POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Informatica
Scuola di Ingegneria dell'Informazione

POLO TERRITORIALE DI COMO

STUDIO SULL'IMPLEMENTAZIONE DEGLI ALGORITMI PER IL MUSICAL INSTRUMENTS ED IL SOUND REINFORCEMENT BASATO SU UN PROCESSORE MULTICORE

RELATORE: Prof. Augusto Sarti

CORRELATORE: Ing. Franco Gazzilli

CANDIDATO: Michele Aretino

MATRICOLA: 835578

Anno Accademico 2016-2017

Sommario

In questa tesi si propongono degli algoritmi che simulano il comportamento dei processori di effetto utilizzati per il trattamento dei segnali audio e si discuterà dell'implementazione di questi algoritmi nei più moderni processori di segnali digitali (DSP, Digital Signal Processor).

Il lavoro svolto nella tesi consiste, a partire da un algoritmo noto per la simulazione del comportamento di un processore di effetto (in questo caso, un riverbero) nell'implementare tale algoritmo in un DSP (Digital Signal Processor).

Per lo scopo si è scelto di implementare un algoritmo per la riverberazione che, pur non appartenendo allo stato dell'arte, è comunque molto gradito ai musicisti per le caratteristiche timbriche che presenta. Nel nostro caso, è stato scelto un algoritmo corrispondente a una tipologia di riverbero avente una timbrica particolare, caratterizzato da una struttura basata su combinazioni di filtri allpass e comb [6].

L'algoritmo in questione, il cui schema a blocchi è discusso nel capitolo 3, è stato sviluppato dalla ditta Montarbo s.r.l. ed è principalmente pensato per venire incontro alle esigenze dei musicisti alla ricerca di un suono particolare. L'algoritmo descrive un riverbero appartenente alla categoria "Ambience", cioè un riverbero con caratteristiche acustiche assimilabili a quelle di una stanza di piccole/medie dimensioni, e ciascuno dei suoi parametri fondamentali è regolabile a partire da un gruppo di filtri comb e allpass.

Sebbene siano state prese, come idea di base per il riverbero, le caratteristiche acustiche di una stanza reale, i parametri scelti per l'algoritmo trattato presentano proprietà timbriche che non corrispondono a nessun ambiente chiuso reale; questa scelta è voluta in quanto questo effetto è pensato per dare al suono un effetto "vintage" particolarmente gradito ai musicisti, piuttosto che per simulare un ambiente particolare.

Una volta definito lo schema di progettazione del riverbero, si sviluppa l'algoritmo su XTime Composer, definendo le istruzioni sia in linguaggio di alto livello (chiamato xC) sia in linguaggio macchina in modo da formare l'interfaccia tra l'elaboratore, in cui sono definite le istruzioni, e il DSP, che si occupa di processare il segnale in ingresso secondo le istruzioni preventivamente definite.

Il risultato del processamento del segnale da parte del DSP, ovvero il segnale riverberato che si vuole ottenere, consisterà in una successione di repliche del segnale di ingresso (che, per semplicità, si tratta di un impulso unitario) che occorrono in corrispondenza di diversi valori di frequenza correlati ai parametri dei singoli building blocks dello schema.

Dal momento che XTime Composer elabora vettori a n dimensioni come segnali in ingresso (dove il numero di campioni n corrisponde alla durata temporale del segnale), le istruzioni descrivono quello che succede in un singolo ciclo di processamento, per cui, una volta terminata la loro esecuzione per il primo ciclo, esse verranno rielaborate nuovamente per i restanti n-1 cicli.

La particolarità di questa tesi consiste implementare l'algoritmo su un DSP innovativo, realizzato dalla ditta XMOS [8]. I DSP di produzione della XMOS prendono il nome di XCORE e consentono di elaborare in alta definizione il segnale audio in ingresso.

Una volta scelto l'algoritmo da implementare, si considera una struttura composta da diversi DSP che lavorano in maniera parallela.

I dati audio provenienti da sorgenti esterne vengono inviati in streaming al DSP tramite un dispositivo di interfacciamento chiamato I2S (di cui si parla più in dettaglio nel capitolo 4) e, una volta dentro, vengono elaborati in alta definizione secondo le istruzioni che definiscono i distinti passi dell'algoritmo di riverberazione scelto.

L'algoritmo scelto per i nostri scopi è un algoritmo piuttosto datato di cui non si dispone più della macchina fisica per l'impostazione dei parametri, per cui si è deciso di simulare tale algoritmo in MATLAB, poi su XTime Composer, che è il linguaggio utilizzato per l'interfacciamento dei DSP XCORE [8,9] al computer (se ne parla più dettagliatamente nel capitolo 3) e, infine, confrondando le due risposte impulsive in uscita.

Nella tesi viene, pertanto, realizzato l'algoritmo del riverbero in MATLAB e, applicando al suo ingresso un impulso unitario, si ricava un segnale di uscita che verrà confrontato con il segnale in uscita dal processore XCORE; nel capitolo 7, in cui viene effettuata la validazione, si riportano in tabella i valori in uscita da MATLAB e dal processore XCORE e questi valori si presentano tra loro molto simili.

Abstract

In this thesis we propose algorithms that simulate the behavior of the effect processors used for the audio signal processing and we will discuss the implementation of these algorithms in the most modern digital signal processors (DSP).

The work carried out in the thesis consists, starting from an algorithm known for simulating the behavior of an effect processor (in this case, a reverb algorithm) in implementing this algorithm in a DSP.

For the purpose it was decided to implement an algorithm for reverberation; although the algorithm does not belong to the state of the art, it is however very pleasing to the musicians due to the timbral characteristics it presents. In our case, an algorithm corresponding to a type of reverb having a particular timbre has been chosen, characterized by a structure based on combinations of allpass and comb filters [6].

The algorithm in question, whose block diagram is discussed in chapter 3, was developed by the Montarbo s.r.l. company and is mainly designed to meet the needs of musicians in search of a particular sound. The algorithm describes a reverberation belonging to the "Ambience" category, that is a reverberation with acoustic characteristics similar to those of a small / medium size room, and each of its fundamental parameters is adjustable starting from a group of comb and allpass filters.

Although the acoustic characteristics of a real room have been taken as the basic idea for reverberation, the parameters chosen for the treated algorithm have timbral properties that do not correspond to any real closed environment; this choice is desired because this effect is designed to give the sound a "vintage" effect particularly pleasing to musicians, rather than to simulate a particular environment.

Once the reverb design scheme has been defined, the algorithm is developed on XTime Composer, defining the instructions both in high level language (called xC) and in machine language in order to form the interface between the computer, in which the instructions are defined, and the DSP, which takes care of processing the input signal according to the previously defined instructions.

The result of the processing of the signal by the DSP, i.e. the reverberated signal to be obtained, will consist of a sequence of replicas of the input signal (which, for simplicity, it is a unitary pulse) that occur at different values frequency related to the parameters of the single building blocks of the scheme.

Since XTime Composer processes n sized vectors as input signals (where the number of samples n corresponds to the signal duration), the instructions describe what happens in a single processing cycle, so, once they are finished executing for the first cycle, they will be reprocessed again for the remaining n-1 cycles.

The particularity of this thesis is to implement the algorithm on an innovative DSP, realized by the company XMOS [8]. The DSPs producted by XMOS are called XCORE and allow the audio signal

to be processed in high definition.

Once the algorithm to be implemented has been chosen, it is considered a structure composed of several DSPs that work in parallel.

The audio data coming from external sources are sent in streaming to the DSP through an interfacing device called I2S (which is discussed in more detail in chapter 4) and, once inside, are processed in high definition according to the instructions that define the distinct steps of the chosen reverberation algorithm.

The algorithm chosen for our purposes is a rather dated algorithm that no longer has the physical machine for setting the parameters, so it was decided to simulate this algorithm in MATLAB, then on XTime Composer, which is the language used for interfacing the XCORE DSP to the computer (more in detail in chapter 3) and finally comparing the two impulsive output responses.

In the thesis, therefore, the reverberation algorithm is realized in MATLAB and, by applying a unitary pulse to its input, an output signal is obtained which will be compared with the output signal from the XCORE processor; in chapter 7, where the validation is carried out, the values in output from MATLAB and the XCORE processor are shown in the table and these values are very similar to each other.

RINGRAZIAMENTI

Ringrazio, innanzitutto, il relatore Prof. Augusto Sarti per la pazienza e per la disponibilità dedicatemi durante il periodo di realizzazione della tesi, e il correlatore Ing. Franco Gazzilli della ditta Montarbo s.r.l. per avermi ospitato e seguito in azienda durante il periodo di tirocinio, avendomi così dato la possibilità di entrare in contatto con un ambiente lavorativo e contribuire alla mia formazione professionale. Infine, un ringraziamento alla mia famiglia e agli altri impiegati della Montarbo s.r.l. per la fiducia e l'incoraggiamento durante la realizzazione della tesi.

Indice

Lista delle figure

Lista delle tabelle

CAPITOLO 1

Introduzione

In generale, lo scopo dei processori di effetto è quello di manipolare le proprietà timbriche di un suono, al fine di rafforzarlo o di conferirgli delle caratteristiche acustiche particolari, in base alle esigenze o al gusto personale dei musicisti.

Per quanto concerne i riverberi, ad esempio, essi sono principalmente pensati per riprodurre artificialmente il fenomeno fisico di riflessione delle onde sonore all'interno di un particolare ambiente (che può essere una stanza, una sala concerti, una cattedrale…).

Va detto, però, che molto spesso la reale necessità che spinge i musicisti a ricorrere al riverbero è quella di rafforzare maggiormente il proprio suono, e perché sia possibile conseguire questo risultato è necessario che il riverbero utilizzato presenti delle caratteristiche acustiche che non corrispondono necessariamente a quelle di un ambiente chiuso realmente esistente.

Dal momento che alcuni di questi riverberi non sono presenti in natura, occorre crearli artificialmente; in questo caso, si parla di riverberi digitali [17].

Al giorno d'oggi, col termine "filtri digitali" si fa riferimento a circuiti integrati che permettono di compiere alcune funzioni matematiche su campioni di segnali discreti nel tempo, al fine di modificare alcuni aspetti del segnale in ingresso analizzati.

I filtri digitali sono costituiti fondamentalmente da filtri ritardanti che, appunto, prendono in ingresso un segnale e ne restituiscono in uscita la versione ritardata e attenuata in ampiezza dello stesso. I filtri maggiormente utilizzati in questo contesto sono i filtri a pettine (comb filters), che restituiscono una sequenza di impulsi equispaziati da una certa frequenza (che, appunto, ricordano i denti di un pettine [6]), e i filtri passatutto (allpass filters), che consentono il passaggio di tutte le frequenze del segnale, ma modificano le relazioni di fase tra le diverse frequenze facendo variare la lunghezza della linea di ritardo secondo una certa frequenza [16]. Oltre a questi filtri, spesso si fa ricorso anche a filtri FIR (Finite Impulse Response [15]), caratterizzati da una risposta impulsiva di durata finita, cioè che si annulla dopo una certa quantità di tempo (in genere piccolissima).

Avendo, dunque, a disposizione questi filtri, la progettazione dei riverberi procede seguendo una serie di regole empiriche atte a riprodurre le caratteristiche fisiche fondamentali di un riverbero, indipendentemente dal fatto che si voglia o meno simulare le caratteristiche fisiche di un ambiente chiuso: in primo luogo, si riproducono le early reflections, di solito tramite una rete costituita da filtri FIR, e, successivamente, il gruppo di riverberazione, allo scopo di addensare le prime riflessioni ottenute dalla rete precedente. Ognuno di questi parametri fondamentali del riverbero può essere pilotato esternamente per mezzo di un gruppo di filtri (building blocks) .

Il gruppo di riverberazione viene, di solito, ricreato per mezzo di una rete costituita da filtri comb e allpass, oppure, eventualmente, una Feedback Delay Network (descritta più nel dettaglio nel capitolo 3).

A seconda di come vengono disposti i vari filtri si ottengono diversi risultati, in quanto le riflessioni che si generano in uscita dai vari filtri vengono addensate diversamente. Nel nostro caso, è stato ritenuto opportuno raggruppare, nel gruppo di riverberazione, i filtri comb in parallelo e a monte dei filtri allpass, collegati in serie tra loro.

In ambito digitale, i riverberi sono definiti in termini di algoritmi matematici, caratterizzati da istruzioni in linguaggio macchina e linguaggio ad alto livello scelte in maniera tale da riprodurre il comportamento dei singoli blocchi funzionali (building blocks) che costituiscono lo schema di funzionamento dell'effetto.

La scelta del particolare algoritmo da applicare tiene conto dei diversi riverberi digitali attualmente presenti sul mercato (riverbero a molla [4], a convoluzione...).

Lo stato dell'arte attuale dei riverberi digitali (di cui fanno parte, ad esempio, i riverberi a convoluzione) riguarda algoritmi più complessi dal punto di vista progettuale che richiedono una quantità di calcolo maggiore ma hanno una timbrica diversa e che, in questo caso, non sono stati presi in considerazione. Questi algoritmi, a differenza di quello da noi trattato, presentano un maggior numero di building blocks che li costituiscono.

CAPITOLO 2

Generalità e parametrizzazione dei riverberi

2.1 Riverbero in ambiente fisico [6]

Il processo di riverberazione inizia con la produzione di un suono da parte di una sorgente sonora localizzata in un punto di un ambiente chiuso. L'onda di pressione acustica si espande radialmente, raggiungendo le pareti della stanza, dove l'energia contenuta nell'onda viene in parte assorbita e in parte riflessa.

Tecnicamente, questa energia viene definita come *riverbero*.

Per ragioni pratiche, si supporrà che la propagazione dell'onda all'interno della stanza termina quando l'intensità del fronte d'onda diventa inferiore all'intensità del livello di rumore ambientale.

Assumendo che esista un percorso diretto tra la sorgente e l'ascoltatore, quest'ultimo percepirà il suono diretto per primo e, a seguire, le riflessioni provenienti dalle pareti più vicine che prendono il nome di *prime riflessioni* (*early reflections*).

Dopo poche centinaia di millisecondi, il numero di onde riflesse risulterà molto alto e il resto del decadimento del riverbero sarà caratterizzato da un insieme di echi che viaggiano in tutte le direzioni, la cui densità è indipendente dalla loro posizione nella stanza; questi echi costituiscono la cosiddetta *coda del riverbero* (*late reverberation*, o *tardo riverbero*).

Considerando un campo acustico perfettamente diffuso, generato da una sorgente sonora, la quantità di energia assorbita da parte delle pareti della stanza è proporzionale alla densità di energia del campo acustico stesso e pertanto l'intensità degli echi che costituiscono la coda del riverbero decade esponenzialmente nel tempo.

Si definisce *tempo di riverberazione* (*reverberation time*) il tempo necessario affinché l'intensità del riverbero durante il decadimento sia inferiore di 60 dB al suo livello iniziale.

2.2 Prime riflessioni [6]

Per lo studio delle prime riflessioni si considera, di solito, il modello geometrico della stanza. Partendo dal presupposto che le dimensioni delle pareti della stanza che riflettono le onde sonore siano di gran lunga superiori alla lunghezza d'onda del suono, l'onda sonora può essere rappresentata come un raggio normale alla superficie del fronte d'onda, e ogni volta che il raggio d'onda incide con la superficie di una parete della stanza esso viene riflesso specularmente.

La Fig.1 rappresenta la riflessione di un raggio su una parete; supponendo che la sorgente che genera il suono si trovi in un punto A dello spazio, si è interessati a sapere come si propaga il suono da questo punto ad un osservatore posto in un punto B (non coincidente con A). Per ricavare il

raggio riflesso possiamo considerare la sorgente immagine A', la cui posizione risulta speculare ad A rispetto al piano della parete. A partire da A' si traccia una linea retta in direzione del punto B, che corrisponde al raggio incidente riflesso, speculare con quello proveniente da A.

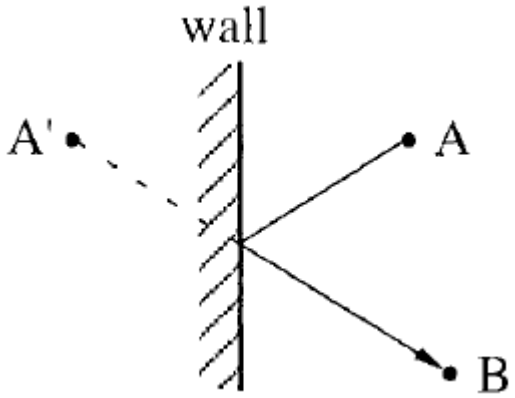


Figura 1 – Singola riflessione di un'onda su una parete e sorgente immagine A'

Il metodo appena descritto per l'individuazione del raggio riflesso è detto *metodo delle sorgenti immagine*.

Il metodo delle sorgenti immagine è sconsigliabile per lo studio delle tarde riflessioni perché in questo caso il numero di sorgenti cresce esponenzialmente ed il modello di riflessione semplificato appena descritto risulterà inaccurato.

Per calcolare la risposta impulsiva nella posizione dell'ascoltatore si sommano i contributi di tutte le sorgenti presenti nella stanza. Ognuna di queste sorgenti contribuisce a creare un impulso ritardato (eco), il cui tempo di ritardo è pari alla distanza tra la sorgente e l'ascoltatore divisa per la velocità del suono.

Tenendo conto dell'espansione sferica del suono (le onde sonore sono, di fatto, onde sferiche), l'ampiezza dell'eco è inversamente proporzionale alla distanza percorsa e direttamente proporzionale al prodotto dei coefficienti di riflessione delle superfici incontrate.

Il modello di Lehnert e Blauert [28] approssima le perdite dipendenti dalla frequenza utilizzando filtri lineari, in maniera tale che lo spettro dell'eco che raggiunge l'ascoltatore sia dato dal prodotto tra le funzioni di trasferimento relative alla storia di quest'eco. Tale relazione è data nella seguente forma:

$$A(\omega) = G(\omega) \prod_{j \epsilon S} \Gamma_j(\omega) \quad (1)$$

Dove $A(\omega)$ è lo spettro dell'eco, S è l'insieme di pareti incontrate dall'onda sonora, Γ_j è la funzione di trasferimento dipendente dalla frequenza che modella la riflessione in corrispondenza della j-esima parete e $G(\omega)$ è la funzione di trasferimento (sempre dipendente dalla frequenza) che modella le perdite dovute all'assorbimento delle pareti e alla propagazione nell'aria del suono.

Le ipotesi semplificative che ci permettono di considerare le riflessioni speculari da parte delle onde sonore a contatto con le pareti non valgono più qualora le dimensioni delle pareti sono almeno pari

a quelle della lunghezza d’onda del suono. In questo caso, il suono riflesso si propagherà in varie direzioni, e questo fenomeno è noto come *diffusione*.

In generale, è noto che la provenienza dei segnali sonori può essere intuita dalle variazioni di pressione sonora che si verificano in corrispondenza del torso, della testa e del padiglione auricolare [29].

Si considera una risposta in frequenza, detta head-related transfer function (HRTF), che descrive questa trasformazione da una specifica posizione della sorgente al timpano. Le HRTF vengono solitamente misurate usando soggetti umani o microfoni a forma di testa (head-shaped microphones) e sono costituite da coppie di risposte, una per l'orecchio sinistro e l’altra per il destro, corrispondenti a un gran numero di posizioni di sorgente attorno alla testa.

Calcolando la funzione di trasferimento binaurale di una stanza usando i modelli geometrici già discussi, dobbiamo convolere ogni eco direzionale con l'HRTF corrispondente alla direzione dell'eco [30]. I segnali direzionali binaurali catturati dalle HRTF sono principalmente la differenza temporale interaurale (ITD) e la differenza di intensità interaurale (IID) che variano in funzione della frequenza. Gli echi provvenienti dalle direzioni laterali (cioè da entrambi i lati dell'ascoltatore) servono a modificare il carattere spaziale del riverbero percepito.

L'ITD di una sorgente sonora posta lateralmente è modellato da un ritardo corrispondente alla differenza nella lunghezza del percorso tra le due orecchie, mentre l'IID può essere modellato per mezzo di un filtraggio passabasso del segnale che arriva all'orecchio opposto (controlaterale).

2.3 Effetti percettivi delle prime riflessioni [6]

Gli effetti percettivi delle prime riflessioni possono essere studiati considerando un campo sonoro costituito da un suono diretto e una singola riflessione ritardata.

Se sia il suono diretto e la prima riflessione si propagano frontalmente in direzione dell’ascoltatore e la riflessione raggiunge quest’ultimo 80msec dopo il suono diretto, allora tale riflessione sarà percepita come un’eco distinta del suono diretto. In caso contrario, cioè se la riflessione impiega meno di 80msec per raggiungere l’ascoltatore dopo il suono diretto, allora quello che si percepirà è un suono dato dalla fusione del suono diretto e della prima riflessione, caratterizzato però da una colorazione tonale dovuta alla cancellazione di fase dei due segnali in un insieme periodico di frequenze.

Talvolta, a seconda del tempo di ritardo, la riflessione potrebbe anche aumentare l’ampiezza del suono diretto. Il ritardo e il guadagno dipendono dal tipo di sorgente sonora utilizzata allo scopo.

Quando la riflessione proviene da una direzione laterale, essa può influenzare profondamente il carattere spaziale del suono. Per piccoli valori di ritardo (inferiori a 5msec), l’eco può causare uno spostamento apparente della posizione della sorgente, mentre valori di ritardo più grandi possono aumentare la dimensione apparente della sorgente, a seconda del suo contenuto in frequenza, o possono dare la sensazione all’ascoltatore di essere completamente circondato dal suono.

Gli studi di Barron e Marshall relativi a questo fenomeno, in cui sono stati usati segnali musicali complessi, determinarono che il grado di impressione spaziale del suono era direttamente correlato

al seno dell'angolo di incidenza della riflessione, il cui valore massimo di 90° corrisponde a una direzione di provenienza completamente laterale [31].

Da questi studi fu proposta una semplice misurazione acustica per la predizione dell'impressione spaziale del suono, detta frazione laterale (LF), espressa come il rapporto tra l'energia iniziale ricevuta da un microfono a dipolo e l'energia iniziale complessiva.

Nel 1995 Hidaka propose una misurazione acustica binaurale che, in termini di accuratezza nel predire l'impressione spaziale, superò di gran lunga la LF; questa misurazione è nota come coefficiente di cross-correlazione interaurale (IACC, *interaural cross-correlation coefficient*) [32], definita come segue:

$$IACF(\tau) = \frac{\int_{t_1}^{t_2} p_L(t) p_R(t + \tau) d\tau}{\left(\int_{t_1}^{t_2} {p_L}^2(t) dt \int_{t_1}^{t_2} {p_R}^2(t) dt \right)^{1/2}}$$

$$IACC = |IACF(\tau)|_{max} \; per - 1 < \tau < +1 \; ms \qquad (2)$$

dove p_L e p_R sono le pressioni misurate all'ingresso dei canali auditivi sinistro e destro, rispettivamente, e i limiti di integrazione t_1 e t_2 sono impostati a 0 e 80 msec, rispettivamente, quando viene calcolato il valore iniziale della $IACC$, detto $IACC_E$. $IACF(\tau)$ è la funzione di cross-correlazione normalizzata delle pressioni alle orecchie sinistra e destra in un intervallo di tempo di lunghezza τ, e $IACC$ è il massimo di questa funzione in un range di valori di ± 1 msec, il quale tiene conto del massimo tempo di ritardo interaurale.

L'intervallo di tempo per il quale $IACF$ assume il suo valore massimo fa una stima della direzione laterale della sorgente sonora [33]. Un aumento della $IACF$ e, di conseguenza, una riduzione della $IACC$ corrispondono ad una maggiore impressione spaziale del suono.

2.4 Tempo di riverberazione [6]

Il concetto di *tempo di riverberazione* deriva dalle ricerche di Sabine [34] nel campo dell'acustica delle stanze.

Uno dei primi esperimenti condotti da Sabine consisteva nel misurare il tempo di decadimento del riverbero, e analizzare come variava introducendo materiale fonoassorbente all'interno della stanza.

Sabine stabilì che il tempo di decadimento del riverbero era direttamente proporzionale al volume della stanza e inversamente proporzionale alla quantità di energia assorbita dalle pareti, secondo la seguente relazione :

$$T_r \propto \frac{V}{A} \qquad (3)$$

dove T_r è il tempo di decadimento del riverbero che deve trascorrere affinché la pressione del suono cali di 60 dB, V è il volume della stanza e A misura l'assorbimento complessivo di energia da parte dei materiali della stanza.

Dal momento che le proprietà di assorbimento dei materiali variano in funzione della frequenza, anche il tempo di riverberazione varierà in funzione della frequenza.

In genere, materiali porosi (come i tappeti o il polistirolo) assorbono una maggiore quantità di energia alle alte frequenze, di conseguenza il tempo di riverberazione decresce al crescere della frequenza.

Si può misurare il tempo di riverberazione eccitando la stanza, al momento in condizioni stazionarie, con un segnale di rumore, disattivando la sorgente sonora, e tracciando la pressione risultante del rumore, elevata al quadrato, in funzione del tempo.

Il tempo necessario affinché l'ampiezza di questa curva (detta *energy decay curve*, *EDC*) scenda al di sotto dei 60 dB è definita, appunto, tempo di riverberazione.

Schroeder dimostrò che il vero valore della EDC può essere calcolato integrando la risposta impulsiva della stanza [35]:

$$EDC(t) = \int_t^{\infty} h^2(\tau) d\tau \tag{4}$$

dove *h(t)* è la risposta impulsiva della stanza. Questo integrale calcola la quantità di energia rimanente nella risposta impulsiva dopo una quantità di tempo pari a *t*.

2.5 Descrizione modale del riverbero [6]

Considerando una stanza ideale (perfettamente rettangolare e con pareti rigide) è possibile descrivere matematicamente, in una forma chiusa, il comportamento del riverbero all'interno di tale stanza. Per fare questo si considera dalll'equazione d'onda acustica e la si risolve applicando le condizioni al contorno imposte dalle pareti della stanza. Le soluzioni di questa equazione, basate sulle frequenze di risonanza delle pareti della stanza, prendono il nome di *modi normali*, o *modi di risonanza*.

Per una stanza rettangolare le frequenze di risonanza sono date dalla relazione (5) [36]:

$$f_n = \frac{c}{2}\sqrt{\left(\frac{n_x}{L_x}\right)^2 + \left(\frac{n_y}{L_y}\right)^2 + \left(\frac{n_z}{L_z}\right)^2} \tag{35}$$

dove:

- f_n è l'n-esima frequenza di risonanza, misurata in Hz
- n_x, n_y e n_z sono numeri interi che possono assumere valori da 0 a ∞
- L_x, L_y e L_z sono le dimensioni della stanza, misurate in metri (larghezza, lunghezza e altezza)
- *c* è la velocità del suono in m/sec

Il numero N_f di modi di risonanza al di sotto della frequenza *f* è dato dalla relazione (6) :

$$N_f \approx \frac{4\pi V}{3c^3} f^3 \tag{6}$$

dove *V* è il volume della stanza ($V = L_x L_y L_z$). Derivando rispetto a *f*, otterremo la densità modale espressa come funzione della frequenza (7):

$$\frac{dN_f}{df} \approx \frac{4\pi V}{c^3} f^2 \tag{7}$$

Pertanto, il numero di modi di risonanza per unità di larghezza di banda (numero di modi per Hz) cresce al crescere del quadrato della frequenza.

Quando una sorgente irradia un suono all'interno di un ambiente chiuso, vengono eccitati uno o più modi normali della stanza. Non appena la sorgente smette di irradiare, i modi continuano a risuonare l'energia acustica immagazzinata; ciascuna di queste risonanze decade a un certo tasso definito dalla costante di smorzamento del relativo modo, la quale dipende dalla capacità di assorbimento delle pareti della stanza.

Il procedimento appena descritto è equivalente a quanto accade nei circuiti elettrici contenenti molte risonanze parallele.

2.6 Modello statistico del riverbero [6]

È possibile utilizzare i modi normali anche per descrivere il comportamento riverberante di una stanza reale, sebbene possa risultare impossibile ottenere una soluzione in forma chiusa come quella vista nel caso ideale.

Si può dimostrare che l'equazione che (7), in generale, vale anche per le stanze con pareti dalle forme irregolari [37].

Alle alte frequenze, la risposta in frequenza complessiva della stanza è determinata da un gran numero di modi normali le cui curve di risonanza tendono a sovrapporsi. Ad ogni frequenza, la risposta in frequenza complessa della stanza è data dalla somma delle risposte modali sovrapposte, che possono essere considerate indipendenti tra loro e distribuite casualmente nello spazio.

Se a tale somma contribuisce un numero di termini sufficientemente elevato, i coefficienti reali e immaginari della risposta in frequenza combinata possono essere modellati come variabili casuali gaussiane indipendenti. Di conseguenza, la risultante risposta all'ampiezza della pressione segue lo stesso andamento della distribuzione di probabilità di Rayleigh.

Questo modello statistico per il riverbero è utilizzabile per frequenze superiori a quelle definite dalla relazione:

$$f_g \approx 2000\sqrt{\frac{T_r}{V}}\ \text{Hz} \tag{8}$$

dove T_r è il tempo di riverbero, misurato in secondi e *V* è il volume della stanza, misurato in metri cubi.

La distanza media dei massimi di frequenza, misurata in Hz, è data dalla formula (9):

$$\Delta f_{max} \approx \frac{4}{T_r} Hz \tag{9}$$

Un'altra statistica interessante è la densità temporale dei modi, il cui valore cresce al crescere del tempo. Questa statistica può essere determinata a partire dal modello delle sorgenti immagine per il riverbero, che per una stanza rettangolare da' luogo a un modello regolare di sorgenti immagine, descritto in Fig.2 :

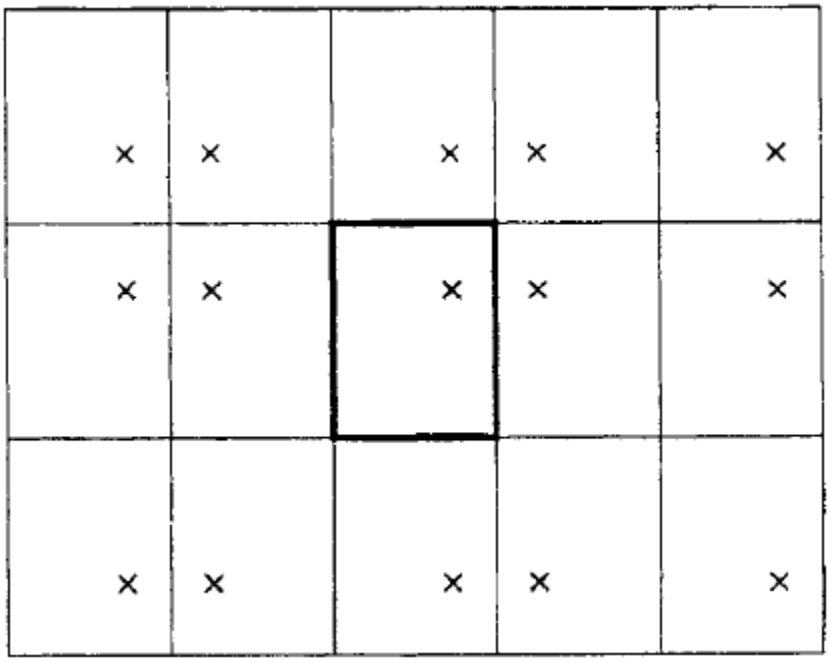


Figura 2 – Modello di sorgenti immagine per una stanza rettangolare

Il numero di echi N_t che occorrono nell'arco di tempo *t* è uguale al numero di sorgenti immagine racchiuse in una sfera di diametro *ct* centrata sull'ascoltatore [37]. Avendo una sorgente immagine per unità di volume, il numero di sorgenti immagine all'interno della sfera può essere stimato dividendo il volume della sfera per il volume della stanza, come in (10) :

$$N_t = \frac{4\pi(ct)^3}{3V} \tag{10}$$

Derivando rispetto a *t*, si ottiene la densità temporale di echi (11):

$$\frac{dN_t}{dt} = \frac{4\pi c^3}{V} t^2 \tag{11}$$

2.7 Metodologie di implementazione dei riverberi

Il riverbero consiste nella modifica di un segnale a partire dalla risposta acustica dell'ambiente in cui è contenuta la sorgente del segnale. È possibile modellare tale risposta (che noi chiameremo RIR, Room Impulse Response) come una risposta impulsiva a durata finita (FIR [2, pagine 2-3]) che si può misurare tra la posizione della sorgente del segnale e quella dell'ascoltatore.

È inoltre possibile modificare le caratteristiche acustiche legate alla stanza che contiene una sorgente di segnale *s(k)* applicando la convoluzione di questo segnale con la risposta impulsiva della stanza RIR(k) (11):

$$s_{rev}(k) = \sum_{l=0}^{N-1} RIR(l)s(k-l) \quad (12)$$

dove il termine a sinistra indica il segnale riverberato risultante mentre N è la lunghezza della risposta impulsiva della camera.

L'effetto di riverberazione associato alla RIR può essere diviso in due parti:

1) Prime riflessioni (*Early Reflections*)
2) Riverbero tardivo (*Late Reverberation*)

Le prime riflessioni sono riconducibili fondamentalmente alla geometria della stanza: esse corrispondono ai primi 80/100 ms di suono riverberato e possono essere opportunamente modellate per mezzo di filtri FIR. Il riverbero tardivo, dopo una certa quantità di tempo, sovrasta le prime riflessioni e si estende per tutta la durata della RIR. In Fig.3 sono rappresentate le ampiezze delle prime riflessioni e della coda del riverbero nei vari istanti di tempo:

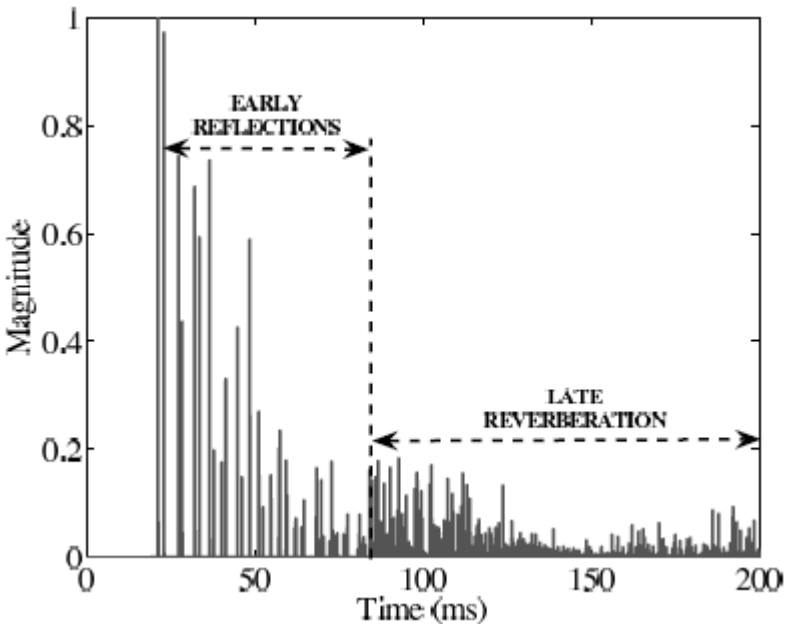


Figura 3 – Ampiezze delle prime riflessioni e della coda del riverbero

Quest'ultima parte di riverbero è quasi diffusa, il che significa che il suono può essere considerato distribuito casualmente. Inoltre, l'ampiezza del suono decade esponenzialmente nel tempo.

Il riverbero tardivo può, dunque, essere modellato per mezzo di filtri IIR (cioè a risposta impulsiva di durata infinita).

Un parametro importante da tenere in considerazione è il tempo impiegato da un suono all'interno di una stanza di ridurre il suo livello di pressione sonora di 60 dB dopo essere stato improvvisamente interrotto. Questo parametro è definito a partire dalla formula di Sabine (13):

$$T_{60} = 0.161(Vol/S_e) \tag{13}$$

Dove *Vol* è il volume della stanza, mentre S_e è l'area effettiva dela stanza, misurata come:

$$S_e = \sum_{i=1}^{L}(1 - r_i)S_i \tag{14}$$

S_i e r_i indicano,rispettivamente, l'area e il coefficiente di riflessione delle *i* pareti della stanza.

2.8 Parametri del riverbero

- *Pre-delay Time*: quantità di tempo che intercorre tra il suono originale e la sua prima riflessione (misurato in ms)
- *Decay (Decadimento)*: la coda del riverbero, ovvero il tempo in cui le riflessioni successive del suono originale perdurano fino ad estinguersi completamente. Ogni singola riflessione del suono originale presenterà, dunque, un valore in ampiezza di una certa quantità inferiore rispetto alla riflessione precedente. Questo parametro, pertanto, regola di quanto si riduce l'ampiezza delle riflessioni man mano che passa il tempo: più questa ampiezza si riduce più corta sarà la coda del riverbero e meno tempo impiegheranno le riflessioni ad estinguersi.
- *Wet/Dry*: parametro che compare nei riverberi digitali e che regola, in pratica, la percentuale di intervento del processore di riverbero sul segnale audio. Più alto è il valore "Wet" più riverberato risulterà il suono.
- *Equalizer*: presente in alcuni riverberi digitali. Di solito consiste in filtri passa-alto o filtri passa-basso e serve per equalizzare le riflessioni del suono originale.

CAPITOLO 3

Generalità sugli algoritmi per la riverberazione

Sebbene l'intento dei riverberatori (e dunque, degli algoritmi che simulano il loro comportamento) sia quello di riprodurre artificialmente, entro certi margini di tolleranza, le riflessioni di un segnale acustico che si verificano all'interno di un ambiente, è bene precisare che la ragione principale per cui vengono apprezzati e utilizzati dai musicisti è di rafforzare il suono, ovvero dargli la giusta spazialità all'interno di un determinato contesto musicale; per ottenere questo risultato, non sempre la timbrica del riverbero corrisponderà a quella di un ambiente reale.

Il riverbero è un fenomeno acustico basato sulla riflessione delle onde sonore su diverse superfici costituenti un ambiente chiuso; esso può presentare una diversa natura a seconda dell'ambiente in cui il suono si propaga, pertanto lo scopo dei riverberatori artificiali è quello di riprodurre questo comportamento. A seconda delle proprietà fisiche dei supporti utilizzati, nonché dalla loro quantità all'interno del dispositivo di riverberazione, il riverbero artificiale ricavato presenterà caratteristiche diverse.

Un riverberatore fisico è un dispositivo elettromeccanico che, mediante l'oscillazione meccanica di un supporto (che può essere una molla, nel caso di riverberi di tipo "spring", o una piastra metallica, nel caso di riverberi di tipo "plate") generata per effetto di un impulso elettrico da parte di un elettrodo, permette di riprodurre artificialmente un particolare riverbero naturale, simulando il percorso del segnale riverberato dalla sorgente all'ascoltatore.

I più moderni riverberatori artificiali sono, invece, realizzati su base di circuiti digitali costituiti fondamentalmente da filtri ritardanti (i più utilizzati sono gli allpass e i comb).

Lo scopo dei filtri utilizzati nei riverberi digitali è, appunto, quello di ritardare il segnale in ingresso e smorzarne l'ampiezza in una certa misura. L'entità del ritardo è determinata dal blocco ritardante, rappresentato negli schemi da un blocco funzionale di funzione di trasferimento z^{-n}, dove n indica il tempo in cui il segnale viene ritardato (solitamente misurato in millisecondi): il segnale che lo attraversa viene ritardato di n millisecondi e poi, successivamente, viene sommato al segnale diretto in ingresso al filtro.

Per quanto riguarda l'attenuazione dell'ampiezza del segnale ritardato, essa la si ottiene mediante un moltiplicatore di guadagno posto nella catena di retroazione del filtro: i valori scelti per questo guadagno sono inferiori a 1, pertanto l'ampiezza del segnale sarà ridotta in proporzione al valore di questo guadagno.

Riprodurre in formato digitale il comportamento di un riverberatore fisico significa prendere un segnale in ingresso in un elaboratore e sottoporlo all'azione di algoritmi matematici, basati principalmente sul ritardo e smorzamento in ampiezza dei segnali acustici in ingresso (come, del resto, avviene nell'algoritmo scelto per questa tesi).

Questi algoritmi sono un'astrazione di quello che succede nei circuiti digitali appena descritti e, pertanto, simulano il comportamento di ogni componente.

3.1 Risposta all'impulso finito (FIR)

Un filtro FIR (Finite Impulse Response) è una tipologia di filtro digitale caratterizzato da una risposta impulsiva di durata finita [15].

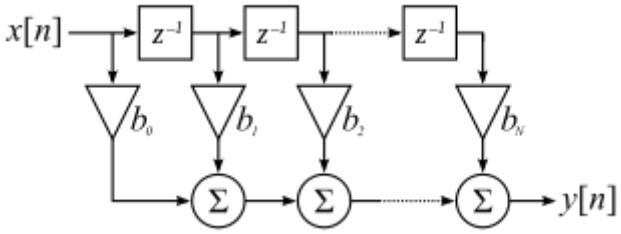


Figura 4 – Diagramma di flusso di un filtro FIR

In Fig.4 si riporta il diagramma di flusso di un filtro FIR, costituito da un certo numero di blocchi ritardanti in cascata, ciascuno con funzione di trasferimento z^{-1}, e da un moltiplicatore di guadagno b_i per il quale viene moltiplicato il segnale in uscita dal blocco ritardante *i*.

Considerando un sistema LTI (ovvero,un sistema dinamico lineare e tempo-invariante) a tempo continuo, se si applica un segnale x(t) all'ingresso di tale sistema si genera in uscita un segnale y(t) dato dalla convoluzione:

$$y(t) = x(t) * h(t) \tag{15}$$

dove *h(t)* è la risposta del sistema quando *x(t)* è una funzione a delta di Dirac.

Un sistema dinamico lineare stazionario e discreto trasforma la successione di ingresso {x} in un'altra successione {y},data dalla convoluzione discreta con la risposta h alla delta di Kronecker (16):

$$y[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k] = \sum_{k=-\infty}^{\infty} x[n-k] \cdot h[k] \tag{16}$$

Per un filtro a tempo discreto, l'uscita è una somma pesata dei valori assunti dall'ingresso al tempo corrente e ai tempi precedenti. Tale operazione è descritta dalla seguente equazione (17):

$$y[n] = h_0 x[n] + h_1 x[n-1] + \cdots + h_n x[n-N] = \sum_{i=0}^{N} h_i x[n-i] \tag{17}$$

Dove h_i sono i coefficienti del filtro, che determinano la risposta impulsiva, e *N* è l'ordine del filtro.

3.2 Modellazione prime riflessioni [6]

In questo paragrafo si discuterà delle varie soluzioni proposte negli anni per riprodurre artificialmente le prime riflessioni di un riverbero.

Gli algoritmi per la simulazione delle prime riflessioni di un riverbero prevedono l'implementazione dei filtri digitali.

In generale, per poter rappresentare una risposta riverberante, una delle tecniche più utilizzate è quella della convoluzione. Tuttavia, risulta sconveniente implementare la convoluzione utilizzando la forma diretta dei filtri FIR (Fig.5, [27]) qualora la dimensione del filtro sia grande.

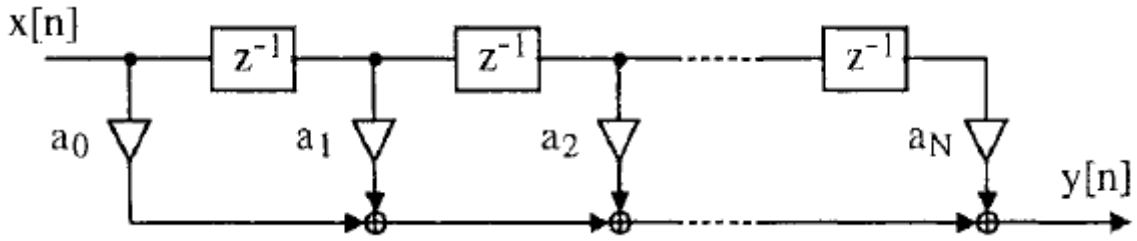


Figura 5 – Forma canonica diretta di un filtro FIR con delays a singolo campione

Come alternativa, si potrebbe utilizzare l'algoritmo di processamento a blocchi, basato sulla Fast Fourier Transform (FFT) [38] per implementare la convoluzione, sebbene anche questa tecnica potrebbe risultare piuttosto problematica per processamenti in tempo reale, a causa del ritardo di propagazione dei segnali di ingresso e uscita che si verifica nell'algoritmo.

Gardner propose una soluzione che eliminasse tale ritardo, che consiste nel segmentare la risposta impulsiva in blocchi di dimensione esponenzialmente crescente [18]. La convoluzione col primo blocco si calcola utilizzando un filtro a forma diretta (come quello rappresentato in Fig.5), mentre la convoluzione con i restanti blocchi si calcola applicando tecniche nel dominio della frequenza.

Per filtri di grandi dimensioni, questo algoritmo risulta molto più efficiente del solo filtro a forma diretta.

Quando la risposta iniziale è ottenuta senza avere imposto alcune condizioni speciali sull'assorbimento o sulla diffusione, essa presenterà un certo numero di impulsi ritardati la cui ampiezza tende ad attenuarsi col tempo. Di conseguenza, sarà possibile implementare il filtro appena discusso utilizzando una struttura di moduli caratterizzati da lunghi ritardi.

Un esempio è dato dalla struttura in Fig.6 [19], proposta da Schroeder per generare uno specifico insieme di prime riflessioni, oltre che un tardo riverbero diffuso.

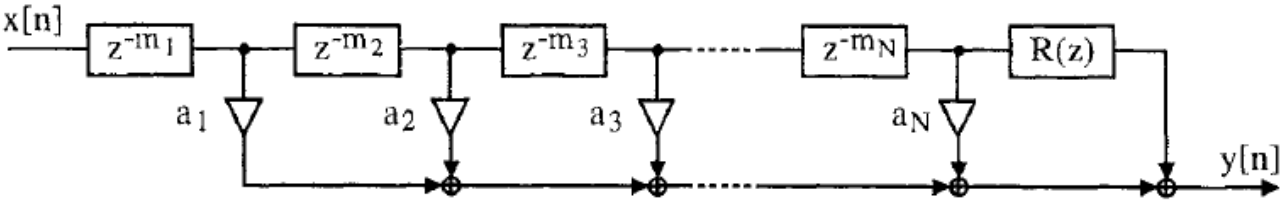


Figura 6 – Combinazione delle prime riflessioni (rappresentate dai moduli di funzione di trasferimento z^{-m_n} e del tardo riverbero (R(z) è la funzione di trasferimento del riverbero) [19]

I filtri FIR che riproducono le prime riflessioni comprendono un insieme di ritardi m_i e un insieme di guadagni a_i, mentre il tardo riverbero è rappresentato dal filtro *R(z)*. Il segnale in ingresso a questo filtro, *x(n)*, è ritardato di una quantità pari a m_1, pertanto l'uscita finale, *y(n)*, seguirà le risposte dei rimanenti filtri FIR.

Moorer propose una soluzione diversa [20], nella quale il tardo riverbero è pilotato dal segnale in uscita dal filtro FIR relativo alla prima riflessione, come è mostrato in Fig.7 :

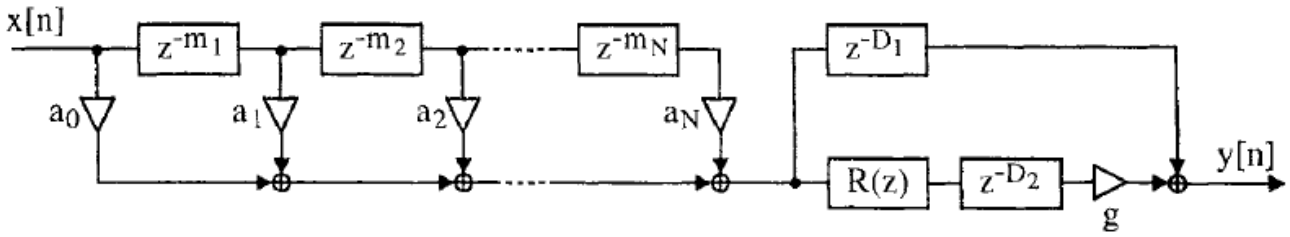


Figura 7 – Filtro di Moorer, costituito dalla cascata di un filtro FIR con un riverberatore R(z)

L'intenzione di Moorer era quella di incrementare la densità degli echi del tardo riverbero. I valori di D_1 e D_2 possono essere impostati in modo che il primo impulso in uscita dal riverberatore corrisponda all'ultimo impulso in uscita dalla sezione di FIR. Il guadagno g, invece, serve a bilanciare la concentrazione di echi del grappolo di riverberazione con quella delle prime riflessioni.

La particolarità di questa struttura è che essa, mediante la manipolazione dei parametri appena descritti, permette di controllare la forma del decadimento del riverbero. Per esempio, se la risposta dei FIR ha uno sviluppo a forma rettangolare molto esteso, mentre il tardo riverbero decade esponenzialmente in maniera relativamente veloce, allora la risposta complessiva in uscita dalla cascata sarà data da una combinazione degli sviluppi dei FIR e di *R(z)*; in altre parole, presenterà inizialmente un andamento costante piatto, seguito da un rapido decadimento.

Modellare le prime riflessioni per mezzo di una rete costituita da filtri FIR porterà, come risultato, ad una risposta in uscita la cui qualità del suono sarà eccessivamente discreta, in particolare se all'ingresso vengono applicati segnali impulsivi molto intensi. Per migliorare la qualità del suono occorrerà applicare la risposta inizial in ingresso e a dei filtri passa-basso (la soluzione più semplice richiede un singolo filtro passa-basso in serie al filtro FIR).

Nel caso della struttura proposta da Schroeder, si potrebbero sostituire i guadagni a_i con dei filtri dipendenti dalla frequenza $A_i(z)$, che hanno la facoltà di modellare le perdite dovute all'assorbimento di energia da parte delle pareti della stanza e alla propagazione del suono nell'aria (queste perdite sono dipendenti dalla frequenza). La struttura di ognuno di questi filtri viene impostata in base alla storia passata delle riflessioni di ogni eco nella stanza, come riportato nell'equazione (1).

3.3 Riverberatori costruiti su filtri comb e filtri allpass [6]

I principali algoritmi che riproducono la coda del riverbero (tra cui il riverberatore di Schroeder [21]) si basano sull'utilizzo ed il funzionamento dei filtri comb e allpass, mentre quelli più recenti sono realizzati secondo metodi generali basati sulle Feedback Delay Networks (FDN), di cui si discuterà tra qualche paragrafo.

Riverberatore di Schroeder

Uno dei principali algoritmi di riverberazione, nonché il primo nella storia ad essere stato concepito per il processamento dei segnali tempo-discreti, si basa sull'implementazione di filtri comb e filtri allpass.

Il filtro comb (*filtro a pettine*), rappresentato in Fig.8 , consiste in un delay il cui segnale in uscita viene fatto ricircolare in ingresso:

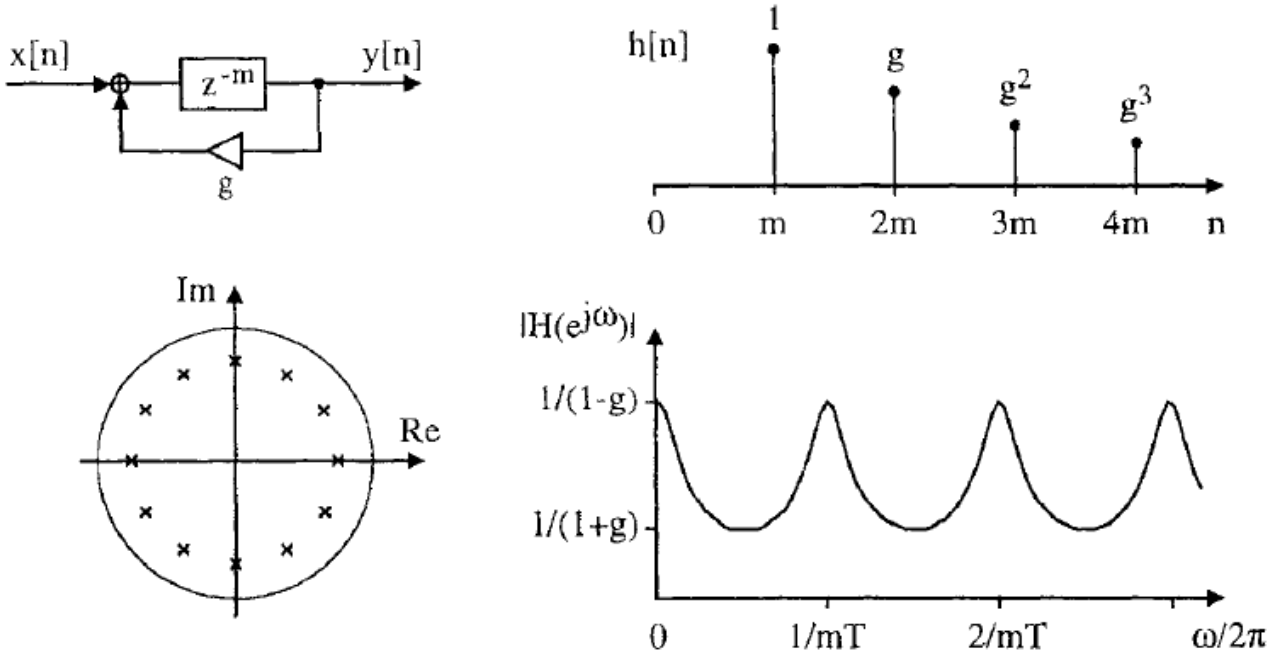


Figura 8 – Filtro comb. In alto a sinistra è riportato il diagramma di flusso. In alto a destra la risposta impulsiva nel tempo. In basso a sinistra il diagramma polare. In basso a destra la risposta in frequenza

La z-trasformata del blocco funzionale del filtro comb è data dalla formula:

$$H(z) = \frac{z^{-m}}{1-gz^{-m}} \tag{18}$$

dove *m* indica la lunghezza del ritardo in campioni (samples), mentre *g* indica il guadagno di feedback. Come è riportato in Fig. , la risposta nel tempo di questo filtro descrive una sequenza di impulsi di ampiezza esponenzialmente decrescente, separati l'uno dall'altro di una distanza pari a *m* campioni.

Sempre in figura, è riportato il diagramma polare del filtro: i poli del sistema sono posizionati in corrispondenza delle *m* radici di *g*, e quindi sono distanziati armonicamente su un cerchio giacente nel piano z. Questo comporta che la risposta in frequenza del filtro presenti una forma a pettine (da cui il nome *comb*, che in inglese significa, appunto, *pettine*), in quanto costituita da *m* picchi periodici che si manifestano in corrispondenza delle frequenze dei poli.

Il filtro comb può essere modificato, mischiando il segnale in ingresso con quello in uscita, in modo da ottenere una risposta in frequenza ad andamento piatto, come riportato in Fig.9 . Il filtro risultante da questa modifica prende il nome di filtro allpass (*filtro passa-tutto*), in quanto la sua risposta in frequenza presenta lo stesso valore di ampiezza per tutte le frequenze.

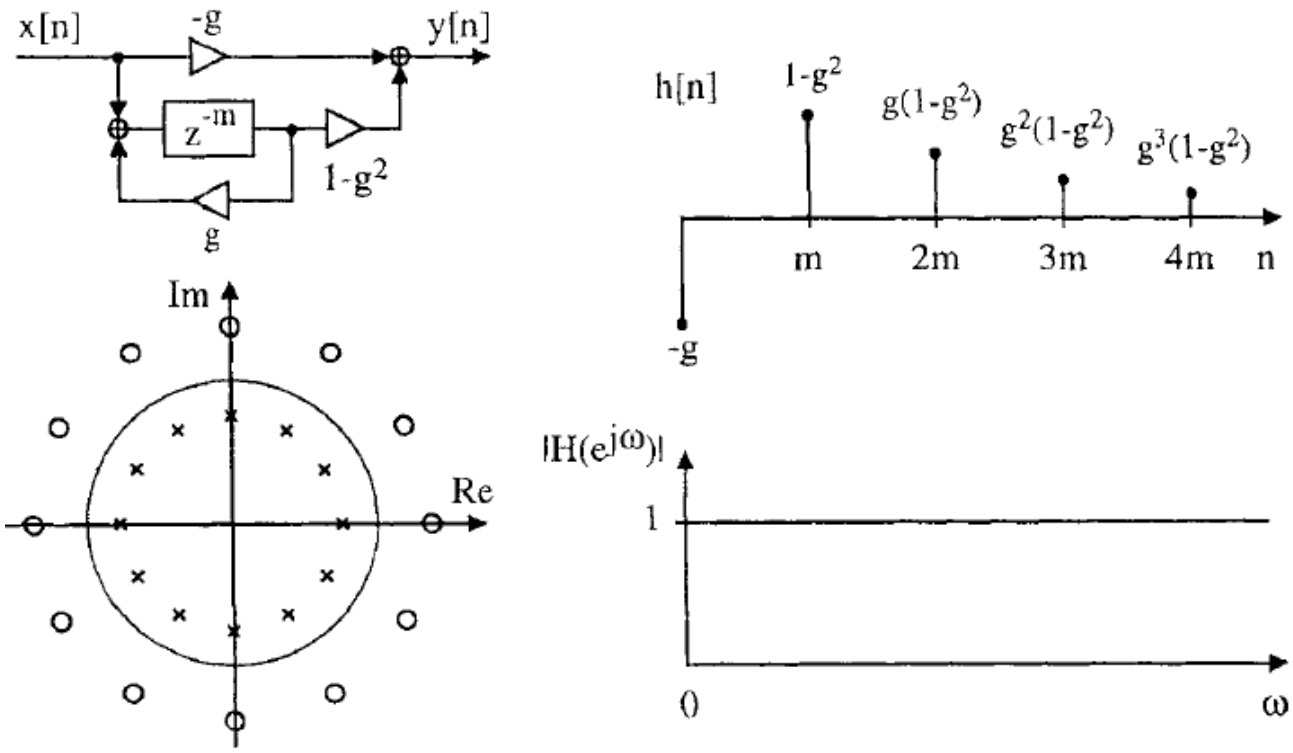


Figura 9 – Filtro allpass. In alto a sinistra il diagramma di flusso. In alto a destra la risposta impulsiva nel tempo. In basso a sinistra il diagramma polare. In basso a destra la risposta in frequenza

La z-trasformata per il filtro allpass è data dalla formula:

$$H(z) = \frac{z^{-m} - g}{1 - gz^{-m}} \tag{19}$$

I poli della funzione di trasferimento del filtro allpass sono gli stessi di quella del filtro comb; nel filtro allpass, però, sono presenti anche gli zeri nelle posizioni reciproche coniugate. Riscrivendo *H(z)* nella forma seguente :

$$H\left(e^{j\omega}\right) = e^{j\omega m}\frac{1 - ge^{+j\omega m}}{1 - ge^{-j\omega m}} \tag{20}$$

è evidente che la risposta in ampiezza del filtro ha ampiezza unitaria, perché il primo termine, $e^{-j\omega m}$, ha ampiezza unitaria, mentre il secondo è un quoziente di complessi coniugati e, di conseguenza, ha anch'esso ampiezza unitaria. Pertanto vale la seguente relazione:

$$\left|H(e^{j\omega})\right| = 1 \tag{21}$$

La risposta in fase del filtro allpass è una funzione non lineare della frequenza, che porta a una diffusione del segnale nel dominio del tempo.

