UNIVERSITÀ POLITECNICA DELLE MARCHE

Corso di laurea magistrale in
INGEGNERIA ELETTRONICA

Studio e sperimentazione di un audio sampler espressivo su scheda embedded STM32

Development and evaluation of an audio sampler on STM32 board

Tesi di laurea di
Giada ALIFFI

Relatore
Leonardo GABRIELLI

ANNO ACCADEMICO 2021/2022

Indice

Elenco delle figure

1 Introduzione

1.1 Obiettivi della tesi

L'obiettivo della presente tesi è la realizzazione di un "sample player" (o sintetizzatore a campioni) su una scheda embedded STM32.
Questo dovrà potenzialmente essere integrato nella prossima versione dell' "ARTinoise Re.corder" [1]. Tale strumento, attualmente, è in grado di funzionare in due modalità. In quella tradizionale, è utilizzabile come un normale flauto soprano acustico; mentre, in quella digitale, viene collegato via Bluetooth Low Energy (BLE) ad un'app mobile in grado di generare suoni di diversa natura.
Lo scopo, dunque, è quello di consentire la generazione del suono in autonomia anche nella modalità digitale. In particolare, sulla base di segnali MIDI ricevuti in ingresso via USB, dovrà essere innescata la riproduzione di note musicali aventi differenti frequenze, volumi e timbri associati a vari strumenti (flauto, oboe, tromba, batteria, violino, ecc...).
Al giorno d'oggi, la sintesi basata su campionamento è una delle più utilizzate, infatti, rispetto ad altri metodi di sintesi digitale come la sintesi per modelli fisici o la sintesi additiva, richiede una potenza di di calcolo molto inferiore. Questo perché la maggior parte delle sfumature dei modelli sonori sono contenute nei campioni preregistrati piuttosto che calcolate in tempo reale. In sostanza, dunque, con i bassi costi di un microcontrollore mediamente performante e di un buon dispositivo d'archiviazione, è possibile ottenere un'ottima emulazione dello strumento reale. Inoltre, a differenza dei sintetizzatori analogici, i circuiti non devono essere duplicati per consentire la riproduzione di più voci contemporaneamente. Quindi, il grado di polifonia delle macchine basate su campioni è generalmente molto più alto.
Il principale svantaggio di tale sintesi è forse la staticità dello spettro generato, la quale si contrappone alla dinamicità riscontrabile nella realtà durante molteplici osservazioni dello stesso segnale. Tali variazioni, dovute in parte alle caratteristiche meccaniche ed elettroniche dello strumento e in parte alla tecnica dell'esecutore, contribuiscono normalmente a rendere il suono più naturale.
Nel caso presente, tuttavia, l'espressività potrà essere introdotta sfruttando i sensori a bordo del "breath controller" ed, eventualmente, ampliando la libreria di campioni registrati.

1.2 Sintesi basata su campionamento: step, accorgimenti e possibili problematiche

La sintesi basata su campionamento, come indicato in [2] consiste nel pilotare per mezzo di messaggi MIDI, la riproduzione di file audio contenenti campioni di strumenti reali. Il processo è descritto come composto da differenti fasi: *"source capture"*, *"editing"*, *"mapping"* e *"tweaking"*.

1.2.1 Prima fase di sintesi: "source capture"

La prima fase, detta di *"source capture"*, coincide con la registrazione della sorgente sonora, la quale deve avvenire con la più alta qualità possibile. Tra le possibili opzioni, c'è quella di utilizzare un microfono vicino all'altoparlante o al punto di emissione sonora dello strumento; tuttavia spesso non rappresenta la scelta migliore in quanto potrebbe comportare l'acquisizione di troppo rumore di tipo elettrico dovuto alla strumentazione, di tipo meccanico dovuto a tasti o pedali o ambientale se non si è in uno studio di registrazione. L'alternativa migliore, dunque, se possibile, sarebbe quella di utilizzare connessioni dirette tra lo strumento e il computer con il quale si registra.

Riguardo alla quantità di suoni da registrare poi, in [3] si specifica come sia importante ottenere i campioni di un numero sufficiente di note tale da ridurre gli effetti negativi dovute a trasposizioni estreme, come per esempio la "munchkinization". Tale effetto è chiamato così perché, sulla voce umana, comporta una variazione del timbro che ricorda la parlata del personaggio Munchkin de "Il mago di Oz" [4].
Dal punto di vista tecnico, tale effetto è dovuto ad un importante teorema riguardante la trasformata di Fourier detto "teorema di Scaling" [5]. Quest'ultimo afferma che se si effettua una compressione nel tempo ("squeeze") di un fattore α, con $\alpha \in \mathbb{R} \setminus \{0\}$ su un segnale, si avrà come conseguenza un'espansione ("stretch") in frequenza dello stesso fattore e viceversa. In formule:

$$x\left(\frac{t}{\alpha}\right) \overset{\mathscr{F}}{\Longrightarrow} |\alpha| \cdot X(\alpha\omega) \tag{1}$$

Infatti, se si calcola la trasformata di Fourier del segnale compresso, si ottiene:

$$\begin{aligned}\mathscr{F}\left\{x\left(\frac{t}{\alpha}\right)\right\} &= \int_{-\infty}^{+\infty} x(\alpha t)\cdot e^{-j\omega t}\,dt \quad \left(\text{se si indica } \tau = \frac{t}{\alpha}\right) \\ &= \int_{-\infty}^{+\infty} x(\tau)\cdot e^{-j\omega(\alpha\tau)}\,d(\alpha\tau) \\ &= |\alpha|\cdot\int_{-\infty}^{+\infty} x(\tau)\cdot e^{-j\omega(\alpha\tau)}\,d(\tau) \\ &= |\alpha|\cdot X(\alpha\omega)\end{aligned} \tag{2}$$

Tra i possibili segnali a cui si può applicare (1) c'è appunto la voce umana. In tal caso, va tenuto in considerazione che lo spettro di quest'ultima è dotato di picchi caratteristici detti "formanti", i quali sono fondamentali per l'intelligibilità del discorso. Infatti, i ciascuna delle 7 vocali della lingua italiana è distinguibile dalle altre grazie ad almeno 5 formanti, situate in zone frequenziali ben precise, come mostrato in fig. 1:

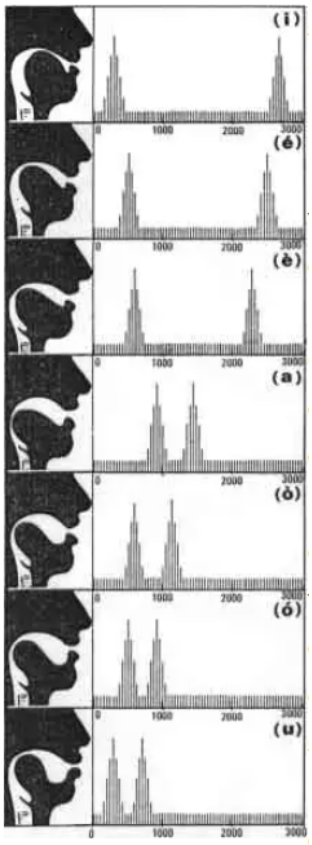

Figura 1: Formanti caratteristiche delle vocali italiane

L'aspetto interessante è che per ogni vocale la posizione delle formanti non cambia pur variando la frequenza che si sta emettendo. Ciò che fa la differenza, dunque, è solo la diversa quantità di armoniche contenute in ciascuna formante. Tale effetto è mostrato per la vocale "a" in fig. 2:

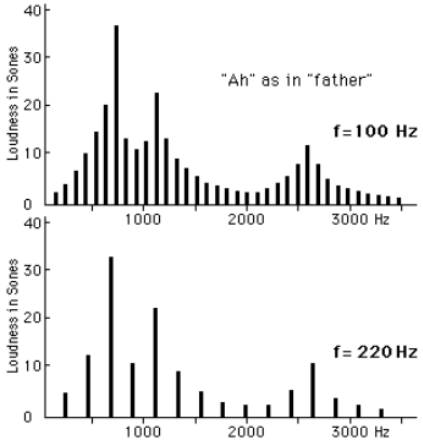

Figura 2: Conservazione delle formanti per una stessa vocale a frequenze diverse

Ciò avviene in quanto, aumentando la frequenza, la prima armonica si sposta visivamente verso destra, e così tutte le altre armoniche successive, che sono in corrispondenza dei multipli interi della fondamentale.
Le prime due formanti, inoltre, contengono sufficienti informazioni affinchè il nostro sistema di audio-codifica identifichi e classifichi un timbro vocalico. In particolare, la frequenza della prima formante F1 è dipende dall'ampiezza dell'apertura della bocca; mentre la frequenza della seconda formante F2 è determinata dalle diverse posizioni della lingua.
Se, quindi, si effettua una compressione temporale di un segnale contenente voce umana, si otterrà come effetto collaterale un allontanamento delle componenti armoniche che caratterizzano le formanti e, quindi, un'alterazione di queste ultime. Ciò causerà lo snaturamento del timbro originale noto come "munchkinization".
Nel caso di una nota generata da uno strumento musicale, invece, si può dire che il timbro delle note più basse è tipicamente diverso da quello delle note alte. Di conseguenza, se il fattore di compressione/espansione α è troppo grande, si potrebbe ottenere un contenuto armonico per la nota generata molto diverso da quello che si avrebbe suonandola sullo strumento reale, creando un effetto innaturale.

Un altro problema poi, come indicato in [6], potrebbe essere la generazione di aliasing nel caso in cui l'espansione in frequenza del fattore α causi la comparsa di armoniche significative sopra a fs/2, con fs frequenza di campionamento.

Sia in [2] che in [3], inoltre, si afferma come un accorgimento per rendere il suono più realistico, sia quello di registrare note a differenti livelli di "velocity" in modo da ottenere un contenuto armonico diverso a seconda della forza con la quale una nota viene suonata.

1.2.2 Seconda fase di sintesi: "editing"

La seconda fase del processo di sintesi presentata in [2] è quella dell' *"editing"*, la quale consiste nell'ascolto del materiale registrato e nella sua correzione, rimuovendo intervalli di silenzio o rumore e ritagliando, dunque, le sole parti d'interesse. In tale contesto, occorre fare in modo tale che il primo campione dell'attacco corrisponda ad uno zero per poi proseguire nel tempo con un aumento graduale dell'ampiezza. Ciò assicura che riproducendo tale nota non si senta un click nella fase iniziale.

1.2.3 Terza fase di sintesi: "mapping"

Una volta ottenuti i file audio desiderati, la terza fase descritta in [2] è quella del *"mapping"*, che consiste nell'associare ciascuno di essi ad una o più note MIDI. In particolare, ciascun file in memoria sarà associato ad una specifica "zone", la quale è indicativa della "root note", cioè della frequenza reale registrata, ed eventualmente di quali altre note andranno generate da essa tramite una trasposizione (interpolazione).
Se, inoltre, sono state registrate note a più livelli di "velocity", andranno creati anche dei "velocity layers", ovvero occorrerà stabilire qual è il confine tra l'utilizzo di un campione piuttosto che di un altro.
Fondamentale è anche scegliere correttamente quale porzione della nota sarà quella da riprodurre a loop nella fase di sustain che segue l'attacco. In tale contesto, occorre fare in modo che l'inizio e la fine siano simili come livello e timbro, per evitare discontinuità innaturali nella riproduzione ciclica.
Riguardo al livello, se la forma d'onda presenta dei passaggi per zero, sarebbe meglio scegliere tali punti come inizio e fine del loop.

Per il timbro, invece, in [3], si specifica che, nel caso in cui fosse troppo diverso, si potrebbe procedere mediante un "cross-fading" tra l'inizio e la fine della forma d'onda, in modo da creare una transizione meno brusca. Tuttavia se questo non venisse implementato in modo opportuno, dovendo il loop essere riprodotto centinaia di volte, l'entità del difetto verrebbe sicuramente a galla in modo amplificato rispetto ad un caso di singola riproduzione.
Inoltre, al fine di evitare "click" o "glitch", si puntualizza anche come sia importante che l'inizio e la fine siano caratterizzati dallo stesso "slope" (pendenza) e dallo stesso tasso di variazione di tale parametro.
In alternativa, se dovesse essere complesso ottenere le specifiche descritte in precedenza, sia in [2] che in [3], si consiglia di applicare dei "fade" per raccordare le parti o addirittura creare un loop di tipo "forward-backward" in cui, una volta arrivati alla fine del campione, si torna indietro, anziché ricominciare dall'inizio.
In [3], tuttavia, si specifica anche come quest'ultima strategia sia efficace solo nel caso di campioni di breve durata, con i quali non è percepibile all'ascolto l'inversione di rotta.
Per quanto riguarda il rilascio, infine, è possibile sia sfumare il sustain che riprodurre altri campioni opportuni alla fine del loop, in base alle preferenze e alle possibilità offerte dallo strumento.

1.2.4 Quarta fase di sintesi: "tweaking"

La quarta ed ultima fase della procedura di sintesi descritta in [2] descritta in [2] è quella di *"tweaking"*, ovvero di sperimentazione e ritocco dei campioni e dei parametri al fine di ottenere il miglior risultato possibile.

1.3 Cenni storici sui campionatori digitali

Affinché tutte le fasi di sintesi descritte in [2] possano essere messe in atto, in [3], si spiega come un campionatore debba essere composto da almeno 3 parti: un ADC per acquisire una forma d'onda analogica e tradurla in una sequenza di valori digitali, una memoria per immagazzinare i campioni e un DAC per riconvertire in analogico il segnale in fase di riproduzione.

In molte fonti come [7], [8], [9] e [10] si racconta la storia dei primi campionatori. Il primo modello di campionatore digitale monofonico disponibile in com-

mercio fu inventato da Harry Mendell nel 1976 e si chiamava "Computer Music Melodian", fig. 3. Quest'ultimo era basato sul computer PDP-8 della Digital Equipment Corporation, campionava ad una frequenza di massimo 22 kHz con una risoluzione di 12 bit.
Il primo utilizzo degno di nota del Melodian avvenne da parte di Stevie Wonder, nel 1979, per la realizzazione della colonna sonora "The Secret Life of Plants". Egli, infatti, creò melodie e ritmi complessi a partire da campionamenti di suoni provenienti dalla natura realizzati proprio per mezzo di tale strumento.

Figura 3: Computer Music Melodian [11]

Successivamente, come indicato in [7] e [8], nel 1979, Peter Fogel e Kim Ryrie inventarono il Fairlight CMI mostrato in fig. 5, ovvero il primo campionatore digitale polifonico. La prima versione campionava utilizzando una risoluzione di 8 bit per campione, a una velocità di 24 kHz, e utilizzava due processori Motorola 6800 a 8 bit. Era dotato di due tastiere da sei ottave, una tastiera alfanumerica e un'unità di visualizzazione video interattiva (VDU) in cui le forme d'onda potevano essere modificate o addirittura disegnate da zero utilizzando una penna ottica . Il software consentiva l'editing, il looping e il missaggio di suoni che potevano quindi essere riprodotti tramite la tastiera o il sequencer basato su software. Tra i primi utilizzatori più noti di tale strumento troviamo Kate Bush nel suo album "Never for ever" del 1980, Peter Gabriel nel suo album omonimo del 1982, i Devo nel loro album "Shout" del 1984 e molti altri.

Figura 4: Fairlight CMI [12]

A fare concorrenza al Fairlight CMI, in quegli anni, fu principalmente il "Synclavier". In [13], si racconta, infatti, come tale campionatore, nella sua seconda versione del 1980, ovvero il "Synclavier II", accostò alla sintesi additiva e FM, anche l'utilizzo di campioni, dando la possibilità di eseguire hard disk recording di 16 tracce a 50 kHz. Il suo corpo, inoltre, era caratterizzato da una tastiera e da un pannello di controllo dal quale era possibile definire con precisione la timbrica della nota suonata.
Tale strumento venne usato da molti musicisti e cantanti famosi quali, per esempio, Michael Jackson nel suo album "Thriller" del 1982, i Genesis nel loro album omonimo del 1983, Frank Zappa negli album "Francesco Zappa" "Jazz from hell" del 1984 e 1986 rispettivamente e molti altri.

