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

Sviluppo di una piattaforma embedded di elaborazione audio digitale su microcontrollore-DSP

Relatore:
Prof. Simone Buso

Laureando:
Giulio Girardi
1187331

Anno Accademico 2020/2021
Data di laurea: 19/07/2022

Indice

Capitolo 1

Presentazione del sistema DSP

In questo documento viene descritto lo sviluppo e il funzionamento di una piattaforma di elaborazione digitale di segnale audio su microcontrollore, sviluppato durante l'esperienza di tirocinio svolta negli ultimi mesi del 2021, presso l'azienda *Protech Engineering* di Castelfranco Veneto.

L'oggetto principale dell'elaborato è un'analisi approfondita della sezione di elaborazione digitale. Questa, però, non può prescindere da una presentazione dell'architettura generale del sistema e dell'aspetto hardware, che è parte integrante del sistema, nel suo ruolo d'interfaccia tra gli algoritmi matematici implementati nel microcontrollore e la realtà fisica dei segnali che vengono elaborati, e da una sezione dedicata alla misura e alla caratterizzazione del sistema nel suo complesso attraverso metriche standard dell'ambito dell'audio professionale.

1.1 Architettura

La piattaforma in esame (fig. 1.1) è organizzata come un tradizionale sistema digitale: da una serie d'ingressi analogici viene prelevato il segnale audio, ulteriormente elaborato in seguito da uno stadio analogico, composto principalmente da stadi di amplificazione e di filtraggio. Il segnale risultante, ancora analogico, è poi convertito da un ADC in una sequenza digitale di bit, processata dal microcontrollore DSP, all'interno del quale sono eseguiti vari algoritmi digitali per l'elaborazione di questo flusso di dati. Tali algoritmi riguardano principalmente il filtraggio del segnale e l'applicazione di *effetti* tipicamente utilizzati nell'ambito dell'audio professionale. Il segnale digitale prodotto in uscita dal DSP è inviato a un DAC, che ne effettua la riconversione a un segnale analogico. Tale segnale, attraverso un altro stadio di amplificazione e filtraggio analogico, raggiunge l'us-

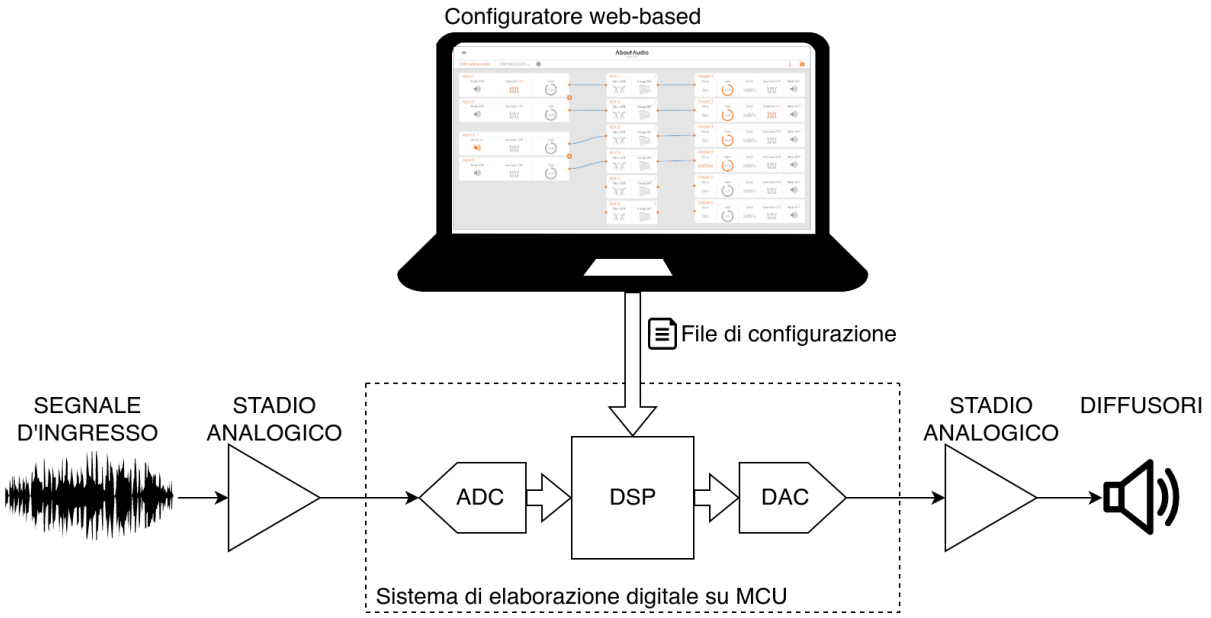


Figura 1.1: Schema del flusso del segnale

cita del sistema, generalmente collegata (tramite altri stadi di amplificazione, totalmente indipendenti) a dei diffusori sonori per la riproduzione del segnale audio.

La peculiarità che contraddistingue questa piattaforma è che la sezione di elaborazione digitale su DSP non è stata codificata in modo definitivo al momento della programmazione del microcontrollore. Al contrario, questa è stata architettata con una struttura altamente modulare che consente di scegliere in *run-time* quali algoritmi eseguire e con quali parametri eseguirli. Data la quantità notevole di parametri modificabili e di combinazioni che possono essere ottenute, la configurazione del sistema avviene in modo visivo tramite un'interfaccia grafica *web-based*, per una maggior portabilità e facilità d'uso. Tramite questa interfaccia è possibile infine generare un file di configurazione, che, caricato sul microcontrollore tramite una connessione USB, applica in tempo reale le impostazioni decise dall'operatore.

1.2 Presentazione hardware

Il cuore della piattaforma è implementato in due circuiti stampati differenti, individuabili nello schema logico in figura 1.2 e in figura 1.3, individuati coi termini *scheda carrier* e *scheda dsp* (la seconda si inserisce a incastro sulla prima, da cui il nome *carrier*):

- la ***scheda carrier*** contiene 2 connettori d'ingresso del segnale audio (e 2 uscite *pass-trough*), gli stadi analogici di filtraggio e amplificazione (in ingresso e in uscita) e i connettori di uscita, per un totale di 4 uscite;

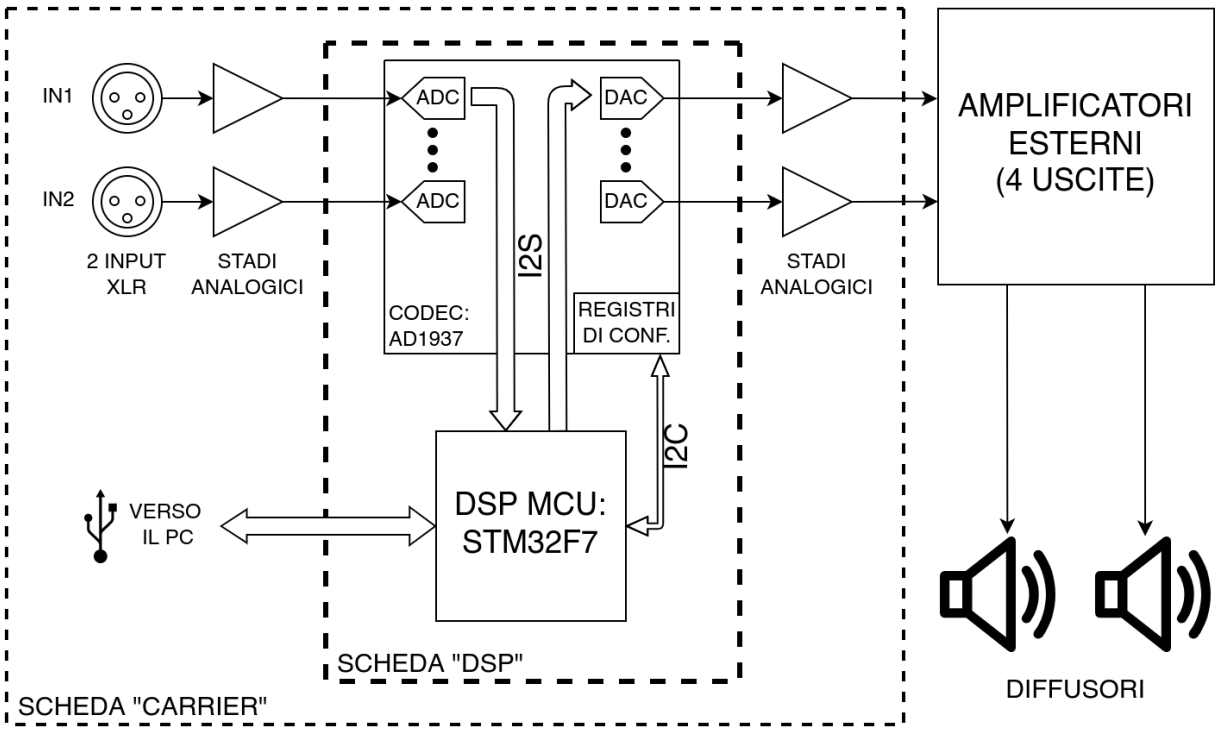
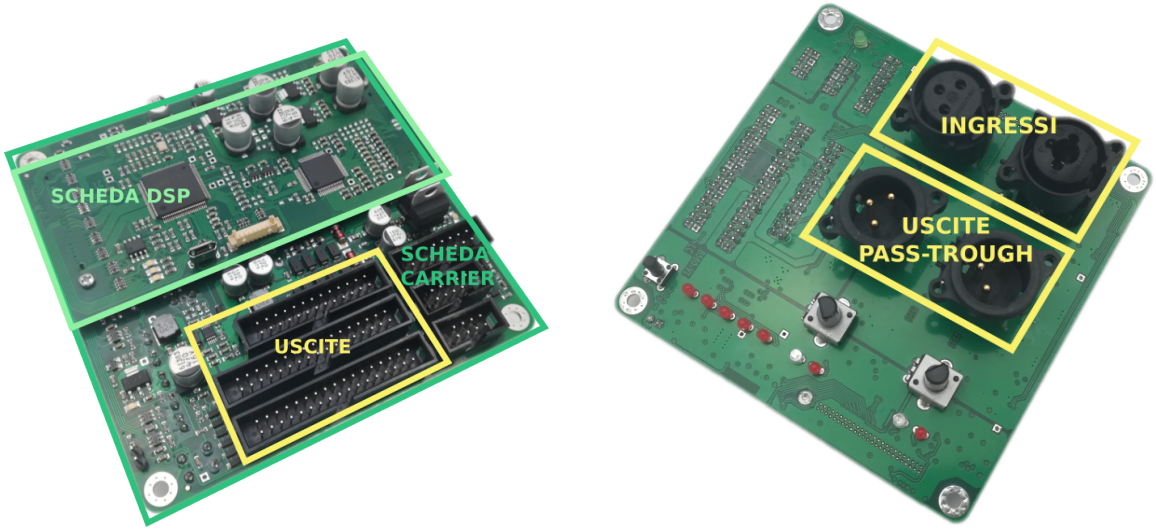


Figura 1.2: Schema logico e delle connessioni delle due schede "carrier" e "dsp"

- la ***scheda dsp*** contiene il microcontrollore (STM32F745, prodotto da ST Microelectronics), che effettua tutte le elaborazioni digitali sul segnale audio, e un codec audio (AD1937, prodotto da Analog Devices). Il codec, a cui è dedicato il capitolo 2, è il componente che converte il segnale analogico ricevuto dalla *scheda carrier* e lo trasforma in uno *stream* di bit (con protocollo I^2S) verso il microcontrollore e, analogamente, converte gli stream di bit generati dal microcontrollore in segnali analogici che sono poi propagati tramite la *scheda carrier* verso le uscite. È presente anche una connessione USB per la configurazione del sistema, che può essere generata tramite un apposito applicativo *web-based*.

Gli ingressi della *scheda carrier* sono dei connettori XLR, molto utilizzati nell'ambito dell'audio professionale. La caratteristica fondamentale di questi connettori è che trasportano un segnale analogico su una linea bilanciata, che consente di collegare dispositivi molto lontani senza perdita di qualità del segnale.

Le uscite, invece, non sono dei connettori standard (a differenza degli ingressi), poiché la *scheda carrier* è stata pensata per essere collegata a degli stadi di amplificazione esterni preesistenti, che pilotano i vari diffusori della cassa acustica in cui il sistema viene installato. Sempre per ragioni di qualità del segnale trasmesso, anche le uscite verso gli amplificatori esterni sono su linee bilanciate.



(a) Retro **(b)** Fronte

Figura 1.3: Foto delle due schede *carrier* e *dsp*

La divisione in due circuiti stampati ha un notevole vantaggio: la *scheda dsp*, infatti, è totalmente indipendente dalla *scheda carrier* e si configura come una piattaforma di elaborazione digitale *general purpose*. È sufficiente, infatti, riprogettare solamente la *scheda carrier* per adattare la piattaforma a qualunque combinazione di dispositivi in ingresso e in uscita, mantenendo completamente invariato l'hardware della *scheda dsp* e richiedendo modifiche minime al firmware che effettua l'elaborazione del segnale, con un notevole risparmio sui costi di progettazione.

Capitolo 2

Codec e trasmissione del segnale digitale

Sebbene il vero cuore di questa piattaforma sia il microcontrollore, all'interno del quale vengono eseguiti tutti gli algoritmi di elaborazione digitale, una funzione altrettanto importante viene svolta dal codec, che è un circuito integrato *esterno* al microcontrollore.

Il codec è appositamente progettato per effettuare la conversione del segnale analogico in segnale digitale (e viceversa) e svolge un ruolo d'interfaccia tra il dominio fisico, descrivibile da segnali puramente analogici e il dominio digitale, dove l'elaborazione di questi segnali (opportunamente trasformati) è molto più semplice.

Il passaggio al mondo digitale, tuttavia introduce una serie di limiti, principalmente relativi alla quantità e alla qualità dell'informazione contenuta nel segnale, che si applicano al momento del passaggio da un dominio all'altro. È importante quindi scegliere il codec prestando attenzione a una serie di caratteristiche (riportate nel foglio tecnico), poiché è principalmente questo componente a determinare le specifiche finali di tutto il sistema.

2.1 Codec: funzionamento e configurazione

Il codec che è stato scelto per questo progetto, l'AD1937[3], è un codec specifico per il settore dell'audio, ma non solo, prodotto dalla Analog Devices. Questo integrato è in grado di effettuare la conversione tra i domini digitale e analogico, in entrambe le direzioni: è infatti dotato di 4 ADC (conversione analogico-digitale) e di 8 DAC (conversione digitale-analogico), raggruppati logicamente in coppie di segnali destro e sinistro, che sono in grado di operare parallelamente.

