UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI INGEGNERIA

Corso di Laurea Triennale in Ingegneria Informatica e dell'Automazione

Studio di un plugin musicale VCV-Rack e porting dello stesso in Juce

Study of a VCV-Rack music plugin and port of the same in Juce

Relatore:

Prof. Stefano Squartini

Tesi di Laurea di:

Samuele Del Moro

Correlatore:

Ing. Leonardo Gabrielli

A.A. **2019 / 2020**

Obiettivo

La presente Tesi ha come obiettivo lo studio del framework Juce per la progettazione di un Plugin DSP in ambito musicale e l'implementazione di un algoritmo specifico sotto tale framework. L'algoritmo realizza un oscillatore secondo la tecnica DPW, adottata per la riduzione dell'aliasing, fenomeno che si verifica con la generazione a tempo discreto di segnali a banda illimitata.

Il lavoro si è svolto partendo da un'implementazione di tale tecnica in VCV Rack di cui è stato fatto il porting su Juce, aggiungendo funzionalità ulteriori come un generatore di inviluppo ADSR ed un filtro Virtual Analog in modo da ottenere una vera e propria voce di sintetizzatore.

Indice

Capitolo 1: Introduzione

1.1 Fondamenti di DSP

Il campionamento è un processo di estrazione dei valori del segnale continuo in punti equidistanti nel tempo. Il teorema del campionamento dice che per campionare correttamente (senza perdita di informazioni) un segnale a banda limitata, è sufficiente campionarlo con una frequenza di campionamento pari almeno al doppio della massima frequenza del segnale (tale frequenza viene anche detta frequenza di Nyquist).

$$Fc > 2Fs$$

Il campionamento di un segnale produce un segnale campionato il cui spettro è composto da diverse repliche del segnale originale, repliche collocate a frequenze multiple della frequenza di campionamento.

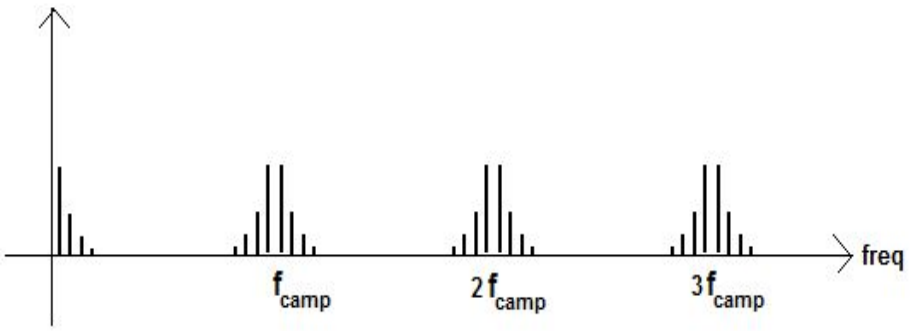


Figura 1. Campionamento di un segnale

Se il segnale da campionare non ha banda limitata, le frequenze più alte delle diverse repliche dello spettro si sovrappongono, interferendo tra loro.

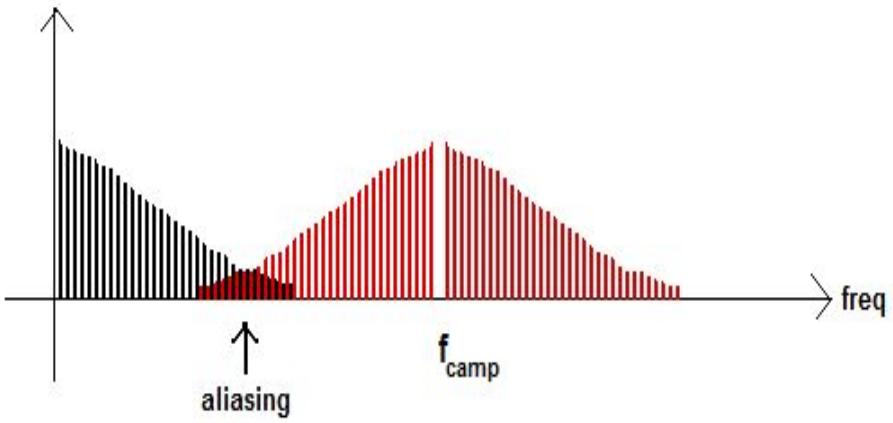


Figura 2. Aliasing

Come si può osservare dalla figura precedente, a causa della sovrapposizione delle "code" dello spettro del segnale originario con quelle prodotte dal campionamento, la ricostruzione corretta del segnale a partire dai campioni non è più possibile. Tale fenomeno prende il nome di **aliasing**.
Una singola componente di frequenza può essere rappresentata come onda sinusoidale e la combinazione di tali onde di diverse frequenze produce suoni complessi.

Altri tipi di forme d'onda di base utilizzati sono:

- Triangolare: ha solo armoniche dispari che decadono come 1/n^2 dove n è il numero armonico e 1 è l'ampiezza della frequenza fondamentale
- Quadrata: con armoniche dispari che decadono come 1/n
- Dente di sega: con armoniche pari e dispari che decadono come 1/n
- Impulso: assume solo valori -1 e 1

1.2 Oscillatore

Gli oscillatori si trovano alla base dell'architettura progettuale di un sintetizzatore. Sono infatti i generatori principali del suono che sarà poi filtrato e processato da altri moduli dello strumento. Sono quindi il primo passo verso la definizione di un timbro, la loro qualità e versatilità influenzerà fortemente il carattere del sintetizzatore.

Compito dell'oscillatore è di generare una variazione di tensione ciclica quindi ripetuta nel tempo definita forma d'onda. Nel caso in cui la tensione variasse in modo casuale e non ripetitivo si otterrebbe un generatore di rumore.

L'oscillatore genera una forma d'onda a frequenza fissa finché non si invia un nuovo comando. La ripetizione dell'onda nel tempo è definita ciclo, se il ciclo si ripete da 20 a 20.000 volte al secondo l'orecchio umano potrà percepirlo. La misurazione di quante volte un ciclo si ripete in un secondo è misurata in Hertz, se un oscillatore emette un suono a 7000 Hz significa che il ciclo si ripete 7000 volte in un secondo.

Questa tensione, una volta amplificata, andrà a far muovere le membrane degli altoparlanti che genereranno un suono udibile. Le infinite possibilità di variazione della tensione corrispondono alla enorme varietà di suoni udibili.

I sintetizzatori analogici normalizzati sono generalmente dotati di un basso numero di oscillatori per ogni voce (da 1 a 3 solitamente), ogni oscillatore può emettere una determinata forma d'onda. Questo tipo di architettura facilita al massimo la creazione di timbri semplici ma rende più impegnativa la creazione di timbri emulativi di buon livello, funzione comunque spesso richiesta dal musicista. Il programmatore deve infatti cercare di utilizzare i pochi generatori a disposizione per creare una forma d'onda complessa.

Gli oscillatori dei sintetizzatori analogici si possono dividere in 2 tipi:

1 **VCO** Voltage Controlled Oscillator (oscillatore controllato dalla tensione).

2 **DCO** Digitally Controlled Oscillator (oscillatore controllato digitalmente)

1.3 Filtri

Il filtro è un sistema lineare invariante nel tempo, che forma lo spettro di frequenza del segnale in ingresso. Nella sintesi del suono i filtri attenuano o amplificano particolari intervalli di frequenza.

Quattro tipi fondamentali di filtri utilizzati in sintesi (presentati nella fig.3) sono:

- Filtro passa basso: attenua le frequenze al di sopra di una certa frequenza
- Filtro passa alto: attenua le frequenze al di sotto di una certa frequenza
- Filtro elimina banda (o filtro notch) : taglia una banda ristretta di frequenza, inclusa in un certo intervallo delimitato da due valori.
- Filtro passa banda: vengono attenuate le frequenze al di fuori di un certo intervallo di frequenze.

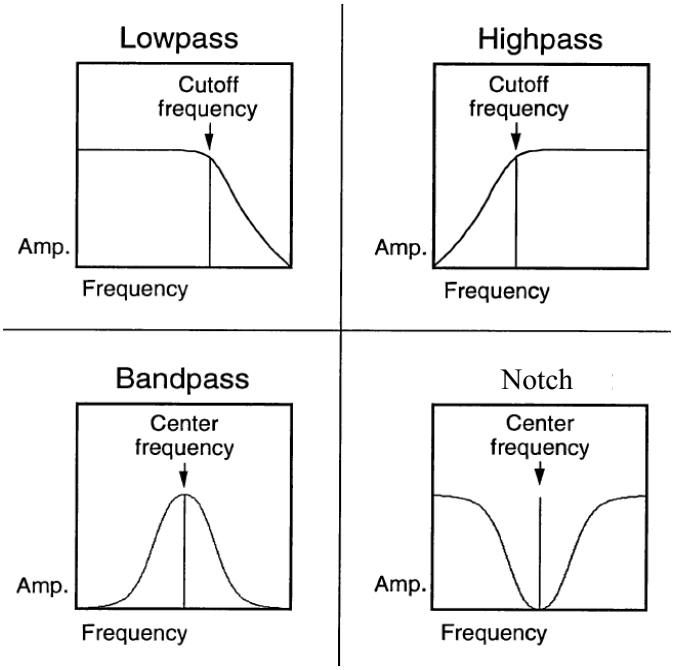


Figura 3. Curve di risposta dei vari tipi i filtri base. Più è bassa la curva di risposta e più la frequenza selezionata è attenuata. Nella figura la frequenza di taglio è marcata.

Un altro parametro importante è il parametro Q o risonanza. E' stato originariamente definito per i filtri passa banda e elimina banda.

$$Q = \frac{f_{center}}{f_{highcutoff} - f_{lowcutoff}}$$

dove f_{center} è la frequenza al centro della banda passante, è uguale alla media geometrica delle frequenze di taglio inferiore e superiore. $f_{highcutoff}$ è la frequenza alla quale la risposta del filtro raggiunge il segmento in banda passante, tipicamente corrisponde a -3 dB, $f_{lowcutoff}$ è invece la frequenza alla quale la risposta del filtro supera il segmento in banda passante, anch'esso tipicamente corrisponde a -3 dB. Avere un alto valore di Q significa che la curva di risposta del filtro è stretta e con un certo guadagno applicato il filtro può diventare risonante. Ciò vuol dire che si avrà come risultato un suono acuto che creerà una componente di segnale aggiuntiva. Il termine risonanza è stato esteso per il filtro passa-basso e passa-alto dove determina indirettamente la dimensione di un picco di guadagno supplementare alla frequenza di taglio (figura 4). La frequenza di taglio è quindi determinata per Q = 0. Aumentando la risonanza del filtro si crea un picco nello spettro ampiezza-frequenza.

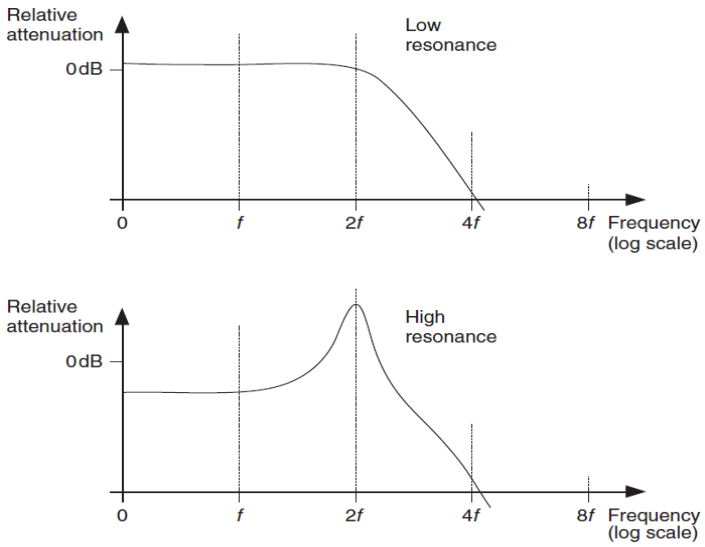


Figura 4. Una curva di risposta del filtro passa basso con valore dei parametri di risonanza basso (fig. sopra) e alto (fig. sotto). La frequenza di taglio è impostata sulla seconda armonica di frequenza

L'ultima proprietà dei filtri che verrà discussa qui è l'attenuazione della banda passante. È espressa in dB per ottava e descrive quanto è ripida la curva di risposta del filtro nella banda passante (gamma di frequenze tra le regioni di attenuazione minima e massima): una regola empirica è che il filtro passa basso unipolare (primo ordine) ha un'attenuazione della banda passante pari a 6 dB e con ogni ordine successivo viene aumentato questo valore di altri 6 dB. Valori tipici dell'attenuazione della banda passante nei sintetizzatori sono 12dB o 24dB.

1.4 Controlli

1.4.1 Inviluppo

Un inviluppo è un mezzo per visualizzare una tendenza generale in una funzione (come l'ampiezza del segnale nel tempo) e sono rappresentati come curve che i massimi locali della funzione raggiungono, ma non superano.

