UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA TRIENNALE IN INGEGNERIA ELETTRONICA

Progetto e implementazione di un equalizzatore audio con microcontrollore STM32F334R8

Relatore:
Prof. Simone Buso

Laureando:
Marco Longo
1219602

Anno Accademico 2021/2022

Data di laurea: 22/09/2022

A te papà,

che, oltre a mamma, saresti il più felice di tutti

nel vedere i tuoi figli laureati.

Indice

Introduzione

In questo documento verrà presentato il progetto di un equalizzatore digitale audio a due bande e due canali realizzato con il microcontrollore *STM32F334R8*. L'idea di questo progetto nasce dalla volontà di approfondire l'affascinante argomento *Digital Signal Processing*, di cui è stata fatta una breve introduzione durante il corso di *Elettronica Industriale*.

In accordo con il Professor Buso è stato quindi deciso di approfondire queste conoscenze per la realizzazione di un semplice equalizzatore audio.

L'idea del progetto è la seguente: utilizzare il μC usato durante il laboratorio del corso per equalizzare un file audio in uscita dallo smartphone. Le fasi del lavoro sono state:

- misurazione del segnale in uscita del cellulare e realizzazione di un circuito di condizionamento per l'ADC del μC;
- studio, simulazione e realizzazione dei filtri digitali necessari;
- realizzazione di un filtro ricostruttore in uscita al DAC e di uno stadio di amplificazione per riprodurre il suono tramite due diffusori (canali destro e sinistro).

Ognuno di questi punti verrà approfondito in un capitolo di questa tesi, preceduti da una breve introduzione dell'*STM32F334R8* e delle periferiche utilizzate per la realizzazione del prototipo, di cui si può vedere uno schema di principio in figura 0.1.

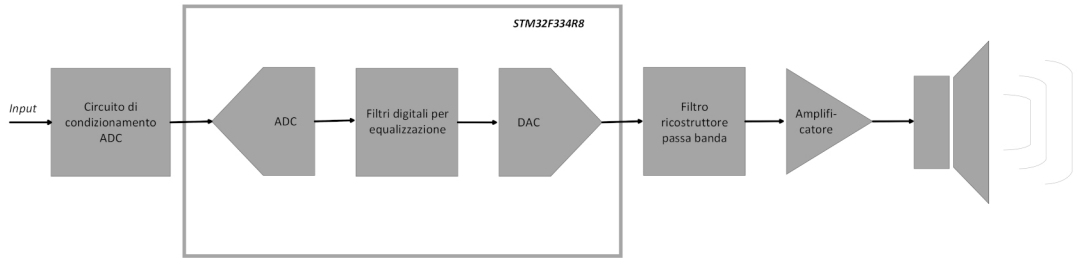


Figura 0.1: Schema a blocchi del progetto di un singolo canale audio; l'altro, poichè uguale, è stato omesso.

Capitolo 1

Scheda di sviluppo e periferiche utilizzate

La scheda *Nucleo F334-R8*, o più semplicemente Nucleo334 (visibile in figura 1.1), di *STMicroelectronics* è una scheda che ospita varie risorse a disposizione del programmatore. Partendo dal microcontrollore, un *STM32F334R8* basato su un core ARM Cortex M4, sono poi disponibili circuiti che permettono alla board di alimentare circuiti esterni a $+5\ V$ e a $+3.3\ V$, due LED (rosso e verde) e due pulsanti (*User* e *Reset*). Inoltre è disponibile nella parte alta della scheda una sezione dedicata alla comunicazione con il PC che permette la programmazione direttamente da computer, nonchè di svolgere funzioni di de-bugging e tracing del codice durante l'esecuzione dello stesso.

Microcontrollore STM32F334R8

Il microcontrollore *STM32F334R8* ha come principali caratteristiche:

- core Cortex M4;
- aritmetica a $32\ bit$ con unità floating point;
- frequenza di clock di $72\ MHz$;
- fino a $64\ kbyte$ di memoria flash;
- fino a $12\ kbyte$ memoria SRAM;
- fino a 51 pin I/O.

Per questo progetto verranno inoltre utilizzati:

- due ADC a canale singolo in modalità *injected*, con risoluzione di $12\ bit$ e ampiezza compresa tra 0 e $3.6\ V$;

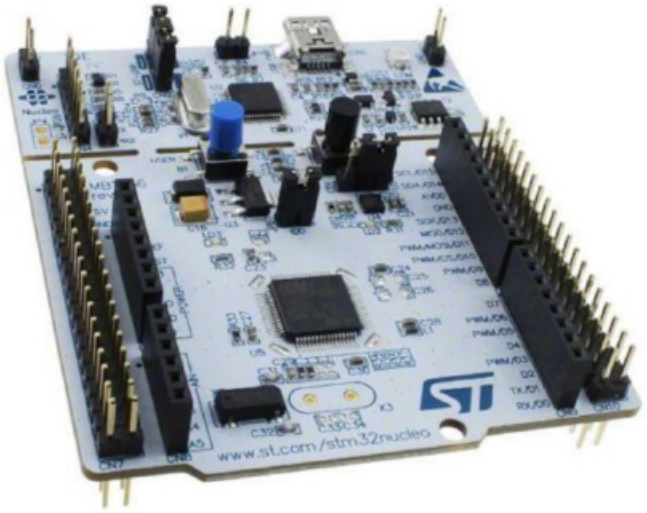


Figura 1.1: Scheda di valutazione *Nucleo F334-R8* di STM.

- due DAC a canale singolo anch'essi con risoluzione di $12\ bit$ e ampiezza compresa tra 0 e $3.6\ V$;
- *TIM3*, un timer general-purpose a $16\ bit$, in modalità free-run.

L'idea alla base del programma è utilizzare il *TIM3* per generare ogni $25\ \mu s$ il segnale di trigger sia per gli ADC che per i DAC, quindi iniziare la conversione di un nuovo dato e scrivere quello calcolato precedentemente in uscita al convertitore digitale-analogico. Quando gli ADC hanno finito la conversione, viene inviata una richiesta di interrupt alla CPU nella quale sono presenti tutti gli algoritmi di Digital Signal Processing[1]. Questa scelta implica che il tempo limite per implementare tutti i filtri è di appunto $25\ \mu s$, corrispondente ad una frequenza di campionamento di $40\ kHz$[2], che si traduce nell'avere a disposizione 1800 periodi di clock tra una IRQ (Interrupt Request) e la successiva. Come si vedrà, grazie all'unità floating point questo non si rivelerà un problema, ma è una considerazione che può essere fatta solo a posteriori poichè, programmando il μC in un linguaggio ad alto livello come il C, il numero di istruzioni macchina dipende dal compilatore usato.

Nel caso si volesse alzare la frequenza di campionamento significativamente, ad esempio a 96 kHz, bisognerebbe ricorrere a soluzioni più efficienti in termini computazionali, ma anche più complesse da gestire, come l'utilizzo del DMA disponibile sul DSC utilizzato.

Il programma principale, interno al main, consisterà semplicemente in un controllo sulla pres-

[1] Ovvero il codice che realizza i vari filtri.

[2] Un segnale audio è compreso tra circa i 16 e i 20000 Hz, per il teorema di Shannon è quindi la frequenza minima di campionamento per un segnale audio.

sione del pulsante *User* (tramite un opportuno codice che implementa un de-bouncer software), che una volta premuto imposterà dei valori prefissati per il guadagno di ciascuna banda e di ciascun canale, contenuti in un buffer circolare.

Convertitore analogico/digitale

Come detto in precedenza, sono presenti due convertitori A/D a $12\ bit$ di tipo *SAR* (ad approssimazioni successive) i quali sono configurabili per funzionare sia in maniera indipendente che in modalità master-slave, con un tempo di conversione nel peggiore dei casi[3] di $210\ ns$.

Nell'applicazione qui trattata vengono utilizzati in *simultaneous injected mode*; dal nome si può evincere che i due ADC acquisiranno *contemporaneamente* i dati, cosa che non sarebbe stata possibile sfruttando più canali (ne sono disponibili 18) di un unico convertitore. I vari canali possono essere configurati come *regular* o *injected*. *Regular* implica che i canali che acquisicano segnali secondo una frequenza regolare, ma non dipendente da eventi interni o esterni al microcontrollore che impongano una rigida sincronizzazione. *Injected*, invece, impone che i segnali vengano acquisiti in corrispondenza di un segnale di trigger (in questa applicazione generato dal *TIM3*) e quindi viene ridotta la possibilità che si manifesti del *jitter*.

Adesso dovrebbe essere più chiaro il perchè è stata scelta la modalità di funzionamento sopracitata.

Convertitore digitale/analogico

A bordo del microcontrollore sono disponibili due convertitori D/A a $12\ bit$. Anche in questo caso sono presenti più canali (due per il DAC1 e uno per il DAC2) ma, per le stesse considerazioni fatte per il convertitore A/D, verranno utilizzate entrambe le periferiche con un solo canale.

La modalità di funzionamento scelta è *triggered output*, ovvero l'uscita viene aggiornata quando riceve un segnale di trigger che, nel caso in analisi, è quello generato dal *TIM3*.

Timer

Poichè l'*STM32F334R8* è un dispositivo orientato al controllo di convertitori switching e agli azionamenti di motori elettrici, sono messi a disposizione molti moduli timer (tra cui uno ad alta risoluzione capace di risolvere eventi ad una distanza temporale fino a $217\ ps$).

[3] Dipende se il canale scelto è *fast* o *slow*.

Dato che una frequenza di campionamento di $40\ kHz$ è relativamente modesta, non è necessario ricorrere all'utilizzo di timer particolarmente prestanti, indi per cui la caratteristica fondamentale per scegliere quale usare è che il suo segnale di trigger interno (*TRGO*) possa essere collegato a tutte le periferiche menzionate fino ad ora. Questo restringe la scelta al solo *TIM3*, che è un timer general purpose a $16\ bit$, con la possibilità di contare in maniera crescente o decrescente e 4 canali a sua disposizione. La modalità di funzionamento è free-run dato che deve portare alto *TRGO* ad intervalli di tempo regolari.

Il valore da impostare nel registro di match, sapendo che la frequenza di clock è di $72\ MHz$ e che si vogliono acquisire i dati ad una frequenza di $40\ kHz$, è:

$$n_{match} = \frac{f_{clock}}{f_{IRQ}} = \frac{72[MHz]}{40[kHz]} = 1800; \tag{1.1}$$

numero sicuramente contenuto in un registro a $16\ bit$ ($1800 < 2^{16} - 1$), rendendo superfluo l'ultilizzo di un prescaler.

Capitolo 2

Circuito di interfaccia con l'ADC

Dopo aver deciso quali periferiche utilizzare e in che modalità, bisogna realizzare un circuito analogico che mappi il segnale in uscita dallo smartphone nella scala accettata dal convertitore analogico-digitale che, come detto nel capitolo 1, è compresa tra 0 e $3.6\ V$.

Il primo passo è quindi quello di misurare la tensione dell'uscita jack. Lo smartphone preso in esame è un iPhone SE del 2016. Sperimentalmente, attraverso l'ausilio di un'applicazione che permette di generare segnali sinusoidali, quadri, a dente di sega e triangolari tra una frequenza compresa tra 0 e $25000Hz$, si trova che la tensione di uscita (che da adesso in poi verrà chiamata v_{in}) sarà $v_{in} \in [-0.750, 0.750][V]$; bisogna quindi realizzare un circuito che amplifichi tale grandezza e aggiunga un offset.

La soluzione adottata è quella in figura 2.1.

Il circuito consiste in un partitore resistivo disaccoppiato tramite un inseguitore di tensione che

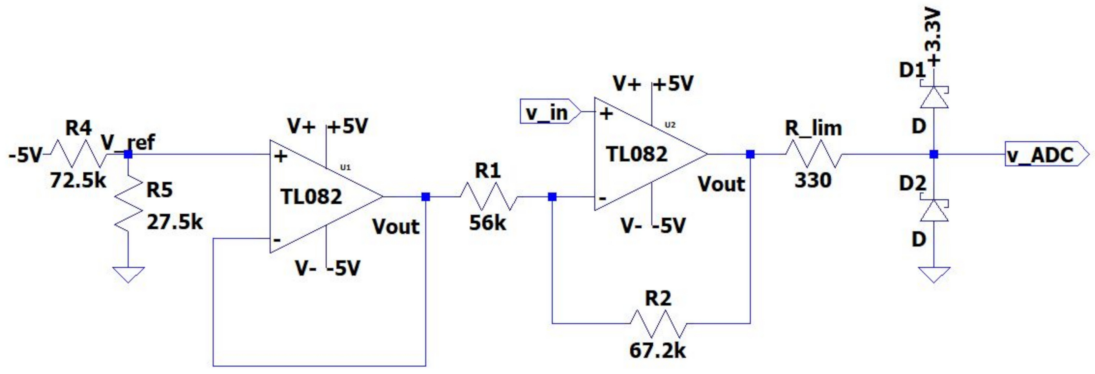


Figura 2.1: Schema del circuito realizzato per mappare il segnale proveniente dal jack audio nel range di valori accettato dall'ADC dell'*STM32F334R8*.

realizza l'offset dello stadio successivo, che consiste in un semplice operazionale in configurazione *non invertente*.

L'OPAMP scelto è il *TL082* che, con una banda di $3\ MHz$ e un *THD* (Total Harmonic Distortion) dello 0.003%, rappresenta un'ottima scelta per applicazioni audio come quella trattata. Nella pratica il partitore resistivo composto da R_4 e R_5 è stato realizzato con un trimmer da $100\ k\Omega$ e, per comodità, anche R_2 è una resistenza variabile. In questa maniera, se si dovesse cambiare la sorgente con una che genera tensioni maggiori o minori, si potrà operare sui trimmer per adattare il segnale ed evitare quindi di incorrere in errori di saturazione o perdita di risoluzione.

