POLITECNICO DI MILANO

FACOLTÀ DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Specialistica in Ingegneria Elettronica

Progettazione di un processore audio digitale con architettura basata su FPGA

Relatore: Prof. Angelo Geraci

Correlatore: Ing. Andrea Abba

Tesi di Laurea di:

Andrea Albano

Matr. 711379

Anno Accademico 2010-2011

Non cercare di diventare un uomo di successo,
ma piuttosto un uomo di valore.

Albert Einstein

Ringraziamenti

Partiamo dalla parte più difficile: i ringraziamenti.

Per prima cosa, ringrazio il Prof. Geraci per avermi coinvolto in questo lavoro, che si è rivelato estremamente appassionante, e per il supporto offertomi in questa impresa. Non di meno, mi preme ringraziare il Prof. Ripamonti, che mi ha permesso, grazie all'uso dei laboratori e all'accoglienza nel suo gruppo, di sviluppare il mio progetto di ricerca con i migliori mezzi. Infine vorrei anche ringraziare il Prof. Ghioni, che pur non coinvolto nel progetto, ha saputo allietare questo viaggio con la sua simpatia.

In secondo luogo vorrei ringraziare Andrea Abba, Antonio Manenti e Andrea Suardi, per il supporto e soprattutto la pazienza che hanno dimostrato nei miei confronti. Senza di loro probabilmente per me non sarebbe stato possibile fare un singolo passo nella realizzazione di questo progetto.

Naturalmente ringrazio tutti i ragazzi del laboratorio: Leu, Paolo, Davide e Biagio per i momenti di sana ilarità che non devono mai mancare anche quando le cose non sembrano andare bene. In particolare devo ringraziare Francesco, Ale e Yelena: penso di poter dire senza indugio di aver trovato degli Amici in loro, e spero che il nostro rapporto continui a mantenersi e ad alimentarsi in futuro.

Un ringraziamento speciale ai Capi del Poli: Luca, Ricky, Bridge, Andrea e Fabio, perchè i migliori restano i migliori.

Devo sinceramente ringraziare Fabio e Antonio per un'infinità di motivi: grazie a loro ho potuto cimentarmi in questa tesi e confrontarmi col mondo della vera elettronica. Un particolare ringraziamento soprattutto per avermi permesso di essere alla fiera di Francoforte dedicata al suono.

Un ringraziamento dovuto anche ai colleghi della pizzeria coi quali ho passato 7 anni fantastici: in primo luogo Mauro e Claudia, Marco, Fatima, Luli, Katiuscia, Luca, Alessio e Walter.

Ora passiamo ai ringraziamenti per le persone che mi sono state vicine in questi anni e non solo (e qua le cose si fanno complicate). In primo luogo devo ringraziare la La, che in questi anni è stata il mio sostegno nei momenti peggiori, e Ale, che è da anni un fratello aggiunto nella mia vita. Un ringraziamento speciale a Ste, alla Fra, alla Vale, alla Raffy e a Dany, coi quali ho condiviso dei momenti indimenticabili, nella speranza che l'amicizia non si affievolisca con il passare del tempo. Un doveroso ringraziamento va anche a Re, per tutte le chiaccherate (a volte poco lucide) del venerdì sera.

Infine, ma solo per dare il risalto che meritano ringrazio la mia famiglia. Grazie a Marco e Erika, perchè nonostante tutto vi voglio molto bene, e grazie a Giorgio. Infine devo ringraziare

infinitamente i miei genitori, che mi hanno messo al mondo e mi hanno permesso di arrivare dove sono, sempre spronandomi a dare il meglio e senza mai limitarmi nelle mie scelte. Dedico a loro questo mio percorso sperando di averli resi orgogliosi di me.

Sommario

Indice tabelle

Indice figure

Abstract (Italiano)

In questa tesi viene presentato un nuovo approccio per la progettazione di un Audio Digital Signal Processor ad alte performance e a basso costo, atto appunto all'elaborazione del segnale audio, basato su un dispositivo FPGA. Il sistema si propone di gestire 4 canali in ingresso e 8 in uscita, con la possibilità di usare segnali analogici o digitali, codificati in standard AES/EBU.

Dopo aver concentrato l'attenzione alla presentazione dei dispositivi FPGA, descrivendone pregi e difetti e facendo in particolare rifrimento alle caratteristice della SPARTAN 6, segue una descrizione teorica della struttura di filtraggio utilizzata, molto adatta a realizzare filtri peak, grazie ai quali è possibile creare catene di equalizzazione, di tipo grafico o parametrico. Il filtro IIR del secondo ordine analizzato, possiede infatti la qualità di poter implementare filtri a campana, caratterizzati da soli 3 parametri indipendenti, che consentono la gestione del guadagno, della banda e del frequenza centrale. Con queste premesse viene presentato un algoritmo particolare per il calcolo dei coefficienti, che permette di ottenere funzioni di trasferimento di tipo "True-Response approssimato", molto adatte all'implementazione di equalizzatori grafici.

Viene quindi descritta la struttura interna implementata nell'FPGA, andando ad analizzare i vari moduli. Tra questi ci si concentrerà soprattutto sui blocchi di Input/Output e sulla struttura che gestisce la routine di ritardo. Particolarmente complessa risulta la gestione di moduli in cui i segnali sono trasmessi in maniera seriale, affiancati da strutture che si sviluppano parallelamente. Un passo molto interessante in questo capitolo, è l'implementazione del filtro IIR del second'ordine.

Il dispositivo sviluppato possiede due interfacce utente: una locale e una remota tramite collegamento USB con pc. Il tutto è gestito da un microcontrollore: verrà fatta una panoramica su alcune funzioni specifiche svolte da esso.

Abstract (English)

In this thesis we present a new approach to the design of a high performance, low cost Digital Audio Signal Processor , based on an FPGA device. The system is designed to handle up to 4 input and 8 output channels, with the chance of using both analogue or digital signals, encoded in standard AES/EBU.

After focusing on the presentation of FPGA devices, describing strengths and weaknesses and referring in particular on the features of the SPARTAN 6, follows a theoretical description of the filtering structure used, very suitable to achieve peak filters, through which it's possible the creation of graphic or parametric equalization chains. The second-order IIR filter analyzed, is able to implement bell filters, characterized by only three parameters, which allow the independent management of the gain, bandwidth and center frequency. With this background we present a particular algorithm for calculating the coefficients, which allows to realize "approximate True-Response" transfer functions, very suitable for the implementation of graphic equalizers.

Afterwards is describes the internal structure implemented in the FPGA, by analyzing the various logical blocks. Among these we will concentrate on the Input/Output's ones and on the structure that handles the routine of delay. Particularly complex is the management of block in which signals are transmitted serially, side by side with structures that develop in parallel. A very interesting step in this chapter, is the implementation of the second-order IIR filter.

The device developed has two user interfaces: a local one and a remote one to PC, via USB connection. Everything is managed by a microcontroller: will be made an overview of some specific functions performed by this processor.

1. Introduzione

La storia dell'elaborazione audio in formato digitale è piuttosto recente. Si può affermare che l'introduzione del compact disc sia l'inizio di questa storia. Nei primi anni '80, il CD mise per la prima volta in risalto i vantaggi della rappresentazione dell'audio digitale: l'*high fidelity*, il *dynamic range* e la robustezza dei sistemi ne sono un esempio. Soprattutto all'inizio, questi vantaggi, si scontravano con la mole di dati che un'elaborazione digitale deve affrontare: i primi formati digitali, a 44.1 o 48 kHz, codificati in modalità PCM (Pulse Code Modulation) a 16 bit, significavano circa 1,5 Mbit/s di dati per un canale stereo. Oggigiorno queste cifre fanno sorridere, se solo si pensa alla mole di dati che vengono processati nell'elaborazioni video odierne.

Elaborazioni audio che prima venivano esclusivamente compiute con hardware totalmente analogici, ora possono essere sviluppate in sistemi digitali, con costi enormemente inferiori e qualità migliore.

In questo contesto si pone questo lavoro di testi: la realizzazione di un oggetto a basso costo che possa farsi carico di tutte quelle elaborazioni più comuni, che possono rendersi utili a un professionista o semplicemente a un audiofilo. Proprio questa necessità di creare un sistema professionale con costi ridotti, ha portato a sceglie come nucleo fondamentale del sistema un FPGA di bassa fascia, in contrapposizione ai sistemi, basati su più costosi DSP, attualmente in commercio.

Questo oggetto si chiama Audio Digital Signal Processor.

2. Il Digital Signal Processor

Un Digital Signal Processor (o DSP come lo chiameremo d'ora in avanti) è un oggetto che, come dice il nome stesso, è preposto all'elaborazione di segnali digitali. In campo audio si indica con il termine DSP un sistema con degli ingressi audio digitali o analogici (nel qual caso è previsto uno stadio di conversione analogico digitale), che vengono processati e posti, dopo adeguata miscelazione, su un certo numero di uscite. Esistono varie tipologie di DSP, verranno quindi ora descritte le funzionalità e le caratteristiche del sistema di cui è oggetto questa tesi.

2.1 Funzionalità

Il sistema si propone di gestire quattro ingressi, sia analogici (capitolo 3), con dinamica di 24 Volt picco picco, sia digitali, in formato AES/EBU (capitolo 4). I quattro ingressi vengono quindi elaborati da un blocco di processamento per ogni canale. L'utente ha la possibilità di scegliere, per ogni canale, se gestirlo singolarmente, o in *link* con altri canali.

Successivamente un mixer si occupa di instradare sulle otto uscite i segnali in ingresso dopo il processamento; anche in questo caso ogni uscita può essere gestita in modo totalmente autonomo o meno a discrezione dell'utente.

Ogni canale in uscita dal mixer passa, infine, attraverso una catena di elaborazione per giungere in uscita. Anche in questo caso, i segnali possono forniti al mondo esterno sia in formato digitale, AES/EBU, sia in formato analogico, il che porta ad avere uno stadio di conversione digitale analogico. Ugualmente a prima le catene di elaborazione possono essere gestite in modo indipendente o, a discrezione dell'utente, possono essere gestite a gruppi.

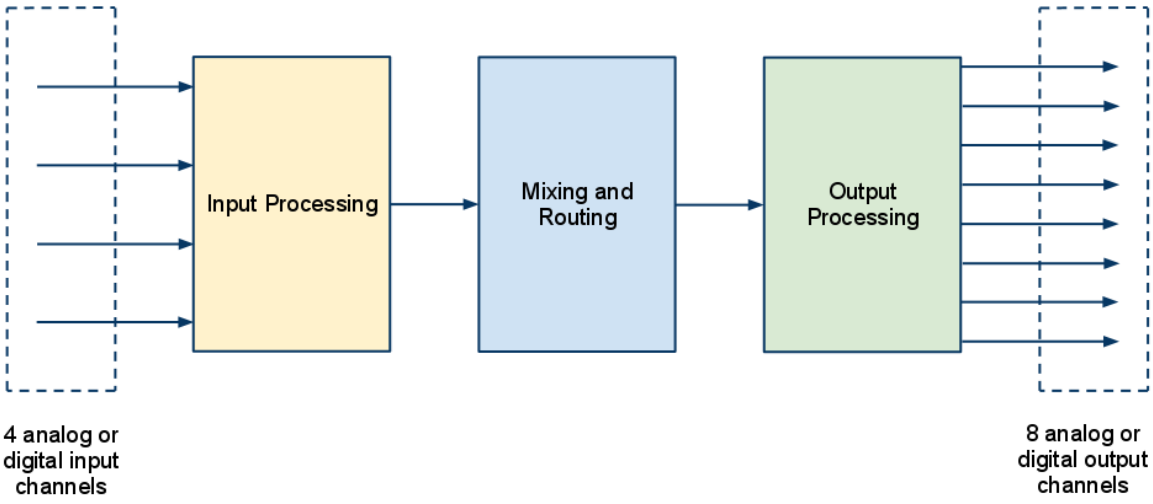

Figura 2.1 - Schema funzionale generale del DSP

In Figura 2.2 sono mostrate le catene di elaborazione d'ingresso.

Il segnale passa, per prima cosa, in uno stadio ritardatore che può essere utile, ad esempio, nel caso in cui le fonti dei segnali siano dei microfoni posti a distanze diverse dalla sorgente. Grazie a questi stadi, i ritardi sui vari canali possono essere riallineati. L'utente può gestire questo elemento andando lui stesso a calcolare i ritardi reciproci o lasciando che sia il sistema a calcolarli, a partire dalla distanza tra le sorgenti.

Successivamente a questo stadio, vi è un blocco di guadagno: a un *master* (guadagno che si applica a tutti i canali) si somma il guadagno di singolo canale.

I segnali arrivano quindi all'equalizzatore, che può essere di tipo grafico, a 31 bande, o parametrico a 9 punti. L'equalizzatore grafico consiste in una catena di 31 filtri IIR del second'ordine, con frequenze d'intervento distanziate a 1/3 di ottava. La risposta dell'eqalizzatore può simulare quella dei classici equalizzatori analogici (in cui le varie campane di filtraggio si influenzano reciprocamente) o essere di tipo *True-Response*, ricalcando i punti del grafico. L'equalizzatore parametrico consente invece l'introduzione di 7 filtri di tipo *peak* (filtri a campana), con possibilità di variazione del centro banda, del fattore di qualità e dell'ampiezza, e due shelving filter (filtri che amplificano o attenuano determinate frequenze, lasciando invariate le altre), per coprire le basse e le alte frequenze.

Ultimo step della catena d'ingresso è il noise gate, ossia un blocco preposto alla riduzione del rumore di fondo. Questo blocco rientra nella categoria dei processori di dinamica.

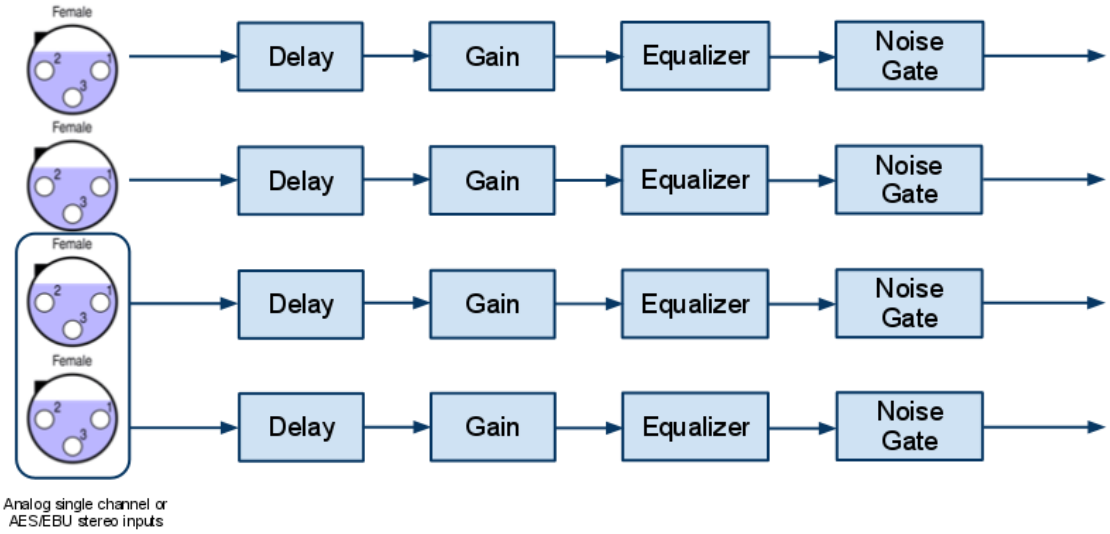

Figura 2.2 - Stadio d'ingresso

In Figura 2.3 è mostrato lo schema a blocchi dello stadio di mixing. Per ognuna delle otto uscite è possibile miscelare i quattro canali d'ingresso a piacimento, fornendo ad ogni segnale il guadagno desiderato.

Per normalizzare le uscite ed evitare problemi di overflow il segnale viene diviso per la somma dei guadagni dati all'ingresso.

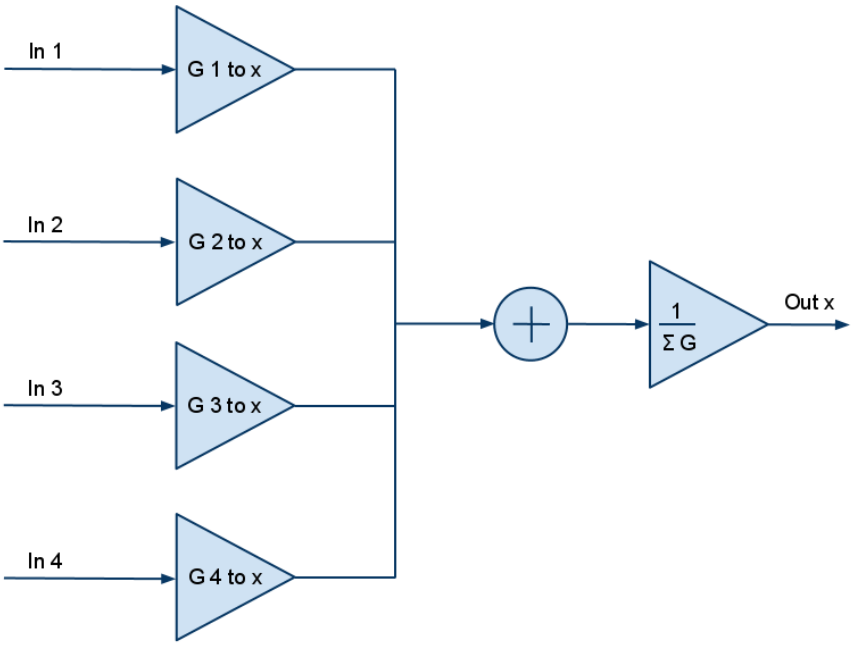

Figura 2.3 - Mixer e Routing

Infine ognuna delle otto uscite passa da una catena di processamento come quella descritta in Figura 2.4.

Il primo elemento di elaborazione è un crossover, che può essere di tipo passa alto, passa basso o passa banda, e permette di selezionare le bande d'interesse per quella determinata uscita. Questo step è fondamentale per poter fornire i segnali agli altoparlanti. Come noto, infatti, esistono diversi tipi di diffusori, ed è necessario, per ottenere le migliori prestazioni, fornire ad essi segnali con caratterizzazione in frequenza adeguata. È lasciata all'utente la scelta del tipo di crossover, tra filtri di Bessel, di Butterworth o di Linkwitz-Riley, e la sua pendenza, a 6, 12, 18 o 24 dB/oct.

Segue, a questo punto un equalizzatore identico a quello della catena d'ingresso, con modalità di funzionamento parametrica o grafica.

Successivamente l'elaborazione del segnale passa al processore di dinamica, che si occupa di gestire la compressione del segnale.

La catena si chiude con uno stadio di guadagno e un blocco ritardatore analogamente a quanto avviene in ingresso.

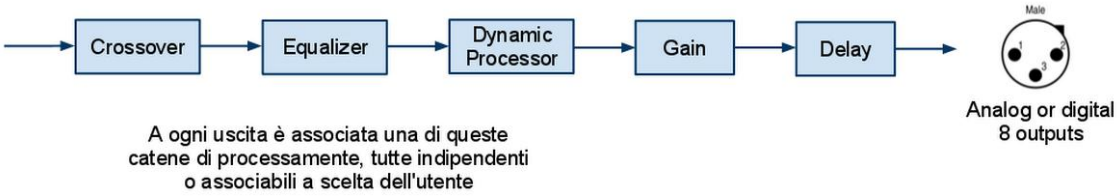

Figura 2.4 - Stadio d'uscita

Tutta la catena appena descritta è a completa disposizione dell'utente, che può modificarne i parametri, sia da una comoda interfaccia utente in posizione frontale, grazie a 24 tasti, un rotary/push controller e un display lcd, sia grazie a una più potente GUI su pc.

È inoltre possibile, grazie a dei *link-channel,* collegare più DSP assieme in modo da aumentare il numero di ingressi e uscite. Il mixer infatti, è creato in modo da poter miscelare fino a 20 canali contemporaneamente.

Ulteriore feature è quella del Real Time Analyzer: grazie a un ingresso aggiuntivo , vi è la possibilità, a impianto spento, di analizzare la risposta in frequenza dell'ambiente, di calcolare il tempo di volo e di avere altre indicazioni molto utili al fine di ottimizzare la resa acustica.

Il sistema funziona internamente usando segnali a 24 bit con frequenza di campionamento a 48 KHz. Tutte le operazioni di filtraggio avvengono in floating point a 36 bit. L'FPGA riesce a compiere circa 1,4 giga FLOPS (Operazioni floating point per secondo).

2.2 Sistema

A livello generale il sistema si presenta come in Figura 2.5.

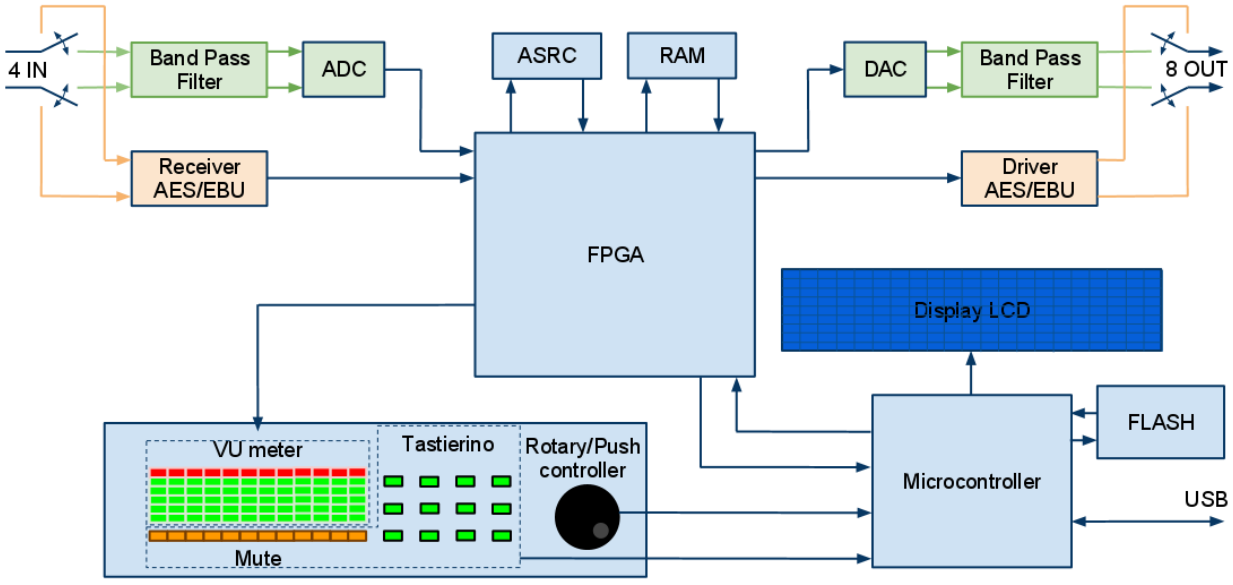

Figura 2.5 - Schema hardware

In alto a sinistra possiamo notare gli ingressi. Grazie a degli switch comandati dal'FPGA, viene scelto se usare gli ingressi analogici o digitali, dirottando i segnali in modo adeguato. Nel primo caso, i segnali differenziali vengono filtrati (capitolo 3.1) e portati agli ADC, che, tramite bus I2S,

forniscono dati a 24 bit campionati alla frequenza di 48KHz. Alternativamente, se la scelta ricade sul segnale digitali, le linee differenziali vengono portate a un ricevitore digitale, che rigenera i livelli e li fornisce in modalità *single-ended* all'FPGA. In questo caso, lo stream di dati viene decodificato e, come vedremo nel capitolo 4.2, viene mandato all'*Asynchronous Sample Rate Converter* (ASRC) per la conversione della frequenza di campionamento.

L'FPGA è l'elemento centrale del progetto (capitolo 8): essa infatti compie l'elaborazione in tempo reale e gestisce i flussi dati. Una RAM esterna è usata come memoria per la generazione dei ritardi.

In alto a destra è possibile notare lo stadio d'uscita. Se il canale deve essere fornito tramite uscita analogica, esso viene trasmesso tramite protocollo I2S al DAC, che effettua la conversione. Il segnale differenziale analogico viene quindi filtrato (capitolo 3.2) e fornito in uscita tramite un selettore. Se invece è selezionata l'uscita differenziale, viene fornito un segnale AES/EBU single ended al driver che provvede a renderlo differenziale e a trasmetterlo al connettore.

In basso possiamo invece notare tutta la parte relativa interfaccia utente, gestita quasi interamente da un microcontrollore (capitolo 9). Il microcontrollore, infatti, si preoccupa di andare a leggere i tasti e il rotary controller e di gestire la visualizzazione su display. Altra funzione del microcontrollore è quella di calcolere e comunicare all'FPGA tutti i parametri di funzionamento. Infine si pone come interfaccia USB per l'uso del sistema con interfaccia grafica su PC.

Il VU meter, composto da 6 led per ogni canale, e i led incorporati in ogni tasto, sono invece gestiti dall'FPGA attraverso 16 latch di tipo D.

Nei prossimi capitoli verrà fornita una descrizione dettagliata di tutti gli aspetti riguardanti la progettazione che mi hanno coinvolto in prima persona.

3. Interfacciamento analogico

Come già detto, il sistema è in grado di ricevere segnali sia analogici che digitali. In ognuno dei due casi il collegamento elettrico avviene mediante un connettore XLR (chiamato anche connettore Cannon) a 3 poli (vedi Figura 3.1).

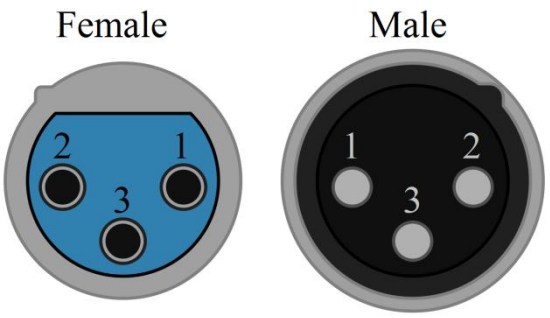

Figura 3.1 - Interfacciamento analogico - Connettori XLR (1 Massa, 2 Hot, 3 Cold)

Nel caso di ingresso analogico, il segnale differenziale, deve essere filtrato per essere poi campionato dall'ADC. Il filtraggio è fondamentale per evitare problemi di *aliasing* e per ridurre i disturbi. Nonostante il segnale in ingresso venga sempre filtrato alla sorgente, bisogna evitare che eventuali interferenti, che si accoppiano alla linea, lunga anche svariati metri, vadano a degradare il rapporto segnale rumore. La specifica del rapporto segnale rumore è, infatti, una delle più stringenti per apparecchiature orientate al mercato professionale.

Andiamo quindi ad analizzare i filtri passabanda d'ingresso e d'uscita.

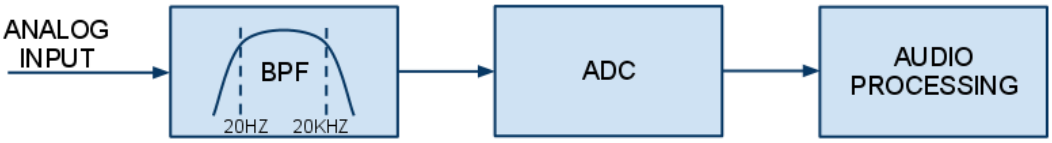

Figura 3.2 - Catena ingresso analogico

Figura 3.3 - Catena uscita analogica

3.1 Filtraggio d'ingresso

In ingresso lo stadio di condizionamento è composto dal filtro passabanda mostrato in Figura 3.4. Ricordiamo che il segnale audio ha una banda udibile che va da circa 20 Hz a 20 KHz.

A bassa frequenza troviamo uno zero nell'origine e un polo a circa 6 Hz. Ne segue una banda passante che arriva a circa 60 KHz dove due poli complessi coniugati fanno scendere la funzione di trasferimento con 40 dB per decade. Il filtro introduce anche uno zero ad alta frequenza che non influenza la funzione di trasferimento.

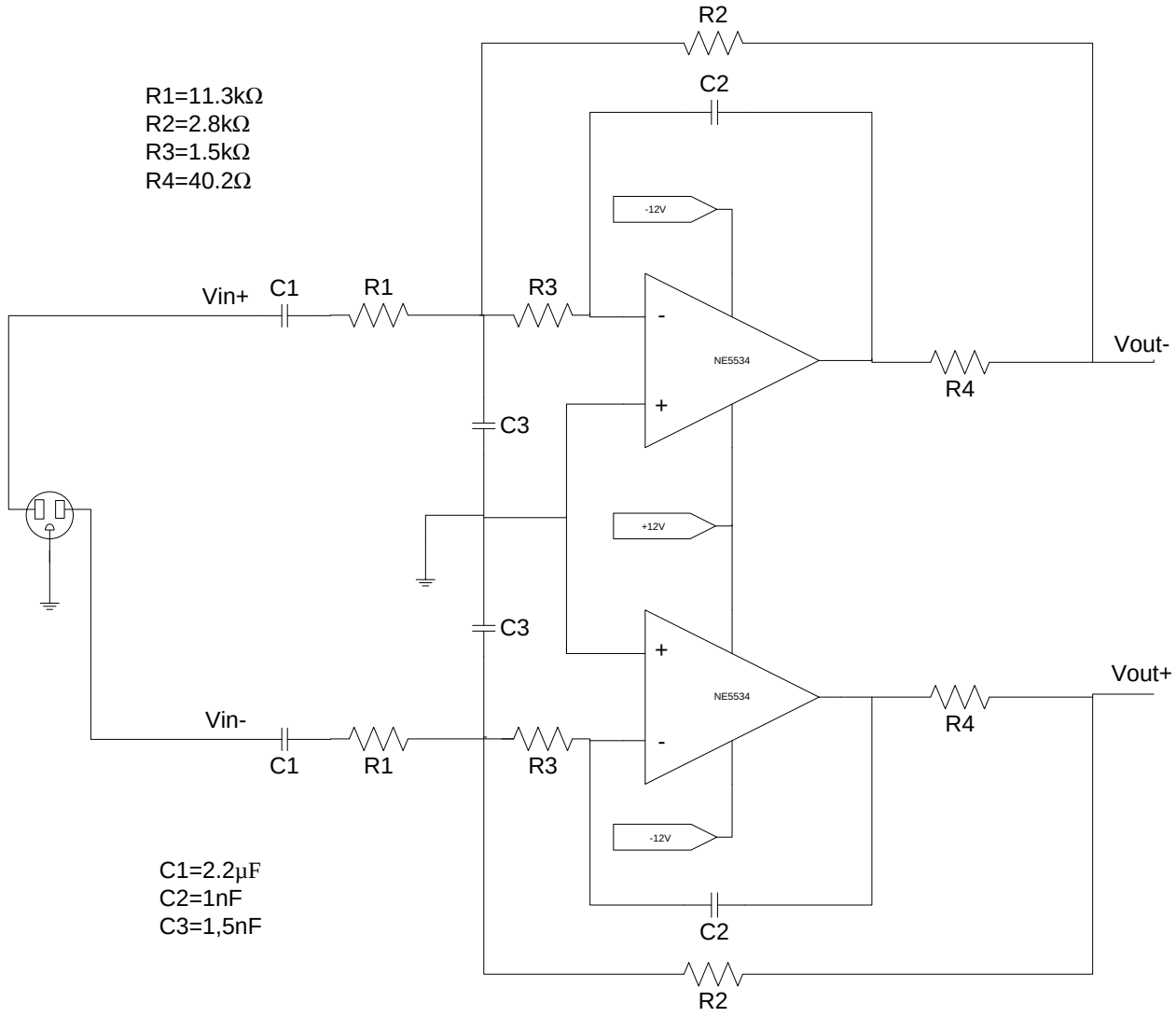

Figura 3.4 - Filtro completo d'ingresso

Per studiare il comportamento si considera solo uno dei due rami speculari, andando ad analizzarlo prima a bassa frequenza e poi nel caso complementare.

Nel primo caso, come si può osservare dall'equivalente di bassa frequenza mostrato in Figura 3.5, il circuito si comporta come un filtro passa alto, con zero nell'origine e un polo a $f_{p1} = \frac{1}{2\pi C_1 R_1} \cong 6.6\ Hz$.

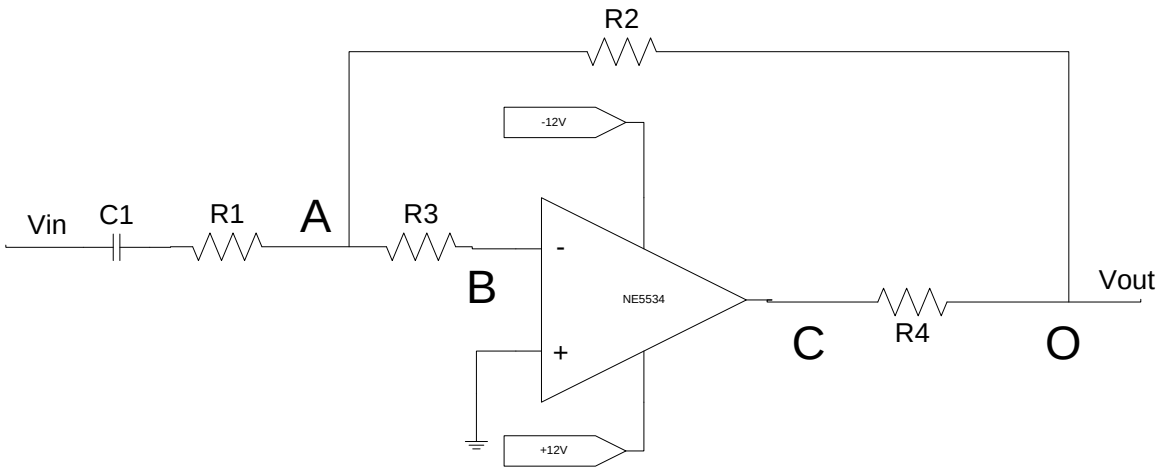

Figura 3.5 - Filtro d'ingresso a bassa frequenza

La funzione di trasferimento può essere ricavata scrivendo la legge di Kirchhoff delle correnti al nodo A.

$$\frac{V_{in}\, sC_1}{1 + sC_1R_1} + \frac{V_{out}}{R_2} = 0 \qquad (1)$$

Da cui

$$\frac{V_{out}}{V_{in}} = -\frac{sC_1R_2}{1 + sC_1R_1} \qquad (2)$$

Questo primo stadio serve a disaccoppiare il segnale in continua e a eliminare una buona parte del rumore 1/f senza intaccare il segnale. Possiamo notare, sostituendo i valori dei componenti, come il modulo della funzione di trasferimento sia 0.247 a media frequenza. Questo per adattare la dinamica del segnale d'ingresso, che ricordiamo essere di 24V picco-picco, ai 6 V del convertitore A/D.

Il secondo filtro impone un polo doppio a $f_{p2,3} = 63.6\, KHz$ con fattore di qualità $Q = 0.54$, in modo da non introdurre picchi di risonanza.