L'*AD1937* è ampiamente configurabile, e i suoi DAC e ADC possono operare con una frequenza di campionamento fino a 192kHz (che significa, per il teorema del campionamento, che è in grado di effettuare la conversione di segnali con banda fino a 96kHz, ampiamente superiore alla banda tipica dei segnali audio) e con una dinamica fino a 24 bit (il segnale analogico è quantizzato in 2^{24} livelli). Nel caso specifico si è utilizzata la combinazione di 24 bit e 96kHz (da specifica del cliente).

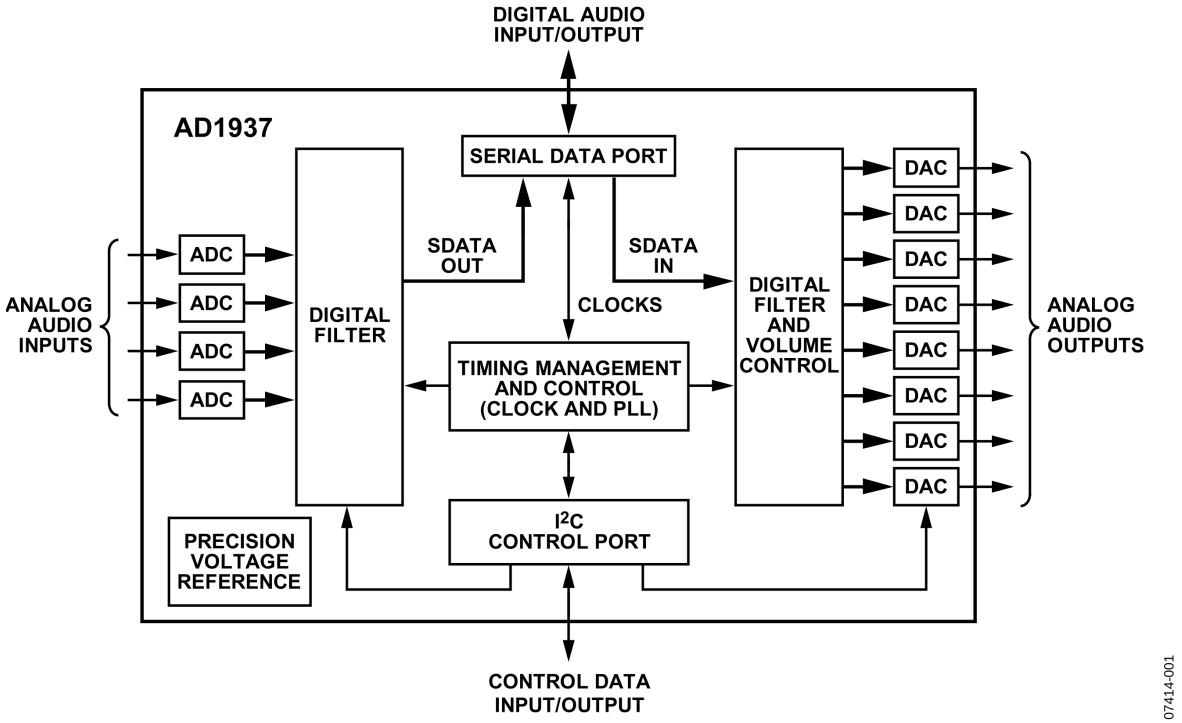


Figura 2.1: Schema logico del codec. [3]

2.1.1 Interfaccia I^2C di configurazione

Come si può vedere dalla figura 2.1, il codec è configurabile da un microcontrollore esterno, tramite una connessione I^2C (indicata dal blocco *I^2C control port*), che espone una serie di registri all'interno dei quali i valori di alcuni bit, come descritto nel foglio tecnico, corrispondono ad alcuni modi di operazione del codec. Il codec, che è uno *slave* nel bus, è controllato dal microcontrollore tramite una periferica I^2C interna configurata da *master*, che permette al firmware di configurare il codec prima di avviare la conversione del segnale.

2.1.2 Interfacce seriali per la trasmissione dell'audio digitale

Il segnale digitale viene scambiato col microcontrollore tramite alcune interfacce seriali (2 per gli ADC e 4 per i DAC), che possono essere configurate (tramite la porta I^2C) per operare in varie modalità e trasmettere utilizzando diversi protocolli, a seconda delle necessità del progettista. In breve, il codec può operare nei seguenti modi:

- **Utilizzando per ogni coppia di canali una singola interfaccia seriale**, per un totale di $2 + 4 = 6$ porte. Questa soluzione è la più semplice da implementare lato firmware, ma è onerosa da un punto di vista dell'hardware: infatti richiede di utilizzare un microcontrollore con almeno 6 periferiche seriali, e altrettanti gruppi di pin liberi da assegnare a ciascuna periferica.

  Nel caso specifico, dal momento che sono stati utilizzati esclusivamente 2 canali (L+R) d'ingresso e 4 d'uscita, per un totale di 3 interfacce e che il microcontrollore dispone di 5 periferiche seriali [7], questa è stata la configurazione scelta.

- In **modalità *TDM* (*time-division multiplexing*)**: una o più interfacce seriali vengono utilizzate a una velocità maggiore per trasmettere i dati generati da più convertitori con un numero inferiore di canali digitali. Questo risultato è ottenuto dividendo la dimensione temporale in una sequenza di *slot*, ciascuno assegnato a un canale differente. Il codec, quindi, invia a rotazione una parola di dati per ogni canale corrispondente allo *slot* valido (*multiplexing*).

  Questa modalità di operazione consente di utilizzare molte meno risorse hardware (pin e periferiche del microcontrollore), ma è più articolata da un punto di vista del software e dal momento che richiede di far funzionare il codec a una frequenza molto più elevata, impedisce l'uso di alcune frequenze di campionamento che non è possibile raggiungere.

- In **modalità *daisy-chain***: due codec possono essere collegati in serie per raggiungere fino a 16 canali DAC e 8 canali ADC. Questa modalità usa una ripartizione *TDM* del flusso di dati, in cui ogni *slot* è assegnato a una combinazione di codec e canale.

Ciascuna interfaccia seriale trasmette i dati attraverso tre segnali digitali:

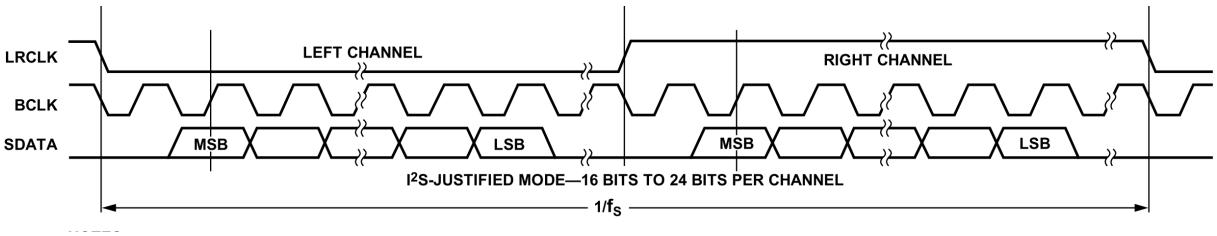


Figura 2.2: Segnali della porta seriale in modalità I^2S. [3]

- Un segnale di dati (**SDATA**), in cui viene trasmesso, in serie, ciascun campione del segnale digitale, bit per bit, coerentemente con i due segnali di clock descritti in seguito.

- Un segnale di clock relativo al bit (**BCLK**): il fronte di salita o di discesa di questo segnale (configurabile) segnala l'istante della trasmissione in cui nel segnale di dati è leggibile ciascuna parola del segnale audio digitale campionato.

  Siccome il codec è stato impostato a una frequenza di campionamento di 96kHz, e ogni interfaccia trasmette 32 bit (di cui al massimo 24 significativi) per due canali, la frequenza di questo segnale è $f_{\mathrm{BCLK}} = 2 \times 32 \times 96 = 6.144\mathrm{MHz}$.

- Un segnale di *framing* (**LRCLK**) che distingue i dati del canale destro dal canale sinistro: gli stati alto e basso di questo segnale indicano che i campioni trasmessi nella linea di dati appartengono rispettivamente al canale destro o sinistro (configurabile).

  La frequenza di questo segnale è $f_{\mathrm{LRCLK}} = 96\mathrm{kHz}$

Alcuni dettagli, legati all'allineamento o alla polarità dei tre segnali possono essere variati tramite i registri di configurazione I^2C per utilizzare diversi protocolli di trasmissione, ampiamente descritti nel foglio tecnico. Nel caso specifico si è usato il protocollo I^2S (*inter-integrated sound*), mostrato in figura 2.2, che tutte le periferiche seriali dal lato del microcontrollore sono in grado d'interpretare.

Per il corretto funzionamento del codec è necessario un ultimo segnale, detto *master clock* (**MCLK**). Come si intuisce dal nome, questo segnale permette all'intero integrato di funzionare. Il *master clock* può essere generato internamente e fornito su un pin d'uscita, (ad esempio nel caso in cui il codec venga configurato come *master*, cioè quando genera i segnali *BCLK* e *LRCLK*), oppure può essere fornito dall'esterno tramite un apposito pin d'ingresso (generalmente quando il codec è *slave*: in questo caso *MCLK*, *LRCLK* e *BCLK* sono tutti generati dal microcontrollore, che è invece *master*). Esistono altre combinazioni, particolarmente esotiche, che non saranno trattate. Il *master clock*, se esterno, può essere

utilizzato come riferimento in un PLL interno al codec, che viene utilizzato per generare un più accurato segnale di clock interno, in cui un eventuale *jitter* del segnale originale viene attenuato.

Nel caso specifico il microcontrollore è stato configurato come *master* e genera tutti i segnali di clock, mentre il codec come *slave*; il *master clock* quindi è generato dal microcontrollore (a una frequenza $f_{\mathrm{MCLK}} = 12.288\mathrm{MHz}$, calcolabile dal foglio tecnico), ma utilizzato tramite il PLL del codec per attenuare l'eventuale *jitter*.

2.2 Microcontrollore e ricezione del segnale digitale

Il codec si occupa esclusivamente della trasformazione dei segnali dal dominio analogico al dominio digitale (e viceversa). L'elaborazione del segnale convertito viene eseguita dal microcontrollore DSP, che lo riceve dal codec tramite delle linee digitali che collegano le periferiche seriali del codec a delle analoghe periferiche del microcontrollore.

Il microcontrollore utilizzato in questo progetto è un STM32F7[7], prodotto dalla STMicroelectronics. Questo microcontrollore non è esclusivamente dedicato ad applicazioni DSP, ma più in generale di tipo industriale. Tuttavia, la presenza di un *core* ARM Cortex-M7, dotato di una *FPU* (*floating point unit*, una periferica hardware dedicata per i calcoli in virgola mobile), l'elevata frequenza di operazione (216 MHz), e la presenza di specifiche istruzioni assembly per l'elaborazione digitale, lo rendono un'ottima scelta anche per le applicazioni DSP.

2.2.1 Periferiche seriali

Lo scambio del segnale digitale col codec, dal punto di vista del microcontrollore, viene gestito da una serie di periferiche integrate in grado d'interpretare il protocollo I^2S e di trasferire il contenuto in bit del segnale nella (o dalla, nel caso dei segnali diretti ai DAC) RAM del microcontrollore per una sua successiva elaborazione.

Come già anticipato nel paragrafo 2.1.2, nel caso specifico sono state utilizzate 3 linee seriali digitali, che sono collegate ad altrettante periferiche seriali integrate del DSP. Data la genericità del microcontrollore, tali periferiche sono in grado di gestire numerosi protocolli e funzioni avanzate non necessarie nel caso specifico. Nel progetto ne sono state utilizzate di due tipologie, una periferica I^2S e due periferiche SAI:

- Le periferiche **I²S (/SPI)** sono le periferiche più avanzate, e in questo microcontrollore possono operare alternativamente come semplici periferiche SPI (*serial peripheral interface*), utilizzate generalmente per la comunicazione tra diversi circuiti integrati, o come periferiche I²S (*inter-integrated sound*), che sono un'estensione delle periferiche SPI e aggiungono la generazione di segnali di *master clock* (**MCLK**) e *framing* (**LRCLK**), e sono invece specificamente utilizzate per la trasmissione di flussi di dati audio digitali.

- Le periferiche **SAI** (*serial audio interface*) sono periferiche specifiche per la trasmissione e la ricezione di flussi di dati audio digitale, in grado di funzionare in modalità I²S, ma anche utilizzando numerosi altri protocolli e modalità di operazione comuni nelle applicazioni di audio digitale.

2.2.2 Trasferimento dei campioni: DMA

Le periferiche seriali, una volta avviate, rendono disponibile alla frequenza di campionamento, su uno specifico registro di lettura dati, ciascun campione prodotto dal codec. Allo stesso tempo, le periferiche seriali d'uscita (quelle collegate ai DAC del codec), trasmettono alla frequenza di campionamento ciascun campione elaborato dal DSP, recuperandolo da uno specifico registro di scrittura dati, che deve essere riempito alla giusta frequenza dal firmware.

Gestire "manualmente" il trasferimento dei campioni dalle periferiche d'ingresso alla RAM, per poi elaborarli singolarmente e scrivere nel registro di scrittura della periferica d'uscita il nuovo valore del campione sarebbe molto oneroso, considerando che, nel caso migliore (e teorico), il tempo disponibile per l'elaborazione è al massimo il periodo di campionamento. Se il nuovo campione non viene fornito in tempo, generalmente viene ritrasmesso quello precedente, producendo degli artefatti sonori (colloquialmente detti "click").

Per una maggiore efficienza di elaborazione, innanzitutto, i campioni vengono trasferiti dai registri delle periferiche alla RAM da una periferica detta DMA (*direct memory access*) che consente il trasferimento dei dati dalla periferica alla RAM in modo diretto, cioè senza che la CPU debba eseguire alcuna istruzione (guadagnando una notevole quantità di tempo macchina). In secondo luogo, la periferica DMA viene impostata per operare in modalità *double-buffering*.

In questa modalità, innanzitutto, i campioni non vengono processati individualmente, ma in *buffer*, cioè in sequenze di campioni, che vengono completamente acquisiti prima di essere elaborati. Il vantaggio è che tutte le operazioni vengono eseguite in blocco sull'intero *buffer*, con un *overhead* molto inferiore rispetto a un'esecuzione campione per campione.

Ogni operazione di elaborazione digitale programmata nel firmware, infatti, è generalmente contenuta in una *funzione*. L'esecuzione di una *funzione*, oltre al tempo necessario a eseguire le operazioni definite dal programmatore (in questo caso l'elaborazione dei campioni digitali), introduce un *overhead* dato dal tempo di accesso alla memoria, dalla lettura delle istruzioni dalla memoria di programmazione, dall'allocazione dello *stack* e da molte altre piccole operazioni necessarie per consentirne l'esecuzione. Elaborando i campioni in blocco tale *overhead* viene ridotto (anche solo in termini di numero d'invocazioni di ciascuna funzione), e tutto il tempo recuperato può essere utilizzato per l'elaborazione del segnale.