Da una rapida analisi si ottiene:

$$v_{ADC} = v_{in} \cdot \left(1 + \frac{R_2}{R_1}\right) + v_{ref} \cdot \frac{R_2}{R_1}; \tag{2.1}$$

sapendo che quando v_{in} è massima l'uscita deve essere di $3.3\ V$[1], si impone:

$$v_{in} \cdot \left(1 + \frac{R_2}{R_1}\right) = \frac{v_{ADC}}{2}; \tag{2.2}$$

$$v_{ref} \cdot \frac{R_2}{R_1} = \frac{v_{ADC}}{2}. \tag{2.3}$$

Il valore del rapporto tra R_2 e R_1, si ottiene dalla 2.2 ponendo $v_{ADC} = 3.3\ V$ e $v_{in} = 0.750\ V$ e dopo semplici passaggi algebrici si trova:

$$\frac{R_2}{R_1} = \frac{6}{5}; \tag{2.4}$$

ponendo quindi $R_1 = 56\ k\Omega$ si trova il valore:

$$R_2 = 56 \cdot \frac{6}{5} = 67.2[k\Omega]. \tag{2.5}$$

Unendo la 2.3 e la 2.4 si trova anche:

$$v_{ref} = \frac{v_{ADC}}{2} \cdot \frac{R_1}{R_2} = -1.375[V]; \tag{2.6}$$

valore ottenibile dall'alimentazione negativa attraverso R_4 e R_5. Per quanto detto prima:

$$R_4 + R_5 = 100[k\Omega] \tag{2.7}$$

$$v_{ref} = -v_{cc} \cdot \frac{R_5}{R_4 + R_5} \tag{2.8}$$

[1] In realtà, come si è visto, il valore massimo accettato dal convertitore A/D è di $3.6\ V$. Conviene però lasciare $0.3\ V$ di scarto che verranno sommati nel caso il diodo schottky D_1 entri in conduzione a causa di sovratensioni o errori, proteggendo così il μC.

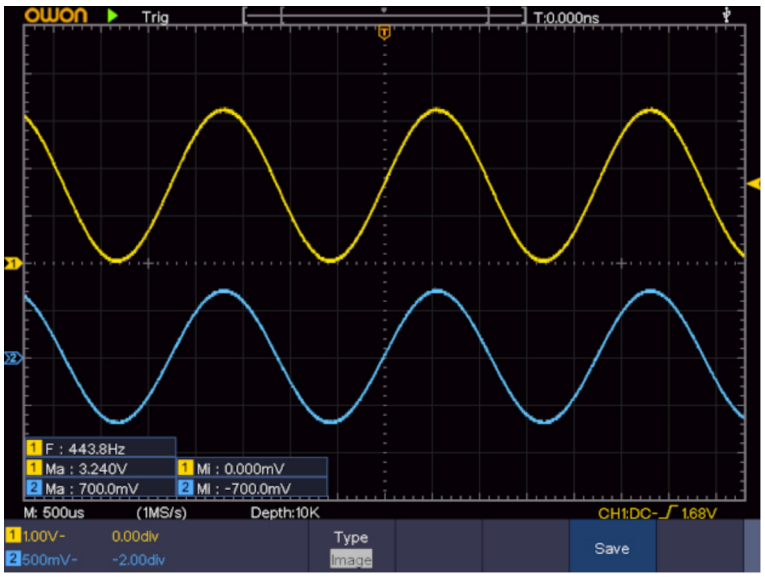


Figura 2.2: Acquisizione con l'oscilloscopio di v_{in} (in azzurro) e v_{ADC} (in giallo).

e, sapendo che l'alimentazione $-v_{cc} = -5\ V$, dopo semplici conti si ottengono $R_4 = 72.5\ k\Omega$ e $R_5 = 27.5\ k\Omega$.

Infine, l'ultima sezione che comprende R_{lim}, D_1 e D_2 è una semplice parte di protezione per l'ADC. I diodi schottky utilizzati sono dei *BAT48*, mentre la resistenza è dimensionata in maniera tale che la corrente che scorre nei diodi non causi la rottura dei componenti[2]. Inserendo la resistenza $R_{lim} = 330\ \Omega$ la corrente è limitata a $4.3\ mA$ nel caso in cui l'uscita superi i $3.3\ V$ (alimentazione ricavata dalla Nucleo334), mentre a circa $-14\ mA$ se quest'ultima fosse inferiore al livello di massa. Il funzionamento di questa parte è banale: se l'uscita dell'OPAMP è maggiore di $3.3\ V$ o inferiore di $0\ V$, il rispettivo diodo entra in conduzione ponendo l'uscita (ingresso del convertitore A/D) rispettivamente a circa $3.6\ V$ e $-0.4\ V$ senza danneggiare la scheda.

In figura 2.2 si può notare un'acquisizione all'oscilloscopio fatta del circuito appena analizzato (la sua realizzazione su breadboard è in figura 2.3), i risultati sono quelli attesi nei limiti di risoluzione dello strumento.

Alimentatore $\pm 5\ V$

A questo punto è doveroso aprire una breve parentesi sull'alimentazione fornita alle varie schede viste fino ad ora e per quelle che verranno analizzate in seguito.

[2]In realtà è una precauzione in più, dato che il *TL082* limita già la corrente massima a circa $26\ mA$.

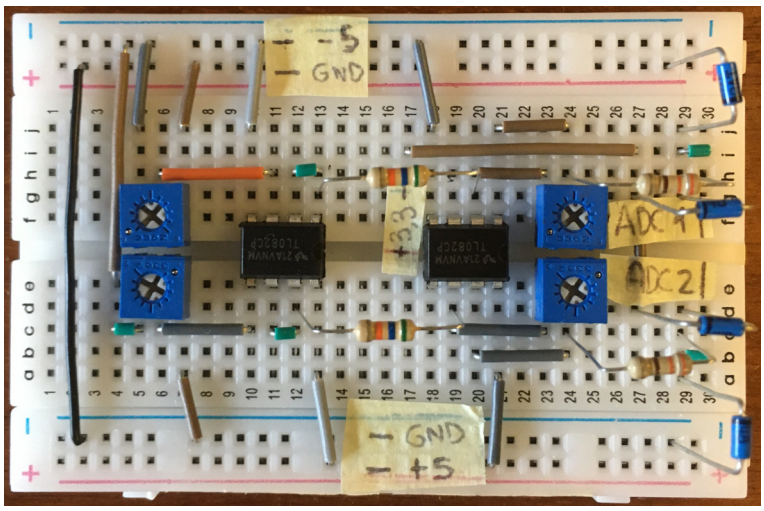


Figura 2.3: Realizzazione su breadboard del circuito di condizionamento di figura 2.1; si noti che si tratta di due circuiti identici, uno per il canale destro e l'altro per il sinistro.

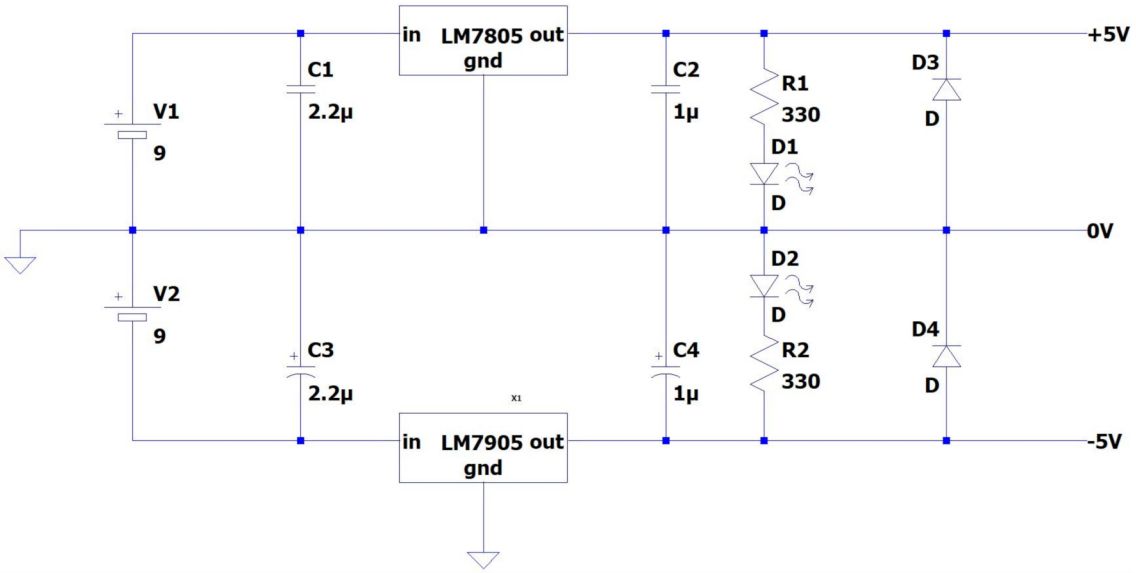


Figura 2.4: Schema circuitale dell'alimentatotre a $\pm 5\ V$.

La scheda Nucleo334 può essere alimentata tramite una presa USB mini direttamente dal PC e a sua volta può fornire una tensione di $5\ V$ o $3.3\ V$, come scritto nel capitolo 1. Dovendo maneggiare segnali audio con valori sia positivi che negativi è necessario avere un'alimentazione duale per gli amplificatori operazionali. Non avendo a disposizione un alimentatore da banco, si è resa necessaria la realizzazione di un alimentatore da $\pm 5\ V$, il cui schema si può vedere in figura 2.4, che è stato progettato a partire dalla sezione *typical applications* del datasheet dell'*LM7805*. L'analisi viene lasciata al lettore.

I livelli di tensione generati e una foto dell'implementazione su breadboard si possono apprezzare rispettivamente in figura 2.5 e 2.6.

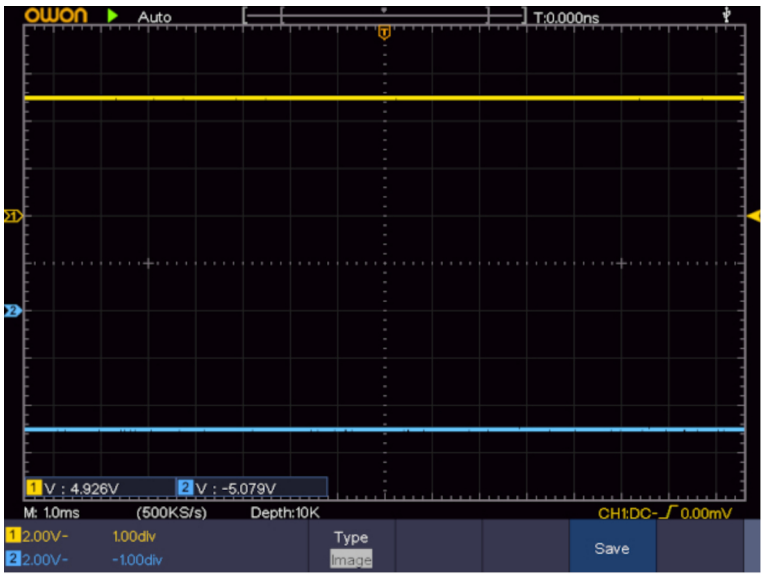


Figura 2.5: Acquisizione con l'oscilloscopio della tensione di uscita positiva (in giallo) e negativa (in azzurro).

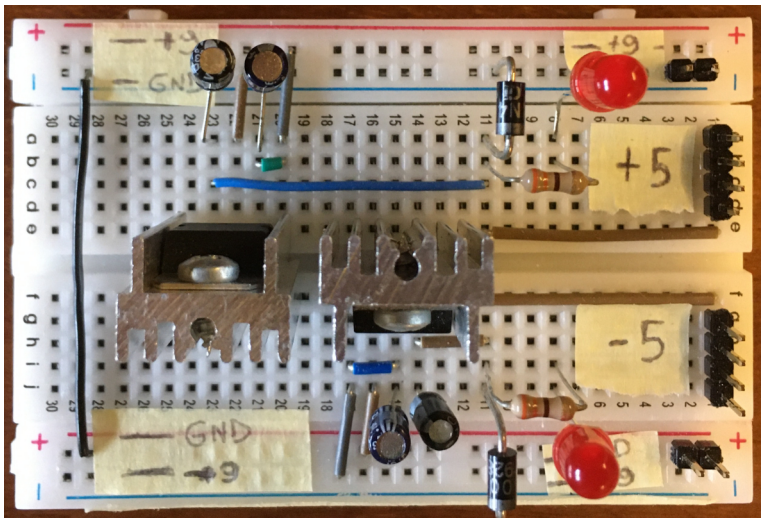


Figura 2.6: Realizzazione su breadboard del circuito di alimentazione di figura 2.4; si noti la simmetria del circuito con l'*LM7805* che genera la tensione a $+5\ V$ e l'*LM7905* che genera la tensione a $-5\ V$.

Capitolo 3

Progetto e implementazione dei filtri digitali

Il contenuto di questo capitolo rappresenta il cuore del progetto di tesi. Siccome quanto verrà qui trattato non è stato, per ovvie ragioni di tempo, svolto durante il corso di laurea, verrà prima presentata una parte teorica in cui si approfondiranno le nozioni di *Digital Signal Processing* necessarie a compredere i vari passaggi per il progetto di un filtro IIR del primo e del secondo ordine, inclusa la sua implementazione software. Verranno poi riportate le varie scelte progettuali e i calcoli fatti per ogni filtro, le simulazioni fatte con l'ausilio di MATLAB e i risultati ottenuti sperimentalmente. Infine, verrà anche aperta una breve parentesi sul funzionamento del programma principale (*main*).