Figura 5: Synclavier

A partire dagli anni ’80 in poi, si svilupparono molti altri campionatori innovativi. Tra questi troviamo la serie degli "E-mu Emulator" [14] prodotta dalla "E-mu Systems" dal 1982 al 1990.
In particolare, la prima versione del 1982, detta "Emulator", fig. 6, era una workstation con tastiera basata su floppy, che permetteva al musicista di campionare suoni, registrandoli e rendendoli poi eseguibili come fossero i suoni di un preset, tramite le note della tastiera. Il floppy da 5 1/4" rendeva possibile la memorizzazione dei campionamenti, e il musicista poteva crearsi una gamma personale, condividerla, o anche comprare librerie già pronte.
Il campionatore era caratterizzato da un rate di campionamento a 27,7 kHz e una risoluzione dei campioni di 8 bit. Inoltre, era dotato di un unico filtro che permetteva di eseguire un solo loop per volta.
Venne distribuito in tre modelli: uno a due (di cui fu venduta una sola unità), uno a quattro ed uno a otto voci.
Stevie Wonder, il quale elogiò lo strumento alla convention NAMM del 1981, ricevette la prima unità messa in commercio (con numero di serie 001).
La seconda versione del 1984, detta "Emulator II", permetteva, rispetto alla prima, una maggior flessibilità nella modifica dei suoni e un miglior controllo in tempo reale. Inoltre, vennero introdotti una seconda unità floppy e un hard-disk da 20 MB.
La terza versione del 1987, detta "Emulator III", era caratterizzato da 4 o 8 MB di memoria e campionava in 16 bit ad un rate di 44 kHz. La qualità del suono era notevolmente migliorata rispetto alle precedenti versioni, con migliori uscite e filtri audio più avanzati. Al tempo, era considerato il più avanzato e professionale campionatore disponibile.
Nello stesso anno (1987), venne introdotto sul mercato l’E-mu SP-1200 [15] il quale aveva specifiche limitate, ovvero lavorava ad una frequenza di campionamento di 26.04 kHz con una risoluzione di 12 bit. Tuttavia, proprio grazie a queste caratteristiche divenne uno dei campionatori più di rilievo nel campo della musica hip-hop. Infatti, il suono suono risultava "caldo" e un po’ "sporco" come quello dei vecchi vinili . Questo modello, inoltre, a differenza dei precedenti, fungeva anche da drum machine e sequencer.

Figura 6: E-mu Emulator I [14]

Un'altra linea di campionatori di rilievo fu realizzata negli anni '80 dalla Akai [16]. Tra questi, troviamo l'Akai S900 del 1986, un campionatore digitale utilizzabile come unità rack. Esso era polifonico a 8 voci e in grado di campionare in 12 bit ad un rate di massimo 40 kHz. Inoltre, possedeva una memoria di 750 kB che consentiva di salvare fino a 12 secondi di suono al rate più alto.
Nel 1988, poi venne introdotto nel mercato l'Akai MPC60 che, oltre ad essere un campionatore digitale, svolgeva anche le funzionalità di una drum machine e un MIDI sequencer. Questo modello era il primo Akai a non essere montato su un rack e ad avere dei pad touch-sensitive.
Sempre nel 1988, ci fu l'Akai S1000 fig. 7 che, molto probabilmente, può essere considerato come il campionatore stereo con risoluzione 16 bit e rate di 44.1 kHz più famoso di sempre. Questo modello era in grado di gestire fino a 16 voci di polifonia, era dotato di una memoria di 32 MB e unità di elaborazione in grado di lavorare con dati di 24 bit. Tra queste ultime, troviamo un filtro digitale (pendenza 18 dB/ottava), un LFO e due generatori d'inviluppo ADSR (per la modulazione d'ampiezza e il filtraggio). L'S1000, inoltre, offriva la possibilità di settare fino a 8 diversi punti di loop più altre funzionalità innovative. Tra queste troviamo l'"autolooping", il "crossfade looping", il "loop in release" (il suono diminuiva di volume gradualmente durante il loop per creare il rilascio), il "loop until release" (il loop si ripeteva fino all'inizio del rilascio), il "reverse and time stretch", ecc...
Nello stesso anno (1988), venne introdotta sul mercato la nuova versione dell'S900, detta S950, la quale, rispetto alla precedente, aveva una frequenza di campionamento massimale di 48 kHz e includeva alcune delle funzionalità del contemporaneo S1000.

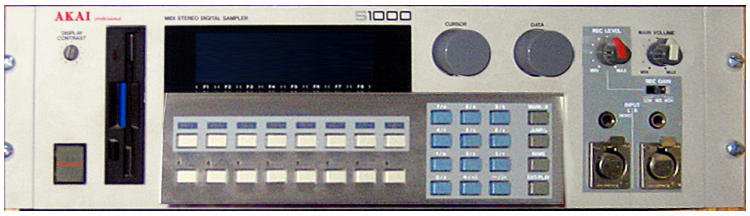

Figura 7: Akai S1000 [16]

Nel corso degli anni poi, i campionatori si sono evoluti sempre di più fino ad arrivare ad oggi in cui, come specificato in [3], gli hardware ad hoc sono stati sostituiti dai computer, aventi potenza di calcolo e capacità d'archiviazioni sufficienti a gestire tutte le funzionalità di un sampler. Tra i principali plugin di campionamento moderni, troviamo "Native Instruments Kontakt 6" fig. 8, "Logic EXS24 mkII", MOTU Machfive 3 e molti altri.

Figura 8: Native Instruments Kontakt 6

1.4 Tecniche d'interpolazione

1.4.1 Il problema dell'interpolazione

Data una sequenza di n numeri reali distinti x_k, chiamati nodi, e per ciascuno di questi sia dato un secondo numero y_k. Il problema dell'interpolazione consiste nell' individuare una funzione f(x) di una certa famiglia tale per cui:

$$f(x_k) = y_k \text{ con } k = 1, ..., n \tag{3}$$

Una coppia (x_k, y_k) viene chiamata punto dato ed f(x) viene detta funzione interpolante, o semplicemente interpolante, per i punti dati.
L'interpolazione ha senso se possiamo presupporre che f(x) rappresenti in maniera significativa la relazione che intercorre tra i valori degli x_k rispetto ai valori dei rispettivi y_k.

1.4.2 Interpolazione polinomiale di Lagrange

Dato un un insieme di punti $[x_k, y_k]$ con k = 0, ..., N, l'interpolazione polinomiale di ordine N [17] [18], con N ≥ 1 afferma che l'immagine di un punto x tale che $x_0 \leq x \leq x_1$ può essere stimata come segue:

$$f(x_0, ..., x_n, D) = \sum_{n=0}^{N} l_n(D) \cdot x_n \tag{4}$$

dove i pesi sono dati da:

$$l_n(D) = \prod_{\substack{j=0 \\ j \neq n}}^{N} \frac{D - x_j}{x_n - x_j} \tag{5}$$

e $D \in [0,1]$ rappresenta lo scostamento tra l'indice del campione da calcolare x e quello del campione noto immediatamente precedente, x_0. Per esempio, dunque, se D = 0, allora $x = x_0$; se D = 1, allora $x = x_1$; se D = 0.5, allora $x = \frac{x_0 + x_1}{2}$; ecc...

Se consideriamo il caso N = 1, allora l'interpolazione viene detta "lineare", in quanto dati due punti (x_0, y_0) e (x_1, y_1), questi verranno congiunti per mezzo di un segmento. Più precisamente, possiamo dire che, ad ogni valore compreso nell'intervallo $[x_0, x_1]$, si associa la media ponderata tra y_0 e y_1. In formule:

$$f(x0, x1, D) = \frac{D - x_1}{x_0 - x_1} \cdot y_0 + \frac{D - x_0}{x_1 - x_0} \cdot y_1 \tag{6}$$

L'interpolante lineare è molto semplice, però ha un grande svantaggio, ovvero il non essere derivabile nei punti x_k. Al contrario, le funzioni polinomiali di ordine N > 1 sono continue e indefinitamente differenziabili in un qualsiasi intervallo.
All'aumentare dell'ordine N scelto, inoltre, la qualità dell'interpolazione è genericamente più alta.
Tra gli svantaggi però troviamo che, più N è elevato, più numerosi saranno i coefficienti da calcolare, il che vuol dire che si ha un maggiore costo computazionale. Inoltre, si può riscontrare il cosiddetto "fenomeno di Runge", il quale consiste in

un aumento, al crescere di N, dell'errore commesso agli estremi dell'intervallo d'interpolazione.
Tra le possibili soluzioni per evitare tale problematica troviamo l'evitare che i punti di interpolazione siano equispaziati tra di loro oppure il suddividere il dominio in domini più piccoli su cui interpolare usando polinomi di ordine inferiore (N = 1 oppure N = 2) [17].

1.4.3 Interpolazione mediante spline di Hermite e Beziér

Un'alternativa all'interpolazione lineare, in cui si utilizza comunque solo una coppia di punti adiacenti (x_i, y_i) e (x_{i+1}, y_{i+1}), è quella data dalle spline [17]. In questo caso le funzioni interpolanti non sono delle rette ma dei polinomi di ordine fisso e basso. I vantaggi di semplicità e basso costo computazionale si uniscono alla possibilità di avere derivate continue come per i polinomi. La spline più semplice a cui possiamo pensare è la spline di Hermite. La spline di Hermite è un polinomio di grado 3 derivabile una volta sola, pertanto, il suo andamento sarà continuo ma non lo sarà la sua curvatura. La spline di Hermite assume la forma seguente:

$$p(x) = a\,x^3 + b\,x^2 + c\,x + d \tag{7}$$

Il polinomio presenta 4 coefficienti incogniti, quindi avremo bisogno di 4 equazioni per risolvere il sistema. Le prime due equazioni si ottengono imponendo il passaggio della spline per i due punti (x_i, y_i) e (x_{i+1}, y_{i+1}), considerati. Le restanti due equazioni si trovano calcolando la derivata prima della spline di Hermite:

$$p'(x) = 3\,a\,x^2 + 2\,b\,x + c \tag{8}$$

e imponendo dei vincoli su di essa agli estremi dell'intervallo x_i e x_{i+1}.
La spline di Hermite naturale, per esempio, prevede che venga imposto:

$$p'(x_i) = 0 \;\;;\;\; p'(x_{i+1}) = 0 \tag{9}$$

Oltre alla spline di Hermite, possiamo trovare la spline di Bézier la quale è definita come segue:

$$C(u) = \sum_{i=0}^{n-1} N_{i,p} \cdot P_i \tag{10}$$

dove u è un vettore di n+p+2 nodi, i P_i rappresentano i punti di controllo, mentre gli $N_{i,p}$ sono coefficienti che dipendono dal punto i-esimo e dal grado del polinomio.

In particolare, gli $N_{i,p}$ possono essere calcolati mediante le formule ricorsive di Cox-De Boor:

$$\begin{aligned} N_{i,0}(u) &= 1 \text{ se } u_i < u < u_{i+1} \text{ altrimenti } N_{i,0}(u) = 0 \\ N_{i,p}(u) &= \frac{u - u_i}{u_{i+p} - u_i} \cdot N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} \cdot N_{i+1,p-1}(u) \end{aligned} \tag{11}$$

Per risolvere il problema della curva di Bézier dobbiamo trovare un set di coefficienti, detti punti di controllo, che servono per definire la curva.
Per calcolare tali punti di controllo incogniti dobbiamo risolvere il seguente sistema lineare:

$$D = N \cdot P \tag{12}$$

dove P rappresenta l'insieme dei punti di controllo incognito, D è l'insieme dei dati a disposizione per cui la curva deve necessariamente passare e N è la matrice dei coefficienti $N_{i,p}$ come definiti prima.

In fig. 9, è mostrato un esempio di applicazione delle tecniche d'interpolazione finora discusse:

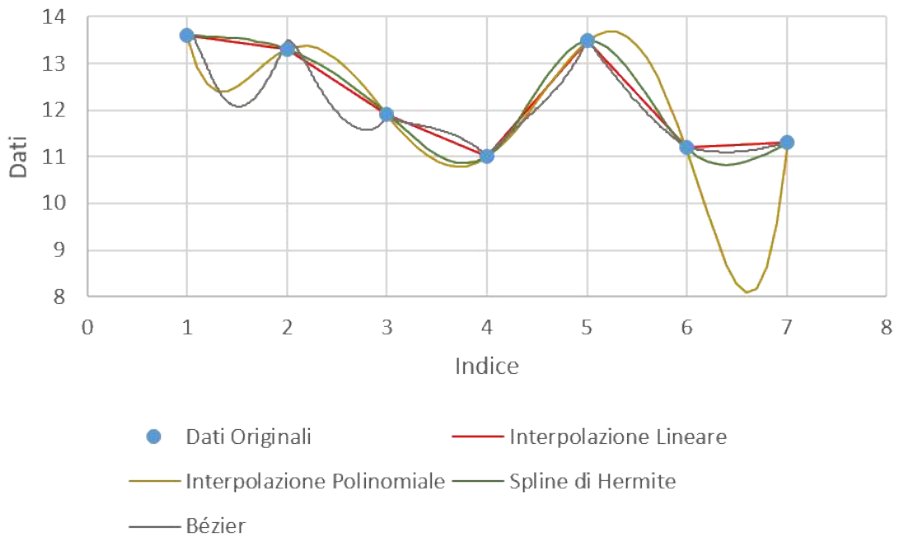

Figura 9: Differenti tipologie d'interpolazione [17]

1.4.4 Interpolazione mediante funzione sinc (seno cardinale)

Un'altra tipologia d'interpolazione è la "Sinc interpolation" [19], la quale consiste nell' applicare al segnale un filtro passa-basso caratterizzato dalla seguente

risposta impulsiva:

$$h(t) = sinc\left(\frac{\omega_s\, t}{2\pi}\right) \tag{13}$$

In tal caso, si otterrebbe una ricostruzione del segnale a tempo continuo, quindi, ricampionando ad un'altra frequenza, sarebbe possibile ottenere i valori del segnale negli istanti desiderati.
Il problema di questa soluzione è però che il filtro impiegato è un IIR e, quindi, anticausale. Di conseguenza, questo sarà irrealizzabile a meno di applicare alla risposta impulsiva una finestratura. È evidente che, più la risposta impulsiva finestrata sarà lunga, più la qualità dell'interpolazione sarà alta ma, d'altro canto, aumenterà il costo computazionale e la memoria occupata dalla eventuale LUT dove andranno salvati i coefficienti.

In fig. 10, è mostrato un confronto tra un'interpolazione di Lagrange di ordine 1 (lineare) e un'interpolazione mediante la funzione sinc:

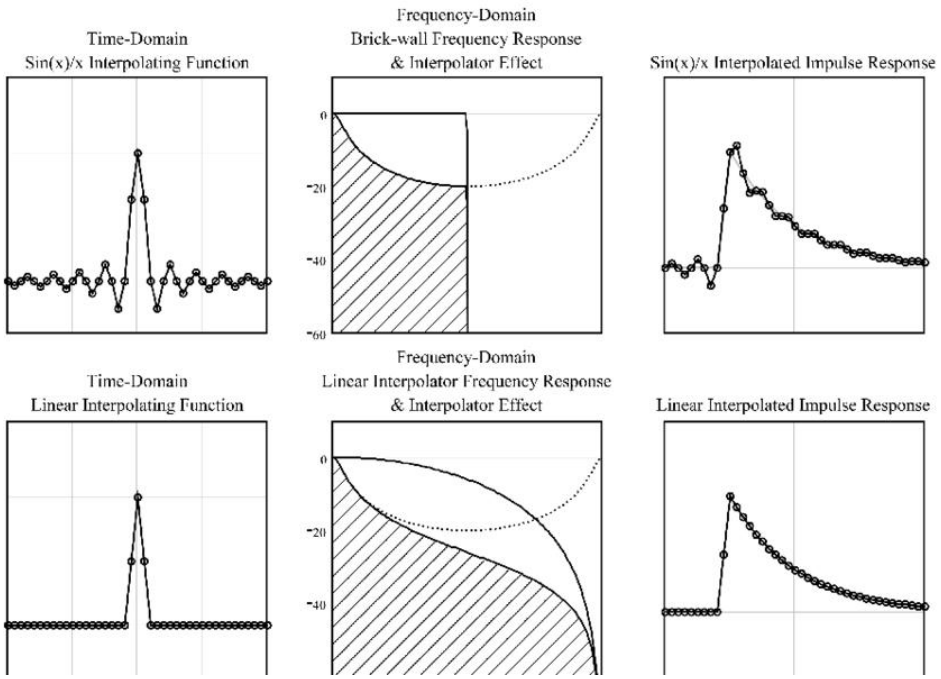

Figura 10: Confronto tra interpolazione lineare e tramite funzione sinc [20]

1.5 Protocollo MIDI

Il **MIDI ("Musical Instrument Digital Interface")** [21] è uno standard nato nel 1982. Esso include la descrizione di un protocollo di comunicazione, un'interfaccia digitale e dei connettori elettrici che, nell'insieme, consentono di mettere in comunicazione diversi strumenti musicali, computer e altri dispositivi audio per la registrazione, l'editing e la riproduzione di musica [22].