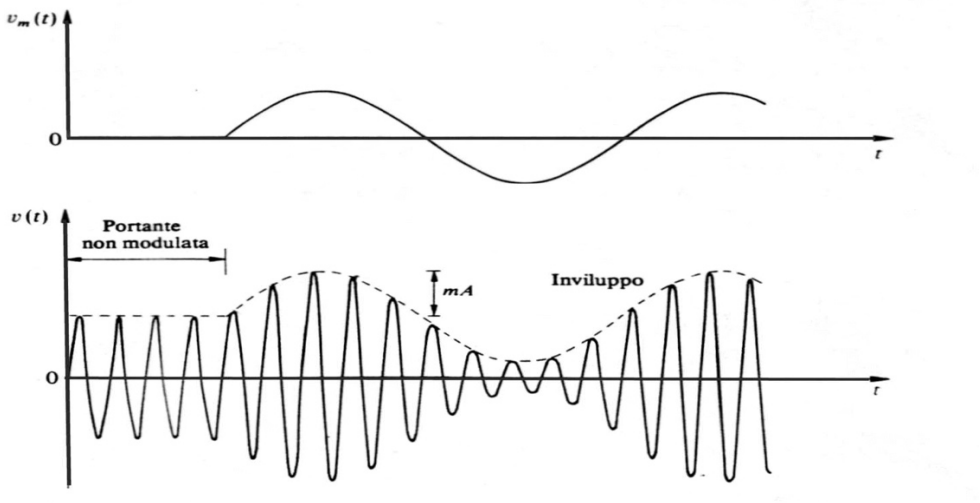


Figura 5. Inviluppo in Ampiezza.

Nelle applicazioni musicali l'inviluppo ha assunto un significato leggermente diverso, essendo sostanzialmente equivalente ad un cambiamento di un particolare parametro nel tempo.

L'uso tipico comprende:

- controllare il volume di una singola nota in tempo dopo la nota sull'evento
- controllare il valore del parametro CutOff in tempo dopo la nota sull'evento
- valori di ampiezza parziali nel tempo nella sintesi additiva, dove ogni componente di frequenza del segnale ha un proprio inviluppo.

Quando l'inviluppo è usato come un cambiamento del volume di una nota nel tempo, alcuni segmenti possono essere estratti da qualsiasi analisi dello strumento musicale:

- Attacco (Attack): un periodo di tempo che impiega il volume per passare da zero al suo valore massimo dopo un gesto appropriato (premendo un tasto, piegando una corda). Può essere veloce (un paio di millisecondi) nel caso di un pianoforte, una chitarra o lento (poche centinaia di millisecondi) nel caso di un Trombone. Tempi di attacco artificialmente lunghi sono spesso utilizzati nella musica elettronica per creare suoni ambient che lentamente si accumulano e si evolvono nel tempo.
- Decadimento (Decay): un periodo di tempo in cui il suono svanisce dal valore di picco al livello più o meno stabile. Questa parte può essere spesso trascurata quando si ascoltano strumenti musicali acustici: esempi di strumenti con parti di decadimento facilmente visibili sono un pianoforte e un clarinetto.
- Sostegno (Sustain): un periodo in cui il suono di uno strumento è a volume costante. La produzione di un volume costante di una nota può essere ottenuta tenendo una chiave o utilizzando un pedale sustain ed è tipicamente presente in strumenti come l'organo o un pianoforte.
- Rilascio (Release): un periodo di tempo dopo la fine dell'interazione musicista-strumento su una nota.

I suddetti segmenti possono essere combinati per creare diversi tipi di buste. Il più comunemente implementato è l'Attack-Decay-Sustain-Release envelope (ADSR), presentato in figura. Inizia con una parte di attack attivata da una nota MIDI sul messaggio e poi aumenta il volume fino a quando il tempo specificato passa e il livello raggiunge il suo picco.

Poi il decay inizia il segmento e il volume diminuisce portando al segmento di sustain dove il volume è costante finché non viene inviato alcun messaggio di note off (o se si tiene il pedale sustain). Quando il tasto o il pedale sustain viene rilasciato (qualunque sia l'ultimo) il segmento di rilascio entra per creare una dissolvenza per il suono. È una metafora semplice ma efficace che permette ad un sintetizzatore di imitare gli strumenti musicali fisici. I parametri controllabili in questo tipo di inviluppo sono:

- Attack Time
- Decay Time
- Sustain Level
- Release Time

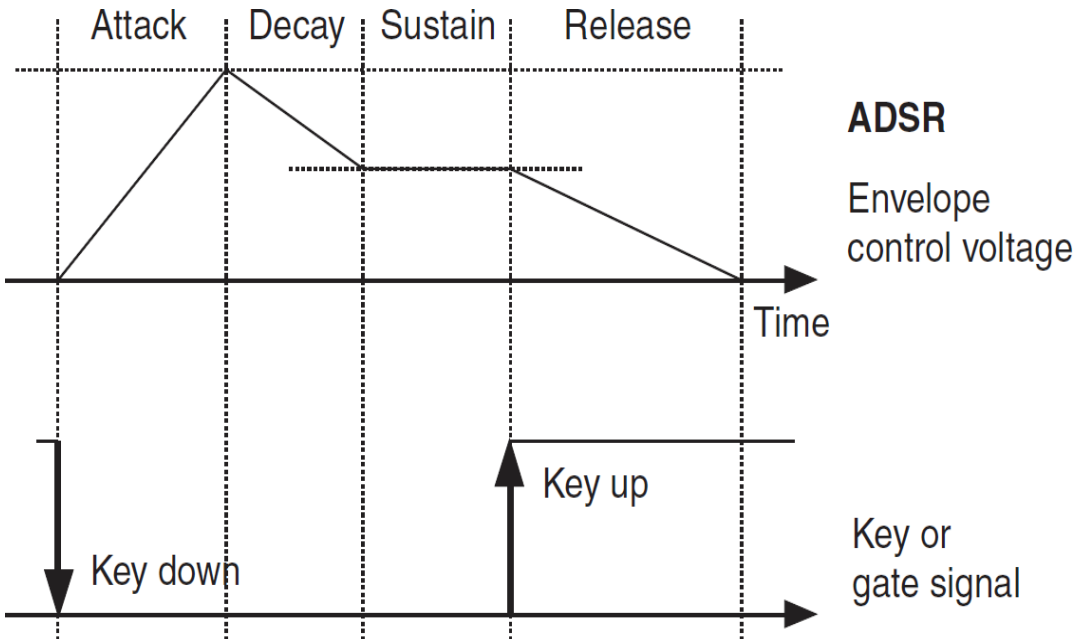


Figura 6. Inviluppo ADSR. Il segmento di attacck inizia con la pressione di un tasto, passa al decay e poi alla parte di sustain. Con il rilascio del tasto l'inviluppo passa immediatamente al segmento di Release (indipendentemente dal segmento in elaborazione)

1.4.2 MIDI

Musical Instrument Digital Interface (MIDI) è un'interfaccia digitale seriale che consente lo scambio di informazioni tra strumenti musicali elettronici e computer. Originariamente i messaggi MIDI sono stati inviati utilizzando cavi con connettori DIN a cinque pin, ma le attuali implementazioni utilizzano connettori e cavi mLAN o USB. Attraverso le interconnessioni della porta alcuni messaggi MIDI vengono inviati e ricevuti. I tipi di porte MIDI includono la porta IN (destinazione o destinazione a cui i messaggi sono inviati), la porta OUT (sorgente che invia i messaggi MIDI) e la porta THRU (porta di uscita che instrada il segnale dalla porta IN). Il MIDI fornisce un'astrazione dei controlli dove vengono generati messaggi particolari, ma il loro utilizzo è completamente all'altezza dell'hardware o del software di destinazione.

I messaggi vengono inviati in modo seriale (uno dopo l'altro) sotto forma di numeri binari il più velocemente possibile, cioè senza alcuna informazione temporale.

Per semplicità, solo il controller della tastiera sarà indirizzato.

Tutte le descrizioni dei messaggi MIDI saranno relative al controller della tastiera. Gli unici messaggi che sono di interesse in questo progetto sono i seguenti:

- Note On : la nota indica che è stato premuto un tasto; contiene informazioni supplementari sotto forma di un numero di nota MIDI e del parametro di *velocity*, un'indicazione della velocità con cui il tasto è stato premuto, che può essere utilizzato per applicare un volume appropriato alla nota.
- Note Off: indica che una determinata nota è stata rilasciata.
- Pitch Bend
- Aftertouch: valore relativo alla pressione supplementare esercitata sulla chiave.

Il MIDI introduce la propria numerazione dei tasti con il numero 69 corrispondente al tasto A4 sulla tastiera (tipicamente accordata a 440 Hz). Altri tasti hanno numeri con un numero aggiunto di semitoni tra di loro e il tasto A4 seguendo la scala cromatica, ad esempio, il G4 è 67, F4 è 65, e così via.

Nel framework JUCE (descritto nel Capitolo 3) i messaggi MIDI vengono passati in un'istanza di classe *Midibuffer* insieme all'istanza di classe *Audiobuffer*. Ogni messaggio nel buffer ha un *timestamp* che indica dove nel buffer audio dovrebbe essere interpretato il messaggio.

Per controllare un plugin attraverso il MIDI si ha bisogno di un controller MIDI e un protocollo di comunicazione tra controller e motore di sintesi. I controller MIDI sono un tipo speciale di interfaccia MIDI con un dispositivo di input integrato. I metodi di input tipici disponibili sui controller MIDI sono tasti di piano e pad sensibili alla pressione, ma la maggior parte include manopole e cursori per il controllo anche di altri parametri.

Il controller MIDI più comunemente usato è quello per tastiera musicale elettronica: quando vengono suonati i tasti, invia i dati MIDI sull'intonazione della nota, sulla forza con cui è stata suonata e sulla sua durata.

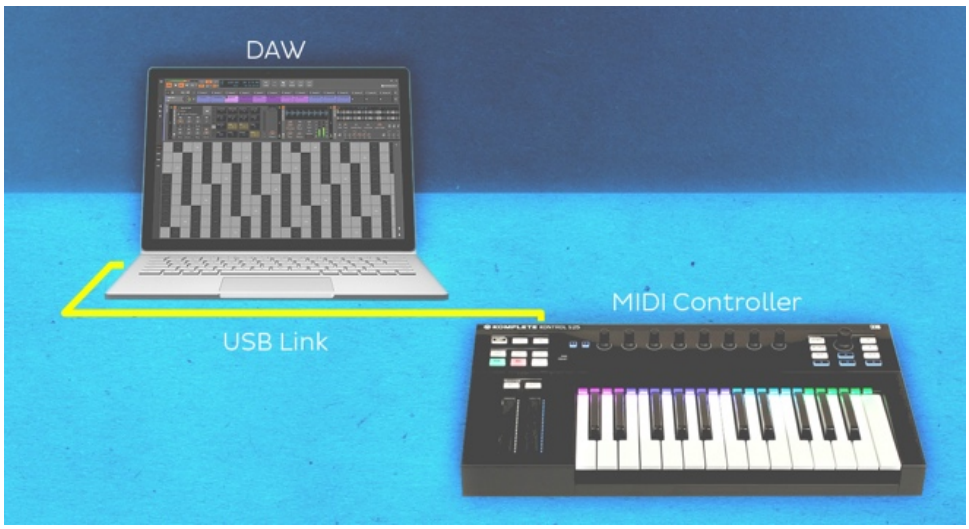
(a)

(b)

Figura 7. Esempi di interfaccia Midi USB (a), e di un controller MIDI (b)

Ci sono tanti modi per usare il MIDI, ma ci sono alcuni tipici flussi di lavoro MIDI che la maggior parte dei produttori di musica usano, come possiamo vedere dagli esempi riportati sotto.

(a)



(b)

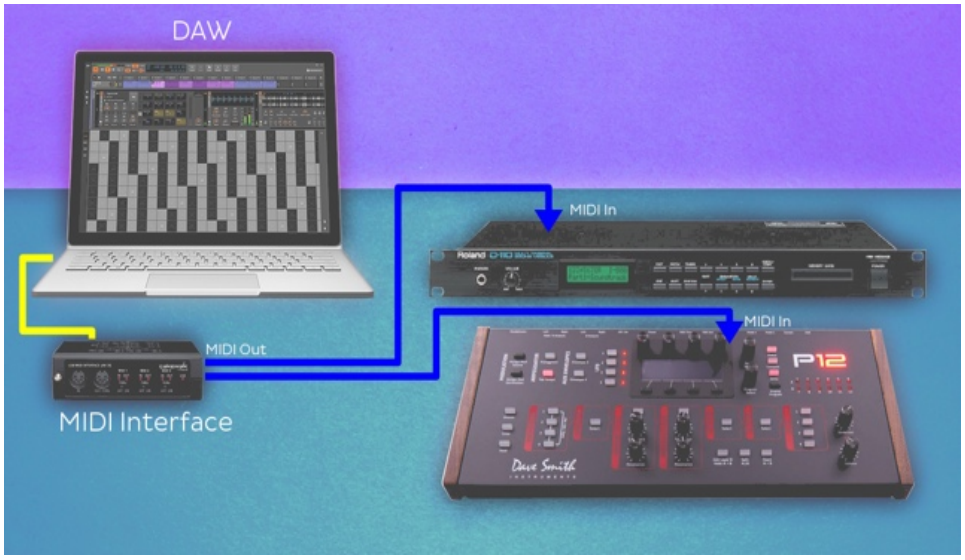


Figura 8. Esempi di setup più comuni. Computer e controller MIDI (a), Computer, interfaccia MIDI e Sintetizzatori (b)

Capitolo 2: Oscillatori Virtual Analog ad aliasing limitato

Un obiettivo degli algoritmi analogici virtuali è stato quello di creare un'implementazione efficiente e priva di aliasing degli oscillatori.
Le forme d'onda ideali, come quella quadra o a dente di sega e ad impulso, hanno spettri di banda infinita e sono soggette a problemi di aliasing quando vengono riprodotte. Esistono diversi modi per implementare un algoritmo in grado di ridurre l'aliasing: attraverso l'oversampling, il campionamento a banda limitata oppure il metodo DPW(Differentiated Parabolic Waveform) che sarà affrontato successivamente.
Questa sezione viene discussa l'implementazione di un oscillatore con riduzione di aliasing attraverso il DPW.

