

UE TSM - Les Séries de Volterra

TD - Partie 2

KANG Jiale

November 15, 2024

Problem 1. *Euler implicate* is defined by

$$s = \frac{1 - z^{-1}}{T}$$

Question 7. Define $\nu = T\omega$, use *Euler implicate* to transform $F_1(s)$ to $F_1(z)$. Show the recurrence equation of input u_n and output y_n .

Solution. Recall that $F_1(s) = \frac{1}{1+s/\omega}$, in \mathcal{Z} -domain, $F_1(z) = \frac{\nu}{\nu+1-z^{-1}}$.

The relation between input and output is $Y(z) = F_1(z)U(z)$. Simplify this equation, we obtain

$$Y(z)(\nu + 1 - z^{-1}) = \nu U(z) \quad (1)$$

Transform equation (1) to time domain,

$$y_n = \frac{\nu}{\nu + 1} u_n + \frac{1}{\nu + 1} y_{n-1} \quad (2)$$

□

Problem 2. Program with FAUST.

Question 8. Implement F_1 with FAUST.

Solution. Equation (2) is a recurrent expression, we denote $a = \frac{\nu}{\nu+1}$, $b = \frac{1}{\nu+1}$. Code for the implementation is shown as Listing 1.

Listing 1: Code for the implementation of F_1

```
import("stdfaust.lib");

process = F1(1000);

F1(fc) = *(a) : + ~ *(b) with {
    nu = 2 * ma.PI * fc / ma.SR;
    a = nu / (nu + 1);
    b = 1 / (nu + 1);
};
```

□

Question 9. Implement a 1-layer 3-order none linear system.

Solution. Recall that block diagram of 3-order none linear system is shown as Figure 1.

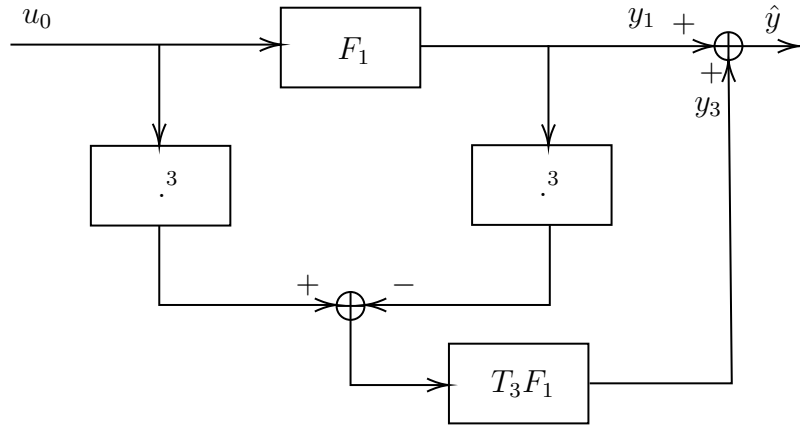


Figure 1: Block Diagram of 3-order None Linear System

According to this diagram , we could implement this system with help of Figure 2, the code of implementation for this system is shown as Listing 2.

