UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea Triennale in Ingegneria Informatica e
dell'Automazione

Progetto di un sintetizzatore virtuale in C++ su piattaforma JUCE

Development of a virtual synthesizer in C++ on the JUCE framework

Candidato:
Leonardo Mannini

Relatore:
Prof. Leonardo Gabrielli

Anno Accademico 2022/2023

Indice

Sommario

In questa tesi verrà presentata la progettazione e la realizzazione di un sintetizzatore virtuale chiamato *leoSynth*, realizzato con JUCE, un framework basato sul linguaggio C++ che fornisce metodi e librerie per l'elaborazione dei segnali audio digitali e la realizzazione della relativa interfaccia utente (GUI - Graphic User Interface).

Nel Capitolo 1 saranno spiegati i concetti fondamentali alla base di un plugin audio digitale, le generalità sulle architetture classiche di sintesi e le tecniche di sintesi. Particolare attenzione sarà dedicata alla sintesi sottrattiva, con l'elenco delle sue componenti principali e l'illustrazione di esempi storici notevoli. In particolare, saranno approfonditi gli oscillatori analogici e digitali, analizzando il problema dell'aliasing e le relative soluzioni, nonché i filtri analogici e digitali, con una breve descrizione delle architetture fondamentali.

Nel Capitolo 2 saranno illustrate le tecnologie utilizzate per realizzare il progetto, nonché le motivazioni alla base delle scelte architetturali e dei linguaggi di programmazione adottati. Sarà quindi presentato il design desiderato del sintetizzatore *leoSynth*, confrontandolo con alcuni esempi di sintetizzatori analogici e digitali.

Nel Capitolo 3 sarà presentato il prodotto finito, mostrando l'interfaccia e spiegando l'implementazione dei componenti utilizzati all'interno del framework. Sarà inoltre presentata la struttura del progetto, spiegato il workflow di sviluppo e forniti esempi di codice.

Nel Capitolo conclusivo saranno inclusi collegamenti a test ed esempi audio, riferimenti per reperire il codice sorgente e una release funzionante del sintetizzatore *leoSynth*. Saranno analizzate le prestazioni e saranno esplorate le possibili direzioni future del progetto, tenendo conto sia delle necessità degli utenti, sia delle eventuali limitazioni prestazionali.

Capitolo 1

Sintetizzatore

Un sintetizzatore è uno strumento musicale elettronico che genera segnali audio, sfruttando diverse tecniche di sintesi (additiva, sottrattiva, FM,...) per "costruire" diverse forme d'onda. Questi suoni vengono poi modificati da altri componenti come filtri, effetti, LFO ed altri, per ottenere timbri particolari. I sintetizzatori moderni sono solitamente controllati attraverso tastiere o *sequencer* che comunicano tramite il protocollo MIDI.

1.1 Tipologie di sintesi del suono

Con sintesi del suono si intende la generazione di segnali audio analogici o digitali.

La sintesi del suono è utilizzata in diversi ambiti: da quello musicale, all'automazione di sistemi e processi (esempio: *text-to-speech*). Il plugin audio progettato discusso in questa relazione appartiene all'ambito musicale, fornendo di fatto un'interfaccia software per per costruire il timbro di uno strumento musicale a partire dalle forme d'onda. Tale strumento può essere controllato esternamente tramite una tastiera, un sequencer esterni, o altri dispositivi.

Esistono molte tecniche di sintesi sviluppate per la realizzazione di sintetizzatori, di seguito sono elencate le principali.

1.1.1 Sintesi additiva

La sintesi additiva è una tecnica di sintesi largamente utilizzata in ambito accademico [1], e si basa sulla somma di segnali elementari (in genere sinusoidali) per l'approssimazione di qualsiasi segnale più complesso, secondo la teoria delle serie di Fourier. In breve, una qualunque funzione periodica può essere scomposta in una somma di infinite "opportune" funzioni o componenti sinusoidali (seno e coseno) che sono dette armoniche quando il rapporto delle loro frequenze rispetto ad una frequenza (detta fondamentale) è intero, oppure inarmoniche negli altri casi. Una forma d'onda complessa $f(t)$ può essere generata aggiungendo un insieme di semplici segnali, tipicamente sinusoidi, con intensità, frequenze, fasi indipendenti fra loro:

$$f(t) = \sum_{k=1}^{K} A_k sin(\omega_k t + \phi_k)$$

dove le costanti A_k, ω_k e ϕ_k sono l'intensità, la frequenza angolare e la fase iniziale della k-esima componente sinusoidale. In generale, ogni sinusoide potrebbe avere intensità, frequenze e fasi variabili nel tempo, ovvero:

$$f(t) = \sum_{k=1}^{K} A_k(t) sin(\theta_k(t))$$

dove

$$\theta_k(t) = \phi_k(t) + \int_0^t \omega_k(\tau) d\tau$$

La frequenza istantanea della componente k-esima al tempo t può essere definita come la derivata nel tempo della funzione $\theta(t)$, ovvero

$$\frac{d}{dt}(\theta_k(t)) = \frac{d}{dt}(\phi_k(t)) + \omega_k(t)$$

Si noti che, nonostante queste siano equazioni presentate a tempo-continuo, è possibile mostrare come sia possibile ottenerle anche nella forma a tempo discreto.

Dato un numero K di sinusoidi con indipendenti valori di $A(t)$, $\omega(t)$ e $\phi(t)$, è possibile quindi rappresentare il sistema che ha in uscita la forma d'onda $f(t)$ tramite il seguente schema di controllo:

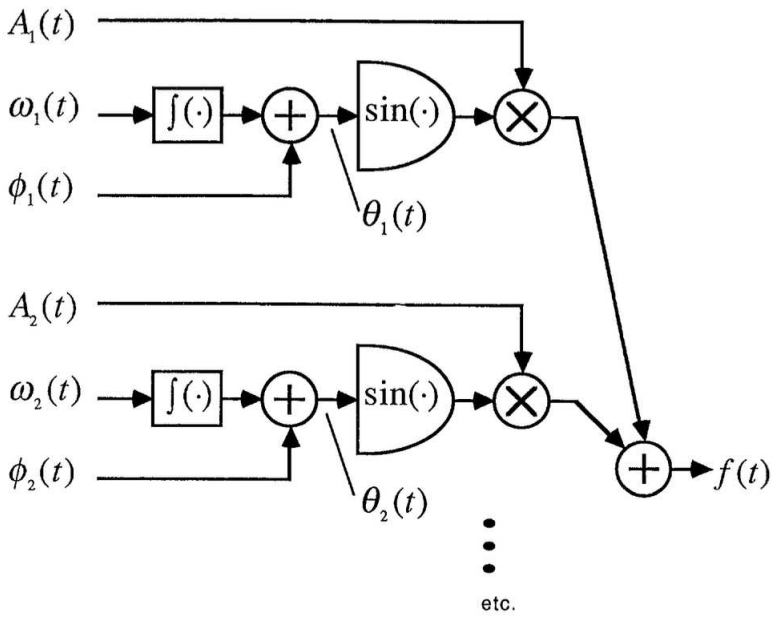

Figura 1.1: Sintesi additiva

Il vantaggio primario di questo approccio è la flessibilità della rappresentazione tramite segnali sinusoidali: la facilità data dalla semplice rappresentazione matematica permette di produrre risultati altamente controllabili e predicibili, ma richiede un

grande numero di oscillatori ed operazioni al secondo per generare un suono elaborato e complesso [2].

Un esempio di strumento che utilizza la sintesi additiva, è il celebre organo Hammond. Questo venne inventato da Laurens Hammond e John M. Hanert nel 1935, ed è uno dei primi esempi di strumenti musicali a sintesi additiva (il primo in assoluto fu il Teleharmonium di Thaddeus Cahill, 1897). L'organo Hammond sommava il suono di diversi generatori elettrici chiamati "ruote foniche" per sintetizzare il suono desiderato. Una ruota fonica consiste si un motore sincrono a corrente alternata ed un dispositivo di trasmissione che pilota una serie di dischi rotanti. Ogni disco contiene un certo numero di denti sul bordo che, a causa delle rotazioni ad una specifica velocità, danno vita ad una specifica frequenza, grazie ad un pick-up che raccoglie la variazione di campo magnetico, come avviene per esempio anche nelle chitarre elettriche. Al variare del numero dei denti della ruota fonica e alla velocità di rotazione, varia la frequenza generata. Poi, ogni frequenza così generata può essere amplificata, ed una singola frequenza fondamentale può essere combinata ad una o più armoniche per produrre suoni più complessi [3]. Mentre i sintetizzatori moderni elettronici utilizzano oscillatori elettrici per generare forme d'onda, l'Hammond utilizzava dispositivi elettromeccanici, più ingombranti, pesanti, costosi e con una manutenzione maggiore.

L'organo Hammond fu rivoluzionario e implementò molte altre novità che aiutarono a commercializzare lo strumento, come i diffusori Leslie che generano un particolare effetto sonoro tridimensionale dovuto alla rotazione degli altoparlanti in esso contenuti [4] [5] [6].

1.1.2 Sintesi sottrattiva

La maggior parte degli strumenti musicali possono essere rappresentati come una camera di risonanza stimolata da onde acustiche con certe proprietà temporali e spettrali.

La sintesi sottrattiva è una tecnica che si basa sul principio che il comportamento di uno strumento musicale si basa sull'interazione di una sorgente di eccitazione ed un risonatore. Il primo è ciò che fornisce un segnale al risonatore che modificherà le proprietà acustiche dell'onda. La tecnica è chiamata sottrattiva perché il risonatore svolge il compito di filtrare dallo spettro armonico le frequenze non desiderate, lasciando così soltanto le frequenze necessarie per ottenere il suono desiderato [7].

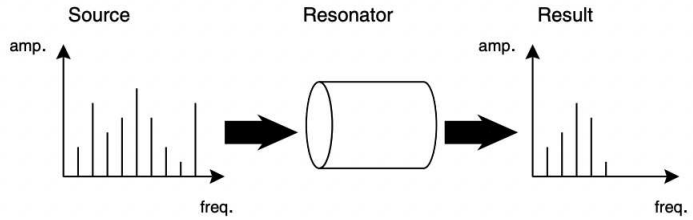

Figura 1.2: E' possibile ottenere una varietà di spettri sonori variando le proprietà acustiche del risonatore

Con la sintesi sottrattiva, a partire da un suono ricco di armoniche, si applicano dei filtri che permettono di escludere frequenze dello spettrogramma non desiderate ed ottenere il timbro ricercato.

Ad oggi, molti sintetizzatori utilizzano la sintesi sottrattiva, poiché permette di ottenere suoni piacevoli, elaborati, ricchi di armoniche, ad un prezzo minore, sia economicamente (per la componentistica) sia per la complessità di elaborazione del segnale.

Il risonatore attua da filtro applicato ad un segnale di eccitazione. La generazione di suoni appropriati più o meno complessi può essere ottenuta sfruttando la combinazione di sorgenti di segnali appropriati con delle specifiche proprietà filtranti di uno o più risonatori. Infatti, esistono diversi tipi di filtro (passa-basso, passa-alto, passa-banda...) che verranno illustrati successivamente nella sezione 1.2.2

Scegliere fra un approccio con sintesi additiva o sottrattiva dipende dall'obiettivo di suono ricercato. La sintesi additiva ha maggior precisione di controllo sulle singole parziali, ma la sintesi sottrattiva può risultare più semplice da gestire in presenza di numerose parziali. Inoltre, con la sintesi sottrattiva è possibile modificare molte parziali alla volta tramite i filtri, ma, se l'obiettivo sonoro è molto differente dalla sorgente da elaborare, potrebbe essere più complicato da ottenere e probabilmente sarà necessario utilizzare diversi tipi di filtro alla volta [8].

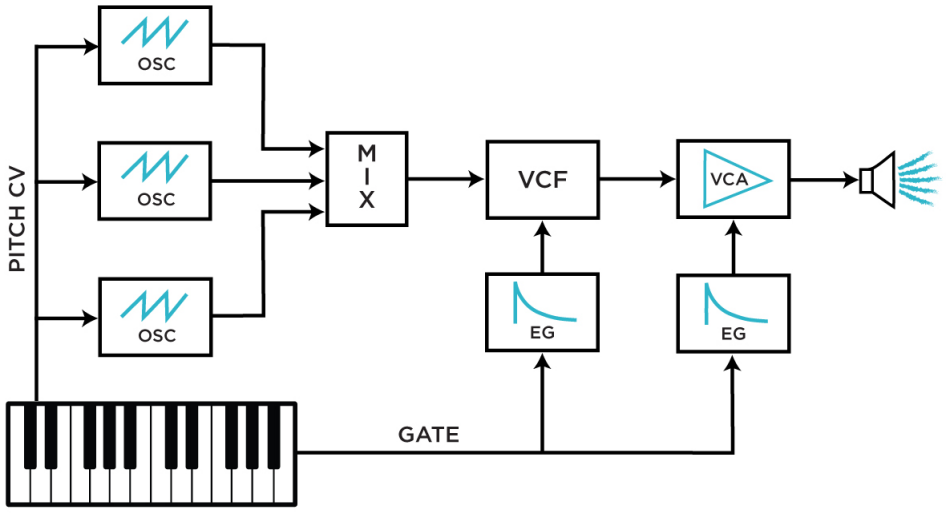

Figura 1.3: Esempio basilare di architettura di sintesi sottrattiva

In una comune architettura di sintesi sottrattiva, il percorso sonoro coinvolge diversi componenti chiave che lavorano insieme per generare il suono desiderato. Il percorso sonoro esteso per una tipica architettura di sintesi sottrattiva parte da uno o più oscillatori, che generano le forme d'onda di base. Queste forme d'onda costituiscono il punto di partenza per la creazione del suono. Un tipico approccio per la sintesi sottrattiva è quello di usare come sorgenti dei segnali di forme notevoli come onde quadre, a dente di sega o triangolari. Queste forme d'onda possono essere generate con componenti elettronici elementari e, a partire da esse, si filtrano le frequenze non necessarie per ottenere il timbro con lo spettro armonico desiderato. Queste forme d'onda generano una serie di armoniche che contribuiscono al timbro del suono. Le armoniche sono multipli interi della frequenza fondamentale e aggiungono complessità e ricchezza al suono. Poiché le forme d'onda complesse contengono una varietà di armoniche, possono essere modificate tramite filtri e altre tecniche di modulazione per ottenere una vasta gamma di suoni. Le onde sinusoidali, d'altro canto, hanno solo un'armonica fondamentale e sono più comuni per la creazione di suoni puri o toni semplici.

Alcune delle forme d'onda menzionate, come il dente di sega, sono particolarmente adatte per emulare il timbro di strumenti reali come fiati, archi e chitarre. Queste forme d'onda possono approssimare meglio le caratteristiche spettrali di tali strumenti.

Gli oscillatori possono essere regolati per impostare la frequenza desiderata del suono. Possono essere controllati manualmente tramite una tastiera o sequencer, o possono essere modulati da altre sorgenti, come un LFO (Low Frequency Oscillator) o un Envelope Generator. Inoltre, spesso è possibile regolare la frequenza in base all'ottava desiderata per creare variazioni nell'altezza del suono.

Dopo che gli oscillatori hanno generato le forme d'onda di base, queste possono essere messe insieme utilizzando un mixer. Il mixer consente di controllare il livello

di ciascun oscillatore e mescolarli tra loro per ottenere una combinazione sonora desiderata. Ad esempio, si possono mescolare più forme d'onda insieme per creare suoni più complessi e stratificati.

Il suono mescolato attraversa quindi un filtro. I filtri svolgono un ruolo cruciale nella sintesi sottrattiva, poiché consentono di plasmare il timbro del suono filtrando o attenuando determinate frequenze. Regolando i parametri del filtro, come la frequenza di taglio e la risonanza, si può ottenere il timbro desiderato.

Gli inviluppi sono generatori di forma d'onda che controllano la dinamica del suono nel tempo. I due inviluppi più comuni utilizzati nella sintesi sottrattiva sono l'inviluppo di ampiezza e l'inviluppo del filtro. L'Envelope Generator consente di controllare l'andamento dell'ampiezza del suono. L'inviluppo del filtro controlla invece il comportamento del filtro nel tempo, permettendo di modulare la forma spettrale del suono durante la sua durata.

Dopo aver attraversato il filtro e gli inviluppi, il segnale audio viene inviato a una sezione di amplificazione finale. Questa sezione controlla il volume finale del suono prima che venga emesso dagli altoparlanti o registrato. Può includere un controllo di volume, panning (posizionamento del suono nell'immagine stereo) e altri parametri di spazializzazione del suono.

Oltre a questi componenti principali, ci possono essere ulteriori elementi presenti nell'architettura di sintesi sottrattiva, come modulatori, effetti audio e altre caratteristiche specifiche del sintetizzatore. Tuttavia, il percorso sonoro descritto sopra rappresenta un flusso di base comune che viene seguito nella sintesi sottrattiva per generare suoni complessi e variabili.

Un esempio di sintetizzatore che utilizza la tecnica di sintesi sottrattiva è il Moog Minimoog, sviluppato dall'azienda americana Moog, pioniera nello sviluppo dei sintetizzatori.

Figura 1.4: Moog Minimoog

Il Moog Minimoog fu il primo sintetizzatore a presentarsi con un prezzo abbordabile e di dimensioni ridotte per i musicisti. Infatti, fino a quel momento, i sintetizzatori erano troppo ingombranti e costosi per un utente di tipo *consumer*.

Il Minimoog è dotato di tre oscillatori controllabili, ciascuno dei quali può generare forme d'onda diverse, come onde quadre, dente di sega e onde sinusoidali (6 tipi diversi di forme d'onda per ciascun oscillatore). Il segnale generato dagli oscillatori passa attraverso un mixer che consente di sommare le diverse forme d'onda e controllare il livello di ciascun oscillatore. Il mixer del Minimoog offre anche la possibilità di utilizzare forme d'onda esterne, come un'entrata audio esterna o un generatore di rumore, per ulteriori possibilità sonore.

Uno degli elementi chiave del percorso sonoro del Minimoog è il filtro passa-basso a $24dB$/ottava, noto come filtro ladder. Questo filtro è famoso per il suo suono "caldo" e caratteristico e può essere modulato per creare variazioni timbriche interessanti. Il filtro è di tipo passa-basso.

Il Minimoog è dotato di due generatori di inviluppo che controllano l'ampiezza e il filtro. Questi inviluppi consentono di modulare la dinamica del suono e controllare come inizia, si sviluppa, viene mantenuto e si spegne il suono. Gli inviluppi possono essere regolati per ottenere una vasta gamma di forme e modulazioni, aggiungendo espressività e movimento al suono. Il percorso sonoro del Minimoog include un amplificatore che controlla il volume del suono prima che venga emesso dagli altoparlanti. Inoltre, il sintetizzatore offre anche funzionalità di panning che consentono di posizionare il suono in diversi punti nello spazio stereo, contribuendo alla spazializzazione del suono, cioè la creazione di un'illusione di posizionamento e movimento del suono

in uno spazio tridimensionale, attraverso l'elaborazione e la riproduzione del suono su un sistema audio grazie al quale è possibile ottenere un'esperienza sonora che va oltre la semplice riproduzione mono o stereo e che simula la percezione di suoni provenienti da diverse direzioni e posizioni nello spazio.

Il Minimoog è stato uno dei primi sintetizzatori a introdurre un'interfaccia con controlli fisici intuitivi. La disposizione dei potenziometri e degli interruttori faceva sì che i musicisti potessero regolare immediatamente i parametri del suono senza dover navigare attraverso complessi menu digitali o utilizzare sequencer esterni. Ciò ha semplificato il processo di creazione sonora e ha consentito ai musicisti di sperimentare in modo più immediato e spontaneo.