1.5.1 Livello hardware

Dal punto di vista hardware, lo standard MIDI prevede l'uso di cavi schermati dotati di connettori DIN a 5 poli, di cui solo i 3 poli centrali vengono utilizzati. Questi 3 poli sono collegati rispettivamente alla tensione di alimentazione (5 V) tramite una resistenza di 220 Ω, a massa e alla linea dati d'interesse. In particolare, le linee dati possibili sono:

- **MIDI IN**: consente al dispositivo di ricevere informazioni.
- **MIDI OUT**: consente al dispositivo di trasmettere informazioni.
- **MIDI THRU**: consente al dispositivo di ritrasmettere i dati ricevuti dalla propria porta IN verso un altro dispositivo.

In fig. 12 è mostrato un esempio di sistema di strumenti e dispositivi collegati tramite protocollo MIDI:

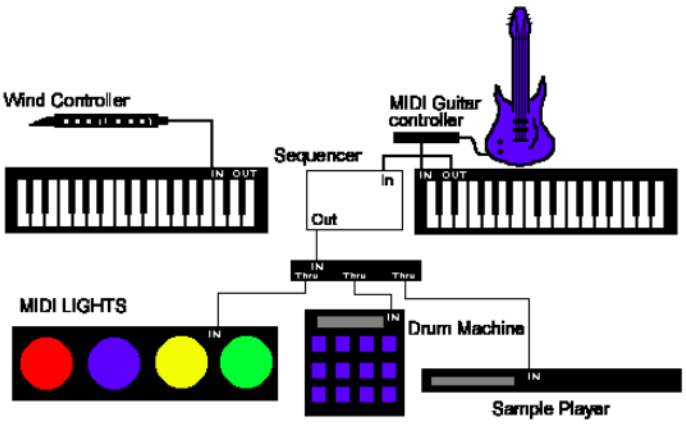

Figura 11: Esempio di sistema MIDI [22]

La grandezza elettrica di riferimento per codificare l'informazione scambiata è la corrente. In altre parole, un livello di corrente alto indicherà un 1 logico, mentre un livello basso indicherà uno 0 logico.
La scelta di usare la corrente, anziché la tensione, serve per rendere il protocollo sufficientemente robusto anche nel caso di utilizzo di cavi molto lunghi, come si hanno spesso nei concerti.

In fig. 12 è mostrato lo schema elettrico di una connessione tra due dispositivi MIDI.

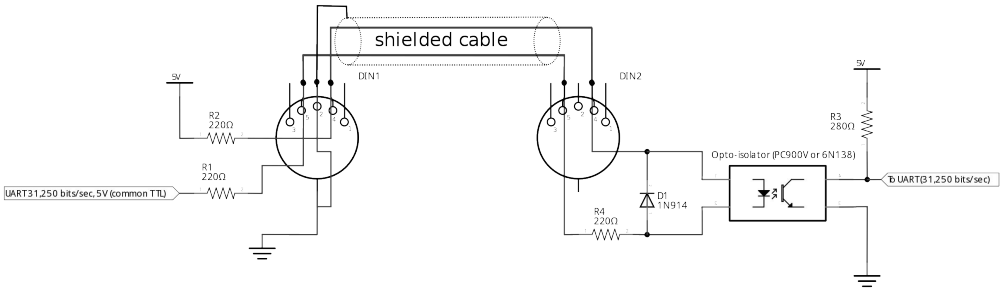

Figura 12: Schema elettrico di una connessione MIDI tra due dispositivi [23]

Come si può notare, sul dispositivo ricevitore sono presenti un optoisolatore (o fotoaccoppiatore) e un fototransistor. Il primo è un particolare diodo LED che, al passaggio di corrente, emette un segnale luminoso; mentre il secondo un dispositivo fotosensibile che capta tale segnale luminoso trasformandolo in un segnale elettrico. Ciò garantisce l'isolamento galvanico rispetto al circuito precedente e, quindi, il blocco di eventuali disturbi elettromagnetici condotti.

Se un dispositivo trasmettitore imprime sulla sua linea dati MIDI OUT una tensione alta (5V), allora nella maglia che collega il dispositivo trasmettitore al ricevitore non scorrerà corrente. Di conseguenza, il LED sarà spento, il fototransistor non condurrà e il resistore di pull-up porterà l'uscita alta (1 logico), fig. 13:

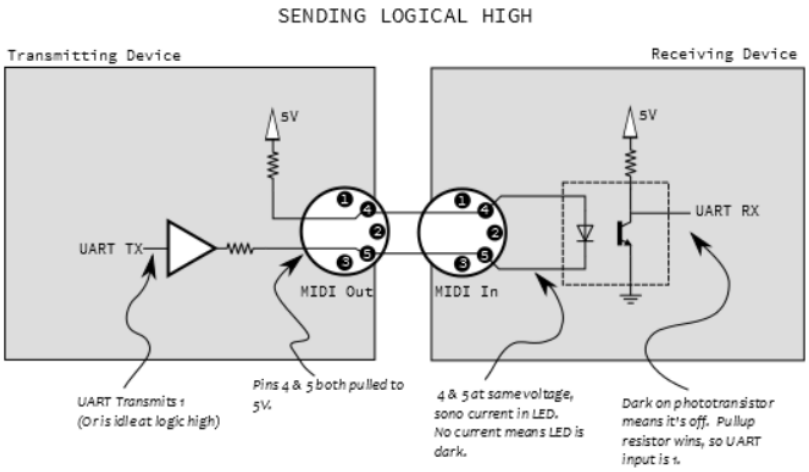

Figura 13: Trasmissione di un 1 logico tramite protocollo MIDI[24]

Se, invece, la tensione impressa è bassa (0 V), data una differenza di potenziale di 5 V e un'impedenza di a 220 Ω, nella maglia che collega trasmettitore e ricevitore, scorrerà una corrente di 5 mA. Di conseguenza, il LED sarà acceso e il fototransistor condurrà, portando l'uscita bassa (0 logico), fig. 14:

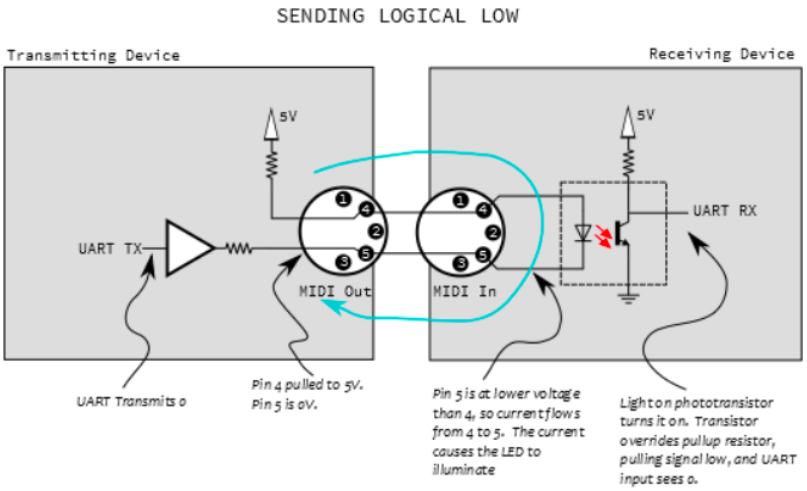

Figura 14: Trasmissione di uno 0 logico tramite protocollo MIDI [24]

1.5.2 Livello software

Ogni messaggio MIDI inizia con uno "Start Bit", durante il quale il livello di corrente scende a 0 mA. Successivamente, viene inviato un byte di dato, partendo dal suo MSB (Most Significant Bit). Infine, è presente uno "Stop Bit", durante il quale il livello di corrente ritorna a 5 mA, fig. 15:

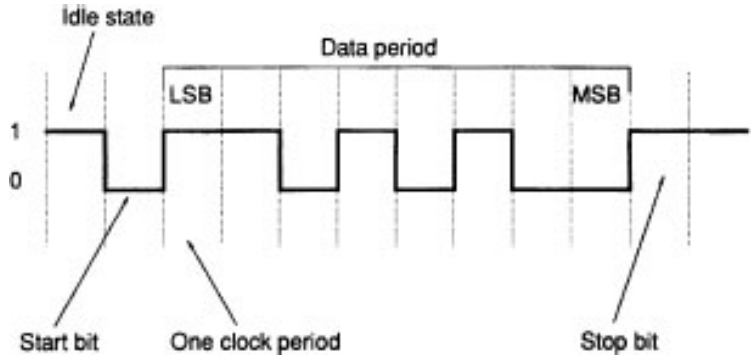

Figura 15: Modalità di trasmissione di un messaggio MIDI [22]

Ci sono due tipi principali di byte inviabili:

1. **Status Byte** → il quale ha l'MSB fisso a 1 ed è diviso in due nibble. Il primo nibble descrive quale evento è accaduto e il secondo indica il numero del canale su cui si lavora.

2. **Data Byte** → il quale ha l'MSB fisso a 0 e rappresenta un dato relativo ad un particolare messaggio MIDI ricevuto.

Tipicamente, i messaggi MIDI sono caratterizzati da una struttura contenente uno "Status Byte" seguito da un certo numero (generalmente 1 o 2) di "Data Byte".
Tra i messaggi MIDI più noti troviamo:

- **Note On** → indica l'inizio di riproduzione di una nota musicale ed è caratterizzato da:

  - Status Byte : 0x9i, con "i" canale
  - Data Byte 1 : 0xNN, codice esadecimale della nota suonata;
  - Data Byte 2 : 0xVV, valore esadecimale della velocity con la quale la nota è stata suonata

- **Note Off** → indica la fine di riproduzione (o rilascio) di una nota musicale ed è caratterizzato da:
  - Status Byte : 0x8i, con "i" canale
  - Data Byte 1 : 0xNN, codice esadecimale della nota suonata;
  - Data Byte 2 : 0xVV, valore esadecimale della velocity con la quale la nota è stata rilasciata
- **Control Change** → indica la modifica di un parametro (expression, volume, vibrato, ecc...) che consente la variazione del suono riprodotto ed è caratterizzato da:
  - Status Byte : 0xBi, con "i" canale
  - Data Byte 1 : 0xNN, codice esadecimale del parametro modificato
  - Data Byte 2 : 0xVV, valore esadecimale assegnato al parametro

In fig. 16, è mostrato un esempio di messaggio MIDI contenente un "Note On":

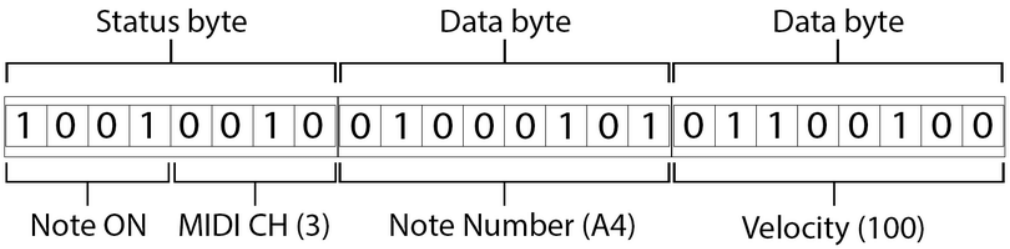

Figura 16: Esempio di messaggio MIDI [25]

2 Descrizione dell'architettura hardware

2.1 Specifiche di partenza

Le specifiche imposte per il lavoro sono:

- Struttura dei campioni caratterizzata da attacco, sostegno e rilascio (eventualmente ottenuto tramite "fade" del sustain).
- Generazione di tutte le note che rientrano nel range d'interesse, tramite lettura e interpolazione di campioni salvati in memoria Flash.
- Volume (guadagno) della singola nota dipendente dalla sua "velocity".
- Se una nota viene suonata mentre un'altra è in esecuzione, il suo attacco non deve essere riprodotto e i due suoni devono essere sottoposti a "cross-fade" di durata opportuna.
- Il contenuto armonico del suono può essere modificato attraverso l'impiego di un filtro passa-basso con frequenza di cut-off variabile con l'"expression".

2.2 Scheda embedded

Per la realizzazione del "sample player", si è scelto di utilizzare la board STM32WB5MM-DK, mostrata in fig. 17:

Figura 17: STM32WB5M Discovery Kit [26]

Il Discovery Kit è progettato dalla ST come una piattaforma completa per lo sviluppo e la sperimentazione di molte applicazioni.
Il cuore della board è sicuramente rappresentato dal microcontrollore STM32 W5MMG. Quest'ultimo compie le elaborazioni per mezzo di due processori a 32-bit, un Arm Cortex-M4 e un Arm Cortex-M0 dedicato al Bluetoot Low Energy (BLE).
Il Cortex-M4 è dotato di un'unità FPU (Floating-Point Unit) che consente di eseguire con maggiore efficacia operazioni matematiche su valori espressi in virgola mobile; ma anche di unità ART (Adaptive Real-time Accelerator) in grado di velocizzare il prelievo ("fetch") delle istruzioni in memoria da parte del processore, sfruttando una cache integrata. Accanto ai processori troviamo anche 1-Mbyte di memoria Flash dedicata al firmware e 256-Kbyte di SRAM.
Il microcontrollore appena descritto è, dunque, integrato sulla board, dove presenta collegamenti a molte periferiche. Tra queste, troviamo il programmatore ST-LINK V/2 collegato tramite UART e SWD, una memoria Flash (la S25FL128SDS MFV001 della Cypress) da 16 MB dotata d'interfaccia QUADSPI, alcuni LED e bottoni connessi alle GPIO, un display OLED con interfaccia SPI, un microfono MEMS che comunica tramite la SAI e molto altro.
Uno schema riassuntivo di queste connessioni è mostrato in fig. 18:

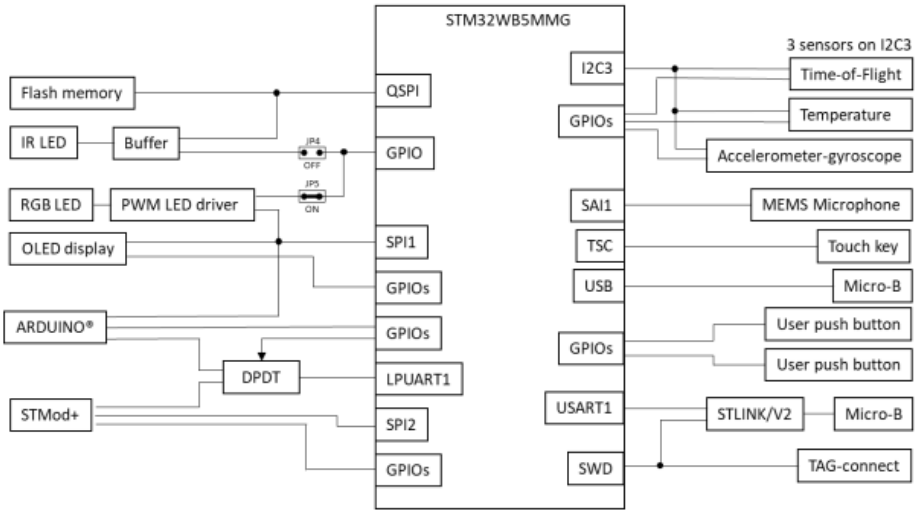

Figura 18: Diagramma a blocchi dell'hardware presente su STM32WB5MM-DK [26]

2.3 Convertitore D/A

Il convertitore digitale-analogico (CODEC) esterno scelto per fornire in uscita (tramite cavo con connettore jack) i segnali audio è il WM8960. Tale dispositivo è inserito in una scheda chiamata "WM8960 Audio HAT", fig. 19, e comunica con il microcontrollore tramite un'interfaccia I^2S. L'inizializzazione è stata fatta in modo da permettergli di lavorare ad una frequenza di campionamento di 48 kHz, con una risoluzione per i campioni di 24 bit.

