UNIVERSITÀ POLITECNICA DELLE MARCHE

Facoltà di Ingegneria

Corso di laurea triennale in ingegneria elettronica

Studio e implementazione algoritmo di pitch shifting sulla piattaforma embedded Audiolino

Study and development of a pitch shifting algorithm on embedded platform Audiolino

Relatore:

Prof. Stefano Squartini

Tesi di Laurea di:

Mattia Massimi

Correlatore:

Ing. Leonardo Gabrielli

A.A. **2020/2021**

Indice

Capitolo 1

INTRODUZIONE

1.1 Finalità dell'algoritmo e nozioni base

Le caratteristiche principali che definiscono un suono sono: intensità, durata, timbro e pitch o altezza del suono. Con pitch si intende quella proprietà dei suoni che permette di ordinarli in una scala musicale, basata principalmente sulla percezione della frequenza associata a quel suono, stabilendo quindi quale suono è più "alto" o più "basso".

Le oscillazioni delle onde vengono di solito caratterizzate in termini di frequenza, oggetto fisico misurabile, mentre si fa riferimento all'altezza del suono a livello qualitativo e percettivo, influenzato anche dal timbro, associandolo indirettamente alla frequenza.

Il pitch è quindi associato ad una determinata frequenza corrispondente a quella del suono emesso dallo strumento in questione, anche se generalmente si fa riferimento alla frequenza fondamentale della nota, poiché i suoni sono più complessi e hanno numerose armoniche, suoni semplici il cui valore di frequenza è multiplo intero della fondamentale e la cui ampiezza e valore definisce quello che viene chiamato *timbro*, grazie anche al quale è possibile distinguere lo strumento che ha emesso una determinata nota (figura 1.1).

Gli algoritmi di pitch shift vanno proprio a modificare questa caratteristica del suono andando ad aumentare o diminuire di un certo numero di semitoni l'altezza del suono.

La formula che lega frequenza e numero di semitoni di distanza da una nota di riferimento secondo il temperamento equabile è $f = 2^{\frac{N}{12}} \cdot f_{rif}$, dove N è il numero di semitoni di distanza dalla nota di riferimento e f la sua frequenza. La frequenza usata come standard internazionale di riferimento e corrispondente al La4 (notazione internazionale A4) è pari a 440Hz. Nella tabella 1.2 sono riportati tutti i valori di frequenza delle note. [1]

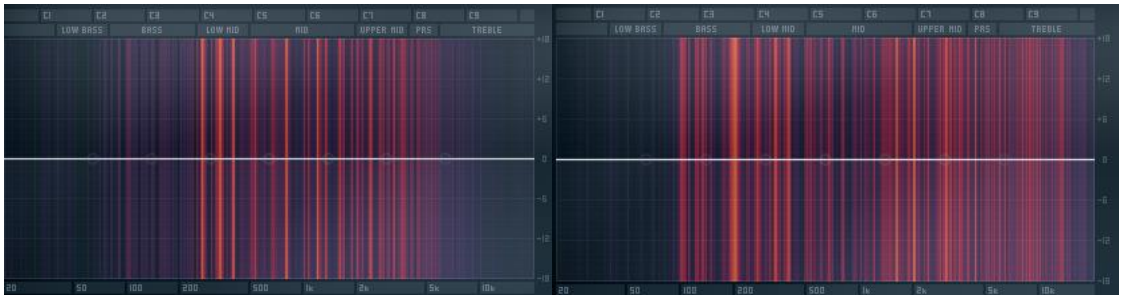

Figura 1.1: Analisi in frequenza di un accordo di Do maggiore suonato con un pianoforte (sinistra) e una chitarra acustica (destra). Ogni riga corrisponde ad un'armonica e l'intensità del colore rappresenta la sua intensità. Si può osservare immediatamente come nel caso della chitarra siano presenti in grande quantità (o a volume maggiore) le armoniche più alte che rendono il suono più brillante all'ascolto.

Note	ottave									
	0	1	2	3	4	5	6	7	8	9
Do	16,35	32,70	65,41	130,8	261,6	523,3	1047	2093	4186	8372
Do#-Reb	17,32	34,65	69,30	138,6	277,2	554,4	1109	2217	4435	8870
Re	18,35	36,71	73,42	146,8	293,7	587,3	1175	2349	4699	9397
Re#-Mib	19,45	38,89	77,78	155,6	311,1	622,3	1245	2489	4978	9956
Mi	20,60	41,20	82,41	164,8	329,6	659,3	1319	2637	5274	10548
Fa	21,83	43,65	87,31	174,6	349,2	698,5	1397	2794	5588	11175
Fa#-Solb	23,12	46,25	92,50	185,0	370,0	740,0	1480	2960	5920	11840
Sol	24,50	49,00	98,00	196,0	392,0	784,0	1568	3136	6272	12544
Sol#-Lab	25,96	51,91	103,8	207,7	415,3	830,6	1661	3322	6645	13290
La	27,50	55,00	110,0	220,0	440,0	880,0	1760	3520	7040	14080
La#-Sib	29,14	58,27	116,5	233,1	466,2	932,3	1865	3729	7459	14917
Si	30,87	61,74	123,5	246,9	493,9	987,8	1976	3951	7902	15804

Tabella 1.2: Qui sono riportate le varie frequenze delle note in tutte le ottave secondo il temperamento equabile

1.2 Introduzione alla piattaforma Audiolino

Audiolino brick è una scheda DSP per effettistica musicale controllabile da computer tramite lo standard MIDI. Viene fornita direttamente con alcuni effetti già caricati al suo interno e pronti all'uso, ma è possibile anche caricarne di propri. Ogni algoritmo può essere eseguito singolarmente ma è possibile collegare più schede in cascata per ottenere un multi-effetto.

La scheda ha un ingresso stereo a 4 pin e un'uscita stereo a 4 pin al quale è possibile attaccare degli adattatori per minijack da 3,5mm da utilizzare come input e output. È inoltre dotata di due potenziometri per andare a modificare in tempo reale i parametri dei vari algoritmi. L'alimentazione è fornita alla scheda tramite la porta microUSB presente sulla scheda collegabile con un qualsiasi cavo USB adibito anche al trasporto dati.

Gli algoritmi sopra citati possono essere caricati e gestiti tramite il software gratuito "Audiolino Brick Editor" scaricabile dal sito audiolino.com. Una volta collegata la scheda al computer sarà sufficiente premere su connect per procedere con la connessione e il riconoscimento della scheda per poi essere in grado di modificare gli algoritmi esistenti o caricarne uno proprio tramite "FW upgrade". [2]

In figura 1.3 si possono vedere alcune delle caratteristiche della board e il significato dei vari pin.