Robert Moog riuscì a riunire le nuove tecnologie come i transistor analogici e il controllo in tensione in unico strumento musicale per renderlo di minori dimensioni, versatile ed economico. Inoltre, grazie ai suggerimenti di musicisti dell'epoca come Mort Garson o Wendy Carlos (che rese popolare per la prima volta al pubblico il lavoro di Moog grazie alle registrazioni di *Switched-On-Bach*), sviluppò le tecnologie che meglio potevano esprimere le richieste degli artisti abituati a strumenti più tradizionali: per esempio, Vladimir Ussachevsky diede a Robert Moog l'idea base per sviluppare poi il sistema di modulazione EG basato su ADSR, e Wendy Carlos suggerì l'importanza del portamento (lo scorrimento graduale da una nota a quella successiva) nella musica per tastiera [9].

In sintesi, il Moog Minimoog è stato rivoluzionario per la sua portabilità, la sua interfaccia intuitiva, il suo suono distintivo e la sua influenza culturale. Ha aperto nuove possibilità creative nella sintesi sonora, ha reso accessibile la sintesi ai musicisti di diverse fasce di prezzo ed è diventato un'icona nella storia dei sintetizzatori.

1.1.3 Sintesi FM

La sintesi per modulazione di frequenza (FM) è una tecnica di sintesi ampiamente utilizzata, in cui il segnale di un'onda detta portante (*carrier*) è modificata in base ad un'altra onda detta modulante *(modulating*), in modo che il ratio in cui la portante varia è la frequenza dell'onda modulante (frequenza di modulazione). I parametri di un segnale modulato in frequenza sono:

- c = frequenza portante o frequenza media,
- m = frequenza di modulazione,
- d = deviazione di frequenza

L'equazione per un'onda modulata in frequenza con un'intensità massima A dove sia la portante e sia la modulante sono due sinusoidi è:

$$e = Asin(\alpha t + Isin\beta t)$$

dove:

- e = Intensità istantanea della portante modulata,
- α = frequenza della portante
- β = frequenza della modulante
- $I = \frac{d}{m}$ = indice di modulazione, ovvero il rapporto tra la deviazione di frequenza e la frequenza di modulazione

Uno dei primi algoritmi per la sintesi audio FM fu sviluppato da John Chowning utilizzando il linguaggio di programmazione MUSIC V. Il programma generava dei sample (rappresentazioni numeriche di un'onda di pressione sonora) sotto forma di dati che rappresentano le caratteristiche fisiche del suono. I sample sono poi memorizzati e poi vengono passati ad un convertitore digitale/analogico, che genera una sequenza di impulsi di tensione le cui intensità sono proporzionali ai sample. Gli impulsi sono poi passati ad un filtro passa-basso e poi mandati al sistema audio [10].

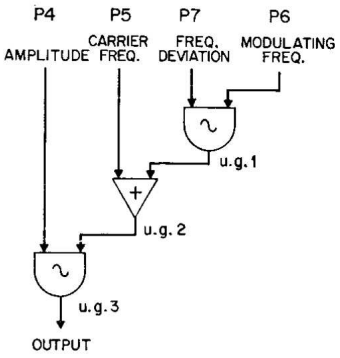

Figura 1.5: Circuito FM rappresentato nella notazione MUSIC V

La figura 1.5 rappresenta il diagramma di uno strumento composto da tre *unit generators*: due oscillatori e un adder. Assegnando i giusti parametri, lo strumento può generare suoni molto elaborati [10].

La sintesi FM utilizzando oscillatori analogici potrebbe generare instabilità nella frequenza del segnale in uscita [11], quindi la sintesi FM è solitamente digitale. La sintesi in Modulazione di Frequenza (FM) è un'ulteriore tecnica ad oggi ancora molto utilizzata.

Dopo che J. Chowning creò la sintesi FM presso la Stanford University, venne concessa la licenza a Yamaha che la utilizzò per creare il DX7. Nonostante ciò, Don Buchla, un altro pioniere nell'ambito della sintesi audio, aveva già implementato in alcuni dei suoi oscillatori (158, 258 e 259) la sintesi FM controllata in tensione [12]. Comunque, lo Yamaha DX7 dominò il mercato negli anni '80 e, una volta scaduto il brevetto (1995) di J. Chowning e della Stanford University, dove aveva svolto la sua attività di ricercatore, la sintesi digitale FM venne implementata liberamente da altri produttori [10].

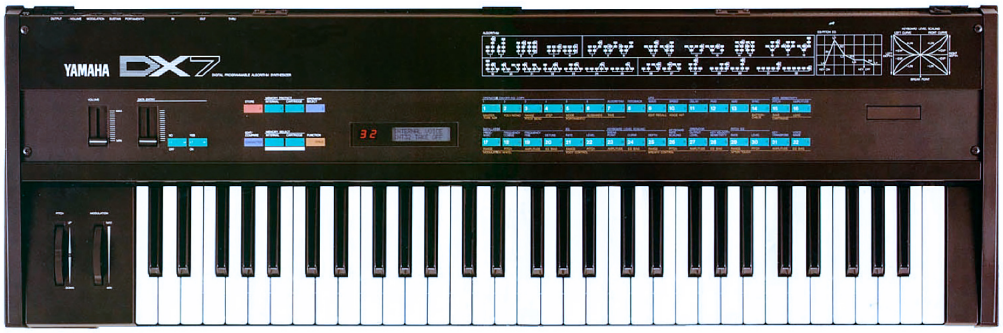

Figura 1.6: Yamaha DX7

Nel DX7, la forma d'onda modulata è nota come operatore, mentre l'onda che modula è detta modulatore. Il sintetizzatore utilizza sei operatori per generare il suono. Ogni operatore è un oscillatore digitale che può generare una forma d'onda sinusoidale.

Il principio di base della sintesi FM nel DX7 è che i modulatori influenzano la frequenza degli operatori di destinazione, generando armoniche superiori e timbri complessi. Questo processo di modulazione delle frequenze permette di ottenere una vasta gamma di suoni, dai suoni metallici e di campane alle sonorità di strumenti a fiato e di chitarre elettriche.

La programmazione del DX7 richiede la gestione delle relazioni tra gli operatori e i modulatori, determinando l'ampiezza, la frequenza e la profondità della modulazione per ottenere il suono desiderato. La complessità e l'interconnessione degli operatori fanno del DX7 uno strumento potente ma anche complesso da programmare. È noto per richiedere un certo grado di sperimentazione e conoscenza approfondita per ottenere risultati soddisfacenti.

Il successo del DX7 e della sua sintesi FM è stato principalmente attribuito alla sua capacità di riprodurre suoni realistici di strumenti acustici, come il pianoforte, il clavicembalo e le chitarre, in modo convincente. La sintesi FM del DX7 ha introdotto un nuovo approccio alla sintesi sonora, che differiva dalla sintesi sottrattiva tradizionale basata su onde quadre e dente di sega.

La popolarità del DX7 è stata notevole grazie alla sua versatilità e alle sue capacità sonore uniche. È stato ampiamente utilizzato nella produzione musicale degli anni '80 e ha influenzato molti generi musicali, dal pop al rock, dall'R&B alla musica elettronica. Ha aperto la strada a una nuova generazione di sintetizzatori digitali e ha dimostrato il potenziale della sintesi FM nella creazione di suoni complessi e dinamici.

1.2 Componenti di un sintetizzatore in sintesi sottrattiva

Nonostante la peculiarità di ogni sintetizzatore sia nella diversità delle tecnologie utilizzate e nelle connessioni tra i diversi moduli funzionali, è possibile identificare e descrivere quali sono le sezioni standard che fanno parte dei sintetizzatori più popolari.

Queste sezioni derivano da decenni di sperimentazioni e realizzati da ricercatori e/o aziende produttrici di strumenti musicali, ma i design tutt'ora ampiamente utilizzati sono derivanti dai primi sintetizzatori costruiti dalla Moog Music Inc., che fu la prima azienda a commercializzare su larga scala sintetizzatori analogici con tecnica di sintesi sottrattiva.

1.2.1 Oscillatore

Gli oscillatori possono essere sia analogici, sia digitali. Gli oscillatori analogici sono basati su circuiti e componenti elettronici che lavorano con segnali di tensione continua. Utilizzano dispositivi elettronici per generare onde continue e variabili nel tempo.

Gli oscillatori digitali operano con segnali digitali, ovvero sequenze discrete di valori binari. Questi oscillatori utilizzano algoritmi e tecniche di elaborazione digitale del segnale per generare forme d'onda. L'elaborazione digitale offre maggiore precisione e flessibilità, consentendo la generazione di una vasta gamma di forme d'onda e funzionalità avanzate.

La scelta tra oscillatori analogici e digitali dipende dalle preferenze e dalle esigenze dell'utilizzatore. Gli oscillatori analogici sono apprezzati per il loro suono caldo e organico, che può essere desiderato in determinati contesti musicali. Tuttavia, gli oscillatori digitali sono una componente comune nei sintetizzatori moderni. Gli oscillatori digitali offrono una serie di vantaggi rispetto ai loro omologhi analogici. Innanzitutto, grazie alla natura digitale del processo, gli oscillatori digitali possono produrre forme d'onda più precise e stabili rispetto agli oscillatori analogici che possono essere influenzati da variazioni di temperatura o da instabilità dei componenti elettronici.

Inoltre, gli oscillatori digitali consentono una maggiore versatilità e flessibilità nella creazione dei suoni. Oltre alle forme d'onda classiche come onde quadre, onde a dente di sega, onde triangolari e onde sinusoidali, gli oscillatori digitali possono generare una vasta gamma di altre forme d'onda complesse e personalizzate. Questo può includere forme d'onda campionate, forme d'onda generiche basate su algoritmi matematici o persino sintesi additiva che combina armoniche multiple [13].

In molti casi, i sintetizzatori e gli strumenti audio moderni combinano sia oscillatori analogici che digitali per sfruttare i vantaggi di entrambe le tecnologie. Questa combinazione consente di ottenere una vasta gamma di suoni e di esplorare nuove possibilità creative nel campo della sintesi sonora.

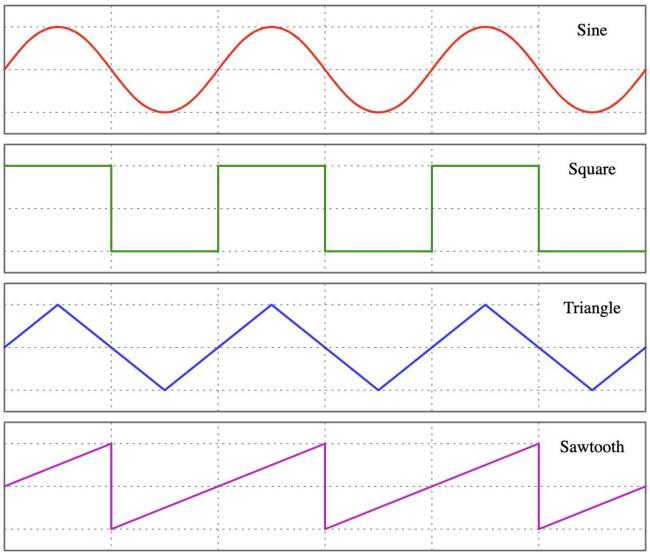

Figura 1.7: Alcuni tipi di forma d'onda

Gli oscillatori sono distinguibili in base al range di frequenze del segnale d'uscita:

1. Oscillatore a bassa frequenza (LFO: Low-Frequency Oscillator): oscillatori che generano frequenze di massimo 20Hz.

2. Oscillatore audio: oscillatori che generano frequenze nel range udibile dall'uomo (circa 20Hz - 20kHz).

Le tecniche di realizzazione degli oscillatori possono variare sia per gli oscillatori analogici che per quelli digitali. Di seguito si riportano alcune categorie fondamentali di oscillatori analogici e digitali.

Oscillatori analogici:

- Oscillatori basati su circuiti RC: Questi oscillatori utilizzano circuiti RC (resistenza-condensatore) come il ponte di Wien, il ponte a sfasamento e il circuito Colpitts. Sono spesso utilizzati negli oscillatori a bassa frequenza;

- Oscillatori basati su LC: Questi oscillatori utilizzano induttanze e condensatori (LC) per creare un'oscillazione. Esempi comuni sono l'oscillatore a sfasamento a cristallo e l'oscillatore a Colpitts a cristallo;

- Oscillatori a transistor: Questi oscillatori utilizzano transistor come elementi di amplificazione e feedback per creare un'oscillazione. Esempi noti sono gli oscillatori a base comune, a collettore comune e a emettitore comune;

- Oscillatori controllati in tensione (VCO): Questi oscillatori sono controllati da una tensione di controllo esterna, che può essere utilizzata per variare la frequenza dell'oscillazione. I VCO sono spesso utilizzati nei sintetizzatori modulari;

Oscillatori digitali :

- Oscillatori wavetable: Questi oscillatori generano forme d'onda riproducendo campioni di forme d'onda predefinite contenuti in una tabella (wavetable). Attraverso l'accelerazione o il rallentamento della frequenza di ricerca nel wavetable, è possibile creare diverse tonalità. Questo approccio unico alla generazione del suono offre interessanti possibilità creative. La ricerca nel wavetable può avvenire in avanti, all'indietro o solo in una parte specifica della tavola stessa [14].

- Oscillatori ad accumulo di fase (Phase Accumulator): Questi oscillatori digitali generano un'oscillazione controllando la fase di un segnale campionato. La frequenza dell'oscillazione è determinata dalla frequenza di campionamento e dall'incremento della fase [15].

- Oscillatori basati su tecniche di riduzione dell'aliasing: Questi oscillatori digitali utilizzano algoritmi di sintesi per generare forme d'onda complesse. Ad esempio, l'algoritmo *Bandlimited Impulse Trains (BLIT)* e l'algoritmo *Bandlimited Step (BLEP)* sono utilizzati per ridurre l'aliasing.

Nei sintetizzatori a sintesi sottrattiva, è comune osservare come siano presenti molto spesso due o più oscillatori. Questo perché, in primo luogo, un singolo oscillatore tenderà a produrre un suono statico, poco elaborato dal punto di vista musicale, date le forme d'onda comunemente generate. Questo non è vero in generale (per esempio le fasi successive di filtraggio o di applicazione di effetti possono aggiungere valore timbrico al suono indipendentemente dal numero di oscillatori), e non è, in generale, un risultato negativo (dipendentemente dal tipo di suono che si vuole ricercare), ma in generale è sicuramente vero che è preferibile avere a disposizione più oscillatori per creare un suono più complesso e, in caso di necessità, utilizzare un numero ridotto di oscillatori tra quelli a disposizione.

Il limite massimo di oscillatori nei sintetizzatori di tipo digitale è generalmente dato dalla necessità di non sovraccaricare troppo la CPU. L'utilizzo di due oscillatori offre una buona combinazione di complessità e controllo del suono. Con due oscillatori è possibile creare timbri ricchi e stratificati, utilizzando diverse forme d'onda e accordando i due oscillatori a intervalli armonici. Questo permette di ottenere suoni più complessi rispetto all'uso di un singolo oscillatore. L'utilizzo di un numero limitato di oscillatori consente di ottimizzare l'uso delle risorse computazionali. Ogni oscillatore richiede una certa quantità di potenza di calcolo, memoria e risorse del processore. Limitare il numero di oscillatori consente di utilizzare in modo efficiente le risorse disponibili, soprattutto in sistemi hardware o software con limitazioni di prestazioni.

Aliasing e tecniche di anti-aliasing

Nella sintesi digitale, i segnali audio vengono campionati a intervalli regolari per convertirli in forma digitale. Questo processo coinvolge la misurazione del valore del segnale audio ad intervalli di tempo discreti. Tuttavia, quando un segnale analogico viene campionato a una frequenza inferiore alla frequenza di Nyquist, possono verificarsi effetti indesiderati di aliasing. La frequenza di Nyquist rappresenta il limite superiore di frequenza per un segnale che viene campionato in modo corretto senza incorrere in fenomeni di aliasing. In altre parole, secondo il teorema di Nyquist-Shannon, la frequenza di campionamento deve essere almeno il doppio della frequenza massima presente nel segnale analogico originale. In pratica, la frequenza di Nyquist determina il limite massimo di frequenza che può essere correttamente rappresentato da un sistema di campionamento digitale. L'aliasing si verifica quando frequenze indesiderate si sovrappongono al segnale riprodotto, causando distorsioni e artefatti indesiderati. Questo accade perché la frequenza di campionamento insufficiente non riesce a catturare correttamente le informazioni sulle alte frequenze presenti nel segnale originale.

Per quanto riguarda gli oscillatori digitali, l'aliasing si manifesta quando l'oscillatore genera frequenze superiori alla metà della frequenza di campionamento. Ad esempio, se la frequenza di campionamento è di 44.1 kHz, qualsiasi componente di frequenza superiore a 22.05 kHz potrebbe causare aliasing.

L'aliasing negli oscillatori digitali può produrre suoni indesiderati e distorti. Le componenti di aliasing possono apparire come parziali spurie che non sono armoniche della fondamentale dell'oscillatore. Questo può deteriorare la qualità del suono e ridurre la fedeltà dell'oscillatore digitale rispetto alla sua controparte analogica.

Per affrontare il problema dell'aliasing negli oscillatori digitali, vengono utilizzate tecniche di anti-aliasing.

Nella sintesi wavetable, una forma d'onda viene campionata e memorizzata in una tabella di campioni (wavetable). Questa tabella di campioni contiene un ciclo completo della forma d'onda. Durante la riproduzione, l'oscillatore accede alla tabella di campioni ad una frequenza fissa (la frequenza di campionamento), e l'accumulatore di fase indica a che posizione in tabella accedere, per poi leggere i campioni corrispondenti per generare il segnale audio.

Il problema dell'aliasing nella sintesi wavetable si verifica quando la frequenza di riproduzione supera la metà della frequenza di campionamento.

Per mitigare l'aliasing nella sintesi wavetable (e non solo), sono state sviluppate diverse tecniche [16].

È stato dimostrato che un segnale che si avvicina molto all'onda a dente di sega ma con meno aliasing può essere prodotto applicando la derivata ad una forma d'onda parabolica a tratti [17]. La versione più semplice di questo algoritmo genera l'approssimazione della forma d'onda a dente di sega a banda limitata. Una versione sovracampionata dell'algoritmo dell'onda parabolica derivata produce una migliore

soppressione degli alias (gli artefatti indesiderati) e lascia spazio all'ottimizzazione utilizzando varie scelte di filtri.

Il metodo (*Differentiated Polynomial Waveforms (DPW*) estende il precedente metodo. Le onde quadre possono essere sintetizzate applicando l'operazione di derivazione alle forme d'onda triangolari. Inoltre, è stato osservato che i metodi *DPW* sono strettamente correlati alle tecniche di interpolazione polinomiale, poiché l'applicazione della derivata ad un segnale *DPW* a dente di sega o quadrato produce rispettivamente un treno di impulsi unipolare o bipolare, costituito da risposte all'impulso di filtri FIR di interpolazione. La valutazione percettiva dei segnali *DPW* a dente di sega mostra che l'aliasing può essere udibile nel metodo *DPW* del secondo ordine introdotto in precedenza a frequenze fondamentali superiori a 600 Hz, quando la frequenza di campionamento è 44100 Hz. Il metodo *DPW* del quarto ordine è privo di alias fino a circa 4,6 kHz e quindi consente la sintesi di praticamente tutte le frequenze fondamentali comunemente utilizzate nella musica [17].

Le tecniche BLIT (Band-Limited Impulse Train) e BLEP (Band-Limited Step) sono metodi utilizzati per ridurre l'aliasing negli oscillatori digitali e nella sintesi wavetable. Entrambe le tecniche si concentrano sulla limitazione delle frequenze indesiderate che si verificano durante la generazione delle forme d'onda.

