Realizzazione degli Oscillatori del sintetizzatore EMS VCS 3 nel linguaggio di programmazione FAUST

Luca Spanedda

24/10/2020

Sommario

L'EMS VCS 3 è un sintetizzatore analogico portatile con una flessibile architettura semimodulare controllata da una matrice. Viene introdotto nel mercato nel 1969 dagli Electronic Music Studios di Londra. In questa breve ricerca, andremo ad analizzare e realizzare, documentando all'interno del codice i vari passaggi, dei metodi per replicare la sua sezione Oscillatori in digitale tramite il linguaggio di programmazione FAUST (Grame). Con il fine ultimo di creare un codice aperto che possa essere un punto d'inizio per chiunque voglia studiare il Linguaggio di Programmazione FAUST (Grame) o realizzare in questo oscillatori virtual-analog con questa metodologia.

1 Introduzione

Con la classe del dipartimento del I Biennio di Musica Elettronica del conservatorio Santa Cecilia di Roma (anno accademico 2019/2020), abbiamo lavorato allo studio e la realizzazione di vari metodi efficaci per riprodurre in digitale l'architettura di un sintetizzatore EMS VCS3, per lo studio e l'esecuzione di un brano di Giorgio Nottoli: Incontro, che ne fa uso. Il lavoro (work in progress) è stato poi documentato e publicato su Github come parte integrante del progetto SEAM (Sustained Electroacoustic Music).
Il lavoro di riproduzione è stato fatto tramite FAUST (Grame) (Functional Audio Stream), un linguaggio di programmazione specifico per il Digital Signal Processing sviluppato da Yann Orlarey, Dominique Fober, e Stéphane Letz nel 2002. Nello specifico, FAUST è un linguaggio di programmazione ad alto livello scritto in C++, che permette di tradurre delle istruzioni date e create appositamente per il digital signal processing (DSP), in un largo raggio di linguaggi di programmazioni non specifici per il dominio dell'Audio Digitale.
In questa ricerca, il mio impegno è stato quello di approfondire e documentare, singolarmente e a posteriori del lavoro già svolto con la classe, la parte di architettura dell'EMS VCS 3 riguardante gli oscillatori, dove nel particolare, per sezione degli oscillatori dell'EMS VCS 3 intendo: Oscillatore 1, Oscillatore 2, e Oscillatore 3. E di cercare dunque dei metodi funzionali e aperti, per replicare gli stessi nel dominio digitale.
Non entrerò dunque nel merito di problematiche legate al filtraggio degli stessi,

e quantomeno nella realizzazione di altre sezioni del sintetizzatore stesso che in questa istanza non ci competono, procederò invece concentrando la mia attenzione sulla realizzazione degli oscillatori controllati in voltaggio (virtual analog), a partire da una matematica che ci permette di generarli, e dunque dalla creazione di un Phasor in FAUST. Procediamo dunque preleminarmente con una breve analisi degli oscillatori dell'EMS VCS3 e dei loro controlli.

2 Oscillatori dell'EMS VCS 3

2.1 La sezione Oscillatore 1

La sezione Oscillatore 1, è un pannello che contiene 4 potenziometri, che corrispondono ognuno ad un controllo specifico di un parametro.
Il potenziometro 1:
permette un controllo della frequenza dell'oscillatore, potendo passare tramite il potenziometro per tutti i valori intermedi che vanno da 1Hz a 10000Hz, tale capacità è estendibile tramite Control Voltage.
Il potenziometro 2:
lo Shape, permette un controllo sulla forma d'onda dell'oscillatore. La scala del potenziometro va da da 0 a 10, a metà della sua capacità rotativa e dunque al valore 5, avremmo come output la forma d'onda di una funzione sinusoidale, mentre quando è a 0 o a 10, avremo della stessa come output il valore assoluto della forma d'onda ed il suo inverso in fase.
In tutto questo abbiamo la capacità di passare per tutti i gradi intermedi generando varie ibridazioni tra i tre stati della forma d'onda.
Il potenziometro 3:
permette un controllo dell'ampiezza di uscita della sinusoide dalla sezione oscillatore 1, la scala del potenziometro anche in questo caso va da 0 a 10.
Il potenziometro 4:
permette un controllo dell'ampiezza di uscita del fasore della sinusoide, come per il potenziometro 2 e 3, la scala del potenziometro 4 va da 0 a 10.
Le due forme d'onda in uscita, sinusoide ed il suo fasore, sono perfettamente correlate in fase, permettendo secondo necessità la creazione di una nuova forma d'onda a partire da una miscelazione delle due.