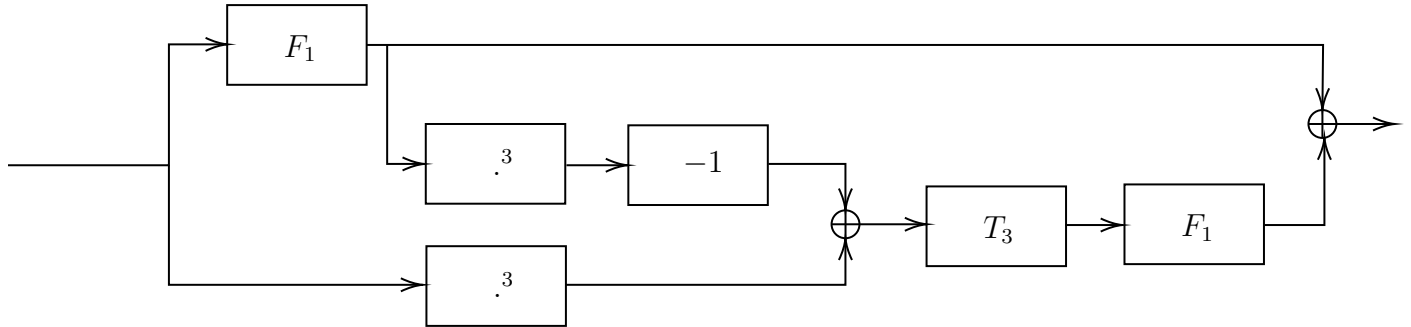


Figure 2: Diagram of Implementation with FAUST

Listing 2: Code for the implementation of 3-order None Linear System

```
import("stdfaust.lib");

process = E(1000);

F1(fc) = *(a) : + ~ *(b) with {
    nu = 2 * ma.PI * fc / ma.SR;
    a = nu / (nu + 1);
    b = 1 / (nu + 1);
};
```

```

NL(fc) = _<: F1(fc), _           // devide into 2 parts in parallel
        : (_ <: _, _), _       // branch 1 devide into 2 parts in
        : _ , ^ (3), ^ (3)     // parallel and parallel with branch 2
        : _ , * (-1), _        // identity, cubic, cubic in parallel
        : _ , * (-1), _        // identity, (-1), identity in
        : _ , (+ : *(T3): F1(fc)) // add with last two branches and
        : +;                    // follow by T3 F1
        : +;                    // two branches add

T3 = -1/3;

```

□

Question 10. Implement a 4-layer 3-order none linear system.

Solution. Recall that the kernel of a 4-layer 3-order none linear system could be expressed as

$$\begin{aligned}
H_1(s_1) &= F_1(s_1)^4 \\
H_2(s_1, s_2) &= 0 \\
H_3(s_1, s_2, s_3) &= \sum_{k=0}^3 F_1(s_1)^k F_1(s_2)^k F_1(s_3)^k F_3(s_1, s_2, s_3) F_1(s_1 + s_2 + s_3)^{3-k}
\end{aligned}$$

Therefore, block diagram of 4-layer 3-order none linear system is shown as Figure 3. Obviously, it parallels 4 \mathcal{M} and 1 \mathcal{S} with input $\{u_0, 0\}$.

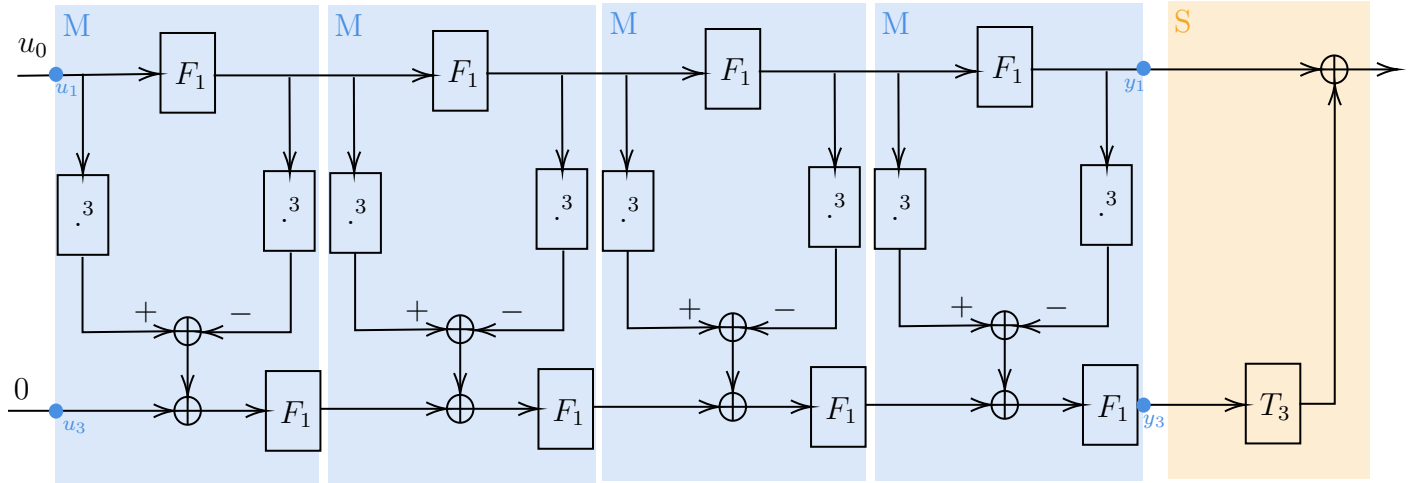


Figure 3: Block Diagram of 4-layer 3-order None Linear System

The code of implementation for this system is shown as Listing 3.