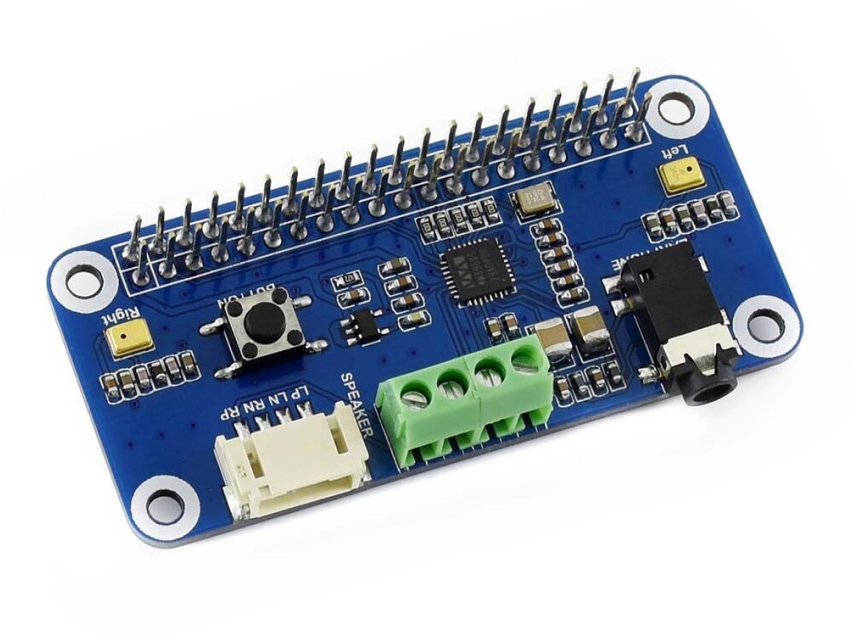

Figura 19: WM8960 Audio HAT [27]

2.4 ARTinoise Re.corder

Una volta generato il firmware, il microcontrollore, la memoria, il CODEC e altri dispositivi opportuni, saranno potenzialmente montati sulla nuova versione del prodotto "ARTinoise Re.Corder" [1], mostrato in fig. 20:

Figura 20: ARTInoise Re.corder [28]

Nella sua versione attuale l'*ARTinoise Re.corder* è apparentemente un flauto soprano acustico tradizionale come molti altri. Tuttavia, esso contiene l'elettronica sufficiente per essere collegato via Bluetooth Low Energy (BLE) ad un'app per smartphone, in grado di generare suoni di strumenti diversi (flauto, oboe, tromba, violino, batteria, ecc...).
Il limite presente e che s'intende superare è, quindi, che lo strumento non è in grado, in modalità digitale, di produrre suoni autonomamente.
In fig. 21, è mostrato uno schema riassuntivo delle parti dello strumento:

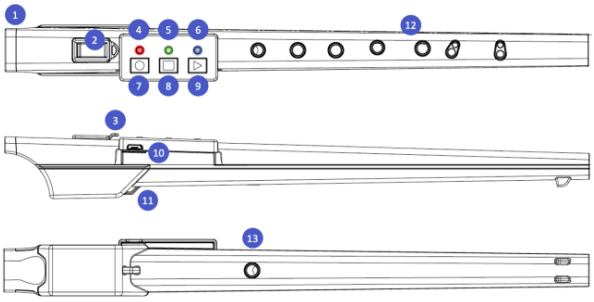

Figura 21: Schematico dell'ARTInoise Re.corder [1]

In particolare, la loro descrizione è riportata di seguito:

1. **Imboccatura e sensori di rilevazione/pressione** → Consente al musicista di soffiare aria nello strumento, generando note e dando espressività al suono. La presenza delle labbra dell'esecutore è rilevata dall'elettronica, in modo da consentire una continua calibrazione e abilitare, in caso, l'uscita. La forza con la quale si soffia determina il livello di pressione dell'aria incanalata, il quale viene rilevato da un sensore apposito.

2. **Labium e finestra** → Il "labium" [29] è una sorta di sottile cuneo sul quale l'aria incanalata impatta, per poi venire deviata in gran parte all'esterno tramite una finestra ed in piccola parte all'interno. Come conseguenza l'aria all'interno dello strumento musicale viene messa in vibrazione, generando un suono. La frequenza di base, oltre a dipendere dalle dimensioni del tubo o della camera di risonanza cambia a seconda della velocità dell'aria, che a sua volta dipende dall'intensità del soffio, dalla conformazione dell'imboccatura e dalle dimensioni del labium.

3. **Silenziatore** → Componente in plastica che viene inserita a tappare la finestra dalla quale esce il suono in modalità tradizionale. Viene usata quando il musicista vuole sfruttare lo strumento in modalità digitale.

4. **Led rosso** → Indica lo stato di ricarica dello strumento. Quando il cavo USB è connesso ad una fonte di alimentazione, il led rimarrà acceso finché la carica non sarà completata. Generalmente, invece, il led è spento e inizia a lampeggiare solo quando la batteria è quasi scarica.

5. **Led verde** → Possiede differenti funzionalità in base alla modalità di utilizzo dello strumento. In generale, però, ogni comando che si invia, sarà confermato da un rapido lampeggiamento del led verde.

6. **Led blu** → Indica lo stato della connessione wireless. All'accensione, tale led inizierà a lampeggiare e, soltanto quando un dispositivo mobile o un computer risulterà collegato allo strumento, il led rimarrà acceso fisso.

7. **Bottone circolare** → Ha la funzione principale di accendere e spegnere lo strumento. In particolare, per l'accensione, è necessaria una pressione di durata minima di 2 secondi. Alcune funzioni secondarie del bottone sono,

per esempio, quella di accoppiamento con dispositivi esterni e quella di registrazione di una traccia nell'app.

8. **Bottone quadrato** → Se premuto mantenendo pigiato il bottone circolare, consente di suonare sull'ottava più alta. Il bottone ha anche altre funzioni secondarie tra cui quella di interrompere la registrazione di una traccia nell'app.

9. **Bottone triangolare** → Se premuto mantenendo pigiato il bottone circolare, consente di suonare sull'ottava più bassa. Il bottone ha anche altre funzioni secondarie tra cui quella di riproduzione di una traccia registrata nell'app.

10. **Porta USB di ricarica** → Porta USB mediante la quale collegare lo strumento ad una fonte di alimentazione per ricaricare la batteria.

11. **Anello per montare il cinturino** → è possibile dotare lo strumento di un cinturino collegato a tale parte per evitare che questo cada mentre si suona.

12. **Fori delle note** → Come in un normale flauto, per riprodurre le note, bisognerà tappare dei buchi con le dita mentre si soffia aria.

13. **Foro posteriore da usare col pollice** → Tale foro consente, se tappato, di cambiare registro/ottava. Esso è dotato di due sensori che rilevano la condizione di semi-chiusura.

Oltre a ciò, all'interno dello strumento, possiamo trovare anche un accelerometro che è utilizzato per aggiungere espressività e sfumature differenti al suono generato.

3 Architettura di sintesi

3.1 Codice usato come base di partenza

Il codice usato come punto di partenza comprendeva:

- Il setup della periferica USB con endpoint MIDI.
- Il gestore e parser per i messaggi MIDI in ingresso.
- Il setup della SAI (serial audio interface) del DAC WM89060 con specificato protocollo di comunicazione I2S, frequenza di campionamento 48 kHz, risoluzione dei campioni pari a 24 bit e modalità stereo.
- L'inizializzazione della periferica DMA. Questa consente l'organizzazione un buffer "AUDIOdac_out[32][2]" di 64 campioni interi di 32-bit ognuno, suddiviso in 2 sotto-buffer da 32 campioni, in cui si alternano un campione dal canale destro e uno dal canale sinistro.
  Periodicamente, tramite l'innesco di un interrupt, il DMA va a leggere dalla RAM del microprocessore 32 campioni, eventualmente elaborati per mezzo di una funzione "AUDIOprocessDSP()", i quali popolano a turno il primo o il secondo sotto-buffer, a seconda della ISR invocata (HAL_SAI_TxHalf CpltCallback(...) oppure HAL_SAI_ TxCpltCallback(...)) e quindi dell'indirizzo specificato dal puntatore "OutBufPtr" (&AUDIOdac_out[0][0] oppure &AUDIOdac_out [HALFBUFFERSIZE][0]).
  Infine, dopo una rapida conversione float-int24, vengono mandati in ingresso al DAC.
- L'implementazione di una funzione "AUDIOprocessDSP()" di esempio la quale consente di mandare in uscita varie forme d'onda (es. sinusoide, forma d'onda triangolare, ecc...) in base al numero di pressioni effettuate sul pulsante della board.
- La costruzione di un piccolo scheduler che consenta di gestire efficacemente l'esecuzione alternata del controllo e parsing dei messaggi MIDI in arrivo e degli interrupt del DMA.

3.2 Gestione dei messaggi MIDI e strutture dati fondamentali

3.2.1 Controllo e parsing di un nuovo messaggio MIDI

L'architettura di sintesi, come già accennato in precedenza, entra visibilmente in funzione quando via USB, il microcontrollore riceve in ingresso un messaggio MIDI.
Il controllo di arrivo di un tale messaggio viene fatto periodicamente, nel loop principale del main, attraverso l'invocazione della funzione "`MidiMessageReady(...)`".
In particolare, se questa ritorna un valore booleano pari a "TRUE", vuol dire che c'è un messaggio MIDI pronto da gestire, quindi verrà invocata la funzione "`handle_MIDI_message()`" a tale scopo e subito dopo verrà revocata la validità del messaggio attraverso la funzione "`MidiInvalid_message(...)`" per evitare che le stesse elaborazioni vengano effettuate molteplici volte.

Nella prima parte della funzione "`handle_MIDI_message()`", vengono eseguite innanzitutto delle istruzioni generiche valide qualsiasi sia la casistica da gestire.
In particolare, dopo aver disabilitato per sicurezza gli interrupt (soprattutto interessano quelli del DMA), ipotizzando, per ora, che il messaggio MIDI in arrivo possa riguardare un evento "NoteOn", "NoteOff" o "ControlChange", si salvano i 3 byte ricevuti all'interno di variabili globali.
Se poi il messaggio MIDI ricevuto non dovesse coincidere effettivamente con uno degli eventi sopra citati, allora si riabilitano subito gli interrupt e si esce dalla funzione. Se invece, si è stato ricevuto un "NoteOn", un "NoteOff" o un "ControlChange", si eseguono istruzioni diverse a seconda del caso.
Infine, troviamo un'altra funzione "`clean_stack()`", il cui scopo sarà maggiormente chiaro più avanti. In linea generale però, è possibile affermare che serve per eliminare da uno stack (array) di strutture, quelle che contengono le informazioni relative alle note ormai concluse.
In fig. 22 è mostrato un diagramma di flusso riassuntivo degli step finora descritti:

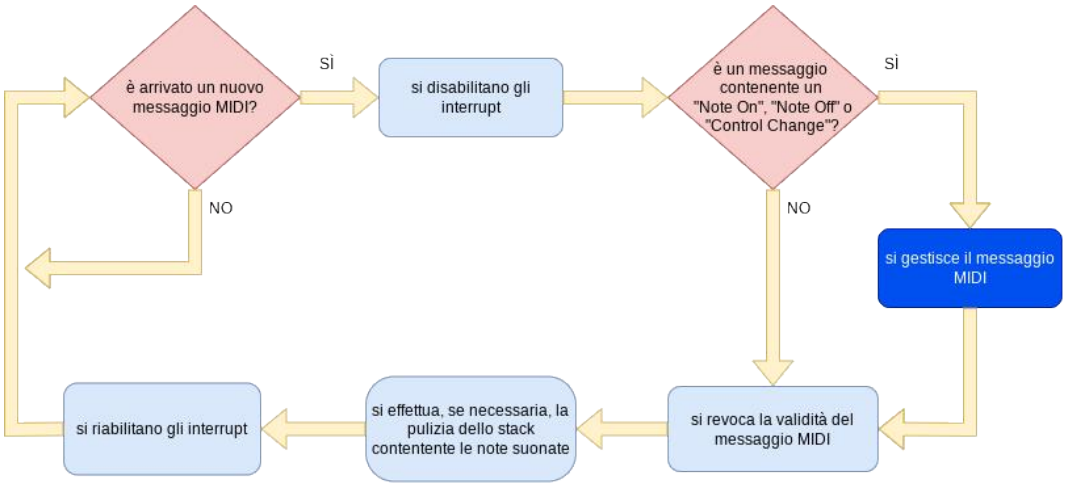

Figura 22: Loop d'istruzioni eseguito nel main

3.2.2 Gestione del Note On

In particolare, con un **"NoteOn"**, il flusso d'istruzioni eseguite è riportato in fig. 23.

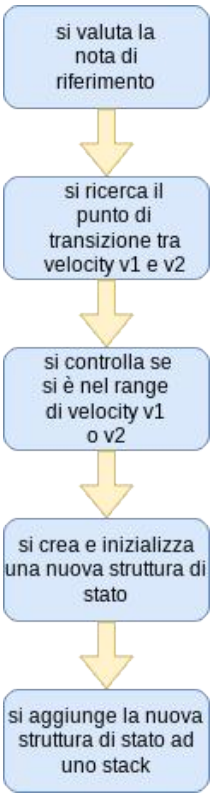

Figura 23: Gestione del Note On

Per il primo step di fig. 23, viene invocata la funzione `"get_reference_note()"`, in modo da verificare se tra i campioni delle note musicali disponibili, i cui numeri associati sono elencati nell'array `"known_notes[]"`, esistono quelli della nota musicale indicata nel "data byte 1" o, in caso negativo, trovare quelli di una possibile altra nota da interpolare.
La scelta di un' eventuale altra nota avviene prendendo la più vicina in frequenza a quella desiderata, quindi quella il cui codice numerico associato è a distanza minore rispetto alle altre.

Successivamente, viene effettuata una ricerca all'interno della memoria Flash per conoscere, in base alla nota di riferimento ottenuta, il punto di transizione tra la "velocity" v1 e la v2.
A partire poi da tale informazione e dal valore di "velocity" specificato nel "data byte 2" del messaggio, viene valutato attraverso la chiamata alla funzione "`get_velocity_range(...)`" se la nota suonata rientra nel range di "velocity" v1 o v2.

Infine, viene invocata la funzione "`initialize_state(...)`" all'interno della quale vengono inizializzati tutti i campi di una nuova struttura di tipo "`sample_state`". La definizione di tale struttura è riportata di seguito:

```
typedef struct{
  MidiType MIDI_event;
  uint8_t note_name;
  float reference_note_period;
  float real_note_period;
  float note_sampling_period;
  uint8_t note_velocity;
  sample_pointers note_pointers;
  float * sample_is_playing;
  enum{
    NOT_INIT = 0,
    ENDED = 1,
    LOOPING = 2,
    STOPPING = 3,
  }stage_flag;
  enum{
    NONE = 0,
    FADING_IN = 1,
    FADING_OUT = 2
  }stage_mod;
  int ramp_fade_cnt;
  int ramp_rel_cnt;
  float D;
  float D_step;
  float gain;
  float filter_weigth;
  float xold;
  float yold;
}sample_state;
```

Il significato dei vari campi della struttura il seguente:

1. **MIDI_event** → Indica il nome dell'evento MIDI che si è verificato facendo riferimento al tipo enum "MidiType_" (es. NoteOn, NoteOff, ecc...).

2. **note_name** → Rappresenta il codice esadecimale relativo alla nota musicale che è stata suonata o interrotta (es.60 per il DO4, 69 per il LA4, ecc...).

3. **reference_note_period** → Coincide con il periodo della nota musicale disponibile tra quelle in memoria che è più vicina come frequenza alla nota musicale suonata.
   I campioni di tale nota di riferimento verranno interpolati al fine di ottenere quelli della nota desiderata. Questo viene calcolato a partire della conoscenza del periodo della nota LA4 (1/440 s) e dalla distanza da essa in termini di semitoni.

4. **real_note_period** → Coincide con il periodo della nota musicale realmente suonata e viene calcolato con la stessa procedura di quello della nota di riferimento.

5. **note_sampling_period** → Rappresenta il periodo di campionamento relativo alla nota musicale suonata (es. se la frequenza di campionamento è 48 kHz, tale periodo sarà $1/48000 \simeq 20.8\ \mu s$).

