UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Corso di Laurea Triennale in Ingegneria Elettronica

Realizzazione di un effetto digitale Octaver in real-time su processore ADSP-SC589

Implementation of a real-time digital Octaver effect on the ADSP-SC589 processor

Tesi di Laurea di:
Gabriele Di Carlo

Relatore: Chiar.mo
Prof. **Leonardo Gabrielli**

Correlatore:
Prof. **Stefano Squartini**

Anno Accademico 2023/2024

A Stitch e Nala

Ringraziamenti

Innanzitutto desidero ringraziare i professori Leonardo Gabrielli e Stefano Squartini per il supporto e la disponibilità dimostrati in questo periodo.
Ringrazio la mia famiglia, gli amici, i colleghi e tutti coloro che hanno fatto parte di questo percorso.

Gabriele Di Carlo

Abstract

La presente tesi ha come obiettivo lo studio e l'implementazione di un effetto Octaver digitale per chitarra in real-time sulla piattaforma Analog Device SHARC SC589 tramite l'ambiente di sviluppo proprietario "SigmaStudio+". Questo ambiente di sviluppo si pone come alternativa ai classici IDE proponendo una programmazione a blocchi, più semplice ed intuitiva rispetto una programmazione puramente testuale. Inizialmente viene descritto il funzionamento dell'effetto Octaver e ne vengono definite le specifiche da rispettare. Una volta introdotto l'effetto Octaver viene discusso il suo design teorico seguito da una implementazione in ambiente MATLAB. Dopo aver verificato il funzionamento dell'effetto progettato ed aver individuato e risolto i problemi riscontrati tramite delle simulazioni in ambiente MATLAB, viene descritta la procedura seguita per l'implementazione dell'effetto sulla piattaforma DSP SHARC SC589. In particolare vengono evidenziati gli aspetti riguardanti il funzionamento dell'ambiente di sviluppo SigmaStudio+, la programmazione della scheda SHARC Audio Module attraverso questo ambiente ed il porting dell'algoritmo MATLAB in SigmaStudio+. Infine vengono riportati e discussi i risultati ottenuti dall'implementazione dell'effetto Octaver real-time confrontandoli con l'implementazione MATLAB.

Indice

Elenco delle figure iii

1 Introduzione 1
- 1.1 Obiettivi della tesi . . . 2
- 1.2 Struttura della tesi . . . 3

2 Studio dell'effetto Octaver 5
- 2.1 Descrizione dell'effetto e delle sue applicazioni . . . 5
- 2.2 Requisiti e specifiche tecniche . . . 5
- 2.3 Design dell'effetto . . . 5
  - 2.3.1 Generazione delle ottave . . . 7
  - 2.3.2 Stima del pitch . . . 9
  - 2.3.3 Filtraggio tempo variante . . . 11
  - 2.3.4 Schema generale dell'effetto . . . 12

3 Simulazione in ambiente MATLAB 14
- 3.1 Algoritmo di base . . . 14
- 3.2 Risultati della simulazione . . . 16
  - 3.2.1 Chirp . . . 16
  - 3.2.2 Sample di chitarra . . . 16
  - 3.2.3 Osservazioni . . . 17
- 3.3 Problemi riscontrati e soluzioni . . . 19

4 Implementazione real-time 23
- 4.1 Piattaforma DSP SHARC SC589 . . . 23
- 4.2 SigmaStudio+ . . . 24
- 4.3 Configurazione del sistema . . . 24
  - 4.3.1 Hardware . . . 25
  - 4.3.2 Software . . . 26
- 4.4 Trasferimento dell'algoritmo . . . 29
  - 4.4.1 Creazione dei blocchi non presenti . . . 30
  - 4.4.2 Implementazione . . . 34
- 4.5 Problemi riscontrati e soluzioni . . . 34
  - 4.5.1 Problema nell'implementazione del blocco di stima del pitch . . . 35
  - 4.5.2 Problema dei volumi delle ottave generate . . . 37

5 Test e validazione dell’implementazione real-time **39**
5.1 Metodologia di test 39
5.2 Risultati dei test 39
5.2.1 Chirp 39
5.2.2 Confronto con simulazione MATLAB 40

6 Conclusioni **42**

7 Appendice **44**
7.1 Codice MATLAB 44
7.2 Codice Algorithm Designer 45

Bibliografia **46**

Elenco delle figure

1.1 Schema di collegamento di un effetto per chitarra 1
1.2 Effetti Octaver commerciali: a sinistra "OC-5" prodotto da Boss, a destra "Sub'n'up" prodotto da tc electronic 2

2.1 Spettrogramma di alcune note suonate da una chitarra $f_s = 48kHz$ 6
2.2 Analisi della generazione delle ottave da un tono puro $f_0 = 1kHz$ 8
2.3 Confronto dei due metodi di generazione dell'ottava inferiore $f_s = 48kHz$ 8
2.4 Analisi della funzione differenza media normalizzata (2.3) applicata in una finestra $N = 1024$ $N_l = 600$ del sample di chitarra in figura 2.1 all'istante $t = 2.4s$ $f_s = 48kHz$ 10
2.5 Schema a blocchi di un filtro passa banda basato su sistemi all-pass 11
2.6 Risposta in frequenza filtro passa banda $f_c = 0.5\frac{f_s}{2}$ e $fb = 0.1\frac{f_s}{2}$ $(Q = 5)$. 12
2.7 Struttura di base dell'effetto 13

3.1 Funzione MATLAB del filtro passa banda tempo variante 15
3.2 Spettrogramma del segnale in ingresso (chirp) 17
3.3 Spettrogramma del segnale in uscita (chirp) 17
3.4 Spettrogramma del segnale in ingresso (sample chitarra). 18
3.5 Spettrogramma del segnale in uscita `windowL=1024` `yinThreshold=0.22` (sample chitarra). 18
3.6 Spettrogramma del segnale in uscita `windowL=512` `yinThreshold=0.22` (sample chitarra). 18
3.7 f_0 nel tempo `windowL=1024` (chirp). 19
3.8 f_0 con filtro passa basso (chirp). 20
3.9 Segnale in uscita con filtro passa basso su f_0 (chirp). 20
3.10 f_0 nel tempo `yinThreshold=0.22` (sample chitarra). 21
3.11 f_0 nel tempo con `yinThreshold=0.01/0.1/0.22`, `windowL=1024` e filtro passa basso (sample chitarra). 21
3.12 f_0 nel tempo con `yinThreshold=0.01/0.1/0.22`, `windowL=512` e filtro passa basso (sample chitarra). 21
3.13 Segnale in uscita con `yinThreshold=0.1` e filtro passa basso, `windowL=512` (sample chitarra). 22

4.1 Sharc Audio Module [1] 23
4.2 Interfaccia utente SigmaStudio+ 24
4.3 ADZS ICE-1000 25

4.4 EVAL-ADUSB2EBZ (USBi) 25
4.5 Routing Scheme dello SHARC Audio Module [2] 28
4.6 Impostazione delle SPORT per lo SHARC Audio Module 28
4.7 Schema a blocchi dell'implementazione real-time 29
4.8 Generazione dell'onda quadra a frequenza $\frac{f_0}{2}$ 31
4.9 Memorizzazione dei campioni all'interno dell'algoritmo di stima del pitch . 33
4.10 Schema dell'effetto Octaver implementato in SigmaStudio+ 34
4.11 Risultato della prima implementazione dell'effetto Octaver real-time con la stima effettuata su 512 campioni 35
4.12 Schema della seconda implementazione su SigmaStudio+ dell'effetto Octaver 36
4.13 Compressione del livello dei toni generati 37
4.14 Test con schema rappresentato in figura 4.12 e `yinThreshold=0.22` 38
4.15 Test con schema rappresentato in figura 4.12 e `yinThreshold=0.1` 38
4.16 Test con schema rappresentato in figura 4.12 e `yinThreshold=0.01` 38

5.1 Effetto applicato ad un chirp sinusoidale $f_0 = [100Hz - 1500Hz]$ 40
5.2 Spettrogramma del segnale in ingresso sample di chitarra 41
5.3 Effetto MATLAB applicato al sample di chitarra 41
5.4 Effetto real-time applicato al sample di chitarra 41

7.1 Algoritmo MATLAB dell'effetto Octaver 44
7.2 Funzione SigmaStudio del filtro passa banda tempo variante 45

1 Introduzione

L'elettrificazione della chitarra è probabilmente la più importante modifica che lo strumento abbia subito nel XX secolo. Questa modifica ha dato vita a uno strumento elettromeccanico che trasforma la vibrazione meccanica delle corde in un segnale elettrico attraverso dei pickup elettromagnetici. Questa nuova struttura dello strumento introduce possibilità di elaborazione del suono altrimenti impossibili da ottenere. Infatti, il segnale elettrico generato può essere elaborato in modo analogico o digitale prima di essere riprodotto tramite un amplificatore, permettendo l'applicazione di una varietà infinita di effetti sul suono della chitarra [3].

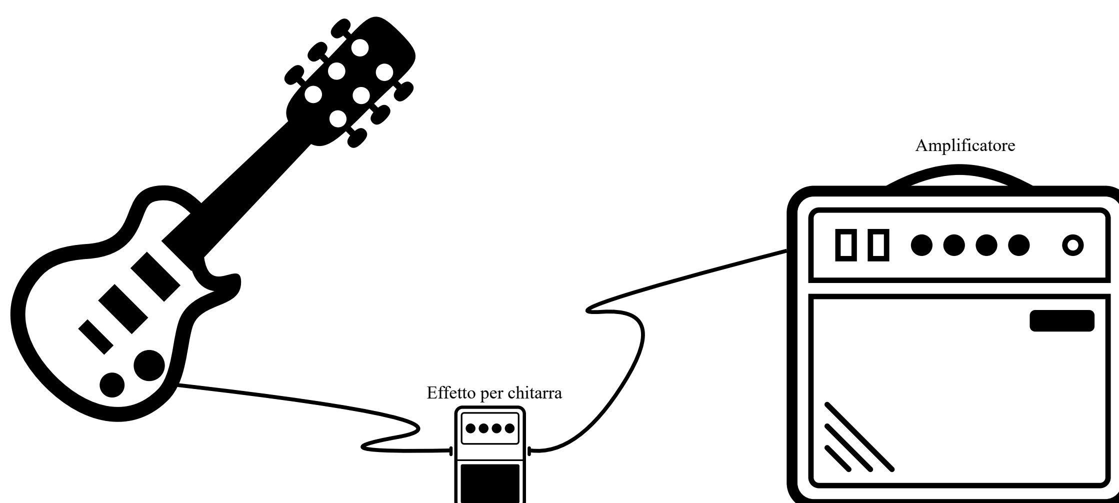

Figura 1.1: Schema di collegamento di un effetto per chitarra

Tra questi effetti, si è preso in esame per questa tesi l'effetto Octaver. Questo effetto, ideato negli anni '60 permette di aggiungere dei toni al segnale originale della chitarra a distanza di ottave, estendendo così la sonorità dello strumento. In questo modo, si possono

ricreare le note tipiche di un basso o delle note più alte per sonorità particolari. Questo effetto venne inizialmente creato attraverso una elaborazione analogica del segnale.
Con l'avanzare delle tecnologie, i produttori hanno creato effetti Octaver sempre più avanzati e con l'introduzione dei microcontrollori, si è passati a effetti digitali, come il BOSS OC5 [4] e il TC Electronic SUB 'N' UP [5], che rappresentano lo stato dell'arte attuale degli effetti Octaver per chitarra. Questi dispositivi implementano funzioni polifoniche, quindi funzionanti anche con accordi e algoritmi di tracking delle note avanzati. Intoltre nel caso del SUB'N'UP di *tc electronic*, viene offerto anche un controllo dell'effetto tramite un'applicazione smartphone.

Figura 1.2: Effetti Octaver commerciali: a sinistra "OC-5" prodotto da Boss, a destra "Sub'n'up" prodotto da tc electronic

1.1 Obiettivi della tesi

Questa tesi si pone come obiettivo principale di descrivere e studiare l'implementazione real-time dell'effetto digitale Octaver per chitarra sulla piattaforma Analog Devices SHARC SC589 tramite l'ambiente di programmazione a blocchi SigmaStudio+.
Per raggiungere l'obiettivo generale, sono stati identificati i seguenti obiettivi specifici:

- **Obiettivo 1**: Studiare l'effetto digitale Octaver ed il suo design.

- **Obiettivo 2**: Implementare l'effetto digitale Octaver in ambiente MATLAB per ottenere un riferimento di effetto funzionante.
- **Obiettivo 3**: Implementare l'effetto digitale Octaver real-time sulla piattaforma DSP SHARC SC589 tramite l'ambiente di sviluppo SigmaStudio+.
- **Obiettivo 4**: Valutare l'efficacia di SigmaStudio+ nell'implementazione dell'effetto Octaver.
- **Obiettivo 5**: Analizzare le prestazioni della piattaforma SHARC SC589 con l'effetto digitale Octaver in esecuzione.
- **Obiettivo 6**: Identificare eventuali vantaggi e limitazioni dell'utilizzo di SigmaStudio+ per lo sviluppo di applicazioni DSP real-time su SHARC SC589.

1.2 Struttura della tesi

In questa sezione viene fornita una panoramica riguardo all'organizzazione dell'elaborato, l'organizzazione segue il percorso logico adottato partendo dallo studio dell'effetto Octaver fino ad arrivare all'implementazione funzionante dell'effetto Octaver in real time:

- **Capitolo 2**: Studio dell'effetto Octaver
  - Definizione delle specifiche di funzionamento dell'effetto Octaver e design teorico.
- **Capitolo 3**: Simulazione in ambiente MATLAB
  - Implementazione in ambiente MATLAB dell'effetto Octaver e simulazioni per validare il design dell'effetto.
  - L'implementazione MATLAB dell'effetto servirà come base di partenza per l'implementazione real-time, mentre le simulazioni verranno utilizzate per una comparazione con le performance dell'effetto in real-time.
  - L'ultima parte del capitolo è rivolta alla risoluzione dei problemi riscontrati.

- **Capitolo 4**: Implementazione real-time
  - Descrizione della piattaforma hardware e software utilizzata. Viene posta l'attenzione sui componenti hardware e software necessari per la programmazione della piattaforma SHARC Audio Module tramite SigmaStudio+ e la relativa procedura di programmazione.
  - Implementazione dell'algoritmo dell'effetto nell'ambiente SigmaStudio+ ponendo particolare attenzione ai problemi ed alle criticità riscontrate.
  - L'ultima parte del capitolo è rivolta alla risoluzione dei problemi riscontrati.
- **Capitolo 5**: Test e validazione dell'implementazione real-time
  - Risultati dei test riguardanti l'implementazione real-time dell'effetto Octaver.
  - Confronto delle performance ottenute in real-time rispetto all'implementazione MATLAB, comprese alcune osservazioni finali riguardanti il funzionamento dell'effetto implementato in real time.
- **Capitolo 6**: Conclusioni
  - Discussione del lavoro svolto e ipotesi su possibili sviluppi futuri.

2 Studio dell’effetto Octaver

In questo capitolo viene descritto l’effetto digitale Octaver e lo studio della sua implementazione.

2.1 Descrizione dell’effetto e delle sue applicazioni

L’effetto Octaver è un effetto audio che consiste nell’aggiungere al suono originale uno o più toni, generati a partire dal segnale in ingresso, a ottave superiori o inferiori rispetto al tono fondamentale del segnale originale, permettendo di ottenere un suono più ricco di armoniche. In commercio esistono già diversi pedali che implementano questo tipo di effetto anche in modo polifonico, quindi funzionanti anche con accordi e non solamente con note singole. Solitamente questo tipo di effetto viene posizionato all’inizio di una eventuale catena di effetti in modo tale da avere in input il suono pulito della chitarra, questo come poi si potrà vedere nella tesi, anche per facilitarne il funzionamento.