Supponiamo di voler creare un riverbero utilizzando un singolo filtro comb o un singolo filtro allpass. Nel caso di un filtro comb, il tempo di riverbero è espresso dalla relazione:

$$\frac{20\log_{10}(g_i)}{m_i T} = \frac{-60}{T_r} \tag{22}$$

dove g_iè il guadagno del filtro, m_i è la lunghezza del ritardo misurata in numero di campioni, e T è il periodo di campionamento. Per ottenere il tempo di riverbero desiderato, possiamo impostare i valori della lunghezza di ritardo e del guadagno di feedback in modo da bilanciare densità modale e e densità di eco.

Il problema di quando si costruisce un riverberatore utilizzando un singolo filtro comb è che, per brevi tempi di ritardo, che producono echi in rapida successione, la risposta in frequenza è caratterizzata da picchi di frequenza ampiamente equispaziati tra loro. Questi picchi corrispondono alle frequenze che vengono riverberate, mentre tutte le altre frequenze non corrispondenti ai valori di picco decadranno rapidamente.

In questa situazione, quando i picchi nella risposta in frequenza sono ampiamente distanziati, il suono uscente dal filtro comb avrà un timbro molto sgradevole.

Per migliorare la qualità del suono si potrebbe aumentare la densità dei picchi nella risposta in frequenza aumentando la lunghezza del ritardo, ma così facendo diminuisce la densità dell'eco nel dominio del tempo e, di conseguenza, il suono riverberato sarà percepito come un insieme discreto e confuso di echi, piuttosto che un decadimento diffuso e regolare.

Per quanto riguarda, invece, il filtro allpass, esso presenta una risposta in ampiezza piatta, perciò si potrebbe pensare di utilizzarlo per risolvere il problema della colorazione timbrica che si presenta nel filtro comb. Tuttavia, la risposta di un filtro allpass suona abbastanza simile a quella del filtro comb (dal momento che il filtro allpass non è altro che un filtro comb modificato), e quindi tende anch'essa a creare una colorazione timbrica molto forte. Questo accade perché l'orecchio umano esegue un'analisi in frequenza a breve termine, mentre le proprietà matematiche del filtro allpass sono definite per un' integrazione a tempo infinito.

Combinando in serie due filtri elementari, si può incrementare notevolmente la densità d'eco, in quanto per ogni eco generata dal primo filtro si produrrà un insieme di echi nel secondo.

È sconsigliabile collegare in serie due filtri comb, perché le sole frequenze che verranno lasciate passare sono quelle corrispondenti ai picchi di entrambi i filtri comb. Al contrario, si possono tranquillamente collegare in serie quanti più filtri allpass si vogliano, dato che la risposta combinata in uscita sarà comunque una risposta tipica dei filtri allpass. Di conseguenza, è utile collegare in serie i filtri allpass al fine di aumentare la densità d'eco senza, però, influenzare la risposta in ampiezza dell'intero sistema.

Un'altra utile struttura è quella che prevede il collegamento in parallelo di filtri comb, caratterizzati da tempi di ritardo molto elevati, poiché la risposta in frequenza risultante conterrà i picchi di frequenza di tutti i singoli filtri comb contenuti nel parallelo. Inoltre, la densità d'eco complessiva sarà data dalla somma delle densità d'eco dei singoli filtri comb.

La combinazione in parallelo di un certo numero di filtri a pettine consente, in teoria, di ottenere una densità arbitraria di picchi di frequenza e di echi.

Il riverbero proposto da Schroeder nel 1962 [21] consiste, dunque, in una struttura composta da filtri comb in parallelo combinati a filtri allpass in serie, come mostrato in Fig.10 :

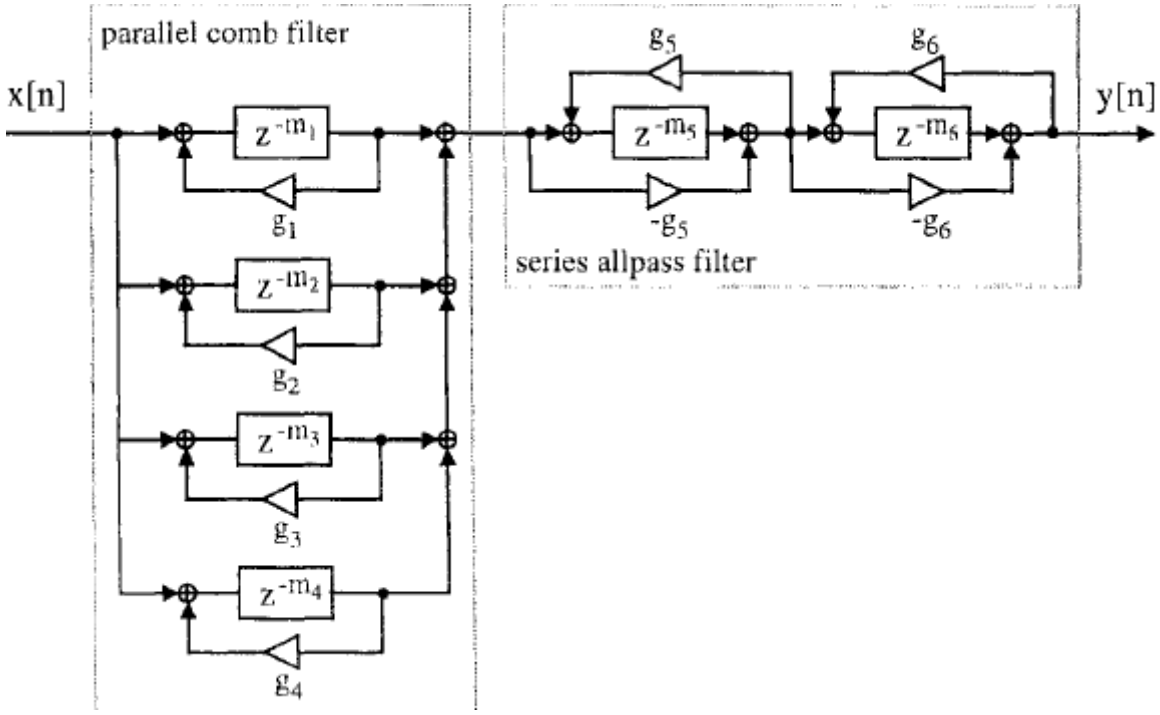


Figura 10 – Diagramma di flusso del riverberatore di Schroeder

I ritardi dei filtri comb sono scelti in modo tale che il rapporto tra il ritardo più grande e quello più piccolo sia pari a 1.5. Come mostrato nell'equazione seguente, il valore dei guadagni g_i è scelto in modo da dare un il tempo di ritardo T_r desiderato

$$g_i = 10^{-3m_iT/T_r} \tag{23}$$

I tempi di ritardo dei due filtri allpass, invece, sono molto più brevi rispetto a quelli dei filtri comb, con valori di guadagno pari a 0.7 per entrambi i filtri.

Con questi valori, i filtri comb producono il lungo decadimento del riverbero, mentre i filtri allpass moltiplicano tra di loro gli echi prodotti un uscita dai filtri comb.

3.4 Parallelo di filtri comb [6]

La z-trasformata di una struttura di filtri comb collegati in parallelo è data dalla seguente formula [23]:

$$H(z) = \sum_{i=1}^{N} \frac{z^{-m_i}}{1 - g_i z^{-m_i}} \tag{24}$$

dove N è il numero di filtri comb costituenti la struttura. I poli della funzione di trasferimento corrispondono alle soluzioni della seguente equazione:

$$\prod_{i=1}^{N} (g_i - z^{m_i}) = 0 \tag{25}$$

Per ogni filtro, i moduli dei poli hanno lo stesso valore, dato dalla (26) :

$$\gamma_i = \sqrt[m_i]{g_i} = 10^{-3T/T_r} \tag{26}$$

Supponendo che i valori g_i dei guadagni di tutti i filtri siano impostati in base allo stesso valore T_r del tempo di riverbero, il valore di γ_i sarà lo stesso per tutti i filtri comb. Pertanto, tutti i modi di risonanza del parallelo di filtri comb decadranno allo stesso modo.

Nel caso in cui i moduli dei poli non fossero uguali, i poli aventi il modulo più alto risuoneranno più a lungo e questi poli sono quelli che determinerebbero la caratteristica tonale del tardo riverbero ([20], [26]). Di conseguenza, per evitare la colorazione tonale nel tardo riverbero, è importante che la relazione (26), riguardante l'uniformità del modulo polare, sia rispettata [23].

Quando i ritardi nei filtri comb sono incommensurabilmente lunghi (il che significa che i valori di ritardo non condividono fattori in comune), le frequenze dei poli saranno tutte distinte tra loro.

Inoltre, nella risposta temporale, considerando due distinti filtri comb, *i* e *k*, non si avrà sovrapposizione degli echi dei due filtri fino ai campioni m_i e m_k.

3.5 Densità modale e densità d'eco [6]

Affinché il riverbero creato artificialmente sia realistico occorre tener conto di due criteri importanti: la densità modale e la densità d'eco.

Densità modale

La densità modale dei filtri comb in parallelo, espressa come numero di modi di risonanza per Hz, è data dalla formula [23]:

$$D_m = \sum_{i=0}^{N-1} \tau_i = N \cdot \tau \tag{27}$$

dove τ_i è la lunghezza dell'i-esimo ritardo misurata in secondi, e τ è la lunghezza media di ritardo.

Da questa formula, risulta evidente che la densità modale dei filtri comb in parallelo sia costante per tutte le frequenze, diversamente da quanto accade nelle stanze reali, in cui la densità modale aumenta al quadrato della frequenza (equazione (7)). Tuttavia, nelle stanze reali, una volta che la densità modale supera un certo valore di soglia, la risposta in frequenza è caratterizzata da massimi di frequenza la cui spaziatura media è costante (equazioni (8) e (9)).

Pertanto, è possibile approssimare la risposta in frequenza di una stanza equiparando la densità modale dei filtri comb in parallelo con quella dei massimi di frequenza nella risposta della stanza [Schroeder, 1962]. La lunghezza totale dei ritardi dei filtri comb, espressa in secondi, è uguale alla densità modale del parallelo di filtri comb, espressa come numero di modi per Hz.

Mettendo ad equazione questa densità modale e quella dei massimi di frequenza delle stanze reali, si ottiene la relazione tra la lunghezza totale dei ritardi e il tempo di riverbero massimo che si vuole simulare [22], espressa come segue:

$$\sum_i \tau_i = D_m > D_f \approx \frac{T_{max}}{4} \tag{28}$$

dove D_f è la densità dei massimi di frequenza, ottenuta sulla base del modello statistico per il tardo riverbero (pari al reciproco di Δ_{max} nell'equazione (9)) e T_{max} è il tempo di riverbero massimo desiderato.

Questa equazione specifica la quantità minima di ritardo totale richiesta.

In risposta a segnali a banda stretta, una bassa densità modale può portare a battiti udibili, dovuti all'eccitazione di due modi di risonanza molto vicini tra loro in frequenza; in questo caso, si produce un battimento ad una frequenza pari alla differenza delle frequenze dei due modi.

Per arginare il fenomeno di battimenti, è possibile scegliere la spaziatura media dei due modi in modo che il periodo medio del battito sia almeno pari al tempo di riverbero [24]. Questo porta alla relazione:

$$\sum_i \tau_i \geq T_{max} \tag{29}$$

Nel caso di una struttura di filtri comb collegati in parallelo, se si considera il vincolo espresso dalla relazione (25) tutti i modi decadranno allo stesso modo, ma non necessariamente lo faranno alla stesso valore di ampiezza iniziale. L'ampiezza dei massimi di frequenza, come si vede nella (11) , è pari a $1/(1-g)$.

Considerando, invece, l'equazione (22), ritardi più lunghi comporteranno guadagni di feedback più piccoli, e di conseguenza picchi meno pronunciati.

Densità d'eco

La densità d'eco per i filtri comb collegati in parallelo è data dalla somma delle densità d'eco dei singoli comb.

Per ogni filtro comb *i* si ottiene in uscita un'eco ad ogni periodo di tempo τ_i, pertanto la densità d'eco complessiva, espressa come il numero di echi per secondo, è data dalla seguente formula [23]:

$$D_e = \sum_{i=0}^{N-1} \frac{1}{\tau_i} \approx \frac{N}{\tau} \tag{30}$$

Questa approssimazione è valida nel caso in cui i ritardi hanno durata simile.

Secondo questa relazione, la densità d'eco è una quantità costante in funzione del tempo, mentre nelle stanze reali tale densità aumenta con il quadrato del tempo (equazione (11)).

Dalle equazioni (26) e (29) possiamo derivare il numero di filtri comb necessari per ottenere una densità modale D_m e una densità d'eco D_e. Questa quantità è espressa dalla seguente relazione [23]:

$$N \approx \sqrt{D_m D_e} \tag{31}$$

Nel caso del riverberatore di Schroeder [21], i valori dei parametri sono scelti in modo tale da avere una densità d'eco pari a 1000 echi per secondo e una densità modale pari a 0.15 picchi per Hz.

3.6 Riverberatori allpass [6]

I riverberatori trattati in questa sezione sono tutti basati su combinazioni in serie di filtri allpass.

Schroeder sperimentò un collegamento di 5 filtri allpass in serie, con ritardi che iniziano dopo 100 msec e che diminuiscono di un fattore approssimativamente pari a 1/3, e con guadagni di feedback pari a 0.7.

Schroeder osservò che il riverbero prodotto da questi 5 filtri in serie era indistinguibile da quello delle stanze reali in termini di colorazione, fintanto che, però, si consideravano segnali in ingresso stazionari.

Altri autori ([20], [23], [26])scoprirono che i filtri allpass in serie sono, in realtà, molto suscettibili alla colorazione tonale, soprattutto se in ingresso vengono applicati segnali impulsivi.

Moorer, ad esempio, sperimentando collegamenti in serie di filtri allpass arrivò alle seguenti conclusioni:

- Maggiore era l'ordine del sistema, più tempo ci metteva la densità d'eco a raggiungere un livello apprezzabile
- La pendenza del decadimento dipendeva dalla scelta dei parametri di ritardo e di guadagno
- Il decadimento presentava un rumore metallico molto fastidioso

La z-trasformata di un collegamento in serie di filtri allpass è data dalla seguente formula:

$$H(z) = \prod_{i=1}^{N} \frac{z^{-m_i} - g_i}{1 - g_i z^{-m_i}} \tag{32}$$

dove m_i e g_i sono, rispettivamente, la lunghezza del ritardo, in campioni, e il guadagno di feedback dell'i-esimo filtro allpass. Esprimendo i guadagni in funzione della lunghezza dei ritardi, come riportato nella (23), si può far sì che i moduli dei poli siano tutti uguali; tuttavia, così facendo non si risolve il problema del suono metallico prodotto in uscita.

Gardner progettò un'altra struttura riverberante basata su filtri allpass annidati, dove il ritardo di un filtro allpass è sostituito da una connessione in serie di un modulo di ritardo con un altro filtro allpass [25]. La struttura di questo riverbero è rappresentata in Fig.11 :

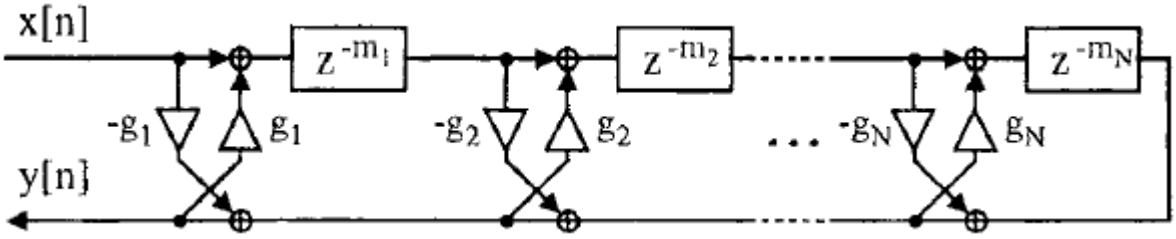


Figura 11 – Struttura di un riverberatore con filtro allpass annidato

mentre in Fig.12 ne è riportata la forma generalizzata:

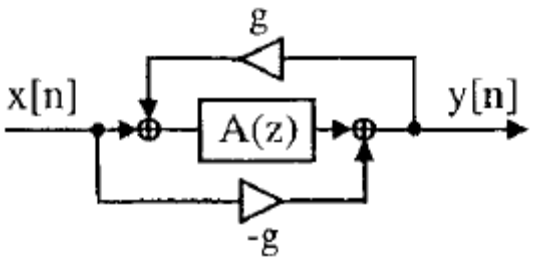


Figura 12 – Forma generalizzata del riverberatore in Fig.11

La funzione di trasferimento di questo riverberatore è data dalla formula:

$$H(z) = \frac{A(z) - g}{1 - gA(z)} \tag{33}$$

La risposta in ampiezza sarà, dunque, data dalla formula:

$$|H(z)|^2 = \frac{|A(z)|^2 - 2gRe\{A(z)\} + g^2}{1 - 2gRe\{A(z)\} + g^2|A(z)|^2} = 1 \; se \; |A(z)| = 1 \tag{34}$$

Questo filtro, nella pratica, non è realizzabile, a meno che non si esprima *A(z)* come la z-trasformata di un modulo di ritardo in serie a un filtro allpass ([22], [25]), altrimenti si forma un ciclo chiuso senza ritardo.

Il vantaggio di implementare un filtro allpass annidato è espresso dal fatto che gli echi creati dal filtro allpass annidato vengono fatti ricircolare sullo stesso tramite il percorso di feedback esterno. Pertanto, la densità d'eco di un filtro allpass annidato aumenta col tempo, come avviene nelle stanze reali.

L'esperimento di Gardner dimostra un'importante proprietà dei filtri allpass: a prescindere da quanti filtri allpass vengono annidati o collegati in serie, la risposta in uscita dell'intero sistema rimarrà sempre quella di un filtro allpass. In questo modo, il sistema risulterà sempre stabile, a prescindere dalla sua complessità.

Gardner uggerì anche una struttura generale per il riverbero monofonico [25], costituita da filtri allpass e mostrata dallo schema in Fig. 13:

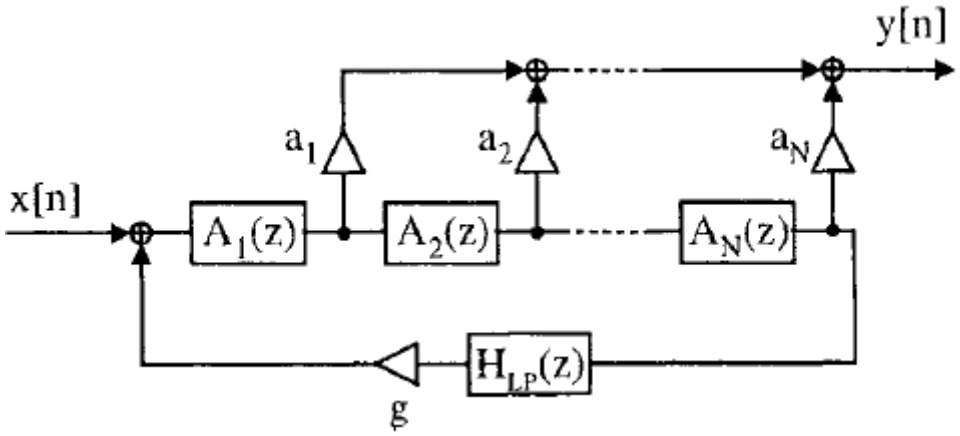


Figura 13 – Struttura di un riverbero in cui sono state aggiunte le perdite da assorbimento all'anello di feedback di un allpass

In Fig.14 è mostrato lo schema completo di un riverbero allpass ad anello di retroazione ideato da Dattorro [1]:

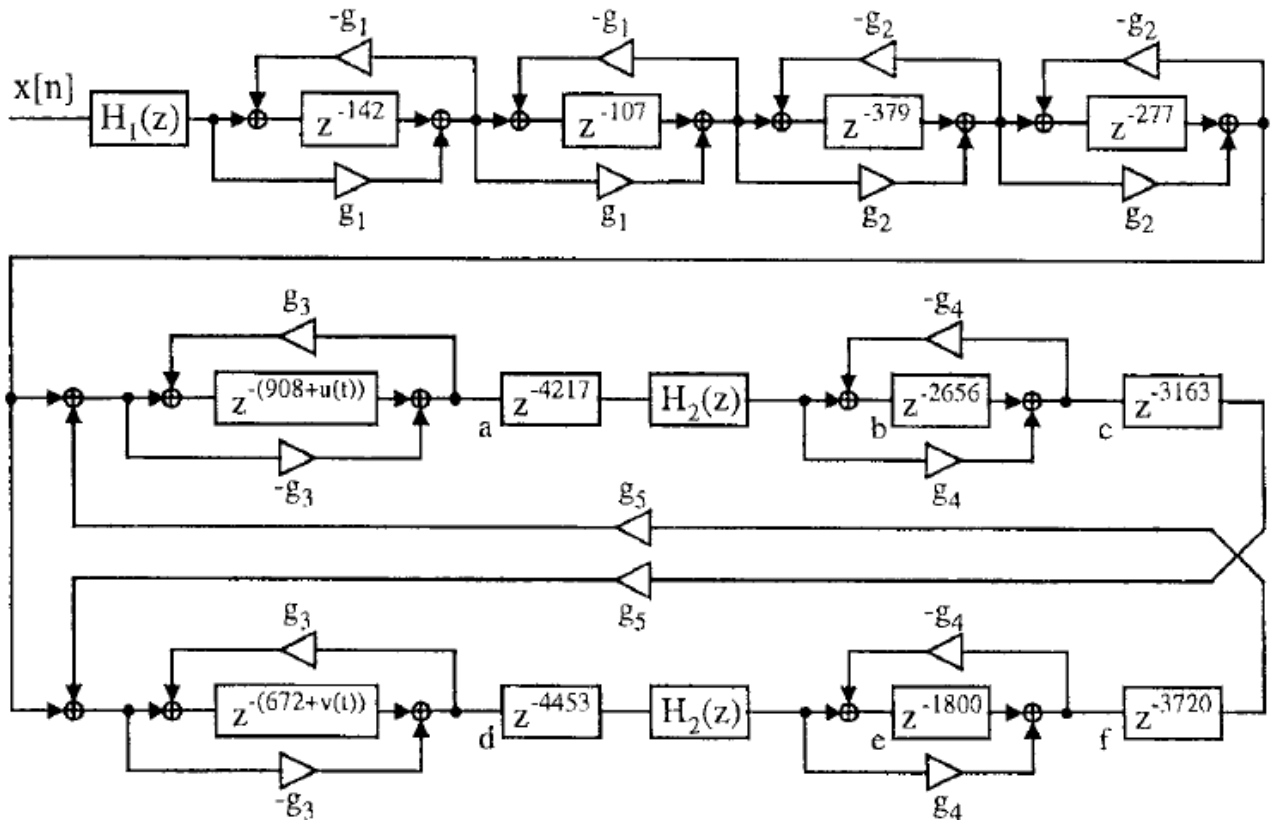


Figura 14 – Riverberatore allpass di Dattorro ad anello di retroazione

il quale attribuisce lo stile del suo riverberatore a Griesinger [26].

Come si può vedere in figura, $H_1(z)$ e $H_2(z)$ sono due filtri passa basso. In particolare, $H_1(z)$ controlla la larghezza di banda del segnale in ingresso al riverberatore, mentre $H_2(z)$ controlla il decadimento del riverbero in relazione alla frequenza. Le uscite stereofoniche, y_L e y_R, sono costituite dai contributi offerti dai diversi ritardi presenti nello schema, in base alla seguente relazione:

$$y_L = a[266] + a[2974] - b[1913] + c[1996] - d[1990] - e[187] - f[1066]$$

$$y_R = d[353] + d[3627] - e[1228] + f[2673] - a[2111] - b[335] - c[121] \quad (35)$$

Il circuito del riverberatore di Dattorro è costruito in modo da simulare il comportamento di un riverberatore elettroacustico di tipo plate, costituito da rapido accumulo di densità d'eco seguito da un decadimento esponenziale del riverbero. Il segnale monofonico in ingresso passa attraverso diversi filtri allpass caratterizzati da brevi tempi di ritardo, e successivamente entra in quello che Dattorro definisce come "serbatoio" del riverbero: si tratta, cioè, di due sistemi distinti e incrociati di filtri allpass annidati la cui struttura è analoga a quella del riverberatore in Fig.13.

Questa struttura è utile per produrre segnali stereofonici in uscita non correlati.

Il riverbero incorpora un modulo di ritardo tempo-variante in ciascuno dei due sistemi incrociati. Lo scopo di questi ritardi è quello di diminuire ulteriormente la colorazione tonale del riverbero alterando dinamicamente le frequenze di risonanza.

Gran parte degli effetti di riverberazione sviluppati dalla Montarbo nel corso degli anni prendono spunto dallo schema di Dattorro appena descritto, il quale rappresenta una soluzione particolare di combinazioni di filtri comb in parallelo e filtri allpass in serie.

Questa struttura, sebbene all'apparenza risulti piuttosto complessa, presenta i seguenti vantaggi:

1) I suoi diversi nodi permettono di controllare alcuni aspetti del suono riverberato, come il tempo di decadimento, il livello di correlazione tra il segnale principale e le sue repliche,lo smorzamento delle alte frequenze e la larghezza di banda del segnale in ingresso,
2) Tra tutte le reti di riverberazione, questa è la più efficiente a livello di complessità computazionale.
3) Può essere facilmente applicata ad una vasta gamma di sorgenti di segnale.

Oltre a tutte queste cose, la rete rappresentata in figura corrisponde anche alla rete di riverberazione più semplice che ci consente di ottenere un buon suono.

3.7 Feedback Delay Networks (FDN)

Le Feedback Delay Networks (FDN) [2, pagine 5-6] sono una generalizzazione delle reti multicanale unitarie, le quali sono la controparte a N dimensioni dei filtri comb e allpass, dove N indica il numero di linee di ritardo nel diagramma in Fig.15:

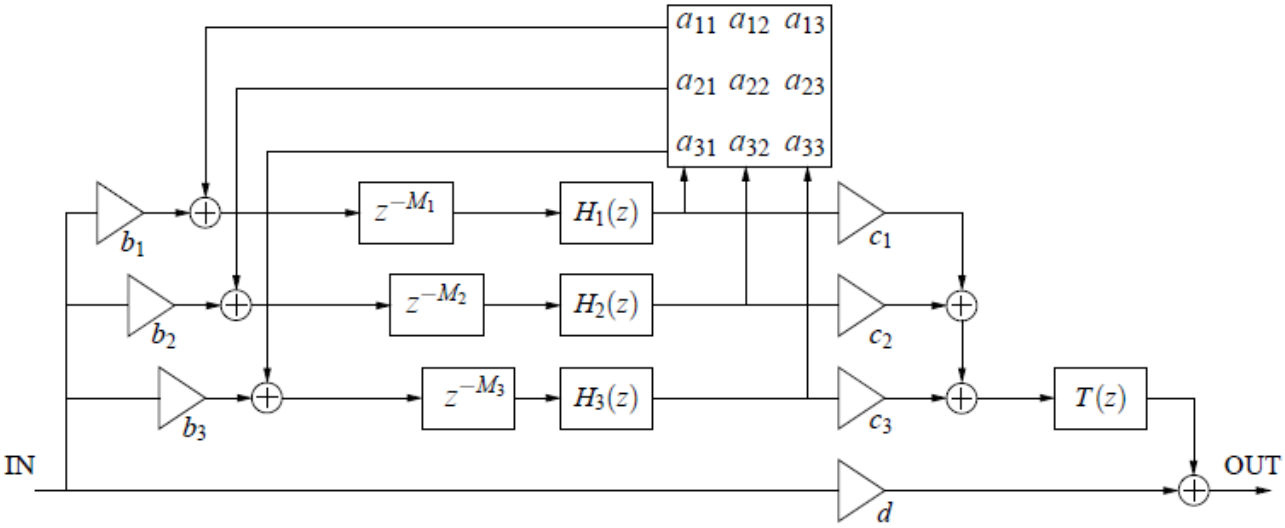


Figura 15 – Feedback Delay Networks

L'intento delle FDN è di far ricircolare il segnale all'interno della rete allo scopo di addensare, ad ogni ciclo, gli echi che si producono al suo interno, il che permette di ridurre considerevolmente la complessità di calcolo in quanto si fa a meno di implementare filtri comb e allpass; tuttavia, l'effetto risultante sarà diverso da quello che si otterrebbe applicando uno schema che segue la forma empirica discussa nel paragrafo precedente (il riverbero ottenuto risulterà molto più addensato).

In questa struttura, i coefficienti $a_{i\,j}$ della matrice quadrata controllano il livello di retroazione (feedback) dell'uscita del j-esimo delay all'ingresso dell'i-esimo delay.

A seconda del valore che si dà a questi coefficienti, questa struttura può generare un riverbero di densità davvero notevole, di gran lunga maggiore rispetto a quello generato da strutture costruite su filtri comb in parallelo.

La scelta del tipo di delay è fatta prendendo spunto dal riverberatore di Schroeder.

Nella struttura del FDN sono implementati dei filtri lowpass (passabasso), caratterizzati dalla funzione di trasferimento espressa nella seguente relazione:

$$H_i(z) = k_i \frac{1 - b_i}{1 - b_i z^{-1}} \tag{36}$$

i cui coefficienti $b_i = 1 - 2/\left(1 + k_i^{(1-1/\varepsilon)}\right)$, con $\varepsilon = T_{60}(\pi)/T_{60}(0)$, e $k_i = 10^{-3M_i T_s/T_{60}(0)}$, fanno sì che il tempo di riverberazione delle componenti in bassa frequenza $T_{60}(0)$ risulti maggiore del tempo di riverberazione delle componenti in alta frequenza $T_{60}(\pi)$.

La funzione di trasferimento *T(z)* si riferisce ad un filtro per la correzione tonale ed è data da:

$$T(z) = g_t \frac{1 - b_t z^{-1}}{1 - b_t} \tag{37}$$

dove $g_t = \sqrt{(T_s/T_{60}(0)\sum M_i)}$ e $b_t = (1 - \sqrt{\varepsilon})/(1 + \sqrt{\varepsilon})$; tale filtro compensa la distorsione che si manifesta nella risposta in frequenza dei filtri *H(z)*.

3.8 Schema generale degli algoritmi per la riverberazione

Al fine di progettare gli algoritmi per la riverberazione sulla base di filtri comb e allpass esistono delle regole empiriche che prevedono l'adozione di uno schema generale composto da tre parti:

1) Una rete composta da filtri FIR per simulare le prime riflessioni dell'ambiente.
2) Una struttura composta da un addensatore di echi, costituita da filtri comb e allpass (i primi, in genere, sono disposti in parallelo tra loro, mentre i secondi in serie) o da una rete FDN (*Feedback Delay Network*, di cui si discuterà più nel dettaglio nel paragrafo successivo).
3) Un certo numero di parametri che servono per la modellazione finale del suono riverberato: in questa fase, praticamente, si può decidere di restituire il suono riverberato in stereo, oppure in multi-uscita, scorrelando completamente l'uscita mono del riverbero dal segnale originario e concentrando una diversa quantità di riverbero in ognuno dei due canali dell'uscita stereo.

Attraverso la procedura appena discussa e disponendo in maniera diversa i filtri si è in grado di ottenere algoritmi di riverberazione che presentano caratteristiche timbriche diverse.

3.9 Anatomia di un riverbero a molla

I primi riverberi a molla furono sviluppati nei primi anni '60 e la loro struttura è divenuta,ormai,lo standard di funzionamento per tutti i moderni riverberi a molla digitali.

La stragrande maggioranza di questi riverberi consiste in un certo numero di molle disposte in parallelo, sebbene in alcuni riverberi le molle sono disposte in serie e tali serie sono, a loro volta, collegate in parallelo ad altre serie di molle.

Altri riverberi ancora implementano delle molle disposte in maniera tale da collegare tra loro le estremità delle molle collegate in parallelo, formando una 'Z'.

Ciascuno di questi riverberi produce un diverso modello di echi, avente una propria complessità.

In Fig.16 è mostrato il diagramma schematico di un riverbero a molla molto semplice, formato solamente da due molle in parallelo [3]:

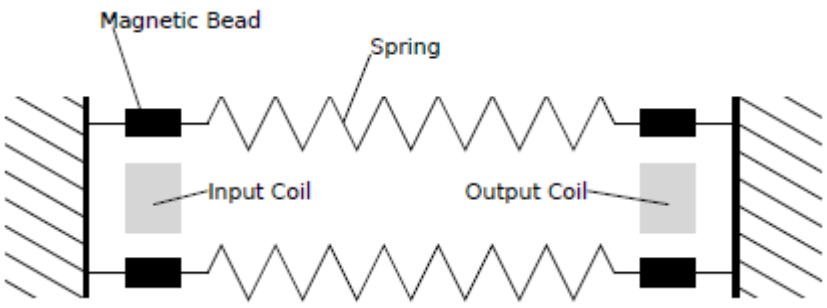


Figura 16 – Modello fisico di un riverberatore a due molle

A entrambe le molle è applicato un trasduttore ad una delle due estremità. Per ciascuna molla si invia un segnale elettrico al trasduttore, il quale lo converte in un segnale meccanico che si propaga lungo le spire della molla, facendola vibrare, fino a raggiungere un altro trasduttore, equivalente al primo, collocato all'estremità opposta della molla.

Il segnale meccanico ricevuto dal secondo trasduttore viene convertito in segnale elettrico; per la precisione, si tratta dello stesso segnale di partenza, ma ritardato di una certa quantità di tempo, proporzionale alla lunghezza della molla.

Il segnale ritardato viene, poi, inviato dal secondo trasduttore al circuito di amplificazione, miscelandosi col segnale audio di partenza.

Il secondo trasduttore (cioè quello che riceve il segnale meccanico che si propaga lungo la molla) è costituito da un nucleo ferromagnetico immerso in un solenoide. L'oscillazione della molla metterà in movimento anche il nucleo e, in virtù della legge di Faraday, tale movimento produrrà un segnale elettrico.

Tra le due molle è, inoltre, presente una bobina di rame: quando viene applicato il segnale in ingresso al trasduttore della prima molla, si genera un campo magnetico che si concatena con la bobina. Essendo,poi,il campo magnetico variabile, dal momento che la molla oscilla, esso crea una corrente indotta sul trasduttore della seconda molla, la quale viene convertita in un segnale meccanico che mette in oscillazione la molla.

3.10 Analisi della risposta impulsiva

La Fig.17 riporta lo spettrogramma della sezione iniziale della risposta impulsiva di una singola molla [3, pag.2]:

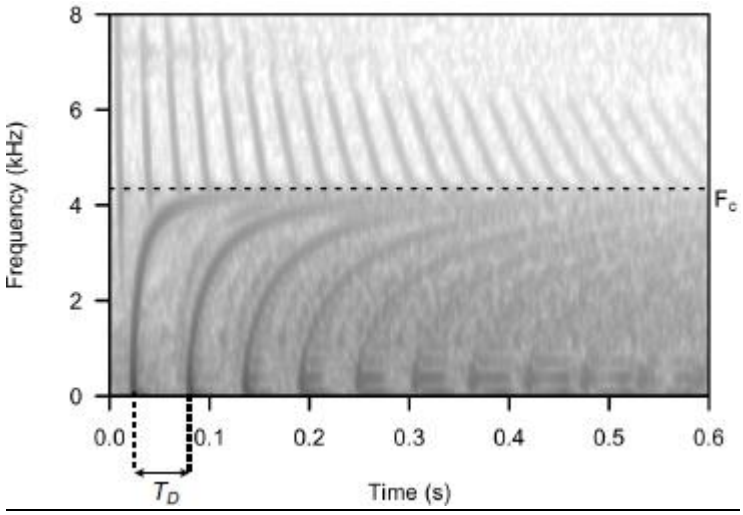


Figura 17 – Spettrogramma della risposta impulsiva di una singola molla

Come si può notare, la risposta impulsiva consiste in una sequenza di echi, la cui forma assunta delinea un comportamento dispersivo.

In particolare, possiamo distinguere due zone nella risposta impulsiva: una zona relativa alle basse frequenze, dove gli echi presentano una natura altamente dispersiva ,e una zona relativa alle alte

frequenze, dove c'è meno dispersione. Gli echi appartenenti alla zona delle alte frequenze,tra l'altro, si susseguono molto più rapidamente rispetto a quelli relativi alla zona delle basse frequenze.

Il punto di transizione tra le due zone è individuato dalla frequenza Fc, in corrispondenza della quale la risposta impulsiva si propaga a velocità minima.

Nella stragrande maggioranza dei riverberi a molla, gli echi della zona delle alte frequenze nella risposta impulsiva in uscita sono caratterizzati da un'ampiezza molto minore rispetto a quella degli echi della zona delle basse frequenze: questo comporta che il suono caratteristico di un riverbero a molla sia maggiormente legato proprio alle basse frequenze, o meglio quelle inferiori a Fc.

I parametri da tenere maggiormente in considerazione in questo caso sono, ovviamente, la frequenza Fc e il periodo di tempo che intercorre tra un eco e quello successivo, indicato con Td.

La seguente relazione esprime la forma esplicita tale tempo di ritardo:

$$T_D \approx \frac{4LR}{r\sqrt{\frac{E}{\rho}}} \tag{38}$$

dove *L* indica la lunghezza del filo che costituisce la molla, *R* è il raggio della sezione di elica formata dalla molla, *r* è il raggio del filo, *E* è il modulo di Young per il materiale che costituisce il filo e ρ è la densità di tale materiale.

Per quanto riguarda, invece, la frequenza di transizione Fc (detta anche frequenza di taglio, o cut frequency) in corrispondenza della quale cambia l'andamento degli echi nella risposta impulsiva in uscita, essa è determinata dalla seguente relazione:

$$F_c \approx \frac{3r\sqrt{\frac{E}{\rho}}}{16\sqrt{5}\pi R^2} \tag{39}$$

3.11 Modello fisico di un riverbero a piastra

Per quanto riguarda il riverbero a piastra, il principio di funzionamento è analogo a quello della molla, con la sola differenza che in questo caso si considera un sistema bidimenzionale in cui le oscillazioni si possono propagare in due diverse direzioni: lungo il lato orizzontale della piastra o lungo quello verticale.

Il riverbero a molla, come abbiamo visto, è invece descritto in termini di un sistema a una sola dimensione, in cui la propagazione delle onde meccaniche avviene lungo una sola direzione.

Come per la molla, sulla piastra vengono applicati due trasduttori: uno che riceve il segnale elettrico in ingresso e lo converte in un segnale meccanico (che si traduce in oscillazioni della piastra lungo una direzione) e un altro che converte il segnale meccanico proveniente dal primo trasduttore in un segnale elettrico, che è lo stesso segnale elettrico applicato in ingresso ma ritardato di una certa quantità. A seconda dei punti della piastra in cui vengono collocati i due trasduttori il tipo di

riverberazione ottenuto sarà diverso (in quanto sarà diversa l'entità delle oscillazioni meccaniche lungo la piastra).

Ciò che, però, contraddistingue il riverbero a piastra da quello a molla sono le caratteristiche di dispersione (ovvero,di smorzamento delle frequenze) relative alla piastra stessa, molto più influenti rispetto al caso della molla.

Per quanto riguarda il segnale in uscita dal riverbero a piastra, bisogna tenere in considerazione il fatto che si ha a che fare con un sistema a due dimensioni, il che significa che le oscillazioni possono propagarsi in due differenti direzioni: lungo il lato orizzontale della piastra o lungo il lato verticale.

Rappresentando la piastra su un sistema di assi cartesiani, i cui valori di ascisse si riferiscono al lato orizzontale della piastra, mentre quelli di ordinata al lato verticale, il segnale di uscita dal riverbero a piastra sarà rappresentato da un vettore *u*, avente due componenti x_{out} e y_{out} (rispettivamente il contributo in uscita dalle propagazioni lungo l'asse orizzontale e quello lungo l'asse verticale), espresso dalla formula:

$$u_{out} = \sum_{m=1}^{M} \sum_{n=1}^{N} q_{mn} \Phi_{mn}(x_{out}, y_{out}) \tag{40}$$

dove q_{mn} è l'ampiezza dei modi di risonanza *m* e *n* (orizzontale e verticale,rispettivamente) mentre ϕ_{mn} è la forma dei modi definita sulla lunghezza orizzontale *x* (per x che va da 0 a L_x) e su quella verticale *y* (per y che va da 0 a L_y).

I segnali di uscita da un riverberatore a piastra hanno la facoltà di propagarsi seguendo traiettorie elittiche, descrite dalle relazioni (41) e (42) :

$$x_{out}(t) = R_x L_x \sin\left(S_x \cdot \frac{2\pi t}{f_s}\right) + 0.5 \tag{41}$$

$$y_{out}(t) = R_y L_y \sin\left(S_y \cdot \frac{2\pi t}{f_s} + \theta\right) + 0.5 \tag{42}$$

dove R_x e R_y indicano i valori massimi della risposta impulsiva in uscita, relativi alle componenti orizzontale e verticale, rispettivamente, S_x e S_y indicano le velocità di propagazione del segnale in uscita, relative alle componenti orizzontale e verticale, rispettivamente, e θ indica lo sfasamento.

3.12 Riverbero a convoluzione

I riverberi a convoluzione permettono di simulare l'acustica di un ambiente a partire dalla sua risposta impulsiva, costruita sulla base delle diverse risposte impulsive che si riscontrano mediamente per quell'ambiente.

Dato un segnale acustico in ingresso, esso può essere riverberato effettuando la convoluzione tra il segnale stesso e la risposta impulsiva dell'ambiente di cui si vuole simulare l'acustica: il suono riverberato in uscita sarà il risultato di questa convoluzione.

Tuttavia, il difetto principale di questo tipo di riverbero è che non è possibile regolarne i parametri caratteristici (dimensione della stanza, tempo di riverbero, etc.) perché il riverbero a convoluzione è predisposto a riprodurre l'acustica di un certo ambiente e questo comporta che i suoi parametri hanno già dei valori di default relativi al suddetto ambiente (che definiscono, tra l'altro, l'aspetto della risposta impulsiva) e non possono essere modificati.

CAPITOLO 4

Modello di riferimento per l'algoritmo scelto

Di seguito, in Fig.18, è riportato il diagramma di flusso del riverbero scelto per la nostra tesi: è un riverbero di tipo Ambience, che di solito viene utilizzato per simulare le riflessioni che avvengono in una stanza di piccole/medie dimensioni, quindi un ambiente caratterizzato da un riverbero non troppo diffuso, le cui riflessioni hanno una breve durata.

Oltre al diagramma di flusso, sono presenti alcune tabelle che riportano i valori dei parametri principali dei building blocks costituenti il riverbero.

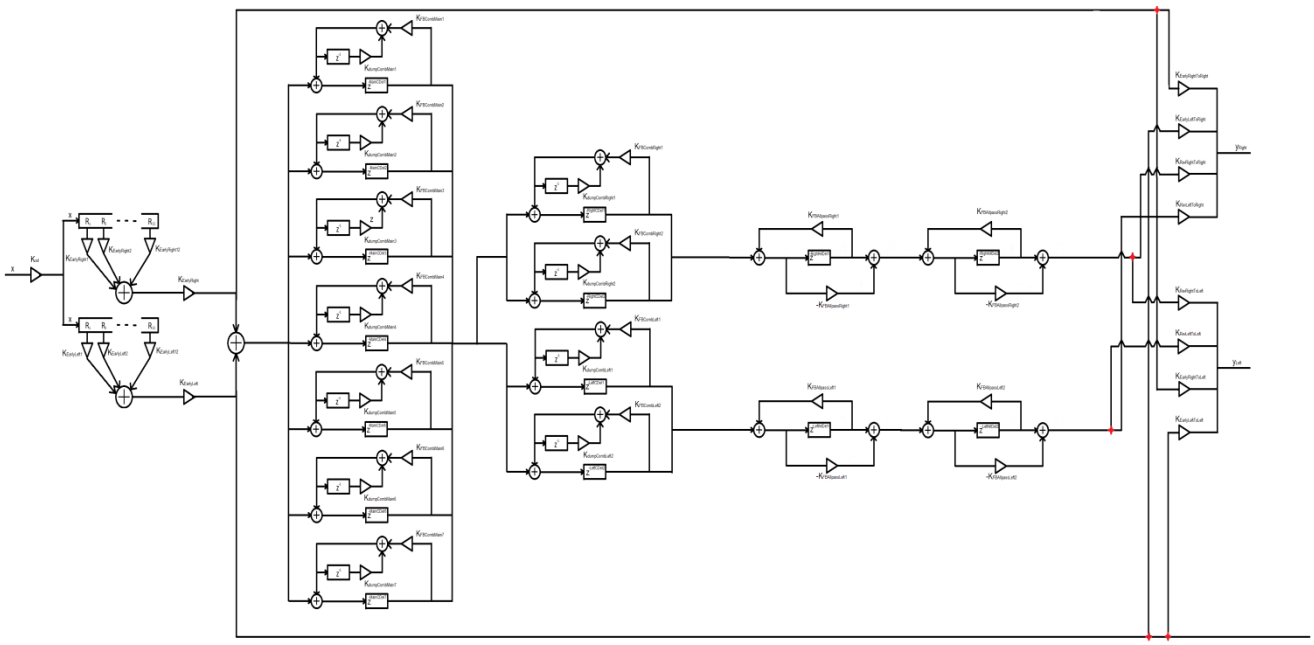


Figura 18 – Diagramma di flusso dell'algoritmo "ambience2" implementato nella tesi

Il modello comprende i seguenti building blocks:

1) 2 blocchi per le prime riflessioni (uno per il canale destro e l'altro per il canale sinistro), ciascuno dei quali è costituito da 2 filtri FIR "sparsi" (nel senso che i delay non sono z^-1 ma z^-m e rappresentano le prime riflessioni che si vogliono rappresentare) con 12 tap (cioè 12 moltiplicatori). Il tempo di riverbero è regolato con il parametro *Delay*, mentre il guadagno è regolato con il parametro *Gain*.

2) 11 filtri comb collegati in parallelo. Di questi 9 comb, 7 appartengono alla categoria *Main*, 2 alla categoria *Right* e altri 2 alla categoria *Left*. L'uscita dei 7 comb di tipo Main andrà sommata a quella dei comb di tipo Right per ottenere il contributo di segnale relativo al canale destro. La somma tra l'uscita dei 7 comb Main con quella dei 2 comb Left darà come risultato il contributo di segnale relativo al canale sinistro.
3) 2 filtri allpass, definiti come *Right Allpass*, collegati in serie: il primo di questi due allpass prende in ingresso il risultato della somma tra l'uscita dei Main comb e l'uscita dei Right comb, lo processa e restituisce in uscita un segnale che andrà in ingresso al secondo allpass della serie.
4) 2 filtri allpass, definiti come *Left Allpass*, collegati in serie: il primo di questi due allpass prende in ingresso il risultato della somma tra l'uscita dei Main comb e l'uscita dei Left comb, lo processa e restituisce in uscita un segnale che andrà in ingresso al secondo allpass della serie.

Il segnale di uscita di questo riverbero sarà dato dalla somma di 8 contributi:

1) Il contributo delle prime riflessioni sul canale destro applicato al canale destro stesso.
2) Il contributo delle prime riflessioni sul canale sinistro applicato al canale destro.
3) Il contributo delle prime riflessioni sul canale destro applicato al canale sinistro.
4) Il contributo delle prime riflessioni sul canale sinistro applicato al canale sinistro stesso.
5) Il contributo dei filtri allpass sul canale destro applicato al canale destro stesso.
6) Il contributo dei filtri allpass sul canale sinistro applicato al canale destro.
7) Il contributo dei filtri allpass sul canale destro applicato al canale sinistro.
8) Il contributo dei filtri allpass sul canale sinistro applicato al canale sinistro stesso.

Si considera, inoltre, una linea di ritardo la cui lunghezza è pari ad un valore superiore alla somma dei contributi di ritardo di ogni singolo building block. Ogni porzione di questa linea di ritardo viene dedicata alla lunghezza di ritardo di ogni singolo building block; per esempio,i primi campioni sono dedicati al primo blocco di ritardo delle early reflections left, mentre il secondo blocco occupa i campioni immediatamente successivi [9].

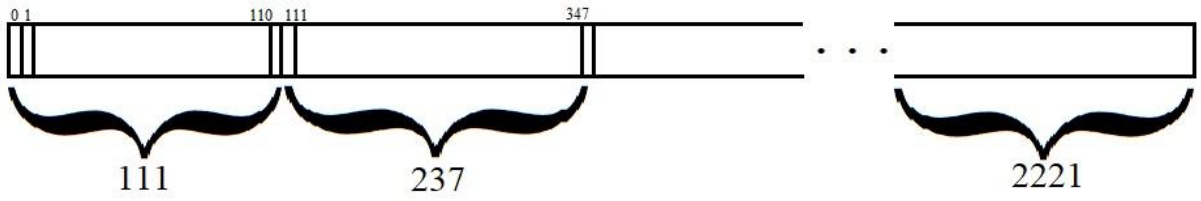


Figura 19 - Ripartizione della linea di ritardo per le EarlyReflectionsRight

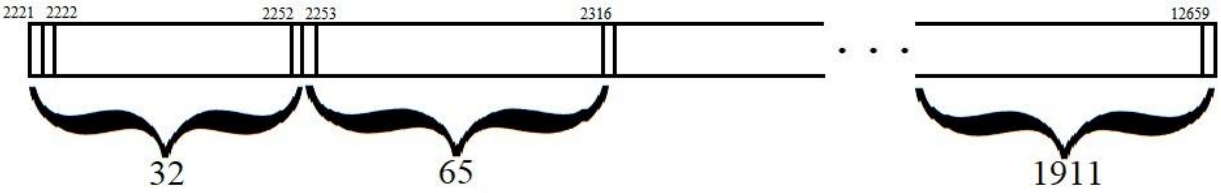


Figura 20 - Ripartizione della linea di ritardo per le EarlyReflectionsLeft

I valori dei tempi di ritardo dei filtri comb si determinano a partire dai modi di risonanza della stanza o dell'ambiente che si vuole simulare.

Per quanto riguarda, invece, il guadagno esso è determinato a partire dalla definizione del guadagno di feedback e del guadagno di dumping.

Per i filtri comb si considera un diagramma di flusso, rappresentato in Fig.21, che comprende, oltre la linea di ritardo z^{-D}e il guadagno di feedback, anche un ramo di dumping, costituito da un ritardo di z^{-1} e da un guadagno di dump k_d, a partire dal quale si determina il decadimento delle alte frequenze. Queste considerazioni valgono per tutti i comb e ciascuno di essi avrà dei propri valori di tempo di ritardo e di guadagno, nonché una diversa entità del decadimento delle alte frequenze.

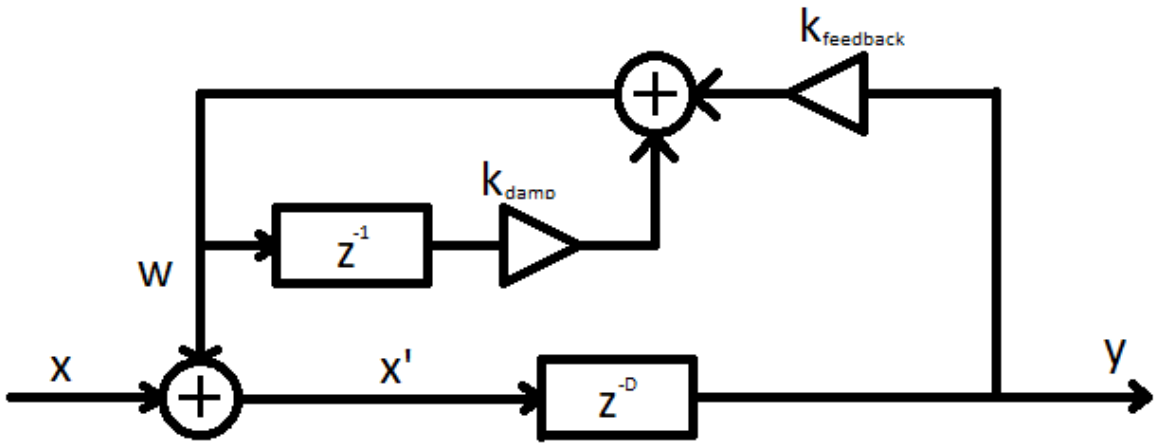


Figura 21 – Diagramma di flusso di un filtro comb, comprensivo di catena di dumping

Per i parametri dei filtri allpass si può pensare di impostare il tempo di ritardo pari a un terzo del tempo di ritardo del filtro comb più lungo, e poi a scalare (per ogni allpass) di una quantità pari a 1/3.

Infine, si controlla la risposta impulsiva di ogni filtro e poi quella di tutta la sequenza per verificare l'eventuale presenza di irregolarità troppo evidenti (buchi o picchi).

Per quanto riguarda, invece, i valori dei coefficienti necessari per il mixing dei segnali all'uscita

A seguire, in Fig.22, i valori di default utilizzati per l'algoritmo (scelti in maniera tale che le irregolarità appena menzionate non risultino troppo evidenti).

Filter 1

input volume	0,7
FM Modulator	
Frequency	3,20E-001
Depth	100
Pointer	500
filter input	1
LPF1 order	
LPF1 type	
LPF1 freq	
LPF2 order	
LPF2 type	
LPF2 freq	
Par freq	8500
Par gain	3
Par Qfact	1

Delay	Gain

Early R

Number	12
1	111 -0,9
2	237 0,9
3	411 -0,9
4	609 0,7
5	877 -0,8
6	1011 0,8
7	1234 -0,5
8	1431 0,4
9	1679 -0,6
10	1845 0,6
11	2001 0,4
12	2221 -0,3
Early Right Volume	0,7

Early L

Number	12
1	32 0,9

2	65	-0,9
3	131	0,8
4	353	0,4
5	531	-0,5
6	752	-0,7
7	971	-0,6
8	1111	0,7
9	1321	0,8
10	1541	-0,9
11	1731	0,5
12	1911	-0,5
Early Left Volume	0,7	

Filter 2

Filter post Early	1
LPF1 order	
LPF1 type	
LPF1 freq	
LPF2 order	
LPF2 type	
LPF2 freq	
Par freq	400
Par gain	-5
Par Qfact	0,9

Comb filters

	Delay	G1	G2	G3
Comb Main Number			7	
	1	0,19999	0,79996	0,0009
	1	0,19999	0,79996	0,0001
	1	0,19999	0,79996	0,0001
	1	0,19999	0,79996	1,00E-005
	1	0,19999	0,79996	0,0001
	3697	0,18366	0,65545	1,00E-005
	3921	0,18033	0,64758	0,0001
Comb Right Number			2	
	1581	0,18366	0,73464	0,5
	1921	0,18033	0,7213	0,5
Comb Left Number			2	
	1811	0,1814	0,72559	0,999
	1771	0,18179	0,72716	0,999

Allpass filters

AllPass Right Number	2	
	Delay	Gain
	2057	0,7
	21	-0,7
AllPass Left Number	2	
	Delay	Gain
	2051	-0,7
	17	0,7
Out Early R to Right	0,9	
Out Early L to Right	0	
Out Rev R to Right	0	
Out Rev L to Right	0	
Out Rev R to Left	0	
Out Rev L to Left	0	
Out Early R to Left	0	
Out Early L to Left	0,9	

Tabella 1 – Valori di default dei parametri coinvolti nel diagramma di flusso dell'algoritmo

CAPITOLO 5

DSP, XMOS e ambiente di sviluppo XTime Composer

Il metodo adottato al nostro scopo consiste nell'implementare l'algoritmo di riverberazione scelto in ambiente XTime Composer, in modo tale che il suono in ingresso possa essere elaborato in tempo reale dal DSP interfacciato, il quale mette in esecuzione le istruzioni caratterizzanti l'algoritmo processando il segnale.

XTime Composer permette fondamentalmente di scrivere e compilare il codice, ma anche di linkare ed eseguire la simulazione di debugging in maniera molto accurata del software scritto su processore XCORE.

Data l'innovazione portata dai DSP, qui verrà spiegata l'architettura interna dei DSP e come le varie problematiche di digital signal processing sono state affrontate.

Questo DSP può essere implementato sia in assembler che in linguaggio C evoluto, permettendo la programmazione in parallelo di tutti i cores all'interno del DSP.

È stato scelto di sviluppare in assembler questo particolare algoritmo per ottimizzare le risorse di calcolo, sfruttando al massimo tutte le potenzialità dei DSP, in quanto nelle applicazioni finali della Montarbo in un singolo ciclo di campionamentovengono implementati non meno di 8 algoritmi come questo contemporaneamente all'interno di ogni processore XCORE.

Quindi, scrivere in assembler l'algoritmo permette di risparmiare tempo e risorse in termini di numero di istruzioni all'interno di ogni thread, ottimizzando il calcolo interno, mentre invece il linguaggio xC, al suo stato attuale, non permette di manipolare istruzioni come la *ldd* e la *std*.

Prima di entrare in merito a come tutto questo viene fatto, è bene descrivere l'ambiente di sviluppo in cui si è lavorato.