La tecnica BLIT si basa sulla generazione di un treno di impulsi a banda limitata. Per ottenere una forma d'onda desiderata, viene generato un treno di impulsi periodico con frequenza molto alta. Successivamente, viene applicato un filtro passa-basso per limitare la banda di frequenze del segnale, eliminando così le componenti ad alta frequenza che causano l'aliasing. Il filtro passa-basso permette solo le frequenze contenute nella banda desiderata di passare, attenuando le altre frequenze al di fuori di essa.

La tecnica BLEP, invece, si concentra sulla correzione delle transizioni di fase durante il passaggio tra campioni successivi. Durante la generazione delle forme d'onda, le transizioni di fase possono causare la comparsa di frequenze indesiderate. La tecnica BLEP compensa queste transizioni di fase aggiungendo un'onda di correzione, di durata limitata, che annulla le discontinuità di fase. Questa correzione permette di evitare l'aliasing causato dalle transizioni brusche tra campioni.

Entrambe le tecniche BLIT e BLEP richiedono una gestione precisa delle transizioni e dei parametri di generazione delle forme d'onda. È necessario calcolare e applicare correttamente i filtri passa-basso o le correzioni di fase al fine di ottenere una limitazione efficace dell'aliasing. Queste tecniche possono essere implementate sia a livello di software che a livello di hardware, in base alle esigenze specifiche dell'applicazione.

Le tecniche BLIT e BLEP sono considerate metodi avanzati per la riduzione dell'aliasing negli oscillatori digitali e nella sintesi wavetable. Consentono di generare forme d'onda più accurate e di alta qualità, riducendo al minimo gli artefatti indesiderati e migliorando la resa sonora complessiva [18].

1.2.2 Filtro (VCF)

Una volta generata una o più onde sonore dagli oscillatori, queste vengono mixate e successivamente il segnale viene mandato ad un filtro. I filtri servono ad attenuare alcune specifiche frequenze di un'onda complessa. Questo è utile per enfatizzare o limitare alcune componenti del suono, modificandolo per far percepire soltanto il contenuto "interessante" da un punto di vista musicale. Esistono molti tipi diversi di filtri, ma essi sono spesso ottenuti combinando un piccolo numero di filtri di tipo elementare [8].

Esistono diversi tipi di filtro:

- *Low-pass*: I filtri passa-basso attenuano o eliminano le componenti di frequenza superiori a una determinata frequenza di taglio (*cutoff*), lasciando passare le componenti di frequenza più basse. Ciò significa che riducono l'energia delle frequenze alte e permettono l'accentuazione delle frequenze basse;

- *High-pass*: I filtri passa-alto fanno esattamente l'opposto dei filtri passa-basso. Attenuano o eliminano le componenti di frequenza inferiori a una determinata frequenza di taglio e lasciano passare le componenti di frequenza più alte. Questo significa che riducono l'energia delle frequenze basse e consentono l'accentuazione delle frequenze alte;

- *Band-pass*: I filtri passa-banda consentono solo il passaggio di una banda di frequenza specifica e attenuano o eliminano le componenti al di fuori di tale banda. Questo tipo di filtro consente di enfatizzare o isolare una determinata gamma di frequenze all'interno dello spettro sonoro;

- *Notch/Band-reject*: Questi filtri sono progettati per attenuare o eliminare selettivamente una stretta banda di frequenze all'interno dello spettro audio. A differenza dei filtri passa-banda che permettono solo il passaggio di una specifica banda di frequenze, i filtri notch/band-reject operano esattamente all'opposto. Attenuano o eliminano le frequenze all'interno della banda selezionata, lasciando passare il resto dello spettro;

La *resonance* (o fattore Q) è un parametro che deriva dagli originali filtri analogici realizzati. Nell'intorno della frequenza di cutoff, si genera un picco di volume che aggiunge sonorità alle frequenze in prossimità di quelle che vengono "tagliate". Indica la risonanza o la reattività del filtro a una determinata frequenza.

Quando il parametro *resonance* viene aumentato, il filtro enfatizza le frequenze vicino alla frequenza di taglio, creando un picco di risonanza nello spettro sonoro. Questo può conferire al suono una caratteristica più pronunciata, risonante o addirittura "metallica". La *resonance* può contribuire a enfatizzare o accentuare parti specifiche dello spettro sonoro e può essere utilizzata per ottenere effetti timbrici particolari o per creare suoni più vivaci e penetranti.

D'altra parte, una *resonance* bassa o assente produce un effetto di smorzamento o attenuazione delle frequenze vicine alla frequenza di taglio, rendendo il suono più "smorzato" o meno risonante.

Il valore del fattore Q o *resonance* può variare da un filtro all'altro. Valori più alti di *resonance* generano una risonanza più accentuata e picchi più pronunciati nello spettro sonoro, mentre valori più bassi producono una risonanza più sottile o quasi impercettibile. La regolazione del parametro di *resonance* offre un ulteriore controllo sulla forma timbrica del suono e può essere utilizzata in modo creativo per modulare l'intensità o il carattere delle frequenze selezionate.

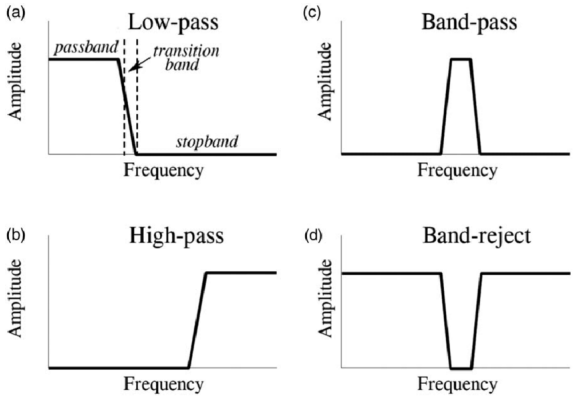

Figura 1.8: Tipi di filtro

I filtri possono essere di tipo analogico o di tipo digitale, ognuno con le sue peculiarità e differenze. I filtri analogici operano su segnali continui nel dominio del tempo e dell'ampiezza, mentre i filtri digitali operano su segnali discretizzati, rappresentati da campioni prelevati a intervalli regolari nel tempo.

Filtri analogici

Filtri Sallen-Key Un semplice design di realizzazione per i filtri di tipo passa-basso, passa-alto e passa-banda è il filtro Sallen-Key, introdotto nel 1955 da R.P. Sallen e E.L. Key [19].

La topologia Sallen-Key è un design di filtro attivo basato su un singolo amplificatore operazionale non invertente, resistori e condensatori. Di seguito sono mostrate le topologie di filtri di tipo passa-basso, passa-alto e passa-banda secondo il design Sallen-Key.

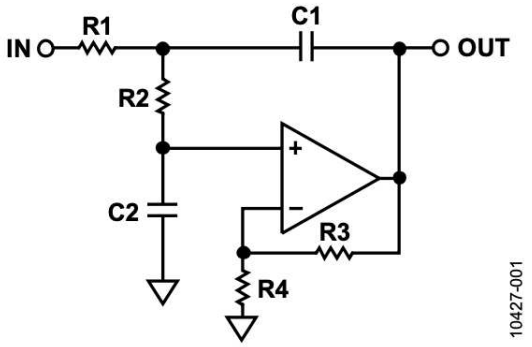

Figura 1.9: Filtro passa-basso di Sallen-Key

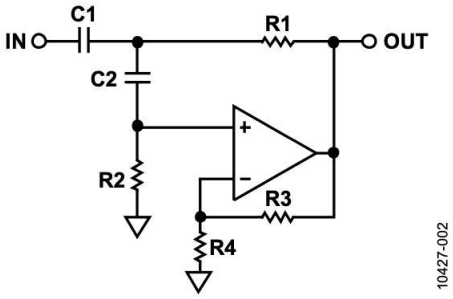

Figura 1.10: Filtro passa-alto di Sallen-Key

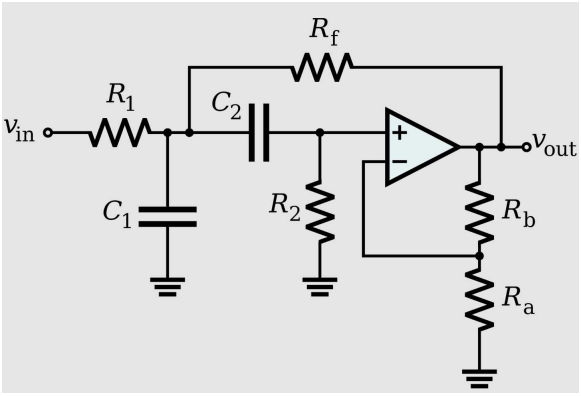

Figura 1.11: Filtro passa-banda di Sallen-Key

Filtro Korg MS-20 Il Korg MS-20 è un sintetizzatore analogico monofonico a sintesi sottrattiva rilasciato nel 1978. L'MS-20 originale aveva in realtà due diversi design di realizzazione del filtro. Il primo era un filtro compatto basato su un chip che Korg chiamò Korg35, basato sul design Sallen-Key.

Figura 1.12: Korg MS-20

Di seguito, viene mostrata la rappresentazione circuitale del Korg35.

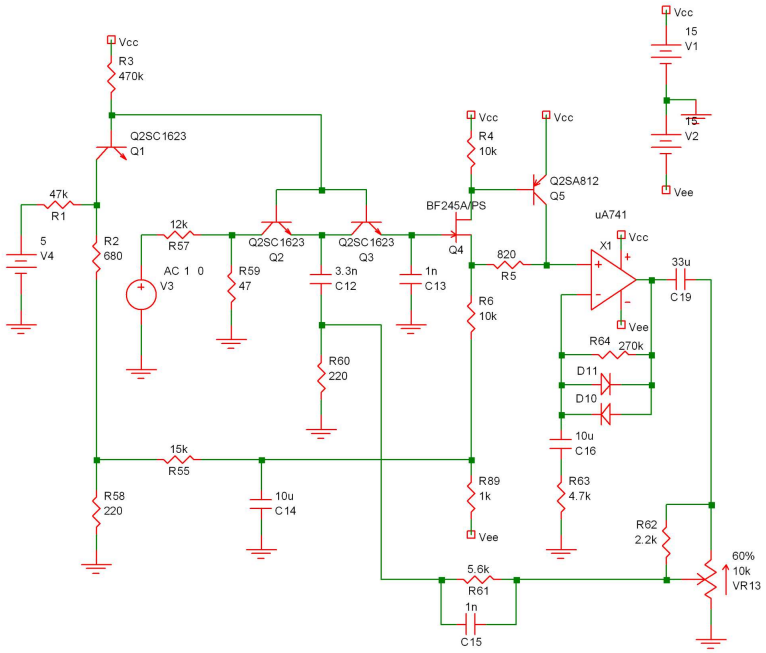

Figura 1.13: 'Korg35', filtro del Korg MS10 e delle prime versioni del Korg MS20 [20]

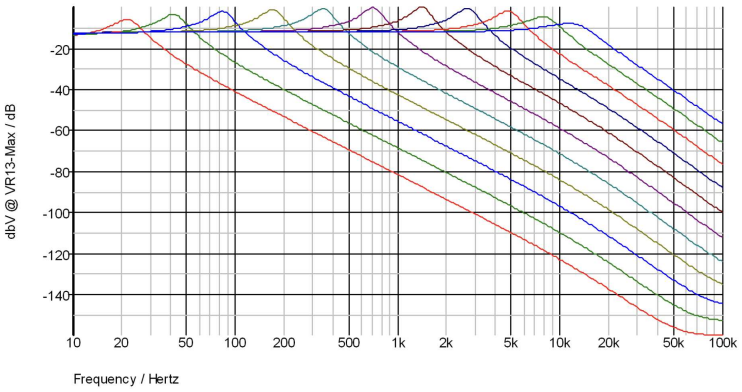

Figura 1.14: Risposta in frequenza del filtro Korg35 [20]

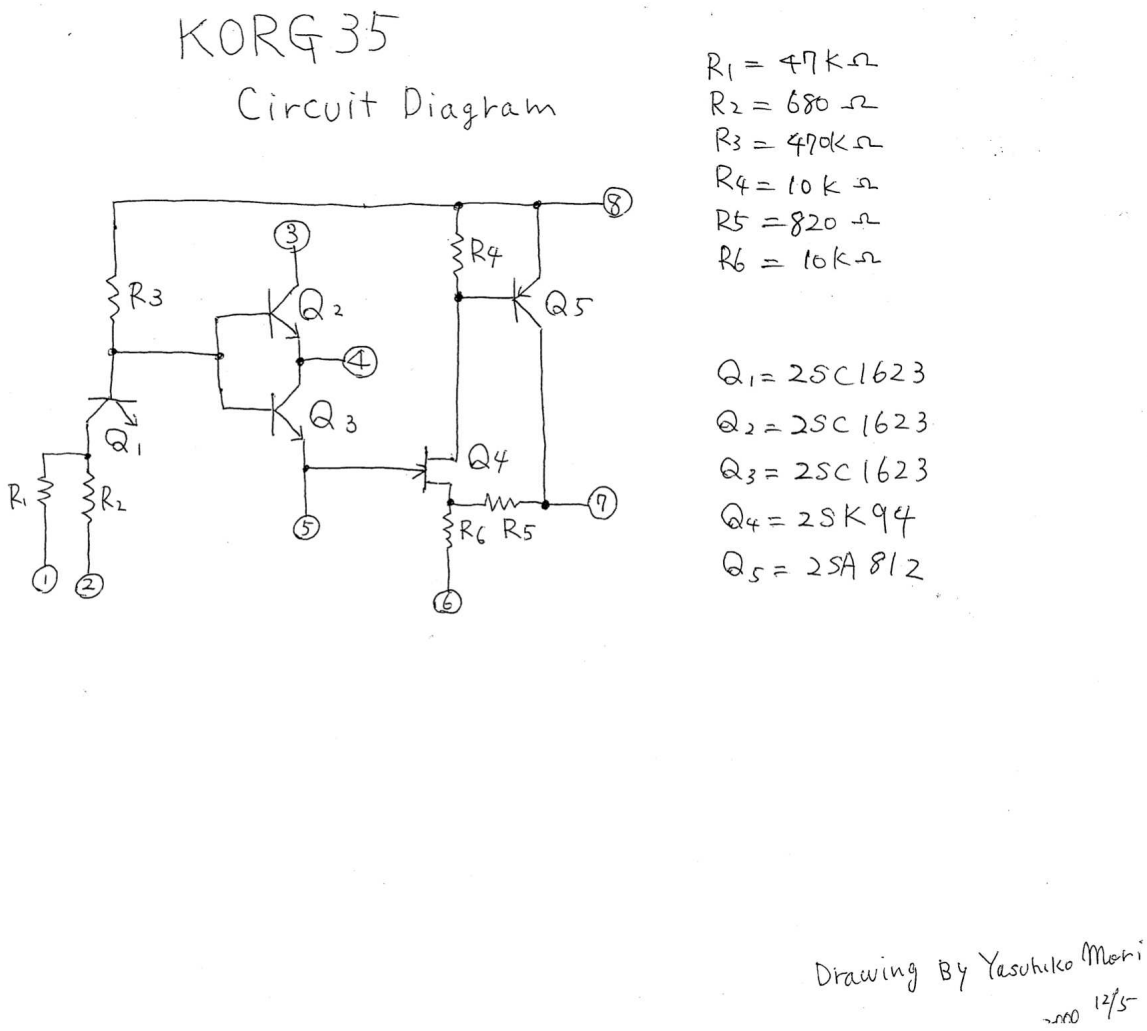

Figura 1.15: Disegni originali di Yasuhiko Mori

Filtro Ladder Il filtro Ladder è utilizzato in molti sintetizzatori Moog.

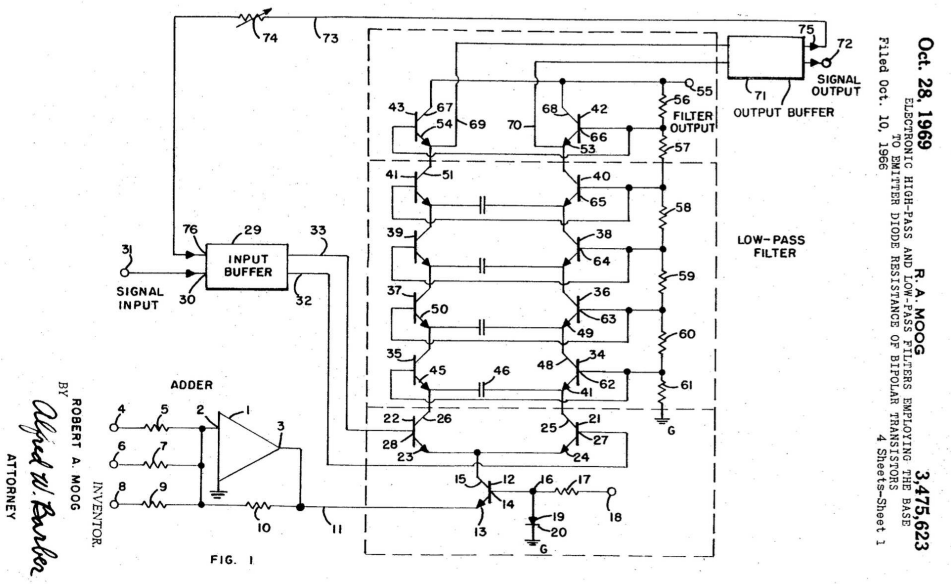

Figura 1.16: Circuito originale del filtro Ladder, come progettato da R. Moog [21].

Filtri digitali

Il modo più semplice per implementare un filtro digitale consiste nell'applicare l'operatore di convoluzione al segnale di ingresso con la risposta all'impulso del filtro digitale. Tutti i possibili filtri lineari possono essere realizzati in questo modo. Quando la risposta all'impulso viene utilizzata in questo modo, viene anche detta il *kernel* del filtro.

Esiste anche un altro modo per creare filtri digitali, chiamato ricorsione. Quando un filtro viene implementato per convoluzione, ogni campione nell'output viene calcolato ponderando i campioni nell'input e sommandoli. I filtri ricorsivi sono un'estensione di questo metodo, utilizzando valori precedentemente calcolati dall'output, oltre ai punti dall'input. Invece di utilizzare un *kernel* del filtro, i filtri ricorsivi sono definiti da un insieme di coefficienti di ricorsione.

Per trovare la risposta all'impulso di un filtro ricorsivo, è sufficiente utilizzare un impulso come input del filtro e analizzare l'output. Le risposte all'impulso dei filtri ricorsivi stabili sono composte da sinusoidi che decadono esponenzialmente in ampiezza. In linea di principio, questo rende le loro risposte all'impulso infinitamente lunghe. Tuttavia, l'ampiezza alla fine scende al di sotto del rumore di arrotondamento del sistema e i campioni rimanenti possono essere ignorati. per questa caratteristica, i filtri ricorsivi sono anche chiamati *Infinite Impulse Response (IIR)*, filtri realizzati per convoluzione sono chiamati filtri *Finite Impulse Response (FIR)* [22].

Un filtro digitale è definito dalla sua funzione di trasferimento o, in altre parole, dalla sua equazione alle differenze. Questa equazione matematica descrive come il filtro risponderà a un determinato input. La progettazione di un filtro digitale

implica la definizione di specifiche appropriate per il problema da risolvere, come ad esempio un filtro passa-basso di secondo ordine con una frequenza di taglio specifica. Successivamente, si procede alla creazione della funzione di trasferimento che rispetti tali specifiche, permettendo al filtro di operare in modo desiderato [23].

La funzione di trasferimento per un filtro digitale lineare tempo-invariante può essere espressa nel dominio Z.

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + ... + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + ... + a_M z^{-M}}$$

I coefficienti del denominatore, a_k, descrivono la parte *autoregressive (AR)* del filtro e i coefficienti del numeratore, b_k, la parte *Moving Average (MA).*

Alla funzione di trasferimento corrispondono N zeri e M poli, e l'ordine del filtro è il valore maggiore fra N ed M. In questa forma, si rappresenta un *filtro ricorsivo.* A differenza dei filtri non ricorsivi, i filtri ricorsivi hanno una dipendenza dagli output precedenti oltre che dagli input correnti. Questa retroazione permette al filtro ricorsivo di avere una risposta in frequenza più complessa e flessibile rispetto ai filtri non ricorsivi a parità di ordine del filtro [23].