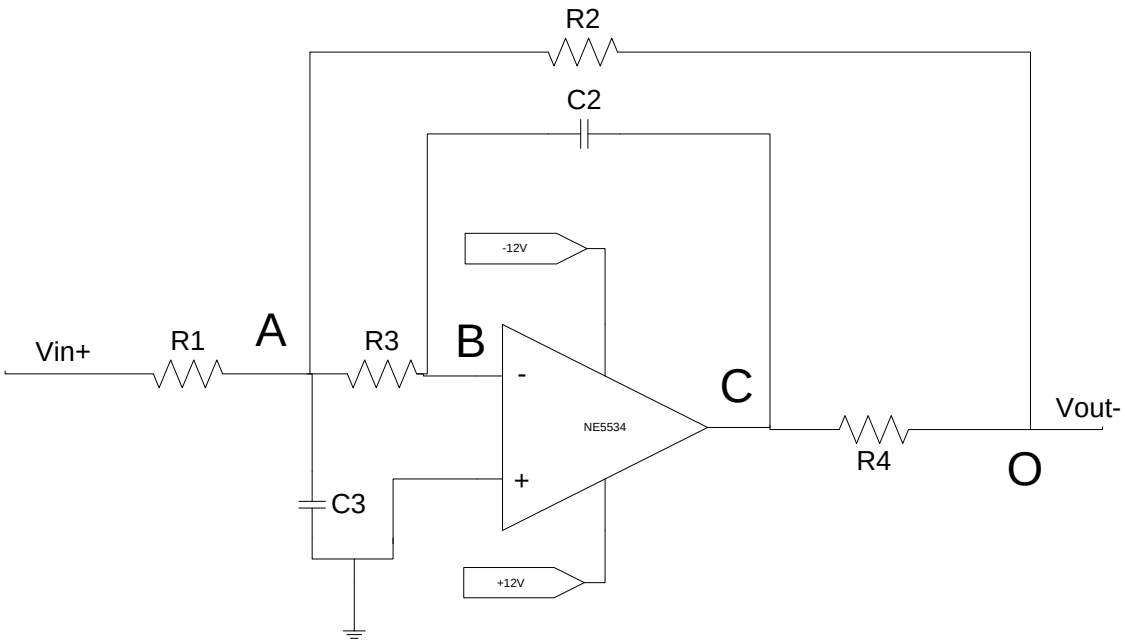

Figura 3.6 - Filtro d'ingresso ad alta frequenza

Ipotizzando C_1 un cortocircuito alle frequenze d'interesse (vedi Figura 3.6) , utilizzando le leggi di Kirchhoff alle correnti ai nodi A, B e O possiamo scrivere il seguente sistema

$$\begin{cases} \dfrac{V_{in}-V_a}{R_1}+\dfrac{V_{out}-V_a}{R_2}-\dfrac{V_a}{R_3}-V_a sC_3=0 & (A) \\ \dfrac{V_{out}-V_c}{R_4}-\dfrac{V_{out}-V_a}{R_2}=0 & (O) \\ V_c sC_2+\dfrac{V_a}{R_3}=0 & (B) \end{cases} \tag{3}$$

per sostituzione si ricava

$$V_c=-\frac{V_a}{sC_2R_3} \tag{4}$$

$$V_a=-V_{out}\frac{R_2-R_4}{R_4}\frac{sC_2R_3}{\frac{R_2}{R_4}+sC_2R_3} \tag{5}$$

e quindi usando la (4) e la (5) nella (A), dopo le opportune semplificazioni si ottiene la (6)

$$\frac{V_{in}}{R_1}+V_{out}\left[\frac{R_2-R_4}{R_4}\frac{sC_2R_3}{\frac{R_2}{R_4}+sC_2R_3}\left(\frac{R_1R_2+R_2R_3+R_1R_3}{R_1R_2R_3}+sC_3\right)+\frac{1}{R_2}\right]=0 \tag{6}$$

L a funzione di trasferimento associata è quindi

$$H(s) = \frac{V_{out}}{V_{in}} = \\ = -\frac{R_2}{R_1}\frac{1 + sC_2R_3\frac{R_4}{R_2}}{1 + sC_2R_3\left[\frac{R_4}{R_2} + \frac{R_1R_2 + R_2R_3 + R_1R_3}{R_1R_2R_3}(R_2 - R_4)\right] + s^2C_2R_3C_3R_2\frac{R_2 - R_4}{R_2}} \quad (7)$$

Sostituendo quindi i valori numerici si ottiene

$$H(s) = -0.247\frac{1 + s21.5\ 10^{-9}[sec]}{1 + s4.6\ 10^{-6}[sec] + s^2 6.2\ 10^{-12}[sec^2]} \quad (8)$$

Come si può facilmente notare il filtro guadagna 0.247 in banda passante, coerentemente a quanto succedeva analizzando il filtro passa alto.

A questo punto possiamo scrivere la funzione di trasferimento complessiva (approssimata) del filtro come

$$H(s) \\ = -\frac{sC_1R_2}{1 + sC_1R_1}\frac{1 + sC_2R_3\frac{R_4}{R_2}}{1 + sC_2R_3\left[\frac{R_4}{R_2} + \frac{R_1R_2 + R_2R_3 + R_1R_3}{R_1R_2R_3}(R_2 - R_4)\right] + s^2C_2R_3C_3R_2\frac{R_2 - R_4}{R_2}} \quad (9)$$

Ovvero, sostituendo i valori numerici

$$H(s) = -\frac{s6.16\ 10^{-3}}{1 + s24.86\ 10^{-3}}\frac{1 + s21.5\ 10^{-9}}{1 + s4.6\ 10^{-6} + s^2 6.2\ 10^{-12}} \quad (10)$$

In Figura 3.7 si può osservare la simulazione matlab del modulo della funzione di trasferimento.

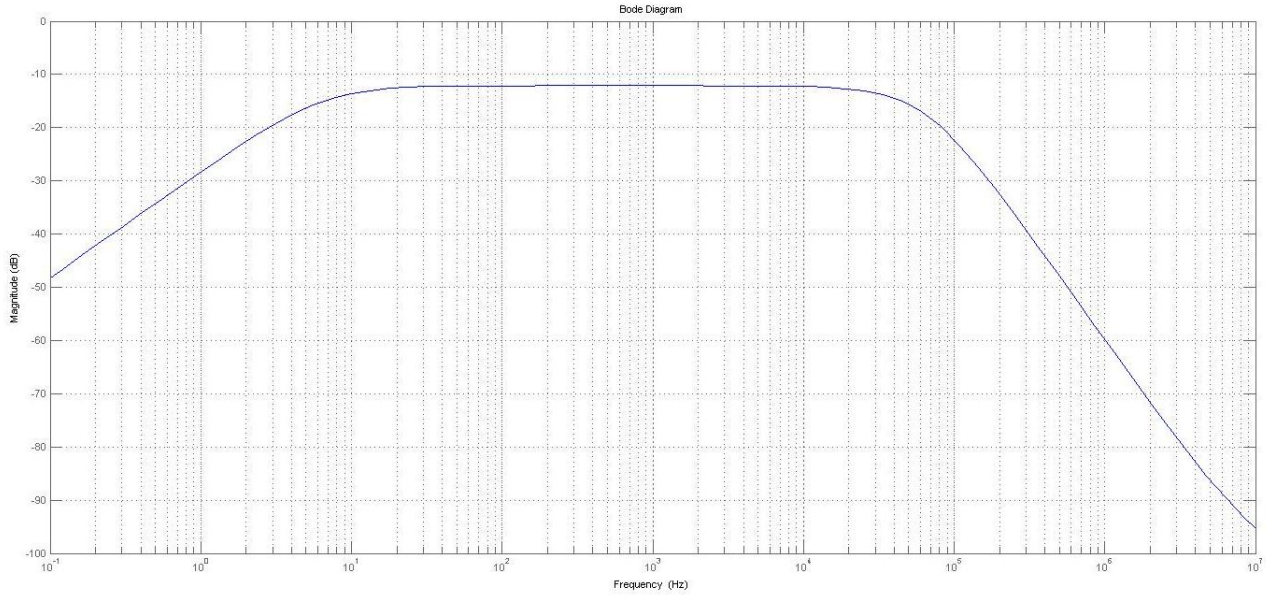

Figura 3.7 - Simulazione funzione di trasferimento con matlab

In Figura 3.8, invece, è rappresentata graficamente la simulazione completa del filtro utilizzando Sapwin, un software preposto alla simulazione di filtri. Come si può facilmente notare le approssimazioni fatte sono del tutto sensate e non portano a errori rilevanti.

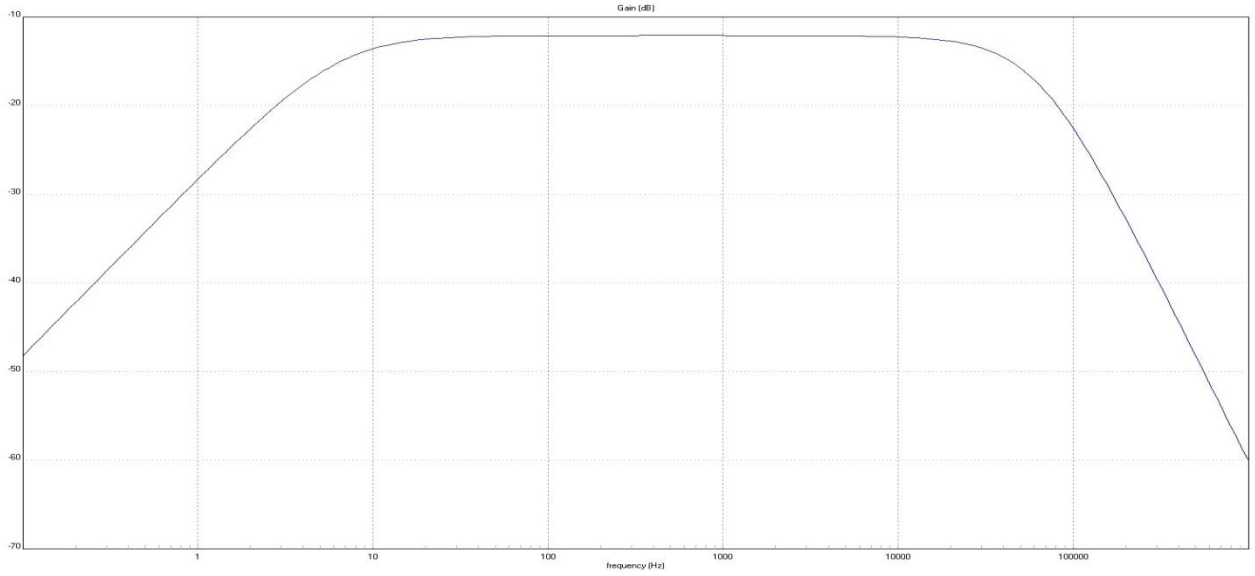

Figura 3.8 - Simulazione filtro completo con Sapwin

3.2 Filtraggio d'uscita

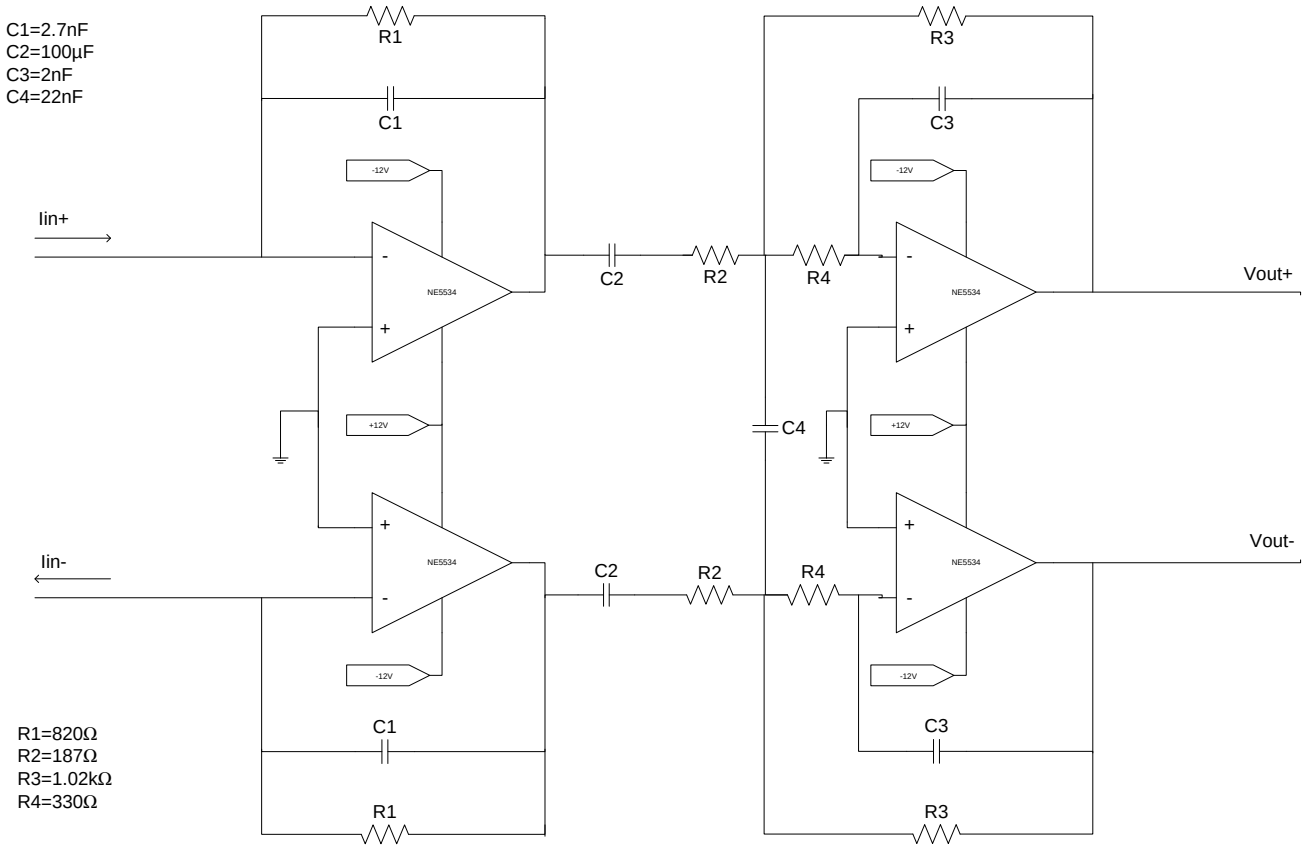

Figura 3.9 - Schema completo filtro d'uscita

Il DAC d'uscita (il PCM1798 di Texas Instruments) è un convertitore in corrente. L'uscita ha una polarizzazione di 2 mA e un valore picco picco di 4mA. Si rende quindi necessaria un'amplificazione, un disaccoppiamento in continua e un filtraggio passabasso oltre i 20 KHz.

Il filtro completo che opera questa elaborazione è mostrato in Figura 3.9. Come consigliato dal produttore il primo filtro incontrato è un passa basso con $f_{p1} = 71.9\ KHz$. Successivamente vi è il disaccoppiamento della corrente continua che introduce uno zero nell'origine e un polo a circa $f_z = 8.5\ Hz$. Infine vi è un filtro del secondo ordine che implementa due poli complessi coniugati a $f_{p2,3} = 30\ KHz$ con fattore di qualità Q=0.87. Il fattore di qualità più elevato del filtro in ingresso, fa si che la frequenza di taglio possa essere messa a frequenza minore senza avere perdita in banda passante. Mentre prima il segnale proveniva da una sorgente già filtrata e quindi il filtro poteva essere più blando, in questo caso bisogna abbattere subito tutti i disturbi e le distorsioni armoniche introdotte, soprattutto, dalle periferiche digitali e dall'alimentatore switching, operando un taglio più netto.

Lo studio del filtro avviene, anche in questo caso, studiando un solo ramo, essendo i due perfettamente identici. Per fare ciò bisogna ricordare che la capacità C_4 andrà considerata vrso massa e grande il doppio.

Passando ad analizzare il primo stadio del filtro, si può notare (Figura 3.10) che esso è un semplice filtro passabasso. La funzione di trasferimento può quindi essere scritta come

$$H_1(s) = \frac{V_{out1}}{I_{in+}} = -\frac{R_1}{1 + sR_1C_1} \tag{11}$$

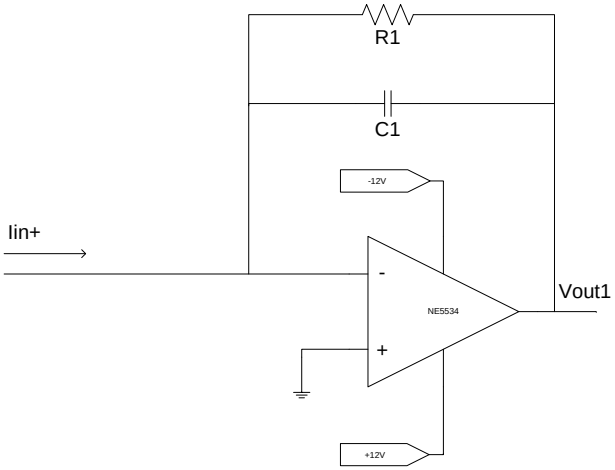

Figura 3.10 - Primo stadio filtro d'uscita

L'analisi del secondo stadio, avviene, come per il filtro d'ingresso, considerando separatamente gli equivalenti a bassa e ad alta frequenza.

Nella prima situazione il circuito equivalente diventa quello in , ovvero un semplice filtro passa alto con guadagno. La funzione di trasferimento in questa situazione è quindi

$$H_2(s)|_{LF} = \frac{V_{out+}}{V_{out1}} = -\frac{sC_2R_3}{1 + sC_2R_2} \tag{12}$$

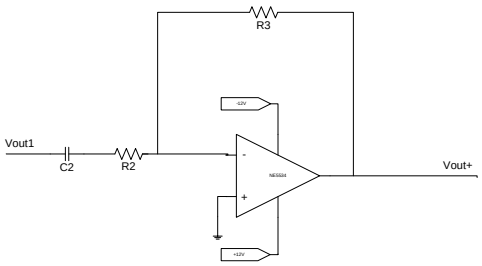

Figura 3.11, ovvero un semplice filtro passa alto con guadagno. La funzione di trasferimento in questa situazione è quindi

$$H_2(s)|_{LF} = \frac{V_{out+}}{V_{out1}} = -\frac{sC_2R_3}{1+sC_2R_2} \tag{13}$$

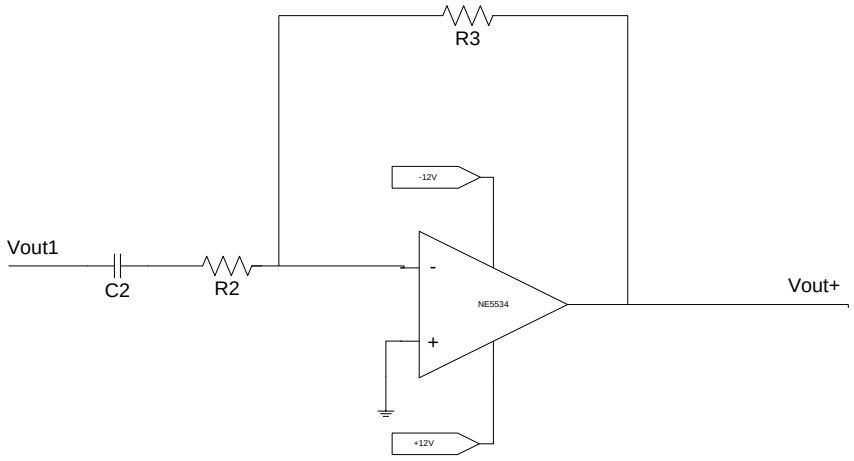

Figura 3.11 - Secondo stadio filtro d'uscita - Bassa frequenza

Infine si analizza il circuito ad alta frequenza (con riferimento alla Figura 3.12).

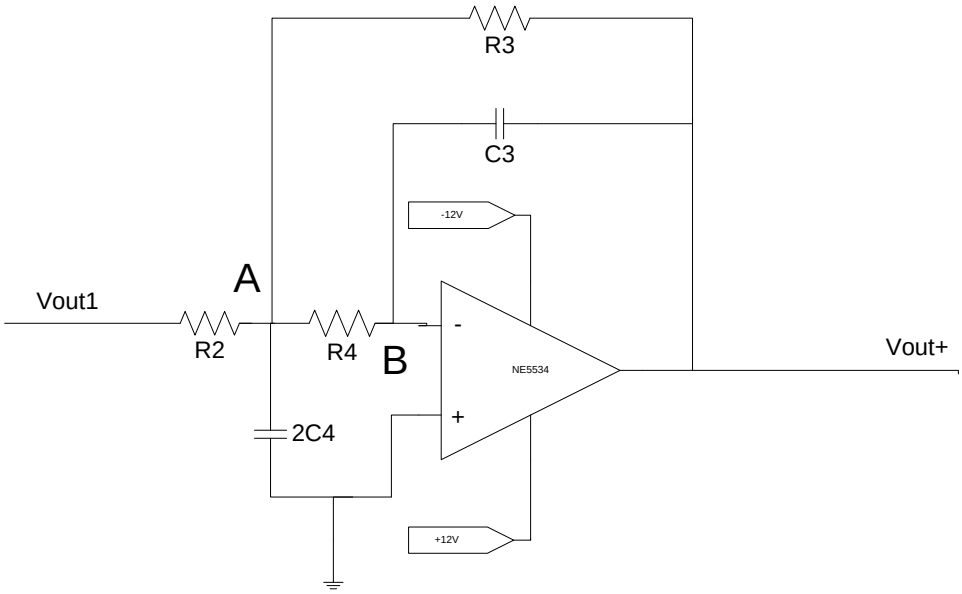

Figura 3.12 - Secondo stadio filtro d'uscita - Alta frequenza

Si scrivono, per prima cosa, le leggi di Kirchhoff alle correnti ai nodi A e B ricavando il seguente sistema

$$\begin{cases} \frac{V_{out1} - V_a}{R_2} + \frac{V_{out} - V_a}{R_3} - \frac{V_a}{R_4} - V_a s 2C_4 = 0 \quad (A) \\ V_{out+} s C_3 + \frac{V_a}{R_4} = 0 \quad (B) \end{cases} \tag{14}$$

Dalla (B) si ottiene

$$V_a = -V_{out+} s C_3 R_4 \tag{15}$$

e quindi, sostituendo la (14) nella (A), dopo le opportune semplificazioni

$$\frac{V_{out1}}{R_2} + V_{out+} \left[s^2 C_3 2C_4 R_4 + s C_3 \frac{R_2R_4 + R_2R_3 + R_3R_4}{R_2R_3} + \frac{1}{R_3} \right] = 0 \tag{16}$$

La funzione di trasferimento ad alta frequenza è quindi

$$H_2(s)|_{HF} = \frac{V_{out+}}{V_{out1}} = -\frac{R_3}{R_2} \frac{1}{1 + sC_3 \frac{R_2R_4 + R_2R_3 + R_3R_4}{R_2} + s^2 C_3 2C_4 R_3 R_4} \tag{17}$$

È immediato notare come, a media frequenza, entrambe le funzioni di trasferimento guadagnino $\frac{R_3}{R_2}$. La funzione di trasferimento complessiva, risulta quindi la (17):

$$H(s) = \frac{V_{out+}}{I_{in+}} = \\ = \frac{R_1}{1 + sR_1C_1} \frac{sC_2R_3}{1 + sC_2R_2} \frac{1}{1 + sC_3 \frac{R_2R_4 + R_2R_3 + R_3R_4}{R_2} + s^2 C_3 2C_4 R_3 R_4} \tag{18}$$

e, sostituendo i valori numerici

$$H(s) \\ = \frac{820[\Omega]}{1 + s2.214\ 10^{-6}[sec]} \frac{s0.102\ [sec]}{1 + s18.7\ 10^{-3}[sec]} \frac{1}{1 + s6.3\ 10^{-6}[sec] + s^2 29.6\ 10^{-12}[sec^2]} \tag{19}$$

Il filtro ha quindi una transimpedenza di 4472Ω a media frequenza; questo fa si che il segnale in uscita potrà avere dinamica di ±9V circa.

Infine, per completezza, in Figura 3.13 e Figura 3.14 si possono comparare, prima il grafico della funzione di trasferimento appena ricavata e simulata con matlab, poi il risultato della simulazione completa del filtro con Sapwin. Anche in questo caso non sono riscontrabili evidenti differenze.

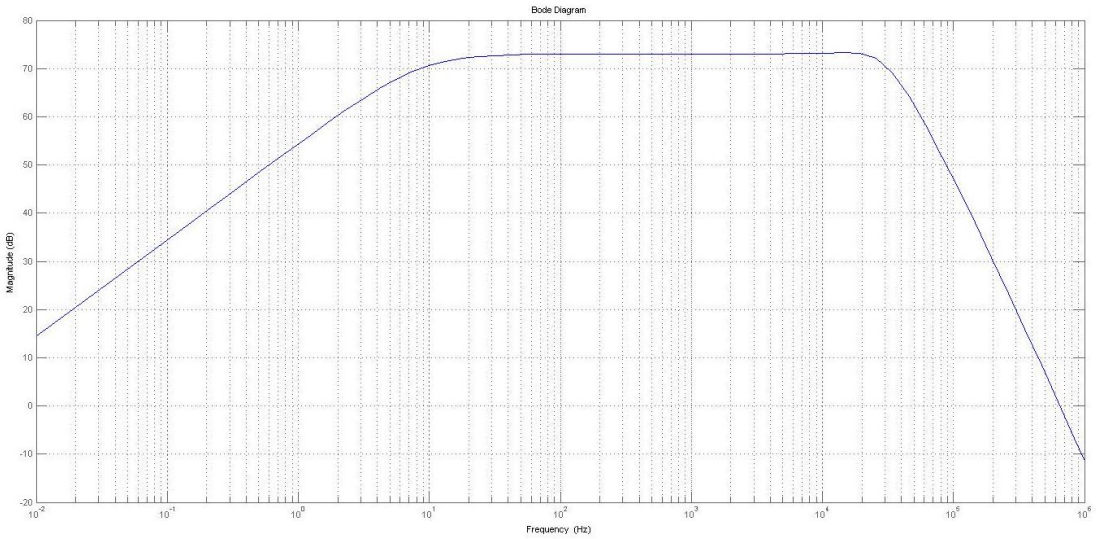

Figura 3.13 - Funzione approssimata filtro d'uscita simulata con matlab

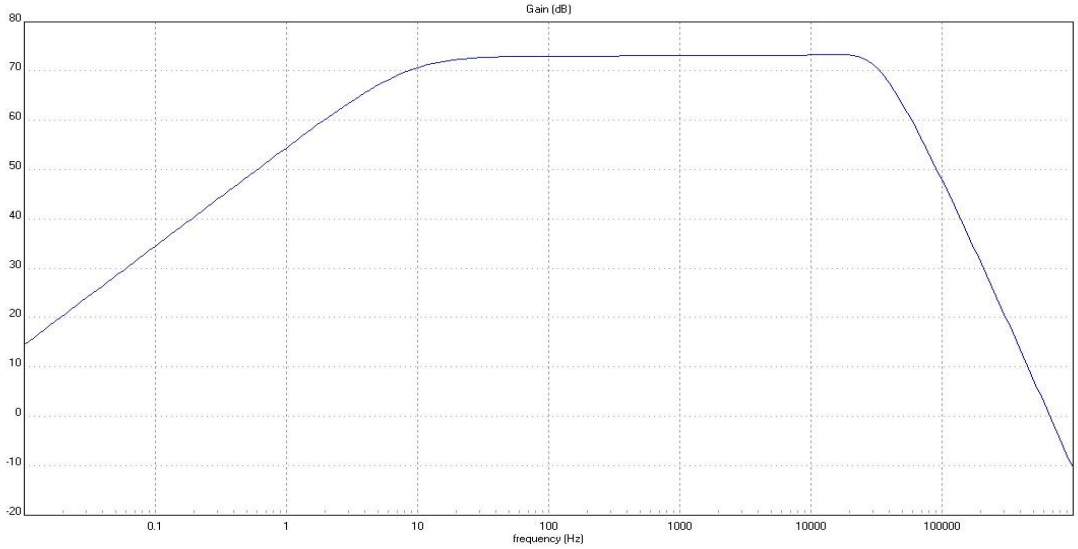

Figura 3.14 - Filtro simulato con Sapwin

4. Interfacciamento digitale

In campo professionale il protocollo largamente più utilizzato per la trasmissione digitale dell'audio *off board* è l'AES/EBU. Negli ultimi anni, in ambito *consumer,* si è molto diffuso lo standard di comunicazione S/PDIF, più conosciuto al grande pubblico, ma che è semplicemente, l'adattamento dello standard professionale a esigenze meno stringenti. In questo capitolo si andrà ad analizzare dettagliatamente lo standard AES/EBU, sia a livello fisico, si a livello di protocollo, andando ad evidenziare i pregi, ma anche i difetti e le problematiche da affrontare quando si ha a che fare con questo tipo di comunicazione.

4.1 Lo standard AES/EBU

L'AES/EBU è uno dei principali formati standard per l'audio digitale. Deriva il suo nome dall'Audio Engineering Society e dall'European Broadcasting Union che lo hanno definito. Oltre ad essere descritto il protocollo di comunicazione, sono definiti anche la tipologia di connettori e le caratteristiche fisiche.

Il protocollo permette la trasmissione di due canali audio a distanze fino a 100 metri. Grande vantaggio di questo standard è quello di non necessitare di un segnale di sincronizzazione, in quanto il segnale di clock può essere estratto dal segnale stesso.

Per questo progetto è stato utilizzato lo standard fisico AES3, bilanciato a 3 cavi, in quanto quello più diffuso nelle applicazioni di questo tipo. Questo permette anche la maggiore flessibilità, infatti, anche se per ottenere le migliori prestazioni andrebbero utilizzati cavi ad hoc, è possibile impiegare gli stessi che si usano per la trasmissione analogica. Inoltre, così facendo, si è potuto realizzare un sistema che con un singolo connettore è capace di gestire sia l'interfaccia analogica che quella digitale.

4.1.1 Requisiti elettrici

Il cavo deve essere bilanciato con impedenza caratteristica nominale di 110 Ω da 0.1 a 128 volte il massimo framerate. L'utilizzo degli stessi cavi usati per la trasmissione analogica fa correre il rischio che l'utente finale impieghi cavi XLR con impedenza costante solo in banda audio: ciò può portare a dei malfunzionamenti se si dovessero utilizzare in lunghi collegamenti. Essendo però un oggetto professionale, si lascia la responsabilità della giusta scelta dei cavi ai tecnici audio che lo utilizzeranno, confidenti della loro preparazione a questo riguardo.

L'ampiezza del segnale differenziale deve essere compresa tra i 2 e i 7 volt picco picco. In questo progetto per la ricezione si utilizza un *receiver* AM26LV32E, che rigenera i livelli e fornisce un segnale single ended all'FPGA, e un *driver* AM26LV31E che, dato il segnale uscente dall'FPGA, lo fornisce in modalità differenziale con dinamica di 3.3V.

4.1.2 Protocollo di trasmissione

Lo AES/EBU è stato sviluppato principalmente nella prospettiva di avere un unico standard che permettesse di trasmettere e ricevere segnali audio sia a 48 KHz che a 44.1 KHz. Per la sua struttura intrinseca, infatti, grazie a questo protocollo, con gli sviluppi successivi alla prima

implementazione, è stata resa possibile la trasmissione di segnali audio a qualsiasi frequenza di campionamento fino a 196 kHz. Questo perché il *baud rate* risultante, non è fissato, ma è semplicemente definito come $128 * f_{sampling}$.

Il protocollo in se è diviso in vari livelli. La struttura di più alto livello è composta da 192 *frames* e prende il nome di *Audio Block*. Ogni frame è quindi numerato da 0 a 191.

Ogni frame è composto da uno *stream* di 64 bit. Ogni bit è trasmesso con codifica BMC (*Biphase Mark Code*): il bit è formato da due stati logici consecutivi, la cui XOR restituisce il valore del bit. La specifica impone che, eccezion fatta per i preamboli, il primo stato logico di un bit sia l'inverso del secondo stato logico del bit precedente. Proprio questa regola che impone almeno una transizione per bit permette al ricevitore di estrarre il clock dal segnale, senza correre il rischio di desincronizzazione a causa di lunghe catene di segnali logici identici.

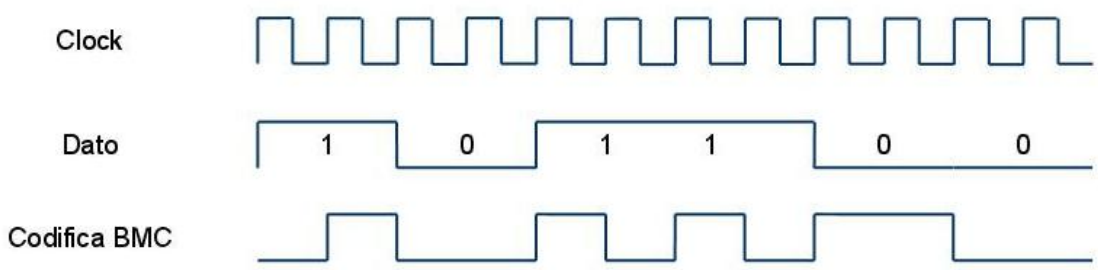

Figura 4.1 - Codifica BMC

Ogni frame è diviso in 2 *sub-frames* da 32 bit che individuano i 2 canali audio. Ogni sub-frame è così suddiviso:

- 4 bit di preambolo
- 24 bit di campione audio
- 1 bit di validità
- 1 bit di user data
- 1 bit di channel status
- 1 bit di parità

Figura 4.2 - Subframe

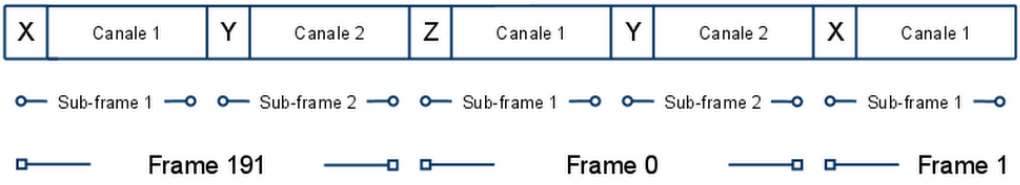

Figura 4.3 - Frames

Gli 8 stati dei 4 bit di preambolo possono essere di 3 tipi diversi, e quindi assumere 6 diverse forme in base all'ultimo bit del frame precedente:

- X - 11100010 o 00011101
- Y - 11100100 o 00011011
- Z - 11101000 o 00010111

Come si può facilmente notare, i primi tre bit del preambolo, non rispettano la regola che impone almeno una transizione per bit: ciò permette al ricevitore di riconoscerli univocamente. Ogni qualvolta viene trasmesso il secondo canale il preambolo sarà di tipo Y, mentre quando viene inviato il primo canale viene inserito il tipo X. Solo quando viene trasmesso il primo canale del frame 0 il preambolo assume la forma di tipo Z. Questo metodo permette l'allineamento del ricevitore al frame 0 anche in caso di errori.

I 24 bit di segnale audio sono trasmessi nell'ordine dall' LSB all' MSB.

Se il bit di validità è 1, indica che il campione mandato non è adatto per una conversione analogica, in quanto, probabilmente il segnale ha componenti in frequenza più alte dei classici 20 kHz.

I bit di user data possono essere usati a piacimento, non vi è una regola che ne definisce un uso preciso. Generalmente sono usati per trasmettere informazioni inerenti il segnale audio.

I bit channel status vengono raggruppati in blocchi di 24 byte, essendo 192 per ogni canale di un audio block. Come mostrato nella Tabella 1, in questi 24 byte vi si possono racchiudere parecchie informazioni , come ad esempio la frequenza di campionamento, il numero di bit per campione ecc.

Tabella 1 - Channel status

	Bit							
Byte	0	1	2	3	4	5	6	7
0	Use of status channel	Linear PCM identification	Audio signal pre-emphasis			Locking of source sample frequency	Sampling frequency	
1	Channel mode				User bit menagement			
2	Use of auxiliary sample bit			Source word lenght and source encoding history			Indication of alignment level	
3.A	Channel number							N=0
3.B	Channel number				Multichannel mode number			N=1
4	Digital audio reference		Reserved	Sampling frequency				SF scaling flag
5	Reserved							
6	Alphanumeric channel origin data							
7								
8								
9								
10	Alphanumeric channel destination data							
11								
12								
13								
14	Local sample address code (32-bit binary)							
15								
16								
17								
18	Time-of-day sample address code (32-bit binary)							
19								
20								
21								
22	Reliability flags							
23	Cyclic redundancy check character							

4.2 L'asynchronous sample rate converter

È già stato detto che lo standard di comunicazione AES/EBU permette al ricevitore di ricostruire il clock a partire dal segnale stesso. Questo metodo da la possibilità quindi di trasmettere uno flusso audio campionato a qualsiasi frequenza, ma pone un grosso problema: il sistema che riceve, che opererà sotto il regime di un clock locale, non sarà sicuramente sincronizzato con il clock del segnale in arrivo. Anche se il valore nominale della frequenza dei due clock è lo stesso, ci sarà comunque una fluttuazione dell'ordine della decina di parti per milione tra i due valori reali. Questo fatto obbliga a inserire nel sistema ricevente un oggetto che preso lo *stream* audio ricevuto a una certa frequenza di campionamento, lo ricampioni alla frequenza di campionamento derivata dal clock locale. Questo oggetto prende il nome di *asynchronous sample rate converter*.

Per realizzare un ASRC vi sono due strade principali. Una strada prevede di tornare nel mondo analogico, quindi passando per un convertitore digitale/analogico e quindi di ricampionare il segnale alla frequenza desiderata. La seconda strada, invece, prevede di rimanere nel mondo digitale.