2.1 Oscillatore Triviale

Generare le forme d'onda con tipologie diversificate: l'onda quadra, a dente di sega o triangolare, è abbastanza semplice nel dominio del tempo. Le varie onde possono essere generate cosi:

- Forma d'onda a dente di sega: implementazione di un modulo *counter* (cioè un contatore crescente che si azzera quando viene raggiunta una certa soglia *th*). Il valore del contatore viene inviato direttamente in uscita.
- Forma d'onda quadrata: usa il contatore come nel dente di sega ma cambia l'uscita ogni volta che viene raggiunta la soglia.
- Forma d'onda triangolare: si genera incrementando e convertendo il segno di quest'ultimo ogni volta che viene superata la soglia.

Questi metodi di generazione vengono comunemente chiamati Triviali perché sono semplici da implementare.

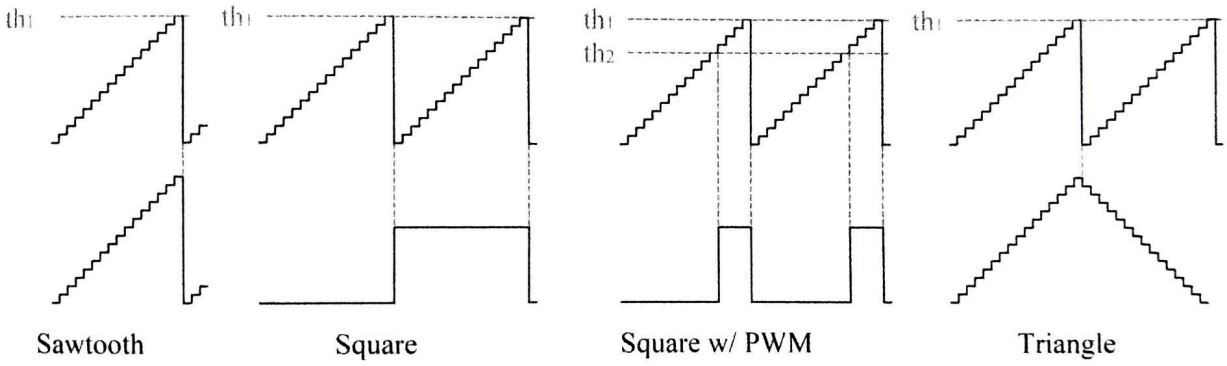


Figura 9. Metodo per la generazione di forme d'onda triviali

2.2 DPW

L'integrale di una derivata di una funzione è la funzione stessa più un valore constante.

$$\int f'(x)dt = f(x) + c \qquad (1)$$

Prendendo in esame un singolo periodo della rampa a dente di sega, si rende noto che è un segmento di una funzione lineare del tipo $f(x) = mx + q$, dove il termine costante q è scelto affinché la forma d'onda abbia DC offset pari a 0 e il coefficiente di pendenza m imposta il tempo di crescita della rampa. Considerando adesso $q=0$ integrando questa funzione lineare e differenziandola otterremo la funzione originale:

$$F(x) = \int f(x) = \int mxdx = \frac{m}{2}x^2 + c \quad \text{(Fase di integrazione)} \qquad (2)$$

$$\frac{dF(x)}{dx} = F'(x) = mx \quad \text{(Fase di Differenziazione)} \qquad (3)$$

Procedendo in questa direzione, non ha senso integrare e differenziare una funzione, tuttavia è stato notato che un segnale simile al dente di sega, ma con aliasing ridotto, può essere generato differenziando un'onda parabolica campionata a tratti.

La forma d'onda parabolica può essere ottenuta integrando un periodo della forma d'onda a dente di sega, che è una funzione lineare all'interno del periodo, ossia:

$$f_2(x) = \int x\,dx = \frac{x^2}{2} + C \qquad (4)$$

Il ridimensionamento può essere trattato separatamente, usando il polinomio x^2 per la sintesi.

La Figura 10 (c) mostra l'onda parabolica campionata e la Figura 10 (d) mostra il suo spettro che decade di circa 12dB/ottava. In linea di principio, la differenziazione (4) ripristina la funzione lineare a tratti.

Quando un segnale a tempo discreto costituito da campioni prelevati dalla curva polinomiale di secondo ordine viene differenziato con un filtro digitale, il risultato è diverso da una classica forma d'onda a dente di sega: uno o più campioni vicino al legame di due periodi adiacenti saranno differenti.

L'esempio è esplicitato nella figura 10 (e) che mostra il segnale a tempo discreto, risultato ottenuto applicando una differenziazione di primo ordine.

Un singolo campione modificato costituisce una transazione graduale da un periodo a quello successivo. Questa proprietà nel dominio del tempo del DPW riduce l'aliasing. La Figura 10 (f) mostra lo spettro della forma d'onda a dente di sega dovuta al DPW del secondo ordine, che ha componenti di aliasing più deboli rispetto allo spettro della forma d'onda a dente di sega triviale.

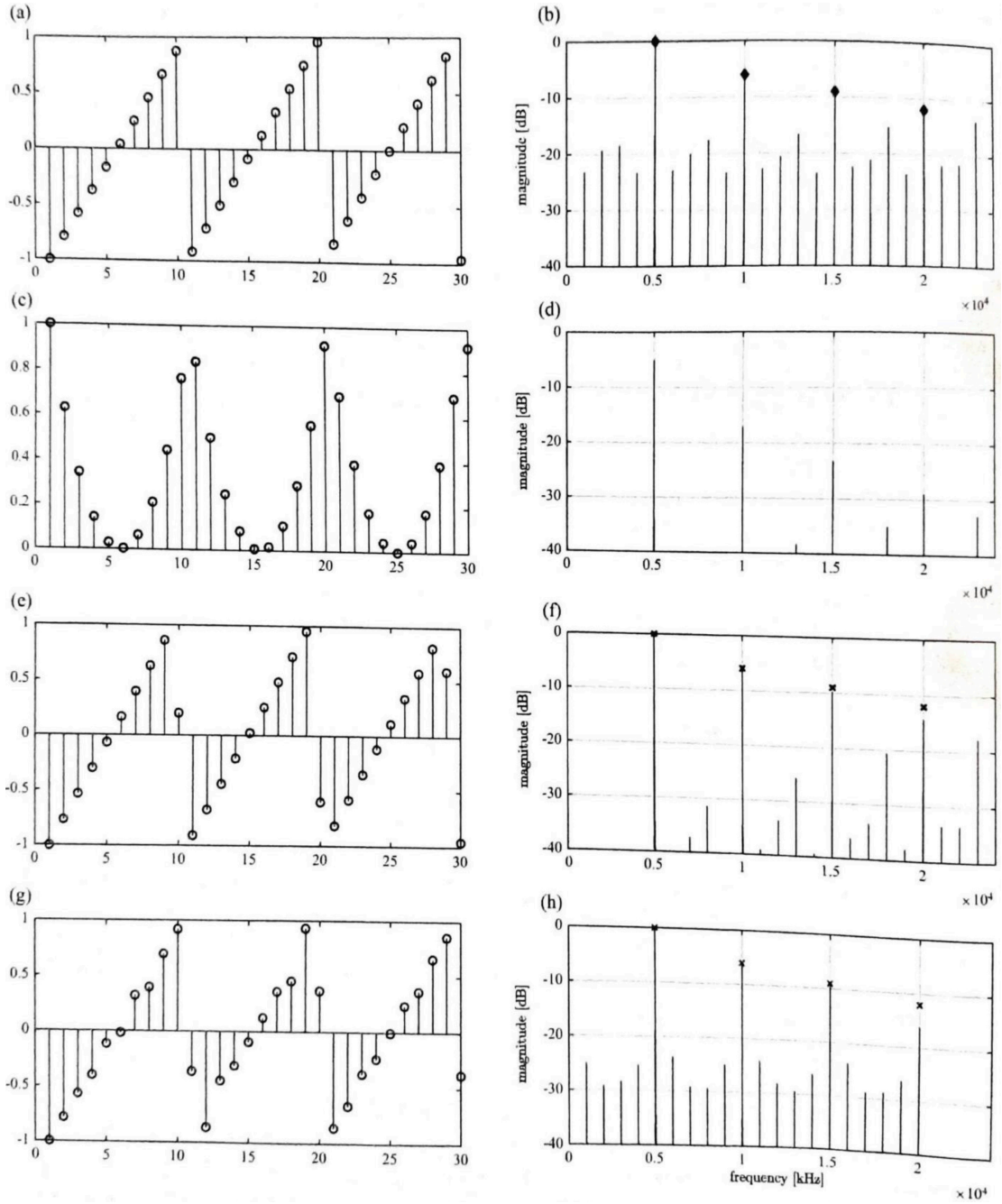


Figura 10. Comparazione tra diversi dente di sega a 5Khz campionati a 48KHz. Onda triviale (a), il quadrato del segnale (funzione parabolica) (c) e la sua derivata prima (e). I relativi spettri si trovano sulla destra (b, d, f).

La Tabella 1 presenta il numero di operazioni necessarie per eseguire le diverse parti dell'algoritmo come la valutazione dei polinomi e il ridimensionamento. Si è visto che il numero di moltiplicazioni, incluso il ridimensionamento, è di circa N e il numero totale di operazioni è di circa 2N, dove N è l'ordine polinomiale e presumendo che la prima differenziazione venga usata per implementare tutte le derivate.

DPW Order	**Polynomial Function**	MUL	ADD	MUL (scaling)	ADD (diff)	Total
N=1	$x\ (trivial\ sawtooth)$	0	0	0	0	0
N=2	x^2	1	0	1	1	3
N=3	$x^3 - x$	2	1	1	2	6

Tabella 1. Funzioni Polinomiali per la generazione di dente di sega usando l'algoritmo DPW con ordini da 1 a 3 per la generazione delle forme d'onda a dente di sega e il loro carico computazionale. I numeri delle moltiplicazioni (MUL) e addizioni (ADD) per valutare ogni polinomio e per implementare N differenziatori e il ridimensionamento.

L'algoritmo DPW è rappresentato nel diagramma di flusso della Figura 11.

Il DPW non è efficace come i metodi BLEP, anche se vanta di una complessità di codice bassa, richiede poca memoria, e ha un costo di calcolo costante.

Un ulteriore vantaggio è la possibilità di iterare il metodo per ottenere una migliore riduzione dell'aliasing.

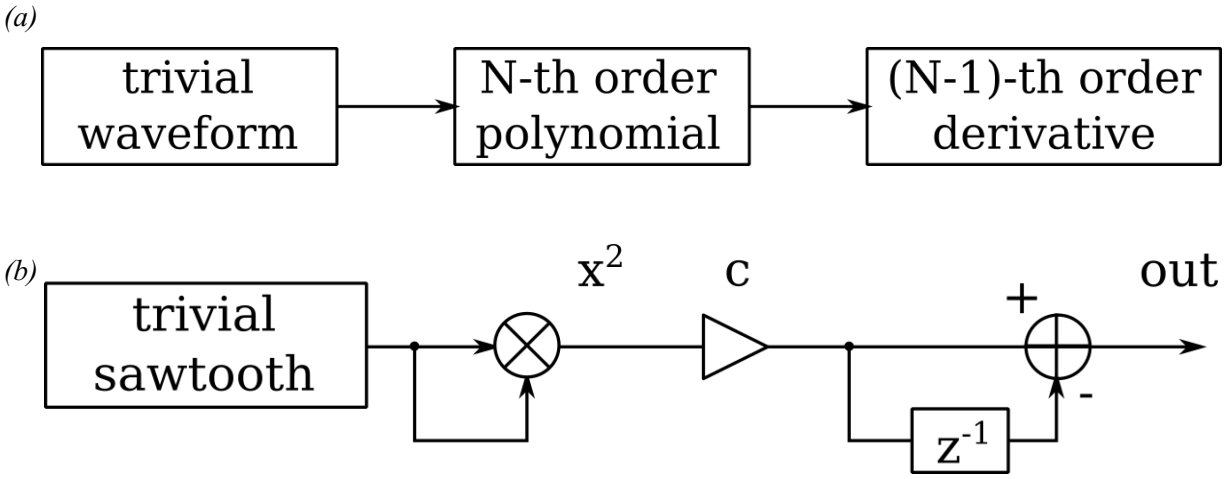


Figura 11. Diagramma di flusso della dell'algoritmo DPW (a), algoritmo DPW di secondo ordine di un dente di sega (b)

Un problema generativo della differenziazione è il processo che riduce l'ampiezza della forma d'onda, specialmente in bassa frequenza. Una soluzione è il ridimensionando della forma d'onda in ogni fase di differenziazione utilizzando un fattore di scala che dipende dall'ordine del polinomio e dal periodo della forma d'onda.

