BELLINZONA SAM

Anno scolastico 2017-18
Lavoro Professionale Individuale

Nome Cognome:	**Naoki Pross**
Professione:	**Elettronico**
Titolo del progetto:	**Spectrum Analyzer**

Azienda:	**CPT Bellinzona** Centro Professionale Tecnico Viale S. Franscini 25 6500 Bellinzona		
Formazione approfondita:	**S.2 Sviluppare prototipi**		
Formatore:	**Rinaldo Geiler, Daniele Kamm**		
Data d'inizio:	**12.04.2018**	Ore a disposizione:	**83 UD**
Data file lavoro:	**15.05.2018**	Ore effettive:	**– UD**

Indice

1 Introduzione

1.1 Contesto

Per portare a termine il percorso formativo per un attestato di capacità federale presso la Scuola Arti e Mestieri di Bellinzona è richiesto lo sviluppo individuale di un progetto di produzione di un prodotto. Per interesse personale nella matematica della trasformata di Fourier mi è stato assegnato di sviluppare un analizzatore spettrale.

1.2 Requisiti

È richiesto di sviluppare circuito per analizzare lo spettro dei segnali di frequenza fino a 10 kHz. Il dispositivo dovrà avere 3 possibili sorgenti: RCA/Cinch e 2 Audio Jack per un microfono e per una sorgente di audio generica. È inoltre richiesto che il calcolo dei dati dello spettrogramma sia eseguito da un microcontroller della Microchip, collegato a due altri dispositivi quali, una display e ad un computer in RS232, per poter visualizzare lo spettrogramma computato.

1.3 Concetti matematici

Il circuito realizzato si appoggia sul concetto matematico di importanza fondamentale nelle discipline come la fisica e l'elettrotecnica della *Trasformata di Fourier*. Questa operazione matematica è fondata su su un principio dimostrato da Joseph Fourier che asserisce che è possibile rappresentare una qualsiasi funzione periodica, in alcuni casi anche non periodica, con una serie di sinusoidi di frequenze multiple ad una di base. L'operazione di *Trasformata* dunque è uno strumento per osservare le frequenze di queste armoniche, esso trasforma una funzione in funzione del tempo $f(t)$ in una funzione rispetto alla frequenza o alla pulsazione $\hat{f}(\omega)$.

Secondariamente, il progetto usufruisce anche di un altro strumento chiamato *Fast Fourier Transform* (FFT) scoperto inizialmente nel 1965 dai matematici J. Cooley e J. Tukey. La FFT è un algoritmo con molte implementazioni che riduce la complessità computazionale della trasformata di fourier discreta da $\mathcal{O}(n^2)$ a $\mathcal{O}(n \log n)$. Questo è necessario perchè le operazioni matematiche da eseguire sono dei prodotti tra numeri complessi, i quali causerebbero dei severi cali di performance.

Tutti i concetti descritti saranno approfonditi nei capitoli seguenti.

1.4 Norme di progetto

2 Hardware

2.1 Schema a blocchi

2.2 Selezione delle entrate

Essendo richiesta dai requisiti la possibilità di selezione tra 3 entrate, è stato utilizzando un semplice multiplexer controllato direttamente dal microcontroller. Per la sua semplicità non sono necessari particolari osservazioni.