XTime Composer è un software che gestisce l'interfacciamento di processori digitali di segnale (DSP) con un elaboratore e che permette all'utente di definire le istruzioni per l'elaborazione dei segnali audio provenienti dai DSP. Questo significa,in poche parole,che se abbiamo un segnale audio nel computer (nella nostra trattazione faremo riferimento a un impulso unitario,per questioni di semplicità) e lo vogliamo passare in ingresso a un DSP in modo da riverberarlo

I programmi XTime Composer girano su un hardware costituito da uno o più tiles, i quali a loro volta contengono diversi processori (cores).

Questi tiles sono di tipo general purpose, nel senso che essi sono in grado di eseguire istruzioni utilizzando linguaggi ad alto livello simili al C (il linguaggio utilizzato prende il nome di XCORE), e inoltre hanno un supporto diretto per il multi-threading, la comunicazione e le operazioni di I/O.

Il supporto per la comunicazione tra processi è costituito da uno switch, e sono presenti connessioni inter-chip che fanno sì che l'intero sistema sia costituito da più chip.

L'architettura su cui si basano i dispositivi XCORE è la XS2.

5.1 Interconnessioni

I tiles di XCORE sono collegati tra loro tramite un sistema di interconnessioni che gestiscono la comunicazione per mezzo di due tipologie di token: i *data token*, che sono semplicemente dei bytes di dati di controllo, e i *control token*, che invece si suddividono nelle seguenti sotto-categorie:

- *Application tokens*: vengono usati dai compilatori o da applicazioni software per implementare comunicazioni a flussi o a pacchetti di dati, codificare strutture dati e controllare i tipi di dati che viaggiano attraverso i canali di comunicazione
- *Special tokens*: definiti dal punto di vista architetturale e possono essere usati sia a livelli di hardware che di software. Vengono utilizzati per effettuare la codifica standard di tipi di dati o strutture dati standard
- *Privileged tokens*: definiti architetturalmente e possono essere usati sia a livelli di hardware che di software privilegiato. Servono per eseguire funzioni di sistema, tra cui la condivisione di risorse hardware, controllo,monitoraggio e debugging. Se si prova a passare questi tokens ad un software non privilegiato, oppure importarli da tale software, il programma crea un'eccezione.
- *Hardware tokens*: usati sono a livello hardware. Servono per controllare le operazioni che avvengono fisicamente nei link. Tentare di trasferire questi token usando istruzioni di uscita causerà un'eccezione

I messaggi che i vari processori si scambiano vengono instradati verso i *channel-ends* (in pratica, l'estremità del canale di comunicazione) di un processore specifico attraverso la rete di interconnessioni xConnect usando un *message header*, il quale contiene una serie di numeri che indicano il chip di destinazione, il processore di destinazione contenuto nel suddetto chip e il numero di terminazione di canale all'interno del processore.

La codifica dei messaggi può avvenire a 24 bit (16 bit indicano l'indirizzo del chip e quello del processore, gli altri 8 l'indirizzo del canale) o a 8 bit (3 bit per gli indirizzi del chip e del processore, 5 per l'indirizzo del canale).

Ogni switch ha un proprio numero identificativo configurabile, pertanto lo switch può essere predisposto alla trasmissione di messaggi modificando il suo numero identificativo con il message header del messaggio da trasmettere. Si fa un confronto bit-per-bit tra il message header e l'identificativo dello switch: se c'è la corrispondenza, lo switch utilizza la seconda componente del message header per instradare il messaggio al tile di destinazione, altrimenti utilizza la prima componente per impostare il chip e il processore di destinazione.

Una volta stabilita la rotta per la trasmissione dei messaggi, anche i token vengono inviati seguendo lo stesso percorso dei messaggi, fino alla trasmissione di due speciali control tokens, rispettivamente END (fine del messaggio) e PAUSE (pausa trasmissione).

5.2 Threads concorrenti

Alcuni tasks all'interno di un dispositivo XCORE possono essere eseguiti concorrentemente in threads, ognuno dei quali esegue una serie di istruzioni secondo un modello di registro a tre operandi.

I thread accedono alle risorse che il sistema mette a disposizione e che consentono sia la comunicazione tra threads diversi oppure tra un thread e l'esterno.

Ogni tile ha un supporto hardware per l'esecuzione concorrente dei threads e questo supporto è caratterizzato da:

- Un insieme di registri per ogni thread
- Uno scheduler che decide quali threads eseguire e in quale ordine
- Un insieme di sincronizzatori per l'esecuzione sincronizzata dei threads
- Un insieme di porte di ingresso e di uscita
- Un insieme di timers per controllare l'esecuzione dei threads in tempo reale
- Un insieme di generatori di clock per far sì che le trasmissioni in ingresso e in uscita dalle porte siano sincronizzate in un dominio temporale esterno
- Un insieme di hardware locks per abilitare le procedure di locking delle risorse a basso livello

Le istruzioni consistono fondamentalmente nell'inizializzazione e terminazione dei threads, e si occupano anche di farli partire, sincronizzarli e interromperli. Sono, inoltre, presenti delle istruzioni per gestire la comunicazione inter-thread e le trasmissioni I/O.

I threads di ogni tile XCORE possono fare queste cose:

- Implementare controllers I/O eseguiti concorrentemente alle applicazioni software
- Consentire la comunicazione tra threads o la trasmissione I/O contemporaneamente all'elaborazione dati
- Ridurre la latenza nelle connessioni tra threads facendo sì che alcuni threads rimangano operativi, mentre altri restano in attesa di comunicare con tiles remoti

I threads che si occupano di gestire la comunicazione e la trasmissione I/O:

- Forniscono comunicazioni e trasmissioni I/O che si verificano a seguito di opportuni eventi, aspettando i threads appena rimossi dallo scheduler
- Supportano in ogni punto del sistema la comunicazione a pacchetti, a flussi di dati o sincronizzata

- Mettono in stallo il processore disabilitando i timers quando tutti i threads di tale processore sono in attesa, allo scopo di risparmiare potenza
- Offrono la possibilità di avere interconnessioni a pipeline e trasmissioni dati I/O bufferizzate

5.3 Instruction set dei tiles

Caratteristiche principali del set di istruzioni di un tile XCORE:

- Istruzioni brevi per garantire un accesso efficiente allo stack e altre regioni di dati allocate dai compilatori, fornendo tra l'altro branching e chiamate a subroutine
- Memoria indirizzata a byte. Tuttavia,gli accessi devono essere allineati su limiti naturali, in modo che, ad esempio, gli indirizzi utilizzati in istruzioni di *load* o *store* a 32 bit presentino i due bit meno significativi impostati a zero. La memoria ordina i propri dati a partire dai byte meno significativi
- Il processore supporta un certo numero di threads, ciascuno dei quali avente un proprio insieme di registri. Le funzioni principali di questi registri prevedono l'accesso allo stack o alle regioni di dati
- Istruzioni I/O per consentire comunicazioni ad alta velocità tra threads appartenenti allo stesso tile oppure tra threads di diversi tiles. Queste istruzioni, oltre ad essere caratterizzate da un'elevata velocità di esecuzione, hanno anche una bassa latenza e sono predisposte a supportare tecniche di programmazione concorrenti ad alto livello

La maggior parte delle istruzioni in un tile hanno 16 bit e molte di esse usano come operandi dei registri in un range che va da 0 a 11, il che consente di eseguire un numero sufficiente di istruzioni a tre indirizzi.

Qualora i 12 registri (da 0 a 11) a disposizione non dovessero bastare, si utilizzano degli indici per estendere il range degli operandi e consentire più operazioni tra registri. I prefissi sono:

- *PFIX*, che concatena i 10 suoi bit più immediati con l'altrettanto immediato operando della successiva istruzione a 16 bit.
- *EOPR*, che concatena il suo insieme di 11 operazioni con l'istruzione successiva

Questi prefissi vengono inseriti automaticamente dai compilatori e dagli assemblers.

Il normale stato di un thread è rappresentato da 12 registri di operando, 4 registri di accesso e 2 registri di controllo.

Le istruzioni utilizzano i 12 operandi (i registri *r0, r1,..., r11*) per eseguire operazioni aritmetiche e logiche, accedere a strutture dati e chiamare subroutines.

La Fig.22 mostra l'elenco dei registri di accesso:

register	number	use
cp	12	constant pool pointer
dp	13	data pointer
sp	14	stack pointer
lr	15	link register

Figura 22 – Registri di accesso

La Fig.23 mostra l'elenco dei registri di controllo:

register	number	use
pc	16	program counter
sr	17	status register

Figura 23 – Registri di controllo

Ogni thread ha, poi, sette registri aggiuntivi, come mostrato in Fig.24:

register	number	use
spc	18	saved pc
ssr	19	saved status
et	20	exception type
ed	21	exception data
sed	22	saved exception data
kep	23	kernel entry pointer
ksp	24	kernel stack pointer

Figura 24 – Registri aggiuntivi per ogni thread

Il registro di stato *sr* contiene le informazioni mostrate in Fig.25:

bit	number	use
eeble	0	event enable
ieble	1	interrupt enable
inenb	2	thread is enabling events
inint	3	thread is in interrupt mode
ink	4	thread is in kernel mode
reserved	5	do not use
waiting	6	thread waiting to execute current instruction
fast	7	thread enabled for fast input-output
di	8	thread is running in dual issue mode
kedi	9	thread switches to dual issue on kernel entry
hipri	10	thread is in high priority mode

Figura 25 – Informazioni sullo stato del registro "sr"

5.4 Implementazione dello scheduler

Dal momento che i threads consentono l'esecuzione in tempo reale di diversi tasks e operazioni I/O è molto importante che l'esecuzione di ogni singolo task avvenga nel modo più efficiente possibile.

Il metodo di scheduling utilizzato in XCORE prevede che diversi threads condividano una memoria e un'interfaccia I/O, garantendo che per n threads da eseguire, ciascuno di essi impieghi 1/n cicli di processore.

Da un punto di vista progettuale del software, questo significa che la minima prestazione accettabile che ci si aspetta da un thread può essere calcolata a partire dal numero di threads concorrenti in un certo punto del programma. L'esecuzione dei singoli threads può, a volte, essere ritardata mentre questi sono in attesa di dati da trasmettere in uscita o da ricevere in ingresso e pertanto i cicli di processore inutilizzati da questi threads passano ad altri threads, quindi da questo punto di vista la prestazione non fa che aumentare. Inoltre, il tempo necessario per far ripartire i threads in attesa è sempre pari a un ciclo di processore.

Quindi, n threads di un insieme possono essere visti come n processori virtuali ciascuno caratterizzato da un clock rate di una quantità pari ad almeno 1/n rispetto al clock rate del processore stesso. Tuttavia, se il numero n di threads è inferiore alla profondità p della pipeline, il clock rate sarà al massimo 1/p.

Ad ogni thread è associato un buffer da 256 bit che può contenere 16 istruzioni brevi, o 8 istruzioni lunghe. Tali istruzioni vengono messe in issue dai threads in esecuzione seguendo una politica di

ordinamento di tipo Round Robin, ignorando i threads non in uso oppure in attesa di un'operazione di sincronizzazione o I/O.

La pipeline consente a tutte le istruzioni l'accesso alla memoria, secondo le seguenti regole per l'esecuzione del fetch di istruzione:

- Ogni istruzione che richiede l'accesso alla memoria viene eseguita durante la fase di tale accesso
- Le istruzioni di branch eseguono il fetch dell'obiettivo di tale branch durante la fase di accesso alla memoria, a meno che non richiedano anche loro l'accesso alla memoria (in tal caso, il buffer verrà lasciato vuoto)
- Vengono eseguiti branch condizionali soltanto ogni volta che vengono eseguiti fetch di istruzioni attorno all'indirizzo di branch
- Tutte le altre istruzioni non di branch (es. le istruzioni ALU) eseguono il fetch durante la fase di accesso alla memoria, allo scopo di caricare il buffer di istruzioni del thread finché non è del tutto saturo
- Se il buffer di istruzioni è vuoto quando deve essere messa in issue un'istruzione, viene messo in issue un *fetch no-op* che, durante la fase di accesso alla memoria, carica il buffer di istruzioni del thread da mettere in issue

Certe istruzioni rendono i thread non eseguibili,ad esempio quando in un canale di ingresso non vengono ricevuti dati; in tal caso, il thread viene lasciato in stallo e viene fatto ripartire dal punto in cui era stato messo in pausa una volta che i dati sono disponibili.

Ad ogni thread è, infatti, assocato un segnale ready request (richiesta pronta), quindi una volta che il thread invia il suo codice identificatore alla risorsa di cui ha bisogno, tale risorsa userà l'identificatore per scegliere il giusto segnale ready request. Il thread in questione verrà,quindi,fatto ripartire da zero reintroducendolo nella sequenza Round Robin e rimettendo in issue l'istruzione di partenza.

Esiste una modalità *fast* per la trasmissione I/O da parte del processore virtuale. Quando un thread si trova in modalità fast, esso non viene rimosso dallo schedule nel caso una sua istruzione non venga completata, ma viene lasciato in esecuzione rimettendo in issue tale istruzione finché non viene completata.

5.5 Notazioni dell'instruction set

- *Bpw*, numero di byte in una word
- *bpw*, numero di bit in una word
- *mem*, memoria
- *pc*, program counter
- *sr*, status register
- *sp*, stack pointer
- *dp*, data pointer

- *cp*, constant pool pointer
- *lr*, link register
- *r0, r1, ... , r11*, registri operandi
- *x* rappresenta uno qualsiasi dei registri operandi
- *X* rappresenta uno qualsiasi dei registri operandi o uno tra *sp*, *dp*, *cp* e *lr*
- *us*, un operando sorgente di tipo unsigned definito nel range 0 ... 11
- *bitp*, uno tra *bpw*, 1, 2, 3, 4, 5, 6, 7, 8, 16, 24 e 32 codificato come *us*
- *u16*, operando unsigned a 16 bit definito nel range 0 ... 65535
- *u32*, operando unsigned a 32 bit definito nel range 0 ... 1048575
- *iw*, issue-width (larghezza di issue),espresso in byte (2 in caso di single issue, 4 in caso di dual issue)

5.6 Architettura interna

Un tile consiste in un insieme di risorse hardware, comprensive di:

1) Un certo numero di processori (cores) che eseguono il codice
2) Un segnale di clock di riferimento (solitamente del valore di 100 MHz)
3) Una memoria
4) Una sotto-sistema di periferiche I/O per il flusso di dati in ingresso e in uscita

Solo il codice in esecuzione su un tile può accedere direttamente alle risorse di quel tile.

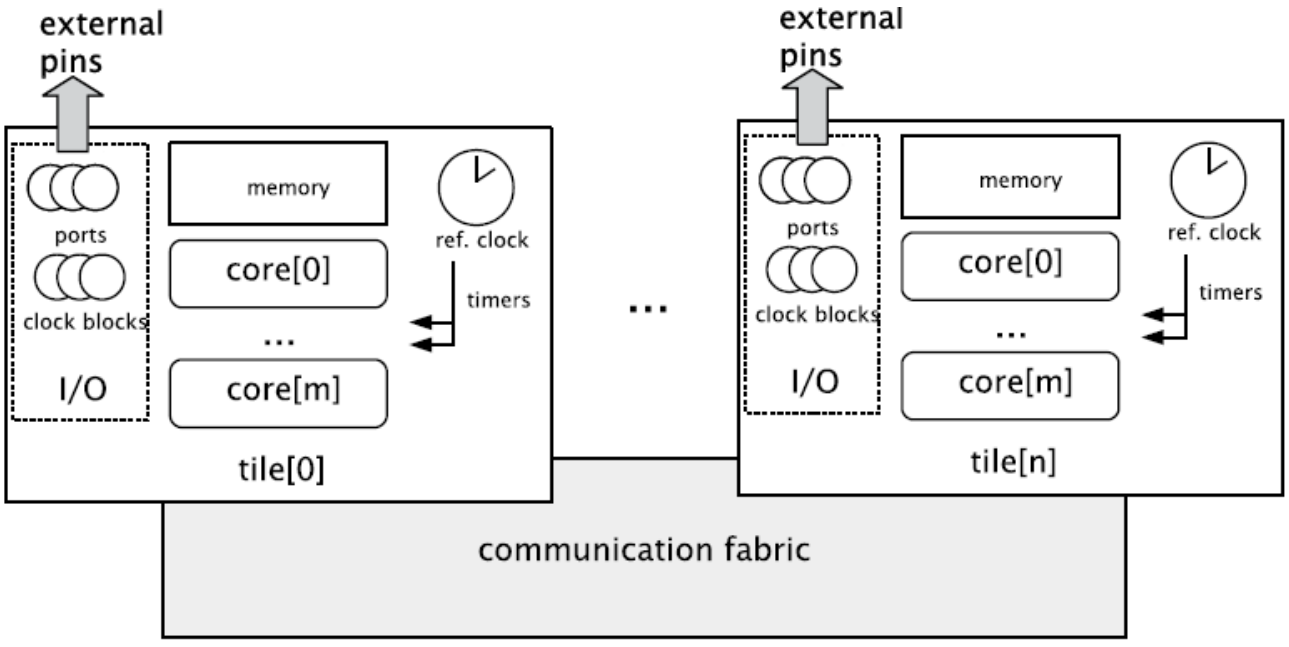


Figura 26 – Architettura di un DSP XCORE

In Fig.26 è mostrata l'architettura di un DSP XCORE.

In particolare, nei sistemi XMOS XS1 si hanno le seguenti caratteristiche:

1) Ogni tile occupa 64KB di memoria
2) La memoria di ogni tile è sprovvista di cache

3) Le periferiche utilizzano il sotto-sistema I/O, il quale non fa uso del bus della memoria per il flusso di dati

I task di diversi tile non condividono memoria, ma possono comunicare tra loro mediante comunicazione inter-core.

5.7 Cores

Un core è un'unità di elaborazione indipendente su cui eseguire il codice. All'interno di un singolo tile, tutti i core vengono eseguiti in parallelo.

Inoltre, a seconda di come è configurato il dispositivo, ogni core può operare ad una diversa velocità, garantendo però un minimo MIPS (Milioni di Istruzioni Per Secondo).

I cores di un certo tile sono messi in comunicazione con i cores di un altro tile per mezzo di un bus di trasmissione dati, detto Communication Fabric, consentendo ad ogni task di effettuare una transazione con un altro task, indipendentemente dal fatto che il secondo task si trovi o meno sullo stesso core (o addirittura sullo stesso tile) del primo task.

5.8 I/O

Il sotto-sistema di periferiche I/O consiste fondamentalmente in porte e blocchi di clock.

1) Le porte consentono l'accesso agli spinotti (pins) del dispositivo, effettuando il campionamento dei dati audio in ingresso provenienti dagli input pins ed instradando i dati audio in uscita direttamente agli output pins. Ogni porta ha la sua dimensione (1 bit,4 bit... 32 bit...). In generale, una porta a n bit campionerà n bit in ingresso in una sola volta, oppure instraderà in uscita n bit.
2) I blocchi di clock forniscono alle porte un segnale di clock per il controllo sincronizzato della lettura e della scrittura. All'interno di ogni porta è presente un registro (shift register) che contiene i dati in ingresso o quelli in uscita, a seconda che la porta in questione sia una porta di ingresso o di uscita. Ad ogni ciclo di clock, la porta di ingresso campiona i dati provenienti dagli input pins e immagazzina i campioni all'interno dello shift register, mentre la porta di uscita preleva tali campioni e li instrada verso gli output pins.

Ogni porta può essere impostate in uno dei due seguenti modi:

1) Divide: la frequenza di clock della porta risulta essere una frazione intera della frequenza di clock di un core del DSP.
2) Externally driven: la frequenza di clock della porta è regolata da un segnale esterno proveniente da un'altra porta di ingresso.

5.9 Parallelismo

I programmi XCORE sono costituiti da tasks che vengono eseguiti in parallelo. L'esecuzione in parallelo di più tasks è reso possibile con il costrutto *par*:

```
par {
        task1(5);
```

```
            task2();
}
```

Questo statement esegue in parallelo *task1* e *task2*, attendendo che l'esecuzione di entrambi sia terminata prima di proseguire.

La funzione *main* definisce tasks su diverse entità hardware; quindi, se ho diversi tasks da eseguire e due tiles a disposizione e se, per esempio, ogni tile dispone di tre cores, la funzione *main* stabilisce a quali core affidare l'esecuzione di un particolare task (es. L'esecuzione del *task1* può essere affidata al *core1* del *tile1*, mentre l'esecuzione di *task2* al *core2* del *tile0*, mentre infine *task3* può essere eseguito dai cores *core1* e *core2* del *tile0*). In ogni caso, tutti i tasks verranno eseguiti parallelamente, a prescindere da quali cores di quali tiles si occupino della loro esecuzione.

Quando più tasks vengono eseguiti sullo stesso core si parla di *cooperative multitasking*, mentre quando più tasks vengono eseguiti attraverso più cores si parla di *task distribuibili*.

Per stabilire a quale/i core/cores assegnare i tasks si usa il costrutto *on* combinato con un *par* all'interno della funzione *main*:

```
int main(){
    par{
        on tile[0]:task1();
        on tile[1].core[0]:task2();
        on tile[1].core[0]:task3();
         }
}
```

In questo caso, *task2* e *task3* vengono allocati sullo stesso core. In generale, se non si specifica nello statement alcun core (come nel caso di *task1*), il task viene allocato automaticamente ad un qualsiasi core.

5.10 Comunicazione

I tasks comunicano tra loro attraverso transazioni, a prescindere dai cores e dai tiles in cui si trovano.

Il compilatore implementa le transazioni utilizzando l'hardware di comunicazione sottostante ai tiles.

5.11 Connessione a interfaccia

Un'interfaccia mette in comunicazione diversi tasks, i quali potrebbero trovarsi anche nello stesso core. Inoltre, le interfacce consentono le notifiche di ricezione di segnali asincroni tra tasks durante le comunicazioni sincrone.

La funzione *interface* definisce un'interfaccia di collegamento tra due o più tasks; all'interno di tale funzione si definiscono il tipo di transazioni che si possono verificare tra i tasks e i dati che essi si scambiano.

Es:

```
interface i{
        void fA(int x, int y);
        void fB(float x);
}
```

In questo esempio, la prima transazione definisce due variabili intere, x e y, che i task si scambiano tra loro, mentre la seconda transazione definisce una variabile float che transita da un task all'altro.

Una connessione a interfaccia tra due tasks è costituita da:

1) Il *client end*, che è l'istruzione da cui parte la struttura d'interfaccia e che inizializza le transazioni
2) Il *server end*, che è l'istruzione in cui termina la struttura d'interfaccia e che risponde alle transazioni

Quando l'istruzione riferita al client end viene passata ad un task, quest'ultimo viene chiamato *client task*: esso può, a questo punto, richiedere l'esecuzione delle sue operazioni ad un altro task che riceve l'istruzione riferita al server end e che, pertanto, viene chiamato *server task*.

Una volta che il server task esegue le operazioni per conto del client task, esso ritorna a quest'ultimo i risultati delle suddette utilizzando la stessa interfaccia di cui si è servito il client task per inviare i dati da elaborare al server task.

L'inizializzazione delle transazioni da parte del client task avviene per mezzo di una sintassi simile ad una chiamata di funzione:

```
void task1(client interface a){
        i.fA(5,10);
}
```

dove i indica il client task, quindi l'istruzione i.fA indica che il client task richiede l'esecuzione delle operazioni incluse nella transazione fA, passando i valori 5 e 10.

Il server task esegue le transazioni commisionategli dal client end per mezzo del costrutto *select*, il quale mette in attesa il task finché uno dei possibili casi al suo interno non si verifica, ovvero finché una delle diverse transazioni non viene inizializzata nel client task. Es:

```
void task2(server interface a){
        select{
            case i.fA(int x, int y);
                 printf("Received fA: %d,%d \n",x,y);
                 break
            case i.fB(float x)
```

```
                    printf("Received fB: %d,%d \n",x);
                    break
                    }
}
```

In questo caso, una volta dichiarato task2 come server task passandogli il server end (indicato come server interface a), il costrutto select lo mette in attesa che il client task inizializzi una delle due transazioni tra fA e fB (select seleziona un solo evento) prima di procedere.

È possibile unire tra loro i tasks in esecuzione dichiarando un'istanza di interfaccia e passandola come argomento ad entrambi i tasks:

```
int main(void){
        interface my_interface i;
        par{
            task1(i);
            task2(i);
        }return 0;
}
```

NOTA: solo uno dei tasks può ricevere il client end e solo uno può ricevere il server end; se più di un task utilizza un server o un client end, ciò causerà un errore di compilazione.

L'attributo [[notification]] dichiara la funione di notifica, la quale viene dichiarata come *slave* per indicare che la direzione di comunicazione va dal server end al client end. Le funzioni di notifica non devono ricevere argomenti e devono presentare un valore di ritorno di tipo void.

5.12 Connessione a canale

I canali mettono in comunicazione i tasks, senza però definire alcun tipo di transazione; essi costituiscono il livello di comunicazione più basso e non possono essere usati per mettere in comunicazione i tasks sullo stesso core.

La connessione tra due tasks per mezzo di un canale avviene tramite il costrutto *chan*:

```
chan c
    par{
        task1(c);
          task2(c);
        }
```

Con i canali vengono introdotti gli operatori speciali <: e :>, i quali servono, rispettivamente, a inviare e a ricevere dati.

Le operazioni I/O dei canali sono sincrone di default,per cui per ogni trasmissione, il task che ha inviato il valore rimane bloccato fino a quando l'input del task dall'altro capo del canale non ha ricevuto il valore. A seconda della quantità di tempo necessaria perché un dato inviato da un task venga ricevuto da un altro task può ridurre le prestazioni: per risolvere questo problema si può fare ricorso ai canali di flusso che consente ai due task lo scambio di dati senza sincronizzazione.

5.13 Tipologie di tasks

Esistono tre diverse tipologie di tasks:

1) Normali: i tasks vengono eseguiti su un singolo core, indipendentemente dagli altri tasks.
2) Combinabili: i tasks possono essere combinati tra loro, in modo che su un singolo core possano essere eseguiti più tasks.
3) Distribuibili: i tasks possono essere eseguiti su diversi cores.

5.14 Campionamento, quantizzazione e interfacciamento con XMOS

I DSP hanno bisogno di un interfacciamento col mondo analogico, in quanto il segnale analogico proveniente da una sorgente esterna deve essere innanzitutto digitalizzato affinché l'algoritmo possa elaborarlo all'interno di un DSP.

I riverberi digitali consistono fondamentalmente in circuiti integrati che contengono, oltre ai già menzionati convertitori A/D e D/A e filtri FIR, anche dei banchi di RAM che svolgono la funzione di buffer.

Il primo passo per ottenere un segnale in uscita da un riverbero digitale consiste nel convertire il segnale analogico in ingresso in segnale digitale. Per farlo occorre un convertitore analogico/digitale che effettua sul segnale due operazioni fondamentali: il campionamento e la quantizzazione.

Campionamento: il campionamento consiste nel prendere la forma d'onda di un segnale in ingresso e suddividerla in un certo numero di campioni, definiti in intervalli di tempo, detti intervalli di campionamento, la cui ampiezza dipende dal valore della cosiddetta *frequenza di campionamento*.

In parole povere,si tratta di prendere diversi spezzoni del segnale in ingresso definiti in diversi istanti di tempo e considerarne il valore in ampiezza. La frequenza di campionamento definisce la lunghezza degli intervalli di campionamento e, di conseguenza, il numero di campioni in cui viene suddiviso il segnale; la lunghezza degli intervalli di campionamento (detta *periodo di campionamento*) è definita come inverso della frequenza di campionamento.

In Fig.27 è mostrato il diagramma di un segnale continuo campionato ad una frequenza di campionamento Fc:

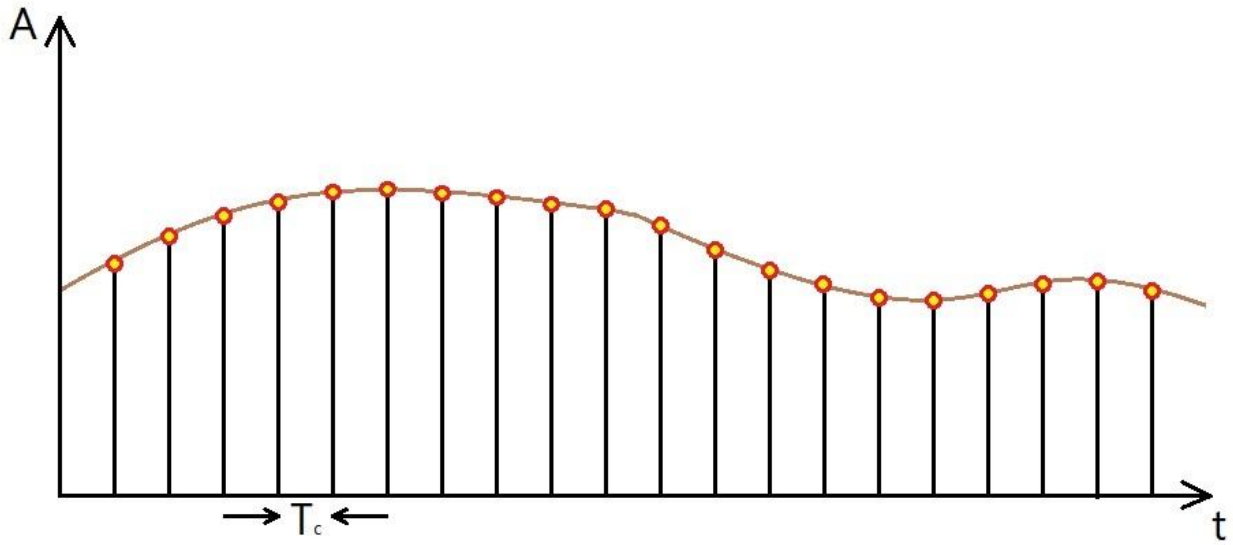


Figura 27 – Campionamento di un segnale

Il campionamento è necessario in quanto i dispositivi elettronici digitali non sono in grado di lavorare con precisione infinita,è necessario convertire il segnale analogico in esame in formato digitale, e quindi *discreto*.

In generale, tanto maggiore è la frequenza di campionamento quanto più accurato è il campionamento del segnale, in quanto un maggior numero di campioni consente di ricostruire più fedelmente il segnale in formato digitale.

La minima frequenza di campionamento necessaria per poter ricostruire accuratamente il segnale a valle è definita a partire dal teorema di Nyquist: la frequenza di campionamento deve essere almeno il doppio della frequenza massima dello spettro del segnale da campionare:

$$f_c > 2 * f_m \tag{43}$$

Qualora il teorema di Nyquist non venisse rispettato, si verifica il fenomeno dell'aliasing, che comporta una distorsione del segnale analogico ricostruito rispetto a quello originale.

Se, infatti, si sottocampiona il segnale nel dominio del tempo, nel corrispondente dominio delle frequenze si verifica il formarsi di frequenze non appartenenti al segnale originario per effetto della sovrapposizione, anche parziale, di quelle già presenti, e viceversa dal dominio delle frequenze al dominio del tempo, distorcendo il segnale originario.

Nel caso dei segnali audio, caratterizzati da uno spettro in frequenza che si estende dai 20Hz ai 20KHz, i convertitori A/D sono impostati in modo effettuare il campionamento con una frequenza di campionamento pari almeno al doppio della frequenza massima dello spettro di un segnale audio, che in questo caso vale 40KHz.

Il valore standard di frequenza di campionamento impostato nei moderni convertitori A/D è, per la precisione, pari a 44100 Hz. Il motivo è che le prime apparecchiature di conversione erano basate sui registratori video.

Il formato NTSC, che è lo standard di conversione americano, utilizza 30 frame di 525 linee al secondo (di cui se ne utilizzano solo 490); con 3 campioni per linea, si ottiene una frequenza di campionamento pari a:

$$f_c = 30 \cdot 490 \cdot 3 = 44100\ Hz \tag{44}$$

Stesso discorso per il formato PAL, lo standard europeo, che usa 25 frame di 625 linee al secondo (di cui solo 588 sono utilizzabili); con 3 campioni per linea, la frequenza di campionamento risulterà sempre pari a 44100 Hz:

$$f_c = 25 \cdot 588 \cdot 3 = 44100\ Hz \tag{45}$$

Talvolta si predilige una frequenza di campionamento più alta, pari a 48000 Hz, onde ridurre il rischio che si verifichi l'aliasing e anche per rendere meno problematici i filtraggi pre e post conversione.

Quantizzazione: la quantizzazione indica un processo in cui un segnale analogico a valori continui viene convertito in un segnale digitale a valori discreti.

Vengono, innanzitutto, decisi i valori minimo e massimo che devono avere i valori discreti del segnale, creando così delle regioni di decisione e la dinamica del quantizzatore.

Quindi, preso un qualsiasi valore in ampiezza del segnale analogico, esso verrà approssimato al valore discreto più prossimo nelle regioni di decisione.

Definendo con *errore di quantizzazione* la differenza tra l'ampiezza del valore analogico con il valore digitale a cui è stato fatto corrispondere, l'errore massimo che si può commettere nel digitalizzare un valore analogico sarà pari alla metà del valore della regione di decisione, nel caso in cui il valore del segnale analogico ricade al confine tra due regioni di decisione.

Il segnale campionato quantizzato è rappresentato in Fig.28:

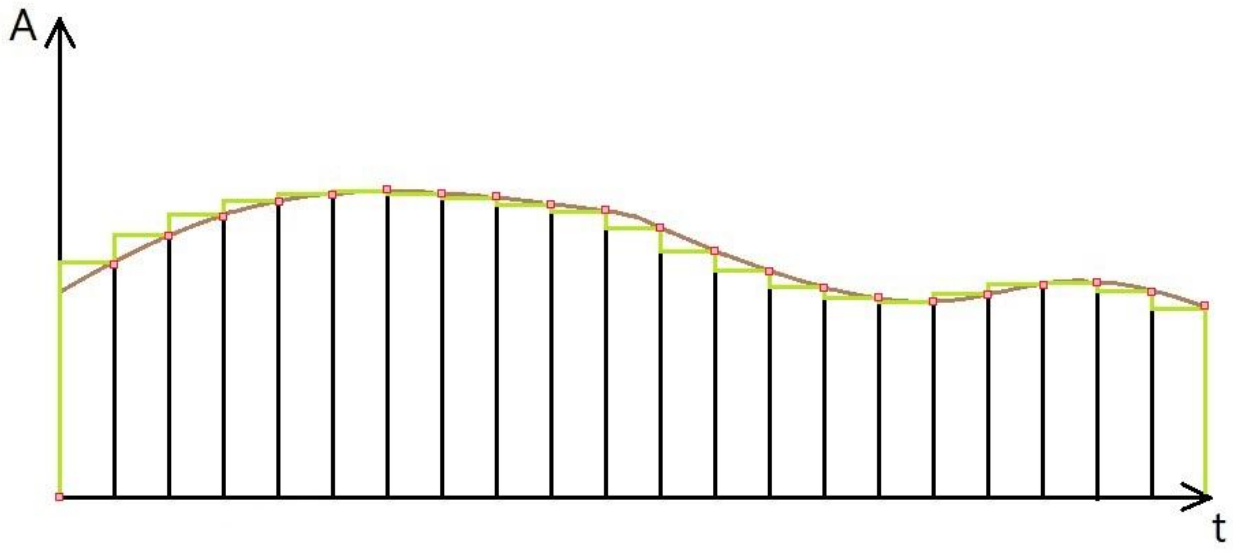


Figura 28 – Quantizzazione di un segnale campionato

L’insieme di questi rumori conduce al Rapporto Segnale/Rumore (Signal to Noise Ratio, SNR), il quale misura l’accuratezza della quantizzazione.

La quantizzazione lineare avviene tramite la tecnica PCM (Pulse Code Modulation), in cui:

- Il numero di livello di decisione è dato da $M = 2^n$, dove n indica il numero di bit di quantizzazione utilizzato dal convertitore
- L’ampiezza di ogni livello è $q = V/M$, dove V è il valore del segnale nell’intervallo considerato
- La varianza è data da $\frac{q^2}{12}$

È chiaro che maggiore è il numero di bit (e,dunque,il numero di livelli di quantizzazione) più accurato è il segnale quantizzato ottenuto.

In campo audio, lo standard di digitalizzazione dei segnali audio corrisponde a una frequenza di campionamento pari a 44100 Hz e un numero di bit di quantizzazione pari a 16, che è anche quello utilizzato per la lettura dei cd musicali. Ciononostante,è possibile lavorare sui segnali audio anche a valori superiodi di fc e di n, basti pensare che a livello professionale la frequenza di campionamento scelta può arrivare addirittura a 196000 Hz e che si possono usare 32 bit per la quantizzazione.

In altre parole, il segnale digitalizzato viene immagazzinato nel primo banco di RAM e successivamente applicato alla catena di FIR; i risultati intermedi del processamento del segnale nei vari building blocks della catena di FIR vengono memorizzati nei successivi banchi di RAM, ed infine il segnale nell’ultimo banco, corrispondente all’uscita complessiva del riverbero, viene convertito in segnale analogico e restituito in uscita [10].

Con un solo chip è possibile realizzare un singolo eco digitale con tempo di riverberazione ridottissimo.

Tuttavia, la digitalizzazione del segnale e la successiva riconversione in segnale analogico causa una perdita di qualità nel suono che è tanto più piccola quanto più accurato è il campionamento del segnale analogico in ingresso.

5.15 Trattamento dei dati da parte del protocollo I^2S

In XMOS, la conversione analogica-digitale è resa possibile da una PCM (Pulse Code Modulation, modulazione a codice di impulsi) lineare a 24 bit di quantizzazione.

Perciò, la dimensione degli intervalli di decisione in cui si suddivide l’intervallo dei possibili valori del segnale digitale è sempre la stessa; qualunque sia il valore quantizzato, l’accuratezza con cui lo si imposta sarà sempre la stessa.

La trasmissione di flussi di dati audio in XMOS avviene per mezzo di un’apposita interfaccia digitale, detta I^2S [7].

La dicitura I^2S indica un protocollo tra due dispositivi, dove uno di essi svolge il ruolo di master, mentre l’altro quello di slave.

Tramite la funzione *send*, I^2S applica i dati presi in ingresso al convertitore AD all'interno del quale vengono campionati. A questo punto, i dati campionati vengono presi dalla funzione *receive*, tramite la quale I^2S colloca i dati all'interno di un buffer. Nel buffer, i dati immagazzinati vengono elaborati dal software XTime Composer secondo le istruzioni definite dall'utente, e infine vengono spediti al convertitore DA tramite una *send*.

L'intero procedimento appena descritto è riassunto nel diagramma di flusso in Fig.29:

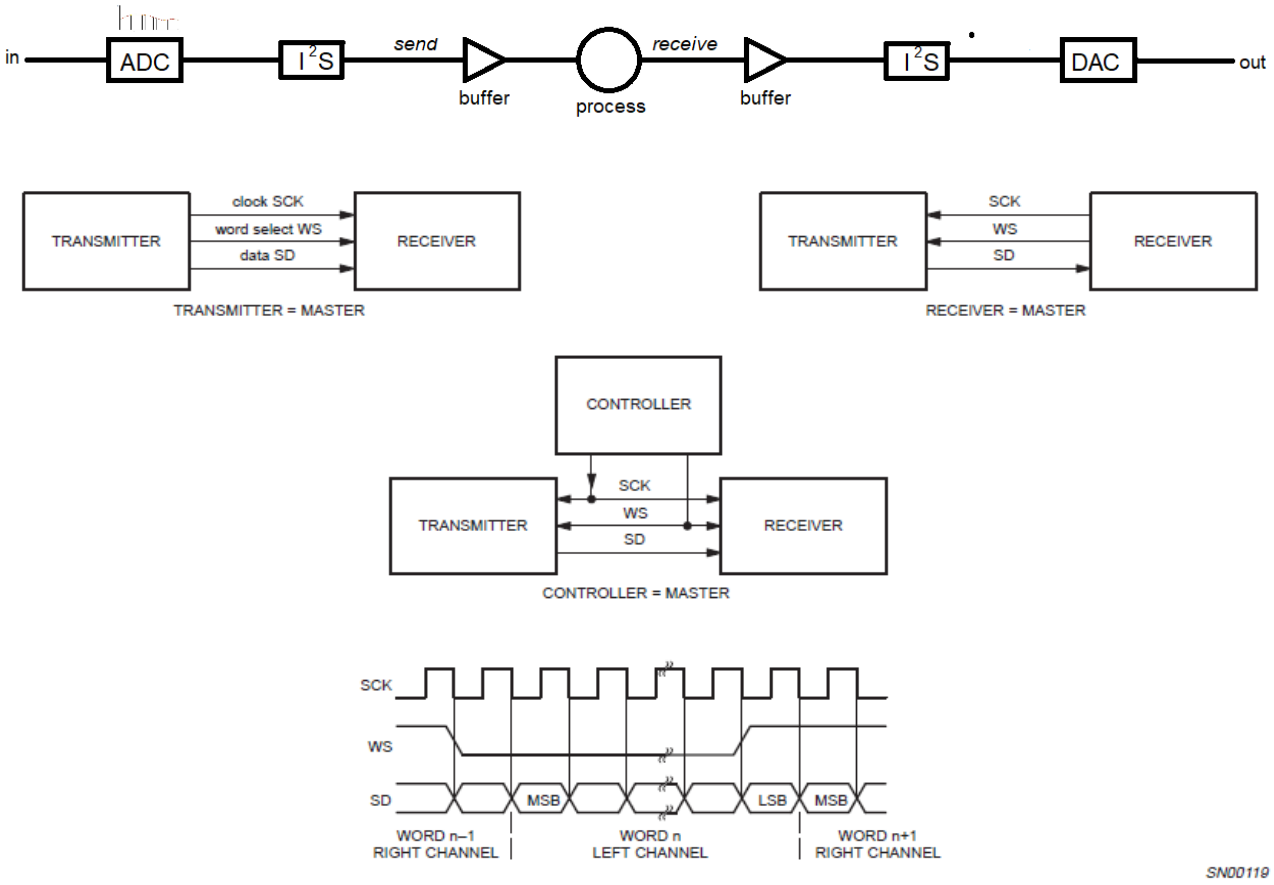


Figura 29 – Diagramma di flusso del trattamento di un segnale in ingresso ad un processore XCORE tramite l'intervento del protocollo I^2S

All'interno dell'ADC, ad ogni ciclo di campionamento, ogni campione viene collocato all'interno del buffer per poi essere processato dall'algoritmo.

I segnali da prendere in considerazione nell'I^2S (Fig.30) sono:

1) La frequenza del processore del PC (o di un oscillatore esterno), indicata come MCLK.
2) Il segnale di bit-clock, detto BCLK,cioè un segnale di onda quadra necessario per sincronizzare la trasmissione di dati da un integrato ad un altro. Ad ogni bit di dati da trasmettere corrisponde un ciclo di BCLK.
3) La frequenza di campionamento del segnale, detta Word Clock (o LRCLK). Ogni mezzo ciclo di LRCLK corrisponde a un canale (es. La prima metà del fronte d'onda corrisponde al canale sinistro, l'altra al canale destro).
4) Il flusso dei dati audio da trasmettere, detto DATA.

I^2S è configurabile a partire dai seguenti parametri:

1) *MCLK_BCLK_RATIO*, che determina il rapporto tra MCLK e BCLK.
2) *MODE*, che definisce il formato dei dati.

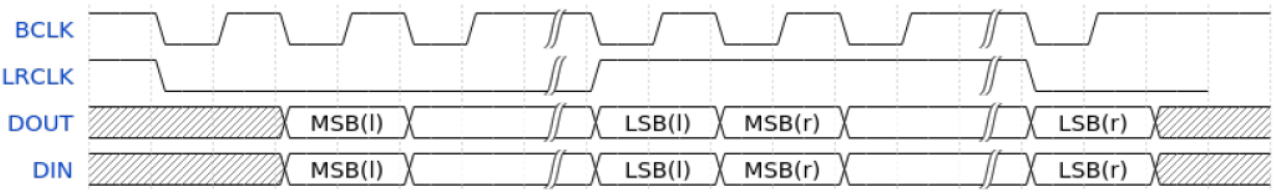


Figura 30 – Segnali coinvolti nell'I^2S

Ad esempio, se MCLK_BCLK_RATIO = 8 e MCLK = 24576 MHz, la frequenza di BCLK risultante sarà pari a 3072 MHz.

Dovendo restituire una quantità di dati in uscita in 64 bit, dividendo BCLK per 64 si ottiene la frequenza di campionamento dei segnali audio da processare, che nel nostro caso sarà pari a 48KHz.

I dati audio possono essere espressi in tre diversi formati:

1) *Left Justified*, quando l'inizio della cresta di un fronte d'onda coincide col bit più significativo del dato.
2) *Right Justified*, quando la fine del fronte d'onda coincide col bit meno significativo.
3) I^2S.

Se il formato dei dati è di tipo I^2S,essi vengono trasferiti a partire da un istante di tempo corrispondente al secondo tratto discendente della forma d'onda di BCLK dopo le transizioni di LRCLK.

In XMOS, un ciclo dura $20 \cdot 10^{-6}$ secondi, di cui una piccola parte è dedicata al'esecuzione delle istruzioni in Assembler contenute all'interno di una struttura dati. Ciascuna di queste istruzioni impiega 10^{-9} secondi per essere eseguita, per cui tante più istruzioni sono contenute all'interno di una funzione quanto più tempo sarà richiesto al programma per eseguirle tutte all'interno di un ciclo.

Questo significa che bisogna utilizzare meno istruzioni possibili per evitare di eccedere la durata dell'intero ciclo, il che comprometterebbe la qualità del suono in uscita.

5.16 Collegare i segnali dell'I^2S ai dispositivi XCORE

È possibile inviare i segnali in formato I^2S a un dispositivo XCORE utilizzando qualsiasi porta di ingresso, a patto che si trovi sullo stesso tile e non si crei alcuna sovrapposizione con i segnali delle altre porte.

Se si considera il dispositivo XMOS come master, il collegamento avviene come descritto in Fig.31:

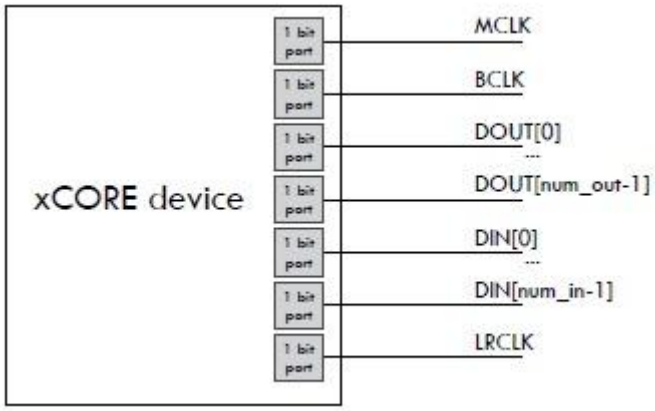
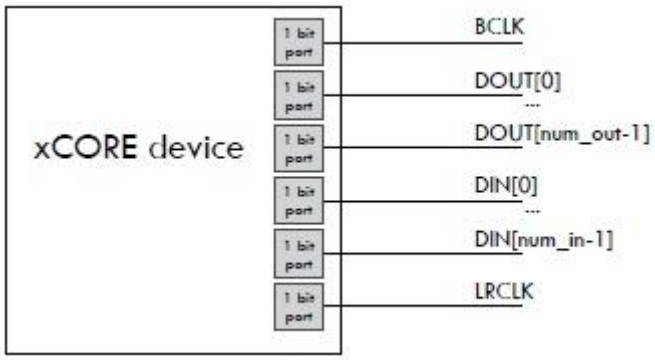


Figura 31 – Connessione segnali I^2S a un dispositivo XCORE in configurazione master

mentre se lo si considera come dispositivo slave, il collegamento sarà quello di Fig.32:



4) *Figura 32 - Connessione segnali I^2S a un dispositivo XCORE in configurazione slave*

5.17 Velocità e prestazioni

- *I^2S Master*: la velocità con cui i dati I^2S fluiscono nel dispositivo XMOS e il numero di cavi necessari al collegamento master dipendono dalla velocità di esecuzione del codice da parte del core logico nel tile considerato.
- *I^2S Slave*: la velocità e il numero di collegamenti dipendono dalla velocità di esecuzione del codice da parte del core,come nel caso del collegamento master, solo che in questo caso le prestazioni dipendono dalla frequenza di bit clock e non da quella del master clock.

5.18 Implementazione del protocollo I^2S in XMOS

Tutte le funzioni del protocollo I^2S vengono introdotte in XMOS includendo la libreria:

```
#include <i2s.h>
```

Le funzioni incluse in tale libreria operano controllando il bus di comunicazione I^2S presente su un core nel dispositivo XCORE implementato.

Quando la libreria riceve un campione o deve inviarne uno,essa effettua delle chiamate (callback) all'applicazione (Fig.33):

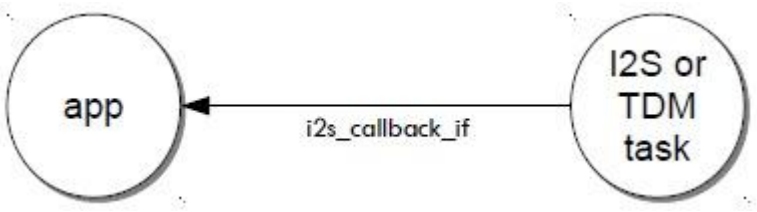


Figura 33 – Chiamate all'applicazione (callbacks) da parte dell'I^2S

Le chiamate vengono eseguite definendo un task i2s_callback_if.

I tasks generati dall'applicazione possono eseguire le istruzioni delle chiamate sullo stesso core implementando un task distribuibile. Esempio:

```
[[distributable]]
void my_application(server i2s_callback_if i2s) {
while (1) {
select {
case i2s.init(i2s_config_t &?i2s_config, tdm_config_t &?tdm_config):
i2s_config.mclk_to_bclk_ratio = 2;
i2c_config.mode = I2S_MODE_LEFT_JUSTIFIED;
...
break;
case i2s.restart_check() -> i2s_restart_t restart:
...
break;
case i2s.receive(size_t index, int32_t sample):
...
break;
case i2s.send(size_t index) -> int32_t sample:
...
break;
}
```

Le chiamate *send* e *receive* passano all'applicazione, come parametro di ingresso, un indice di canale. La chiamata *init* fornisce le strutture per configurare il bus di comunicazione da utilizzare; a questo punto, l'applicazione può configurare i parametri del bus (*MCLK/BCLK RATIO*, *LRCLK*...).

5.19 I^2S: utilizzo in modalità master

Il master task viene istanziato all'interno di uno statement *par*, dunque come un task che viene eseguito in parallelo ad altri tasks. Il collegamento dell'applicazione avviene per mezzo di un'interfaccia i2s_callback_if.

Il seguente codice istanzia un componente master I^2S e lo collega a tale interfaccia:

```
out buffered port:32 p_dout[2] = {XS1_PORT_1D, XS1_PORT_1E};
in buffered port:32 p_din[2] = {XS1_PORT_1I, XS1_PORT_1K};
```

```
port p_mclk = XS1_PORT_1M;
out buffered port:32 p_bclk = XS1_PORT_1A;
out buffered port:32 p_lrclk = XS1_PORT_1C;
clock mclk = XS1_CLKBLK_1;
clock bclk = XS1_CLKBLK_2;
int main(void) {
        i2s_callback_if i_i2s;
        configure_clock_src(mclk, p_mclk);
        start_clock(mclk);
        par {
                i2s_master(i_i2s, p_dout, 2, p_din, 2,p_bclk, p_lrclk, bclk, mclk);
                my_application(i_i2s);
        }
        return 0;
}
```

5.20 I^2S: utilizzo in modalità slave

Lo slave task viene anch'esso istanziato come task da eseguire in parallelo e, come avviene in modalità master, l'applicazione lo collega all'interfaccia di connessione nella seguente maniera:

```
out buffered port:32 p_dout[2] = {XS1_PORT_1D, XS1_PORT_1E};
in buffered port:32 p_din[2] = {XS1_PORT_1I, XS1_PORT_1K};
in port p_bclk = XS1_PORT_1A;
in port p_lrclk = XS1_PORT_1C;
clock bclk = XS1_CLKBLK_1;
int main(void) {
        par {
                i2s_slave(i2s_i, p_dout, 2, p_din, 2,
                p_bclk, p_lrclk, bclk);
                my_application(i_i2s);
        }
        return 0;
}
```

5.21 Numerazione dei canali

L'interfaccia di chiamata numera i canali predisposti alla spedizione e alla ricezione di sample di dati in modo da passare i loro indici come parametro di ingresso delle chiamate *send* e *receive*, rispettivamente.

I canali di numero pari corrispondono ai data sample di sinistra (cioè, ai campioni corrispondenti ai fronti d'onda di numero pari) e i canali di numero dispari a quelli di destra (cioè, ai campioni corrispondenti ai fronti d'onda di numero dispari). Per esempio, in un sistema contenente 4 porte d'ingresso e 4 porte d'uscita, dichiarato come segue:

```
out buffered port:32 p_dout[4] = {XS1_PORT_1A, XS1_PORT_1B, XS1_PORT_1C,
XS1_PORT_1D};
```

in buffered port:32 p_din[4] = {XS1_PORT_1E, XS1_PORT_1F, XS1_PORT_1G, XS1_PORT_1H};

i canali saranno numerati come in Fig.34:

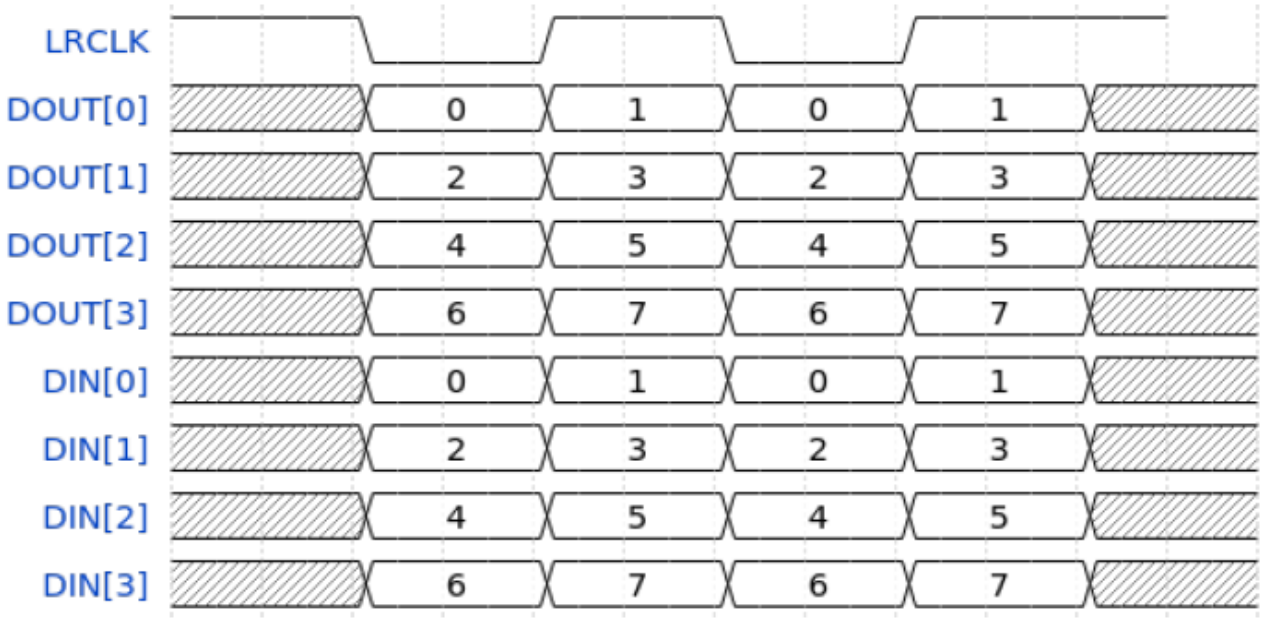


Figura 34 – Numerazione dei canali di ingresso e in uscita nell'I^2S

5.22 Chiamate in sequenza

Le chiamate *send* e *receive* occorrono in un ordine predefinito.

La sequenza consiste in:

- Il contenuto ricevuto dai canali pari.
- Il contenuto inviato dai canali pari.
- Il contenuto ricevuto dai canali dispari.
- Il contenuto inviato dai canali dispari.

Dal momento che le porte del dispositivo XCORE contengono un buffer al loro interno, ci sarà una certa sequenza di chiamate *send* successive alla chiamata *init* e una sequenza finale di chiamate *receive* successive ad una chiamata di *restart* o di *shutdown*.

Initial send:	S0 S2 S1 S3
Frame:	R0 R2 S0 S2 R1 R3 S1 S3
Frame:	R0 R2 S0 S2 R1 R3 S1 S3
...	...
Frame:	R0 R2 S0 S2 R1 R3 S1 S3
Final receive:	R0 R2 R1 R3

Figura 35 – Sequenza di callbacks "send" e "receive"all'interno di un dispositivo XCORE

5.23 Configurazione del clock

Nel protocollo I^2S in modalità master, spetta all'applicazione impostare e inizializzare un clock interno da usare come master clock prima di chiamare il componente.

Ad esempio, il seguente codice configura ed inizializza un clock:

```
configure_clock_src(mclk, p_mclk);
start_clock(mclk);
```

5.24 Creazione di istanze I^2S

i2s_master: la componente master. Questo task applica il protocollo I^2S ai pin del dispositivo, facendo in modo che il flusso di dati che li attraversa si presenti nel suddetto formato. *i2s_master* effettua chiamate attraverso l'interfaccia *i2s_callback_if* per inviare e ricevere dati dall'applicazione che gira all'interno del dispositivo XCORE, che, essendo un dispositivo master, coordina il master clock e il bit clock.

```
void
i2s_master(client i2s_callback_if i2s_i,out buffered port:32 p_dout[num_out],static const
size_t num_out,in buffered port:32 p_din[num_in],static const size_t num_in,out buffered
port:32 p_bclk,out buffered port:32 p_lrclk,
clock bclk,const clock mclk)
```

dove:

- *i2s_i* è la chiamata di interfaccia per connettere l'applicazione.
- *p_dout* è un array di porte dati in uscita.
- *num_out* è il numero di porte dati in uscita.
- *p_din* è un array di porte dati in ingresso.
- *num_in* è il numero di porte dati in ingresso.
- *p_bclk* è la porta d'uscita del bit clock.
- *p_lrclk* è la porta d'uscita del word clock.
- *bclk* è il bit clock.
- *mclk* è il master clock.

i2s_slave: la componente slave. Questo task effettua chiamate all'applicazione per ricevere dati attraverso l'interfaccia direttamente dal componente in cui gira l'applicazione. Il componente in questione, questa volta, è un componente slave, per cui il word clock e il master clock sono definiti e coordinati esternamente.

```
void
i2s_slave(client i2s_callback_if i2s_i,out buffered port:32 p_dout[num_out],static const
size_t num_out,in buffered port:32 p_din[num_in],static const size_t num_in,in port p_bclk,
in buffered port:32 p_lrclk,clock bclk)
```

i2s_callback_if: interfaccia per mezzo della quale vengono chiamati gli eventi che si possono verificare durante l'esecuzione di un task I^2S.