DPW Order	**Polynomial Function (SAW)**	**Basic Scaling Factor**	**Improved Scaling Factor**
N=1	$x\ (trivial\ sawtooth)$	1	1
N=2	x^2	$P/4$	$\pi/[4\sin\left(\frac{\pi}{P}\right)]^2$
N=3	$x^3 - x$	$P^2/24$	$\pi^2/(6\cdot\left[2\sin\left(\frac{\pi}{P}\right)\right]^2)$

Tabella 2. Funzioni Polinomiali per la generazione di dente di sega usando l'algoritmo DPW con ordini da 1 a 3 e i relativi fattori scalari da usare durante la differenziazione

Capitolo 3: Il Framework JUCE

Uno dei cambiamenti significativi intervenuti con l'introduzione del software nella musica è stato il concetto di un plug-in audio come sostituto di un rack di moduli separati (sintetizzatori, sequencer, effetti ecc.) e tali plug-in possono essere utilizzati nel software DAW.

Il plug-in audio è un software che può avere più ingressi e uscite audio e il suo comportamento è determinato dalle istruzioni contenute all'interno del plug-in compilato dal codice sorgente che è stato creato dai programmatori. Il plug-in audio può essere fornito in una varietà di formati tra cui:

- VST (Virtual Studio Technology): creato da Steinberg, il formato VST è stato ampiamente adottato da molte DAW come il formato principale e ha visto lo sviluppo in una forma di terza versione chiamata VST3 (gli strumenti plug-in, come i sintetizzatori, che accettano input MIDI, hanno la lettera 'i' aggiunta alla fine del nome del formato).
- AU (Audiounits): nasce come supporto del sistema operativo MAC X OS, e utilizzato dalle DAW in esecuzione su questo sistema (supporto nativo significa minore latenza e integrazione plug-in stabile).
- AAX (Avid Audio extension): creato dalla società Avid e utilizzato nella DAW della loro produzione denominata Pro Tools.
- RTAS (Real-Time Audio Suite): formato utilizzato nel Pro Tools DAW fino alla versione 10.

Nelle DAW è possibile costruire complessi set-up di plug-in e routing con vantaggi rispetto ai rack e ai banchi analogici, data la mancanza di cavi necessari per le interconnessioni (tutte le connessioni sono gestite dal software host) e per un rapido richiamo preimpostato.

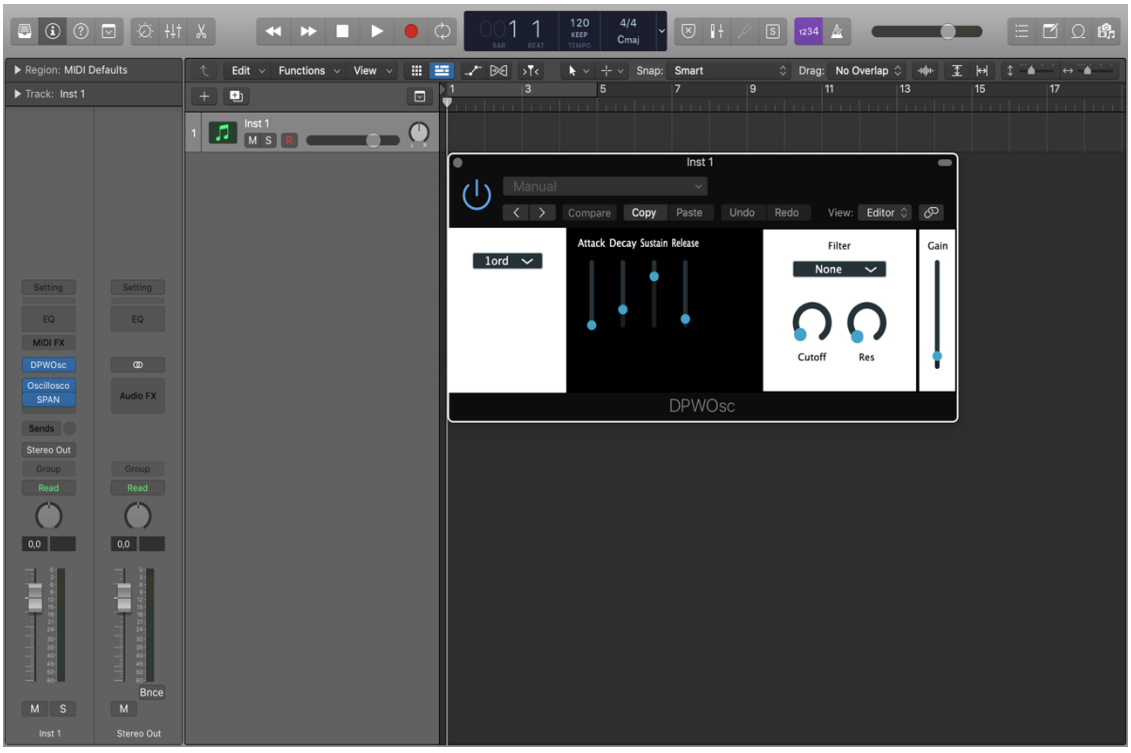


Figura 12. Logic Pro X è una DAW per MacOS

Punto nodale della trattazione sono gli strumenti virtuali in plug-in audio con un ingresso MIDI e uscite audio (in genere uno o due, ma c'è la possibilità di averne di più). Gli strumenti emettono audio in base ai messaggi ricevuti attraverso la porta di ingresso MIDI, ai parametri esterni (impostati dall'utente e gestiti dalla DAW) e allo stato interno.

Nella maggior parte dei set-up l'uscita MIDI del controller della tastiera o l'uscita MIDI del sequencer è collegata all'ingresso MIDI dello strumento con uscita audio collegata a una traccia di registrazione della DAW.

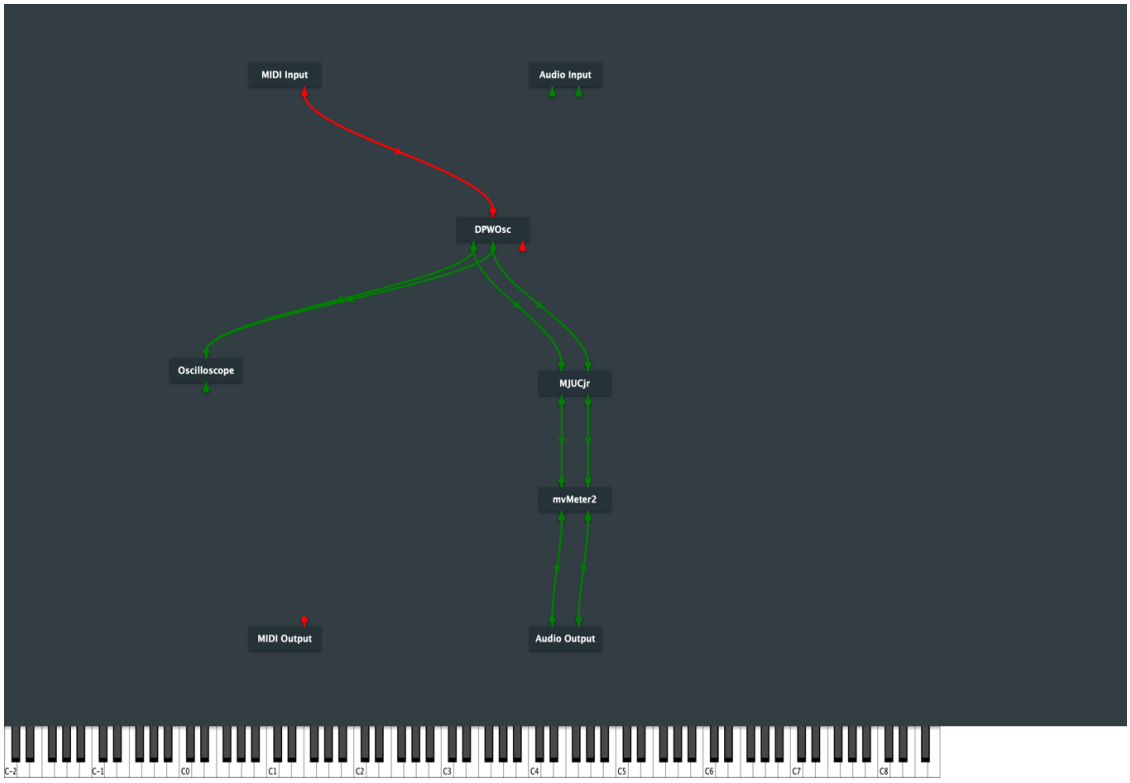


Figura 13. Esempio di catena di plugin attraverso l'host di JUCE. In rosso le connessioni MIDI, in verde le connessioni audio tra i vari plugin.

La creazione di plug-in audio richiede capacità di programmazione e viene effettuata su diversi livelli di astrazione: attraverso blocchi di connessione di unità preparate o attraverso linee di scrittura di codice.

L'integrazione del codice scritto autonomamente in un plug-in completo può essere un processo lungo e faticoso, a supporto di ciò sono stati sviluppati framework software per aiutare i programmatori in questo processo.

JUCE è un framework costruito in C++ per lo sviluppo audio multipiattaforma. Supporta lo sviluppo su PC, Mac, Linux, Android e IOS per i formati plug-in tra cui AU, VST e AAX ed è destinato a facilitare il processo di creazione di un'applicazione, in particolare quella che coinvolge l'elaborazione del suono. JUCE consente di creare plug-in audio in vari formati tra cui VST, VST3, AU e AAX fornendo il programma Projucer in grado di generare i file di progetto per piattaforme o IDE particolari, come Microsoft Visual Studio, e classi multiple per la progettazione e l'elaborazione del suono e dell'interfaccia grafica (GUI). È possibile generare formati plug-in concreti, tutti utilizzando lo stesso codice di base (Shared_Code). Si richiede al programmatore di scrivere gli interni del plug-in audio, e non le connessioni API esterne, consentendo uno sviluppo più veloce e affidabile.

Scrivere un plug-in audio con JUCE vuol dire fornire l'implementazione di tre metodi (Funzionalità della classe *AudioProcessor*) responsabili della preparazione, elaborazione e pulizia delle risorse. Non implementando il metodo di elaborazione, si avrà l'effetto di un plugin bypassatto.

Il metodo *ProcessBlock()* è chiamato ad una velocità che permette alla scheda audio di riprodurre i campioni(*Samples*) elaborati ad una frequenza di campionamento (*SampleRate*), dipendente dal set-up della DAW. Il programmatore deve riempire l'*Audiobuffer* con campioni da riprodurre e può utilizzare il *Midibuffer*, fornito per questo scopo, che contiene tutti i messaggi MIDI, con le date esatte (*TimeStamps*), che informeranno il verificarsi nell'*Audiobuffer*. Quest'ultimo contiene un array di campioni (uno per ogni canale) di una lunghezza specifica (sempre dipendente dalla DAW). Quando viene rilasciato il metodo *processBlock()*, i campioni generati sono accodati in modo tale che la scheda audio li riproduca. Se l'intero processo impiega meno di 10 ms circa, la latenza dovrebbe essere percettibile per l'ascoltatore.

Capitolo 4: Progettazione

Il progetto si compone in due parti di software:

- La libreria *Maximilian:* una libreria per la sintesi audio e l'elaborazione del segnale costruita in C++. A differenza di JUCE, dà all'utente un maggiore controllo sull'applicazione e i suoi moduli DSP sono più semplici da implementare. Ho deciso d'importare questa libreria come uno strumento di terze parti per accelerare lo sviluppo partendo da segmenti di codice già pronti.
- VST3 client plugin: il client della libreria in C++ con un'interfaccia utente grafica, costruita utilizzando il framework JUCE. Il client non fornisce altro che una connessione tra l'utente e l'API della libreria. Il plug-in VST3 consente l'utilizzo in DAW o qualsiasi altro software di hosting. Non esegue alcuna elaborazione del suono da solo.

4.1 Percorso di Sintesi

Il percorso illustrato è uno schema che mostra l'ordine di particolari moduli di sintesi in un sintetizzatore. Il percorso di sintesi di una singola voce in questo progetto è descritto in figura 14.

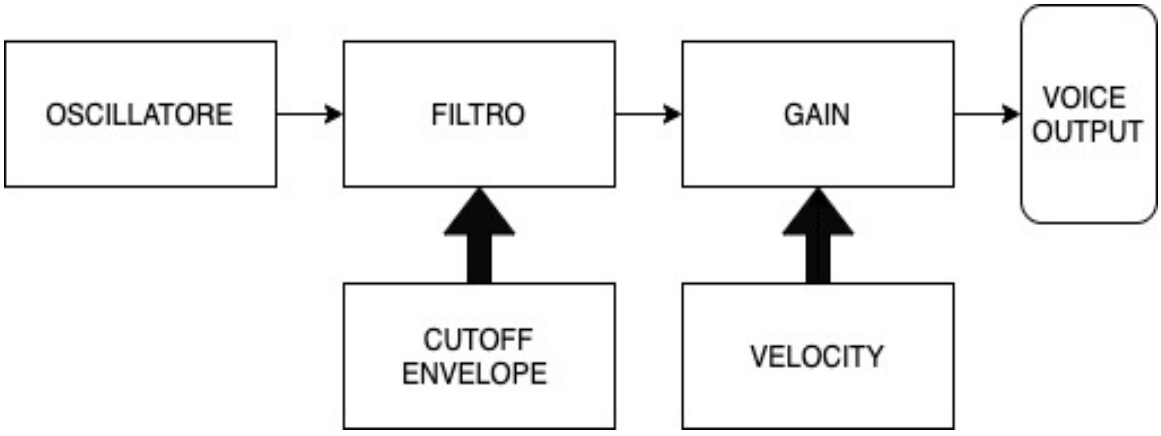


Figura 14. Percorso di sintesi semplificato di una singola voce. Non tutti i parametri di controllo sono rappresentati.