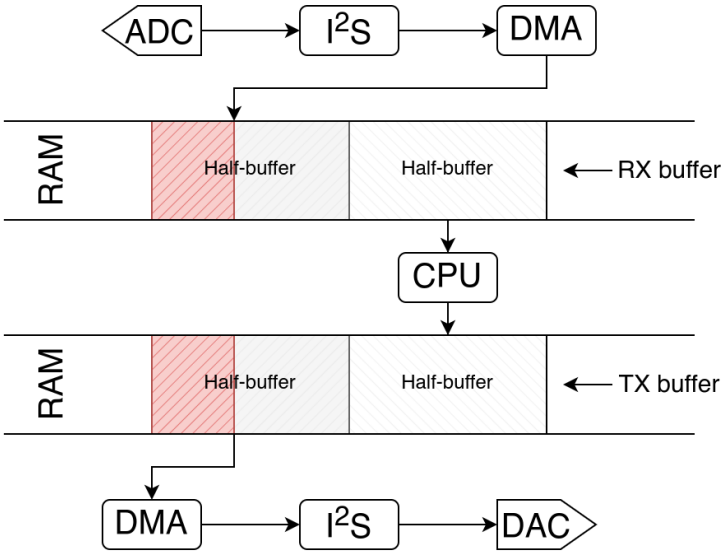


Figura 2.3: *Double-buffering*: ricezione e trasmissione del primo *half-buffer* mentre la CPU elabora il secondo

In secondo luogo, utilizzando la tecnica del *double-buffering*, ciascun *buffer* viene diviso in due *half-buffer*, che vengono scritti (o letti, nel caso dei DAC) alternativamente, in modo tale che mentre il DMA accede a uno degli *half-buffer* l'altro sia "libero", e accessibile alla CPU che quindi non rischia di operare su dati corrotti (dal momento che il DMA lavora indipendentemente dalla CPU, non è infatti possibile sapere quale sia lo stato dei dati copiati finché questo non ha terminato la sua operazione).

Dalla figura 2.3 è possibile vedere il funzionamento generale del sistema: a ciascuna periferica seriale è associato un *double-buffer*. Mentre il DMA trasferisce il segnale digitale dal codec al DSP (e viceversa) per ciascuna periferica d'ingresso (e uscita), utilizzando solo un *half-buffer*, la CPU elabora il segnale nell'altro *half-buffer* di ciascuna periferica d'ingresso e salva il segnale risultante nell'*half-buffer* dedicato alla periferica d'uscita, pronto per essere mandato al codec nel momento in cui il trasferimento corrente è stato completato.

L'uso di *buffer* ha però uno svantaggio non trascurabile: il maggior tempo disponibile per l'elaborazione e la maggior resilienza alle interruzioni ad alta priorità da parte della CPU, infatti, sono controbilanciate da un maggior ritardo tra ingresso e uscita, che è pari alla dimensione del *buffer* per il periodo di campionamento ($\Delta t_{delay} = \frac{S_b}{F_s} = \frac{384}{96000} = 4\text{ms}$ nel caso specifico).

Per migliorare ulteriormente le prestazioni del trasferimento dei campioni digitali tramite il DMA, è stata sfruttata un'ulteriore ottimizzazione messa a disposizione dal microcontrollore utilizzato (generalmente presente in molti microcontrollori della famiglia ARM): l'area di memoria degli STM32F7 è infatti divisa in varie regioni con diverse caratteristiche [8]. La regione più rilevante per questa applicazione è la regione di RAM denominata DRAM-TCM (*data RAM on tightly coupled memory interface*). Tale regione di memoria è strettamente collegata alla CPU e, per la precisione, non è dotata di *cache*. Nel firmware, i buffer gestiti dal DMA sono stati posizionati in questa regione: ciò significa che da un lato la CPU risparmia il tempo dei *cache miss*, rendendo deterministico il tempo di accesso, e dall'altro riduce la probabilità di corruzione dei dati. Se questa regione fosse dotata di *cache*, infatti, potrebbe verificarsi un aggiornamento dei dati in memoria da parte del DMA senza l'invalidazione della *cache*, causando la lettura di dati errati da parte della CPU (che avviene tramite la *cache*, quando presente).

2.2.3 Sincronizzazione delle periferiche

Dal momento che il sistema elabora più canali di trasmissione contemporaneamente, è fondamentale che il trasferimento dei campioni a opera del DMA avvenga in modo sincronizzato tra ciascun canale. Infatti la CPU, prima di avviare il processo di elaborazione del segnale, deve attendere che *tutti* gli *half-buffer* delle periferiche dei vari canali siano stati trasferiti (fig. 2.4). Un eventuale ritardo di una qualsiasi delle periferiche viene quindi sottratto al tempo di processing: infatti il trasferimento degli *half-buffer* viene

completato con un periodo $T_{DMA} = \frac{S_{hb}}{F_s}$ costante (che dipende dalla dimensione S_{hb} del *buffer* e dalla frequenza di campionamento) e il periodo disponibile per l'elaborazione è $T_{proc} = T_{DMA} - \Delta t_{sync}$, dove Δt_{sync} è il ritardo di sincronizzazione (fig. 2.5). L'obiettivo è di ridurre il ritardo di sincronizzazione per massimizzare il tempo disponibile per il processing.

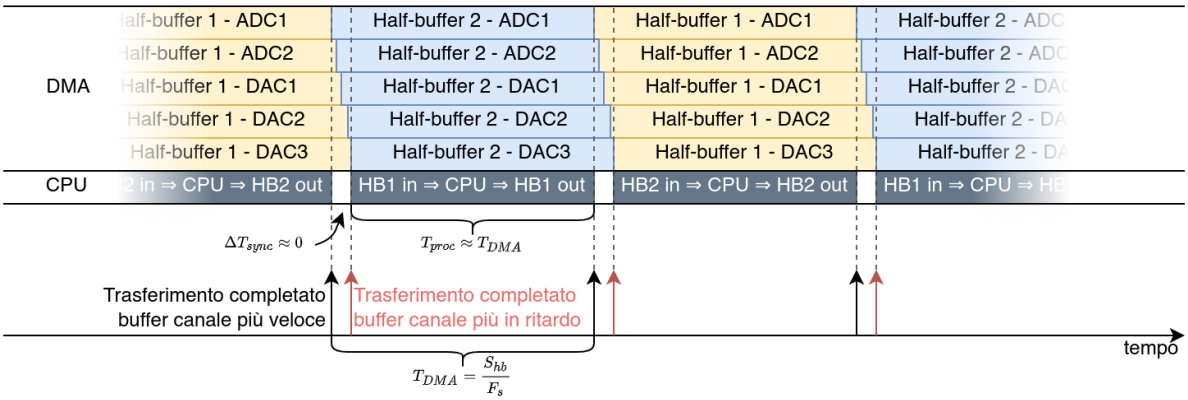


Figura 2.4: Trasferimenti DMA ed elaborazione CPU con periferiche sincronizzate

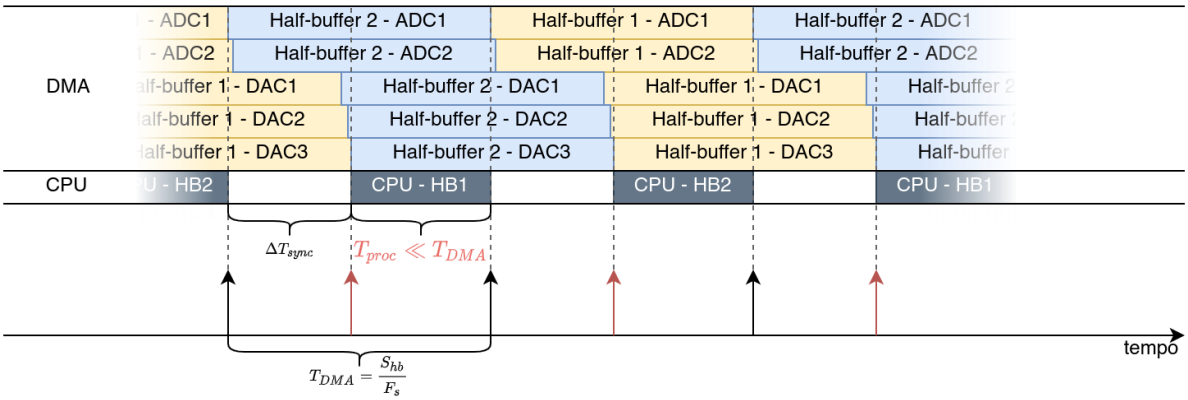


Figura 2.5: Trasferimenti DMA ed elaborazione CPU con periferiche non sincronizzate - il tempo disponibile per il processing è molto ridotto

Il modo più semplice per garantire questa specifica è che tutti i canali seriali condividano i segnali di clock, dal momento che sono questi a determinare il funzionamento del codec (è questa infatti la definizione di *slave* nel *bus*, funzione svolta dal *codec*). Esistono dei sistemi di sincronizzazione interna al microcontrollore [8], però questi si applicano solamente ad alcune combinazioni delle varie periferiche utilizzate. La soluzione adottata dunque è di tipo hardware: le due linee di clock (*BCLK* e *LRCLK*) di ciascun canale seriale, sono state unite al livello del circuito stampato. In fase di configurazione delle periferiche, una sola di esse è stata configurata come *master*, in modo che generi i segnali di clock, mentre tutte le altre sono state impostate come *slave*, per essere tutte sincroniz-

zate con il *master*. Avviando infine le periferiche configurate come *slave* per prime, si ha che quando la periferica *master* viene a sua volta avviata e genera i segnali di clock, le altre inizino contemporaneamente a riempire ciascun *half-buffer*.

Un'ulteriore complicazione è data dal fatto che a livello *hardware*, i canali seriali relativi agli ADC sono completamente separati da quelli dei DAC (le linee di clock sono fisicamente separate). L'unica banale soluzione per la sincronizzazione, in questo caso, è di avviare (a livello firmware) le periferiche il più possibile nello stesso istante, in modo tale che i *buffer* siano riempiti quasi contemporaneamente, con un ritardo trascurabile.

Capitolo 3

Elaborazione digitale del segnale

In questo capitolo viene presentato il sistema di elaborazione digitale al cuore di questa piattaforma, implementato sul microcontrollore ARM STM32F7. Come già anticipato, si tratta di un microcontrollore *general purpose*, utilizzato prevalentemente in ambito industriale, ma altrettanto adatto all'applicazione di elaboratore audio DSP se abbinato, come in questo caso, a un codec audio esterno.

Il vantaggio di utilizzare un microcontrollore generico è che non è limitato a svolgere unicamente le operazioni di elaborazione del segnale, ma può essere utilizzato per fornire funzioni aggiuntive, quali ad esempio la gestione di una porta USB, per consentire la configurazione tramite PC, o il monitoraggio di parametri di corretto funzionamento dell'hardware. Per ovvie ragioni tutte queste attività hanno una priorità inferiore all'attività di elaborazione, che deve essere in grado di fornire i campioni processati entro un tempo specifico e dunque può interromperle in qualsiasi momento non appena un nuovo *buffer* di campioni è disponibile.

Tali funzioni secondarie, per quanto fondamentali al corretto funzionamento dell'intero sistema, sono state citate per completezza ma non saranno affrontate in questo capitolo, in cui sarà approfondita unicamente la sezione di elaborazione.

3.1 Panoramica del sistema

Il sistema di elaborazione DSP (schematizzato in figura 3.1) è organizzato in tre macroblocchi, ciascuno contenente un certo numero (2 o 4) di catene di *effetti* (per *effetto* si intende una generica operazione di elaborazione digitale che può essere considerata come un blocco indipendente dal resto del sistema, dotato di un *ingresso*, di un'*uscita* ed eventualmente di *parametri* di configurazione), in ciascuna catena gli effetti sono collegati in

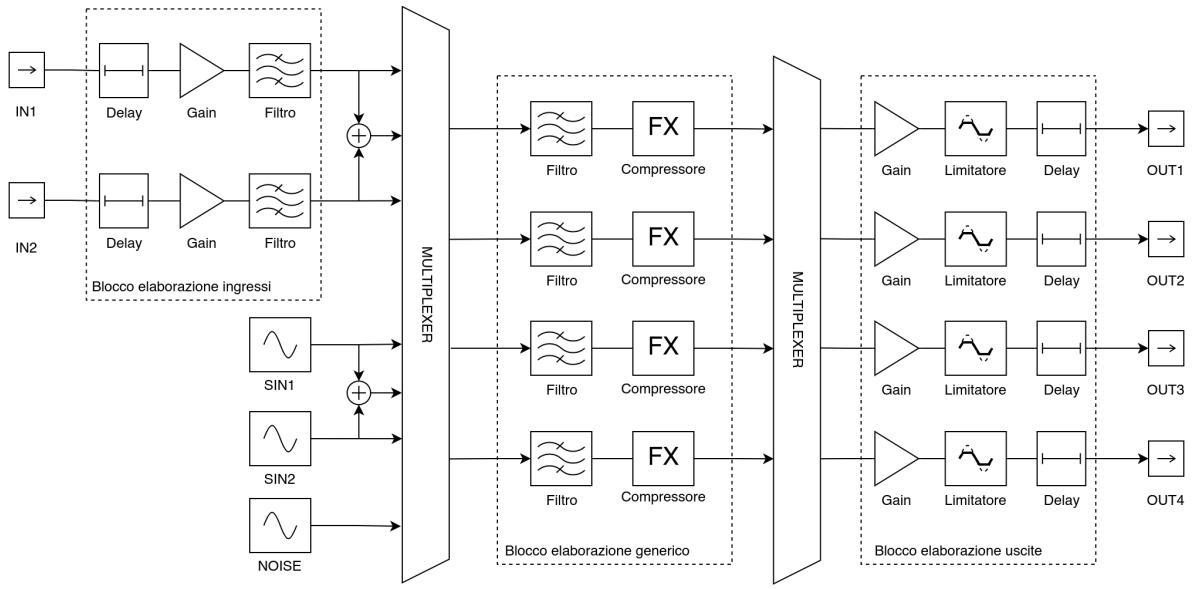


Figura 3.1: Diagramma della *pipeline* di elaborazione del sistema DSP

serie e completamente configurabili tramite USB da un PC esterno. I tre macroblocchi sono:

- Un **blocco d'ingresso**, in cui ciascuna catena è collegata direttamente a uno degli ingressi del sistema, serviti dai 2 ADC. In ciascuna catena sono presenti in sequenza un **delay**, che è un *effetto* che ritarda il segnale di un certo numero di millisecondi, un **guadagno**, che scala i segnali di una costante, e una **sezione di filtraggio**, che può essere configurata per operare come un filtro FIR (fino a un ordine di 1023) oppure come un filtro IIR biquadratico.