2.2 Requisiti e specifiche tecniche

In questa realizzazione sperimentale è stato fissato l’obiettivo di implementare un effetto Octaver monofonico, funzionante con note singole, che vada ad aggiungere due toni, uno all’ottava superiore e uno all’ottava inferiore del segnale in ingresso. Questo per evitare di complicare ulteriormente l’implementazione real-time in un nuovo ambiente di sviluppo, ponendo comunque una base per future implementazioni.

2.3 Design dell’effetto

Nel caso del suono di una nota di chitarra (come in generale di qualsiasi strumento musicale), andando ad analizzare lo spettro, il segnale presenta diversi contributi in frequenza. In

particolare è utile individuare il contributo a frequenza minore il quale va a definire la frequenza fondamentale del suono, mentre le restanti componenti spettrali sono armoniche multiple di essa. La frequenza fondamentale va a determinare il pitch del suono, ovvero la nostra percezione di quanto questo sia acuto o grave. La percezione del pitch segue una scala logaritmica e non lineare, questo significa che la differenza percepita tra due suoni con frequenze fondamentali 100Hz e 200Hz è simile a quella tra 1000Hz e 2000Hz nonostante la differenza assoluta sia maggiore. Proprio per questo, un'ottava è un intervallo musicale corrispondente ad un raddoppio o dimezzamento della frequenza, a seconda che sia un'ottava ascendente o discendente. Generare un tono all'ottava superiore rispetto ad uno di riferimento, corrisponde a generare il tono al doppio della frequenza del tono di riferimento, al contrario generare questo tono all'ottava inferiore corrisponde a generarlo alla metà della frequenza di riferimento. In questo caso il tono di riferimento sarà il contributo a frequenza più bassa dello spettro del segnale, ovvero quello alla frequenza fondamentale.

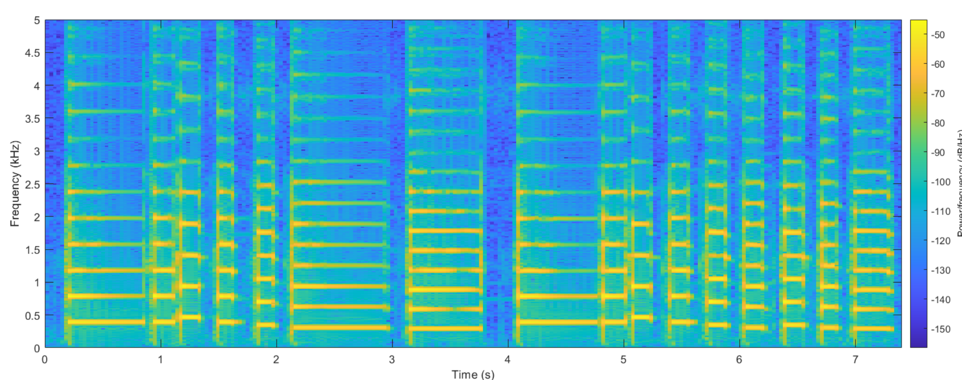

Figura 2.1: Spettrogramma di alcune note suonate da una chitarra $f_s = 48kHz$

Come si può osservare dallo spettrogramma in figura 2.1, il suono di una chitarra è composto da diverse armoniche concentrate nello spettro in particolare al di sotto dei 5kHz, le note fondamentali possono essere individuate dai contributi spettrali a frequenza minore che nel caso della chitarra elettrica sono comprese nel range di frequenze che vanno da 80Hz a circa 1200Hz. Sempre osservando lo spettrogramma in figura 2.1 si può notare la presenza all'interno del segnale originale del tono a frequenza doppia. L'obiettivo

dell'effetto è generare due toni, uno a frequenza doppia e uno a frequenza dimezzata rispetto a alla nota fondamentale. Questi toni devono essere generati a partire direttamente dal segnale originale ed infine essere sommati al suono originale con volumi controllati dall'utente.

2.3.1 Generazione delle ottave

La generazione delle armoniche e subarmoniche a partire dal suono in ingresso può essere ottenuta con il calcolo del valore assoluto e tramite il conteggio degli attraversamenti per lo zero del segnale [6]. L'idea è quella di generare il tono all'ottava superiore tramite il calcolo del valore assoluto. Mentre per generare il tono a frequenza dimezzata, è necessario moltiplicare il segnale in ingresso con un segnale generato contando i passaggi del segnale in ingresso attraverso lo zero. Il segnale da moltiplicare deve assumere il valore: 1 al primo attraversamento, 0 al secondo attraversamento mantenendo questo valore anche al terzo e quarto, al quinto attraversamento si riporta a 1 e così via.
La funzione valore assoluto di x è definita nell'equazione 2.1.

$$|x| = \begin{cases} x & \text{se } x \geq 0 \\ -x & \text{se } x < 0 \end{cases} \tag{2.1}$$

Viene utile andare ad analizzare il comportamento di queste operazioni con un segnale puramente sinusoidale a frequenza prefissata per verificarne il funzionamento e le prestazioni, figura 2.2.
Andando ad analizzare il risultato ottenuto applicando i metodi sopra elencati in figura 2.2 si può osservare come vengano generati i toni a frequenza doppia e mezza, oltre a questi toni ricercati vengono generati anche dei contributi "indesiderati", contributi che necessiteranno di essere eliminati tramite appositi filtri. I filtri che andranno a rimuovere i contributi armonici indesiderati dovranno anche variare nel tempo andando a seguire l'andamento della frequenza fondamentale del segnale.
Con il presupposto di conoscere attraverso una stima la frequenza fondamentale del segnale, è stato anche riadattato il metodo di generazione del tono all'ottava inferiore pensando ad una semplificazione rispetto al conteggio degli attraversamenti per lo zero. Que-

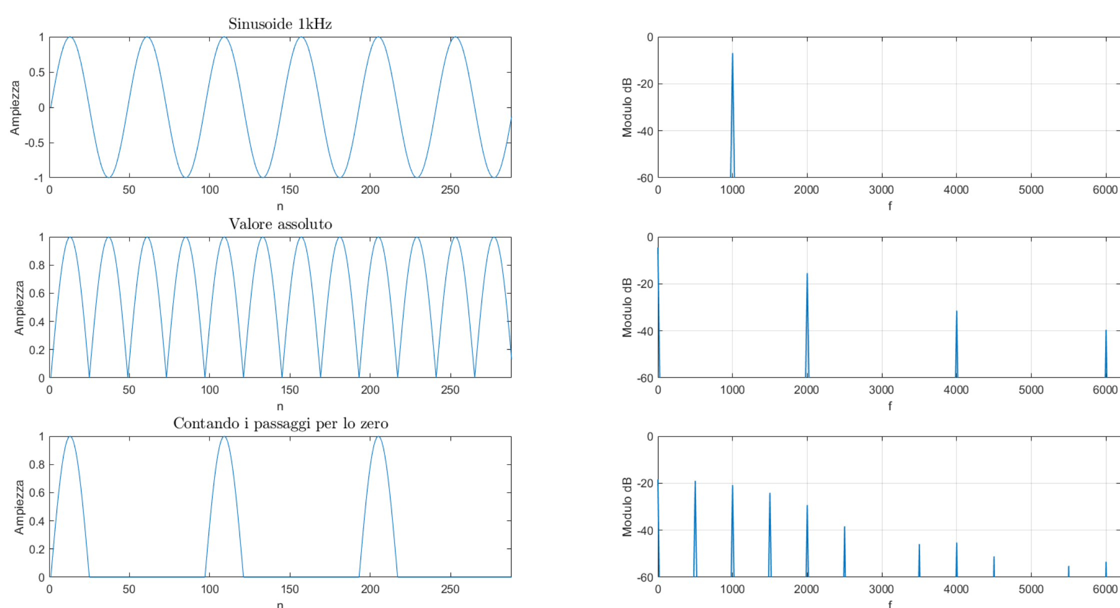

Figura 2.2: Analisi della generazione delle ottave da un tono puro $f_0 = 1kHz$

sto nuovo metodo prevede di moltiplicare il segnale originale con un'onda quadra a frequenza pari alla metà della fondamentale e duty cycle del 50%, rappresentato in figura 2.3.

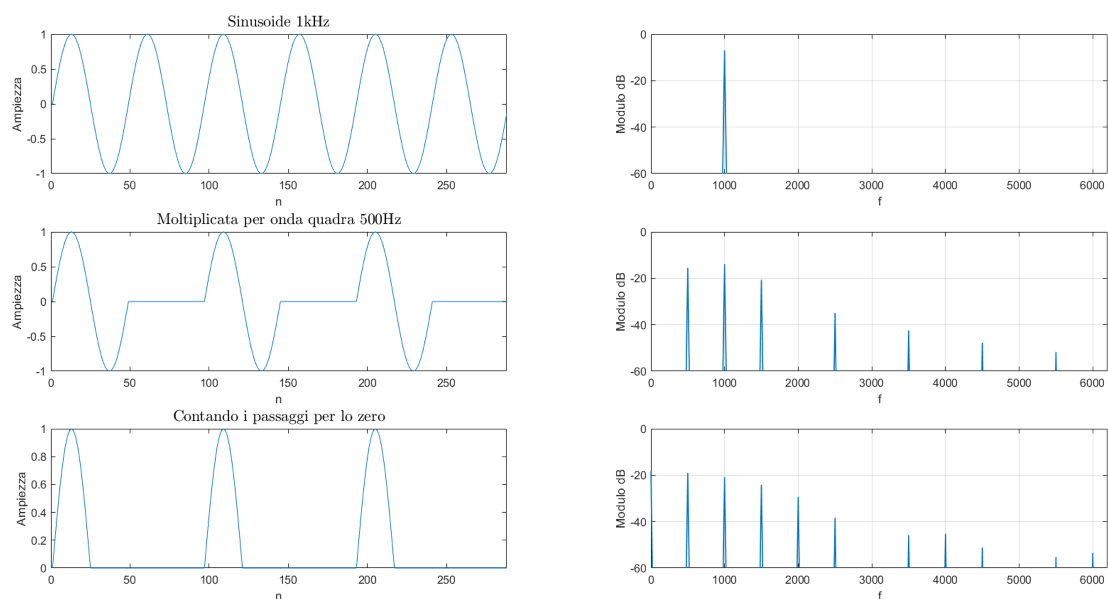

Figura 2.3: Confronto dei due metodi di generazione dell'ottava inferiore $f_s = 48kHz$

Andando ad analizzare il risultato ottenuto applicando il metodo di moltiplicazione con l'onda quadra e confrontandolo con il metodo di conteggio dei passaggi per lo zero (figura 2.3), si può osservare che i toni vengono generati con un valore maggiore in modulo mentre viene a mancare il contributo in continua ($f = 0$), questo metodo si rivela quindi una buona alternativa al precedente.
Dallo studio dei metodi per la generazione delle ottave nascono due nuove necessità:

- Stimare la frequenza fondamentale del segnale in ingresso.
- Attuare un filtraggio tempo variante centrato sui toni da estrarre.

2.3.2 Stima del pitch

La necessità di estrarre il pitch di un segmento di un segnale audio è molto diffusa, infatti la frequenza fondamentale una volta estratta può essere utilizzata per comandare altri algoritmi di processamento, come nel caso in oggetto. Per questo sono stati sviluppati e studiati molti algoritmi per stimare la frequenza fondamentale in modo sempre più accurato e mirato al segnale da identificare.
La scelta per questo progetto è ricaduta sull'algoritmo YIN [7], in particolare per la sua semplicità di implementazione data anche dall'elaborazione nel dominio del tempo.
L'algoritmo YIN per la stima del pitch si basa sul calcolo della funzione differenza media normalizzata (2.3) effettuato su finestre di segnale di dimensione N. La dimensione delle finestre utilizzate per la stima è un parametro fondamentale dal quale ne dipende il funzionamento dell'algoritmo, infatti non sarà possibile stimare frequenze corrispondenti a periodi maggiori della dimensione della finestra. L'algoritmo consiste nel calcolare i valori della funzione 2.3 per diversi valori di l (al massimo corrispondenti alla dimensione della finestra) e nell'andare a ricercare il minimo della funzione, il valore di l in corrispondenza del minimo della funzione andrà quindi a rappresentare la stima del periodo fondamentale T_0 del segnale all'interno della finestra, dal quale sfruttando poi la relazione (2.2) sarà quindi possibile determinare la stima della frequenza fondamentale.

$$F_0 = f_s/T_0 \tag{2.2}$$

$$d'_t[l] = \begin{cases} 1 & \text{se } l = 0 \\ \frac{d_t(l)}{\frac{1}{l}\sum_{n=1}^{l} d_t[n]} & altrimenti \end{cases} \tag{2.3}$$

Con $d_t(l)$ funzione differenza cumulata definita come:

$$d_t[l] = \sum_{n=1}^{N-1} (x[n] - x[n+l])^2 \tag{2.4}$$

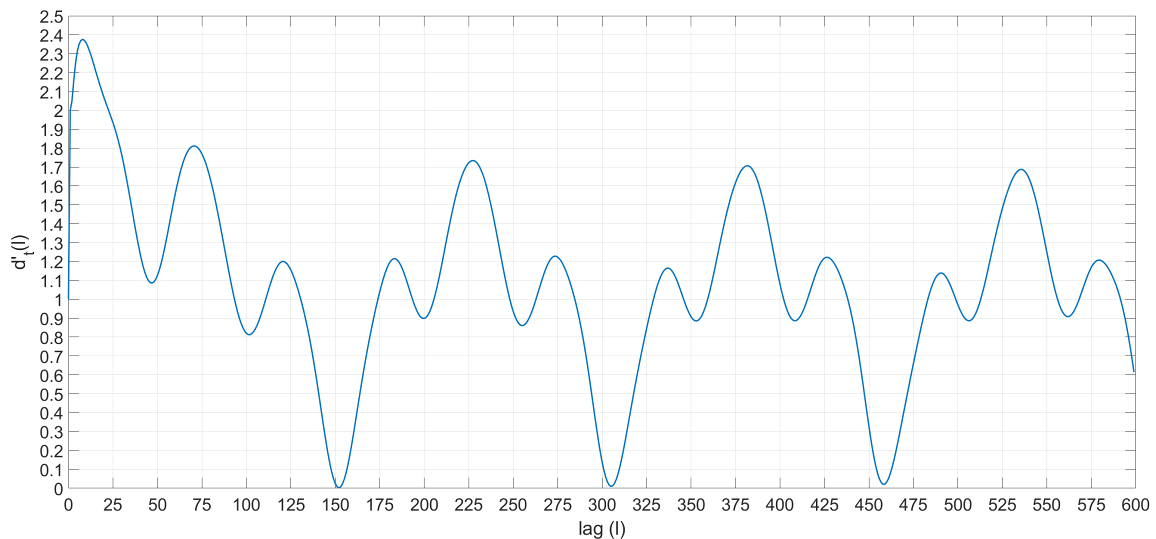

Figura 2.4: Analisi della funzione differenza media normalizzata (2.3) applicata in una finestra $N = 1024$ $N_l = 600$ del sample di chitarra in figura 2.1 all'istante $t = 2.4s$ $f_s = 48kHz$

Come ogni algoritmo di stima, YIN non è perfetto, infatti la periodicità dei segnali può portare a stimare frequenze troppo basse e quindi errate. Questo problema può essere parzialmente risolto attraverso una ricerca del minimo non assoluto, la ricerca deve essere eseguita a partire dal valore $l = 1$ andando a crescere e valutando il confronto tra un valore di soglia yin_{th} ed il valore assunto dalla funzione per quel determinato valore di l: quando il valore della funzione scende al di sotto del valore di soglia impostato si va a ricercare il punto di inversione, il valore di $l = T_0$ corrispondente al punto di inversione rappresenterà il minimo della funzione, ovvero la stima del periodo fondamentale del segnale. La soglia introdotta necessiterà di essere aggiustata in base ai segnali analizzati, poiché il suo valore ottimale è dipendente dal segnale. Nel caso venga impostato

un valore di soglia troppo elevato si potrebbe ottenere il problema opposto: delle stime di frequenze troppo elevate. Nel caso del suono di una chitarra le frequenze sono relativamente basse quindi questa soglia dovrà essere impostata ad un valore sufficientemente basso che permetta di non andare a confondere la frequenza fondamentale con i contributi delle armoniche.