La risposta all'impulso è una caratterizzazione del comportamento del filtro.

Nel caso di filtri FIR lineari tempo-invarianti, la risposta all'impulso y_n è esattamente uguale alla sequenza dei coefficienti del filtro, e quindi:

$$y_n = \sum_{k=0}^{N} b_k x_{n-k} = \sum_{k=0}^{N} h_k x_{n-k}$$

La forma generale di un filtro IIR è invece:

$$\sum_{m=0}^{M} a_m y_{n-m} = \sum_{k=0}^{N} b_k x_{n-k}$$

Un filtro IIR è sempre ricorsivo. Mentre è possibile che un filtro ricorsivo abbia una risposta all'impulso finita, un filtro non ricorsivo ha sempre una risposta all'impulso finita.

Nei sistemi a tempo discreto, il filtro digitale viene spesso implementato convertendo la funzione di trasferimento in un'equazione differenziale a coefficiente costante lineare (LCCD) tramite la trasformata Z.

$$y[n] = -\sum_{k=1}^{M} a_k y[n-k] + \sum_{k=0}^{N} b_k x[n-k]$$

Dove:

- y = Output, ovvero il valore filtrato
- x = Input

- n = Il numero del campione, il numero dell'iterazione o il numero del periodo di tempo.

Dopo che un filtro è stato progettato, deve essere realizzato sviluppando un diagramma di flusso del segnale che descriva il filtro in termini di operazioni sulle sequenze di campioni.

Una data funzione di trasferimento può essere realizzata in molti modi. Di seguito, due possibili schemi a blocchi per implementare direttamente un filtro IIR.

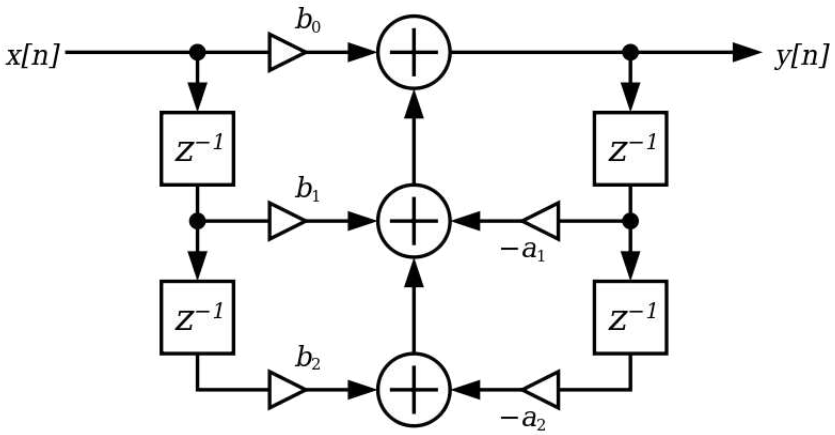

Figura 1.17: Forma diretta I

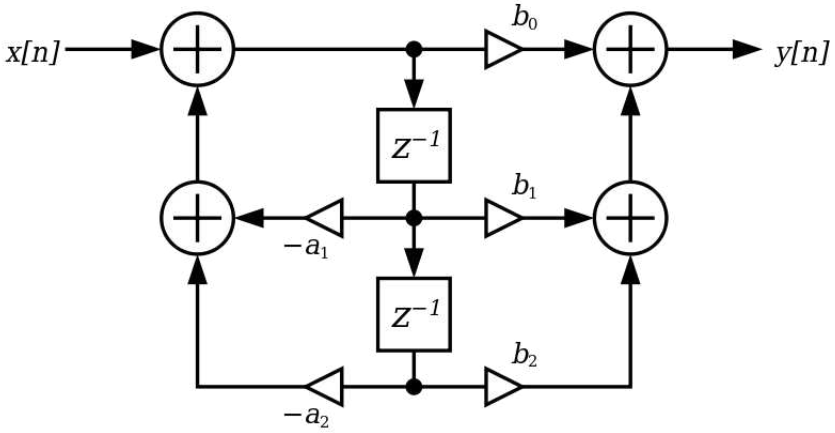

Figura 1.18: Forma diretta II

I filtri implementati per mezzo della convoluzione possono avere prestazioni di gran lunga migliori rispetto ai filtri che utilizzano la ricorsione, ma vengono eseguiti molto più lentamente.

La *Moving Average (MA)* viene utilizzata nel dominio del tempo, il *windowed-sinc* viene utilizzato nel dominio della frequenza. La tecnica chiamata *convoluzione FFT*

(Fast-Fourier Transform) è un algoritmo per aumentare la velocità di convoluzione, consentendo ai filtri FIR di essere eseguiti più velocemente.

Per quanto riguarda i filtri ricorsivi, il *filtro ricorsivo unipolare* viene utilizzato nel dominio del tempo, mentre il filtro *Chebyshev* viene utilizzato nel dominio della frequenza. I filtri ricorsivi con una risposta personalizzata sono progettati con tecniche iterative. Come mostrato nella tabella **??**, la convoluzione e la ricorsione sono tecniche mutualmente esclusive; è necessario utilizzare l'una o l'altra per una particolare applicazione [22].

Per esempio, un filtro digitale che utilizza una struttura con due poli e due zeri (Two-Pole-Two-Zero) nella sua funzione di trasferimento è progettato per consentire una modulazione veloce dei coefficienti del filtro senza introdurre discontinuità o artefatti indesiderati nel segnale audio. L'equazione di trasferimento può essere scritta come:

$$H(z) = \frac{(b_0 + b_1 z^{-1} + b_2 z^{-2})}{(1 + a_1 z^{-1} + a_2 z^{-2})}$$

1.2.3 Envelope Generator (EG)

L'Envelope Generator è un componente che permette di controllare come il volume di un suono cambi nel tempo.

Oltre che il volume, l'Envelope Generator può controllare anche altri componenti, ad esempio l'inviluppo del filtro [24].

L'Envelope Generator normalmente viene controllato in base a quattro parametri che vengono riassunti con l'acronimo ADSR (Attack-Decay-Sustain-Release), che indica uno schema concettuale di realizzazione dell'inviluppo generato da un EG.:

- *Attack*: L'*attack* rappresenta la fase iniziale del suono, che definisce quanto velocemente il suono raggiunge il suo picco di volume massimo. È il tempo impiegato dal suono per passare da un livello di volume iniziale a un livello massimo, determinando l'impatto iniziale e l'intensità del suono. Ad esempio, un attacco rapido crea un suono di percussione acuto, mentre un attacco lento produce un suono più graduale.

- *Decay*: Il *decay* rappresenta la fase successiva dopo l'attacco, in cui il suono diminuisce gradualmente dal suo picco di volume massimo al livello di sustain. Questa fase definisce quanto velocemente il suono si attenua dopo l'attacco iniziale. Il tempo di decadimento determina la durata della parte iniziale del suono, prima di raggiungere il livello di sustain.

- *Sustain*: Il *sustain* rappresenta la fase in cui il suono viene mantenuto a un livello costante dopo il decadimento. In questa fase, il suono si mantiene al livello di volume desiderato finché il musicista tiene premuto il tasto o il segnale viene mantenuto attivo. Questa fase può variare in durata a seconda di quanto tempo viene mantenuto il suono.

- *Release*: Il *release* rappresenta la fase finale in cui il suono si attenua gradualmente dopo che il tasto è stato rilasciato o il segnale è cessato. Indica quanto velocemente il suono si estingue una volta che il tasto è stato rilasciato. Un rilascio rapido crea una dissolvenza rapida del suono, mentre un rilascio più lento permette al suono di durare più a lungo prima di scomparire.

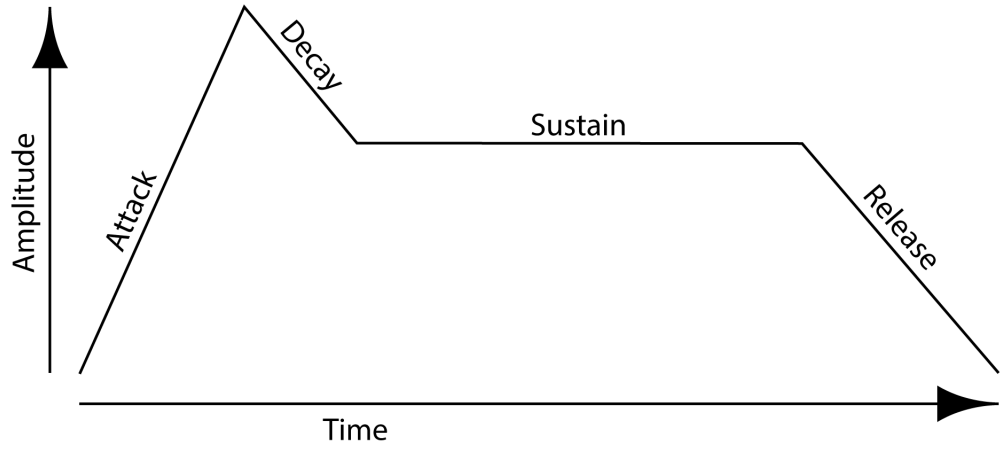

Figura 1.19: ADSR: Attack, Decay, Sustain, Release

L'Envelope Generator è ampiamente utilizzato nella sintesi sonora e può essere applicato a diverse caratteristiche del suono, come il volume, la modulazione di frequenza, la forma d'onda e altro ancora. Consente ai musicisti e ai sound designer di modellare la dinamica dei suoni in modo preciso, consentendo un maggiore controllo e una maggiore espressività nella creazione musicale e nella progettazione sonora.

I generatori di inviluppo differiscono dai VCO e VCF perché in input hanno dei trigger (o "gate"), dei picchi di tensioni relativamente alti che iniziano il ciclo del generatore di inviluppo (EG) (esistono anche alcuni Envelope Generator avanzati con parametri controllabili in tensione).

Figura 1.20: Modulo EG (doppio) del Technosaurus Selector

In Figura 1.20, il sistema ADSR del *Technosaurus Selector*, che ha un doppio generatore di inviluppo consistente anche di un delay (tale approccio è simile anche nel Korg MS20, che ha anche un controllo "Hold Time"): ciò significa che c'è un periodo di tempo variabile dopo che la nota è suonata, prima che l'inviluppo venga attivato. Il Moog Minimoog, invece, possiede solo controlli ADS (Attack, Decay, Sustain), e il Release Time è controllato dalla stessa manopola che controlla la fase di Decay. Questa limitazione venne rimossa da Robert Moog nei successivi Crumar Spirit e Moog Voyager [25]

1.2.4 LFO

L'LFO (*low-frequency oscillator*) è un oscillatore che ha la peculiarità di generare frequenze molto basse, solitamente sotto i 20Hz. Gli LFO vengono utilizzati per modulare parametri audio o controllare l'ampiezza, la frequenza o altri parametri di un suono nel tempo. E' utile per realizzare effetti tipici degli strumenti acustici.

Per esempio, il segnale di un LFO può essere utilizzato per modificare nel tempo la frequenza di *cutoff* di un filtro ("*wobble*"), il pitch di un oscillatore ("*vibrato*"), il volume ("*tremolo*"), ecc.

Oppure, un LFO può essere utilizzato per modulare i parametri di un filtro, come la frequenza di taglio o la risonanza, producendo gli effetti *wah-wah* o di *sweeping*.

Può essere anche utilizzato per creare effetti di modulazione spaziale come il *chorus*, il *flanger* e il *phaser*, che aggiungono una sensazione di movimento e spazialità al suono, oppure può modulare i parametri di un inviluppo creando variazioni ritmiche nel tempo.

1.2.5 Amplifier (VCA)

Un VCA (Voltage-Controlled Amplifier) è un componente che determina il livello di volume in output del segnale audio generato dal sintetizzatore. E' utilizzato, di solito, nella forma di un controllo per il volume finale di un segnale. Nei sintetizzatori non modulari di piccole dimensioni di solito il VCA non è separato dalle altre sezioni in modo marcato. Può esserci, inoltre, anche un controllo per il *drive*, che sovraccarica leggermente il VCA in uscita per creare un effetto di distorsione sonora, una funzione incidentale divenuta popolare grazie al Moog Minimoog originale, poi resa molto più esplicitamente disponibile.

Il funzionamento di un VCA è basato sulla variazione di una tensione di controllo, che determina l'ampiezza del segnale audio in uscita. Il segnale audio in ingresso passa attraverso l'amplificatore e la sua ampiezza viene modificata in base al valore della tensione di controllo applicata al VCA.

Un VCA può essere controllato in diversi modi. Uno dei più comuni è l'utilizzo di un segnale di controllo generato da un generatore di inviluppo. Il generatore di inviluppo produce una forma d'onda che descrive l'andamento temporale dell'amplificazione del segnale. Ad esempio, può avere un attacco graduale, un decadimento progressivo, un livello di *sustain* costante e un rilascio graduale. Questo permette di plasmare il suono in base alle variazioni desiderate nel tempo.

Un altro metodo di controllo comune è tramite l'utilizzo di un LFO, che permette quindi di modulare l'ampiezza del segnale audio ritmicamente, creando effetti come il tremolo.

Un aspetto importante dei VCA è la loro risposta alla tensione di controllo. Alcuni VCA hanno una risposta lineare, il che significa che l'ampiezza del segnale in uscita varia in modo proporzionale alla tensione di controllo. Altri VCA possono avere risposte più complesse, come curve di risposta esponenziali o curve di saturazione non lineari. Questo consente di ottenere una maggiore flessibilità nella modulazione dell'ampiezza del segnale.

I VCA sono ampiamente utilizzati nella sintesi sonora e nella produzione musicale per creare una varietà di effetti e modulazioni. Possono essere impiegati per controllare il volume di una voce o di uno strumento, creare effetti di modulazione ritmica, generare forme d'onda complesse attraverso la modulazione incrociata di VCA multipli e molto altro ancora.

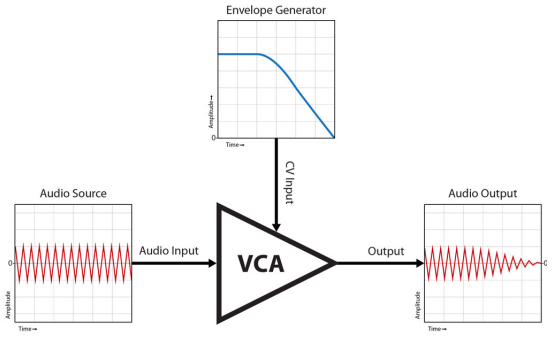

Figura 1.21: VCA: Un segnale con intensità controllata da un segnale esterno

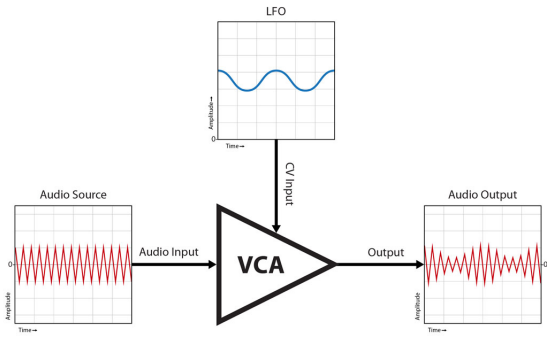

Figura 1.22: VCA controllato da un LFO per generare un Tremolo

1.2.6 Effetti

Gli effetti audio possono essere componenti dei sintetizzatori o stand-alone. Si utilizzano per alterare il suono di uno strumento musicale aggiungendo varie proprietà a seconda del tipo di effetto utilizzato.

Effetti dinamici

Gli effetti *dinamici* modificano il volume dello strumento.

- *Compressori*: servono ad attenuare il livello di volume di un suono, impostando una soglia massima. Inoltre, spesso sono utilizzati anche per stabilizzare la gamma dinamica di un canale audio, innalzando tramite un *gain* il volume quando troppo basso.

  I compressori hanno un parametro, il *ratio*, che determina quanta compressione viene applicata: per esempio, se il ratio è 2:1, ad un segnale che eccede la soglia massima di $4dB$ verrà applicata una riduzione di $2dB$. Se il ratio è ∞:1 (o comunque molto alto), il compressore è chiamato *limitatore*: esso di fatto imposterà una soglia massima che non potrà essere oltrepassata, e tutto il

segnale generato verrà compresso per ottenere un'intensità del suono sempre uguale o minore del valore desiderato.

- *Gate*: attenuano il volume di un segnale audio finché questo non supera un certo valore di soglia. In questo modo, è possibile eliminare una certa quantità di rumore nei momenti in cui lo strumento non sta suonando, ovvero nei momenti in cui quel rumore potrebbe risultare più udibile.

Effetti time-based

Gli effetti time-based si basano tutti sul principio del *delay*: riprodurre il segnale sonoro ritardandolo nel tempo.

- *Delay*: è realizzato prendendo il segnale audio suonato e riproducendolo dopo un certo intervallo di tempo. Il numero di volte per cui il segnale viene ripetuto è chiamato *feedback*.
- *Echo*: concettualmente è molto simile al delay, in quanto si tratta di segnale ripetuto dopo un certo intervallo di tempo; la differenza è che le successive "riflessioni" del segnale hanno dei tempi generalmente più lunghi e ad ogni riflessione il volume del suono diminuisce.
- *Reverb*: è un effetto utilizzato per simulare le riflessioni di un segnale prodotte in un determinato ambiente (ad esempio una chiesa o un teatro). A differenza del *delay* e dell'*echo*, il *reverb* simula moltissime più riflessioni (anche milioni) ed è per questo più pesante computazionalmente.

Effetti di Equalizzazione (EQ)

L'equalizzazione viene utilizzata per regolare il bilanciamento delle frequenze nel suono. Permette di aumentare o ridurre le frequenze specifiche per correggere, migliorare o modificare il timbro complessivo del suono.

Altri effetti

Esistono innumerevoli altri tipi di effetti: per esempio, gli effetti di modulazione, che utilizzano un segnale *modulator*) per modulare il segnale *carrier* (*chorus*, *flanging*, *phaser*, *phase shifting*, etc.). Oppure, effetti come la distorsione o l'overdrive modificano il suono introducendo distorsione o saturazione armonica e creando un suono più aggressivo o "sporco": sono particolarmente popolari nelle chitarre elettriche e negli amplificatori per ottenere un caratteristico suono distorto.

1.3 MIDI

MIDI (Music Instrument Digital Interface) è il principale protocollo di comunicazione tra strumenti digitali e *DAW (Digital Audio Workstation)*. Nacque nel 1983,

con la necessità delle diverse aziende produttrici di strumenti musicali di avere un linguaggio comune con cui far comunicare le proprie macchine.

Il linguaggio MIDI è basato su messaggi che vengono scambiati dopo eventi come la pressione di un tasto. Il messaggio non rappresenta il suono stesso, ma rappresenta una sequenza di istruzioni e le caratteristiche del suono che poi il sintetizzatore utilizzerà per generare il suono.

Grazie a questo, il linguaggio MIDI è molto leggero in memoria ed è quindi rapido lo scambio di messaggi.