Funzioni coinvolte:

- *init*: inizializza le chiamate agli eventi. Viene effettuata al primo avvio, oppure dopo una *restart*. La sintassi è la seguente:

  ```
  void init(i2s_config_t & ?i2s_config,tdm_config_t & ?tdm_config)
  ```

  dove *i2s_config* viene considerata solo se il componente XCORE effettua il controllo diretto del bus i2s, mentre *tdm_config* se il componente controlla il bus TDM.

- *restart_check*: utilizzata per verificare se sussistono le condizioni per il riavvio dell'applicazione. Sintassi:

  ```
  i2s_restart_t restart_check()
  ```

  Il valore di ritorno sarà o *I2S_NO_RESTART*, o *I2S_RESTART*, o *I2S_SHUTDOWN*.

- *receive*: utilizzata per ricevere un campione dati in ingresso, quando è letto dal componente I^2S. Sintassi:

  ```
  void receive(size_t index, int32_t sample)
  ```

  dove *index* esprime l'indice del campione dati nel frame, mentre *sample* è il campione dati stesso, espresso come variabile a 32 bit.

- *send*: richiede l'invio di un dato in uscita: ciò avverrà quando il componente I^2S richiederà un nuovo sample. Sintassi:

  ```
  int32_t send(size_t index)
  ```

  dove *index* è l'indice del campione dati nel frame richiesto dal componente XCORE. Il campione dati sarà ritornato in uscita come una variabile a 32 bit.

CAPITOLO 6

Simulazione in MATLAB

Il lavoro eseguito in MATLAB consiste fondamentalmente in un file di progetto .m, di nome *ambience2.m*, e di quattro funzioni: *EarlyReflectionsRight.m*, *EarlyReflectionsLeft.m*, *CombFilter.m* e *AllpassFilter.m*.

Le righe di codice all'interno di ogni funzione descrivono le operazioni che ciascun building block deve eseguire, mentre quelle definite all'interno del progetto descrivono il funzionamento dell'intero algoritmo di riverbero, in cui l'esecuzione dei singoli building blocks viene innescata dalle chiamate alle relative funzioni.

6.1 *EarlyReflectionsRight.m*

Questa funzione riguarda la catena di 12 filtri che producono le prime 12 riflessioni del segnale in ingresso percepite al canale destro,ed è definita come segue:

```
function [DelayLineOut,stEarlyRight,OutEarlyRight,Pointer_rd,Pointer_wr] =
EarlyReflectionsRight(InEarlyRight,Pointer_Current_rd,Pointer_Current_wr,EarlyRi
ghtDelay,StartingDelay,ER_G1,EarlyRightGain,ER_G3,DelayLine,state,index)
```

La funzione, dunque, riceve come parametri in ingresso:

- Il segnale in ingresso al building block, *InEarlyRight*, che in questo caso è il segnale in uscita dalla prima biquadratica.
- La posizione corrente del puntatore in fase di lettura, *Pointer_Current_rd*.
- La posizione corrente del puntatore in fase di scrittura, *Pointer_Current_wr*.
- Il tempo di ritardo del generico filtro, *EarlyRightDelay*.
- La posizione di partenza del puntatore sulla linea di ritardo in corrispondenza di dove incomincia il FIR considerato, *StartingDelay*.
- Il valore del guadagno del FIR corrente, *EarlyRightGain*.
- La lunghezza della linea di ritardo (che assume lo stesso valore per tutti i building blocks).
- Il valore della variabile di stato (facente riferimento al blocco di f.d.t. z^{-1} nello schema a blocchi), *state*.
- L'indice del vettore *state*, *index*, che assume valori da 1 a 12 per indicare qual è il FIR corrente.

I valori ER_G1 ed ER_G3, che peraltro compaiono anche nella funzione *CombFilter.m*, sono lasciati uguali a zero. Il motivo è che il comportamento delle prime riflessioni è assimilabile a quello del filtro comb, con la sola assenza delle catene di feedback e di dump.

In uscita,la funzione produce le seguenti variabili:

- *DelayLineOut*, la posizione corrente del vettore *DelayLine*, che deve essere aggiornata ad ogni chiamata di funzione.
- *stEarlyRight*, la posizione corrente del vettore *state*.
- *OutEarlyRight*, il segnale in uscita dalla catena dei FIR.
- *Pointer_rd*, la posizione corrente del puntatore in fase di lettura.
- *Pointer_wr*, la posizione corrente del puntatore in fase di scrittura.

Inizialmente, la funzione definisce diverse variabili che corrispondono ai nodi interni dello schema a blocchi (cioè le varie forme assunte dal segnale in ingresso nei vari punti dello schema).

```
d=DelayLine(Pointer_Current_rd);
a=ER_G1*d;
b=ER_G3*state(index);
c=a+b;
state(index)=c;
```

La variabile d indica il segnale in uscita dalla linea di ritardo (che nello schema corrisponde al segnale in ingresso stesso ritardato di una certa quantità di tempo): il suo valore corrisponde al valore dell'elemento del vettore *DelayLine* nella posizione corrispondente al valore di *Pointer_Current_rd*. In altre parole, si considera l'intera linea di ritardo (lunga 44000 campioni, nel nostro caso) e si prende il suo valore in corrispondenza al campione #(valore di *Pointer_Current_rd*): questo sarà il valore assunto da d, cioè il segnale in uscita dal blocco ritardante del FIR, che nella fattispecie corrisponde al secondo tra tutti i sample che costituiscono il ritardo del FIR (che è, dunque, il primo valore ad essere letto alla prima iterazione).

Seguono le istruzioni per la definizione del segnale nella catena di feedback, `a=ER_G1*d;` , e di quello nella catena di dump, `b=ER_G3*state(index);` , che in questo caso sono entrambi nulli per il motivo spiegato in precedenza. Questi due segnali andranno, poi, sommati e il risultato della loro somma andrà immagazzinato nella variabile c, che corrisponderà anche al valore corrente della variabile di stato relativa al FIR corrente.

```
if Pointer_Current_rd==EarlyRightDelay
     Pointer_Current_rd=StartingDelay+1;
     else Pointer_Current_rd=Pointer_Current_rd+1;
    end
```

In queste righe viene descritto il comportamento del puntatore in lettura ad ogni iterazione. Se la posizione corrente del puntatore in fase di lettura corrisponde all'ultimo campione del ritardo del FIR corrente (*EarlyRightDelay*), tale puntatore viene saturato e riportato al primo campione del ritardo, che sarà dunque la posizione corrente del puntatore all'iterazione successiva.

Questo si fa imponendo il valore di *Pointer_Current_rd* pari a *StartingDelay+1*; la somma a 1 è necessaria in quanto il ritardo del primo FIR inizia a *StartingDelay*=0, ma essendo *Pointer_rd* un vettore, il valore minimo di ogni elemento di un vettore definito in MATLAB deve essere pari a 1, per cui come risultato si avrà *Pointer_Current_rd = 0 + 1 = 1*.

Se, invece, il puntatore in lettura si trova in un altro campione, esso si sposta semplicemente di un campione in avanti, il che equivale ad aumentare di un'unità il valore *Pointer_Current_rd*.

Ogni volta che si aggiorna la posizione corrente del puntatore in lettura, si aggiorna anche il valore di *DelayLine* nella posizione corrispondente al valore del puntatore in fase di scrittura:

```
DelayLine(Pointer_Current_wr)=InEarlyRight+c;
```

e questo valore corrisponde al segnale in ingresso al blocco ritardante del FIR (dato dalla somma tra il segnale in ingresso al building block e il valore della variabile c, che nel caso delle prime riflessioni è sempre pari a zero, quindi di fatto si considera solo il segnale in ingresso).

Una volta fatto questo aggiornamento, si ripete la stessa procedura vista per *Pointer_Current_rd* applicata alla posizione corrente del puntatore in fase di scrittura:

```
  if Pointer_Current_wr==EarlyRightDelay
    Pointer_Current_wr=StartingDelay+1;
else Pointer_Current_wr=Pointer_Current_wr+1;
end
```

In generale,ogni campione letto in una iterazione verrà successivamente scritto nell'iterazione successiva.

Infine, abbiamo la definizione delle variabili di uscita dalla funzione:

```
DelayLineOut = DelayLine;
stEarlyRight=state;
OutEarlyRight=EarlyRightGain*d;
Pointer_rd=Pointer_Current_rd;
Pointer_wr=Pointer_Current_wr;
```

Si aggiorna,dunque, *DelayLine* con i suoi nuovi elementi (la versione aggiornata viene chiamata,in questo caso, *DelayLineOut*), la variabile di stato, il puntatore nelle sue nuove posizioni in lettura e in scrittura e si definisce anche il segnale in uscita dal building block, *OutEarlyRight*, il cui valore sarà dato dal prodotto del segnale d in uscita dal blocco ritardante per il guadagno in uscita *EarlyRightGain* (che corrisponde al contributo di guadagno relativo al FIR corrente).

6.2 *EarlyReflectionsLeft.m*

```
function [DelayLineOut,stEarlyLeft,OutEarlyLeft,Pointer_rd,Pointer_wr] =
EarlyReflectionsLeft(InEarlyLeft,Pointer_Current_rd,Pointer_Current_wr,EarlyLeft
Delay,StartingDelay,EL_G1,EarlyLeftGain,EL_G3,DelayLine,state,index)
```

Questa funzione fa esattamente le stesse cose di cui si è discusso in merito alla funzione *EarlyReflectionsRight.m*, solamente che in questo caso cambiano i tempi di ritardo e i guadagni di ogni FIR, che produrranno le prime riflessioni del segnale in ingresso al canale sinistro.

Inoltre, il primo campione del ritardo relativo al primo FIR di questa seconda catena si troverà nella posizione successiva a quella del'ultimo campione del ritardo dell'ultimo FIR di

EarlyReflectionsRight.m; questo significa che il valore di *StartingDelay* riferito al primo FIR di *EarlyReflectionsLeft.m* non sarà pari a 0.

6.3 *CombFilter.m*

```
function [DelayLineOut,stComb,OutComb,Pointer_rd,Pointer_wr] =
CombFilter(InComb,CombDelay,StartingDelay,G1,G2,G3,Pointer_Current_rd,Pointer_Cu
rrent_wr,DelayLine,state,index)
```

Questa funzione riceve in ingresso gli stessi parametri di *EarlyReflections*, con nomi ovviamente diversi e con i guadagni di feedback e di dump che questa volta non sono nulli. Di conseguenza, il calcolo dei segnali di feedback e di dump descritto dalle seguenti righe di codice:

```
a=G1*d;
b=G3*state(index);
```

potrebbe dar luogo a risultati non nulli.

Come nelle funzioni delle prime riflessioni, anche qui si calcolano il segnale in uscita dal blocco ritardante,

```
d=DelayLine(Pointer_Current_rd);
```

e la variabile di stato,

```
state(index)=c;
```

Si aggiorna la posizione corrente del puntatore in fase di lettura nel consueto modo:

```
 if Pointer_Current_rd==CombDelay
Pointer_Current_rd=StartingDelay+1;
else Pointer_Current_rd=Pointer_Current_rd+1;
end
```

il valore di *DelayLine* che verrà scritto dal puntatore:

```
DelayLine(Pointer_Current_wr)=InComb+c;
```

e la posizione corrente del puntatore in fase di scrittura:

```
 if Pointer_Current_wr==CombDelay
   Pointer_Current_wr=StartingDelay+1;
else Pointer_Current_wr=Pointer_Current_wr+1;
end
```

Infine, la funzione calcola i valori delle variabili di uscita:

```
DelayLineOut = DelayLine;
stComb=state;
OutComb=G2*d;
Pointer_rd=Pointer_Current_rd;
Pointer_wr=Pointer_Current_wr;
```

6.4 *AllpassFilter.m*

```
function [DelayLineOut,OutAllpass,Pointer_rd,Pointer_wr] =
AllpassFilter(InAllpass,AllpassDelay,StartingDelay,AllpassGain,Pointer_Current_r
d,Pointer_Current_wr,DelayLine)
```

A differenza delle funzioni descritte sinora, in *AllpassFilter.m* non è presente la variabile di stato in quanto nello schema a blocchi relativo al filtro allpass non vi è alcun blocco z^-1.

In questa funzione, oltre a definire una variabile d per l'uscita dal blocco ritardante:

```
d=DelayLine(Pointer_Current_rd);
```

si definiscono altre variabili per i segnali intermedi. La variabile a, ad esempio:

```
a=AllpassGain*d;
```

definisce il segnale in uscita dal blocco ritardante che viene passato alla catena di feedback e moltiplicato per il relative guadagno (che, in questo caso, è espresso come *AllpassGain*). La variabile b, invece:

```
b=InAllpass+a;
```

definisce il segnale in ingresso al blocco ritardante,dato dalla somma dal segnale in ingresso all'intero building block col valore di b. Quest'ultimo segnale, a sua volta, nello schema viene passato ad una seconda catena di feedback, venendo moltiplicato per un guadagno uguale in modulo a quello dell'altra catena di feedback, ma opposto di segno:

```
c=-(AllpassGain)*b;
```

Definite queste variabili, si aggiorna la posizione corrente del puntatore in lettura:

```
if Pointer_Current_rd==AllpassDelay;
      Pointer_Current_rd=StartingDelay+1;
      else Pointer_Current_rd=Pointer_Current_rd+1;
end
```

si scrive sulla linea di ritardo il valore di b:

```
DelayLine(Pointer_Current_wr)=b;
```

e si aggiorna la posizione corrente del puntatore in scrittura:

```
if Pointer_Current_wr==AllpassDelay;
      Pointer_Current_wr=StartingDelay+1;
  else Pointer_Current_wr=Pointer_Current_wr+1;
    end
```

In conclusione, la funzione restituisce in uscita i valori delle variabili in uscita:

```
DelayLineOut = DelayLine;
OutAllpass=c+d;
Pointer_rd=Pointer_Current_rd;
Pointer_wr=Pointer_Current_wr;
```

Si noti, per quanto riguarda la variabile in uscita, che il suo valore corrisponde alla somma delle variabili c e d: questo significa che il segnale in uscita dal filtro allpass corrisponderà alla somma tra il segnale uscente dal blocco ritardante (cioè d) e il segnale b amplificato di un guadagno negative (cioè c).

6.5 *ambience2.m*

Discuteremo, ora, di come è stato realizzato in MATLAB il riverbero ambience per la trattazione del nostro esperimento.

Per prima cosa, si definisce come segnale in ingresso un impulso unitario, cioè un segnale x(t) avente ampiezza unitaria in t=0 e ampiezza nulla per tutti gli altri valori di t >0. La risposta impulsiva del segnale in ingresso è rappresentata in Fig.37.

In MATLAB, questo segnale è stato definito come un vettore x di 100000 elementi:

```
x=zeros(100000,1);
```

il cui valore in corrispondenza del primo elemento è 1:

```
x(1)=1;
```

L'andamento nel tempo di questo segnale viene, poi, rappresentato tramite una funzione *plot*, definendo, in primo luogo, un vettore t di 100000 elementi:

```
t = (1:1:length(x));
```

che fungerà da ascisse per il nostro grafico.
In pratica, il primo elemento di t varrà 1, il secondo 2, etc. fino all'ultimo che varrà 100000.
A questo punto, è possibile tracciare il grafico di x(t) (Fig.36), passando alla funzione *plot* il vettore t come ascissa e il vettore x come ordinata:

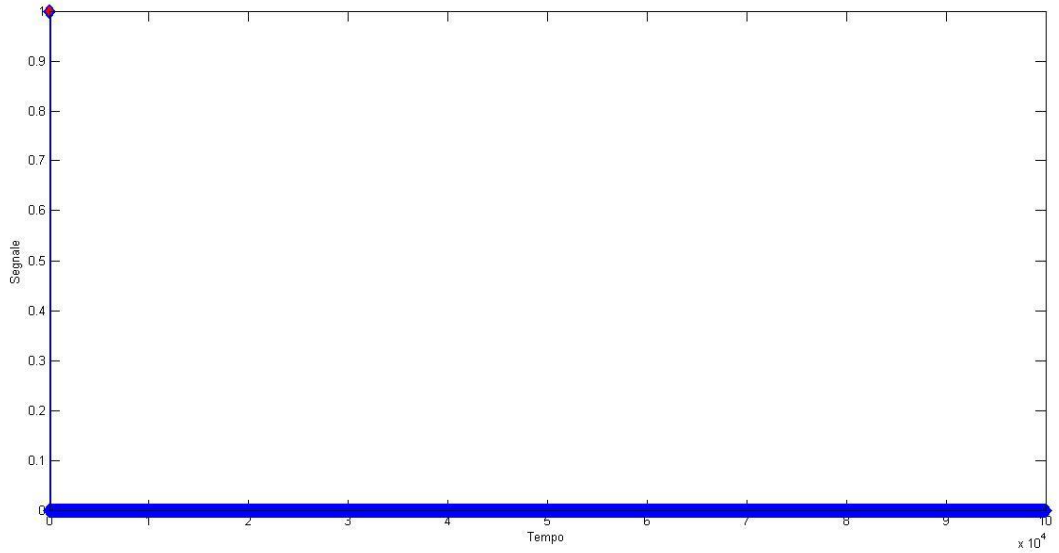


Figura 36 – Impulso unitario in ingresso

La fase successiva consiste nella dichiarazione delle variabili che andranno, poi, passate in ingresso alle funzioni di cui si è già parlato.

Si definisce la linea di ritardo, *DelayLine*, di dimensione *Dline = 44000*; quindi, la linea di ritardo sarà un vettore colonna contenente 44000 campioni, e ogni frazione di tale linea di ritardo sarà dedicata ad un preciso building block, specificando la dimensione di tale frazione in base al contributo di ritardo offerto da quel building block.
A seguire, abbiamo i contributi di ritardo di ogni building block.

È bene tener presente che ogni variabile relativa al contributo di ritardo di un building block è stata definita in termini di vettore perché nello schema è presente una certa quantità di building blocks appartenenti ad una diversa categoria (comb, allpass...), per cui il numero di elementi di ogni vettore sarà pari al numero di building blocks appartenenti alla categoria considerata presenti nello schema e il cui valore sarà il contributo di ritardo offerto dal rispettivo building block.
Ad esempio, avendo 12 filtri che compongono le Early Reflections Right, si definisce un vettore a 12 dimensioni in cui ogni elemento è riferito ad un generico filtro; quindi, in questo caso, il valore del primo elemento di *EarlyRightDelay* sarà il tempo di ritardo relativo al primo filtro delle Early Reflections Right,che vale 111.
Per quanto riguarda *CombDelayMain*, i valori dei primi 5 elementi sono stati deliberatamente impostati a 0 perché, essendo il tempo di ritardo dei primi 5 comb appartenenti alla categoria Main pari a 1, il loro contributo all'interno dello schema del riverbero è pressoché ininfluente e si prendono in considerazione,invece,i ritardi degli ultimi due Main Comb.

Queste righe, invece, definiscono i guadagni:

- *ER_G1*, *EarlyRightGain*, *ER_G3* indicano i guadagni da passare alla funzione *EarlyReflectionsRight.m* (*ER_G1* e *ER_G3* sono entrambi impostati a zero; l'unico motivo per cui sono stati contemplati è per sfruttare la stessa struttura della funzione *CombFilter.m* dove, invece, non sono nulli, dal momento che i FIR delle prime riflessioni si comportano come un filtro comb senza catene di feedback e di dump).

- *EL_G1*, *EarlyLeftGain*, *EL_G3* indicano i guadagni da passare alla funzione *EarlyReflectionsLeft.m*.
- *MainG1*, *MainG2*, *MainG3* indicano i guadagni dei filtri comb appartenenti alla categoria Main da passare alla funzione *CombFilter.m*.
- *RightG1*, *RightG2*, *RightG3* indicano i guadagni dei filtri comb appartenenti alla categoria Right da passare alla funzione *CombFilter.m*.
- *LeftG1*, *LeftG2*, *LeftG3* indicano i guadagni dei filtri comb appartenenti alla categoria Left da passare alla funzione *CombFilter.m*.
- *AllpassGainRight* indica i guadagni dei filtri allpass appartenenti alla categoria Right da passare alla funzione *AllpassFilter.m*.
- *AllpassGainLeft* indica i guadagni dei filtri allpass appartenenti alla categoria Left da passare alla funzione *AllpassFilter.m*.
- *EarlyRightToRight* indica il guadagno per il quale deve essere moltiplicato il segnale in uscita dalle prime riflessioni Right, per poi essere passato al canale destro.
- *EarlyLeftToRight* indica il guadagno per il quale deve essere moltiplicato il segnale in uscita dalle prime riflessioni Left, per poi essere passato al canale sinistro.
- *AllpassRightToRight* indica il guadagno per il quale deve essere moltiplicato il segnale in uscita dal secondo filtro allpass Right, per poi essere passato al canale destro.
- *AllpassLeftToRight* indica il guadagno per il quale deve essere moltiplicato il segnale in uscita dal secondo filtro allpass Left, per poi essere passato al canale destro.
- *AllpassRightToLeft* indica il guadagno per il quale deve essere moltiplicato il segnale in uscita dal secondo filtro allpass Right, per poi essere passato al canale sinistro.
- *AllpassLeftToLeft* indica il guadagno per il quale deve essere moltiplicato il segnale in uscita dal secondo filtro allpass Left, per poi essere passato al canale sinistro.

Come per i ritardi, anche nel caso dei guadagni sono state definiti dei vettori il cui numero di elementi corrisponde al numero di building blocks appartenenti alla categoria di riferimento.

Le variabili di stato da passare,rispettivamente,alle funzioni *CombFilter.m* e *EarlyReflections.m* (sia Right che Left) sono definite dai vettori *state* e *stateEarly*, rispettivamente.

I prossimi vettori, espressi nella forma *Pointer_rd_XXX* e *Pointer_wr_XXX* (dove *XXX* indica il building block specifico) riguardano le posizioni correnti dei puntatori, sia in fase di lettura che di scrittura, sulle porzioni di linea di ritardo appartenenti ai diversi building blocks.

Nel caso delle prime riflessioni (sia Right che Left), si definiscono vettori a 12 dimensioni, in quanto si hanno in tutto 12 filtri per i quali ciascun ritardo occupa una diversa posizione della linea di ritardo *DelayLine*. Nel caso dei filtri comb, invece, le dimensioni sono 7 perché si hanno 7 filtri comb Main; si è voluto mantenere la stessa dimensione anche per i Right e i Left comb, in modo da poter passare lo stesso vettore di stato, *state* (anch'esso a 7 dimensioni), per tutti i comb, altrimenti si sarebbe dovuto definire un'altra funzione, diversa da *CombFilter.m*, che si occupasse dei comb Right e Left.

Infine, per gli allpass la dimensione dei vettori *Pointer* è 2.

Si definisce, poi, la posizione di partenza delle porzioni di linea di ritardo per ciascun building block, i cui valori corrispondono alla posizione all'interno del vettore *DelayLine*. Tale posizione di partenza è espressa dai vettori *startingDelayXXX* (dove *XXX* indica sempre lo specifico building block).

Si definiscono, inoltre, i vettori *endingDelayXXX*, che indicano la posizione terminale delle porzioni di linea di ritardo per ciascun building block.

Le righe di codice successive provvedono a impostare la posizione di partenza del puntatore in lettura e in scrittura per ciascun building block, secondo la seguente sintassi:

Pointer_rd_XXX(i) = startingDelayXXX(i) + 1;

Pointer_wr_XXX(i) = startingDelayXXX(i) + 2;

Si noti che, in *Pointer_rd_XXX(i)* si è sommato a 1 il valore di *startingDelayXXX(i)*: se, infatti, questo valore fosse pari a 0 (cioè, pari alla posizione di partenza della linea di ritardo), il corrispondente valore di *Pointer_rd_XXX(i)* risulterà pari a 1, ovvero al minimo valore che può assumere un indice di un vettore.

In *Pointer_wr_XXX(i)*, invece, il valore di *startingDelayXXX(i)* viene sommato a 2 in quanto, in un buffer circolare, la posizione di scrittura deve essere immediatamente successiva a quella di lettura.

Infine, si definiscono i segnali di uscita da ciascun building block. In particolare:

- *OutEarlyReflections* indica il segnale in uscita dalle prime 24 riflessioni del segnale in ingresso, inteso come somma tra i contributi in uscita *OutEarlyRight* e *OutEarlyLeft.*
- *OutRComb* indica il segnale in uscita dal parallelo di comb nel canale destro, espresso come somma tra l'uscita dei main comb, *OutCombMain,* e l'uscita dei right comb, *OutRightComb*.
- *OutLComb* indica il segnale in uscita dal parallelo di comb nel canale sinistro, espresso come somma tra l'uscita dei main com, *OutCombMain,* e l'uscita dei left comb, *OutLeftComb*.
- *Output* è il segnale in uscita dall'intero circuito del riverbero.

Una volta dichiarate le variabili, inizia la compilazione vera e propria: si istanzia un ciclo for in cui si preleva un campione del segnale di ingresso alla volta e lo si fa processare da ogni building block lungo il diagramma di flusso del riverbero:

```
for n=1:length(x){
```

Per n=1, dunque, viene passato il primo campione di x(t), pari a 1, all'ingresso del blocco di filtri per le prime riflessioni sul canale destro, calcolando campione per campione il corrispondente segnale in uscita da ogni filtro:

```
      [DelayLine,stateEarly,OutEarlyRighti(n),Pointer_rd_EarlyRight(i),Pointer_w
r_EarlyRight(i)] =
EarlyReflectionsRight(x(n),Pointer_rd_EarlyRight(i),Pointer_wr_EarlyRight(i),end
ingDelayEarlyRight(i),startingDelayEarlyRight(i),ER_G1(i),EarlyRightGain(i),ER_G
3(i),DelayLine,stateEarly,i);
```

dove *i* si riferisce ad un particolare filtro.
Viene, dunque, chiamata *i = 12* volte la funzione *EarlyReflectionsRight*,che riceve i seguenti valori di ingresso:

- x(n)
- *Pointer_rd_EarlyRight(i)*, riferito all'i-esimo FIR
- *Pointer_we_EarlyRight(i)*
- *endingDelayEarlyRight(i)*
- *startingDelayEarlyRight(i)*
- *ER_G1(i)*
- *EarlyRightGain(i)*
- *ER_G3(i)*
- *DelayLine*
- *stateEarly(i)*

e produce le seguenti uscite:

- *DelayLine* (lo stesso vettore passato in ingresso, ma aggiornato al termine dell'iterazione precedente)
- *stateEarly*
- *OutEarlyRight_i(n),* cioè l'n-esimo campione del segnale in uscita dall'i-esimo FIR
- *Pointer_rd_EarlyRight(i)*
- *Pointer_wr_EarlyRight(i)*

Ad ogni iterazione, dunque, la funzione costruisce il vettore di uscita *OutEarlyRight_i* aggiungendo il valore dell'elemento corrispondente all'n considerato.
I valori di *endingDelayEarlyRight* e di *startingDelayEarlyRight* sono scelti in maniera tale che ogni qualvolta il puntatore raggiunge l'ultimo campione della porzione di linea di ritardo associata a tale filtro (campione #111 per il primo filtro, (#111 + #237) per il secondo e così via) esso viene saturato e riportato alla posizione iniziale di tale porzione (campione #1 per il primo filtro, (#1 + #111) per il secondo...).
Una volta ottenuti tutti e 12 i vettori *OutEarlyRight_i* essi vengono sommati in un unico vettore *OutEarlyRight*, che esprime l'uscita complessiva del blocco dei 12 filtri sul canale destro:

```
OutEarlyRight(n) = OutEarlyRight1(n) + OutEarlyRight2(n) + OutEarlyRight3(n) +
OutEarlyRight4(n) + OutEarlyRight5(n) + OutEarlyRight6(n) + OutEarlyRight7(n) +
OutEarlyRight8(n) + OutEarlyRight9(n) + OutEarlyRight10(n) + OutEarlyRight11(n)
+ OutEarlyRight12(n);
```

Come risultato,si otterrà una sequenza di impulsi centrati in corrispondenza del tempo di ritardo di ognuno dei 12 filtri. Tale sequenza è rappresentata in Fig.37:

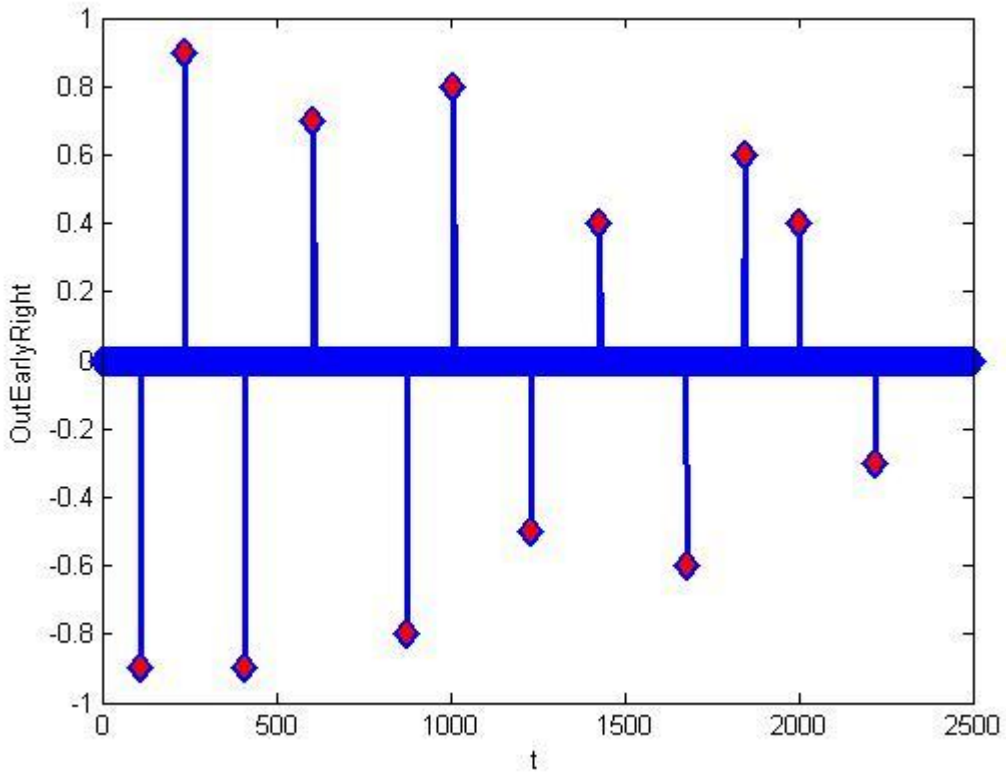


Figura 37 – Risposta impulsiva in uscita dalle EarlyReflectionsRight

Lo stesso procedimento verrà seguito per ottenere *OutEarlyLeft*, cioè l'uscita complessiva del blocco dei 12 FIR sul canale sinistro:

Qui, per quanto riguarda i valori di *endingDelayEarlyLeft* e di *startingDelayEarlyLeft*, essi corrispondono al sample immediatamente successivo a quello in cui si trovano *endingDelayEarlyRight* e *startingDelayEarlyRight* relativamente all'ultimo FIR delle prime riflessioni al canale destro.
In Fig.38 è riportato il grafico delle prime 12 riflessioni al canale sinistro:

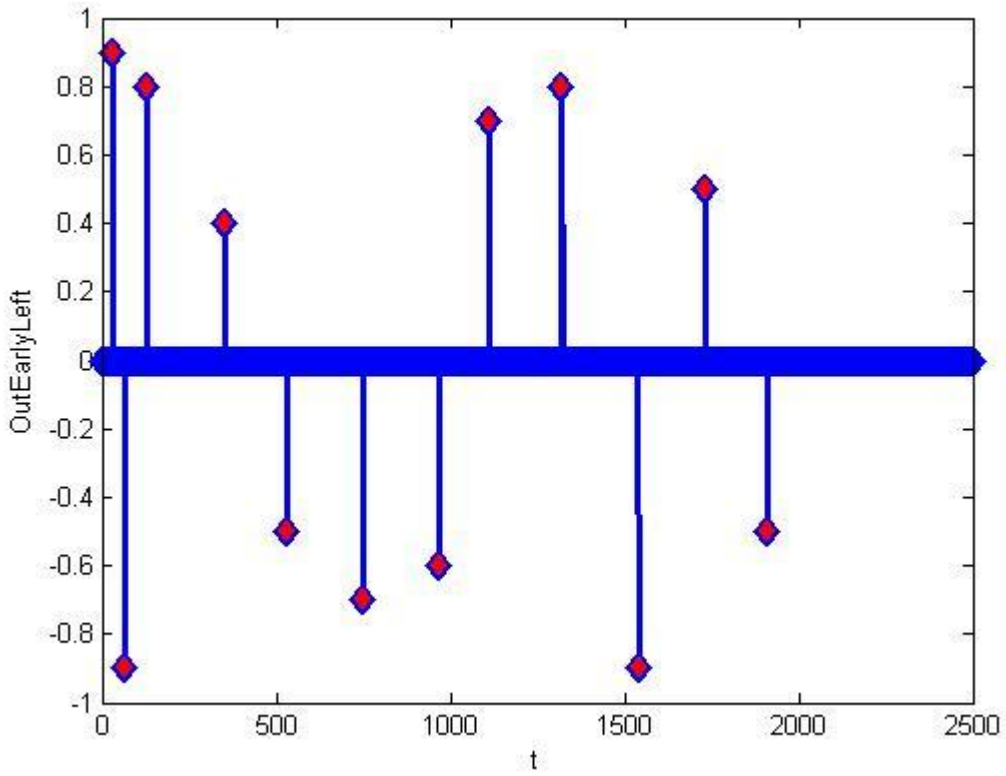


Figura 38 – Risposta impulsiva in uscita dalle EarlyReflectionsLeft

Si moltiplica, poi, per 0.7 (il valore del volume di uscita delle prime riflessioni) sia *OutEarlyRight* sia *OutEarlyLeft*, e si sommano tra loro questi due valori per ottenere l'uscita complessiva del building block relativa alle prime riflessioni, come rappresentato in Fig.39:

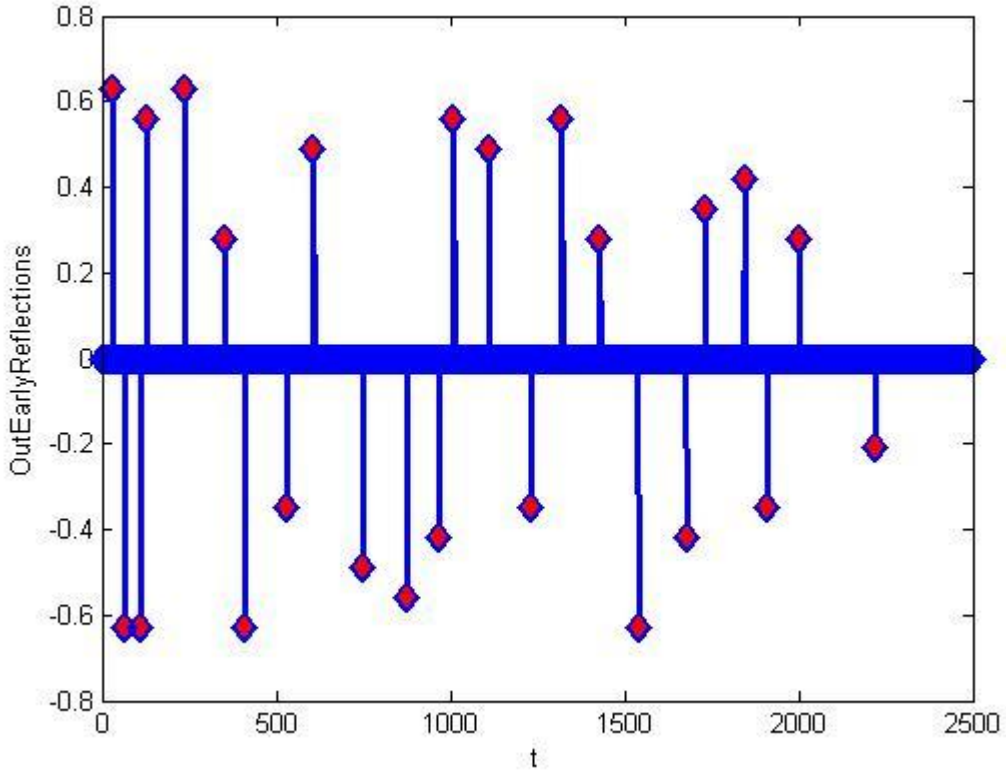


Figura 39 – Risposta impulsiva in uscita dall'intero blocco di Early Reflections

Il segnale di uscita *OutEarlyReflections* appena ottenuto andrà, poi, applicato in ingresso al parallelo dei comb.
Il lavoro di un filtro comb, in questo caso, consiste nel generare diverse repliche del segnale in ingresso, che si susseguono per un periodo pari al tempo di ritardo del comb stesso, e ogni replica presenterà un'ampiezza minore e un andamento esponenzialmente decrescente più pronunciato rispetto alla replica precedente. Come si è visto per le Early Reflections, si definiscono le variabili di uscita anche della funzione comb:

```
[DelayLine,state,OutCombXi(n),Pointer_rd_XComb(i),Pointer_wr_XComb(i)] =
CombFilter(OutEarlyReflections(n),endingDelayXComb(i),startingDelayXComb(i),XG1(
i),XG2(i),XG3(i),Pointer_rd_XComb(i),Pointer_wr_XComb(i),DelayLine,state,i);
```

Anche in questo caso, *i* corrisponde ad uno specifico comb, mentre *X* sta a indicare il suo gruppo di appartenenza (cioè, Main, Right o Left).

Sono state calcolate solo le uscite relative ai comb Main #6 e #7, dato che i primi 5 comb Main offrono un contributo di uscita molto piccolo che può essere, dunque, trascurato.
Le uscite dai due comb Main vengono sommate in un'unica variabile di uscita, detta *OutCombMain*, la cui risposta in uscita è riportata in Fig.40:

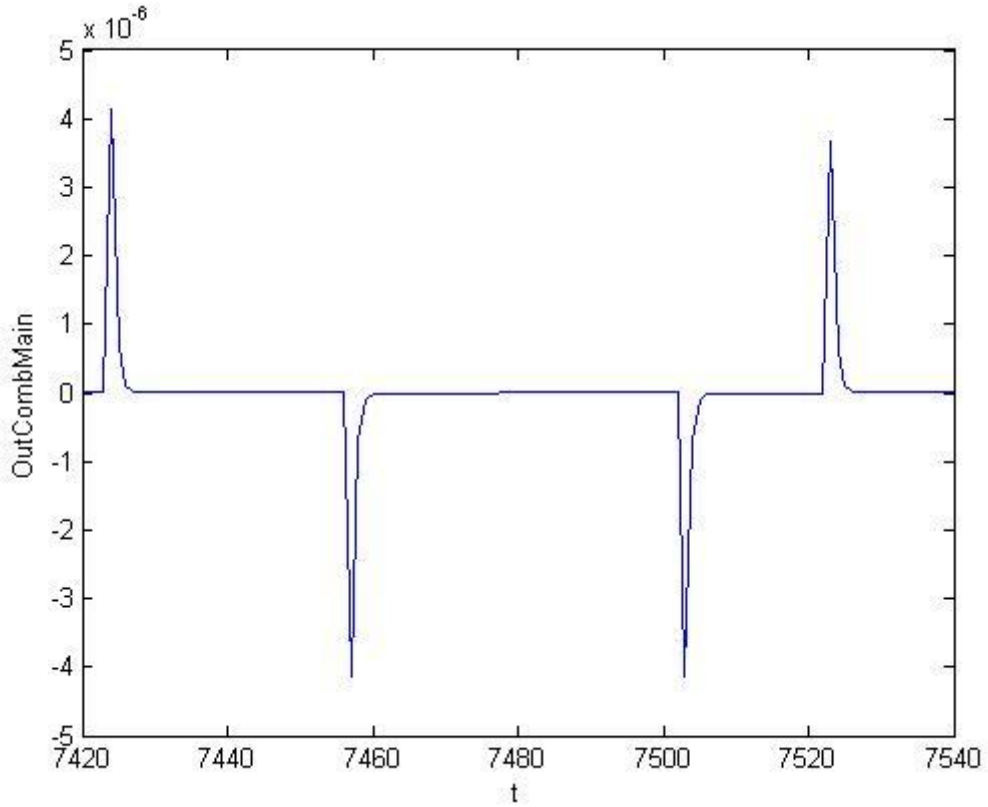


Figura 40 – Risposta impulsiva in uscita dal blocco di filtri comb Main (primi quattro impulsi)

L'effetto delle catene di feedback e di dump,come si evince dalla figura precedente, è quello di conferire un andamento curvilineo al tratto discendente degli impulsi (o ascendente, se gli impulsi hanno ampiezza negativa), che dal punto di vista acustico si traduce in un prolungamento del suono nel tempo (la cui quantità dipende dai guadagni di feedback e di dump); ad ogni iterazione, questo

effetto risulta sempre più accentuato, oltre che accompagnato ad un'attenuazione in ampiezza degli impulsi.
La Fig.41 riporta un'inquadratura ravvicinata delle repliche di tre degli impulsi in uscita dal parallelo dei Main Comb:

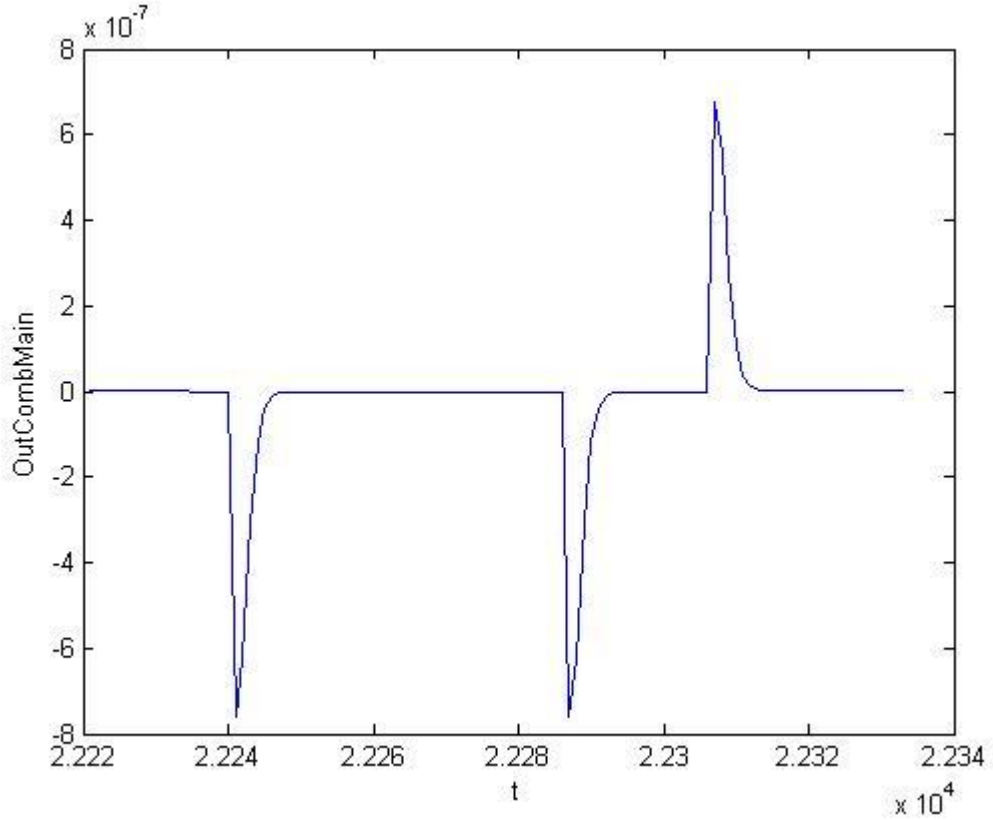


Figura 41 – Dettaglio sull'andamento delle prime repliche degli impulsi nella risposta impulsiva dei Main Comb

Allo stesso modo, si ricavano le uscite dei comb Right e quelle dei comb Left.

A questo punto, si fa la somma tra *OutCombMain* e *OutCombRight* in modo da ottenere l'uscita dal parallelo dei comb da inviare al canale destro e alla prima serie di filtri allpass.

Stesso discorso per l'uscita del parallelo dei comb dai inviare al canale sinistro e alla seconda serie di filtri allpass, calcolata come la somma tra *OutCombMain* e *OutCombLeft.*

I due segnali appena calcolati, *OutRComb* e *OutLComb*, vengono, dunque, inviati in ingresso ai rispettivi filtri allpass, *AllpassRight1* e *AllpassLeft1*.
Un filtro allpass restituisce una sequenza di repliche del segnale di ingresso, che occorrono in corrispondenza di istanti di tempo che sono multipli interi del tempo di ritardo dell'allpass, e alcune di queste repliche potrebbero presentare valori negativi di ampiezza, a seconda del valore dei guadagni di feedback del filtro allpass (uguali in modulo, ma opposti in segno).
OutRComb viene passata in ingresso al primo dei due filtri allpass Right collegati in serie:

```
[DelayLine,OutAllpassRight1(n),Pointer_rd_RightAllpass(1),Pointer_wr_RightAllpas
s(1)] =
AllpassFilter(OutRComb(n),endingDelayRightAllpass(1),startingDelayRightAllpass(1
),AllpassGainRight(1),Pointer_rd_RightAllpass(1),Pointer_wr_RightAllpass(1),Dela
yLine);
```

mentre *OutLComb* viene passata in ingresso al primo dei due filtri allpass Left collegati in serie:

```
[DelayLine,OutAllpassLeft1(n),Pointer_rd_LeftAllpass(1),Pointer_wr_LeftAllpass(1
)] =
AllpassFilter(OutLComb(n),endingDelayLeftAllpass(1),startingDelayLeftAllpass(1),
AllpassGainLeft(1),Pointer_rd_LeftAllpass(1),Pointer_wr_LeftAllpass(1),DelayLine
);
```

Il secondo filtro allpass Right prenderà in ingresso, dunque, il segnale prodotto in uscita dal primo allpass della serie, cioè *OutAllpassRight1*e la stessa cosa verrà fatta dal secondo filtro allpass Left, che riceverà in ingresso il segnale *OutAllpassLeft1* prodotto dal primo allpass della serie

I due segnali di uscita verranno rinominati come *OutAllpassRight* e *OutAllpassLeft*, rispettivamente. La risposta impulsiva in uscita dalle due serie di allpass è riportata in Fig.42 e in Fig.43:

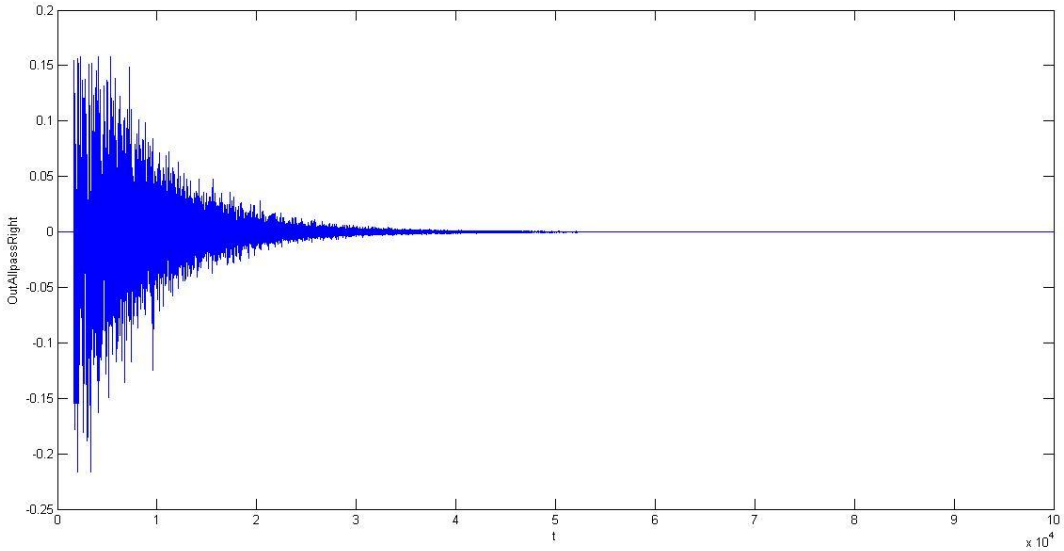


Figura 42 - Grafico di OutAllpassRight

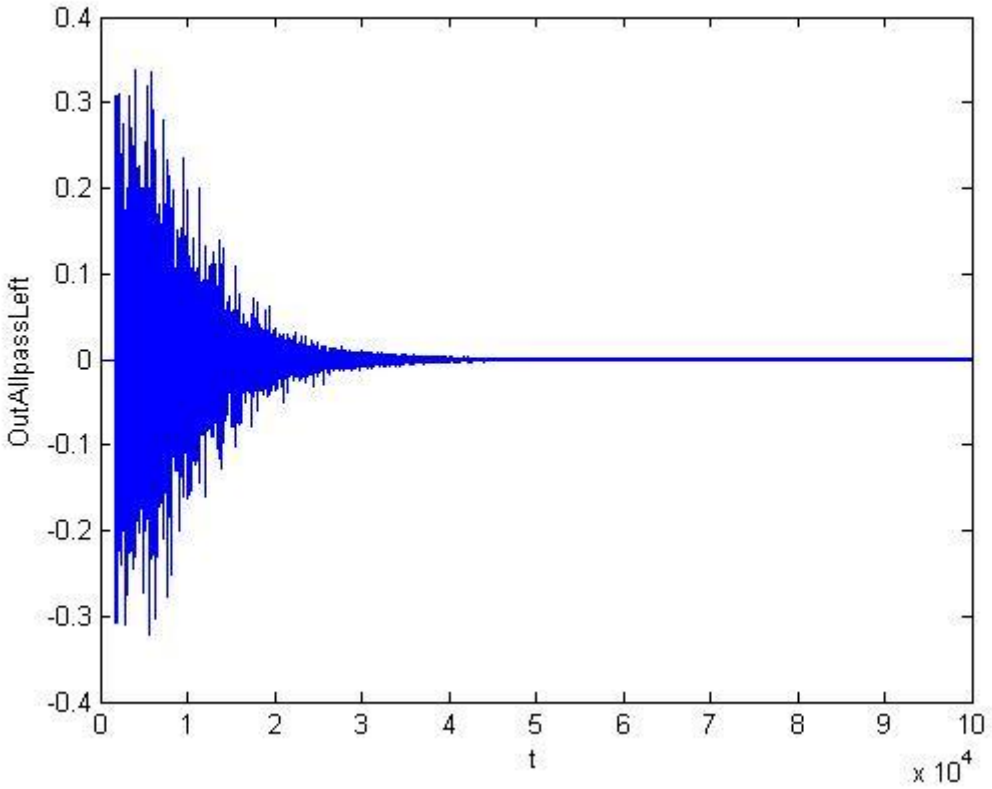


Figura 43 - Grafico di OutAllpassLeft

A questo punto, vengono definiti gli otto contributi al segnale di uscita.
Quattro dei quali (*OutEarlyR2R*, *OutEarlyL2R*, *OutEarly R2R* e *OutEarlyL2L*) rappresentano il contributo da parte delle prime riflessioni del segnale di ingresso, sia dal canale destro che dal sinistro, ai diversi canali di uscita:

- *OutEarlyR2R* è il contributo di uscita delle prime riflessioni prodotte al canale destro, moltiplicato per un guadagno di uscita ed instradato al canale destro.

- *OutEarlyL2R* è il contributo di uscita delle prime riflessioni prodotte al canale sinistro, moltiplicato per un guadagno di uscita ed instradato al canale destro.
- *OutEarlyR2L* è il contributo di uscita delle prime riflessioni prodotte al canale destro, moltiplicato per un guadagno di uscita ed instradato al canale sinistro.
- *OutEarlyL2L* è il contributo di uscita delle prime riflessioni prodotte al canale sinistro, moltiplicato per un guadagno di uscita ed instradato al canale sinistro.

Gli altri quattro segnali esprimono il contributo da parte dei filtri allpass ai diversi canali di uscita:

- *OutAllpassR2R* è il contributo di uscita della serie di allpass Right, moltiplicato per un guadagno di uscita ed instradato al canale destro.
- *OutAllpassL2R* è il contributo di uscita della serie di allpass Left, moltiplicato per un guadagno di uscita ed instradato al canale destro.
- *OutAllpassR2L* è il contributo di uscita della serie di allpass Right, moltiplicato per un guadagno di uscita ed instradato al canale sinistro.
- *OutAllpassL2L* è il contributo di uscita della serie di allpass Left, moltiplicato per un guadagno di uscita ed instradato al canale sinistro.

Infine, si sommano questi 8 segnali per ottenere il segnale in uscita dal riverbero, cioè *Output*, dopodiché, il ciclo for termina.
Tracciando con la funzione *plot* il grafico del segnale in uscita, considerando lo stesso asse t definito inizialmente, si ottiene il risultato riportato in Fig.44:

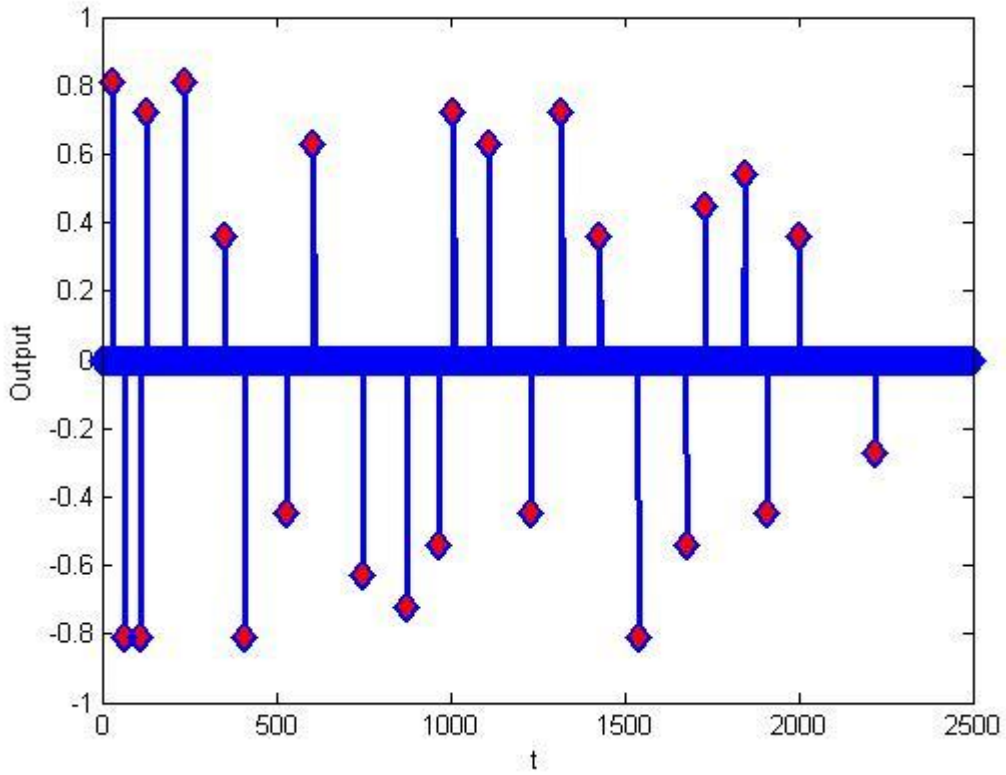


Figura 44 – Risposta impulsiva in uscita del riverbero

Il segnale di uscita presenta, dunque, gli stessi 24 impulsi risultanti dalla combinazione delle prime 12 riflessioni prodotte al canale destro con le prime 12 riflessioni prodotte al canale sinistro, moltiplicati di un fattore pari a 0.9.

Si noti, inoltre,che è assente il contributo degli allpass, dal momento che i quattri segnali discussi poco fa (*OutAllpassR2R*, *OutAllpassL2R*, *OutAllpassR2L*, *OutAllpassL2L*) sono moltiplicati per i loro rispettivi guadagni di uscita, tutti uguali a zero, per cui la somma che definisce *Output* terrà conto del solo contributo delle prime 12 riflessioni prodotte al canale destro e instradate al canale destro assieme al contributo delle prime 12 riflessioni prodotte al canale sinistro e instradate al canale sinistro.

CAPITOLO 7

Simulazione in XMOS

L'ambiente di lavoro che XMOS mette a disposizione per lo sviluppo degli algoritmi è detto XTime Composer, che è appunto una IDE (Integrate Development Environment) che permette di avere, in maniera integrata, editor, compilatore, linker, simulatore e debug. I linguaggi di programmazione che si possono usare con i DSP della XMOS sono C, xC (per la programmazione parallela), C++ e assembler.

Le istruzioni di un progetto XMOS sono definite in tre distinti files:

1) File *audio_effects.h*, in cui vengono inizializzati in linguaggio xC (che è un linguaggio che deriva dal C) i parametri dei building blocks utilizzati per costruire i processori di effetto, nonché le strutture dati per rappresentare ogni singolo building block (es. Filtro comb, filtro allpass...).

2) File *audio_effects.xC*, in cui si definiscono i valori dei parametri dei building blocks. Nello stesso file vengono, inoltre, definite le chiamate alle librerie e alle strutture dati definite nel file *audio_effects.h*, in modo da specificare a quale struttura dati si riferisce il parametro a cui si vuole assegnare un certo valore. All'interno di questo file è anche presente il codice che descrive l'esecuzione del processore di effetto in questione, attraverso le chiamate alle strutture dati definite nel file .h e definendo delle strutture di comunicazione tra XMOS e il DSP (che possono essere canali o interfacce).