Il segnale viene generato dall'oscillatore triviale, filtrato con la frequenza di *cutoff* seguendo l'inviluppo fornito (di tipo ADSR), successivamente viene applicato il guadagno risultante dalle informazioni di velocità della nota sull'evento. L'uscita della voce è un segnale pronto per essere miscelato con l'uscita di altre voci.

4.2 Architettura

La struttura generale di questo progetto è simile a un tipico progetto JUCE, dove le diverse parti sono divise in componenti gerarchiche.

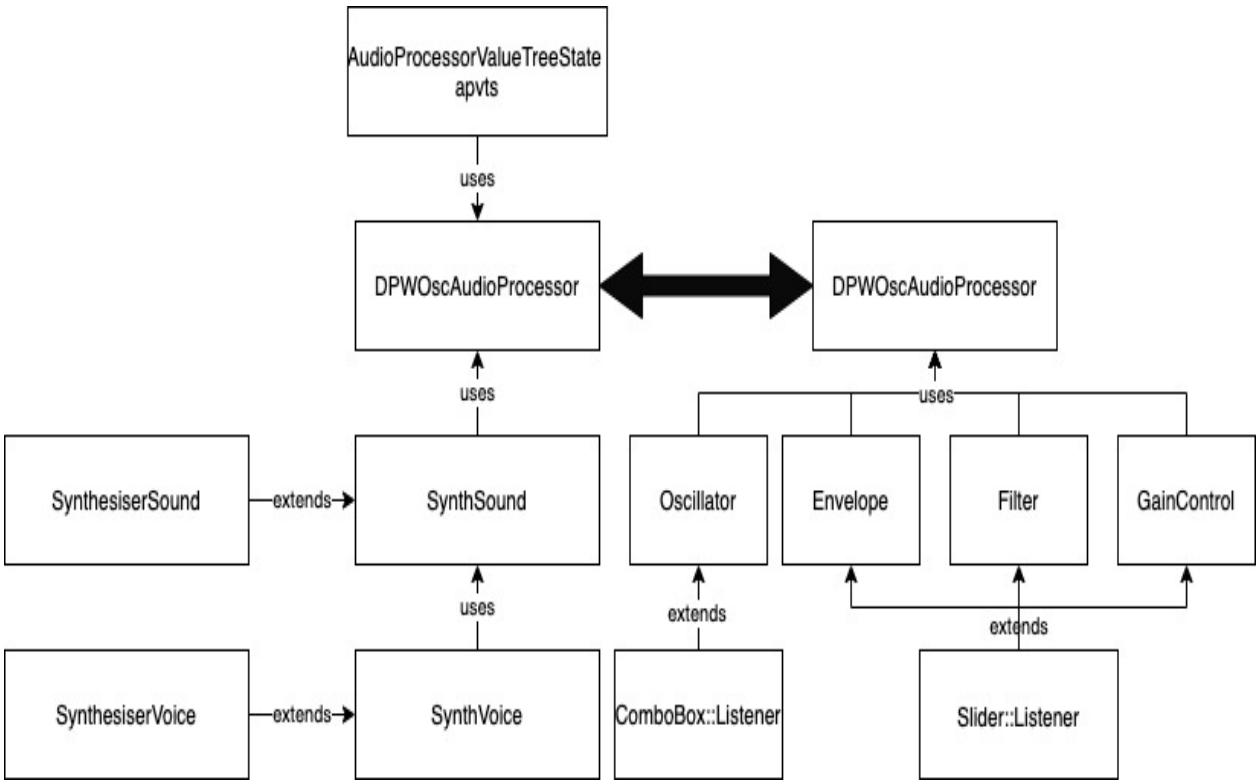


Figura 15. Diagramma della struttura

Come mostrato nella figura 15, i due componenti principali sono *DPWOscAudioProcessor* e *DPWOscAudioProcessorEditor*, che sono responsabili rispettivamente per gli algoritmi DSP e UI.

4.2.1 Componenti DSP

Sull'aspetto DSP, la classe *SynthVoice* si estende dalla classe *Synthesiservoice* integrata di JUCE, dove il genitore si occupa del campionamento, dell'input e dell'output del suono. Include inoltre la classe *SynthSound* che si estende dalla classe *SynthesiserSound*. Una classe passiva che semplicemente descrive ciò che il suono è, quali note midi e canali possono attivarlo.

Al fine di rendere questo progetto utilizzabile come applicazione Standalone e plugin utilizzabile in DAW, ho deciso di convertire il messaggio midi in frequenza d'onda per produrre un output.

SynthesiserVoice segue un protocollo guidato da eventi, il che significa che mantiene un ciclo di messaggi su un thread separato per rilevare eventi diversi dai dispositivi di input come "note on" e "note off". Quando viene rilevato un evento, viene attivata una funzione di callback per eseguire determinate operazioni. Successivamente *RenderNextBlock()* viene chiamato dal componente DSP principale per l'output del blocco. Dal momento che il componente DSP funziona su thread audio dove il campionamento avviene ad un tasso molto elevato, è fondamentale escludere qualsiasi messaggio di input/output standard in quanto potrebbero rallentare il sistema e causare un ritardo.

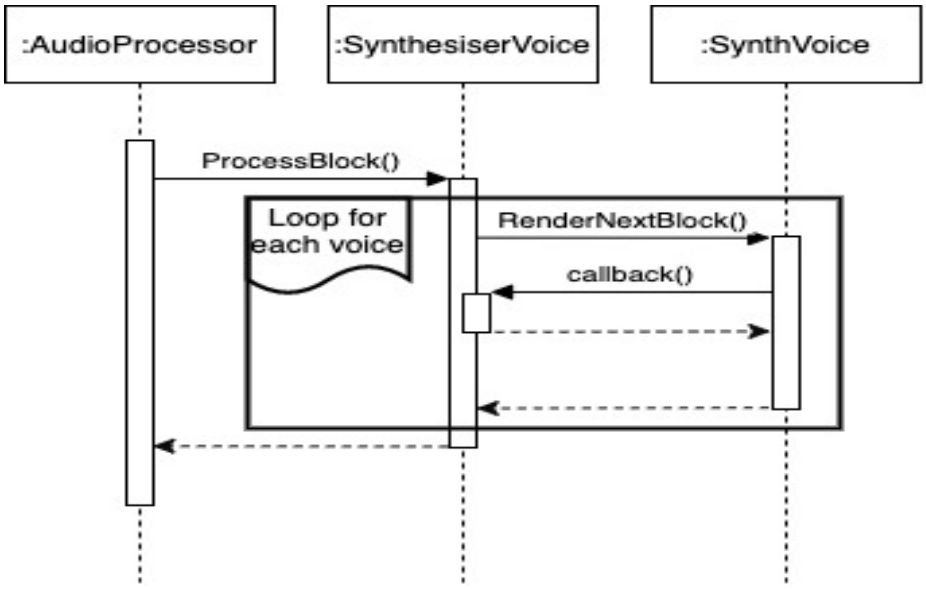


Figura 16. Diagramma delle sequenze del rendering audio

4.2.2 Componenti UI

Sul lato UI, ho eseguito il modo convenzionale per creare un'applicazione GUI in JUCE con *DPWOscAudio ProcessorEditor* come componente principale dell'interfaccia utente.

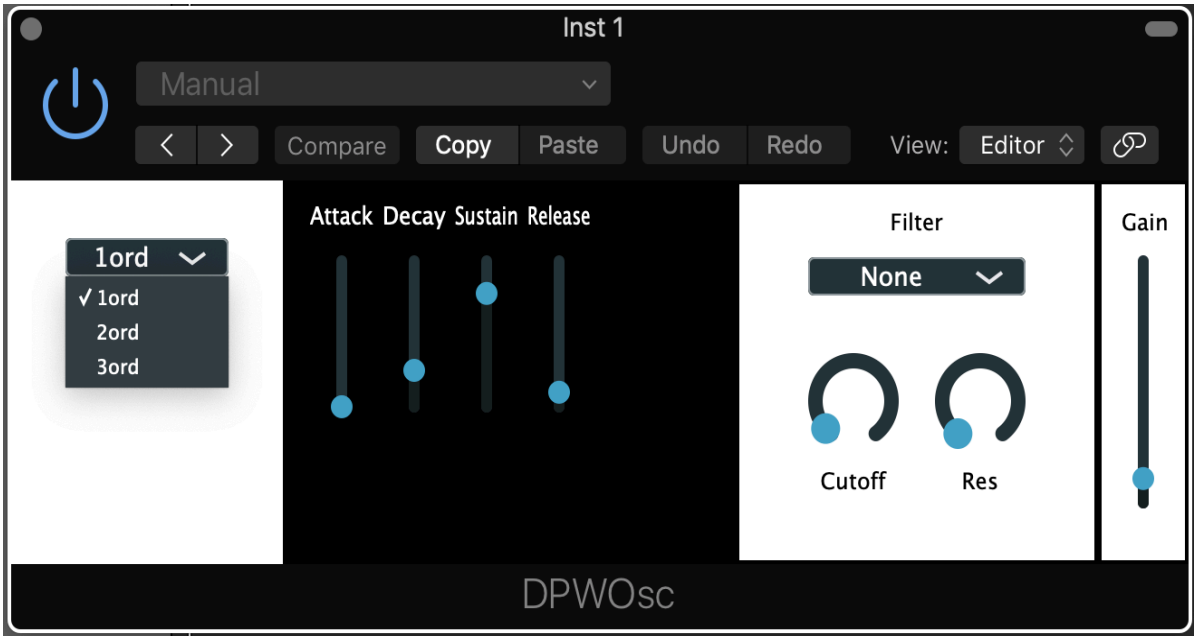


Figura 17. Interfaccia Plugin. Da sinistra verso destra troviamo la selezione dell'ordine della forma d'onda, il blocco relativo all'ADSR, selezione tipo di filto e knob relativi al Cutoff e Risonanza, e infine il Gain.

I suoi component figli sono *Oscillator*, *Envelope*, *Filter* e *GainControl*. Il componente *Oscillator* è principalmente responsabile della visualizzazione delle opzioni della forma d'onda in una casella combinata, che consente all'utente di selezionare i diversi gradi di un'onda dente di sega.

La componente *Envelop*e aiuta l'utente a modificare il comportamento dell'onda. In questo caso specifico, ho costruito un sintetizzatore ADSR, che ha come parametri "*attack*", "*decay*", "*sustain*" e "*release*". E l'utente è in grado di modificare i parametri attraverso i cursori.

La componente *Filter* consente la scelta di un filtro HiPass o LowPass, controllati dai relativi parametri "CutOff" e "Res" che si occupano rispettivamente della frequenza da dare al filtro, e della risonanza.

Infine la componente *GainControl* si occupa del volume di uscita attraverso un cursore.

4.2.3 Comunicazione tra le due componenti

La mappatura del controllo nell'interfaccia utente al componente del processore è la parte più critica di questo progetto. Ricordando i *Listeners* riguardanti *Oscillator*, *Envelope*, *Filter e GainControl*, che monitorano il cambiamento di un certo parametro dal processore abbiamo più classi indipendenti e avremmo bisogno di 9 parametri diversi tra componenti diversi. Potremmo impostare un riferimento per ciascuno, ma questo approccio diventerebbe facilmente disordinato quando ci sono più *thread* e un singolo conflitto potrebbe portarci in un'area di comportamento indefinita. Pertanto, ho adottato la classe *AudioProcessorValueTreeState*, che racchiude i parametri e li memorizza in un oggetto (*apvts*). Questo oggetto è memorizzato all'interno del processore *DPWOscAudioProcessor*, che è una classe singleton (cioè un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza). In questo modo, possiamo accedere ai parametri da qualsiasi classe affinché abbiamo accesso alla classe *DPWOscAudioProcessor*.

4.3 Generazione Segnale

Per la generazione del dente di sega sono partito da un metodo già presente nella libreria di terze parti Maximilian come in figura 18.

```
324 double maxiOsc::saw(double frequency) {
325     //Sawtooth generator. This is like a phasor but goes between -1 and 1
326     phase += (1./(maxiSettings::sampleRate/(frequency)));
327     if ( phase >= 1.0 ) phase -=2.0;
328     output=phase;
329     return(output);
330 
331     }
```

Figura 18. Metodo per la generazione del dente di sega

In ingresso troviamo la frequenza da applicare al segnale e attraverso l'uso dell'istruzione *if* lo si fa variare nell'intervallo [-1,1]. In Output si avrà un segnale del tipo dente di sega che varia in quell'intervallo come mostrato nella Figura 19.