3.1 Trasformata bilineare e forme dirette

Per poter eleaborare un segnale proveniente da una certa sorgente, il μC deve prima acquisirlo attraverso il convertitore A/D, ovvero deve *discretizzarlo* e *quantizzarlo*. Il segnale viene dunque trasformato da analogico a digitale (o numerico). Il segnale digitale consiste in una sequenza di valori appartenenti ad un insieme finito, acquisiti ad una distanza temporale pari al passo di campionamento T.

Un filtro è un algoritmo che, presa una sequenza $x(kT)$ in ingresso, restituisce in uscita una nuova sequenza $y(kT)$.

Un filtro numerico è sempre ricavabile da un suo corrispettivo analogico attraverso un processo di discretizzazione. Un filtro così ottenuto viene detto *IIR* (Infinite Impulse Response) poichè,

proprio come un filtro analogico, la sua risposta all'impulso si assesterà dopo un tempo infinito. Esistono anche i filtri *FIR* (Finite Impulse Response), senza un corrispettivo analogico, la cui risposta impulsiva, come si può dedurre dal nome, si assesterà dopo un tempo finito.

Il processo di dicretizzazione qui trattato è la *trasformata bilineare* o *di Tustin*.

Presa una f.d.t. (funzione di trasferimento) nel dominio di Laplace bisogna effettuare la seguente sostituzione:

$$s = \frac{2}{T} \cdot \frac{z-1}{z+1}. \tag{3.1}$$

Per capire intuitivamente il senso di questa operazione, si ricorda che ai fini di valutare la risposta in frequenza di una funzione di trasferimento si pone:

$$s = j\Omega; \tag{3.2}$$

dove $\Omega \in [0, \infty][rad/s]$ e rappresenta le "frequenze analogiche".

Ricordando dalla teoria della trasformata $\mathcal{Z}$ che:

$$z = e^{j\omega}; \tag{3.3}$$

e che questo vuole dire che le "frequenze digitali" normalizzate utili sono $\omega \in [0, \pi]$ e non estese fino all'infinito.

Sostituendo la 3.2 e la 3.3 nella 3.1 si ottiene:

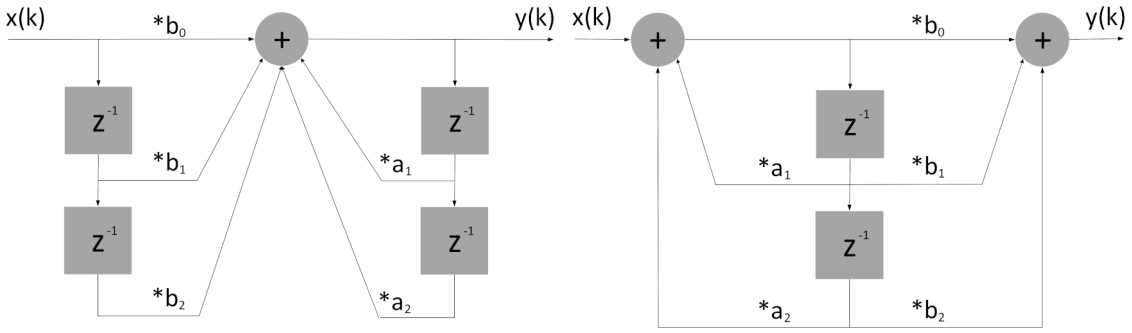
$$j\Omega = \frac{2}{T} \cdot \frac{e^{j\omega} - 1}{e^{j\omega} + 1}; \tag{3.4}$$

$$j\Omega = \frac{2}{T} \cdot j \tan \frac{\omega}{2}; \tag{3.5}$$

$$\Omega = \frac{2}{T} \cdot \tan \frac{\omega}{2}. \tag{3.6}$$

Ecco che il motivo della sostituzione ora appare più chiaro: la trasformata bilineare non fa altro che mappare tramite la funzione *tangente* una funzione di trasferimento dal dominio analogico (che si estende all'infinito) al dominio digitale (che è limitato a π).

Ovviamente non si tratta di una mappa lineare, anzi la tangente può essere approssimata come tale nella parte "centrale" vicino all'origine, e più ci si avvicina a π più sarà presente un effetto di distorsione, chiamato *warping*. Per compensare questo effetto viene eseguito il cosiddetto *pre-warping*, che consiste semplicemente nell'utilizzare la relazione 3.6, ovvero si fissa la "frequenza digitale" normalizzata ω_c desiderata per il filtro, si ricava quindi la relativa "frequenza analogica" Ω_c e poi si procede alla discretizzazione.



(a) I forma diretta

(b) II forma diretta

Figura 3.1: Schemi a blocchi equivalenti di un filtro numerico del II ordine.

Una volta che è stata eseguita la sostituzione e dopo qualche passaggio algebrico si otterrà una f.d.t. a tempo discreto[1] del tipo:

$$H(z^{-1}) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}}. \tag{3.7}$$

Sempre dalla teoria della trasformata $\mathcal{Z}$, una funzione di trasferimento si traduce come un'equazione alle differenze nel dominio del tempo:

$$y(k) = a_1 \cdot y(k-1) + a_2 \cdot y(k-2) + b_0 \cdot x(k) + b_1 \cdot x(k-1) + b_2 \cdot x(k-2); \tag{3.8}$$

dove $y(k)$ e $x(k)$ rappresentano rispettivamente l'uscita e l'ingresso del sistema al passo k (il passo di campionamento T è implicito).

Essendo un'equazione alle differenze, adesso si può intuire come un filtro digitale possa essere implementato via software, si tratta solamente di risolvere poche operazioni MAC (Multiply And Accumulate).

Senza scendere in dettagli poco utili a questa trattazione, anche se l'argomento è molto interessante, i coefficienti del tipo b_0, b_1 e b_2 sono detti coefficienti FIR, mentre a_1 e a_2 vengono detti coefficienti IIR.

L'equazione 3.8 può essere tradotta in più schemi a blocchi equivalenti osservabili in figura 3.1. Il blocco con l'operatore z^{-1} è un blocco di ritardo. Un filtro del primo ordine è implementabile togliendo un blocco di ritardo da ogni "colonna".

Il vantaggio di utilizzare lo schema di figura 3.1 (a) piuttosto che (b) o viceversa dipende dall'aritmetica della CPU sulla quale verrà implementato l'algoritmo. Una rapida occhiata permette

[1] Gli esempi che seguiranno da ora in poi saranno per semplicità del primo o secondo ordine, ma sono facilmente generalizzabili a un ordine n.

di notare nella figura (a) la presenza di due "buffer di ritardo" (le "colonne" con i blocchi z^{-1}) ma di un solo accumulatore, che rende ottimale tale schema per applicazioni con aritmetiche *fixed point*. Al contrario, la II forma diretta (b) è più indicata con aritmetiche *floating point* poichè è presente un solo buffer, ma sono presenti due accumulatori, caratteristica molto svantaggiosa se si utilizzasse una rappresentazione a virgola fissa: infatti, sarebbe difficoltoso capire su quale dei due accumulatori occorrerebbe il primo overflow in caso di de-bugging del codice.

I filtri di questo equalizzatore audio, poichè come detto nel capitolo 1, l'*STM32F334R8* è provvisto di unità floating point, saranno implementati nella II forma diretta. Un esempio generale del codice in C è il seguente:

```
float genericFilter(float a1, float a2, float b0, float b1,
                    float b2, float input_value, float buffer[]){
    //inizializzazione degli accumulatori
    float input_acc = 0;
    float output_acc = 0;

    //calcolo dell'uscita
    input_acc = input_value;
    input_acc = input_acc + (a1*buffer[0]);
    input_acc = input_acc + (a2*buffer[1]);
    output_acc = b0*input_acc;
    output_acc = output_acc + (b1*buffer[0]);
    output_acc = output_acc + (b2*buffer[1]);

    //aggiornamento dei buffer
    buffer[1] = buffer[0];
    buffer[0] = input_acc;

    return output_acc;
}
```

Per riassumere quanto scritto in queste pagine, i passaggi necessari al progetto di un filtro sono:

1. scegliere la frequenza di taglio f_c e ricavare quindi $\omega_c = 2\pi f_c T$;
2. eseguire il pre-warping di ω_c per ricavare Ω_c;

3. scegliere il tipo di filtro analogico $H(s)$, come ad esempio Butterworth (verrà usato solo questo tipo), ellittico, etc... ;

4. applicare la trasformata bilineare ad $H(s)$ per ottenere una $H(z^{-1})$ nella forma della 3.7;

5. scegliere se implementare il filtro nella I o II forma diretta e scrivere il relativo codice.

3.2 Filtri di equalizzazione

Arrivati a questo punto è doverosa una premessa. Questo progetto non vuole essere un prodotto per il mondo audio professionale, anzi sicuramente alcune delle scelte prese potrebbero non essere condivise da chi lavora quotidianamente in tale ambito. Bisogna ribadire che questo lavoro di tesi nasce dalla volontà di approfondire l'argomento *Digital Signal Processing* e un'applicazione possibile era questa inerente all'audio.

Lo scopo di questo equalizzatore è quello di suddividere lo spettro delle frequenze audio (approssimativamente tra 16 e 20000 Hz) in due bande: *Bass* (fino a 440 Hz) e *Treble* (da 784 Hz) e riuscire ad amplificarle o attenuarle separatamente in modo da migliorare il mix. Per fare ciò è stato deciso di dividere le frequenze con un filtro passa basso e un filtro passa alto entrambi del primo ordine, in modo che una volta riprodotto non si senta lo "stacco" netto tra le due bande, ma ci sia un passaggio più "morbido" tra bassi e alti.

Prima di fare ciò, bisogna ricordare che il convertitore A/D converte valori compresi tra 0 e 3.6 V, quindi è presente una componente continua, o a frequenza 0 Hz, che va eliminata con un filtro passa alto del secondo ordine insieme alle bassissime frequenze, ovvero sotto i 20 Hz, nelle qualii ricadono disturbi, ad esempio passi, che è bene eliminare.

Un'altra fonte di disturbi frequenti è la rete elettrica, che in Italia è a 50 Hz, e per eliminare questa frequenza verrà utilizzato un particolare filtro detto *ad intaglio* (o *notch*).

Per concludere, l'ultima nota utile riproducibile dal pianoforte è il B_8, che corrisponde ad una frequenza di 7902.133 Hz, tutto quello che c'è oltre a tale frequenza sono le armoniche che determinano il timbro di alcuni strumenti (ad esempio i piatti della batteria), ma soprattutto è presente molto rumore. La decisione è dunque stata quella di rimuovere il contenuto dello spettro oltre il B_8 con un filtro passa basso del secondo ordine.

La catena di equalizzazione per un singolo canale è quindi quella in figura 3.2. Per l'altro canale sarà identica a differenza dei guadagni delle bande.

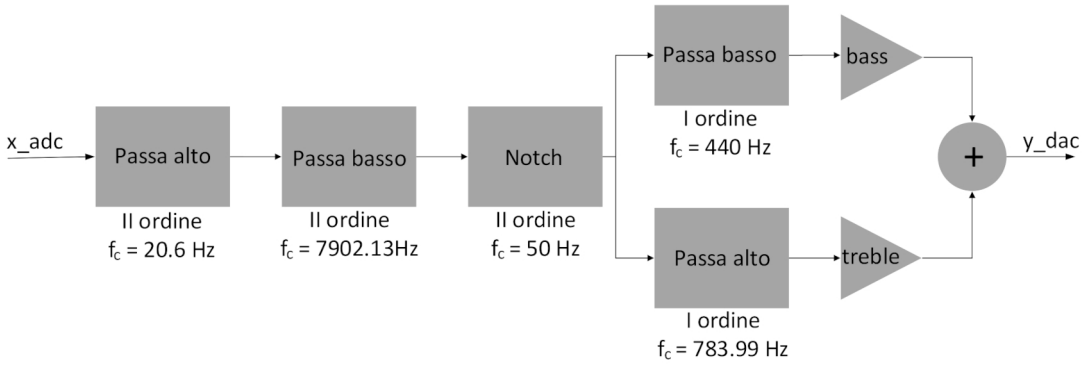


Figura 3.2: Catena di filtraggio dell'equalizzatore, si presti attenzione allo sfasamento quando le bande vengono sommate.