2.2 La sezione Oscillatore 2

La sezione Oscillatore 2, è uguale nei potenziometri e nelle caratteristiche di controllo alla sezione oscillatore 1, ma con due differenze sostanziali: la prima differenza è che le due forme d'onda in uscita sono una onda quadra e una triangolare, e non una sinusoide con controllo di shape ed il fasore (statico nella sua forma d'onda).
La seconda differenza è relativa al controllo tramite il potenziometro shape, sulla forma d'onda dell'oscillatore 2, che a differenza della sezione oscillatore 1 modifica le fasi di entrambe le forme d'onda in uscita.
L'onda quadra viene modificata in maniera proporzionale alla lunghezza di una delle due fasi rispetto ad un relativo accorciarsi dell'altra (a valore 5 del poten-

ziometro, le fasi della nostra onda quadra sono equivalenti).
Mentre l'onda triangolare modifica le proprie fasi in modo da divenire: una dente di sega a 0, una triangolare a 5, e un fasore a 10, ovviamente passando per tutte le ibridazioni dettate dai valori intermedi. Anche in questo caso le fasi sono identiche per entrambe le uscite in ampiezza essendo le forme d'onda ricavate dallo stesso Fasore.

2.3 La sezione Oscillatore 3

La sezione Oscillatore 3, è uguale alla sezione oscillatore 2, in questo caso l'unica variazione è relativa range di controllo di frequenze possibili dall'oscillatore tramite il primo potenziometro: che vanno da un minimo di 0.25Hz, ed arrivano sino ad una frequenza massima di 500Hz, permettendone dunque l'applicazione come segnale di controllo (LFO), grazie ad una precisione nel controllo a basse frequenze.

3 Realizzazione degli Oscillatori dell'EMS VCS 3 in digitale tramite FAUST Grame

3.1 Breve introduzione sul funzionamento degli Oscillatori Digitali

Gli oscillatori digitali virtual analog genericamente condividono una comune topologia, che consiste in un accumulatore di fase dove ogni campione si somma al precedente generando una semplice forma d'onda che chiamiamo Phasor, e dove secondo una soglia predeterminata, i campioni che son stati accumulati vengono poi riscalati, scaricando questa accumulazione di fase al valore di partenza 0, e permettendo dunque così un ricominciare del processo ciclico. Il processo designato è alla base di due principali modi differenti genericamente utilizzati per la generazione di forme d'onda nel dominio digitale:
Un metodo prevede una lettura ciclica di una tabella di valori (campioni), salvati precedentemente in memoria, e che nella loro interezza formano il periodo di un segnale. Questi campioni vengono letti da un segnale di controllo, che è appunto il nostro Phasor e che ha il compito di puntare alle locazioni di memoria e ricostruire dalla nostra tabella salvata in memoria una determinata forma d'onda. Il processo avviene tante volte al secondo quanti sono gli Hz desiderati a cui il fasore sta leggendo. Questi oscillatori sono chiamati Table-lookup Oscillators (differenti dalla sintesi per Wavetable essendo la lettura di tavole statiche)
Un secondo metodo, prevede invece una elaborazione matematica del Phasor, tramite delle formule matematiche che ci permettono di ricavare diversi oscillatori partendo da quest'ultimo. Questo metodo viene chiamato Virtual Analog. Il vantaggio di questo secondo metodo sta nel fatto che a differenza del primo, non abbiamo una tabella fissa da leggere salvata precedentemente in memoria e composta da un totale discreto di campioni, che definisce il nostro periodo del segnale. Ma invece, di volta in volta, possiamo cambiare il numero di campioni che vanno a comporre il nostro Phasor, ottenendo di conseguenza segnali molto definiti nella loro interezza dal numero di campioni che si decide di utilizzare a