3) File *asm_effect.S*, in cui si definiscono le istruzioni in linguaggio macchina che definiscono il comportamento di ogni singolo building block, inteso come elaborazione in diversi stadi del segnale in ingresso. In ogni istruzione vengono implementati dei registri (identificati con la dicitura $r) per l'immagazzinamento dei valori provvisori dei dati da elaborare.

Ogni istruzione in XMOS ha un size di 32 bit e impiega 1 o 2 cicli macchina per essere eseguita. Si possono usare in linea di massima 12 registri (da $r0 a $r11) per immagazzinare temporaneamente i dati intermedi. Inoltre, è data la possibilità di eseguire due istruzioni in parallelo (definendo le suddette all'interno di parentesi grafe) in un singolo ciclo.

Il compilatore ha a disposizione un numero massimo di core definito dal tipo di dispositivo XMOS utilizzato (da un minimo di 4 core fino ad un massimo di 32) e tali cores vengono assegnati automaticamente (solitamente, le funzioni I2S vengono assegnate al core 1, o comunque al primo core libero che il compilatore trova, mentre il main con le altre funzioni xC al core 0).

In seguito, per trasportare i dati da un core all'altro, vengono utilizzate le primitive di collegamento dell'I2S.

Per il risultato finale della simulazione, è stata modificata la chiamata alla funzione *asm_effect.S* per mettere in ingresso un impulso unitario; la stessa lunghezza di risposta all'impulso è stata salvata su file, proprio come è stato fatto in MATLAB.

7.1 Riverbero in XMOS

Nel file .h vengono definite le strutture dati relative ad ogni singolo building block,tramite il comando *typedef struct* . All'interno di ogni struttura vengono dichiarati i parametri relativi al building block considerato (espressi come variabili intere o unsigned),vale a dire il segnale in ingresso, i guadagni, i tempi di ritardo, la dimensione della linea di ritardo (che è la stessa per tutti i building blocks) e i valori correnti dei puntatori in lettura e in scrittura ad ogni campione della linea di ritardo.

Con la direttiva *external*, seguita dal nome della struttura di riferimento e dal nome della variabile da dichiarare, si definisce una variabile esterna che richiama i parametri contenuti all'interno della struttura di riferimento; questo viene fatto con lo scopo di attribuire nel file .xC un valore esplicito ai parametri della struttura di riferimento specificando il nome della variabile external.

Sempre nel file .h, una volta definite le strutture di tutti i building blocks, viene chiamata la funzione audio_effects, di tipo void, passandole come argomenti un canale di flusso aperto di nome *c_dsp_eq* (che serve per ricevere i campioni audio in ingresso e restituire, poi, i campioni aggiornati in uscita), un canale aperto di nome *c_gain* (che serve per ricevere gli aggiornamente del guadagno regolabile dall'esterno) e una costante di nome *num_chans* (il numero dei canali del segnale).

Questa funzione, quando viene chiamata,esegue il contenuto del file .Xc, cioè le istruzioni per il processamento del segnale in ingresso.

Nel file .xC vengono definiti i valori dei parametri coinvolti nel riverbero, compresi la lunghezza della linea di ritardo ed il valore del puntatore alla linea di ritardo. I valori dei parametri dei building blocks definiti nel file .h si definiscono come segue:

nomeStruttura.nomeParametro=valore;

Il segnale impulsivo in ingresso viene dichiarato nel file .xC come una sequenza di k valori (abbiamo imposto *k=100000*,cioè un segnale a impulso unitario costituito da 100000 samples): tramite un ciclo *for* si impone pari a 1 il valore corrispondente a *k=0* e pari a 0 tutti gli altri valori di k diversi da *k=0*.

Alla definizione del suddetto segnale segue un ciclo while in cui si chiama la funzione *asm_effect()* che applica l'algoritmo del riverbero ad ogni singolo sample del segnale, costruendo al tempo stesso il segnale in uscita sample per sample.

Una spiegazione più ad alto livello di quello che succede in questo ambiente di sviluppo è data dal diagramma di flusso in Fig.45, dove è spiegato ad alto livello come lavorano ad ogni ciclo di campionamento le funzioni di I^2S (callback), le funzioni definite nel file .xC e quelle in assembler:

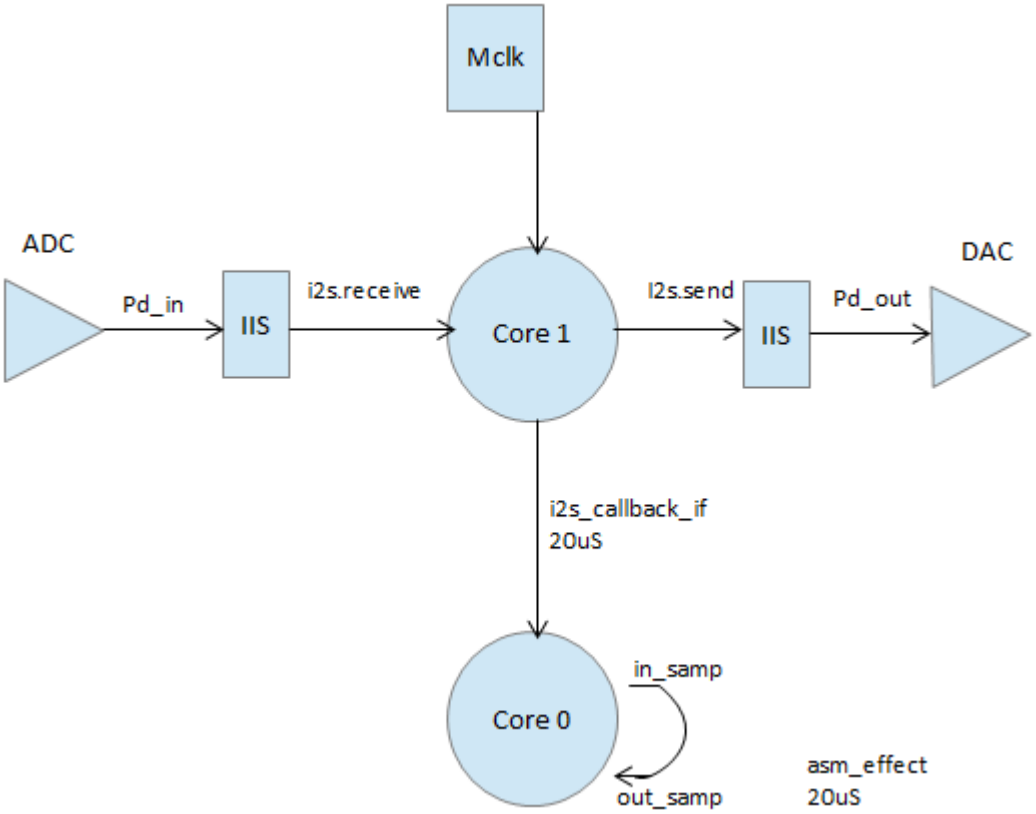


Figura 45 – Diagramma di flusso dei dati del protocollo I^2S

Dall'ADC arrivano i campioni del segnale in ingresso; essi vengono immediatamente passati all'architettura I^2S per mezzo della callback *Pd_in*, la quale ha, appunto, lo scopo di definire i dati in ingresso con cui dovrà lavorare l'I^2S.

A questo punto avviene l'assegnazione delle funzioni dell'I^2S al core 1 (la scelta del core è automatica, a meno che non siano state definite delle istruzioni sull'assegnazione a un core specifico): ciò viene fatto per mezzo di una callback di nome *i2s.receive* che alloca i dati campionati in ingresso all'interno di un buffer del core 1.

A partire dallo stesso core 1, viene istanziata un'altra callback, *i2s_callback_if*, che in un arco di tempo della durata di 20µs assegna le funzioni incluse nel file .xC (incluso il main) ad un altro core disponibile (in questo caso il core 0), ovviamente diverso dal core 1.

All'interno del core 0 vengono eseguite le istruzioni in linguaggio macchina definite all'interno della funzione *asm_effect()*, sempre in una quantità di tempo della durata di 20µs. Ad ogni ciclo di campionamento, la funzione *asm_effect()* prende come parametri i campioni in ingresso (*in_samp*) e al termine dello stesso ciclo vengono prodotti i relativi campioni in uscita (*out_samp*). Al ciclo di campionamento successivo, alla funzione *asm_effect()* vengono passati come parametri sia i campioni in ingresso, *in*_samp, sia quelli in uscita dal ciclo precedente, *out_samp*.

Lo svolgimento delle operazioni all'interno di ogni core è coordinato dal segnale di clock principale, *MCLK*, che può corrispondere al segnale di clock del sistema oppure al clock interno del DSP interfacciato.

I dati risultanti dall'applicazione di *asm_effects()* devono, poi, essere spediti in uscita: per fare questo, si istanzia una callback *i2s.send* tramite la quale I2S salva temporaneamente i dati in un secondo buffer, per poi passarli in ingresso ad un DAC, tramite la callback *Pd_out*, il quale provvede a convertire i campioni in uscita nel segnale analogico risultante in uscita.

7.2 Applicazione dell'algoritmo

All'interno della funzione *asm_effect.S* sono presenti le istruzioni in linguaggio macchina che definiscono il procedimento dell'algoritmo di riverbero. Ciascuna istruzione si basa sul caricamento e sul salvataggio di valori esadecimali in un registro destinazione e questi valori fanno riferimento a delle variabili nelle varie locazioni di memoria di una struttura dati definita nel file *audio_effects.h*.

Quindi,prima di eseguire un'operazione di salvataggio o di caricamento del valore di uno di questi parametri, occorre innanzitutto caricare l'indirizzo di memoria della struttura dati in cui è contenuto tramite lo statement *ldaw*, in modo tale che ogni indirizzo di memoria (espresso tramite un offset) faccia riferimento a una specifica locazione di memoria della struttura dati caricata.

Le prime istruzioni di ogni building block elaborano il segnale in uscita dalla linea di ritardo al k corrente (in sostanza, se si considera come segnale in ingresso *x(k)*, il segnale prodotto all'iterazione k sarà *x(k-1)*).

Il valore di questo segnale corrisponderà ai 16 bit più significativi della word contenuta nella locazione di memoria corrispondente a Ptr_rdR0, considerando come indice Ptr_rd_currentR1 (in breve, si prende il contenuto del puntatore in fase di lettura, Ptr_rdR0, in corrispondenza della posizione corrente, Ptr_rd_currentR1, e se ne immagazzinano i 16 bit più significativi nel registro destinazione espresso dall'istruzione *ld16*).

Il valore del segnale in uscita dalla linea di ritardo andrà, poi, salvato nella locazione di memoria relativa alla variabile di stato (*state*), dopodiché si avanza al sample successivo della linea di ritardo.

Se il sample corrente della linea di ritardo corrispone all'ultimo sample, allora avviene la saturazione della linea: in altre parole, il puntatore viene riportato immediatamente al primo sample della linea di ritardo in modo da ripercorrerla ad ogni iterazione da capo a fondo.

Per verificare la posizione corrente del puntatore si applica uno statement *eq*, che confronta il valore della variabile relativa alla posizione corrente del puntatore con quello della variabile relativa alla dimensione della linea di ritardo. Si possono verificare due casi:

1) I contenuti delle due variabili sono diversi: si esegue l'istruzione prevista dalla condizione *eq*, cioè una *bf* (*branch false*), la quale impone un branch a un certo punto del codice (specificato da una tag) e incrementa il valore della variabile relativa alla posizione corrente di un'unità. Questa procedura descrive l'avanzamento della posizione corrente del puntatore al sample successivo.

2) I contenuti delle due variabili sono uguali: in questo caso, si impone uguale a zero il valore della variabile relativa alla posizione corrente del puntatore. Questa procedura descrive la saturazione del puntatore e il passaggio del puntatore al primo sample della linea di ritardo.

Questa operazione verrà eseguita sia in fase di lettura che in fase di scrittura (quest'ultima avrà luogo dopo che è stata ricavata l'uscita al building block considerato).

Il processamento del segnale all'interno del building block è descritto dalle istruzioni *maccs* (moltiplica e accumula). Una operazione *maccs* esegue il prodotto dei contenuti di due words, salvando i 32 bit più significativi del prodotto in un registro destinazione (che nel nostro caso prende il nome di *RHighR4*) e i 32 bit meno significativi nell'altro registro destinazione (*RLowR6*). Se in uno dei due registri destinazione è già presente un valore diverso da zero (che potrebbe corrispondere a un segnale applicato direttamente in ingresso a un nodo sommatore del building block) allora il risultato del prodotto andrà a sommarsi (o meglio,accumularsi) col prodotto già presente nei registri.

Tutto quello che si verifica nelle istruzioni *maccs* descrive il percorso del segnale in ingresso all'interno di un building block: tale segnale viene moltiplicato per i vari guadagni, sommato ai vari segnali, ed infine restituito in uscita.

Il segnale in uscita dal building block viene sottoposto ad un'istruzione *lextract*, la quale estrae 32 bit del contenuto dei due registri destinazione delle *maccs*, li normalizza a 28 bit e poi li salva all'interno di un registro destinazione. Un'istruzione successiva *stw* salverà il contenuto del registro destinazione della *lextract* all'interno di una locazione di memoria corrispondente all'uscita del building block corrente (oppure all'ingresso del building block immediatamente successivo, eventualmente).

A questa operazione segue l'incremento della posizione corrente del puntatore in fase di scrittura al sample successivo della linea di ritardo (a meno che la posizione corrente non si trovi già all'ultimo sample: in questo caso, avviene quello che è stato descritto in precedenza per il puntatore in lettura, cioè si fa saturare il puntatore riportandolo al primo sample della linea di ritardo).

CAPITOLO 8

Validazione, conclusioni e direttive future

Una volta ottenuti i segnali riverberati sia da MATLAB che da XTime Composer li si mette a confronto: ci si aspetta,al termine dell'esperimento,che i risultati coincidano, almeno dal punto di vista degli istanti di tempo in cui occorrono le repliche dell'impulso.

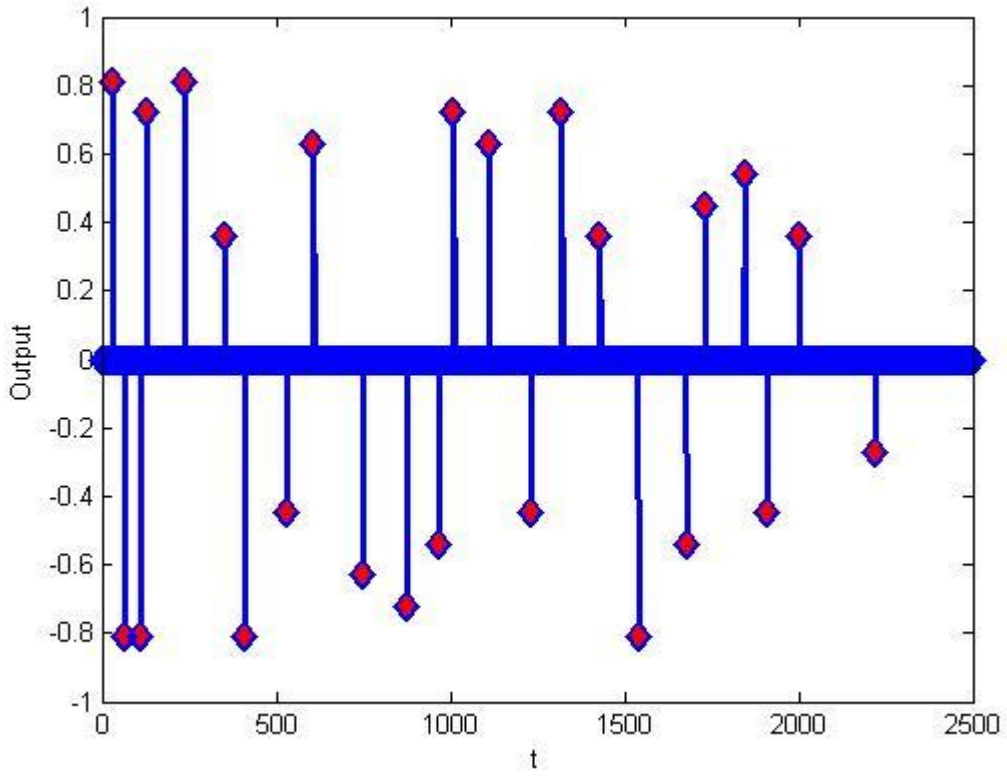


Figura 46 - Segnale in uscita da MATLAB

Nella seguente tabella sono riportati i primi 256 valori della risposta in uscita calcolata; la prima colonna riporta il numero di campione considerato, la seconda il rispettivo valore ottenuto dalla simulazione in XMOS e la terza il valore ottenuto dalla simulazione in MATLAB:

Numero Sample	Valore Xmos	Valore Matlab	Numero Sample	Valore Xmos	Valore Matlab	Numero Sample	Valore Xmos	Valore Matlab	Numero Sample	Valore Xmos	Valore Matlab
0	0	0	64	-0.729	-0.81	128	0	0	192	0	0
1	0	0	65	0	0	129	0	0	193	0	0
2	0	0	66	0	0	130	0.648	0.72	194	0	0
3	0	0	67	0	0	131	0	0	195	0	0
4	0	0	68	0	0	132	0	0	196	0	0
5	0	0	69	0	0	133	0	0	197	0	0
6	0	0	70	0	0	134	0	0	198	0	0
7	0	0	71	0	0	135	0	0	199	0	0
8	0	0	72	0	0	136	0	0	200	0	0

9	0	0	73	0	0	137	0	0	201	0	0
10	0	0	74	0	0	138	0	0	202	0	0
11	0	0	75	0	0	139	0	0	203	0	0
12	0	0	76	0	0	140	0	0	204	0	0
13	0	0	77	0	0	141	0	0	205	0	0
14	0	0	78	0	0	142	0	0	206	0	0
15	0	0	79	0	0	143	0	0	207	0	0
16	0	0	80	0	0	144	0	0	208	0	0
17	0	0	81	0	0	145	0	0	209	0	0
18	0	0	82	0	0	146	0	0	210	0	0
19	0	0	83	0	0	147	0	0	211	0	0
20	0	0	84	0	0	148	0	0	212	0	0
21	0	0	85	0	0	149	0	0	213	0	0
22	0	0	86	0	0	150	0	0	214	0	0
23	0	0	87	0	0	151	0	0	215	0	0
24	0	0	88	0	0	152	0	0	216	0	0
25	0	0	89	0	0	153	0	0	217	0	0
26	0	0	90	0	0	154	0	0	218	0	0
27	0	0	91	0	0	155	0	0	219	0	0
28	0	0	92	0	0	156	0	0	220	0	0
29	0	0	93	0	0	157	0	0	221	0	0
30	0	0	94	0	0	158	0	0	222	0	0
31	0.729	0.81	95	0	0	159	0	0	223	0	0
32	0	0	96	0	0	160	0	0	224	0	0
33	0	0	97	0	0	161	0	0	225	0	0
34	0	0	98	0	0	162	0	0	226	0	0
35	0	0	99	0	0	163	0	0	227	0	0
36	0	0	100	0	0	164	0	0	228	0	0
37	0	0	101	0	0	165	0	0	229	0	0
38	0	0	102	0	0	166	0	0	230	0	0
39	0	0	103	0	0	167	0	0	231	0	0
40	0	0	104	0	0	168	0	0	232	0	0
41	0	0	105	0	0	169	0	0	233	0	0
42	0	0	106	0	0	170	0	0	234	0	0
43	0	0	107	0	0	171	0	0	235	0	0
44	0	0	108	0	0	172	0	0	236	0.729	0.81
45	0	0	109	0	0	173	0	0	237	0	0
46	0	0	110	-0.729	-0.81	174	0	0	238	0	0
47	0	0	111	0	0	175	0	0	239	0	0
48	0	0	112	0	0	176	0	0	240	0	0
49	0	0	113	0	0	177	0	0	241	0	0
50	0	0	114	0	0	178	0	0	242	0	0
51	0	0	115	0	0	179	0	0	243	0	0
52	0	0	116	0	0	180	0	0	244	0	0
53	0	0	117	0	0	181	0	0	245	0	0
54	0	0	118	0	0	182	0	0	246	0	0
55	0	0	119	0	0	183	0	0	247	0	0
56	0	0	120	0	0	184	0	0	248	0	0
57	0	0	121	0	0	185	0	0	249	0	0
58	0	0	122	0	0	186	0	0	250	0	0
59	0	0	123	0	0	187	0	0	251	0	0
60	0	0	124	0	0	188	0	0	252	0	0
61	0	0	125	0	0	189	0	0	253	0	0

62	0	0	126	0	0	190	0	0	254	0	0
63	0	0	127	0	0	191	0	0	255	0	0

Tabella 2 – Primi 256 valori della risposta in uscita calcolata, rispettivamente, nella simulazione in XMOS, dopo aver ripulito il segnale, e nella simulazione in MATLAB

Da notare che i valori di MATLAB e quelli di XMOS sono uguali a meno di qualche piccolo riscalamento (che fa abbassare lievemente i valori in uscita dalla simulazione XMOS), necessario per evitare la saturazione. Questo è dovuto al fatto che l'esperimento è stato condotto in un sistema chiuso, considerando un segnale numerico in ingresso, generato artificialmente, rappresentato da una stringa di *n* campioni il cui primo campione è pari a 1 e tutti gli altri sono pari a 0.

Non sono stati usati né microfoni né qualsiasi altro stimolo esterno per prelevare ed elaborare il segnale in tempo reale, pertanto i parametri del sistema da tenere in considerazione sono sempre gli stessi. La differenza tra i risultati ottenuti in Matlab e quelli ottenuti in XMOS è dovuta alla precisione matematica.

8.1 Conclusioni

L'algoritmo di riverbero di cui si è discusso in questa tesi può essere considerato un punto di partenza per la progettazione di effetti di riverberazione più particolari e ricercati.

Un'idea potrebbe essere quella di implementare una struttura di linee di ritardo più ottimizzata, organizzata in maniera molto simile ad una FDN.

Le prove da fare in questa direzione sarebbero diverse:

1) Prelevare il segnale uscente non da un filtro comb ma attraverso una seconda linea di ritardo in maniera da regolare la fase del segnale di ogni filtro.
2) Provare a fare una struttura ricircolante (per esempio iniziando con soli 2 filtri comb) mandando i segnali di feedback dei due comb anche negli ingressi degli altri filtri. Tale modifica dovrebbe creare un addenzamento notevole sulle code del riverbero, modificandone il suono. In questo caso, una cura adeguata deve essere presa per evitare che il riverbero diventi instabile.

Un'altra idea sarebbe sfruttare al massimo le potenzialità offerte dai DSP XCORE, implementando una programmazione parallela su più core (nella nostra trattazione ne abbiamo usato solo uno) ed utilizzare i DSP come se fossero una VLIW (Very Large Instruction Word Architecture). Questo permette di avere maggiori risorse di calcolo a disposizione per poter implementare algoritmi più complessi (ciò che non sarebbe pensabile nel caso di un solo core).

APPENDICE

A - Istruzioni per accesso alla memoria

Istruzioni per accesso alla memoria tramite stack pointer:

- *LDWSP*, carica una word dallo stack.

  $$D \leftarrow mem[sp + u_{16} \times Bpw]$$

  Nel registro D viene caricata la word contenuta nella locazione di memoria il cui indirizzo è dato dal risultato della somma tra parentesi quadre.

- *STWSP*, salva una word nello stack.

  $$mem[sp + u_{16} \times Bpw] \leftarrow S$$

  Il contenuto del registro S viene salvato nella locazione di memoria il cui indirizzo è dato dal risultato della somma tra parentesi quadre.

- *LDAWSP*, carica l'indirizzo di una word contenuta nello stack.

  $$D \leftarrow sp + u_{16} \times Bpw$$

  Al registro D andrà il risultato della somma *sp + u16 x Bpw*.

Istruzioni per l'accesso tramite data pointer:

- *LDWDP*, carica il contenuto di una word dai dati.

  $$D \leftarrow mem[dp + u_{16} \times Bpw]$$

  In D verrà memorizzata la word contenuta in memoria,il cui indirizzo è dato dal risultato della somma tra parentesi quadre (si noti che, in questo caso, al posto di *sp* compare *dp* perché adesso stiamo considerando il data pointer, e non più lo stack pointer).

- *STWDP*, salva il contenuto di una word nei dati.

  $$mem[dp + u_{16} \times Bpw] \leftarrow S$$

  Il contenuto di S andrà salvato nella locazione di memoria di indirizzo pari al risultato della somma tra parentesi quadre.

- *LDAWDP*, carica l'indirizzo di una word contenuta nei dati.

  $$D \leftarrow dp + u_{16} \times Bpw$$

In D verrà caricato il risultato di tale somma.

Istruzioni di accesso a valori costanti e a indirizzi di programmi:

- *LDC*, carica una costante.

$$D \leftarrow u_{16}$$

Il valore contenuto in D sarà la costante u16 (in questo caso, un valore a 16 bit, dunque esadecimale).

- *LDWCP*, carica il contenuto di una word da una fonte costante (constant pool).

$$D \leftarrow mem[cp + u_{16} \times Bpw]$$

In D verrà caricato il contenuto della word il cui indirizzo è dato dal risultato della somma tra parentesi quadre.

- *LDAWCP*, carica l'indirizzo di una word contenuta nella constant pool.

$$r11 \leftarrow cp + u_{16} \times Bpw$$

Nel registro operando *r11* (ne è stato scelto uno qualsiasi tra i 12 disponibili) verrà caricato l'indirizzo della constant pool corrispondente al risultato di questa somma.

- *LDWCPL*, carica il contenuto di una word di tipo long (nel senso che è costituita da un maggior numero di bit) che si trova nella constant pool.

$$r11 \leftarrow mem[cp + u_{20} \times Bpw]$$

Nel registro *r11* verrà caricato il contenuto della word il cui indirizzo corrisponde al risultato della somma tra parentesi quadre.

Istruzioni di accesso che usano come indirizzo base un qualsiasi registro operando, che viene combinato con un offset. Nel caso di accesso a word, l'operando in questione può essere o una piccola costante o un altro registro operando. Le istruzioni sono le seguenti:

- *LDW*, carica una word.

$$d \leftarrow mem[b + i \times Bpw]$$

- *STW*, salva una word.

$$mem[b + i \times Bpw] \leftarrow s$$

- *LDAW*, carica l'indirizzo di una word.

$$d \leftarrow b + i \times Bpw$$

Nel caso di accesso a grandezze a 16 bit, l'indirizzo base viene combinato con un registro operando scalato. Il bit meno significativo del registro risultante deve essere zero. Le istruzioni sono:

- *LD16S*, carica 16 bit di un valore di tipo signed, estendendo il segno all'intera word.

$$d \leftarrow sext(mem[b + i \times 2], 16)$$

  In d verrà caricato il contenuto, col segno, di una word il cui indirizzo è dato dai 16 bit più significativi del risultato della somma *b + i x 2*.

- *ST16*, salva un valore a 16 bit in una locazione di memoria di indirizzo pari a *b + i x 2*.

$$mem[b + i \times 2] \leftarrow s$$

- *LDA16*, carica l'indirizzo di una quantità espressa in 16 bit.

$$d \leftarrow b + i \times 2$$

Nel caso in cui si voglia accedere a quantità a 8 bit, l'indirizzo base è combinato con un operando non scalato (quindi si avrà *LD8U*), che deve sempre essere un registro operando.

XMOS offre la possibilità di accedere a coppie di word in un'unica istruzione. Per fare questo, è necessario che l'indirizzo sia allineato su un margine di due word, cioè deve essere un multiplo di *Bpw x 2*.

Per caricare il contenuto di due word devono essere specificati due registri destinazione (uno per caricare il contenuto della prima word, e l'altro per caricare il contenuto della seconda), mentre per salvare il contenuto di due word devono essere specificati due registri sorgente.

Istruzioni su coppie di word:

- *LDDSP*, carica due word dallo stack.

$$d \leftarrow mem[sp + u_s \times Bpw \times 2]$$
$$e \leftarrow mem[sp + u_s \times Bpw \times 2 + Bpw]$$

  Nel registro d verrà caricato il contenuto della word il cui indirizzo è dato dal risultato della somma tra parentesi quadre. Nel registro e, invece, verrà caricato il contenuto della seconda word; si noti che il valore tra parentesi quadre è un multiplo di *Bpw x 2*, in quanto il contenuto tra parentesi quadre nella riga precedente viene sommato a *Bpw*.

- *STDSP*, salva due word nello stack.

$$mem[sp + u_s \times Bpw \times 2] \leftarrow x$$
$$mem[sp + u_s \times Bpw \times 2 + Bpw] \leftarrow y$$

- *LDD*, carica due word.

$$d \leftarrow mem[b + i \times Bpw \times 2]$$
$$e \leftarrow mem[b + i \times Bpw \times 2 + Bpw]$$

- *STD*, salva due word.

$$mem[b + i \times Bpw \times 2] \leftarrow x$$
$$mem[b + i \times Bpw \times 2 + Bpw] \leftarrow y$$

Le istruzioni che seguono sono le stesse espressioni di valutazione utilizzate in linguaggio Assembler:

- *ADD*, operazione di somma.
$$d \leftarrow l + r$$
Il contenuto di l e quello di r vengono sommati tra loro, e il risultato della somma viene immagazzinato in d.

- *ADDI*, operazione di somma di valori immediati.
$$d \leftarrow l + u_s$$
Il contenuto di l viene sommato ad una costante unsigned *us*, e il risultato di tale somma viene immagazzinato in d.

- *SUB*, operazione di sottrazione.
$$d \leftarrow l - r$$
Al valore di l viene sottratto quello di r,e il risultato della differenza viene immagazzinato in d.

- *SUBI*, operazione di sottrazione di valori immediati.
$$d \leftarrow l - u_s$$
Al valore di l viene sottratta una costante unsigned *us*, e il risultato della differenza viene immagazzinato in d.

- *NEG*, operazione di inversione di segno.
$$d \leftarrow -s$$
Il segno del contenuto di s viene invertito (quindi,se il contenuto di s è positivo, con *NEG* diventerà negativo) e poi salvato in d.

- *EQI*, operazione di uguaglianza con valori immediati.
$$d \leftarrow l = u_s$$
Si impone il contenuto di l uguale al valore di *us* e lo si salva in d.

- *EQ*, operazione di uguaglianza.
$$d \leftarrow l = r$$
Si impone il contenuto di l uguale a quello di r e lo si salva in d.

- *LSU*, less-than (minore di) unsigned.

$$d \leftarrow l < r$$

Si verifica se il contenuto di l è minore del contenuto di r e si salva l'esito di tale verifica in d (1 se l < r, 0 viceversa).

- *LSS*, less-than signed.

$$d \leftarrow l <_{sgn} r$$

Si verifica se il valore (con segno) di l è minore di quello di r (anche questo con segno) e si salva l'esito della verifica in d.

- *NOP*, nessuna operazione da svolgere.

Istruzione di branch condizionato:

- *BF*, branch false. Dopo aver introdotto un'istruzione che definisce una condizione (es. una *EQ*, che ritorna 1 se l'uguaglianza è verificata, 0 altrimenti), se la condizione precedente è falsa si esegue l'istruzione specificata nella *bf*, in caso contrario l'istruzione della *bf* viene ignorata e si passa alla riga di codice successiva.

Altre istruzioni:

- *LEXTRACT:*

$$\texttt{LEXTRACT} \qquad d, l, r, x, bitp$$

$$d \leftarrow (l : r)[\text{bit } bitp + x - 1..x]$$

Data una coppia di registri, l e r, *LEXTRACT* estrae i bit di l e r in corrispondenza di una certa posizione (data da *bitp*), e relativi al registro x, e li salva in d.

- *MACCS*:

$$\begin{aligned} & \texttt{MACCS} \qquad d, e, x, y \\ e \quad & \leftarrow \quad tmp[bpw - 1...0] \\ d \quad & \leftarrow \quad tmp[2 \times bpw - 1...bpw] \\ & \text{where } tmp \leftarrow (d_{signed} : e) + x_{signed} \times y_{signed} \end{aligned}$$

MACCS effettua il prodotto tra due word, *x* e *y*, con segno (signed), ottenendo come risultato una double word che andrà, poi, passata a un accumulatore di tipo double. L'accumulatore è comprensivo di due registri, *d* ed *e*, che servono entrambi da sorgente e destinazione.

I contenuti di *x* e *y*,dunque,vengono moltiplicati tra loro; il risultato di tale prodotto è una word a 64 bit, di cui i 32 più significativi verranno salvati in d, mentre gli altri 32 meno significativi in e.

B - Codice MATLAB

EarlyReflectionsRight.m

```
function [DelayLineOut,stEarlyRight,OutEarlyRight,Pointer_rd,Pointer_wr] =
EarlyReflectionsRight(InEarlyRight,Pointer_Current_rd,Pointer_Current_wr,EarlyRi
ghtDelay,StartingDelay,ER_G1,EarlyRightGain,ER_G3,DelayLine,state,index)

d=DelayLine(Pointer_Current_rd);
a=ER_G1*d;
b=ER_G3*state(index);
c=a+b;
state(index)=c;

    if Pointer_Current_rd==EarlyRightDelay
     Pointer_Current_rd=StartingDelay+1;
     else Pointer_Current_rd=Pointer_Current_rd+1;
    end

DelayLine(Pointer_Current_wr)=InEarlyRight+c;

    if Pointer_Current_wr==EarlyRightDelay
        Pointer_Current_wr=StartingDelay+1;
    else Pointer_Current_wr=Pointer_Current_wr+1;
    end

DelayLineOut = DelayLine;
stEarlyRight=state;
OutEarlyRight=EarlyRightGain*d;
Pointer_rd=Pointer_Current_rd;
Pointer_wr=Pointer_Current_wr;
```

EarlyReflectionsLeft.m

```
function [DelayLineOut,stEarlyLeft,OutEarlyLeft,Pointer_rd,Pointer_wr] =
EarlyReflectionsLeft(InEarlyLeft,Pointer_Current_rd,Pointer_Current_wr,EarlyLeft
Delay,StartingDelay,EL_G1,EarlyLeftGain,EL_G3,DelayLine,state,index)

d=DelayLine(Pointer_Current_rd);
a=EL_G1*d;
b=EL_G3*state(index);
c=a+b;
state(index)=c;

    if Pointer_Current_rd==EarlyLeftDelay
     Pointer_Current_rd=StartingDelay+1;
     else Pointer_Current_rd=Pointer_Current_rd+1;
    end

DelayLineOut = DelayLine;
stEarlyLeft=state;
OutEarlyLeft=EarlyRightGain*d;
Pointer_rd=Pointer_Current_rd;
Pointer_wr=Pointer_Current_wr;
```

CombFilter.m

```
function [DelayLineOut,stComb,OutComb,Pointer_rd,Pointer_wr] =
CombFilter(InComb,CombDelay,StartingDelay,G1,G2,G3,Pointer_Current_rd,Pointer_Cu
rrent_wr,DelayLine,state,index)

a=G1*d;
b=G3*state(index);
d=DelayLine(Pointer_Current_rd);
state(index)=c;

    if Pointer_Current_rd==CombDelay
     Pointer_Current_rd=StartingDelay+1;
     else Pointer_Current_rd=Pointer_Current_rd+1;
    end

DelayLine(Pointer_Current_wr)=InComb+c;

      if Pointer_Current_wr==CombDelay
        Pointer_Current_wr=StartingDelay+1;
    else Pointer_Current_wr=Pointer_Current_wr+1;
    end

DelayLineOut = DelayLine;
stComb=state;
OutComb=G2*d;
Pointer_rd=Pointer_Current_rd;
Pointer_wr=Pointer_Current_wr;
```

AllpassFilter.m

```
function [DelayLineOut,OutAllpass,Pointer_rd,Pointer_wr] =
AllpassFilter(InAllpass,AllpassDelay,StartingDelay,AllpassGain,Pointer_Current_r
d,Pointer_Current_wr,DelayLine)

d=DelayLine(Pointer_Current_rd);
a=AllpassGain*d;
b=InAllpass+a;
c=-(AllpassGain)*b;

      if Pointer_Current_rd==AllpassDelay;
            Pointer_Current_rd=StartingDelay+1;
            else Pointer_Current_rd=Pointer_Current_rd+1;
    end

DelayLine(Pointer_Current_wr)=b;

      if Pointer_Current_wr==AllpassDelay;
            Pointer_Current_wr=StartingDelay+1;
        else Pointer_Current_wr=Pointer_Current_wr+1;
          end


      DelayLineOut = DelayLine;
      OutAllpass=c+d;
      Pointer_rd=Pointer_Current_rd;
      Pointer_wr=Pointer_Current_wr;
```

ambience2.m

```
x=zeros(100000,1);
x(1)=1;
t = (1:1:length(x));

plot(t,x,'bd','LineWidth',2,'MarkerEdgeColor','b','MarkerFaceColor','r',
'MarkerSize',10);
      xlabel('Tempo'); ylabel('Segnale');

DLine = 44000;
      DelayLine=zeros(1,DLine);

EarlyRightDelay =[111;237;411;609;877;1011;1234;1431;1679;1845;2001;2221];
      EarlyLeftDelay = [32;65;131;353;531;752;971;1111;1321;1541;1731;1911];
      CombDelayMain = [0;0;0;0;0;3697;3921];
      CombDelayRight = [1581;1921;0;0;0;0;0];
      CombDelayLeft = [1811;1771;0;0;0;0;0];
      AllpassDelayRight = [2057;21];
      AllpassDelayLeft = [2051;17];

ER_G1 = [0;0;0;0;0;0;0;0;0;0;0;0];
      EarlyRightGain = [-0.9;0.9;-0.9;0.7;-0.8;0.8;-0.5;0.4;-0.6;0.6;0.4;-0.3];
      ER_G3 = [0;0;0;0;0;0;0;0;0;0;0;0];
      EL_G1 = [0;0;0;0;0;0;0;0;0;0;0;0];
      EarlyLeftGain = [0.9;-0.9;0.8;0.4;-0.5;-0.7;-0.6;0.7;0.8;-0.9;0.5;-0.5];
      EL_G3 = [0;0;0;0;0;0;0;0;0;0;0;0];
      MainG1 = [0.79996;0.79996;0.79996;0.79996;0.79996;0.65545;0.64758];
      MainG2 = [0.0009;0.0001;0.0001;0.00001;0.0001;0.00001;0.0001];
      MainG3 = [0.19999;0.19999;0.19999;0.19999;0.19999;0.16386;0.1619];
      RightG1 = [0.73464;0.7213;0;0;0;0;0];
      RightG2 = [0.5;0.5;0;0;0;0;0];
      RightG3 = [0.18366;0.18033;0;0;0;0;0];
      LeftG1 = [0.72559;0.72716;0;0;0;0;0];
      LeftG2 = [0.999;0.999;0;0;0;0;0];
      LeftG3 = [0.1814;0.18179;0;0;0;0;0];
      AllpassGainRight = [0.7;-0.7];
      AllpassGainLeft = [-0.7;0.7];
      EarlyRightToRight = 0.9;
      EarlyLeftToRight = 0;
      AllpassRightToRight = 0;
      AllpassLeftToRight = 0;
      AllpassRightToLeft = 0;
      AllpassLeftToLeft = 0;
      EarlyRightToLeft = 0;
      EarlyLeftToLeft = 0.9;


state=zeros(7,1);
stateEarly=zeros(12,1);


Pointer_rd_EarlyRight=zeros(12,1);
Pointer_wr_EarlyRight=zeros(12,1);
Pointer_rd_EarlyLeft=zeros(12,1);
Pointer_wr_EarlyLeft=zeros(12,1);
Pointer_rd_MainComb=zeros(7,1);
Pointer_wr_MainComb=zeros(7,1);
Pointer_rd_RightComb=zeros(7,1);
Pointer_wr_RightComb=zeros(7,1);
Pointer_rd_LeftComb=zeros(7,1);
Pointer_wr_LeftComb=zeros(7,1);
Pointer_rd_RightAllpass=zeros(2,1);
Pointer_wr_RightAllpass=zeros(2,1);
```

```
Pointer_rd_LeftAllpass=zeros(2,1);
Pointer_wr_LeftAllpass=zeros(2,1);


startingDelayEarlyRight = [0;
    EarlyRightDelay(1);
    EarlyRightDelay(1)+EarlyRightDelay(2);
    EarlyRightDelay(1)+EarlyRightDelay(2)+EarlyRightDelay(3);
    EarlyRightDelay(1)+EarlyRightDelay(2)+EarlyRightDelay(3)+EarlyRightDelay(4);

EarlyRightDelay(1)+EarlyRightDelay(2)+EarlyRightDelay(3)+EarlyRightDelay(4)+Earl
yRightDelay(5);

EarlyRightDelay(1)+EarlyRightDelay(2)+EarlyRightDelay(3)+EarlyRightDelay(4)+Earl
yRightDelay(5)+EarlyRightDelay(6);

EarlyRightDelay(1)+EarlyRightDelay(2)+EarlyRightDelay(3)+EarlyRightDelay(4)+Earl
yRightDelay(5)+EarlyRightDelay(6)+EarlyRightDelay(7);

EarlyRightDelay(1)+EarlyRightDelay(2)+EarlyRightDelay(3)+EarlyRightDelay(4)+Earl
yRightDelay(5)+EarlyRightDelay(6)+EarlyRightDelay(7)+EarlyRightDelay(8);

EarlyRightDelay(1)+EarlyRightDelay(2)+EarlyRightDelay(3)+EarlyRightDelay(4)+Earl
yRightDelay(5)+EarlyRightDelay(6)+EarlyRightDelay(7)+EarlyRightDelay(8)+EarlyRig
htDelay(9);

EarlyRightDelay(1)+EarlyRightDelay(2)+EarlyRightDelay(3)+EarlyRightDelay(4)+Earl
yRightDelay(5)+EarlyRightDelay(6)+EarlyRightDelay(7)+EarlyRightDelay(8)+EarlyRig
htDelay(9)+EarlyRightDelay(10);

EarlyRightDelay(1)+EarlyRightDelay(2)+EarlyRightDelay(3)+EarlyRightDelay(4)+Earl
yRightDelay(5)+EarlyRightDelay(6)+EarlyRightDelay(7)+EarlyRightDelay(8)+EarlyRig
htDelay(9)+EarlyRightDelay(10)+EarlyRightDelay(11)];

startingDelayEarlyLeft = [startingDelayEarlyRight(12)+EarlyRightDelay(12);
    startingDelayEarlyRight(12)+EarlyRightDelay(12)+EarlyLeftDelay(1);

startingDelayEarlyRight(12)+EarlyRightDelay(12)+EarlyLeftDelay(1)+EarlyLeftDelay
(2);

startingDelayEarlyRight(12)+EarlyRightDelay(12)+EarlyLeftDelay(1)+EarlyLeftDelay
(2)+EarlyLeftDelay(3);

startingDelayEarlyRight(12)+EarlyRightDelay(12)+EarlyLeftDelay(1)+EarlyLeftDelay
(2)+EarlyLeftDelay(3)+EarlyLeftDelay(4);

startingDelayEarlyRight(12)+EarlyRightDelay(12)+EarlyLeftDelay(1)+EarlyLeftDelay
(2)+EarlyLeftDelay(3)+EarlyLeftDelay(4)+EarlyLeftDelay(5);

startingDelayEarlyRight(12)+EarlyRightDelay(12)+EarlyLeftDelay(1)+EarlyLeftDelay
(2)+EarlyLeftDelay(3)+EarlyLeftDelay(4)+EarlyLeftDelay(5)+EarlyLeftDelay(6);

startingDelayEarlyRight(12)+EarlyRightDelay(12)+EarlyLeftDelay(1)+EarlyLeftDelay
(2)+EarlyLeftDelay(3)+EarlyLeftDelay(4)+EarlyLeftDelay(5)+EarlyLeftDelay(6)+Earl
yLeftDelay(7);

startingDelayEarlyRight(12)+EarlyRightDelay(12)+EarlyLeftDelay(1)+EarlyLeftDelay
(2)+EarlyLeftDelay(3)+EarlyLeftDelay(4)+EarlyLeftDelay(5)+EarlyLeftDelay(6)+Earl
yLeftDelay(7)+EarlyLeftDelay(8);

startingDelayEarlyRight(12)+EarlyRightDelay(12)+EarlyLeftDelay(1)+EarlyLeftDelay
```

```
(2)+EarlyLeftDelay(3)+EarlyLeftDelay(4)+EarlyLeftDelay(5)+EarlyLeftDelay(6)+Earl
yLeftDelay(7)+EarlyLeftDelay(8)+EarlyLeftDelay(9);

startingDelayEarlyRight(12)+EarlyRightDelay(12)+EarlyLeftDelay(1)+EarlyLeftDelay
(2)+EarlyLeftDelay(3)+EarlyLeftDelay(4)+EarlyLeftDelay(5)+EarlyLeftDelay(6)+Earl
yLeftDelay(7)+EarlyLeftDelay(8)+EarlyLeftDelay(9)+EarlyLeftDelay(10);

startingDelayEarlyRight(12)+EarlyRightDelay(12)+EarlyLeftDelay(1)+EarlyLeftDelay
(2)+EarlyLeftDelay(3)+EarlyLeftDelay(4)+EarlyLeftDelay(5)+EarlyLeftDelay(6)+Earl
yLeftDelay(7)+EarlyLeftDelay(8)+EarlyLeftDelay(9)+EarlyLeftDelay(10)+EarlyLeftDe
lay(11)];

startingDelayMainComb =
[0;0;0;0;0;startingDelayEarlyLeft(12)+EarlyLeftDelay(12);startingDelayEarlyLeft(
12)+EarlyLeftDelay(12)+CombDelayMain(6)];

startingDelayRightComb =
[startingDelayMainComb(7)+CombDelayMain(7);startingDelayMainComb(7)+CombDelayMai
n(7)+CombDelayRight(1);0;0;0;0;0];

startingDelayLeftComb =
[startingDelayRightComb(2)+CombDelayRight(2);startingDelayRightComb(2)+CombDelay
Right(2)+CombDelayLeft(1);0;0;0;0;0];

startingDelayRightAllpass =
[startingDelayLeftComb(2)+CombDelayLeft(2);startingDelayLeftComb(2)+CombDelayLef
t(2)+AllpassDelayRight(1)];

startingDelayLeftAllpass =
[startingDelayRightAllpass(2)+AllpassDelayRight(2);startingDelayRightAllpass(2)+
AllpassDelayRight(2)+AllpassDelayLeft(1)];


endingDelayEarlyRight = [EarlyRightDelay(1);
    EarlyRightDelay(1)+EarlyRightDelay(2);
    EarlyRightDelay(1)+EarlyRightDelay(2)+EarlyRightDelay(3);
    EarlyRightDelay(1)+EarlyRightDelay(2)+EarlyRightDelay(3)+EarlyRightDelay(4);

EarlyRightDelay(1)+EarlyRightDelay(2)+EarlyRightDelay(3)+EarlyRightDelay(4)+Earl
yRightDelay(5);

EarlyRightDelay(1)+EarlyRightDelay(2)+EarlyRightDelay(3)+EarlyRightDelay(4)+Earl
yRightDelay(5)+EarlyRightDelay(6);

EarlyRightDelay(1)+EarlyRightDelay(2)+EarlyRightDelay(3)+EarlyRightDelay(4)+Earl
yRightDelay(5)+EarlyRightDelay(6)+EarlyRightDelay(7);

EarlyRightDelay(1)+EarlyRightDelay(2)+EarlyRightDelay(3)+EarlyRightDelay(4)+Earl
yRightDelay(5)+EarlyRightDelay(6)+EarlyRightDelay(7)+EarlyRightDelay(8);

EarlyRightDelay(1)+EarlyRightDelay(2)+EarlyRightDelay(3)+EarlyRightDelay(4)+Earl
yRightDelay(5)+EarlyRightDelay(6)+EarlyRightDelay(7)+EarlyRightDelay(8)+EarlyRig
htDelay(9);

EarlyRightDelay(1)+EarlyRightDelay(2)+EarlyRightDelay(3)+EarlyRightDelay(4)+Earl
yRightDelay(5)+EarlyRightDelay(6)+EarlyRightDelay(7)+EarlyRightDelay(8)+EarlyRig
htDelay(9)+EarlyRightDelay(10);

EarlyRightDelay(1)+EarlyRightDelay(2)+EarlyRightDelay(3)+EarlyRightDelay(4)+Earl
yRightDelay(5)+EarlyRightDelay(6)+EarlyRightDelay(7)+EarlyRightDelay(8)+EarlyRig
htDelay(9)+EarlyRightDelay(10)+EarlyRightDelay(11);
```

```
EarlyRightDelay(1)+EarlyRightDelay(2)+EarlyRightDelay(3)+EarlyRightDelay(4)+Earl
yRightDelay(5)+EarlyRightDelay(6)+EarlyRightDelay(7)+EarlyRightDelay(8)+EarlyRig
htDelay(9)+EarlyRightDelay(10)+EarlyRightDelay(11)+EarlyRightDelay(12)];

endingDelayEarlyLeft = [endingDelayEarlyRight(12)+EarlyLeftDelay(1);
    endingDelayEarlyRight(12)+EarlyLeftDelay(1)+EarlyLeftDelay(2);

endingDelayEarlyRight(12)+EarlyLeftDelay(1)+EarlyLeftDelay(2)+EarlyLeftDelay(3);

endingDelayEarlyRight(12)+EarlyLeftDelay(1)+EarlyLeftDelay(2)+EarlyLeftDelay(3)+
EarlyLeftDelay(4);

endingDelayEarlyRight(12)+EarlyLeftDelay(1)+EarlyLeftDelay(2)+EarlyLeftDelay(3)+
EarlyLeftDelay(4)+EarlyLeftDelay(5);

endingDelayEarlyRight(12)+EarlyLeftDelay(1)+EarlyLeftDelay(2)+EarlyLeftDelay(3)+
EarlyLeftDelay(4)+EarlyLeftDelay(5)+EarlyLeftDelay(6);

endingDelayEarlyRight(12)+EarlyLeftDelay(1)+EarlyLeftDelay(2)+EarlyLeftDelay(3)+
EarlyLeftDelay(4)+EarlyLeftDelay(5)+EarlyLeftDelay(6)+EarlyLeftDelay(7);

endingDelayEarlyRight(12)+EarlyLeftDelay(1)+EarlyLeftDelay(2)+EarlyLeftDelay(3)+
EarlyLeftDelay(4)+EarlyLeftDelay(5)+EarlyLeftDelay(6)+EarlyLeftDelay(7)+EarlyLef
tDelay(8);

endingDelayEarlyRight(12)+EarlyLeftDelay(1)+EarlyLeftDelay(2)+EarlyLeftDelay(3)+
EarlyLeftDelay(4)+EarlyLeftDelay(5)+EarlyLeftDelay(6)+EarlyLeftDelay(7)+EarlyLef
tDelay(8)+EarlyLeftDelay(9);

endingDelayEarlyRight(12)+EarlyLeftDelay(1)+EarlyLeftDelay(2)+EarlyLeftDelay(3)+
EarlyLeftDelay(4)+EarlyLeftDelay(5)+EarlyLeftDelay(6)+EarlyLeftDelay(7)+EarlyLef
tDelay(8)+EarlyLeftDelay(9)+EarlyLeftDelay(10);

endingDelayEarlyRight(12)+EarlyLeftDelay(1)+EarlyLeftDelay(2)+EarlyLeftDelay(3)+
EarlyLeftDelay(4)+EarlyLeftDelay(5)+EarlyLeftDelay(6)+EarlyLeftDelay(7)+EarlyLef
tDelay(8)+EarlyLeftDelay(9)+EarlyLeftDelay(10)+EarlyLeftDelay(11);

endingDelayEarlyRight(12)+EarlyLeftDelay(1)+EarlyLeftDelay(2)+EarlyLeftDelay(3)+
EarlyLeftDelay(4)+EarlyLeftDelay(5)+EarlyLeftDelay(6)+EarlyLeftDelay(7)+EarlyLef
tDelay(8)+EarlyLeftDelay(9)+EarlyLeftDelay(10)+EarlyLeftDelay(11)+EarlyLeftDelay
(12)];

endingDelayMainComb =
[0;0;0;0;0;endingDelayEarlyLeft(12)+CombDelayMain(6);endingDelayEarlyLeft(12)+Co
mbDelayMain(6)+CombDelayMain(7)];

endingDelayRightComb =
[endingDelayMainComb(7)+CombDelayRight(1);endingDelayMainComb(7)+CombDelayRight(
1)+CombDelayRight(2);0;0;0;0;0];

endingDelayLeftComb =
[endingDelayRightComb(2)+CombDelayLeft(1);endingDelayRightComb(2)+CombDelayLeft(
1)+CombDelayLeft(2);0;0;0;0;0];

endingDelayRightAllpass =
[endingDelayLeftComb(2)+AllpassDelayRight(1);endingDelayLeftComb(2)+AllpassDelay
Right(1)+AllpassDelayRight(2)];
```

```
endingDelayLeftAllpass =
[endingDelayRightAllpass(2)+AllpassDelayLeft(1);endingDelayRightAllpass(2)+Allpa
ssDelayLeft(1)+AllpassDelayLeft(2)];


Pointer_rd_EarlyRight(1)=startingDelayEarlyRight(1)+2;
Pointer_wr_EarlyRight(1)=startingDelayEarlyRight(1)+1;
Pointer_rd_EarlyRight(2)=startingDelayEarlyRight(2)+2;
Pointer_wr_EarlyRight(2)=startingDelayEarlyRight(2)+1;
Pointer_rd_EarlyRight(3)=startingDelayEarlyRight(3)+2;
Pointer_wr_EarlyRight(3)=startingDelayEarlyRight(3)+1;
Pointer_rd_EarlyRight(4)=startingDelayEarlyRight(4)+2;
Pointer_wr_EarlyRight(4)=startingDelayEarlyRight(4)+1;
Pointer_rd_EarlyRight(5)=startingDelayEarlyRight(5)+2;
Pointer_wr_EarlyRight(5)=startingDelayEarlyRight(5)+1;
Pointer_rd_EarlyRight(6)=startingDelayEarlyRight(6)+2;
Pointer_wr_EarlyRight(6)=startingDelayEarlyRight(6)+1;
Pointer_rd_EarlyRight(7)=startingDelayEarlyRight(7)+2;
Pointer_wr_EarlyRight(7)=startingDelayEarlyRight(7)+1;
Pointer_rd_EarlyRight(8)=startingDelayEarlyRight(8)+2;
Pointer_wr_EarlyRight(8)=startingDelayEarlyRight(8)+1;
Pointer_rd_EarlyRight(9)=startingDelayEarlyRight(9)+2;
Pointer_wr_EarlyRight(9)=startingDelayEarlyRight(9)+1;
Pointer_rd_EarlyRight(10)=startingDelayEarlyRight(10)+2;
Pointer_wr_EarlyRight(10)=startingDelayEarlyRight(10)+1;
Pointer_rd_EarlyRight(11)=startingDelayEarlyRight(11)+2;
Pointer_wr_EarlyRight(11)=startingDelayEarlyRight(11)+1;
Pointer_rd_EarlyRight(12)=startingDelayEarlyRight(12)+2;
Pointer_wr_EarlyRight(12)=startingDelayEarlyRight(12)+1;

Pointer_rd_EarlyLeft(1)=startingDelayEarlyLeft(1)+2;
Pointer_wr_EarlyLeft(1)=startingDelayEarlyLeft(1)+1;
Pointer_rd_EarlyLeft(2)=startingDelayEarlyLeft(2)+2;
Pointer_wr_EarlyLeft(2)=startingDelayEarlyLeft(2)+1;
Pointer_rd_EarlyLeft(3)=startingDelayEarlyLeft(3)+2;
Pointer_wr_EarlyLeft(3)=startingDelayEarlyLeft(3)+1;
Pointer_rd_EarlyLeft(4)=startingDelayEarlyLeft(4)+2;
Pointer_wr_EarlyLeft(4)=startingDelayEarlyLeft(4)+1;
Pointer_rd_EarlyLeft(5)=startingDelayEarlyLeft(5)+2;
Pointer_wr_EarlyLeft(5)=startingDelayEarlyLeft(5)+1;
Pointer_rd_EarlyLeft(6)=startingDelayEarlyLeft(6)+2;
Pointer_wr_EarlyLeft(6)=startingDelayEarlyLeft(6)+1;
Pointer_rd_EarlyLeft(7)=startingDelayEarlyLeft(7)+2;
Pointer_wr_EarlyLeft(7)=startingDelayEarlyLeft(7)+1;
Pointer_rd_EarlyLeft(8)=startingDelayEarlyLeft(8)+2;
Pointer_wr_EarlyLeft(8)=startingDelayEarlyLeft(8)+1;
Pointer_rd_EarlyLeft(9)=startingDelayEarlyLeft(9)+2;
Pointer_wr_EarlyLeft(9)=startingDelayEarlyLeft(9)+1;
Pointer_rd_EarlyLeft(10)=startingDelayEarlyLeft(10)+2;
Pointer_wr_EarlyLeft(10)=startingDelayEarlyLeft(10)+1;
Pointer_rd_EarlyLeft(11)=startingDelayEarlyLeft(11)+2;
Pointer_wr_EarlyLeft(11)=startingDelayEarlyLeft(11)+1;
Pointer_rd_EarlyLeft(12)=startingDelayEarlyLeft(12)+2;
Pointer_wr_EarlyLeft(12)=startingDelayEarlyLeft(12)+1;

Pointer_rd_MainComb(6)=startingDelayMainComb(6)+2;
Pointer_wr_MainComb(6)=startingDelayMainComb(6)+1;
Pointer_rd_MainComb(7)=startingDelayMainComb(7)+2;
Pointer_wr_MainComb(7)=startingDelayMainComb(7)+1;
Pointer_rd_RightComb(1)=startingDelayRightComb(1)+2;
Pointer_wr_RightComb(1)=startingDelayRightComb(1)+1;
Pointer_rd_RightComb(2)=startingDelayRightComb(2)+2;
```

```
Pointer_wr_RightComb(2)=startingDelayRightComb(2)+1;
Pointer_rd_LeftComb(1)=startingDelayLeftComb(1)+2;
Pointer_wr_LeftComb(1)=startingDelayLeftComb(1)+1;
Pointer_rd_LeftComb(2)=startingDelayLeftComb(2)+2;
Pointer_wr_LeftComb(2)=startingDelayLeftComb(2)+1;

Pointer_rd_RightAllpass(1)=startingDelayRightAllpass(1)+2;
Pointer_wr_RightAllpass(1)=startingDelayRightAllpass(1)+1;
Pointer_rd_RightAllpass(2)=startingDelayRightAllpass(2)+2;
Pointer_wr_RightAllpass(2)=startingDelayRightAllpass(2)+1;
Pointer_rd_LeftAllpass(1)=startingDelayLeftAllpass(1)+2;
Pointer_wr_LeftAllpass(1)=startingDelayLeftAllpass(1)+1;
Pointer_rd_LeftAllpass(2)=startingDelayLeftAllpass(2)+2;
Pointer_wr_LeftAllpass(2)=startingDelayLeftAllpass(2)+1;


OutEarlyRight1 = zeros(size(x));
OutEarlyRight2 = zeros(size(x));
OutEarlyRight3 = zeros(size(x));
OutEarlyRight4 = zeros(size(x));
OutEarlyRight5 = zeros(size(x));
OutEarlyRight6 = zeros(size(x));
OutEarlyRight7 = zeros(size(x));
OutEarlyRight8 = zeros(size(x));
OutEarlyRight9 = zeros(size(x));
OutEarlyRight10 = zeros(size(x));
OutEarlyRight11 = zeros(size(x));
OutEarlyRight12 = zeros(size(x));
OutEarlyRight = zeros(size(x));

OutEarlyLeft1 = zeros(size(x));
OutEarlyLeft2 = zeros(size(x));
OutEarlyLeft3 = zeros(size(x));
OutEarlyLeft4 = zeros(size(x));
OutEarlyLeft5 = zeros(size(x));
OutEarlyLeft6 = zeros(size(x));
OutEarlyLeft7 = zeros(size(x));
OutEarlyLeft8 = zeros(size(x));
OutEarlyLeft9 = zeros(size(x));
OutEarlyLeft10 = zeros(size(x));
OutEarlyLeft11 = zeros(size(x));
OutEarlyLeft12 = zeros(size(x));
OutEarlyLeft = zeros(size(x));

OutEarlyReflections = zeros(size(x));

OutCombMain6 = zeros(size(x));
OutCombMain7 = zeros(size(x));
OutCombRight1 = zeros(size(x));
OutCombRight2 = zeros(size(x));
OutCombLeft1 = zeros(size(x));
OutCombLeft2 = zeros(size(x));
OutCombMain = zeros(size(x));
OutCombRight = zeros(size(x));
OutCombLeft = zeros(size(x));

OutRComb = zeros(size(x));
OutLComb = zeros(size(x));

OutAllpassRight1 = zeros(size(x));
OutAllpassRight2 = zeros(size(x));
```

```
OutAllpassLeft1 = zeros(size(x));
OutAllpassLeft2 = zeros(size(x));

OutAllpassRight = zeros(size(x));
OutAllpassLeft = zeros(size(x));

OutEarlyR2R = zeros(size(x));
OutEarlyL2R = zeros(size(x));
OutAllpassR2R = zeros(size(x));
OutAllpassL2R = zeros(size(x));
OutAllpassR2L = zeros(size(x));
OutAllpassL2L = zeros(size(x));
OutEarlyR2L = zeros(size(x));
OutEarlyL2L = zeros(size(x));

Output = zeros(size(x));


for n=1:length(x)

[DelayLine,stateEarly,OutEarlyRight1(n),Pointer_rd_EarlyRight(1),Pointer_wr_Earl
yRight(1)] =
EarlyReflectionsRight(x(n),Pointer_rd_EarlyRight(1),Pointer_wr_EarlyRight(1),end
ingDelayEarlyRight(1),startingDelayEarlyRight(1),ER_G1(1),EarlyRightGain(1),ER_G
3(1),DelayLine,stateEarly,1);

[DelayLine,stateEarly,OutEarlyRight2(n),Pointer_rd_EarlyRight(2),Pointer_wr_Earl
yRight(2)] =
EarlyReflectionsRight(x(n),Pointer_rd_EarlyRight(2),Pointer_wr_EarlyRight(2),end
ingDelayEarlyRight(2),startingDelayEarlyRight(2),ER_G1(2),EarlyRightGain(2),ER_G
3(2),DelayLine,stateEarly,2);

[DelayLine,stateEarly,OutEarlyRight3(n),Pointer_rd_EarlyRight(3),Pointer_wr_Earl
yRight(3)] =
EarlyReflectionsRight(x(n),Pointer_rd_EarlyRight(3),Pointer_wr_EarlyRight(3),end
ingDelayEarlyRight(3),startingDelayEarlyRight(3),ER_G1(3),EarlyRightGain(3),ER_G
3(3),DelayLine,stateEarly,3);

[DelayLine,stateEarly,OutEarlyRight4(n),Pointer_rd_EarlyRight(4),Pointer_wr_Earl
yRight(4)] =
EarlyReflectionsRight(x(n),Pointer_rd_EarlyRight(4),Pointer_wr_EarlyRight(4),end
ingDelayEarlyRight(4),startingDelayEarlyRight(4),ER_G1(4),EarlyRightGain(4),ER_G
3(4),DelayLine,stateEarly,4);

[DelayLine,stateEarly,OutEarlyRight5(n),Pointer_rd_EarlyRight(5),Pointer_wr_Earl
yRight(5)] =
EarlyReflectionsRight(x(n),Pointer_rd_EarlyRight(5),Pointer_wr_EarlyRight(5),end
ingDelayEarlyRight(5),startingDelayEarlyRight(5),ER_G1(5),EarlyRightGain(5),ER_G
3(5),DelayLine,stateEarly,5);

[DelayLine,stateEarly,OutEarlyRight6(n),Pointer_rd_EarlyRight(6),Pointer_wr_Earl
yRight(6)] =
EarlyReflectionsRight(x(n),Pointer_rd_EarlyRight(6),Pointer_wr_EarlyRight(6),end
ingDelayEarlyRight(6),startingDelayEarlyRight(6),ER_G1(6),EarlyRightGain(6),ER_G
3(6),DelayLine,stateEarly,6);

[DelayLine,stateEarly,OutEarlyRight7(n),Pointer_rd_EarlyRight(7),Pointer_wr_Earl
yRight(7)] =
EarlyReflectionsRight(x(n),Pointer_rd_EarlyRight(7),Pointer_wr_EarlyRight(7),end
ingDelayEarlyRight(7),startingDelayEarlyRight(7),ER_G1(7),EarlyRightGain(7),ER_G
3(7),DelayLine,stateEarly,7);
```

```
[DelayLine,stateEarly,OutEarlyRight8(n),Pointer_rd_EarlyRight(8),Pointer_wr_Earl
yRight(8)] =
EarlyReflectionsRight(x(n),Pointer_rd_EarlyRight(8),Pointer_wr_EarlyRight(8),end
ingDelayEarlyRight(8),startingDelayEarlyRight(8),ER_G1(8),EarlyRightGain(8),ER_G
3(8),DelayLine,stateEarly,8);

[DelayLine,stateEarly,OutEarlyRight9(n),Pointer_rd_EarlyRight(9),Pointer_wr_Earl
yRight(9)] =
EarlyReflectionsRight(x(n),Pointer_rd_EarlyRight(9),Pointer_wr_EarlyRight(9),end
ingDelayEarlyRight(9),startingDelayEarlyRight(9),ER_G1(9),EarlyRightGain(9),ER_G
3(9),DelayLine,stateEarly,9);

[DelayLine,stateEarly,OutEarlyRight10(n),Pointer_rd_EarlyRight(10),Pointer_wr_Ea
rlyRight(10)] =
EarlyReflectionsRight(x(n),Pointer_rd_EarlyRight(10),Pointer_wr_EarlyRight(10),e
ndingDelayEarlyRight(10),startingDelayEarlyRight(10),ER_G1(10),EarlyRightGain(10
),ER_G3(10),DelayLine,stateEarly,10);

[DelayLine,stateEarly,OutEarlyRight11(n),Pointer_rd_EarlyRight(11),Pointer_wr_Ea
rlyRight(11)] =
EarlyReflectionsRight(x(n),Pointer_rd_EarlyRight(11),Pointer_wr_EarlyRight(11),e
ndingDelayEarlyRight(11),startingDelayEarlyRight(11),ER_G1(11),EarlyRightGain(11
),ER_G3(11),DelayLine,stateEarly,11);

[DelayLine,stateEarly,OutEarlyRight12(n),Pointer_rd_EarlyRight(12),Pointer_wr_Ea
rlyRight(12)] =
EarlyReflectionsRight(x(n),Pointer_rd_EarlyRight(12),Pointer_wr_EarlyRight(12),e
ndingDelayEarlyRight(12),startingDelayEarlyRight(12),ER_G1(12),EarlyRightGain(12
),ER_G3(12),DelayLine,stateEarly,12);


OutEarlyRight(n) = OutEarlyRight1(n) + OutEarlyRight2(n) + OutEarlyRight3(n) +
OutEarlyRight4(n) + OutEarlyRight5(n) + OutEarlyRight6(n) + OutEarlyRight7(n) +
OutEarlyRight8(n) + OutEarlyRight9(n) + OutEarlyRight10(n) + OutEarlyRight11(n)
+ OutEarlyRight12(n);

[DelayLine,stateEarly,OutEarlyLeft1(n),Pointer_rd_EarlyLeft(1),Pointer_wr_EarlyL
eft(1)] = EarlyReflectionsLeft
(x(n),Pointer_rd_EarlyLeft(1),Pointer_wr_EarlyLeft(1),endingDelayEarlyLeft(1),st
artingDelayEarlyLeft(1),EL_G1(1),EarlyLeftGain(1),EL_G3(1),DelayLine,stateEarly,
1);

[DelayLine,stateEarly,OutEarlyLeft2(n),Pointer_rd_EarlyLeft(2),Pointer_wr_EarlyL
eft(2)] = EarlyReflectionsLeft
(x(n),Pointer_rd_EarlyLeft(2),Pointer_wr_EarlyLeft(2),endingDelayEarlyLeft(2),st
artingDelayEarlyLeft(2),EL_G1(2),EarlyLeftGain(2),EL_G3(2),DelayLine,stateEarly,
2);

[DelayLine,stateEarly,OutEarlyLeft3(n),Pointer_rd_EarlyLeft(3),Pointer_wr_EarlyL
eft(3)] = EarlyReflectionsLeft
(x(n),Pointer_rd_EarlyLeft(3),Pointer_wr_EarlyLeft(3),endingDelayEarlyLeft(3),st
artingDelayEarlyLeft(3),EL_G1(3),EarlyLeftGain(3),EL_G3(3),DelayLine,stateEarly,
3);

[DelayLine,stateEarly,OutEarlyLeft4(n),Pointer_rd_EarlyLeft(4),Pointer_wr_EarlyL
eft(4)] = EarlyReflectionsLeft
(x(n),Pointer_rd_EarlyLeft(4),Pointer_wr_EarlyLeft(4),endingDelayEarlyLeft(4),st
artingDelayEarlyLeft(4),EL_G1(4),EarlyLeftGain(4),EL_G3(4),DelayLine,stateEarly,
4);

[DelayLine,stateEarly,OutEarlyLeft5(n),Pointer_rd_EarlyLeft(5),Pointer_wr_EarlyL
eft(5)] = EarlyReflectionsLeft
```

```
(x(n),Pointer_rd_EarlyLeft(5),Pointer_wr_EarlyLeft(5),endingDelayEarlyLeft(5),st
artingDelayEarlyLeft(5),EL_G1(5),EarlyLeftGain(5),EL_G3(5),DelayLine,stateEarly,
5);

[DelayLine,stateEarly,OutEarlyLeft6(n),Pointer_rd_EarlyLeft(6),Pointer_wr_EarlyL
eft(6)] = EarlyReflectionsLeft
(x(n),Pointer_rd_EarlyLeft(6),Pointer_wr_EarlyLeft(6),endingDelayEarlyLeft(6),st
artingDelayEarlyLeft(6),EL_G1(6),EarlyLeftGain(6),EL_G3(6),DelayLine,stateEarly,
6);

[DelayLine,stateEarly,OutEarlyLeft7(n),Pointer_rd_EarlyLeft(7),Pointer_wr_EarlyL
eft(7)] = EarlyReflectionsLeft
(x(n),Pointer_rd_EarlyLeft(7),Pointer_wr_EarlyLeft(7),endingDelayEarlyLeft(7),st
artingDelayEarlyLeft(7),EL_G1(7),EarlyLeftGain(7),EL_G3(7),DelayLine,stateEarly,
7);

[DelayLine,stateEarly,OutEarlyLeft8(n),Pointer_rd_EarlyLeft(8),Pointer_wr_EarlyL
eft(8)] = EarlyReflectionsLeft
(x(n),Pointer_rd_EarlyLeft(8),Pointer_wr_EarlyLeft(8),endingDelayEarlyLeft(8),st
artingDelayEarlyLeft(8),EL_G1(8),EarlyLeftGain(8),EL_G3(8),DelayLine,stateEarly,
8);

[DelayLine,stateEarly,OutEarlyLeft9(n),Pointer_rd_EarlyLeft(9),Pointer_wr_EarlyL
eft(9)] = EarlyReflectionsLeft
(x(n),Pointer_rd_EarlyLeft(9),Pointer_wr_EarlyLeft(9),endingDelayEarlyLeft(9),st
artingDelayEarlyLeft(9),EL_G1(9),EarlyLeftGain(9),EL_G3(9),DelayLine,stateEarly,
9);

[DelayLine,stateEarly,OutEarlyLeft10(n),Pointer_rd_EarlyLeft(10),Pointer_wr_Earl
yLeft(10)] = EarlyReflectionsLeft
(x(n),Pointer_rd_EarlyLeft(10),Pointer_wr_EarlyLeft(10),endingDelayEarlyLeft(10)
,startingDelayEarlyLeft(10),EL_G1(10),EarlyLeftGain(10),EL_G3(10),DelayLine,stat
eEarly,10);

[DelayLine,stateEarly,OutEarlyLeft11(n),Pointer_rd_EarlyLeft(11),Pointer_wr_Earl
yLeft(11)] = EarlyReflectionsLeft
(x(n),Pointer_rd_EarlyLeft(11),Pointer_wr_EarlyLeft(11),endingDelayEarlyLeft(11)
,startingDelayEarlyLeft(11),EL_G1(11),EarlyLeftGain(11),EL_G3(11),DelayLine,stat
eEarly,11);

[DelayLine,stateEarly,OutEarlyLeft12(n),Pointer_rd_EarlyLeft(12),Pointer_wr_Earl
yLeft(12)] =
EarlyReflectionsLeft(x(n),Pointer_rd_EarlyLeft(12),Pointer_wr_EarlyLeft(12),endi
ngDelayEarlyLeft(12),startingDelayEarlyLeft(12),EL_G1(12),EarlyLeftGain(12),EL_G
3(12),DelayLine,stateEarly,12);

         OutEarlyLeft(n) = OutEarlyLeft1(n) + OutEarlyLeft2(n) +
OutEarlyLeft3(n) + OutEarlyLeft4(n) + OutEarlyLeft5(n) + OutEarlyLeft6(n) +
OutEarlyLeft7(n) + OutEarlyLeft8(n) + OutEarlyLeft9(n) + OutEarlyLeft10(n) +
OutEarlyLeft11(n) + OutEarlyLeft12(n);


OutEarlyReflections(n) = 0.7*(OutEarlyRight(n)) + 0.7*(OutEarlyLeft(n));

[DelayLine,state,OutCombMain6(n),Pointer_rd_MainComb(6),Pointer_wr_MainComb(6)]
=
CombFilter(OutEarlyReflections(n),endingDelayMainComb(6),startingDelayMainComb(6
),MainG1(6),MainG2(6),MainG3(6),Pointer_rd_MainComb(6),Pointer_wr_MainComb(6),De
layLine,state,6);

[DelayLine,state,OutCombMain7(n),Pointer_rd_MainComb(7),Pointer_wr_MainComb(7)]
=
```

```
CombFilter(OutEarlyReflections(n),endingDelayMainComb(7),startingDelayMainComb(7
),MainG1(7),MainG2(7),MainG3(7),Pointer_rd_MainComb(7),Pointer_wr_MainComb(7),De
layLine,state,7);


      OutCombMain(n) = OutCombMain6(n) + OutCombMain7(n);


[DelayLine,state,OutCombRight1(n),Pointer_rd_RightComb(1),Pointer_wr_RightComb(1
)] =
CombFilter(OutEarlyReflections(n),endingDelayRightComb(1),startingDelayRightComb
(1),RightG1(1),RightG2(1),RightG3(1),Pointer_rd_RightComb(1),Pointer_wr_RightCom
b(1),DelayLine,state,1);

[DelayLine,state,OutCombRight2(n),Pointer_rd_RightComb(2),Pointer_wr_RightComb(2
)] =
CombFilter(OutEarlyReflections(n),endingDelayRightComb(2),startingDelayRightComb
(2),RightG1(2),RightG2(2),RightG3(2),Pointer_rd_RightComb(2),Pointer_wr_RightCom
b(2),DelayLine,state,2);

      OutCombRight(n) = OutCombRight1(n) + OutCombRight2(n);


[DelayLine,state,OutCombLeft1(n),Pointer_rd_LeftComb(1),Pointer_wr_LeftComb(1)]
=
CombFilter(OutEarlyReflections(n),endingDelayLeftComb(1),startingDelayLeftComb(1
),LeftG1(1),LeftG2(1),LeftG3(1),Pointer_rd_LeftComb(1),Pointer_wr_LeftComb(1),De
layLine,state,1);

[DelayLine,state,OutCombLeft2(n),Pointer_rd_LeftComb(2),Pointer_wr_LeftComb(2)]
=
CombFilter(OutEarlyReflections(n),endingDelayLeftComb(2),startingDelayLeftComb(2
),LeftG1(2),LeftG2(2),LeftG3(2),Pointer_rd_LeftComb(2),Pointer_wr_LeftComb(2),De
layLine,state,2);

      OutCombLeft(n) = OutCombLeft1(n) + OutCombLeft2(n);


OutRComb(n) = OutCombMain(n) + OutCombRight(n);

OutLComb(n) = OutCombMain(n) + OutCombLeft(n);

[DelayLine,OutAllpassRight1(n),Pointer_rd_RightAllpass(1),Pointer_wr_RightAllpas
s(1)] =
AllpassFilter(OutRComb(n),endingDelayRightAllpass(1),startingDelayRightAllpass(1
),AllpassGainRight(1),Pointer_rd_RightAllpass(1),Pointer_wr_RightAllpass(1),Dela
yLine);

[DelayLine,OutAllpassLeft1(n),Pointer_rd_LeftAllpass(1),Pointer_wr_LeftAllpass(1
)] =
AllpassFilter(OutLComb(n),endingDelayLeftAllpass(1),startingDelayLeftAllpass(1),
AllpassGainLeft(1),Pointer_rd_LeftAllpass(1),Pointer_wr_LeftAllpass(1),DelayLine
);

[DelayLine,OutAllpassRight2(n),Pointer_rd_RightAllpass(2),Pointer_wr_RightAllpas
s(2)] =
AllpassFilter(OutAllpassRight1(n),endingDelayRightAllpass(2),startingDelayRightA
llpass(2),AllpassGainRight(2),Pointer_rd_RightAllpass(2),Pointer_wr_RightAllpass
(2),DelayLine);

[DelayLine,OutAllpassLeft2(n),Pointer_rd_LeftAllpass(2),Pointer_wr_LeftAllpass(2
)] =
```

```
AllpassFilter(OutAllpassLeft1(n),endingDelayLeftAllpass(2),startingDelayLeftAllp
ass(2),AllpassGainLeft(2),Pointer_rd_LeftAllpass(2),Pointer_wr_LeftAllpass(2),De
layLine);

OutAllpassRight(n) = OutAllpassRight2(n);
      OutAllpassLeft(n) = OutAllpassLeft2(n);


OutEarlyR2R(n) = OutEarlyRight(n)*EarlyRightToRight;
OutEarlyL2R(n) = OutEarlyLeft(n)*EarlyLeftToRight;
OutAllpassR2R(n) = OutAllpassRight(n)*AllpassRightToRight;
OutAllpassL2R(n) = OutAllpassLeft(n)*AllpassLeftToRight;
OutAllpassR2L(n) = OutAllpassRight(n)*AllpassRightToLeft;
OutAllpassL2L(n) = OutAllpassLeft(n)*AllpassLeftToLeft;
OutEarlyR2L(n) = OutEarlyRight(n)*EarlyRightToLeft;
OutEarlyL2L(n) = OutEarlyLeft(n)*EarlyLeftToLeft;


Output(n) = OutEarlyR2R(n) + OutEarlyL2R(n) + OutAllpassR2R(n) +
OutAllpassL2R(n) + OutAllpassR2L(n) + OutAllpassL2L(n) + OutEarlyR2L(n) +
OutEarlyL2L(n);
```