Quest'ultimo è il metodo più utilizzato perché riesce a mantenere più basso il deterioramento del rapporto segnale rumore. Il principio di funzionamento è molto semplice concettualmente: il segnale in ingresso viene sovracampionato a una frequenza molto maggiore della frequenza di campionamento, utilizzando un interpolatore opportuno (ad esempio un interpolatore di *Lagrange*). Quello che si ottiene è un segnale che si presenta ai nostri occhi come se fosse quasi continuo. A questo punto avviene una decimazione: a ogni colpo di clock del ricevitore, viene restituito il valore più prossimo temporalmente. Con opportuni accorgimenti si possono ottenere efficienze davvero elevate. Nel nostro progetto questo lavoro viene svolto da un componente dedicato , l'SRC4184 di Texas Instruments.

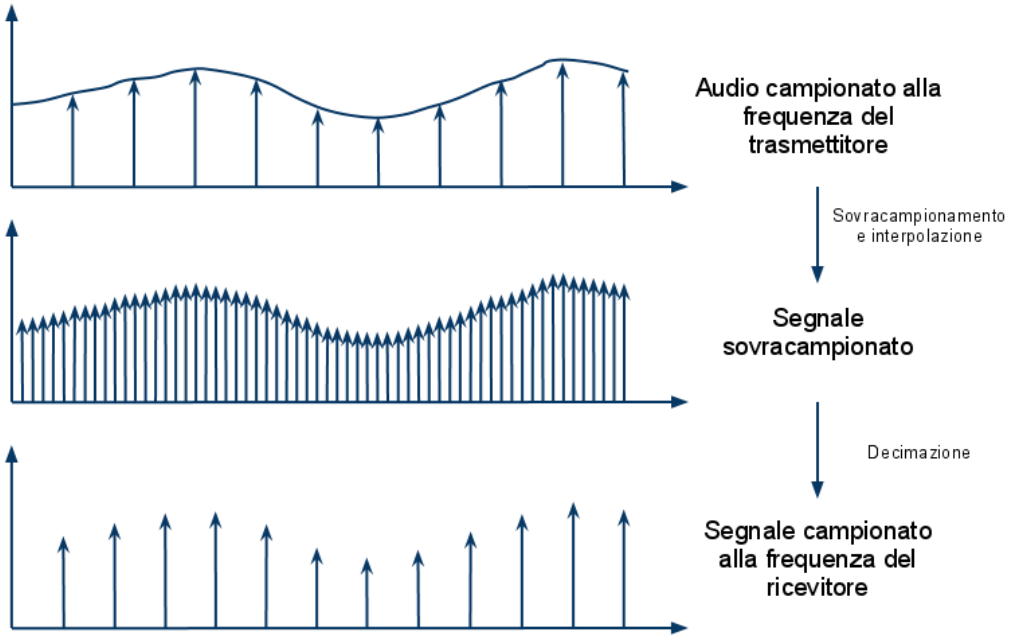

Figura 4.4 – ASRC

5. FPGA Xilinx Spartan 6: panoramica

In questo capitolo viene fatta una panoramica sul fulcro del progetto: il dispositivo FPGA.

Una conoscenza delle caratteristiche di questa logica è fondamentale per comprendere le potenzialità di elaborazione, che ha portato a un sempre maggiore impiego nello sviluppo di apparati elettronici sempre più complessi.

5.1 Caratteristiche del dispositivo

Negli ultimi decenni nel campo della progettazione elettronica, l'incessante progresso tecnologico ha reso di vitale importanza la capacità di realizzare prodotti ad elevata duttilità verso le esigenze dell'utente finale e con tempistiche che rispettino le dinamiche di mercato. Questi due concetti, fondamentali per le aziende che vogliono essere competitive in campo economico, sono ben noti come *customization* e *time-to-market*. I dispositivi logici programmabili, insieme a microprocessori, microcontrollori e DSP, sono tra i circuiti integrati che meglio si prestano a queste esigenze. La filosofia che sta alla base di questi dispositivi, è infatti quella di poter sviluppare rapidamente sistemi specializzati, in modo semplice, e di poterne modificare le funzionalità molto velocemente all'occorrenza, senza dover modificare la struttura hardware del dispositivo.

I dispositivi logici programmabili più importanti in tal senso sono le FPGA (*Field Programmable Gate Array*), ormai divenuti elementi fondamentali in buona parte dei sistemi elettronici digitali. La forza di questi circuiti integrati, sta infatti nel grande potenziale di calcolo che offrono, oltre ad un'elevata duttilità di progetto. Negli ultimi anni le FPGA stanno infatti prendendo il sopravvento sui principali antagonisti, gli ASIC *Gare Array*, grazie al drastico abbattimento dei tempi di attesa, oltre a un notevole risparmio economico in caso di volumi di produzione ridotto. La convenienza nell' utilizzo di dispositivi ASIC emerge solo in presenza di produzioni elevate o quando vi è la necessita di un'elevatissima densità di porte logiche. Questi componenti specializzati riescono infatti ad ottimizzare le risorse, in termini di area di silicio principalmente, in modo più efficiente. Va comunque puntualizzato, che ormai sempre più spesso, anche in caso di scelta di questo tipo di componente, la prototipazione avviene quasi sempre tramite l'utilizzo di FPGA, in modo da abbattere i rischi dovuti a errori di progettazione.

Attualmente, almeno tre società al mondo si sono specializzate nella produzione di FPGA: Xilinx, Altera e Lattice. Per questo progetto è stata scelta una Spartan 6 prodotta da Xilinx, quindi ci concentreremo sui dispositivi prodotti da questa azienda, anche se le strutture interne sono abbastanza simili per tutti i produttori.

Il concetto che sta alla base di un oggetto del genere è tanto semplice, quanto potente: una matrice di piccoli blocchi circuitali programmabili e una rete di interconnessioni configurabili a piacimento. Questo permette all'utente di aver accesso ad una piattaforma hardware integrata completamente riconfigurabile sulla quale è possibile implementare qualsiasi architettura hardware (nei limiti delle risorse del dispositivo).

La grande potenzialità è appunto quella di essere totalmente *general-purpose*, di poter quindi utilizzare questo dispositivo in una molteplicità di applicazioni. Addirittura è possibile personalizzare il singolo prodotto utilizzando la stessa struttura, a seconda delle esigenze precise dell'utilizzatore finale.

Vi è una prima macro differenziazione tra le FPGA, che si dividono in OTP (*one time programmable*, programmabili una sola volta) e quelle riprogrammabili. Le prime, grazie a degli *antifuse* (sostanzialmente dei fusibili integrati che funziono in maniera opposta, e cioè sono degli aperti finché una certa corrente non fa avvenire il contatto) una volta programmate non permettono modifiche. Questo fa si che siano più affidabili, in quanto molto resistenti a fattori come le radiazioni. D'altro canto non permettendo modifiche, non consentono aggiornamenti *firmware* e modifiche successive. In genere sono utilizzate in applicazioni militari e aerospaziali, dove l'affidabilità è di primaria importanza e i budget di costi non sono molto stringenti.

In questo progetto viene invece utilizzato il tipo riprogrammabile. In questi dispositivi è una SRAM o una EEPROM che contiene l'informazione relativa le connessioni e le funzioni logiche. Rimane il limite di essere particolarmente vulnerabili alle radiazioni, aspetto del tutto irrilevante per le applicazioni audio.

Fin'ora ci si è concentrati sugli aspetti positivi relativi all'utilizzo di questi dispositivi. Il più grosso svantaggio che interessa questa classe di componenti è dovuto proprio all'estrema flessibilità della struttura: la possibilità di creare qualsiasi connessione e funzione logica è dovuta alla presenza massiccia di diramazioni gestite da *pass transistor*; questi, a seconda della programmazione, modificano il percorso del segnale, introducendo però elevate resistenze serie e capacità parassite che limitano la velocità di propagazione dello stesso. Questo limite non è presente negli ASIC ad esempio, in cui la struttura, fissata a priori, viene integrata direttamente senza la presenza, ovviamente, di questi transistor.

Altro limite, meno importante, ma comunque da non sottovalutare, è quello della complessità di sintesi: l'implementazione del sistema così come progettato, è molto laboriosa e sono necessari elaboratori particolarmente potenti per poter permettere al progettista di lavorare in maniera efficiente e rapida. Inoltre, nella maggior parte dei casi, il sistema per essere messo a punto, necessita di essere simulato. Anche questo porta a dover utilizzare computer potenti e software di simulazione adeguati.

Per questo progetto, come già citato, è stata usata un'FPGA appartenente alla famiglia delle Spartan 6, uno tra i più nuovi prodotti di XIlinx. Nei prossimi paragrafi faremo riferimento alle caratteristiche generali dei dispositivi di questa azienda, andando più in dettaglio per quanto riguarda questa famiglia.

Nella Tabella 2 sono riportati tutti i dispositivi facenti parte della famiglia, con le relative caratteristiche. Di particolare interesse sono il numero di *slice*, e cioè delle risorse riconfigurabili, di *DSP slices* e di *Block RAM*, fattori fondamentali da considerare quando bisogna scegliere il componente giusto per il proprio progetto.

Tabella 2 - Elenco dei dispositivi della famiglia Spartan 6 e loro caratteristiche

Device	Logic Cells(1)	Configurable Logic Blocks (CLBs)			DSP48A1 Slices(3)	Block RAM Blocks		CMTs(5)	Memory Controller Blocks (Max)	Endpoint Blocks for PCI Express	Maximum GTP Transceivers	Total I/O Banks	Max User I/O
		Slices(2)	Flip-Flops	Max Distributed RAM (Kb)		18 Kb(4)	Max (Kb)						
XC6SLX4	3,840	600	4,800	75	8	12	216	2	0	0	0	4	132
XC6SLX9	9,152	1,430	11,440	90	16	32	576	2	2	0	0	4	200
XC6SLX16	14,579	2,278	18,224	136	32	32	576	2	2	0	0	4	232
XC6SLX25	24,051	3,758	30,064	229	38	52	936	2	2	0	0	4	266
XC6SLX45	43,661	6,822	54,576	401	58	116	2,088	4	2	0	0	4	358
XC6SLX75	74,637	11,662	93,296	692	132	172	3,096	6	4	0	0	6	408
XC6SLX100	101,261	15,822	126,576	976	180	268	4,824	6	4	0	0	6	480
XC6SLX150	147,443	23,038	184,304	1,355	180	268	4,824	6	4	0	0	6	576
XC6SLX25T	24,051	3,758	30,064	229	38	52	936	2	2	1	2	4	250
XC6SLX45T	43,661	6,822	54,576	401	58	116	2,088	4	2	1	4	4	296
XC6SLX75T	74,637	11,662	93,296	692	132	172	3,096	6	4	1	8	6	348
XC6SLX100T	101,261	15,822	126,576	976	180	268	4,824	6	4	1	8	6	498
XC6SLX150T	147,443	23,038	184,304	1,355	180	268	4,824	6	4	1	8	6	540

5.1.1 La logica riconfigurabile

I blocchi logici configurabili (CLB) sono formati da una coppia di *slices*, poste fianco a fianco, come parte di due colonne verticali. Vi sono tre tipi di *slices* nella Spartan 6, ognuna delle quali è composta da quattro *Look-Up-Table* (LUT), otto *flip-flop* e altra logica variabile. Le LUT servono ad avere supporto logico sequenziale e combinatorio *general-purpose*.

Un quarto delle *slices* è composto da SLICEM, che possono essere configurate come LUT a sei ingressi e un'uscita o in coppia come LUT con cinque ingressi identici e due uscite indipendenti. Questa tipologia di *slice* può essere usata anche come RAM distribuita singola a 64 bit o doppia a 32 bit o come *shift register* a 32 o a 16 bit con lunghezza indirizzabile. L'uscita può essere memorizzara in un *flip-flop* interno. Per le operazioni aritmetiche, i segnali si possono propagare su una catena ad alta velocità lungo la colonna delle *slices*.

Un altro quarto delle *slices* è composto da SLICEL, che sono identiche al tipo M, se non per il fatto che non dispongono della funzionalità di *shift register*.

L'altra metà delle *slices* prende il nome di SLICEX che non dispone del bus dati per le operazioni aritmetiche.

5.1.2 Clock Management e Clock Distribution

Ogni Spartan 6 ha fino a 6 *Clock Management Tile* (CMT) in grado di gestire le funzioni avanzate sui segnali di sincronismo. Sono composti da due *digital clock manager* (DCM) e un *phase locked loop* (PLL) che possono essere utilizzati singolarmente o in cascata, a seconda di come si vuole processare il segnale di clock.

Il DCM è una struttura che permette, a partire dal clock di partenza, di avere un segnale sfasato di 0°, 90°, 180° e 270°. Questo blocco può generare un segnale a frequenza doppia dell'originale e di sfasarlo di 180°. Inoltre la frequenza di sincronismo può essere diviso per qualsiasi intero compreso tra 2 e 16, o per N/2, con N intero compreso tra 3 e 15.

Un sintetizzatore di frequenza interna può inoltre generare tutte quelle frequenze ottenibili moltiplicando la frequenza d'ingresso per un numero M, intero compreso tra 2 e 32, e dividendo per D, intero compreso tra 1 e 32.

Ogni FPGA ha un gran numero di linee espressamente progettate per portare il segnale di clock ai vari dispositivi. Queste linee hanno la caratteristica di poter gestire un elevato fanout, hanno basso ritardo di propagazione e bassissimo skew.

Queste linee si dividono in due categorie: quelle globali, 16 in totale, che viaggiano all'interno del dispositivo e possono raggiungere ogni flip-flop, e quelle di input/output che possono prendere o fornire il segnale sui pin di interfacciamento con l'esterno dell'FPGA.

5.1.3 Block RAM

Abbiamo già parlato di come è possibile sfruttare le *slices* per creare dei blocchi di memoria distribuita all'occorrenza. Questa funzionalità è molto utile se l'allocazione di memoria necessaria è piccola. Quando bisogna creare moduli di memoria più capienti è possibile invece utilizzare dei blocchi che implementano esclusivamente la possibilità di essere utilizzati per questo scopo. Questi moduli prendono il nome di *Block* RAM (BRAM). Ogni dispositivo ha tra i 12 e i 268 *dual-port* BRAM, ognuno con capacità di memorizzare 18 kbit. Le due porte della memoria sono completamente indipendenti, e condividono esclusivamente i dati memorizzati.

Ogni accesso, sia in lettura che in scrittura, è controllato dal clock. Un *latch* in uscita fa si che il dato letto sia disponibile al colpo di clock successivo.

La larghezza del dato è completamente modificabile dal progettista. Si può andare da 16K x 1 a 512 x 36 (con un *parity bit* per byte se necessario).

5.1.4 Digital Signal Processing

Nelle FPGA Spartan-6 sono implementati dei blocchi logici chiamati DSP48A1. Questi componenti sono delle *slices* dedicate a svolgere operazioni come moltiplicazioni e accumulazioni. Sono costituiti in modo da essere completamente personalizzabili: ogni *slice* di questo tipo contiene al suo interno un moltiplicatore a 18 bit in complemento a due e un accumulatore a 48 bit, entrambi capaci di operare fino a quasi 400 MHz, sfruttando il meccanismo delle pipeline.

Grazie a questi blocchi logici è possibile eseguire una molteplicità di tipologie di processamento. Di grande interesse per il nostro progetto è la capacità di eseguire operazioni in *floating point*. Grazie agli *ip-core* la complessità nella progettazione di tali strutture si riduce pesantemente, rendendole accessibili anche senza dover avere un elevato *know-how* sull'argomento.

5.1.5 Interfaccia esterna

Il sistema che verrà implementato su FPGA dovrà necessariamente interfacciarsi con il mondo esterno. Per ricevere dati, elaborarli e fornire i risultati. Nelle FPGA Xilinx nessun componente interno è in comunicazione diretta con i pin esterni, ma solo attraverso catene di buffer. Questo per non gravare capacitivamente i nodi interni, e quindi rallentare il sistema, e per rendere l'FPGA adattabile agli standard logici più disparati (LVCMOS, LVTTL, HSTL, SSTL, PCI, LVDS ecc.).

A tal scopo, i pin esterni sono connessi direttamente a degli *Input Output Block* (IOB), che possono porre il pin in ingresso, in uscita o in *tristate*.

5.1.6 Le interconnessioni programmabili

Tutti i CLB e gli IOB devono poter essere elettricamente connessi tra di loro per poter scambiare dati e, quindi, costituire il sistema digitale. A bordo delle FPGA, tutte le connessioni devono essere programmabili e consentire a un qualunque CLB di raggiungere un qualunque altro CLB presente sul dispositivo o un qualunque blocco di IOB. Affinché quanto illustrato venga realizzato è necessario che ciascun CLB e ciascun IOB abbia associata una matrice di connessione (GRM, *General Routing Matrix*). Questa può essere vista, di fatto, come un ulteriore blocco che raccoglie tutte le connessioni del CLB o dell'IOB associato e lo collega con il resto del sistema. Questi blocchi si trovano sui nodi della rete di interconnessione globale.

Uno dei punti deboli delle FPGA è sempre stato il ritardo di propagazione introdotto dalla rete configurabile di interconnessioni dovuto al gran numero di punti di diramazione e, quindi, di capacità equivalenti introdotte sui connettori. Avviene quindi che tanto più è elevato il numero di matrici di interconnessione attraversate da una risorsa di connessione, tanto più questa introdurrà ritardo nella propagazione del segnale. Proprio per sopperire a questo problema, anche in questo caso, come per la gestione dei segnali di clock, si utilizzano strutture molto complesse per la loro ottimizzazione.

Vengono impiegati generalmente quattro tipi di connessioni differenti:

- *Long lines*, che connettono tra di loro le matrici di connessione di 6 in 6;
- *Hex lines*, che connettono tra di loro le matrici di 3 in 3;
- *Double lines*, che connettono tra di loro matrici di 2 in 2;
- *Direct lines*, che connettono tra di loro direttamente ogni CLB con i vicini.

5.2 La programmazione

5.2.1 Design entry

La prima fase che bisogna affrontare quando ci si trova a dover progettare un sistema basato su FPGA è quella definita come design entry. Essa consiste nel "descrivere" al sistema di programmazione il progetto da implementare.

Questa descrizione può avvenire in tre modi diversi: mediante schematico, attraverso linguaggi descrittivi di alto livello o utilizzando diagrammi di stato.

Il primo metodo consiste nel rappresentare tramite schematico la funzionalità del sistema, mediante l'uso di porte logiche e blocchi più complessi attraverso librerie.

L'ultimo, invece, prevede la descrizione del sistema tramite diagrammi di stato, mostrando le connessioni tra i vari stati e le condizioni di transizione tra uno e l'altro.

Il metodo più utilizzato e che consente una maggior efficienza, è, però, quello che si avvale dei linguaggi descrittivi di alto livello. La descrizione di un progetto attraverso le istruzioni di un codice standard ad alto livello fornisce tutti i vantaggi dell'uso di un linguaggio di programmazione strutturato. In particolare, nel caso dei dispositivi logici programmabili il progetto risulta in tal modo adatto a essere facilmente modificato, esportato, gerarchizzato e ne è accelerata la fase di sviluppo, soprattutto nel caso di grandi progetti in cui al team di sviluppo viene garantita un'interfaccia standard tra i diversi progettisti. Inoltre, la descrizione del progetto mediante un linguaggio è completamente indipendente dal dispositivo elettronico prescelto (*technology independent*), in quanto non viene considerato alcun componente reale ma ne viene indica il solo aspetto logico funzionale. Di conseguenza anche la simulazione del progetto descritto risulta totalmente *technology independent*, fattore questo che garantisce l'adattabilità e la possibilità di riutilizzare il progetto alla luce della continua e rapida evoluzione dei dispositivi in cui può essere implementato.

I due principali linguaggi di programmazione ad alto livello per dispositivi logici programmabili sono il VHDL (di cui verrà discusso nel capitolo 5.3) e il *Verilog*.

5.2.2 Simulazione funzionale

Quando si progettano sistemi complessi, è difficile avere certezza che il codice scritto rispecchi esattamente le caratteristiche volute, soprattutto quando si ha a che fare con moduli che funzionano a frequenze di clock differenti. Nasce quindi la necessità di poter simulare il codice in maniera funzionale, senza doverlo necessariamente implementare fisicamente.

Grazie a software preposti, è possibile simulare la struttura creata, fornendo degli stimoli opportuni in ingresso al sistema e valutare le risposte. In questa fase è possibile andare a esaminare l'evoluzione di qualsiasi segnale all'interno del sistema.

Una volta effettuata questa fase di debugging, si può tornare alla fase di *design entry* per correggere eventuali errori, o passare alla fase di sintesi, se il risultato è soddisfacente.

5.2.3 Sintesi

Mentre nelle due fasi precedenti si era ancora svincolati dalla tecnologia, è con questa operazione che si va a legarsi all'hardware.

In questa fase viene infatti realizzato, dal compilatore, il fle di programmazione per configurare il componente che implementa, il progetto, tenendo conto delle caratteristiche del dispositivo.

La sintesi avviene in 4 passi: *Translation*, *Mapping*, *Place & Route* e *Bitstream Generation*.

Nel primo passo, ancora indipendente dalla tecnologia, il compilatore esamina la descrizione del progetto, verificandone la coerenza dal punto di vista logico, di connessione dei segnali, dei pin e di altri fattori. Inoltre viene prodotta una descrizione logica del progetto, su un unico livello gerarchico.

Nel *Mapping*, sulla base di quanto elaborato al passo precedente, il software ricava una descrizione fisica del progetto, in base ai blocchi logici disponibili nel dispositivo. Sempre in questa fase, vengono accorpate o eliminate tutte quelle parti logiche che risultano ridondanti o inutili (segnali non connessi in uscita o mai usati).

Nel passo successivo, il *Place & Route*, ai blocchi precedentemente identificati vengono associati i componenti reali del dispositivo, andando a realizzare tutte le connessioni necessarie, rispettando i vincoli di *timing*. Gli algoritmi che realizzano questa fase sono particolarmente complessi allo scopo di ottimizzare le risorse impiegate e minimizzare i ritardi di propagazione, nell'ottica di minimizzare lo spazio occupato e massimizzare la frequenza di funzionamento. Il progettista ha la facoltà di decidere quale dei due aspetti privilegiare, oltre a poter imporre vincoli di *timing* (su certi cammini particolarmente importanti), nonché decidere dove localizzare certe risorse (tipica è l'associazione dei segnali di I/O a precisi pin del package).

Nell'ultima fase, viene infine creato un file, detto *bitstream*, che rispecchia il risultato della fase precedente. Con questo file è possibile configurare il dispositivo.

5.2.4 Simulazione di timing

Analogamente alla simulazione funzionale, in questa fase si verifica che il sistema svolga, esattamente quello per cui è stato progettato. Questa simulazione, è però più completa rispetto alla prima, perché tiene conto dei ritardi effettivi sui segnali, introdotti dalle connessioni e dai blocchi di elaborazione, potendo conoscere l'effettiva disposizione dei componenti. Questa possibilità ulteriore di simulazione si rende molto utile quando si vuole spingere le prestazioni del dispositivo al massimo.

5.2.5 Downloading

Nell'ultima fase si procede a programmare fisicamente il dispositivo, caricando il file bitstream.

A seconda del dispositivo, il download avviene attraverso pin preposti con modalità che possono variare. Il programma viene quindi memorizzato in alcuni casi in una memoria interna all'FPGA, in altri in una esterna.

5.3 Il linguaggio VHDL

Il linguaggio descrittivo principale è il VHDL, acronimo di Vhsic (*Very high speed integrated circuit*) *Hardware Description Language*, linguaggio di descrizione hardware per circuiti integrati ad alta velocità. Nato agli inizi degli anni '80 nell'ambito del programma VHSIC (Very High Speed Integrated Circuit) del Dipartimento della Difesa statunitense, con lo scopo sviluppare nuove generazioni di circuiti integrati, il VHDL ha il duplice obiettivo di essere un linguaggio con istruzioni orientate alla descrizione circuitale e di essere uno standard di interfacciamento tra vari progettisti. In un ventennio ha subito diverse fasi di standardizzazione, segno questo dell'effettiva efficacia del suo impiego, divenendo per esempio standard IEEE dal 1987.

La caratteristica principale del linguaggio VHDL, diversamente dai classici linguaggi sequenziali (come C e Java) è la caratteristica di eseguire le operazioni parallelamente. Due comandi

successivi, vengono quindi eseguiti contemporaneamente, ed è necessario porre condizioni sulle transizioni del clock per sincronizzare le operazioni, e fare in modo che vengano eseguite quando i segnali di ingresso riportano effettivamente i valori corretti. Se necessario è anche possibile la creazione di istruzioni sequenziali, grazie i processi, che si attivano sotto particolari stimoli.

La programmazione VHDL è quindi molto diversa dalla programmazione sequenziale e richiede una grande attenzione alle problematiche di temporizzazione. Anche per questo motivo, la simulazione funzionale per la verifica del codice è di grande aiuto ed è impensabile la programmazione di sistemi complessi senza una verifica in tal senso.

5.4 Il software

Un ultimo accenno viene dato ai software usati per la realizzazione della parte di progetto su FPGA. Per ogni fase di lavoro sono stati utilizzati software altamente specializzati:

- Xilinx ISE 12.1, per la parte di Editing, Sintesi, Mapping e Place & Route;
- Mentor Graphics Modelsim 6.6b SE, per la parte di simulazione funzionale;
- Xilinx ChipScope Pro Analizer, per la verifica del corretto funzionamento del sistema, dopo l'implementazione reale;
- MatLab 7.8.0, per la creazione dei vettori di stimolo e la verifica delle simulazioni più complesse.

6. Altri componenti

6.1 Il microcontrollore (LM3S9B90)

Il microcontrollore scelto per il progetto è un Stellaris LM3S9B90 di Texas Instrument.

La scelta di questo dispositivo è dovuta alle numerose caratteristiche richieste al microcontrollore, tutte soddisfatte da questo componente.

Le caratteristiche principali sono le seguenti:

- *core* ARM Cortex-M3
- frequenza operativa massima di 80 MHz
- 256 kB di memoria flash
- 96 kB di SRAM

La grande potenzialità di questo microcontrollore risiede, però, nella gran quantità di periferiche di comunicazione implementati, tra cui:

- 1 Ethernet 10/100
- 2 CAN
- 1 USB
- 2 SSI/SPI
- 2 I2C
- 1 I2S
- 3 UART

Infine, tralasciando altre caratteristiche di minor importanza per il progetto, a completare l'elenco ci sono le 60 porte di interfacciamento *input/output general purpose*.

6.1.1 ARM Cortex-M3

La prima caratteristica da soddisfare è l'elevata potenza di calcolo richiesta: il processore interno deve infatti eseguire complessi calcoli in floating point a 64 bit in tempi molto brevi, in modo che il sistema non risulti lento agli occhi dell'utente finale.

Il microcontrollore possiede un processore ARM a 32 bit della famiglia Cortex, in grado di arrivare a circa 1.25 MIPS/MHz. Può gestire fino a 47 interrupt, suddividedoli in 8 livelli di differente priorità. Tra le varie caratristiche, fondamentale è quella di poter eseguire moltiplicazioni a 32bit in un singolo colpo di clock. Negli ultimi anni i processori ARM si sono contraddistiniti per le performance di elaborazione e il ridotto consumo energetico, tanto da essere utilizzati nei più moderni smartphone.

In Figura 6.1 è mostrato lo schema funzionale del processore.

Infine ricordiamo che la programmazione avviene direttamente in C grazie alle librerie fornite dal produttore. Per la compilazione del codice ci si affida al development tools di Keil.

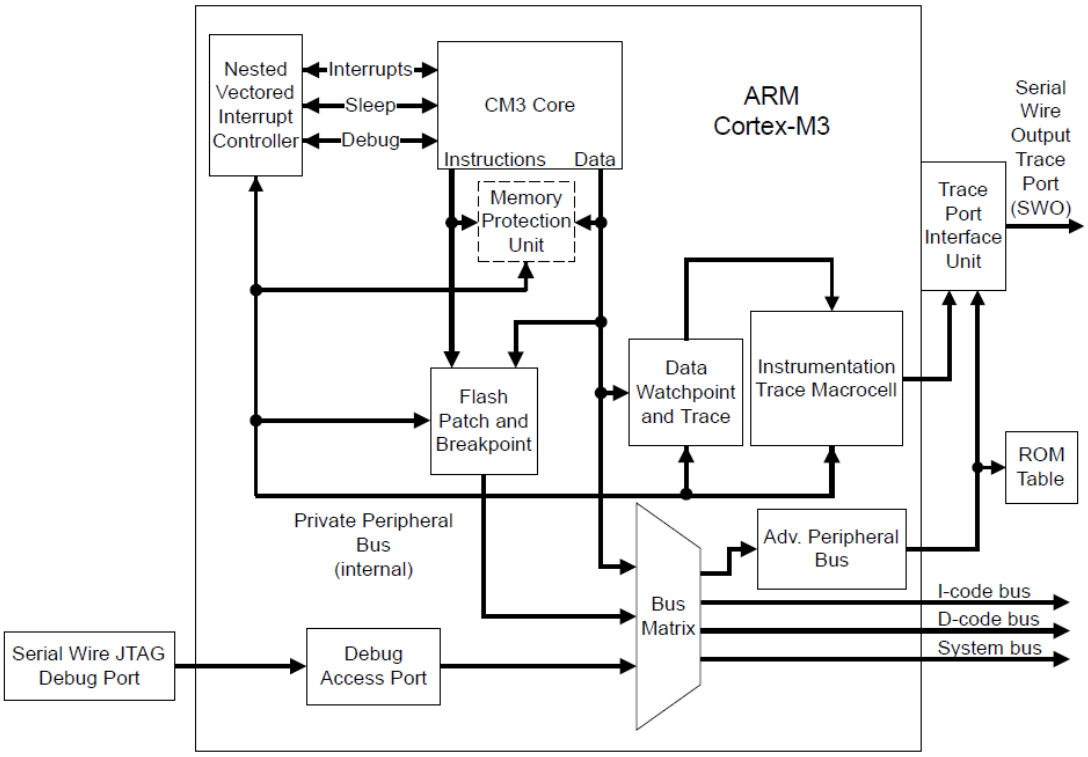

Figura 6.1 - Il microcontrollore - Schema funzionale ARM

6.1.2 Synchronous Serial Interface (SSI/SPI)

La comunicazione dei coefficienti e dei parametri all'FPGA avviene tramite protocollo SPI (vedi Appendice B.II. SPI (Serial Peripheral Interface) per le caratteristiche di questo standard di comunicazione).

Il microcontrollore implementa 2 di questi moduli, molto versatili e personalizzabili. I moduli possono operare sia in master mode che come slave. La lunghezza del dato può essere da 4 a 16 bit. La velocità di trasmissione può essere programmata, fino ad arrivare a un massimo di 4MHz.

In Figura 6.2 è possibile osservare lo schema funzionale della periferica.

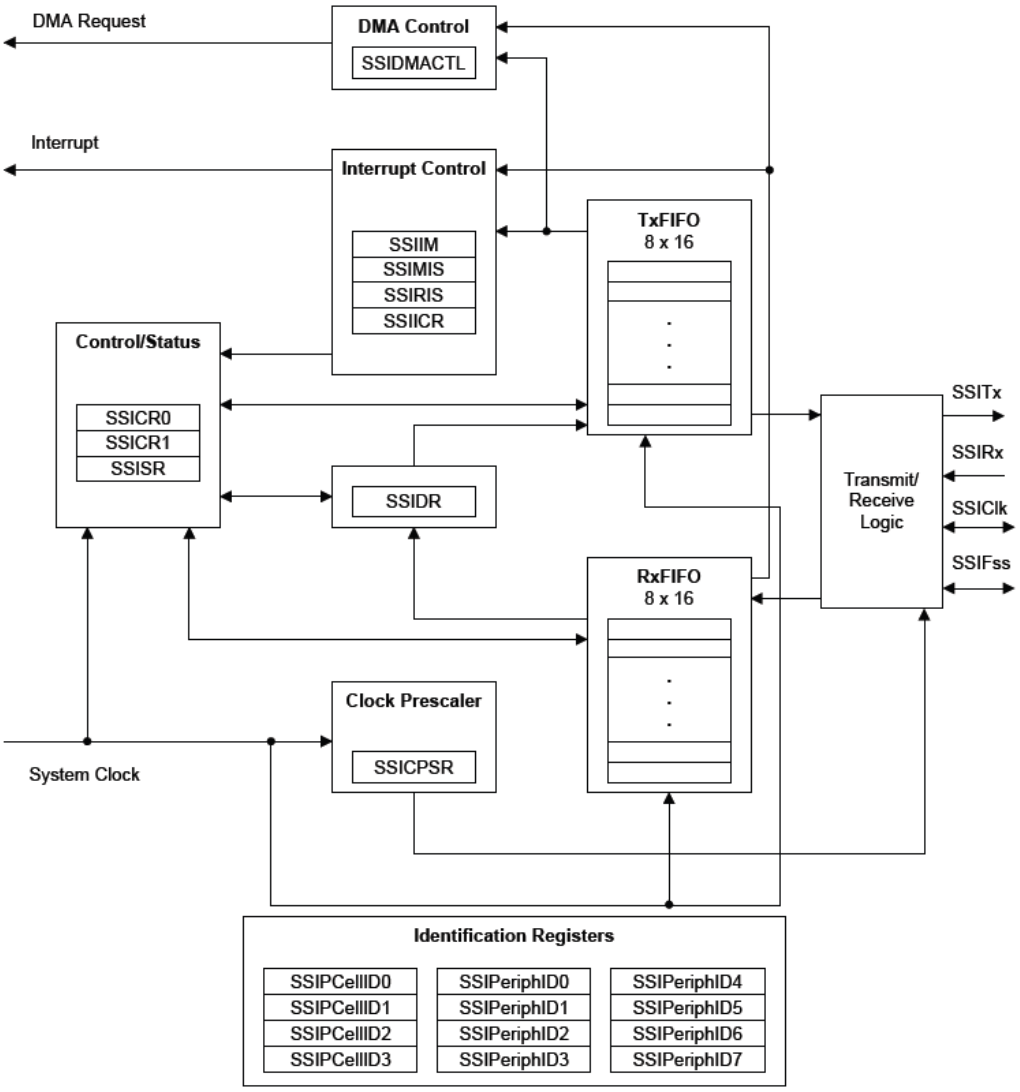

Figura 6.2 - Schema funzionale modulo SSI

6.1.3 Inter-Integrated Circuit Interface (I2C)

La lettura dei tasti presenti sul frontalino avviene tramite un controller, il MAX7347 di Maxim. La comunicazione con tale dispositivo è su bus I2C (per approfondire questo standard si veda il capitolo 0).

Il microcontrollore possiede un modulo preposto a questo standard, capace di funzionare sia in modalità master che in modalità slave. Sono supportate le frequenze di trasmissione a 100 kbit/s e 400 kbit/s.

Il modulo offre la possibilità di generare degli *interrupt*: per esempio al termine di ogni comunicazione, in caso di modalità master, o, in modalità slave, quando viene riconosciuto il proprio indirizzo sul bus dati; questo permette di compiere altre operazioni durante le trasmissioni.

In Figura 6.3 si può notare lo schema funzionale del modulo.

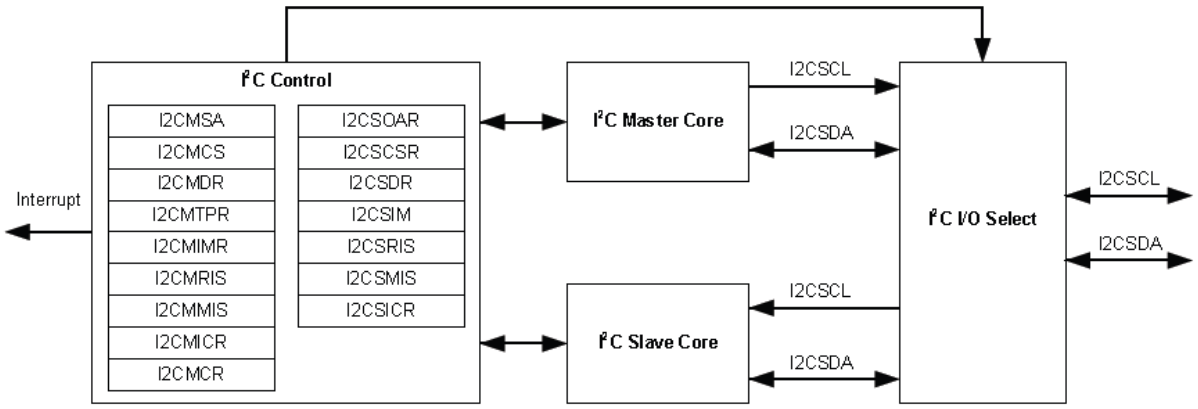

Figura 6.3 - Schema funzionale modulo I2C

6.1.4 Universal Serial Bus Controller (USB)

Oltre a poter controllare tutte le funzioni del DSP tramite la comoda interfaccia locale, il progetto si propone di creare un'interfaccia grafica su pc. La comunicazione, per questa prima implementazione avviene tramite porta USB. Il microcontrollore supporta nativamente anche questo tipo di protocollo.