priori e che sfruttano qualitativamente al massimo il sistema.
Il metodo che si è deciso di utilizzare per questo lavoro di realizzazione in FAUST, è nello specifico il secondo metodo esposto. Procedendo in questo modo, è stato fatto si nel seguente codice che al variare della frequenza di campionamento, si utilizzi una definizione differente di campioni che andiamo ad utilizzare e spartire nel processo, per ricostruire nella sua interezza la definizione del nostro segnale. Questo totale di campioni da spartire nel nostro periodo del segnale, e che determina il nostro Phasor, è definito in partenza dalla nostra frequenza di campionamento. A seguito il Codice scritto con commenti interni ed esterni, per le varie sezioni di codice, che ne illustrano e ne documentano la sua architettura e realizzazione.

3.2 Codice in FAUST (GRAME)

La prima parte del codice è dedicata a richiamare la libreria standard di faust e a definire manualmente una costante o più, qualora ce ne fosse il bisogno.
La libreria standard di FAUST si occupa di richiamare tutte le librerie che sono state già scritte, e che permettono dunque l'accesso ad una serie di utilità precedentemente scritte all'interno di queste. In questo caso, nonostante la libreria standard di FAUST permetta di richiamare una serie di costanti al suo interno, incluse quelle interessate da noi, ho deciso di riportare il valore della costante del Pi Greco in Double Precision manualmente, in modo da poterla estendere e modificare agevolmente in caso fosse necessario.

```
// INIZIO DEL CODICE

// REPLICA DEGLI OSCILLATORI DELL'EMS VCS3


        /* ------------------------------------
            LIBRERIA STANDARD & COSTANTI
        ------------------------------------ */
        // LIBRERIA standard di FAUST
        import("stdfaust.lib");

        // COSTANTI:
        // 2 PI Greco (In Double Precision)
            pigreco = 3.14159265358979323846264338327950288419716
            939937510582097494459230781640628620899;
            due_pigreco = pigreco*2;
        /* ------------------------------------ */
```

Il process in FAUST ci permette di di assegnare le uscite di riferimento del codice. In questo caso l'uscita scritta nel codice ci permette di avere la sezione oscillatore 1,2,3 in uscita sul primo canale, e una seconda copia della sezione oscillatore 1,2,3 su un secondo canale (funzione ottenuta tramite una somma

delle tre sezioni per entrambi i canali e impostando tutte le ampiezze a 0). La funzione richiamata ci permette di puntare ad alla sezione Oscillatore interessata: oscillatore1, oscillatore2, o oscillatore3, e invece i controlli scritti come argomento della funzione (1,2,3,4) corrispondono rispettivamente a: frequenza, shape, ampiezza 1, ampiezza 2, della sezione oscillatore richiamata.

```
// USCITE:
// il process ci permette di uscire dai canali
// a partire dal canale 1 di default a seguire.
// è necessario usare una , tra un canale e il seguente.


// LEGENDA CONTROLLI OSCILLATORI SU USCITA:
// oscillatore1/2/3(frequenza, shape,
                    amp forma d'onda 1, amp forma d'onda 2)
process =
            // Uscite dal canale 1
            oscillatore1(440, 5.01, 0., 0.)+
            oscillatore2(440, 5.01, 0., 0.)+
            oscillatore3(440, 5.01, 0., 0.),

            // Uscite dal canale 2
            oscillatore1(440, 5.01, 0., 0.)+
            oscillatore2(440, 5.01, 0., 0.)+
            oscillatore3(440, 5.01, 0., 0.);
 /* ------------------------------------ */
```