Listing 3: Code for the implementation of 4-layer 3-order None Linear System

```

import("stdfaust.lib");

```

```

process = fmoog(1000);

T3 = -1/3;
fmoog(fc) = _, 0 : M : M : M : M : S with {
    F1 = *(a) : + ~ *(b) with {
        nu = 2 * ma.PI * fc / ma.SR;
        a = nu / (nu + 1);
        b = 1 / (nu + 1);
    };

    M(u1, u3) = F1(u1), F1(u3 + u1^3 - F1(u1)^3);

    S(y1, y3) = y1 + T3 * y3;
};

```

□

Question 11. At what amplitude does distortion begin to be perceptible? To what order the sound seems to be real?

Solution. When we augment amplitude of the sound, the distortion begins to happen. We could perceive this distortion until its amplitude > 1.5 . When its amplitude < 2 , the sound seems to be real, because higher the amplitude could be, more non-harmonic frequencies will appear.

□

Question 12. When changing ω , is the stability of this filter still guaranteed?

Solution. As the cutoff frequency ω increases, the stability of the system can no longer be guaranteed. Because high frequencies non-harmonic components appear when we increase ω . This indicates that the system has entered the nonlinear region, resulting in non-harmonic distortion, and it may generate self-excited oscillations under the high cut off frequency. □

A Simulation code in FAUST

```

import("stdfaust.lib");

process = source(freq, delta) : drive(fmoog(fc));

T3 = -1/3 * checkbox("NL");

fmoog(fc) = _, 0 : M : M : M : M : S with {
    F1 = *(a) : + ~ *(b) with {
        nu = 2 * ma.PI * fc / ma.SR;
        a = nu / (nu + 1);
        b = 1 / (nu + 1);
    };

    M(u1, u3) = F1(u1), F1(u3 + u1^3 - F1(u1)^3);
};

```

```

    S(y1, y3) = y1 + T3 * y3;
};

drive(C) = *(g) : C : /(g) with {
    g = vslider("drive[style:knob]", 1, 0.1, 20, 0.1);
};

source(f1, df) = os.square(f1) + os.square(f1 + df) : *(0.5);
freq = vslider("freq[style:knob][scale:log][unit:Hz]", 330, 20, 5000, 1);
delta = vslider("delta[style:knob][unit:Hz]", 0.1, 0.05, 2, 0.05);
fc = vslider("fc[style:knob][scale:log][unit:Hz]", 1000, 20, 5000, 1);

```

B Auto Documentation

fmoog

November 15, 2024

compile'options	-lang cpp -ct 1 -es 1 -mcd 16 -mdd 1024 -mdy 33 -single -ftz 0
filename	fmoog.dsp
maths.lib/author	GRAME
maths.lib/copyright	GRAME
maths.lib/license	LGPL with exception
maths.lib/name	Faust Math Library
maths.lib/version	2.8.0
name	fmoog
oscillators.lib/lf sawpos:author	Bart Brouns, revised by Stéphane Letz
oscillators.lib/lf sawpos:licence	STK-4.3
oscillators.lib/name	Faust Oscillator Library
oscillators.lib/sawN:author	Julius O. Smith III
oscillators.lib/sawN:license	STK-4.3
oscillators.lib/version	1.5.1
platform.lib/name	Generic Platform Library
platform.lib/version	1.3.0

This document provides a mathematical description of the Faust program text stored in the `fmoog.dsp` file. See the notice in Section 3 (page 4) for details.

