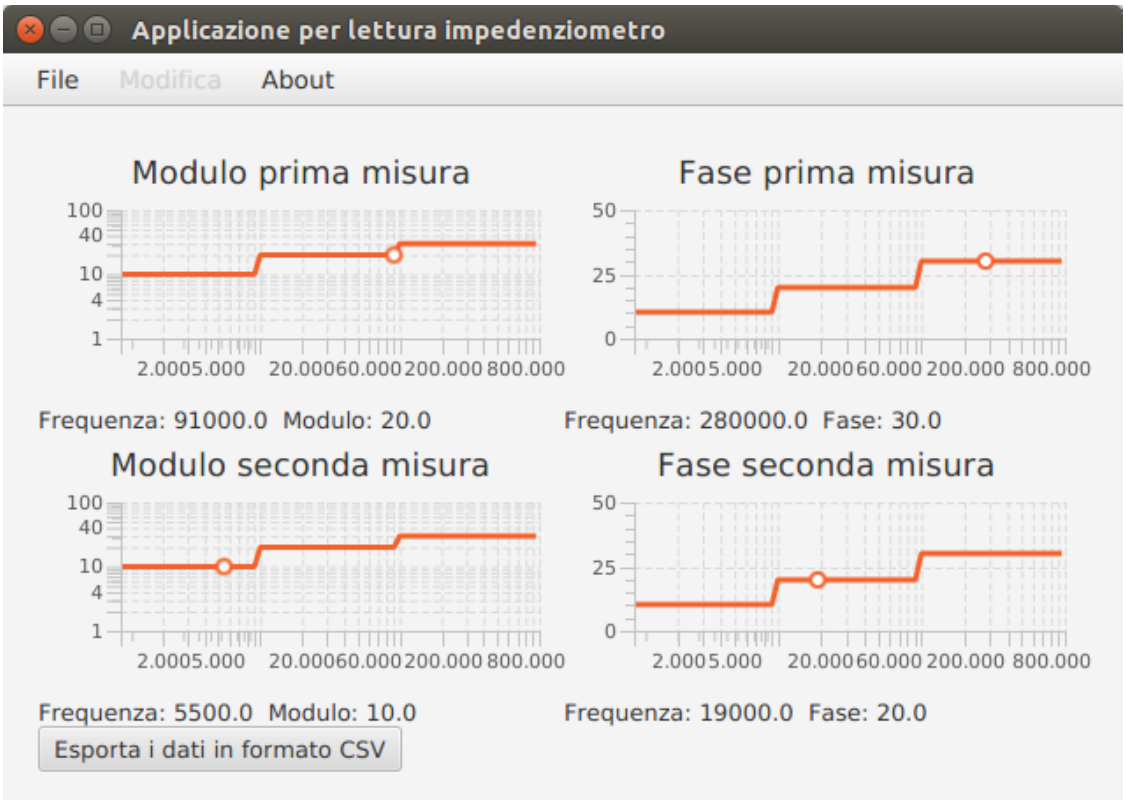