Il controller USB è in grado di arrivare alla velocità di 12 Mbps e supporta tutte le operazioni definita dallo standard USB 2.0. In Figura 6.4 è mostrato lo schema funzionale della periferica.

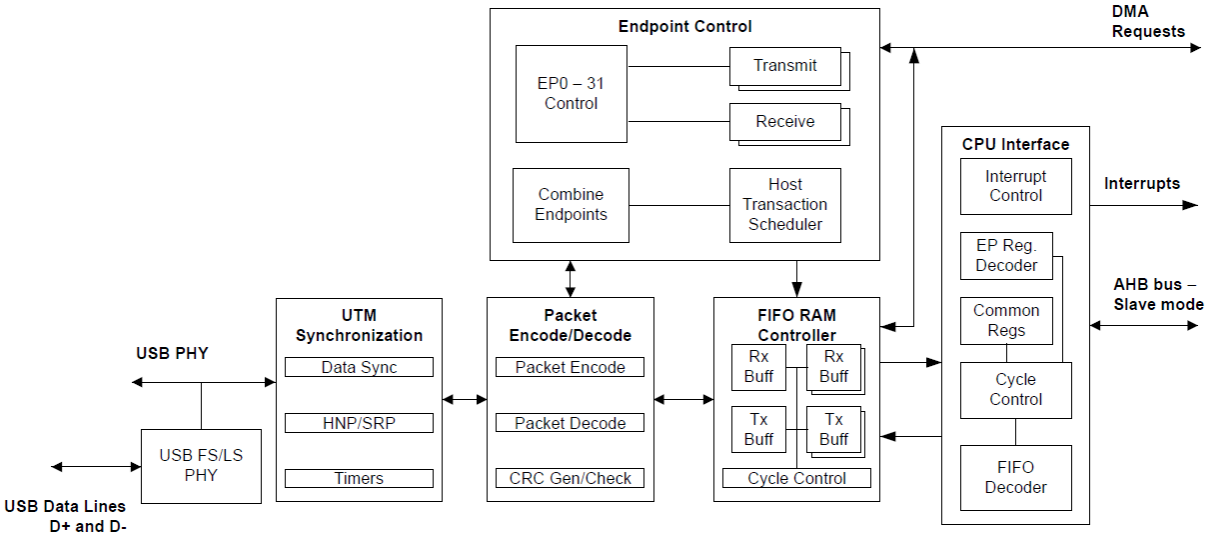

Figura 6.4 - Schema funzionale USB controller

6.2 Il sample rate converter (SRC4184)

Come spiegato nel capitolo 4.2, il *sample rate converter* è un elemento fondamentale quando si vuole ricevere uno *stream* audio in formato AES/EBU e non si vuole degradare significativamente il rapporto segnale rumore.

L'SRC4184 di Texas Instruments, oltre al costo limitato, ha varie caratteristiche che lo rendono il componente adatto a questo tipo di progetto.

In primo luogo la possibilità di comunicazione tramite protocollo I2S. Essendo questo protocollo già usato per altri scopi all'interno del progetto, il suo utilizzo anche allo scopo di comunicare con l'SRC ha comportato solo l'adattamento a questo specifico uso, eliminando le fasi di creazione di una periferica ad hoc.

Inoltre il dispositivo è in grado di calcolare automaticamente il rapporto tra la frequenza di campionamento del segnale in ingresso e quella in uscita; il suo valore può essere compreso tra 16:1 e 1:16, rientrando ampiamente nelle specifiche.

Altro aspetto rilevante, consente un *dynamic range* di 128 dB.

Vi sono poi una serie di altre funzioni minori per le quali si rimanda al *datasheet*, tutte però programmabili tramite comunicazione SPI, in modo semplice e intuitivo.

6.3 ADC (PCM4220) e DAC (PCM1798)

Il PCM4220 di Texas Instruments è un convertitore analogico/digitale di tipo *Sigma Delta* a due canali. Ha una dinamica d'ingresso massima di 6V differenziale e un *dynamic range* tipico di 123 dB.

Il PCM1798, invece, è un convertitore digitale/analogico, con uscita in corrente. Ha dinamica di 4mA picco picco e *dynamic range* tipico di 123 dB, il che lo rende il componente più critico nella catena analogica.

In entrambi i componenti, la trasmissione dei dati campionati può avvenire tramite protocollo I2S.

7. Il filtro IIR del second'ordine

7.1 Il filtro passa tutto del second'ordine

Il filtro passatutto è un blocco di elaborazione computazionalmente efficiente che può essere utile in molte applicazioni di signal processing. Generalmente, per prima cosa, il progettista seleziona la funzione di trasferimento adatta alle caratteristiche di selettività in frequenza e di risposta in fase. Successivamente, viene scelta la struttura con cui implementare tale funzione di trasferimento tra le numerose possibilità ponendo attenzione alle specifiche di rumore di tipo *round off* e di sensitività dei coefficienti. Per definizioni e proprietà della trasformata Z e dei filtri si faccia riferimento all'appendice.

7.1.1 Definizioni e proprietà

La definizione di filtro passa tutto è molto semplice: il modulo della funzione di trasferimento $A(e^{j\omega})$ è uguale a 1 per qualunque ω.

$$\left|A(e^{j\omega})\right|^2 = 1 \;\forall\; \omega \tag{20}$$

I poli della funzione di trasferimento hanno la caratteristica di apparire a coppie con gli zeri e in modo da essere i reciproci. La funzione di trasferimento può quindi essere scritta nel modo seguente:

$$A(z) = e^{j\theta} \prod_{i=1}^{N} \frac{\gamma_i^* - z^{-1}}{1 - \gamma_i z^{-1}} \tag{21}$$

Per ragioni di stabilità $|\gamma_i| < 1$ in modo che tutti poli siano interni al cerchio di raggio unitario.

Se supponiamo che A(z) sia reale, essa può essere espressa nel seguente modo

$$A(z) = \frac{z^{-M} D(z^{-1})}{D(z)} \tag{22}$$

Dove M rappresenta l'ordine del filtro. Infatti il polinomio al numeratore è ottenuto dal polinomio al denominatore semplicemente invertendo l'ordine dei coefficienti.

Questa caratteristica di simmetria permette, attraverso opportuni conti matematici, di risalire a un filtro computazionalmente efficiente col minor numero di moltiplicazioni possibile. Prendiamo ad esempio

$$A(z) = \frac{Y(z)}{U(z)} = \frac{a_2 + a_1 z^{-1} + z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{23}$$

che corrisponde all'equazione nel dominio del tempo

$$y(n) = a_2[u(n) - y(n-2)] + a_1[u(n-1) - y(n-1)] + u(n-2) \tag{24}$$

Con il raccoglimento di termini eseguito nella (23) sono necessarie solo due moltiplicazioni per implementare il filtro.

Con procedimenti simili un filtro passatutto arbitrario di ordine M può essere sintetizzato con solo M moltiplicazioni, contro le 2M+1 necessarie nella realizzazione della forma diretta I.

Un'interessante struttura per la realizzazione di filtri passatutto è il *lattice filter* di Gray e Merkel; questa struttura è scritta in forma ricorsiva come

$$z^{-1}A_{m-1}(z) = \frac{A_m(z) - k_m}{1 - k_m A_m(z)} \qquad (25)$$

Con $m = M, M-1, \ldots, 1$ e $k_m = A_m(\infty)$.

Ora, se $A_m(z)$ è una funzione passatutto stabile di ordine M, può essere verificato che $|k_m| < 1$ e che $A_{m-1}(z)$ è anch'essa una funzione passatutto stabile di ordine M-1.

L'interpretazione strutturale di questa forma è quella mostrata in Figura 7.1.

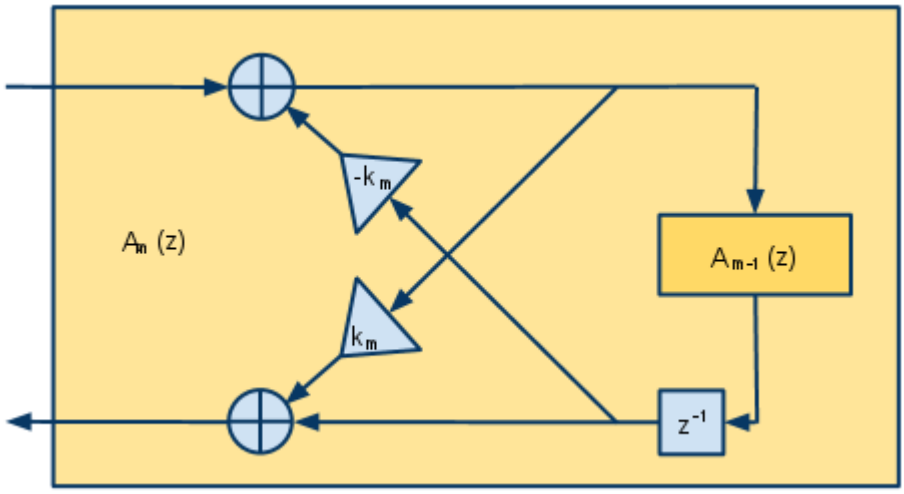

Figura 7.1 - *lattice filter* di Gray e Markel

Reiterando il procedimento, si arriva alla realizzazione del filtro "lattice" di ordine M, come mostrato Figura 7.2, mettendone M elementari in cascata.

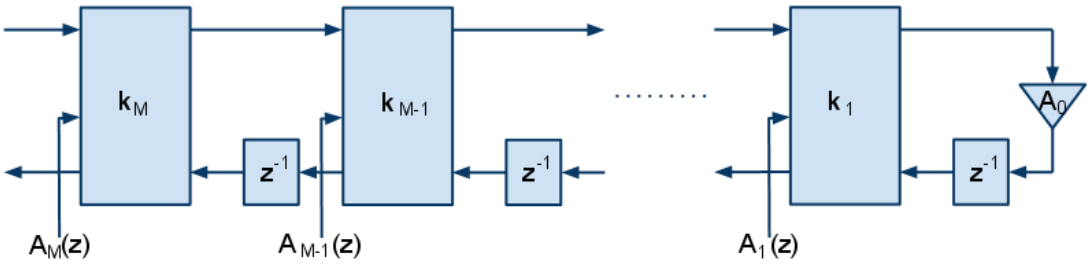

Figura 7.2 - Implementazione attraverso la cascata di *lattice filter* di una funzione passatutto di ordine M

Precedentemente, abbiamo già citato che il vantaggio nell'utilizzo di una struttura del genere, è quello di necessitare di un solo moltiplicatore per ordine di complessità del filtro: con la struttura descritta in Figura 7.1 però questo non si verifica. In realtà essa è equivalente a quella descritta in Figura 7.3 che realizza esattamente questo obiettivo.

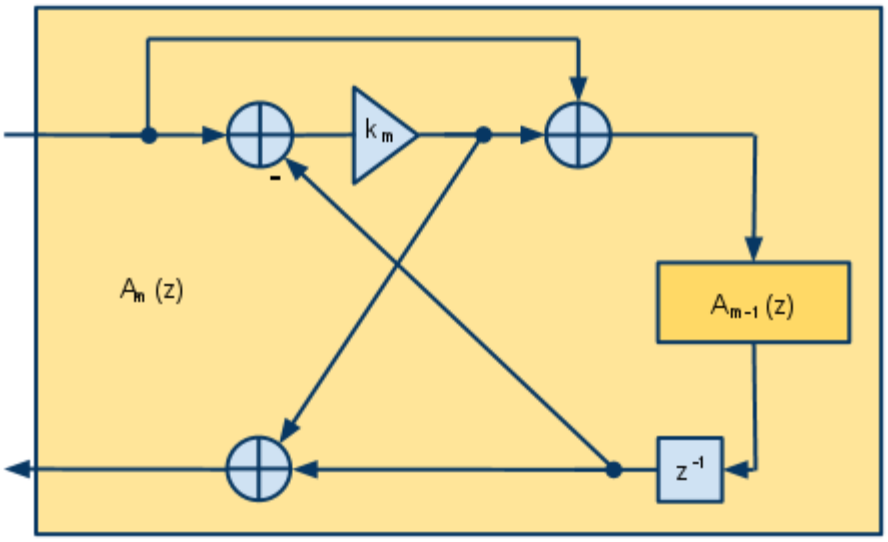

Figura 7.3 - Struttura *lattice* a singolo moltiplicatore

7.2 Il filtro Notch digitale

Una prima applicazione, in cui il filtro passatutto può essere utilizzato, per la realizzazione di funzioni di trasferimento più complesse, è il filtro *notch*. Questo tipo di filtraggio serve a eliminare, dal segnale in elaborazione, la componente a una determinata frequenza.

La più semplice implementazione di un filtro notch digitale è quella mostrata in Figura 7.4 e realizza la seguente funzione di trasferimento

$$G(z) = \frac{1}{2}[1 + A(z)] \tag{26}$$

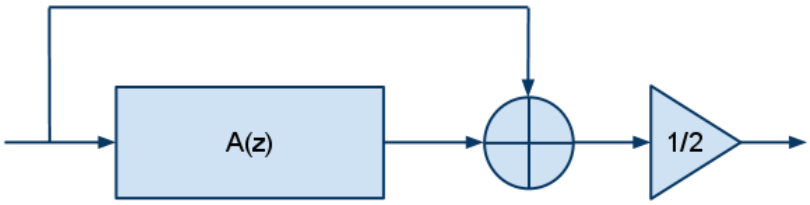

Figura 7.4 - Filtro notch del primo ordine

La funzione all-pass scelta è del second'ordine, in modo che lo sfasamento introdotto da $A(e^{j\omega})$ con ω che va da 0 a π sia -2π radianti. Ne consegue che

$$\left|G(e^{j0})\right| = \left|G(e^{j\pi})\right| = 1 \tag{27}$$

$$\left|G(e^{j\omega_0})\right| = 0 \tag{28}$$

dove ω_0 è quella frequenza angolare per cui il filtro passatutto introduce uno sfasamento di π radianti sul segnale.

Per realizzare il filtro passatutto, si usa la struttura schematizzata in Figura 7.5, che altro non è che la cascata di due filtri passatutto a singolo moltiplicatore. La funzione di trasferimento di implementata è la seguente

$$A(z) = \frac{k_2 + k_1(1 + k_2)z^{-1} + z^{-2}}{1 + k_1(1 + k_2)z^{-1} + k_2 z^{-2}} \tag{29}$$

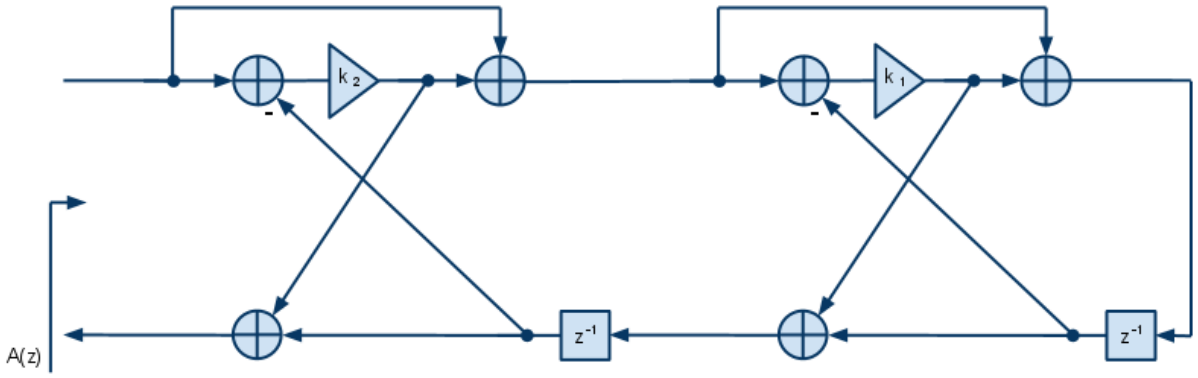

Figura 7.5 - All-pass filter del second'ordine

Il vantaggio maggiore nell'utilizzo di questa struttura è che permette di modificare in modo indipendente la frequenza centrale ω_0 e la banda di attenuazione a meno 3 dB (Ω).

Infatti si ha che

$$k_1 = -cos\omega_0 \tag{30}$$

$$k_2 = \frac{1 - \tan(\Omega/2)}{1 + \tan(\Omega/2)} \tag{31}$$

il che rende molto leggera l'operazione di calcolo dei coefficienti al momento dell'implementazione reale del sistema.

In Figura 7.6 e in Figura 7.7 sono riportati i grafici (forniti da Matlab) della simulazione del filtro, al variare dei coefficienti. In particolare, nella prima figura, è stato simulato un notch alla frequenza di 10 KHz, variando il valore del coefficiente k_2 in modo da modificare la larghezza della banda a -3 dB, portandola prima a 1 kHz, poi a 2 kHz e infine a 3 kHz. Nella seconda figura, invece, è riportata la simulazione di un filtro con larghezza di banda costante a 3 KHz, ma variazione del coefficiente k_1, in modo da centrare il filtro a 5, 10, 15 e 20 KHz.

Nella seconda figura possiamo anche notare un grosso problema nell'uso dei filtri digitali, ricavati dai cugini analogici attraverso la trasformazione bilineare: ad alta frequenza (e cioè nell'intorno di π) il comportamento del filtro digitale si discosta dall'ideale analogico.

L'unità di misura utilizzata sull'asse delle ascisse è l'Hertz, al posto dei più corretti radianti visto che la correlazione dipende dalla frequenza di campionamento. La scelta, d'ora in avanti di adoperare già questa unità di misura è fatta in un'ottica di implementazione nel sistema che è oggetto di questa tesi. Infatti la frequenza di campionamento utilizzata è 48 KHz (uno standard nell'audio professionale). Si è voluto quindi contestualizzare fin da subito le simulazioni.

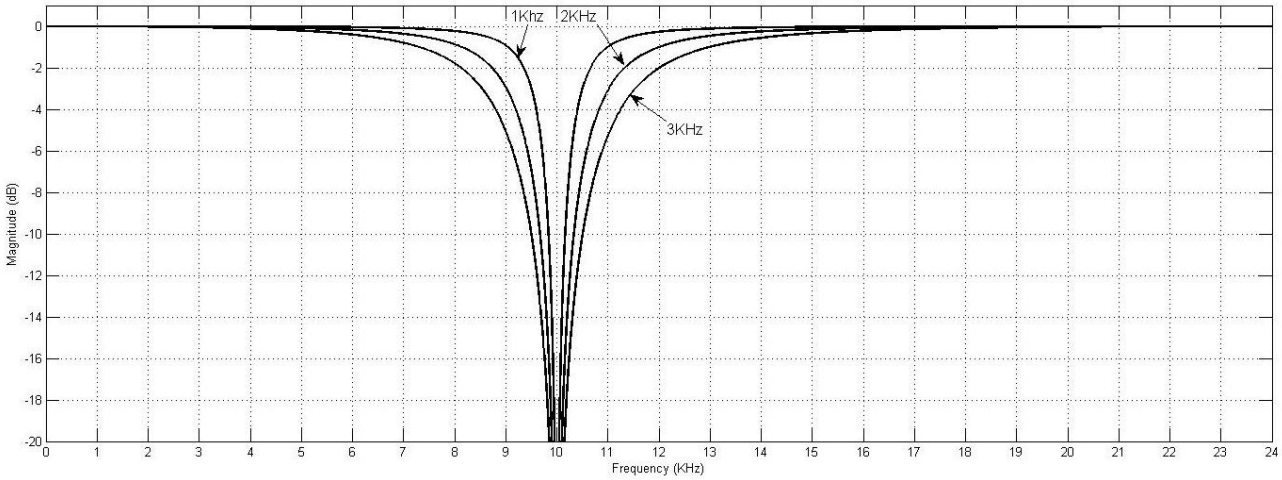

Figura 7.6 - Simulazione filtro notch, variazione della banda a -3 dB

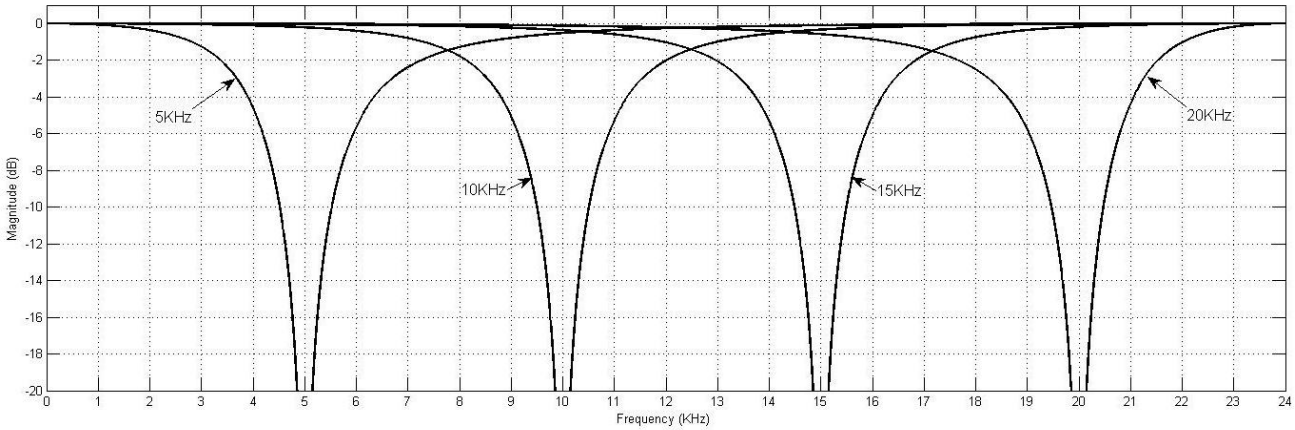

Figura 7.7 - Simulazione filtro notch, variazione del centro banda

7.3 Filtri doppiamente complementari

Due filtri sono detti complementari se la banda passante del primo coincide con la banda bloccata dell'altro.

Detti G e H le funzioni di trasferimento dei due filtri, possono verificarsi due situazioni di complementarità:

$$\left|G\left(e^{j\omega}\right) + H(e^{j\omega})\right| = 1 \qquad \forall\omega \tag{32}$$

o

$$\left|G\left(e^{j\omega}\right)\right|^2 + \left|H\left(e^{j\omega}\right)\right|^2 = 1 \qquad \forall\omega \tag{33}$$

Quando entrambe le condizioni sono verificate si parla di doppia complementarità e quindi si ha che

$$\left|G\left(e^{j\omega}\right) + H(e^{j\omega})\right| = \left|G\left(e^{j\omega}\right)\right|^2 + \left|H\left(e^{j\omega}\right)\right|^2 = 1 \qquad \forall\omega \tag{34}$$

Una situazione del genere comporta che i due filtri siano sempre in quadratura della fase. Inoltre, vedendo $G\left(e^{j\omega}\right)$ e $H\left(e^{j\omega}\right)$ come fasori nel piano complesso, deve succedere che $[G\left(e^{j\omega}\right) + H(e^{j\omega})]$ e $[G\left(e^{j\omega}\right) - H(e^{j\omega})]$ debbano avere lo stesso modulo unitario.

Analiticamente succede quindi che

$$G(z) + H(z) = A_1(z) \tag{35}$$

$$G(z) - H(z) = A_2(z) \tag{36}$$

dove $A_1(z)$ e $A_2(z)$ sono delle funzioni passatutto. Esplicitando G e H si trova quindi che

$$G(z) = \frac{1}{2}[A_1(z) + A_2(z)] \tag{37}$$

$$H(z) = \frac{1}{2}[A_1(z) - A_2(z)] \tag{38}$$

la cui l'implementazione strutturale è mostrata in Figura 7.8

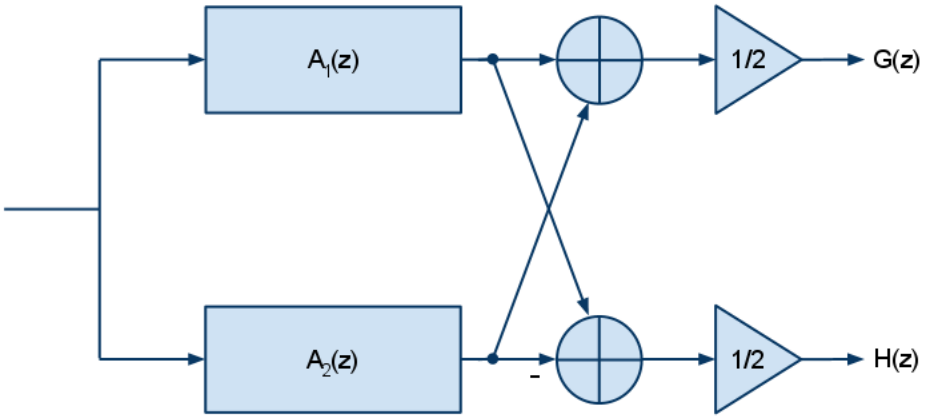

Figura 7.8 - Struttura del filtro doppiamente complementare come composizione di due all-pass filter

7.4 Filtri accordabili

Grazie alla struttura vista nel paragrafo precedente, si possono ricavare, grazie a piccole modifiche, dei filtri molto interessanti.

Immaginiamo di prendere

$$A_1(z) = 1 \tag{39}$$

$$A_2(z) = \frac{K_2 + K_1(1+K_2)z^{-1} + z^{-2}}{1 + K_1(1+K_2)z^{-1} + K_2 z^{-2}} \tag{40}$$

Possiamo immediatamente riconoscere $A_2(z)$ come la funzione di trasferimento dell'*all-pass filter* di Figura 7.5. Applicando queste due funzioni passatutto allo schema in Figura 7.8, per quanto concerne G(z), esso risulta equivalente a quello mostrato in Figura 7.4.

Grazie alla definizione di doppia complementarità, H(z) risulta quindi essere un filtro passabanda con frequenza centrale identica a quella del filtro notch.

Prendiamo ora in considerazione la funzione di trasferimento ottenuta come combinazione delle due, modificata attraverso un coefficiente K

$$F(z) = G(z) + KH(z) \tag{41}$$

ottenuta dal sistema schematizzato in Figura 7.9

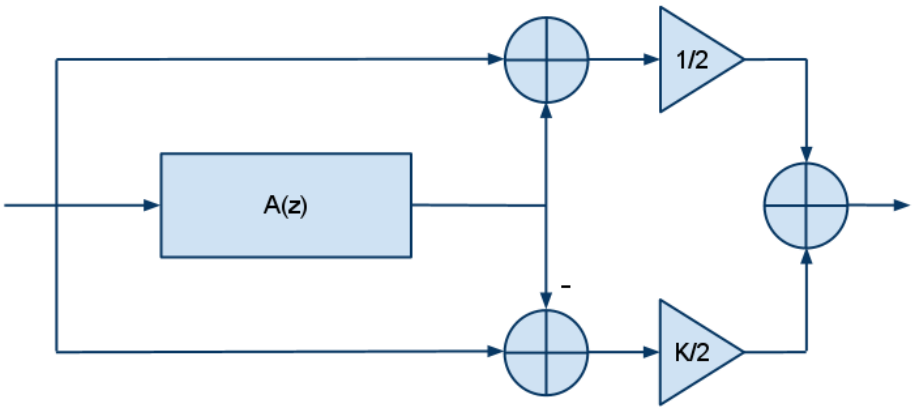

Figura 7.9 - Struttura del filtro di equalizzazione accordabile

Si può dimostrare che

$$\left|F\left(e^{j\omega}\right)\right|^2 = \left|G\left(e^{j\omega}\right)\right|^2 + K^2\left|H\left(e^{j\omega}\right)\right|^2 \tag{42}$$

Ricordandoci che H è un filtro passabanda doppiamente complementare con G, ne consegue che la funzione di trasferimento F così ottenuta rappresenta un *peak filter*. In campo audio questo tipo

di filtro è molto utilizzato per eseguire l'equalizzazione del segnale. Il grande vantaggio nel realizzare questo filtro, eseguendo tutti i passaggi che ci hanno portato alla sua formulazione, è che ora abbiamo un filtro i cui parametri possono essere modificati indipendentemente l'un dall'altro.

Alle due precedenti condizioni su k_1 e k_2 si aggiunge ora la terza su K

$$K = F(e^{j\omega}) \tag{43}$$

In Figura 7.10 viene riportata la simulazione Matlab del filtro al variare del coefficiente K, mantenendo gli altri due coefficienti costanti.

Nelle due figure successive viene invece mantenuto costante il gadagno, e fatti variare gli altri due parametri.

Rimane solo un problma da affrontare: questo filtro, infatti, con il calcolo dei coefficienti, così come mostrato nelle equazioni (29), (39) e (42), è perfetto quando dev'essere implementato in un equalizzatore parametrico, in quanto rende indipendenti il guadagno, la frequenza centrale e il fattore di qualità; necessita, però, di alcune modifiche quando serve a realizzare un equalizzatore grafico. Infatti dalla Figura 7.10 si può notare come la forma del filtro, per guadagni superiori a 1, si allarghi sempre più, andando a modificare fortemente tutto il range di frequenze interessate.

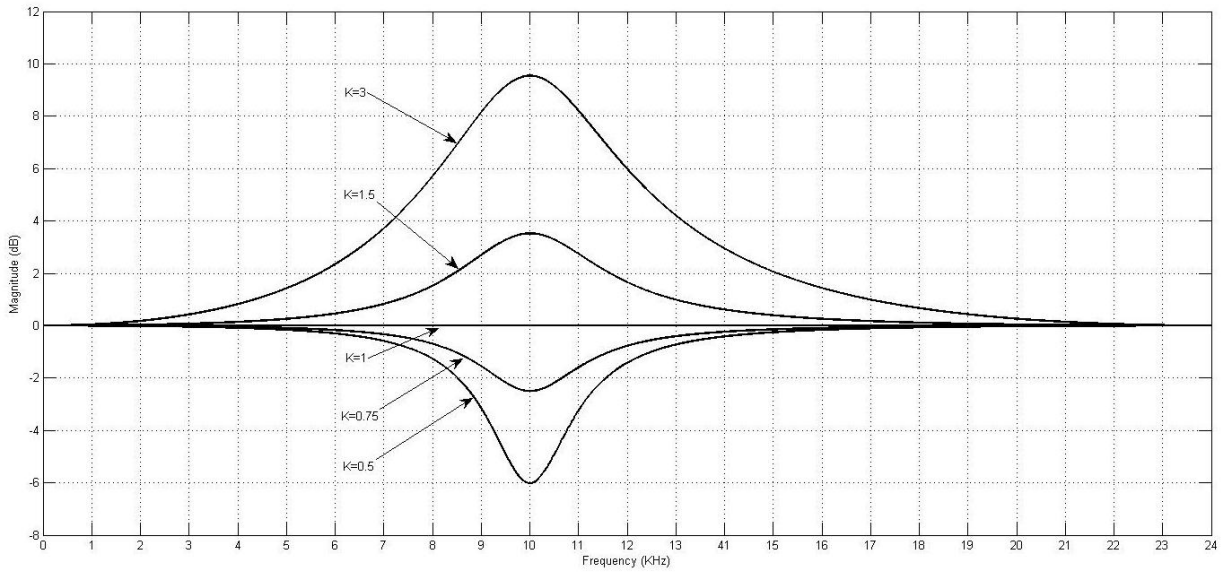

Figura 7.10 - Simulazione filtro peak al variare di K

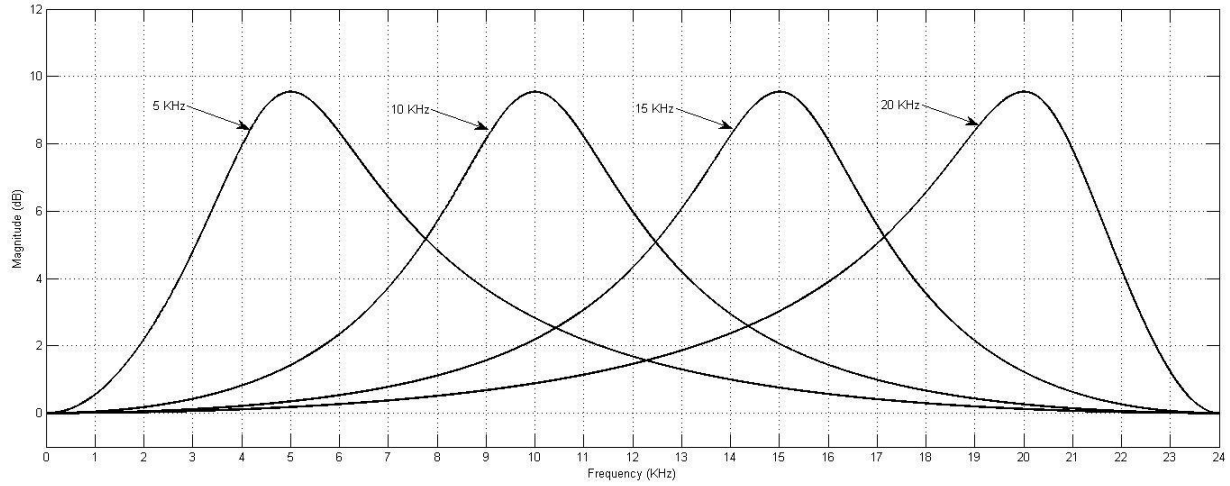

Figura 7.11 - Simulazione filtro peak al variare del centro banda

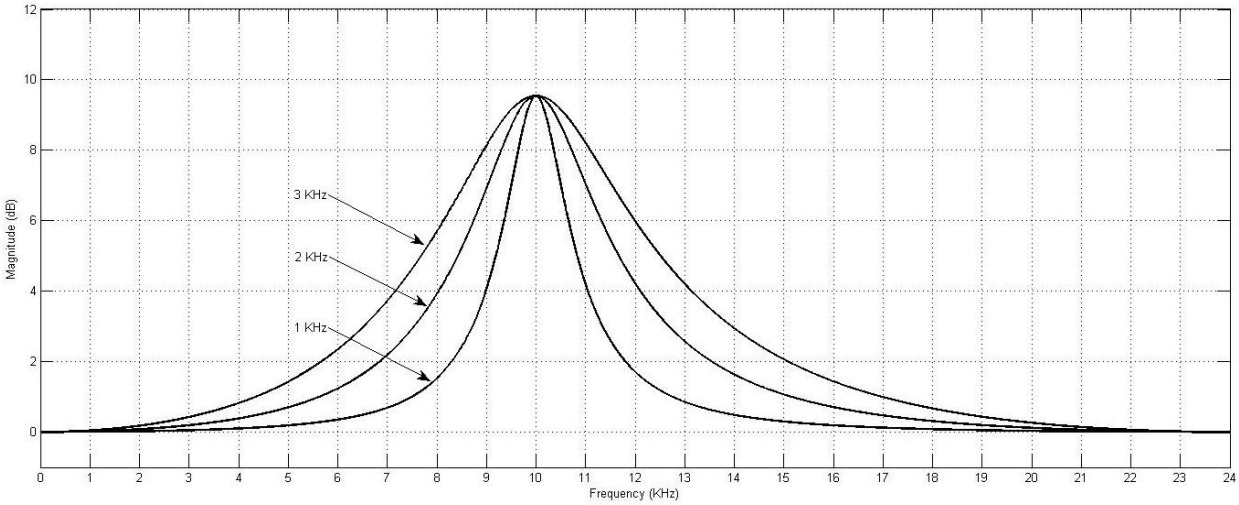

Figura 7.12 - Simulazione filtro peak al variare della banda a -3 dB

È per questo motivo che il calcolo dei coefficienti viene leggermente modificato quando si ha a che fare con questo tipo di equalizzatore, secondo la seguente formula

$$\begin{cases} k_2 = \dfrac{K - \tan(\Omega/2)}{K + \tan(\Omega/2)} & se\ K > 1 \\ k_2 = \dfrac{1 - \tan(\Omega/2)}{1 + \tan(\Omega/2)} & se\ K \leq 1 \end{cases} \tag{44}$$

Si può pensare che gli sforzi fatti per ottenere un filtro che abbia i tre parametri indipendenti siano andati persi. Questo non avviene quando si ha a che fare con un equalizzatore parametrico, l'utente finale può modificare tutti i parametri a piacimento, e si mantiene quindi il vantaggio dell'indipendenza. Nel caso in cui invece si usi l'equalizzatore grafico, il coefficiente k_2 deve essere ricalcolato a ogni modifica del relativo filtro. L'impatto sulle prestazioni del sistema è però molto limitato in quanto il microcontrollore, che è l'elemento preposto a svolgere tali calcoli, ha prestazioni computazionali parecchio elevate.