I parametri di funzionamento della scheda sono illustrati nella tabella 1.4. "STARTUP_PROGRAM" è il comando che viene chiamato all'accensione e farà avviare la scheda. I successivi "POTx_POLARITY" selezionano la polarità (normal/reverse) del potenziometro in questione e con l'ultimo parametro è possibile selezionare se usare il potenziometro 4 come potenziometro o come selettore di programma.

Per leggere i parametri si utilizza il comando DEV_PARAM_GET e per scriverli DEV_PARAM_SET. Da notare che i parametri così facendo saranno scritti in RAM, per salvarli in memoria occorrerà usare il comando STORE_DEV_PARAM che salverà tutti i parametri in memoria contemporaneamente.

Per quanto riguarda invece i parametri relativi all'algoritmo, per le stesse operazioni sono presenti analoghi comandi: ALG_PARAM_SET, ALG_PARAM_GET. Questi saranno invece salvati in memoria insieme al programma. [3]

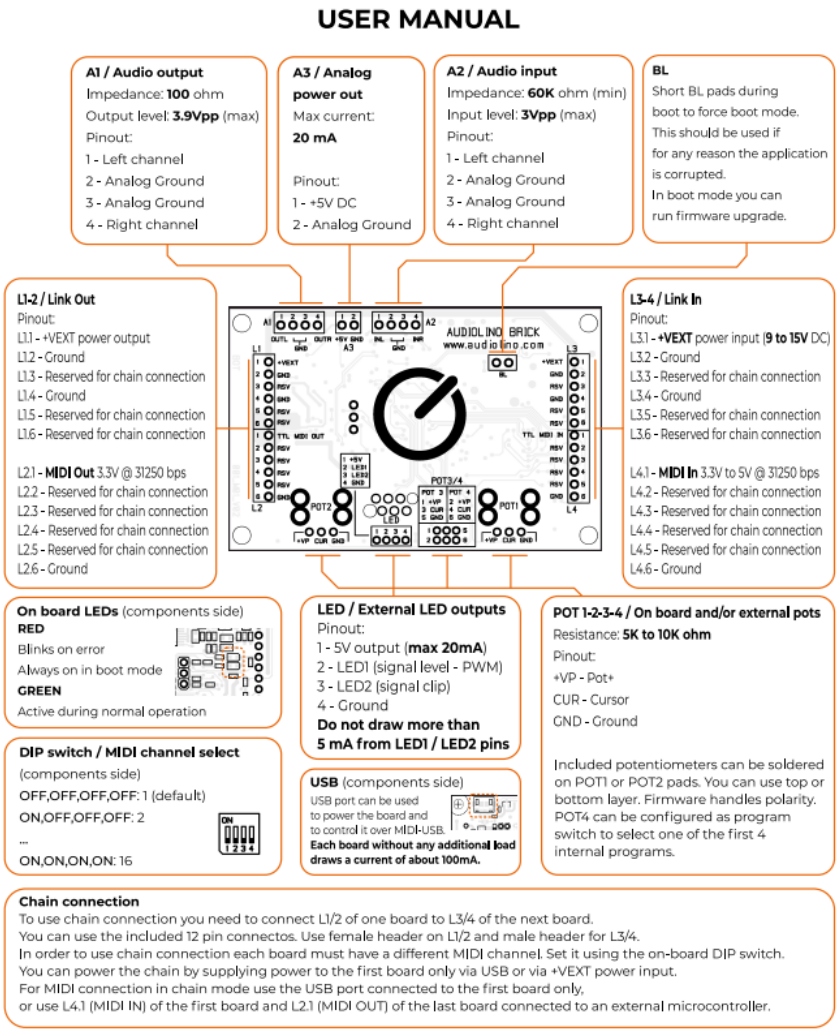

Figura 1.3: Immagine presa dal datasheet della board che illustra i vari pin [4]

Address	Name	Min	Max	Default	Access
0	STARTUP_PROGRAM	0	31	0	RW
1	POT1_POLARITY	0	1	0	RW
2	POT2_POLARITY	0	1	0	RW
3	POT3_POLARITY	0	1	0	RW
4	POT4_POLARITY	0	1	0	RW
5	POT4_MODE	0	31	0	RW

Tabella 1.4

Capitolo 2

ALGORITMO DI PITCH SHIFTING TRAMITE DUE DELAY-LINE

2.1 Le delay-line

Con delay-line si intende, come suggerito dal nome, una linea di ritardo e in ambito DSP possono essere utilizzate per ritardare un segnale di un certo numero di campioni. Le delay-line diventano interessanti se si riesce a far variare fluidamente e in maniera continua nel tempo i suoi parametri, come ad esempio il ritardo *d* che viene introdotto ogni istante di tempo.

Il primo problema in questa operazione si ha nel definire e trattare i ritardi frazionari che inevitabilmente verranno introdotti visto che il ritardo varierà in maniera continua. Infatti presa ad esempio $x[n]$ definita solo per valori interi di n, un'espressione del tipo $x[n-d]$ non è determinata per valori di d non interi. L'uscita desiderata si troverà quindi a metà tra due campioni successivi e sarà quindi necessaria un'interpolazione tra i due valori per poter avere un'uscita. Nel codice che verrà successivamente descritto viene utilizzata un'interpolazione lineare per risolvere questo problema.

Il secondo problema che si incontra è il valore massimo di ritardo che la linea può assumere. Ipotizzando infatti di voler variare in maniera continua e crescente il valore del ritardo d introdotto dalla delay-line per un tempo medio-lungo, cosa che sarà utile e necessaria per lo scopo del codice, sarà necessario creare una delay-line di lunghezza indefinita e si andrà a saturare la memoria del supporto utilizzato, dato che sarebbe necessario porre il massimo ritardo a valori estremamente elevati (al limite infiniti) e quindi questo metodo non può essere utilizzato indefinitamente. Per risolvere questo problema ci sono varie strade:

- Variare il ritardo introdotto in maniera continua alternando intervalli crescenti e decrescenti in modo da mantenere una variazione continua di d ma mantenendo il valore confinato tra un massimo e un minimo ben definiti;
- Utilizzare due delay-line con lo stesso range di valori (o ritardo massimo) crescenti o decrescenti, trasformando la retta indefinita precedente in un dente di sega confinato tra due valori. Il problema che viene introdotto da questa soluzione è il modo in cui viene trattata l'inevitabile discontinuità di salto

presente nel dente di sega che se lasciata come tale genererebbe forti artefatti audio in uscita. Proprio a questo scopo viene utilizzata una coppia di delay-line di cui una sfasata rispetto all'altra di metà periodo, in modo da poter saltare da una all'altra ogni volta che in una vi è la discontinuità e operare un cross-fading tra le due uscite per avere un passaggio fluido senza clippling o artefatti nella commutazione continua tra le due linee.