Nell'esempio in figura 2.4 si può notare come il primo picco corrisponda circa al valore $l = 155$ che con una frequenza di campionamento $f_s = 48kHz$ corrisponde ad una frequenza stimata di $\frac{f_s}{l_{min}} = \frac{48000}{155} \approx 310Hz$ in accordo con il tono che compare nello spettrogramma in figura 2.1.

2.3.3 Filtraggio tempo variante

Per ottenere i toni desiderati ottenuti dai processi di generazione delle ottave descritti precedentemente, si devono eliminare i contributi armonici dei toni principali attraverso dei filtri passa banda centrati sulla frequenza del tono da estrarre. Per questo motivo è necessario che le caratteristiche dei filtri cambino nel tempo andando a seguire l'andamento della frequenza fondamentale del segnale, questi filtri devono quindi poter essere comandati dalla frequenza stimata. Per effettuare un filtraggio passa banda tempo variante in modo agevole è necessario ricercare un tipo di filtro che permetta di variare la sua frequenza di centro banda (f_c) tramite un parametro, il tipo di filtro che permette questo nel modo più semplice è il filtro di tipo IIR (Infinite Impulse Response). Nello specifico è stata scelta una struttura di filtri IIR del secondo ordine parametrici basata su sistemi all-pass descritti in [6].

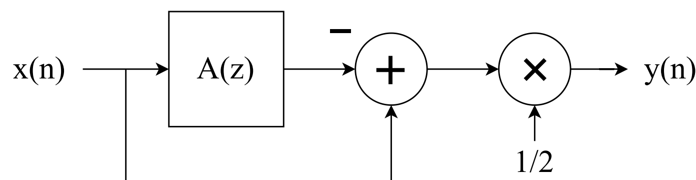

Figura 2.5: Schema a blocchi di un filtro passa banda basato su sistemi all-pass

$$A(z) = \frac{-c + d(1-c)z^{-1} + z^{-2}}{1 + d(1-c)z^{-1} - cz^{-2}} \tag{2.5}$$

$$c = \frac{\tan(\pi f_b/f_S) - 1}{\tan(\pi f_b/f_S) + 1} \tag{2.6}$$

$$d = -cos(2\pi f_c/f_S) \tag{2.7}$$

La struttura del filtro scelto è quella rappresentata nello schema in figura 2.5. L'equazione $A(z)$ (2.5) rappresenta la funzione di rete del sistema all-pass utilizzato, si può osservare come questa dipenda da due coefficienti c e d che a loro volta sono dipendenti dai parametri di interesse, ovvero la frequenza di centro banda f_c e la larghezza della banda passante f_b (corrispondente all'attenuazione di -3 dB). La larghezza della banda passante può essere legata a sua volta alla frequenza di centro banda tramite il fattore di qualità definito come $Q = \frac{f_c}{f_b}$. Questa struttura rende possibile controllare le caratteristiche del filtro tramite il valore della frequenza stimata che varierà nel tempo andando ad eseguire un inseguimento del pitch del segnale in ingresso. Un esempio di risposta in frequenza di un filtro di questo tipo è mostrato in figura 2.6.

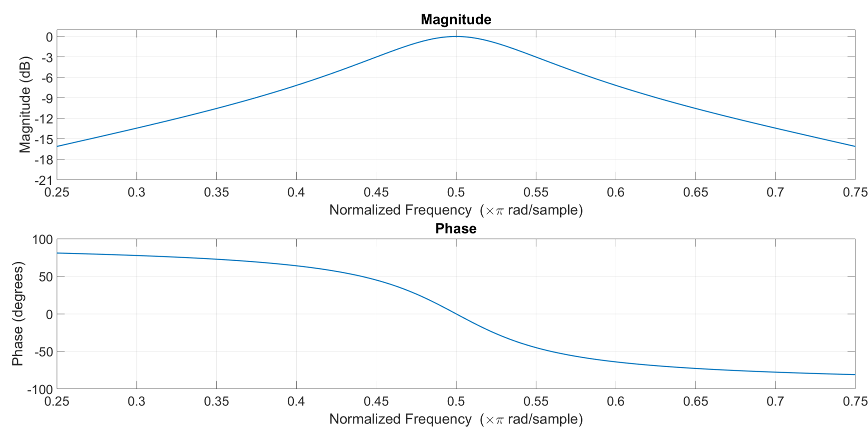

Figura 2.6: Risposta in frequenza filtro passa banda $f_c = 0.5\frac{f_s}{2}$ e $fb = 0.1\frac{f_s}{2}$ $(Q = 5)$

2.3.4 Schema generale dell'effetto

Una volta studiati gli algoritmi necessari per la generazione dei toni che vanno a caratterizzare l'effetto, è possibile definire un primo schema a blocchi che ne riassuma il design, questo è mostrato in figura 2.7. Oltre ai blocchi previsti precedentemente è stato aggiunto un filtro sul segnale in ingresso centrato sulla frequenza fondamentale (con una banda

non troppo stretta) la cui uscita viene utilizzata per generare le ottave, questo per fare in modo di diminuire l'effetto dei contributi armonici nella generazione dei toni. Infine sono stati aggiunti dei controlli di volume dei toni e del segnale in ingresso prima di essere sommati in uscita, in modo da poter avere un controllo sulla sonorità dell'effetto che potrà poi essere regolato a piacere del musicista.

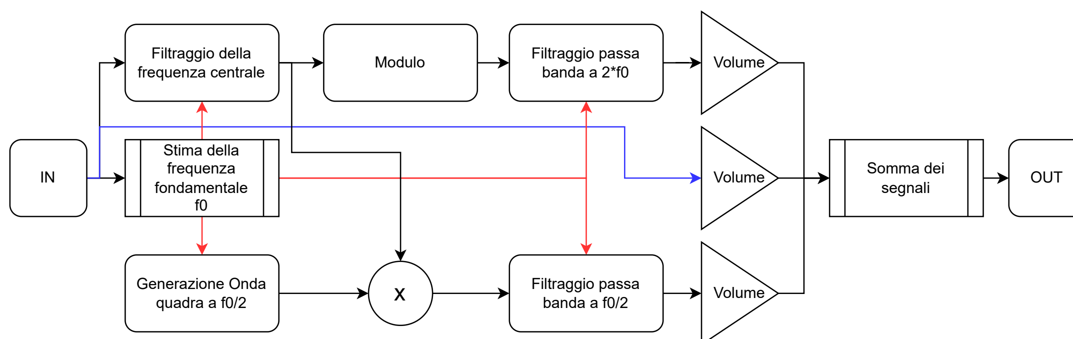

Figura 2.7: Struttura di base dell'effetto

3 Simulazione in ambiente MATLAB

In questo capitolo viene descritta l'implementazione dell'effetto Octaver in ambiente MATLAB, inoltre vengono mostrati e discussi i risultati ottenuti dalle simulazioni.

3.1 Algoritmo di base

L'implementazione in ambiente MATLAB ricalca come struttura lo schema a blocchi in figura 2.7, si va quindi a discutere l'implementazione dei vari blocchi che comporranno l'effetto visti nel capitolo precedente e la loro integrazione all'interno del codice.
Come approccio di programmazione è stato seguito quello di una elaborazione dei singoli campioni del segnale questo per poter simulare le caratteristiche tempo varianti dell'effetto, gli algoritmi di generazione dei toni lavorano sui singoli campioni, mentre la stima del pitch viene eseguita su finestre di dimensioni definite che si vanno a sovrapporre parzialmente, la frequenza di aggiornamento della frequenza stimata risulterà quindi minore rispetto a quella di campionamento.
L'implementazione dei filtri tempo-varianti e dell'algoritmo di stima del pitch sono state prese da [6] e riadattate per lo scopo.
L'implementazione dell'algoritmo di stima del pitch non viene qui riportata ma è possibile trovarla nella [8]. Un parametro importante di questa funzione risulta essere la soglia `yinThreshold`, dalla quale dipende il presentarsi degli errori di stima, questa soglia deve essere aggiustata in base al comportamento risultante dalle simulazioni. Inoltre riguardo alla stima della frequenza viene anche aggiunta la condizione di mantenere la stima precedente nel caso la funzione non riesca a produrre una stima.
In figura 3.1 viene mostrata la funzione MATLAB utilizzata per filtrare i campioni del segnale, si può notare come insieme al campione di segnale filtrato, in output vengano anche

restituiti i coefficienti del filtro da mantenere in memoria per la corretta elaborazione del campione successivo.

```
function [y,xh] = myApbandpass (x, Wc, Wb,xh)
    % [y,xh] = myApbandpass (x, Wc, Wb, xh)
    % y = campione in input (x) filtrato
    % xh = ultimi coefficienti del filtro
    % Wc frequenza di centro banda normalizzata 2*fc/fS.
    % Wb larghezza di banda normalizzata 2*fb/fS.
    c = (tan(pi*Wb/2)-1) / (tan(pi*Wb/2)+1);
    d = -cos(pi*Wc);
    xh_new = x - d*(1-c)*xh(1) + c*xh(2);
    ap_y = -c * xh_new + d*(1-c)*xh(1) + xh(2);
    xh = [xh_new, xh(1)];
    y = 0.5 * (x - ap_y);
end
```

Figura 3.1: Funzione MATLAB del filtro passa banda tempo variante

Per la generazione delle ottave non sono state create apposite funzioni ma avviene tutto direttamente all'interno del ciclo come descritto in figura 7.1, il segnale in ingresso viene prima filtrato attorno alla f_0 stimata e poi utilizzato (`fundamental`) per la generazione delle ottave "grezze" con ancora il contributo armonico nei seguenti modi:

- Ottava superiore: ottenuta calcolando il modulo del segnale filtrato in ingresso.
- Ottava inferiore: ottenuta andando a modulare il segnale con un'onda quadra ($\delta = 50\%$) generata da un segnale a dente di sega comandato dalla f_0 stimata, questo per limitare la discontinuità delle variazioni di frequenza.

Una volta generate le ottave grezze queste vengono poi filtrate dai filtri tempo varianti tramite la funzione in figura 3.1, in questo modo si ottengono i sample contenenti i toni all'ottava superiore ed inferiore che andranno a costituire il risultato dell'applicazione dell'effetto.
Parametri di rilevante importanza all'interno del codice sono:

- dimensione della finestra per la stima del pitch (`windowL`)
- fattore di sovrapposizione delle finestre (`windowOverlap`)

- soglia di rilevazione del pitch (`yinThreshold`)
- minima frequenza stimabile (`fMin`)
- fattori di qualità dei filtri (`Ql`, `Qh` e `Q`)
- volumi e guadagni in uscita (`gH`,`gL` e `gM`)

3.2 Risultati della simulazione

3.2.1 Chirp

La prima simulazione, per verificare il funzionamento dell'effetto, è stata eseguita utilizzando come ingresso un segnale sinusoidale puro che varia in frequenza linearmente ottenuto tramite la funzione MATLAB $chirp$. La sinusoide generata parte da una frequenza $f_{start} = 80Hz$ e arriva ad una frequenza $f_{stop} = 1500Hz$ in un intervallo di tempo $T_{chirp} = 5s$, la frequenza di campionamento utilizzata è $f_s = 48kHz$, lo spettrogramma del segnale originale è mostrato in figura 3.2. Per l'effetto, i parametri impostati sono: `windowL=1024`, `windowOverlap=8`, `fMin=80`, `yinThreshold=0.22`, `Q=1`, `Qh=3`, `Ql=12`, `gH=gL=1.5`, `gM=1`. Lo spettrogramma del segnale ottenuto in uscita è presentato in figura 3.3.

3.2.2 Sample di chitarra

Per la seconda simulazione è stato utilizzato come segnale in ingresso un sample di chitarra il cui spettrogramma viene mostrato in figura 3.4. La frequenza di campionamento del file audio è $f_s = 44.1kHz$. Per l'effetto, i parametri impostati sono: `windowL=1024`, `windowOverlap=8`, `fMin=80Hz`, `yinThreshold=0.22`, `Q=1`, `Qh=3`, `Ql=12`, `gH=gL=3`, `gM=1`. Il risultato è mostrato in figura 3.5.

La terza simulazione è stata realizzata con lo stesso ingresso della seconda e stessi parametri, andando questa volta a dimezzare la dimensione della finestra di stima del pitch: `windowL=512` con un conseguente innalzamento della frequenza minima stimabile `fMin=100Hz`, il risultato è mostrato in figura 3.6.

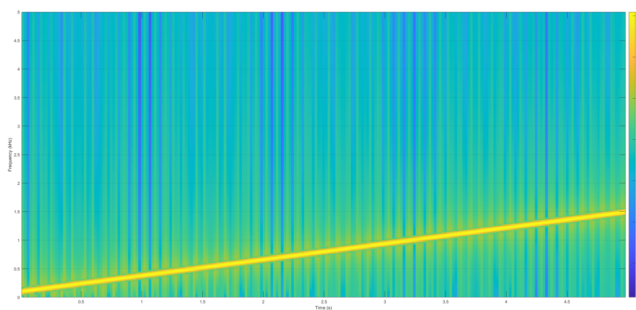

Figura 3.2: Spettrogramma del segnale in ingresso (chirp)

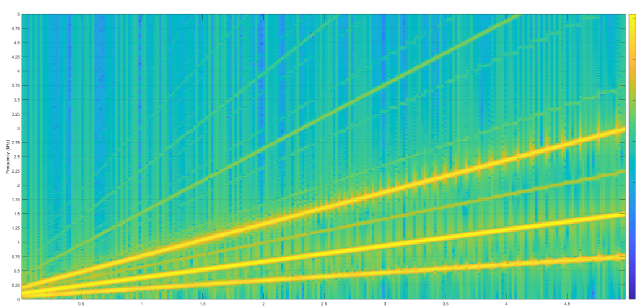

Figura 3.3: Spettrogramma del segnale in uscita (chirp)

3.2.3 Osservazioni

Da queste simulazioni è possibile osservare il funzionamento dell'effetto che va ad aggiungere il tono all'ottava inferiore e a rinforzare il tono presente all'ottava superiore. Si nota anche la presenza di toni indesiderati, in particolare nel chirp ma sono accettabilmente attenuati rispetto al segnale utile ($\approx -30dB$). Al contempo si nota anche la generazione di contributi spettrali intorno ai toni generati in particolare nelle figure 3.3 e 3.6, questi contributi andando ad ascoltare il suono in uscita vanno a creare degli artefatti che degradano la qualità audio in uscita rendendo l'effetto non funzionante correttamente, per questo motivo è necessario studiarne la causa e correggere il problema. Infine un ultimo problema ma di minore rilevanza risiede nella variazione dell'ampiezza dei segnali generati la quale risulta essere dipendente dal segnale in ingresso, per questo problema la soluzione subito adottata è stata quella di andare a variare i guadagni dei segnali (`gL` e `gH`) dopo una prima fase di analisi. Una eventuale soluzione potrebbe essere quella di andare a comprimere i segnali e poi amplificarli oppure adottare una soluzione che automatizzi la regolazione di questi guadagni.