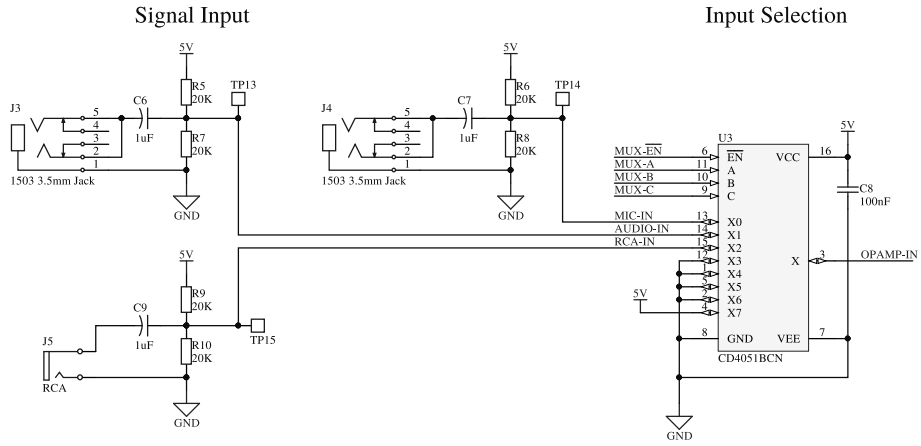

Figura 2.1: Circuito di selezione delle entrate

Tutte le entrate dispongono di un condensatore di disaccoppiamento seguito da un partitore di tensione simmetrico per aggiungere un offset della metà dell'alimentazione. Il valore delle resistenze di 20 kΩ è scelto per avere un impedenza rispetto al connettore pari all'impedenza caratteristica dei cavi audio di 10 kΩ.

2.3 Circuito di entrata

Il segnale di cui si analizza lo spettro, prima di essere campionato, viene adattato mediate un circuito di amplificazione e filtraggio. Esso è necessario per due ragioni. Il circuito di amplificazione è presente per ovvie ragioni, quali per poter adattare il circuito nel caso in cui si dovesse avere in entrata un segnale di ampiezza molto piccola. Il secondo circuito invece, di filtraggio, è necessario per rimuovere disturbi di alta frequenza che potrebbero compromettere la qualità del campionamento. Questa è una tipica configurazione prima di un circuito di conversione AD (analogico - digitale), ed è conosciuto anche come circuito di filtraggio *anti-alias*.

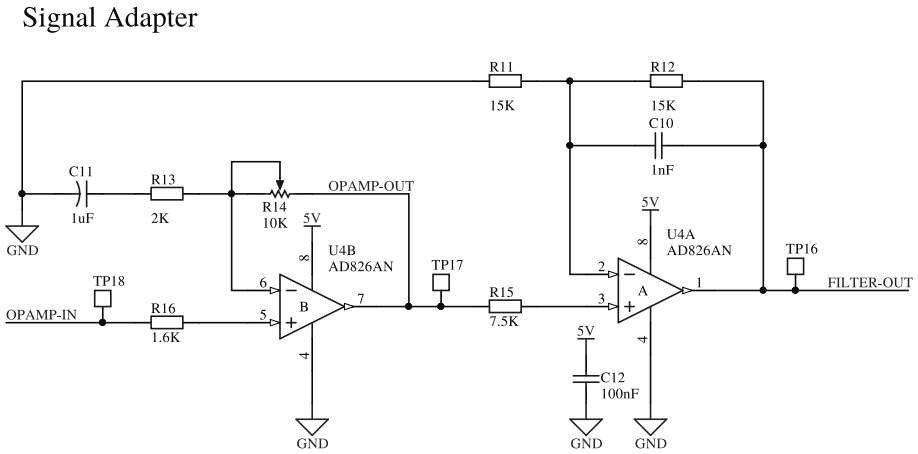

Figura 2.2: Circuito di adattamento del segnale in entrata

È importante notare che per questa applicazione si è scelto utilizzare degli opamp *rail to rail*, che hanno una tensione di saturazione vicina a quella di alimentazione. Essi sono necessari per poter raggiungere tensioni vicino allo 0 V, che non sarebbero possibili con un opamp normale siccome l'alimentazione del circuito è asimmetrica tra 0 e 12 V.

Amplificatore. Come si può notare a sinistra nella figura 2.2, il circuito di amplificazione non ha una configurazione tipica. Esso è basato su una configurazione non invertente ma dispone di un consensatore (C11) che modifica la retroazione in modo da reagire unicamente alla componente AC del segnale. Questo permette di amplificare la componente alternata ignorando l'offset del segnale, perciò di *non* utilizzare un'alimentazione simmetrica ± 5 V.

L'amplificazione di questo amplificatore è comunque data dal rapporto $1 + R14/R13$ che permette un un guadagno fino a 6 oppure 15,5 dB.

Filtro. A destra della figura 2.2 vi è il circuito di filtraggio, realizzato utilizzando un tipico filtro passa basso attivo di primo ordine. Esso è dimensionato con una frequenza di taglio di 10 kHz poichè quest'ultimo è il limite di Nyquist, conosciuto anche dal teorema di Shannon, il quale stata che la frequenza di campionamento deve essere almeno doppia della frequenza dell'armonica di frequenza maggiore.