In figura 2.1 è riportato un esempio delay-line variabile dove viene evidenziata la relazione tra l'input e l'output nel tempo. La delay-line presa in considerazione ha come lunghezza massima D e per ogni campione dell'output è presente un campione (dove necessario interpolato) dell'input alla delay-line. Se chiamiamo n il numero del campione in output l'asse verticale indica quindi $y[n] = n - d[n]$ dove $d[n]$ è il delay in numero di campione.

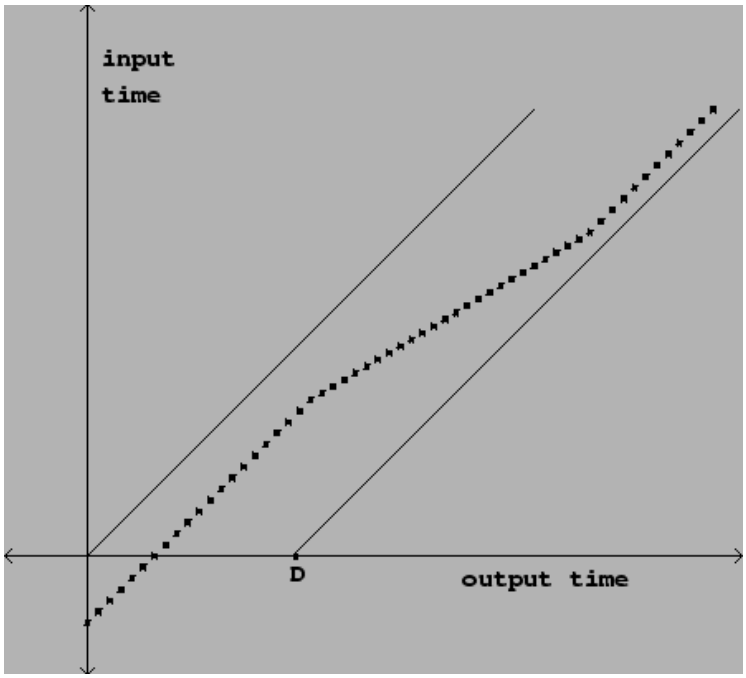

Figura 2.1: rappresentazione grafica della funzione che lega gli indici temporali di uscita e ingresso di una delay-line

Nella stessa figura possiamo notare le due linee diagonali che indicano la massima ampiezza della delay-line e quindi la regione in cui la funzione $y[n]$ deve rimanere.

Se $d[n]$ non varia al variare di n e quindi del campione selezionato, i campioni che usciranno dalla delay-line lo faranno alla stessa velocità con la quale sono entrati e quindi anche il suono risultante sarà invariato.

Se invece $d[n]$ varia in funzione di n, ad esempio con andamento crescente, il suono risultante dai campioni in uscita sarà trasposto verso il basso (pitch down), se decrescente, verso l'alto (pitch up). Osservando il grafico di figura 2.1 si può dedurre che

nel tratto iniziale non è presente alcuna variazione di pitch finchè la pendenza della retta è pari a 1, successivamente ci sarà un intervallo in cui la pendenza sarà minore di 1 ottenendo quindi un pitch down, per poi ritornare alla tonalità originale con la retta che torna a pendenza unitaria. Un'ulteriore osservazione che è possibile fare riguarda la pendenza della retta che determina la quantità di pitch down: più la trasposizione sarà ampia, più sarà pendente la retta e prima si raggiungerà la diagonale che delimita la fascia massima di valori di ritardo ammissibili per tale delay-line e questo riporta di nuovo al secondo problema trattato precedentemente in questo capitolo

Questo fenomeno è chiamato effetto Doppler ed esiste anche in natura: se l'osservatore e/o la sorgente del suono si muovono entrambi rispetto al mezzo di propagazione (aria) il suono udito dall'osservatore verrà percepito alterato, tipico esempio sono la sirena di un treno o la sirena di un'ambulanza il cui suono viene percepito con tonalità sempre più bassa man mano che la sorgente si allontana da noi. L'aria attraverso la quale il suono viaggia può essere infatti considerata come una delay-line. Modificare il ritardo della delay-line corrisponde ad allontanare o avvicinare l'osservatore alla sorgente.

2.2 Pitch shifting

Dalla descrizione della delay-line si è visto come utilizzare quest'ultima come strumento per adoperare una trasposizione di tonalità ad un suono sfruttando l'effetto Doppler.

Volendo ora ritornare ai due problemi principali che si incontrano nella costruzione corretta di una delay-line descritti nel paragrafo 2.1, si può osservare che la prima soluzione al problema della limitatezza del valore massimo della delay-line non è altro che un vibrato a livello sonoro. Infatti non si fa altro che alternare ritardi crescenti e decrescenti periodicamente nel tempo che hanno come risultato pitch up e pitch down alternati periodicamente e l'ammontare della trasposizione ovviamente dipenderà dalla pendenza della retta (figura 2.2). Se ad esempio si volesse implementare un vibrato come quello mostrato in figura, sarebbe sufficiente utilizzare

$$d[n] = d_0 + a \cdot \cos(\omega n)$$

dove d_0 è il ritardo medio, a la variazione massima del ritardo, mentre ω rappresenterà quanto velocemente varia sotto forma di velocità angolare.