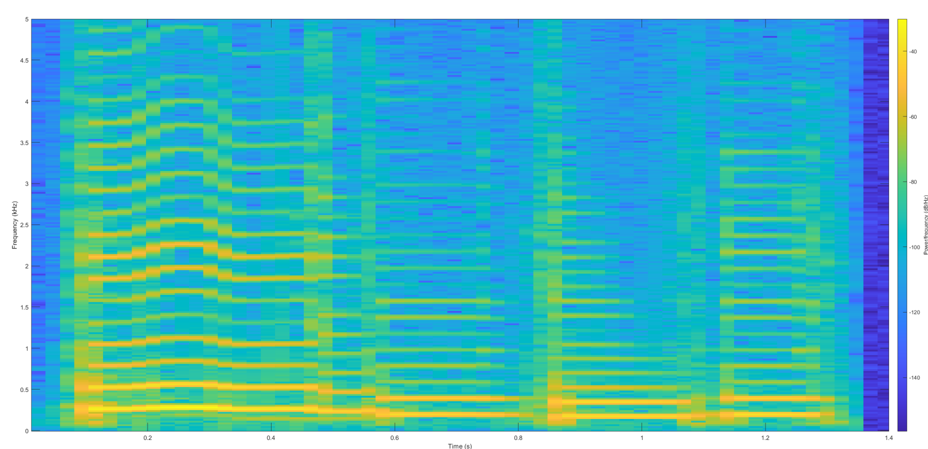

Figura 3.4: Spettrogramma del segnale in ingresso (sample chitarra).

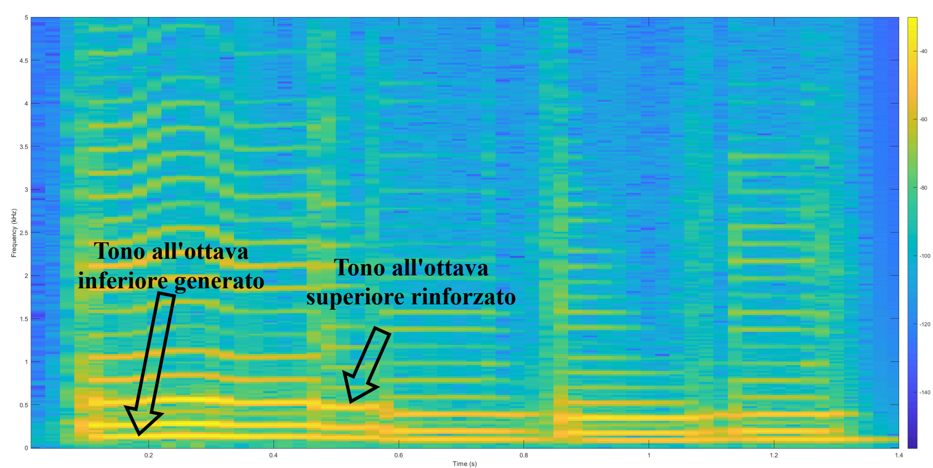

Figura 3.5: Spettrogramma del segnale in uscita `windowL=1024 yinThreshold=0.22` (sample chitarra).

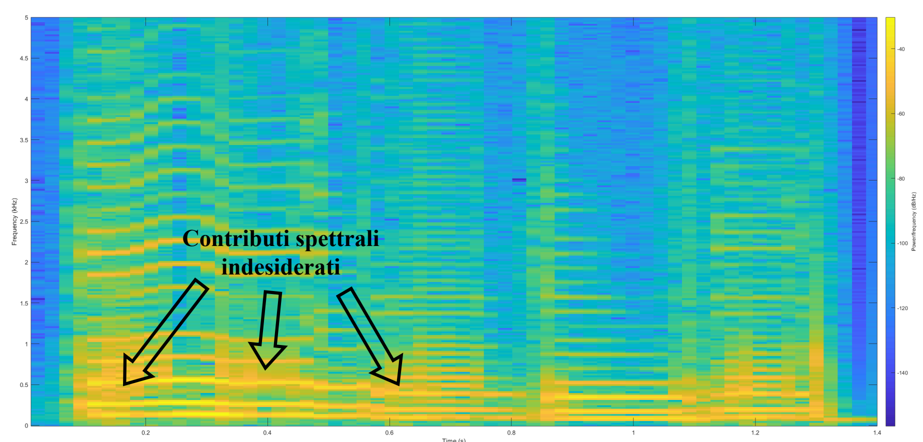

Figura 3.6: Spettrogramma del segnale in uscita `windowL=512 yinThreshold=0.22` (sample chitarra).

3.3 Problemi riscontrati e soluzioni

I problemi riscontrati nelle simulazioni risiedono principalmente nel segnale di controllo f_0 generato dalla stima del pitch, infatti da questo dipende il funzionamento dell'intero effetto essendo il segnale di controllo dei coefficienti dei filtri tempo varianti. Andando ad analizzare l'andamento del segnale $f0$ nel caso del chirp come segnale in ingresso, rappresentato in figura 3.7, possiamo osservare come l'algoritmo di stima del pitch abbia difficoltà all'aumentare della frequenza a differenziare due frequenze diverse: questo porta ad un andamento discontinuo del segnale $f0$. Questo comportamento discontinuo porta ad avere degli artefatti nelle ottave generate dovute ai rapidi cambiamenti dei coefficienti dei filtri comandati dal segnale $f0$.

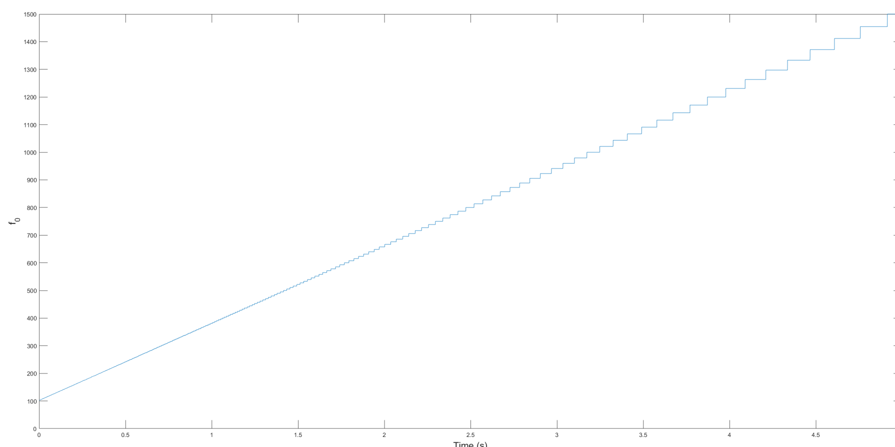

Figura 3.7: f_0 nel tempo `windowL=1024` (chirp).

Per risolvere il problema dovuto alle discontinuità del segnale la soluzione scelta è aggiungere un filtro passa basso IIR del secondo ordine sul segnale f_0 con frequenza di taglio $\approx 20Hz$, il filtro è stato preso da [6] ed è un filtro passa basso basato sullo stesso funzionamento dei filtri passa banda utilizzati per filtrare il segnale audio, la sua implementazione si trova in [8]. In questo modo i cambiamenti di frequenza vengono resi più dolci e gli artefatti in uscita vengono ridotti notevolmente, figure 3.8, 3.9.
Andando invece ad osservare i grafici dell'andamento del segnale $f0$ con ingresso il sample di chitarra in figura 3.10 si possono notare dei cambiamenti molto rapidi del segnale dovuti a delle stime errate da parte dell'algoritmo, questi vanno a peggiorare nel caso di

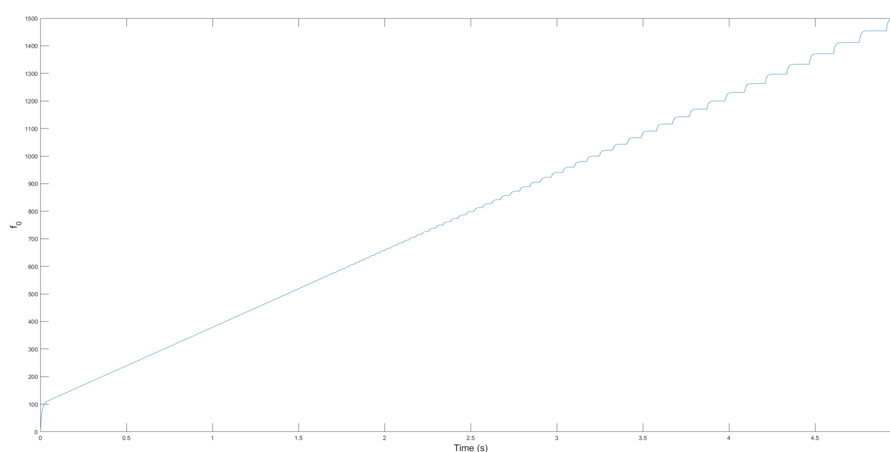

Figura 3.8: f_0 con filtro passa basso (chirp).

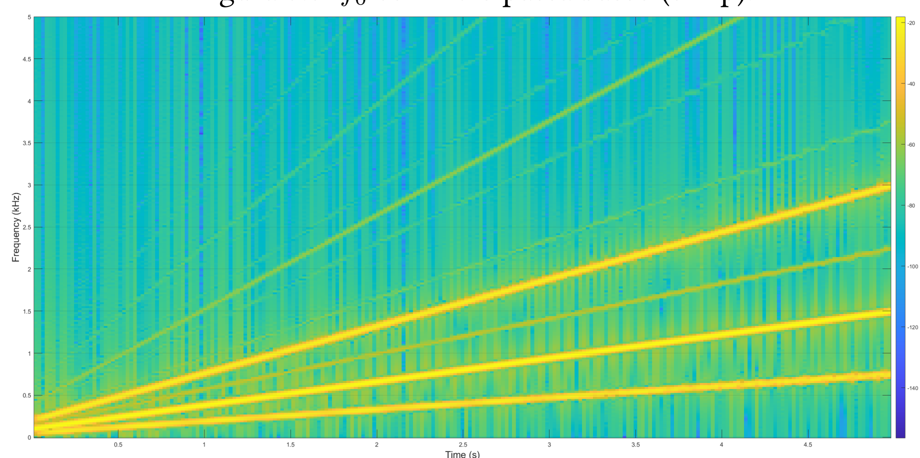

Figura 3.9: Segnale in uscita con filtro passa basso su f_0 (chirp).

utilizzo di una finestra di lunghezza minore andando ad evidenziare un andamento instabile. L'introduzione del filtro passa basso non riesce a sopperire alle stime errate che causano dei picchi nel segnale o che portano ad una errata generazione dei toni, queste stime possono essere migliorate andando ad affinare il parametro di soglia dell'algoritmo YIN (`yinThreshold`), per questo sono state eseguite delle simulazioni andando a confrontare i risultati con diversi valori di `yinThreshold` con le due diverse dimensioni di finestre (512 e 1024 campioni), gli andamenti di f_0 sono riportati nelle figure 3.11 e 3.12.

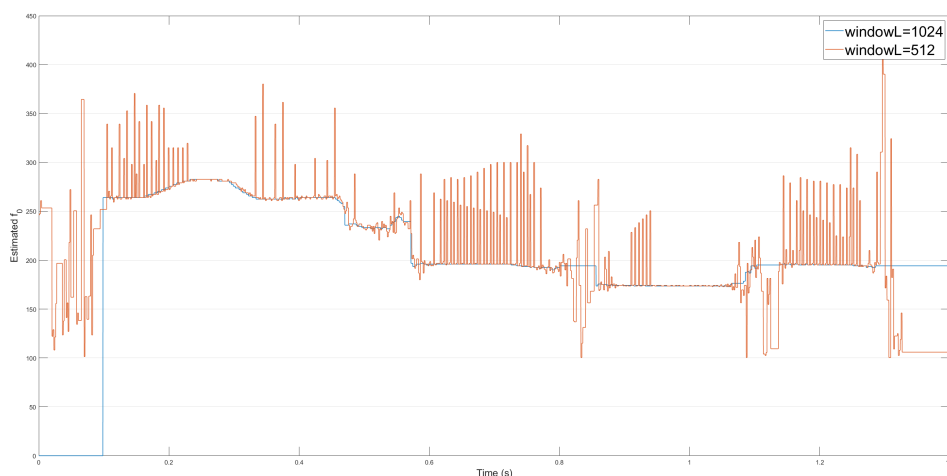

Figura 3.10: f_0 nel tempo `yinThreshold=0.22` (sample chitarra).

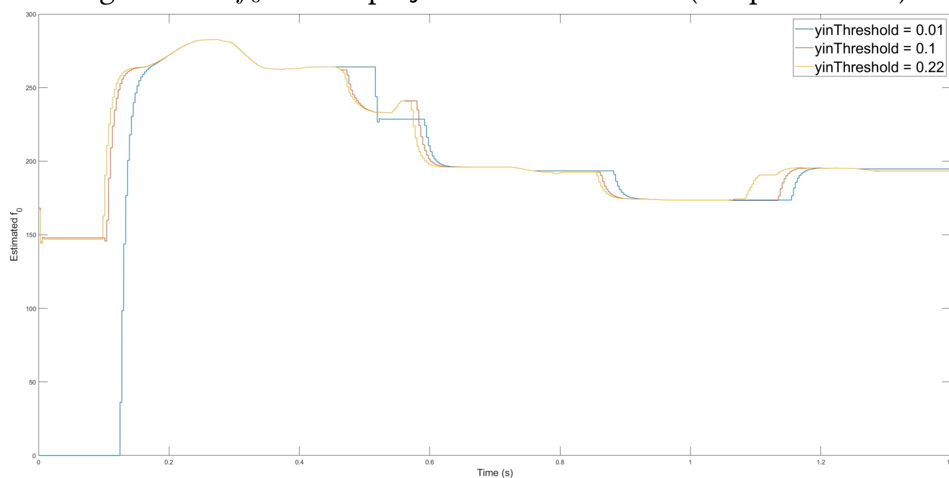

Figura 3.11: f_0 nel tempo con `yinThreshold=0.01/0.1/0.22`, `windowL=1024` e filtro passa basso (sample chitarra).

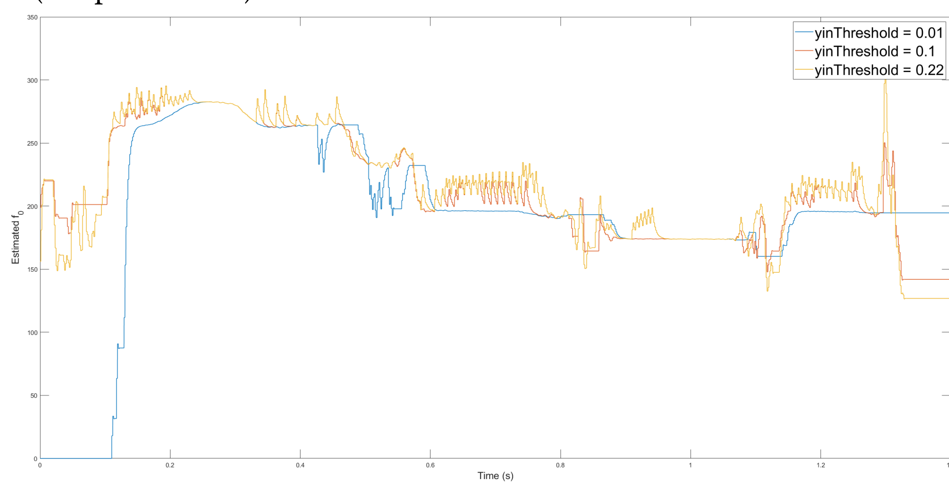

Figura 3.12: f_0 nel tempo con `yinThreshold=0.01/0.1/0.22`, `windowL=512` e filtro passa basso (sample chitarra).

Andando ad analizzare l'andamento del segnale f_0 nelle figure 3.11 e 3.12, è possibile osservare come al diminuire del valore di soglia `yinThreshold` si ha un miglioramento dal punto di vista della pulizia del segnale: i valori stimati assumono un andamento più regolare, migliore per il controllo degli algoritmi di generazione delle ottave. Come contro al diminuire del valore di soglia si può invece notare un incremento del ritardo nell'andare a stimare la frequenza del segnale: questo è dovuto alle mancate stime (che portano al mantenimento dell'ultima frequenza stimata) nel caso in cui la funzione 2.3 non scenda al di sotto del valore di soglia fissato. Il ritardo della stima dato dall'utilizzo di valori di soglia molto piccoli e la pulizia del segnale che risulta molto migliore andando ad utilizzare una finestra di dimensione maggiore, portano alla necessità di utilizzare una finestra sufficientemente grande (1024 campioni) per ottenere un buon risultato, con un valore di soglia `yinThreshold=0.1` il quale risulta essere un compromesso tra il ritardo e la pulizia del segnale. Mentre se si vuole utilizzare una finestra di dimensione più piccola risulta obbligatorio utilizzare un valore di soglia basso `yinThreshold=0.01` per ottenere un segnale sufficientemente regolare per controllare l'algoritmo ma comunque non regolare come quello ottenuto con l'utilizzo di una finestra di dimensione maggiore. Il risultato viene mostrato nella figura 3.13.