C - Codice XTime Composer

audio_effects.h

```
#ifndef _AUDIO_EFFECTS_H_
#define _AUDIO_EFFECTS_H_

#include <stddef.h>

#define Size_delay_rd1   20000

typedef struct{
    int OutputComb_rd1;
    unsigned Pointer_CurrentComb_rd1;
    unsigned Pointer_Comb_rd1;
    unsigned SizeComb_rd1;
    int KFeedbackComb;
    int KDump1;
    int KDump2;
    int StateDump;

    int KOutputComb;
    int InputComb;
    int  OutputComb_wr1;
    unsigned Pointer_CurrentComb_wr1;
} T_comb_Ram;

extern T_comb_Ram CombPtr;

typedef struct{
    unsigned Pointer_CurrentAllpass_rd1;
    unsigned Pointer_CurrentAllpass_wr1;
    unsigned Pointer_Allpass_rd1;
    unsigned SizeAllpass_rd1;
    int KFeedbackAllpass;
    int StateAllpass;
    int KOutputAllpass;
    int InputAllpass;

    int     OutputAllpass_rd1;
    int     OutputAllpass_wr1;
} T_allpass_Ram;

extern T_allpass_Ram  AllpassPtr;

typedef struct{
      unsigned Pointer_Twentyfour;
      unsigned SizeTwentyfour;
      unsigned Pointer_CurrentTwentyfour1_rd1;
      unsigned Pointer_CurrentTwentyfour2_rd1;
      unsigned Pointer_CurrentTwentyfour3_rd1;
      unsigned Pointer_CurrentTwentyfour4_rd1;
      unsigned Pointer_CurrentTwentyfour5_rd1;
```

```
unsigned Pointer_CurrentTwentyfour6_rd1;

unsigned Pointer_CurrentTwentyfour7_rd1;
unsigned Pointer_CurrentTwentyfour8_rd1;
unsigned Pointer_CurrentTwentyfour9_rd1;
unsigned Pointer_CurrentTwentyfour10_rd1;

unsigned Pointer_CurrentTwentyfour11_rd1;
unsigned Pointer_CurrentTwentyfour12_rd1;
unsigned Pointer_CurrentTwentyfour13_rd1;
unsigned Pointer_CurrentTwentyfour14_rd1;
unsigned Pointer_CurrentTwentyfour15_rd1;
unsigned Pointer_CurrentTwentyfour16_rd1;
unsigned Pointer_CurrentTwentyfour17_rd1;
unsigned Pointer_CurrentTwentyfour18_rd1;

unsigned Pointer_CurrentTwentyfour19_rd1;
unsigned Pointer_CurrentTwentyfour20_rd1;
unsigned Pointer_CurrentTwentyfour21_rd1;
unsigned Pointer_CurrentTwentyfour22_rd1;

unsigned Pointer_CurrentTwentyfour23_rd1;
unsigned Pointer_CurrentTwentyfour24_rd1;
unsigned Pointer_CurrentTwentyfour_wr1;
int InputTwentyfour;
int K1Twentyfour;
int State1Twentyfour;
int K2Twentyfour;
int State2Twentyfour;

int K3Twentyfour;
int State3Twentyfour;
int K4Twentyfour;
int State4Twentyfour;

int K5Twentyfour;
int State5Twentyfour;
int K6Twentyfour;
int State6Twentyfour;
int K7Twentyfour;
int State7Twentyfour;
int K8Twentyfour;
int State8Twentyfour;

int K9Twentyfour;
int State9Twentyfour;
int K10Twentyfour;
int State10Twentyfour;

int K11Twentyfour;
int State11Twentyfour;
int K12Twentyfour;
int State12Twentyfour;
int K13Twentyfour;
int State13Twentyfour;
int K14Twentyfour;
```

```
        int State14Twentyfour;

        int K15Twentyfour;
        int State15Twentyfour;
        int K16Twentyfour;
        int State16Twentyfour;

        int K17Twentyfour;
        int State17Twentyfour;
        int K18Twentyfour;
        int State18Twentyfour;
        int K19Twentyfour;
        int State19Twentyfour;
        int K20Twentyfour;
        int State20Twentyfour;

        int K21Twentyfour;
        int State21Twentyfour;
        int K22Twentyfour;
        int State22Twentyfour;

        int K23Twentyfour;
        int State23Twentyfour;
        int K24Twentyfour;
        int State24Twentyfour;
        int     OutputTwentyfour_rd1;
        int     OutputTwentyfour_wr1;

} T_twentyfour_Ram;

extern T_twentyfour_Ram  TwentyfourPtr;

typedef struct
 {

 //   int Dummy;
    int InputBq;
    int ka0;
    int ka1;
    int ka2;
    int kb1;
    int kb2;
    int sta1;
    int sta2;
    int stb1;
    int stb2;
    int OutputBq;
    int NumBq;

 } T__bquad_Ram;

 extern T__bquad_Ram FilterDelay;

 typedef struct{

 //    int InputVolume;
```

```
    T__bquad_Ram Biquad1;
    T_twentyfour_Ram EarlyReflectionsRight;
    T_twentyfour_Ram EarlyReflectionsLeft;
    T__bquad_Ram Biquad2;
    T_comb_Ram CombMain1;
    T_comb_Ram CombMain2;
    T_comb_Ram CombMain3;
    T_comb_Ram CombMain4;
    T_comb_Ram CombMain5;
    T_comb_Ram CombMain6;
    T_comb_Ram CombMain7;
    T_comb_Ram CombRight1;
    T_comb_Ram CombRight2;
    T_comb_Ram CombLeft1;
    T_comb_Ram CombLeft2;
    T_allpass_Ram AllpassRight1;
    T_allpass_Ram AllpassRight2;
    T_allpass_Ram AllpassLeft1;
    T_allpass_Ram AllpassLeft2;
    int EarlyRightVol;
    int EarlyLeftVol;
    int KEarlyRightToRight;
    int KEarlyLeftToRight;
    int KRevRightToRight;
    int KRevLeftToRight;
    int KRevRightToLeft;
    int KRevLeftToLeft;
    int KEarlyRightToLeft;
    int KEarlyLeftToLeft;
    int Ambience2OutputRight;
    int Ambience2OutputLeft;

} T_rev_Ram;

 extern T_rev_Ram  RevPtr;

typedef struct
{
    int     delta_delay_rd1;
    unsigned Pointer_Currentdelay_rd1;
    unsigned   Pointer_delay_rd1;

    unsigned Sizedelay_rd1;
    int     kfeedB_delay_rd1;
    int     k1dump_delay_rd1;
    int     k2dump_delay_rd1;
    int     stdump_delay_rd1;

    int     delta_delay_wr1;
    unsigned Pointer_Currentdelay_wr1;
    unsigned input_delay;
    int     K1to2out;

    T__bquad_Ram FilterDelay;
```

```
    unsigned Pointer_delay_rd2;
    unsigned Pointer_delta_delay_rd2;
    unsigned Pointer_delay_wr;

    int kfeedB_delay_rd2;
    int k1dump_delay_rd2;
    int k2dump_delay_rd2;
    int stdump_delay_rd2;



} T__Pointer_Ram;

extern T__Pointer_Ram Pointer_ptr;

void audio_effects(streaming chanend c_dsp_eq,
                   chanend c_gain,
                   static const size_t num_chans);

#endif
```

audio_effect.xC

```
#include <audio_effects.h>
#include <stdint.h>
#include <stddef.h>
#include <debug_print.h>
#include <xscope.h>
extern void asm_effect(void);
#define LEFT_IN 0
#define LEFT_OUT 1

short Delay_1[44000];
T__Pointer_Ram Pointer_ptr;
int32_t in_samps[4];
T_comb_Ram CombPtr;
T_allpass_Ram AllpassPtr;
T_twentyfour_Ram TwentyfourPtr;
T_rev_Ram  RevPtr;
T__bquad_Ram FilterDelay;


// Sample buffer to hold samples after processing
int32_t out_samps[4];
int prova = 0; // Start with volume at 0
int k;
void audio_effects(streaming chanend c_dsp,
                   //client startkit_led_if i_led,
                   //client startkit_button_if i_button,
                   chanend c_gain,
                   static const size_t num_chans)
{
  // Unprocessed input audio sample buffer
```

```
  //int32_t in_samps[num_chans] = {0};

  // Sample buffer to hold samples after processing
  //int32_t out_samps[num_chans] = {0};

    int32_t gain = 0; // Start with volume at 0

// ---------------------------------------- init Struct effect
  Pointer_ptr.Pointer_delay_wr=1;
  Pointer_ptr.Sizedelay_rd1=sizeof(Delay_1)/2;
  Pointer_ptr.Pointer_Currentdelay_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
  Pointer_ptr.Pointer_delay_rd1=((unsigned) &Delay_1);
  Pointer_ptr.kfeedB_delay_rd1=0x3FFFFFF;
  Pointer_ptr.k1dump_delay_rd1=0x7fd4397;
  Pointer_ptr.k2dump_delay_rd1=179384;
  Pointer_ptr.Pointer_Currentdelay_wr1=0;
  Pointer_ptr.delta_delay_wr1=0;
  Pointer_ptr.K1to2out=0x07FFFFFF;
  Pointer_ptr.delta_delay_rd1=0x7Ff8000;
  Delay_1[0]=0x7FFF;
  Delay_1[1]=0x7FFF;
  Delay_1[7]=0x7FFF;
  Delay_1[8]=0x7FFF;


  Pointer_ptr.FilterDelay.NumBq=1;
  Pointer_ptr.FilterDelay.ka0=1;
  Pointer_ptr.FilterDelay.ka1=2;
  Pointer_ptr.FilterDelay.ka2=3;
  Pointer_ptr.FilterDelay.kb1=4;
  Pointer_ptr.FilterDelay.kb2=5;

  Pointer_ptr.FilterDelay.sta1=1;
  Pointer_ptr.FilterDelay.sta2=2;
  Pointer_ptr.FilterDelay.stb1=3;
  Pointer_ptr.FilterDelay.stb2=4;

  CombPtr.OutputComb_rd1=0x7Ff8000;
  CombPtr.Pointer_CurrentComb_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
  CombPtr.Pointer_Comb_rd1=((unsigned) &Delay_1);
  CombPtr.SizeComb_rd1=sizeof(Delay_1)/2;
  CombPtr.KFeedbackComb=0x3FFFFFF;
  CombPtr.KOutputComb=0x3FFFFFF;
  CombPtr.KDump1=0x7fd4397;
  CombPtr.KDump2=179384;

  CombPtr.OutputComb_wr1=0;
  CombPtr.Pointer_CurrentComb_wr1=0;

  AllpassPtr.OutputAllpass_rd1=0x7Ff8000;
  AllpassPtr.Pointer_CurrentAllpass_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
  AllpassPtr.Pointer_Allpass_rd1=((unsigned) &Delay_1);
  AllpassPtr.SizeAllpass_rd1=sizeof(Delay_1)/2;
  AllpassPtr.KFeedbackAllpass=0x3FFFFFF;
  AllpassPtr.KOutputAllpass=0x3FFFFFF;
```

```
TwentyfourPtr.Pointer_Twentyfour=((unsigned)&Delay_1);
TwentyfourPtr.SizeTwentyfour=sizeof(Delay_1)/2;
TwentyfourPtr.Pointer_CurrentTwentyfour1_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour2_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour3_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour4_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour5_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour6_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour7_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour8_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour9_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour10_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour11_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour12_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour13_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour14_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour15_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour16_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour17_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour18_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour19_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour20_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour21_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour22_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour23_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.Pointer_CurrentTwentyfour24_rd1=sizeof(Delay_1)/2-Size_delay_rd1;
TwentyfourPtr.K1Twentyfour=10;
TwentyfourPtr.K2Twentyfour=20;
TwentyfourPtr.K3Twentyfour=30;
TwentyfourPtr.K4Twentyfour=40;
TwentyfourPtr.K5Twentyfour=50;
TwentyfourPtr.K6Twentyfour=60;
TwentyfourPtr.K7Twentyfour=70;
TwentyfourPtr.K8Twentyfour=80;
TwentyfourPtr.K9Twentyfour=90;
TwentyfourPtr.K10Twentyfour=100;
TwentyfourPtr.K11Twentyfour=110;
TwentyfourPtr.K12Twentyfour=120;
TwentyfourPtr.K13Twentyfour=130;
TwentyfourPtr.K14Twentyfour=140;
TwentyfourPtr.K15Twentyfour=150;
TwentyfourPtr.K16Twentyfour=160;
TwentyfourPtr.K17Twentyfour=170;
TwentyfourPtr.K18Twentyfour=180;
TwentyfourPtr.K19Twentyfour=190;
TwentyfourPtr.K20Twentyfour=200;
TwentyfourPtr.K21Twentyfour=210;
TwentyfourPtr.K22Twentyfour=220;
TwentyfourPtr.K23Twentyfour=230;
TwentyfourPtr.K24Twentyfour=240;
TwentyfourPtr.OutputTwentyfour_rd1=0x7Ff8000;
TwentyfourPtr.OutputTwentyfour_wr1=0;

RevPtr.Biquad1.InputBq=0xb333333;
RevPtr.Biquad1.ka0=0x1;
RevPtr.Biquad1.ka1=0;
```

```
  RevPtr.Biquad1.ka2=0;
  RevPtr.Biquad1.kb1=0;
  RevPtr.Biquad1.kb2=0;
  RevPtr.Biquad1.sta1=0;
  RevPtr.Biquad1.sta2=0;
  RevPtr.Biquad1.stb1=0;
  RevPtr.Biquad1.stb2=0;
  RevPtr.Biquad1.OutputBq=0x0;
  RevPtr.Biquad1.NumBq=1;
  RevPtr.EarlyReflectionsRight.Pointer_Twentyfour=((unsigned)&Delay_1);
  RevPtr.EarlyReflectionsRight.SizeTwentyfour=13667;
  RevPtr.EarlyReflectionsRight.Pointer_CurrentTwentyfour1_rd1=1;
  RevPtr.EarlyReflectionsRight.Pointer_CurrentTwentyfour_wr1=0;
  RevPtr.EarlyReflectionsRight.K1Twentyfour=0xfffffffff199999a;
  RevPtr.EarlyReflectionsRight.Pointer_CurrentTwentyfour2_rd1=(111+1);
  RevPtr.EarlyReflectionsRight.K2Twentyfour=0xe666666;
  RevPtr.EarlyReflectionsRight.Pointer_CurrentTwentyfour3_rd1=(111+237+1);
  RevPtr.EarlyReflectionsRight.K3Twentyfour=0xfffffffff199999a;
  RevPtr.EarlyReflectionsRight.Pointer_CurrentTwentyfour4_rd1=(111+237+411+1);
  RevPtr.EarlyReflectionsRight.K4Twentyfour=0xb333333;
  RevPtr.EarlyReflectionsRight.Pointer_CurrentTwentyfour5_rd1=(111+237+411+609+1);
  RevPtr.EarlyReflectionsRight.K5Twentyfour=0xfffffffff3333333;
  RevPtr.EarlyReflectionsRight.Pointer_CurrentTwentyfour6_rd1=(111+237+411+609+877+1);
  RevPtr.EarlyReflectionsRight.K6Twentyfour=0xccccccd;

RevPtr.EarlyReflectionsRight.Pointer_CurrentTwentyfour7_rd1=(111+237+411+609+877+1011+1)
;
  RevPtr.EarlyReflectionsRight.K7Twentyfour=0xfffffffee666666;

RevPtr.EarlyReflectionsRight.Pointer_CurrentTwentyfour8_rd1=(111+237+411+609+877+1011+1
234+1);
  RevPtr.EarlyReflectionsRight.K8Twentyfour=0x6666666;

RevPtr.EarlyReflectionsRight.Pointer_CurrentTwentyfour9_rd1=(111+237+411+609+877+1011+1
234+1431+1);
  RevPtr.EarlyReflectionsRight.K9Twentyfour=0xfffffffff6666666;

RevPtr.EarlyReflectionsRight.Pointer_CurrentTwentyfour10_rd1=(111+237+411+609+877+1011+
1234+1431+1679+1);
  RevPtr.EarlyReflectionsRight.K10Twentyfour=0x999999a;

RevPtr.EarlyReflectionsRight.Pointer_CurrentTwentyfour11_rd1=(111+237+411+609+877+1011+
1234+1431+1679+1845+1);
  RevPtr.EarlyReflectionsRight.K11Twentyfour=0x6666666;

RevPtr.EarlyReflectionsRight.Pointer_CurrentTwentyfour12_rd1=(111+237+411+609+877+1011+
1234+1431+1679+1845+2001+1);
  RevPtr.EarlyReflectionsRight.K12Twentyfour=0xfffffffffb333336;
  RevPtr.EarlyReflectionsRight.OutputTwentyfour_rd1=0xb333333;
  RevPtr.EarlyReflectionsRight.OutputTwentyfour_wr1=0;
  RevPtr.EarlyRightVol=0x7ffffff;
  RevPtr.EarlyReflectionsLeft.Pointer_Twentyfour=((unsigned)&Delay_1);
  RevPtr.EarlyReflectionsLeft.SizeTwentyfour=10450;
  RevPtr.EarlyReflectionsLeft.Pointer_CurrentTwentyfour1_rd1=(13667+1);
  RevPtr.EarlyReflectionsLeft.Pointer_CurrentTwentyfour_wr1=13667;
  RevPtr.EarlyReflectionsLeft.K1Twentyfour=0xe666666;
```

```
  RevPtr.EarlyReflectionsLeft.Pointer_CurrentTwentyfour2_rd1=(13667+32+1);
  RevPtr.EarlyReflectionsLeft.K2Twentyfour=0xfffffffff199999a;
  RevPtr.EarlyReflectionsLeft.Pointer_CurrentTwentyfour3_rd1=(13667+32+65+1);
  RevPtr.EarlyReflectionsLeft.K3Twentyfour=0xccccccd;
  RevPtr.EarlyReflectionsLeft.Pointer_CurrentTwentyfour4_rd1=(13667+32+65+131+1);
  RevPtr.EarlyReflectionsLeft.K4Twentyfour=0x6666666;
  RevPtr.EarlyReflectionsLeft.Pointer_CurrentTwentyfour5_rd1=(13667+32+65+131+353+1);
  RevPtr.EarlyReflectionsLeft.K5Twentyfour=0xfffffffff8000000;
  RevPtr.EarlyReflectionsLeft.Pointer_CurrentTwentyfour6_rd1=(13667+32+65+131+353+531+1);
  RevPtr.EarlyReflectionsLeft.K6Twentyfour=0xfffffffff4ccccd;

RevPtr.EarlyReflectionsLeft.Pointer_CurrentTwentyfour7_rd1=(13667+32+65+131+353+531+752+
1);
  RevPtr.EarlyReflectionsLeft.K7Twentyfour=0xfffffffff6666666a;

RevPtr.EarlyReflectionsLeft.Pointer_CurrentTwentyfour8_rd1=(13667+32+65+131+353+531+752+
971+1);
  RevPtr.EarlyReflectionsLeft.K8Twentyfour=0xb333333;

RevPtr.EarlyReflectionsLeft.Pointer_CurrentTwentyfour9_rd1=(13667+32+65+131+353+531+752+
971+1111+1);
  RevPtr.EarlyReflectionsLeft.K9Twentyfour=0xccccccd;

RevPtr.EarlyReflectionsLeft.Pointer_CurrentTwentyfour10_rd1=(13667+32+65+131+353+531+752
+971+1111+1321+1);
  RevPtr.EarlyReflectionsLeft.K10Twentyfour=0xfffffffff199999a;

RevPtr.EarlyReflectionsLeft.Pointer_CurrentTwentyfour11_rd1=(13667+32+65+131+353+531+752
+971+1111+1321+1541+1);
  RevPtr.EarlyReflectionsLeft.K11Twentyfour=0x8000000;

RevPtr.EarlyReflectionsLeft.Pointer_CurrentTwentyfour12_rd1=(13667+32+65+131+353+531+752
+971+1111+1321+1541+1731+1);
  RevPtr.EarlyReflectionsLeft.K12Twentyfour=0xfffffffff8000000;
  RevPtr.EarlyReflectionsLeft.OutputTwentyfour_rd1=0xb333333;
  RevPtr.EarlyReflectionsLeft.OutputTwentyfour_wr1=0;
  RevPtr.EarlyLeftVol=0x7ffffff;
  RevPtr.Biquad2.ka0=0x1;
  RevPtr.Biquad2.ka1=0;
  RevPtr.Biquad2.ka2=0;
  RevPtr.Biquad2.kb1=0;
  RevPtr.Biquad2.kb2=0;
  RevPtr.Biquad2.sta1=0;
  RevPtr.Biquad2.sta2=0;
  RevPtr.Biquad2.stb1=0;
  RevPtr.Biquad2.stb2=0;
  RevPtr.Biquad2.OutputBq=0x0;
  RevPtr.Biquad2.NumBq=1;
  RevPtr.CombMain1.OutputComb_rd1=0x7Ff8000;
  RevPtr.CombMain1.Pointer_CurrentComb_rd1=43100;
  RevPtr.CombMain1.Pointer_Comb_rd1=((unsigned) &Delay_1);
  RevPtr.CombMain1.SizeComb_rd1=1;
  RevPtr.CombMain1.KFeedbackComb=0x3FFFFFF;
  RevPtr.CombMain1.KDump1=0x33328b7;
  RevPtr.CombMain1.KDump2=0xccca2db;
  RevPtr.CombMain1.KOutputComb=0x3afb8;
```

```
RevPtr.CombMain1.OutputComb_wr1=0;
RevPtr.CombMain1.Pointer_CurrentComb_wr1=43050;
RevPtr.CombMain1.InputComb=0;
RevPtr.CombMain2.OutputComb_rd1=0x7Ff8000;
RevPtr.CombMain2.Pointer_CurrentComb_rd1=43200;
RevPtr.CombMain2.Pointer_Comb_rd1=((unsigned) &Delay_1);
RevPtr.CombMain2.SizeComb_rd1=1;
RevPtr.CombMain2.KFeedbackComb=0x3FFFFFF;
RevPtr.CombMain2.KDump1=0x33328b7;
RevPtr.CombMain2.KDump2=0xccca2db;
RevPtr.CombMain2.KOutputComb=0x68db;
RevPtr.CombMain2.OutputComb_wr1=0;
RevPtr.CombMain2.Pointer_CurrentComb_wr1=43150;
RevPtr.CombMain2.InputComb=0;
RevPtr.CombMain3.OutputComb_rd1=0x7Ff8000;
RevPtr.CombMain3.Pointer_CurrentComb_rd1=43300;
RevPtr.CombMain3.Pointer_Comb_rd1=((unsigned) &Delay_1);
RevPtr.CombMain3.SizeComb_rd1=1;
RevPtr.CombMain3.KFeedbackComb=0x3FFFFFF;
RevPtr.CombMain3.KDump1=0x33328b7;
RevPtr.CombMain3.KDump2=0xccca2db;
RevPtr.CombMain3.KOutputComb=0x68db;
RevPtr.CombMain3.OutputComb_wr1=0;
RevPtr.CombMain3.Pointer_CurrentComb_wr1=43250;
RevPtr.CombMain3.InputComb=0;
RevPtr.CombMain4.OutputComb_rd1=0x7Ff8000;
RevPtr.CombMain4.Pointer_CurrentComb_rd1=43400;
RevPtr.CombMain4.Pointer_Comb_rd1=((unsigned) &Delay_1);
RevPtr.CombMain4.SizeComb_rd1=1;
RevPtr.CombMain4.KFeedbackComb=0x3FFFFFF;
RevPtr.CombMain4.KDump1=0x33328b7;
RevPtr.CombMain4.KDump2=0xccca2db;
RevPtr.CombMain4.KOutputComb=0xa7c;
RevPtr.CombMain4.OutputComb_wr1=0;
RevPtr.CombMain4.Pointer_CurrentComb_wr1=43350;
RevPtr.CombMain4.InputComb=0;
RevPtr.CombMain5.OutputComb_rd1=0x7Ff8000;
RevPtr.CombMain5.Pointer_CurrentComb_rd1=43500;
RevPtr.CombMain5.Pointer_Comb_rd1=((unsigned) &Delay_1);
RevPtr.CombMain5.SizeComb_rd1=1;
RevPtr.CombMain5.KFeedbackComb=0x3FFFFFF;
RevPtr.CombMain5.KDump1=0x33328b7;
RevPtr.CombMain5.KDump2=0xccca2db;
RevPtr.CombMain5.KOutputComb=0xa7c;
RevPtr.CombMain5.OutputComb_wr1=0;
RevPtr.CombMain5.Pointer_CurrentComb_wr1=43450;
RevPtr.CombMain5.InputComb=0;
RevPtr.CombMain6.OutputComb_rd1=0x7Ff8000;
RevPtr.CombMain6.Pointer_CurrentComb_rd1=(24117+1);
RevPtr.CombMain6.Pointer_Comb_rd1=((unsigned) &Delay_1);
RevPtr.CombMain6.SizeComb_rd1=3697;
RevPtr.CombMain6.KFeedbackComb=0x3FFFFFF;
RevPtr.CombMain6.KDump1=0x29f2baa;
RevPtr.CombMain6.KDump2=0xa7cb924;
RevPtr.CombMain6.KOutputComb=0xa7c;
RevPtr.CombMain6.OutputComb_wr1=0;
```

```
RevPtr.CombMain6.Pointer_CurrentComb_wr1=24117;
RevPtr.CombMain6.InputComb=0;
RevPtr.CombMain7.OutputComb_rd1=0x7Ff8000;
RevPtr.CombMain7.Pointer_CurrentComb_rd1=(24117+3697+1);
RevPtr.CombMain7.Pointer_Comb_rd1=((unsigned) &Delay_1);
RevPtr.CombMain7.SizeComb_rd1=3921;
RevPtr.CombMain7.KFeedbackComb=0x3FFFFFF;
RevPtr.CombMain7.KDump1=0xac94;
RevPtr.CombMain7.KDump2=0xa5c7cd9;
RevPtr.CombMain7.KOutputComb=0x68db;
RevPtr.CombMain7.OutputComb_wr1=0;
RevPtr.CombMain7.Pointer_CurrentComb_wr1=(24117+3697);
RevPtr.CombMain7.InputComb=0;
RevPtr.CombRight1.OutputComb_rd1=0x7Ff8000;
RevPtr.CombRight1.Pointer_CurrentComb_rd1=(31735+1);
RevPtr.CombRight1.Pointer_Comb_rd1=((unsigned) &Delay_1);
RevPtr.CombRight1.SizeComb_rd1=1581;
RevPtr.CombRight1.KFeedbackComb=0x3FFFFFF;
RevPtr.CombRight1.KDump1=0x2f04578;
RevPtr.CombRight1.KDump2=0xbc115e0;
RevPtr.CombRight1.KOutputComb=0x8000000;
RevPtr.CombRight1.OutputComb_wr1=0;
RevPtr.CombRight1.Pointer_CurrentComb_wr1=31735;
RevPtr.CombRight1.InputComb=0;
RevPtr.CombRight2.OutputComb_rd1=0x7Ff8000;
RevPtr.CombRight2.Pointer_CurrentComb_rd1=(31735+1581+1);
RevPtr.CombRight2.Pointer_Comb_rd1=((unsigned) &Delay_1);
RevPtr.CombRight2.SizeComb_rd1=1921;
RevPtr.CombRight2.KFeedbackComb=0x3FFFFFF;
RevPtr.CombRight2.KDump1=0x2e16723;
RevPtr.CombRight2.KDump2=0xb8a71de;
RevPtr.CombRight2.KOutputComb=0x8000000;
RevPtr.CombRight2.OutputComb_wr1=(31735+1581);
RevPtr.CombRight2.Pointer_CurrentComb_wr1=0;
RevPtr.CombRight2.InputComb=0;
RevPtr.CombLeft1.OutputComb_rd1=0x7Ff8000;
RevPtr.CombLeft1.Pointer_CurrentComb_rd1=(35237+1);
RevPtr.CombLeft1.Pointer_Comb_rd1=((unsigned) &Delay_1);
RevPtr.CombLeft1.SizeComb_rd1=1811;
RevPtr.CombLeft1.KFeedbackComb=0x3FFFFFF;
RevPtr.CombLeft1.KDump1=0x2e703b0;
RevPtr.CombLeft1.KDump2=0xb9c0442;
RevPtr.CombLeft1.KOutputComb=0x19931fc;
RevPtr.CombLeft1.OutputComb_wr1=35237;
RevPtr.CombLeft1.Pointer_CurrentComb_wr1=0;
RevPtr.CombLeft1.InputComb=0;
RevPtr.CombLeft2.OutputComb_rd1=0x7Ff8000;
RevPtr.CombLeft2.Pointer_CurrentComb_rd1=(35237+1811+1);
RevPtr.CombLeft2.Pointer_Comb_rd1=((unsigned) &Delay_1);
RevPtr.CombLeft2.SizeComb_rd1=1771;
RevPtr.CombLeft2.KFeedbackComb=0x3FFFFFF;
RevPtr.CombLeft2.KDump1=0x2e89ca1;
RevPtr.CombLeft2.KDump2=0xba27286;
RevPtr.CombLeft2.KOutputComb=0xffbe76c;
RevPtr.CombLeft2.OutputComb_wr1=0;
RevPtr.CombLeft2.Pointer_CurrentComb_wr1=(35237+1811);
```

```
RevPtr.CombLeft2.InputComb=0;
RevPtr.AllpassRight1.Pointer_CurrentAllpass_rd1=(38819+1);
RevPtr.AllpassRight1.Pointer_CurrentAllpass_wr1=38819;
RevPtr.AllpassRight1.Pointer_Allpass_rd1=((unsigned) &Delay_1);
RevPtr.AllpassRight1.SizeAllpass_rd1=2057;
RevPtr.AllpassRight1.KFeedbackAllpass=0xb333333;
RevPtr.AllpassRight1.KOutputAllpass=0x3FFFFFF;
RevPtr.AllpassRight1.InputAllpass=0;
RevPtr.AllpassRight2.Pointer_CurrentAllpass_rd1=(38819+2057+1);
RevPtr.AllpassRight2.Pointer_CurrentAllpass_wr1=(38819+2057);
RevPtr.AllpassRight2.Pointer_Allpass_rd1=((unsigned) &Delay_1);
RevPtr.AllpassRight2.SizeAllpass_rd1=21;
RevPtr.AllpassRight2.KFeedbackAllpass=0xfffffffff4cccccc;
RevPtr.AllpassRight2.KOutputAllpass=0x3FFFFFF;
RevPtr.AllpassRight2.InputAllpass=0;
RevPtr.AllpassLeft1.Pointer_CurrentAllpass_rd1=(40897+1);
RevPtr.AllpassLeft1.Pointer_CurrentAllpass_wr1=40897;
RevPtr.AllpassLeft1.Pointer_Allpass_rd1=((unsigned) &Delay_1);
RevPtr.AllpassLeft1.SizeAllpass_rd1=2051;
RevPtr.AllpassLeft1.KFeedbackAllpass=-0xb333333;
RevPtr.AllpassLeft1.KOutputAllpass=0x3FFFFFF;
RevPtr.AllpassLeft1.InputAllpass=0;
RevPtr.AllpassLeft2.Pointer_CurrentAllpass_rd1=(40897+2051+1);
RevPtr.AllpassLeft2.Pointer_CurrentAllpass_wr1=(40897+2051);
RevPtr.AllpassLeft2.Pointer_Allpass_rd1=((unsigned) &Delay_1);
RevPtr.AllpassLeft2.SizeAllpass_rd1=17;
RevPtr.AllpassLeft2.KFeedbackAllpass=0xb333333;
RevPtr.AllpassLeft2.KOutputAllpass=0x3FFFFFF;
RevPtr.AllpassLeft2.InputAllpass=0;
RevPtr.KEarlyRightToRight=0xe666666;
RevPtr.KEarlyLeftToRight=0;
RevPtr.KRevRightToRight=0;
RevPtr.KRevLeftToRight=0;
RevPtr.KRevRightToLeft=0;
RevPtr.KRevLeftToLeft=0;
RevPtr.KEarlyRightToLeft=0;
RevPtr.KEarlyLeftToLeft=0xe666666;

debug_printf("Biquad effect off\n");

for (k = 0;k<100000;k++) {
    if (k == 0)
       RevPtr.Biquad1.InputBq = 0x7ffffff;
    else
       RevPtr.Biquad1.InputBq = 0;

    asm_effect();


}
while(1) {

       asm_effect();

  }
```

```
    // Send/Receive samples
  /*  for (size_t i = 0; i < num_chans; i++) {
      c_dsp :> in_samps[i];
      //in_samps[i] = in_samps[i]>>8;
      out_samps[i] = in_samps[i];
      //out_samps[i]= out_samps[i]<<8;
      c_dsp <: out_samps[i];
    }
    //c_dsp :> in_samps[0];
    Pointer_ptr.input_delay=in_samps[0]>>4;
    if (in_samps[0] > prova)
        prova=in_samps[0];
    //RevPtr();
    out_samps[0]=Pointer_ptr.delta_delay_rd1<<4;*/
    //c_dsp <: out_samps[0];
//    asm_effect();
///// per adesso   xscope_int(LEFT_IN, in_samps[1]);
///// per adesso   xscope_int(LEFT_OUT, out_samps[1]);
    //xscope_int(GAIN, gain);

    select{
    // load K_&P_small   carica koef e pointer gli passa indirizzo e dato senza mute
    // load algorit  carica un algoritmo. Fa un jump sull algoritmo da caricare poi carica
    //               poi carica il codice
    // load preset       carica tutto un preset ( K&PTR)
    // mute_alg      esegue il mute dell' algoritmo
    // mute_all      esegue il mute dell' effetto

/*    case i_button.changed():
      if (i_button.get_value() == BUTTON_DOWN) {
        cur_effect++;
        if (cur_effect == NUM_BIQUAD_TYPES + 1)
          cur_effect = 0;

        i_led.set_multiple(0b111111111, LED_OFF);
        if (cur_effect == 0) {
          debug_printf("Biquad effect off\n");
          break;
        }
        enum biquad_type_t btype = cur_effect - 1;

        debug_printf("Effect: %s: Freq 1000Hz\n", filt_names[btype]);
        for (size_t i = 0; i < num_chans; i++) {
          init_biquad_state(btype, 1000, 48000, BIQUAD_Q_BUTTERWORTH,
                            biquad_state[i]);
        }
        i_led.set_multiple(1 << btype, LED_ON);
      }
      break; // end of button handling case
      */
    case c_gain :> gain:
      break;
    default:
      break;
    }
```

```
    // Do DSP processing ...
    //for (size_t i = 0; i < num_chans; i++) {
      //if (cur_effect != 0)
      //  out_samps[i] = apply_biquad(in_samps[i], biquad_state[i]);
      //else
    // out_samps[i] = in_samps[i];
    // }

    // Apply gain
    //for (size_t i = 0; i < num_chans; i++)  {
    //  out_samps[i] = apply_gain(out_samps[i], gain);
    //}
  }
```

asm_effect.S

```
.text
#define     Offset_delta_delay_rd1                  0
#define     Offset_Pointer_Currentdelay_rd1     1
#define     Offset_Pointer_delay_rd1                2
#define     Offset_Sizedelay_rd1                    3
#define     Offset_kfeedB_delay_rd1                     4
#define     Offset_k1dump_delay_rd1                     5
#define     Offset_k2dump_delay_rd1                     6
#define     Offset_stdump_delay_rd1                     7

#define     Offset_delta_delay_wr1                      8
#define     Offset_Pointer_Currentdelay_wr1         9
#define     Offset_Input_delay                              10
#define     Offset_Sizedelay_wr1                    11
#define     Offset_Pointer_delay_wr1                10

#define     Offset_K1to2out                             11

// Comb filter

#define             Offset_OutputComb_rd1               0
#define             Offset_Pointer_CurrentComb_rd1              1
#define             Offset_Pointer_Comb_rd1             2
#define             Offset_SizeComb_rd1                 3
#define             Offset_KFeedbackComb                4
#define             Offset_StateDump            5
#define             Offset_KDump1               6
#define             Offset_KDump2               7

#define             Offset_KOutputComb                  8
#define             Offset_InputComb            9
#define             Offset_OutputComb_wr1               10
#define             Offset_Pointer_CurrentComb_wr1              11

// Allpass filter

#define             Offset_Pointer_CurrentAllpass_rd1           0
#define             Offset_Pointer_CurrentAllpass_wr1           1
#define             Offset_Pointer_Allpass_rd1          2
```

```
#define		Offset_SizeAllpass_rd1		3
#define		Offset_KFeedback_Allpass	4
#define		Offset_StateAllpass		5
#define		Offset_KOutput_Allpass		6
#define		Offset_InputAllpass		7

#define		Offset_OutputAllpass_rd1	8
#define		Offset_OutputAllpass_wr1	9

// 24 filters

#define		Offset_Pointer_Twentyfour		0
#define		Offset_SizeTwentyfour			1
#define		Offset_Pointer_CurrentTwentyfour1_rd1		2
#define		Offset_Pointer_CurrentTwentyfour2_rd1		3
#define		Offset_Pointer_CurrentTwentyfour3_rd1		4
#define		Offset_Pointer_CurrentTwentyfour4_rd1		5
#define		Offset_Pointer_CurrentTwentyfour5_rd1		6
#define		Offset_Pointer_CurrentTwentyfour6_rd1		7

#define		Offset_Pointer_CurrentTwentyfour7_rd1		8
#define		Offset_Pointer_CurrentTwentyfour8_rd1		9
#define		Offset_Pointer_CurrentTwentyfour9_rd1		10
#define		Offset_Pointer_CurrentTwentyfour10_rd1		11

#define		Offset_Pointer_CurrentTwentyfour11_rd1
	1+Offset_Pointer_CurrentTwentyfour10_rd1
#define		Offset_Pointer_CurrentTwentyfour12_rd1
	2+Offset_Pointer_CurrentTwentyfour10_rd1
#define		Offset_Pointer_CurrentTwentyfour13_rd1
	3+Offset_Pointer_CurrentTwentyfour10_rd1
#define		Offset_Pointer_CurrentTwentyfour14_rd1
	4+Offset_Pointer_CurrentTwentyfour10_rd1
#define		Offset_Pointer_CurrentTwentyfour15_rd1
	5+Offset_Pointer_CurrentTwentyfour10_rd1
#define		Offset_Pointer_CurrentTwentyfour16_rd1
	6+Offset_Pointer_CurrentTwentyfour10_rd1
#define		Offset_Pointer_CurrentTwentyfour17_rd1
	7+Offset_Pointer_CurrentTwentyfour10_rd1
#define		Offset_Pointer_CurrentTwentyfour18_rd1
	8+Offset_Pointer_CurrentTwentyfour10_rd1

#define		Offset_Pointer_CurrentTwentyfour19_rd1
	9+Offset_Pointer_CurrentTwentyfour10_rd1
#define		Offset_Pointer_CurrentTwentyfour20_rd1
	10+Offset_Pointer_CurrentTwentyfour10_rd1
#define		Offset_Pointer_CurrentTwentyfour21_rd1
	11+Offset_Pointer_CurrentTwentyfour10_rd1
#define		Offset_Pointer_CurrentTwentyfour22_rd1
	12+Offset_Pointer_CurrentTwentyfour10_rd1

#define		Offset_Pointer_CurrentTwentyfour23_rd1
	1+Offset_Pointer_CurrentTwentyfour22_rd1
#define		Offset_Pointer_CurrentTwentyfour24_rd1
	2+Offset_Pointer_CurrentTwentyfour22_rd1
```

```
#define		Offset_Pointer_CurrentTwentyfour_wr1
	3+Offset_Pointer_CurrentTwentyfour22_rd1
#define		Offset_InputTwentyfour
	4+Offset_Pointer_CurrentTwentyfour22_rd1
#define		Offset_K1_Twentyfour
	5+Offset_Pointer_CurrentTwentyfour22_rd1
#define		Offset_State1_Twentyfour
	6+Offset_Pointer_CurrentTwentyfour22_rd1
#define		Offset_K2_Twentyfour
	7+Offset_Pointer_CurrentTwentyfour22_rd1
#define		Offset_State2_Twentyfour
	8+Offset_Pointer_CurrentTwentyfour22_rd1

#define		Offset_K3_Twentyfour
	9+Offset_Pointer_CurrentTwentyfour22_rd1
#define		Offset_State3_Twentyfour
	10+Offset_Pointer_CurrentTwentyfour22_rd1
#define		Offset_K4_Twentyfour
	11+Offset_Pointer_CurrentTwentyfour22_rd1
#define		Offset_State4_Twentyfour
	12+Offset_Pointer_CurrentTwentyfour22_rd1

#define		Offset_K5_Twentyfour		1+Offset_State4_Twentyfour
#define		Offset_State5_Twentyfour	2+Offset_State4_Twentyfour
#define		Offset_K6_Twentyfour		3+Offset_State4_Twentyfour
#define		Offset_State6_Twentyfour	4+Offset_State4_Twentyfour
#define		Offset_K7_Twentyfour		5+Offset_State4_Twentyfour
#define		Offset_State7_Twentyfour	6+Offset_State4_Twentyfour
#define		Offset_K8_Twentyfour		7+Offset_State4_Twentyfour
#define		Offset_State8_Twentyfour	8+Offset_State4_Twentyfour

#define		Offset_K9_Twentyfour		9+Offset_State4_Twentyfour
#define		Offset_State9_Twentyfour	10+Offset_State4_Twentyfour
#define		Offset_K10_Twentyfour		11+Offset_State4_Twentyfour
#define		Offset_State10_Twentyfour	12+Offset_State4_Twentyfour

#define		Offset_K11_Twentyfour		1+Offset_State10_Twentyfour
#define		Offset_State11_Twentyfour	2+Offset_State10_Twentyfour
#define		Offset_K12_Twentyfour		3+Offset_State10_Twentyfour
#define		Offset_State12_Twentyfour	4+Offset_State10_Twentyfour
#define		Offset_K13_Twentyfour		5+Offset_State10_Twentyfour
#define		Offset_State13_Twentyfour	6+Offset_State10_Twentyfour
#define		Offset_K14_Twentyfour		7+Offset_State10_Twentyfour
#define		Offset_State14_Twentyfour	8+Offset_State10_Twentyfour

#define		Offset_K15_Twentyfour		9+Offset_State10_Twentyfour
#define		Offset_State15_Twentyfour	10+Offset_State10_Twentyfour
#define		Offset_K16_Twentyfour		11+Offset_State10_Twentyfour
#define		Offset_State16_Twentyfour	12+Offset_State10_Twentyfour

#define		Offset_K17_Twentyfour		1+Offset_State16_Twentyfour
#define		Offset_State17_Twentyfour	2+Offset_State16_Twentyfour
#define		Offset_K18_Twentyfour		3+Offset_State16_Twentyfour
#define		Offset_State18_Twentyfour	4+Offset_State16_Twentyfour
#define		Offset_K19_Twentyfour		5+Offset_State16_Twentyfour
#define		Offset_State19_Twentyfour	6+Offset_State16_Twentyfour
```

```
#define		Offset_K20_Twentyfour		7+Offset_State16_Twentyfour
#define		Offset_State20_Twentyfour	8+Offset_State16_Twentyfour

#define		Offset_K21_Twentyfour		9+Offset_State16_Twentyfour
#define		Offset_State21_Twentyfour	10+Offset_State16_Twentyfour
#define		Offset_K22_Twentyfour		11+Offset_State16_Twentyfour
#define		Offset_State22_Twentyfour	12+Offset_State16_Twentyfour

#define		Offset_K23_Twentyfour		1+Offset_State22_Twentyfour
#define		Offset_State23_Twentyfour	2+Offset_State22_Twentyfour
#define		Offset_K24_Twentyfour		3+Offset_State22_Twentyfour
#define		Offset_State24_Twentyfour	4+Offset_State22_Twentyfour
#define		Offset_OutputTwentyfour_rd1
	5+Offset_State22_Twentyfour
#define		Offset_OutputTwentyfour_wr1
	6+Offset_State22_Twentyfour
#define		Offset_Ambience2OutputRight
	7+Offset_State22_Twentyfour
#define		Offset_Ambience2OutputLeft
	8+Offset_State22_Twentyfour
```