Filtro passa alto del II ordine

Si passa ora al progetto dei singoli filtri che compongono l'equalizzatore di figura 3.2, cominciando dal passa alto del II ordine. Seguendo il procedimento descritto nella sezione 3.1, bisogna innanzitutto fissare la "frequenza digitale" desiderata, in questo caso $20.6\ Hz$ (corrispondente alla nota E_0) per trovare la relativa pulsazione normalizzata:

$$\omega_c = 2\pi f_c T; \tag{3.9}$$

applicando ora la 3.6 si trova:

$$\Omega_c = 129.43[rad/s]. \tag{3.10}$$

L'equazione di un filtro *analogico* di Butterworth del II ordine è:

$$H(s) = \frac{\tau^2 s^2}{1 + \sqrt{2}\tau s + \tau^2 s^2}; \tag{3.11}$$

dove $\tau = 1/\Omega_c$. Si applica ora la trasformata bilineare sostituendo nella 3.11 la 3.1:

$$H(z^{-1}) = \frac{\tau^2 \cdot \frac{4}{T^2} \cdot \frac{(1-z^{-1})^2}{(1+z^{-1})^2}}{1 + \sqrt{2}\tau \cdot \frac{2}{T} \cdot \frac{1-z^{-1}}{1+z^{-1}} + \tau^2 \cdot \frac{4}{T^2} \cdot \frac{(1-z^{-1})^2}{(1+z^{-1})^2}}; \tag{3.12}$$

e, dopo una serie di lunghi e noiosi passaggi algebrici si trova:

$$H(z^{-1}) = \frac{\frac{4\tau^2}{4\tau^2+2\sqrt{2}\tau T+T^2} + \frac{-8\tau^2}{4\tau^2+2\sqrt{2}\tau T+T^2} \cdot z^{-1} + \frac{4\tau^2}{4\tau^2+2\sqrt{2}\tau T+T^2} \cdot z^{-2}}{1 - \frac{8\tau^2-2T^2}{4\tau^2+2\sqrt{2}\tau T+T^2} \cdot z^{-1} - \frac{-4\tau^2+2\sqrt{2}\tau T-T^2}{4\tau^2+2\sqrt{2}\tau T+T^2} \cdot z^{-2}}; \tag{3.13}$$

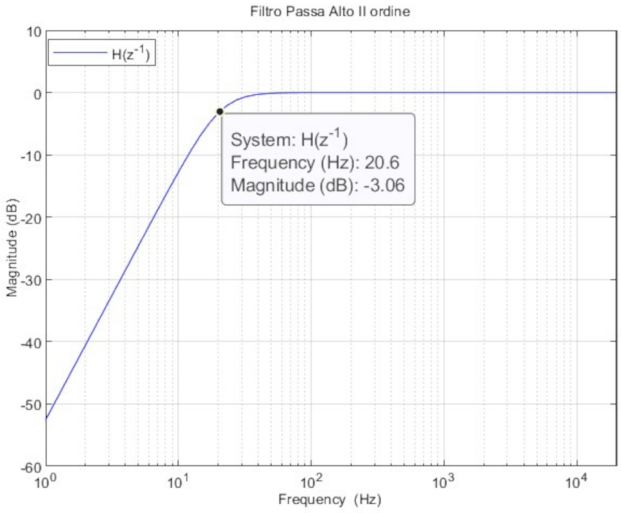


Figura 3.3: Modulo della risposta in frequenza del filtro, la cui funzione di trasferimento è espressa dalla 3.13.

che è nella forma della 3.7. Si possono dunque ricavare i coefficienti FIR e IIR:

$$b_0 \approx 0.997714531;$$

$$b_1 \approx -1.995429062;$$

$$b_2 \approx 0.997714531;$$

$$a_1 \approx 1.995423839;$$

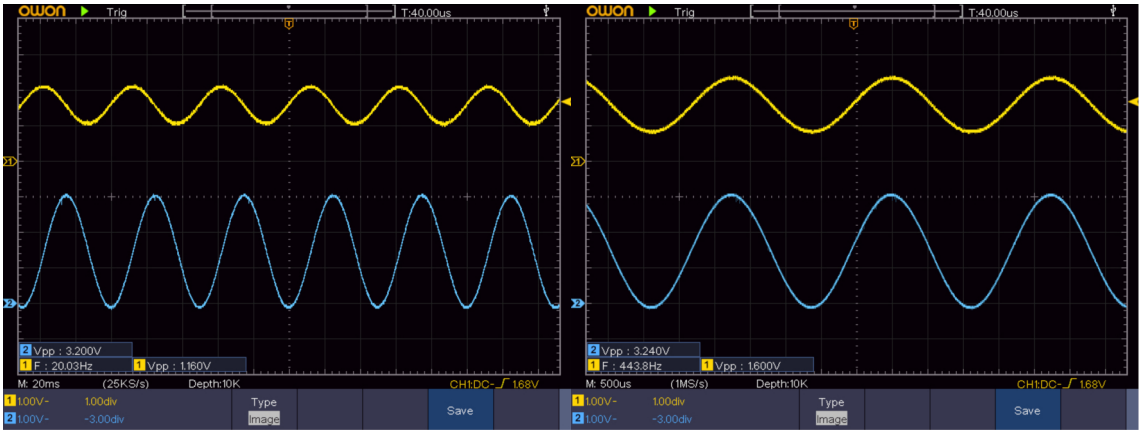
$$a_2 \approx -0.995434285;$$

i quali sono facilmente rappresentabili con un'adeguata approssimazione in virgola mobile a precisione singola.

La simulazione di questo filtro restituisce la risposta in frequenza visibile in figura 3.3, che corrisponde a quanto ci si aspetta da un filtro passa alto del II ordine di Butterworth. La fase non è rilevante e per questo non è mostrata. Utilizzando l'oscilloscopio e mettendo in ingresso un'onda sinusoidale all'ADC e sapendo che questa in uscita al DAC, senza essere amplificata, è ampia la metà[2], si ottengono i risultati sperimentali apprezzabili in figura 3.4 (a), (b) e (c).

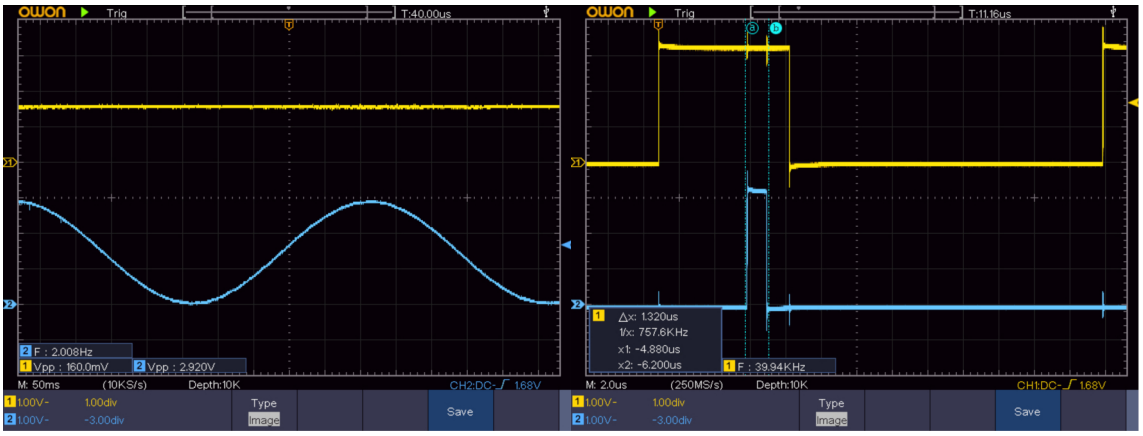
La figura 3.4 (d) invece mostra che la durata dell'algoritmo è inferiore a $1.320\ \mu s$ poichè il tempo misurato comprende anche quello delle istruzioni di alzare e abbassare il pin monitor, che consente di fare questa stima. Il risultato è rassicurante poichè, come affermato nel capitolo 1, il tempo a disposizione per gestire la routine di interrupt nella quale è presente il codice dei

[2]Questo è per evitare problemi di saturazione quando l'ingresso non sarà un segnale sinusoidale ma una vera traccia audio con un contenuto armonico molto ricco.



(a) Frequenza di taglio: 20.6 Hz

(b) In banda: 444 Hz



(c) Fuori banda: 2 Hz

(d) Durata dell'algoritmo

Figura 3.4: Acquisizioni sperimentali, nelle figure (a), (b) e (c) il segnale in ingresso all'ADC è quello azzurro, il segnale in uscita al DAC dopo essere stato filtrato è quello giallo. In figura (d) invece si può notare una stima grossolana della durata temporale della routine di interrupt (giallo) e del solo algoritmo del filtro (azzurro).

filtri è di $25\ \mu s$. Poichè i filtri da implementare tra i due canali sono 10, questo fa ben sperare che con la frequenza di campionamento scelta le temporizzazioni non dovrebbero essere una parte critica o particolarmente problematica.

Filtro passa basso del II ordine

La procedura è identica a quella seguita prima per il filtro passa alto del II ordine. Si determina la "frequenza di taglio digitale" desiderata $f_c = 7902.13\ Hz$ e si ricava la relativa pulsazione di taglio normalizzata:

$$\omega_c = 2\pi f_c T. \tag{3.14}$$

Si ricava quindi la pulsazione analogica:

$$\Omega_c = 57189.06[rad/s]. \tag{3.15}$$

L'equazione generale di un filtro passa basso di Butterworth del secondo ordine è:

$$H(s) = \frac{1}{1 + \sqrt{2}\tau s + \tau^2 s^2}; \tag{3.16}$$

e applicando la trasformata bilineare, dopo una serie di conti per mettersi nella forma della 3.7, si ottiene:

$$H(z^{-1}) = \frac{\frac{T^2}{4\tau^2+2\sqrt{2}\tau T+T^2} + \frac{2T^2}{4\tau^2+2\sqrt{2}\tau T+T^2} \cdot z^{-1} + \frac{T^2}{4\tau^2+2\sqrt{2}\tau T+T^2} \cdot z^{-2}}{1 - \frac{8\tau^2-2T^2}{4\tau^2+2\sqrt{2}\tau T+T^2} \cdot z^{-1} - \frac{-4\tau^2+2\sqrt{2}\tau T-T^2}{4\tau^2+2\sqrt{2}\tau T+T^2} \cdot z^{-2}}. \tag{3.17}$$

I valori dei coefficienti FIR e IIR saranno quindi:

$$b_0 \approx 0.2026287711;$$

$$b_1 \approx 0.4052575422;$$

$$b_2 \approx 0.2026287711;$$

$$a_1 \approx 0.3877642089;$$

$$a_2 \approx -0.1982792933.$$

Un'analisi con MATLAB determina la risposta in frequenza di figura 3.5. Poichè la frequenza di taglio è abbastanza vicina a quella di Nyquist $f_{Nyq} = f_{samp}/2 = 20000\ Hz$, si vede molto bene l'effetto di warping che fa discostare il comportamento di questo filtro rispetto al suo corrispettivo analogico.

Utilizzando l'oscilloscopio e mettendo in ingresso un'onda sinusoidale all'ADC e misurando l'uscita del DAC, si ottengono i risultati sperimentali in figura 3.6 (a), (b) e (c). Si noti come più

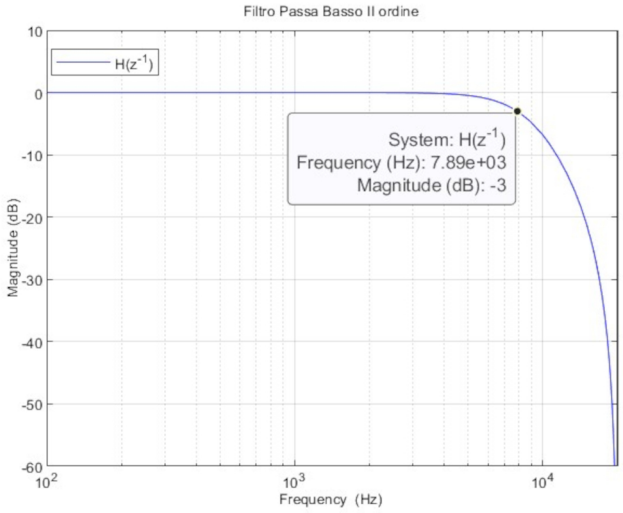


Figura 3.5: Modulo della risposta in frequenza del filtro, la cui funzione di trasferimento è espressa dalla 3.17.

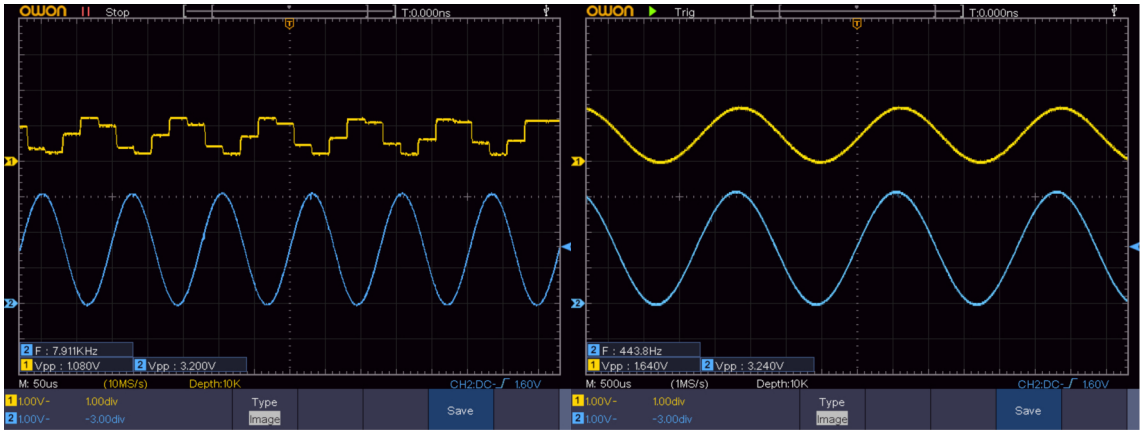
ci si avvicina alla frequenza a cui opera il DAC più l'effetto di quantizzazione si faccia sentire, effetto che può essere attenuato o eliminato con un opportuno filtraggio in uscita al convertitore D/A.

Come per il filtro precedente, in figura 3.6 (d) si vede il peso del codice del filtro sul tempo totale della routine di interrupt.