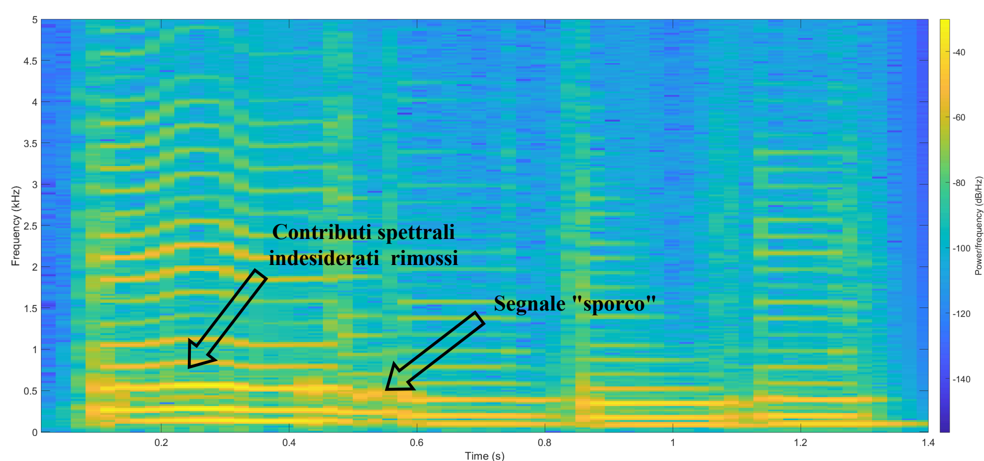

Figura 3.13: Segnale in uscita con `yinThreshold=0.1` e filtro passa basso, `windowL=512` (sample chitarra).

4 Implementazione real-time

In questo capitolo viene trattata l'implementazione dell'effetto in real-time. Partendo dalla descrizione della piattaforma e dell'ambiente di sviluppo, si passa quindi a descrivere la configurazione del sistema sia dal punto di vista hardware che software con una spiegazione di come si programma la scheda attraverso il tool SigmaStudio+ [9], infine si tratta il trasferimento su SigmaStudio+ dell'algoritmo precedentemente implementato in ambiente MATLAB e delle relative problematiche e soluzioni trovate.

4.1 Piattaforma DSP SHARC SC589

La piattaforma utilizzata per l'implementazione real-time dell'effetto è la SHARC Audio Module (ADZS-SC589-MINI) della Analog Devices, mostrata in figura 4.1 [10]. Questa piattaforma consente la prototipazione, lo sviluppo e la distribuzione di applicazioni audio, tra cui l'elaborazione di effetti, sistemi audio multicanale, sintetizzatori MIDI e molti altri progetti audio basati su DSP, il suo cuore è lo SHARC ADSP-SC589 di Analog Devices, esso combina due core DSP in virgola mobile da 450 MHz ed un core ARM®Cortex®-A5 da 450 MHz. La programmazione di questa piattaforma è supportata da SigmaStudio+ [11].

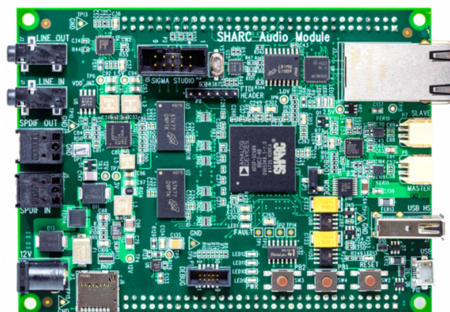

Figura 4.1: Sharc Audio Module [1]

4.2 SigmaStudio+

Lo strumento di sviluppo grafico SigmaStudio®+ è il software di programmazione, sviluppo e messa a punto per i processori audio SHARC® e SigmaDSP [12] di Analog Devices. SigmaStudio+ mette a disposizione (e da la possibilità di creare) diversi blocchi di elaborazione audio che possono essere collegati insieme come in uno schema (figura 4.2), il compilatore genera il codice per il sistema DSP embedded e predispone una interfaccia di controllo per l'impostazione e l'ottimizzazione dei parametri in tempo reale. Questo strumento consente quindi di accorciare il tempo necessario allo sviluppo di applicazioni DSP semplificando il lavoro di programmazione [13].

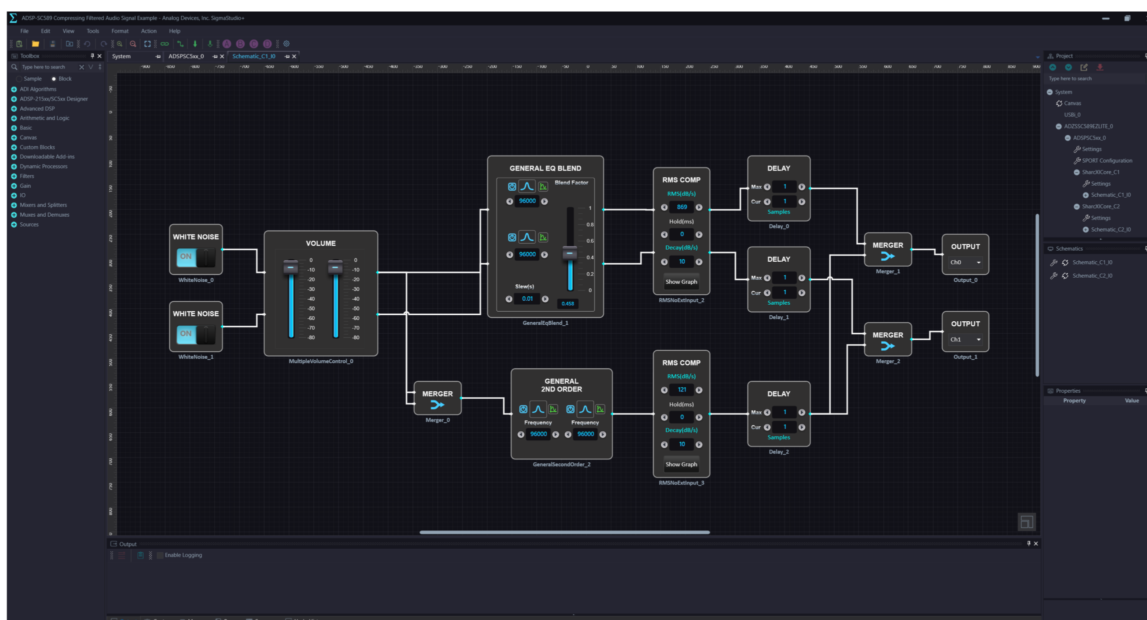

Figura 4.2: Interfaccia utente SigmaStudio+

4.3 Configurazione del sistema

Per poter programmare lo SHARC Audio Module tramite SigmaStudio+ sono necessari dei componenti hardware / software aggiuntivi ed il corretto setup dei vari software utilizzati.

4.3.1 Hardware

Lato hardware per poter programmare la scheda sono necessarie altre due schede oltre lo SHARC Audio Module, un'interfaccia JTAG (ICE-1000) e un adattatore I2C SPI (USBi). Queste dovranno essere collegate alla scheda e poi al computer per essere utilizzate con i rispettivi software [14].

ICE-1000

Per poter caricare il framework di SigmaStudio+ sulla piattaforma SHARC è necessaria un'interfaccia JTAG, in questo caso è stato utilizzato l'emulatore $ICE-1000$ [15] mostrato in figura 4.3.

Figura 4.3: ADZS ICE-1000

EVAL-ADUSB2EBZ

Per poter caricare il programma sviluppato in SigmaStudio+ e poter effettuare il tuning dei parametri in real-time è necessaria una interfaccia USB I2C-SPI, Analog Devices per questo mette a disposizione la scheda $EVAL-ADUSB2EBZ$ [16] mostrata in figura 4.4.

Figura 4.4: EVAL-ADUSB2EBZ (USBi)

4.3.2 Software

Per poter programmare lo SHARC Audio Module tramite SigmaStudio+ ed in generale i processori della serie SC5xx è necessario installare anche CrossCore Embedded Studio e dei pacchetti aggiuntivi per il supporto dei processori SC5xx [17].

CrossCore Embedded Studio

L'ambiente di sviluppo *CrossCore Embedded Studio* (CCES), di proprietà della Analog Devices, è un IDE basato su Eclipse che permette di sviluppare, debuggare e implementare algoritmi scritti in C/C++ sui processori della famiglia SHARC [18]. Nel caso di programmazione dei processori SHARC SC5xx tramite SigmaStudio+, CrossCore Embedded Studio viene utilizzato da SigmaStudio+ per la compilazione del codice, mentre deve essere utilizzato dall'utente in modalità di debug per caricare il framework di SigmaStudio relativo alla scheda che si deve programmare tramite l'interfaccia JTAG precedentemente descritta [19].

Caricamento del Framework

Caricare il Framework di SigmaStudio è un'operazione fondamentale per poter programmare la scheda con SigmaStudio+, per fare questo è necessario prima di tutto collegare la scheda all'$ICE1000$ e aprire *CrossCore Embedded Studio* sul proprio computer. La procedura per il caricamento del framework viene descritta sul sito del produttore [19], nel caso di utilizzo dello SHARC Audio Module è necessario selezionare i file $CORE$ e $.dxe$ presenti all'interno della cartella "ADSP-SC589-SAM" e non quelli in "ADSP-SC589" altrimenti SigmaStudio+ non riuscirà a comunicare con la scheda dando errori in fase di caricamento del programma. Nota: Questa procedura va ripetuta ogni volta che la scheda viene resettata dal pulsante di reset oppure spenta. Una volta eseguito il caricamento del Framework seguendo la procedura è possibile andare a caricare il programma da SigmaStudio+.

Setting del progetto SigmaStudio+ per lo SHARC Audio Module

Per creare un programma per lo SHARC Audio Module in SigmaStudio+ è necessario utilizzare il blocco relativo alla piattaforma $ADZS - SC589 - EZLITE$, questo non nasce con le impostazioni delle Serial Port (SPORT) corrette per lo SHARC Audio Module poiché il codec presente sul SAM è collegato (come mostrato in figura 4.5) su pin diversi rispetto alla board EZLITE, infatti andando a caricare il codice di default non si ottiene nessun output audio (e neanche input). Per risolvere questo problema è necessario andare alla voce "SPORT Configuration" e settare i parametri delle SPORT relative alla scheda SHARC Audio Module nel modo descritto in seguito [2]:

- Impostare le porte nella sezione $Sport\ Selection$ come in figura 4.6
- Nella sezione $Configure\ Sport$:
  - Selezionare la porta SPORT2A, impostare $DAI\ Port = DAI0$ e $DAI\ Pin = 1$ lasciando il resto di default
  - Selezionare la porta SPORT1A, impostare $DAI\ Port = DAI0$ e $DAI\ Pin = 2$ lasciando il resto di default
  - Selezionare la porta SPORT0A, impostare $DAI\ Port = DAI0$ e $DAI\ Pin = 13$ lasciando il resto di default

Eseguita questa operazione è necessario anche andare nella sezione delle impostazione dei core e caricare i file $.dxe$ relativi ai core precedentemente utilizzati nella procedura di caricamento del framework altrimenti SigmaStudio+ non riuscirà a compilare il codice.
Una volta seguiti questi passaggi ed in seguito al caricamento del Framework come descritto precedentemente è possibile andare a caricare il programma creato in SigmaStudio+ sulla scheda. Il caricamento del programma avviene tramite l'adattatore "EVAL-ADUSB2EBZ", e viene iniziato cliccando in SigmaStudio+ il pulsante "Link Compile Download". Una volta finito di caricare il programma in automatico SigmaStudio+ si metterà in una modalità che permette di variare in real-time alcuni parametri dei blocchi presenti nello schema del programma creato.

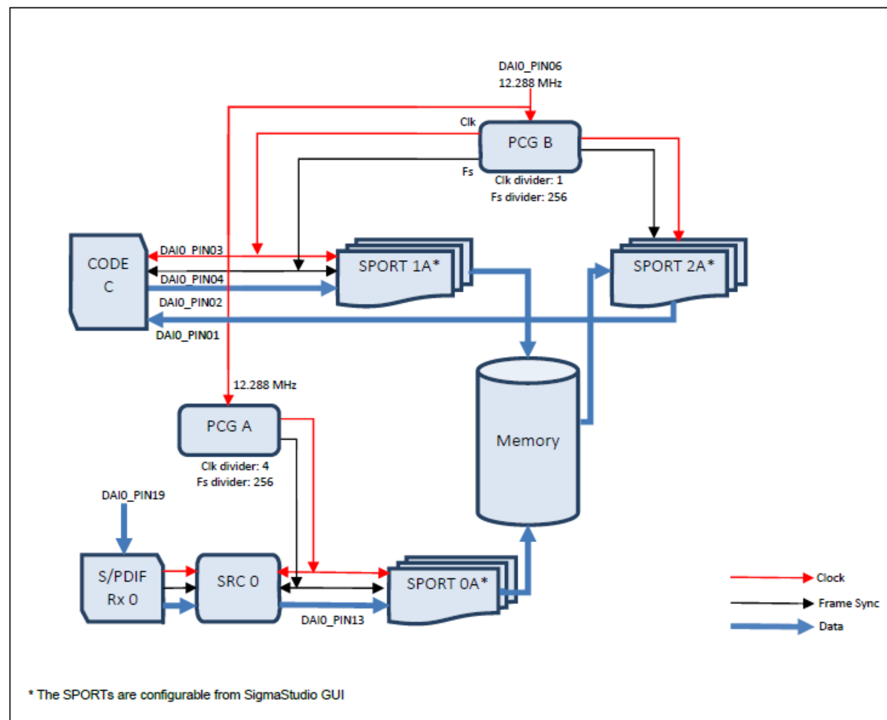

Figura 4.5: Routing Scheme dello SHARC Audio Module [2]

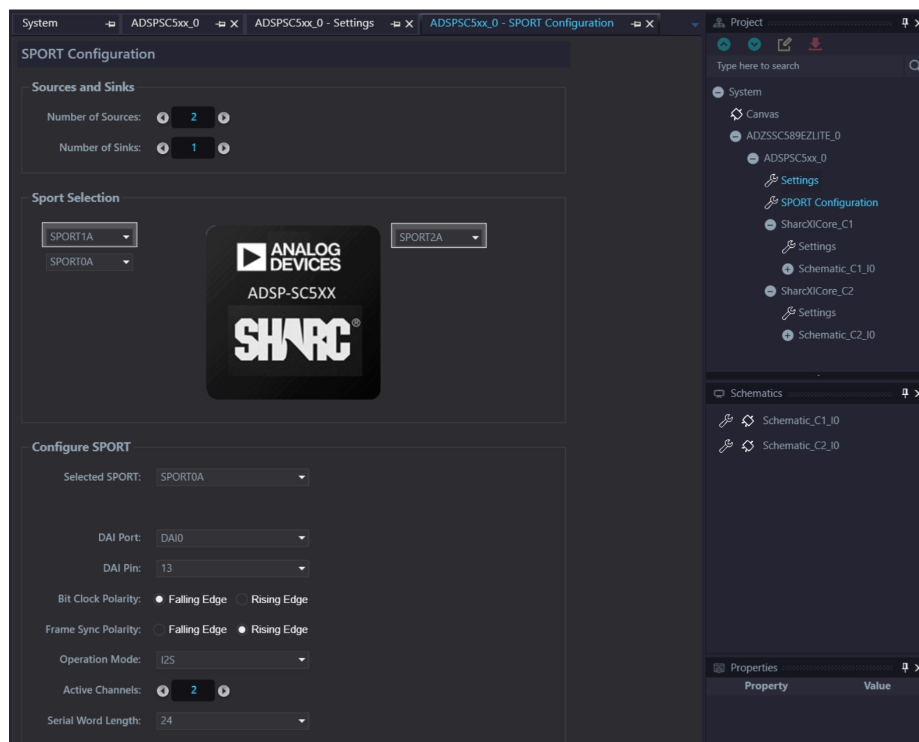

Figura 4.6: Impostazione delle SPORT per lo SHARC Audio Module

4.4 Trasferimento dell'algoritmo

Come primo passo per il trasferimento dell'algoritmo è stato creato un nuovo progetto per la piattaforma SHARC Audio Module come descritto nel paragrafo precedente, si sottolinea che l'implementazione è stata eseguita utilizzando solamente un core della piattaforma SC589, le impostazioni del progetto sono state lasciate di default:

- Frequenza di campionamento $f_s = 48kHz$ (determinata dal blocco di input)
- Processing a blocchi di segnale (determinato dal tipo di blocchi utilizzati nello schema)
- Dimensione dei blocchi $blockSize = 64$ (determinata dal blocco di input)

SigmaStudio+ è un tool basato sulla programmazione a blocchi, per questo come primo passo è stato necessario individuare lo schema a blocchi da implementare, questo è rappresentato in figura 4.7.