// Riverbero Ambience 2

```
#define		Offset_InputVolume		0
#define		Offset_Biquad1			1
#define		Offset_Biquad2			2
#define		Offset_EarlyReflectionsRight		3
#define		Offset_EarlyReflectionsLeft		4
#define		Offset_EarlyRightVol		5
#define		Offset_EarlyLeftVol		6
#define		Offset_CombMain1		7

#define		Offset_CombMain2		8
#define		Offset_CombMain3		9
#define		Offset_CombMain4		10
#define		Offset_CombMain5		11

#define		Offset_CombMain6		1+Offset_CombMain5
#define		Offset_CombMain7		2+Offset_CombMain5
#define		Offset_CombRight1		3+Offset_CombMain5
#define		Offset_CombRight2		4+Offset_CombMain5
#define		Offset_CombLeft1		5+Offset_CombMain5
#define		Offset_CombLeft2		6+Offset_CombMain5
#define		Offset_AllpassRight1		7+Offset_CombMain5
#define		Offset_AllpassRight2		8+Offset_CombMain5

#define		Offset_AllpassLeft1		9+Offset_CombMain5
#define		Offset_AllpassLeft2		10+Offset_CombMain5
#define		Offset_KEarlyRightToRight		11+Offset_CombMain5
#define		Offset_KEarlyLeftToRight		12+Offset_CombMain5

#define		Offset_KRevRightToRight
	1+Offset_KEarlyLeftToRight
#define		Offset_KRevLeftToRight
	2+Offset_KEarlyLeftToRight
```

```
#define		Offset_KRevRightToLeft
	3+Offset_KEarlyLeftToRight
#define		Offset_KRevLeftToLeft
	4+Offset_KEarlyLeftToRight
#define		Offset_KEarlyRightToLeft
	5+Offset_KEarlyLeftToRight
#define		Offset_KEarlyLeftToLeft
	6+Offset_KEarlyLeftToRight
#define		Offset_OutEarlyRightToRight
	7+Offset_KEarlyLeftToRight
#define		Offset_OutEarlyLeftToRight
	8+Offset_KEarlyLeftToRight

#define		Offset_OutRevRightToRight
	9+Offset_KEarlyLeftToRight
#define		Offset_OutRevLeftToRight
	10+Offset_KEarlyLeftToRight
#define		Offset_OutRevRightToLeft
	11+Offset_KEarlyLeftToRight
#define		Offset_OutRevLeftToLeft
	12+Offset_KEarlyLeftToRight

#define		Offset_OutEarlyRightToLeft
	1+Offset_OutRevLeftToLeft
#define		Offset_OutEarlyLeftToLeft
	2+Offset_OutRevLeftToLeft

// Biquad

#define	Offset_InputBq
	(Offset_K1to2out+2)
#define	Offset_ka0			(Offset_InputBq+1)
#define	Offset_Ka1				(Offset_InputBq+2)
#define	Offset_Ka2			(Offset_InputBq+3)
#define	Offset_kb1				(Offset_InputBq+4)
#define	Offset_kb2				(Offset_InputBq+5)
#define	Offset_sta1			(Offset_InputBq+6)
#define	Offset_sta2			(Offset_InputBq+7)
#define	Offset_stb1			(Offset_InputBq+8)
#define	Offset_stb2			(Offset_InputBq+9)
#define	Offset_OutputBq		(Offset_InputBq+10)



#define Pointer_strctR7	r7


//#define t		r11

#define Ptr_rdR0			r0
#define Ptr_rd_currentR1	r1
#define SampleR3			r3
#define delta_rdR4		r4
#define Sizedelay_rdR2		r2

#define Ptr_wrR0			r0
```

```
#define Ptr_wr_currentR1        r1
#define delta_wrR4              r4
#define Sizedelay_wrR2          r2


#define KfeedbackR8             r8
#define K1dumpR1        r1
#define K2dumpR10       r10
#define StdumpR3        r3
#define     RhighR4             r4
#define     RlowR6              r6
#define     K1to2outR9  r9

#define p r2
#define in_delayR10 r10

// Comb Filter
#define InputCombR0                         r0
#define KFeedbackCombR1             r1
#define KDump1CombR2                    r2
#define KDump2CombR3                    r3
#define RhighR4                                 r4
#define StateDumpCombR5             r5
#define RlowR6                                          r6
#define KOutputCombR9                   r9
#define OutputCombR10                   r10
#define SizeDelayCombR11            r11

#define XnmCombR10                  r10

// Allpass Filter
#define InputAllpassR0                      r0
#define KFeedbackAllpassR1          r1
#define StateAllpassR2                      r2
#define RhighR4                 r4
#define RlowR6                  r6
#define KOutputAllpassR9                r9
#define OutputAllpassR10                r10
#define SizeDelayAllpassR11             r11

#define XnmAllpassR10               r10

// Parallel of twentyfour gains
#define InputTwentyfourR2           r2
#define RhighR4                 r4
#define RlowR6                  r6
#define StateTwentyfourR2           r2
#define OutputTwentyfourR10             r10
#define SizeTwentyfourR11           r11
#define K1TwentyfourR1                  r1
#define K2TwentyfourR2                  r2
#define K3TwentyfourR3                  r3
#define K4TwentyfourR5                  r5
#define K5TwentyfourR9                  r9
#define K6TwentyfourR10                 r10
#define K7TwentyfourR11                 r11
```

```
#define K8TwentyfourR1                  r1
#define K9TwentyfourR2                  r2
#define K10TwentyfourR3                 r3
#define K11TwentyfourR5                 r5
#define K12TwentyfourR9                 r9
#define K13TwentyfourR10        r10
#define K14TwentyfourR11        r11
#define K15TwentyfourR1                 r1
#define K16TwentyfourR2                 r2
#define K17TwentyfourR3                 r3
#define K18TwentyfourR5                 r5
#define K19TwentyfourR9                 r9
#define K20TwentyfourR10        r10
#define K21TwentyfourR11        r11
#define K22TwentyfourR1                 r1
#define K23TwentyfourR2                 r2
#define K24TwentyfourR3                 r3
#define State1TwentyfourR5                      r5
#define State2TwentyfourR9                      r9
#define State3TwentyfourR10             r10
#define State4TwentyfourR11             r11
#define State5TwentyfourR1                      r1
#define State6TwentyfourR2                      r2
#define State7TwentyfourR3                      r3
#define State8TwentyfourR5                      r5
#define State9TwentyfourR9                      r9
#define State10TwentyfourR10            r10
#define State11TwentyfourR11            r11
#define State12TwentyfourR1             r1
#define State13TwentyfourR2             r2
#define State14TwentyfourR3             r3
#define State15TwentyfourR5             r5
#define State16TwentyfourR9             r9
#define State17TwentyfourR10            r10
#define State18TwentyfourR11            r11
#define State19TwentyfourR1             r1
#define State20TwentyfourR2             r2
#define State21TwentyfourR3             r3
#define State22TwentyfourR5             r5
#define State23TwentyfourR9             r9
#define State24TwentyfourR10            r10
#define Xnm1TwentyfourR10               r10
#define Xnm2TwentyfourR10               r10
#define Xnm3TwentyfourR10               r10
#define Xnm4TwentyfourR10               r10
#define Xnm5TwentyfourR10               r10
#define Xnm6TwentyfourR10               r10
#define Xnm7TwentyfourR10               r10
#define Xnm8TwentyfourR10               r10
#define Xnm9TwentyfourR10               r10
#define Xnm10TwentyfourR10                      r10
#define Xnm11TwentyfourR10                      r10
#define Xnm12TwentyfourR10                      r10
#define Xnm13TwentyfourR10                      r10
#define Xnm14TwentyfourR10                      r10
#define Xnm15TwentyfourR10                      r10
```

```
#define Xnm16TwentyfourR10                  r10
#define Xnm17TwentyfourR10                  r10
#define Xnm18TwentyfourR10                  r10
#define Xnm19TwentyfourR10                  r10
#define Xnm20TwentyfourR10                  r10
#define Xnm21TwentyfourR10                  r10
#define Xnm22TwentyfourR10                  r10
#define Xnm23TwentyfourR10                  r10
#define Xnm24TwentyfourR10                  r10

//Riverbero Ambience
#define     InputVolumeR0                 r0
#define     EarlyRightVolR2               r2
#define     EarlyLeftVolR3                r3
#define     EarlyRightOutputR5            r5
#define     EarlyLeftOutputR9             r5
#define     CombMain1OutputR5                   r5
#define     CombMain2OutputR9                   r9
#define     CombMain3OutputR5                   r5
#define     CombMain4OutputR9                   r9
#define     CombMain5OutputR5                   r5
#define     CombMain6OutputR9                   r9
#define     CombMain7OutputR5                   r5
#define     CombRight1OutputR9                  r9
#define     CombRight2OutputR5                  r5
#define     CombLeft1OutputR9             r9
#define     CombLeft2OutputR5             r5
#define     AllpassRight1OutputR5               r5
#define     AllpassRight2OutputR9               r9
#define     AllpassLeft1OutputR5                r5
#define     AllpassLeft2OutputR9                r9
#define     KEarlyR2RR5                   r5
#define     KEarlyL2RR9                   r9
#define     KRevR2RR5                     r5
#define     KRevL2RR10                    r10
#define     KRevR2LR5                     r5
#define     KRevL2LR10              r10
#define     KEarlyR2LR5                   r5
#define     KEarlyL2LR9                   r9
#define     EarlyR2ROutputR5              r5
#define     EarlyL2ROutputR9              r9
#define     RevR2ROutputR5                r5
#define     RevL2ROutputR9                r9
#define     RevR2LOutputR5                r5
#define     RevL2LOutputR9                r9
#define     EarlyR2LOutputR5              r5
#define     EarlyL2LOutputR9              r9
#define     RhighR4                       r4
#define     RlowR6                        r6

//------------------------ register filter Biquad-------------------------------
#define In_bqDelr0              r0
#define Ka0_bqDelr1             r1
#define Ka1_bqDelr3             r3
#define Ka2_bqDelr2             r2
#define Kb1_bqDelr3             r3
```

```
#define Kb2_bqDelr2                 r2
#define Sta1_bqDelr5                r5
#define Sta2_bqDelr8                r8
#define Stb1_bqDelr5                r5
#define Stb2_bqDelr8                r8

#define Pointer_strctR10    r10

#define bit_28x24    20
#define bit_28x28    27

#define MACNORM bit_28x28
#define DELAYNORM 8

#define STACKWORDS 16

.extern Pointer_k
.extern Pointer_ptr
.extern CombPtr
.extern AllpassPtr
.extern TwentyfourPtr
.extern RevPtr

.align 8
.globl asm_effect
.type  asm_effect, @function
.cc_top asm_effect.function

#define c_source1 r0


asm_effect:
.issue_mode dual
        DUALENTSP_lu6 16

        std r0, r1, sp[2]
        std r2, r3, sp[3]
        std r4, r5, sp[4]
        std r6, r7, sp[5]
        std r8, r9, sp[6]
        std r10, r11, sp[7]

rev:
// Il segnale di ingresso al riverbero, InputVolume, viene inizialmente passato in ingresso ad un
filtro
// biquadratico
                ldaw                    Pointer_strctR7, dp[RevPtr];

                ldc             r8, MACNORM;

                {ldc    RhighR4, 0;
                        ldc RlowR6, 0}
                ldd             Ka0_bqDelr1,In_bqDelr0,Pointer_strctR7[(Offset_InputBq-
Offset_InputBq)/2];
                maccs RhighR4,RlowR6,Ka0_bqDelr1,In_bqDelr0;
```

```
                ldd             Ka2_bqDelr2,Ka1_bqDelr3,Pointer_strctR7[(Offset_Ka1-
Offset_InputBq)/2];
                ldd             Sta2_bqDelr8,Sta1_bqDelr5,Pointer_strctR7[(Offset_sta1-
Offset_InputBq)/2];
                maccs RhighR4,RlowR6,Ka1_bqDelr3,Sta1_bqDelr5;
                maccs RhighR4,RlowR6,Ka2_bqDelr2,Sta2_bqDelr8;

                ldc             r1, MACNORM;

                ldd             Kb2_bqDelr2,Kb1_bqDelr3,Pointer_strctR7[(Offset_kb1-
Offset_InputBq)/2];
                ldd             Stb2_bqDelr8,Stb1_bqDelr5,Pointer_strctR7[(Offset_stb1-
Offset_sta1)/2];
                maccs RhighR4,RlowR6,Kb1_bqDelr3,Stb1_bqDelr5;
                maccs RhighR4,RlowR6,Kb2_bqDelr2,Stb2_bqDelr8;
                lextract r2, RhighR4, RlowR6, r8, 32;

                ldc     r3,(Offset_InputTwentyfour+12);
                stw     r2,Pointer_strctR7[r3];
                ldc     r3,(Offset_InputTwentyfour+90);
                stw     r2,Pointer_strctR7[r3];

                ldc     r9, (Offset_SizeTwentyfour+12);
                ldw     Sizedelay_rdR2, Pointer_strctR7[r9];
                add     Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq      r9, Ptr_rd_currentR1, Sizedelay_rdR2;
        bf      r9, ambience2_early_refl_right1;
        ldc     Ptr_rd_currentR1, 0;

// Il segnale in uscita dalla biquadratica viene passato in ingresso a due componenti in parallelo
che ne producono
// le prime riflessioni: il primo componente è relativo al canale destro,il secondo al canale sinistro.
ambience2_early_refl_right1:
                ldc                     r3, (Offset_Pointer_CurrentTwentyfour1_rd1+12);
                ldw                     Ptr_rd_currentR1,Pointer_strctR7[r3];
                ldc                     r2,(Offset_Pointer_Twentyfour+12)/2;
                ldd                     SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[r2];
                ld16s           Xnm1TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

                add                     r3, r3, 1;
                ldc                     r5, (Offset_State1_Twentyfour+12);
                stw                     Xnm1TwentyfourR10,Pointer_strctR7[r5];

                add             Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq              r9, Ptr_rd_currentR1, SizeTwentyfourR11;
                bf              r9, ambience2_early_refl_right2;
                ldc             Ptr_rd_currentR1, 0x0;

ambience2_early_refl_right2:
                ldc             r2,(Offset_Pointer_CurrentTwentyfour1_rd1+12);
                stw             Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldw                     Ptr_rd_currentR1,Pointer_strctR7[r3];
                ldd
        SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[(Offset_Pointer_Twentyfour+12)/2];
                ld16s           Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];
```

```
                add             r3, r3, 1;
                ldc             r5, (Offset_State2_Twentyfour+12);
                stw             Xnm2TwentyfourR10,Pointer_strctR7[r5];

                add         Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq          r9, Ptr_rd_currentR1, SizeTwentyfourR11;
                bf          r9, ambience2_early_refl_right3;
                ldc         Ptr_rd_currentR1, 0x0;

ambience2_early_refl_right3:
                ldc         r2,(Offset_Pointer_CurrentTwentyfour2_rd1+12);
                stw         Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldw             Ptr_rd_currentR1,Pointer_strctR7[r3];
                ldd
        SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[(Offset_Pointer_Twentyfour+12)/2];
                ld16s       Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

                add             r3, r3, 1;
                ldc             r5, (Offset_State3_Twentyfour+12);
                stw             Xnm2TwentyfourR10,Pointer_strctR7[r5];

                add         Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq          r9, Ptr_rd_currentR1, SizeTwentyfourR11;
                bf          r9, ambience2_early_refl_right4;
                ldc         Ptr_rd_currentR1, 0x0;

ambience2_early_refl_right4:
                ldc         r2,(Offset_Pointer_CurrentTwentyfour3_rd1+12);
                stw         Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldw             Ptr_rd_currentR1,Pointer_strctR7[r3];
                ldd
        SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[(Offset_Pointer_Twentyfour+12)/2];
                ld16s       Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

                add             r3, r3, 1;
                ldc             r5, (Offset_State4_Twentyfour+12);
                stw             Xnm2TwentyfourR10,Pointer_strctR7[r5];

                add         Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq          r9, Ptr_rd_currentR1, SizeTwentyfourR11;
                bf          r9, ambience2_early_refl_right5;
                ldc         Ptr_rd_currentR1, 0x0;

ambience2_early_refl_right5:
                ldc         r2,(Offset_Pointer_CurrentTwentyfour4_rd1+12);
                stw         Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldw             Ptr_rd_currentR1,Pointer_strctR7[r3];
                ldd
        SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[(Offset_Pointer_Twentyfour+12)/2];
                ld16s       Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

                add             r3, r3, 1;
```

```
        ldc             r5, (Offset_State5_Twentyfour+12);
        stw             Xnm2TwentyfourR10,Pointer_strctR7[r5];

        add         Ptr_rd_currentR1,Ptr_rd_currentR1,1;
        eq          r9, Ptr_rd_currentR1, SizeTwentyfourR11;
        bf          r9, ambience2_early_refl_right6;
        ldc         Ptr_rd_currentR1, 0x0;

ambience2_early_refl_right6:
        ldc         r2,(Offset_Pointer_CurrentTwentyfour5_rd1+12);
        stw         Ptr_rd_currentR1,Pointer_strctR7[r2];

        ldw             Ptr_rd_currentR1,Pointer_strctR7[r3];
        ldd
    SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[(Offset_Pointer_Twentyfour+12)/2];
        ld16s       Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

        add             r3, r3, 1;
        ldc             r5, (Offset_State6_Twentyfour+12);
        stw             Xnm2TwentyfourR10,Pointer_strctR7[r5];

        add         Ptr_rd_currentR1,Ptr_rd_currentR1,1;
        eq          r9, Ptr_rd_currentR1, SizeTwentyfourR11;
        bf          r9, ambience2_early_refl_right7;
        ldc         Ptr_rd_currentR1, 0x0;

ambience2_early_refl_right7:
        ldc         r2,(Offset_Pointer_CurrentTwentyfour6_rd1+12);
        stw         Ptr_rd_currentR1,Pointer_strctR7[r2];

        ldw             Ptr_rd_currentR1,Pointer_strctR7[r3];
        ldd
    SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[(Offset_Pointer_Twentyfour+12)/2];
        ld16s       Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

        add             r3, r3, 1;
        ldc             r5, (Offset_State7_Twentyfour+12);
        stw             Xnm2TwentyfourR10,Pointer_strctR7[r5];

        add         Ptr_rd_currentR1,Ptr_rd_currentR1,1;
        eq          r9, Ptr_rd_currentR1, SizeTwentyfourR11;
        bf          r9, ambience2_early_refl_right8;
        ldc         Ptr_rd_currentR1, 0x0;

ambience2_early_refl_right8:
        ldc         r2,(Offset_Pointer_CurrentTwentyfour7_rd1+12);
        stw         Ptr_rd_currentR1,Pointer_strctR7[r2];

        ldw             Ptr_rd_currentR1,Pointer_strctR7[r3];
        ldd
    SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[(Offset_Pointer_Twentyfour+12)/2];
        ld16s       Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

        add             r3, r3, 1;
        ldc             r5, (Offset_State8_Twentyfour+12);
        stw             Xnm2TwentyfourR10,Pointer_strctR7[r5];
```

```
                add             Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq              r9, Ptr_rd_currentR1, SizeTwentyfourR11;
                bf              r9, ambience2_early_refl_right9;
                ldc             Ptr_rd_currentR1, 0x0;

ambience2_early_refl_right9:
                ldc             r2,(Offset_Pointer_CurrentTwentyfour8_rd1+12);
                stw             Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldw                     Ptr_rd_currentR1,Pointer_strctR7[r3];
                ldd
        SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[(Offset_Pointer_Twentyfour+12)/2];
                ld16s           Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

                add                     r3, r3, 1;
                ldc                     r5, (Offset_State9_Twentyfour+12);
                stw                     Xnm2TwentyfourR10,Pointer_strctR7[r5];

                add             Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq              r9, Ptr_rd_currentR1, SizeTwentyfourR11;
                bf              r9, ambience2_early_refl_right10;
                ldc             Ptr_rd_currentR1, 0x0;

ambience2_early_refl_right10:
                ldc             r2,(Offset_Pointer_CurrentTwentyfour9_rd1+12);
                stw             Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldw                     Ptr_rd_currentR1,Pointer_strctR7[r3];
                ldd
        SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[(Offset_Pointer_Twentyfour+12)/2];
                ld16s           Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

                add                     r3, r3, 1;
                ldc                     r5, (Offset_State10_Twentyfour+12);
                stw                     Xnm2TwentyfourR10,Pointer_strctR7[r5];

                add             Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq              r9, Ptr_rd_currentR1, SizeTwentyfourR11;
                bf              r9, ambience2_early_refl_right11;
                ldc             Ptr_rd_currentR1, 0x0;

ambience2_early_refl_right11:
                ldc             r2,(Offset_Pointer_CurrentTwentyfour10_rd1+12);
                stw             Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldw                     Ptr_rd_currentR1,Pointer_strctR7[r3];
                ldd
        SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[(Offset_Pointer_Twentyfour+12)/2];
                ld16s           Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

                add                     r3, r3, 1;
                ldc                     r5, (Offset_State11_Twentyfour+12);
                stw                     Xnm2TwentyfourR10,Pointer_strctR7[r5];

                add             Ptr_rd_currentR1,Ptr_rd_currentR1,1;
```

```
                eq              r9, Ptr_rd_currentR1, SizeTwentyfourR11;
                bf              r9, ambience2_early_refl_right12;
                ldc             Ptr_rd_currentR1, 0x0;

ambience2_early_refl_right12:
                ldc             r2,(Offset_Pointer_CurrentTwentyfour11_rd1+12);
                stw             Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldw                     Ptr_rd_currentR1,Pointer_strctR7[r3];
                ldd
        SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[(Offset_Pointer_Twentyfour+12)/2];
                ld16s           Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

                add                     r3, r3, 1;
                ldc                     r5, (Offset_State12_Twentyfour+12);
                stw                     Xnm2TwentyfourR10,Pointer_strctR7[r5];

                add             Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq              r9, Ptr_rd_currentR1, SizeTwentyfourR11;
                bf              r9, ambience2_early_refl_right_maccs;
                ldc             Ptr_rd_currentR1, 0x0;

ambience2_early_refl_right_maccs:
                ldc             r2,(Offset_Pointer_CurrentTwentyfour12_rd1+12);
                stw             Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldc             r8, MACNORM;
          {ldc          r11,(Offset_K1_Twentyfour+12)/2 ;
        ldc     r3, (Offset_K2_Twentyfour+12)/2}
                ldd             State1TwentyfourR5,K1TwentyfourR1,Pointer_strctR7[r11];
                ldd             State2TwentyfourR9,K2TwentyfourR2,Pointer_strctR7[r3];
                {ldc            RhighR4, 0;
                ldc             RlowR6, 0}
                maccs RhighR4,RlowR6,K1TwentyfourR1,State1TwentyfourR5;
                maccs RhighR4,RlowR6,K2TwentyfourR2,State2TwentyfourR9;

                {ldc            r2, (Offset_K3_Twentyfour+12)/2;
                ldc             r9,(Offset_K4_Twentyfour+12)/2;}
                ldd             State3TwentyfourR10,K3TwentyfourR3,Pointer_strctR7[r2];
                ldd             State4TwentyfourR11,K4TwentyfourR5,Pointer_strctR7[r9];
                maccs RhighR4,RlowR6,K3TwentyfourR3,State3TwentyfourR10;
                maccs RhighR4,RlowR6,K4TwentyfourR5,State4TwentyfourR11;

                {ldc            r5, (Offset_K5_Twentyfour+12)/2;
                ldc             r11,(Offset_K6_Twentyfour+12)/2;}
                ldd             State5TwentyfourR1,K5TwentyfourR9,Pointer_strctR7[r5];
                ldd             State6TwentyfourR2,K6TwentyfourR10,Pointer_strctR7[r11];
                maccs RhighR4,RlowR6,K5TwentyfourR9,State5TwentyfourR1;
        maccs RhighR4,RlowR6,K6TwentyfourR10,State6TwentyfourR2;

        {ldc            r5,(Offset_K7_Twentyfour+12)/2;
           ldc          r9,(Offset_K8_Twentyfour+12)/2;}
        ldd             State7TwentyfourR3,K7TwentyfourR11,Pointer_strctR7[r5];
        ldd             State8TwentyfourR5,K8TwentyfourR1,Pointer_strctR7[r9];
        maccs RhighR4,RlowR6,K7TwentyfourR11,State7TwentyfourR3;
              maccs RhighR4,RlowR6,K8TwentyfourR1,State8TwentyfourR5;
```

```
    {ldc  r1, (Offset_K9_Twentyfour+12)/2;                                  ldc
  r5, (Offset_K10_Twentyfour+12)/2;}
    ldd         State9TwentyfourR9,K9TwentyfourR2,Pointer_strctR7[r1];
    ldd         State10TwentyfourR10,K10TwentyfourR3,Pointer_strctR7[r5];
    maccs RhighR4,RlowR6,K9TwentyfourR2,State9TwentyfourR9;
    maccs RhighR4,RlowR6,K10TwentyfourR3,State10TwentyfourR10;

    {ldc  r2, (Offset_K11_Twentyfour+12)/2;                                 ldc
  r3, (Offset_K12_Twentyfour+12)/2;}
    ldd         State11TwentyfourR11,K11TwentyfourR5,Pointer_strctR7[r2];
    ldd         State12TwentyfourR1,K12TwentyfourR9,Pointer_strctR7[r3];
    maccs RhighR4,RlowR6,K11TwentyfourR5,State11TwentyfourR11;
        maccs RhighR4,RlowR6,K12TwentyfourR9,State12TwentyfourR1;

        lextract r9, RhighR4, RlowR6, r8, 32;
        ldc             r1, (Offset_OutputTwentyfour_rd1+12);
        stw             r9,Pointer_strctR7[r1];

        ldc             r2,(Offset_Pointer_Twentyfour+12)/2;
        ldd             SizeTwentyfourR11,Ptr_wrR0,Pointer_strctR7[r2];

        ldc             r5,(Offset_Pointer_CurrentTwentyfour_wr1+12);
        ldw             Ptr_wr_currentR1, Pointer_strctR7[r5];

        st16            r9, Ptr_wrR0[Ptr_wr_currentR1];
        add             Ptr_wr_currentR1, Ptr_wr_currentR1, 1;
        eq              r9, Ptr_wr_currentR1, SizeTwentyfourR11;
        bf              r9, ambience2_early_refl_left1;
        ldc     Ptr_wr_currentR1, 0x0;

ambience2_early_refl_left1:
        ldc                     r9, (Offset_Pointer_CurrentTwentyfour_wr1+12);
        stw                     Ptr_rd_currentR1, Pointer_strctR7[r9];

        ldc                     r3, (Offset_Pointer_CurrentTwentyfour1_rd1+90);
        ldw                     Ptr_rd_currentR1,Pointer_strctR7[r3];
        ldc                     r5, (Offset_Pointer_Twentyfour+90)/2;
        ldd                     SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[r5];
        ld16s           Xnm1TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

        add                     r3, r3, 1;
        ldc                     r5, (Offset_State1_Twentyfour+90);
        stw                     Xnm1TwentyfourR10,Pointer_strctR7[r5];

        add             Ptr_rd_currentR1,Ptr_rd_currentR1,1;
        eq              r9, Ptr_rd_currentR1, SizeTwentyfourR11;
        bf              r9, ambience2_early_refl_left2;
        ldc             Ptr_rd_currentR1, 0x0;

ambience2_early_refl_left2:
        ldc             r2,(Offset_Pointer_CurrentTwentyfour1_rd1+90);
        stw             Ptr_rd_currentR1,Pointer_strctR7[r2];

        ldw                     Ptr_rd_currentR1,Pointer_strctR7[r3];
        ldc                     r5, (Offset_Pointer_Twentyfour+90)/2;
```

```
                ldd             SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[r5];
                ld16s       Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

                add             r3, r3, 1;
                ldc             r5, (Offset_State2_Twentyfour+90);
                stw             Xnm2TwentyfourR10,Pointer_strctR7[r5];

                add         Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq          r9, Ptr_rd_currentR1, SizeTwentyfourR11;
                bf          r9, ambience2_early_refl_left3;
                ldc         Ptr_rd_currentR1, 0x0;

ambience2_early_refl_left3:
                ldc         r2,(Offset_Pointer_CurrentTwentyfour2_rd1+90);
                stw         Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldw             Ptr_rd_currentR1,Pointer_strctR7[r3];
                ldc             r5, (Offset_Pointer_Twentyfour+90)/2;
                ldd             SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[r5];
                ld16s       Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

                add             r3, r3, 1;
                ldc             r5, (Offset_State3_Twentyfour+90);
                stw             Xnm2TwentyfourR10,Pointer_strctR7[r5];

                add         Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq          r9, Ptr_rd_currentR1, SizeTwentyfourR11;
                bf          r9, ambience2_early_refl_left4;
                ldc         Ptr_rd_currentR1, 0x0;

ambience2_early_refl_left4:
                ldc         r2,(Offset_Pointer_CurrentTwentyfour3_rd1+90);
                stw         Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldw             Ptr_rd_currentR1,Pointer_strctR7[r3];
                ldc             r5, (Offset_Pointer_Twentyfour+90)/2;
                ldd             SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[r5];
                ld16s       Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

                add             r3, r3, 1;
                ldc             r5, (Offset_State4_Twentyfour+90);
                stw             Xnm2TwentyfourR10,Pointer_strctR7[r5];

                add         Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq          r9, Ptr_rd_currentR1, SizeTwentyfourR11;
                bf          r9, ambience2_early_refl_left5;
                ldc         Ptr_rd_currentR1, 0x0;

ambience2_early_refl_left5:
                ldc         r2,(Offset_Pointer_CurrentTwentyfour4_rd1+90);
                stw         Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldw             Ptr_rd_currentR1,Pointer_strctR7[r3];
                ldc             r5, (Offset_Pointer_Twentyfour+90)/2;
                ldd             SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[r5];
                ld16s       Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];
```

```
                add             r3, r3, 1;
                ldc             r5, (Offset_State5_Twentyfour+90);
                stw             Xnm2TwentyfourR10,Pointer_strctR7[r5];

                add         Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq          r9, Ptr_rd_currentR1, SizeTwentyfourR11;
                bf          r9, ambience2_early_refl_left6;
                ldc         Ptr_rd_currentR1, 0x0;

ambience2_early_refl_left6:
                ldc         r2,(Offset_Pointer_CurrentTwentyfour5_rd1+90);
                stw         Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldw             Ptr_rd_currentR1,Pointer_strctR7[r3];
                ldc             r5, (Offset_Pointer_Twentyfour+90)/2;
                ldd             SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[r5];
                ld16s       Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

                add             r3, r3, 1;
                ldc             r5, (Offset_State6_Twentyfour+90);
                stw             Xnm2TwentyfourR10,Pointer_strctR7[r5];

                add         Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq          r9, Ptr_rd_currentR1, SizeTwentyfourR11;
                bf          r9, ambience2_early_refl_left7;
                ldc         Ptr_rd_currentR1, 0x0;

ambience2_early_refl_left7:
                ldc         r2,(Offset_Pointer_CurrentTwentyfour6_rd1+90);
                stw         Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldw             Ptr_rd_currentR1,Pointer_strctR7[r3];
                ldc             r5, (Offset_Pointer_Twentyfour+90)/2;
                ldd             SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[r5];
                ld16s       Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

                add             r3, r3, 1;
                ldc             r5, (Offset_State7_Twentyfour+90);
                stw             Xnm2TwentyfourR10,Pointer_strctR7[r5];

                add         Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq          r9, Ptr_rd_currentR1, SizeTwentyfourR11;
                bf          r9, ambience2_early_refl_left8;
                ldc         Ptr_rd_currentR1, 0x0;

ambience2_early_refl_left8:
                ldc         r2,(Offset_Pointer_CurrentTwentyfour7_rd1+90);
                stw         Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldw             Ptr_rd_currentR1,Pointer_strctR7[r3];
                ldc             r5, (Offset_Pointer_Twentyfour+90)/2;
                ldd             SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[r5];
                ld16s       Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

                add             r3, r3, 1;
```

```
                ldc             r5, (Offset_State8_Twentyfour+90);
                stw             Xnm2TwentyfourR10,Pointer_strctR7[r5];

                add         Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq          r9, Ptr_rd_currentR1, SizeTwentyfourR11;
                bf          r9, ambience2_early_refl_left9;
                ldc         Ptr_rd_currentR1, 0x0;

ambience2_early_refl_left9:
                ldc         r2,(Offset_Pointer_CurrentTwentyfour8_rd1+90);
                stw         Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldw             Ptr_rd_currentR1,Pointer_strctR7[r3];
                ldc             r5, (Offset_Pointer_Twentyfour+90)/2;
                ldd             SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[r5];
                ld16s       Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

                add             r3, r3, 1;
                ldc             r5, (Offset_State9_Twentyfour+90);
                stw             Xnm2TwentyfourR10,Pointer_strctR7[r5];

                add         Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq          r9, Ptr_rd_currentR1, SizeTwentyfourR11;
                bf          r9, ambience2_early_refl_left10;
                ldc         Ptr_rd_currentR1, 0x0;

ambience2_early_refl_left10:
                ldc         r2,(Offset_Pointer_CurrentTwentyfour9_rd1+90);
                stw         Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldw             Ptr_rd_currentR1,Pointer_strctR7[r3];
                ldc             r5, (Offset_Pointer_Twentyfour+90)/2;
                ldd             SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[r5];
                ld16s       Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

                add             r3, r3, 1;
                ldc             r5, (Offset_State10_Twentyfour+90);
                stw             Xnm2TwentyfourR10,Pointer_strctR7[r5];

                add         Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq          r9, Ptr_rd_currentR1, SizeTwentyfourR11;
                bf          r9, ambience2_early_refl_left11;
                ldc         Ptr_rd_currentR1, 0x0;

ambience2_early_refl_left11:
                ldc         r2,(Offset_Pointer_CurrentTwentyfour10_rd1+90);
                stw         Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldw             Ptr_rd_currentR1,Pointer_strctR7[r3];
                ldc             r5, (Offset_Pointer_Twentyfour+90)/2;
                ldd             SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[r5];
                ld16s       Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

                add             r3, r3, 1;
                ldc             r5, (Offset_State11_Twentyfour+90);
                stw             Xnm2TwentyfourR10,Pointer_strctR7[r5];
```

```
                add             Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq              r9, Ptr_rd_currentR1, SizeTwentyfourR11;
                bf              r9, ambience2_early_refl_left12;
                ldc             Ptr_rd_currentR1, 0x0;

ambience2_early_refl_left12:
                ldc             r2,(Offset_Pointer_CurrentTwentyfour11_rd1+90);
                stw             Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldw                     Ptr_rd_currentR1,Pointer_strctR7[r3];
                ldc                     r5, (Offset_Pointer_Twentyfour+90)/2;
                ldd                     SizeTwentyfourR11,Ptr_rdR0,Pointer_strctR7[r5];
                ld16s           Xnm2TwentyfourR10,Ptr_rdR0[Ptr_rd_currentR1];

                add                     r3, r3, 1;
                ldc                     r5, (Offset_State12_Twentyfour+90);
                stw                     Xnm2TwentyfourR10,Pointer_strctR7[r5];

                add     Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq      r9, Ptr_rd_currentR1, Sizedelay_rdR2;
                bf              r9, ambience2_early_refl_left_maccs;
                ldc             Ptr_rd_currentR1, 0x0;

ambience2_early_refl_left_maccs:
                ldc             r2,(Offset_Pointer_CurrentTwentyfour12_rd1+90);
                stw             Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldc             r8, MACNORM;
          {ldc          r11,(Offset_K1_Twentyfour+90)/2 ;
        ldc     r3, (Offset_K2_Twentyfour+90)/2}
                ldd             State1TwentyfourR5,K1TwentyfourR1,Pointer_strctR7[r11];
                ldd             State2TwentyfourR9,K2TwentyfourR2,Pointer_strctR7[r3];
                {ldc            RhighR4, 0;
                ldc             RlowR6, 0}
                maccs RhighR4,RlowR6,K1TwentyfourR1,State1TwentyfourR5;
                maccs RhighR4,RlowR6,K2TwentyfourR2,State2TwentyfourR9;

                {ldc            r2, (Offset_K3_Twentyfour+90)/2;
                ldc             r9,(Offset_K4_Twentyfour+90)/2;}
                ldd             State3TwentyfourR10,K3TwentyfourR3,Pointer_strctR7[r2];
                ldd             State4TwentyfourR11,K4TwentyfourR5,Pointer_strctR7[r9];
                maccs RhighR4,RlowR6,K3TwentyfourR3,State3TwentyfourR10;
                maccs RhighR4,RlowR6,K4TwentyfourR5,State4TwentyfourR11;

                ldc             r5, (Offset_K5_Twentyfour+90)/2;
                ldd             State5TwentyfourR1,K5TwentyfourR9,Pointer_strctR7[r5];
                ldc             r11,(Offset_K6_Twentyfour+90)/2;
                ldd             State6TwentyfourR2,K6TwentyfourR10,Pointer_strctR7[r11];
                maccs RhighR4,RlowR6,K5TwentyfourR9,State5TwentyfourR1;
        maccs RhighR4,RlowR6,K6TwentyfourR10,State6TwentyfourR2;

        ldc             r5,(Offset_K7_Twentyfour+90)/2;
        ldd             State7TwentyfourR3,K7TwentyfourR11,Pointer_strctR7[r5];
        ldc             r9,(Offset_K8_Twentyfour+90)/2;
        ldd             State8TwentyfourR5,K8TwentyfourR1,Pointer_strctR7[r9];
```

```
maccs RhighR4,RlowR6,K7TwentyfourR11,State7TwentyfourR3;
        maccs RhighR4,RlowR6,K8TwentyfourR1,State8TwentyfourR5;

        ldc   r1, (Offset_K9_Twentyfour+90)/2;
        ldd           State9TwentyfourR9,K9TwentyfourR2,Pointer_strctR7[r1];
        ldc   r5, (Offset_K10_Twentyfour+90)/2;
        ldd           State10TwentyfourR10,K10TwentyfourR3,Pointer_strctR7[r5];
        maccs RhighR4,RlowR6,K9TwentyfourR2,State9TwentyfourR9;
        maccs RhighR4,RlowR6,K10TwentyfourR3,State10TwentyfourR10;

        ldc   r2, (Offset_K11_Twentyfour+90)/2;
        ldd           State11TwentyfourR11,K11TwentyfourR5,Pointer_strctR7[r2];
        ldc   r3, (Offset_K12_Twentyfour+90)/2;
        ldd           State12TwentyfourR1,K12TwentyfourR9,Pointer_strctR7[r3];
        maccs RhighR4,RlowR6,K11TwentyfourR5,State11TwentyfourR11;
              maccs RhighR4,RlowR6,K12TwentyfourR9,State12TwentyfourR1;

              lextract r9, RhighR4, RlowR6, r8, 32;
              ldc             r1, (Offset_OutputTwentyfour_rd1+90);
              stw             r9,Pointer_strctR7[r1];

              ldc             r2, (Offset_Pointer_Twentyfour+90)/2;
              ldd             SizeTwentyfourR11,Ptr_wrR0,Pointer_strctR7[r2];

              ldc             r5,(Offset_Pointer_CurrentTwentyfour_wr1+90);
              ldw             Ptr_wr_currentR1, Pointer_strctR7[r5];

              st16            r9, Ptr_wrR0[Ptr_wr_currentR1];
              add             Ptr_wr_currentR1, Ptr_wr_currentR1, 1;
              eq              r9, Ptr_wr_currentR1, SizeTwentyfourR11;
              bf              r9, ambience2_biquad2;
              ldc     Ptr_wr_currentR1, 0x0;

ambience2_biquad2:
              ldc             r3,(Offset_Pointer_CurrentTwentyfour_wr1+90);
              stw             Ptr_rd_currentR1,Pointer_strctR7[r3];

              {ldc    RhighR4, 0;
                      ldc RlowR6, 0}
              ldc             r9,(Offset_OutputTwentyfour_rd1+12);
              ldw             OutputTwentyfourR10,Pointer_strctR7[r9];
              ldc             r5,(Offset_EarlyRightVol+347);
              ldw             EarlyRightVolR2,Pointer_strctR7[r5];
              maccs RhighR4,RlowR6,EarlyRightVolR2,OutputTwentyfourR10;
              ldc             r9,(Offset_OutputTwentyfour_rd1+90);
              ldw             OutputTwentyfourR10,Pointer_strctR7[r9];
              ldc             r5,(Offset_EarlyLeftVol+347);
              ldw             EarlyLeftVolR3,Pointer_strctR7[r5];
              maccs RhighR4,RlowR6,EarlyLeftVolR3,OutputTwentyfourR10;

              lextract r2, RhighR4, RlowR6, r8, 32;

              ldc     r3,(Offset_InputBq+155);
              stw     r2,Pointer_strctR7[r3];
```

```
        {ldc    RhighR4, 0;
                ldc RlowR6, 0}                                          //
        ldc     r3,(Offset_InputBq+155)/2;
        ldd             Ka0_bqDelr1,In_bqDelr0,Pointer_strctR7[r3];
        maccs RhighR4,RlowR6,Ka0_bqDelr1,In_bqDelr0;
        ldc             r10,(Offset_Ka1+155)/2;
        ldd             Ka2_bqDelr2,Ka1_bqDelr3,Pointer_strctR7[r10];
        ldc             r9,(Offset_sta1+155)/2;
        ldd             Sta2_bqDelr8,Sta1_bqDelr5,Pointer_strctR7[r9];
        maccs RhighR4,RlowR6,Ka1_bqDelr3,Sta1_bqDelr5;
        maccs RhighR4,RlowR6,Ka2_bqDelr2,Sta2_bqDelr8;
        ldc             r1, MACNORM;
        ldc             r9,(Offset_kb1+155)/2;
        ldd             Kb2_bqDelr2,Kb1_bqDelr3,Pointer_strctR7[r9];
        ldc             r11,(Offset_stb1+155)/2;
        ldd             Stb2_bqDelr8,Stb1_bqDelr5,Pointer_strctR7[r11];
        maccs RhighR4,RlowR6,Kb1_bqDelr3,Stb1_bqDelr5;
        maccs RhighR4,RlowR6,Kb2_bqDelr2,Stb2_bqDelr8;
        lextract r0, RhighR4, RlowR6, r1, 32;

        ldc             r10,(Offset_InputComb+180);
        stw             r0,Pointer_strctR7[r10];
        ldc             r10,(Offset_InputComb+192);
        stw             r0,Pointer_strctR7[r10];
        ldc             r10,(Offset_InputComb+204);
        stw             r0,Pointer_strctR7[r10];
        ldc             r10,(Offset_InputComb+216);
        stw             r0,Pointer_strctR7[r10];
        ldc             r10,(Offset_InputComb+228);
        stw             r0,Pointer_strctR7[r10];
        ldc             r10,(Offset_InputComb+240);
        stw             r0,Pointer_strctR7[r10];
        ldc             r10,(Offset_InputComb+252);
        stw             r0,Pointer_strctR7[r10];
        ldc             r10,(Offset_InputComb+264);
        stw             r0,Pointer_strctR7[r10];
        ldc             r10,(Offset_InputComb+276);
        stw             r0,Pointer_strctR7[r10];
        ldc             r10,(Offset_InputComb+288);
        stw             r0,Pointer_strctR7[r10];
        ldc             r10,(Offset_InputComb+300);
        stw             r0,Pointer_strctR7[r10];

ambience2_main_comb_1:
        ldc             r2,(Offset_Pointer_CurrentComb_rd1+180);
        ldw             Ptr_rd_currentR1,Pointer_strctR7[r2];
        ldc             r2,(Offset_Pointer_Comb_rd1+180)/2;
        ldd             SizeDelayCombR11,Ptr_rdR0,Pointer_strctR7[r2];
        ld16s   XnmCombR10,Ptr_rdR0[Ptr_rd_currentR1];
        add     Ptr_rd_currentR1,Ptr_rd_currentR1,1;
        eq              r9, Ptr_rd_currentR1, SizeDelayCombR11;
        bf              r9, ambience2_main_comb_maccs_1;
        ldc     Ptr_rd_currentR1, 0x0;

ambience2_main_comb_maccs_1:
        ldc             r2, (Offset_Pointer_CurrentComb_rd1+180);
```

```
                stw             Ptr_rd_currentR1, Pointer_strctR7[r2];

                ldc             r9, (Offset_InputComb+180);
                ldw         InputCombR0,Pointer_strctR7[r9];
                ldc             r11, (Offset_KOutputComb+180);
                ldw             KOutputCombR9,Pointer_strctR7[r11];

                ldc             r8, MACNORM;

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,KOutputCombR9,XnmCombR10;
                lextract r3, RhighR4, RlowR6, r8, 32;
                ldc             r9,(Offset_OutputComb_rd1+180)/2;
                std Ptr_rd_currentR1,r3,Pointer_strctR7[r9];

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                ldc             r11, (Offset_KOutputComb+180);
                ldw             KOutputCombR9,Pointer_strctR7[r11];
                ldc             r3, (Offset_KFeedbackComb+180)/2;
                ldd             KDump1CombR2, KFeedbackCombR1, Pointer_strctR7[r3];
                maccs RhighR4,RlowR6,KFeedbackCombR1,KOutputCombR9;
                lextract r10, RhighR4, RlowR6, r8, 32;

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                ldc             r9, (Offset_KDump2+180)/2;
                ldd             StateDumpCombR5, KDump2CombR3, Pointer_strctR7[r9];
                maccs RhighR4,RlowR6,KDump1CombR2,r10;
                maccs RhighR4,RlowR6,KDump2CombR3,StateDumpCombR5;
                lextract r11, RhighR4, RlowR6, r8, 32;
                ldc             r5,(Offset_StateDump+182);
                stw r11, Pointer_strctR7[r5];

                {add RhighR4,r11,0;                                     ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,InputCombR0,SizeDelayCombR11;
                lextract r5, RhighR4, RlowR6, r8, 32;

                {ldc    RhighR4, 0;                                                 ldc
RlowR6, 0;}
                ldc             r10, (Offset_KOutputComb+180);
                ldw             KOutputCombR9,Pointer_strctR7[r10];
                maccs RhighR4, RlowR6,KOutputCombR9,r5;
                lextract r9, RhighR4, RlowR6, r8, 32;

                ldc             r3,(Offset_Pointer_Comb_rd1+180);
                ldw         Ptr_wrR0,Pointer_strctR7[r3];
                ldc             r3,(Offset_Pointer_CurrentComb_wr1+180);
                ldw             Ptr_wr_currentR1, Pointer_strctR7[r3];
                st16    r9,Ptr_wrR0[Ptr_wr_currentR1];

                add     Ptr_wr_currentR1,Ptr_wr_currentR1,1;
                eq              r2, Ptr_wr_currentR1, SizeDelayCombR11;
                bf              r2, ambience2_main_comb_2;
                ldc     Ptr_wr_currentR1, 0x0;

ambience2_main_comb_2:
                ldc             r2,(Offset_Pointer_CurrentComb_wr1+180);
```

```
                stw             Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldc             r2,(Offset_Pointer_CurrentComb_rd1+192);
                ldw             Ptr_rd_currentR1,Pointer_strctR7[r2];
                ldc             r2,(Offset_Pointer_Comb_rd1+192)/2;
                ldd             SizeDelayCombR11,Ptr_rdR0,Pointer_strctR7[r2];
                ld16s           XnmCombR10,Ptr_rdR0[Ptr_rd_currentR1];
                add             Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq              r9, Ptr_rd_currentR1, SizeDelayCombR11;
                bf              r9, ambience2_main_comb_maccs_2;
                ldc             Ptr_rd_currentR1, 0x0;

ambience2_main_comb_maccs_2:
                ldc             r9, (Offset_Pointer_CurrentComb_rd1+192);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

                ldc             r9, (Offset_InputComb+192);
                ldw         InputCombR0,Pointer_strctR7[r9];
                ldc             r11, (Offset_KOutputComb+192);
                ldw             KOutputCombR9,Pointer_strctR7[r11];

                ldc             r8, MACNORM;

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,KOutputCombR9,XnmCombR10;
                lextract r3, RhighR4, RlowR6, r8, 32;
                ldc             r9,(Offset_OutputComb_rd1+192)/2;
                std Ptr_rd_currentR1,r3,Pointer_strctR7[r9];

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                ldc             r11, (Offset_KOutputComb+192);
                ldw             KOutputCombR9,Pointer_strctR7[r11];
                ldc             r3, (Offset_KFeedbackComb+192)/2;
                ldd             KDump1CombR2, KFeedbackCombR1, Pointer_strctR7[r3];
                maccs RhighR4,RlowR6,KFeedbackCombR1,KOutputCombR9;
                lextract r10, RhighR4, RlowR6, r8, 32;

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                ldc             r9, (Offset_KDump2+192)/2;
                ldd             StateDumpCombR5, KDump2CombR3, Pointer_strctR7[r9];
                maccs RhighR4,RlowR6,KDump1CombR2,r10;
                maccs RhighR4,RlowR6,KDump2CombR3,StateDumpCombR5;
                lextract r11, RhighR4, RlowR6, r8, 32;
                ldc             r5,(Offset_StateDump+192);
                stw r11, Pointer_strctR7[r5];

                {add RhighR4,r11,0;                                         ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,InputCombR0,SizeDelayCombR11;
                lextract r5, RhighR4, RlowR6, r8, 32;

                {ldc    RhighR4, 0;                                                 ldc
RlowR6, 0;}
                ldc             r10, (Offset_KOutputComb+192);
                ldw             KOutputCombR9,Pointer_strctR7[r10];
                maccs RhighR4, RlowR6,KOutputCombR9,r5;
                lextract r9, RhighR4, RlowR6, r8, 32;
```

```
        {add RhighR4,r9,0;                                          ldc RlowR6, 0;}
        ldc             r3,(Offset_Pointer_Comb_rd1+192);
        ldw         Ptr_wrR0,Pointer_strctR7[r3];
        ldc             r3,(Offset_Pointer_CurrentComb_wr1+192);
        ldw             Ptr_wr_currentR1, Pointer_strctR7[r3];
        st16    r9,Ptr_wrR0[Ptr_wr_currentR1];

        add     Ptr_wr_currentR1,Ptr_wr_currentR1,1;
        eq              r2, Ptr_wr_currentR1, SizeDelayCombR11;
        bf              r2, ambience2_main_comb_3;
        ldc     Ptr_wr_currentR1, 0x0;

ambience2_main_comb_3:
        ldc             r9, (Offset_Pointer_CurrentComb_wr1+192);
        stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

        ldc             r2,(Offset_Pointer_CurrentComb_rd1+204);
        ldw             Ptr_rd_currentR1,Pointer_strctR7[r2];
        ldc             r2,(Offset_Pointer_Comb_rd1+204)/2;
        ldd             SizeDelayCombR11,Ptr_rdR0,Pointer_strctR7[r2];
        ld16s   XnmCombR10,Ptr_rdR0[Ptr_rd_currentR1];
        add     Ptr_rd_currentR1,Ptr_rd_currentR1,1;
        eq              r9, Ptr_rd_currentR1, SizeDelayCombR11;
        bf              r9, ambience2_main_comb_maccs_3;
        ldc     Ptr_rd_currentR1, 0x0;

ambience2_main_comb_maccs_3:
        ldc             r9, (Offset_Pointer_CurrentComb_rd1+204);
        stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

        ldc             r9, (Offset_InputComb+204);
        ldw         InputCombR0,Pointer_strctR7[r9];
        ldc             r11, (Offset_KOutputComb+204);
        ldw             KOutputCombR9,Pointer_strctR7[r11];

        ldc             r8, MACNORM;

        {ldc    RhighR4, 0;                                         ldc RlowR6, 0;}
        maccs RhighR4,RlowR6,KOutputCombR9,XnmCombR10;
        lextract r3, RhighR4, RlowR6, r8, 32;
        ldc             r9,(Offset_OutputComb_rd1+204)/2;
        std Ptr_rd_currentR1,r3,Pointer_strctR7[r9];

        {ldc    RhighR4, 0;                                         ldc RlowR6, 0;}
        ldc             r11, (Offset_KOutputComb+204);
        ldw             KOutputCombR9,Pointer_strctR7[r11];
        ldc             r3, (Offset_KFeedbackComb+204)/2;
        ldd             KDump1CombR2, KFeedbackCombR1, Pointer_strctR7[r3];
        maccs RhighR4,RlowR6,KFeedbackCombR1,KOutputCombR9;
        lextract r10, RhighR4, RlowR6, r8, 32;

        {ldc    RhighR4, 0;                                         ldc RlowR6, 0;}
        ldc             r9, (Offset_KDump2+204)/2;
        ldd             StateDumpCombR5, KDump2CombR3, Pointer_strctR7[r9];
        maccs RhighR4,RlowR6,KDump1CombR2,r10;
```

```
                maccs RhighR4,RlowR6,KDump2CombR3,StateDumpCombR5;
                lextract r11, RhighR4, RlowR6, r8, 32;
                ldc             r5,(Offset_StateDump+204);
                stw r11, Pointer_strctR7[r5];

                {add RhighR4,r11,0;                                     ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,InputCombR0,SizeDelayCombR11;
                lextract r5, RhighR4, RlowR6, r8, 32;

                {ldc    RhighR4, 0;                                                     ldc
RlowR6, 0;}
                ldc             r10, (Offset_KOutputComb+204);
                ldw             KOutputCombR9,Pointer_strctR7[r10];
                maccs RhighR4, RlowR6,KOutputCombR9,r5;
                lextract r9, RhighR4, RlowR6, r8, 32;

                {add RhighR4,r9,0;                                      ldc RlowR6, 0;}
                ldc             r3,(Offset_Pointer_Comb_rd1+204);
                ldw       Ptr_wrR0,Pointer_strctR7[r3];
                ldc             r3,(Offset_Pointer_CurrentComb_wr1+204);
                ldw             Ptr_wr_currentR1, Pointer_strctR7[r3];
                st16    r9,Ptr_wrR0[Ptr_wr_currentR1];

                add     Ptr_wr_currentR1,Ptr_wr_currentR1,1;
                eq              r2, Ptr_wr_currentR1, SizeDelayCombR11;
                bf              r2, ambience2_main_comb_4;
                ldc     Ptr_wr_currentR1, 0x0;

ambience2_main_comb_4:
                ldc             r9, (Offset_Pointer_CurrentComb_wr1+204);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

                ldc             r2,(Offset_Pointer_CurrentComb_rd1+216);
                ldw             Ptr_rd_currentR1,Pointer_strctR7[r2];
                ldc             r2,(Offset_Pointer_Comb_rd1+216)/2;
                ldd             SizeDelayCombR11,Ptr_rdR0,Pointer_strctR7[r2];
                ld16s   XnmCombR10,Ptr_rdR0[Ptr_rd_currentR1];
                add     Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq              r9, Ptr_rd_currentR1, SizeDelayCombR11;
                bf              r9, ambience2_main_comb_maccs_4;
                ldc     Ptr_rd_currentR1, 0x0;

ambience2_main_comb_maccs_4:
                ldc             r9, (Offset_Pointer_CurrentComb_rd1+216);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

                ldc             r9, (Offset_InputComb+216);
                ldw       InputCombR0,Pointer_strctR7[r9];
                ldc             r11, (Offset_KOutputComb+216);
                ldw             KOutputCombR9,Pointer_strctR7[r11];

                ldc             r8, MACNORM;

                {ldc    RhighR4, 0;                                     ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,KOutputCombR9,XnmCombR10;
                lextract r3, RhighR4, RlowR6, r8, 32;
```

```
                ldc             r9,(Offset_OutputComb_rd1+216)/2;
                std Ptr_rd_currentR1,r3,Pointer_strctR7[r9];

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                ldc             r11, (Offset_KOutputComb+216);
                ldw             KOutputCombR9,Pointer_strctR7[r11];
                ldc             r3, (Offset_KFeedbackComb+216)/2;
                ldd             KDump1CombR2, KFeedbackCombR1, Pointer_strctR7[r3];
                maccs RhighR4,RlowR6,KFeedbackCombR1,KOutputCombR9;
                lextract r10, RhighR4, RlowR6, r8, 32;

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                ldc             r9, (Offset_KDump2+216)/2;
                ldd             StateDumpCombR5, KDump2CombR3, Pointer_strctR7[r9];
                maccs RhighR4,RlowR6,KDump1CombR2,r10;
                maccs RhighR4,RlowR6,KDump2CombR3,StateDumpCombR5;
                lextract r11, RhighR4, RlowR6, r8, 32;
                ldc             r5,(Offset_StateDump+216);
                stw r11, Pointer_strctR7[r5];

                {add RhighR4,r11,0;                                     ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,InputCombR0,SizeDelayCombR11;
                lextract r5, RhighR4, RlowR6, r8, 32;

                {ldc    RhighR4, 0;                                                     ldc
RlowR6, 0;}
                ldc             r10, (Offset_KOutputComb+216);
                ldw             KOutputCombR9,Pointer_strctR7[r10];
                maccs RhighR4, RlowR6,KOutputCombR9,r5;
                lextract r9, RhighR4, RlowR6, r8, 32;

                {add RhighR4,r9,0;                                              ldc RlowR6, 0;}
                ldc             r3,(Offset_Pointer_Comb_rd1+216);
                ldw         Ptr_wrR0,Pointer_strctR7[r3];
                ldc             r3,(Offset_Pointer_CurrentComb_wr1+216);
                ldw             Ptr_wr_currentR1, Pointer_strctR7[r3];
                st16    r9,Ptr_wrR0[Ptr_wr_currentR1];

                add     Ptr_wr_currentR1,Ptr_wr_currentR1,1;
                eq              r2, Ptr_wr_currentR1, SizeDelayCombR11;
                bf              r2, ambience2_main_comb_5;
                ldc     Ptr_wr_currentR1, 0x0;

ambience2_main_comb_5:
                ldc             r9, (Offset_Pointer_CurrentComb_wr1+216);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

                ldc             r2,(Offset_Pointer_CurrentComb_rd1+228);
                ldw             Ptr_rd_currentR1,Pointer_strctR7[r2];
                ldc             r2,(Offset_Pointer_Comb_rd1+228)/2;
                ldd             SizeDelayCombR11,Ptr_rdR0,Pointer_strctR7[r2];
                ld16s   XnmCombR10,Ptr_rdR0[Ptr_rd_currentR1];
                add     Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq              r9, Ptr_rd_currentR1, SizeDelayCombR11;
                bf              r9, ambience2_main_comb_maccs_5;
                ldc     Ptr_rd_currentR1, 0x0;
```

```
ambience2_main_comb_maccs_5:
                ldc             r9, (Offset_Pointer_CurrentComb_rd1+228);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

                ldc             r9, (Offset_InputComb+228);
                ldw         InputCombR0,Pointer_strctR7[r9];
                ldc             r11, (Offset_KOutputComb+228);
                ldw             KOutputCombR9,Pointer_strctR7[r11];

                ldc             r8, MACNORM;

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,KOutputCombR9,XnmCombR10;
                lextract r3, RhighR4, RlowR6, r8, 32;
                ldc             r9,(Offset_OutputComb_rd1+228)/2;
                std Ptr_rd_currentR1,r3,Pointer_strctR7[r9];

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                ldc             r11, (Offset_KOutputComb+228);
                ldw             KOutputCombR9,Pointer_strctR7[r11];
                ldc             r3, (Offset_KFeedbackComb+228)/2;
                ldd             KDump1CombR2, KFeedbackCombR1, Pointer_strctR7[r3];
                maccs RhighR4,RlowR6,KFeedbackCombR1,KOutputCombR9;
                lextract r10, RhighR4, RlowR6, r8, 32;

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                ldc             r9, (Offset_KDump2+228)/2;
                ldd             StateDumpCombR5, KDump2CombR3, Pointer_strctR7[r9];
                maccs RhighR4,RlowR6,KDump1CombR2,r10;
                maccs RhighR4,RlowR6,KDump2CombR3,StateDumpCombR5;
                lextract r11, RhighR4, RlowR6, r8, 32;
                ldc             r5,(Offset_StateDump+228);
                stw r11, Pointer_strctR7[r5];

                {add RhighR4,r11,0;                                         ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,InputCombR0,SizeDelayCombR11;
                lextract r5, RhighR4, RlowR6, r8, 32;

                {ldc    RhighR4, 0;                                                 ldc
RlowR6, 0;}
                ldc             r10, (Offset_KOutputComb+228);
                ldw             KOutputCombR9,Pointer_strctR7[r10];
                maccs RhighR4, RlowR6,KOutputCombR9,r5;
                lextract r9, RhighR4, RlowR6, r8, 32;

                {add RhighR4,r9,0;                                              ldc RlowR6, 0;}
                ldc             r3,(Offset_Pointer_Comb_rd1+228);
                ldw         Ptr_wrR0,Pointer_strctR7[r3];
                ldc             r3,(Offset_Pointer_CurrentComb_wr1+228);
                ldw             Ptr_wr_currentR1, Pointer_strctR7[r3];
                st16    r9,Ptr_wrR0[Ptr_wr_currentR1];

                add     Ptr_wr_currentR1,Ptr_wr_currentR1,1;
                eq              r2, Ptr_wr_currentR1, SizeDelayCombR11;
                bf              r2, ambience2_main_comb_6;
```

```
                ldc     Ptr_wr_currentR1, 0x0;

ambience2_main_comb_6:
                ldc             r9, (Offset_Pointer_CurrentComb_wr1+228);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

                ldc             r2,(Offset_Pointer_CurrentComb_rd1+240);
                ldw             Ptr_rd_currentR1,Pointer_strctR7[r2];
                ldc             r2,(Offset_Pointer_Comb_rd1+240)/2;
                ldd             SizeDelayCombR11,Ptr_rdR0,Pointer_strctR7[r2];
                ld16s   XnmCombR10,Ptr_rdR0[Ptr_rd_currentR1];
                add     Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq              r9, Ptr_rd_currentR1, SizeDelayCombR11;
                bf              r9, ambience2_main_comb_maccs_6;
                ldc     Ptr_rd_currentR1, 0x0;

ambience2_main_comb_maccs_6:
                ldc             r9, (Offset_Pointer_CurrentComb_rd1+240);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

                ldc             r9, (Offset_InputComb+240);
                ldw         InputCombR0,Pointer_strctR7[r9];
                ldc             r11, (Offset_KOutputComb+240);
                ldw             KOutputCombR9,Pointer_strctR7[r11];

                ldc             r8, MACNORM;

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,KOutputCombR9,XnmCombR10;
                lextract r3, RhighR4, RlowR6, r8, 32;
                ldc             r9,(Offset_OutputComb_rd1+240)/2;
                std Ptr_rd_currentR1,r3,Pointer_strctR7[r9];

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                ldc             r11, (Offset_KOutputComb+240);
                ldw             KOutputCombR9,Pointer_strctR7[r11];
                ldc             r3, (Offset_KFeedbackComb+240)/2;
                ldd             KDump1CombR2, KFeedbackCombR1, Pointer_strctR7[r3];
                maccs RhighR4,RlowR6,KFeedbackCombR1,KOutputCombR9;
                lextract r10, RhighR4, RlowR6, r8, 32;

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                ldc             r9, (Offset_KDump2+240)/2;
                ldd             StateDumpCombR5, KDump2CombR3, Pointer_strctR7[r9];
                maccs RhighR4,RlowR6,KDump1CombR2,r10;
                maccs RhighR4,RlowR6,KDump2CombR3,StateDumpCombR5;
                lextract r11, RhighR4, RlowR6, r8, 32;
                ldc             r5,(Offset_StateDump+240);
                stw r11, Pointer_strctR7[r5];

                {add RhighR4,r11,0;                                     ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,InputCombR0,SizeDelayCombR11;
                lextract r5, RhighR4, RlowR6, r8, 32;

                {ldc    RhighR4, 0;                                                     ldc
RlowR6, 0;}
```

```
                ldc             r10, (Offset_KOutputComb+240);
                ldw             KOutputCombR9,Pointer_strctR7[r10];
                maccs RhighR4, RlowR6,KOutputCombR9,r5;
                lextract r9, RhighR4, RlowR6, r8, 32;

                {add RhighR4,r9,0;                                              ldc RlowR6, 0;}
                ldc             r3,(Offset_Pointer_Comb_rd1+240);
                ldw       Ptr_wrR0,Pointer_strctR7[r3];
                ldc             r3,(Offset_Pointer_CurrentComb_wr1+240);
                ldw             Ptr_wr_currentR1, Pointer_strctR7[r3];
                st16    r9,Ptr_wrR0[Ptr_wr_currentR1];

                add     Ptr_wr_currentR1,Ptr_wr_currentR1,1;
                eq              r2, Ptr_wr_currentR1, SizeDelayCombR11;
                bf              r2, ambience2_main_comb_7;
                ldc     Ptr_wr_currentR1, 0x0;

ambience2_main_comb_7:
                ldc             r9, (Offset_Pointer_CurrentComb_wr1+240);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

                ldc             r2,(Offset_Pointer_CurrentComb_rd1+252);
                ldw             Ptr_rd_currentR1,Pointer_strctR7[r2];
                ldc             r2,(Offset_Pointer_Comb_rd1+252)/2;
                ldd             SizeDelayCombR11,Ptr_rdR0,Pointer_strctR7[r2];
                ld16s   XnmCombR10,Ptr_rdR0[Ptr_rd_currentR1];
                add     Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq              r9, Ptr_rd_currentR1, SizeDelayCombR11;
                bf              r9, ambience2_main_comb_maccs_7;
                ldc     Ptr_rd_currentR1, 0x0;

ambience2_main_comb_maccs_7:
                ldc             r9, (Offset_Pointer_CurrentComb_rd1+252);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

                ldc             r9, (Offset_Pointer_CurrentComb_rd1+252);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

                ldc             r9, (Offset_InputComb+252);
                ldw       InputCombR0,Pointer_strctR7[r9];
                ldc             r11, (Offset_KOutputComb+252);
                ldw             KOutputCombR9,Pointer_strctR7[r11];

                ldc             r8, MACNORM;

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,KOutputCombR9,XnmCombR10;
                lextract r3, RhighR4, RlowR6, r8, 32;
                ldc             r9,(Offset_OutputComb_rd1+252)/2;
                std Ptr_rd_currentR1,r3,Pointer_strctR7[r9];

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                ldc             r11, (Offset_KOutputComb+252);
                ldw             KOutputCombR9,Pointer_strctR7[r11];
                ldc             r3, (Offset_KFeedbackComb+252)/2;
                ldd             KDump1CombR2, KFeedbackCombR1, Pointer_strctR7[r3];
```

```
                maccs RhighR4,RlowR6,KFeedbackCombR1,KOutputCombR9;
                lextract r10, RhighR4, RlowR6, r8, 32;

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                ldc             r9, (Offset_KDump2+252)/2;
                ldd             StateDumpCombR5, KDump2CombR3, Pointer_strctR7[r9];
                maccs RhighR4,RlowR6,KDump1CombR2,r10;
                maccs RhighR4,RlowR6,KDump2CombR3,StateDumpCombR5;
                lextract r11, RhighR4, RlowR6, r8, 32;
                ldc             r5,(Offset_StateDump+252);
                stw r11, Pointer_strctR7[r5];

                {add RhighR4,r11,0;                                         ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,InputCombR0,SizeDelayCombR11;
                lextract r5, RhighR4, RlowR6, r8, 32;

                {ldc    RhighR4, 0;                                                 ldc
RlowR6, 0;}
                ldc             r11, (Offset_KOutputComb+252);
                ldw             KOutputCombR9,Pointer_strctR7[r11];
                maccs RhighR4, RlowR6,KOutputCombR9,r5;
                lextract r9, RhighR4, RlowR6, r8, 32;

                {add RhighR4,r9,0;                                              ldc RlowR6, 0;}
                ldc             r3,(Offset_Pointer_Comb_rd1+252);
                ldw       Ptr_wrR0,Pointer_strctR7[r3];
                ldc             r3,(Offset_Pointer_CurrentComb_wr1+252);
                ldw             Ptr_wr_currentR1, Pointer_strctR7[r3];
                st16    r9,Ptr_wrR0[Ptr_wr_currentR1];

                add     Ptr_wr_currentR1,Ptr_wr_currentR1,1;
                eq              r2, Ptr_wr_currentR1, SizeDelayCombR11;
                bf              r2, ambience2_right_comb_1;
                ldc     Ptr_wr_currentR1, 0x0;

ambience2_right_comb_1:
                ldc             r9, (Offset_Pointer_CurrentComb_wr1+252);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

                ldc             r2,(Offset_Pointer_CurrentComb_rd1+264);
                ldw             Ptr_rd_currentR1,Pointer_strctR7[r2];
                ldc             r2,(Offset_Pointer_Comb_rd1+264)/2;
                ldd             SizeDelayCombR11,Ptr_rdR0,Pointer_strctR7[r2];
                ld16s   XnmCombR10,Ptr_rdR0[Ptr_rd_currentR1];
                add     Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq              r9, Ptr_rd_currentR1, SizeDelayCombR11;
                bf              r9, ambience2_right_comb_maccs_1;
                ldc     Ptr_rd_currentR1, 0x0;

ambience2_right_comb_maccs_1:
                ldc             r9, (Offset_Pointer_CurrentComb_rd1+264);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

                ldc             r9, (Offset_Pointer_CurrentComb_rd1+264);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];
```

```
ldc             r9, (Offset_InputComb+264);
ldw       InputCombR0,Pointer_strctR7[r9];
ldc             r11, (Offset_KOutputComb+264);
ldw             KOutputCombR9,Pointer_strctR7[r11];

ldc             r8, MACNORM;

{ldc    RhighR4, 0;                                          ldc RlowR6, 0;}
maccs RhighR4,RlowR6,KOutputCombR9,XnmCombR10;
lextract r3, RhighR4, RlowR6, r8, 32;
ldc             r9,(Offset_OutputComb_rd1+264)/2;
std Ptr_rd_currentR1,r3,Pointer_strctR7[r9];

{ldc    RhighR4, 0;                                          ldc RlowR6, 0;}
ldc             r11, (Offset_KOutputComb+264);
ldw             KOutputCombR9,Pointer_strctR7[r11];
ldc             r3, (Offset_KFeedbackComb+264)/2;
ldd             KDump1CombR2, KFeedbackCombR1, Pointer_strctR7[r3];
maccs RhighR4,RlowR6,KFeedbackCombR1,KOutputCombR9;
lextract r10, RhighR4, RlowR6, r8, 32;

{ldc    RhighR4, 0;                                          ldc RlowR6, 0;}
ldc             r9, (Offset_KDump2+264)/2;
ldd             StateDumpCombR5, KDump2CombR3, Pointer_strctR7[r9];
maccs RhighR4,RlowR6,KDump1CombR2,r10;
maccs RhighR4,RlowR6,KDump2CombR3,StateDumpCombR5;
lextract r11, RhighR4, RlowR6, r8, 32;
ldc             r5,(Offset_StateDump+264);
stw r11, Pointer_strctR7[r5];

{add RhighR4,r11,0;                                    ldc RlowR6, 0;}
maccs RhighR4,RlowR6,InputCombR0,SizeDelayCombR11;
lextract r5, RhighR4, RlowR6, r8, 32;

{ldc    RhighR4, 0;                                                ldc
RlowR6, 0;}
ldc             r10, (Offset_KOutputComb+264);
ldw             KOutputCombR9,Pointer_strctR7[r10];
maccs RhighR4, RlowR6,KOutputCombR9,r5;
lextract r9, RhighR4, RlowR6, r8, 32;

{add RhighR4,r9,0;                                          ldc RlowR6, 0;}
ldc             r3,(Offset_Pointer_Comb_rd1+264);
ldw       Ptr_wrR0,Pointer_strctR7[r3];
ldc             r3,(Offset_Pointer_CurrentComb_wr1+264);
ldw             Ptr_wr_currentR1, Pointer_strctR7[r3];
st16    r9,Ptr_wrR0[Ptr_wr_currentR1];

add     Ptr_wr_currentR1,Ptr_wr_currentR1,1;
eq              r2, Ptr_wr_currentR1, SizeDelayCombR11;
bf              r2, ambience2_right_comb_2;
ldc     Ptr_wr_currentR1, 0x0;

ambience2_right_comb_2:
ldc             r9, (Offset_Pointer_CurrentComb_wr1+264);
stw             Ptr_rd_currentR1, Pointer_strctR7[r9];
```

```
                ldc             r2,(Offset_Pointer_CurrentComb_rd1+276);
                ldw             Ptr_rd_currentR1,Pointer_strctR7[r2];
                ldc             r2,(Offset_Pointer_Comb_rd1+276)/2;
                ldd             SizeDelayCombR11,Ptr_rdR0,Pointer_strctR7[r2];
                ld16s   XnmCombR10,Ptr_rdR0[Ptr_rd_currentR1];
                add     Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq              r9, Ptr_rd_currentR1, SizeDelayCombR11;
                bf              r9, ambience2_right_comb_maccs_2;
                ldc     Ptr_rd_currentR1, 0x0;

ambience2_right_comb_maccs_2:
                ldc             r9, (Offset_Pointer_CurrentComb_rd1+276);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

                ldc             r9, (Offset_Pointer_CurrentComb_rd1+276);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

                ldc             r9, (Offset_InputComb+276);
                ldw         InputCombR0,Pointer_strctR7[r9];
                ldc             r11, (Offset_KOutputComb+276);
                ldw             KOutputCombR9,Pointer_strctR7[r11];

                ldc             r8, MACNORM;

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,KOutputCombR9,XnmCombR10;
                lextract r3, RhighR4, RlowR6, r8, 32;
                ldc             r9,(Offset_OutputComb_rd1+276)/2;
                std Ptr_rd_currentR1,r3,Pointer_strctR7[r9];

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                ldc             r11, (Offset_KOutputComb+276);
                ldw             KOutputCombR9,Pointer_strctR7[r11];
                ldc             r3, (Offset_KFeedbackComb+276)/2;
                ldd             KDump1CombR2, KFeedbackCombR1, Pointer_strctR7[r3];
                maccs RhighR4,RlowR6,KFeedbackCombR1,KOutputCombR9;
                lextract r10, RhighR4, RlowR6, r8, 32;

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                ldc             r9, (Offset_KDump2+276)/2;
                ldd             StateDumpCombR5, KDump2CombR3, Pointer_strctR7[r9];
                maccs RhighR4,RlowR6,KDump1CombR2,r10;
                maccs RhighR4,RlowR6,KDump2CombR3,StateDumpCombR5;
                lextract r11, RhighR4, RlowR6, r8, 32;
                ldc             r5,(Offset_StateDump+276);
                stw r11, Pointer_strctR7[r5];

                {add RhighR4,r11,0;                                     ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,InputCombR0,SizeDelayCombR11;
                lextract r5, RhighR4, RlowR6, r8, 32;

                {ldc    RhighR4, 0;                                                     ldc
RlowR6, 0;}
                ldc             r10, (Offset_KOutputComb+276);
                ldw             KOutputCombR9,Pointer_strctR7[r10];
```

```
                maccs RhighR4, RlowR6,KOutputCombR9,r5;
                lextract r9, RhighR4, RlowR6, r8, 32;

                {add RhighR4,r9,0;                                              ldc RlowR6, 0;}
                ldc             r3,(Offset_Pointer_Comb_rd1+276);
                ldw         Ptr_wrR0,Pointer_strctR7[r3];
                ldc             r3,(Offset_Pointer_CurrentComb_wr1+276);
                ldw             Ptr_wr_currentR1, Pointer_strctR7[r3];
                st16    r9,Ptr_wrR0[Ptr_wr_currentR1];

                add     Ptr_wr_currentR1,Ptr_wr_currentR1,1;
                eq              r2, Ptr_wr_currentR1, SizeDelayCombR11;
                bf              r2, ambience2_left_comb_1;
                ldc     Ptr_wr_currentR1, 0x0;

ambience2_left_comb_1:
                ldc             r9, (Offset_Pointer_CurrentComb_wr1+276);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

                ldc             r2,(Offset_Pointer_CurrentComb_rd1+288);
                ldw             Ptr_rd_currentR1,Pointer_strctR7[r2];
                ldc             r2,(Offset_Pointer_Comb_rd1+288)/2;
                ldd             SizeDelayCombR11,Ptr_rdR0,Pointer_strctR7[r2];
                ld16s   XnmCombR10,Ptr_rdR0[Ptr_rd_currentR1];
                add     Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq              r9, Ptr_rd_currentR1, SizeDelayCombR11;
                bf              r9, ambience2_left_comb_maccs_1;
                ldc     Ptr_rd_currentR1, 0x0;

ambience2_left_comb_maccs_1:
                ldc             r9, (Offset_Pointer_CurrentComb_rd1+288);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

                ldc             r9, (Offset_Pointer_CurrentComb_rd1+288);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

                ldc             r9, (Offset_InputComb+288);
                ldw         InputCombR0,Pointer_strctR7[r9];
                ldc             r11, (Offset_KOutputComb+288);
                ldw             KOutputCombR9,Pointer_strctR7[r11];

                ldc             r8, MACNORM;

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,KOutputCombR9,XnmCombR10;
                lextract r3, RhighR4, RlowR6, r8, 32;
                ldc             r9,(Offset_OutputComb_rd1+288)/2;
                std Ptr_rd_currentR1,r3,Pointer_strctR7[r9];

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                ldc             r11, (Offset_KOutputComb+288);
                ldw             KOutputCombR9,Pointer_strctR7[r11];
                ldc             r3, (Offset_KFeedbackComb+288)/2;
                ldd             KDump1CombR2, KFeedbackCombR1, Pointer_strctR7[r3];
                maccs RhighR4,RlowR6,KFeedbackCombR1,KOutputCombR9;
                lextract r10, RhighR4, RlowR6, r8, 32;
```

```
                {ldc     RhighR4, 0;                                                    ldc RlowR6, 0;}
                ldc             r9, (Offset_KDump2+288)/2;
                ldd             StateDumpCombR5, KDump2CombR3, Pointer_strctR7[r9];
                maccs RhighR4,RlowR6,KDump1CombR2,r10;
                maccs RhighR4,RlowR6,KDump2CombR3,StateDumpCombR5;
                lextract r11, RhighR4, RlowR6, r8, 32;
                ldc             r5,(Offset_StateDump+288);
                stw r11, Pointer_strctR7[r5];

                {add RhighR4,r11,0;                                                ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,InputCombR0,SizeDelayCombR11;
                lextract r5, RhighR4, RlowR6, r8, 32;

                {ldc     RhighR4, 0;                                                        ldc
RlowR6, 0;}
                ldc             r10, (Offset_KOutputComb+288);
                ldw             KOutputCombR9,Pointer_strctR7[r10];
                maccs RhighR4, RlowR6,KOutputCombR9,r5;
                lextract r9, RhighR4, RlowR6, r8, 32;

                {add RhighR4,r9,0;                                                 ldc RlowR6, 0;}
                ldc             r3,(Offset_Pointer_Comb_rd1+288);
                ldw         Ptr_wrR0,Pointer_strctR7[r3];
                ldc             r3,(Offset_Pointer_CurrentComb_wr1+288);
                ldw             Ptr_wr_currentR1, Pointer_strctR7[r3];
                st16    r9,Ptr_wrR0[Ptr_wr_currentR1];

                add     Ptr_wr_currentR1,Ptr_wr_currentR1,1;
                eq              r2, Ptr_wr_currentR1, SizeDelayCombR11;
                bf              r2, ambience2_left_comb_2;
                ldc     Ptr_wr_currentR1, 0x0;

ambience2_left_comb_2:
                ldc             r9, (Offset_Pointer_CurrentComb_wr1+288);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

                ldc             r2,(Offset_Pointer_CurrentComb_rd1+300);
                ldw             Ptr_rd_currentR1,Pointer_strctR7[r2];
                ldc             r2,(Offset_Pointer_Comb_rd1+300)/2;
                ldd             SizeDelayCombR11,Ptr_rdR0,Pointer_strctR7[r2];
                ld16s   XnmCombR10,Ptr_rdR0[Ptr_rd_currentR1];
                add     Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq              r9, Ptr_rd_currentR1, SizeDelayCombR11;
                bf              r9, ambience2_left_comb_maccs_2;
                ldc     Ptr_rd_currentR1, 0x0;

ambience2_left_comb_maccs_2:
                ldc             r9, (Offset_Pointer_CurrentComb_rd1+300);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

                ldc             r9, (Offset_InputComb+300);
                ldw         InputCombR0,Pointer_strctR7[r9];
                ldc             r11, (Offset_KOutputComb+300);
                ldw             KOutputCombR9,Pointer_strctR7[r11];
```

```
                ldc             r8, MACNORM;

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,KOutputCombR9,XnmCombR10;
                lextract r3, RhighR4, RlowR6, r8, 32;
                ldc             r9,(Offset_OutputComb_rd1+300)/2;
                std Ptr_rd_currentR1,r3,Pointer_strctR7[r9];

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                ldc             r11, (Offset_KOutputComb+300);
                ldw             KOutputCombR9,Pointer_strctR7[r11];
                ldc             r3, (Offset_KFeedbackComb+300)/2;
                ldd             KDump1CombR2, KFeedbackCombR1, Pointer_strctR7[r3];
                maccs RhighR4,RlowR6,KFeedbackCombR1,KOutputCombR9;
                lextract r10, RhighR4, RlowR6, r8, 32;

                {ldc    RhighR4, 0;                                             ldc RlowR6, 0;}
                ldc             r9, (Offset_KDump2+300)/2;
                ldd             StateDumpCombR5, KDump2CombR3, Pointer_strctR7[r9];
                maccs RhighR4,RlowR6,KDump1CombR2,r10;
                maccs RhighR4,RlowR6,KDump2CombR3,StateDumpCombR5;
                lextract r11, RhighR4, RlowR6, r8, 32;
                ldc             r5,(Offset_StateDump+300);
                stw r11, Pointer_strctR7[r5];

                {add RhighR4,r11,0;                                     ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,InputCombR0,SizeDelayCombR11;
                lextract r5, RhighR4, RlowR6, r8, 32;

                {ldc    RhighR4, 0;                                                     ldc
RlowR6, 0;}
                ldc             r10, (Offset_KOutputComb+300);
                ldw             KOutputCombR9,Pointer_strctR7[r10];
                maccs RhighR4, RlowR6,KOutputCombR9,r5;
                lextract r9, RhighR4, RlowR6, r8, 32;

                {add RhighR4,r9,0;                                              ldc RlowR6, 0;}
                ldc             r3,(Offset_Pointer_Comb_rd1+300);
                ldw         Ptr_wrR0,Pointer_strctR7[r3];
                ldc             r3,(Offset_Pointer_CurrentComb_wr1+300);
                ldw             Ptr_wr_currentR1, Pointer_strctR7[r3];
                st16    r9,Ptr_wrR0[Ptr_wr_currentR1];

                add     Ptr_wr_currentR1,Ptr_wr_currentR1,1;
                eq              r2, Ptr_wr_currentR1, SizeDelayCombR11;
                bf              r2, ambience2_comb_output;
                ldc     Ptr_wr_currentR1, 0x0;

ambience2_comb_output:
                ldc             r9, (Offset_Pointer_CurrentComb_wr1+300);
                stw             Ptr_rd_currentR1, Pointer_strctR7[r9];

                {ldc    RhighR4, 0;                                                     ldc
RlowR6, 0;}
                ldc             r2,(Offset_OutputComb_rd1+180);
                ldw             CombMain1OutputR5, Pointer_strctR7[r2];
```

```
                ldc             r3,1;
                maccs RhighR4,RlowR6,r3,CombMain1OutputR5;
                ldc             r2,(Offset_OutputComb_rd1+192);
                ldw             CombMain2OutputR9, Pointer_strctR7[r2];
                ldc             r3,1;
                maccs RhighR4,RlowR6,r3,CombMain2OutputR9;
                ldc             r3,(Offset_OutputComb_rd1+204);
                ldw             CombMain3OutputR5, Pointer_strctR7[r3];
                ldc             r2,1;
                maccs RhighR4,RlowR6,r2,CombMain3OutputR5;
                ldc             r2,(Offset_OutputComb_rd1+216);
                ldw             CombMain4OutputR9, Pointer_strctR7[r2];
                ldc             r3,1;
                maccs RhighR4,RlowR6,r3,CombMain4OutputR9;
                ldc             r3,(Offset_OutputComb_rd1+228);
                ldw             CombMain5OutputR5, Pointer_strctR7[r3];
                ldc             r2,1;
                maccs RhighR4,RlowR6,r2,CombMain5OutputR5;
                ldc             r2,(Offset_OutputComb_rd1+240);
                ldw             CombMain6OutputR9, Pointer_strctR7[r2];
                ldc             r3,1;
                maccs RhighR4,RlowR6,r3,CombMain6OutputR9;
                ldc             r3,(Offset_OutputComb_rd1+252);
                ldw             CombMain7OutputR5, Pointer_strctR7[r3];
                ldc             r2,1;
                maccs RhighR4,RlowR6,r2,CombMain7OutputR5;
                ldc             r2,(Offset_OutputComb_rd1+264);
                ldw             CombRight1OutputR9, Pointer_strctR7[r2];
                ldc             r3,1;
                maccs RhighR4,RlowR6,r3,CombRight1OutputR9;
                ldc             r3,(Offset_OutputComb_rd1+288);
                ldw             CombRight2OutputR5, Pointer_strctR7[r3];
                ldc             r2,1;
                maccs RhighR4,RlowR6,r2,CombRight2OutputR5;

                lextract r10, RhighR4, RlowR6, r8, 32;
                ldc             r2, (Offset_InputAllpass+312);
                stw             r10, Pointer_strctR7[r2];
                {ldc    RhighR4, 0;                                                     ldc
RlowR6, 0;}
                ldc             r2,(Offset_OutputComb_rd1+180);
                ldw             CombMain1OutputR5, Pointer_strctR7[r2];
                ldc             r3,1;
                maccs RhighR4,RlowR6,r3,CombMain1OutputR5;
                ldc             r2,(Offset_OutputComb_rd1+192);
                ldw             CombMain2OutputR9, Pointer_strctR7[r2];
                ldc             r3,1;
                maccs RhighR4,RlowR6,r3,CombMain2OutputR9;
                ldc             r3,(Offset_OutputComb_rd1+204);
                ldw             CombMain3OutputR5, Pointer_strctR7[r3];
                ldc             r2,1;
                maccs RhighR4,RlowR6,r2,CombMain3OutputR5;
                ldc             r2,(Offset_OutputComb_rd1+216);
                ldw             CombMain4OutputR9, Pointer_strctR7[r2];
                ldc             r3,1;
                maccs RhighR4,RlowR6,r3,CombMain4OutputR9;
```

```
ldc             r3,(Offset_OutputComb_rd1+228);
ldw             CombMain5OutputR5, Pointer_strctR7[r3];
ldc             r2,1;
maccs RhighR4,RlowR6,r2,CombMain5OutputR5;
ldc             r2,(Offset_OutputComb_rd1+240);
ldw             CombMain6OutputR9, Pointer_strctR7[r2];
ldc             r3,1;
maccs RhighR4,RlowR6,r3,CombMain6OutputR9;
ldc             r3,(Offset_OutputComb_rd1+252);
ldw             CombMain7OutputR5, Pointer_strctR7[r3];
ldc             r2,1;
maccs RhighR4,RlowR6,r2,CombMain7OutputR5;
ldc             r2,(Offset_OutputComb_rd1+300);
ldw             CombLeft1OutputR9, Pointer_strctR7[r2];
ldc             r3,1;
maccs RhighR4,RlowR6,r3,CombLeft1OutputR9;
ldc             r3,(Offset_OutputComb_rd1+312);
ldw             CombLeft2OutputR5, Pointer_strctR7[r3];
ldc             r2,1;
maccs RhighR4,RlowR6,r2,CombLeft2OutputR5;

lextract r10, RhighR4, RlowR6, r8, 32;
ldc             r2, (Offset_InputAllpass+332);
stw             r10, Pointer_strctR7[r2];
```