Filtro notch

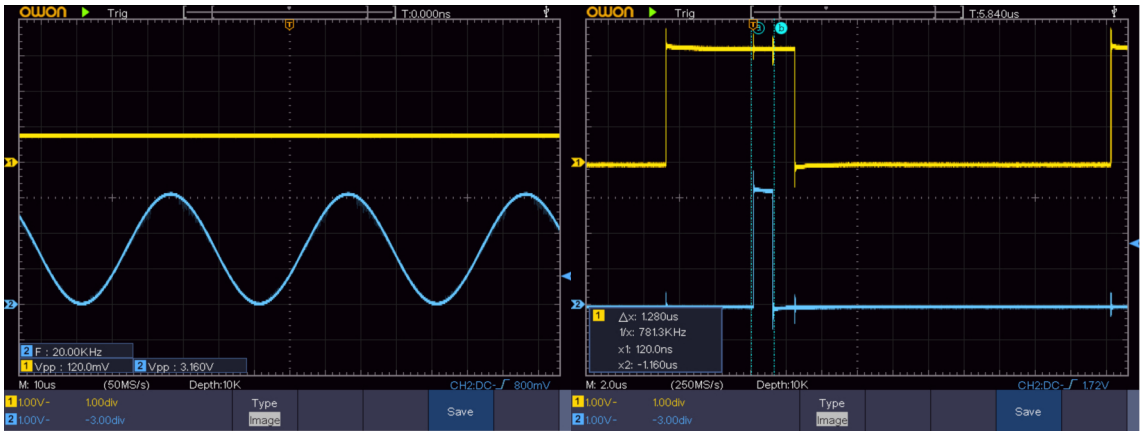
Questa tipologia di filtro consiste nell'interazione tra una coppia di poli e zeri complessi coniugati ed è l'unico trattato in questa tesi che non è stato ricavato tramite la tecnica della trasformata di Tustin. La coppia di zeri giace sulla circonferenza unitaria nel piano complesso, mentre la coppia di poli è all'interno di questa. Se si avvicinano i poli agli zeri in modo che questi non coincidano, ma che stiano sulla circonferenza di raggio $\rho = 0.99$ o più (ma in ogni caso *sempre minore di* 1), si ottiene un particolare filtro elimina banda stretta che idealmente potrebbe eliminare selettivamente una singola frequenza. L'equazione di tale filtro è:

$$H(z^{-1}) = \frac{1 - 2\cos(2\pi f_n T) \cdot z^{-1} + z^{-2}}{1 - 2\rho\cos(2\pi f_n T) \cdot z^{-1} - \rho^2 \cdot z^{-2}}. \tag{3.18}$$



(a) Frequenza di taglio: 7902.13 Hz

(b) In banda: 444 Hz



(c) Fuori banda: 20 kHz

(d) Durata dell'algoritmo

Figura 3.6: Acquisizioni sperimentali, nelle figure (a), (b) e (c) il segnale in ingresso all'ADC è quello azzurro, il segnale in uscita al DAC dopo essere stato filtrato è quello giallo. In figura (d) invece si può notare una stima grossolana della durata temporale della routine di interrupt (giallo) e del solo algoritmo del filtro (azzurro).

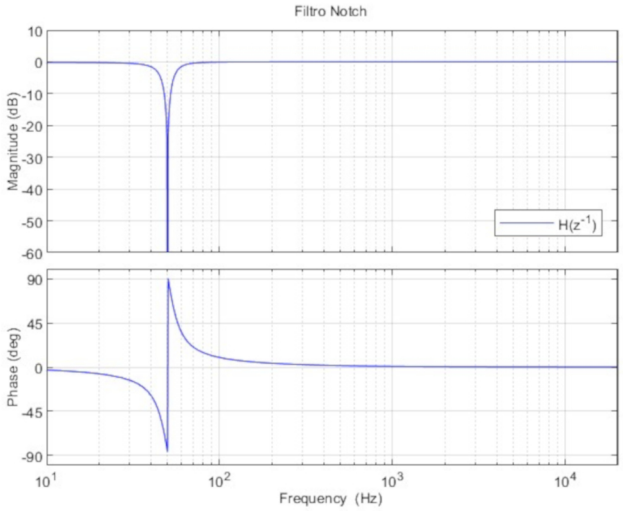


Figura 3.7: Risposta in frequenza di modulo e fase del filtro notch la cui equazione è espressa dalla 3.18.

Sempre confrontando la 3.18 con l'equazione generale di un filtro del secondo ordine, si possono ricavare i seguenti coefficienti:

$$b_0 = 1;$$
$$b_1 = -2\cos(2\pi f_n T);$$
$$b_2 = 1;$$
$$a_1 = 2\rho\cos(2\pi f_n T);$$
$$a_2 = -\rho^2;$$

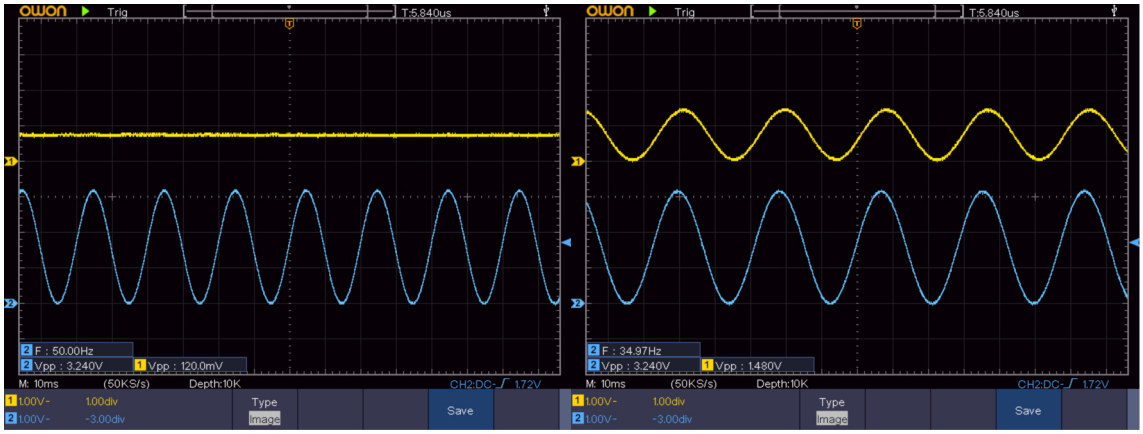
dove, dalla breve trattazione appena svolta, si può intuire che il termine ρ indica la selettività del filtro mentre l'angolo, ovvero l'argomento del coseno, determina invece la frequenza su cui si vuole posizionare l'intaglio.

Ponendo quindi $\rho = 0.999$ e $f_n = 50\ Hz$ e simulando il filtro con l'ausilio di MATLAB si ottiene la risposta in frequenza di figura 3.7.

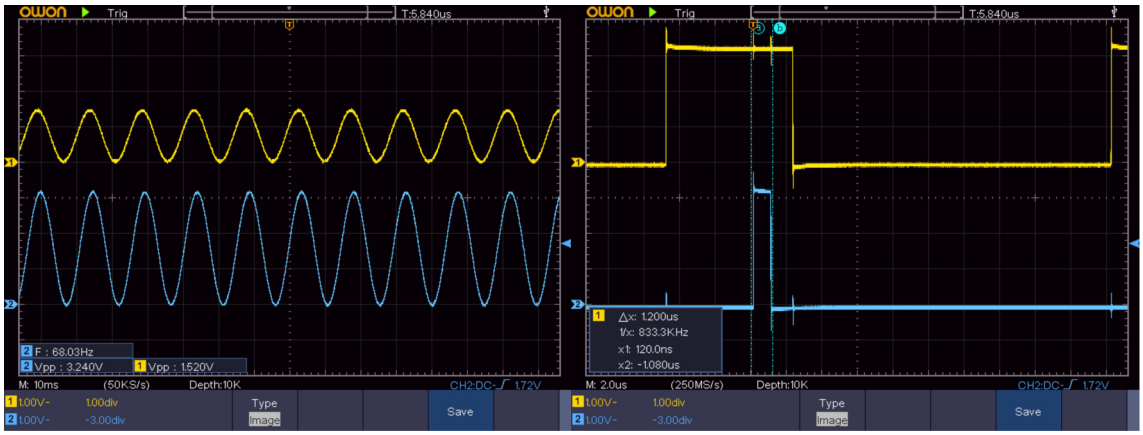
Come per i due filtri precedenti, ponendo in ingresso al convertitore A/D un segnale sinusoidale e collegando l'uscita del DAC all'oscilloscopio, sono stati acquisiti i risultati in figura 3.8 (a), (b) e (c). La durata dell'algoritmo è visibile invece nella (d).

Filtro passa basso del I ordine

Si passa ora all'analisi dei filtri che dividono lo spettro delle frequenze audio in due bande. La prima sarà quella che va dai $20.6\ Hz$ (E_0) ai $440\ Hz$ (A_4). La scelta di farli del primo ordine rispetto al secondo sarà più chiara una volta che si vedrà l'insieme della catena di filtraggio.



(a) Frequenza intaglio: $50\ Hz$ (b) $35\ Hz$



(c) $62\ Hz$ (d) Durata dell'algoritmo

Figura 3.8: Acquisizioni sperimentali, nelle figure (a), (b) e (c) il segnale in ingresso all'ADC è quello azzurro, il segnale in uscita al DAC dopo essere stato filtrato è quello giallo. In figura (d) invece si può notare una stima grossolana della durata temporale della routine di interrupt (giallo) e del solo algoritmo del filtro (azzurro).

Esattamente come è stato fatto per quelli di ordine superiore, si fissa la frequenza di taglio $f_c = 440\ Hz$ e si calcola la "pulsazione di taglio digitale" normalizzata:

$$\omega_c = 2\pi f_c T; \tag{3.19}$$

si passa quindi a calcolare la rispettiva "pulsazione analogica" attraverso la 3.6:

$$\Omega_c = 2765.70[rad/s]. \tag{3.20}$$

L'equazione generica di un filtro passa basso del I ordine è:

$$H(s) = \frac{1}{1+\tau s}; \tag{3.21}$$

dove $\tau = 1/\Omega_c$. Si applica quindi la sostituzione della trasformata bilineare ottenendo così:

$$H(z^{-1}) = \frac{1}{1+\tau \cdot \frac{2}{T} \cdot \frac{1-z^{-1}}{1+z^{-1}}}; \tag{3.22}$$

e dopo semplici passaggi algebrici si ottiene:

$$H(z^{-1}) = \frac{\frac{T}{2\tau+T} + \frac{T}{2\tau+T} \cdot z^{-1}}{1 - \frac{2\tau-T}{2\tau+T} \cdot z^{-1}}. \tag{3.23}$$

Tutti i dati per calcolare i coefficienti sono stati fissati ricavando:

$$b_0 = 0.0334160466;$$

$$b_1 = 0.0334160466;$$

$$a_1 = 0.9331679068.$$

La simulazione di tale filtro produce la risposta in frequenza di figura 3.9, dove si può vedere molto chiaramente l'effetto dell'unico polo e anche quello dovuto al warping avvicinandosi a f_{Nyq}. Di particolare interesse è, in questo caso e nel successivo, anche la fase, questo perchè le due bande andranno poi sommate algebricamente e se lo sfasamento tra le due dovesse avvicinarsi a $180°$ si avrà un effetto distruttivo invece che costruttivo.

Sempre ponendo in ingresso al convertitore A/D un segnale sinusoidale a diverse frequenze, è possibile valutare sperimentalmente il corretto funzionamento del filtro osservando l'uscita direttamente dal convertitore D/A, visibile in figura 3.10 (a), (b) e (c).

Un aspetto interessante di figura 3.10 (d) è che il tempo impiegato dall'algoritmo è diminuito rispetto ai filtri precedenti. Questo fatto è dovuto alla complessità del codice che è diminuita, dovendo solo salvare il valore dell'accumulatore precedente e non gli ultimi due, oltre che eseguire meno operazioni MAC.

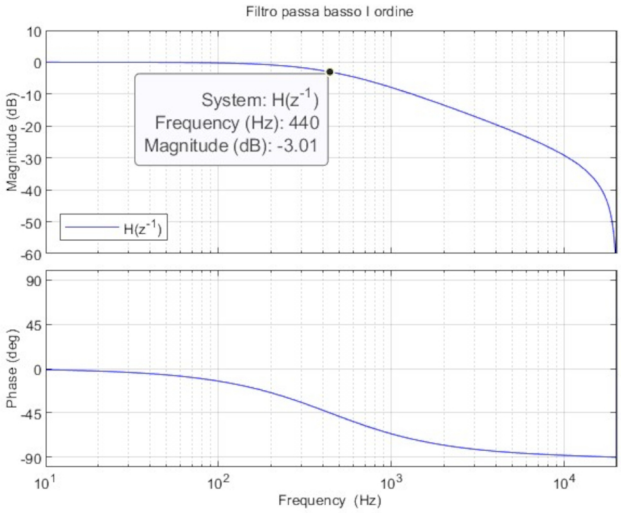
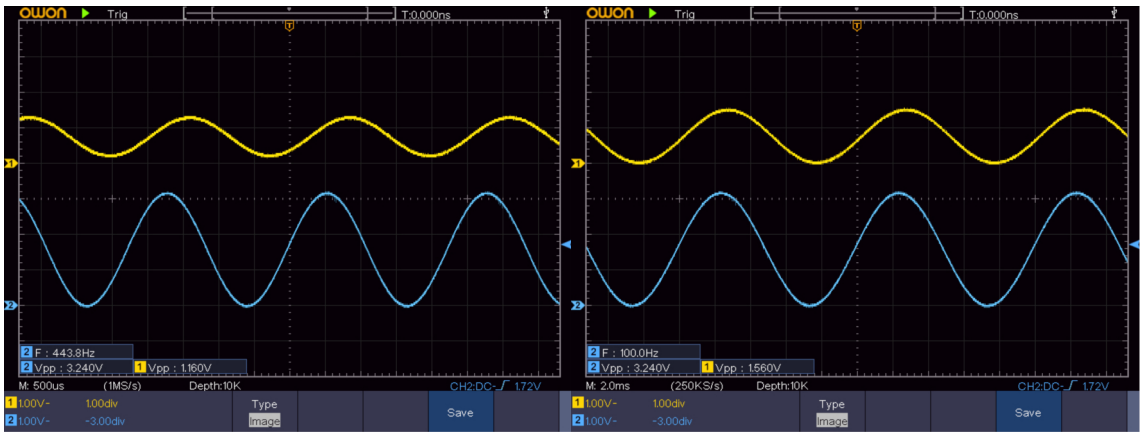
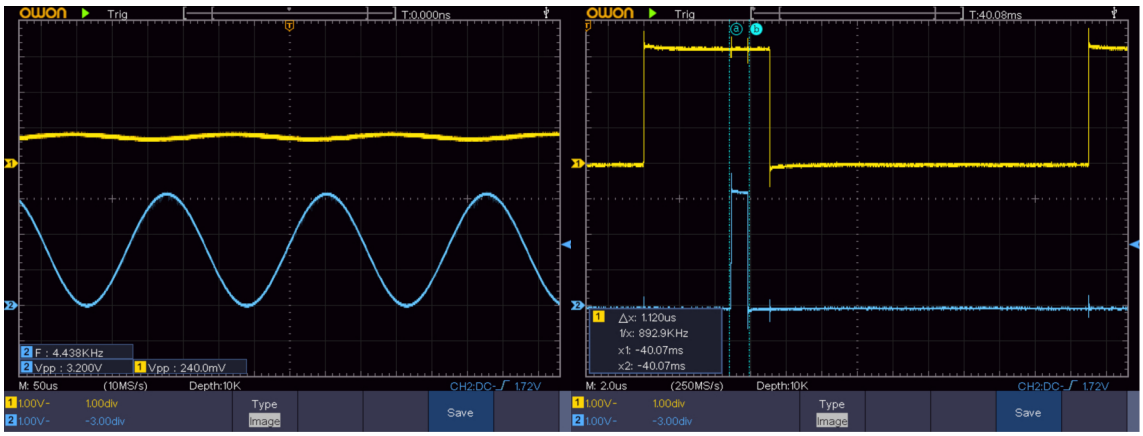


Figura 3.9: Risposta in frequenza di modulo e fase del filtro la cui equazione è espressa dalla 3.23.