Procediamo ora analizzando l'architettura che è stata utilizzata per la creazione della sezione Oscillatore 1. Il nome della funzione ed i suoi argomenti scritti all'interno della funzione stessa sono: "oscillatore1(frequency1, scaledshape1, scaledampsine1, scaledampsaw1)", dove le funzioni dei vari argomenti sono rispettivamente controllo di: frequenza, shape, ampiezza forma d'onda 1, ampiezza forma d'onda 2, e vengono illustrate nei commenti all'interno del codice. "= funzioneuscita1", Sta per la funzione che viene richiamata all'interno del codice e che viene rappresentata da "oscillatore1" ed i suoi argomenti. Il "with" seguito dalla parentesi graffa aperta, è dato come punto iniziale del codice che a sua volta comprende tutte le variabili che compongono la funzione "oscillatore1", ed all'interno delle variabili, vengono passati i vari argomenti della funzione per renderne efficace l'utilizzo desiderato. Il termine del codice della funzione è designato da una parentesi graffa chiusa seguita da un ";"" che sta ad indicare il termine del codice della funzione. Una prima parte del codice è pensata in modo da tradurre i controlli del valore, degli argomenti dati alla funzione (che sono i stessi valori numerici dei 4 potenziometri della sezione oscillatore del VCS3) in valori utilizzati all'interno del codice definiti a loro volta da alcune variabili. A seguito, viene poi convertito il valore dell'argomento di funzione della shape, in una serie di valori che permettono il controllo della forma d'onda in uscita dell'Oscillatore, dividendo il numero che va da 0 a 10 in tre controlli di ampiezza

con un minimo di 0 e un massimo di 1 indipendenti: uno ascendente da 5 a 10, uno discendente da 0 a 5, e uno triangolare (ascendente da 0 a 5 e discendente da 5 a 0).
Si passa poi nel vivo della sezione oscillatore 1, ovvero alla parte di codice relativa alla generazione della funzione stessa: nella prima parte troviamo in origine un Fasore generato da un accumulatore di fase che viene riscalato periodicamente e controllato in frequenza, basandoci sulla funzione ma.SR della libreria di FAUST che ci passa come costante il valore attuale della frequenza di campionamento utilizzata, e a seguito troviamo le forme d'onda utilizzate ricavate dal fasore stesso. Infine nell'ultima parte del codice della funzione, troviamo la somma delle onde ricavate dal fasore e l'uscita dalla funzione "funzioneuscita1" che verrà poi richiamata a significare la funzione stessa "oscillatore1".

```
/* ------------------------------------
   SEZIONE OSCILLATORE 1
   ------------------------------------ */

// Codice della Funzione dell'Oscillatore 1
// richiamato sul process.

// CONTROLLI che vengono richiamati come argomento della funzione:
// frequency1 = frequenza da 1. a 10000.
// shape1 = forma d'onda da 0. a 1. per la sine
// (nel VCS3 da 0. a 10.)