ambience2_right_allpass_1:

```
ldc                 r2,(Offset_Pointer_CurrentAllpass_rd1+312);
ldw                 Ptr_rd_currentR1,Pointer_strctR7[r2];
ldc                 r2,(Offset_Pointer_Allpass_rd1+312)/2;
ldd                 SizeDelayAllpassR11,Ptr_rdR0,Pointer_strctR7[r2];
ld16s           XnmAllpassR10,Ptr_rdR0[Ptr_rd_currentR1];
add             Ptr_rd_currentR1,Ptr_rd_currentR1,1;
eq              r9, Ptr_rd_currentR1, SizeDelayAllpassR11;
bf              r9, ambience2_right_allpass_maccs_1;
ldc             Ptr_rd_currentR1, 0x0;
```

ambience2_right_allpass_maccs_1:

```
ldc             r2, (Offset_Pointer_CurrentAllpass_rd1+312);
stw             Ptr_rd_currentR1,Pointer_strctR7[r2];

ldc             r3, (Offset_KFeedback_Allpass+312)/2;
ldd             StateAllpassR2,KFeedbackAllpassR1,Pointer_strctR7[r3];
ldc             r3, (Offset_InputAllpass+312);
ldw             InputAllpassR0,Pointer_strctR7[r3];

ldc             r3, (Offset_KOutput_Allpass+312);
ldw             KOutputAllpassR9,Pointer_strctR7[r3];


{add            RhighR4, InputAllpassR0, 0;             ldc RlowR6, 0;}
neg r5,KFeedbackAllpassR1;
maccs RhighR4,RlowR6,r5,XnmAllpassR10;
lextract r3, RhighR4, RlowR6, r8, 32;

{add    RhighR4, XnmAllpassR10, 0;              ldc RlowR6, 0;}
```

```
            maccs RhighR4,RlowR6,KFeedbackAllpassR1,r3;
            lextract r5, RhighR4, RlowR6, r8, 32;

            {ldc     RhighR4, 0;
        ldc RlowR6, 0;}
            maccs RhighR4,RlowR6,KOutputAllpassR9,r5;
            lextract r9, RhighR4, RlowR6, r8, 32;
            ldc             r1,(Offset_InputAllpass+322);
            stw             r9,Pointer_strctR7[r1];
            ldc                     r2,(Offset_Pointer_Allpass_rd1+312)/2;
            ldd                     SizeDelayAllpassR11,Ptr_rdR0,Pointer_strctR7[r2];
            ldc             r5,(Offset_Pointer_CurrentAllpass_wr1+312);
            ldw             Ptr_wr_currentR1, Pointer_strctR7[r5];

            st16    r3,Ptr_wrR0[Ptr_wr_currentR1];
            add             Ptr_wr_currentR1, Ptr_wr_currentR1, 1;
            eq              r9, Ptr_wr_currentR1, SizeDelayAllpassR11;
            bf              r9, ambience2_right_allpass_2;
            ldc     Ptr_wr_currentR1, 0x0;

ambience2_right_allpass_2:
            ldc             r2, (Offset_Pointer_CurrentAllpass_wr1+312);
            stw             Ptr_rd_currentR1,Pointer_strctR7[r2];

            ldc                     r2,(Offset_Pointer_CurrentAllpass_rd1+322);
            ldw                     Ptr_rd_currentR1,Pointer_strctR7[r2];
            ldc                     r2,(Offset_Pointer_Allpass_rd1+322)/2;
            ldd                     SizeDelayAllpassR11,Ptr_rdR0,Pointer_strctR7[r2];
            ld16s           XnmAllpassR10,Ptr_rdR0[Ptr_rd_currentR1];
            add             Ptr_rd_currentR1,Ptr_rd_currentR1,1;
            eq              r9, Ptr_rd_currentR1, SizeDelayAllpassR11;
            bf              r9, ambience2_right_allpass_maccs_2;
            ldc             Ptr_rd_currentR1, 0x0;

ambience2_right_allpass_maccs_2:
            ldc             r2, (Offset_Pointer_CurrentAllpass_rd1+322);
            stw             Ptr_rd_currentR1,Pointer_strctR7[r2];

            ldc             r3, (Offset_KFeedback_Allpass+322)/2;
            ldd             StateAllpassR2,KFeedbackAllpassR1,Pointer_strctR7[r3];
            ldc             r3, (Offset_InputAllpass+322);
            ldw             InputAllpassR0,Pointer_strctR7[r3];

            ldc             r3, (Offset_KOutput_Allpass+322);
            ldw             KOutputAllpassR9,Pointer_strctR7[r3];


            {add            RhighR4, InputAllpassR0, 0;                     ldc RlowR6, 0;}
            neg r5,KFeedbackAllpassR1;
            maccs RhighR4,RlowR6,r5,XnmAllpassR10;
            lextract r3, RhighR4, RlowR6, r8, 32;
            {add    RhighR4, XnmAllpassR10, 0;                  ldc RlowR6, 0;}
            maccs RhighR4,RlowR6,KFeedbackAllpassR1,r3;
            lextract r5, RhighR4, RlowR6, r8, 32;
            {ldc     RhighR4, 0;
        ldc RlowR6, 0;}
```

```
                maccs RhighR4,RlowR6,KOutputAllpassR9,r5;
                lextract r9, RhighR4, RlowR6, r8, 32;
                ldc             r1,(Offset_OutputAllpass_rd1+322);
                stw             r9,Pointer_strctR7[r1];
                ldc                     r2,(Offset_Pointer_Allpass_rd1+312)/2;
                ldd                     SizeDelayAllpassR11,Ptr_rdR0,Pointer_strctR7[r2];
                ldc             r5,(Offset_Pointer_CurrentAllpass_wr1+322);
                ldw             Ptr_wr_currentR1, Pointer_strctR7[r5];

                st16    r3,Ptr_wrR0[Ptr_wr_currentR1];
                add             Ptr_wr_currentR1, Ptr_wr_currentR1, 1;
                eq              r9, Ptr_wr_currentR1, SizeDelayAllpassR11;
                bf              r9, ambience2_right_output;
                ldc     Ptr_wr_currentR1, 0x0;

ambience2_right_output:
                ldc             r2, (Offset_Pointer_CurrentAllpass_wr1+322);
                stw             Ptr_rd_currentR1,Pointer_strctR7[r2];

                {ldc    RhighR4, 0;
        ldc RlowR6, 0;}
                ldc             r1,(Offset_OutputAllpass_rd1+322);
                ldw             AllpassRight2OutputR9, Pointer_strctR7[r1];
                ldc             r2,(Offset_KRevRightToRight+332);
                ldw             KRevR2RR5, Pointer_strctR7[r2];
                maccs RhighR4,RlowR6,KRevR2RR5,AllpassRight2OutputR9;
                ldc             r3,(Offset_OutputAllpass_rd1+322);
                ldw             AllpassRight2OutputR9, Pointer_strctR7[r3];
                ldc             r2,(Offset_KRevRightToLeft+332);
                ldw             KRevR2LR5, Pointer_strctR7[r2];
                maccs RhighR4,RlowR6,KRevR2LR5,AllpassRight2OutputR9;
                ldc             r2,(Offset_OutputTwentyfour_rd1+12);
                ldw             OutputTwentyfourR10,Pointer_strctR7[r2];
                ldc             r3,(Offset_KEarlyRightToRight+332);
                ldw             KEarlyR2RR5,Pointer_strctR7[r3];
                maccs RhighR4,RlowR6,KEarlyR2RR5,OutputTwentyfourR10;
                ldc             r2,(Offset_OutputTwentyfour_rd1+12);
                ldw             OutputTwentyfourR10,Pointer_strctR7[r2];
                ldc             r3,(Offset_KEarlyRightToLeft+332);
                ldw             KEarlyR2LR5,Pointer_strctR7[r3];
                maccs RhighR4,RlowR6,KEarlyR2LR5,OutputTwentyfourR10;

                lextract r10, RhighR4, RlowR6, r8, 32;
                ldc             r2,(Offset_Ambience2OutputRight+284);
                stw             r10, Pointer_strctR7[r2];

ambience2_left_allpass_1:

                ldc                     r2,(Offset_Pointer_CurrentAllpass_rd1+332);
                ldw                     Ptr_rd_currentR1,Pointer_strctR7[r2];
                ldc                     r2,(Offset_Pointer_Allpass_rd1+332)/2;
                ldd                     SizeDelayAllpassR11,Ptr_rdR0,Pointer_strctR7[r2];
                ld16s           XnmAllpassR10,Ptr_rdR0[Ptr_rd_currentR1];
                add             Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq              r9, Ptr_rd_currentR1, SizeDelayAllpassR11;
                bf              r9, ambience2_left_allpass_maccs_1;
```

```
                ldc             Ptr_rd_currentR1, 0x0;

ambience2_left_allpass_maccs_1:
                ldc             r2, (Offset_Pointer_CurrentAllpass_rd1+332);
                stw             Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldc             r3, (Offset_KFeedback_Allpass+332)/2;
                ldd             StateAllpassR2,KFeedbackAllpassR1,Pointer_strctR7[r3];
                ldc             r3, (Offset_InputAllpass+332);
                ldw             InputAllpassR0,Pointer_strctR7[r3];

                ldc             r3, (Offset_KOutput_Allpass+332);
                ldw             KOutputAllpassR9,Pointer_strctR7[r3];


                {add            RhighR4, InputAllpassR0, 0;                 ldc RlowR6, 0;}
                neg r5,KFeedbackAllpassR1;
                maccs RhighR4,RlowR6,r5,XnmAllpassR10;
                lextract r3, RhighR4, RlowR6, r8, 32;
                {add    RhighR4, XnmAllpassR10, 0;              ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,KFeedbackAllpassR1,r3;
                lextract r5, RhighR4, RlowR6, r8, 32;
                {ldc    RhighR4, 0;
        ldc RlowR6, 0;}
                maccs RhighR4,RlowR6,KOutputAllpassR9,r5;
                lextract r9, RhighR4, RlowR6, r8, 32;
                ldc             r1,(Offset_InputAllpass+342)
                stw             r9,Pointer_strctR7[r1];
                ldc                     r2,(Offset_Pointer_Allpass_rd1+312)/2;
                ldd                     SizeDelayAllpassR11,Ptr_rdR0,Pointer_strctR7[r2];
                ldc             r5,(Offset_Pointer_CurrentAllpass_wr1+332);
                ldw             Ptr_wr_currentR1, Pointer_strctR7[r5];

                st16    r3,Ptr_wrR0[Ptr_wr_currentR1];
                add             Ptr_wr_currentR1, Ptr_wr_currentR1, 1;
                eq              r9, Ptr_wr_currentR1, SizeDelayAllpassR11;
                bf              r9, ambience2_left_allpass_2;
                ldc     Ptr_wr_currentR1, 0x0;

ambience2_left_allpass_2:
                ldc             r2, (Offset_Pointer_CurrentAllpass_wr1+332);
                stw             Ptr_rd_currentR1,Pointer_strctR7[r2];

                ldc                     r2,(Offset_Pointer_CurrentAllpass_rd1+342);
                ldw                     Ptr_rd_currentR1,Pointer_strctR7[r2];
                ldc                     r2,(Offset_Pointer_Allpass_rd1+342)/2;
                ldd                     SizeDelayAllpassR11,Ptr_rdR0,Pointer_strctR7[r2];
                ld16s           XnmAllpassR10,Ptr_rdR0[Ptr_rd_currentR1];
                add             Ptr_rd_currentR1,Ptr_rd_currentR1,1;
                eq              r9, Ptr_rd_currentR1, SizeDelayAllpassR11;
                bf              r9, ambience2_left_allpass_maccs_2;
                ldc             Ptr_rd_currentR1, 0x0;

ambience2_left_allpass_maccs_2:
                ldc             r2, (Offset_Pointer_CurrentAllpass_rd1+342);
                stw             Ptr_rd_currentR1,Pointer_strctR7[r2];
```

```
        ldc         r3, (Offset_KFeedback_Allpass+342)/2;
        ldd         StateAllpassR2,KFeedbackAllpassR1,Pointer_strctR7[r3];
        ldc         r3, (Offset_InputAllpass+342);
        ldw         InputAllpassR0,Pointer_strctR7[r3];

        ldc         r3, (Offset_KOutput_Allpass+342);
        ldw         KOutputAllpassR9,Pointer_strctR7[r3];


        {add        RhighR4, InputAllpassR0, 0;                 ldc RlowR6, 0;}
        neg r5,KFeedbackAllpassR1;
        maccs RhighR4,RlowR6,r5,XnmAllpassR10;
        lextract r3, RhighR4, RlowR6, r8, 32;
        {add    RhighR4, XnmAllpassR10, 0;              ldc RlowR6, 0;}
        maccs RhighR4,RlowR6,KFeedbackAllpassR1,r3;
        lextract r5, RhighR4, RlowR6, r8, 32;
        {ldc    RhighR4, 0;
    ldc RlowR6, 0;}
        maccs RhighR4,RlowR6,KOutputAllpassR9,r5;
        lextract r9, RhighR4, RlowR6, r8, 32;
        ldc         r1,(Offset_OutputAllpass_rd1+342);
        stw         r9,Pointer_strctR7[r1];
        ldc             r2,(Offset_Pointer_Allpass_rd1+312)/2;
        ldd             SizeDelayAllpassR11,Ptr_rdR0,Pointer_strctR7[r2];
        ldc         r5,(Offset_Pointer_CurrentAllpass_wr1+342);
        ldw         Ptr_wr_currentR1, Pointer_strctR7[r5];

        st16    r3,Ptr_wrR0[Ptr_wr_currentR1];
        add         Ptr_wr_currentR1, Ptr_wr_currentR1, 1;
        eq          r9, Ptr_wr_currentR1, SizeDelayAllpassR11;
        bf          r9, ambience2_left_output;
        ldc     Ptr_wr_currentR1, 0x0;

ambience2_left_output:
        ldc         r2, (Offset_Pointer_CurrentAllpass_wr1+342);
        stw         Ptr_rd_currentR1,Pointer_strctR7[r2];

        {ldc    RhighR4, 0;
    ldc RlowR6, 0;}
        ldc         r1,(Offset_OutputAllpass_rd1+342);
        ldw         AllpassLeft2OutputR9, Pointer_strctR7[r1];
        ldc         r2,(Offset_KRevLeftToRight+332);
        ldw         KRevL2RR10, Pointer_strctR7[r2];
        maccs RhighR4,RlowR6,KRevL2RR10,AllpassLeft2OutputR9;
        ldc         r1,(Offset_OutputAllpass_rd1+342);
        ldw         AllpassLeft2OutputR9, Pointer_strctR7[r1];
        ldc         r2,(Offset_KRevLeftToLeft+332);
        ldw         KRevL2LR10, Pointer_strctR7[r2];
        maccs RhighR4,RlowR6,KRevL2LR10,AllpassLeft2OutputR9;
        ldc         r2,(Offset_OutputTwentyfour_rd1+90);
        ldw         OutputTwentyfourR10,Pointer_strctR7[r2];
        ldc         r3,(Offset_KEarlyLeftToRight+332);
        ldw         KEarlyL2RR9,Pointer_strctR7[r3];
        maccs RhighR4,RlowR6,KEarlyL2RR9,OutputTwentyfourR10;
        ldc         r2,(Offset_OutputTwentyfour_rd1+90);
```

```
                ldw             OutputTwentyfourR10,Pointer_strctR7[r2];
                ldc             r3,(Offset_KEarlyLeftToLeft+332);
                ldw             KEarlyL2LR9,Pointer_strctR7[r3];
                maccs RhighR4,RlowR6,KEarlyL2LR9,OutputTwentyfourR10;

                lextract r10, RhighR4, RlowR6, r8, 32;
                ldc             r2, (Offset_Ambience2OutputLeft+284);
                stw             r10, Pointer_strctR7[r2];

.globl asm_effect.nstackwords
.set asm_effect.nstackwords, 0
.size asm_effect, .-asm_effect
.cc_bottom asm_effect.function
```

Bibliografia

[1] J.Dattorro - ***Effect Design – Part 1: Reverberators and Other Filters*** - *J. Audio Eng. Soc, Vol. 45, No 9, 1997 September*. In *Simple Reverberation Network*, pagine 661-664.

[2] A.A.de Lima, S.L.Neto, L.W.P.Biscainho, F.P.Freeland, B.C.Bispo, R.A.de Jesus, R.Schafer, A.Said, B.Lee, T.Kalker – ***Quality Evaluation of Reverberation in Audioband Speech Signals***. In *3.3 - Feedback Delay Networks*

[3] S.Bilbao, J.Parker – ***Spring Reverberation: A Physical Perspective*** - *Proc. of the 12th Int. Conference on Digital Audio Effects (DAFx-09), Como, Italy, September 1-4, 2009.*

[4] S.Bilbao, J.Parker - ***Perceptual and Numerical Aspects of Spring Reverberation Model*** - *Proceedings of 20th International Symposium on Music Acoustics (Associated Meeting of the International Congress on Acoustics), 25-31 August 2010, Sydney and Katoomba, Australia.*

[5] S.Willemsen, S.Serafin, J.R.Jensen – ***Virtual Analog Simulation and Extensions of Plate Reverberation*** - *Proceedings of the 14th Sound and Music Computing Conference, July 5-8, Espoo, Finland - SMC2017-315.*

[6] Mark Kahrs, Karlheinz Brandenburg – ***Applications of Digital Signal Processing to Audio and Acoustics***

[7] XMOS - ***I2S/TDM Library*** - www.xmos.com, XM007055.

[8] XMOS – ***xCore-200: The XMOS XS2 Architecture*** - *REV 1.0, 2015/04/01, XMOS © 2015.*

[9] XMOS – ***XMOS Programming Guide*** - *Document Number: XM004440A, Publication Date: 2014/10/9, XMOS © 2014.*

[10] F.Romani (UniPi) – ***Introduzione all'Audio Digitale*** - *http://pages.di.unipi.it/romani/DIDATTICA/AD/AD%208%20for.pdf*

[11] http://www.audiosonica.com/it/corso/post/16/Teoria_del_suono_Comportamento_del_suono

[12] Università di Modena e Reggio Emilia - *Fisica Onde Musica: Risonanza*

[13] Erik Jansson - ***Acoustics for violin and guitar makers - Chapter II: Resonance and Resonators*** - *TMH, Speech, Music and Hearing*

[14] https://it.wikipedia.org/wiki/Risonanza_acustica

[15] Phillips, C.l., Parr, J.M., & Riskin, E.A - ***Signals, systems and Transforms*** - *Prentice Hall, 2007*

[16] R. R. Spencer, M. S. Ghausi - ***Introduction to Electronic Circuit Design*** - *2003, Prentice Hall*

[17] A. Antoniou - ***Digital Filters: Analysis, Design, and Applications*** - *New York, NY: McGraw-Hill, 1993*

[18] Gardner, W. G. (1995) - ***Efficient convolution without input-output Delay*** - *J. Audio Eng. Soc.*, 43(3): 127–136

[19] Schroeder, M. R. (1970b) - ***Digital simulation of sound transmission in reverberant spaces*** - *J. Acoust. Soc. Am.*, 47(2):424–431

[20] Moorer, J. A. (1979) - ***About This Reverberation Business*** - *Computer Music Journal*, 3(2):3255–3264

[21] Schroeder, M. R. (1962) - ***Natural Sounding Artificial Reverberation*** - *J. Audio Eng. Soc.*, 10(3)

[22] Jot, J. M. (1992b) - ***Etude et réalisation d'un spatialisateur de sons par modèles physiques et perceptifs (Design and implementation of a sound spatializer based on physical and perceptual models, in French*** - PhD thesis, Telecom Paris

[23] Jot, J. M. and Chaigne, A. (1991) - ***Digital delay networks for designing artificial reverberators*** - In *Proc. Audio Eng. Soc. Conv.* Preprint 3030

[24] Stautner, J. and Puckette, M. (1982) - ***Designing multichannel Reverberators*** - *Computer Music Journal*, 6(1):52–65

[25] Gardner, W. G. (1992) - ***A realtime multichannel room simulator*** - *J. Acoust. Soc. Am.*, 92(4 (A)):2395. http://sound.media.mit.edu/papers.html

[26] Griesinger, D. (1989) - ***Practical processors and programs for digital reverberation*** - In *Proc. Audio Eng. Soc. 7th Int. Conf.*, pages 187–195, Toronto, Ontario, Canada. Audio Eng. Society

[27] Gardner, W. G. (1995) - ***Efficient convolution without input-output delay*** - *J. Audio Eng. Soc.*, 43(3): 127–136

[28] Lehnert, H. and Blauert, J. (1992) - ***Principles of binaural room simulation*** - *Applied Acoustics*, 36:259–291

[29] Blauert, J. (1983) - ***Spatial Hearing*** - MIT Press, Cambridge, MA

[30] Wightman, F. L. and Kistler, D. J. (1989) - ***Headphone simulation of free-field listening. I: Stimulus synthesis*** - *J. Acoust. Soc. Am.*, 85(2):858–867.

[31] Barron, M. and Marshall, A. H. (1981) - ***Spatial Impression Due to Early Lateral Reflection in Concert Halls: The Derivation of a Physical Measure*** - *J. Sound and Vibration*, 77(2):211–232

[32] Hidaka, T., Beranek, L. L., and Okano, T. (1995) - ***Interaural cross-correlation, lateral fraction, and low- and high-frequency sound levels as measures of acoustical quality in concert halls*** - *J. Acoust. Soc. Am.*, 98(2):988– 1007

[33] Blauert, J. and Cobben, W. (1978) - ***Some consideration of binaural cross-correlation analysis*** - *Acustica*, 39(2):96–104

[34] Sabine, W. C. (1972) - ***Reverberation*** - In Lindsay, R. B., editor, *Acoustics: Historical and Philosophical Development.* Dowden, Hutchinson, and Ross, Stroudsburg, PA. Originally published in 1900

[35] Schroeder, M. R. (1965) - ***New method of measuring reverberation time*** - *J. Acoust. Soc. Am.*, 37:409–412

[36] Beranek, L. L. (1986) – ***Acoustics*** - American Institute of Physics, New York, NY

[37] Kuttruff, H. (1991) - ***Room Acoustics*** - Elsevier Science Publishing Company, New York, NY

[38] Oppenheim, A. V. and Schafer, R. W. (1989) - ***Discrete-Time Signal Processing*** - Prentice Hall, Englewood Cliffs, New Jersey