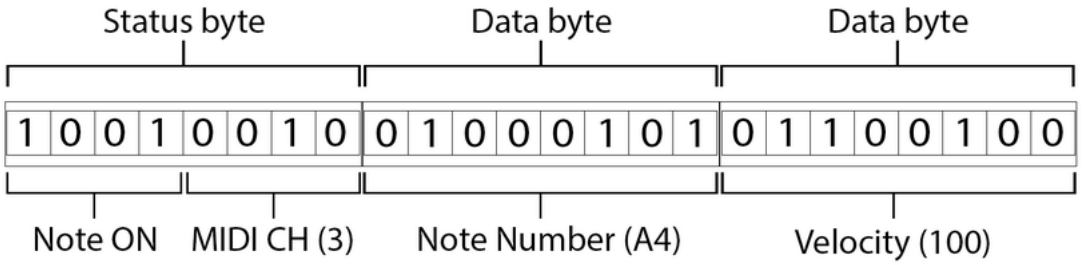

Figura 1.23: Esempio di MIDI message

I messaggi MIDI possono essere degli *STATUS byte* o dei *DATA bytes*: nel primo caso, il bit 7 è impostato a 1 e nel secondo è impostato a 0. Lo *STATUS Byte* determina il tipo di messaggio, il *DATA Byte* determina le informazioni caratteristiche di quel messaggio. Lo status byte, inoltre, contiene informazioni riguardanti il canale midi utilizzato: infatti, è possibile avere fino a 16 canali midi in comunicazione contemporaneamente, permettendo cioè di gestire 16 strumenti diversi.

I messaggi scambiabili con il protocollo MIDI sono **NOTE ON** e **NOTE OFF**, che rappresentano la pressione e il rilascio di un tasto di un sintetizzatore.

Nel caso del **NOTE ON**, le informazioni inviate nel *STATUS Byte* sono la comunicazione del tipo di evento (**NOTE ON**), e il canale. Le informazioni inviate nel *DATA Byte* sono il numero identificativo della nota, e la velocity, ovvero la forza di pressione del tasto. Il segnale **NOTE OFF** invia le stesse informazioni del **NOTE OFF** ma al posto della *velocity* comunica la *release velocity*.

Alcuni eventi MIDI: **NOTE ON**, **NOTE OFF**, **PAUSE**, **Aftertouch Polyphonic**, **Control Change**, **Program Change**, **Channel Aftertouch**,**Pitch Bend Change**, ecc.

Capitolo 2

Design del'architettura di sintesi

Il sintetizzatore leoSynth è stato progettato con l'intenzione di essere un sintetizzatore utilizzabile da un musicista, con l'ausilio di un dispositivo di input come una tastiera. Deve essere quindi in grado di generare dei suoni mediamente complessi, i cui parametri possono essere controllati semplicemente da dei potenziometri virtuali così da poter essere suonato in tempo reale.

2.1 Il framework

Per sviluppare il *leoSynth*, è stato adottato JUCE, un framework open-source, cross-platform basato sul linguaggio C++, molto utilizzato nello sviluppo desktop e mobile di plugin audio grazie alla presenza di librerie audio e GUI particolarmente intuitive ed efficaci. È stato creato da Jules Storer e offre una vasta gamma di funzionalità per semplificare lo sviluppo di software audio multipiattaforma.

JUCE si distingue per la sua portabilità, consentendo agli sviluppatori di scrivere il codice una volta sola e renderlo eseguibile su diverse piattaforme come Windows, macOS e Linux. Questa caratteristica semplifica il supporto multipiattaforma, riducendo il tempo e gli sforzi richiesti per adattare l'applicazione a diversi sistemi operativi.

Inoltre, JUCE offre una vasta gamma di strumenti e classi per la gestione delle operazioni audio. Grazie all'interfaccia di alto livello fornita da JUCE, gli sviluppatori possono concentrarsi sulla logica di elaborazione audio, utilizzando gli strumenti della libreria per semplificare le operazioni tecniche come la gestione dei buffer audio, l'elaborazione in tempo reale e l'interfacciamento con le librerie di basso livello dell'hardware audio.

Un'altra caratteristica di JUCE è il supporto per la creazione di interfacce utente grafiche personalizzate per le applicazioni audio. La libreria offre una vasta gamma di componenti grafici e strumenti che consentono agli sviluppatori di creare interfacce utente intuitive e accattivanti per le loro applicazioni audio.

JUCE gode anche di una documentazione completa e di un'attiva community di sviluppatori. La documentazione fornisce spiegazioni dettagliate sulle funzionalità di JUCE, esempi di codice e guide di riferimento, semplificando l'apprendimento e l'utilizzo della libreria. Inoltre, la community di JUCE è coinvolta e collaborativa,

offrendo supporto, suggerimenti e risorse agli sviluppatori che utilizzano JUCE nei loro progetti.

L'organizzazione del codice e le funzionalità di JUCE verranno discusse più approfonditamente nei capitoli successivi.

2.1.1 Ambiente di sviluppo

Projucer è un'applicazione compresa nel framework JUCE che gestisce le dipendenze dei progetti JUCE, gestisce la creazione, la modifica, e la cancellazioni di nuovi file di progetto, e di impostare il tipo di applicazione desiderata (plugin, synth, ...).

Figura 2.1: Il Projucer

L'IDE (*integrated development environment*) usata nel caso del synth "leoSynth" è *Xcode* per MacOS, mentre per compilare il progetto su Windows è stato utilizzato Visual Studio 2022.

2.1.2 Perché C++?

C++ è un linguaggio di programmazione multi-paradigma estremamente flessibile, ed è un linguaggio molto utilizzato nello sviluppo di applicazioni audio. E' un linguaggio *compilato*, ovvero i programmi vengono generati da un programma esterno chiamato *compilatore* che traduce le istruzioni scritte in C++ dal programmatore in linguaggio macchina.

Oltre ai linguaggi *compilati* (come C++ o C), esistono linguaggi *interpretati*, come Python o Java, linguaggi con un livello più alto di astrazione. Tuttavia, possono essere inefficienti nel caso di operazioni che richiedono tempi di elaborazione molto brevi (come nel caso di una routine di sintesi audio) [26].

La scelta di utilizzare C++ e JUCE per la creazione di plugin audio è motivata da diversi fattori.

Innanzitutto, il linguaggio di programmazione C++ offre notevoli vantaggi in termini di prestazioni e controllo. Grazie alla sua natura a basso livello di astrazione, C++ consente di ottimizzare l'elaborazione audio in tempo reale, riducendo al minimo la latenza e garantendo una risposta veloce. Inoltre, consente un accesso diretto alla memoria, fornendo un controllo dettagliato sulle operazioni audio.

JUCE, una libreria C++ ampiamente utilizzata nel campo dell'audio professionale, offre un'ampia gamma di funzionalità che semplificano lo sviluppo di plugin audio. Essa fornisce un'astrazione di alto livello per gestire operazioni comuni come la gestione dei buffer audio e l'interfacciamento con l'hardware audio a basso livello. Grazie a JUCE, gli sviluppatori possono scrivere codice una volta sola e renderlo eseguibile su diverse piattaforme come Windows, macOS e Linux, facilitando il supporto multipiattaforma.

La combinazione di C++ e JUCE offre quindi numerosi vantaggi. Gli sviluppatori possono creare plugin audio ad alte prestazioni, ottimizzati per l'elaborazione in tempo reale e con un controllo preciso sui dettagli implementativi. Inoltre, JUCE semplifica lo sviluppo fornendo un'interfaccia coerente e strumenti per la creazione di interfacce utente grafiche personalizzate.

Nel linguaggio C++, si distinguono tre diversi tipi di file:

- *Source Files*: sono file di testo contenente codice sorgente in C++, che verranno poi utilizzati per *compilare* il programma.
- *Headers*: sono file che contengono definizioni di parti di codice che verranno poi implementate successivamente nei Source Files: questo rende più leggibile il codice e, se la progettazione è coerente in tutte le parti del programma, permette di risparmiare errori impedendo al programmatore di modificare la struttura delle classi durante lo sviluppo.
- *Librerie*: sono dei file contenenti funzioni, definizioni di strutture dati, etc. che vengono aggiunti al programma nella fase chiamata *linking*, e vengono utilizzati perché, se già presenti nel linguaggio/framework, permettono di risparmiare tempo nella stesura del codice: per esempio, nel framework JUCE, sono già presenti librerie per la manipolazione audio e per la creazione di interfacce grafiche.

2.2 Design Pattern MVC

Un design dattern è una strategia con cui organizzare il codice di un programma (tipicamente ad oggetti). E' fondamentale da decidere in fase di progettazione quali pattern implementare in un qualsiasi progetto per garantire leggibilità, accelerare lo sviluppo stesso, e garantire maggior durabilità nel tempo.

Per il progetto realizzato, si è deciso di adottare il design pattern *Model-View-Controller (MVC)* [27]

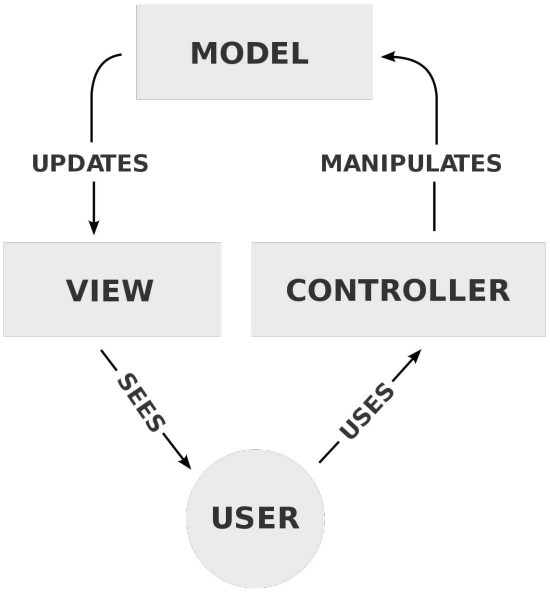

Figura 2.2: MVC: Modello, Vista e Controller

Il MVC è uno dei design pattern più utilizzati, ed è basato su 3 moduli funzionali interagenti fra di loro:

1. *Model*: le componenti del modello si occupano di mantenere informazioni e metodi sui dati, indipendentemente dall'interfaccia utente.

2. *View*: A partire dai dati immagazziati nel *model*, la *view* sceglie come questi dati devono essere visualizzati dal punto di vista dell'utente.

3. *Controller*: è il "cervello" dell'applicazione, accetta gli input degli utenti e li converte in comandi da elaborare per poi indirizzarli verso il Model o le View.

L'obiettivo principale di MVC è separare le responsabilità e promuovere la modularità e la riusabilità del codice. Ogni componente ha un ruolo specifico e ben definito, il che rende più facile la manutenzione, l'estensione e la comprensione dell'applicazione.

Inoltre, l'uso di MVC favorisce la scalabilità dell'applicazione. Poiché il modello, la vista e il controllore sono separati, è possibile modificare o sostituire uno di essi senza influire sugli altri componenti. Ad esempio, è possibile aggiungere nuove viste o modelli senza dover riscrivere tutto il sistema.

La scelta del MVC per questo tipo di progetto, quindi, permette di dividere così il *leoSynth*: nel *model* vi sono i vari componenti del sintetizzatore (oscillatori, filtri, ecc...), il loro funzionamento, le loro caratteristiche e i loro metodi, nelle *view* verranno descritte le modalità con cui questi componenti verranno visualizzati a schermo e come l'utente potrà interagire con essi, e nel *controller* compariranno tutte le funzioni relative al routing del segnale audio generato, alla connessione fra i

model e le *view* e all'impostazione ed all'aggiornamento di parametri fondamentali (modificabili dall'utente con l'interazione con la *view*).

2.3 VST

VST (Virtual Studio Technology) è un protocollo sviluppato da Steinberg e rilasciato nel 1996, nato come metodo per aggiungere effetti audio in real-time a tracce audio registrate digitalmente.

Il formato diventò rapidamente popolare data la sua natura *open*, per cui altre aziende cominciarono a vedere il suo potenziale e sviluppare software VST.

I plugin VST sono componenti software che possono essere integrati all'interno di un'applicazione audio esistente, consentendo di aggiungere nuove funzionalità e effetti audio. I plugin VST possono essere di diversi tipi, come effetti audio (come riverberi, compressori, equalizzatori, ecc.) o strumenti virtuali (come sintetizzatori, campionatori, drum machine, ecc.).

La tecnologia VST opera attraverso un'interfaccia standardizzata che consente ai plugin di comunicare con l'applicazione ospitante. I plugin VST ricevono l'audio di input dall'applicazione, elaborano il segnale audio in base ai loro algoritmi specifici e restituiscono il segnale di output elaborato all'applicazione ospitante. In questo modo, i plugin VST possono essere utilizzati per aggiungere effetti, modulazioni e altre elaborazioni audio al suono generato all'interno dell'applicazione.

Un aspetto importante della tecnologia VST è la sua portabilità e compatibilità tra diverse piattaforme. Gli sviluppatori possono creare plugin VST utilizzando diversi linguaggi di programmazione, come C++ o Java, e possono essere eseguiti su sistemi operativi come Windows e macOS. Ciò permette agli utenti di utilizzare i plugin VST su diverse piattaforme e all'interno di diverse DAW senza dover ricomprare o reinstallare i plugin per ogni ambiente.

Inoltre, l'ecosistema dei plugin VST è molto vasto e diversificato, con un'ampia gamma di plugin disponibili sviluppati da diverse aziende e sviluppatori indipendenti. Ciò offre agli utenti una vasta scelta di strumenti e effetti audio per arricchire le proprie produzioni musicali. I plugin VST vengono generalmente aperti in una DAW (Digital Audio Workstation) come Ableton, Cubase, FL Studio.

Esistono tre tipi di VST:

1. *VST instruments*: generano audio, possono essere o sintetizzatori virtuali o campionatori virtuali. Alcuni di questi riproducono il suono e le scelte tecnologiche di sintetizzatori hardware più famosi.

2. *VST effects*: processano invece di generare audio, svolgendo la stessa funzione di processori audio hardware (come un reverb).

3. *VST MIDI effects*: processano i messaggi MIDI e indirizzano le informazioni MIDI ad altri strumenti o plugin.

2.4 Definizione delle specifiche

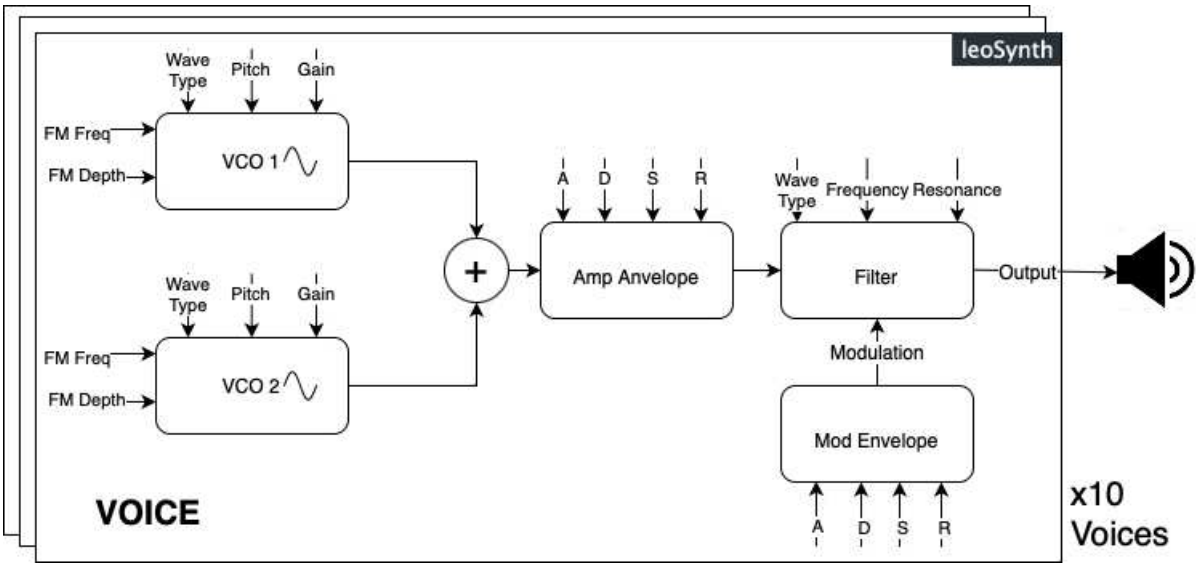

Figura 2.3: leoSynth: Block Diagram

Il sintetizzatore sarà dotato di due oscillatori indipendenti, ciascuno dei quali conterrà i seguenti controlli.

Il controllo *Wave Type* permette di selezionare la forma d'onda desiderata per ciascun oscillatore. Saranno disponibili le seguenti opzioni: sinusoidale, quadra, triangolare e a dente di sega.

La *FM Frequency* consente di regolare la frequenza di modulazione in frequenza (FM) per ciascun oscillatore. Congiuntamente, la *FM Depth* regola l'intensità dell'effetto di modulazione in frequenza (FM) per ciascun oscillatore. Determina quanto il segnale FM influisce sulla frequenza dell'oscillatore. Grazie a questo controllo e alla FM Frequency, è possibile creare un effetto vibrato, ovvero una variazione periodica più o meno grande della frequenza per ognuno degli oscillatori.

Il controllo di *Gain* regola l'ampiezza del segnale dell'oscillatore. Permette di controllare il livello di uscita del segnale audio generato dall'oscillatore.

Il *Pitch* consente di regolare la tonalità o l'altezza del suono prodotto dall'oscillatore. Modificando il valore di Pitch, è possibile controllare la frequenza dell'oscillatore in modo preciso o tramite l'input da tastiera MIDI. I due oscillatori potranno così essere impostati in modo che, dato lo stesso input da tastiera, genereranno due onde sonore a frequenze diverse, in funzione di questo controllo: in questo modo, sarà possibile creare degli effetti armonici dipendenti dal rapporto fra le altezze delle note generate dai due oscillatori, sfruttando per esempio particolari intervalli musicali. Oppure, creando una minima differenza relativa sul pitch dei due oscillatori, si potrà arricchire il suono sfruttando il fenomeno dei battimenti: questo è un effetto acustico che si verifica quando due suoni con frequenze simili, ma leggermente diverse, vengono

riprodotti simultaneamente. I battimenti sono percepiti come variazioni periodiche nell'intensità del suono.

Il sintetizzatore sarà poi dotato di un envelope ADSR per il controllo dell'ampiezza del suono. Saranno disponibili i parametri Attack, Decay, Sustain e Release per modulare l'inviluppo dell'ampiezza del suono nel tempo. Ciò consentirà di creare variazioni dinamiche nel volume del suono in risposta agli input dell'utente.

Il sintetizzatore includerà un filtro con i seguenti parametri: *filter type*, *filter freq* e *filter res*. Saranno disponibili le tipologie più comuni di filtro: passa-basso, passa-alto e passa-banda, per modulare lo spettro del suono. Il parametro *filter freq* permetterà di regolare la frequenza di taglio del filtro, mentre il parametro *filter res* consentirà di regolare la risonanza (fattore Q) del filtro.

Sarà presente anche un envelope ADSR di modulazione per modulare la frequenza di cut-off del filtro. Saranno disponibili i parametri Attack, Decay, Sustain e Release per modulare l'inviluppo della modulazione nel tempo. Ciò consentirà di creare variazioni timbriche complesse e dinamiche nel suono.

Si è scelto di utilizzare due oscillatori per i motivi spiegati precedentemente nel capitolo 1.2.1: due oscillatori costituiscono una quantità sufficiente a garantire una minima complessità sonora, garantendo anche una tranquillità per quanto riguarda l'occupazione di risorse da parte del plugin.

Il percorso del suono sarà nell'ordine con cui son stati presentati i componenti: una volta ricevuto in input un segnale da una tastiera (o da un'altro controller), i due oscillatori genereranno una forma d'onda dipendentemente dall'input e dai parametri impostati nell'interfaccia utente. Dopo di questo, i segnali verranno mixati e la loro intensità sarà regolata dall'Amp Envelope. Dopodiché, sarà il filtro selezionato a modificare il timbro del suono, dipendentemente dai parametri frequenza, risonanza e ai valori del Mod Envelope.

Rispetto ad altri sintetizzatori, è un esempio molto basilare di come dei semplici componenti possono interagire fra di loro: il filtro non ha parametri particolari oltre ai più classici e gli oscillatori generano le forme d'onda più basilari senza una grande scelta timbrica, ma la possibilità di regolare una modulazione di frequenza ad hoc per ogni oscillatore unita alla possibilità di regolare gli inviluppi dell'intensità del suono e della frequenza di taglio riuscirà a restituire un'esperienza che permetterà di realizzare suoni di una complessità non troppo banale.