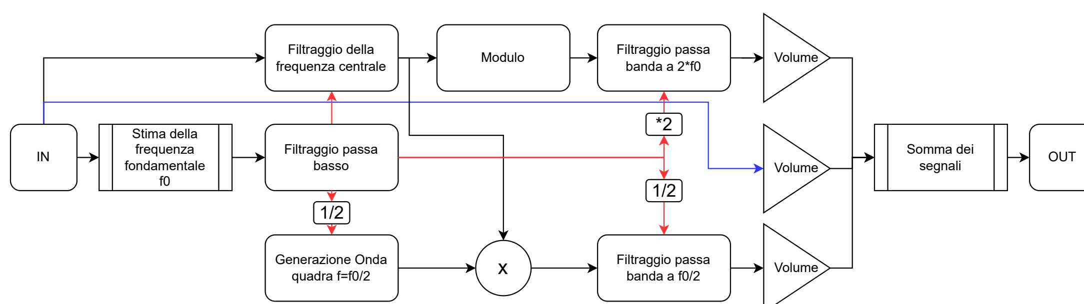

Figura 4.7: Schema a blocchi dell'implementazione real-time

Dallo schema a blocchi è possibile individuare quali sono quelli necessari per l'implementazione dell'effetto Octaver:

- Stimatore della frequenza fondamentale f_0
- Filtro passa-basso del segnale f_0
- Filtro passa-banda con frequenza centrale controllata da un segnale

- Calcolo del valore assoluto
- Generatore di una onda quadra con frequenza controllata da un segnale
- Moltiplicatore di segnali
- Controllo del volume di un segnale
- Mixer di segnali

Dei blocchi necessari non tutti sono presenti all'interno della libreria fornita da SigmaStudio+, infatti non sono presenti i seguenti:

- Stimatore della frequenza fondamentale f_0
- Filtro passa-banda con frequenza centrale controllata da un segnale
- Generatore di una onda quadra con frequenza controllata da un segnale

Per la generazione dell'onda quadra con frequenza controllata da un segnale è stata trovata una soluzione utilizzando i blocchi messi a disposizione nella libreria, mostrata in figura 4.8. La generazione dell'onda quadra avviene con l'utilizzo del blocco VCO (Voltage Controlled Oscillator) [20] il quale genera un segnale sinusoidale con frequenza pari alla frequenza normalizzata posta in ingresso ($f_{out} = \frac{f_{in}}{f_s}$), l'uscita di questo blocco viene posta in ingresso ad un blocco di comparazione che compara il segnale con il valore 0, se il segnale è positivo in uscita il blocco restituisce il valore 1, altrimenti se il segnale è negativo restituisce il valore 0. In questo modo all'uscita del blocco di comparazione si ottiene un segnale ad onda quadra con frequenza f_{in} e $delta = 50\%$.
Per ottenere invece il filtraggio tempo variante e la stima della frequenza fondamentale del segnale è stata sfruttata la possibilità di creare dei blocchi personalizzati data da SigmaStudio+.

4.4.1 Creazione dei blocchi non presenti

SigmaStudio+ mette a disposizione dell'utente il blocco $Algorithm\ Designer$ [21], questo blocco permette di creare dei blocchi personalizzati che andranno ad eseguire il codice

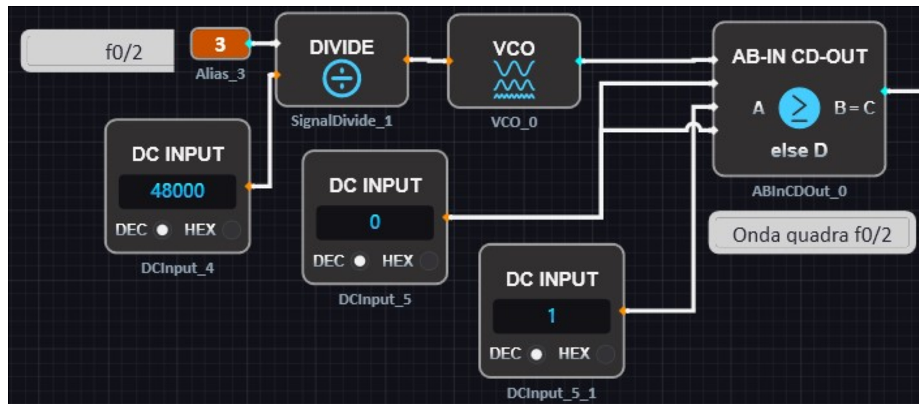

Figura 4.8: Generazione dell'onda quadra a frequenza $\frac{f_0}{2}$

scritto dal programmatore. Il codice che andrà ad eseguire il blocco deve essere scritto in linguaggio C seguendo le convenzioni imposte dal tool [22], durante la creazione è anche necessario impostare: i segnali di input/output, i parametri di lavoro, la quantità memoria di stato utilizzata e l'interfaccia grafica del blocco [23].
Tramite $Algorithm\ Designer$ sono stati implementati su SigmaStudio+, facendo un porting degli algoritmi utilizzati nell'ambiente MATLAB:

- Filtro passa banda tempo variante
- Algoritmo di stima del pitch YIN

Filtro passa banda tempo variante

La creazione del blocco del filtro passa banda tempo variante segue il codice della funzione MATLAB presentata in figura 3.1, il codice della funzione è stato tradotto in linguaggio C adottando le convenzioni di programmazione imposte dal tool ed adattando il processing a blocchi di segnale.
Per questa implementazione sono stati impostati:

- 3 Input
  - Segnale audio
  - Segnale di controllo della frequenza
  - Segnale di controllo del fattore di qualità

- 1 Output
  - Segnale audio filtrato
- 2 spazi di *State Memory*
  - Per salvare i coefficienti del filtro tra l'elaborazione di un blocco e quello successivo.

È fondamentale utilizzare la memoria di stato per salvare i dati che devono essere mantenuti tra una chiamata della funzione e quella successiva, questo perché il compilatore fornisce un puntatore alla memoria riservata durante la fase di creazione del blocco, utilizzando delle variabili questo non è assicurato. L'accesso ai dati di input, output e memoria, viene fornito dalla "struct" posta in ingresso alla funzione, la quale contiene anche i parametri dello schema del progetto quali: frequenza di campionamento e dimensione dei blocchi, in questo modo è possibile rendere l'implementazione del blocco indipendente dalla frequenza e dalla dimensione dei blocchi utilizzata nel progetto [22]. In figura 7.2 viene riportato il codice del blocco del filtro passa banda tempo variante implementato tramite *Algorithm Designer* su SigmaStudio+.

Stima del pitch

La creazione del blocco di Stima del pitch va a seguire la funzione MATLAB dell'algoritmo YIN, anche in questo caso il codice è stato tradotto in C e riadattato per il processamento a blocchi del segnale, infatti nel caso dell'implementazione MATLAB in ingresso alla funzione si trova direttamente un blocco di segnale di dimensione `windowL`, in questo caso i blocchi processati non sono di dimensione `windowL` ma dipendenti dalle impostazioni del progetto (nel caso in studio 64), quindi per ottenere un blocco di dimensione maggiore è necessario mantenere in memoria i restanti `windowL-blockSize` campioni di segnale. La soluzione adottata per gestire questa memorizzazione è quella di utilizzare una porzione di memoria di stato *StateB* per mantenere in memoria i precedenti campioni tramite un array di variabili float che viene aggiornato ad ogni chiamata della funzione, rappresentato in figura 4.9.

```
algBlockSize = windowL; //windowL definita come costante
prjBlockSize =
    pBlkAlgoInfo->pInputs[0].pBlockProperties->nBlockSize;
buffer = (float*)(pBlkAlgoInfo->pStateB);
pInput = pBlkAlgoInfo->pInputs[0].pSamples;
for(i = 0;i<algBlockSize-prjBlockSize;i++){
    buffer[i] = buffer[i+prjBlockSize];
}
for(i = 0;i<prjBlockSize;i++){
    buffer[algBlockSize-prjBlockSize+i] = pInput[i];
}
```

Figura 4.9: Memorizzazione dei campioni all'interno dell'algoritmo di stima del pitch

In questa implementazione la stima della frequenza fondamentale viene eseguita ad ogni chiamata della funzione, quindi per ogni blocco di dimensione `prjBlockSize` di campioni in ingresso, questo porta ad avere un fattore di overlap delle finestre pari a $\frac{algBlockSize}{prjBlockSize}$. Per questa implementazione sono stati impostati:

- 2 Input
  - Segnale audio
  - Valore `yinThreshold`
- 1 Output
  - Segnale di controllo f_0
- Spazi di memoria $stateB$ e $stateC$
  - 1024 spazi di memoria $stateB$ per la memorizzazione dei campioni.
  - 1 spazio di memoria $stateC$ per la memorizzazione dell'ultimo valore di frequenza stimato.

Il restante codice segue l'algoritmo YIN presentato in precedenza, con la condizione di mantenere il valore di frequenza stimata precedentemente nel caso non riesca ad eseguire una stima. Alla fine della funzione viene caricato su tutti i sample del blocco di dimensione `prjBlockSize` in uscita il valore di f_0 stimato.

L'intera implementazione di questo blocco tramite *Algorithm Designer* non viene riportata per la sua lunghezza ma è possibile trovarla in [8].

4.4.2 Implementazione

In seguito alla creazione dei blocchi viene riportata in figura 4.10 l'implementazione dell'effetto Octaver nell'ambiente di sviluppo SigmaStudio+. Si sottolineano in questa implementazione i parametri del blocco di stima del pitch non visibili: `windowL=512`, e `fMin=100Hz`.

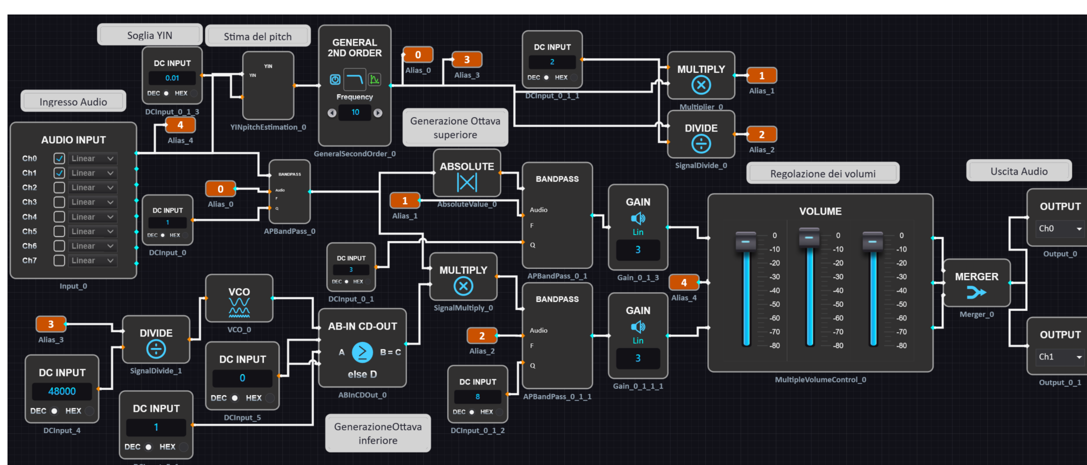

Figura 4.10: Schema dell'effetto Octaver implementato in SigmaStudio+

4.5 Problemi riscontrati e soluzioni

Con una dimensione della finestra di 512 campioni l'effetto sembra funzionare correttamente con dei toni puri sinusoidali in ingresso mentre nascono dei problemi quando l'effetto viene utilizzato con dei suoni di chitarra: l'uscita audio ottenuta è degradata dagli artefatti dovuti alle irregolarità del segnale di controllo f_0. Inoltre mancano alcune generazioni delle ottave oppure sono errate: questo è dovuto alle stime errate della frequenza fondamentale da parte del blocco di stima del pitch. Il risultato ottenuto è mostrato in figura 4.11 dove vengono evidenziati i problemi riscontrati.

Il risultato ottenuto era prevedibile poiché i problemi nell'andamento del segnale f_0 erano già stati riscontrati nelle simulazioni MATLAB dell'effetto con l'utilizzo di una fine-

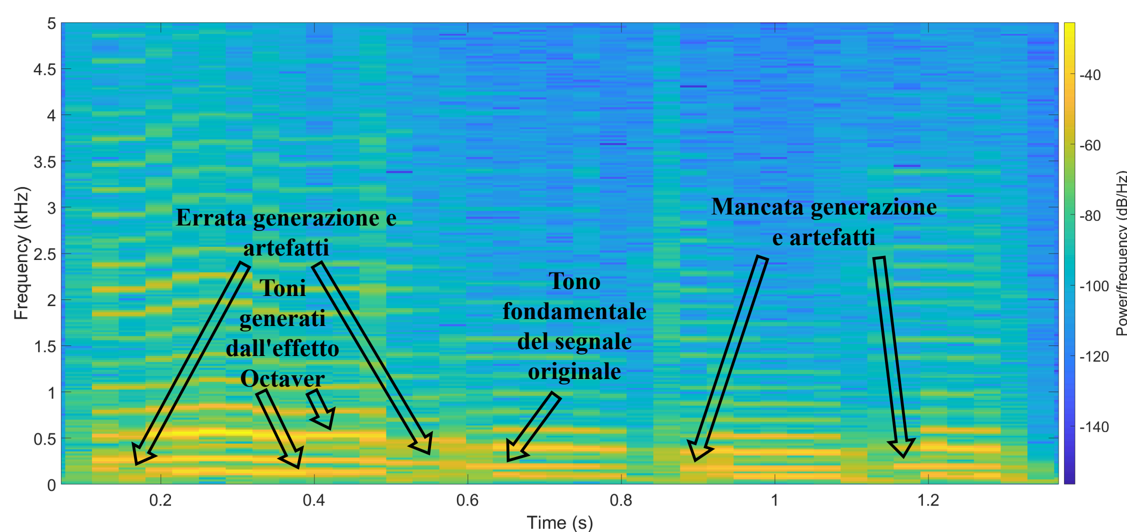

Figura 4.11: Risultato della prima implementazione dell'effetto Octaver real-time con la stima effettuata su 512 campioni

stra di 512 campioni, questi problemi sono stati messi in risalto con l'applicazione dell'effetto in real-time ed è quindi necessario trovare una soluzione per ottenere il corretto funzionamento dell'effetto.