Volendo invece implementare un pitch-shifter che mantenga il pitch desiderato nel tempo, è necessario ricorrere all'utilizzo di due delay-line opportunamente definite, con stesso ritardo massimo e shiftate l'una rispetto all'altra di mezzo periodo e combinate tramite cross-fading come illustrato nel paragrafo 2.1. Lo shift di mezzo periodo fa si che quando avverrà la commutazione e una si troverà nel punto di discontinuità l'altra sarà esattamente a metà (figura 2.3). [5]

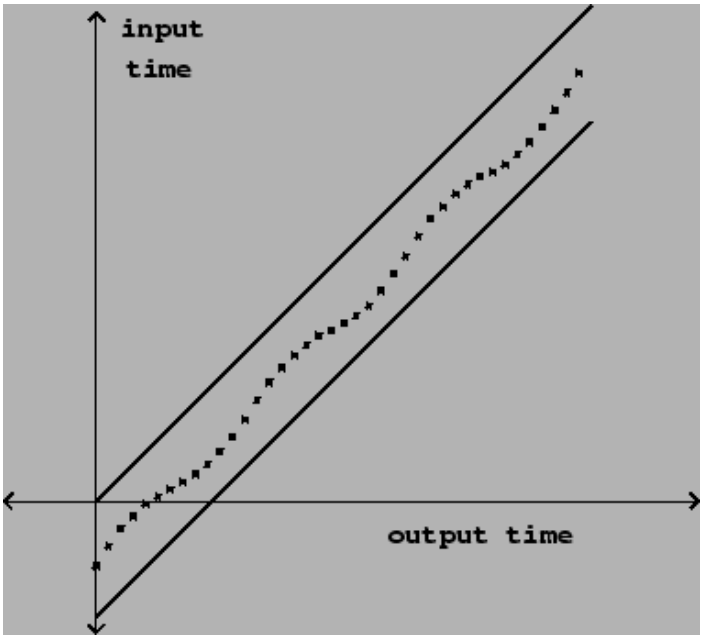

Figura 2.2: vibrato

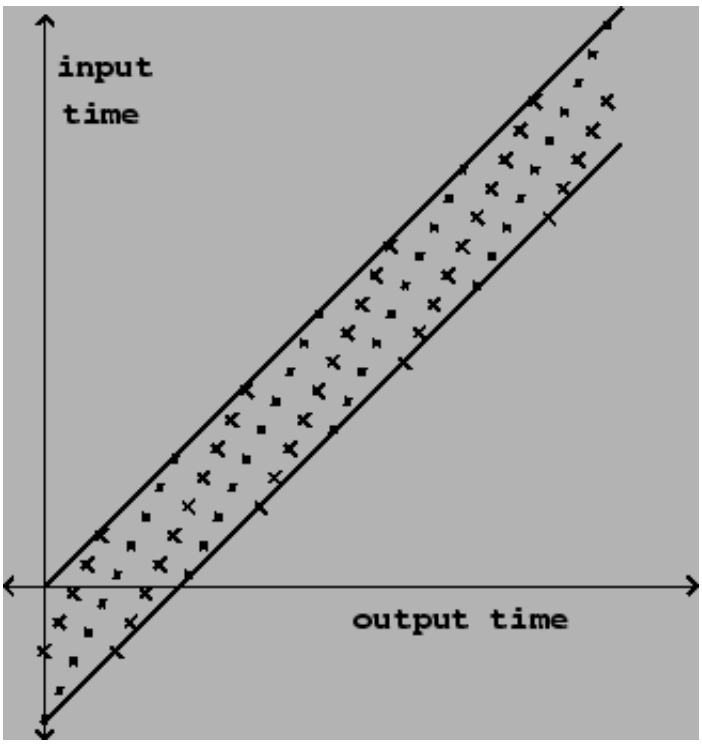

Figura 2.3: pitch shifting con due delay-line. Con il processo sopra descritto si rimane sempre all'interno dei limiti di ritardo massimo della delay-line in questione

2.3 Descrizione codice MATLAB

L'algoritmo MATLAB inizia con la definizione della lunghezza massima della delay-line, corrispondente anche alla finestra di elaborazione, denominata "framelength", che verrà impostata a 4096 nei successivi esempi e dello shift desiderato in numero di semitoni. Lo shift verrà poi convertito nel pitch change ratio tramite la formula $pitch\ change\ ratio = 2^{\frac{n\_semitoni}{12}}$ dalla quale poi si definirà il verso (valutando se il pcr è maggiore o minore di 1) e l'ampiezza della rampa che verrà generata (WIDTH). La seconda parte di codice qui riportato si occupa invece della generazione vera e propria delle rampe nei due casi, chiamate t1 e t2, di cui possiamo avere un esempio grafico osservando la figura 2.5.

Successivamente il codice procede con la definizione della finestra che verrà utilizzata per il cross-fading, in particolare in questo caso è stata utilizzata una finestra di Hanning. Ne verrà naturalmente generata una seconda identica shiftata di mezzo periodo che andrà applicata alla seconda delay-line in modo da rendere possibile la somma delle uscite delle due delay-line così modulate. È riportato un esempio di uscite dalle delay-line modulate in figura 2.6.

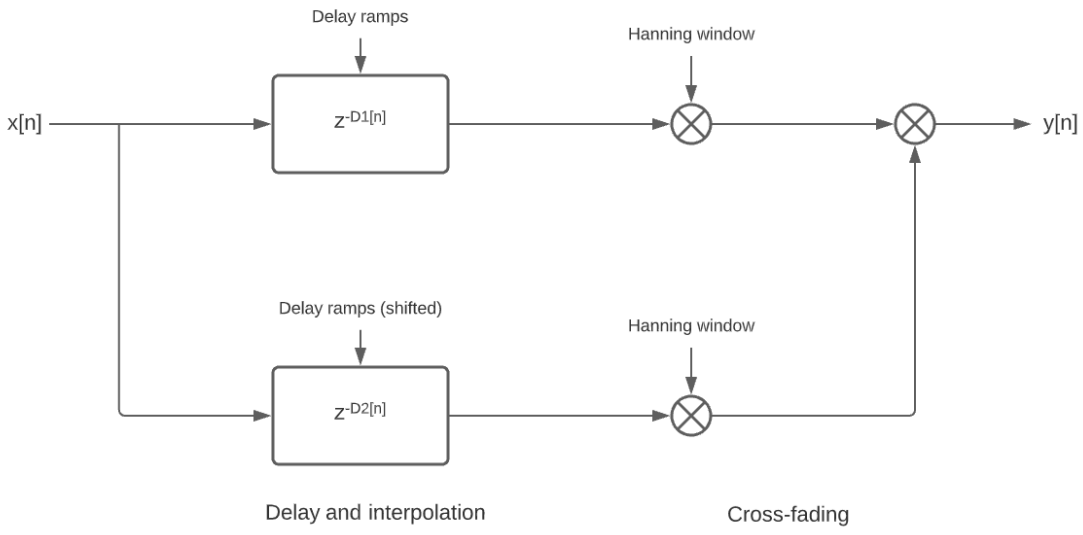

Figura 2.4: Schema a blocchi del codice implementato

Conversione e impostazione rampe:

```
ratio = 2^(shift/12)
if ratio < 1                                        %pitch down
    verse = 0;
    WIDTH = (framelength - 1)*(1 - ratio);
else
    verse = 1;                                      %pitch up
    WIDTH = (framelength - 1)*(ratio - 1);
    if ratio == 2
     WIDTH = framelength - 2;
    end
end
```