(a) Frequenza di taglio: 440 Hz

(b) In banda 100 Hz



(c) Fuori banda 4440 Hz

(d) Durata dell'algoritmo

Figura 3.10: Acquisizioni sperimentali, nelle figure (a), (b) e (c) il segnale in ingresso all'ADC è quello azzurro, il segnale in uscita al DAC dopo essere stato filtrato è quello giallo. In figura (d) invece si può notare una stima grossolana della durata temporale della routine di interrupt (giallo) e del solo algoritmo del filtro (azzurro).

Filtro passa alto del I ordine

Per quest'ultimo filtro la procedura è identica alla precedente. L'unica accortezza da avere però è dove porre la frequenza di taglio f_c. Dato che la banda *Treble* verrà poi sommata a quella *Bass* non si può fissare $f_c = 440\ Hz$ perchè il passaggio a $-3\ dB$ fa sì che l'ampiezza del segnale sia il 70% di quella di partenza. Bisogna quindi fare in modo di porre la f_c quando il filtro precedente avrà raggiunto i $-10.7\ dB$. Infatti poichè la somma non deve superare gli $0\ dB$:

$$-3[dB] = 20\log_{10} x; \tag{3.24}$$

$$x = 10^{-\frac{3}{20}} \approx 0.7; \tag{3.25}$$

$$20\log_{10} 0.3 = -10.7[dB]. \tag{3.26}$$

Controllando quindi dal grafico di figura 3.9 la frequenza a cui corrispondono i $-10.7\ dB$ si trova la frequenza di taglio $f_c = 783.99\ Hz$ (nota G_5). Si può allora procedere esattamente come per il filtro precedente:

$$\omega_c = 2\pi f_c T. \tag{3.27}$$

Si esegue ora il pre-warping:

$$\Omega_c = 4932.19[rad/s]. \tag{3.28}$$

La f.d.t. di un filtro passa alto del primo ordine è:

$$H(s) = \frac{\tau s}{1+\tau s}; \tag{3.29}$$

e operando la trasformata bilineare, dopo qualche manipolazione matematica:

$$H(z^{-1}) = \frac{\frac{2\tau}{2\tau+T} + \frac{-2\tau}{2\tau+T}\cdot z^{-1}}{1 - \frac{2\tau-T}{2\tau+T}\cdot z^{-1}}. \tag{3.30}$$

I coefficienti saranno quindi:

$$b_0 = 0.9419279147;$$

$$b_1 = -0.9419279147;$$

$$a_1 = 0.8838558294.$$

In figura 3.11 si vede la risposta in frequenza calcolata con MATLAB.

I risultati sperimentali ottenuti e la durata dell'algoritmo sono apprezzabili in figura 3.12.

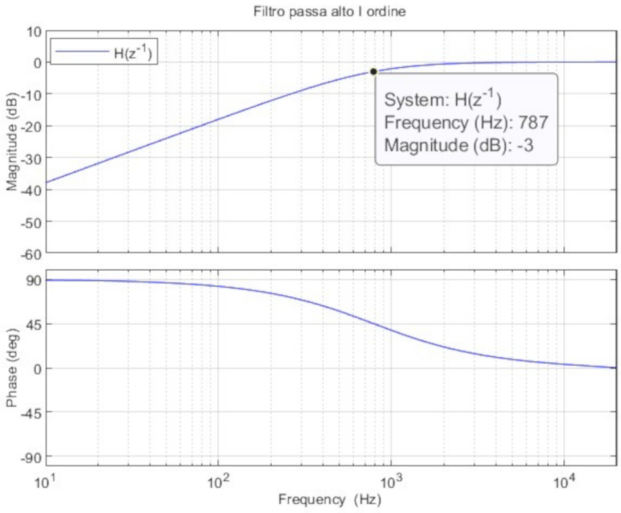
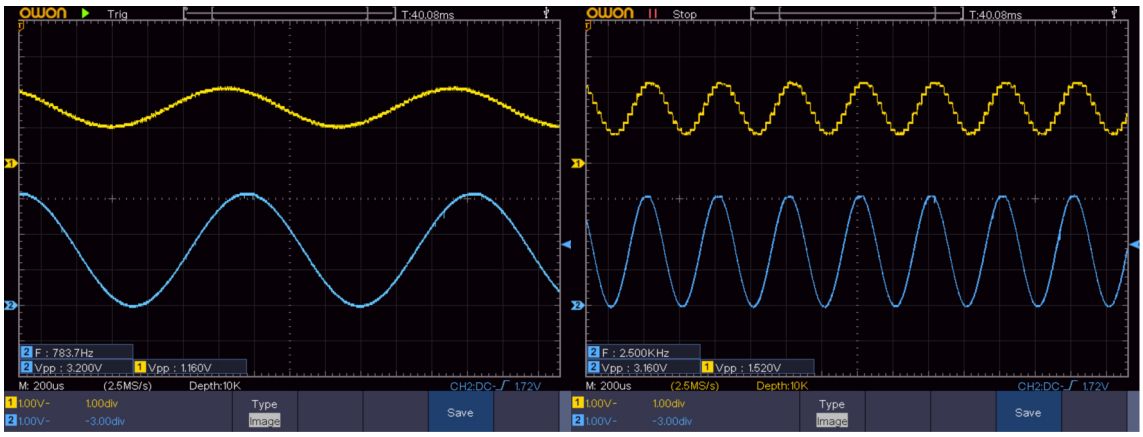
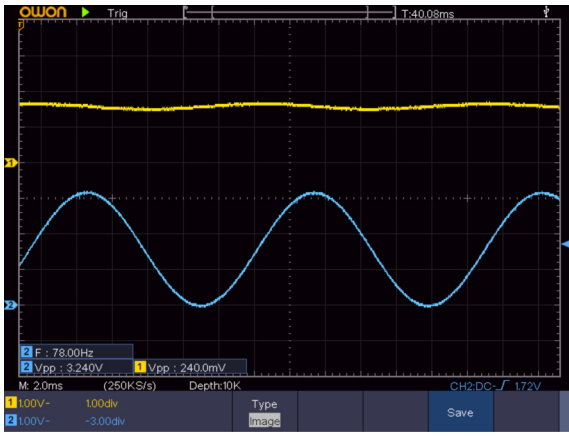


Figura 3.11: Risposta in frequenza di modulo e fase del filtro la cui equazione è espressa dalla 3.30.

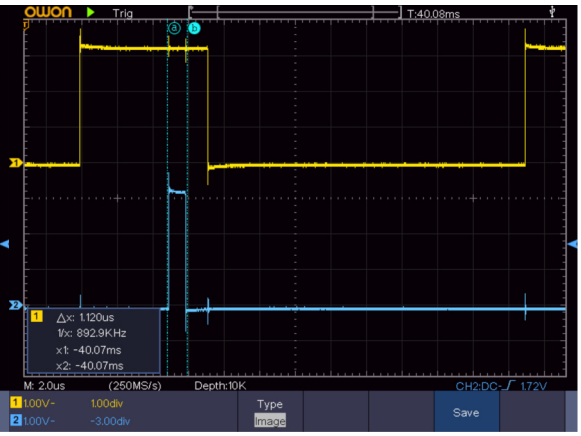


(a) Frequenza di taglio: 783.99 Hz

(b) In banda 2500 Hz

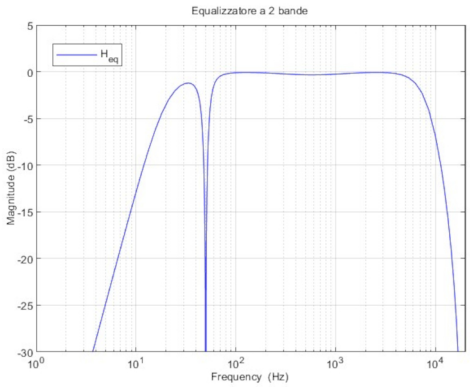


(c) Fuori banda 78 Hz

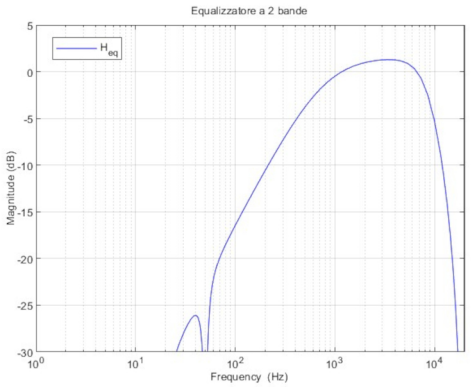


(d) Durata dell'algoritmo

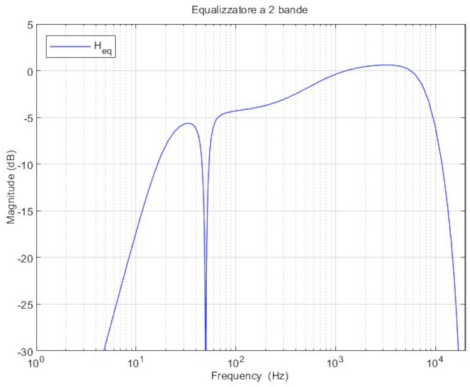
Figura 3.12: Acquisizioni sperimentali, nelle figure (a), (b) e (c) il segnale in ingresso all'ADC è quello azzurro, il segnale in uscita al DAC dopo essere stato filtrato è quello giallo. In figura (d) invece si può notare una stima grossolana della durata temporale della routine di interrupt (giallo) e del solo algoritmo del filtro (azzurro).



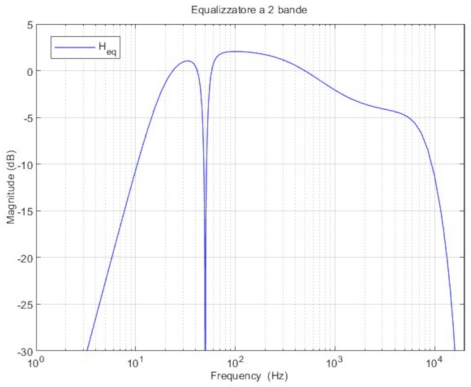
(a) $Bass = 1, Treble = 1$



(b) $Bass = 0, Treble = 1.2$



(c) $Bass = 0.6, Treble = 1.1$



(d) $Bass = 1.3, Treble = 0.6$

Figura 3.13: Risposta in frequenza dell'equalizzatore audio con diversi gain per ciascuna banda.

3.3 Funzionamento dell'equalizzatore

Dopo aver progettato i singoli filtri, questi vanno messi insieme per formare l'equalizzazione di un canale, secondo lo schema di figura 3.2. La risposta in frequenza complessiva è visibile nella figura 3.13. Si possono osservare le differenze dovute alle diverse amplificazioni delle bande *Bass* e *Treble*. Si è scelto di dividere le due bande con un filtro del primo ordine rispetto ad uno di ordine superiore poichè, in questa maniera, le frequenze intermedie tra le due bande non vengono penalizzate troppo con attenuazioni improvvise. Mettendo quindi un filtro la cui pendenza è più lieve, questo passaggio è più "dolce" e il mix audio appare più omogeneo.

La durata totale della routine di interrupt si vede in figura 3.14 e comprende: gestione della richiesta di interrupt, lettura del registro di uscita dell'ADC con il dato convertito, algoritmi di filtraggio e scrittura dei dati sul registro del convertitore D/A per i canali audio destro e sinistro. Come si può notare la durata è molto inferiore a quella che si otterrebbe sommando tutti i tempi impiegati da ogni filtro. Questo perchè le istruzioni di portare al valore logico 1 un pin di I/O e

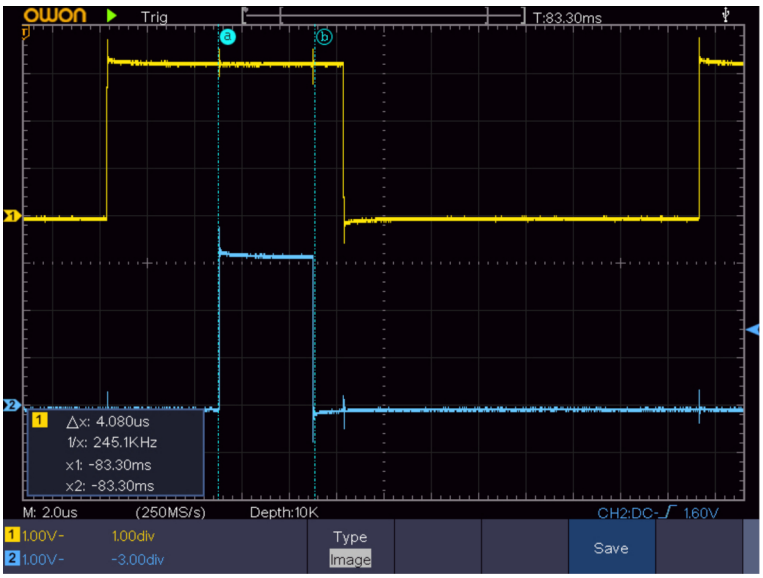


Figura 3.14: Stima del tempo occupato dalla routine di interrupt (in giallo) e di quello occupato dai soli filtri (in azzurro).

di abbassarlo a 0 sono molte onerose dal punto di vista del tempo impiegato e falsano quindi di molto la stima fatta fino ad ora, in ogni caso utile per fornire un tempo indicativo.