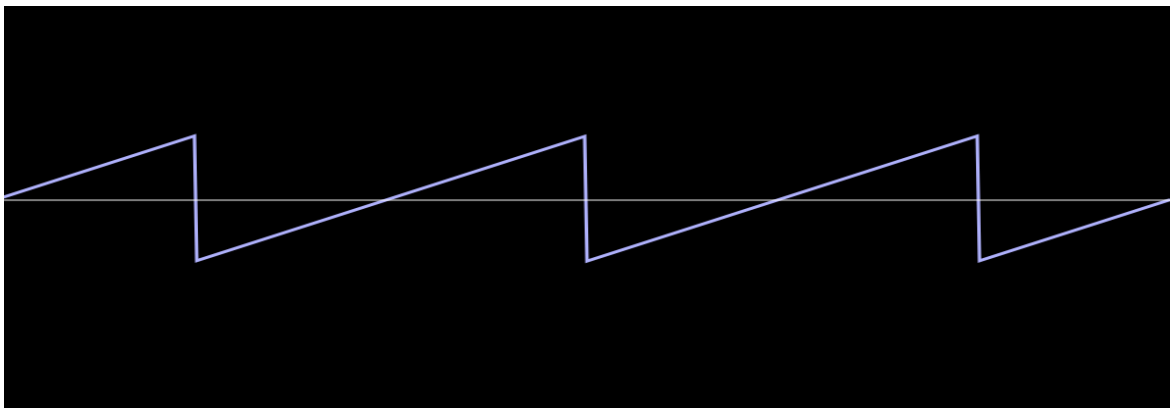


Figura 19. Forma d'onda generata dal metodo saw().

Successivamente vi saranno due metodi che implementano un algoritmo DPW per la generazione del dente di sega ma con una riduzione di *aliasing*, e sono: *saw2ord()* e *saw3ord()* che implementano rispettivamente l'algoritmo di secondo e terzo ordine.

```
333  double maxiOsc::saw2ord(double frequency){
334
335
336      double triv = saw(frequency);
337      double poly = triv*triv;
338      double gain1 = ((maxiSettings::sampleRate/frequency)/4);
339
340      output = (poly - poly_prec);
341      poly_prec = poly;
342
343      return (output*gain1);
344
345  }
```

Figura 20. Implementazione dell'algoritmo DPW di secondo ordine.

Con la variabile *triv* si indica il segnale generato attraverso al funzione *saw(),* con *poly,* il polinomio della funzione parabolica, e con *gain1* si indica il fattore di scala rispetto all'ordine della funzione (vedi Tabella 1).

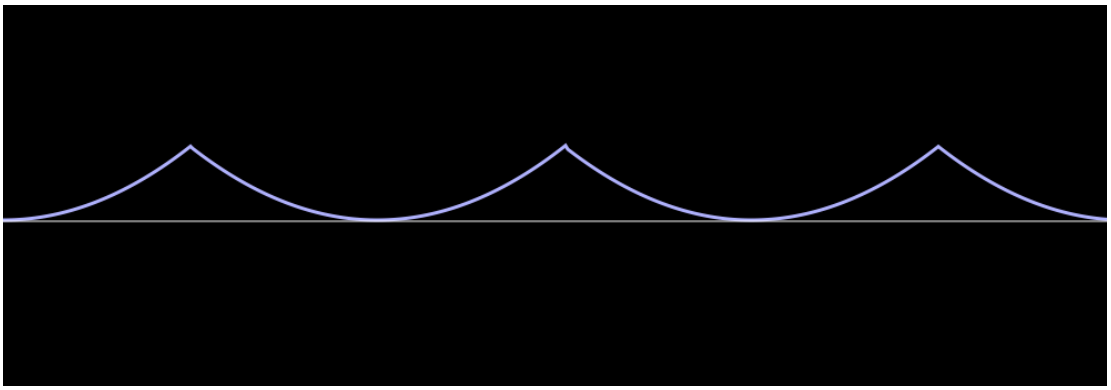


Figura 21. Forma d'onda della funzione parabolica generata tramite la quadratura del segnale.

Come già spiegato nel capitolo DPW, per ottenere il segnale desiderato partendo dalla funzione parabolica di secondo ordine, bisogna differenziare tale funzione che in questo caso si fa sottraendo ad essa una variabile, chiamata *poly_prec*, inizializzata a zero, che indica il polinomio all'istante precedente. La risultante di tutto ciò si moltiplica per il fattore di scala perché, come già visto, la differenziazione ha il problema di produrre una riduzione di ampiezza della forma d'onda.

Questo metodo ci restituisce un segnale simile ad un dente di sega ma con un *aliasing* ridotto rispetto a quello generato dal metodo *saw()*.

```
347 double maxiOsc::saw3ord(double frequency){
348     double triv = saw(frequency);
349     double poly = ((triv*triv*triv)-triv);
350     double gain1 = (((maxiSettings::sampleRate/frequency)*(maxiSettings::sampleRate/frequency))/24);
351     double saw1 = (poly-poly_prec2);
352     poly_prec2 = poly;
353 
354     output = ((saw1 - poly_prec)*gain1);
355     poly_prec = saw1;
356 
357     return (output);
358 }
```

Figura 22. Implementazione dell'algoritmo di terzo ordine.

Il metodo presente in figura 22 è simile al metodo precedente, ma con alcune varianti: il valore della variabile *poly* (vedi tabella 2) che essendo una funzione parabolica di terzo ordine, necessita di essere differenziata due volte al fine di arrivare ad avere il segnale desiderato.

4.4 Risultati

In questa sezione saranno valutati la qualità audio dei segnali prodotti utilizzando i metodi DPW proposti e come viene modulato il segnale attraverso l'ADSR.

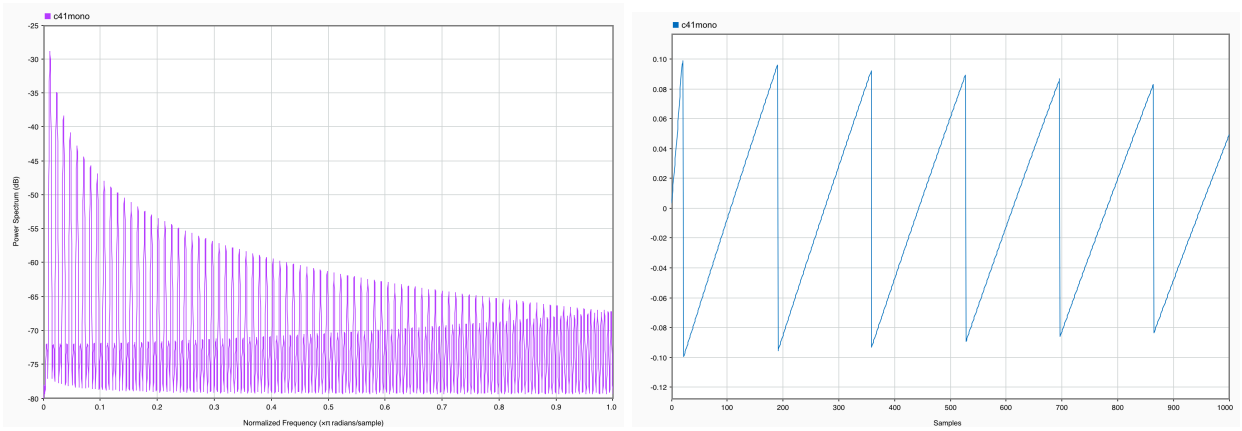
Calcolando lo spettro FFT di ogni tono da un segmento di segnale lungo 1s, per le seguenti note C4(262 Hz), C7(2093 Hz) e C8(4186 Hz), è interessante notare come migliora la qualità quando si aumenta l'ordine polinomiale. Per ottenere ciò facciamo riferimento all'SNR (signal-to-noise ratio) cioè il rapporto tra l'ampiezza del segnale utile e ampiezza del rumore di fondo in un certo istante.

Le seguenti figure mostrano esempi di spettri e le relative forme d'onda di segnali a dente di sega prodotto utilizzando il metodo triviale e il metodo DPW degli ordini 2 e 3.

Dalla Figura 23 si può notare come l'aliasing presente nel metodo triviale (a) sia molto ostile, causando così un segnale distorto. Utilizzando però il metodo DPW (b) e (c) si può osservare come l'aliasing viene notevolmente ridotto. Più si aumenta il grado e più si avrà una riduzione significativa. Avendo aliasing ridotto rispetto al metodo triviale, si avrà anche un valore di SNR maggiore, perché essendo il rapporto tra il segnale utile e il rumore di fondo (in questo caso è l'aliasing), maggiore è il rapporto e maggiore sarà l'energia del segnale rispetto a quella di tutto ciò che è considerato rumore.

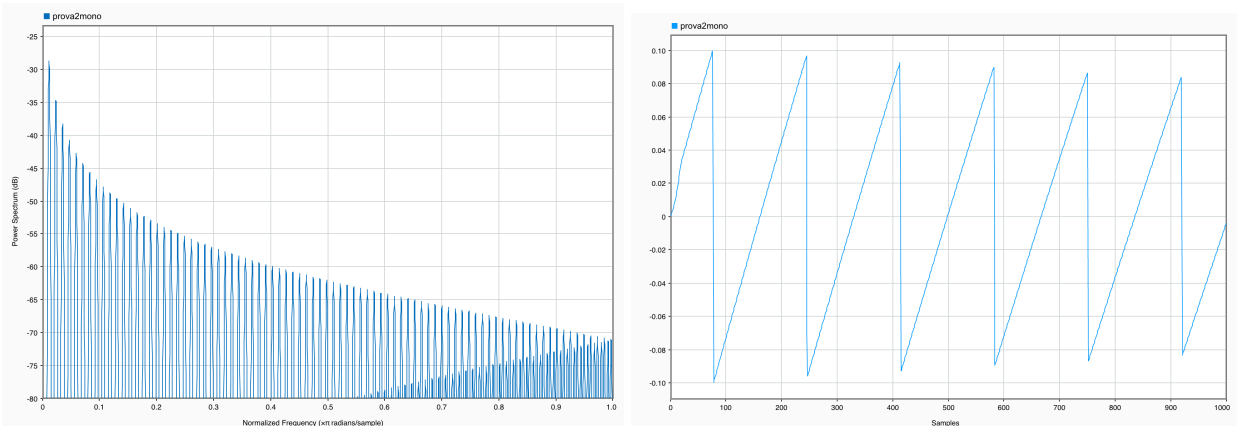
Come evidenziato nel capitolo 2.2, il segnale generato dalla differenziazione del metodo DPW non è un vero e proprio segnale a dente di sega, ma è un'approssimazione di esso. Lo si può notare molto bene nelle forme d'onda della Figura 25 (b) (c) dove uno o più campioni vicini al legame di due periodi adiacenti sono differenti da quello generato dal metodo triviale ed è proprio questa proprietà nel dominio del tempo del DPW che riduce l'aliasing.

(a)



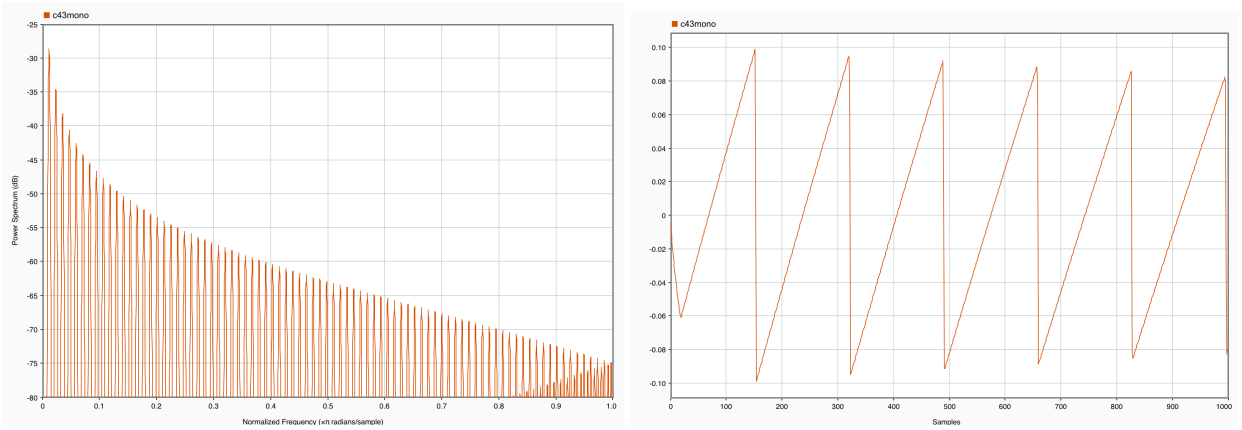
SNR =19,26 dB

(b)



SNR = 29,47 dB

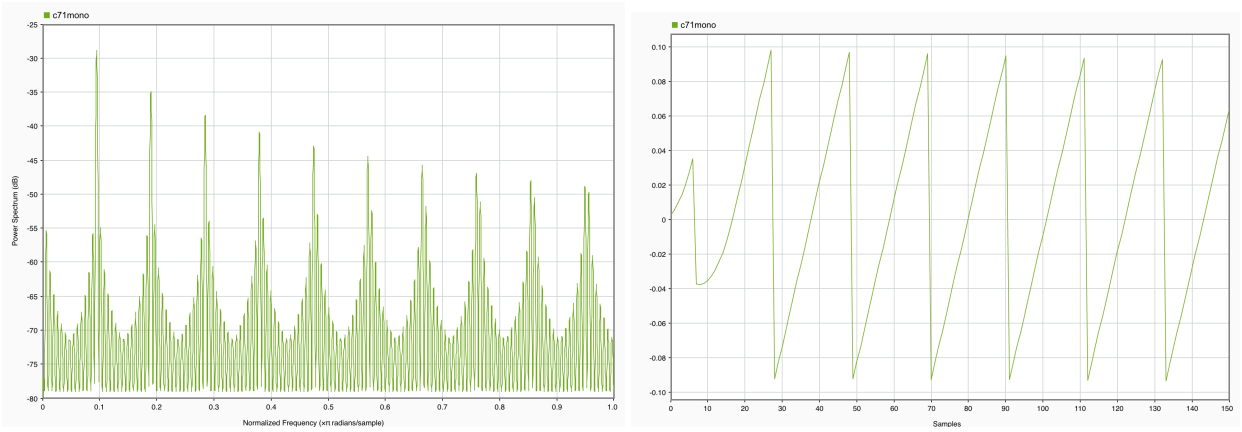
(c)