// amposcillatore1 = ampiezza della sine da 0. a 1.
// (nel VCS3 da 0. a 10.)
// ampsaw1 = ampiezza della saw da 0. a 1. (nel VCS3 da 0. a 10.)


// FUNZIONE:
oscillatore1(frequency1, scaledshape1,
scaledampsine1, scaledampsaw1)
= funzioneuscita1
with{

/* ------------------------------------ */
// RISCALAMENTO DEI CONTROLLI dati in argomento alla funzione:
shape1 = scaledshape1/10;
amposcillatore1 = scaledampsine1/10;
ampsaw1 = scaledampsaw1/10;

    /* ------------------------------------
       CONTROLLO DELLA SHAPE
       ------------------------------------ */

    // CALCOLI SHAPE
    // il valore della shape dato è da 0. a 1.
```

```
    // calcoli a seguire su questo:

    // rampa negativa 1-A-0
    shape1neg = (shape1*-1)+1;
    // condizione discendente se > 0.5
    shape1maj05 = (shape1 > 0.5)*shape1neg;
    // condizione ascendente se < 0.5
    shape1neg05 = (shape1 < 0.5)*shape1;
    // rampa negativa 1-A-(-1)
    shape1negpos = (shape1*2-1);


        // OUTS DELLA SHAPE
        // rampa triangolare finale 0-A-1-A-0
        shape1final = (shape1maj05 + shape1neg05)*2;
        // rampa tri negativa
        // 2 - seconda metà
        shape1cos = (shape1negpos > 0)*shape1negpos;
        // rampa tri negativa
        // 1 - prima metà
        shape1sin = ((shape1negpos < 0)*shape1negpos)*-1;

/* ------------------------------------ */

    // OSCILLATORE

    /* ------------------------------------
       FASORE
       ------------------------------------ */

    // FASORE
    // da cui derivano matematicamente le forme d'onda:

    decimale(step)= step-int(step);
    // reset interi
    fasoreosc1 = (frequency1/ma.SR) : (+ : decimale) ~ _;
    // controllo frequenza,
    // e loop ad ogni reset


    // forme d'onda ricavate a partire dal fasore:

    /* ------------------------------------
       SINUSOIDE COMPLETA
       ------------------------------------ */
    sine2pi = fasoreosc1 * due_pigreco;
    // fasore moltiplico *2 PI
    sinusoideout = sin(sine2pi)*shape1final;
    // Uscita Sinusoide con controllo Shape
```

```
    /* ------------------------------------
       SENO
       ------------------------------------ */
    sinepi = fasoreosc1 * pigreco;
    // fasore moltiplico *2 PI
    senofinal = ((sin(sinepi)-0.5)*2);
    senoout = senofinal*shape1sin;
    // Uscita Seno

    /* ------------------------------------
       COSENO
       ------------------------------------ */
    coseno = (senofinal*-1);
    cosenoout = coseno*shape1cos;
    // Uscita Coseno


    // OUT FUNZIONE
    funzioneosc1 = (sinusoideout+senoout+cosenoout)
    * amposcillatore1;
    // uscita oscillatore con controllo della shape e ampiezza
    funzionefasore1 = (fasoreosc1*2-1.)*ampsaw1;
    // uscita rampa ricavata dal fasore
    // con solo controllo ampiezza


    funzioneuscita1 = funzioneosc1+funzionefasore1;
    // uscita dalla funzione
};
```