Per modificare il valore del gain delle singole bande, viene utilizzato il pulsante *User* disponibile a bordo della scheda Nucleo334. Di conseguenza, anzichè modificare singolarmente (ad esempio attraverso dei potenziometri) il guadagno di ciascuna banda, sono stati creati dei valori prefissati (quelli visibili in figura 3.13) che ad ogni pressione del pulsante vengono aggiornati ciclicamente essendo memorizzati in un buffer circolare. Questa parte di codice è quindi presente nel *main* e non nella routine di interrupt. L'unica accortezza da avere in questo caso è quella di predisporre un anti-rimbalzo per il pulsante; questo perchè, quando viene premuto, altera il suo stato più volte prima di stabilizzarsi e può causare problemi in fase di lettura da parte del DSC. La scelta adottata è stata quella di predisporre un anti-rimbalzo software che consiste nell'attendere a vuoto per un periodo di tempo di qualche millisecondo al primo cambiamento di stato del pulsante e nel aggiornare i valori una volta che questo viene rilasciato.

Il codice C è il seguente:

```
//Vettori con i guadagni selezionabili  di ciascuna banda di
//ciascun canale
float bsx[4] = {1, 0, 1.3, 0.6};
float tsx[4] = {1, 1.2, 0.6, 1.1};
float bdx[4] = {1, 1.2, 1.3, 0.6};
```

```
float tdx[4] = {1, 0, 0.6, 1.1};

//User button normalmente alto
if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == 0){
    //Attendere che si stabilizzi il pulsante
    HAL_Delay(50);
    //Attendere che il pulsante venga rilasciato
    while(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == 0){}
    i++;
    if(i>3){
        i=0;
    }
    gain_bsx = bsx[i];
    gain_tsx = tsx[i];
    gain_bdx = bdx[i];
    gain_tdx = tdx[i];
}
```

Inoltre, per far capire all'utente quale pre-set è al momento selezionato, è stato sfruttato il led presente sulla Nucleo334, che lampeggia a 4 velocità differenti in base al valore della variabile i.

Capitolo 4

Filtro ricostruttore e amplificatore audio

In quest'ultimo capitolo verrà analizzato lo stadio di uscita dell'equalizzatore audio, composto da un filtro passa banda analogico, il cui scopo è quello di eliminare la componente continua in uscita dal DAC nonchè le armoniche ad alta frequenza introdotte dalla ricostruzione del segnale analogico (ben visibili in figura 3.6 (a) ad esempio), oltre che da un amplificatore audio, poichè un amplificatore operazionale come il *TL082* non è in grado di erogare la corrente necessaria a pilotare un diffusore.

Filtro di uscita

Per il progetto di questo filtro è stato utilizzato il software gratuito *Filter Design Tool* di *Texas Instruments*. Mediante questo tool disponibile online sul sito del produttore, si è in grado di progettare senza dover fare calcoli un filtro analogico anche di ordine elevato.

Per quanto concerne questa tesi, l'idea è quella di realizzare un filtro passa-banda del secondo ordine per le motivazioni date pocanzi. La scelta di fermarsi al secondo ordine è dettata dalla limitatezza dei componenti elettronici a disposizione. Una volta avviato il tool si sceglie il tipo di filtro (per questa applicazione passa-banda) e si danno le specifiche di frequenza centrale e larghezza di banda. Per riuscire a ricostruire il segnale, andrebbero eliminate tutte le armoniche multiple della frequenza di campionamento (si ricorda essere $f_{samp} = 40\ kHz$) che si orginano. In favore di avere una buona attenuazione ai $40\ kHz$ si è quindi deciso di porre la frequenza di taglio inferiore $f_{c_{inf}} = 7\ Hz$ e la frequenza di taglio superiore $f_{c_{sup}} = 8.8\ kHz$, anche se questo implica un'ulteriore attenuazione della banda delle altissime frequenze (che, come è stato già detto, è spesso ricca di rumore ma anche di informazione utile).

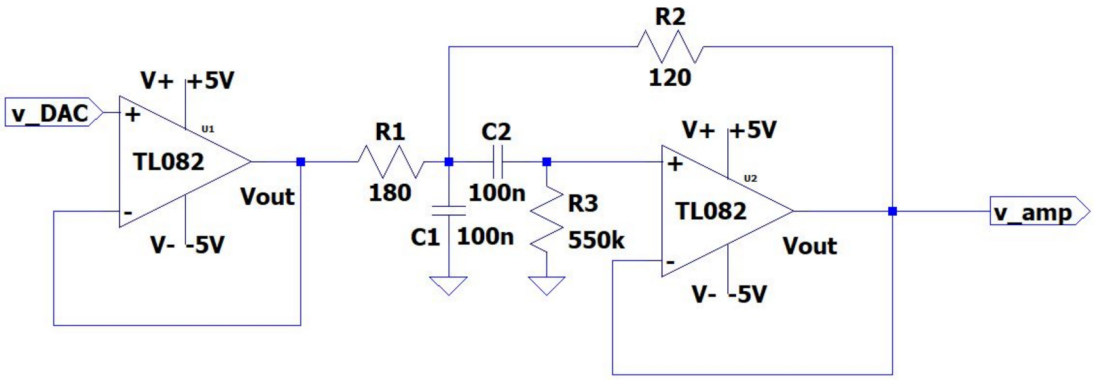


Figura 4.1: Schema circuitale del filtro di uscita del convertitore D/A. Si noti che il filtro è preceduto da un buffer per disaccoppiare il μC dai circuiti esterni.

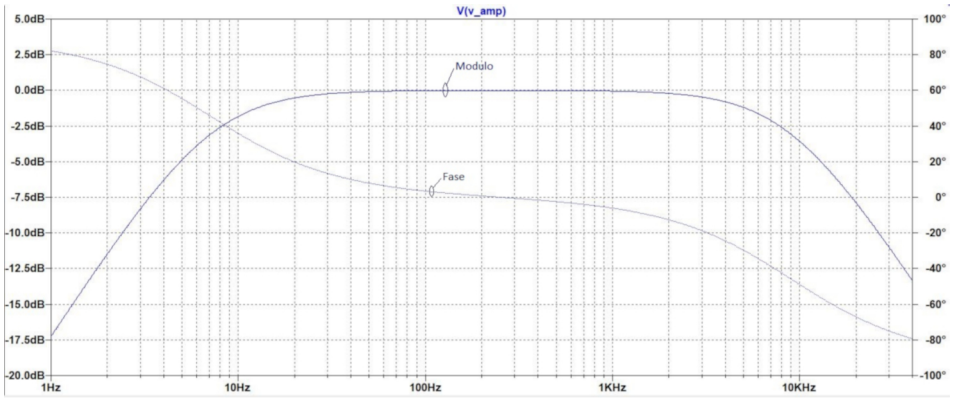


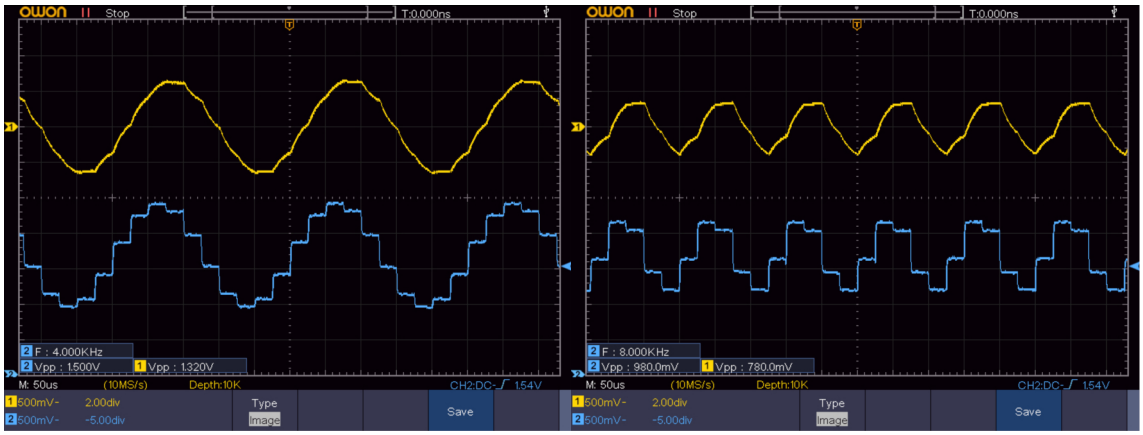
Figura 4.2: Risposta in frequenza del circuito di figura 4.1.

Dopo aver scelto di volere un filtro di *Butterworth* con topologia *Sallen-Key*, selezionato l'OPAMP utilizzato e la serie di resistenze e condensatori, è stato restituito lo schema elettrico di figura 4.1. I risultati della simulazione *LTSpice* sono visibili in figura 4.2.

I risultati sperimentali ottenuti confrontando la tensione v_{DAC} di uscita del convertitore D/A rispetto a quella in uscita al filtro ricostruttore v_{amp} sono visibili in figura 4.3. Come si può osservare, la ricostruzione è tutt'altro che perfetta perchè la $f_{c_{sup}}$ è stata posta meno di una decade prima della f_{samp} e di conseguenza non riesce a rimuoverla completamente. Una soluzione per migliorare questo problema potrebbe essere quella di alzare l'ordine del filtro oppure la frequenza di campionamento.

Amplificatore audio

Come accennato all'inizio del capitolo, il *TL082* non è in grado di fornire la corrente necessaria al pilotaggio di un piccolo diffusore, richiedendo quindi l'uso di uno stadio amplificatore.



(a) Frequenza $4\ kHz$ (b) Frequenza $8\ kHz$

Figura 4.3: Acquisizioni sperimentali del funzionamento del filtro di uscita; in azzurro si può notare il segnale non filtratro con gli scalini tipici della quantizzazione, mentre in giallo il segnale ricostruito, più simile ad una sinusoide.

L'*LM386* è un integrato pensato appositamente per piccoli amplificatori a bassa potenza ed è quindi ideale per questa applicazione. In figura 4.4 si può vedere il suo schema interno preso direttamente dal suo *datasheet*. Si può notare la semplicità del circuito, composto da uno stadio differenziale seguito da uno stadio di amplificazione, formato da un transistor in configurazione a emettitore comune, e infine lo stadio di uscita in classe AB. Sempre facendo fede al datasheet, dalla sezione *typical applications* è stato tratto il circuito di figura 4.5. Il partitore di tensione in ingresso è necessario piochè la tensione di ingresso deve essere compresa tra i $\pm 0.4\ V$. Sapendo che $v_{amp} \in [-1.65, 1.65][V]$ e che l'obbiettivo è che la tensione di uscita sia all'incirca compresa tra $\pm 2.5\ V$:

$$v_{R_4} = v_{R_2} \cdot 20; \tag{4.1}$$

$$v_{R_2} = \frac{v_{R_4}}{20} = 0.125[V]; \tag{4.2}$$

$$v_{R_2} = v_{amp} \cdot \frac{R_2}{R_1 + R_2}. \tag{4.3}$$

Dalla 4.3 si ottiene:

$$\frac{v_{R_2}}{v_{amp}} = \frac{R_2}{R_1 + R_2}; \tag{4.4}$$

imponendo ora $R_1 = 100\ k\Omega$ e che R_2 è un trimmer da $10\ k\Omega$, si ottiene:

$$\frac{0.125[V]}{1.65[V]} = \frac{10 - x[k\Omega]}{110[k\Omega]}; \tag{4.5}$$

$$x = 10 - 110 \cdot \frac{0.125}{1.65} = 1667[\Omega]. \tag{4.6}$$

Questo risultato indica quindi che il trimmer (che regola il volume generale) non può raggiungere il finecorsa se si desidera avere l'uscita compresa tra i $\pm 2.5\ V$. In realtà, alimentado

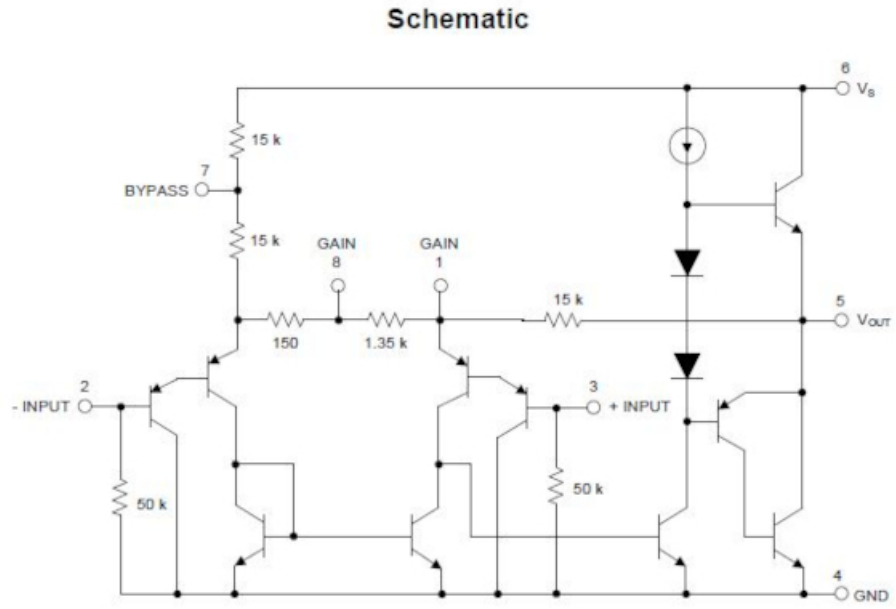


Figura 4.4: Schema dell'integrato *LM386*.