L'interfaccia del sintetizzatore sarà costituita da 5 moduli funzionali: uno per ogni oscillatore, uno per l'Amp Envelope, uno per il filtro, e uno per il Mod Envelope. Tutti i controlli saranno delle manopole, degli Slider o dei menu a cascata (nel caso della scelta del tipo di onda o tipo di filtro).

Il sintetizzatore sarà composto da 10 voci, sarà quindi possibile generare 10 note contemporaneamente, ognuna controllata dagli stessi parametri degli oscillatori, del filtro e dei due envelope. Son stati scelte 10 voci perché, dato che il sintetizzatore è stato pensato per essere suonato in tempo reale da musicisti, con 10 voci si garantisce il massimo di voci nel massimo in cui tutte e 10 le dita stiano suonando

contemporaneamente. In questo senso, si può dire che il sintetizzatore sarà polifonico a 10 voci.

I filtri implementati sono di tipo passa-basso, passa-alto e passa-banda perché sono i tipi implementati di base all'interno del framework Juce.

Le forme d'onda scelte (sinusoide, quadra, dente di sega, triangolare) sono le onde più utilizzate nella sintesi sonora a causa delle loro caratteristiche distintive e delle possibilità timbriche che offrono.

La sinusoide è la forma d'onda più semplice e pura, caratterizzata da un singolo picco che si ripete in modo regolare nel tempo. Essa produce un suono puro e privo di armoniche aggiuntive, risultando ideale per generare toni fondamentali o suoni di base.

L'onda quadra è caratterizzata da una forma d'onda con un ciclo di attacco repentino e un ciclo di rilascio repentino, creando una serie di armoniche dispari nella sua spettro. Questo tipo di forma d'onda è ampiamente utilizzato per generare suoni ricchi di armoniche e con un carattere più aggressivo, come suoni di basso, suoni di sintetizzatori e suoni percussivi.

La forma d'onda triangolare ha un profilo che si sviluppa linearmente dal picco al punto di inversione e poi ritorna al picco successivo. Essa produce una serie di armoniche con una diminuzione lineare dell'intensità, fornendo suoni dolci e morbidi. La triangolare è spesso utilizzata per suoni di pad, suoni di chitarra o suoni che richiedono una componente armonica ben definita.

Il dente di sega, come suggerisce il nome, ha una forma d'onda che ricorda i denti di una sega. Essa contiene un insieme di armoniche che diminuiscono in intensità man mano che ci si sposta verso le frequenze più alte. Il dente di sega viene spesso utilizzato per creare suoni brillanti, ricchi di armoniche e con una sensazione di movimento ascendente o discendente.

L'utilizzo combinato di queste forme d'onda all'interno di un sintetizzatore offre un'ampia gamma di opzioni timbriche per gli utenti medi. Consentono di creare suoni musicalmente interessanti, adattabili a diversi generi musicali e contesti compositivi. La varietà di spettri timbrici generati da queste forme d'onda consente di esprimere creatività e sperimentare nella progettazione dei suoni, offrendo agli utenti la possibilità di modellare il suono in base alle proprie preferenze musicali e alle esigenze del brano.

2.5 Confronto con altre architetture

2.5.1 Korg Minilogue XD

Come esempio, viene di seguito visualizzata l'interfaccia di funzionamento di un KORG Minilogue XD e il suo relativo schema a blocchi, un sintetizzatore analogico molto comune presentato dall'azienda giapponese KORG nel 2016.

Figura 2.4: KORG Minilogue XD

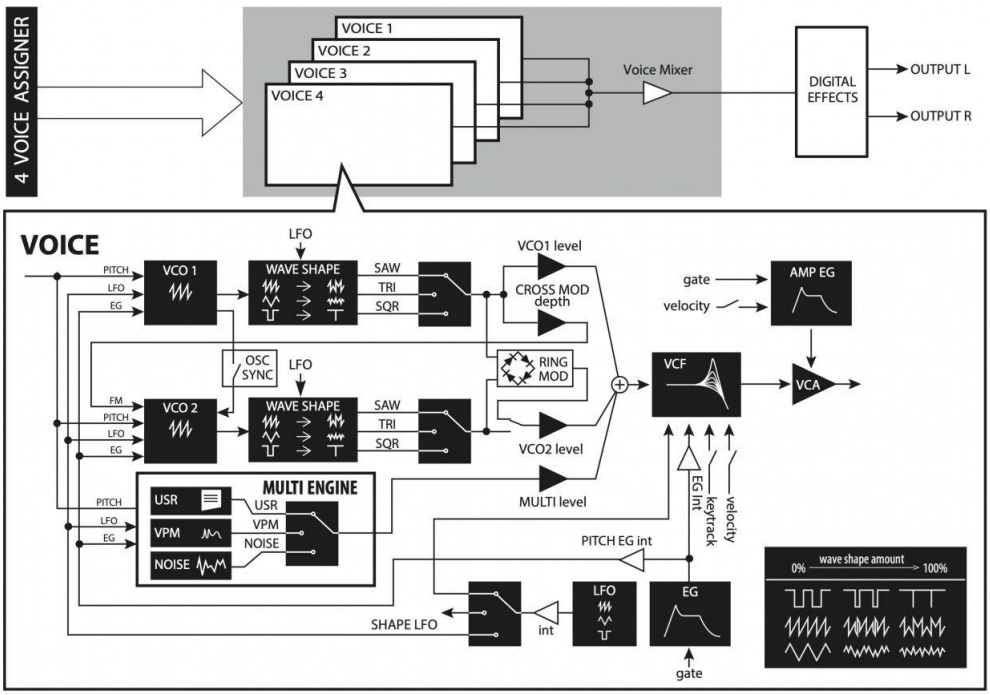

Figura 2.5: KORG Minilogue Block Diagram [28]

Il sintetizzatore è diviso in quattro sezioni: i *controlli master*, i *controlli di sintesi*, gli *effetti*, e il *sequencer*.

Con *controlli master* si intendono parametri ed impostazioni "globali" della tastiera: il tempo, il volume Master, il portamento, l'ottava (per la tastiera) e il modulation stick (assegnabile a vari parametri a scelta).

Nella sezione dei *controlli di sintesi* sono presenti gli oscillatori: due analogici e uno digitale, a differenza del leoSynth che ha 2 oscillatori digitali. Per ognuno di essi è possibile modificare la forma d'onda, l'ottava, il pitch, ecc. Poi, c'è una banda per "'mixare'" gli oscillatori, ovvero regolare i loro volumi, singolarmente. I due oscillatori analogici possono anche essere utilizzati insieme alla ring modulation: è una tecnica di modulazione audio che coinvolge due segnali di ingresso. Questa tecnica si basa sulla moltiplicazione di due segnali per creare nuove componenti di frequenza che non sono presenti nei segnali di ingresso originali.

Nella ring modulation, il segnale di ingresso principale (carrier) e il segnale di modulazione (modulator) vengono moltiplicati tra loro campione per campione. Il risultato di questa moltiplicazione è un segnale di uscita che contiene le frequenze somma e differenza dei segnali di ingresso.

La ring modulation genera una serie di frequenze armoniche che sono un multiplo delle frequenze del segnale di ingresso. Se il segnale di modulazione ha una frequenza fissa, il risultato sarà una serie di picchi spettrali a frequenze multiple della frequenza del segnale carrier. Se il segnale di modulazione ha una frequenza variabile, si otterranno modulazioni di frequenza.

L'effetto sonoro risultante dalla ring modulation è spesso descritto come "metallico" o "robotico" a causa delle nuove componenti di frequenza che vengono generate.

Dopodiché, c'è il filtro di tipo *low-pass*, con i controlli per la frequenza di *cutoff* e per la *resonance*. E' possibile anche controllare il *drive* (un'amplificazione post-filtro) e il livello di *key-tracking* (attivare il *key-track* vuol dire rendere proporzionale il rapporto tra pitch e frequenza di *cutoff*). Poi, è presente la parte relativa ai *EG* (Envelope Generators: generatori di inviluppo), che permettono di controllare contemporaneamente il volume e uno a scelta tra il *pitch* degli oscillatori analogici e la frequenza di cutoff del filtro. Infine, è presente anche un *LFO* applicabile al *pitch* dell'oscillatore digitale, la forma d'onda o la frequenza di cutoff del filtro.

A differenza del leoSynth, nel Minilogue XD è presente soltanto un filtro passa-basso mentre, esattamente come nel leoSynth, sono presenti due generatori di inviluppo per controllare rispettivamente VCA e il filtro.

Gli *effetti* presenti (digitali) sono: *modulation*, *reverb* e *delay*, utilizzabili in modo esclusivo fra di loro e regolabili tramite due controlli *time* e *depth*.

Il *sequencer* è un dispositivo che permette di registrare dati di automazione per registrare informazioni sull'audio da riprodurre (note, velocity, e qualsiasi altro parametro). Nel caso del *Korg Minilogue*, il *sequencer* permette di salvare informazioni su 16 "step", salvando per ognuno l'automazione di 4 manopole a scelta, permettendo quindi di realizzare suoni con un timbro nettamente diverso per ogni step [28].

2.5.2 Full Bucket Music Kern

Il *Full Bucket Music Kern* è un VST gratuito creato per essere completamente controllato da tastiere MIDI, esattamente come il leoSynth. L'interfaccia è molto semplice, ed è molto simile a quella del leoSynth.

Figura 2.6: Full Bucket Music Kern

Il *Kern* ha due oscillatori che possono generare onde quadre o a dente di sega, ma la forma d'onda selezionata è la stessa per entrambi gli oscillatori. L'oscillatore 2 può essere trasposto di ±24 note, oppure, con una sensibilità maggiore di ±1 nota.

La frequenza degli oscillatori può essere modulata dall'LFO. La modulazione di frequenza di entrambi gli oscillatori da parte dell'LFO (*vibrato*) può sempre essere applicata grazie alla *modulation wheel*.

C'è anche un controllo per il *portamento*, ovvero la capacità del suono di spostarsi da una nota all'altra in modo graduale, senza salti repentini. È anche noto come "*glide*" o "*slide*". Il portamento viene spesso utilizzato per creare effetti melodici e per ottenere una transizione fluida tra le note.

Il filtro si basa su un Design a Zero-Delay Feedback e fornisce due modalità: Smooth (un passa-basso a 4 poli con non linearità moderate e potenziale auto-oscillazione) e Dirty, un passa-basso a 2 poli con potenziale ma nessuna auto-oscillazione. Ci sono poi i controlli per modificare *cut-off* e *resonance*. La frequenza di taglio del filtro può essere modulata simultaneamente ed entrambe positivamente o negativamente da quattro fonti: inviluppo del filtro, LFO, *key-track* e *velocity*. Con *key-track* si intende la relazione tra l'altezza delle note suonate sulla tastiera e un parametro specifico del sintetizzatore, in questo caso appunto la frequenza di taglio: in questo modo, questa sarà dipendente dal tasto premuto sulla tastiera. Con *velocity*, invece, si intende la forza con cui viene premuta la nota sulla tastiera.

L'amplificatore offre solo i parametri *Volume* e *Velocity*; quest'ultimo controlla il influenza della velocità sul volume di uscita.

In questo caso è presente anche un effetto *Chorus* che può essere attivato o disattivato. Inoltre è possibile impostare la velocità dei due LFO triangolari che modulano il *Chorus* così come la profondità di modulazione [29].

Rispetto al *leoSynth*, quindi, questo sintetizzatore ha lo stesso numero di oscillatori ma un diverso tipo di controllo su di essi: le forme d'onda hanno una limitata varietà, ma è possibile controllare con maggior accuratezza il rapporto tra le frequenze dei due oscillatori. Il filtro è solo passa-basso, ma è presente qui un LFO per controllare sia la frequenza di cut-off, sia la frequenza degli oscillatori. La parte relativa all'Amplifier è sostanzialmente identica, nonostante la mancanza di un controllo incrementale sul *Sustain*, così come la limitata implementazione dell'inviluppo per il filtro (dotato solo di controlli per *Attack*, *Decay* e switch per il *Sustain*), ma è stato implementato un basilare effetto Chorus, assente nel *leoSynth*.

Capitolo 3

leoSynth

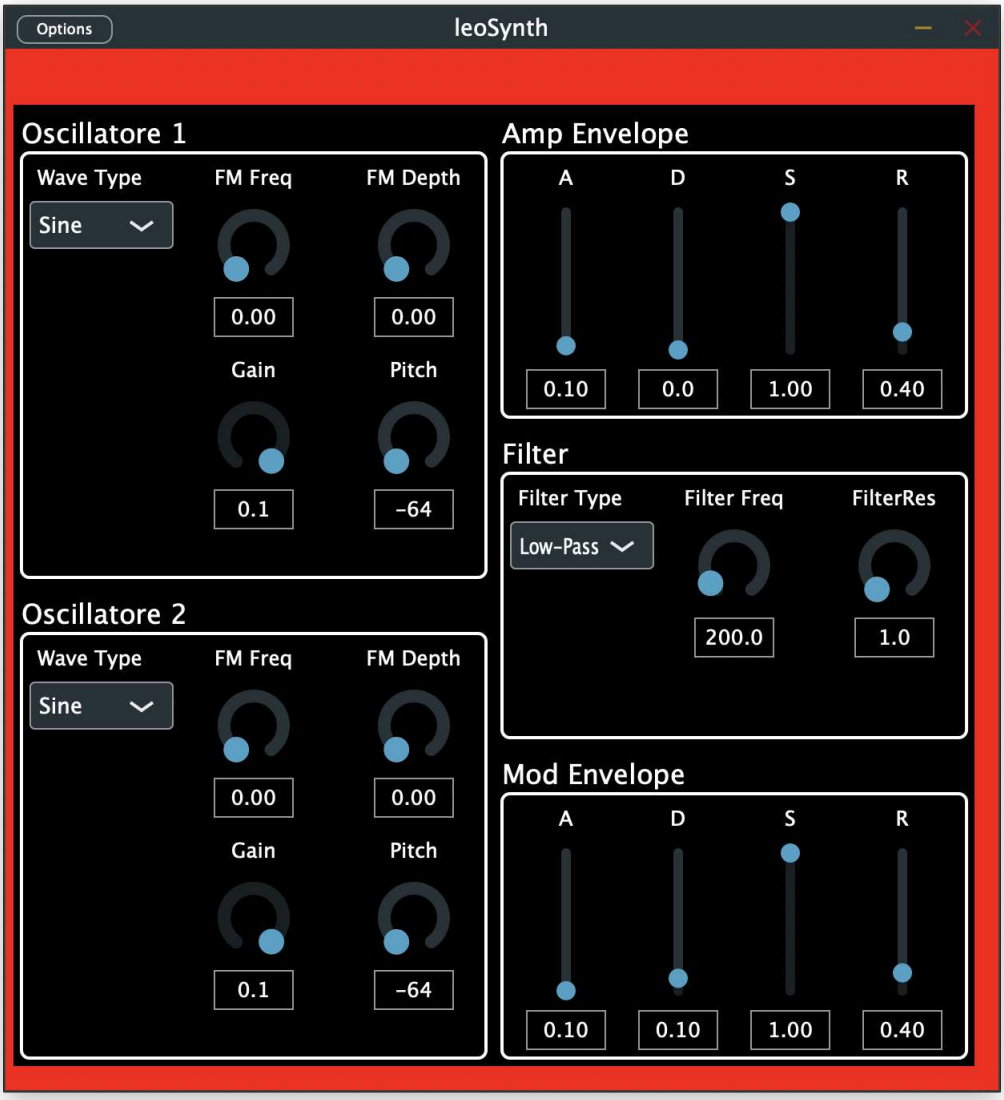

Figura 3.1: Interfaccia del leoSynth

3.1 Dettagli tecnici dei componenti utilizzati

3.1.1 Juce: Oscillatore

JUCE offre diverse funzionalità per la creazione di oscillatori che utilizzano varie tecniche. Queste funzionalità consentono di creare oscillatori personalizzati e complessi, ad esempio JUCE permette di generare *WaveTable* personalizzate o di utilizzare *WaveTable* predefinite da file audio o librerie esistenti.

Inoltre, JUCE offre un'ampia gamma di funzionalità DSP per manipolare il segnale audio. È possibile utilizzare queste funzioni per implementare oscillatori più complessi, come oscillatori a modulazione di frequenza (FM), oscillatori granulari o oscillatori con algoritmi di sintesi additiva.

JUCE consente anche di generare il segnale dell'oscillatore in modo dinamico durante l'esecuzione dell'applicazione. Questo può essere utile per creare oscillatori con forme d'onda personalizzate o generare forme d'onda in base a parametri dinamici.

Fornisce inoltre funzionalità di interpolazione per migliorare la qualità e la fluidità dell'oscillatore. È possibile utilizzare le funzioni di interpolazione di JUCE per interpolare i campioni audio durante la generazione del segnale, garantendo una transizione regolare tra i valori di campionamento.

Nel progetto realizzato, gli oscillatori funzionano con una generazione dinamica del segnale, cioè tramite una funzione definita dall'utente (o dal framework stesso) che elaborano il sample successivo ad ogni istante di tempo. Il sample rate predefinito in Juce è 44.1kHz, modificabile a piacere dallo sviluppatore.

Gli oscillatori sono implementati come classi derivate dalla classe base juce :: Oscillator . Questa classe base fornisce un'interfaccia comune per tutti gli oscillatori, definendo metodi per l'accesso e la modifica dei parametri dell'oscillatore, come la frequenza, l'ampiezza e la forma d'onda.

Per utilizzare un oscillatore in JUCE, è necessario creare un'istanza di una classe derivata da juce :: Oscillator, come ad esempio juce :: SineWave. Ad esempio:

```
juce::SineWave oscillator;
oscillator.setFrequency(440.0); // Imposta la frequenza a 440 Hz
oscillator.setAmplitude(0.5); // Imposta l'ampiezza a 0.5
                                (intervallo 0-1)

// Genera un campione audio utilizzando l'oscillatore
float sample = oscillator.getNextSample();
```

Nell'esempio sopra, viene creato un oscillatore sinusoidale (juce :: SineWave) e vengono impostati alcuni parametri come la frequenza e l'ampiezza. Successivamente, viene generato un campione audio utilizzando il metodo getNextSample(), che restituisce il valore del prossimo campione dell'oscillatore.

JUCE fornisce anche altre classi di oscillatori, come juce :: SquareWave, juce :: SawtoothWave, juce :: TriangleWave e così via, che possono essere utilizzate in modo simile.

In particolare, all'interno di questo progetto, è stata creata una funzione in grado di cambiare la forma d'onda da generare a seconda della scelta effettuata nel menù a tendina nell'interfaccia utente.

```
void OscData::setWaveType (const int choice)
{
    switch (choice)
    {
        case 0:
            // Sine
            initialise ([](float x) { return std::sin (x); });
            break;

        case 1:
            // Saw wave
            initialise ([](float x) {
            return x / juce::MathConstants<float>::pi; });
            break;

        case 2:
            // Square wave
            initialise ([](float x)
            { return x < 0.0f ? -1.0f : 1.0f; });
            break;
    }
}
```

Per evitare l'aliasing, è stata utilizzata la tecnica dell'oversampling.

Per implementare l'oversampling, si crea un oggetto dsp::Oversampling specificando il fattore di oversampling desiderato.

Prima di iniziare l'elaborazione del segnale, si inizializza l'oggetto dsp::Oversampling chiamando il metodo prepare(), passando il sample rate di 44.1kHz come parametro per indicare la frequenza di campionamento iniziale.