Generazione rampe:

```
if verse == 0                           %rampa nel caso pitch down
    t1 = 0: 1/(framelength-1) :1;
    t2 = mod (t1+1/2, 1);
else
    t1 = 1: -1/(framelength-1) :0;    %rampe nel caso pitch up
    t2 = mod (t1+1/2, 1);
end
```

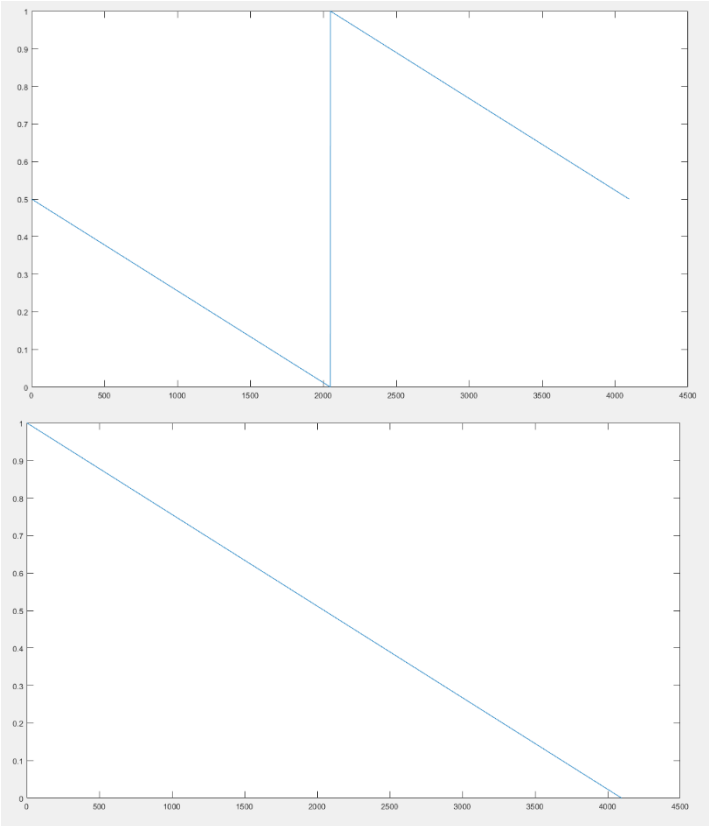

Figura 2.5: Rampe ottenute richiedendo uno shift di 7 semitoni con lunghezza della finestra pari a 4096 campioni. Si può osservare come descritto in precedenza che una è la versione shiftata dell'altra.

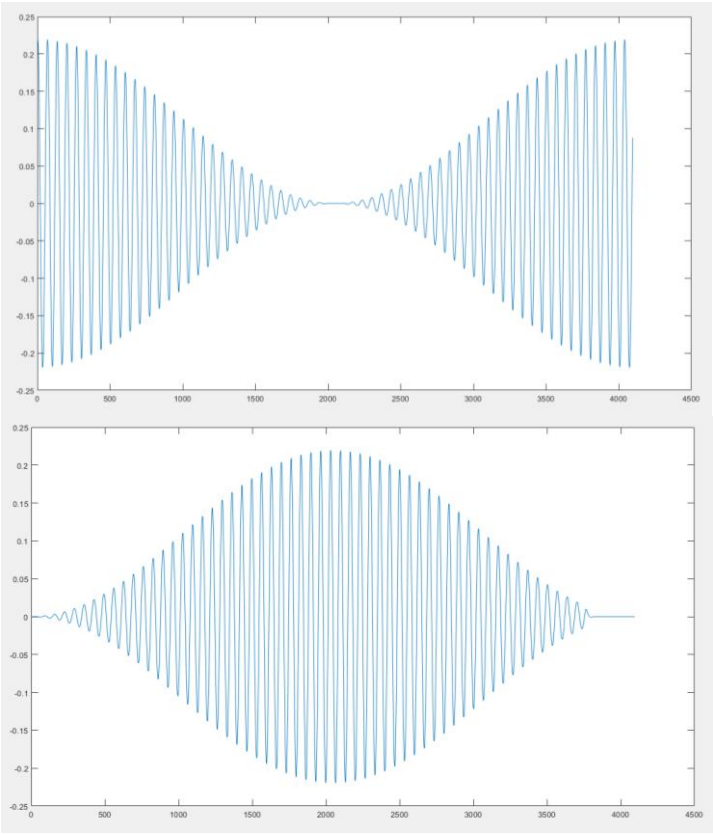

Figura 2.6: Uscite ottenute mandando in ingresso una sinusoide a 440Hz richiedendo uno shift di 7 semitoni.

Il nucleo principale del codice è il ciclo che mette insieme le operazioni sopra descritte (interpolazione e cross-fading) per ogni blocco di 4096 campioni del segnale in ingresso. Come è possibile notare nel codice sottostante i valori delle rampe vengono utilizzati suddivisi in parte intera (i1, i2) e parte frazionaria (frac1, frac2), per poter successivamente procedere con l'interpolazione.

```
for n = 0:M-1
  for index = 1:framelength
    Delayline = [x(n*framelength+index); Delayline(1:framelength-1)];

    ind1    = i1(index);
    ind2    = i2(index);
    frac01 = frac1(index);
    frac02 = frac2(index);

    % interpolazione delle delay-line
    y1(index,1) = Delayline(ind1+1)*frac01 + Delayline(ind1)*(1-
frac01) ;

    y2(index,1) = Delayline(ind2+1)*frac02 + Delayline(ind2)*(1-
frac02);

  end
  out(n*framelength+1 : (n+1)*framelength)=y1.*WIN1 + y2.*WIN2;
  end
```

I risultati di tale codice sono stati analizzati all'interno di MATLAB e sono stati messi a confronto con quelli risultanti dal codice C nel paragrafo 2.5.

2.4 Descrizione codice C

Il medesimo codice è stato poi riscritto anche in linguaggio C come passaggio intermedio prima di passare al porting sulla board Audiolino, dove è stato ovviamente necessario qualche accorgimento in più.

Il codice C inizia con alcune impostazioni iniziali che vengono richieste all'utente tramite *scanf* e sono rispettivamente la lunghezza della finestra in campioni, lo shift in semitoni desiderato e la frequenza della sinusoide che verrà utilizzata come ingresso. Come test infatti viene generata una sinusoide ad una determinata frequenza già all'interno del codice.