La sezione oscillatore 2, e la sua relativa funzione "oscillatore2", nella sua architettura non è sostanzialmente differende dalla sezione oscillatore 1 che è stata documentata. Troviamo anche qui una architettura composta rispettivamente da: funzione e argomenti di funzione (che corrispondono nella loro funzione a quelli dell'oscillatore 1), riscalamento dei controlli di argomento della funzione, calcolo della shape, creazione del fasore, ed infine forme d'onda ricavate a partire dal fasore stesso seguite dalla relativa uscita dalla funzione. La sostanziale differenza avviene nelle forme d'onda create dal Phasor in uscita dalla funzione, che non sono più la sinusoide ed il suo valore assoluto positivo e negativo ed il Phasor stesso, ma divengono onda quadra e Phasor, con il controllo della shape comune per entrambe le due forme d'onda, che nel caso dell'onda quadra permette un bilanciamento della fase negativa e positiva relativo al controllo, e nel caso del fasore il controllo della shape permette tre stati ricavati dal Phasor stesso che divengono rispettivamente: dente di sega, triangolare, e Fasore stesso.

```
/* ------------------------------------
   SEZIONE OSCILLATORE 2
```

```
   ------------------------------------ */

// Codice della Funzione dell'Oscillatore 2
// richiamato sul process.

// CONTROLLI che vengono richiamati come argomento della funzione:
// frequency2 = frequenza da 1. a 10000.
// shape2 = forma d'onda da 0. a 1. per quadra/saw
// (nel VCS3 da 0. a 10.)
// amposcillatore2 = ampiezza della quadra da 0. a 1.
// (nel VCS3 da 0. a 10.)
// ampsaw2 = ampiezza della saw da 0. a 1. (nel VCS3 da 0. a 10.)


// FUNZIONE:
oscillatore2(frequency2, scaledshape2,
scaledampsquare2, scaledampsaw2) =
funzioneuscita2
with{

/* ------------------------------------ */
// RISCALAMENTO DEI CONTROLLI dati in argomento alla funzione:
shape2 = scaledshape2/10;
ampquadra2 = scaledampsquare2/10;
amptri2 = scaledampsaw2/10;

    /* ------------------------------------
       CONTROLLO DELLA SHAPE
       ------------------------------------ */

    // CALCOLI SHAPE
    // il valore della shape dato è da 0. a 1.
    // calcoli a seguire su questo:

    // rampa negativa 1-A-0
    shape2neg = (shape2*-1)+1;
    // condizione discendente se > 0.5
    shape2maj05 = (shape2 > 0.5)*shape2neg;
    // condizione ascendente se < 0.5
    shape2neg05 = (shape2 < 0.5)*shape2;
    // rampa negativa 1-A-(-1)
    shape2negpos = (shape2 *2 -1);


        // OUTS DELLA SHAPE
        // rampa triangolare finale 0-A-1-A-0
        shape2final = (shape2maj05 + shape2neg05)*2;
        // rampa tri negativa
        // 2 - seconda metà
        shape2phasor = (shape2negpos > 0)*shape2negpos;
```

```
        // rampa tri negativa
        // 1 - prima metà
        shape2saw = ((shape2negpos < 0)*shape2negpos)*-1;

/* ------------------------------------ */

    // OSCILLATORE

    /* ------------------------------------
       FASORE
       ------------------------------------ */

    // FASORE
    // da cui derivano matematicamente le forme d'onda:

    decimale2(step2)= step2-int(step2);
    // reset interi
    fasoreosc2 = (frequency2/ma.SR) : (+ : decimale2) ~ _;
    // controllo frequenza,
    // e loop ad ogni reset


    // forme d'onda ricavate a partire dal fasore:

    /* ------------------------------------
       QUADRA
       ------------------------------------ */
    quadra1out = (fasoreosc2 > shape2*-1 +1) *2 -1;
    // Imposto condizione su fasore (vero o falso; 1 o 0)
    // la shape modifica la soglia della condizione,
    // e trasformo il segnale rendendolo bipolare

    /* ------------------------------------
       DENTE DI SEGA A RAMPA
       ------------------------------------ */
    sawrampa1out = (fasoreosc2 *2 -1) * shape2phasor;
    // trasformo il fasore in bipolare
    // e imposto un controllo della shape ascendente da 0.5 a 1.

    /* ------------------------------------
       DENTE DI SEGA
       ------------------------------------ */
    sawimpulse1out = (fasoreosc2 *-1 +0.5) * 2 * shape2saw;
    // trasformo il fasore in bipolare ma ribaltando le fasi
    // e imposto un controllo della shape discendente da 0. a 0.5

    /* ------------------------------------
       TRIANGOLARE
       ------------------------------------ */
    negative1tophasor = fasoreosc2 *-1 +1;
```

```
    // ribalto fase del fasore - da 1. a 0.
    tri1maj05 = ((fasoreosc2 > 0.5) * negative1tophasor);
    // passa solo il segnale > di 0.5 uscendo come 1.
    // ma moltiplico l'1. per la seconda parte del
    // fasore ribaltato
    // ottenendo un movimento da 0.5 a 0.
    tri1min05 = (fasoreosc2 < 0.5) * fasoreosc2;
    // passa solo il segnale < di 0.5 uscendo come 1.
    // ma moltiplico l'1. per la prima parte del
    // fasore (normale)
    tri1sum = tri1maj05 + tri1min05;
    // costruisco la mia onda triangolare con le somme
    tri1out = (tri1sum - 0.25) * 4 * shape2final;
    // riscalo rendendo l'onda triangolare bipolare
    // imposto il controllo della shape triangolare:
    // da 0. a 0.5 ascendente e da 0.5 a 1. discendente


    // OUT FUNZIONE
    oscquadra1out = quadra1out * ampquadra2;
    // uscita oscillatore con controllo della shape e ampiezza
    triangolare1out = (tri1out + sawimpulse1out + sawrampa1out)
    * amptri2;
    // uscita oscillatore con controllo della shape e ampiezza


    funzioneuscita2 = oscquadra1out+triangolare1out;
    // uscita dalla funzione
};
```