1 Mathematical definition of process

The *fmoog* program evaluates the signal transformer denoted by **process**, which is mathematically defined as follows:

1. Output signal y such that

$$y(t) = p_{28}(t) \cdot (r_2(t) - p_{27}(t) \cdot r_1(t))$$

2. Input signal (none)
3. User-interface input signals u_{si} for $i \in [1, 4]$ and u_{c1} such that

"delta" (Hz)	$u_{s1}(t) \in [0.05, 2]$	(default value = 0.1)
"freq" (Hz)	$u_{s2}(t) \in [20, 5000]$	(default value = 330)
"drive"	$u_{s3}(t) \in [0.1, 20]$	(default value = 1)
"fc" (Hz)	$u_{s4}(t) \in [20, 5000]$	(default value = 1000)
"NL"	$u_{c1}(t) \in \{0, 1\}$	(default value = 0)

4. Intermediate signals p_i for $i \in [1, 28]$, s_i for $i \in [1, 12]$, r_i for $i \in [1, 11]$ and q_i for $i \in [1, 2]$ such that

$$p_1(t) = \max(u_{s2}(t) + u_{s1}(t), 23.4489496824621)$$

$$p_2(t) = \max(20, |p_1(t)|)$$

$$p_3(t) = k_2 \cdot p_2(t)$$

$$p_4(t) = \left(\frac{k_3}{p_2(t)} \right)$$

$$p_5(t) = \max\left(0, \min\left(2047, \frac{k_4}{p_1(t)}\right)\right)$$

$$p_6(t) = \text{int}(p_5(t))$$

$$p_7(t) = \lfloor p_5(t) \rfloor$$

$$p_8(t) = (p_7(t) + (1 - p_5(t)))$$

$$p_9(t) = \max(u_{s2}(t), 23.4489496824621)$$

$$p_{10}(t) = \max(20, |p_9(t)|)$$

$$p_{11}(t) = k_2 \cdot p_{10}(t)$$

$$p_{12}(t) = \left(\frac{k_3}{p_{10}(t)} \right)$$

$$p_{13}(t) = \max\left(0, \min\left(2047, \frac{k_4}{p_9(t)}\right)\right)$$

$$p_{14}(t) = \text{int}(p_{13}(t))$$

$$p_{15}(t) = p_{14}(t) \oplus 1$$

$$p_{16}(t) = \lfloor p_{13}(t) \rfloor$$

$$p_{17}(t) = (p_{13}(t) - p_{16}(t))$$

$$p_{18}(t) = (p_{16}(t) + (1 - p_{13}(t)))$$

$$p_{19}(t) = p_6(t) \oplus 1$$

$$p_{20}(t) = (p_5(t) - p_7(t))$$

$$p_{21}(t) = \left(\frac{1}{p_2(t)} \right)$$

$$p_{22}(t) = \left(\frac{1}{p_{10}(t)} \right)$$

$$\begin{aligned}
p_{23}(t) &= k_5 \cdot u_{s_4}(t) \cdot u_{s_3}(t) \\
p_{24}(t) &= k_6 \cdot u_{s_4}(t) \\
p_{25}(t) &= \left(\frac{1}{p_{24}(t) + 1} \right) \\
p_{26}(t) &= 0.5 \cdot u_{s_3}(t) \\
p_{27}(t) &= 0.3333333333333333 \cdot u_{c_1}(t) \\
p_{28}(t) &= \left(\frac{1}{u_{s_3}(t)} \right)
\end{aligned}$$

$$\begin{aligned}
s_1(t) &= 1 \ominus r_7(t-1) \\
s_2(t) &= 2 \cdot r_6(t) + -1^2 \\
s_3(t) &= (s_2(t) - s_2(t-1)) \\
s_4(t) &= r_7(t-1) \\
s_5(t) &= p_4(t) \cdot s_4(t) \cdot s_3(t) \\
s_6(t) &= 2 \cdot r_{10}(t) + -1^2 \\
s_7(t) &= (s_6(t) - s_6(t-1)) \\
s_8(t) &= p_{12}(t) \cdot s_4(t) \cdot s_7(t) \\
s_9(t) &= (k_3 \cdot s_4(t) \cdot (p_{22}(t) \cdot s_7(t) + p_{21}(t) \cdot s_3(t)) - p_{20}(t) \cdot s_5(t-p_{19}(t)) \\
&\quad + p_{18}(t) \cdot s_8(t-p_{14}(t)) + p_{17}(t) \cdot s_8(t-p_{15}(t)) + p_8(t) \cdot s_5(t-p_6(t))) \\
s_{10}(t) &= r_3(t)^3 \\
s_{11}(t) &= r_4(t)^3 \\
s_{12}(t) &= r_5(t)^3
\end{aligned}$$