I blocchi fondamentali del codice, come ad esempio generazione rampe, generazione finestre e ciclo che svolge le operazioni principali dell'algoritmo, sono analoghi a quelli descritti nel paragrafo 2.3, con l'aggiunta dei formalismi richiesti dal linguaggio C: definizione iniziale delle variabili, allocazione dinamica della memoria (eseguita con il comando *malloc*) e gestione degli array tramite puntatori. Proprio con lo scopo di facilitare la gestione e la creazione degli array è stata creata un'apposita funzione chiamata "make_vector" che prendendo in ingresso gli estremi *a* e *b*, un passo *step* e un puntatore, riempie la locazione di memoria puntata da tale puntatore con un vettore di numeri da *a* a *b* con passo pari a *step* (vedi codice nella pagina successiva).

La finestra utilizzata in questo caso è stata così definita " WIN1[i] = 0.5 - 0.5*cos(2.0 * pi * x[i]) " dove x[i] è semplicemente un vettore di valori progressivi che creato tramite la funzione c "linspace".

```
void make_vector(double a, double step, double b, double u[])
{
    double c;
    int i;

    /* Numero di punti */
    c = floor(b/step - a/step) + 1;

    for(i = 0; i < c; ++i)
        u[i] = a + i*step;
}
```

Un'applicazione di questa funzione all'interno del codice è stata la generazione del segnale sinusoidale da usare come "test input" citata precedentemente. In questo caso viene utilizzata per generare l'asse dei tempi, qui chiamato ramp.

```
make_vector(1/(double)Fs, 1/(double)Fs, TIME, ramp);

    for(i=0; i<Fs*TIME; i++)  signal[N+i] = sin(2.0*pi*
frequency * ramp[i]);
```

Una volta svolte tutte le operazioni necessarie allo shift, vengono salvati su file di testo sia la sinusoide generata e utilizzata come ingresso, sia l'uscita che è risultata dalle elaborazioni in modo da poter valutare i risultati ottenuti.

Tali file di testo sono stati poi analizzati, ascoltati e confrontati su MATLAB. Di seguito sono riportati i risultati posti a confronto.

2.5 Confronto risultati

Parametri utilizzati in entrambi i codici:

- Lunghezza finestra di elaborazione e rampe: 4096
- Pitch shift: +7 semitoni
- Sinusoide in input a frequenza 440Hz (La) campionata a 44,1kHz. Nel codice MATLAB per generare la sinusoide è stato utilizzato un sintetizzatore software esterno, la cui uscita è stata salvata su un file ".wav" e poi elaborata da MATLAB.

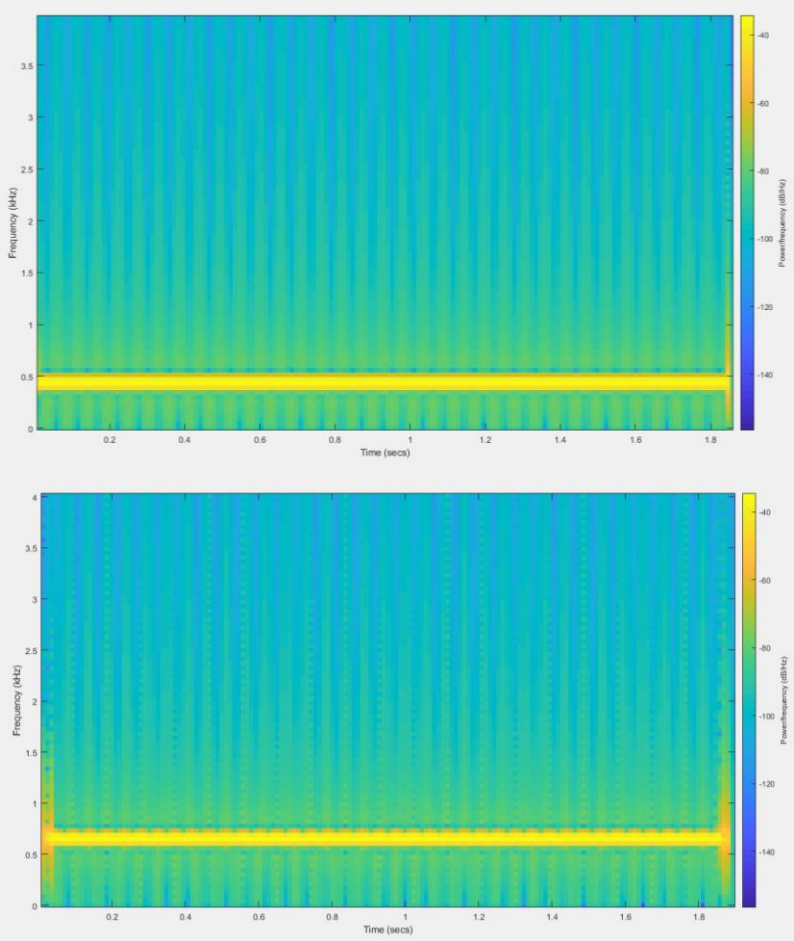

Figura 2.7: Spettrogramma della sinusoide in ingresso a 440Hz e di quella in uscita ottenuta con il codice MATLAB a circa 660Hz, corrispondente quindi al Mi (frequenza esatta 659Hz). Si possono notare dei piccoli artefatti all'inizio e alla fine della sinusoide, in particolare quello iniziale è dovuto alle operazioni svolte dal codice e quello finale al modo in cui è stata generata la sinusoide iniziale e il suo successivo troncamento imperfetto. Un artefatto meno amplificato è infatti presente anche in ingresso.