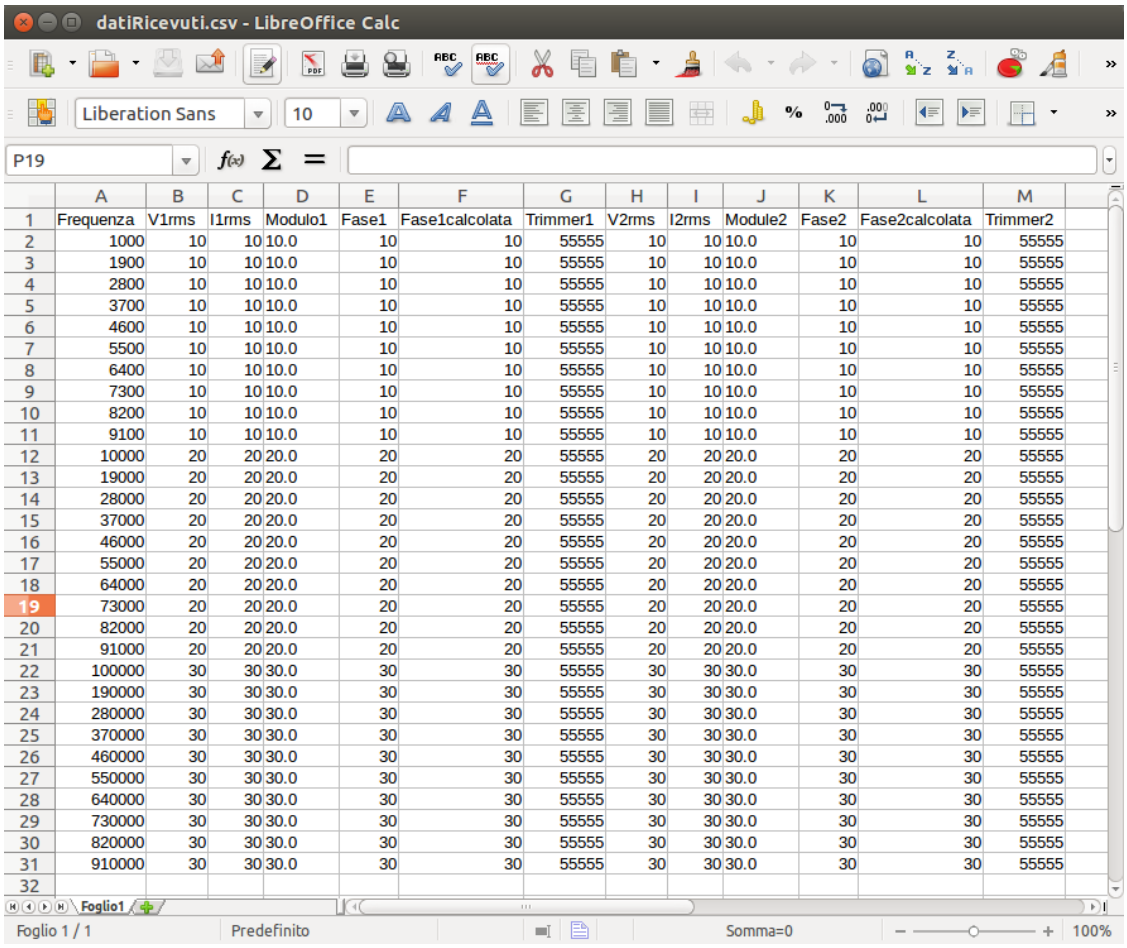