- Un **blocco generico di elaborazione**, in cui sono presenti quattro catene di effetti indipendenti, ciascuna dotata di un ingresso e di un'uscita collegabili tramite dei *multiplexer* a una qualsiasi delle catene dei blocchi d'ingresso o uscita. In ciascuna di queste catene sono presenti una **sezione di filtraggio**, configurabile esclusivamente come filtro IIR biquadratico (a causa di vincoli di memoria del microcontrollore) e un **compressore** (il compressore è un *effetto* che abbassa dinamicamente il guadagno del segnale quando questo supera un certo limite).

- Un **blocco d'uscita**, in cui l'uscita di ciascuna catena è direttamente collegata a una delle 4 uscite del sistema, servite dai 4 DAC. Ciascuna di queste catene è dotata di un **guadagno**, di un **limitatore** (il limitatore è un caso particolare di compressore, in cui il guadagno del segnale, se questo supera un certo limite, non viene semplicemente ridotto ma riportato esattamente al limite) e di un **delay**.

In aggiunta a questi tre macroblocchi, per semplificare operazioni di misura e di debugging sono disponibili anche due generatori indipendenti di segnali sinusoidali e un generatore di rumore, collegati logicamente al *multiplexer* tra il blocco d'ingresso e il blocco generico di elaborazione, che possono essere utilizzati per testare i vari effetti.

3.2 Librerie ARM CMSIS DSP

Per l'implementazione di buona parte degli algoritmi precedentemente elencati si è fatto uso di una specifica libreria fornita da ARM, la CMSIS-DSP Software Library [4], progettata appositamente per sfruttare al meglio il set d'istruzioni del core ARM Cortex-M all'interno del microcontrollore scelto per il progetto.

In secondo luogo, utilizzare una libreria consente di programmare le routine di elaborazione senza doversi preoccupare eccessivamente della corretta implementazione di algoritmi già universalmente noti, riducendo notevolmente il tempo necessario per lo sviluppo.

Tale libreria fornisce, generalmente, per ogni routine di elaborazione digitale multiple versioni, una per ciascun tipo di dato supportato: interi a 8 bit, interi a 16 bit, interi a 32 bit e numeri in virgola mobile a 32 bit (*floating point*).

Le funzioni disponibili in questa libreria sono due per ciascun algoritmo: la prima, identificata col suffisso `_init`, serve a inizializzare una `struct` in memoria, che contiene eventuali coefficienti, *buffer* o variabili di stato di ogni istanza del relativo algoritmo. La seconda funzione (senza il suffisso) accetta come parametri la `struct` di dati relativa all'algoritmo e due puntatori a dei *buffer*: uno d'ingresso che viene elaborato dalla funzione (senza modificarlo) e uno d'uscita, dove viene salvato il risultato dell'elaborazione.

3.2.1 Rappresentazione numerica del segnale

Il codec, che è a 24 bit, genera (e accetta) dei campioni interi a 24 bit con segno. Siccome il canale di comunicazione I^2S è a 32 bit il segnale viene comunque trasmesso a 24 bit effettivi, ma allineati a sinistra con un *padding* di 8 bit, nella regione degli LSB (*least significant bit*).

Una volta ricevuti dal microcontrollore, i campioni non ancora elaborati vengono salvati nei *buffer* in RAM mantenendo la rappresentazione come interi a 32 bit allineati a sinistra.

Per una maggiore semplicità degli algoritmi di elaborazione (non è necessario considerare problemi di scalatura e quantizzazione), e grazie alla presenza di una FPU (*floating point unit*, ovvero una periferica hardware per eseguire le operazioni tra i numeri in virgola mobile) si è scelto di convertire i campioni dal formato `int32_t` (interi a 32 bit con segno) al formato `float32_t` (numero a virgola mobile nello standard IEEE 754), riportandoli nell'intervallo $[-1, 1]$. La conversione tra i due formati si ottiene dividendo il valore intero per 2^{31} (l'effetto di questa operazione è di spostare la virgola decimale di 31 bit a sinistra, non 32 perché l'ultimo bit è quello di segno).

La rappresentazione in virgola mobile è molto differente dalla rappresentazione intera, pertanto non è scontato che le due rappresentazioni si equivalgano senza perdita d'informazione; bisogna infatti verificare che:

- La quantità di numeri rappresentabili tra -1.0 e 1.0 in virgola mobile sia almeno pari alla quantità d'interi rappresentabili con 24 bit. Nei `float32_t` i 23 bit di *mantissa* e il bit di segno permettono di rappresentare 2^{24} numeri diversi. Per quanto riguarda l'esponente sono possibili 127 scelte poiché il valore deve essere nell'intervallo $[-1, 1]$ e quindi l'esponente deve essere negativo o nullo. Si ha dunque che:

$$N_{\text{float32\_t}} = 127 \times 2^{24} > 2^{24} = N_{\text{int24\_t}} \tag{3.1}$$

- La distribuzione dei valori in virgola mobile sia tale da contenere tutti i valori presenti anche nella rappresentazione intera (opportunamente scalati). Infatti nella rappresentazione a virgola mobile c'è una densità di punti molto più elevata in un intorno di 0, che verso gli estremi dell'intervallo. La distanza tra due numeri adiacenti in rappresentazione intera (scalati nell'intervallo $[-1, 1]$) è $\delta_{\text{int24\_t}} = \frac{1-(-1)}{2^{23}}$. La distanza massima agli estremi dell'intervallo $[-1, 1]$ nella rappresentazione a virgola mobile è invece data dalla seguente equazione:

$$\delta_{\text{float32\_t}} = 2^{1-n+e} = 2^{1-23+0} = \frac{1}{2^{22}} = \delta_{\text{int24\_t}} \tag{3.2}$$

  dove $e = 0$ è l'esponente della rappresentazione a virgola mobile degli estremi dell'intervallo, ± 1.0, rappresentati come $\pm 1.0 \times 2^0$, $n = 23$ è il numero di bit nella mantissa.

Tali due proprietà assicurano che ogni numero rappresentabile da un intero a 24 bit sia rappresentabile anche da un numero a virgola mobile a 32 bit quando scalato nell'intervallo [-1,1].

3.3 Analisi di alcuni algoritmi DSP

Nella sezione seguente saranno analizzati alcuni dei principali algoritmi DSP che sono stati implementati o rielaborati durante l'attività di tirocinio.

Nella scelta degli algoritmi si è dovuto individuare un delicato compromesso tra la configurabilità di quest'ultimi e l'utilizzo del tempo di calcolo disponibile: infatti, come descritto nel capitolo 2.2, l'elaborazione viene eseguita su dei *buffer* divisi in due *half-buffer*. Mentre un *half-buffer* è riempito dal DMA, l'altro è libero per essere elaborato dalla CPU, che ha un tempo $T_{DMA} = \frac{S_{hb}}{F_s} = 2\text{ms}$ per portare a termine tutte le elaborazioni prima che tale *half-buffer* venga trasmesso dal DMA al DAC.

Lo strumento più utile per misurare la prestazione degli algoritmi è stata una periferica nativa del core Cortex-M, denominata DWT (*Data Watchpoint and Trace*), che contiene al suo interno un contatore dei cicli di clock. Per misurare l'impatto sul tempo macchina di ciascun algoritmo è sufficiente calcolare la differenza tra il valore del contatore prima e dopo l'esecuzione dell'algoritmo: dividendo tale valore per la frequenza di operazione del microcontrollore è possibile determinare le tempistiche di esecuzione di ogni operazione.

3.3.1 Calcolo RMS e filtraggio DC

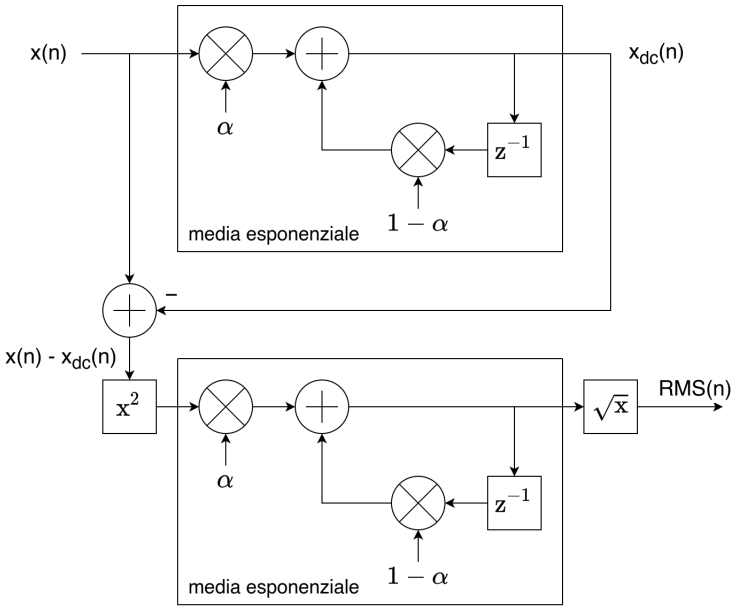


Figura 3.2: Diagramma a blocchi del calcolo dell'RMS (sotto) e del filtraggio DC (sopra)

A scopo di misurazione e di test, all'interno della piattaforma è disponibile una funzione per misurare l'ampiezza del segnale di qualunque ingresso. Tale funzione effettua il calcolo del valore RMS del segnale campionato, riportandolo in scala logaritmica

(viene utilizzata l'unità di misura dBFS, ovvero decibel relativi al fondo scala del segnale digitale).

Il calcolo della media di un segnale per ottenere l'RMS, dovrebbe essere fatto su tutto il suo supporto, ma dal momento che si tratta di un segnale in tempo reale, dove ciò non è realizzabile (dal momento che il supporto ha teoricamente un'estensione infinita), si sostituisce l'operazione di media aritmetica con una media esponenziale (evidenziata in figura 3.2) per dare un peso maggiore ai campioni più recenti, tenendo comunque conto dell'evoluzione del segnale.

È necessario poi aggiungere un ulteriore stadio di filtraggio alla misura: per quanto il codec sia disaccoppiato in continua dagli ingressi (e uscite) analogici, infatti, per costruzione esso riporta comunque una componente continua spuria nel segnale digitale, che tende a sfalsare notevolmente la misura dell'RMS (anche perché la componente continua non ha senso in un sistema di elaborazione audio). Per questo è stato implementato uno stadio che sottrae al segnale d'ingresso la sua componente continua (calcolata con un'altra media esponenziale) prima di fornirla allo stadio di misura RMS.

3.3.2 Filtri IIR biquadratici

I filtri IIR biquadratici (che per brevità saranno in seguito indicati col termine tecnico *biquad*), sono un sottoinsieme dei filtri IIR (acronimo che sta per *infinite impulse response*, ovvero *risposta impulsiva infinita*), per la precisione sono i filtri IIR del secondo ordine, aventi due poli e due zeri [6].

In termini di trasformata z, un filtro *biquad* viene normalmente rappresentato con la seguente funzione di trasferimento [2], utilizzata generalmente anche nei calcoli:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \tag{3.3}$$

Le funzioni utilizzate dalla libreria ARM CMSIS DSP per implementare nel software gli stadi di *biquad*, tuttavia, implementano questi filtri eseguendo la seguente equazione alle differenze [4]:

$$y[n] = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + a_1 y[n-1] + a_2 y[n-2] \tag{3.4}$$

che in termini di trasformata z corrisponde alla funzione di trasferimento

$$H(z) = \frac{1 - a_1 z^{-1} - a_2 z^{-2}}{b_0 + b_1 z^{-1} + b_2 z^{-2}} \tag{3.5}$$

Per passare da una forma all'altra è sufficiente normalizzare i coefficienti dell'eq. 3.3 a b_0 e invertire i due coefficienti al numeratore.

I filtri IIR sono molto usati, poiché generalmente richiedono pochi coefficienti (che occupano spazio in memoria) e conseguentemente anche poche operazioni per l'esecuzione (nel caso della *biquad* implementata con l'eq. 3.4 si tratta solo di 5 somme e moltiplicazioni).

Si tende in generale a evitare l'implementazione di filtri di ordine elevato, poiché sono più soggetti a errori di quantizzazione e problemi di stabilità [5]. Per questo motivo si cerca piuttosto di utilizzare filtri del secondo ordine (*biquad*) in cascata. Il passaggio da un filtro di ordine elevato a una cascata di *biquad* è semplice: è sufficiente estrarre dalla funzione di trasferimento del primo coppie di poli e zeri che diventano i due poli e zeri di ciascun filtro biquadratico in cascata.

Le funzioni della libreria ARM CMSIS DSP adottano questo approccio; nel caso particolare di questa piattaforma sono state inserite, in ciascun blocco di filtraggio, 12 *biquad* in cascata individualmente configurabili.

Calcolo dei coefficienti

Per una maggior semplicità di configurazione, i blocchi di filtri biquadratici non sono direttamente configurabili attraverso i coefficienti della funzione di trasferimento. Questi vengono invece calcolati, al momento della configurazione, da una routine specifica che riceve dai 2 ai 4 parametri che descrivono il funzionamento del filtro in modo intuitivo per l'operatore. Tali parametri sono:

- La **tipologia** del filtro: *passa-alto*, *passa-basso*, *passa-banda*, *passa-tutto*, *notch*, *high-shelf*, *low-shelf*, *peaking* etc. (ciascuna di queste tipologie è ottenibile con un filtro *biquad*).

- La **frequenza** f_0 di applicazione, ovvero la frequenza di taglio nel caso dei filtri *passa-basso* e *passa-alto*, la frequenza di centro banda nel caso del *passa-banda*, etc.

- Un **guadagno** G, che si applica solo ad alcuni filtri, quali *high-shelf*, *low-shelf* e *peaking*.

- Un **fattore di qualità** Q, che si applica solo ad alcuni filtri, come *passa-alto* e *passa-basso*, di cui controlla lo smorzamento della risonanza, il *passa-banda* del quale

controlla l'ampiezza della banda passante e i filtri *peaking* e *notch*, di cui controlla la larghezza del picco.

Le formule per il calcolo dei coefficienti sono state prese direttamente da [2], a cui si rimanda per i calcoli esatti e le funzioni di trasferimento di ciascun filtro. In generale, però, la tecnica utilizzata è quella della **trasformata bilineare** [5]:

- Per prima cosa ciascun filtro viene rappresentato nel dominio analogico (ovvero il dominio della trasformata di Laplace) con la sua caratteristica funzione di trasferimento, parametrizzata secondo i valori elencati precedentemente.