2.4 Microcontroller

2.5 Visualizzazione

3 Software

3.1 Campionamento

FIGURE 13-1: TIMER2/4/6 BLOCK DIAGRAM

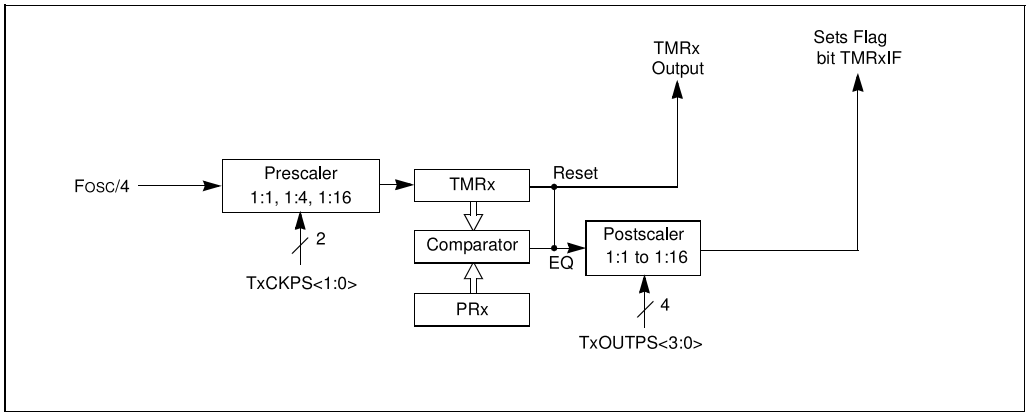

Figura 3.1: Schema a blocchi del TIMER2. Fonte: Microchip PIC18F2X/4XK22 datasheet

3.2 Trasferimento dei dati

Per trasferire i dati campionati era inizialmente stato scelto di utilizzare una struttura dati binaria. In seguito però per

3.3 Interfaccia al Computer

Per preferenze principalmente personali è stato scelto di realizzare l'interfaccia al computer utilizzando il moderno linguaggio di programmazione C++ (versione ≥ 11) utilizzando un framework (Qt) che sarà descritto successivamente. I vantaggi dati dall'utilizzo del C++ anzichè linguaggi interpretati come il Python, linguaggi compilati in bytecode come Java, o con runtime particolari come LabView / CVI sono molteplici. Innanzitutto tutti gli strumenti necessari per lo sviluppo hanno mezzi o varianti *open source* / *libre*, di conseguenza gratuiti e in molti casi multipiattaforma. Al contrario invece dei sistemi proprietari come quelli offerti da National Instruments che sono estremamente costosi e possono essere utilizzati unicamente sulle piattaforme con un supporto ufficiale. Tra i vari linguaggi di programmazione non proprietari il C++ è comunque in posizione di vantaggio siccome è tra i più performanti in quanto non richiede alcun interpreter (bytecode o non) o nessuna runtime, stando quindi alla pari con il linguaggio C guadagnando però i vantaggi dell'astrazione data dalla programmazione ad oggetti.

3.3.1 Librerie e codice di terzi

Per ridurre i tempi dedicati alla realizzazione del programma, seppur mantenendo una buona qualità, è stato scelto di utilizzare le seguenti librerie.

- Serial (`http://wjwwood.io/serial/`): Utilizzata come interfaccia multipiattaforma per l'accesso di basso livello alla porta seriale del sistema operativo sottostatante. Descrizione dal sito:

> This is a cross-platform library for interfacing with rs-232 serial like ports written in C++. It provides a modern C++ interface with a workflow designed to look and feel like PySerial, but with the speed and control provided by C++.

- QCustomPlot (http://qcustomplot.com): Utilizzata per produrre il grafico all'interno del software, per visualizzare i dati dal microcontroller. Descrizione dal sito:

  > QCustomPlot is a Qt C++ widget for plotting and data visualization. It has no further dependencies and is well documented. This plotting library focuses on making good looking, publication quality 2D plots, graphs and charts, as well as offering high performance for realtime visualization applications.