Infine troviamo la sezione oscillatore 3 e la sua corrispondente funzione "oscillatore3", che è identica alla sezione oscillatore 2, ma con una differenza teorica: in digitale e in questo caso, permettendoci il calcolatore di determinare valori molto specifici nel dominio della virgola mobile, non abbiamo lo stesso problema di valori da passare alla frequenza legato invece all'uso di un potenziometro nel dominio analogico, che invece per permetterci una maggiore precisione nel controllo dell'oscillatore ci impone un riscalamento del raggio di azione della resistenza variabile dedicato al controllo. Qualora si decidesse di utilizzare una interfaccia utente grafica, o un interfaccia esterna o interna di qualsiasi tipo, è necessario tener conto che è adeguato mappare i valori in modo che corrispondano a quelli del potenziometro analogico dedicato alla frequenza, così da far fede ai controlli originali del VCS3. In questa sede, possiamo permetterci invece il lusso di interagire passando alla funzione qualsiasi valore di frequenza necessiti alla nostra occorenza.

```
/* ------------------------------------
   SEZIONE OSCILLATORE 3
```

```
   ------------------------------------ */

// Codice della Funzione dell'Oscillatore 3
// richiamato sul process.

// CONTROLLI che vengono richiamati come argomento della funzione:
// frequency3 = frequenza da 0.25 a 500.
// shape3 = forma d'onda da 0. a 1. per quadra/saw
// (nel VCS3 da 0. a 10.)
// amposcillatore3 = ampiezza della quadra da 0. a 1.
// (nel VCS3 da 0. a 10.)
// ampsaw3 = ampiezza della saw da 0. a 1. (nel VCS3 da 0. a 10.)


// FUNZIONE:
oscillatore3(frequency3, scaledshape3,
scaledampsquare3, scaledampsaw3) =
funzioneuscita3
with{

/* ------------------------------------ */
// RISCALAMENTO DEI CONTROLLI dati in argomento alla funzione:
shape3 = scaledshape3/10;
ampquadra3 = scaledampsquare3/10;
amptri3 = scaledampsaw3/10;

    /* ------------------------------------
       CONTROLLO DELLA SHAPE
       ------------------------------------ */

    // CALCOLI SHAPE
    // il valore della shape dato è da 0. a 1.
    // calcoli a seguire su questo:

    // rampa negativa 1-A-0
    shape3neg = (shape3*-1)+1;
    // condizione discendente se > 0.5
    shape3maj05 = (shape3 > 0.5)*shape3neg;
    // condizione ascendente se < 0.5
    shape3neg05 = (shape3 < 0.5)*shape3;
    // rampa negativa 1-A-(-1)
    shape3negpos = (shape3 *2 -1);


        // OUTS DELLA SHAPE
        // rampa triangolare finale 0-A-1-A-0
        shape3final = (shape3maj05 + shape3neg05)*2;
        // rampa tri negativa
        // 2 - seconda metà
        shape3phasor = (shape3negpos > 0)*shape3negpos;
```

```
        // rampa tri negativa
        // 1 - prima metà
        shape3saw = ((shape3negpos < 0)*shape3negpos)*-1;

/* ------------------------------------ */

    // OSCILLATORE

    /* ------------------------------------
       FASORE
       ------------------------------------ */

    // FASORE
    // da cui derivano matematicamente le forme d'onda:

    decimale3(step3)= step3-int(step3);
    // reset interi
    fasoreosc3 = (frequency3/ma.SR) : (+ : decimale3) ~ _;
    // controllo frequenza,
    // e loop ad ogni reset


    // forme d'onda ricavate a partire dal fasore:

    /* ------------------------------------
       QUADRA
       ------------------------------------ */
    quadra2out = (fasoreosc3 > shape3*-1 +1) *2 -1;
    // Imposto condizione su fasore (vero o falso; 1 o 0)
    // la shape modifica la soglia della condizione,
    // e trasformo il segnale rendendolo bipolare

    /* ------------------------------------
       DENTE DI SEGA A RAMPA
       ------------------------------------ */
    sawrampa2out = (fasoreosc3 *2 -1) * shape3phasor;
    // trasformo il fasore in bipolare
    // e imposto un controllo della shape ascendente da 0.5 a 1.

    /* ------------------------------------
       DENTE DI SEGA
       ------------------------------------ */
    sawimpulse2out = (fasoreosc3 *-1 +0.5) * 2 * shape3saw;
    // trasformo il fasore in bipolare ma ribaltando le fasi
    // e imposto un controllo della shape discendente da 0. a 0.5

    /* ------------------------------------
       TRIANGOLARE
       ------------------------------------ */
    negative2tophasor = fasoreosc3 *-1 +1;
```

```
    // ribalto fase del fasore - da 1. a 0.
    tri2maj05 = ((fasoreosc3 > 0.5) * negative2tophasor);
    // passa solo il segnale > di 0.5 uscendo come 1.
    // ma moltiplico l'1. per la seconda parte
    // del fasore ribaltato
    // ottenendo un movimento da 0.5 a 0.
    tri2min05 = (fasoreosc3 < 0.5) * fasoreosc3;
    // passa solo il segnale < di 0.5 uscendo come 1.
    // ma moltiplico l'1. per la prima parte
    // del fasore (normale)
    tri2sum = tri2maj05 + tri2min05;
    // costruisco la mia onda triangolare con le somme
    tri2out = (tri2sum - 0.25) * 4 * shape3final;
    // riscalo rendendo l'onda triangolare bipolare
    // imposto il controllo della shape triangolare:
    // da 0. a 0.5 ascendente e da 0.5 a 1. discendente


    // OUT FUNZIONE
    oscquadra2out = quadra2out * ampquadra3;
    // uscita oscillatore con controllo della shape e ampiezza
    triangolare2out = (tri2out + sawimpulse2out + sawrampa2out)
    * amptri3;
    // uscita oscillatore con controllo della shape e ampiezza


    funzioneuscita3 = oscquadra2out+triangolare2out;
    // uscita dalla funzione
};

// FINE DEL CODICE
```