6. **note_velocity** → È un numero da 0 a 127 che indica la "forza" con la quale una nota viene suonata e, di conseguenza, il volume con la quale essa deve essere riprodotta. La scala utilizzata è tipicamente logaritmica ed è caratterizzata da un guadagno di 0 dB se la "velocity" è pari a 127 (cioè massima) e -30 dB se è pari a 1 (cioè minima).
   La "velocity" pari a 0 non è contemplata nei "NoteOn", ovvero è associata dal MIDI ad un "NoteOff".

7. **stage_flag** → Rappresenta la fase di riproduzione attuale della nota musicale suonata, la quale viene scelta tra le seguenti opzioni:

   (a) **LOOPING**→ La nota sta suonando l'attacco o il loop del sustain e ancora non è arrivato il "NoteOff".

   (b) **STOPPING** → È arrivato il "NoteOff" quindi la nota si prepara, in base al tipo di loop, ad eseguire il rilascio o il "fade" (diminuzione graduale del guadagno) del sustain. La durata attuale del fade è impostata a circa 50 ms ma è facilmente modificabile cambiando il valore del `#define release_len 2399.f` con il numero di campioni che, considerando una frequenza di campionamento di 48 kHz, coprirebbero la durata desiderata per tale fase.

(c) **ENDED**→ La nota ha terminato la sua esecuzione, quindi la struttura ad essa associata è prossima ad essere rimossa dallo stack.

(d) **NOT_INIT**→ La struttura non è stata inizializzata, ovvero non contiene informazioni valide su una nota da suonare.

8. **stage_mod** → Rappresenta la modalità con la quale viene gestita dal punto di vista del guadagno e della fase di riproduzione una nota di cui è arrivato il "NoteOn".
   Le possibilità sono:

   (a) **NONE** → La nota viene suonata al momento senza eseguire nessun tipo di "fade", ovvero con un guadagno costante che dipende solo dalla struttura del campione e dalla velocity.

   (b) **FADING_IN** → La nota è stata suonata nel mentre era in riproduzione un'altra nota (legato), quindi, per creare un effetto più naturale, s'introdurrà senza eseguire l'attacco ed effettuando un fade (aumento graduale del guadagno) del sustain. Analogamente, al fade in uscita di cui sopra, anche qui la durata è impostata per circa 50 ms ma è facilmente modificabile cambiando il valore del `#define fading_len 2399.f` con il numero di campioni che, considerando una frequenza di campionamento di 48 kHz, coprirebbero la durata desiderata per tale fase.

   (c) **FADING_OUT**→ La nota è ancora in riproduzione ma nel frattempo ne è arrivata una nuova quindi si procede su di essa con un fade (diminuzione graduale del guadagno) del sustain. Per la durata, valgono le stesse identiche considerazioni fatte nel caso di "FADING_IN".

9. **note_pointers** → È una struttura di tipo "sample_pointers" contenente le seguenti informazioni:

   - **note_number** → È il codice esadecimale associato al campione di nota in questione.
   - **velocity_range** → Rappresenta il range di "velocity" a cui sono associati i campioni della nota in questione, ovvero, in base all'enum omonimo, "v1" per volumi bassi o "v2" per volumi più alti. Il confine tra "v1" e "v2" dipende dalla nota specifica ed è indicato all'interno

di una struttura di tipo "note_velocity_transition", dal campo "transition".
L'unico altro campo presente in tale struttura è il campo "note_number" con il codice numerico relativo alla nota in questione.

- **base_pointer** → È il puntatore all'inizio del vettore contenente i campioni di attacco, sostegno ed, eventualmente, rilascio relativi alla nota in questione.
- **sustain_start_offset, sustain_end_offset, release_start_offset, release_end_offset** → Sono gli offset di campioni (float) da aggiungere a "base_pointer" per ottenere il puntatore all'inizio o alla fine del sustain o release rispettivamente;
- **loop_type** → Indica la struttura di loop disponibile per la nota in riproduzione. In particolare, si possono avere due possibilità:
  (a) **LOOP_AND_RELEASE** → Si hanno a disposizione i campioni per il rilascio, quindi nel momento in cui arriva il "NoteOff", vengono riprodotti quelli.
  (b) **LOOP_AND_FADING** → Non si hanno a disposizione i campioni per il rilascio, quindi nel momento in cui arriva il "NoteOff", si procedere con un fade (diminuzione graduale del guadagno) del sustain.

10. **ramp_fade_cnt, ramp_rel_cnt** → Sono i contatori dei campioni suonati durante le fasi di fade o release rispettivamente. Quando tale contatore arriva a "fading_len" o "release_len" (cioè attualmente a 2399), si ha un cambio di stato nel codice.

11. **sample_is_playing** → È il puntatore al vettore contenente i campioni del segnale che si sta riproducendo attualmente.

12. **D** → È il valore attuale del parametro D utilizzato all'interno dell'algoritmo d'interpolazione per determinare la distanza tra un campione da calcolare e il successivo

13. **D_step** → È il singolo incremento di D, partendo una nota musicale di riferimento. Il suo valore determina, dunque, anche quanto la lunghezza del vettore contenente i reali campioni da suonare si discosta da quella del

vettore contenente i campioni della nota di riferimento. In particolare, la lunghezza del vettore della nota di riferimento va divisa per "D_step" al fine di ottenere la lunghezza di quello della nota desiderata.

14. **gain** → È il guadagno da associare ad una nota musicale da riprodurre ed è calcolato in base al valore di velocity ricevuto.

    Il guadagno da dare alla nota, viene calcolato attraverso i seguenti step:

    - Alla "velocity" specificata nel messaggio MIDI, si associa linearmente un valore tra -30 dB (v=1) e 0 dB (v=127).
    - Il guadagno in dB viene tradotto in lineare, ottenendo un valore compreso tra 0.03 e 1.

    L'andamento del guadagno in funzione della velocity è mostrato in fig. 24:

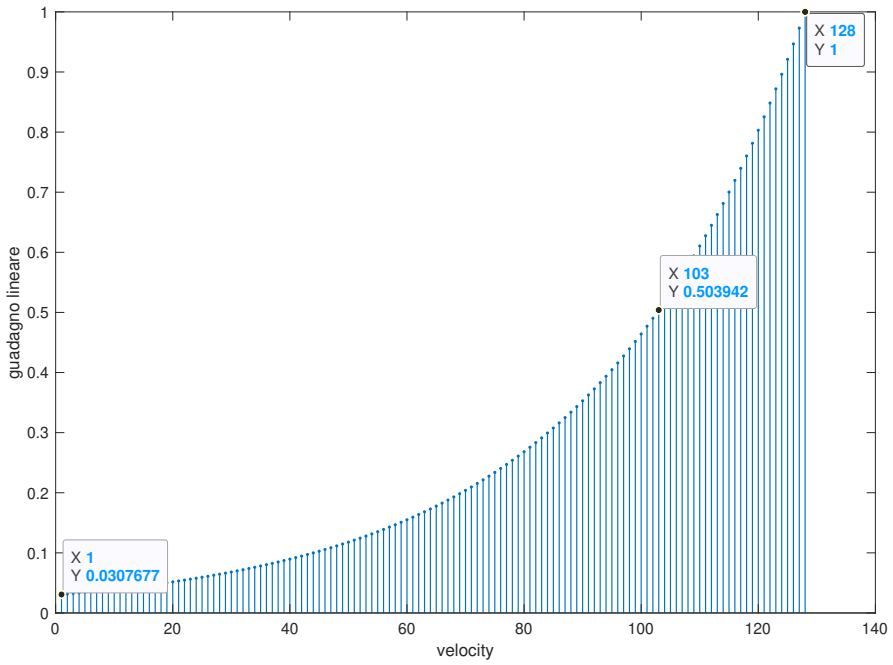

Figura 24: Andamento del guadagno della nota in funzione della velocity

15. **filter_weight, xold, yold** → Sono informazioni su un filtro passa-basso con frequenza di cut-off variabile in funzione dell'expression che viene applicato sulle varie note suonate. In particolare, le informazioni riguardano ri-

spettivamente, come si vedrà meglio successivamente, il parametro variabile "c" usato nei pesi, lo stato precedente dell'ingresso e quello dell'uscita.

Lo scopo del filtro è quello di cambiare il contenuto armonico della nota suonata in base al valore di "expression" riscontrato; quindi rendere il suono più scuro e cupo nel caso di "expression" basse e più chiaro e squillante nel caso di "expression" alte, senza mai togliere la fondamentale. In particolare, la frequenza di cut-off del filtro può variare da un minimo di 2000 Hz, nel caso di velocity pari a 1, fino ad un massimo di 10000 Hz, nel caso di velocity pari a 127.
Il filtro impiegato è un semplice filtro a singolo polo, la cui funzione di trasferimento discreta è stata ottenuta partendo dalla funzione di trasferimento analogica di un filtro RC:

$$H(s) = \frac{1}{1 + sRC} \tag{14}$$

e applicando ad essa la trasformazione bilineare, come spiegata [30], ovvero usando il mapping:

$$s = \frac{2}{T} \frac{1 - z^{-1}}{1 + z^{-1}} \tag{15}$$

In [30], si spiega come (15) comporti un mapping dell'asse immaginario positivo e negativo del piano S rispettivamente sulla metà superiore e inferiore della circonferenza unitaria del piano Z. Inoltre, il semipiano sinistro del piano S, si mapperà all'interno della circonferenza unitaria del piano Z, mentre quello destro all'esterno della stessa.
Da ciò, si può intuire come tale tecnica di trasformazione S-Z preservi la stabilità. Inoltre, rispetto ad altre tecniche come l'invarianza all'impulso, non presenta problemi di aliasing.
L'unico inconveniente, dunque, sarebbe rappresentato dalla compressione dell'asse delle frequenze, la quale causa una distorsione di fase e un'alterazione della posizione delle frequenze che determinano le transizioni tra una banda e l'altra.
Per attenuare quest'ultima problematica, sarebbe possibile agire mediante "prewarping", ovvero predistorcere tali frequenze già al principio.

Il risultato ottenuto applicando (15) è il seguente:

$$H[z] = \frac{\sum_{k=0}^{1} b_k \cdot z^{-k}}{\sum_{k=0}^{1} a_k \cdot z^{-k}} = \frac{1 + z^{-1}}{(1 + \frac{2 \cdot RC}{T}) + (1 - \frac{2 \cdot RC}{T}) z^{-1}} \tag{16}$$

Se si indica, per semplicità:

$$c = \frac{2 \cdot RC}{T} = \frac{2}{T \cdot \omega_{\text{cut-off}}} \tag{17}$$

si ha:

$$H[z] = \frac{1 + z^{-1}}{(1 + c) + (1 - c) z^{-1}} \tag{18}$$

Dalla (18), i coefficienti dell'equazione alle differenze seguente:

$$\sum_{k=0}^{1} a_k \cdot y[n-k] = \sum_{k=0}^{1} b_k \cdot x[n-k] \tag{19}$$

risultano essere $b_0 = 1,\ b_1 = 1,\ a_0 = 1 + c,\ a_1 = 1 - c$.

Quindi si può scrivere che l'uscita al tempo "n" è data da:

$$y[n] = \frac{x[n] + x[n-1] - (1 - c) \cdot y[n-1]}{(1 + c)} \tag{20}$$

L'equazione (20) è implementata nella funzione `"filtro_dinamica(...)"` e viene, quindi, utilizzata per il filtraggio dei singoli campioni durante la riproduzione.
Come si è accennato precedentemente, gli stati precedenti x[n-1]("xold"), y[n-1]("yold") e c ("filter_weight") sono campi della struttura di tipo "sample_state", dunque la loro inizializzazione avviene proprio nella funzione `"initialize_state(...)"`.
Qui, gli stati "xold" e "yold" semplicemente vengono posti a 0 per poi essere aggiornati dinamicamente durante la riproduzione.
Per quanto riguarda il parametro "c", invece:

- Si crea innanzitutto un mapping non lineare secondo la funzione $y = x^2$ tra la "expression" e la $f_{\text{cut-off}}$, facendo in modo che quest'ultima rientri nel range 2000-10000 Hz.
- Si applica la eq. (17) per ottenere il valore del peso "c", tenendo conto che $\omega_{\text{cut-off}} = 2\pi \cdot f_{\text{cut-off}}$

L'andamento della frequenza di cut-off in funzione dell'expression è riportato di in fig. 25:

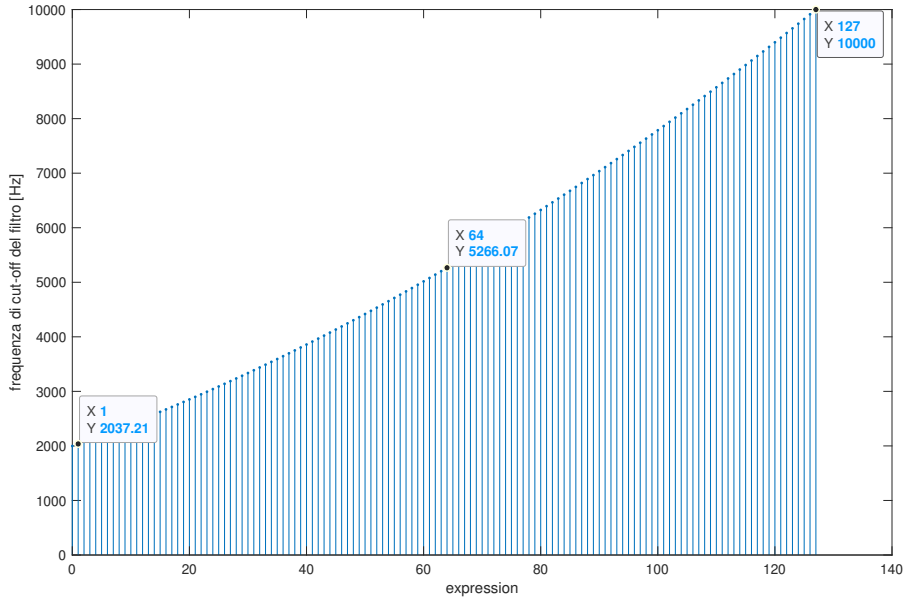

Figura 25: Andamento della frequenza di cut-off del filtro in funzione dell'expression

Successivamente, La funzione "`calcolo_filtro_expression(...)`", che effettua tali calcoli, viene chiamata la prima volta all'avvio con un valore di "expression" (cc_value) di default pari a 64 poi ogni volta che arriva un "Control Change" di tipo 11. Ciò serve ad aggiornare il valore di "c" da associare ai campi "filter_weight" delle strutture di stato.

L'ultima cosa che viene fatta dopo l'invocazione di "`initialize_state(...)`" è la chiamata alla funzione "`add_state(...)`", nella quale si prende la struttura di tipo "`sample_state`" appena inizializzata e la si aggiunge nel primo posto libero di uno stack (array di strutture "`sample_state`") di lunghezza indicata dal `#define stack_len` (per ora posto a 5).
In tale stack, dovranno essere presenti istante per istante tutte le strutture con le informazioni relative alle note in riproduzione, in modo tale da poter calcolare per ognuna di esse un valore e, infine, sommarli tutti insieme per ottenere il campione finale.

Prima di effettuare il reale inserimento della struttura tramite la funzione "`push_struct(...)`", vengono eseguite anche altre operazioni.
In particolare, se già nell'array sono presenti altre strutture, le note relative a queste strutture vengono cambiate di modalità passando a "FADING_OUT". Ciò forzerà la loro progressiva diminuzione del guadagno fino ad arrivare alla loro estinzione. Inoltre, se ciò accade, verrà anche messa la variabile "flag_no_attack" a 1, in modo tale da cambiare anche la modalità della nota in entrata da "NONE" a "FADING_IN", consentendo di iniziare la sua riproduzione direttamente dal "sustain" con un aumento graduale del guadagno.
Come già accennato in precedenza, la "pulizia" dello stack dalle strutture che non servono più, verrà effettuata poi nel loop del main, attraverso l'invocazione della funzione "`pop_struct (...)`".

3.2.3 Gestione del Note Off

Nel caso di **"NoteOff"**, invece, le istruzioni eseguite sono riassunte in fig. 26.

Figura 26: Gestione del Note Off

In poche parole, viene invocata la funzione "`stop_note(...)`" che ricerca all'interno dello stack la struttura contenente il nome della nota per cui è arrivato il "NoteOff" cambiando il suo stato a "STOPPING". Ciò forzerà, in base al tipo di loop, la riproduzione del rilascio o la diminuzione graduale del guadagno del "sustain", fino ad arrivare comunque all'estinzione di tale nota.