3.3.2 Qt Framework

La dipendenza principale utilizzata per realizzare la grafica è il framework di Qt. Oggi Qt è un'azienda indipendente che vende un supporto commerciale per lo sviluppo di applicazioni su praticamente ogni piattaforma. Qt è un framework maturo che esiste oramai da 22 anni ed è disponibile sia con una licenza commerciale che con le licenze open source LGPL e GPL.

La toolchain di Qt aggiunge al normale sviluppo un preprocessore speciale chiamato MOC che genera in automatico il codice dalle strutture grafiche realizzate con QtDesigner. Il resto della toolchain è composta da componenti tipici che possono essere intercambiati liberamente poichè Qt supporta gcc/g++, clang, MSVC e MinGW. Per compilare il codice è dunque necessario un compiler qualsiasi di C++ e l'IDE QtCreator, oppure qmake. Pochè questi pacchetti offrono il preprocessore MOC. Per progetti open source entrambi sono offerti gratuitamente dal sito ufficiale www.qt.io.

3.3.3 Compilazione sotto Linux

Il programma è stato realizzato in parte sotto Debian 9.4 Stretch ed in parte sotto Fedora 27. Per entrambi i sistemi sono necessarie le dipendenze per lo sviluppo in Qt5.

Una volta installate le dipendenze dalla cartella di progetto è possibile utilizzare il makefile per compilare le dipendenze e il codice.

```
$ make build-deps   # dependencies
$ make              # spectrum analyzer code
```

Purtroppo la liberia QCustomPlot utilizza un sistema di build molto particolare che richiede molte dipendenze. Perciò in alcuni casi è preferibile scaricare dal seguente link l'ultima versione dei due files `qcustomplot.cpp` e `qcustomplot.hpp` ed immetterli manualmente nella cartella `lib/qcustomplot/`.

http://qcustomplot.com/index.php/download

Ed infine si compila con

```
$ make serial       # build only Serial library dep
$ make              # build spectrum analyzer code
```

3.3.4 Compilazione manuale sotto Linux

Per compilare manualmente il progetto sono necessari pochi steps grazie a qmake. Come per il caso precedente la libreria QCustomPlot può essere scaricata dal sito.

1. Scaricare le dipendenze.

```
$ git submodule init
$ git submodule update
```

2. Compilare la libreria Serial

```
$ mkdir -p lib/build-serial
$ qmake -makefile -o Makefile -Wall "CONFIG+=releae" \
    -o lib/build-serial/ lib/serial

$ make -C lib/build-serial/
```

3. Compilare la libreria QCustomPlot

```
$ cd lib/qcustomplot/src
$ sed -i -e 's/qmake474/qmake/' release.py
$ ./release.py
$ cd ../.. # go back to project root dir
```

4. Compilare il progetto

```
$ mkdir -p build-deskop
$ qmake -makefile -o Makefile -Wall "CONFIG+=release" \
    -o build-desktop/ src-desktop

$ make -C build-desktop
```

3.3.5 Compilazione sotto Windows

Per compilare il progetto in Windows è necessario installare QtCreator dal sito ufficiale www.qt.io.

1. Installare QtCreator, Qt $\geq$ 5.0 (consigliato 5.10.0)
2. Installare MinGW oppure MSVC + Visual Studio (consigliato MinGW)
3. Inizializzare ed aggiornare i submoduli di Git oppure clonare recursivamente il progetto.
4. Scaricare dal sito http://qcustomplot.com/index.php/download i documenti `qcustomplot.cpp` `qcustomplot.hpp` della libreria QCustomPlots ed immetterli nella cartella `lib/qcustomplots/`
5. Aprire il progetto `lib/serial/serial.pro` ed impostare la build directory sia per release che per debug in `lib/build-serial`.
6. Compilare la libreria Serial come release.
7. Aprire il progetto `src-desktop/SpectrumAnalyzer.pro` ed impostare la cartella di build sia per release che debug nella cartella `build-desktop/`
8. Compilare il progetto SpectrumAnalyzer come release
9. Controllare che lo script `deploy-desktop.cmd` abbia le variabili `QT_PATH` e `QT_VERSION` che corrispondano con la vostra l'installazione.
10. Eseguire lo script `deploy-desktop.cmd`.

Nella cartella `build-desktop` sarà pronto l'eseguibile con tutte le librerie dinamiche (DLL) necessarie.