- Alla funzione di trasferimento in s viene applicata la trasformata bilineare, che mappa lo spazio della trasformata s nello spazio della trasformata z, operando la seguente sostituzione:
$$s = \frac{2}{T_s}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right) \tag{3.6}$$
dove T_s è il periodo di campionamento.

- La nuova funzione di trasferimento in z viene riportata alla forma del filtro *biquad*, per estrarre il valore di ciascun coefficiente in termini dei parametri del filtro analogico.

3.3.3 Filtri FIR

L'altra categoria di filtri digitali comunemente utilizzati è quella dei filtri FIR (*finite impulse response*, ovvero *risposta impulsiva finita*). I filtri FIR sono descritti dalla seguente funzione di trasferimento:
$$\sum_{n=0}^{M} b_n z^{-n} \tag{3.7}$$
dove M è l'ordine del filtro e $\{b_0, b_1, ..., b_{M-1}\}$ sono i coefficienti del filtro (detti anche *taps* in gergo tecnico). L'equazione alle differenze corrispondente è:
$$y[n] = b_0 x[n] + b_1 x[n-1] + ... + b_{M-1} x[n-M+1] \tag{3.8}$$

In questa piattaforma, i blocchi di filtraggio utilizzati come FIR possono essere interamente configurati scegliendo l'ordine M e il valore di ciascun coefficiente.

Rispetto ai filtri IIR, i filtri FIR richiedono molta più memoria (a parità di caratteristiche del filtro ottenuto) per il salvataggio dei coefficienti, che possono raggiungere un

numero molto elevato, e un tempo di calcolo molto maggiore (in un filtro di ordine M richiede M moltiplicazioni e addizioni, a differenza delle 5 tipiche delle biquad).

Per queste ragioni, nel caso particolare di questo sistema DSP, si è deciso d'implementare i filtri FIR soltanto nei blocchi d'ingresso, che sono solo due, per limitare l'impatto sulla percentuale del tempo di elaborazione, e di limitare il numero complessivo di coefficienti utilizzabili a 1024: la somma del numero di *taps* di ciascun filtro non deve superare questo valore. La scelta è stata dettata dalla necessità di salvare la configurazione (per mantenerla tra i riavvii del sistema) su una memoria esterna EEPROM, comunemente nell'ordine delle centinaia di KB[1].

Allo stato attuale, a differenza delle *biquad*, non è stato implementato alcun meccanismo per semplificare la progettazione dei filtri FIR in modo intuitivo per l'operatore (in termini di tipologia di filtro e parametri semplici). Per questo motivo, anche durante la validazione della piattaforma, sono stati utilizzati strumenti di terze parti per il calcolo dei coefficienti date delle semplici specifiche in frequenza[2].

Il motivo di questa scelta è che generalmente i filtri FIR non vengono utilizzati manualmente per equalizzare la risposta in frequenza dei diffusori collegati al sistema, per questo scopo sono più che sufficienti le *biquad*. Il caso d'uso specifico per i filtri FIR è principalmente quello di equalizzare la risposta in frequenza dell'ambiente in cui si trova il diffusore, misurando la risposta impulsiva dell'ambiente e calcolando attraverso software specifici[3] un filtro in grado di compensarla.

[1] I soli 1024 coefficienti occupano già $B = M * 4\text{B} = 4\text{KB}$

[2] Ad es. `http://t-filter.engineerjs.com/`

[3] Ad es. *Room Eq Wizard* `https://www.roomeqwizard.com`

Capitolo 4

Misure audio

In un sistema di elaborazione digitale come quello presentato finora è fondamentale validare tutte le sue componenti: gli stadi di conversione analogico-digitale e digitale-analogico, lo stadio di elaborazione puramente digitale, ma anche il sistema nella sua interezza.

Il corretto funzionamento della sezione di elaborazione digitale è in buona parte garantito dall'uso di una libreria (quale la ARM CMSIS DSP) fornita in dotazione e già validata dal produttore del microcontrollore, che consente di evitare possibili errori di pura implementazione degli algoritmi.

Anche gli stadi di conversione tra i due domini del segnale sono facilmente testabili, introducendo alcuni strumenti di misura all'interno del firmware del sistema: ad esempio, per testare gli stadi d'ingresso (ADC) è stato implementato il calcolo di una serie di grandezze caratteristiche dei segnali, quali il valore di picco, il valore in unità di volume (*VU meter*) o il valore RMS, che è stato approfondito nel capitolo 3.3.1. Per testare lo stadio d'uscita invece il sistema è stato provvisto di generatori di segnale (sinusoidali e di rumore bianco e rosa) liberamente collegabili a ciascuna delle uscite digitali (DAC). Tramite questi sistemi è possibile non solo validare il sistema, ma anche valutarne le caratteristiche specifiche in termini d'integrità del segnale attraverso le misure tipiche (SNR, THD+N, etc.).

Altrettanta attenzione però merita l'analisi del sistema nella sua interezza, in particolar modo la possibilità di controllare che dall'ingresso negli stadi analogici, alla conversione digitale, all'uscita, nuovamente analogica, il segnale sia correttamente elaborato preservando la sua integrità nei limiti garantiti dalle specifiche.

4.1 Strumentazione: PrismSound dScope III

Figura 4.1: Pannello frontale del dScope III

In particolar modo per la verifica del funzionamento del sistema nel suo complesso, si è fatto uso di uno strumento di misura molto avanzato dedicato alle applicazioni audio: il *PrismSound dScope Series III*[1].

Come si può vedere in figura 4.1, lo strumento non è dotato di un'interfaccia grafica: si tratta infatti principalmente di un'interfaccia hardware, che si occupa solamente dell'acquisizione e della generazione dei segnali di test, ed collegata tramite USB a un software PC che è il vero cuore dello strumento di misura. Questa architettura è particolarmente vantaggiosa perché permette di sfruttare l'intera potenza di calcolo di un PC, ma nello stesso tempo un'interfaccia hardware a elevata precisione e per la gestione dei segnali.

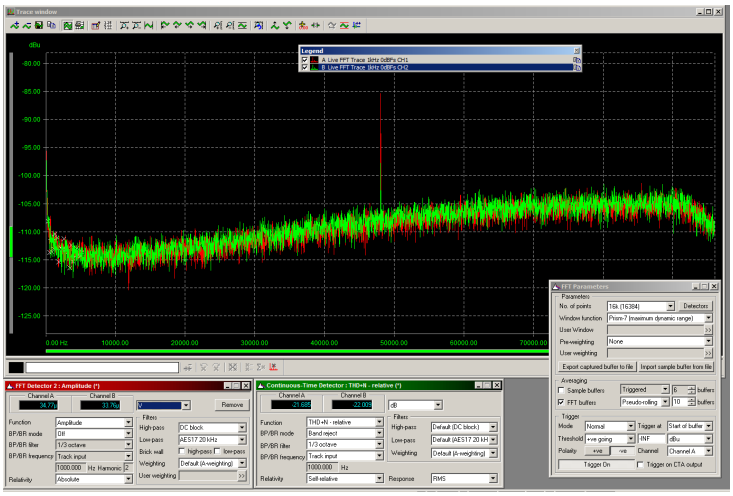


Figura 4.2: Interfaccia PC del dScope III

L'interfaccia grafica su PC (figura, 4.2) è notevolmente avanzata, e permette di eseguire una grande quantità di operazioni, di cui sono elencate in breve le più utilizzate per questo progetto:

[1] `http://www.prismsound.com/test_measure/products_subs/dscope/dscope_home.php`

- **Generazione di segnali**: è possibile generare varie tipologie di segnali completamente parametrizzabili nella forma d'onda (sinusoidale, quadra, triangolare, personalizzata) e nelle relative grandezze (ampiezza, valore RMS, energia, potenza, frequenza).

- **Misure di grandezze caratteristiche dei segnali**: quali ampiezza, fase, SNR (rapporto segnale rumore), frequenza, distorsione (THD+N).

- **Misure sullo spettro del segnale**: FFT (*fast Fourier transform*), risposta in frequenza tramite *sweep* AC.

4.1.1 Connessione al sistema

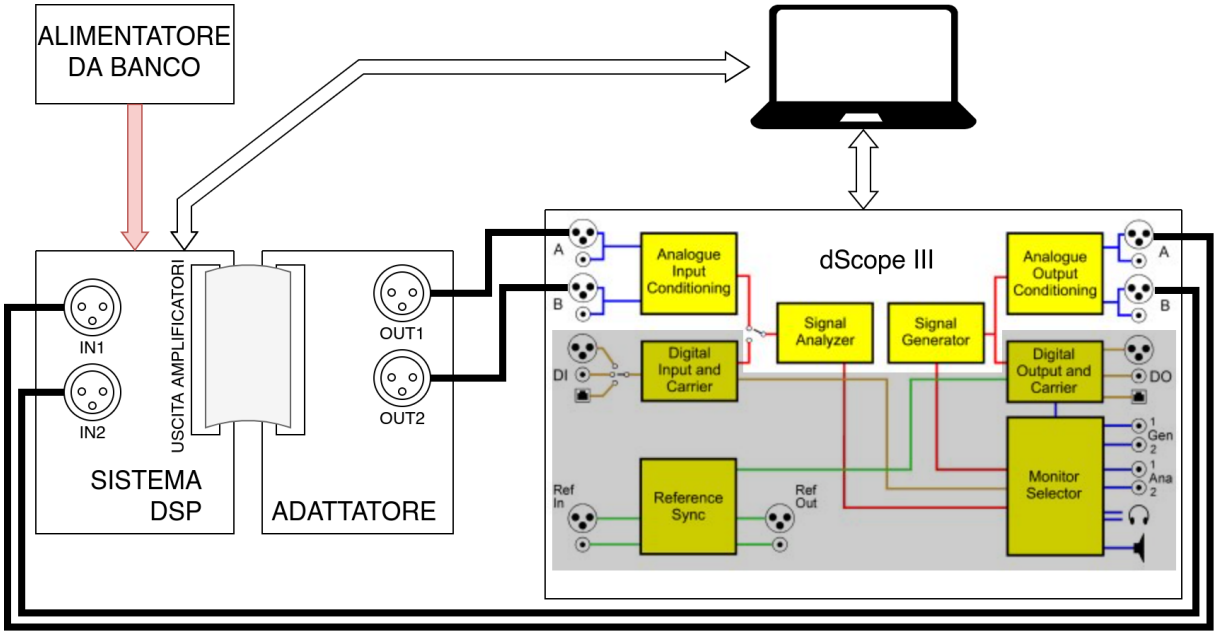


Figura 4.3: Configurazione banco di prova del sistema

Il *dScope III* è dotato di una serie di uscite (analogici e digitali) in grado di generare segnali di qualsiasi forma, da tipiche sinusoidi, completamente regolabili, a (se necessario) segnali personalizzati ottenuti da un file *WAV* nel PC. Questi generatori vengono generalmente collegati agli ingressi del sistema, permettendo d'iniettare nel dispositivo un segnale di riferimento con parametri noti, requisito fondamentale per fare una misura accurata della catena di elaborazione.

Il segnale in uscita dal dispositivo, a sua volta, viene collegato agli ingressi analogici dello strumento (che ne possiede anche di digitali, con protocolli specifici del settore audio, ma non utilizzati in questo progetto). Questi ne consentono l'acquisizione ad alta qualità

(elevatissimo rapporto segnale rumore e ridotta distorsione introdotti dall'apparato di misura).

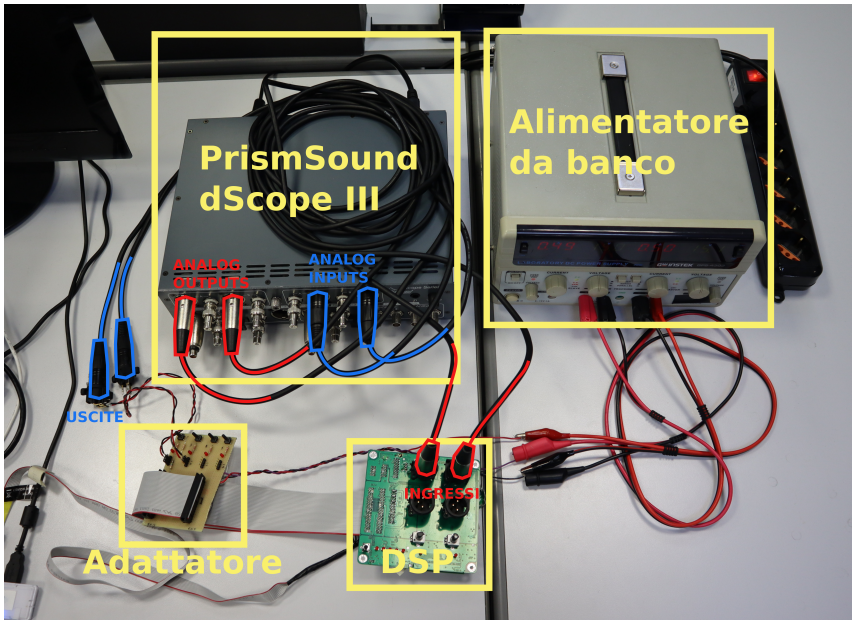


Figura 4.4: Foto del banco di prova

L'uscita del dispositivo, come spiegato nel capitolo 1.2, è accessibile da una serie di connettori proprietari per la connessione ad alcuni modelli di amplificatori esterni. Per questo motivo, per validare il sistema, è stato utilizzato un adattatore che da un lato presenta il connettore proprietario complementare a quello presente in scheda, e dall'altro espone dei connettori XLR per il collegamento allo strumento di misura.

Il segnale acquisito dagli ingressi dello strumento è una versione modificata del segnale inizialmente generato dallo stesso strumento (in base alla configurazione del sistema sotto test). Confrontandolo quindi opportunamente con il segnale originale, è possibile caratterizzare il tipo e i parametri dell'elaborazione eseguita dal dispositivo in esame e verificare che corrisponda alla configurazione effettivamente impostata.

4.2 Caratterizzazione del sistema

Prima di mettere in commercio qualsiasi sistema di elaborazione audio, è necessario caratterizzarne e descriverne le caratteristiche tecniche in un *datasheet*. Esistono numerosi standard (il più noto è l'AES17) e misure tipiche [1] che sono universalmente riconosciute come fondamentali per descrivere il funzionamento di un sistema di elaborazione audio.

Nel caso particolare sono state effettuate le misure più caratteristiche (tutte tramite il *dScope III*):

- Massimo segnale in ingresso
- Massimo segnale in uscita
- Risposta in frequenza
- SNR (*signal to noise ratio*)
- THD+N (*total harmonic distortion plus noise*)

Le misure descritte in seguito sono state effettuate sul sistema digitale nel suo complesso, configurato senza applicare alcuna elaborazione tra ingresso e uscita e considerando (configurato quindi come canale passante o *pass-trough*), per semplicità, solo un canale d'ingresso e un canale d'uscita.