Durante il ciclo di elaborazione del segnale, si applica l'oversampling utilizzando il metodo dsp::Oversampling::processSamplesUp(). Questo aumenterà la frequenza di campionamento e consentirà di elaborare il segnale a una frequenza più alta. Successivamente, si possono applicare le operazioni di elaborazione desiderate ai campioni.

Dopo aver completato l'elaborazione a una frequenza superiore, si riduce nuovamente la frequenza di campionamento al sample rate di 44.1kHz utilizzando il metodo dsp::Oversampling::processSamplesDown(). Questo processo applica un filtro anti-aliasing interno e riduce la frequenza di campionamento al valore desiderato.

Si ripetono questi passaggi per ogni blocco di campioni da elaborare nel tuo segnale audio.

L'implementazione dell'oversampling come tecnica di anti-aliasing potrebbe causare alcuni problemi prestazionali, come l'aumento della complessità computazionale (dovuta alla necessità di elaborare un numero maggiore di campioni al secondo), o il

maggior utilizzo di memoria (a causa dell'allocazione di buffer audio più grandi per gestire i campioni a una frequenza di campionamento più alta). Tuttavia, testando il software, probabilmente grazie alla semplicità, non sembra dare alcun tipo di problema durante l'utilizzo.

Il *morphing* è un termine utilizzato nell'ambito della sintesi sonora per descrivere il processo di transizione fluida e graduale tra due o più forme d'onda o suoni. In sostanza, si tratta di combinare due o più suoni in modo tale che si fondano insieme in modo continuo, creando un suono ibrido che contiene caratteristiche di entrambe le fonti sonore.

Durante il morphing, i parametri delle forme d'onda originali vengono modificati gradualmente nel tempo per passare da una forma d'onda all'altra. Questo può includere il cambiamento di ampiezza, frequenza, forma, fase e altre caratteristiche sonore. Il risultato è una transizione morbida e controllata tra i suoni di partenza, creando un suono unico e interessante.

In JUCE, è possibile effettuare il morphing (o crossfading) tra le forme d'onda utilizzando diverse tecniche e componenti offerti dalla libreria. Ciò può essere realizzato mediante l'utilizzo di sintesi wavetable, interpolazione e l'uso di tavole di forma d'onda. È possibile creare tavole di forma d'onda campionando i valori desiderati e utilizzare l'interpolazione per transizionare gradualmente tra le forme d'onda desiderate. JUCE offre classi come dsp::LookupTable e componenti come dsp:: Oscillator che consentono di gestire e generare tavole di forma d'onda e controllare i parametri di morphing. In questo modo, è possibile ottenere transizioni fluide e controllate tra le diverse forme d'onda all'interno di un progetto audio.

3.1.2 Juce: ADSR

All'interno del framework JUCE, l'ADSR (Attack, Decay, Sustain, Release) è implementato tramite la classe juce :: ADSR. Questa classe fornisce un envelope generator completo con le fasi di attacco, decay, sustain e release, che può essere utilizzato per modulare i parametri audio.

Si possono utilizzare i metodi noteOn e noteOff per attivare e disattivare l'envelope. Quando noteOn viene chiamato, l'envelope inizia la fase di attacco e passa alle fasi di decay e sustain. Quando noteOff viene chiamato, l'envelope passa alla fase di release.

Durante l'esecuzione dell'audio, puoi ottenere il valore corrente dell'envelope chiamando il metodo getNextSample.

Il valore restituito rappresenta l'ampiezza dell'envelope nel range da 0.0 a 1.0, che può essere utilizzato per modulare i parametri audio come il volume o altri parametri desiderati.

L'utilizzo della classe juce :: ADSR semplifica l'implementazione e il controllo degli envelope generator ADSR all'interno del tuo progetto JUCE. Puoi facilmente configurare i tempi e i livelli desiderati e utilizzare i valori dell'envelope per modulare i parametri audio in modo dinamico.

Gli inviluppi all'interno del framework JUCE possono essere sia lineari che esponenziali, a seconda delle impostazioni specificate. La classe juce::ADSR supporta entrambi i tipi di inviluppo.

Per impostazione predefinita, l'inviluppo Attack e Release di Juce::ADSR sono esponenziali, mentre l'inviluppo Decay è lineare. Tuttavia, si possono eventualmente configurare gli inviluppi per essere lineari o esponenziali utilizzando i metodi appropriati. Per il progetto del *leoSynth* sono utilizzati i tipi di inviluppo predefiniti, nel modo appena descritto.

I limiti di range per i tempi e i livelli dell'inviluppo dipendono dalla configurazione specifica dell'envelope generator e possono essere personalizzati a seconda delle esigenze. Nel *leoSynth*, si utilizzano dei range personalizzati scelti in base all'esperienza di utilizzo. Una volta definiti i range per ogni parametro, vengono normalizzati grazie alla struttura juce::NormalisableRange<**float**>, una classe che rappresenta un mapping tra un intervallo arbitrario di valori e un intervallo $[0:1]$ normalizzato. Per esempio, per l'attack:

```
params.push_back
  (std::make_unique
    <juce::AudioParameterFloat>
        ("ATTACK", "Attack",
            juce::NormalisableRange<float>
                { 0.01f, 3.0f, 0.01f }, 0.1f));
```

Quest'istruzione crea un range normalizzato da 0 a 1 di valori a partire dall'intervallo $[0.01 : 3.0]$, con una sensibilità di 0.01 e uno *skew factor* di 0.1. lo skew factor è un parametro aggiuntivo che può essere utilizzato per modificare la forma dell'inviluppo. Lo skew factor regola la simmetria tra la fase di attacco e la fase di decadimento dell'inviluppo.

Normalmente, senza l'uso dello skew factor, l'inviluppo ADSR ha una forma simmetrica in cui la durata dell'attacco è uguale alla durata del decadimento. Tuttavia, utilizzando lo skew factor, si può modificare questa simmetria per ottenere una forma personalizzata dell'inviluppo.

Lo *skew factor* può assumere valori compresi tra -1.0 e 1.0. Quando lo *skew factor* è impostato su -1.0, l'inviluppo presenta un attack molto rapido e un decade più lento. Al contrario, quando lo *skew factor* è impostato su 1.0, l'*attack* è più lento e il *decade* è più rapido. Quando lo *skew factor* è impostato su 0.0 (valore predefinito), l'inviluppo è simmetrico.

3.1.3 Juce: Filtri

All'interno della classe dsp:: StateVariableFilter :: Filter di JUCE, i filtri sono realizzati utilizzando una struttura IIR (Infinite Impulse Response) a variabile di stato (*State Variable Filter (SVF)*. Questa struttura permette di eseguire il filtraggio basso, banda e passa-alto su un segnale audio, con un'attenuazione di $12dB$ per ottava.

La struttura TPT è utilizzata per implementare il filtro. Questa struttura, progettata per una modulazione veloce, offre una risposta in frequenza precisa e stabile.

Il filtro utilizza la seguente equazione alle differenze:

```
auto& ls1 = s1[(size_t) channel];
auto& ls2 = s2[(size_t) channel];

auto yHP = h * (inputValue - ls1 * (g + R2) - ls2);

auto yBP = yHP * g + ls1;
ls1      = yHP * g + yBP;

auto yLP = yBP * g + ls2;
ls2      = yBP * g + yLP;
```

dove yBP, yHP e yLP sono le rispettive uscite bandpass, highpass, lowpass.

Per ottenere l'attenuazione di $12dB$ per ottava, i coefficienti del filtro vengono calcolati in base alla frequenza di taglio desiderata utilizzando le formule appropriate per il tipo di filtro (basso, banda o passa-alto).

La struttura TPT consente una modulazione veloce, il che significa che i coefficienti del filtro possono essere regolati rapidamente senza introdurre discontinuità o artefatti indesiderati nel segnale audio [30].

Complessivamente, la classe dsp:: StateVariableFilter :: Filter di JUCE implementa un filtro IIR con una struttura TPT per eseguire il filtraggio basso, banda e passa-alto su un segnale audio, con $12dB$ di attenuazione per ottava e modulazione veloce.

3.2 Organizzazione del codice

Come già detto, il codice è organizzato secondo il pattern MVC. Nella root folder dell'applicazione è presente il *controller*, ovvero il file `PluginProcessor`.

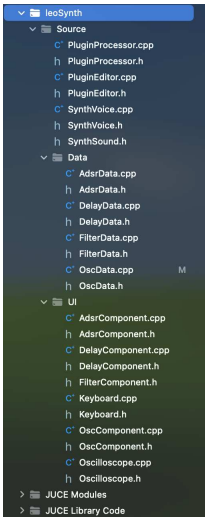

Figura 3.2: Controller, View, Data (Model)

Oltre al controller, sono presenti altri file nella root folder:

1. *PluginEditor*: Di fatto è il *view* del programma, imposta l'interfaccia grafica dal livello d'astrazione più alto possibile, ma, nel farlo, utilizza componenti che vengono dichiarati e configurati nella cartella *UI*. Di fatto, quindi, questo file serve da *"aggregatore"* di tutti i componenti visuali del programma.

2. `SynthSound`: E' una classe che rappresenta il suono prodotto dal sintetizzatore, ma, di fatto, il rendering audio viene poi fatto dalla classe SynthVoice. In generale, un suono potrebbe essere composto da tante voci diverse fra loro (ognuna processata in modo differenze): nel caso del *leoSynth*, il suono sarà composto da dieci voci ma tutte dello stesso tipo: di conseguenza, `SynthSound` è una semplice eredità dalla classe `juce::SynthesiserSound` [31].

3. `SynthVoice`: Questa classe rappresenta una *voce* del sintetizzatore: effettua tutto il rendering relativo ad una singola voce: rappresenta ciò che il `juce::Synthesiser` può utilizzare per riprodurre un `juce::SynthesiserSound`. Una voce tiene un singolo suono alla volta, e il `Synthesiser` ha un'array di voci che possono suonare in modo polifonico.

3.3 Il Synth

Ad ogni componente del *leoSynth* è associato un file Data (*Model*), contenente proprietà e metodi di ogni componente, ed un file Component (*View*), che descrive

il modo in cui i componenti saranno visibili dall'utente. Quindi, la logica e il funzionamento del sintetizzatore sono descritti nal file *PluginProcessor*, che ha la funzione di "collegare" la parte visuale ai dati dei componenti.

Di seguito, verrà analizzato il procedimento logico alla base del funzionamento di tutti i componenti del programma, in questo caso per scegliere la forma d'onda di un oscillatore, al fine di illustrare il percorso funzionale e comprendere la scelta architetturale presa e le sue conseguenze.

3.3.1 Esempi di codice

Il file *header* contiene la dichiarazione di tutte le proprietà e metodi dell'oscillatore: eredita un oggetto `juce::dsp::Oscillator` dalle librerie *JUCE*, un `Gain`, l'ultima nota suonata e i parametri necessari per la modulazione FM. Tra i metodi, notiamo quello che aggiornerà la forma d'onda (`setWaveType`), che viene riportato più avanti.

```
class OscData : public juce::dsp::Oscillator<float>
{
public:
    void prepareToPlay (double sampleRate,
        int samplesPerBlock,
        int outputChannels);
    void setWaveType (const int choice);
    void setGain (const float levelInDecibels);
    void setPitch(int pitch);
    void setWaveFrequency (const int midiNoteNumber);
    void getNextAudioBlock (juce::dsp::AudioBlock<float>& block);
    void updateFm (const float freq, const float depth);
    float processNextSample(float input);

private:
    void processFmOsc (juce::dsp::AudioBlock<float>& block);
    juce::dsp::Oscillator<float> fmOsc {
        [](float x)
        { return std::sin (x); } };
    juce::dsp::Gain<float> gain;
    float fmMod { 0.0f };
    float fmDepth { 0.0f };
    int lastMidiNote { 0 };
};
```

La *view* ha, nell'*header*, la dichiarazione di tutti i vari componenti visuali che verranno visualizzati a schermo, con i relativi "Attachment" (connessione fra oggetti di input e variabili)

```
class OscComponent  : public juce::Component
{
public:

    OscComponent (juce::AudioProcessorValueTreeState& apvts,
        juce::String waveSelectorId,
```

```
        juce::String fmFreqId,
        juce::String fmDepthId,
        juce::String gainId,
        juce::String pitchId,
        juce::String oscName);
    ~OscComponent() override;
    void paint (juce::Graphics&) override;
    void resized() override;
    void setTitle(juce::String newTitle);
    juce::String getTitle();

private:
    juce::String title;
    juce::ComboBox oscWaveSelector;
    std::unique_ptr
        <juce::AudioProcessorValueTreeState::ComboBoxAttachment>
        oscWaveSelectorAttachment;
    juce::Slider fmFreqSlider;
    juce::Slider fmDepthSlider;
    juce::Slider gainSlider;
    juce::Slider pitchSlider;
   //etc...
   //...
};
```

Il *source code* della *view* contiene semplicemente istruzioni per impostare il layout del componente.

```
void OscComponent::paint (juce::Graphics& g)
{
    auto bounds = getLocalBounds().reduced (5);
         auto labelSpace = bounds.removeFromTop (25.0f);
         g.fillAll (juce::Colours::black);
         g.setColour (juce::Colours::white);
         g.setFont (20.0f);
         g.drawText (OscComponent::getTitle(),
            labelSpace.withX (5), juce::Justification::left);
         g.drawRoundedRectangle (bounds.toFloat(), 5.0f, 2.0f);
         //etc...
         //...
         //...
}
```

Nel *controller*, inizialmente vengono dichiarati le variabili modificabili direttamente dall'utente finale, che vengono tutti memorizzati in una struttura dati chiamata `AudioProcessorValueTreeState (apvts)`. Una volta inizializzato un parametro, identificandolo con un ID e assegnandogli un nome, è possibile accederci e leggere il valore regolato dall'utente (tramite l'`Attachment`), per poi passare il valore ai metodi del *Model* (in questo caso, `osc1.setWaveType(oscWaveChoice)`).

Quest'operazione viene svolta per ogni parametro di ogni componente di ogni voce. Alla fine, si effettua un aggiornamento di tutte le componenti del synth

(`adsr.updateADSR, filterAdsr.updateADSR` e `voice->updateFilter`).

```
//...
params.push_back (std::make_unique<juce::AudioParameterChoice>
        ("OSCWAVETYPE2", // parameter id
        "Osc 2 Wave Type",  //parameter name
        juce::StringArray {
                "Sine",
                "Saw", //possible values names,
                "Square"

        }, 0));  //standard value is 0 (sine)
    //ecc ecc...
    //...
    //...
for (int i = 0; i < synth.getNumVoices(); ++i)
    {
        if (auto voice = dynamic_cast<SynthVoice*>(synth.getVoice(i)))
        {

            //OSC
            auto& oscWaveChoice = *apvts.getRawParameterValue
                    ("OSCWAVETYPE");
            auto& osc1Pitch = *apvts.getRawParameterValue
                    ("OSC1PITCH");
            //etc...
            //stessa cosa per gli altri parametri
            //...
            for (int i=0; i<getTotalNumOutputChannels();i++)
            {
                osc1[i].setWaveType(oscWaveChoice);
                osc1[i].setPitch(osc1Pitch);
                //...
                osc2[i].setWaveType(oscWaveChoice2);
                osc2[i].setPitch(osc2Pitch);
                //...
            }
            adsr.updateADSR(/* ... */);
            filterAdsr.updateADSR(/*... */);
            voice->updateFilter(/* ...*/);

        }
    }
```

Capitolo 4

Risultati e Conclusioni

4.1 Prestazioni

Di seguito sono mostrati alcuni grafici in tempo e/o frequenza di alcuni segnali segnali generati dal *leoSynth*, con relativa schermata d'impostazione del plugin.

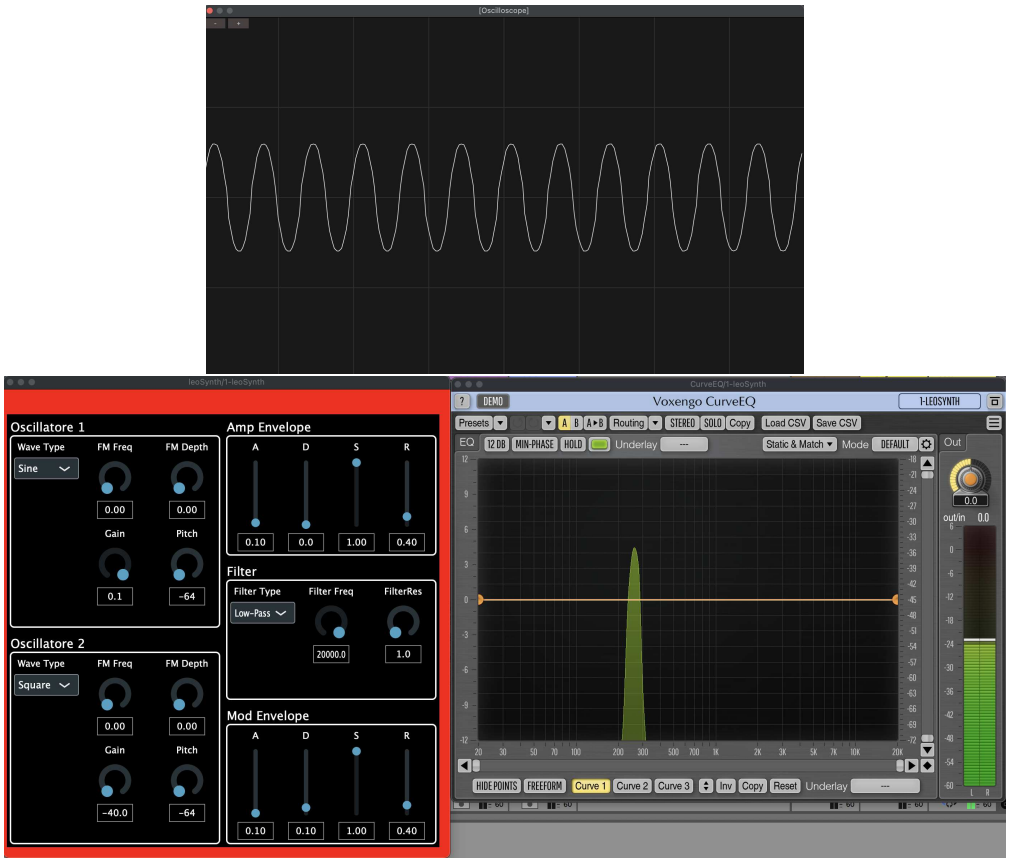

Figura 4.1: Sine Wave: semplice sinusoide. Un unico picco compare alla frequenza fondamentale (DO4, 262Hz)