Figura 20. Grafici di modulo e fase. Doppia misura.

La figura 20 invece mostra il caso in cui arduino è configurato per inviare fase e doppia misura. Tutte le tre decadi sono abilitate.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Frequenza	V1rms	I1rms	Modulo1	Fase1	Fase1calcolata	Trimmer1	V2rms	I2rms	Module2	Fase2	Fase2calcolata	Trimmer2
2	1000	10	10	10.0	10	10	55555	10	10	10.0	10	10	55555
3	1900	10	10	10.0	10	10	55555	10	10	10.0	10	10	55555
4	2800	10	10	10.0	10	10	55555	10	10	10.0	10	10	55555
5	3700	10	10	10.0	10	10	55555	10	10	10.0	10	10	55555
6	4600	10	10	10.0	10	10	55555	10	10	10.0	10	10	55555
7	5500	10	10	10.0	10	10	55555	10	10	10.0	10	10	55555
8	6400	10	10	10.0	10	10	55555	10	10	10.0	10	10	55555
9	7300	10	10	10.0	10	10	55555	10	10	10.0	10	10	55555
10	8200	10	10	10.0	10	10	55555	10	10	10.0	10	10	55555
11	9100	10	10	10.0	10	10	55555	10	10	10.0	10	10	55555
12	10000	20	20	20.0	20	20	55555	20	20	20.0	20	20	55555
13	19000	20	20	20.0	20	20	55555	20	20	20.0	20	20	55555
14	28000	20	20	20.0	20	20	55555	20	20	20.0	20	20	55555
15	37000	20	20	20.0	20	20	55555	20	20	20.0	20	20	55555
16	46000	20	20	20.0	20	20	55555	20	20	20.0	20	20	55555
17	55000	20	20	20.0	20	20	55555	20	20	20.0	20	20	55555
18	64000	20	20	20.0	20	20	55555	20	20	20.0	20	20	55555
19	73000	20	20	20.0	20	20	55555	20	20	20.0	20	20	55555
20	82000	20	20	20.0	20	20	55555	20	20	20.0	20	20	55555
21	91000	20	20	20.0	20	20	55555	20	20	20.0	20	20	55555
22	100000	30	30	30.0	30	30	55555	30	30	30.0	30	30	55555
23	190000	30	30	30.0	30	30	55555	30	30	30.0	30	30	55555
24	280000	30	30	30.0	30	30	55555	30	30	30.0	30	30	55555
25	370000	30	30	30.0	30	30	55555	30	30	30.0	30	30	55555
26	460000	30	30	30.0	30	30	55555	30	30	30.0	30	30	55555
27	550000	30	30	30.0	30	30	55555	30	30	30.0	30	30	55555
28	640000	30	30	30.0	30	30	55555	30	30	30.0	30	30	55555
29	730000	30	30	30.0	30	30	55555	30	30	30.0	30	30	55555
30	820000	30	30	30.0	30	30	55555	30	30	30.0	30	30	55555
31	910000	30	30	30.0	30	30	55555	30	30	30.0	30	30	55555
32													

Figura 21. File CSV aperto con LibreOffice Calc

La figura 21 infine mostra come i dati salvati in formato CSV siano stati facilmente aperti da LibreOffice Calc. I dati rappresentano una doppia misura, con invio anche di valori di fase e trimmer. I valori con virgola erano, prima dell'esportazione, gestiti dal software come double, mentre quelli senza virgola come interi.

3.3 Test del sistema completo

Il software è stato testato direttamente con l'impedenziometro.
L'applicazione è stata configurata, attraverso il file *settings.json,* in modo che prevedesse la trasmissione dei soli valori efficaci e di una sola misura.
La impedenza di test collegata agli elettrodi era una resistenza da 1Kohm.
La figura 22 mostra il modulo calcolato tramite i valori ricevuti. Come era facile aspettarsi il grafico non è costante ma oscilla intorno ad un valore.

Ovviamente i dati sono fuorvianti perché, come abbiamo già detto, non considerano il guadagno degli stadi di amplificazione del segnale. Inoltre i valori indicati sugli assi non sono corretti in quanto l'immagine risale ad un periodo in cui il progetto dei grafici non era ancora definitivo. L'importanza di questo test risiede nel essere riusciti a far comunicare i due sistemi indipendenti, ma che parlano lo stesso linguaggio.