In un sistema digitale è importante verificare che tali misure siano sufficientemente vicine a quelle specificate nel foglio tecnico del codec: infatti in termini di queste metriche, un sistema analogico ottiene risultati molto migliori di un sistema digitale (si veda la sezione 4.2.5), quindi il codec di fatto rappresenta il componente "collo di bottiglia", che determina il limite alle prestazioni. Misurare, quindi, delle metriche simili a quelle del codec significa che l'elettronica di supporto a quest'ultimo (eventuali stadi di amplificazione o filtraggio) non introduce alcuna *aggiuntiva* degradazione del segnale.

È molto importante, infine, specificare le condizioni in cui viene effettuata la misurazione, perché spesso condizioni differenti producono misure differenti. Ad esempio, effettuando le stesse misure elencate in precedenza su tutti i canali contemporaneamente invece che singolarmente, i risultati peggiorano in modo evidente (esistono grandezze specifiche per quantificare anche queste non idealità, ad esempio il *cross-talk*).

Le stesse misure sono state poi effettuate anche sul canale *pass-trough* puramente analogico, presente nella *scheda dsp*, che replica su un connettore di uscita aggiuntivo il segnale analogico d'ingresso, non facendolo passare per il sistema di elaborazione digitale, ma attraverso un operazionale che svolge unicamente la funzione di *buffer*.

4.2.1 Massimo segnale in ingresso e uscita

Tale misura caratterizza la massima ampiezza del segnale che il sistema è in grado di accettare in ingresso e la massima ampiezza del segnale che è in grado di produrre in uscita, senza che questi siano distorti in maniera significativa. Quando il segnale, infatti, supera questi limiti (determinati dai componenti elettronici del sistema) causa generalmente un

fenomeno di saturazione, provocando una notevole distorsione, chiaramente udibile in un sistema audio.

I valori di queste grandezze sono espressi in dBu (0dBu = 0.775V), e sono particolarmente importanti perché permettono di conoscere quello che viene chiamato *fondo scala* del sistema, cioè il segnale massimo di riferimento per il sistema. Conoscendo il *fondo scala* è possibile poi riportare le misure di ampiezza con un'altra unità di misure, il dBFS, in cui lo 0dBFS rappresenta l'ampiezza massima (il *fondo scala*, appunto). Queste misure sono molto importanti anche perché consentono all'utente di verificare se, nella catena di elaborazione audio, ciascun dispositivo di elaborazione sia compatibile con il successivo, ed eventualmente di aggiungere un guadagno intermedio per adattare l'ampiezza del segnale.

Convenzionalmente l'ampiezza massima è l'ampiezza di un segnale sinusoidale a 1kHz in ingresso che misura in uscita una distorsione (THD) pari all'1%. Nel sistema in questione l'ampiezza massima sia in ingresso che in uscita è di 16dBu (4.88V), che nelle misure seguenti è stato utilizzato come *fondo scala*.

4.2.2 Risposta in frequenza

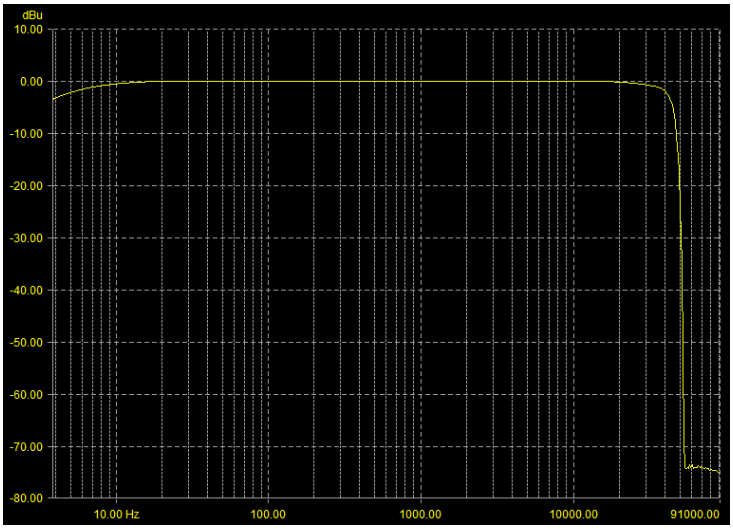


Figura 4.5: Risposta in frequenza del sistema

Tale misura è meno rilevante per l'utente finale, ma è molto importante per dimostrare il corretto funzionamento del sistema: trattandosi infatti di un sistema campionato, è necessario che prima del campionamento sia applicato un filtro passa-basso con una frequenza di taglio pari alla metà della frequenza di campionamento, per rimuovere eventuali fenomeni di *aliasing*.

In secondo luogo, trattandosi di un sistema audio si aggiunge anche un filtro passa-alto che rimuova la componente spettrale del segnale al di sotto dei 20Hz, il limite inferiore della banda audio.

In figura 4.5 è possibile osservare che è presente una banda passante fino a circa 42kHz (a -3dB), compatibilmente alla frequenza di Nyquist $F_N = \frac{F_s}{2} = \frac{96000}{2} = 48kHz$. Poco al di sotto dell'estremità inferiore della banda audio, inoltre, la risposta in frequenza inizia a calare, grazie ai filtri di rimozione della componente continua posti agli ingressi del codec.

In secondo luogo è importante osservare che nella banda passante la risposta in frequenza sia il più vicina possibile agli 0dB, a indicare che gli stadi analogici e di conversione analogico-digitale non modificano il segnale in modo rilevante. Osservando il grafico si nota che ciò è particolarmente vero in banda audio, mentre oltre i 20kHz la risposta tende a calare leggermente, a causa della presenza del filtro passa-basso.

4.2.3 Fondo di rumore e SNR

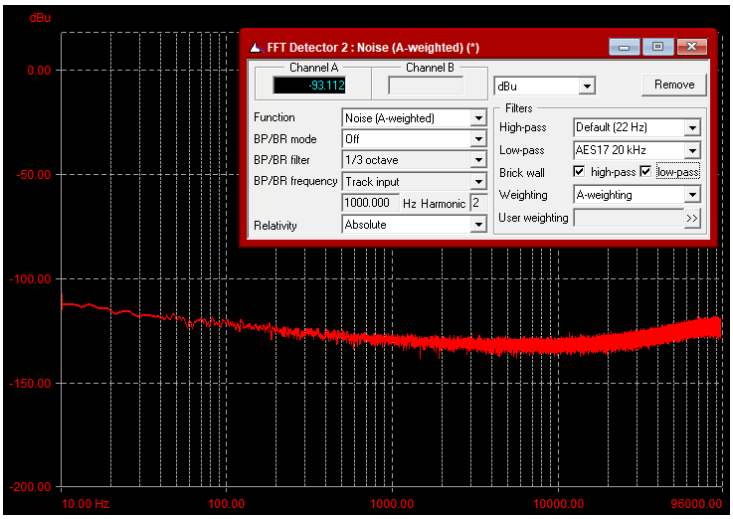


Figura 4.6: Fondo di rumore del sistema (nel grafico è visualizzato il dominio della frequenza, nella finestra è riportata la misura automatica)

Nell'ambito audio, questa è una delle misure più importanti per caratterizzare la qualità dell'elaborazione del segnale da parte del sistema. Questa misura consente, infatti di determinare il *range* (cioè l'escursione) massimo dell'intensità del segnale che può essere elaborato dal sistema. Se il limite superiore, infatti, è dato dal massimo segnale che non viene distorto, il limite inferiore, invece è dato dal più piccolo segnale che non viene confuso con il rumore intrinseco del sistema, dovuto principalmente al codec.

In questa misura infatti viene calcolata l'ampiezza del segnale in uscita dal sistema quando l'ingresso è cortocircuitato (e quindi nullo). Di conseguenza, il segnale misurato in uscita è per forza il rumore intrinseco del sistema, e qualsiasi segnale di ampiezza inferiore sarà indistinguibile dal rumore. L'intensità del rumore a ingresso cortocircuitato rappresenta quindi il limite inferiore del *range dinamico* del sistema. Come si può osservare in figura 4.6, il fondo di rumore è a -93dBu.

È importante notare che nel calcolo di questo valore sono state applicati un filtro passa-basso a 20kHz (definito secondo lo standard AES17) e la *pesatura A*, che è una specifica curva in frequenza che viene sommata allo spettro del segnale. Tali calibrazioni hanno lo scopo di dare un peso maggiore alla misura al centro della banda audio, dove l'orecchio umano è più sensibile al rumore.

Conoscendo il fondo di rumore e il fondo scala (cioè l'ampiezza massima del segnale), è possibile infine calcolare la metrica più importante: il SNR (*signal to noise ratio*), cioè il rapporto tra il massimo segnale processabile dal sistema e il fondo di rumore, che in decibel si esprime come:

$$\mathrm{SNR}_{dB} = S_{max} - N_{floor} = 16\mathrm{dBu} - (-93\mathrm{dBu}) = 109\mathrm{dB} \tag{4.1}$$

Tale valore può essere confrontato con quello definito nel *datasheet* del codec, che riporta un valore tipico a 25°C di 110dB, concludendo che il rumore introdotto dal circuito esterno al codec è sostanzialmente trascurabile rispetto a quello intrinseco del codec e non causa alcuna degradazione del segnale.

4.2.4 Total harmonic distortion (THD)

La distorsione è un'altra misura fondamentale nell'ambito dell'elaborazione dei segnali, poiché caratterizza quanto un sistema ne mantiene l'integrità, da un punto di vista delle sue componenti armoniche. In un sistema con bassa THD il segnale d'ingresso è riportato in uscita praticamente invariato, se invece la THD è elevata il segnale in uscita è sommato ad altri segnali indesiderati armonicamente correlati al segnale originale.

La THD si misura eccitando il sistema con un segnale sinusoidale (di solito a 1kHz), con un'ampiezza specificata (di solito -1dBFS o -20dBFS) e calcolando, tramite un'analisi spettrale, l'ampiezza delle armoniche del segnale originale, che è direttamente proporzionale al valore della THD; la formula per il calcolo della THD è infatti:

$$\mathrm{THD} = \sqrt{\frac{\sum_{i=2}^{\infty} V_{i,\mathrm{RMS}}^2}{\sum_{i=1}^{\infty} V_{i,\mathrm{RMS}}^2}} \tag{4.2}$$

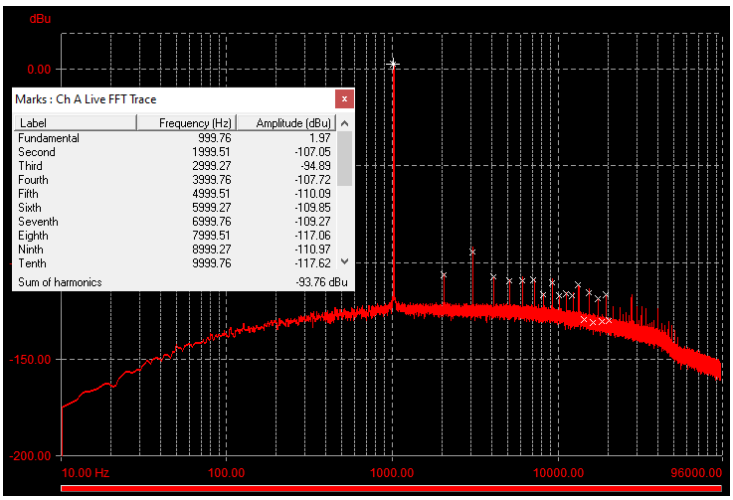


Figura 4.7: Spettro del segnale in uscita e misurazione della THD+N. Nella finestra sono riportate le ampiezze di ogni armonica

dove $V_{i,\mathrm{RMS}}$ è l'ampiezza dell'i-esima armonica del segnale misurato.

Anche in questa misura si applicano le pesature già descritte nella sezione 4.2.3, per renderla più significativa nell'ambito audio.

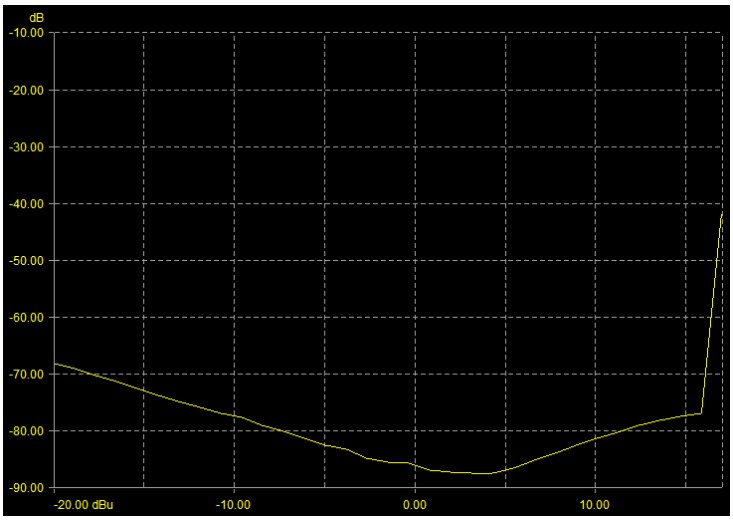


Figura 4.8: Variazione della THD+N al variare dell'ampiezza del segnale in ingresso

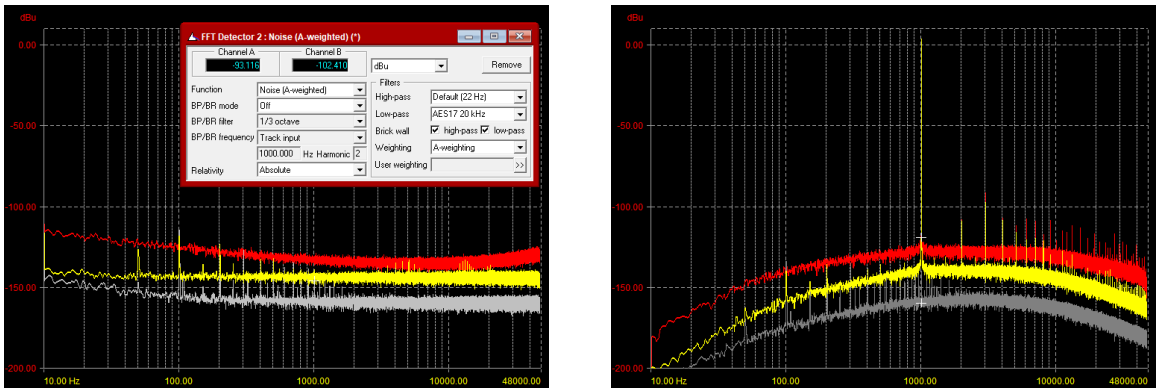
Nel caso dei sistemi campionati, a causa della notevole rilevanza del rumore generato dal codec, si preferisce la misura THD+N, che tiene conto anche del fondo di rumore. È interessante notare che al calare dell'ampiezza della fondamentale (data dal segnale di riferimento), la THD tende a peggiorare.