SNR = 35,44 dB

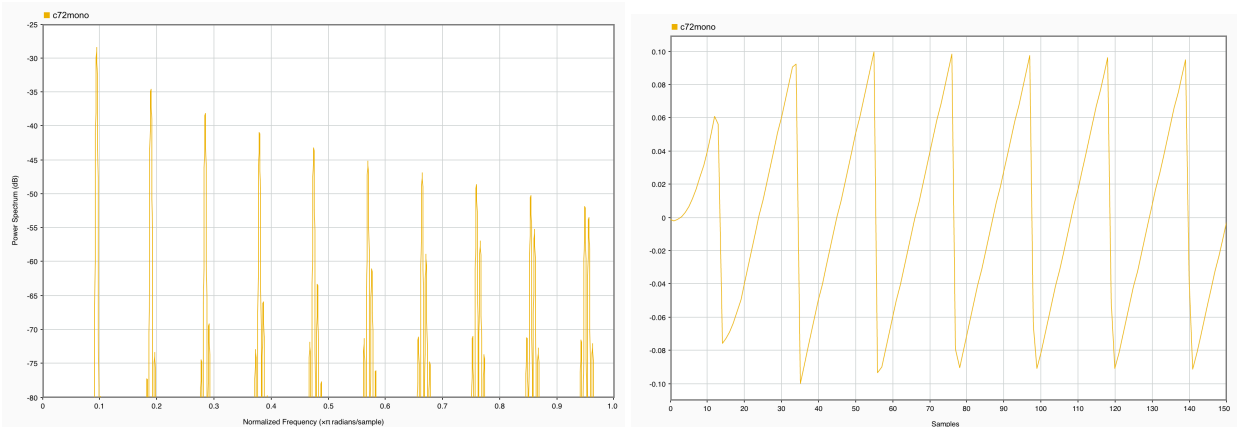
Figura 23.Forme d'onda e spettri relativi ai diversi ordini e il loro relativo SNR a una frequenza C4 (262Hz) con una frequenza di campionamento pari a 44100 Hz. (a) primo ordine, (b) secondo ordine, (c) terzo ordine

(a)



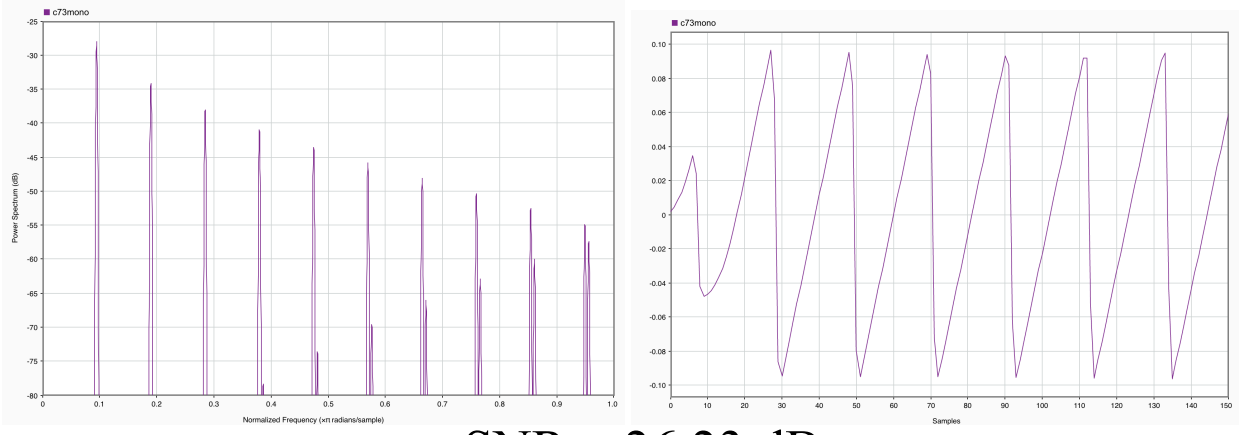
SNR = 10,22 dB

(b)



SNR = 20,33 dB

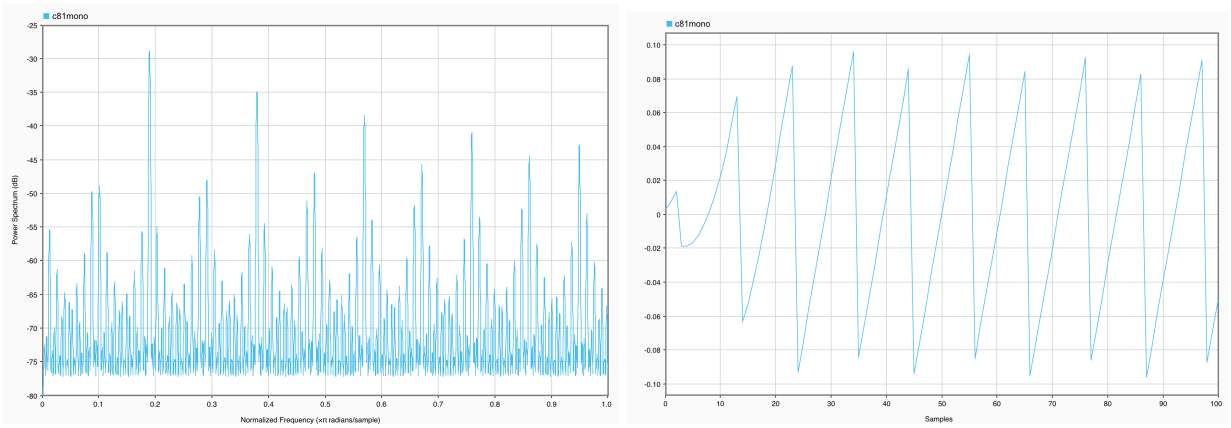
(c)



SNR = 26,23 dB

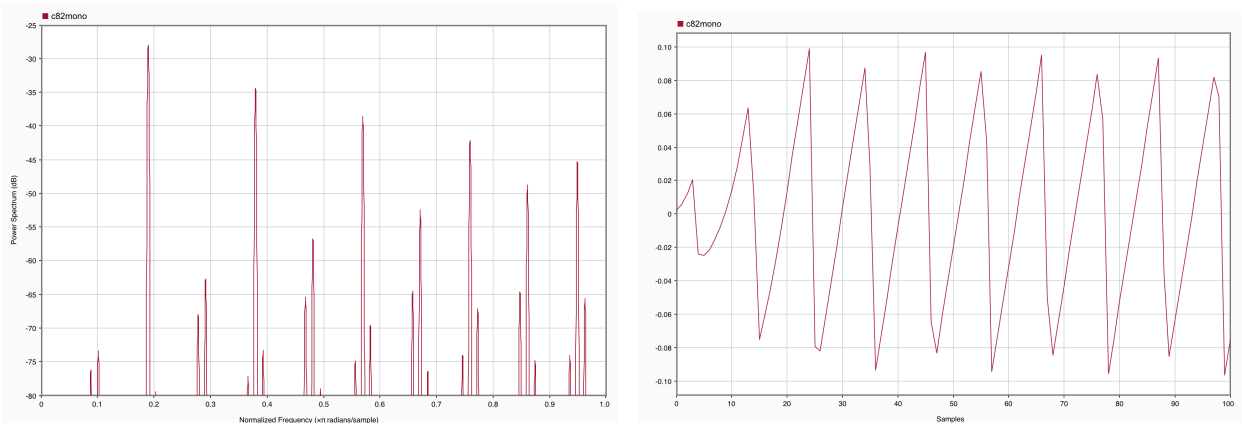
Figura 24. Forme d'onda e spettri relativi ai diversi ordini e il loro relativo SNR a una frequenza C7 (2093Hz) con una frequenza di campionamento pari a 44100 Hz. (a) primo ordine, (b) secondo ordine, (c) terzo ordine

(a)



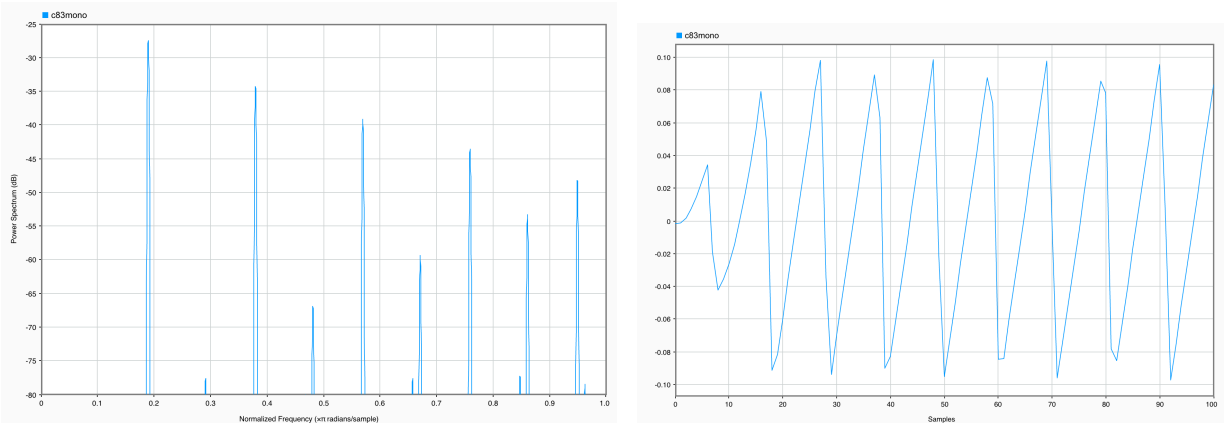
SNR = 7,41 dB

(b)



SNR = 18,18 dB

(c)



SNR = 24,64 dB

Figura 25. Forme d'onda e spettri relativi ai diversi ordini e il loro relativo SNR a una frequenza C8 (4186Hz) con una frequenza di campionamento pari a 44100 Hz. (a) primo ordine, (b) secondo ordine, (c) terzo ordine

Nelle prossime figure sono rappresentati alcuni esempi di inviluppi del segnale attraverso l'ADSR.

In figura 17 è rappresentato il plugin con i suoi parametri di partenza, e la sua forma d'onda è rappresentata in figura 26. Come visto nel capitolo 1.4.1, per creare un inviluppo si fa riferimento ai parametri attack, decay, sustain e release. Aumentando il parametro attack andiamo a diminuire il tempo che il suono impiega a raggiungere il suo volume massimo (figura 27). In figura 28 è rappresentato lo spettro di un inviluppo in cui il parametro decay è stato aumentato. Infatti è possibile notare come il suono dal suo picco di massimo diminuisce più lentamente rispetto al segnale con parametri di default in figura 26.

Diminuendo il sustain invece diminuisce il volume che si ha dopo la fase di attack finché la nota rimane ON (figura 29). Infine in figura 30 si può notare come la coda sia più lunga rispetto al segnale di default, questo perché aumentando il tempo di release aumentiamo il tempo in cui il suono svanisce dopo aver lasciato il tasto (note off).

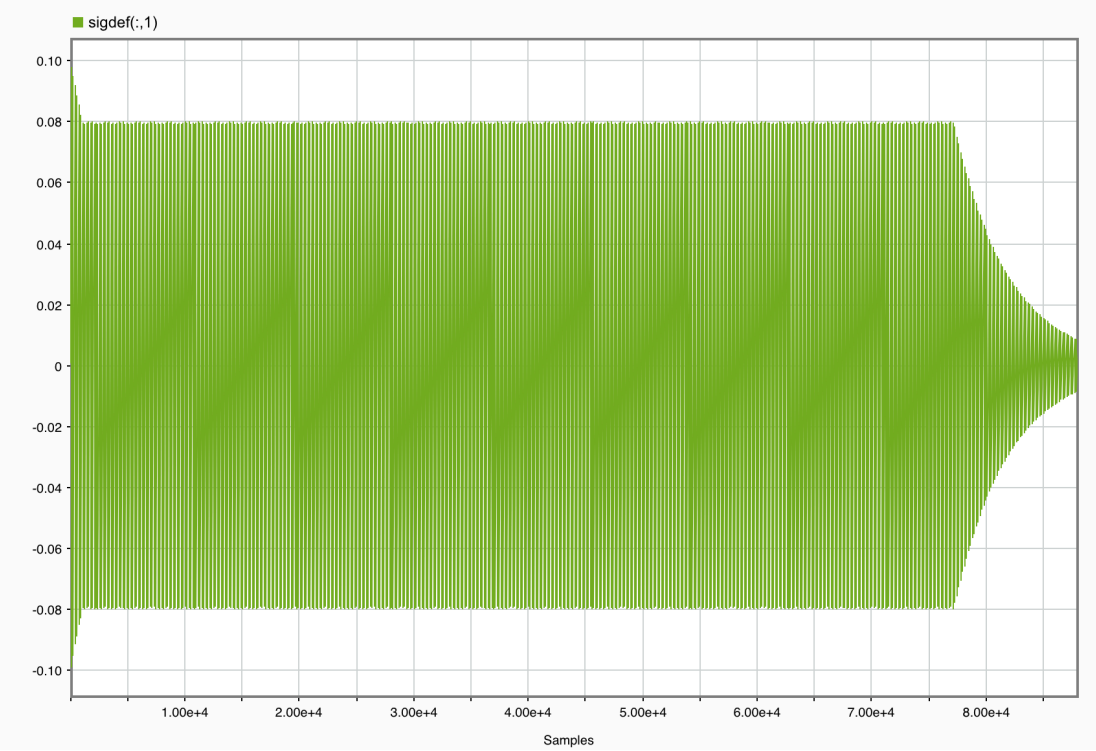


Figura 26. Forma d'onda del segnale con parametri ADSR di default.