Il vantaggio della modifica può essere considerato visivamente in Figura 7.13, attraverso la comparazione tra il primo metodo di calcolo a fattore di qualità costante, e il secondo, a larghezza costante.

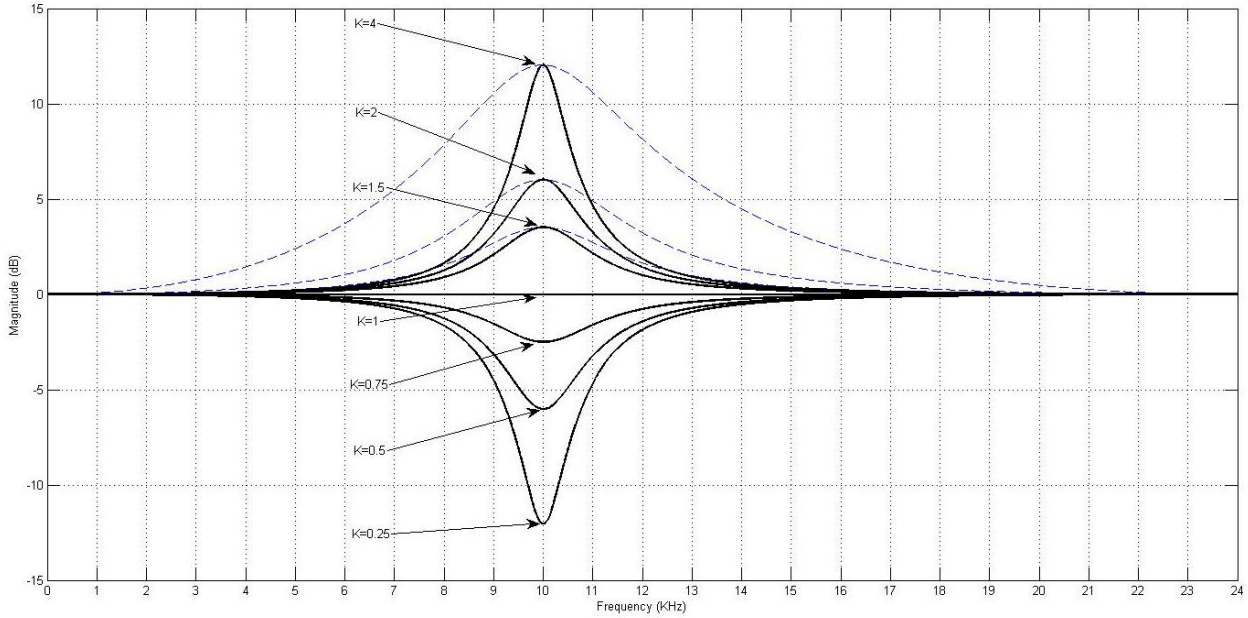

Figura 7.13 - Calcolo secondo coefficiente con algoritmo alternativo

7.5 Algoritmo di compensazione

In questo paragrafo andremo ad analizzare, attraverso la simulazione Matlab, l'implementazione di un equalizzatore grafico che sfrutti il filtro studiato nei capitoli precedenti. L'equalizzatore grafico è piuttosto comune negli apparecchi audio di ogni tipo. A seconda dell'ambiente di utilizzo, possiede un certo numero di bande prefissate, a ognuna delle quali è associato un filtro regolabile in ampiezza. Il numero di bande dipende dall'applicazione prescelta. Si va dalle tre bande (alti, medi e bassi) nei modelli di minor qualità, alle 31 in quelli professionali. Nel nostro caso ci si propone di creare un prodotto commerciale, quindi il target è di creare un equalizzatore a 31 punti di intervento, distanziati da 1/3 di ottava.

Due frequenze distanziate da un'ottava, sono definite come due frequenze l'una il doppio dell'altra. Dire che l'equalizzatore ha 31 punti d'intervento a 1/3 di ottava significa dire che vi sono 3 filtri per ogni ottava, e quindi per ogni raddoppio. Classicamente quindi sono definite 31 frequenze che rispettino (in modo leggermente approssimato) questa suddivisione, prendendo come punto di riferimento l'1 KHz. Avremo quindi 31 filtri centrati alle seguenti frequenze (espresse in Hertz): 20, 25, 31, 40, 50, 62, 80, 100, 125, 160, 200, 250, 320, 400, 500, 640, 800, 1000, 1250, 1600, 2000, 2500, 3200, 4000, 5000, 6400, 8000, 10000, 13000, 16000 e 20000.

Nasce il problema di definire quanto fare larghe le bande di intervento in modo da ottenere le funzioni di trasferimento volute. Il problema sorge principalmente in quanto un grosso guadagno di una banda, influisce sulle adiacenti modificandone il guadagno. A tal riguardo esistono varie filosofie, e per spiegarle sono state fatte simulazioni su un caso particolare che mostra in modo chiaro il problema. Poniamo di essere nella situazione in cui 7 bande consecutive vengano fatte guadagnare il massimo (12 dB di consuetudine).

In Figura 7.14 è mostrata la simulazione della prima filosofia di pensiero: bande strette che non si influenzano. La banda di ogni filtro è infatti calcolata come la metà della differenza tra la frequenza successiva e la precedente. È immediato notare come l'influenza reciproca dei vari filtri sia limitata, ma anche il fatto che il ripple al centro di due bande consecutive sia notevole. Questo approccio, proprio per le questo problema non viene mai utilizzato.

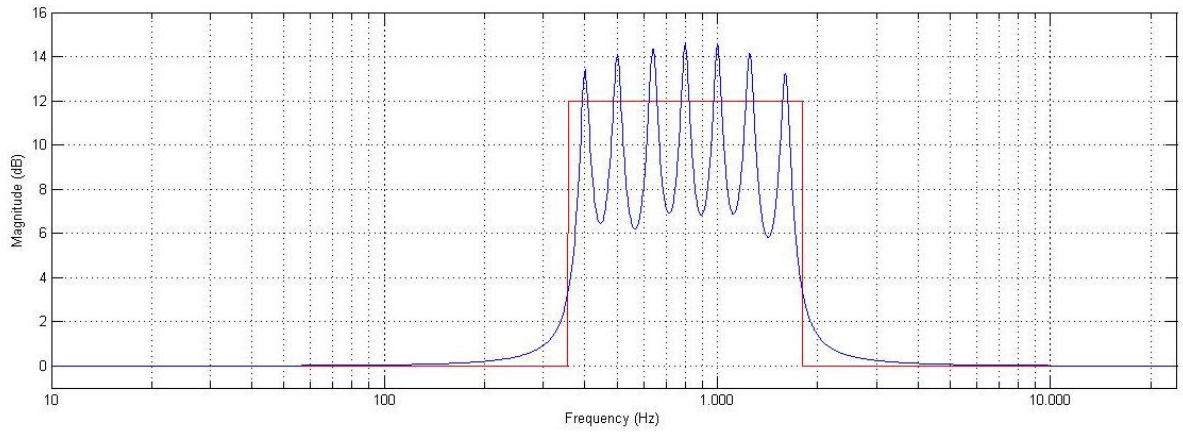

Figura 7.14 - Equalizzatore Grafico - Larghezza di banda calcolata come (frequenza successiva - frequenza precedente)/2

In Figura 7.15, invece è mostrata la simulazione della seconda filosofia, che predilige l'uniformità del guadagno. La banda è quindi il doppio di quella calcolata nel caso precedente. Come si può facilmente notare, il ripple si riduce notevolmente a scapito di un guadagno che diverge da quello

voluto. Anche in questo caso siamo in una situazione limite, e quindi non è così impensabile utilizzare questo tipo di soluzione, che infatti è quella più comune, in quanto rispecchia l'andamento in frequenza dei vecchi equalizzatori analogici. Non bisogna mai dimenticare quanto la tradizione, in campo audio, sia un aspetto parecchio rilevante.

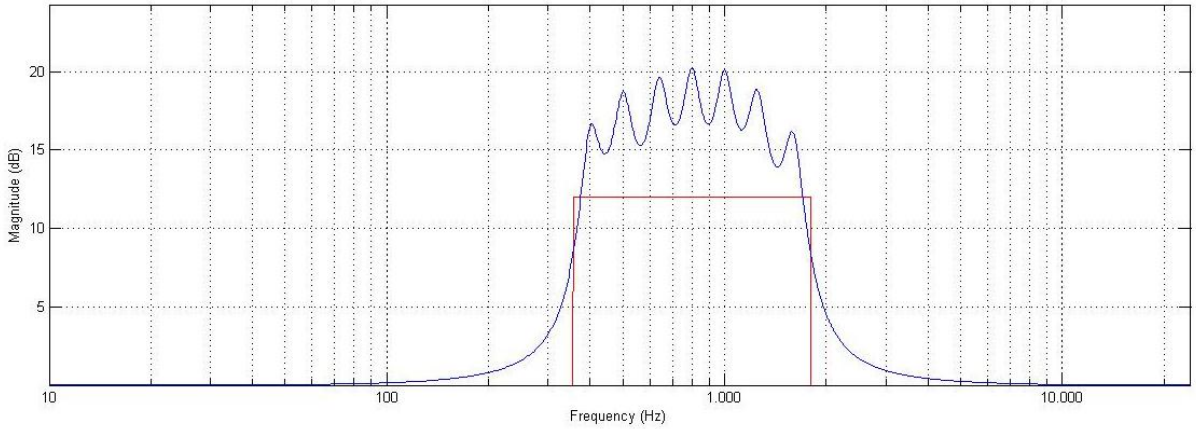

Figura 7.15 - Equalizzatore Grafico - Larghezza di banda calcolata come (frequenza successiva - frequenza precedente)

Non soddisfatti di questo tipo di risposta in frequenza, e avendo a disposizione un microcontrollore molto potente ci siamo chiesti se fosse possibile creare un algoritmo che potesse avere dei risultati migliori, in modo da emulare un comportamento di tipo *True-Response*.

Dopo vari tentativi che e miravano a trovare delle formule chiuse con cui calcolare i coefficienti a priori, ci si è resi conto che il problema era troppo complesso per essere risolto. Anche un approccio a look-up table avrebbe richiesto una quantità di memoria spropositata, e un tempo di raccolta dati inimmaginabili.

Si è intrapresa quindi un'altra strada inconsueta per questo tipo di problemi. L'idea di base è la seguente: grazie alla potenza di calcolo del microcontrollore, è possibile in tempi ragionevoli, simulare la risposta in frequenza della catena di filtri, in alcuni punti; a questo punto è possibile calcolare l'errore tra la funzione reale e quella ideale in quei punti. Infine bisogna usare questi valori calcolati per ridurre l'errore stesso. Sostanzialmente si tratta di una retroazione molto atipica, che può vagamente ricordare una soluzione ai minimi quadrati, e difficilmente studiabile con i meccanismi classici. Attraverso un grande sforzo di simulazione e validazione dei risultati, è stato possibile realizzare un algoritmo di calcolo che si può considerare un buon compromesso tra qualità e costo computazionale.

7.5.1 Calcolo della funzione di trasferimento

Primo punto da affrontare per lo sviluppo dell'algoritmo è stato dove e come calcolare il modulo della funzione di trasferimento complessiva.

Siccome per ogni filtro si devono aggiustare due parametri si è pensato di calcolare la funzione in due punti per ogni filtro, ovvero nel punto dove agisce, per regolarne il guadagno, e nel punto intermedio tra le frequenze centrali di due filtri contigui, per regolarne la banda.

Per il calcolo la questione è più complessa. Consideriamo innanzi tutto una generica F(z)

$$F(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} \tag{45}$$

ricordando che

$$e^{j\omega} = \cos\omega + j\sin\omega \tag{46}$$

si ha

$$F\left(e^{j\omega}\right) = \frac{a_0 + a_1\cos\omega + j\,a_1\sin\omega + a_2\cos^2\omega - a_2\sin^2\omega + j a_2\cos\omega\sin\omega}{1 + b_1\cos\omega + j\,b_1\sin\omega + b_2\cos^2\omega - b_2\sin^2\omega + j b_2\cos\omega\sin\omega} \tag{47}$$

e quindi

$$\left|F\left(e^{j\omega_0}\right)\right| = \frac{|N|\big|_{\omega_0}}{|D|\big|_{\omega_0}} \tag{48}$$

dove

$$|N|\big|_{\omega_0} = \sqrt{[a_0 + a_1\cos\omega_0 + a_2\cos^2\omega_0 - a_2\sin^2\omega_0]^2 + [a_1\sin\omega_0 + a_2\cos\omega_0\sin\omega_0]^2} \tag{49}$$

$$|D|\big|_{\omega_0} = \sqrt{[1 + b_1\cos\omega_0 + b_2\cos^2\omega_0 - b_2\sin^2\omega_0]^2 + [b_1\sin\omega_0 + b_2\cos\omega_0\sin\omega_0]^2} \tag{50}$$

Riconducendoci al nostro caso particolare abbiamo che

$$F(z) = \frac{0.5(1 + k_2 + K - Kk_2) + k_1(1 + k_2)z^{-1} + 0.5(1 + k_2 - K + Kk_2)z^{-2}}{1 + k_1(1 + k_2)z^{-1} + k_2 z^{-2}} \tag{51}$$

Basta ora sostituire nella (48) e nella (49) i coefficienti per calcolare il modulo della funzione in un punto ω_0.

Per facilitare il compito del microcontrollore, in fase di inizializzazione vengono creati due vettori con il seno e il coseno di ω_0 per tutte e 61 le posizioni in cui deve essere calcolato il modulo in modo da non dover eseguire il calcolo a ogni iterazione.

Per ogni ω_0 viene quindi calcolato il modulo di ogni filtro e moltiplicato per il valore contenuto in un vettore ausiliario (vettore inizializzato a 1 all'inizio della procedura). Alla fine del processo il vettore conterrà tutti e 61 i valori della funzione di trasferimento complessiva. I cicli di calcolo saranno in totale 61x31=1891.

7.5.2 Correzione del parametro K

La prima correzione che viene effettuata è quella sui coefficienti K dei vari filtri.

Nella prima fase vengono quindi presi in considerazione solo i parametri dispari del vettore in cui è contenuto il modulo della funzione di trasferimento, a cui corrisponde il guadagno reale di centro banda di ogni filtro.

Per ognuno viene per prima cosa calcolato il rapporto (R) tra il guadagno ideale in quel punto (G) e il modulo simulato dal microcontrollore (che a meno errori minimali di round off può essere considerato a buon ragione quello reale, e che chiameremo H)

$$R = \frac{G}{H} \tag{52}$$

Si pone ora il problema di come usare questo parametro R per modificare il parametro K e sistemare il guadagno della funzione di trasferimento. L'idea è quella di moltiplicare il coefficiente corrente per R. Facendo così, però, si va incontro a effetti di divergenza, in quanto la funzione di trasferimento in ogni punto, data la larghezza di banda dei filtri, può essere fortemente influenzata da quelli adiacenti. Si creano quindi situazioni in cui in passi successivi H alternativamente è prima maggiore, poi minore di G, aumentando a ogni iterazione l'errore commesso.

Per ovviare a questo problema, una prima implementazione dell'algoritmo prevedeva di elevare R a un coefficiente α<1.

$$K_{new} = K_{old} R^{\alpha} \tag{53}$$

Attraverso il buon senso e la validazione delle simulazioni, è stato scelto α=0.4, valore che evita in ogni condizione la divergenza del sistema, e lo fa convergere in circa 10 iterazioni.

Ci si è però resi conto dello sforzo computazionale che prevedo l'operazione di elevazione a potenza per un microcontrollore. La formula che si usa quindi per calcolare il nuovo coefficiente K, ricavata tramite linearizzazione attorno al punto R=1, è la seguente

$$K_{new} = K_{old}(0.6 + 0.4R) \tag{54}$$

che offre circa la stessa efficacia della precedente, ma è molto più leggera a livello di costo computazionale.

7.5.3 Correzione della larghezza di banda dei filtri

La seconda modifica effettuata dall'algoritmo è quella sulla larghezza delle bande dei singoli filtri, allo scopo di ridurre il ripple: in linea di principio, quello che avviene è che, quando più filtri contigui guadagnano in modo simile, la larghezza di banda viene aumentata, per diminuire l'ampiezza degli ondeggiamenti della funzione di trasferimento; quando, invece, si riscontrano forti differenze di guadagno tra due filtri consecutivi, la larghezza viene diminuita, al fine di favorire le discontinuità.

Abbiamo già detto come il modulo della funzione di trasferimento viene calcolato nel punto mediano a due filtri consecutivi. Per ognuno di questi punti viene calcolato un rapporto (Q) che esprime quantitativamente la sua posizione rispetto al modulo della funzione di trasferimento nelle frequenze d'azione dei filtri adiacenti. Chiamando ω_0 la frequenza di questo punto, ω_{-1} quella del filtro precedente e ω_1 quella del successivo il rapporto viene calcolato nel seguente modo

$$Q = \frac{\left|F\left(e^{j\omega_{-1}}\right)\right| + \left|F\left(e^{j\omega_1}\right)\right|}{2|F(e^{j\omega_0})|} \tag{55}$$

dove tutti i moduli erano già stati calcolati precedentemente.

Bisogna precisare che se il modulo del punto centrale è compreso tra i moduli dei due punti adiacenti, l'algoritmo non interviene, poiché ci si trova in una situazione accettabile e per non complicare ulteriormente un processo già computazionalmente molto pesante.

Se invece il valore è superiore o inferiore a entrambi i due punti contigui si possono verificare due situazioni, gestite in maniera complementare.

Se avviene che

$$\frac{\left|F\left(e^{j\omega_{-1}}\right)\right| + \left|F\left(e^{j\omega_1}\right)\right|}{2} < 1 \tag{56}$$

le bande dei due filtri vengono divise per il fattore Q^{β}, dove è stato introdotto il fattore β che serve ad evitare di far divergere l'algoritmo, in modo analogo a quanto succedeva con il guadagno. Grazie alla simulazione è stato possibile scegliere un valore adeguato per questo parametro, e si ha $\beta = 0.2$.

$$BW_{-1\,new} = \frac{BW_{-1\,old}}{Q^{\beta}} \tag{57}$$

$$BW_{1\,new} = \frac{BW_{1\,old}}{Q^{\beta}} \tag{58}$$

In caso contrario invece le due correzioni avvengono in senso opposto, ovvero

$$BW_{-1\,new} = BW_{-1\,old}Q^{\beta} \tag{59}$$

$$BW_{1\,new} = BW_{1\,old}Q^{\beta} \tag{60}$$

Come nel caso precedente, è stata sostituita l'operazione di elevamento a potenza con la sua linearizzazione.

Nel primo caso si ha quindi

$$BW_{-1\,new} = \frac{BW_{-1\,old}}{0.8 + 0.2Q} \tag{61}$$

$$BW_{1\,new} = \frac{BW_{1\,old}}{0.8 + 0.2Q} \tag{62}$$

mentre nel secondo

$$BW_{-1\,new} = BW_{-1\,old}(0.8 + 0.2Q) \tag{63}$$

$$BW_{1\,new} = BW_{1\,old}(0.8 + 0.2Q) \tag{64}$$

7.5.4 Iterazione dell'algoritmo

Sempre attraverso la simulazione Matlab, dopo alcune prove, è stato chiaro che si ottengono miglioramenti significativi reiterando il procedimento appena descritto circa 6 volte. Per ottenere buoni risultati anche nelle situazioni più critiche, l'algoritmo viene quindi reiterato 10 volte. Lo schema concettuale delle operazioni svolte dal microcontrollore per il calcolo dei coefficienti è mostrato in Figura 7.16.

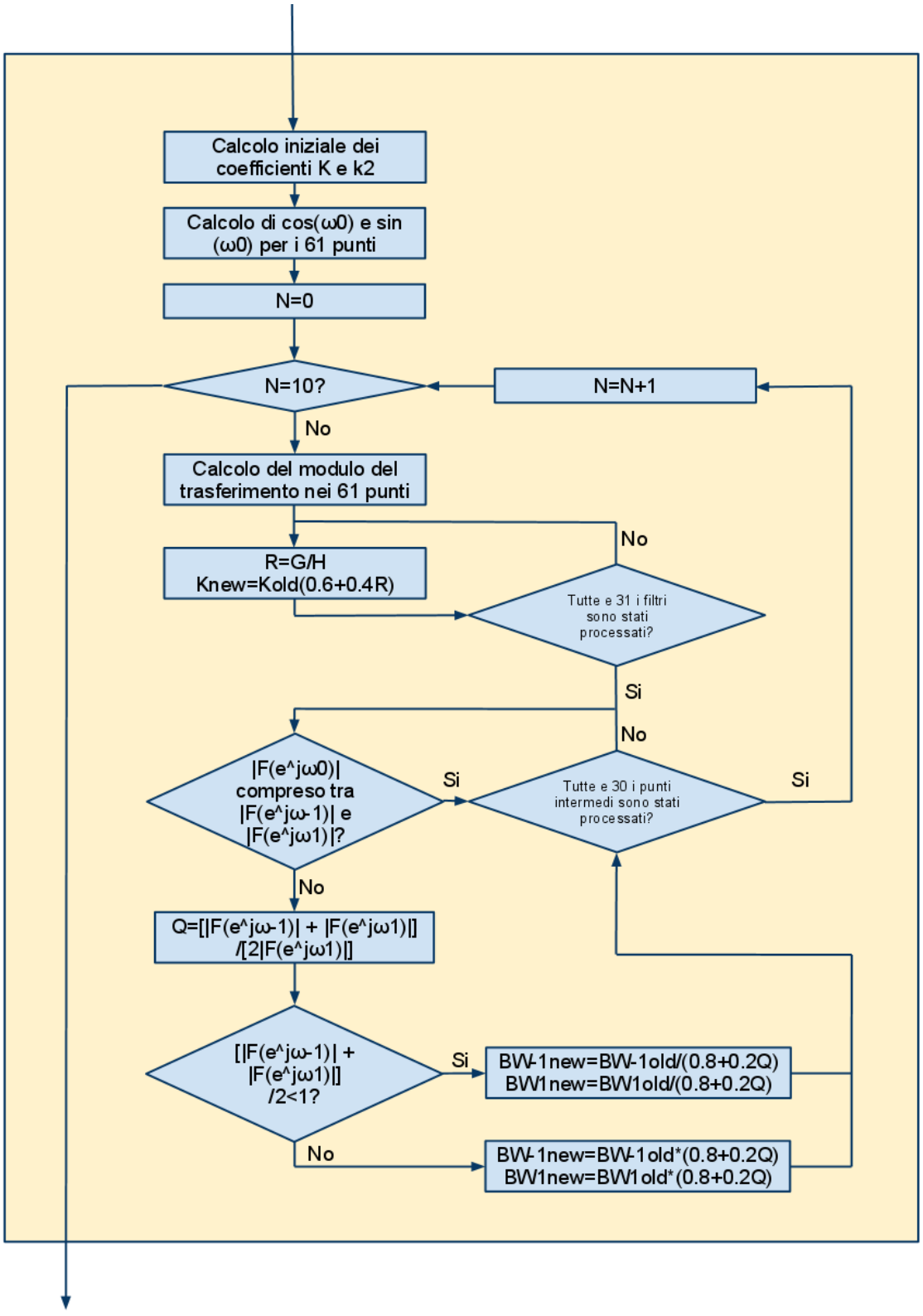

Figura 7.16 - Algoritmo di calcolo dei coefficienti dell'equalizzatore grafico

In Figura 7.17 è mostrato il risultato della simulazione utilizzando le correzioni con le potenze, in Figura 7.18 quello con le linearizzazioni. Le differenze sono minime, e i vantaggi notevoli.

Se poi si confrontano questi risultati con le simulazioni in assenza di elaborazione dei coefficienti di Figura 7.14 e di Figura 7.15 appare molto chiaro come i gli sforzi necessari a sviluppare questo tipo di algoritmo non siano stati vani. Il miglioramento è evidente sotto ogni aspetto.

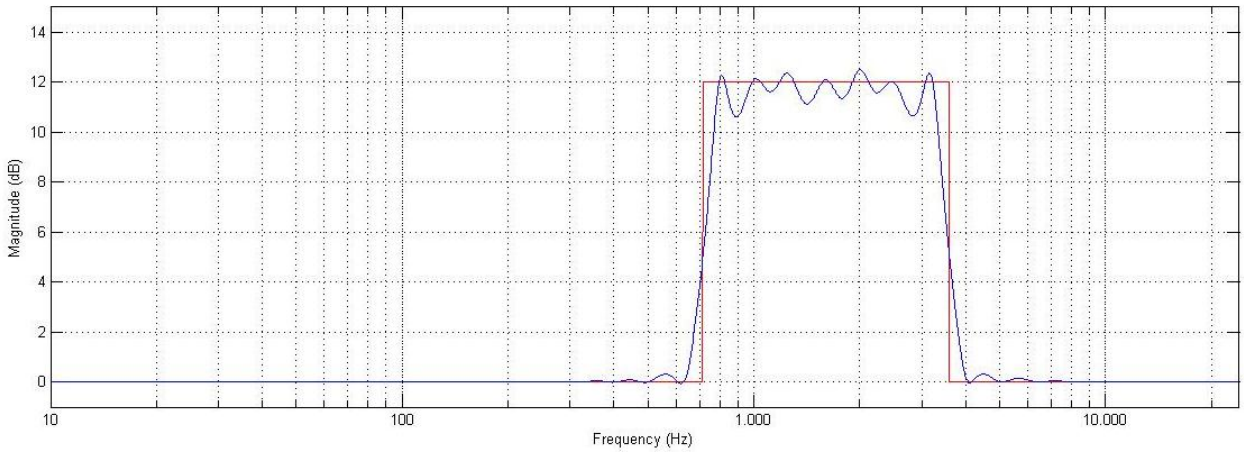

Figura 7.17 - Simulazione algoritmo generale

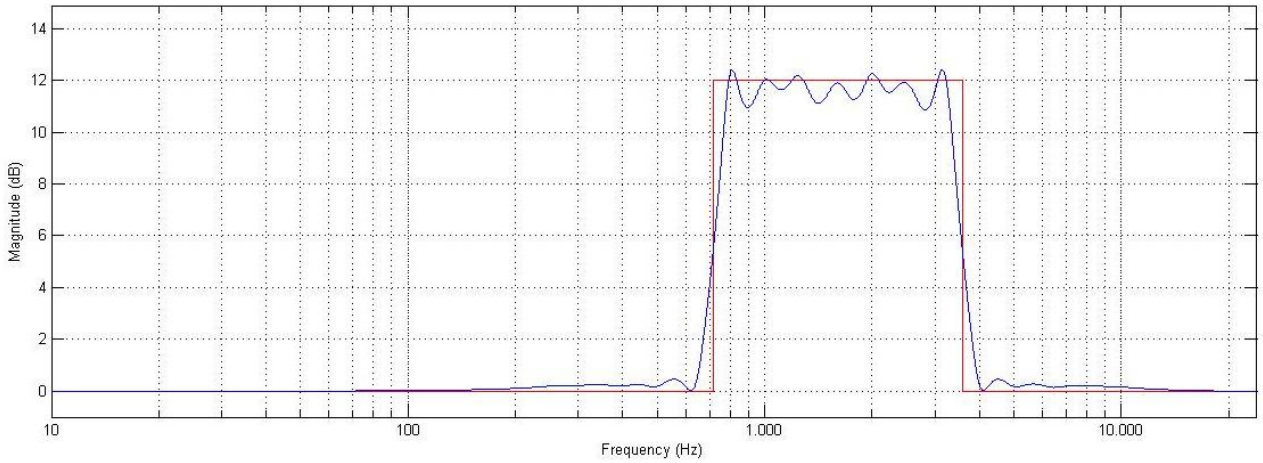

Figura 7.18 - Simulazione algoritmo ottimizzato con linearizzazione delle potenze

8. Struttura FPGA

Passiamo in questa sezione alla descrizione del progetto implementato nell'FPGA.

La programmazione è avvenuta in linguaggio VHDL e l'implementazione è stata generata mediante l'utilizzo del programma fornito da Xilinx, ISE 12.1 come spiegato nel capitolo 5.

La struttura generale del progetto è mostrata in Figura 8.1.

I quattro canali d'ingresso e gli otto d'uscita viaggiano su un unico bus di 24 bit: il sincronizzatore si occupa di generare un vettore di 4 bit che indica quale canale è presente sul bus (*channel select*), un segnale di validità della durata di un colpo di clock (*data valid*), che indica quando sul bus è è disponibile un nuovo dato, e un segnale di sincronia (*sync data*), che viene asserito il colpo di clock precedente alla variazione del segnale sul bus. Il segnale di validità indica ai blocchi in lettura quando un nuovo dato è presente, il segnale di sincronia avvisa i moduli in scrittura quando devono essere asseriti i nuovi campioni.

In alto a sinistra si può notare il ricevitore: una parte si occupa della ricezione digitale, decodificando il segnale AES/EBU e inviandolo al *sample rate converter* per il ricampionamento alla frequenza locale, mentre l'altra è dedicata alla ricezione analogica, tramite due moduli I2S, che interpretano i segnali provenienti dagli ADC. I segnali ricevuti, vengono portati all'uscita dell'elemento tramite un multiplexer, che seleziona i canali analogici piuttosto che quelli digitali in base ai segnali *A/D select*. I quattro canali d'ingresso viaggiano ancora su bus PCM separati.

Un multiplexer in ingresso si occupa di inserire i quattro canali d'ingresso nel bus PCM comune: ogni volta che viene asserito il segnale di sincronia, viene letto il canale corrente e viene posto in ingresso al generatore di ritardo uno dei quattro ingressi, piuttosto che il bus in uscita dal processore.

Il delay generator, con l'ausilio di una memoria RAM esterna, inserisce un ritardo puro ai vari canali in modo indipendentemente controllabile. I segnali in uscita, dopo essere sincronizzati, vengono portati al processore e al de-multiplexer.

Il processore, compie l'elaborazione sui vari canali, e li riporta al multiplexer d'ingresso.

Infine il de-multiplexer, agisce sui canali d'uscita, fornendoli al trasmettitore, che, in base ai *D/A select*, li invia tramite I2S ai DAC, o in formato AES/EBU ai driver per la trasmissione digitale.

Contestualmente, un modulo SPI slave si preoccupa della ricezione e dello smistamento dei comandi provenienti dal microcontrollore.

Tutte le operazioni avvengono in modalità sincrona, con temporizzazione derivante da un clock a 98.304 MHz, gestito da un *Clock Manager*.

Nei paragrafi verranno analizzati i singoli moduli nel dettaglio.

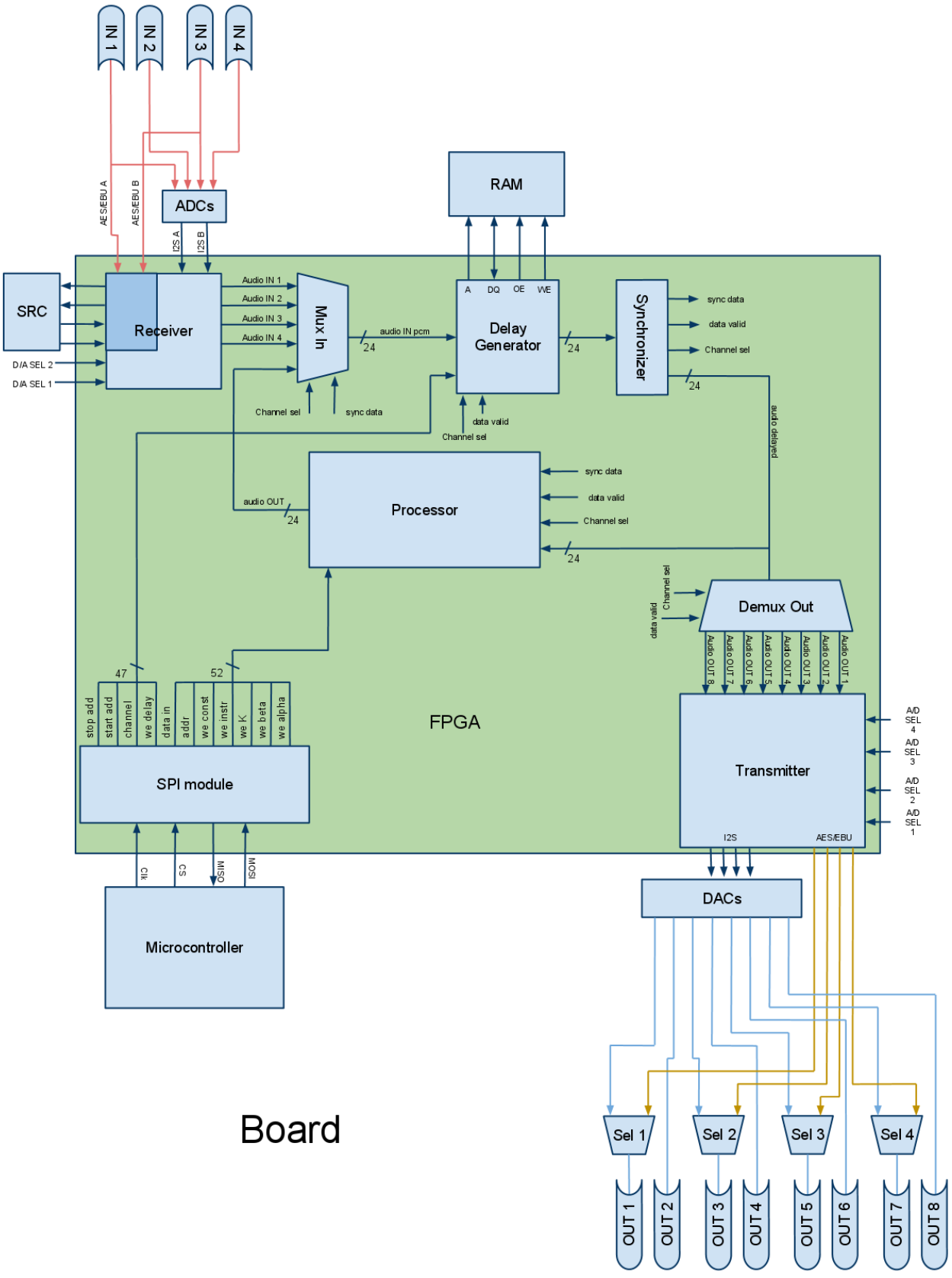

Figura 8.1 - Struttura generale FPGA

8.1 Ricevitore

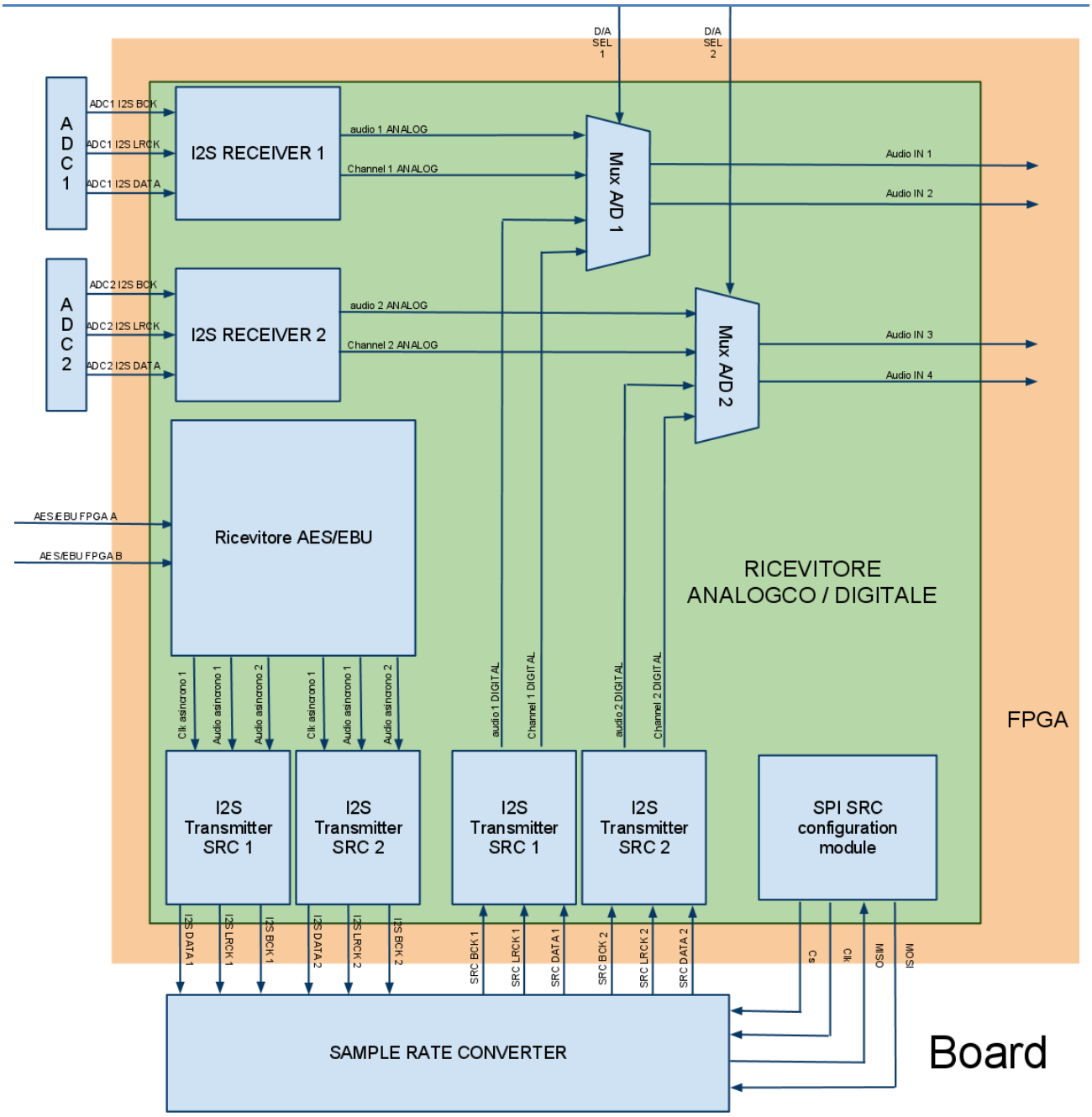

Figura 8.2 - Ricevitore

Il ricevitore è quell'apparato che si occupa di leggere i flussi di dati audio in ingresso e di fornirli in modalità pcm a 24 bit al sistema. Come abbiamo già visto, i flussi possono arrivare o dagli adc, e quindi dai bus I2S, o dal tranceiver AES/EBU. Il sistema è quindi composto da 2 ricevitori I2S e 2 ricevitori AES/EBU. Saranno poi due mux d'uscita a determinare, in base agli appositi segnali di controllo, se selezionare i canali analogici o quelli digitali.