3.3.6 Architettura

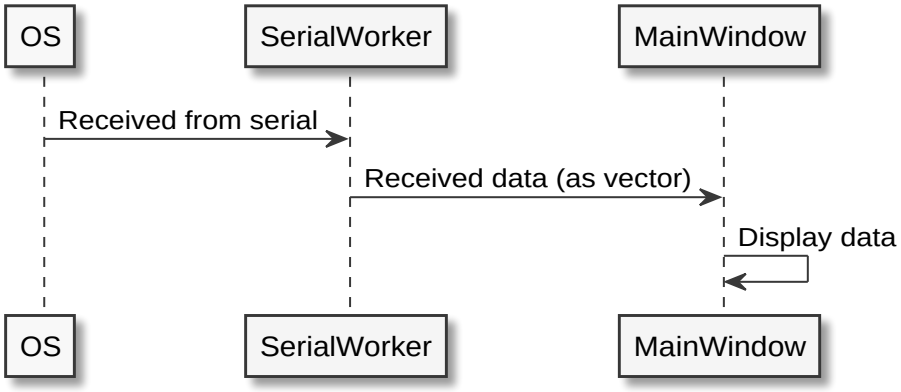

Figura 3.2: Diagramma delle sequenze

3.4 Interfaccia al Display

3.5 Fast Fourier Transform

4 Problemi riscontrati / Bugs

Bibliografia

[1] *Example item title*, (online), Author and other informations,
https://www.example.com

5 Trasformata di Fourier

5.1 Nozioni preliminarie

5.1.1 Regressione lineare con il metodo dei minimi quadrati

La regressione lineare è un'approssimazione di una serie di dati ad una funzione lineare. Questa retta di approssimazione può essere calcolata in molteplici modi, per questo progetto è di interesse utilizzare il *metodo dei minimi quadrati*. Sarà dunque esplicato come trovare i coefficienti di una retta a $m+1$ termini partendo da N punti di riferimento.

$$r(x, a_0, \ldots, a_m) = a_0 + x \sum_{i=1}^{m} a_i \tag{5.1.1.1}$$

Consideriamo di avere gli insiemi X e Y entrambi con N termini di cui si prende le coppie ordinate di valori (x_k, y_k) $x_k \in X$, $y_k \in Y$, ossia i punti dato di cui eseguire la regressione. Il metodo dei minimi quadrati trova i coefficienti della retta minimizzando il quadrato della differenza tra il valore stimato dalla retta $r(x_k)$ e il valore reale y_k.

$$\min((r(x_k) - y_k)^2) \quad \forall x_k \in X,\, y_k \in Y$$

Definiamo quindi la funzione da minimizzare ε

$$\varepsilon(a_0, \ldots, a_m) = \sum_{k=1}^{N} \Big[r(x_k, a_0, \ldots, a_m) - y_k \Big]^2 \tag{5.1.1.2}$$

Da cui si computa le derivati parziali rispetto ai coefficienti ricercati, ottenendo un sistema di equazioni lineare. Ciò corrisponde anche ad affermare che il *gradiente* di ε è un vettore $\in \mathbb{R}^{m+1}$ con tutte le componenti a 0.

$$\nabla\varepsilon = \langle 0, \ldots, 0 \rangle$$

A questo punto si può procedere risolvendo il sistema con l'algebra lineare definendo la matrice di trasformazione $\mathbf{A}$ e il vettore dei termini noti $\vec{u}$

$$\nabla\varepsilon = \mathbf{A}\langle a_0, \ldots, a_m \rangle + \vec{u} \iff \langle a_0, \ldots, a_m \rangle = \mathbf{A}^{-1}(-\vec{u})$$

5.1.2 Funzione armonica

Una funzione armonica, sinusoidale, può essere descritta in molteplici modi. Iniziamo dunque osservando le forme più semplici, ossia la forma trigonometrica.

$$f(x) = a \cdot \sin(\omega x + \varphi) \tag{5.1.2.1}$$
$$f(x) = b \cdot \cos(\omega x + \vartheta) \tag{5.1.2.2}$$