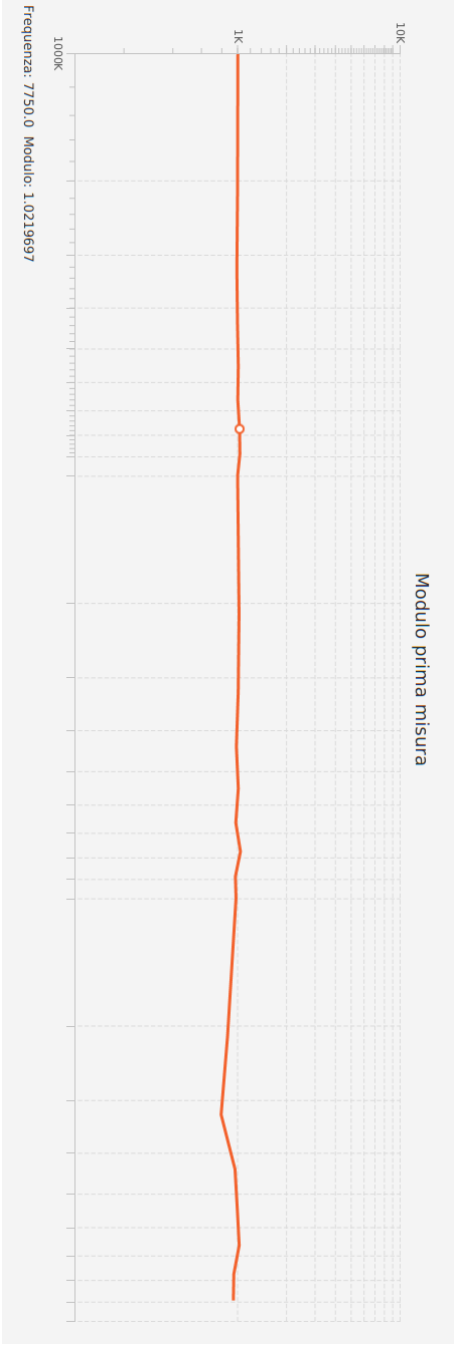


Figura 22. Grafico del modulo. Prova sistema completo

Conclusioni

Lo sviluppo della piattaforma software si è dimostrato altamente stimolante. Durante la sua evoluzione, come accade normalmente, si sono manifestati problemi di varia natura, ma che sono stati progressivamente risolti. Una aggiunta fondamentale è il legame fra i valori letti e i valori dei trimmer digitali. Senza questa relazione la applicazione non riesce (come è stato detto più volte) a calcolare i valori reali misurati perché non vengono considerati i guadagni degli stadi a monte degli RMS detector.
Alcune scelte di progetto sono discutibili. La ricezione delle stringhe nel firmware potrebbe essere implementata in diversi modi. Una alternativa è abilitare una interrupt che appende i caratteri ricevuti tramite UART ad un array e scansionarlo per verificarne il contenuto. Inoltre, l'utilizzo di due thread separati durante la misurazione non è una scelta ottimale. Lo stesso risultato si sarebbe ottenuto usando ad esempio un *ChangeListener,* il quale avrebbe notificato la modifica dello stato del sistema, senza la necessità di lanciare un secondo thread.
Nonostante ciò il sistema si è dimostrato funzionante e soprattutto funzionale. Il software permette una gestione ad alto livello del sistema di misura di impedenze ed è in grado di elaborare i dati in grafici logaritmici e eventualmente salvarli in file CSV come richiesto.

Bibliografia

1. Oracle http://docs.oracle.com/javase/8/javase-clienttechnologies.htm
2. Oracle http://docs.oracle.com/javase/8/docs/api/
3. JSSC https://code.google.com/p/java-simple-serial-connector/
4. JSON-simple https://code.google.com/p/json-simple/
5. Github https://gist.github.com/kevinsdooapp/3227204
6. Stackoverflow http://stackoverflow.com/questions/27418189/javafx-center-an-resizable-arc-in-a-borderpane
7. Giulia Luciani. “Strumento per la diagnosi di carcinoma BCC basato su spettroscopia empedenziometrica”. 2013-2014
8. Microchip. “dsPIC33EPXXXGM3XX/6XX/7XX 16-Bit Digital Signal Controllers with High-Speed PWM, Op Amps and Advanced Analog Features”. 2013-2014
9. Microchip. “Universal Asynchronous Receiver Transmitter (UART)”. 2009-2013

Ringraziamenti

Desidero ringraziare il professore Aldo Romani, relatore, il dottore Marco Crescentini, co-relatore e la dottoressa Giulia Luciani per la loro pazienza e disponibilità che hanno dimostrato durante il periodo di lavoro.
Ringrazio il collega Fabio che spesso mi ha fornito materiale di studio.
Mille grazie alla mia famiglia. A mio fratello Christian che giudica i fatti da punti di vista alternativi, a mio padre Stefano che mi ha trasmesso la passione per il mondo della informatica e elettronica e mia madre Loretta che mi ha pazientemente sopportato.
Grazie alla mia compagna di vita e amica Sylvia, con cui ho condiviso questi tre anni, non privi di ansia e a volte frustrazione ma che in fondo hanno donato grande soddisfazione.