Per gestire il flusso analogico sono stati creati due semplici ricevitori I2S in modalità slave. I due canali codificati su ogni bus vengono trasformati in flusso pcm su un'unica linea. Un segnale di controllo, definisce quale canale è in quel momento sul bus.

Il percorso degli ingressi digitali è invece più complesso. È stato creato un doppio ricevitore AES/EBU: per ognuno dei due bus il blocco restituisce i 2 canali audio e il clock estratto dal bit stream. Tutti i dati ausiliari vengono ignorati, anche se, per sviluppi futuri, data la struttura modulare molto versatile, sarà possibile recuperare queste funzionalità se dovesse essere necessario.

Come visto nel capitolo 4.2, il segnale così ricevuto avrà frequenza di campionamento diversa rispetto ai 48 KHz del sistema, anche se la frequenza nominale dovesse essere la stessa. Infatti, anche un minimo errore di questo parametro, può portare a un deterioramento non indifferente del rapporto segnale rumore. Questo è dovuto alla perdita di campioni, in caso la sorgente avesse frequenza maggiore, o alla loro mancanza, nel caso contrario.

Bisogna quindi passare i dati al sample rate converter esterno. Per fare ciò sono stati creati due trasmettitori I2S in modalità master, che generano i segnali di sincronia a partire dal clock estratto dall'AES/EBU. L'SRC a questo punto elabora i dati e li restituisce ancora in formato I2S ad altri due ricevitori, che decodificano il flusso e forniscono i canali in uscita con la stessa modalità dei ricevitori analogici.

L'SRC4184 necessita di una configurazione iniziale. Durante l'accensione della scheda, in caso di modifiche del valore dei D/A select (che definiscono l'utilizzo dei canali analogici piuttosto di quelli digitali) e in caso di malfunzionamenti, un modulo si occupa di inizializzare il dispositivo, attraverso un bus SPI dedicato: quando una delle tre condizioni si verifica, comunica quali canali sono attivi in base agli D/A select e disattiva il mute. Inoltre, per prevenire eventuali altri problemi, a ogni comunicazione vengono settati i registri del sample rate converter indicando come protocollo di comunicazione l'I2S a 24 bit e la modalità di interpolazione più precisa, quella a 64 campioni (con ritardo di elaborazione maggiore, cosa che però non disturba particolarmente).

8.2 Trasmettitore

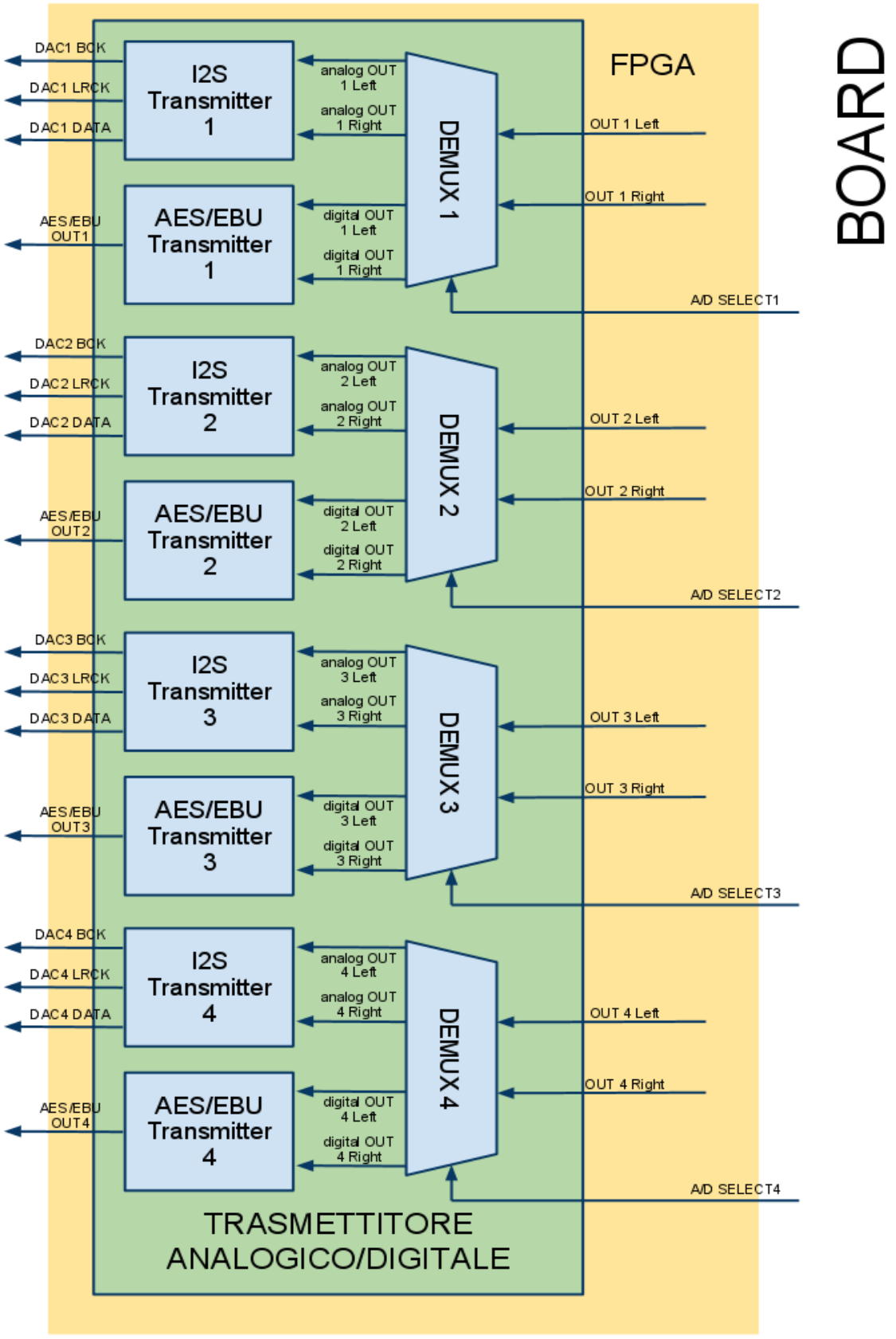

Figura 8.3 - Trasmettitore

La struttura del trasmettitore è molto semplice. Il modulo è formato di quattro blocchi logici identici, composti da un trasmettitore I2S, un trasmettitore AES/EBU e un de multiplexer che fornisce i dati a un componente piuttosto che all'altro in base a valore dell' *A/D select* associato.

Il segnale in uscita dai moduli I2S vengono quindi trasmessi ai DAC esterni, mentre i flussi generati dai trasmettitori AES/EBU raggiungono il driver LVDS.

Lo stream digitale, in questo caso, contrariamente a quanto succede per il ricevitore, che ignora i dati di controllo, contiene le informazioni relative alla frequenza di campionamento e al numero di bit di segnale.

8.3 Sincronizzatore

Il sincronizzatore è quello che può essere definito il *direttore d'orchestra* del sistema.

Inizialmente il sistema era stato progettato per far viaggiare i dati PCM su bus separati. Questo rendeva però necessario, in ingresso e in uscita ai blocchi di elaborazione, che lavorano in maniera sequenziale, l'implementazione di multiplexer e de-multiplexer di dimensioni enormi. Le risorse necessarie divergevano quindi molto rapidamente fin dall'implementazione dei primi moduli, e soprattutto il valore delle LUT necessarie, lasciava poco spazio a eventuali ottimizzazioni di questa strada realizzativa.

Si è scelto quindi di creare un bus unico su cui far circolare i dati, un solo multiplexer d'ingresso e un solo de-multiplexer d'uscita.

Il sincronizzatore svolge due funzioni: quella principale di porsi ad arbitro del bus e quella di sincronizzare i dati in arrivo dal generatore di ritardi. Questa funzione si rende necessaria proprio per come è costituito il delay generator e sarebbe potuta essere implementata direttamente in quel modulo; per ragioni di secondaria importanza, che potrebbero essere definite estetiche, è stato scelto di far allineare i flussi di dati da questo elemento.

Il sincronizzatore basa le sue attività su un contatore incrementato dal clock principale a 98 MHz. Il contatore viene resettato con periodo di 20.8 µs (1/48 KHz); questo periodo principale viene diviso equamente in 12 parti, ciascuna corrispondente a un canale. Come si può vedere dalla Figura 8.4, al colpo di clock corrispondente a una delle 12 soglie viene incrementato il valore del vettore channel select e viene asserito il bit di *data valid*. Al colpo di clock precedente, viene invece posto a 1 il bit di *sync data*.

Il *data valid* permette quindi ai moduli posti in ascolto sul bus di capire quando un nuovo dato vi è presente, il *sync data*, invece, permette ai blocchi in scrittura sul bus di sapere quando possono immettere un nuovo dato, facendo attenzione a leggere il *channel select*, incrementandolo di 1. Questo perché il dato che verrà scritto, sarà visto dal sistema il colpo di clock successivo, quando il *channel sel* sarà stato incrementato.

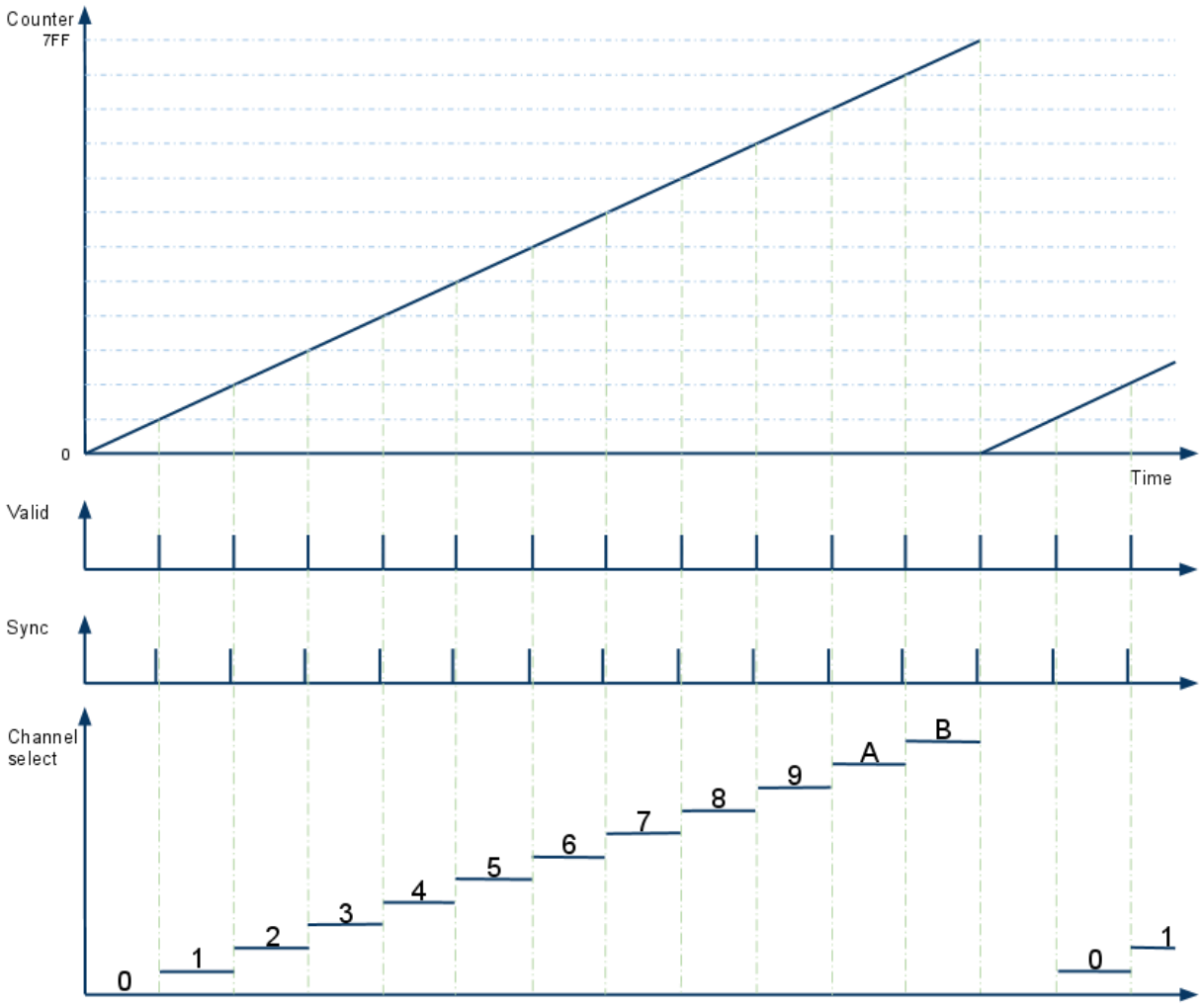

Figura 8.4 - Sincronizzatore

8.4 Generatore di ritardi

Il generatore di ritardi si basa su una memoria ram esterna all'FPGA con capacità massima di 32 Megabit (la MT45W2MW16PAFA di Micron). La comunicazione tra i dispositivi avviene su un bus parallelo che è composto da 21 bit d'indirizzo, 16 di dato, 1 bit di word enable e 1 bit di output enable.

La memoria è strutturata in parole da 16 bit, mentre i dati che a noi interessa memorizzare hanno lunghezza di 24 bit. Nascono quindi due alternative di progettazione.

La prima è quella di sfruttare interamente la memoria, mettendo 2 dati ogni 3 parole di memoria. Questa è stata la scelta che inizialmente è stata portata avanti. Seguendo questa strada, nascono però parecchi problemi: il modulo di controllo della memoria deve accumulare due dati prima di andarli a scrivere, e deve successivamente leggerne due. Si crea una complessità di sincronizzazione e di progettazione non indifferente, con problemi anche sulla una quantità di risorse utilizzate assolutamente inaccettabile.

La seconda alternativa è quella di memorizzare un dato in 2 parole di memoria, lasciando inutilizzati 8 bit ogni 32. Questa soluzione, che in fin dei conti spreca il 25% della memoria, risulta conveniente, in quanto la ram è sovradimensionata rispetto alle specifiche che deve rispettare, e permette di utilizzare la semplice struttura mostrata in Figura 8.5.

In basso possiamo notare tre catene di registri a scorrimento: due fissi, che contengono gli indirizzi di inizio e fine per ogni canale, e una variabile che contiene l'indirizzo corrente.

Ogni qual volta arriva un segnale di *data valid* dal sincronizzatore, viene attivata la catena. La temporizzazione viene gestita dalla struttura composta dai registri ritardatori del *data valid* (in figura sono quelli posti in alto a sinistra).

Per prima cosa vengono fatti scorrere i registri in avanti di una posizione. Al passo successivo vengono comparati l'indirizzo corrente con quello di fine: se sono uguali l'indirizzo successivo dovrà essere quello di inizio, poiché per ogni canale è allocato un ben determinato spazio sulla RAM, e quello corrente è l'ultimo indirizzo disponibile a tale canale. Se invece sono diversi, l'indirizzo corrente viene incrementato di 2, in quanto ogni dato viene memorizzato in 2 locazioni.

L'indirizzo corretto viene quindi inviato al modulo di lettura, che interroga la ram e restituisce il dato memorizzato in quelle due locazioni di memoria un tempo τ prima. Il tempo di ritardo può essere facilmente calcolato come $\tau = \frac{indirizzo\ fine - indirizzo\ inizio}{2} * \frac{1}{f_{sample}}$.

Quando il modulo di lettura ha terminato le sue operazioni, attiva il modulo di scrittura che va a inserire, nella stessa locazione di memoria, il nuovo dato audio in ingresso.

Al reset del sistema gli indirizzi vengono settati sul ritardo minimo. È il microcontrollore che si occupa successivamente di fornire gli indirizzi di inizio e fine per ogni canale. Ogni qualvolta questi valori vengono cambiati, il microcontrollore li comunica tutti in ordine, questi vengono

memorizzati e inseriti attraverso un multiplexer nel primo registro. L'operazione richiede quindi un periodo di campionamento per essere compiuta. Inoltre, finché il canale con ritardo maggiore non ha finito un ciclo completo, i dati presenti non sono validi: in uscita vengono quindi posti solo zeri finché il sistema non va a regime.

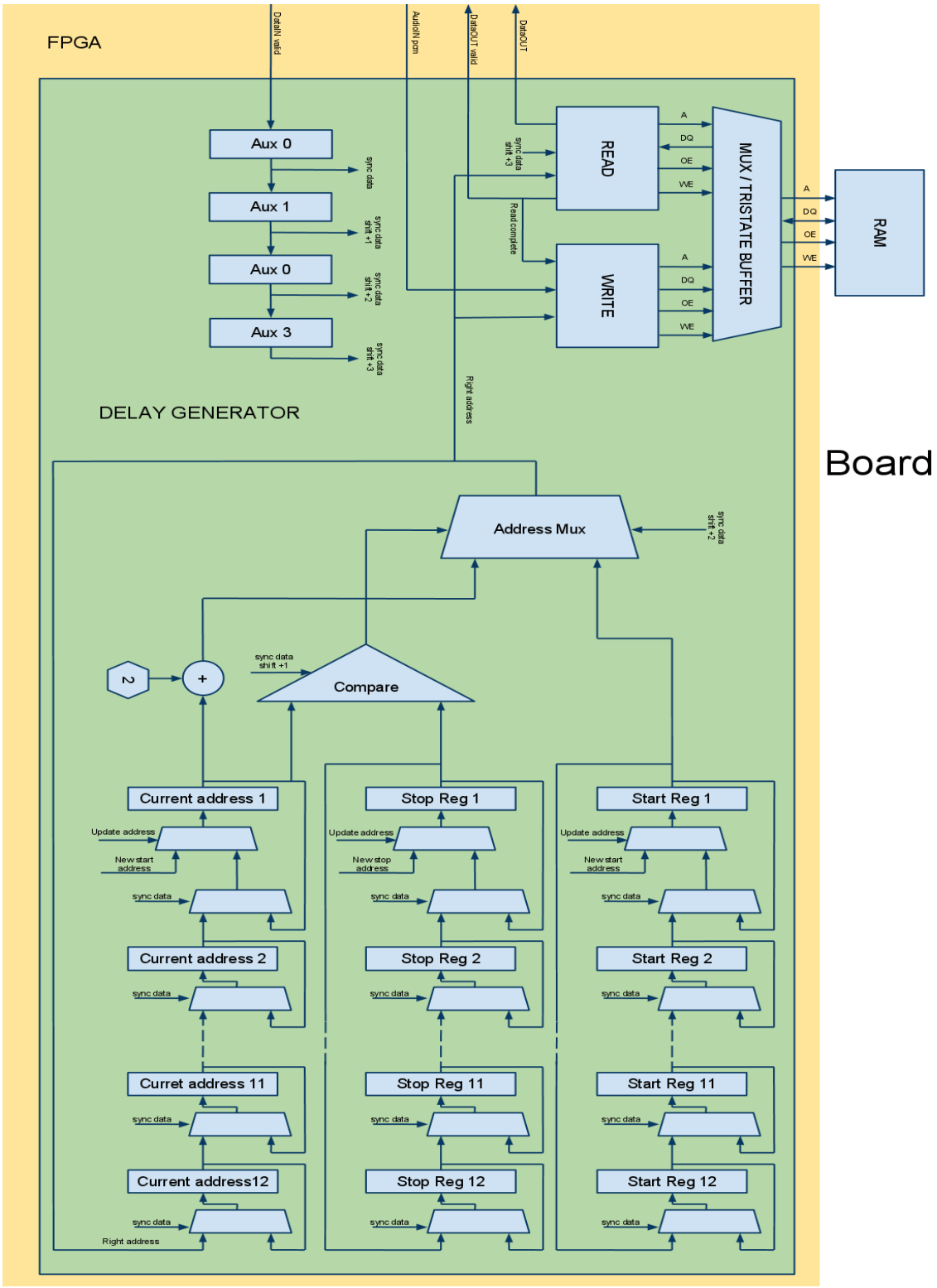

Figura 8.5 - Generatore di ritardi

8.4.1 Read and Write Ram

Per le operazioni di lettura e scrittura della ram sono stati creati appositamente due moduli. Questi blocchi vengono attivati da un segnale di controllo fornito dal blocco di generazione dei ritardi, che si occupa anche di gestire il tristate buffer, il cui scopo è di mettere il bus dati in modalità ingresso quando opera il modulo di lettura, in uscita quando ad agire è il blocco di scrittura.

Come rappresentato in Figura 8.6 quando il segnale *Read Go* viene asserito, il modulo imposta sul bus il nuovo indirizzo e contestualmente pone a 0 le linee di controllo CE e OE. Il colpo di clock successivo le asserisce a 1 e aspetta che la ram metta sul bus dati la parola di 16 bit. Questo deve avvenire in un tempo massimo di 85 ns (come da specifica su datasheet della memoria RAM). Per una maggior sicurezza, e considerate le non stringenti tempistiche, il modulo aspetta 12 colpi di clock, pari circa a 120 ns. A questo punto i 16 bit letti vengono trasferiti nei bit più significativi del vettore che verrà poi indirizzato in uscita. Il passo successivo consiste nell'incrementare l'indirizzo e compiere un'altra lettura con la stessa modalità. I primi 8 dei 16 bit letti vengono trasferiti nei bit meno significativi dello stesso vettore che a questo punto viene posto in uscita. Contestualmente, viene asserito un segnale che indica la fine dell'operazione di lettura. Il modulo di delay azzera quindi il *Read Go*, porta in uscita il dato letto e passa alla modalità di scrittura.

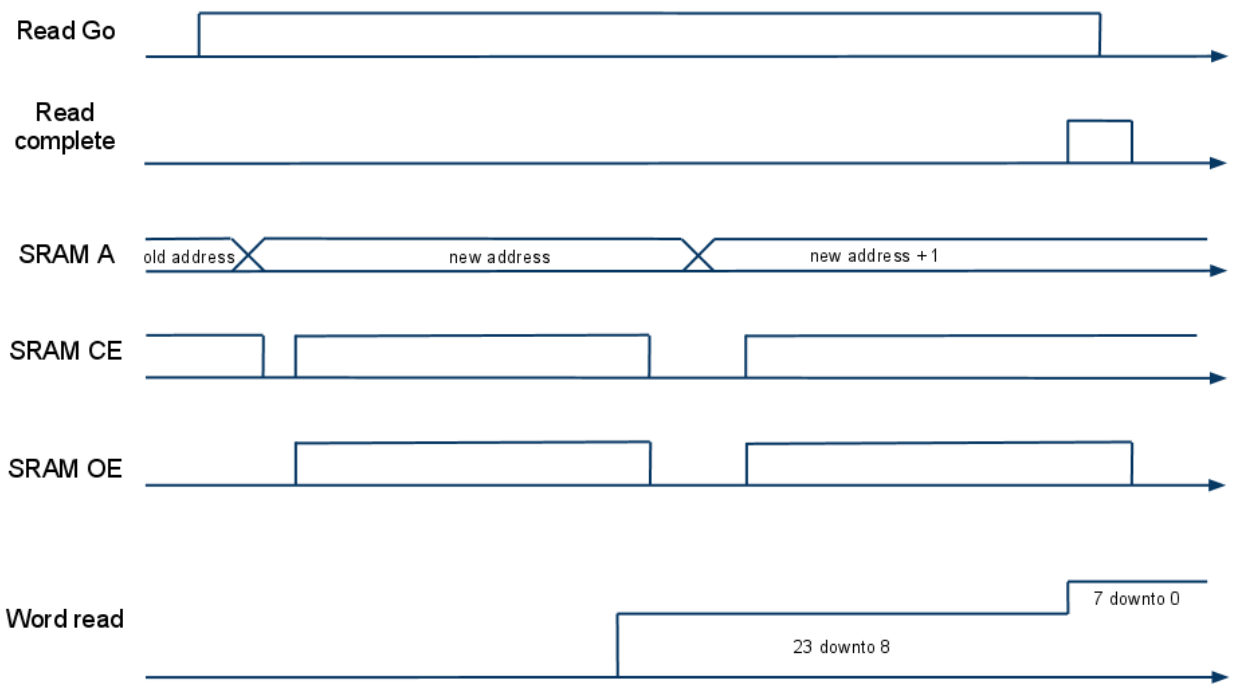

Figura 8.6 - Protocollo lettura ram

Analogamente a quanto avviene per la lettura, il modulo di scrittura viene attivato dal blocco di generazione dei ritardi attraverso il segnale di *Write Go*. Per prima cosa viene asserito il primo indirizzo sul bus dedicato. Successivamente, vengono convogliati i primi 16 bit sul bus dati e contestualmente viene portato a 0 il bit WE. Dopo un'attesa minima di 12 colpi di clock (i 120 ns contro gli 85 ns massimi di tempo di memorizzazione, analogamente a quanto avveniva in lettura),

il WE viene alzato, viene incrementato l'indirizzo e si ripete la procedura, mandando sul bus dati gli ultimi 8 bit di dato seguiti da 8 zeri.

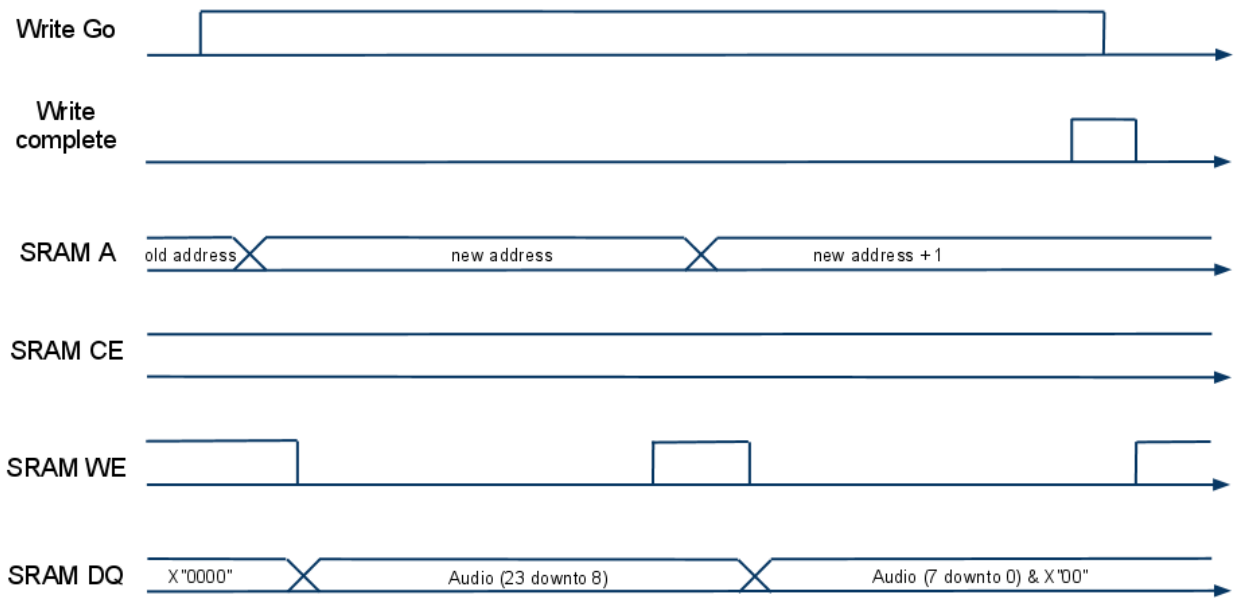

Figura 8.7 - Protocollo di scrittura ram

8.5 Serial Processor Router

Il serial processor router è la struttura temporal computing che si occupa di instradare i dati seguendo la giusta temporizzazione. Per prima cosa carica i dati audio in ingresso è quelli posti sul *linkbus*. Successivamente va a compiere le conversioni da intero a virgola mobile carica i dati agli ingressi delle catene del filtro IIR. Dopo aver aspettato che i dati vengano elaborati, e riconvertiti in interi, va a leggere i risultati e li pone in uscita, compiendo anche il mixing.

Tutto questo procedimento è gestito da una macchina stati (Figura 8.8). In Figura 8.9 e Figura 8.10 sono descritti i segnali di controllo su cui si basa questo modulo.