4.5.1 Problema nell'implementazione del blocco di stima del pitch

Riguardo all'implementazione presentata in figura 4.10 sono nati dei problemi relativi al blocco di stima del pitch (YIN). Come sottolineato alla fine del capitolo 3 il funzionamento dell'effetto Octaver è totalmente dipendente dalla stima della frequenza fondamentale ed il comportamento dell'algoritmo di stima del pitch è fortemente dipendente dalla dimensione della finestra utilizzata per calcolare la stima.
Il blocco di stima del pitch creato con `windowL=1024` porta a dei problemi nell'esecuzione del programma sulla scheda, i più rilevanti sono:

- Problema nel caricamento del programma sulla scheda
- Assenza di audio in uscita
- Presenza di segnali indesiderati in uscita

Per risolvere questi problemi sono state provate diverse soluzioni nella creazione del blocco mirando ad ottimizzare il codice C e l'utilizzo della memoria (es.Utilizzo di un array circolare, split dell'array su due memorie), le soluzioni provate non sono riuscite a risolvere il problema ed è quindi stato inizialmente testato l'effetto utilizzando una finestra con `windowL=512` (con questa dimensione il blocco di stima del pitch funziona).

Soluzione adottata

La soluzione trovata per risolvere il limite dato dalla finestra di 512 campioni per il blocco di stima del pitch è stata quella di decimare il segnale all'ingresso del blocco di un fattore 2 attraverso il blocco $Down\ Sampler$ fornito dalla libreria di SigmaStudio+, per poi interpolare di un fattore 2 il segnale di controllo f_0 filtrato all'uscita del filtro passa basso con il relativo blocco $Up\ Sampler$. Dall'uscita del blocco di interpolazione il segnale viene poi utilizzato come in precedenza, il nuovo schema viene riportato in figura 4.12.

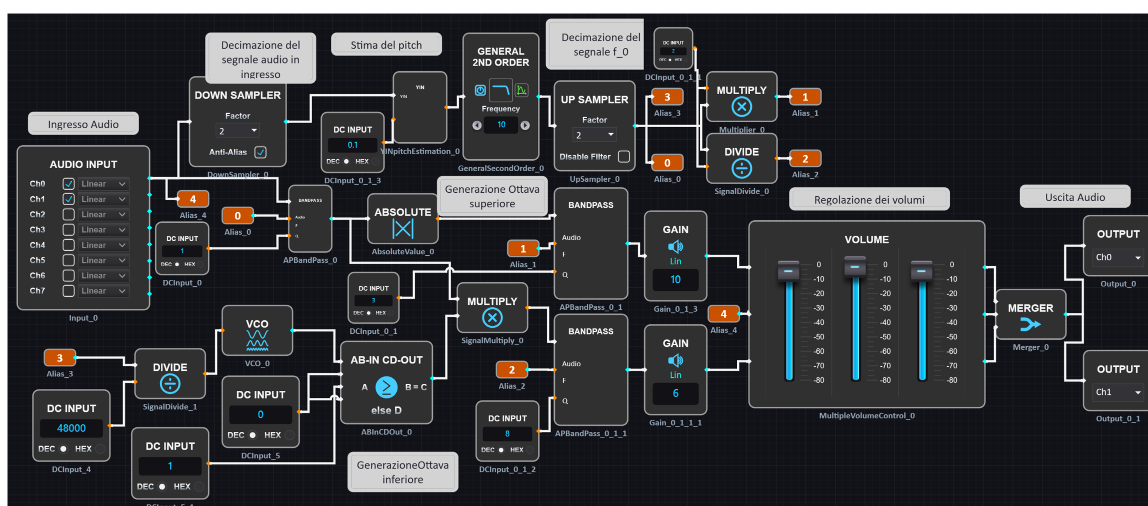

Figura 4.12: Schema della seconda implementazione su SigmaStudio+ dell'effetto Octaver

Il valore di soglia del blocco di stima del pitch è stato scelto in modo sperimentale analizzando quale valore portasse ad un migliore segnale in uscita eseguendo dei test con diversi sample di chitarra. Alcuni dei risultati dei test eseguiti sono riportati nelle figure 4.14, 4.15, 4.16. I test con differenti valori di `yinThreshold` sono stati resi semplici dalla possibilità di effettuare un tuning in tempo reale dei parametri offerta da SigmaStudio+. Il valore

`yinThreshold=0.1` si è rivelato il migliore dal punto di vista della regolarità del segnale e delle stime errate, valori minori portano a mancate generazioni delle ottave, mentre valori maggiori di `yinThreshold` introducono delle irregolarità nel segnale f_0 andando a generare degli artefatti nel segnale audio in uscita.

4.5.2 Problema dei volumi delle ottave generate

Un altro problema che è stato notato durante i test dell'effetto Octaver real-time è stato quello di avere il volume dei segnali dei toni generati variabili in base al segnale in ingresso e di norma molto più bassi rispetto al livello del segnale in ingresso. Questo problema comporta la necessità di regolare i guadagni ad ogni cambiamento del tipo di segnale in ingresso per ottenere i medesimi livelli in uscita oltre che a evitare il clipping del segnale.

Soluzione

La soluzione provata per risolvere questo problema è stata quella di utilizzare i blocchi messi a disposizione da $SigmaStudio+$: Peak Compressor, Gain e Limiter seguendo lo schema utilizzato mostrato in figura 4.13. In questo modo il suono generato all'uscita dei filtri passa banda relativi alla generazione delle ottave viene: compresso, amplificato e limitato ad un valore in modo da prevenire il clipping e ridurne la dinamica, . Infine rimane comunque a disposizione dell'utente la possibilità di variare i volumi dei 3 segnali (ingresso, ottava superiore ed inferiore) tramite il mixer visibile nello schema in figura 4.12.

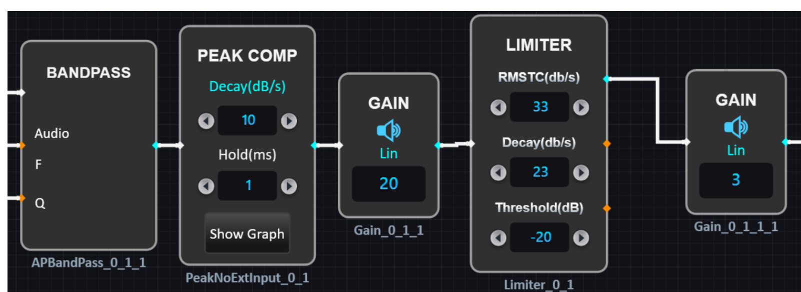

Figura 4.13: Compressione del livello dei toni generati

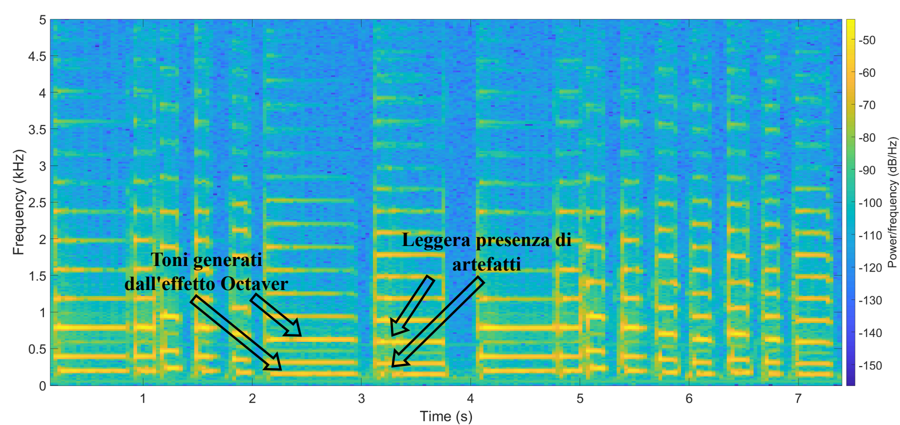

Figura 4.14: Test con schema rappresentato in figura 4.12 e `yinThreshold=0.22`

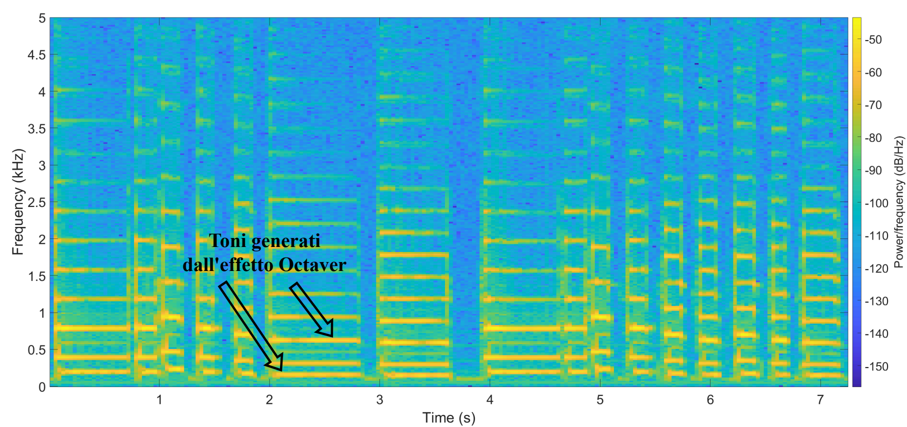

Figura 4.15: Test con schema rappresentato in figura 4.12 e `yinThreshold=0.1`

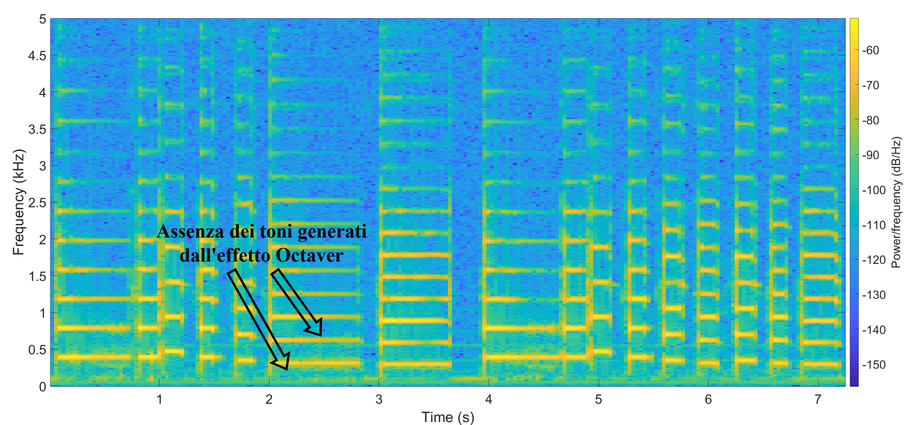

Figura 4.16: Test con schema rappresentato in figura 4.12 e `yinThreshold=0.01`

5 Test e validazione dell'implementazione real-time

In questo capitolo vengono mostrati i risultati dei test in real-time dell'effetto e confrontati con i risultati delle simulazioni MATLAB.

5.1 Metodologia di test

Per eseguire il test dell'effetto in real-time implementato sulla piattaforma SC589 sono state utilizzate delle tracce audio contenenti dei segnali di test:

- Chirp ($f_0 = [100Hz - 1500Hz]$) figura 5.1
- Sample di chitarra confrontato con la simulazione MATLAB figura 5.4

Questi segnali di test sono stati riprodotti dal lettore musicale del computer e posti in ingresso alla sheda SHARC SC589 utilizzando un cavo jack $3.5mm$, il segnale in uscita elaborato dalla scheda è stato campionato da un altro computer utilizzando l'ingresso di linea collegato tramite un altro cavo jack $3.5mm$ all'uscita jack della scheda, utilizzando una frequenza di campionamento $f_s = 44.1kHz$. I segnali campionati sono stati salvati in file "wav" per poter essere analizzati in ambiente MATLAB.

5.2 Risultati dei test

5.2.1 Chirp

Andando ad osservare il risultato ottenuto con un segnale sinusoidale che varia linearmente la sua frequenza nel tempo (figura 5.1), si può notare come l'effetto funzioni correttamente fino ad una frequenza di circa 1kHz, dopo questa frequenza vengono generati degli artefatti notevolmente visibili nell'andamento dello spettro del segnale dovuti

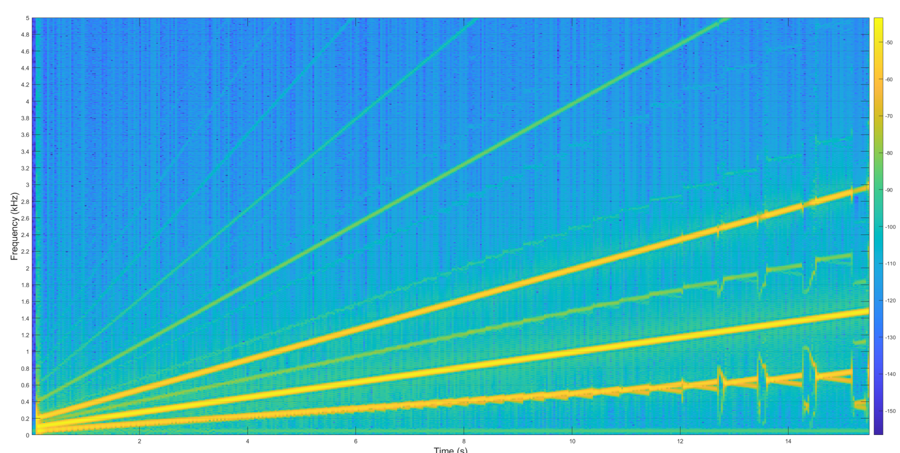

Figura 5.1: Effetto applicato ad un chirp sinusoidale $f_0 = [100Hz - 1500Hz]$

a delle stime errate della frequenza fondamentale. Da questa analisi può essere individuato un limite dell'implementazione real time dell'effetto che quindi sembra funzionare correttamente per note con $f_0 < 1kHz$.

5.2.2 Confronto con simulazione MATLAB

Come ultima analisi viene riportato un test comparativo tra l'effetto Octaver implementato sulla piattaforma real-time e l'effetto Octaver simulato nell'ambiente MATLAB, per fare questo test entrambi gli effetti sono stati applicati sullo stesso sample di chitarra, del quale lo spettrogramma è mostrato in figura 5.2.
I risultati ottenuti dall'applicazione dell'effetto sono riportati in:

- Figura 5.3 effetto MATLAB
- Figura 5.4 effetto real-time su piattaforma SC589

Confrontando il risultato dell'effetto Octaver real-time con il risultato ottenuto tramite la simulazione MATLAB è possibile osservare come il comportamento ottenuto sia pressoché uguale, questo dimostra il corretto funzionamento dell'implementazione real-time sulla piattaforma SHARC SC589. Può essere notata una piccola differenza nel risultato real-time dell'effetto rispetto a quello simulato, probabilmente dovuta ai rumori e alle non idealità nel caso dell'implementazione reale, infatti il segnale generato è leggermente più sporco a livello armonico ma la qualità audio non è compromessa.