$$\begin{aligned}
r_7(t) &= 1 \\
r_6(t) &= q_1(t) - \lfloor q_1(t) \rfloor \\
r_{10}(t) &= q_2(t) - \lfloor q_2(t) \rfloor \\
r_5(t) &= p_{25}(t) \cdot (r_5(t-1) + p_{23}(t) \cdot s_9(t)) \\
r_4(t) &= p_{25}(t) \cdot (r_4(t-1) + p_{24}(t) \cdot r_5(t)) \\
r_3(t) &= p_{25}(t) \cdot (r_3(t-1) + p_{24}(t) \cdot r_4(t)) \\
r_2(t) &= p_{25}(t) \cdot (r_2(t-1) + p_{24}(t) \cdot r_3(t)) \\
r_{15}(t) &= p_{25}(t) \cdot \left(r_{15}(t-1) + p_{24}(t) \cdot \left(p_{26}(t) \cdot s_9(t)^3 - s_{12}(t) \right) \right) \\
r_{14}(t) &= p_{25}(t) \cdot (r_{14}(t-1) + p_{24}(t) \cdot (r_{15}(t) + s_{12}(t) - s_{11}(t))) \\
r_{13}(t) &= p_{25}(t) \cdot (r_{13}(t-1) + p_{24}(t) \cdot (r_{14}(t) + s_{11}(t) - s_{10}(t))) \\
r_1(t) &= p_{25}(t) \cdot \left(r_1(t-1) + p_{24}(t) \cdot \left(r_{13}(t) + s_{10}(t) - r_2(t)^3 \right) \right)
\end{aligned}$$

$$q_1(t) = \begin{cases} p_3(t) + r_6(t-1) & \text{if } s_1(t) = 0 \\ 0 & \text{if } s_1(t) = 1 \end{cases} \quad q_2(t) = \begin{cases} p_{11}(t) + r_{10}(t-1) & \text{if } s_1(t) = 0 \\ 0 & \text{if } s_1(t) = 1 \end{cases}$$

5. Constants k_i for $i \in [1, 6]$ such that

$$k_1 = \min(192000, \max(1, f_S))$$

$$k_2 = \left(\frac{1}{k_1}\right)$$

$$k_3 = 0.25 \cdot k_1$$

$$k_4 = 0.5 \cdot k_1$$

$$k_5 = \left(\frac{\pi}{k_1}\right)$$

$$k_6 = \left(\frac{2 \cdot \pi}{k_1}\right)$$

2 Block diagram of process

The block diagram of `process` is shown on Figure 1 (page 4).

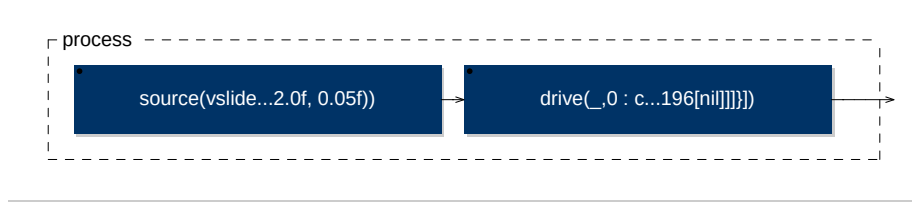


Figure 1: Block diagram of `process`

3 Notice

- This document was generated using Faust version 2.75.16 on November 15, 2024.
- The value of a Faust program is the result of applying the signal transformer denoted by the expression to which the `process` identifier is bound to input signals, running at the f_S sampling frequency.
- Faust (*Functional Audio Stream*) is a functional programming language designed for synchronous real-time signal processing and synthesis applications. A Faust program is a set of bindings of identifiers to expressions

that denote signal transformers. A signal s in S is a function mapping¹ times $t \in \mathbb{Z}$ to values $s(t) \in \mathbb{R}$, while a signal transformer is a function from S^n to S^m , where $n, m \in \mathbb{N}$. See the Faust manual for additional information (<http://faust.grame.fr>).

- Every mathematical formula derived from a Faust expression is assumed, in this document, to having been normalized (in an implementation-dependent manner) by the Faust compiler.
- A block diagram is a graphical representation of the Faust binding of an identifier I to an expression E ; each graph is put in a box labeled by I . Subexpressions of E are recursively displayed as long as the whole picture fits in one page.