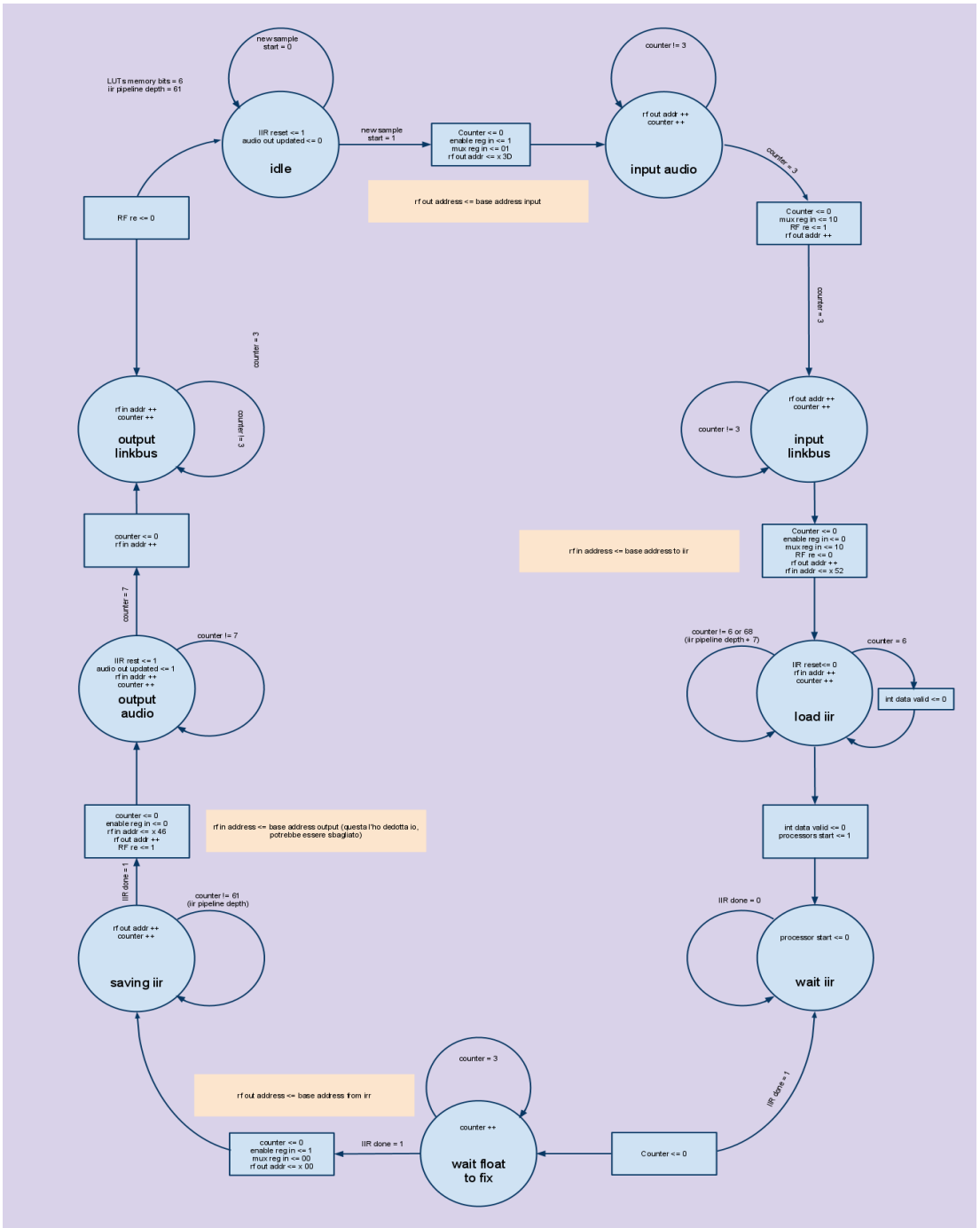

Figura 8.8 - Serial Processor Router

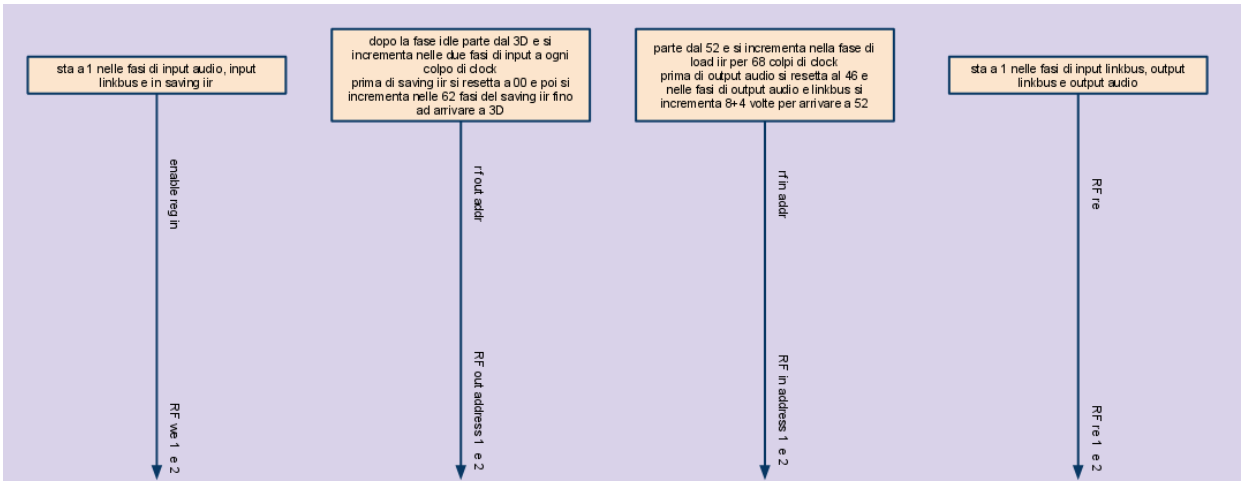

Figura 8.9 - Segnali di controllo Router A

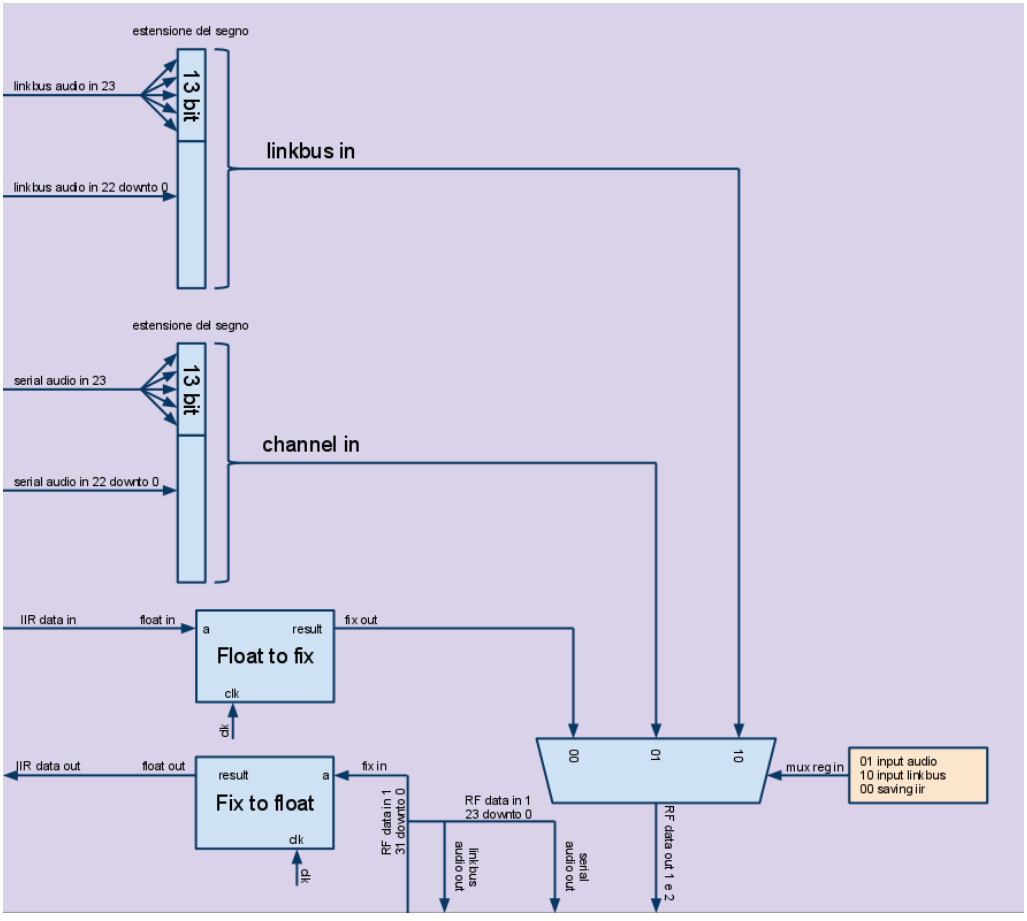

Figura 8.10 - Segnali di controllo Router B

8.6 Filtro IIR

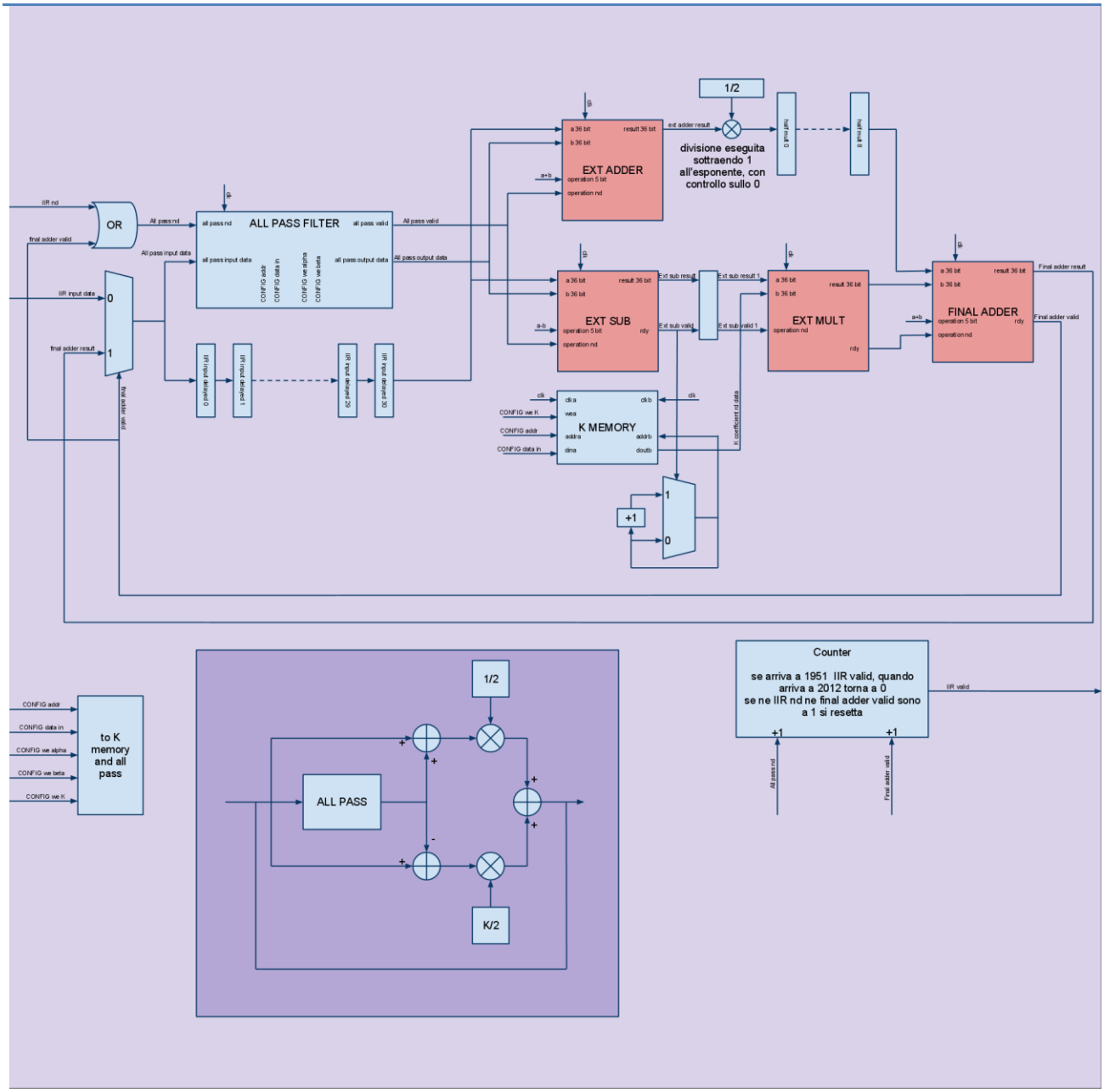

Figura 8.11 - 2nd order IIR

Uno dei moduli più importanti è quello che sintetizza il filtro IIR del second'ordine (di cui si è parlato in termini teorici nel capitolo 7).

In Figura 8.11 è mostrata, in basso, la struttura logica, in alto, la sua implementazione su FPGA.

La cosa più complessa è gestire le temporizzazioni, in modo che i giusti dati arrivino al colpo di clock corretto al blocco logico designato.

L'all-pass filter, di cui discuteremo nel prossimo paragrafo, ha un tempo di latenza di 31 colpi di clock. A tal scopo sono stati realizzati in parallelo ad esso 31 shift register che forniscono i dati d'ingresso ai due adder iniziali in contemporanea all'uscita dei dati dopo l'elaborazione del filtro.

I blocchi di addizione e sottrazione sono realizzati attraverso il medesimo ip core, che grazie a un segnale di controllo può fare entrambe le operazioni. La latenza dell'adder è di 10 colpi di clock.

A questo punto un ramo va moltiplicato per un coefficiente K/2, l'altro va semplicemente diviso per 2. La divisione per 2 risulta molto semplice: basta sottrarre infatti 1 all'esponente, facendo attenzione che il numero da dividere non sia già 0 o che l'operazione non porti all'*overflow*, il che porta a replicare il numero in uscita.

Per quanto riguarda il moltiplicatore vero e proprio, i coefficienti K sono memorizzati in una *dual-port ram* creata con l'apposito ip core. Sulla porta d'uscita, l'indirizzo viene incrementato ad ogni colpo di clock e, contestualmente, sul bus dati viene fornito il corrispondente valore della costante. I coefficienti devono essere quindi memorizzati nel giusto ordine. Il microcontrollore, può modificare un dato valore attraverso la comunicazione spi: il modulo di comunicazione ricevuto il giusto comando, può asserire il write enable, fornire l'indirizzo e il nuovo coefficiente da memorizzare, il tutto sulla porta d'ingresso.

Il moltiplicatore è un altro blocco logico generato a partire da un ip core floating point, e il suo tempo di latenza è di 9 colpi di clock. Questo rende necessario inserire 9 shift register sul ramo superiore.

I dati arrivano infine all'ultimo addizionatore, che dopo un tempo di latenza ancora di 10 colpi di clock li restituisce in uscita.

La latenza totale del sistema è quindi data dalla somma di tutte le latenze, e cioè di 61. Quindi posso creare 61 catene separate di filtraggio. Bisogna ora determinare il numero massimo di filtri che può comporre ognuna di queste catene: il clock funziona a 1024 volte la frequenza di campionamento, quindi con una semplice divisione si trova che posso inserire 33 filtri per catena. In realtà i filtri sono 32, in quanto il router esterno prima di inviare i nuovi 61 dati in ingresso, vuole ricevere tutti e 61 i valori in uscita, in modo da poter resettare il filtro e ricominciare per un nuovo ciclo.

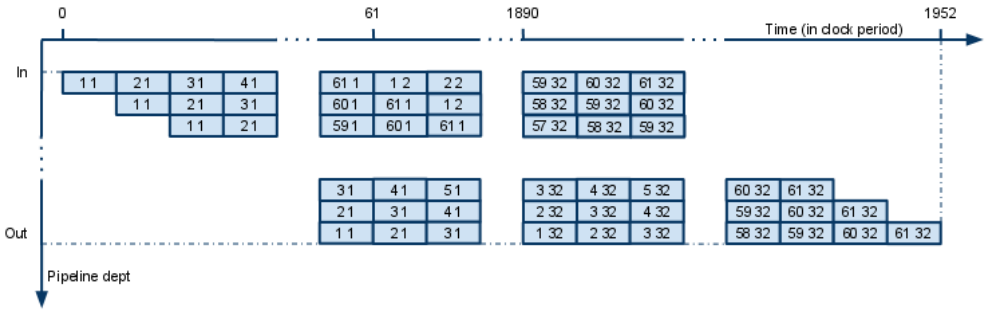

Figura 8.12 - Pipeline 2nd order filter

In Figura 8.12 si può vedere la struttura della pipeline del filtro schematizzata: per i primi 61 colpi di clock successivi al reset il sistema accetta gli ingressi che gli vengono forniti dal router esterno. Dal colpo di clock successivo in ingresso tornano i valori che hanno raggiunto l'uscita. In figura è

schematizzata la posizione dei dati all'inizio e alla fine del ciclo: la prima cifra indica la catena a cui appartiene il dato, il secondo valore indica il numero di filtro a cui è arrivata l'elaborazione dei dati.

8.6.1 All pass filter

Il cuore del filtro IIR del second'ordine è il filtro passatutto, di cui si tratta in termini teorici nel capitolo 7. La struttura fisica realizzata è schematizzata in Figura 8.13 e in Figura 8.14 .

In basso nelle due figure sono schematizzate le funzioni logiche, in alto la loro realizzazione pratica. Si può subito notare che i due macroblocchi sono praticamente identici: l'uscita del secondo sommatore del primo blocco va in ingresso al secondo, mentre l'uscita del secondo sommatore del secondo modulo rientra all'ingresso della memoria che realizza il ritardo nello stesso modulo.

Come si può notare osservando la catena di andata il ritardo introdotto dal primo blocco formato da adder, multiplier e registro è di 20 colpi di clock. Vengono quindi inseriti 20 registri per far arrivare i dati al medesimo colpo di clock all'adder in uscita, che introduce ulteriori 10 colpi di clock di ritardo. Un registro d'ingresso aggiunge un ritardo, e la latenza totale del sistema è di 31 colpi di clock. Lo stesso risultato lo si può ottenere con la catena di ritorno. Osservandola attentamente si capisce il motivo del registro d'ingresso, che serve a emulare il ritardo della *delay memory*. Successivamente 20 shift register ritardano i segnali, aspettando l'uscita del multiplier.

Come avveniva per la memoria contenente i coefficienti K nel modulo del filtro IIR, i moltiplicatori alfa e beta sono memorizzati in 2 *dual-port ram*, in ordine, in modo da poter semplicemente incrementare il campo indirizzo ad ogni colpo di clock. All'avvio del sistema tali memorie sono configurate con valori di default: sarà poi il microcontrollore ad occuparsi di inserire i valori del *preset* nel giusto ordine.

Anche la *delay memory* è una dual port ram, l'indirizzo di scrittura e l'indirizzo di lettura sono gestiti separatamente in modo da emulare il ritardo di un campione alla frequenza di campionamento.

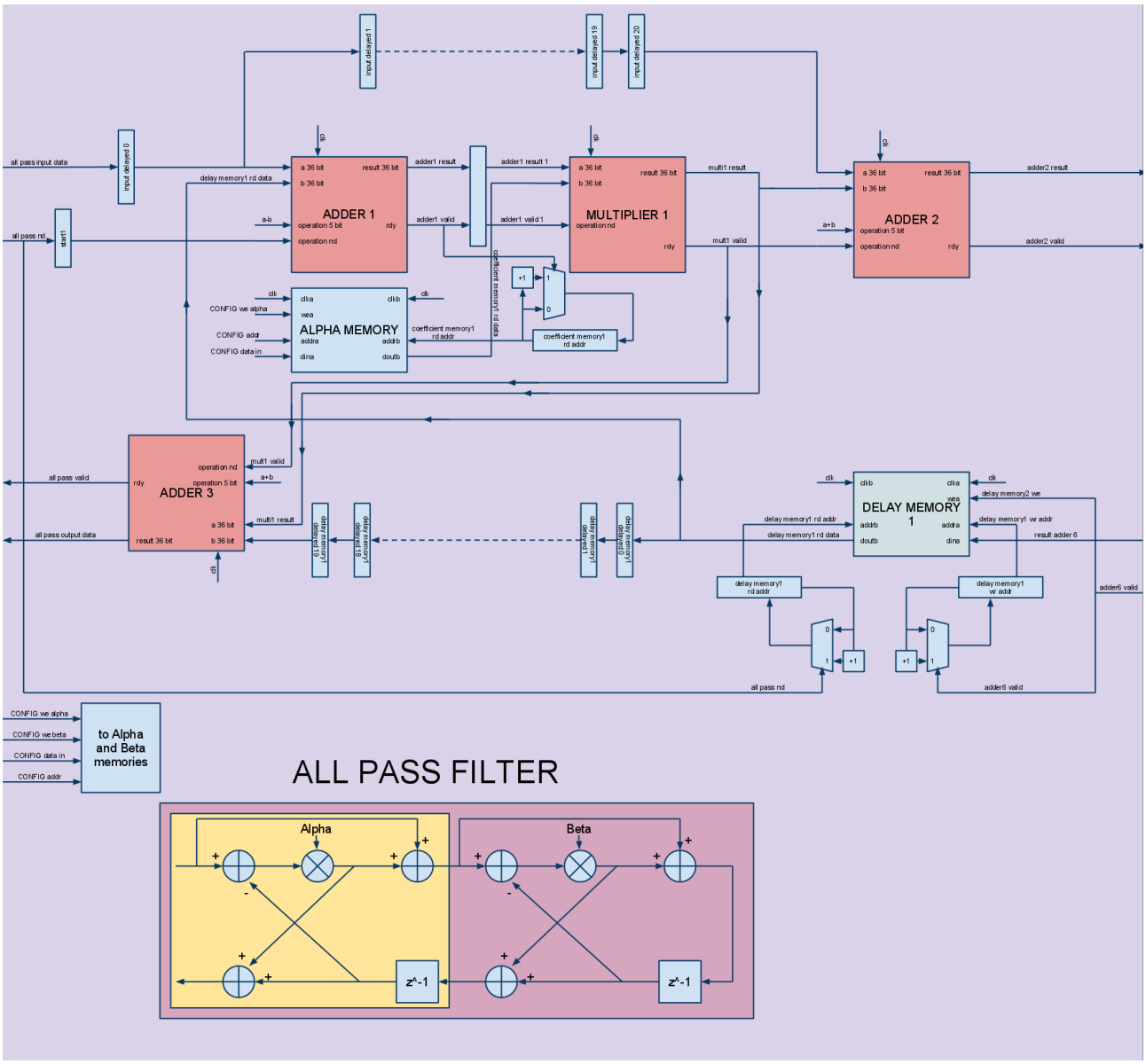

Figura 8.13 - All pass filter A

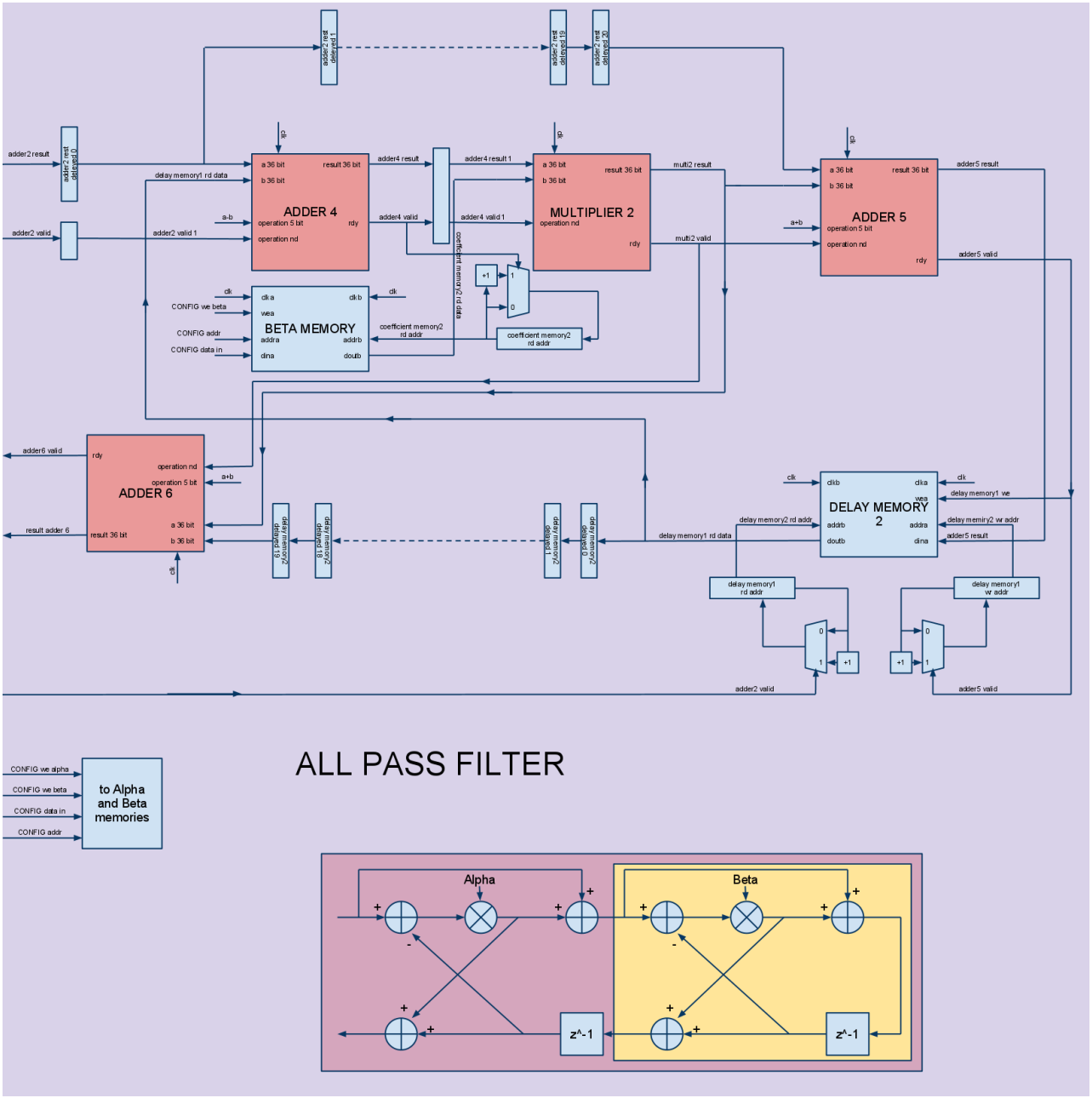

Figura 8.14 - All pass filter B

8.7 Gestione comunicazoni col microcontrollore

Il bus di comunicazione tra microcontrollore e FPGA è di tipo SPI. Il protocollo utilizzato è stato creato ad hoc per l'applicazione.

Ogni comando consta di 7 byte, 56 bit in totale. I primi 9 bit sono di controllo in modalità *one hot*. I primi 3 bit identificano le 3 memorie coefficienti, i 2 bit successivi identificano la memoria programma e la memoria coefficienti della alu: quando sono asseriti viene alzato il write enable della rispettiva memoria, gli 11 bit successivi vengono mandati nel bus indirizzo e viene quindi memorizzato il dato che corrisponde ai 36 bit successivi. Se il dato sia indirizzato alla memoria istruzioni solo gli ultimi 10 bit indirizzo vengono usati; nel caso invece il destinatario è la memoria costanti della alu, il numero di bit indirizzo scende a 8 e il numero di bit di dato va a 32. I bit inutillizzati vengono scartati a priori.

Il sesto bit di controllo identifica un cambio negli indirizzio di inizio e fine nella catena di delay. In questo caso cambia il protocollo: il bit successivo ai control bit è inutilizzato, i 4 bit seguenti identificano il numero di canale da andare a modificare (da 0 a 3 gli ingressi, da 4 a 11 le uscite). I successivi 21 bit identificano l'indirizzo di stop e gli ultimi 21 l'indirizzo di start. La procedura di sostituzione degli indirizzi è gestita direttamente dal blocco di generazione dei ritardi.

I bit di controllo non utilizzati sono stati previsti per le implementazioni future del crossover, del processore di dinamica e di tutte quelle comunicazioni minori (il *mute* ad esempio).

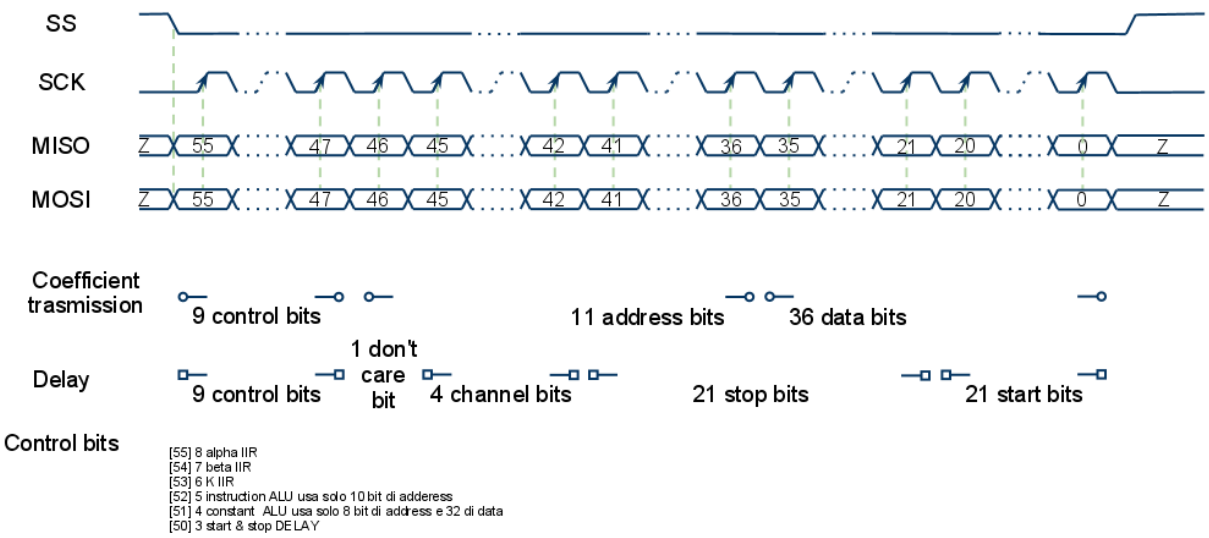

Figura 8.15 - Protocollo SPI di comunicazione

9. Struttura microcontrollore

In questo capitolo viene descritto il programma inserito nel microcontrollore.

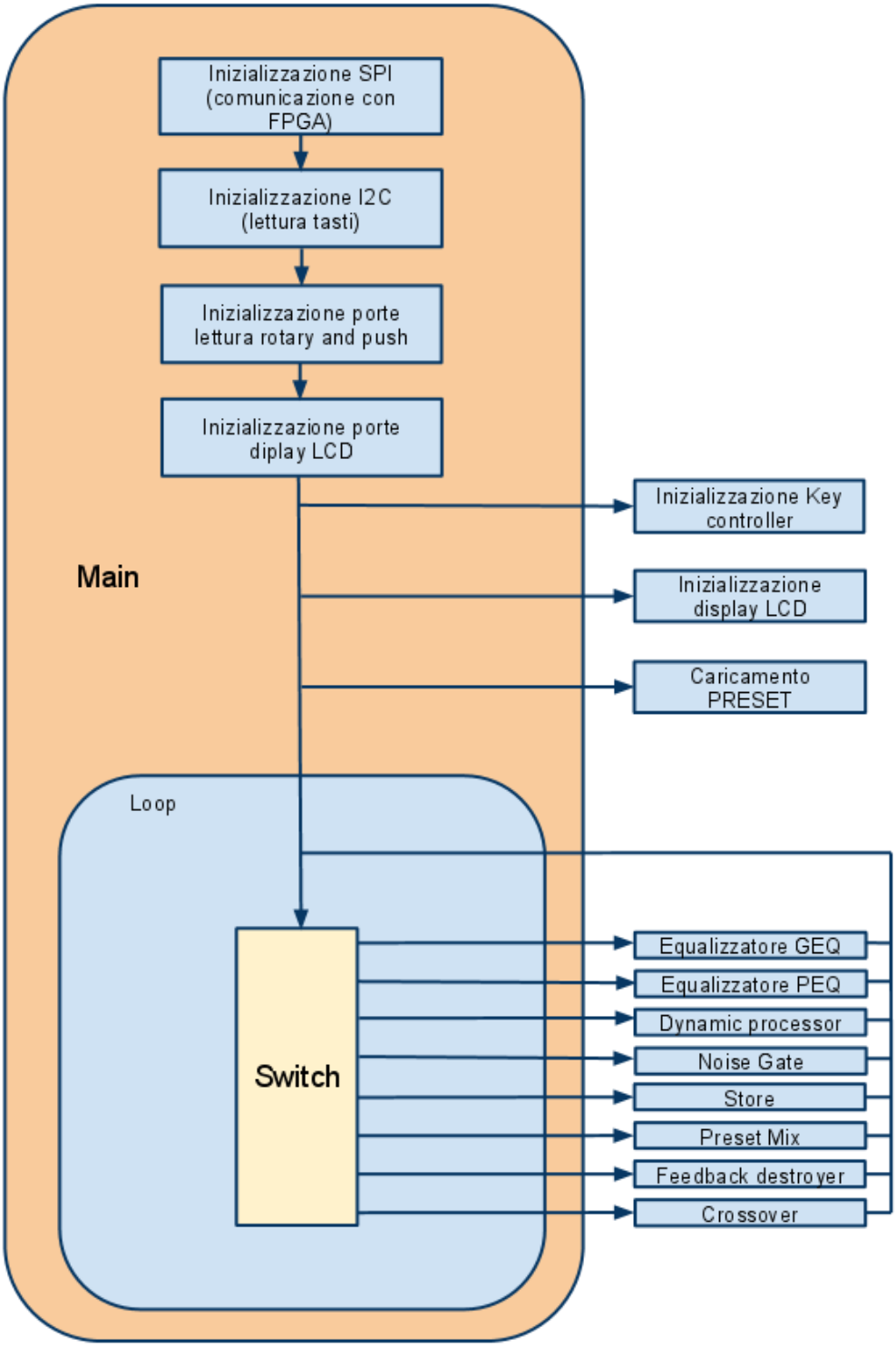

Figura 9.1 – Main

9.1 Main

La struttura implementata nel microcontrollore è completamente modulare. Questo per poter gestire più facilmente eventuali modifiche di singole parti, senza dover modificare tutto il programma. Per prima cosa, nel *main* vengono inizializzate tutte le periferiche. Si passa poi in un *loop* infinito che gestisce i passaggi tra i vari menù.

Come si può notare in Figura 9.1, le prime istruzioni attivano la periferica SSI0, che si occupa della comunicazione via SPI con l'FPGA, la periferica I2C0, che si occupa di andare a leggere il controller dei tasti. Vengono quindi predisposti come ingressi i pin preposti alla lettura del *rotary encoder* e del relativo tasto di *push*. Infine vengono attivate le porte collegate al display LCD. Vengono quindi inizializzate le periferiche esterne e viene caricato il preset attraverso funzioni esterne.

Finita la fase di inizializzazione, si entra in un loop infinito. Questo ciclo legge una variabile globale in cui è memorizzato il menù da visualizzare e richiama la funzione ad esso associata. All'avvio, il primo menù visualizzato è il *preset mix*. La routine di gestione di ogni fuunzione ha quindi una prima parte di inizializzazione del menù che si occupa di visualizzare sul display la schermata e di caricare i dati necessari. A questa fase segue solitamente un loop che, utilizzando il sistema del *polling* si mette in ascolto del *key controller*, del *rotary encoder* e del tasto di *push*, andando a tradurre in azioni concrete i vari input. Se in questa fase viene premuto un tasto il cui scopo è andare a visualizzare un altro menù la funzione termina, restituendo al main il valore del menù successivo.

9.2 Inizializzazione del Key controller

In fase di accensione il key controller (il MAX7347 di Maxim) è nello stato di *shutdown*. Attraverso il bus I2C, rispettando le regole di comunicazione descritte nel datasheet del componente, viene scritto il *register file* in modo da attivarlo. Successivamente vengono anche modificati valori di rilievo sulle caratteristiche di funzionamento: l'inibizione della protezione antirimbalzo viene impostata a 15ms, la frequenza di lettura a 4 Hz e l'*autorepeat* dei dati a 750ms.

9.3 Lettura tasti, rotary e push

Ogni menù, dopo le relative inizializzazioni, si pone in uno stato di attesa in cui le periferiche di interfacciamento esterno vengono interrogate continuamente da una funzione apposita, che a ogni ciclo di lettura restituisce un valore indicante l'azione compiuta o un valore di default se non è stato letto nulla.

9.3.1 Lettura del key controller

Delle tre procedure di lettura è la più complessa. Non utilizzando gli interrupt, per capire se è stato premuto un tasto viene interrogato un registro specifico del key controller. Il microcontrollore invia quindi una richiesta di lettura di tale registro, attraverso la periferica I2C. Se il bit che indica se effettivamente un tasto è stato premuto è asserito, si entra in un ciclo che va a leggere il registro in cui è contenuto il valore del tasto premuto, e successivamente analizza il risultato.

Se il tasto premuto è quello che identifica la seconda funzione si entra in un loop dal quale si esce solo quando viene premuto un altro tasto. Con lo stesso principio precedentemente esposto la funzione continua ad interrogare il controller. Quando viene letto un tasto vi sono due opzioni: se il tasto non possiede una seconda funzione si esce dal loop e il programma continua come se non fasse stato premuto nulla; se invece la seconda pressione è coerente, viene restituito il valore della funzione richiamata.

Se invece il primo tasto premuto è uno dei 12 mute, una sottofunzione interna si occupa di comunicare direttamente all'FPGA la situazione dei mute. Questa sottofunzione viene eseguita direttamente dalla routine di lettura dei tasti, in quanto qualunque sia il menù che ha richiamato la funzione di lettura tasti, l'operazione da eseguire è sempre la stessa.

Se il primo tasto premuto non rientra in queste due categorie, viene direttamente restituito il suo valore alla funzione principale.

9.3.2 Lettura del rotary controller

La lettura del rotary controller è molto semplice: il dispositivo è collegato attraverso due linee al microcontrollore. Ogni volta che vi è un movimento della manopola, una delle due linee cambia il suo stato. Il processo di lettura è quindi molto semplice e consiste in una macchina a stati che, in base allo stato precedente e allo stato attuale, restituisce la direzione di rotazione se è effettivamente avvenuta. In Figura 9.2 è mostrato lo schema concettuale della macchina a stati.

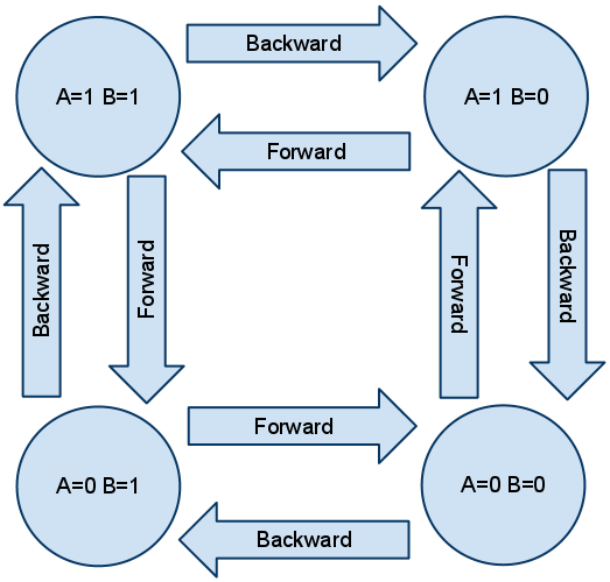

Figura 9.2 - Struttura Microcontrollore - Rotary routine

9.3.3 Lettura del tasto push

L'identificazione della pressione del tasto "push" è la parte più semplice dell'intera routine di lettura: in fase di riposo la linea in uscita dal dispositivo si trova alla tensione di alimentazione. Quando avviene la pressione, la linea viene cortocircuitata a massa. Basta quindi monitorare la linea tenendo memoria dello stato precedente: quando si verifica un passaggio da stato logico alto a basso, viene fornito il segnale di "push".

9.4 Gestione protocollo di comunicazione

I dati da inviare all'FPGA vengono gestiti da una routine di comunicazione seguendo il protocollo analizzato nel capitolo 8.7. La procedura si avvale del modulo SPI integrato nel microcontrollore. La difficoltà nella comunicazione dei parametri sta principalmente nella differente struttura dei dati: il microcontrollore infatti opera con parametri *floating point* a 64 bit, l'FPGA invece opera con numeri a 36 bit.

9.4.1 Conversione floating point da 64 a 36 bit

Quando si inviano dei coefficienti alle memorie bisogna fare i conti col fatto che i dati che viaggiano nel filtro IIR sono *floating point* a 36 bit con 7 bit di esponente. Non essendo uno standard, non si ha la possibilità di dichiarare variabili di questo tipo nel microcontrollore. Si è quindi scelto di lavorare con variabili di tipo *double* (che corrispondono al tipo *floating point* a 64 bit) , per poi eseguire la conversione quando vi è la necessita di trasmettere i dati.