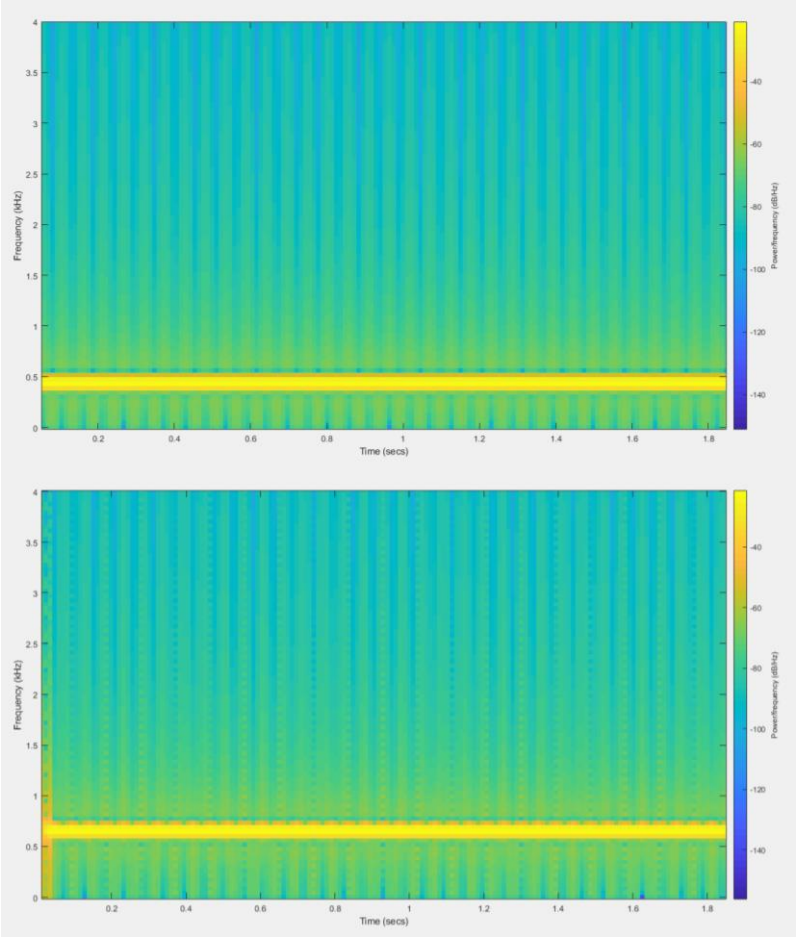

Figura 2.8: Spettrogramma della sinusoide in ingresso a 440Hz e di quella in uscita ottenuta con il codice C a circa 660Hz. Come si può notare il risultato è il medesimo del codice MATLAB. Gli artefatti non sono presenti in coda poiché la sinusoide in ingresso è stata generata numericamente ed è molto più pulita alle estremità.

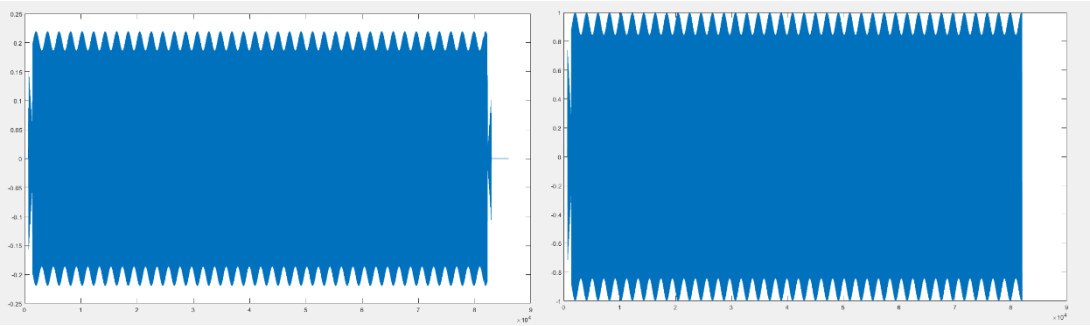

Figura 2.9: A sinistra è riportata l'uscita nel tempo ottenuta dal codice MATLAB e a destra quella ottenuta con il codice C. Entrambe presentano un inevitabile ripple dovuto alla modulazione e cross-fading attuate utilizzando una finestra di tipo cosinusoidale. La differenza in ampiezza è dovuta a un diverso tipo di normalizzazione in fase di scrittura su file, ininfluente ai fini della valutazione. Si può osservare anche nel tempo l'artefatto in testa e in coda alla sinusoide nel primo risultato e solo in testa nel secondo, dove è invece presente un taglio netto alla fine.

La modulazione in ampiezza che risulta nel segnale in uscita dipende sia dalla frequenza del segnale in ingresso sia dalla lunghezza delle rampe (e quindi la larghezza della finestra), poiché varia la frequenza con cui avviene il cross-fading. Negli spettrogrammi di figura 2.7 e 2.8 è possibile anche notare i disturbi introdotti dai salti tra le delay-line che si presentano con un periodo pari alla lunghezza delle rampe. Tali disturbi sono molto attenuati e risultano impercettibili.

Pitch shift e lunghezza delle rampe sono inoltre collegati. Infatti al diminuire della lunghezza delle rampe si avrà una perdita di qualità dovuta all'infittirsi dei disturbi prodotti dal continuo salto da una delay-line all'altra e inoltre, prendendo rampe troppo corte e un pitch shift troppo elevato, l'algoritmo non riesce a modificare il segnale come dovrebbe, non riuscendo quindi a raggiungere lo shift desiderato.

Capitolo 3

PORTING DEL CODICE C SU BOARD AUDIOLINO

La trascrizione dell'algoritmo in C è stato un passaggio intermedio finalizzato al porting sulla board. Con porting si intende un processo di trasposizione di un software su una piattaforma diversa da quella originale, implicando anche delle necessarie modifiche alla struttura.

Dall'azienda Swapp è stato fornito uno scheletro di progetto da riempire con il codice che svolge la vera e propria elaborazione, già configurato e con tutto il necessario affinché la board funzioni correttamente. I file relativi all'algoritmo sono 5, 3 file sorgente e 2 header file.

3.1 Analisi dell'header file

Il file "pitch_dl_priv" è un header file che contiene una serie di strutture e macro utili che verranno poi sfruttate dal codice. È quindi un file di "servizio" che viene utilizzato dagli altri file tramite la direttiva "#include". Di seguito sono riportati alcuni esempi di strutture utilizzate e che verranno successivamente citate.

```
typedef struct
{
	float gkt;
	float amp;
	float dry;
	float wet;
	float pitch;
	float t1[4096];
	float t2[4096];
	float delay1[4096];
	float delay2[4096];
	float i1[4096];
	float i2[4096];
```

```
        float frac1[4096];
        float frac2[4096];
        float WIN1[4096];
        float WIN2[4096];

} PITCH_DL_DESC;
```

Questa struttura ha all'interno array inizializzati per le variabili che vengono utilizzate all'interno del codice e i parametri fondamentali.

```
typedef struct
{
        float *WIN1;
        float *WIN2;
        float *i1;
        float *i2;
        float *frac1;
        float *frac2;
        int calls;
} RAMPS;
```

Seconda struttura necessaria che raccoglie i puntatori agli array precedenti per renderli accessibili a tutti i file del codice rendendoli variabili globali.

```
typedef struct
{
        PITCH_DL_DESC   *p_desc;
        PITCH_DL_STATE *p_state;
        RAMPS *p_ramps;
        float *p_in[ALG_PITCH_DL_N_IN];
        float *p_out[ALG_PITCH_DL_N_OUT];

        PITCH_DL_ST state;
        bool start_requested;
        bool term_requested;
        int32_t update_flag;
        int16_t      param_value[PITCH_DL_PARAM__NUM];

        ALG_RAMP_GROUP_RES  *p_global_ramp_group;
        ALG_RAMP_COEFF_RES  *p_global_ramp_coeff;

} PITCH_DL_PRIV;
```

La principale struttura che racchiude all'interno i puntatori alle strutture principali, possiamo infatti notare *p_desc, *p_state e *p_ramps. Gestisce inoltre i campioni in input e output tramite *p_in e *p_out.