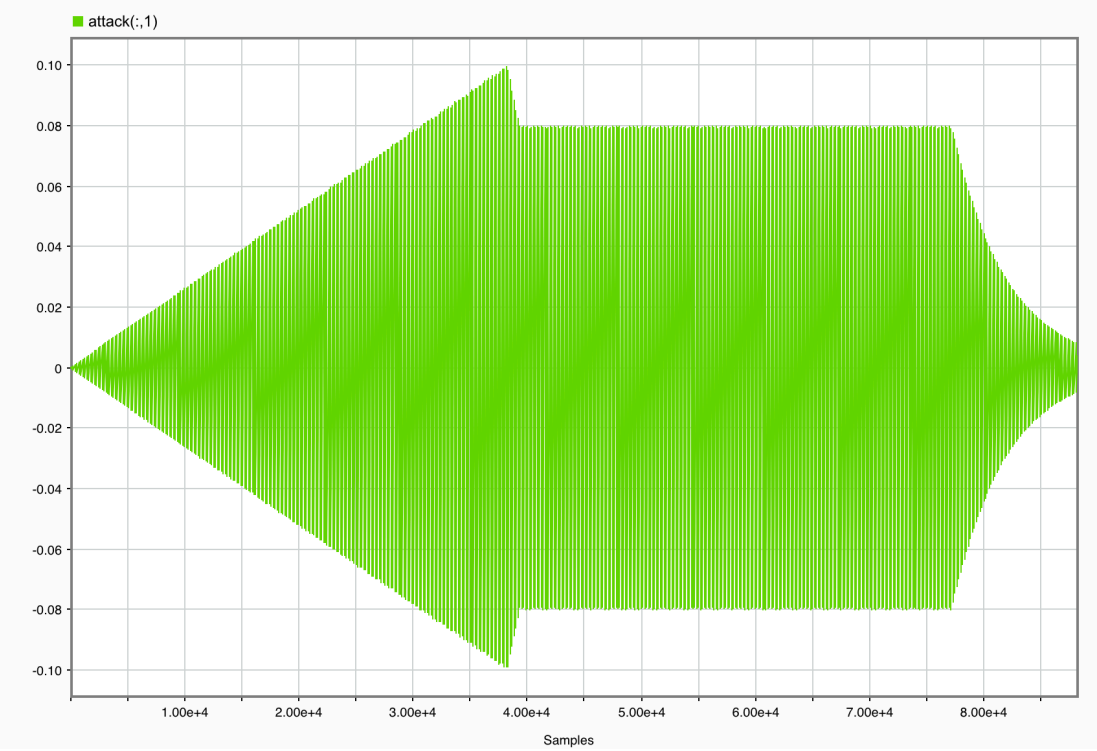


Figura 27. Forma d'onda di un segnale con parametro di attack aumentato

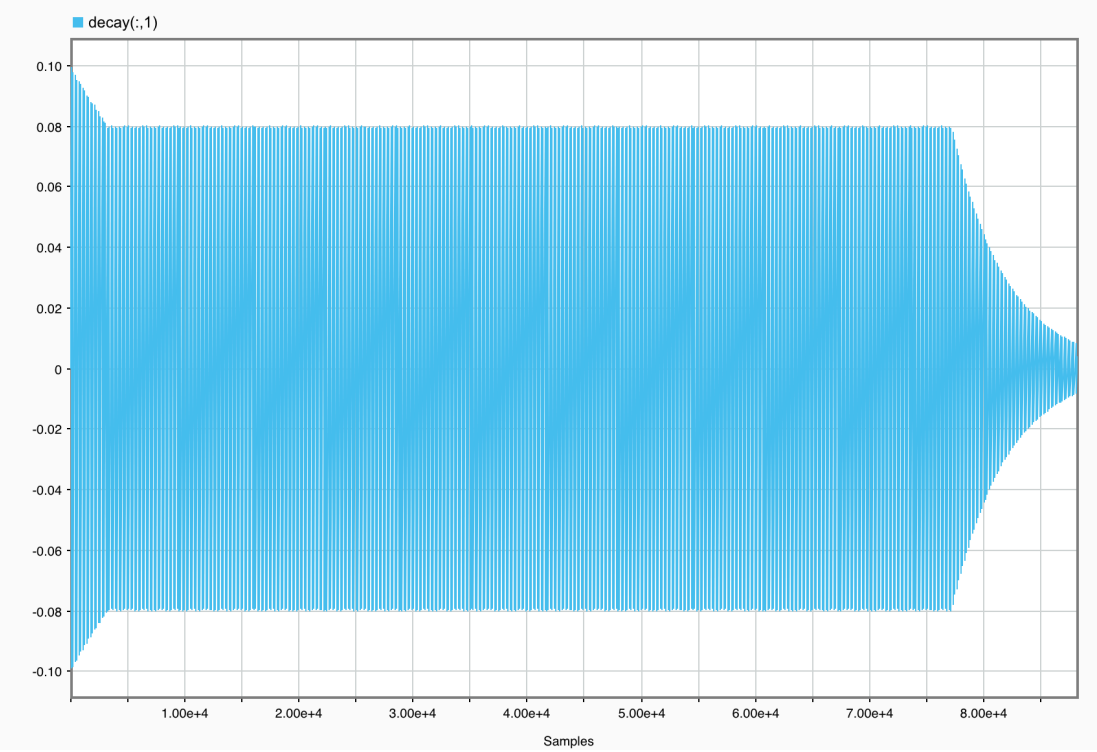


Figura 28. Forma d'onda di un segnale con parametro di decay aumentato.

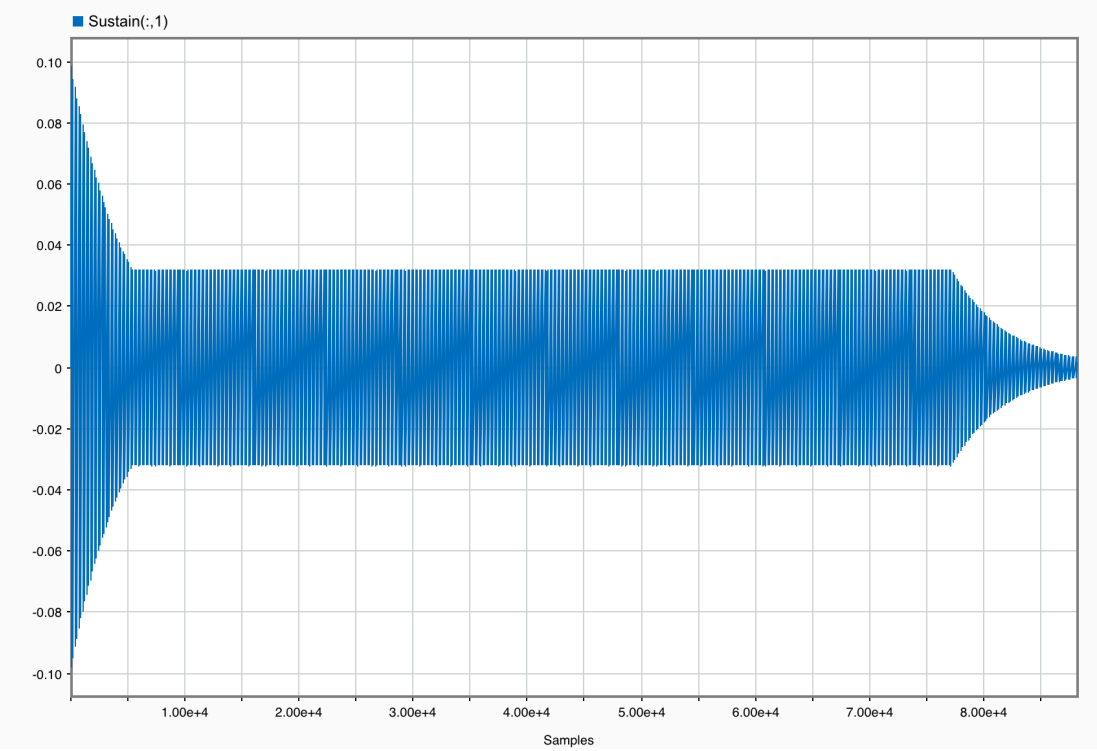


Figura 29. Forma d'onda di un segnale con paramentro di sustain diminuito.

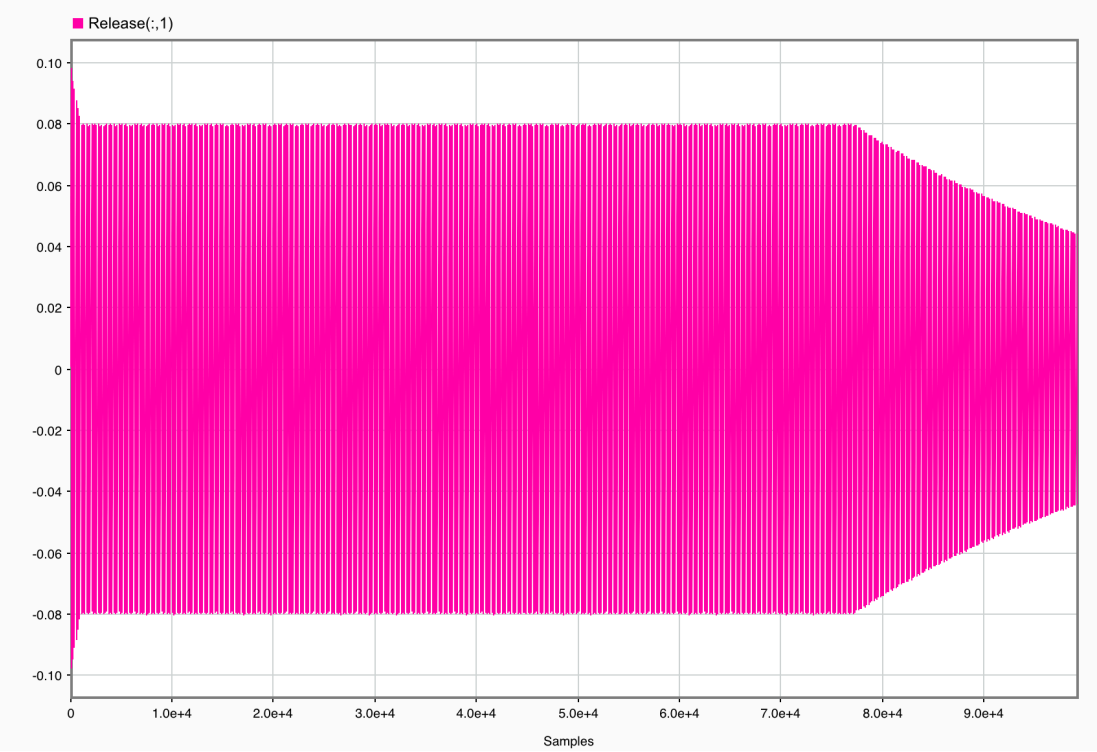


Figura 30. Forma d'onda di un segnale con parametro di release aumentato

Conclusioni

La presente tesi ha come obiettivo lo studio del framework Juce per la progettazione di un Plugin DSP in ambito musicale e l'implementazione di un algoritmo DPW sotto tale framework. Il contesto teorico è descritto nei primi due capitoli. Il framework Juce è descritto nel capitolo 3. La descrizione di quanto è stato fatto e il relativo risultato è descritto nel capitolo 4.

La conclusione del mio progetto è che il plugin è in grado di produrre un segnale a dente di sega e modularlo attraverso la componente ADSR e l'applicazione di un filtro. Nucleo centrale è il pluguin che è in grado di produrre un segnale con una riduzione di aliasing attraverso l'implementazione del metodo DPW, i cui algoritmi e strutture utilizzate si sono dimostrati efficaci ed efficienti.

Bibliografia e Sitografia

[1] Leonardo Gabrielli, *Developing virtual synthesizers with VCVRack*, Routledge, 2020

[2] https://digilander.libero.it/desdeus/Trasmissione/Campionamento.htm

[3] http://www.sintetizzatore.com/index.php/oscillatori

[4] https://www.esperimentanda.com/come-usare-la-tecnologia-midi-nella-musica-messaggi-eventi-interfacce-cavi-midi-software-hardware/

[5] Martin Robinson, *Getting started with JUCE,* Packt Publishing, 2013

[6] https://www.youtube.com/channel/UCpKb02FsH4WH4X_2xhIoJ1A

[7] Vesa Välimäki, Juhan Nam, Julius O. Smith and Jonathan Abel, *"Alias-suppressed oscillators based on differentiated polynomial waveforms."* IEEE Transactions on Audio, Speech and Language Processing 18.4 (2010a): 786-798