Infatti, come si può osservare in figura 4.8, al variare dell'ampiezza del segnale di riferimento dal *fondo scala* in giù, si misura inizialmente una THD elevata intorno al *fondo scala*, a causa della saturazione del segnale; al calare dell'ampiezza la THD raggiunge un minimo, quando il segnale non satura più, e poi tende a peggiorare nuovamente a causa

della diminuzione dell'ampiezza della fondamentale. Si osserva che il minimo di distorsione è intorno ai 2dBu, dove la THD + N = −88dB. In figura 4.7 è riportato il segnale in uscita nel dominio della frequenza nel caso di THD+N minima: sono chiaramente visibili le componenti armoniche indesiderate a frequenza multipla del segnale di riferimento.

Anche il valore della THD può essere confrontato con il valore, riportato nel *datasheet* del codec, di -94dB quando un solo canale è attivo. È evidente che in questo caso il circuito introduce una distorsione non trascurabile; è importante notare, però, che la scheda su cui sono state effettuate tutte le misure è un prototipo, il cui layout è stato ampiamente modificato a mano durante lo sviluppo, possibilmente introducendo delle non linearità.

4.2.5 Confronto tra i canali digitale e analogico



(a) Fondo di rumore

(b) THD+N

Figura 4.9: Confronto dei canali digitale (traccia rossa) e analogico (traccia gialla). La traccia grigia rappresenta il fondo di rumore dello strumento di misura

Le stesse misure descritte nelle sezioni precedenti sono state replicate anche per il canale analogico *pass-trough* presente nel dispositivo.

Si è osservato che tutte le metriche descritte in precedenza hanno dato risultati notevolmente migliori nel canale analogico rispetto al canale digitale (coerenti per di più con le specifiche dell'operazionale utilizzato come *buffer* nel canale analogico):

Misura	**Analogico**	**Digitale**
SNR (fig. 4.9.a)	119dB	103dB
THD (3dBu@1kHz) (fig. 4.9.b)	-98dB	-88dB

È evidente, quindi, che in termini di misure caratteristiche, un sistema analogico è ancora molto più performante di un sistema digitale, che a causa delle operazioni di

campionamento e quantizzazione introduce più rumore e non idealità nella catena di elaborazione del segnale.

Tuttavia, per quanto peggiori, le misure ottenute col canale digitale possono essere considerate più che soddisfacenti per un'applicazione nell'ambito dell'audio, dal momento che, ai livelli misurati, il rumore e le non idealità sono praticamente impercettibili all'orecchio umano. Al contrario, accettare questo piccolo compromesso permette di ottenere infinite possibilità di elaborare il segnale, una volta che questo è stato trasferito nel dominio digitale, dove l'elaborazione è molto più semplice da implementare, più deterministica e più flessibile.

4.3 Esempio: progettazione e misura di un FIR

In questa sezione si descrive un esempio di funzionamento del sistema DSP, configurato per operare come semplice filtro FIR, dalla progettazione del filtro tramite MATLAB, alla configurazione della piattaforma di elaborazione tramite il configuratore *web-based* alla misurazione del sistema tramite il *dScope III*, per verificare che il comportamento complessivo del sistema rispetti la configurazione scelta.

4.3.1 Progettazione in MATLAB

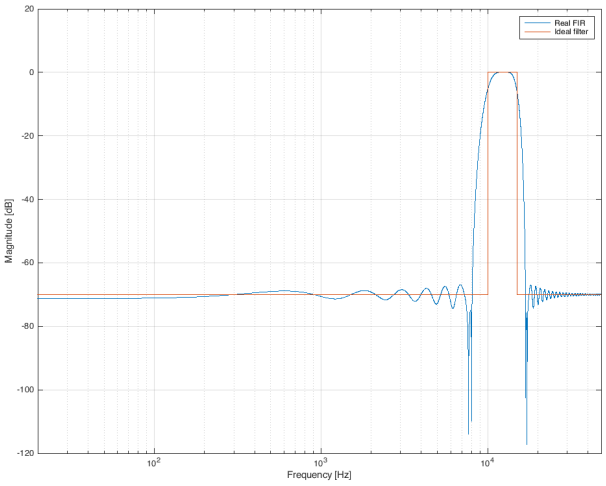


Figura 4.10: Filtro FIR sintetizzato in MATLAB

Per progettare un filtro FIR in MATLAB si può usare la funzione `fir2`, che permette di specificare il numero di coefficienti desiderati, seguito da due vettori (in frequenza e

guadagno) che identificano i punti della risposta in frequenza ideale che si vuole ottenere dal filtro[2].

Il filtro desiderato (si tratta di un esempio, non ha una reale applicazione pratica, ma dimostra bene la flessibilità del sistema) è in questo caso un filtro passa-banda, con banda passante nell'intervallo tra 10 e 15 kHz. Sempre arbitrariamente si è scelto di usare 151 coefficienti.

I due vettori che descrivono la risposta in frequenza desiderata sono:

```
f = [0   1   10000 10000 15000 15000 48000]; % Hz
m = [-70 -70 -70   0     0     -70   -70  ]; % dB
```

La risposta in frequenza richiesta è stata rappresentata anche in figura 4.10 con la traccia arancione.

Nel codice precedente la frequenza viene specificata nell'intervallo tra 0 e 48kHz, dal momento che il sistema digitale lavora a una frequenza di campionamento di 96kHz, quindi per il teorema di Nyquist, la massima frequenza di un segnale elaborabile è $\frac{F_s}{2} = 48\text{kHz}$.

Per calcolare i coefficienti del filtro FIR desiderato è sufficiente eseguire il seguente codice, che invoca la funzione `fir2`:

```
b = fir2(150,f/48000,db2mag(m));
```

dove 150 è l'ordine del filtro (che è il numero di coefficienti meno uno), `b` è il vettore contenente i coefficienti calcolati del filtro e ciascun vettore di frequenza e guadagno è stato rispettivamente scalato nell'intervallo $[0, 1]$ e riportato da scala logaritmica a scala lineare, come richiesto dalla funzione `fir2`.

Il grafico, infine, è stato generato col seguente codice:

```
[h,w] = freqz(b,1,10000,Fs);
semilogx(w,mag2db(abs(h)),f,m);
grid on;
xlim([20 48000]);
xlabel("Frequency [Hz]");
ylabel("Magnitude [dB]");
legend("Real FIR", "Ideal filter");
```

[2] https://www.mathworks.com/help/signal/ref/fir2.html

Dal grafico in figura 4.10 (dove viene ignorata la fase) si può osservare che il filtro generato da MATLAB (traccia blu) è ragionevolmente vicino all'obiettivo richiesto (traccia arancione). Un risultato ancora più preciso si potrebbe raggiungere alzando l'ordine del filtro.

4.3.2 Configurazione e anteprima tramite web-configurator

Attraverso il web-configurator è possibile creare visivamente e in modo intuitivo una configurazione per testare il filtro appena creato: tutti gli effetti digitali sono raggruppati in 2 blocchi d'ingresso, 4 blocchi di filtraggio generiche e 4 blocchi d'uscita (analogamente a quanto descritto nel capitolo 3.1), collegabili tra loro tramite *drag-n-drop* e i parametri di ciascun effetto possono essere modificati cliccando sul simbolo corrispondente.

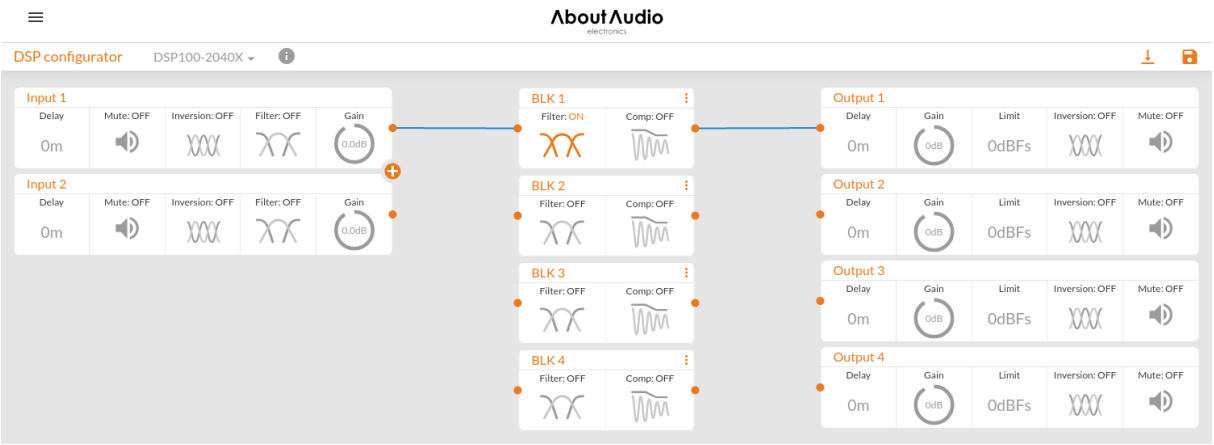


Figura 4.11: Configurazione di esempio tramite web-configurator

La configurazione scelta è molto semplice: tutti gli ingressi e tutte le uscite sono state tenute alle impostazioni di *default*, ed è stato fatto un singolo collegamento dal blocco d'ingresso IN1 al blocco di filtraggio BLK1 all'output OUT1, come visibile in figura 4.11.

Cliccando il simbolo *filter* nel blocco di filtraggio BLK1, è possibile modificare e visualizzare l'anteprima del filtro che si vuole assegnare a tale sezione di elaborazione. Si è scelta la modalità di operazione FIR e sono stati inseriti i coefficienti calcolati al punto precedente tramite MATLAB. Il web-configurator, per una maggiore facilità d'uso, riporta anche l'anteprima della risposta in frequenza attesa (visibile in figura 4.12).

Terminata la configurazione è possibile scaricare un file in formato testuale che contiene una lista di direttive per impostare tutti i parametri del sistema, al quale può essere inviato tramite USB (una volta connesso al PC, infatti, il sistema si presenta come un dispositivo USB *mass-storage*). Di seguito viene riportato un breve estratto delle sezioni più rilevanti:

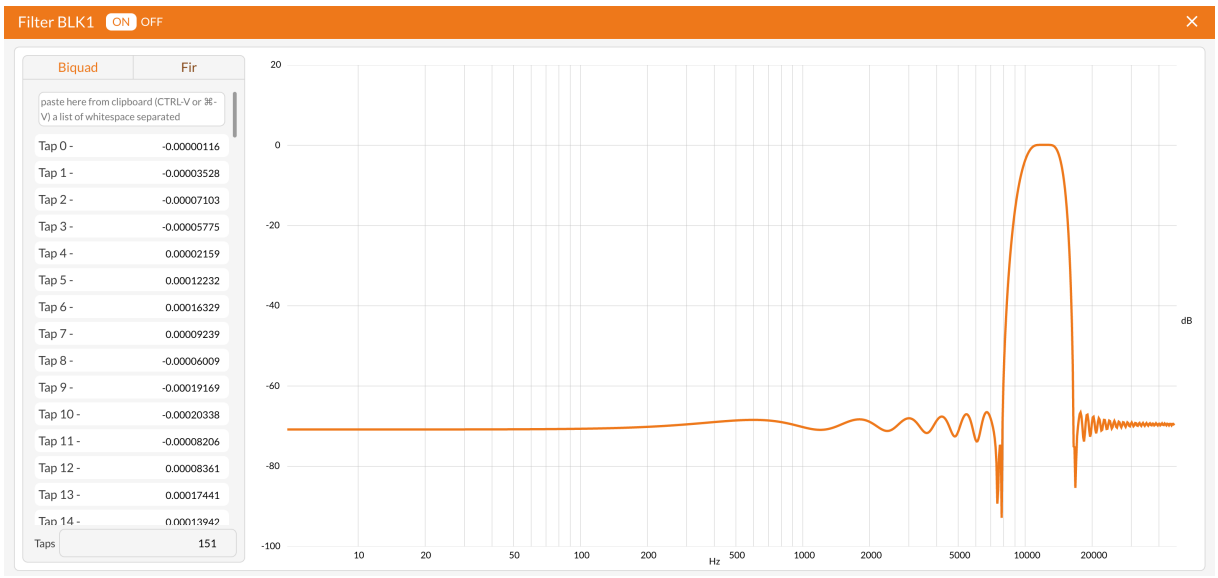


Figura 4.12: Anteprima del filtro FIR nel web-configurator

```
SELECT BLK1 FIR      # Imposta il filtro BLK1 come FIR
FIR BLK1 151         # Imposta il numero di coeff.
# Imposta i coefficienti del filtro (molte righe simili)
CFIR BLK1 0 -0.00000116 -0.00003528 -0.00007103 -0.00005775 0.00002159
# Imposta i collegamenti tra i vari blocchi
ROUTE IN1 BLK1
ROUTE BLK1 OUT1
```

Gli stessi comandi potrebbero essere riportati manualmente anche tramite una *console* seriale, sempre accessibile tramite USB, in caso l'operatore volesse sperimentare in tempo reale sulla configurazione.

4.3.3 Misura e validazione

Non appena il file contenente la nuova configurazione viene caricato sul dispositivo, questa viene applicata immediatamente: tra l'ingresso IN1 e l'uscita OUT1, dunque, il sistema si comporta come un filtro passa-banda con le caratteristiche specificate in precedenza.

Attraverso un setup del banco di prova come quello descritto al capitolo 4.1.1, è possibile utilizzare il *dScope III* per verificare che il funzionamento del sistema sia corrispondente alla configurazione, controllando che la risposta in frequenza tra IN1 e OUT1 sia equivalente a quella calcolata tramite MATLAB e visibile nell'anteprima del web-configurator.

Tra le varie misurazioni messe a disposizione dal *dScope III*, la più adatta per misurare la risposta in frequenza è lo *sweep* AC. In questa modalità il generatore di segnale dello strumento viene impostato per generare un segnale sinusoidale con ampiezza e frequenza note. Per il teorema della risposta in frequenza, se un sistema LTI con funzione di trasferimento H, viene stimolato con un segnale sinusoidale $x(t) = A\sin(\omega t)$ di pulsazione ω produce in uscita un segnale

$$y(t) = A|H(j\omega)|\sin(\omega t + \angle H(j\omega)) \tag{4.3}$$

che è il segnale $x(t)$ scalato del valore assoluto e sfasato dell'argomento della risposta in frequenza alla pulsazione ω. È sufficiente quindi confrontare i due segnali d'ingresso (generato dallo strumento) e d'uscita del sistema, per estrarre modulo e fase della sua risposta in frequenza.