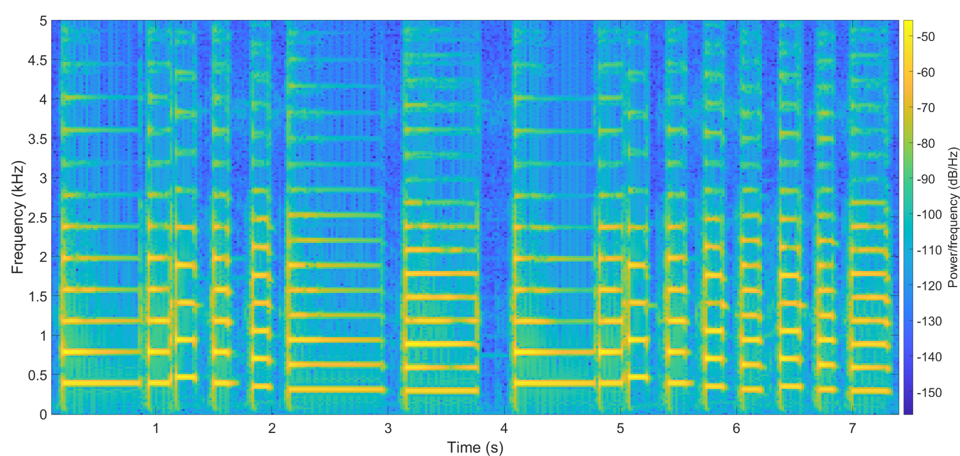

Figura 5.2: Spettrogramma del segnale in ingresso sample di chitarra

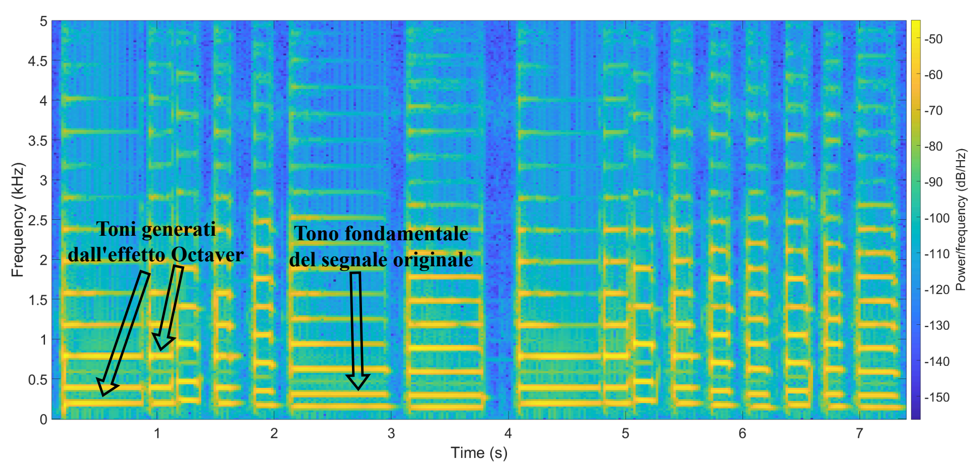

Figura 5.3: Effetto MATLAB applicato al sample di chitarra

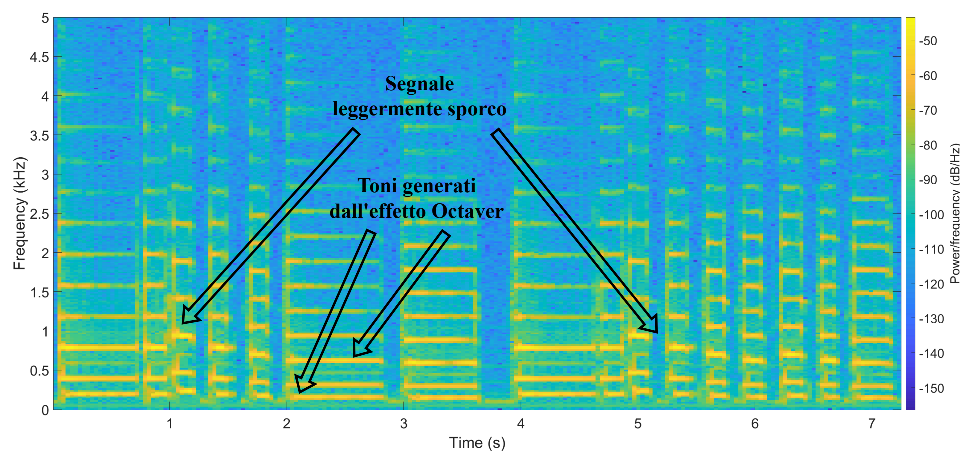

Figura 5.4: Effetto real-time applicato al sample di chitarra

6 Conclusioni

Lo sviluppo dell'effetto Octaver in real-time tramite SigmaStudio+ sulla piattaforma Analog Devices SC589, presentato in questo elaborato, mostra come SigmaStudio+ renda semplice ed intuitiva la programmazione di piattaforme DSP del produttore Analog Devices anche da parte di chi non ha esperienza nella scrittura del software di processori DSP. L'approccio di programmazione a blocchi fornito da SigmaStudio+ semplifica notevolmente la programmazione di un processore DSP, infatti vengono forniti all'interno della libreria molti blocchi che svolgono le funzioni più utilizzate nell'ambito del processing audio (filtri crossover, compressori, processing multirate, etc.), in questo modo il programmatore deve pensare solo a come utilizzare questi blocchi collegandoli tra di loro ed impostando i parametri, senza dover pensare al codice relativo all'implementazione. L'utilizzo della programmazione a blocchi presenta però un punto debole quando si necessita di una funzione che non è presente tra quelle messe a disposizione, questo viene sopperito in parte dalla possibilità di creare dei blocchi personalizzati, data dall'ambiente di sviluppo utilizzato. Con questa possibilità l'utente può comunque riuscire ad implementare una funzione mancante, il problema nasce successivamente all'implementazione di alcune funzioni, come la stima del pitch in questo elaborato. Infatti può succedere che creando un blocco funzionante con determinati parametri di utilizzo (memoria e operazioni), all'aumento della memoria utilizzata oppure del numero di operazioni effettuate, questo blocco smetta di funzionare iniziando a dare problemi di diverso tipo all'intero programma e non solo alla parte collegata direttamente ad esso. A questo problema si aggiunge l'assenza di una funzione di debug del codice, rendendo difficile l'individuazione della causa del problema.

Si è voluto mettere l'accento sull'implementazione in ambiente SigmaStudio+ e le rela-

tive problematiche riscontrate perché questo elaborato si pone anche come supporto per chi volesse utilizzare questo ambiente di sviluppo per programmare una piattaforma DSP SHARC.

Comunque, tramite dei compromessi l'implementazione dell'effetto Octaver sulla piattaforma è riuscita, infatti l'effetto oltre che con segnali audio di chitarra registrati è stato anche testato con un segnale audio proveniente direttamente da una chitarra elettrica ed il risultato ottenuto è stato quello desiderato tipico di un effetto Octaver.

La struttura dell'effetto Octaver presentata in questo elaborato è ridotta al minimo necessario per il funzionamento di questo tipo di effetto, questo con l'obiettivo di non complicare ulteriormente la sua implementazione nell'ambiente SigmaStudio+, fornendo al contempo una base di partenza per eventuali sviluppi futuri. Durante lo studio dell'implementazione dell'effetto sono stati illustrati alcuni dei problemi riscontrati e delle possibili soluzioni, questo per mostrare a cosa prestare attenzione durante uno sviluppo di un effetto di questo tipo. L'implementazione presentata può essere utilizzata come base per sviluppi futuri relativi sia all'algoritmo di base dell'effetto che alla sua implementazione in SigmaStudio+. Gli sviluppi dell'algoritmo dell'effetto possono mirare a migliorare il tracking del pitch del segnale audio, la generazione dei toni ad esempio andando ad utilizzare tecniche di pitch shifting in modo da ricreare un suono più naturale oppure l'implementazione di un effetto Octaver polifonico. Mentre per quanto riguarda l'implementazione dell'effetto in SigmaStudio+, possibili sviluppi includono una implementazione che vada ad utilizzare entrambi i core dello SHARC SC589 ed una ottimizzazione generale dello schema e dei blocchi creati.

7 Appendice

7.1 Codice MATLAB

```
for i=1 : L     % L numero di campioni del segnale
    % stima del pitch
    if((mod(i,windowL/windowOverlap)==1)&&(i+windowL<L))
        f0nf = YIN(data(i:i+windowL),Fs,fMin,yinThreshold);
        if (f0nf < fMin )
            f0nf = f0;
        else
            f0 = f0nf;
        end
    end
    % Filtro il segnale in ingresso attorno alla f0 stimata
    fb = f0/Q;
    [fundamental(i),xhf] = myApbandpass(data(i),2*f0/(Fs),fb/Fs,xhf);
    % generazione dell'ottava superiore
    rawH = abs(fundamental(i));
    %Segnale a dente di sega con frequenza f0
    increment = f0/(2*Fs);
    moduloCounter = moduloCounter + increment;
    if moduloCounter > threshold
        moduloCounter = moduloCounter - threshold;
    end
    % fundamental * onda quadra a frequenza f0/2
    if ((moduloCounter < sThreshold))
        rawL = fundamental(i);
    else
        rawL = 0;
    end
    fbl = f0/Ql;
    fbh = f0/Qh;
    % estrazione toni ottave
    [low(i),xhl] = myApbandpass(rawL,f0/(Fs),fbl/Fs,xhl);
    [high(i),xhh] = myApbandpass(rawH,4*f0/(Fs),fbh/Fs,xhh);
end
out = gM*data + gH*high + gL*low;
```

Figura 7.1: Algoritmo MATLAB dell'effetto Octaver

7.2 Codice Algorithm Designer

```
#include "adi_ss_extmod.h"
#include <math.h>
#pragma section("seg_pmco")
void BPROCESS_APBandPass(SSBlockAlgo* pBlkAlgoInfo)
{
    int sample, blockSize;
    float fs,x,c,d,xh_new,ap_y;
    float Wc,Wb;
    float *pInput, *pOutput, *fInput, *QInput;
    float *memory;
    float xh[2];
    float PI = 3.14159265358979323846;
    //puntatore alla stateMemory
    memory = (float*)(pBlkAlgoInfo->pStateB);
    blockSize =
        pBlkAlgoInfo->pInputs[0].pBlockProperties->nBlockSize;
    //frequenza di campionamento
    fs = pBlkAlgoInfo->pInputs[0].pBlockProperties->nSamplingRate;
    // ricarico gli ultimi coefficienti del filtro
    xh[0] = memory[0];
    xh[1] = memory[1];
    // segnale audio in ingresso -> input 1
    pInput = pBlkAlgoInfo->pInputs[0].pSamples;
    // segnale di controllo in ingresso -> input 2
    fInput = (float*)pBlkAlgoInfo->pInputs[1].pSamples;
    // segnale di controllo in ingresso -> input 3
    QInput = (float*)pBlkAlgoInfo->pInputs[2].pSamples;
    // segnale audio in uscita
    pOutput = pBlkAlgoInfo->pOutputs[0].pSamples;
    //ciclo su tutti i sample del blocco da filtrare
    for(sample = 0; sample < blockSize; sample++)
    {
        Wc = 2*fInput[sample]/fs;
        Wb = Wc/QInput[sample]; // Q = f0/BW -> BW[Hz] = f0[Hz]/Q
        x = pInput[sample];
        // calcolo dei coefficienti del filtro
        c = (tan(PI*Wb/2)-1) / (tan(PI*Wb/2)+1);
        d = -cos(PI*Wc);
        xh_new = x - d*(1-c)*xh[0] + c*xh[1];
        ap_y = -c * xh_new + d*(1-c)*xh[0] + xh[1];
        xh[1] = xh[0];
        xh[0] = xh_new;
        pOutput[sample] = 0.5 * (x - ap_y);
    }
    // salvataggio dei coefficienti per la successiva elaborazione
    memory[0] = xh[0];
    memory[1] = xh[1];
}
```

Figura 7.2: Funzione SigmaStudio del filtro passa banda tempo variante

Bibliografia

[1] Analog Devices, Inc. (2024) Sharc audio module (revision 1.4). Ultimo accesso: 2 luglio 2024. [Online]. Available: https://wiki.analog.com/resources/tools-software/sharc-audio-module/hardware/main-board/rev1_4

[2] ——. (2024) Audio input-output modes. Ultimo accesso: 1 luglio 2024. [Online]. Available: https://wiki.analog.com/resources/tools-software/sigmastudiov2/targetintegration/targetapplication/audioiomodes

[3] O. Lähdeoja, B. Navarret, S. Quintans, A. Sedes, O. Lähdeoja, and A. Sédes, "The electric guitar: An augmented instrument and a tool for musical composition," *Journal of interdisciplinary music studies*, vol. 4 2, pp. 37–54, 01 2010.

[4] Roland Corporation. (2024) Oc-5. Ultimo accesso: 4 luglio 2024. [Online]. Available: https://www.boss.info/it/products/oc-5/

[5] tc electronic. (2024) Sub 'n' up octaver. Ultimo accesso: 4 luglio 2024. [Online]. Available: https://www.tcelectronic.com/product.html?modelCode=0709-AGR

[6] U. Zölzer, Ed., *DAFX: Digital Audio Effects*, 2nd ed. Wiley, 2011.

[7] A. de Cheveigné and H. Kawahara, "Yin, a fundamental frequency estimator for speech and music." *The Journal of the Acoustical Society of America*, vol. 111 4, pp. 1917–30, 2002. [Online]. Available: https://api.semanticscholar.org/CorpusID:1607434

[8] G. D. Carlo. (2024) Octaverfx. [Online]. Available: https://github.com/Gabry-xdvr/OctaverFX.git

[9] Analog Devices, Inc. (2024) Sigmastudio+. Ultimo accesso: 2 luglio 2024. [Online]. Available: https://www.analog.com/en/resources/evaluation-hardware-and-software/software/sigmastudio-plus.html

[10] ——. (2024) Sharc audio module (adzs-sc589-mini). Ultimo accesso: 2 luglio 2024. [Online]. Available: https://wiki.analog.com/resources/tools-software/sharc-audio-module

[11] ——. (2024) Adsp-sc5xx sigmastudio+ support. Ultimo accesso: 1 luglio 2024. [Online]. Available: https://wiki.analog.com/resources/tools-software/sigmastudiov2/supportedplatforms/adspsc5xx

[12] ——. (2024) Sigmadsp audio processors. Ultimo accesso: 1 luglio 2024. [Online]. Available: https://www.analog.com/en/product-category/sigmadsp-audio-processors.html

[13] ——. (2024) Sigmastudio+ wiki. Ultimo accesso: 1 luglio 2024. [Online]. Available: https://wiki.analog.com/resources/tools-software/sigmastudiov2

[14] ——. (2024) Sigmastudio+ sc5xx hardware setup. Ultimo accesso: 1 luglio 2024. [Online]. Available: https://wiki.analog.com/resources/tools-software/sigmastudiov2/gettingstarted/adsp-215xx_and_adsp-sc5xx

[15] ——. (2024) Emulator-adsp. Ultimo accesso: 1 luglio 2024. [Online]. Available: https://www.analog.com/en/resources/evaluation-hardware-and-software/evaluation-boards-kits/emulators.html

[16] ——. (2024) Eval-adusb2ebz. Ultimo accesso: 1 luglio 2024. [Online]. Available: https://www.analog.com/en/resources/evaluation-hardware-and-software/evaluation-boards-kits/eval-adusb2ebz.html

[17] ——. (2024) Pc software setup adsp-214xx/adsp-215xx/adsp-sc5xx. Ultimo accesso: 1 luglio 2024. [Online]. Available: https://wiki.analog.com/resources/tools-software/sigmastudiov2/gettingstarted/swsetup

[18] ——. (2024) Crosscore embedded studio. Ultimo accesso: 1 luglio 2024. [Online]. Available: https://www.analog.com/en/resources/evaluation-hardware-and-software/software/adswt-cces.html

[19] ——. (2024) Load application using cces. Ultimo accesso: 1 luglio 2024. [Online]. Available: https://wiki.analog.com/resources/tools-software/sigmastudiov2/gettingstarted/load_executable_using_cces

[20] ——. (2024) Sigmastudio+ voltage controlled oscillator. Ultimo accesso: 2 luglio 2024. [Online]. Available: https://wiki.analog.com/resources/tools-software/sigmastudiov2/modules/sources/vco

[21] ——. (2024) Sigmastudio+ algorithm designer. Ultimo accesso: 2 luglio 2024. [Online]. Available: https://wiki.analog.com/resources/tools-software/sigmastudiov2/usingsigmastudio/designermodule

[22] ——. (2024) Algorithm designer coding conventions. Ultimo accesso: 2 luglio 2024. [Online]. Available: https://wiki.analog.com/resources/tools-software/sigmastudiov2/usingsigmastudio/algorithm_designer_coding_conventions

[23] ——. (2024) Algorithm designer designing module. Ultimo accesso: 2 luglio 2024. [Online]. Available: https://wiki.analog.com/resources/tools-software/sigmastudiov2/usingsigmastudio/designing_module