- $\forall x \in \mathbb{R}$,

$$\text{int}(x) = \begin{cases} \lfloor x \rfloor & \text{if } x > 0 \\ \lceil x \rceil & \text{if } x < 0 \\ 0 & \text{if } x = 0 \end{cases}.$$

- This document uses the following integer operations:

<i>operation</i>	<i>name</i>	<i>semantics</i>
$i \oplus j$	integer addition	$\text{normalize}(i + j)$, in \mathbb{Z}
$i \ominus j$	integer substraction	$\text{normalize}(i - j)$, in \mathbb{Z}

Integer operations in Faust are inspired by the semantics of operations on the n -bit two's complement representation of integer numbers; they are internal composition laws on the subset $[-2^{n-1}, 2^{n-1} - 1]$ of \mathbb{Z} , with $n = 32$. For any integer binary operation \times on \mathbb{Z} , the \otimes operation is defined as: $i \otimes j = \text{normalize}(i \times j)$, with

$$\text{normalize}(i) = i - N \cdot \text{sign}(i) \cdot \left\lfloor \frac{|i| + N/2 + (\text{sign}(i) - 1)/2}{N} \right\rfloor,$$

where $N = 2^n$ and $\text{sign}(i) = 0$ if $i = 0$ and $i/|i|$ otherwise. Unary integer operations are defined likewise.

- The `fmoog-mdoc/` directory may also include the following subdirectories:
 - `cpp/` for Faust compiled code;
 - `pdf/` which contains this document;
 - `src/` for all Faust sources used (even libraries);
 - `svg/` for block diagrams, encoded using the Scalable Vector Graphics format (<http://www.w3.org/Graphics/SVG/>);
 - `tex/` for the \LaTeX source of this document.

¹Faust assumes that $\forall s \in S, \forall t \in \mathbb{Z}, s(t) = 0$ when $t < 0$.

4 Faust code listings

This section provides the listings of the Faust code used to generate this document, including dependencies.

Listing 1: fmoog.dsp

```
1 declare filename "fmoog.dsp";
2 declare name "fmoog";
3 import("stdfaust.lib");
4
5 process = source(freq, delta) : drive(fmoog(fc));
6
7 T3 = -1/3 * checkbox("NL");
8
9 fmoog(fc) = _, 0 : M : M : M : M : S with {
10   F1 = *(a) : + ~ *(b) with {
11     nu = 2 * ma.PI * fc / ma.SR;
12     a = nu / (nu + 1);
13     b = 1 / (nu + 1);
14   };
15
16   M(u1, u3) = F1(u1), F1(u3 + u1^3 - F1(u1)^3);
17
18   S(y1, y3) = y1 + T3 * y3;
19 };
20
21 drive(C) = *(g) : C : /(g) with {
22   g = vslider("drive[style:knob]", 1, 0.1, 20, 0.1);
23 };
24
25 source(f1, df) = os.square(f1) + os.square(f1 + df) : *(0.5);
26 freq = vslider("freq[style:knob][scale:log][unit:Hz]", 330, 20, 5000, 1);
27 delta = vslider("delta[style:knob][unit:Hz]", 0.1, 0.05, 2, 0.05);
28 fc = vslider("fc[style:knob][scale:log][unit:Hz]", 1000, 20, 5000, 1);
```

Listing 2: stdfaust.lib

```
1 //##### stdfaust.lib #####
2 // The purpose of this library is to give access to all the Faust standard libraries
3 // through a series of environments.
4 //#####
5
6 aa = library("aanl.lib");
7 sf = library("all.lib");
8 an = library("analyzers.lib");
9 ba = library("basics.lib");
10 co = library("compressors.lib");
11 de = library("delays.lib");
12 dm = library("demos.lib");
13 dx = library("dx7.lib");
14 en = library("envelopes.lib");
15 fd = library("fds.lib");
16 fi = library("filters.lib");
17 ho = library("hoa.lib");
18 it = library("interpolators.lib");
19 ma = library("maths.lib");
20 mi = library("mi.lib");
21 ef = library("misceffects.lib");
22 os = library("oscillators.lib");
```

```

23 no = library("noises.lib");
24 pf = library("phaflangers.lib");
25 pl = library("platform.lib");
26 pm = library("physmodels.lib");
27 qu = library("quantizers.lib");
28 rm = library("reducemaps.lib");
29 re = library("reverbs.lib");
30 ro = library("routes.lib");
31 sp = library("spats.lib");
32 si = library("signals.lib");
33 so = library("soundfiles.lib");
34 sy = library("synths.lib");
35 ve = library("vaeffects.lib");
36 vl = library("version.lib");
37 wa = library("webaudio.lib");
38 wd = library("wdmodels.lib");