3.2.4 Gestione del Control Change

Infine, nel caso di gestione di un **"ControlChange"**, le istruzioni eseguite sono riassunte in fig. 27.

Figura 27: Gestione del Control Change

In pratica, come già accennato in precedenza, se il CC è ti tipo 11, viene chiamata la funzione `"calcolo_filtro_expression()"` per ricalcolare il peso "c" in base al nuovo valore di "expression" settato e poi `"aggiornamento_filtro_expression(...)"` per aggiornare il campo "filter_weight" di tutte le strutture di stato nello stack.

3.3 Elaborazione dei campioni

3.3.1 Loop di istruzioni eseguite ad ogni interrupt del DMA

Il loop d'istruzioni eseguito per ciascuna delle 16 coppie di campioni sinistro e destro, nella funzione "AUDIO_process_DSP()" chiamata periodicamente ad ogni interrupt del DMA, è sintetizzato in fig. 28.

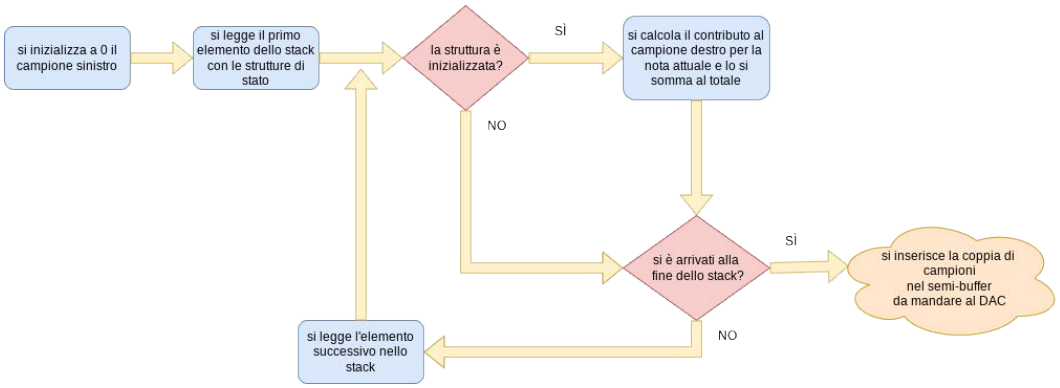

Figura 28: Elaborazione dei campioni

Come si può notare, per i campioni del canale sinistro (`"left_channel_samples[]"`), essendo lo strumento da simulare monofonico, semplicemente viene effettuata un'assegnazione del valore 0; mentre per i campioni del canale destro(`"right`

`_channel_samples[]`") è presente un loop che ha come obiettivo quello di scorrere tutto lo stack e, per ogni struttura non vuota che trova, ricavare, attraverso la funzione "`calcolo_campioni()`" uno dei contributi da sommare insieme per ottenere il valore definitivo, fig. 29.

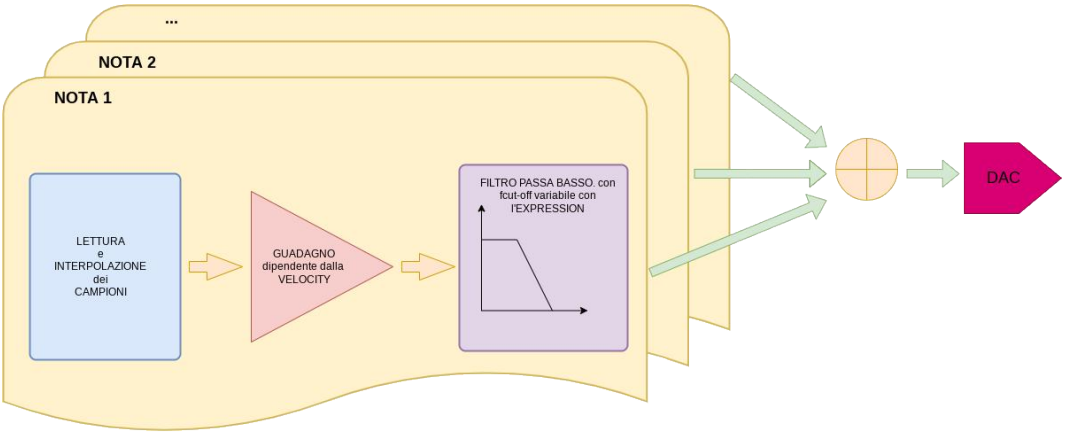

Figura 29: Calcolo campioni del canale destro

Successivamente, i 16 valori raccolti per gli array "`right_channel_samples[]`" e "`left_channel_samples[]`" vengono convertiti in "int24" e utilizzati per riempire una specifica metà dell'array "`AUDIOdac_out []`[]" (dimensioni 32x2), fig. 30.

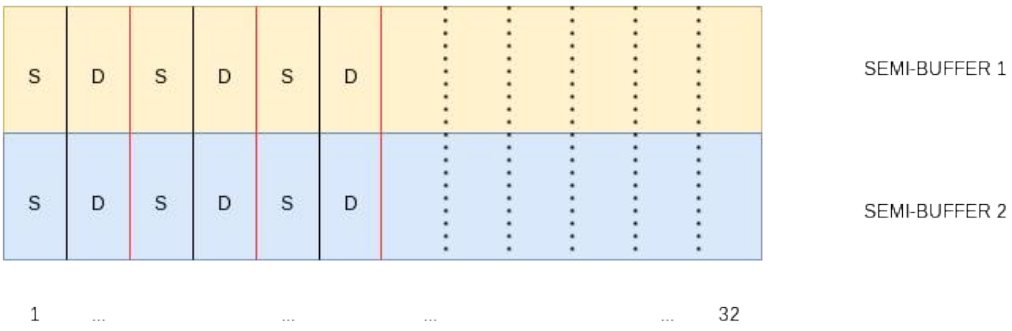

Figura 30: Doppio buffer dei campioni

L'indirizzo di partenza di tale metà è indicato dalla variabile "OutBufPtr". In particolare, sommando a tale variabile una quantità dispari, si otterrà l'indirizzo a cui scrivere il singolo campione destro; mentre sommandogli una quantità pari si otterrà quello a cui scrivere il singolo campione sinistro.

3.3.2 Lettura e interpolazione dei campioni

Le istruzioni eseguite all'interno della funzione "`calcolo_campioni(...)`" hanno come obiettivo quello di calcolare il campione da riprodurre per una particolare nota, a partire dalla sua nota di riferimento, sfruttando un'interpolazione di Lagrange di ordine N [18].
Allo scopo di ridurre i tempi di elaborazione e dare la possibilità di riproduzione di almeno 2 note contemporaneamente, è stato impiegato un interpolatore di ordine 1 (o lineare). L'unico appunto da fare rispetto alla teoria di base dell'algoritmo d'interpolazione è che qui se il campione x_0 coincide con l'ultimo disponibile e non si ha a disposizione il campione x_1, quest'ultimo viene a sua volta calcolato attraverso la stessa procedura d'interpolazione prima di procedere oltre.
All'interno del codice, seguendo gli stessi principi, è stata creata anche una funzione a parte per realizzare un'eventuale interpolazione del terzo ordine. Per utilizzarla, basta cambiare il nome della funzione invocata nella "process".
Il campione ottenuto dall'interpolatore viene poi moltiplicato per il parametro "gain" della struttura, ovvero un guadagno variabile con la velocity.
Successivamente, avviene il filtraggio del campione ottenuto tramite il filtro con frequenza di cut-off variabile in funzione dell'expression, come indicato in voce 14.

Infine, in base ai valori dei campi "stage_flag" " stage_mod" della struttura, quindi alla fase e alla modalità di riproduzione della nota, vengono eseguite istruzioni diverse. In particolare:

- se si è nella fase di **LOOPING** → semplicemente si incrementa il campo D del valore contenuto nel campo D_step della stessa struttura e se per caso questo dovesse assumere un valore maggiore di 1, si diminuisce iterativamente di un'unità il valore finché esso non ritorna ad essere minore o uguale a 1. Per ogni diminuzione si aumenta di uno step il puntatore del campione della nota di riferimento usata per la riproduzione. Se per caso poi tale puntatore dovesse superare il valore massimo, semplicemente lo si rimette pari al valore d'inizio del loop di sustain.

- se si è nella fase di **STOPPING** → prima di effettuare le istruzioni già descritte per la fase di LOOPING, se il tipo di loop è "LOOP_AND_FADING", si

moltiplica il campione di "sustain" ottenuto per un guadagno che,ad ogni giro, scenderà di 1/release_len. Quando tale guadagno arriverà a 0 vorrà dire che la nota avrà terminato la sua riproduzione, quindi potrà passare nello stato "ENDED" ed estinguersi del tutto. Se, invece, il tipo di loop è "LOOP_AND_RELEASE", si dovrà semplicemente controllare se si è arrivati alla fine dei campioni di rilascio e, solo in tal caso, procedere con il passaggio di stato a "ENDED" ed estinzione della nota.

- se è attiva la modalità di **FADING_OUT** → vale un discorso analogo al caso di "STOPPING" con loop di tipo "LOOP_AND_FADING".

- se è attiva la modalità di **FADING_IN** → prima di effettuare le istruzioni già descritte per la fase di LOOPING, si moltiplica il campione di "sustain" ottenuto per un guadagno che,ad ogni giro, salirà di 1/fading_len. Quando tale guadagno arriverà a 1, vorrà dire che sarà terminato il fade in ingresso e si potrà passare alla modalità "NONE" con guadagno costante.

4 Implementazione su target

La board STM32WB5MM-DK utilizzata è dotata di una memoria NOR-flash da 128 Mbit, ovvero la S25FL128SDSMFV001 della CYPRESS, la quale è in grado di comunicare con il microcontrollore STM32WB5MMG tramite il protocollo seriale QSPI.
Tale memoria aggiuntiva è stata, dunque, utilizzata per il salvataggio dei campioni audio necessari all'architettura di sintesi.

4.0.1 Configurazione del QSPI

All'interno dell'interfaccia IOC messa a disposizione dalla STM32CubeIDE, è stato possibile configurare la periferica QUADSPI, in modo tale che essa comunicasse con la memoria flash a bordo della scheda.

A tale scopo, i pin impiegati sono stati:

- **QUADSPI_SCLK** → PA3 (pin 56) : rappresenta il segnale di clock di riferimento.
- **QUADSPI_NCS** → PD3 (pin 70) : rappresenta il "chip-select". Quando esso va basso, il dispositivo connesso (qui memoria flash) è selezionato e pronto per la comunicazione.
- **QUADSPI_BK1_IO0, QUADSPI_BK1_IO1, QUADSPI_BK1_IO2, QUADSPI_BK1_IO3** → PB9 (pin 11), PD5 (pin 80), PD6 (pin 81), PD7 (pin 67): indicano le 4 linee dati attraverso le quali la periferica scambia informazioni con il dispositivo connesso (qui memoria flash).

i quali sono stati inizializzati tutti senza né pull-up né pull-down.

La modalità di utilizzo deL QSPI scelta è stata quella con:

- **CPOL = 0** → clock al livello logico basso quando è a riposo.
- **CPHA = 0** → campionamento dei dati sul fronte di salita.

la quale è compatibile con la NOR-flash con la quale si vuole comunicare.

Tenendo conto poi che il clock del microcontrollore va a 64 MHz, è stato impostato un prescaler pari a 3, in modo da ottenere una frequenza di comunicazione per ciascuna delle 4 linee di 64 MHz/(3+1) = 16 MHz. Se si tiene conto che i dati viaggiano su 4 linee, la frequenza in realtà sarà il quadruplo ovvero 64 MHz, il che significa una **velocità di trasferimento di 64 Mbit/s** circa, se si trascura la presenza di eventuali "dummy-byte" scambiati.

Infine, essendo la NOR-flash da 16 MB totali, si è indicato come parametro "FlashSize" 23, in modo da avere a disposizione 23+1 = **24 bit per indirizzare tale memoria**.

Fatto ciò, nel codice è stata creata una funzione in modo da inizializzare il QSPI in **modalità "Memory-mapped"**. In tal modo, lo spazio di memoria della NOR-flash verrà considerato all'interno di quello del micro a partire dall'indirizzo 0x900 0000.
Una volta invocata tale funzione, dunque, se dal firmware si vorrà accedere al contenuto di uno degli indirizzi della memoria, es. 0x10000, basterà aggiungere prima l'offset 0x9000000 per poi usare la normale aritmetica dei puntatori, ovvero float value = *((float*)0x90010000).

4.0.2 Preparazione dei file binari e caricamento dei dati

I campioni audio da utilizzare sono stati forniti dall'azienda in formato SF2, dunque è stato possibile aprire tale file con un soundfont editor come Polyphone.
Osservando in fig. 31 la schermata dell'editor per un generico campione, è possibile notare come vengano fornite diverse informazioni utili, tra cui la frequenza di campionamento, qui 48 kHz, e anche gli indici dei campioni, a partire da 0, che costituiscono l'inizio e la fine del loop del sustain, cioè qui 68156 e 78939.
Nel caso presente, la fine del loop coincide anche con la fine del segnale audio, quindi non sono forniti i campioni relativi al rilascio.

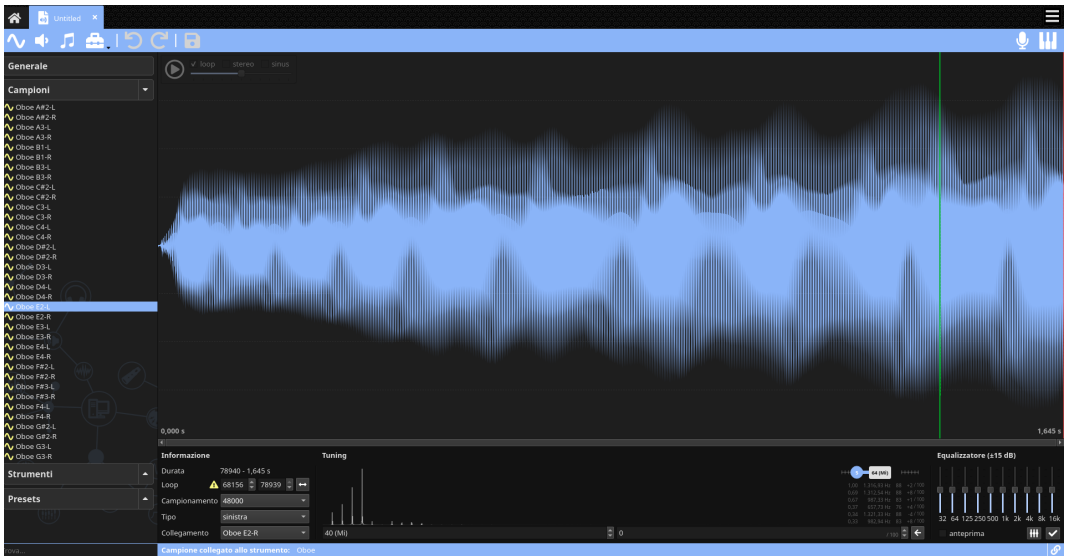

Figura 31: Schermata dal soundfont editor Polyphone

Tramite la toolbar in alto è stato possibile esportare i campioni d'interesse in formato wav, per poi aprirli in matlab e generare un file binario "campioni.bin" con l'elenco dei valori "float" da caricare in memoria.