Conoscendo la formula di Eulero (**??**)

$$e^{i\varphi} = \cos(\varphi) + i \cdot \sin(\varphi) \tag{5.1.2.3}$$

possiamo riscrivere $f(x)$ nei seguenti modi

$$f(x) = \frac{a}{2i} \cdot (e^{i(x\omega+\varphi)} - e^{-i(x\omega+\varphi)}) \tag{5.1.2.4}$$
$$f(x) = \frac{b}{2} \cdot (e^{i(x\omega+\vartheta)} + e^{-i(x\omega+\vartheta)}) \tag{5.1.2.5}$$

5.1.3 Proprietà di ortogonalità del seno e del coseno

Per avere delle fondamenta solide prima dell'introduzione dell'argomento principale, saranno dimostrate le proprietà di ortogonalità del seno e coseno. Considerando il periodo T, dunque di frequenza $2\pi/T$.

Intuizione geometrica

Dimostrzioni algebriche

1.

$$\int_0^T \sin(\frac{m2\pi x}{T})\,\mathrm{d}x = 0 \quad \forall m \in \mathbb{Z}$$

$$\begin{aligned}\int_0^T \sin(\frac{m2\pi x}{T})\,\mathrm{d}x &= \Big[-\frac{T}{2\pi m} \cdot \cos\big(\frac{2\pi}{T} m x\big)\Big]_0^T \\ &= -\frac{T}{2\pi m} \cdot \cos\big(2\pi m\big) + \frac{T}{2\pi m} \cdot \cos\big(0\big) \\ &= 0\end{aligned}$$

2.

$$\int_0^T \cos(\frac{m2\pi x}{T})\,\mathrm{d}x = 0 \quad \forall m \in \mathbb{Z}^*$$

$$\begin{aligned}\int_0^T \cos(\frac{m2\pi x}{T})\,\mathrm{d}x &= \Big[\frac{T}{2\pi m} \cdot \sin\big(\frac{2\pi}{T} m x\big)\Big]_0^T \\ &= \frac{T}{2\pi m} \cdot \sin\big(2\pi m\big) + \frac{T}{2\pi m} \cdot \sin\big(0\big) \\ &= 0\end{aligned}$$

Nota: Se $m = 0$

$$\int_0^T \cos(\frac{m2\pi x}{T})\,\mathrm{d}x = T$$

3.

5.2 Polinomio Trigonometrico

Analogamente a come è definito un polinomio P "normale" di grado N, è possibile definire anche un polinomio trigonometrico T.

$$P_N(x) = \sum_{n=0}^{N} a_n x^n \qquad a_n \in \mathbb{R},\ a_N \neq 0$$

$$T_N(x) = \sum_{n=0}^{N} c_n e^{i\omega n x} \qquad c_n \in \mathbb{C},\ \omega \in \mathbb{R},\ c_N \neq 0$$

Questo polinomio è detto *trigonometrico* perchè utilizzando la formula di eulero $e^{i\varphi} = \cos(\varphi) + i\sin(\varphi)$ si può espandere nel seguente modo.

$$T_N(x) = \sum_{n=0}^{N} \left[a_n \cdot \cos(\omega n x) + i b_n \cdot \sin(\omega n x)\right] \qquad a_n, b_n \in \mathbb{C}$$

È definito inoltre il polinomio trogonometrico *reale* come

$$T_N(x) = \sum_{n=0}^{N} \left[a_n \cdot \cos(\omega n x) + b_n \cdot \sin(\omega n x)\right] \qquad a_n, b_n \in \mathbb{R}$$

Quest'ultimo mediante delle identità trigonometriche può essere riscritto anche nel modo seguente.

$$T_N(x) = \sum_{n=0}^{N} A_n \cdot \cos(\omega n x - \varphi)$$

In tutti i casi possiamo osservare che il polinomio trogonometrico è una somma di sinusoidi di frequenze multiple ad una base $\omega = 2\pi f$. Se descritto mediante la terminologia dell'algebra lineare, si può anche osservare che un polinomio trigonometrico è una combinazione lineare nello spazio funzionale ortonormato dalle basi $\sin(\omega n x)$ e $\cos(\omega n x)$.

5.3 Serie di Fourier

5.4 Trasformata di Fourier discreta

5.5 Trasformata di Fourier

5.6 Fast Fourier Transform

5.6.1 Motivazioni e Complessità temporale

5.6.2 Proprietà dei numeri complessi