```

Listing 3: oscillators.lib

```

1  //##### oscillators.lib #####
2  // This library contains a collection of sound generators. Its official prefix is 'os'.
3  //
4  // The oscillators library is organized into 9 sections:
5  //
6  // * [Wave-Table-Based Oscillators] (#wave-table-based-oscillators)
7  // * [Low Frequency Oscillators] (#low-frequency-oscillators)
8  // * [Low Frequency Sawtooths] (#low-frequency-sawtooths)
9  // * [Alias-Suppressed Sawtooth] (#alias-suppressed-sawtooth)
10 // * [Alias-Suppressed Pulse, Square, and Impulse Trains] (#alias-suppressed-pulse-square-and-
    impulse-trains)
11 // * [Filter-Based Oscillators] (#filter-based-oscillators)
12 // * [Waveguide-Resonator-Based Oscillators] (#waveguide-resonator-based-oscillators)
13 // * [Casio CZ Oscillators] (#casio-cz-oscillators)
14 // * [PolyBLEP-Based Oscillators] (#polyblep-based-oscillators)
15 //
16 // #### References
17 // * <https://github.com/grame-cncm/faustlibraries/blob/master/oscillators.lib>
18 //#####
19
20 /*****
21 *****/
22 FAUST library file, GRAME section
23
24 Except where noted otherwise, Copyright (C) 2003-2017 by GRAME,
25 Centre National de Creation Musicale.
26 -----
27 GRAME LICENSE
28
29 This program is free software; you can redistribute it and/or modify
30 it under the terms of the GNU Lesser General Public License as
31 published by the Free Software Foundation; either version 2.1 of the
32 License, or (at your option) any later version.
33
34 This program is distributed in the hope that it will be useful,
35 but WITHOUT ANY WARRANTY; without even the implied warranty of
36 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
37 GNU Lesser General Public License for more details.
38
39 You should have received a copy of the GNU Lesser General Public
40 License along with the GNU C Library; if not, write to the Free
41 Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
42 02111-1307 USA.
43
44 EXCEPTION TO THE LGPL LICENSE : As a special exception, you may create a

```

```

45 larger FAUST program which directly or indirectly imports this library
46 file and still distribute the compiled code generated by the FAUST
47 compiler, or a modified version of this compiled code, under your own
48 copyright and license. This EXCEPTION TO THE LGPL LICENSE explicitly
49 grants you the right to freely choose the license for the resulting
50 compiled code. In particular the resulting compiled code has no obligation
51 to be LGPL or GPL. For example you are free to choose a commercial or
52 closed source license or any other license if you decide so.
53 *****
54 *****/
55
56 ma = library("maths.lib");
57 ba = library("basics.lib");
58 fi = library("filters.lib");
59 si = library("signals.lib");
60
61 declare name "Faust Oscillator Library";
62 declare version "1.5.1";
63
64 // This library contains platform specific constants
65 pl = library("platform.lib");
66
67 //=====Oscillators based on mathematical functions=====
68 //
69 // Note that there is a numerical problem with several phasor functions built using the
70 // internal
71 // 'phasor_imp'. The reason is that the incremental step is smaller than 'ma.EPSILON', which
72 // happens with very small frequencies,
73 // so it will have no effect when summed to 1, but it will be enough to make the fractional
74 // function wrap
75 // around when summed to 0. An example of this problem can be observed when running the
76 // following code:
77 //
78 // 'process = os.phasor(1.0, -.001);'
79 //
80 // The output of this program is the sequence 1, 0, 1, 0, 1... This happens because the
81 // negative incremental
82 // step is greater than '-ma.EPSILON', which will have no effect when summed to 1, but it
83 // will be significant
84 // enough to make the fractional function wrap around when summed to 0.
85 //
86 // The incremental step can be clipped to guarantee that the phasor will
87 // always run correctly for its full cycle, otherwise, for increments smaller than 'ma.
88 // EPSILON',
89 // phasor would initially run but it'd eventually get stuck once the output gets big enough.
90 //
91 // All functions using 'phasor_imp' are affected by this problem, but a safer
92 // version is implemented, and can be used alternatively by setting 'SAFE=1' in the
93 // environment using
94 // [explicit substitution](https://faustdoc.grame.fr/manual/syntax/#explicit-substitution)
95 // syntax.
96 //
97 // For example: 'process = os[SAFE=1;].phasor(1.0, -.001);' will use the safer implementation
98 // of 'phasor_imp'.
99 //=====
100 //=====Wave-Table-Based Oscillators=====
101 // Oscillators using tables. The table size is set by the
102 // [pl.tablesize](https://github.com/grame-cncm/faustlibraries/blob/master/platform.lib)
103 // constant.
104 //=====
105 // Global parameter to use the safer version of 'phasor_imp', but which
106 // could be used in other functions as well.
107
108 SAFE = 0; // 0: use the faster version, 1: use the safer version
109
110 //-----'(os.)sinwaveform'-----