Quando in sezione 3.2.2 si è parlato dei campi che compongono una struttura di tipo "sample_state", si è detto che questa comprende, a sua volta, un'altra struttura di tipo "note_pointers". Quest'ultima dovrebbe contenere le informazioni acquisite, grazie a Polyphone, riguardo alla posizione in memoria dei campioni della nota di riferimento usata. Anche tali strutture, come le liste dei campioni, andranno salvate nella memoria Flash.
In particolare, la scelta progettuale è stata quella di dedicargli la prima pagina in modo da essere facilmente rintracciabili, come illustrato di seguito:

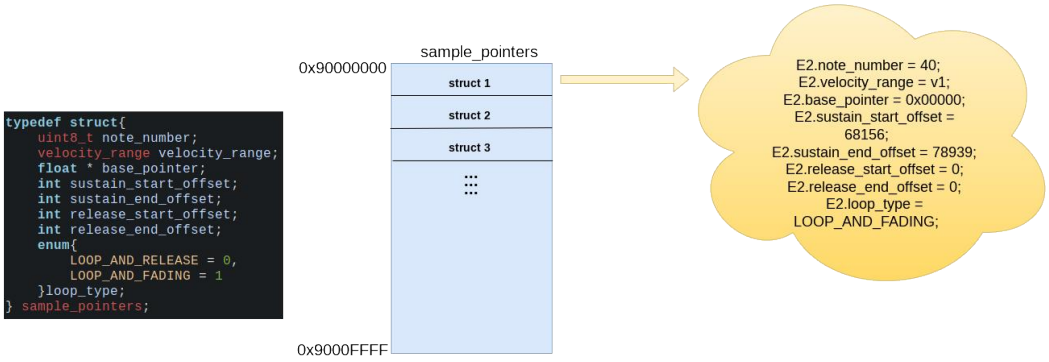

Figura 32: Struttura della prima pagina della NOR-Flash

Affinché sia possibile capire, inoltre, in base al valore di velocity ricevuto tramite messaggio MIDI, se occorre utilizzare i campioni a v1 o v2, è necessario tenere in memoria anche le informazioni riguardanti il punto di transizione tra v1 e v2 per ciascuna delle note di riferimento. Tali informazioni, contenute all'interno di strutture di tipo "note_velocity_transition", sono state salvate nella seconda pagina di memoria, come illustrato di seguito:

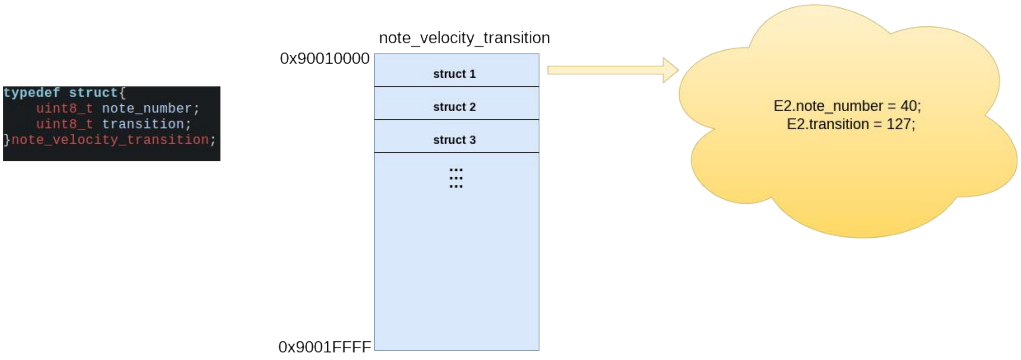

Figura 33: Struttura della seconda pagina della NOR-Flash

I file binari "puntatori.bin" e "transizioni.bin" con l'elenco dei dati relativi a tali strutture sono stati creati manualmente attraverso un programma C opportuno.

Le pagine della memoria flash dalla terza in poi, infine, si è scelto di dedicarle alla lista dei campioni relativi alle note di riferimento, come illustrato di seguito:

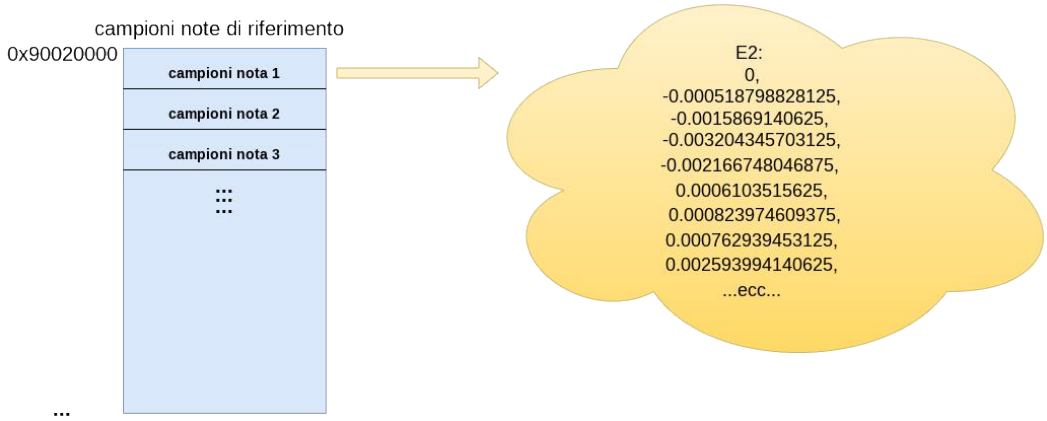

Figura 34: Struttura delle pagine della NOR-Flash dalla terza in poi

Per il caricamento effettivo dei file binari in Flash, è stato usato il software STM32Cube Programmer. Qui, infatti, dopo aver collegato e connesso la board, grazie ad un "external loader", è stato possibile popolare la memoria Flash esterna,a partire da un indirizzo specificato, coi dati relativi ad un file binario salvato su pc:

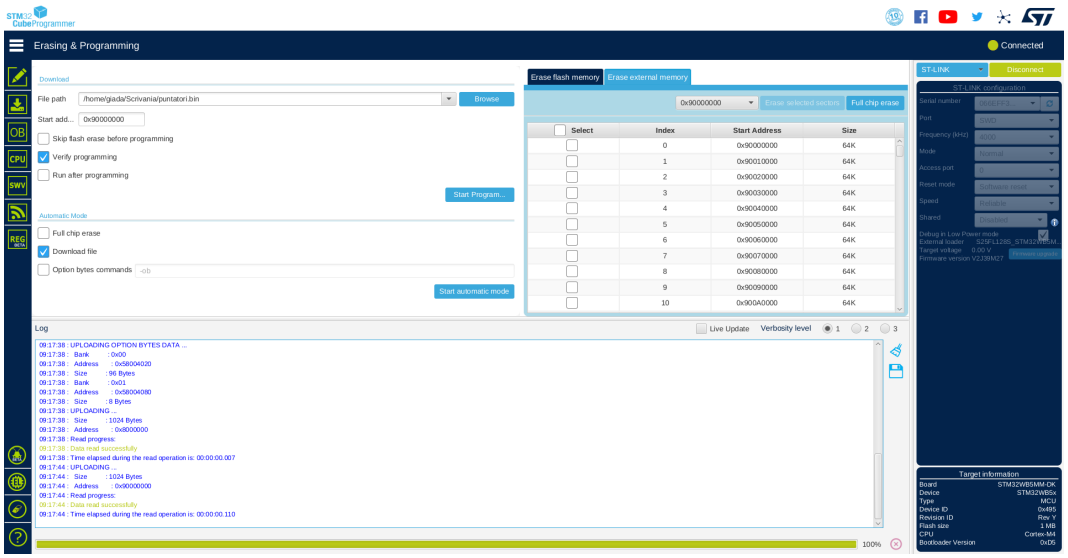

Figura 35: Esempio di caricamento in flash di un file binario tramite STM32CubeProgrammer

Per avere la conferma riguardo il successo dell'operazione effettuata, è possibile anche leggere tale memoria, come illustrato di seguito:

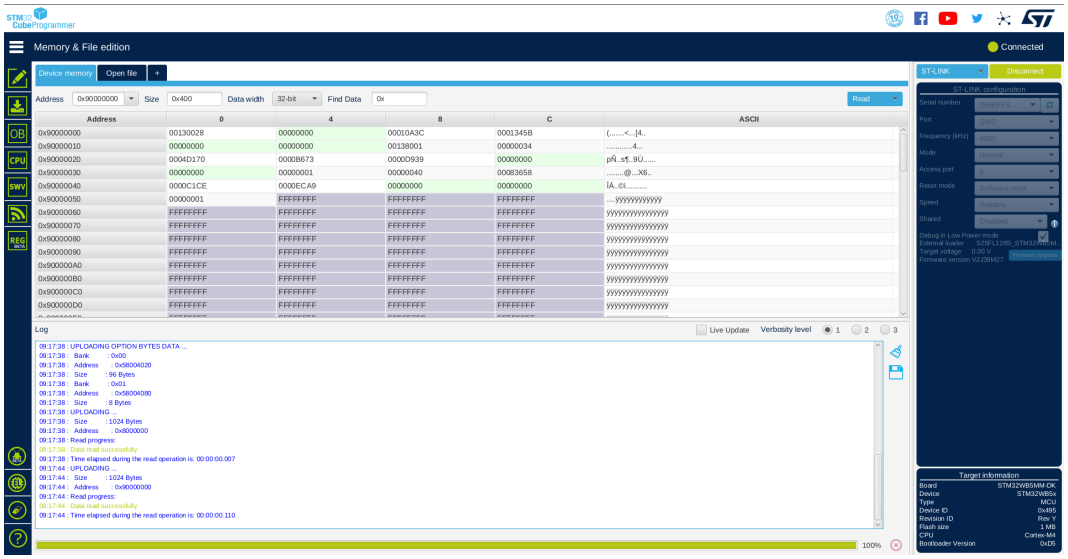

Figura 36: Esempio di lettura della flash tramite STM32CubeProgrammer

5 Validazione

5.1 Test su segnali sinusoidali

Allo scopo di semplificare i test dell'algoritmo descritto in sezione 3.1 e, quindi, rendere più chiari, da eventuali analisi nel tempo e in frequenza, eventuali bug, sono stati usati dapprima dei campioni più semplici di quelli che si sarebbero dovuti inserire nel prodotto finale.

In particolare, grazie ad un breve codice MATLAB, è stato possibile generare 3 segnali sinusoidali, uno per ogni ottava di maggiore interesse, divisi in attacco, sostegno e rilascio.
L'attacco e il rilascio sono stati generati in modo tale da durare un numero intero di periodi del seno in questione, tale da avvicinarsi il più possibile alla durata desiderata, per esempio qui 50 ms.
Inoltre, allo scopo di simulare ciò che succede nella realtà, i campioni dell'attacco sono stati moltiplicati per una rampa crescente da 0 a 1 e passo pari al periodo di campionamento; mentre quelli del rilascio per una rampa decrescente da 1 a 0 e passo pari al periodo di campionamento.
Il sostegno, invece, comprende 2 periodi centrali del seno a volume costante, quindi è tale da raccordarsi perfettamente all'attacco e al rilascio senza discontinuità.
Per esempio, il grafico di attacco, sostegno e rilascio concatenati ottenuti per la sinusoide a 440 Hz (LA4) è riportato in sezione 5.1:

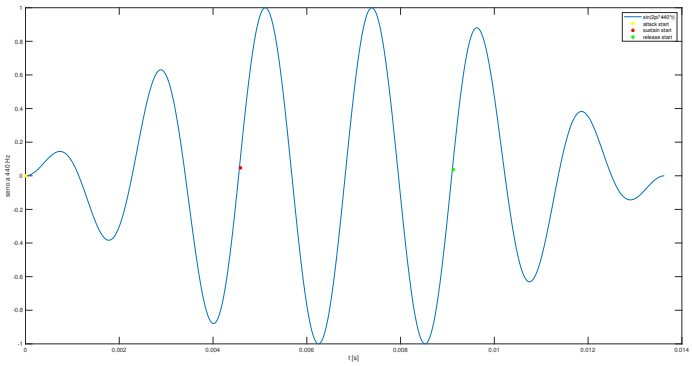

Figura 37: Funzione sinusoidale a f = 440Hz divisa in attacco sostegno e rilascio

5.1.1 Analisi nel tempo tramite oscilloscopio

Con una procedura analoga a quella descritta in sezione 4, i campioni sinusoidali e le strutture ad essi annesse, sono stati caricati in Flash e successivamente sono stati effettuati dei test tramite oscilloscopio.
Innanzitutto, per una nota generica, è stata verificata la correttezza della forma d'onda generata in tutte le fasi di riproduzione.
Nelle immagini seguenti, il segnale giallo mostra l'evoluzione di generiche note nelle varie fasi di riproduzione (attacco, sostegno e rilascio):

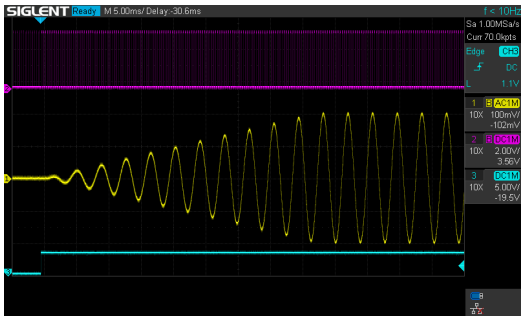

Figura 38: Esempio di attacco e sustain di una generica nota

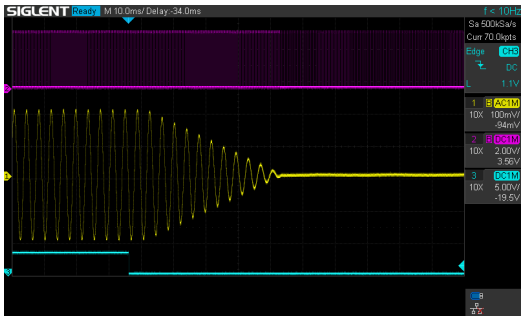

Figura 39: Esempio di sustain e rilascio di una generica nota

In entrambi i casi, si può notare che la forma d'onda risulta perfettamente sinusoidale senza rilevanti imperfezioni.
Il segnale azzurro è un gradino che va alto al "NoteOn" e torna basso al "NoteOff".
Il segnale viola è una successione di impulsi, ciascuno dei quali rappresenta il tempo di calcolo dei campioni relativi ad uno specifico semi-buffer; in particolare dei 16 del canale destro, in quanto quelli del canale sinistro vengono semplicemente messi a 0.

È possibile affermare che, nel caso di gestione di una singola nota, la durata del singolo impulso è di circa di 96 μs, come mostrato di seguito:

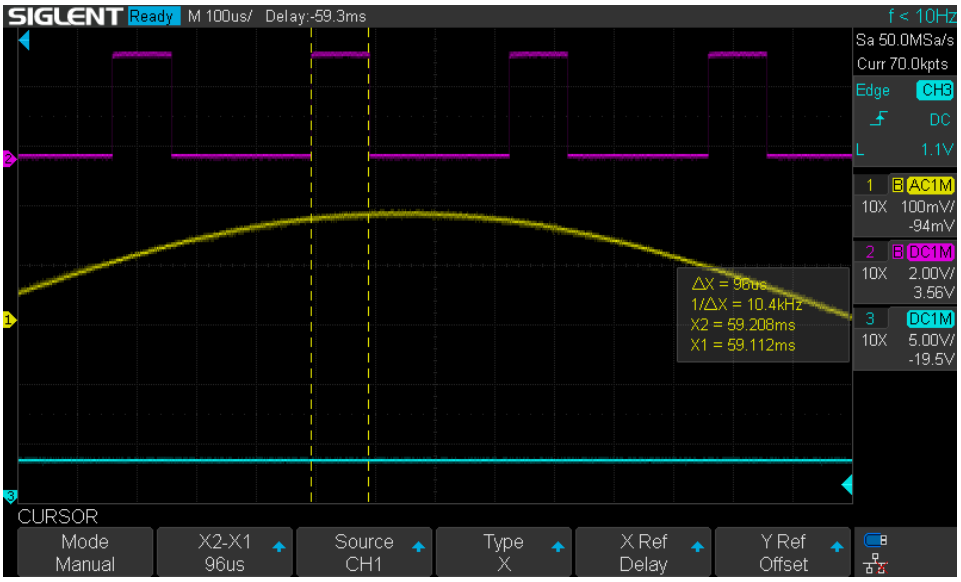

Figura 40: Durata del calcolo dei campioni di metà buffer

Tuttavia, se si considera il caso di gestione di due note, ovvero un "legato", si può notare che il singolo impulso aumenta la sua durata:

Figura 41: Esempio di legato

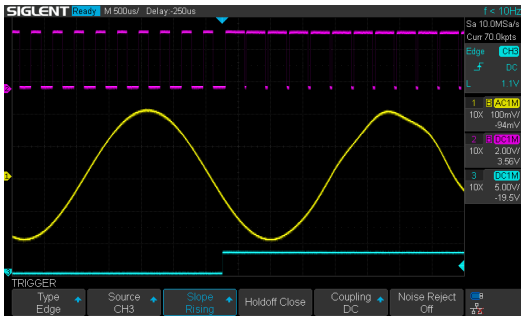

Figura 42: Aumento del tempo di elaborazione nel caso di un "legato"

In particolare, aumentando il grado di risoluzione, si può affermare che tale durata sale a circa 296 μs, come mostrato in figura:

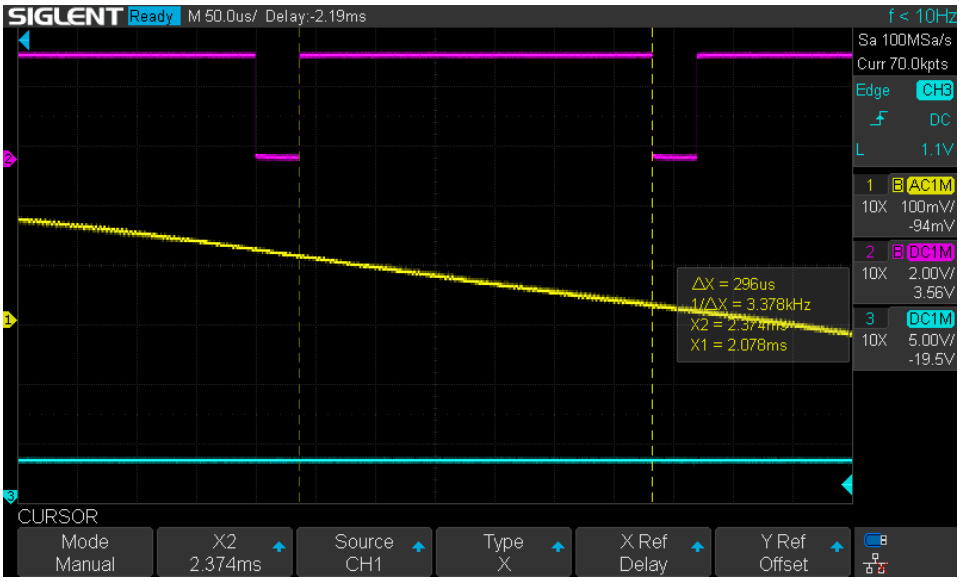

Figura 43: Durata dell'elaborazione nel caso di un "legato"

Se si considera che il tempo disponibile per l'elaborazione di metà buffer è il tempo di riproduzione dell'altra metà buffer, ovvero, considerando una frequenza di campionamento di 48 kHz, circa 333 μs; è possibile affermare che anche con due note, si rientra nelle specifiche.
Dovendo poi tale algoritmo girare per un flauto elettronico, non è necessario sperimentare il funzionamento con più di due note.