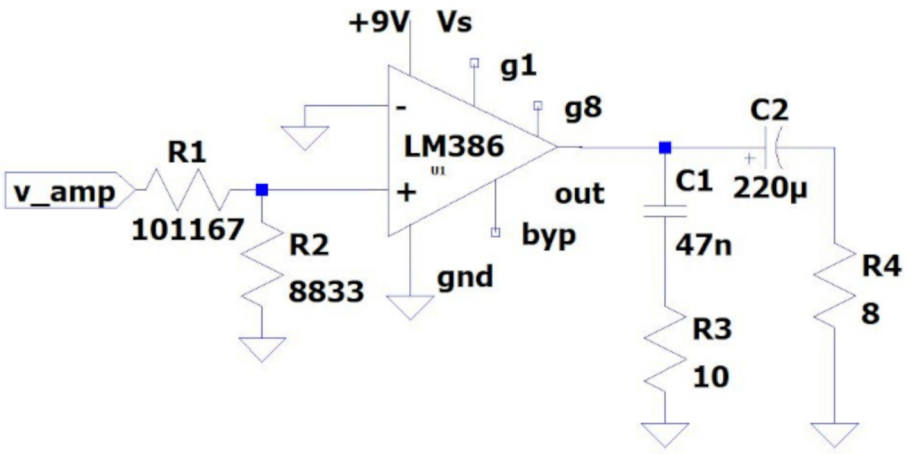


Figura 4.5: Amplificatore audio con *LM386* e $gain = 20$.

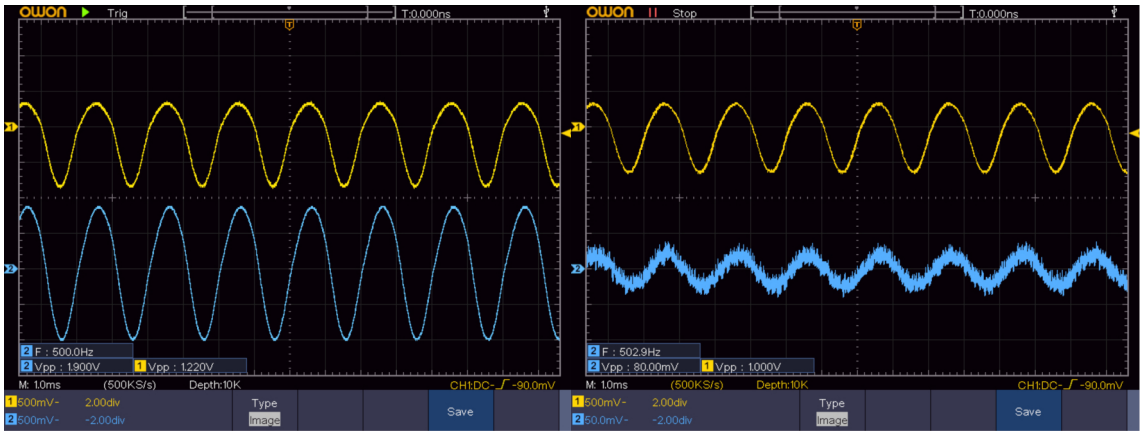
l'integrato a $9\ V$, anche se il trimmer raggiungesse il finecorsa si otterrebbe:

$$v_{R_2} = v_{amp} \cdot \frac{R_2}{R_1 + R_2} = 0.15[V]; \tag{4.7}$$

$$v_{R_4} = 20 \cdot 0.15[V] = 3[V]; \tag{4.8}$$

risultato tranquillamente accettabile in quanto per questo valore di uscita e di alimentazione non viene introdotta ulteriore distorsione.

I risultati sperimentali ottenuti sono visibili in figura 4.6 da cui si possono notare i limiti di questo prototipo. Dovendo infatti abbassare di molto con il partitore la tensione in ingresso all'amplificatore ed essendo il prototipo montato su breadboard, si nota che il segnale accumula molto rumore, anch'esso amplificato.



(a) v_{amp} (in azzurro) e v_{R_4} (in giallo)

(b) v_{R_2} (in azzurro) e v_{R_4} (in giallo)

Figura 4.6: Acquisizioni sperimentali del funzionamento dell'amplificatore. Il valore del trimmer R_2 è inferiore a quanto calcolato nella 4.6.

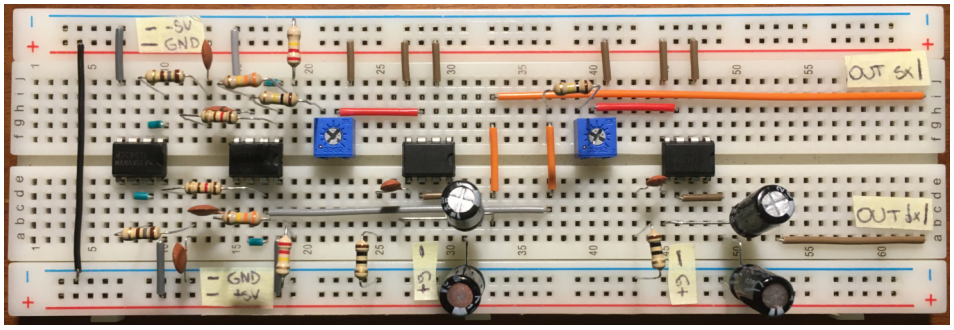


Figura 4.7: Realizzazione su breadboard dei circuiti di figura 4.1 e 4.5.

In figura 4.7, invece, è visibile il circuito di uscita, comprendente filtro e amplificatore montato su breadboard. Un occhio particolarmente attento si accorgerà del fatto che rispetto allo schema elettrico sono presenti due grossi condensatori elettrolitici ($220\ \mu F$) in più. Questi condensatori sono messi in parallelo tra l'alimentazione dell'*LM386* e massa, in modo da stabilizzarle il più possibile. Se non ci fossero, la traccia audio risulterebbe estremamente rumorosa o addirittura inascoltabile.

Capitolo 5

Conclusioni

Dopo aver analizzato ciascuna scheda analogica e aver visto come funziona il microcontrollore, si può passare alla conclusione con delle considerazioni finali.

Il prototipo è l'unione di tutte le schede viste fino ad ora ed è visibile in figura 5.1. Le schede sono state semplicemente collegate tra loro attraverso dei jumper. Naturalmente questo protipo, pur essendo funzionante, ha tutti i limiti di un'implementazione su breadboard per quanto riguarda il rumore ed in generale i disturbi (ad esempio non è presente un piano di massa). Inoltre la sezione di uscita, in particolar modo il circuito di amplificazione, non sfrutta al massimo le potenzialità dell'integrato *LM386* in favore di un minor numero di componenti e quindi di una semplicità del suo funzionamento.

In tutto questo bisogna però tener conto che quello realizzato è un *prototipo*; sicuramente se venisse disegnata una PCB a regola d'arte seguendo quindi tutti i consigli di layout presenti nei

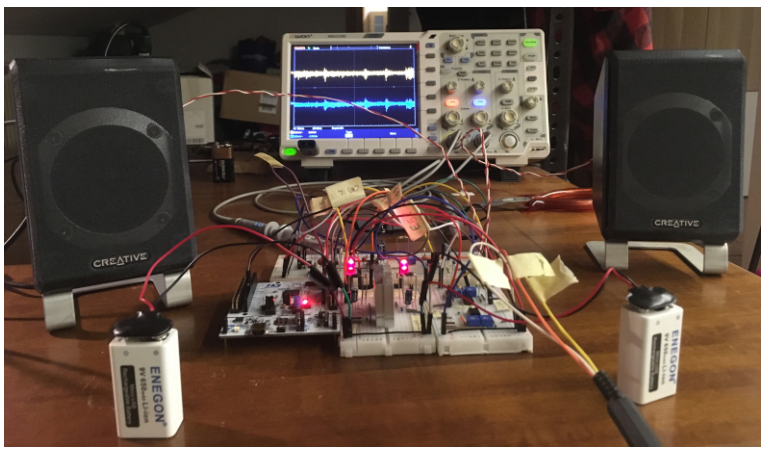


Figura 5.1: Prototipo dell'equalizzatore realizzato; si possono notare sullo schermo dell'oscilloscopio le tracce audio destra (in giallo) e sinistra (in azzurro).

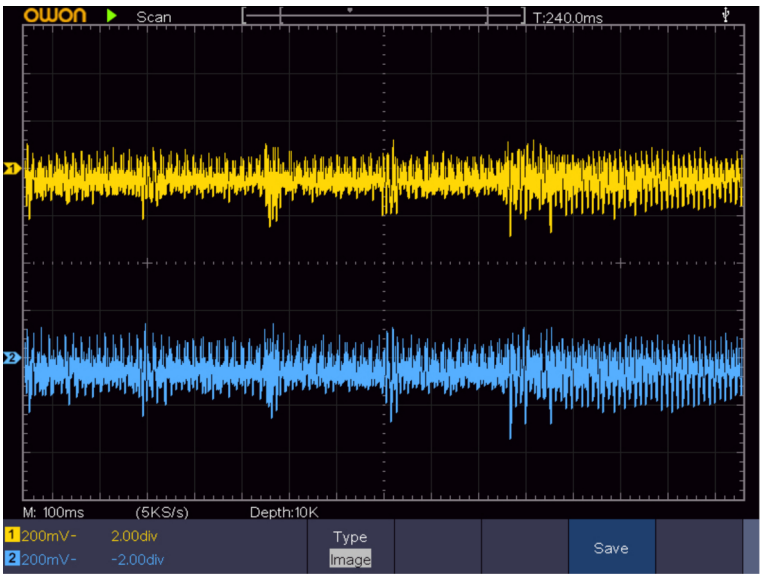


Figura 5.2: Acquisizione all'oscilloscopio di una traccia audio equalizzata. Le due tracce sono i differenti canali destro (in giallo) e sinistro (in azzurro).

datasheet dei vari componenti, il risultato finale ne gioverebbe molto. Non è però stato ritenuto un punto fondamentale per questa tesi il cui scopo è quello di *approfondire* il filtraggio digitale rispetto a quanto visto durante il corso di laurea, obiettivo da ritenersi raggiunto. Tutti gli altri punti, per quanto stimolanti e interessanti, erano una parte sì necessaria alla realizzazione del prototipo, ma non la parte principale.

A concludere questo lavoro di tesi è la figura 5.2 che mostra una traccia stereo equalizzata dal prototipo realizzato, acquisita all'oscilloscopio.

Bibliografia

[1] Buso, Simone (2018), *Esercitazioni con il microcontrollore STM32F334R8 per il corso di Microcontrollori e DSP*.

[2] Buso, Simone (2018), *Introduzione alle applicazioni industriali di MICROCONTROLLORI E DSP*, Esculapio.

[3] Digital Signal Processing, (2018), *Bilinear transform (Py)*, [YouTube, playlist] disponibile a: https://www.youtube.com/playlist?list=PLxWwb-b9LnpDoW0fbZYIC-vdx3U8dcdBa.

[4] Digital Signal Processing, (2017), *2nd order IIR filters (Py/C++)*, [YouTube, playlist] disponibile a: https://www.youtube.com/playlist?list=PLxWwb-b9LnpCGd9epAGZ4p0fCk9lyR3c-.

[5] FAIRCHILD SEMICONDUCTOR, (2001), *LM78xx datasheet (Rev 1.0.1)*, disponibile a: https://www.alldatasheet.com/datasheet-pdf/pdf/82833/FAIRCHILD/LM7805.html.

[6] STMicroelctronics, (2010), *STM32's ADC modes and theirapplications, application note (Rev 1)*, disponibile a: https://www.st.com/resource/en/datasheet/bat48.pdf.

[7] STMicroelctronics, (2011), *BAT48, datasheet (Rev 2)*, disponibile a: https://www.st.com/resource/en/datasheet/bat48.pdf.

[8] STMicroelctronics, (2010), *How to get the best ADC accuracy in STM32 microcontrollers, application note (Rev 8)*, disponibile a: https://www.st.com/resource/en/application_note/cd00211314-how-to-get-the-best-adc-accuracy-in-stm32-microcontrollers-stmicroelectronics.pdf.

[9] STMicroelectronics, (2021), *NUCLEO-xxxxRx, datasheet (Rev 15)*, disponibile a: https://www.st.com/en/evaluation-tools/nucleo-f334r8.html.

[10] STMicroelectronics, (2018), *STM32F334x8, datasheet (Rev 9)*, disponibile a: https://www.st.com/en/microcontrollers-microprocessors/stm32f334r8.html.

[11] STMicroelctronics, (2010), *STM32's ADC modes and their applications, application note (Rev 1)*, disponibile a: https://www.st.com/resource/en/application_note/an3116-stm32s-adc-modes-and-their-applications-stmicroelectronics.pdf.

[12] Texas Instruments, (2017), *LM386, datasheet (Rev may 2017)*, disponibile a: https://www.ti.com/lit/ds/symlink/lm386.pdf.

[13] Texas Instruments, (2013), *TL08xx, datasheet (Rev april 2013)*, disponibile a: https://www.ti.com/lit/ds/symlink/lm386.pdf.