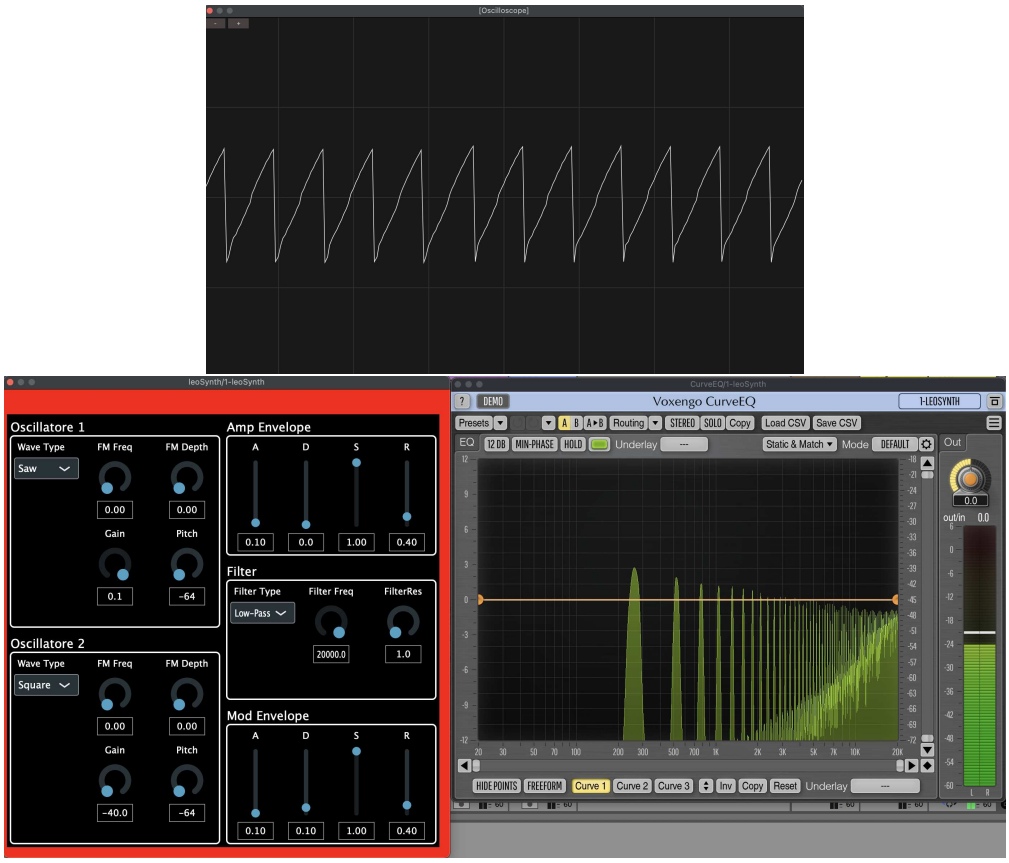

Figura 4.2: Sawtooth Wave: dente di sega. il suo spettro contiene armoniche sia pari sia dispari della frequenza fondamentale. Poiché contiene tutte le intere armoniche, è una delle migliori forme d'onda da utilizzare per la costruzione di altri suoni utilizzando la sintesi sottrattiva. (DO4, 262Hz)

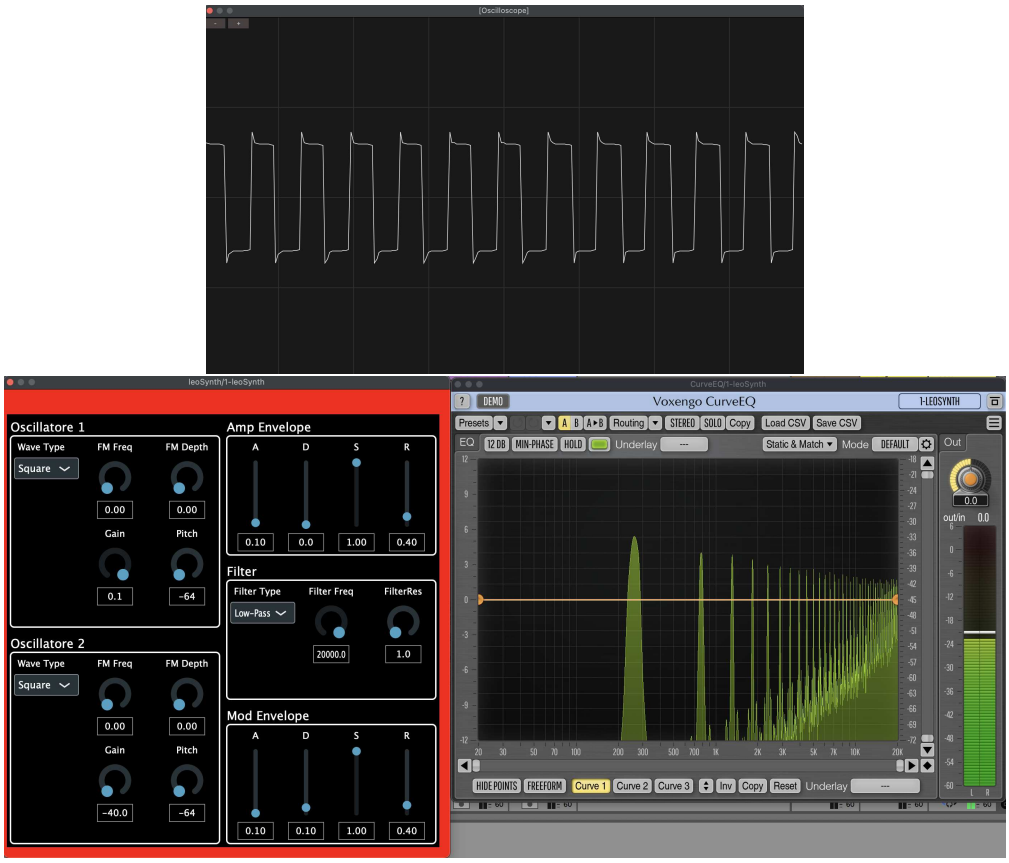

Figura 4.3: Square Wave: onda quadra. Nel suo spettro sono presenti esclusivamente le armoniche dispari. (DO4, 262Hz)

Figura 4.4: Square Wave + Filter Resonance @ 5kHz: Il segnale generato dall'onda quadra viene tagliato dalla frequenza 5kHz in poi, con un'enfasi di 9.7. (DO4, 262Hz)

Figura 4.5: Chord: Square + Saw + Filter Resonance @5kH. Accordo di Do Maggiore con enfasi intorno alla frequenza 5kHz. (DO4, 262Hz) + (MI4, 330Hz) + (SOL4, 392Hz)

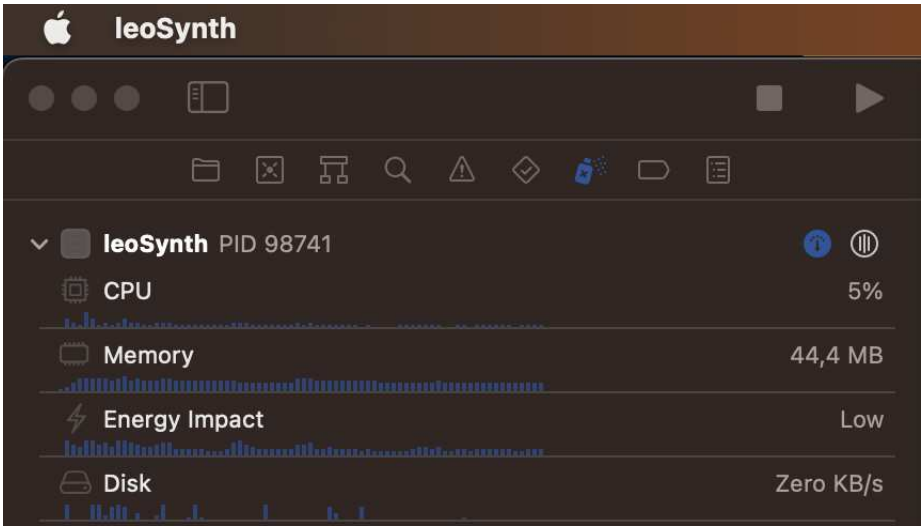

Figura 4.6: Prestazioni, secondo l'IDE Xcode

Il plugin non sembra avere un particolare impatto sulle prestazioni del computer dove è stato testato (Macbook Air con chip Apple M1).

Durante il periodo di utilizzo del plugin in modo autonomo, l'occupazione della CPU è rimasta stabilizzata intorno al 5-6%, con picchi massimi del 10%. L'impatto sulla memoria è stato altrettanto trascurabile, con un'occupazione massima di circa 50MB.

Si può presumere che, data la presenza attuale di 10 voci ed una così bassa occupazione della CPU, che si potrebbero aumentare le voci senza incorrere in particolari problemi prestazionali. Per esempio, aumentando a 20 le voci, si ha un'occupazione della CPU del 20%, ed una occupazione della memoria pressoché invariata (55MB). Aumentando fino a 100 voci, l'occupazione della CPU ha raggiunto picchi del 60%: tale potrebbe essere un eccessivo utilizzo della CPU, ipotizzando il caso di voler utilizzare il *leoSynth* insieme ad altri strumenti o plugin.

E' importante notare come, data la natura per cui è stato creato il sintetizzatore, una tale quantità di voci risulterebbe, salvo rare eccezioni, inutile. Ipoteticamente, sarebbe una scelta più adeguata decidere di impiegare ulteriore potenza di calcolo aggiungendo moduli al sintetizzatore, come per esempio effetti. Le voci potrebbero essere aumentate, ma pragmaticamente non si sentirebbe la necessità di avere la possibilità di suonare utilizzando più di 15-20 voci.

4.2 Esempi audio

Sono state effettuate alcune registrazioni per illustrare la gamma timbrica e il complessivo risultato ottenuto con lo sviluppo del *leoSynth*.

Le tracce sono ascoltabili su https://github.com/leonardoman9/SYNTH/tree/master/recs, oppure dai seguenti link:

- Test componente per componente;
- *J.S. Bach: Preludio in Do maggiore BWV 846*: in questa registrazione è stato aggiunto del riverbero in post-produzione.

4.3 Release

Il file .vst3 è stato compilato su Xcode 14.0 e può essere aperto tramite un qualsiasi DAW che supporti il formato.

Il software offre supporto nativo per controller USB esterni, è sufficiente caricare il plugin in un canale audio e fornirgli in input il controller desiderato.

Il source code e la Release possono essere visualizzati e scaricati dal seguente link: https://github.com/leonardoman9/SYNTH.

4.4 Direzioni future

Allo stato attuale, il *leoSynth* è costituito dai moduli funzionali più elementari per un sintetizzatore a sintesi sottrattiva. Tuttavia, ci sono alcune interessanti feature che possono essere integrate per arricchire ulteriormente l'esperienza degli utenti e fornire maggiore versatilità nell'utilizzo dello strumento.

Innanzitutto, una delle caratteristiche chiave che potrebbe essere implementata è la possibilità di salvare e richiamare preset sonori. Questo consentirebbe agli utenti di memorizzare e condividere le loro configurazioni sonore preferite, consentendo un rapido accesso a sonorità specifiche e facilitando il processo creativo.

Per implementare un sistema di salvataggio dei preset in JUCE, è possibile creare una classe dedicata che gestisce l'intero processo di salvataggio e caricamento dei preset. All'interno di questa classe, si possono definire metodi per il salvataggio ed il caricamento di un preset. Le classi fornite da JUCE, come juce :: File e juce :: XmlElement, possono essere utilizzate per gestire il salvataggio dei dati del preset in un formato appropriato, come XML o JSON. Inoltre, è possibile creare un'interfaccia utente che permetta all'utente di selezionare, creare o rinominare i preset. Questa interfaccia può utilizzare componenti grafici di JUCE, come juce :: ListBox o juce :: ComboBox, per visualizzare l'elenco dei preset disponibili e consentire all'utente di interagire con essi. Inoltre, si può considerare la possibilità di consentire all'utente di esportare e importare i preset in file separati utilizzando la classe juce :: FileChooser.

In aggiunta, l'espansione del synth digitale potrebbe prevedere l'integrazione di nuovi oscillatori. L'aggiunta di oscillatori supplementari, come onde diverse, permetterebbe agli utenti di esplorare una gamma ancora più ampia di timbri e texture sonore, ampliando le possibilità creative. Questa possibilità è possibile grazie al fatto che, allo stato attuale, il software non presenta problemi di prestazioni o occupazione di memoria.

Oltre all'aggiunta di nuovi oscillatori, l'inclusione di effetti audio rappresenterebbe un'aggiunta significativa al synth digitale. Effetti come riverberi, delay, chorus e flanger potrebbero essere implementati per offrire agli utenti la possibilità di modulare, spazializzare e colorare il suono in modi unici e interessanti. Ciò amplierebbe le possibilità espressive del synth, consentendo agli utenti di creare atmosfere sonore più complesse e coinvolgenti.

In JUCE, gli effetti audio possono essere implementati utilizzando la classe juce :: AudioProcessor. Questa classe fornisce un'interfaccia per creare e gestire effetti audio personalizzati.

Per implementare un effetto, si creerà una classe derivata da juce :: AudioProcessor. Questa classe conterrà la logica e l'algoritmo specifici dell'effetto desiderato.

All'interno della classe dell' effetto, si dovranno implementare i metodi virtuali forniti da juce :: AudioProcessor, come ad esempio processBlock(), che rappresenta il metodo principale dell'effetto. Questo metodo riceve e processa i buffer audio in input e genera i buffer audio in output. All'interno di processBlock(), si può applicare l'effetto audio utilizzando le tecniche desiderate, ad esempio applicando filtri, modulazioni, delay, riverberi o qualsiasi altro tipo di elaborazione audio.

Un'altra caratteristica che potrebbe essere aggiunta al synth digitale per arricchire l'esperienza sonora è un oscillatore a bassa frequenza (LFO). L'aggiunta di un LFO consentirebbe agli utenti di introdurre modulazioni periodiche e cicliche ai parametri del suono, creando effetti di vibrato, tremolo, modulazione di ampiezza e altre variazioni periodiche.

L'inclusione di un LFO aprirebbe nuove possibilità creative, consentendo agli utenti di esplorare la modulazione di parametri chiave come la frequenza dell'oscillatore, l'ampiezza del segnale, la forma d'onda e altri parametri selezionati. Ciò consentirebbe di aggiungere movimento e vivacità al suono, creando effetti pulsanti, sincopi ritmiche o modulazioni sottili per arricchire ulteriormente le composizioni musicali.

La flessibilità del LFO potrebbe essere ampliata fornendo diverse forme d'onda tra cui scegliere, come sinusoidale, triangolare, quadra o campionata. Inoltre, sarebbe possibile implementare funzionalità avanzate come la sincronizzazione dell'LFO al tempo del brano, la regolazione della velocità di modulazione e la possibilità di assegnare l'LFO a più parametri simultaneamente per ottenere effetti complessi e interazioni sonore intriganti.

Infine, nel codice sono già presenti file del *model* per sviluppare un oscilloscopio ed una tastiera virtuale per utilizzare lo strumento senza l'ausilio di un controller esterno.

Un oscilloscopio consente di visualizzare il segnale audio generato dal sintetizzatore nel dominio del tempo. Ciò offre all'utente un feedback visivo immediato sulle caratteristiche e sulla forma d'onda del segnale audio prodotto dal sintetizzatore. Un oscilloscopio può aiutare a identificare problemi come distorsioni, clipping, sfasamenti o irregolarità nel segnale audio. Inoltre, può essere utilizzato per visualizzare i

cambiamenti nel segnale durante la modifica dei parametri o l'applicazione di effetti, consentendo all'utente di ottenere un controllo visivo e accurato sul suono prodotto.

Per implementare una tastiera virtuale utilizzando JUCE, si può creare una classe che estende la classe Component. Questa classe si occuperà di gestire gli eventi di input MIDI necessari per il controllo della tastiera virtuale. All'interno di VirtualKeyboardComponent, si potranno sovrascrivere i metodi handleNoteOn e handleNoteOff per gestire rispettivamente l'evento di premuta e rilascio di una nota sulla tastiera virtuale. La grafica della tastiera virtuale potrà essere disegnata nel metodo paint, utilizzando le funzioni di disegno fornite da JUCE. Per consentire all'utente di suonare le note tramite il mouse, si potranno gestire gli eventi di input del mouse all'interno dei metodi mouseDown, mouseDrag e mouseUp.

Bibliografia

[1] Roads, C. *The Computer Music Tutorial*. MIT Press, 1996.

[2] Maher, R. C. Sinewave additive synthesis revisited. *Journal of The Audio Engineering Society*, 1991.

[3] Cahill, T. Art of and apparatus for generating and distributing music electrically, US patent 580035, 1897-04-06.

[4] Brice, R. *Music engineering*. Newnes, 2001.

[5] Bush, D. E.; Kassel, R. *The organ: an encyclopedia*, volume 3. Psychology Press, 2006.

[6] Pekonen, J. and Pihlajamäki, T. and Välimäki, V. Computationally efficient hammond organ synthesis. In *Proceedings of the 14th International Conference on Digital Audio Effects (DAFx-11), Paris, France*, pages 19–23, 2011.

[7] Miranda, E. R. *Computer Sound Design, synthesis techniques and programming*. 2002.

[8] Creasey, D. *Audio Processes - Musical analysis, modification, synthesis and control*.

[9] Pinch, T. In the moog. In *Journées d'Informatique Musicale*, 2011.

[10] J. Chowning. *The Synthesis of Complex Audio Spectra by Means of Frequency Modulation*. Journal of the Audio Engineering Society, 1973.

[11] Mcguire, S.; Matějů, Z. *The Art of Digital Orchestration*. 2020.

[12] Dodge, C.; Jerse, T. A. *Computer music: synthesis, composition, and performance*. Macmillan Library Reference, 1985.

[13] Smith, III., Julius, O. Viewpoints on the history of digital synthesis. In *Proceedings of the International Computer Music Conference*, 1991.

[14] Massie, D.C. Wavetable sampling synthesis. *Applications of Digital Signal Processing to Audio and Acoustics*, pages 311–341, 2002.

[15] Kadam, S., Sasidaran, D., Awawdeh, A., Johnson, L., and Soderstrand, M. Comparison of various numerically controlled oscillators. In *The 2002 45th Midwest Symposium on Circuits and Systems, 2002. MWSCAS-2002.*, volume 3, pages III–III. IEEE, 2002.

[16] Gabrielli, L., D'Angelo, S., La Pastina, P.P., Squartini, S. Antiderivative antialiasing for arbitrary waveform generation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:2743–2753, 2022.

[17] Valimaki, V., Nam, J., Smith, J.O., Abel, J.S. Alias-suppressed oscillators based on differentiated polynomial waveforms. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(4):786–798, 2010.

[18] Valimaki, V., Huovilainen, A. Antialiasing oscillators in subtractive synthesis. *IEEE Signal Processing Magazine*, 24(2):116–125, 2007.

[19] Sallen, R. P.; Key, E. L. A practical method of designing rc active filters. *IRE Transactions on Circuit Theory*, 2(1):74–85, 1955.

[20] Stinchcombe, T. E. A Study Of The Korg MS10 & MS20 filters. *Online, Aug*, 30, 2006.

[21] Moog, R. A. Electronic high-pass and low-pass filters employing the base to emitter diode resistance of bipolar transistors, October 28 1969. US Patent 3,475,623.

[22] Smith, Steven W et al. The scientist and engineer's guide to digital signal processing, 1997.

[23] Oppenheim, A. V., Willsky, A. S., Nawab, S. H., Ding, J. *Signals and systems*, volume 2. Prentice hall Upper Saddle River, NJ, 1997.

[24] Deutsch, R.; Deutsch, L. J. Adsr envelope generator, March 21 1978. US Patent 4,079,650.

[25] Jenkins, M. *Analog synthesizers: understanding, performing, buying: from the legacy of Moog to software synthesis*. Routledge, 2019.

[26] Boulanger, R. and Lazzarini, V. and Mathews, M.V. *The Audio Programming Book*. MIT Press, 2010.

[27] Gamma, E. and Helm, R. and Johnson, R. and Vlissides, J. M. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1 edition, 1994.

[28] Korg. *Minilogue xd Owner's Manual*. Korg.

[29] Björn, A. *Kern Performance Synthesizer*. Full Bucket Music.

[30] Zavalishin, V. The art of va filter design. *Native Instruments, Berlin, Germany*, 2012.

[31] Juce documentation. Online. Accessed: 2022-09-13.

[32] Manning, P. *Electronic and computer music*. Oxford University Press, 2013.

[33] Chowning, J. The synthesis of complex audio spectra by means of frequency modulation. *Computer Music Journal*, 1(2):46–54, 1977.

[34] Moog, R. A. Voltage Controlled Electronic Music Modules. *Journal of the Audio Engineering Society*, 13(3):200–206, 1965.

[35] Calzolari, P.; Graffi, S. *Elementi di Elettronica*. 1972.

[36] Gonzalez, G. *Foundations of Oscillator Circuit Design*. 2006.

[37] Sedra, A.; Smith, K.C. Microelectronic circuits 8th edition. 2020.

[38] Stinchcombe, T. E. Derivation Of The Transfer Function Of The Moog Ladder Filter. *Online*, 2005.