Fine del Codice dunque determinato nella sua interezza dalle tre funzioni oscillatore, e richiamabili a piacimento per la loro esecuzione nel process iniziale.

4 Conclusioni

Il fine di questa ricerca è quello di creare un codice aperto che possa essere un punto d'inizio, o di informazione, verso la realizzazione di questi oscillatori analogici del VCS3 in un porting efficace nel dominio digitale, e che possa al contempo informare un utente potenzialmente inesperto verso l'utilizzo pratico del linguaggio di programmazione FAUST (Grame). Ma come accennato precedentemente, seppur qui vengono presentati dei metodi per la realizzazione di questi oscillatori, non è da escludere che però possano non essere il metodo più efficace per realizzare oscillatori nel dominio digitale: una serie di problematiche tra cui prima di tutte l'aliasing (problema affrontato durante il corso delle lezioni dell'anno accademico 2019/2020, a cui sono state trovate varie soluzioni non trattate qui). Non è stato infatti di mio interesse in questa esposizione, ma sono

problemi tipici introdotti dalla generazione di questa tipologia di oscillatori. In ogni caso non affrontare problematiche legate al filtraggio del segnale ma solo legate alla sua generazione, è comunque uno step verso il progresso di futuri studi o realizzazioni che possano portare al miglioramento di questa tipologia di oscillatori virtual-analog.

5 Referenze

5.1 Testi

1 Alias-Free Digital Synthesis of Classic Analog Waveforms, Tim Stilson, Julius Smith, CCRMA Music Department, Stanford University

2 Audio Signal Processing in Faust, Julius O. Smith III, Center for Computer Research in Music and Acoustics (CCRMA), Department of Music, Stanford University (Stanford, California 94305 USA)

3 Beat Frei, Digital Sound Generation, Institute for Computer Music and Sound Technology (ICST) (Zurich University of the Arts, Baslerstrasse 30, CH-8048 Zürich, Switzerland)

4 Electronic Music Studios (London) Ltd., EMS VCS3 Users Manual (49 Deodar road, London, S.W.15, England).

5 Etienne Gaudrain, Yann Orlarey. A FAUST Tutorial. manual, 2003. hal-02158895

6 M. V. Mathews, The Digital Computer as a Musical Instrument, Science, New Series, Vol. 142, No. 3592 (Nov. 1, 1963), pp. 553-557

7 Robert Bristow-Johnson, Wavetable Synthesis 101, A Fundamental Perspective, Wave Mechanics, Inc. (45 Kilburn St., Burlington VT 05401 USA)

5.2 Sitografia

1 Orlarey Y., Fober D., Letz S., FAUST. Functional Programming Language for Real Time Signal Processing, https://faust.grame.fr/index.html, consultato il 21/10/2020.

2 Silvi G., S-E-A-M. Sustained Electroacoustic Music, https://github.com/s-e-a-m/, consultato il 21/10/2020.