3.2 Analisi dei file sorgente

Il file “pitch_dl_main.c” è il file in cui risiede il cuore dell’algoritmo. Nella prima parte vengono svolte operazioni di routine come quella di gestione risorse/memoria e la definizione dei parametri utilizzati. Da notare che il pitch shift da numero di semitoni è stato convertito in *cents* per esigenze della scheda. Ogni semitono è diviso in 100 cents quindi avremo un range che va da +1200 a -1200 (+1 ottava, -1 ottava).

La funzione “pitch_dl_init” è la prima funzione presente all’interno del main e si occupa di inizializzare variabili e puntatori che verranno utilizzati all’interno del codice, nonché di generare le finestre cosinusoidali utilizzate per il cross-fading analogamente ai codici già analizzati, con l’unica accortezza di salvare il i valori che definiscono le finestre negli appositi array “globali” chiamati WIN1 e WIN2, presenti della struttura PITCH_DL_DESC.

Successivamente sono definite una serie di funzioni di “servizio”: la funzione di start che analizza quando viene richiesto uno start del codice, la funzione analoga di terminazione e una funzione di set e get parameter relativa ai parametri che vengono utilizzati.

Il codice principale è scritto all’interno della funzione “pitch_dl_run”, tale funzione viene chiamata ciclicamente ogni 64 campioni, poiché il buffer in uscita e in ingresso della scheda è pari a un massimo di 64 campioni. L’elaborazione avviene un canale alla volta, per cui è stato necessario inserire un ciclo esterno che selezionasse prima un canale e poi l’altro.

```
for(ch=0; ch<ALG_PITCH_DL_N_CH;ch++)
	{
		const float *p_in = p_priv->p_in[ch];
		float *p_out = p_priv->p_out[ch];
```

Alla fine di ogni ciclo verrà mandato in out il risultato elaborato sfruttando il puntatore *p_out sopra citato.

Poiché il buffer e l’elaborazione è limitata a 64 campioni per volta, è stato necessario inserire una variabile globale (visibile nella struttura PITCH_DL_PRIV) chiamata “calls”, che tiene traccia del punto in cui il codice è arrivato a leggere la rampa della delay-line per evitare di ricominciare a leggere i valori da capo a ogni chiamata successiva della funzione “run”. Questo è stato fatto sommando un indice con valore pari al valore attuale della variabile “calls” all’interno del processo di lettura delle rampe. A ogni fine ciclo questa variabile viene incrementata di 64 fino a che non raggiunge 4096. Raggiunto tale valore la variabile viene azzerata poiché si è arrivati alla fine della rampa.

```
                ind1    = i1[call+index];
                ind2    = i2[call+index];
                frac01 = frac1[call+index];
                frac02 = frac2[call+index];


                                [...]


if(call==4032)
      p_ramps->calls=0;
p_ramps->calls=call+bufsize;
```

Il secondo file principale è il file "pitch_dl_update" all'interno del quale è presente la parte dedicata all'aggiornamento delle variabili e quindi tutto quello che è dipendente dalla variazione dello shift in semitoni. In particolare quando viene modificato un parametro, tramite una bitmask viene settato un flag e viene chiamata la funzione dedicata all'update delle variabili. All'interno troviamo la parte di generazione delle rampe e definizione dei parametri WIDTH e *verse*. Tali rampe una volta calcolate vengono salvate all'interno delle strutture presenti nell'header, con i valori già separati in parte intera e parte frazionaria.

```
//indici
          for(i = 0; i < N; i++) delay1[i] = t1[i]*width + 1.0;
          for(i = 0; i < N; i++) delay2[i] = t2[i]*width + 1.0;
          for(i = 0; i < N; i++) i1[i]    = floor(delay1[i]);
          for(i = 0; i < N; i++) i2[i]    = floor(delay2[i]);
          for(i = 0; i < N; i++) frac1[i] = delay1[i] - i1[i];
          for(i = 0; i < N; i++) frac2[i] = delay2[i] - i2[i];

             RAMPS *p_ramps=p_priv->p_ramps;

             p_ramps->i1=i1;
             p_ramps->i2=i2;
             p_ramps->frac1=frac1;
             p_ramps->frac2=frac2;
```

Capitolo 4

CONCLUSIONI

Con questo lavoro si è arrivati ad adattare e preparare un codice esistente di un algoritmo di pitch shifting per l'implementazione su board Audiolino fornita dall'azienda SWAPP tech. È stato prima eseguito uno studio di 4 algoritmi con il medesimo scopo ma implementati con tecniche differenti (ognuno con pro e contro specifici), per poi procedere con la loro successiva validazione e verifica dell'effettiva attendibilità dei risultati, testandoli usando come input segnali semplici e analizzando poi l'accuratezza del risultato sfruttando MATLAB e software esterni. L'azienda proprietaria ha poi selezionato per l'implementazione l'algoritmo ottenuto con la tecnica delle due delay-line.

Si è poi proseguito con il confronto tra i risultati ottenuti in MATLAB e quelli ottenuti dal codice C dell'algoritmo basato sulle due delay-line. Il codice C è stato usato come step intermedio prima di passare su board.

Come ultimo passo è stato necessario adattare il codice alla struttura file già esistente e necessaria al funzionamento di un qualsiasi codice all'interno della board Audiolino, tale passaggio è stato fatto utilizzando come IDE STM32Cube.

Bibliografia e riferimenti

[1] https://tecnologiamusicale.wordpress.com/2012/07/30/la-relazione-tra-le-frequenze-e-le-note-musicali/

[2] http://www.audiolino.com/

[3] http://www.audiolino.com/downloads/brick/FW01001a%20-%20Audiolino%20Brick%20Firmware%20v1.0.0.pdf – Audiolino Brick Firmware

[4] http://www.audiolino.com/downloads/brick/UM01001a%20-%20Audiolino%20Brick%20User%20Manual.pdf – Audiolino User Manual

[5] http://msp.ucsd.edu/ - Miller Puckette