Lo strumento di misura automatizza questa operazione, variando progressivamente la frequenza del segnale generato in un intervallo specificato e misurando per ogni iterazione modulo e fase della risposta del sistema.

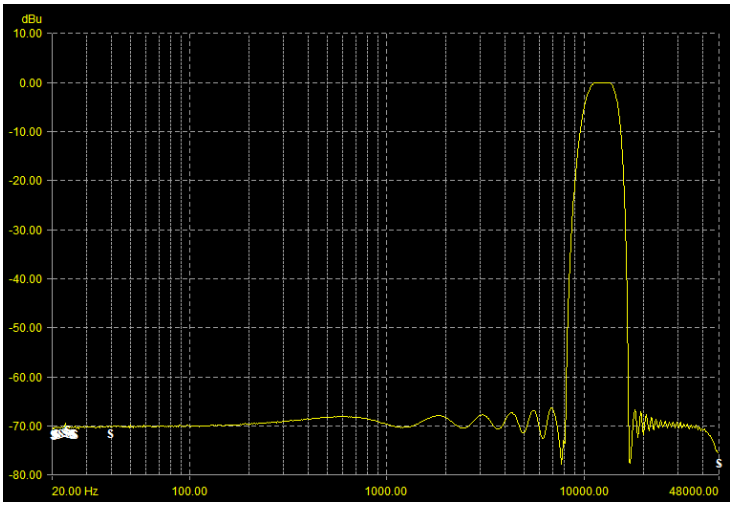


Figura 4.13: Risposta in frequenza del sistema misurata tramite *dScope III*

Il risultato di questa misurazione può essere infine rappresentato in un grafico (in figura 4.13), fornendo una risposta in frequenza che corrisponde con ottima approssimazione alla caratteristica del filtro calcolata su MATLAB.

Capitolo 5

Web-configurator

Per quanto accessoria e non fondamentale per il funzionamento della piattaforma DSP, la *web-app* di configurazione svolge comunque un ruolo fondamentale all'interno di questo progetto, poiché rappresenta un'interfaccia intuitiva e visuale per l'operatore non necessariamente tecnico e in grado di programmare questo tipo di prodotto da una linea di comando testuale (che è l'interfaccia alternativa e di basso livello con cui si può configurare questa piattaforma).

L'interfaccia grafica, infatti, rappresentando il sistema DSP suddiviso nei tre blocchi, d'ingresso, di filtraggio e d'uscita, e permettendo di collegare tramite *drag-n-drop* ciascuno di questi blocchi per formare una catena di elaborazione del segnale, rievoca l'operazione, ben nota ai professionisti di quest'ambito, di collegare tra di loro degli *effetti* e blocchi di elaborazione fisici attraverso dei cavi.

In secondo luogo, l'interfaccia grafica semplifica notevolmente anche la progettazione dei filtri, in particolar modo delle *biquad*, che vengono generalmente utilizzate per l'equalizzazione manuale.

Ogni blocco di filtraggio configurato come *biquad* è composto dalla cascata di fino a 12 singole *biquad* individualmente configurabili in termini di tipologia, frequenza significativa ed eventuali altri parametri. Nell'interfaccia (in figura 5.1) è possibile vedere sia il grafico della risposta in frequenza complessiva derivante dalla cascata di ciascuna delle 12 *biquad*, sia modificare direttamente sul grafico i parametri di ciascuna di esse, trascinando dei cursori per avere un'anteprima in tempo reale come di cambia la risposta in frequenza.

Una volta terminata la configurazione, è possibile esportare un file di configurazione testuale contenente tutte le direttive, interpretabili dalla piattaforma DSP, necessarie per applicare le impostazioni scelte nella *web-app*, che è anche in grado di aprire un file di

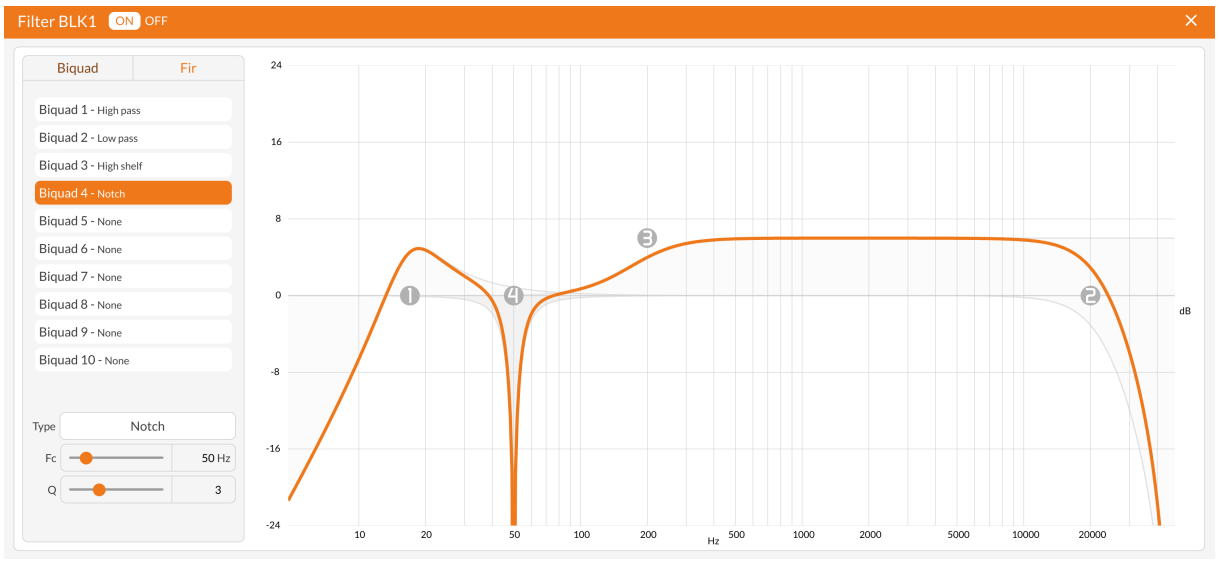


Figura 5.1: Pannello di configurazione dei filtri *biquad*, a sinistra è visibile la configurazione delle 12 *biquad* in cascata, a destra la risposta in frequenza finale del filtro. Gli indicatori grigi numerati possono essere spostati in tempo reale per variare interattivamente la configurazione del filtro

configurazione preesistente, permettendo di modificarlo in un secondo momento.

Parallelamente allo sviluppo del firmware DSP, è stato portato avanti anche lo sviluppo di una nuova interfaccia grafica sperimentale, con l'obiettivo di avere un'applicazione più facilmente estensibile della *web-app* originale, che per motivi architetturali non lo è altrettanto.

5.1 Architettura dell'applicazione

L'applicazione di configurazione è stata sviluppata in JavaScript (per la precisione TypeScript, che è un linguaggio che compila in JavaScript), utilizzando un *framework* molto utilizzato per creare le interfacce utente *web-based*: si tratta di *React.js*, sviluppato e mantenuto da *Meta*.

React consente di creare interfacce grafiche *reattive*: utilizzando il JSX, un linguaggio specifico che è un misto di JavaScript e HTML (che è tradizionalmente utilizzato per creare le pagine web), è possibile descrivere il layout e il contenuto di ciascuna pagina di un'interfaccia grafica come pura funzione di uno *stato*, che generalmente è tenuto separato dal codice relativo all'interfaccia.

La peculiarità di React è la possibilità di aggiornare automaticamente i vari elementi grafici dell'applicazione non appena lo stato dell'applicazione dovesse cambiare. In questo

modo il programmatore si può concentrare semplicemente nel descrivere l'interfaccia in funzione dello stato dell'applicazione, senza preoccuparsi della logica di aggiornamento. Eventuali comandi impartiti dall'utente possono generare dei cambiamenti allo stato, che a loro volta provocano l'aggiornamento automatico dell'interfaccia.

Poiché React è una libreria puramente orientata alle interfacce grafiche, non possiede un sistema di gestione dello stato particolarmente avanzato; per svolgere questo compito è stata infatti scelta un'altra libreria (sempre sviluppata da *Meta*), chiamata *Recoil.js*. *Recoil* permette di definire, separatamente dall'interfaccia grafica, delle unità di stato, dette *atoms*. Ciascuno dei vari elementi grafici costruiti con *React* può essere collegato a uno o più *atoms*, in modo tale che al variare del suo contenuto anche gli elementi corrispondenti siano ridisegnati nella pagina web. Negli *atoms* viene salvato lo stato generico dell'applicazione, ma soprattutto tutte le configurazioni dei vari blocchi di elaborazione e lo stato delle interconnessioni tra i vari blocchi.

Lo stato contenuto negli *atoms* viene letto e scritto anche da due blocchi logici separati dall'interfaccia grafica: un sistema di esportazione, che a partire dallo stato della configurazione è in grado di generare un file di configurazione, e un sistema di parsing, che attraverso delle *espressioni regolari* è in grado di estrarre da un file preesistente le informazioni sulla configurazione del sistema e di popolare adeguatamente lo stato dell'applicazione.

5.2 Risposta in frequenza in JavaScript

Un problema che è emerso durante lo sviluppo del web-configurator è stato il calcolo della risposta in frequenza di un generico filtro (IIR o FIR), necessario per poterla raffigurare in un grafico.

Dato un sistema o un filtro con una funzione di trasferimento $H(z)$, è possibile calcolare la risposta in frequenza tramite la sostituzione $z = e^{j\omega}$. La risposta in frequenza $H(e^{j\omega})$ è dunque una funzione complessa, il che la rende impossibile da calcolare direttamente in JavaScript, che è un linguaggio che non supporta i numeri complessi.

Per risolvere questo problema si è considerata una generica funzione di trasferimento, applicabile sia a filtri FIR che IIR:

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}} \tag{5.1}$$

la cui risposta in frequenza $H(e^{j\omega})$, tramite la formula di Eulero, si può scrivere come:

$$H(e^{j\omega}) = \frac{\sum_{k=0}^{M} b_i cos(-k\omega) + j \sum_{k=0}^{M} b_i sin(-k\omega)}{\sum_{k=0}^{N} a_i cos(-k\omega) + j \sum_{k=0}^{N} a_i sin(-k\omega)} \tag{5.2}$$

Da questa forma è possibile ricavare facilmente le formule per calcolare modulo e fase, in modo tale che non vi sia l'unità complessa j:

$$|H(e^{j\omega})| = \sqrt{\frac{\left(\sum_{k=0}^{M} b_i cos(-k\omega)\right)^2 + \left(\sum_{k=0}^{M} b_i sin(-k\omega)\right)^2}{\left(\sum_{k=0}^{N} a_i cos(-k\omega)\right)^2 + \left(\sum_{k=0}^{N} a_i sin(-k\omega)\right)^2}} \tag{5.3}$$

$$\angle H(e^{j\omega}) = \arctan\left(\frac{\sum_{k=0}^{M} b_i sin(-k\omega)}{\sum_{k=0}^{M} b_i cos(-k\omega)}\right) - \arctan\left(\frac{\sum_{k=0}^{N} a_i sin(-k\omega)}{\sum_{k=0}^{N} a_i cos(-k\omega)}\right) \tag{5.4}$$

Tramite queste formule è possibile calcolare, separatamente, modulo e fase di qualunque filtro digitale dati i coefficienti $\{b_0 + b_1 + ... + b_M\}$ e $\{a_0 + a_1 + ... + a_N\}$ a una qualsiasi frequenza ω.

Di seguito è riportata la funzione in JavaScript che esegue questo calcolo:

```
tf(b, a, w) {
  let sumsinb = 0, sumcosb = 0, sumsina = 0, sumcosa = 0, magsq = 0;
  for (let k = 0; k < b.length; k++) {
    sumsinb += b[i] * Math.sin(-w * i);
    sumcosb += b[i] * Math.cos(-w * i);
  }
  magsq = sumcosb * sumcosb + sumsinb * sumsinb;
  for (let k = 0; k < a.length; k++) {
    sumsina += a[i] * Math.sin(-w * i);
    sumcosa += a[i] * Math.cos(-w * i);
  }
  magsq /= sumcosa * sumcosa + sumsina * sumsina;
  return {
    mag: 10 * Math.log10(magsq), // La radice è eseguita nel logaritmo
    phase: Math.atan2(sumsinb, sumcosb) - Math.atan2(sumsina, sumcosa),
  };
}
```

È possibile osservare che il calcolo viene effettuato utilizzando solamente le funzioni della libreria `Math`, nativa del linguaggio, senza la necessità di utilizzare librerie esterne

di calcolo con i numeri complessi. È fondamentale, infatti, che questa funzione sia il più semplice e il più veloce possibile poiché essa è parte di un percorso critico del codice responsabile del disegno dei grafici: ogni volta che un parametro di un filtro viene modificato, questa funzione viene chiamata anche centinaia di volte al secondo se la modifica viene fatta in tempo reale dall'operatore, quindi un tempo di calcolo eccessivo limiterebbe notevolmente la responsività e l'usabilità dell'applicazione.

Bibliografia

[1] D. Bohn. *Audio Specifications*. http://www.rane.com/pdf/ranenotes/Audio_Specifications.pdf, 2000. accessed 2022.

[2] R. Bristow-Johnson. *Cookbook formulae for audio equalizer biquad filter coefficients*. https://webaudio.github.io/Audio-EQ-Cookbook/audio-eq-cookbook.html. accessed 2022.

[3] Analog Devices Inc. *AD1937 datasheet (rev. b)*. https://www.analog.com/media/en/technical-documentation/data-sheets/ad1937.pdf, 2010. accessed 2022.

[4] Arm Ltd. *CMSIS DSP Software Library*. https://www.keil.com/pack/doc/CMSIS/DSP/html/index.html. accessed 2022.

[5] R.G. Lyons. *Understanding Digital Signal Processing*. Pearson Education, 3rd edition, 2010.

[6] J.O. Smith. *Introduction to Digital Filters with Audio Applications*. http://ccrma.stanford.edu/~jos/filters/, accessed 2022. online book.

[7] STMicroelectronics. *STM32F745xx STM32F746xx datasheet*. https://www.st.com/resource/en/datasheet/stm32f745ve.pdf, 2016. accessed 2022.

[8] STMicroelectronics. *STM32F75xxx and STM32F74xxx advanced Arm®-based 32-bit MCUs - reference manual*. https://www.st.com/resource/en/reference_manual/rm0385-stm32f75xxx-and-stm32f74xxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf, 2016. accessed 2022.