5.1.2 Analisi in frequenza in MATLAB

In fig. 44, è riportata l'fft di una generica nota MI4. Da tale immagine, si può notare come effettivamente sia presente l'armonica fondamentale a circa f0 = 330 Hz, più tutta una serie di altre armoniche che costituiscono un rumore di fondo. Alcune di esse, potenzialmente dovute all'azione dell'interpolatore lineare, si ripetono ogni f0/2 $\simeq$ 165 Hz circa. Tuttavia, essendo tali armoniche almeno 50 dB sotto la fondamentale, risultano difficilmente udibili per l'orecchio umano.
Infine, in corrispondenza di 3005 Hz circa si può notare un picco abbastanza rilevante. La sua natura potrebbe essere dovuta alla presenza di un loop di funzioni che si ripete periodicamente ogni 16 campioni, ovvero ogni 1/3005 $\simeq$ 333 μs.

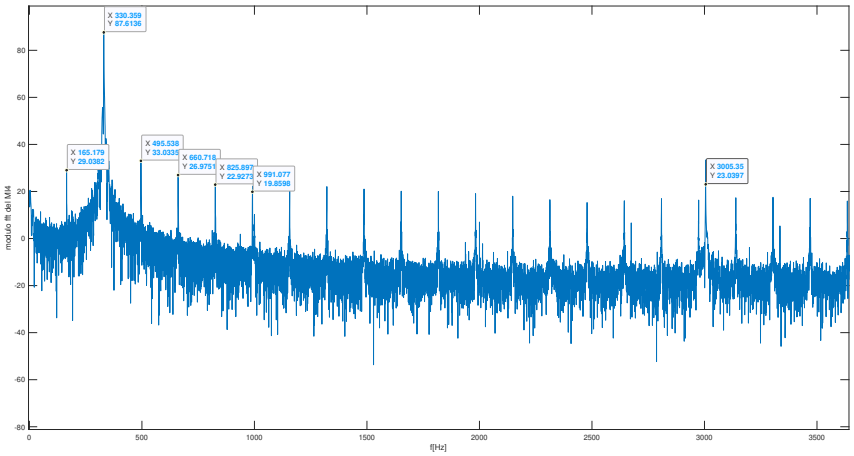

Figura 44: FFT della nota MI4

Abbassando la velocity da 100 a 20, le componenti presenti rimangono pressoché le stesse con guadagni diversi. In particolare, si può notare come l'armonica fondamentale passi da 87 dB a 68 dB:

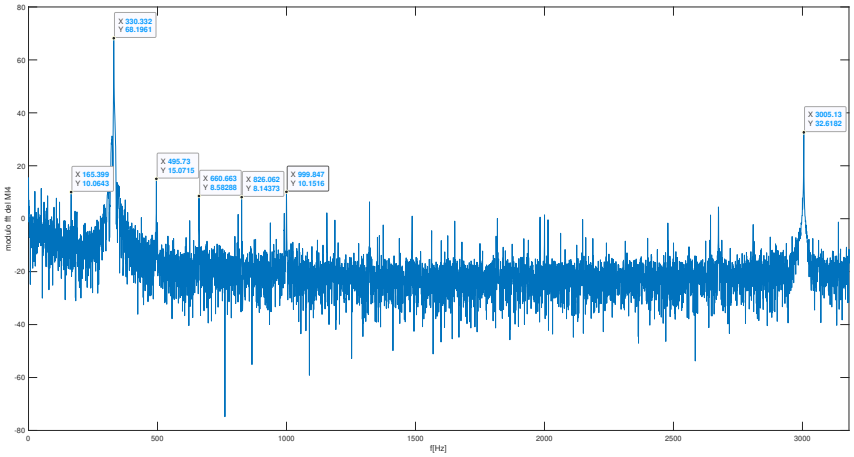

Figura 45: FFT della nota MI4

Nel caso di un legato, la situazione è più o meno simile, tranne per il fatto che essendo presenti due armoniche fondamentali, le componenti dovute all'interpolatore sono molto più fitte:

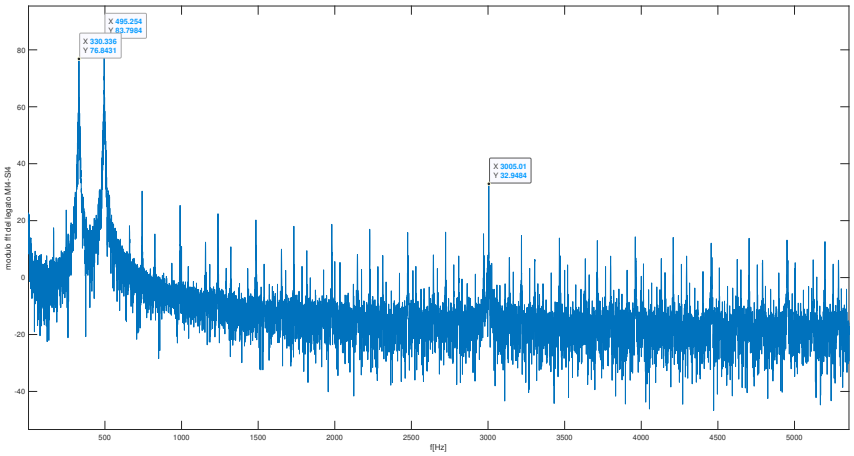

Figura 46: FFT del legato MI4-SI4

5.2 Test su campioni di oboe

5.2.1 Validazione del filtro controllato dall'expression

Rispetto ai segnali sinusoidali, i campioni di oboe presentano uno spettro più ricco di componenti armoniche. Su di essi, dunque, è stato più facile sperimentare l'effetto del filtro con frequenza di cut-off variabile in funzione dell'expression.

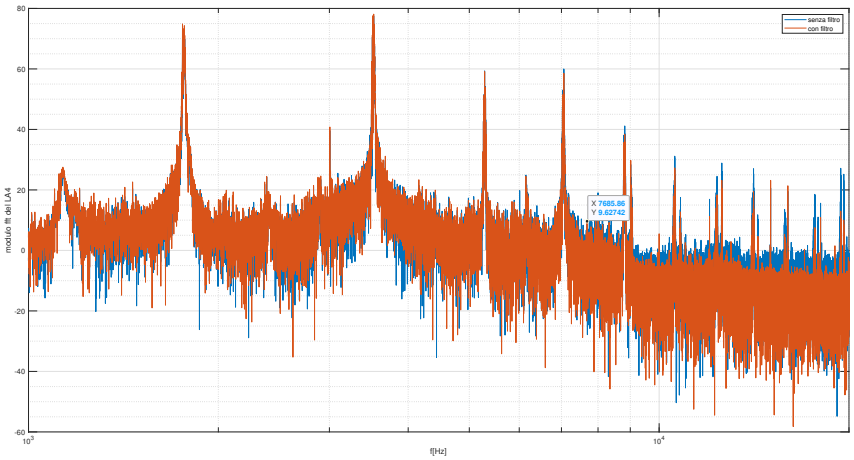

Figura 47: FFT del LA4 a velocity 100 prima e dopo l'applicazione del filtro

Un esempio è mostrato in fig. 47, dove è possibile osservare un LA4 a velocity 100, prima e dopo l'applicazione del filtro. Ciò che si può notare è che, in presenza del filtro, a partire da dalla frequenza 7690 Hz circa, si ha un calo d'ampiezza delle armoniche. In particolare, essendo il filtro di primo ordine, la discesa è di -20 dB a decade.

6 Conclusioni

Il lavoro svolto ha prodotto ottimi risultati in quanto è stato ottenuto un sintetizzatore a campioni in grado di generare correttamente tramite interpolazione, a partire da un solo campione per ottava, tutte le note che rientrano nel range d'interesse.
Il volume della singola nota dipende dal valore di velocity utilizzato, ovvero a ciascun valore da 1 a 127 sarà associato un guadagno che va dai -30 dB agli 0 dB che, tradotto in lineare, vuol dire da 0.03 circa a 1.
I suoni sono dotati di un attacco iniziale e, a seguire, di un loop di sustain che si ripete periodicamente fino al rilascio, quando poi verrà sfumato fino ad estinguersi.
Nonostante la monofonia del sistema, se una nota viene suonata mentre un'altra è in esecuzione, il suo attacco non verrà riprodotto e i due suoni verranno sottoposti a cross-fade di durata 50 ms. Dopo di ciò, la nota in uscita sarà del tutto estinta, mentre quella in entrata avrà raggiunto il guadagno di regime, dipendente dal valore di velocity ad essa collegato.
A seconda del valore di expression settato, il quale è variabile tra 1 e 127, il contenuto armonico del suono viene modificato attraverso l'impiego di un filtro passabasso del primo ordine con frequenza di cut-off variabile tra i 2000 e i 10000 Hz. Ciò contribuirà a rendere il suono più scuro e cupo per expression più basse e, al contrario, più chiaro e brillante per expression più alte.
Eventuali punti di miglioramento potrebbero riguardare la velocità di lettura e di elaborazione dei campioni, la quale rappresenta attualmente un collo di bottiglia per il sistema. Infatti, i campioni, per via della loro lunghezza, sono salvati in una memoria NOR-Flash esterna anziché in RAM, il che contribuisce a rendere la loro acquisizione più lenta, non consentendo la gestione di più di due suoni per volta.
La parte di elaborazioni di campioni, invece, viene fatta sfruttando un normale processore ARM Cortex-M4 con FPU, anziché un vero processore DSP. Ciò impedisce di parallelizzare i calcoli sui campioni float contenuti nei buffer.
Un altro limite attuale dell'algoritmo è che, se si prolunga sufficientemente il tempo di riproduzione di una nota, soprattutto sulle note alte, è possibile udire ogni qualche centinaio di campioni un lieve click.
La causa di quest'imperfezione risiede nel fatto che, una volta configurata la fre-

quenza di clock per il micro pari a 64 MHz, sfruttando HSE ("high-speed external oscillator"), molto più stabile dell'HSI ("high-speed internal oscillator"), per via delle limitate potenzialità del PLL, non è stato possibile ottenere per la periferica SAI, una frequenza di lavoro di 48000 Hz esatti. In altre parole, usando le moltiplicazioni e divisioni per gli interi a disposizione, è stato possibile soltanto avvicinarsi a tale frequenza, ottenendo circa 48076 Hz.
Per questa ragione, anche il PLL del DAC è stato configurato, in modo da consentirgli di lavorare ad una frequenza circa uguale a quella del micro, ma comunque differente per alcune cifre decimali.
Tale problematica potrebbe essere eventualmente risolta usando un DAC che utilizzi direttamente il clock del micro, anziché generarne uno internamente o, viceversa, dando la possibilità al micro di utilizzare il clock del DAC.

Riferimenti bibliografici

[1] *ARTInoise*, http://www.artinoise.com.

[2] Sam McGuire, Nathan Van Der Rest, *The musical art of synthesis*. 2016.

[3] Martin Russ, *Sound Synthesis and Sampling*. 2008.

[4] *Munchkinization*, https://electronicmusic.fandom.com/wiki/Munchkinization.

[5] *Scaling theorem*, https://ccrma.stanford.edu/~jos/st/Scaling_Theorem.html.

[6] Steven W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*. 1998.

[7] *Music Technology Research: The History of the Sampler*, https://flismusicblog.wordpress.com/2015/05/13/music-technology-research-the-history-of-the-sampler/.

[8] *Campionatore (strumento musicale)*, https://www.no-regime.com/ru-it/wiki/Computer_Music_Melodian.

[9] Mark Vail, *Vintage Synthesizers*. 2000.

[10] ——, *The Synthesizer: a comprehensive guide*. 2014.

[11] *A brief history of sampling*, https://www.thomann.de/blog/en/a-brief-history-of-sampling/.

[12] *Vintage rewind: Fairlight CMI*, https://musictech.com/features/opinion-analysis/vintage-rewind-fairlight-cmi/.

[13] *Synclavier: musica (digitale) maestro!* https://www.appuntidigitali.it/3970/synclavier-musica-digitale-maestro/.

[14] *E-mu Emulator)*, https://it.wikipedia.org/wiki/E-mu_Emulator.

[15] *E-mu SP1200)*, https://www.vintagesynth.com/emu/sp1200.php.

[16] *Sampler)*, https://en.wikipedia.org/wiki/Sampler_(musical_instrument).

[17] *Interpolare: una guida per principianti!* https://www.mathisintheair.org/wp/2021/02/interpolare-una-guida-per-principianti/.

[18] Vesa Valimaki, "Fractional Delay Filter Design Based on Truncated Lagrange Interpolation," 2007.

[19] *Sinc Interpolation for Signal Reconstruction*, https://demonstrations.wolfram.com/SincInterpolationForSignalReconstruction/.

[20] *Comparison of Sinc Interpolation with Linear Interpolation*, https://www.researchgate.net/figure/Comparison-of-Sinc-Interpolation-with-Linear-Interpolation_fig1_297707712.

[21] "MIDI 1.0 Detailed Specification," 1985.

[22] *MIDI*, https://web.archive.org/web/20120830211425/http://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol1/aps2/.

[23] *MIDIschema*, https://commons.wikimedia.org/wiki/File:MIDI_IN_OUT_schematic.svg.

[24] *MIDIhw*, https://learn.sparkfun.com/tutorials/midi-tutorial/hardware--electronic-implementation.

[25] *An example of MIDI message*, https://www.researchgate.net/figure/An-example-of-MIDI-message_fig1_343709022.

[26] *Discovery kit with STM32WB5MMG MCU*, https://www.st.com/en/evaluation-tools/stm32wb5mm-dk.html.

[27] *WM8960 Audio HAT*, https://www.waveshare.com/wiki/WM8960_Audio_HAT.

[28] *ARTInoise Re.corder Blue*, https://www.thomann.de/it/artinoise_re.corder_blue.htm.

[29] *Labium(musica)*, https://it.wikipedia.org/wiki/Labium_(musica).

[30] Oppenheim, Alan, *Discrete Time Signal Processing*. 2010.