Il *floating point* è così definito: il primo bit è di segno, seguito dai bit dell'esponente (11 nel caso dei *double*, 7 nel formato utilizzato sull'FPGA) con i restanti bit che svolgono il ruolo di mantissa (52 nei *double*, 28 per quanto riguarda l'FPGA). Bisogna poi ricordare che l'esponente, dopo essere calcolato, viene espresso con un offset positivo di $2^{N-1}-1$, dove N è il numero di bit dell'esponente. Il numero così ottenuto è quindi sempre positivo (a meno di eventuali *overflow* o *underflow*).

La conversione consiste quindi nel copiare il bit di segno e i primi 28 bit della mantissa. I bit di esponente vanno invece adattati, sottraendo 1023 e sommando 63. In Figura 9.3 è visualizzata l'operazione da compiere.

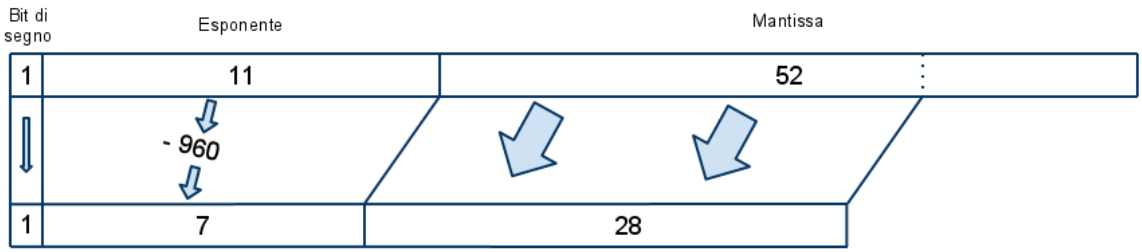

Figura 9.3 - Struttura microcontrollore - Conversione floating point da 64 a 36 bit

```
punt=(unsigned char*) &parametro;
for (c=0; c<8; c++)
        {
        vettore_double[c]=punt[c];
        }
aux=((vettore_double[7]<<4)&0x07F0)|((vettore_double[6]>>4)&0x000F);
aux=aux-960;
BYTE[0] = (control >> 1)&0xFF;
BYTE[1]=((control << 7)&0x80)|((address>>4)&0x7F);
BYTE[2]=((address<<4)&0xF0)|((vettore_double[7]>>4)&0X08)|((aux>>4)&0x07);
BYTE[3]=((aux<<4)&0X00F0)|((vettore_double[6])&0x000F);
BYTE[4]=((vettore_double[5])&0XFF);
BYTE[5]=((vettore_double[4])&0XFF);
BYTE[6]=((vettore_double[3])&0XFF);
```

Il codice diventa come quello riportato nella casella di testo: viene scritto in un puntatore l'indirizzo del double di riferimento; successivamente vengono copiati i byte della variabile in un

array di char in modo da poter agire sui singoli bit. Attraverso un sistema di *shift* e di mascherature, viene inserito nella variabile aux l'esponente. Successivamente si crea la stringa di 7 byte da inviare.

Nel codice d'esempio si vede chiaramente come nei primi 9 bit vengono messi i bit di control, successivamente l'indirizzo e poi il dato da trasmettere appena calcolato.

10. Conclusioni

10.1 FPGA, vantaggi e svantaggi

L'innovazione prodotta dal progetto di tesi sta nell'utilizzo dell'FPGA come fulcro del sistema. Nessun oggetto commerciale, infatti, propone questa soluzione. Il vantaggio enorme che porta con se questo tipo di approcio, in contrasto a quello classico che fonda le sue radici sull'uso di uno o più DSP, sta nella riduzione sostanziale di componenti. Utilizzando i DSP, infatti, si rende necessario l'uso di dispositivi di interfacciamento, capaci di gestire i vari tipi di segnali logici. Utilizzando un FPGA invece, è possibile utilizzare risorse interne a questo scopo.

A fronte di questi vantaggi principalmente economici, che emergono alla produzione di un numero elevato di oggetti, si presentano una serie di sfide ardue per i progettisti. La programmazione su FPGA in linguaggio VHDL è infatti molto più complicata e richiede tempi di progettazione e simulazione elevati.

Altro aspetto negativo lo si incontra in fase di processamento del segnale: la struttura di elaborazione dell'FPGA è infatti molto statica. Il filtraggio implementato non può essere variato sensibilmente in fase di utilizzo e ciò è un grosso svantaggio se si vogliono implementare filtri di natura diversa da quello realizzato. Non c'è da stupirsi infatti se, una struttura ideata per svolgere funzioni parallele, incontra problemi di staticità quando viene utilizzata per eseguire elaborazioni in maniera seriale.

10.2 Sviluppi futuri

La struttura così realizzata implementa una buona parte delle funzioni che un oggetto del genere deve supportare. Per rendere il sistema completo è però necessario implementare le funzioni di crossover e del processore di dinamica.

Per quanto riguarda il crossover sarà necessario sviluppare un sistema che modifichi, in maniera semplice, il filtro IIR realizzato, in modo da poter aggiungere un grado di libertà alla struttura (con l'aggiunta quindi di un coefficiente di caratterizzazione) in modo da poter implementare filtri di Butterworth e di Bessel.

La struttura del processore di dinamica è invece già realizzabile grazie al processore realizzato internamente all'FPGA. Bisognerà quindi studiare un algoritmo efficiente che lo realizzi.

Se si volesse invece cambiare filosofia, la cosa migliore da fare sarebbe quella di affiancare un FPGA a un DSP. Si potrebbe infatti pensare, di sfruttare i vantaggi di entrambi: da una parte abbiamo un dispositivo molto adatto al routing, che potrebbe eseguire solo le operazioni di interfacciamento con il mondo esterno e piccole elaborazioni (i ritardi e il mixing), dall'altra vi è un dispositivo molto flessibile e adatto all'elaborazione, che quindi può essere utilizzato per compiere i filtraggi IIR opportuni e per implementare il processore di dinamica.

Per la realizzazione di un prodotto completo, in futuro si potrebbe anche pensare di concentrarsi sugli aspetti di interfacciamento utente, in modo da fornire una consolle facile da usare, veloce nella sua configurazione e piacevole dal punto di vista estetico. Questi aspetti non sono stati ritenuti di grande importanza per un oggetto prototipale, ma che diventerebbero fondamentali in caso di una visione commerciale del prodotto.

Appendice A – Segnali digitali

A.I. Segnali digitali e operazioni

Per permettere a un sistema di elaborazione digitale (DSP) di manipolare un segnale analogico, il segnale deve essere campionato e convertito in formato binario. Quindi, un segnale digitale è definito come una sequenza di numeri funzione dell'indice di tempo n, che corrisponde al tempo nT se il segnale è stato campionato da un segnale analogico x(t) con un periodo di campionamento di T secondi.

Il periodo di campionamento può essere espresso come

$$T = \frac{1}{f_s} \tag{65}$$

Dove f_s è la frequenza di campionamento espressa in Hertz.

Si possono definire tre operazioni base che è possibile eseguire sui segnali per elaborarli.

La somma tra due segnali (di seguito vista matematicamente e per mezzo del diagramma a blocchi):

$$y[n] = x_1[n] + x_2[n] \tag{66}$$

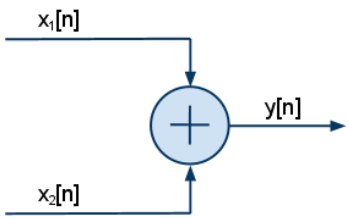

Figura A.1 - Somma

La moltiplicazione:

$$y(n) = \alpha x(n) \tag{67}$$

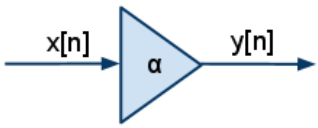

Figura A.2 - Moltiplicazione

Il ritardo

$$y[n] = x[n-k] \tag{68}$$

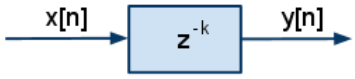

Figura A.3 - Ritardo

Queste tre operazioni possono essere combinate a creare l'equazione differenziale di I/O del DSP, che definisce matematicamente la relazione tra ingresso e uscita. Ad esempio posso scrivere:

$$y[n] = b_0 x[n] + b_1 x[n-1] \tag{69}$$

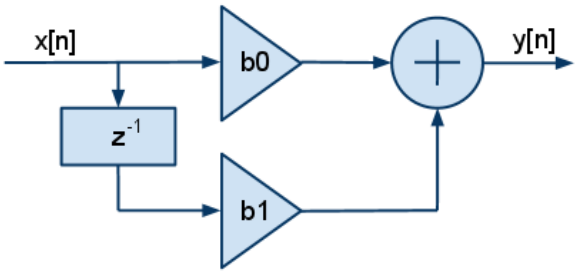

Figura A.4 - Schema a blocchi di un semplice sistema DSP

La definizione di energia di un segnale digitale è la seguente:

$$E_x = \sum_{n=-\infty}^{+\infty} |x[n]|^2 \tag{70}$$

Per un segnale periodico di periodo N, l'energia è infinita, ma può essere data la definizione di potenza:

$$P_x = \frac{1}{N} \sum_{n=0}^{N-1} |x[n]|^2 \tag{71}$$

A.II. La trasformata Z

La trasformata Z è una tecnica molto efficace per lo studio e l'analisi di segnali e sistemi digitali. È molto utile per rappresentare diagrammi a blocchi e funzioni di trasferimento.

La trasformata z di un segnale digitale x[n] è definita come

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]\, z^{-n} \tag{72}$$

Dove z è una variabile complessa. L'insieme dei valori z per i quali X(z) esiste è chiamata regione di convergenza. In generale, il valore di z può essere espresso in forma polare $z = re^{j\theta}$.

Proprietà

Per un segnale causale vale la seguente equazione

$$X(z) = \sum_{n=0}^{\infty} x[n]\, z^{-n} \tag{73}$$

Se un segnale y[n] è la versione ritardata di k campioni di x[n]

$$\begin{aligned} Y(z) = \sum_{n=-\infty}^{\infty} y[n]\, z^{-n} = \sum_{n=-\infty}^{\infty} x[n-k]\, z^{-(n-k)+k} = \\ = z^{-k} \sum_{m=-\infty}^{\infty} x[m]\, z^{-m} = z^{-k} X(z) \end{aligned} \tag{74}$$

Se ne ricava che un ritardo di k campioni nel dominio del tempo corrisponde alla moltiplicazione per z^{-k} nel dominio Z.

Se y[n] è la convoluzione lineare tra due sgnali nel dominio del tempo

$$y[n] = x[n] * h[n] = \sum_{i=-\infty}^{\infty} x[i]h[n-i] = \sum_{i=-\infty}^{\infty} h[i]x[n-i] \tag{75}$$

La trasformata Z di y[n] sarà data da

$$Y(z) = X(z)H(z) \tag{76}$$

La convoluzione nel dominio del tempo equivale, infatti, alla moltiplicazione nel dominio della trasformata.

A.III. Sistemi linari tempo invarianti

Si consideri il sistema digitale definito dalla seguente equazione

$$y[n] = b_0 x[n] + b_1 x[n-1] \tag{77}$$

Quando il segnale d'ingresso x[n] è l'impulso unitario (x[n]=1 per n=0 , x[n]=0 altrove), l'uscita assumerà la seguente forma:

$$y[n] = h[n] = \begin{cases} b_0 & per\ n = 0 \\ b_1 & per\ n = 1 \\ 0 & altrove \end{cases} \tag{78}$$

h[n] è chiamata "risposta all'impulso" di un sistema digitale

$$\begin{gathered} y[n] = x[n] * h[n] = \sum_{i=-\infty}^{\infty} x[i]h[n-i] = \\ = h[0]x[n] + h[1]x[n-1] = b_0x[n] + b_1x[n-1] \end{gathered} \tag{79}$$

Che è identica all'equazione di partenza. Ne consegue, quindi, che un sistema digitale può essere espresso in maniera identica sia con l'equazione alle differenze, sia con la risposta all'impulso.

Viene chiamata funzione di trasferimento di un sistema digitale la trasformata Z di h[n]

$$H(z) = b_0 + b_1z^{-1} = \frac{Y(z)}{X(z)} \tag{80}$$

Si consideri ora il sitema descritto nell'equazione (76) nel quale x[n] è combinazione lineare di due segnali

$$x[n] = a_1x_1[n] + a_2x_2[n] \tag{81}$$

Ne consegue che

$$\begin{aligned} y[n] &= b_0x[n] + b_1x[n-1] \\ &= b_0(a_1x_1[n] + a_2x_2[n]) + b_1(a_1x_1[n-1] + a_2x_2[n-1]) \\ &= a_1(b_0x_1[n] + b_1x_1[n-1]) + a_2(b_0x_2[n] + b_1x_2[n-1]) \\ &= a_1y_1[n] + a_2y_2[n] \end{aligned} \tag{82}$$

Dove $y_1[n]$ e $y_2[n]$ sono le uscite del sistema dovute agli ingressi $x_1[n]$ e $x_2[n]$. Un sistema che soddisfa questa proprietà è chiamato *lineare.*

Definito un operatore Ơ tale che

$$y[n] = Ơ(x[n]) \tag{83}$$

Si dice che il sistema è *tempo invariante* rispetto all'operazione Ơ se avviene che

$$y[n-k] = \mathcal{O}(x[n-k]) \tag{84}$$

Un sistema che soddisfa entrambe queste ultime due proprietà è chiamato sistema *lineare tempo invariante* (LTI).

A.IV. Filtri digitali

Un filtro digitale è un sistema combinazione di operazioni lineari. Il filtraggio è una delle operazioni più comuni tecniche usate nell'elaborazione di segnali. Un filtro permette a certe componenti in frequenza di un segnale di passare attraverso il sistema senza essere modificate, mentre blocca altre componenti non volute.

I filtri digitali si dividono in classi: filtri con risposta all'impulso finita (FIR) e filtri con risposta all'impulso infinita (IIR).

I filtri FIR si chiamano così, perché, posto in ingresso un impulso ad un istante n, l'uscita del filtro va a zero in un numero definito L di campioni. La funzione nel dominio del tempo che caratterizza questi filtri è del tipo:

$$y[n] = \sum_{k=1}^{N} b_k x[n-k] \tag{85}$$

Mentre la funzione di trasferimento associata è la seguente

$$H(z) = b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_N z^{-N} = \sum_{k=1}^{N} b_k z^{-k} \tag{86}$$

Ciò che più interessa in questa trattazione, però, sono i filtri IIR. La risposta all'impulso di questi filtri è una sequenza infinita. La funzione di trasferimento associata a questo tipo di filtro è nella forma:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_M z^{-M}} = \frac{\sum_{k=1}^{N} b_k z^{-k}}{1 + \sum_{i=1}^{M} a_i z^{-i}} \tag{87}$$

A cui è associata la forma nel dominio del tempo:

$$y[n] = \sum_{k=1}^{N} b_k x[n-k] - \sum_{i=1}^{M} a_k x[n-i] \tag{88}$$

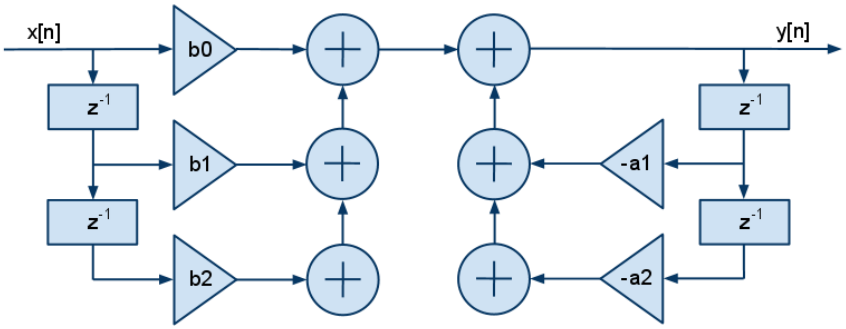

Figura A.5 - Realizzazione in forma diretta I

Il diagramma di flusso in Figura A.5 illustra l'equazione (87) utilizzando la forma diretta I.

Uno degli aspetti principali nella progettazione di un filtro, è la ripidità

In generale, il vantaggio primario nell'uso dei filtri IIR, rispetto ai filtri FIR, è quello di poter ottenere frequenze di taglio con caratteristiche più ripide a parità di ordine, riducendo la complessità hardware. Questo a scapito di una fase non lineare e problemi relativi alla precisione finita dei calcolatori, che danno luogo ai così detti errori di *round-off*.

Il metodo più comunemente usato per ricavare il filtro digitale, consiste nel partire dalla funzione di trasferimento analogica nel dominio della trasformata S e applicare la trasformazione bilineare. Questa trasformazione avviene sostituendo in H(s) al posto della variabile s la seguente

$$s = \frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}} \tag{89}$$

Dove T è il periodo di campionamento. La comodità nell'uso di questo tipo di trasformazione, sta nella garanzia di stabilità del filtro di arrivo, se il filtro analogico di partenza è stabile. Lo svantaggio è una discrepanza di guadagno del filtro per frequenze vicine a $\frac{f_s}{2}$.

A.V. Analisi in frequenza

Definiamo la *trasformata di fourier tempo-discreta* (DTFT) come

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n} \tag{90}$$

Ne consegue direttamente che la DTFT è periodica di periodo 2π. Similmente, si può pensare di valutare la funzione di trasferimento di un sistema tempo discreto lungo il cerchio di raggio unitario descritto nel piano Z, per rappresentare la risposta in frequenza del sistema come

$$H(\omega) = H(z)|_{z=e^{j\omega}} \tag{91}$$

Da cui si possono ottenere informazioni sul modulo e la fase del sistema di riferimento.

Appendice B – Protocolli di comunicazione

B.I. I2S (Integrated Interchip Sound)

L'I2S è uno standard che definisce un protocollo di comunicazione per trasmettere flussi audio *on board*. Il bus consiste in 3 linee: la linea di clock, la linea di *word select* (anche chiamata *left/right clock*) e almeno una linea dati. La frequenza di clock generalmente è 64 volte la frequenza di campionamento del segnale audio da trasmettere.

Lo stato del *left/right clock* definisce quale canale è in trasmissione sul bus dati. La sincronia generale avviene proprio sulle discontinuità di questo clock, in quanto, sul fronte di salita del clock principale successivo a una commutazione del LRCK, sarà disponibile sulla linea dati il MSB del segnale audio. Il protocollo prevede 32 bit di dato per ogni canale, anche se usualmente se ne usano 24 per l'audio professionale.

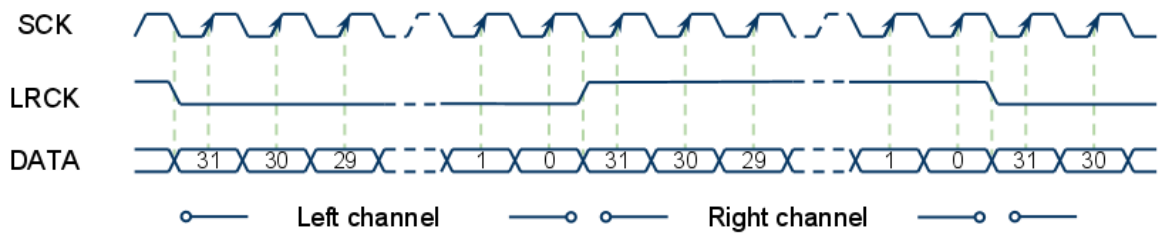

Figura 0.1 - Diagramma temporale I2S

Vi sono tre configurazioni possibili come mostrato in Figura 0.2:

- Trasmettitore master, ricevitore slave. Il trasmettitore provvede a generare i clock.
- Ricevitore master, trasmettitore slave. È il ricevitore che comanda i sincronismi.
- Ricevitore e trasmettitore entrambi slave. Un controller esterno si occupa di fornire i segnali di clock.

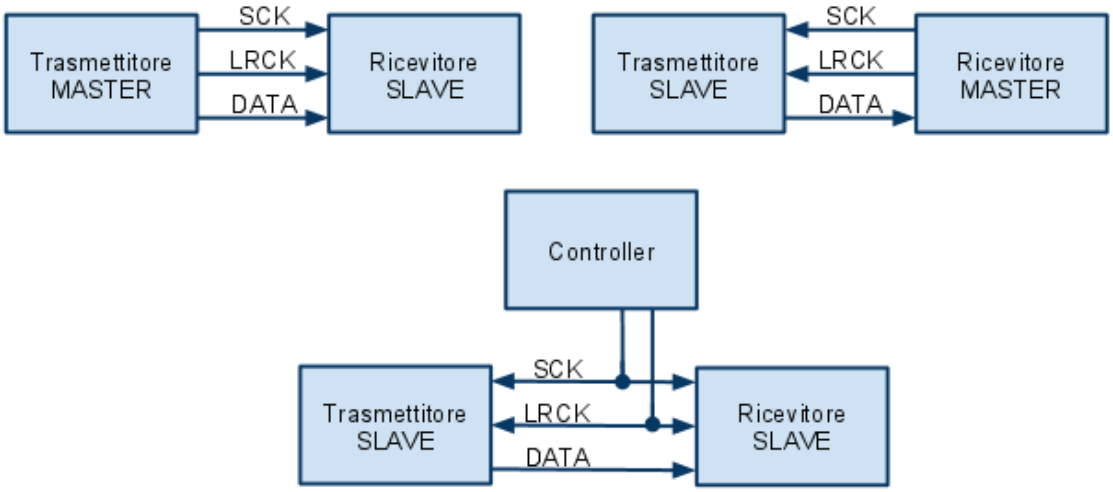

Figura 0.2 - Configurazioni I2S

B.II. SPI (Serial Peripheral Interface)

Il bus SPI è uno standard di comunicazione sincrono a quattro fili, che opera in modalità *full duplex*. I dispositivi comunicano in configurazione master o slave, e sono ammessi slave multipli.

Una linea di *Slave Select* (SS) per ogni periferica, identifica il dispositivo a cui è consentito comunicare.

Una linea di *Serial Clock* (SCK) in uscita dal modulo master detta i sincronismi al bus.

Infine vi sono due linee dati: la *Master Output, Slave Input* (MOSI), utilizzata dal master per comunicare informazioni agli slave, e la *Master Input, Slave Output* (MISO), su cui viaggiano le informazioni dagli slave al master.

Quando bus è in *stand-by* il master lascia ferma la linea di clock e tiene al valore logico alto lo Slave Select. Quando deve avvenire la comunicazione, l'SS del dispositivo *slave* prescelto viene abbassato e successivamente viene fornito il segnale di clock.

Vi sono quattro possibili modalità di funzionamento, identificate da due parametri: il CPOL e il CPHA. Se il CPOL è uguale a 0, in situazione di stand-by il clock è a valore logico basso. In questo caso, se il CPHA è uguale a 0 il dato viene letto dai due dispositivi durante il *rising-edge*, mentre viene imposto sulla linea durante il *falling-edge*. In caso contrario il dato è imposto durate il *rising-edge* e viene letto sul *falling-edge*.

In caso di CPOL uguale a 1, in situazione di riposo il clock assume valore logico alto. L'asserzione e la lettura del dato in base al CPHA è inversa rispetto al caso precedente.

In Figura 0.3 è mostrato, a titolo di esempio, come avviene la trasmissione nel caso in cui CPOL=0 e CPHA=0.

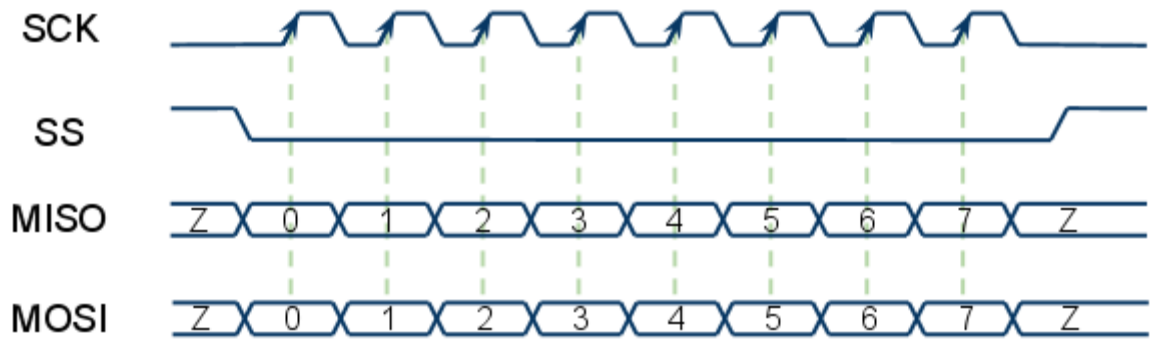

Figura 0.3 - Diagramma temporale SPI, CPOL=0, CPHA=0

Come si può vedere in Figura 0.4 vi sono due modalità principali di interconnessione: la prima (mostrata a sinistra), molto semplice, associa uno *slave select* per ogni *slave* presente: il master comunica direttamente con la periferica prescelta. Nella seconda lo *slave select* è unico, la linea in uscita dal master va al primo dispositivo, quella in uscita dal dispositivo va al secondo, e così via, finchè la linea in uscita dall'ultimo dispositivo torna al master. La comunicazione in questo caso è

più complicata: è divisa in due fasi che si alternano ripetutamente. Durante la prima fase i moduli ricevono il dato in ingresso e comunicano in uscita il proprio dato. Nel periodo successivo si limitano a ritrasmettere quello che avevano ricevuto nella prima fase, e a ricevere quindi quello che era stato ricevuto dal dispositivo precedente. Il protocollo di incapsulamento dei dati risulta quindi alquanto complesso, dovendo specificare al suo interno, quale modulo è interessato da una determinata informazione.

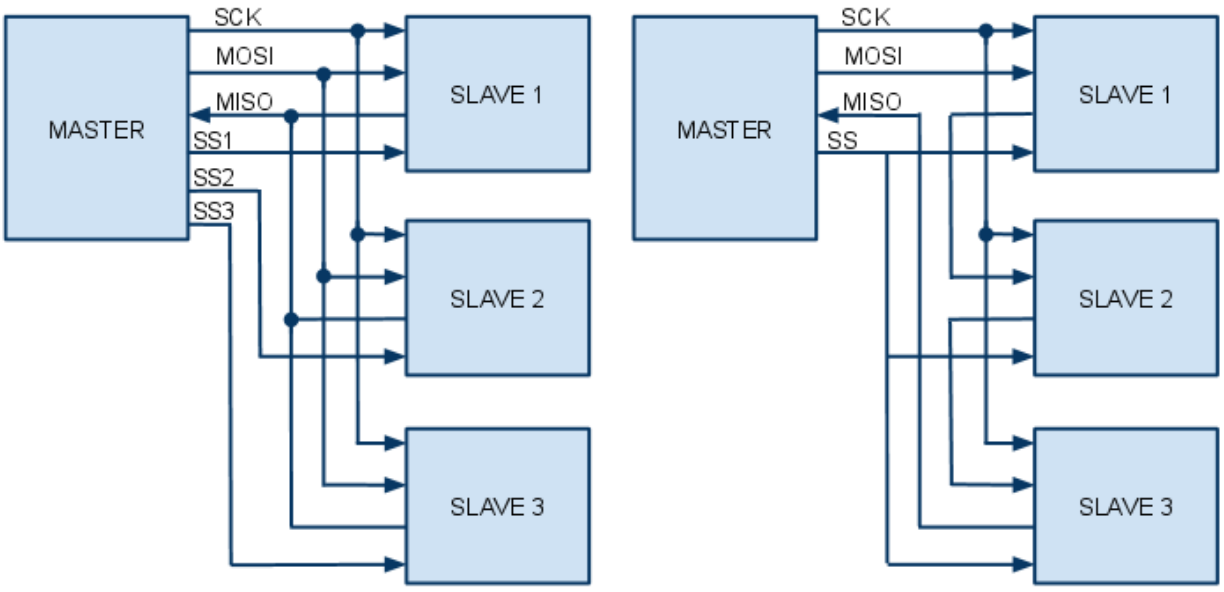

Figura 0.4 - Configurazione a slave indipendenti a sinistra, cooperativi a destra

I vantaggi nell'utilizzo di tale protocollo sono molteplici: innanzi tutto è uno standard che ammette la trasmissione e la ricezione contemporanea, quindi con un innalzamento del *throughput* (rispetto al principale concorrente che può essere identificato nel protocollo I2C); inoltre è molto versatile e consente di trasmettere parole di qualsivoglia lunghezza e contenuto; è molto semplice da implementare e ha un consumo ridotto, in quanto non necessita di complicati circuiti di ricezione; infine non è necessario fornire un indirizzo univoco a ogni periferica che viene connessa al bus.

Gli svantaggi sono invece limitati sostanzialmente all'uso di quattro linee, al posto delle 2 presenti nell'I2C, e a una minore flessibilità in termini di sistema: l'aggiunta di una periferica, infatti, comporta l'aggiunta di una linea di slave select, cosa non necessaria quando ad essere utilizzato è il suo antagonista.

B.III. I2C (Inter Integrated Circuit)

L'I2C è uno standard di comunicazione seriale a due linee. Ogni componente che si connette al bus ha un indirizzo unico. Il dispositivo che incomincia la comunicazione è il master, che si occupa di gestire il segnale di clock. La struttura del protocollo permette di creare una struttura multimaster (Figura 0.5), in cui, se la linea non è già utilizzata per un'altra trasmissione, ogni dispositivo può prenderne il controllo e diventare master. Generalmente, però, questo bus di comunicazione viene usato con un master solo e più slave.

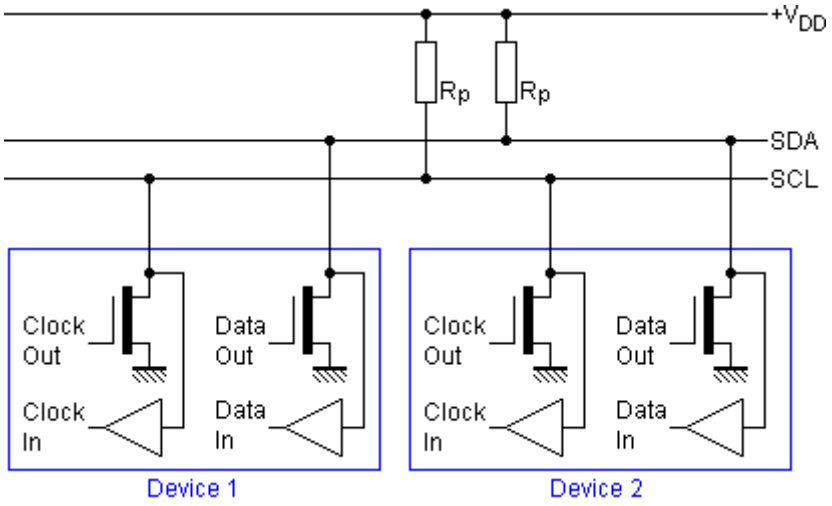

Figura 0.5 - Schema connessioni I2C

Lo standard definisce anche la velocità di trasmissione, che può assumere 3 valori:

- 100kbps Standard Mode
- 400kbps Fast Mode
- 3.4Mbps High Speed Mode

Le due linee, una di dato e una di clock, hanno un pull-up resistivo, quindi a riposo entrambe le linee sono alla tensione di alimentazione.

Quando il master vuole iniziare a comunicare, impone sulla linea la *start condition*, che consiste nell'imporre lo 0 sulla linea dati mentre la linea di clock è ancora a 1. Questa condizione, come quella di stop, è univocamente identificabile, in quanto durante la trasmissione dei dati, le transizioni avvengono sempre quando il clock è a livello logico basso.

Successivamente alla condizione di start, viene abbassata anche la linea di clock e imposto il primo bit sul canale dati. Ogni qual volta il clock va a livello logico basso viene posto un bit sul bus dal *master*. Il bit viene letto dagli slave quando il clock è alto. Il primo byte, trasmesso sempre dal master, è composto da 7 bit di indirizzo e 1 bit che indica al dispositivo indirizzato se l'operazione che si vuole svolgere è di lettura (1) o di scrittura (0).

Al colpo di clock successivo la trasmissione dell'ottavo bit, il master mette la linea dati in *tristate*. Il dispositivo che si riconosce nell'indirizzo trasmesso, a questo punto, impone lo zero sulla linea dati. Questo bit si chiama di *acknowledge*, e indica al master che la trasmissione è andata a buon fine. I moduli che non si riconoscono nell'indirizzo trasmesso dal master, invece, aspettano senza fare nulla fino alla ricezione di una condizione di stop.

Al successivo colpo di clock inizia la trasmissione dei dati, o da parte del master o da parte dello slave, in base al bit di read/write. Il clock invece è sempre imposto dal master. Dopo ogni 8 bit vi è sempre un colpo di clock dedicato all'acknowledge, in modo da identificare eventuali errori di trasmissione. Il numero di byte che si possono trasmettere è totalmente arbitrario.

Quando la trasmissione deve concludersi il master impone la condizione di stop, che consiste nell'alzare il bus dati mentre il clock è alto. Lo schema in Figura 0.6 può aiutare visivamente a capire il funzionamento del protocollo.

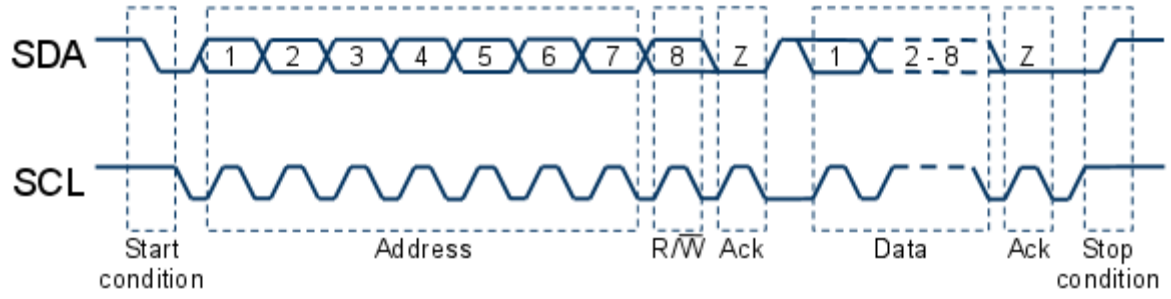

Figura 0.6 - Diagramma temporale

Il vantaggio principale nell'uso di questo protocollo, sta nella struttura hardware molto semplice. Con solo due linee, si possono interfacciare fino a 128 componenti, sia master che slave. Inoltre l'aggiunta di una periferica sulla linea non comporta nessuna modifica al resto della rete di comunicazione, il che lo rende un protocollo particolarmente flessibile, molto comodo soprattutto in fase di prototipazione.

Gli svantaggi principali invece sono la procedura di comunicazione abbastanza complessa e il non poter effettuare comunicazioni full duplex.

Bibliografia

1. **Philips Semiconductors.** *I2S bus specification.* 1986.

2. **European Broadcasting Union.** *Specification of the digital audio interface (The AES/EBU interface).* Grand-Saconnex (Geneva) Switzerland : s.n., 2005. Tech. 3250-E - Third edition.

3. *The digital all-pass filter: a versatile signal processing building block.* **Regalia, Phillip A., Mitra, Sanjit K. e Vaidyanathan, P. P.** 1988. Proceedings of the IEEE. Vol. 76.

4. **Texas Instrument.** Stellaris® LM3S9B90 Microcontroller - Data sheet. [Online] http://www.luminarymicro.com/products/lm3s9b90.html.

5. **Kuo, Sen M. e Gan, Woon-Seng.** *Digital Signal Processors - Architectures, Implementations and Applications.* Upper Saddle River : Pearson, 2005.

6. **Zolzer, Udo.** *Digital Audio Signal Processing.* s.l. : John Wiley and Sons, 2008.

7. **Zölzer, Udo, et al., et al.** *Digital Audio Effects.* s.l. : John Wiley & Sons, 2002.

8. **Watkinson, John.** *An introduction to digital audio.* s.l. : Focal Press, 2002.

9. **Zwolinski, Mark.** *VHDL. Progetto di sistemi digitali.* s.l. : Pearson Paravia Bruno Mondad, 2007.

10. **Micron.** MT45W2MW16PAFA-85 WT PSRAM Data Sheet. [Online] http://download.micron.com/pdf/datasheets/psram/Async%20CellularRAM_16_32.pdf.

11. **Xilinx.** Spartan 6 Data Sheet. [Online] http://www.xilinx.com/support/documentation/data_sheets/ds162.pdf.

12. **Texas Instuments.** SRC4184. [Online] http://focus.ti.com/lit/ds/symlink/src4184.pdf.

13. **Xilinx.** Audio/Video Connectivity Solution for Virtex-II Pro and Virtex-4 FPGAs. [Online] 15 10 2008.