```

```

102 // Sine waveform ready to use with a 'rdtable'.
103 //
104 // ### Usage
105 //
106 // '''
107 // sinwaveform(tablesize) : _
108 // '''
109 //
110 // Where:
111 //
112 // * 'tablesize': the table size
113 //-----
114 sinwaveform(tablesize) =
115     sin(float(ba.period(tablesize)) * (2.0 * ma.PI) / float(tablesize));
116
117 //-----'(os.)coswaveform'-----
118 // Cosine waveform ready to use with a 'rdtable'.
119 //
120 //
121 // ### Usage
122 //
123 // '''
124 // coswaveform(tablesize) : _
125 // '''
126 //
127 // Where:
128 //
129 // * 'tablesize': the table size
130 //-----
131 coswaveform(tablesize) =
132     cos(float(ba.period(tablesize)) * (2.0 * ma.PI) / float(tablesize));
133
134 // Possibly faster version using integer arithmetic
135 phasor_env(freq, N) = environment {
136
137     //----- GLOBAL PARAMS
138     nbits = 31;
139     tablesize = 1<<N;
140     accuracy = int(nbits - N);
141     mask = (1<<nbits)-1;
142     inc(step) = int(tablesize * step * (1<<accuracy));
143
144     //----- LAMBDA DSP CASE
145     lambda(inc_op) = (inc_op : &(mask)) ~ _ : >>(accuracy) : /(tablesize);
146
147     //----- MINIMAL CASE
148     hsp(0,0) = lambda(+ (inc(freq/ma.SR)'));
149
150     //----- GENERAL CASE
151     hsp(reset,phase) = lambda(select2(hard_reset,+(inc(freq/ma.SR)),inc(phase)))
152     with {
153         hard_reset = (1-1')|reset;
154     };
155 };
156 declare phasor_env author "Pierre Mascarade Relano, Maxime Sirbu, Stéphane Letz";
157
158 // Generic phasor with 'reset' and 'phase' parameters to be specialised in concrete use-cases
159 phasor_imp(freq, reset, phase) = (select2(hard_reset, +(incr(SAFE)), phase) : ma.decimal) ~
160 with {
161     incr_aux = freq/ma.SR;
162
163     // Faster but less accurate version
164     incr(0) = incr_aux;
165
166     // To make sure that the incremental step is greater or equal to EPSILON or
167     // less than or equal to -EPSILON to avoid numerical problems.

```

```

168 // A frequency of 0Hz can still be used to freeze the phasor.
169 incr(1)= (freq != 0) * ba.if(freq < 0, min(-1.0 * ma.EPSILON, incr_aux), max(ma.EPSILON,
    incr_aux));
170
171 // To correctly start at 'phase' at the first sample
172 hard_reset = (1-1')|reset;
173 };
174
175 // Possibly faster version using integer arithmetic
176 // phasor_imp(freq, reset, phase) = phasor_env(freq, 16).hsp(reset, phase);
177
178 // Version to be used with tables
179 phasor_table(tablesize, freq, reset, phase) = phasor_imp(freq, reset, phase) : *(float(
    tablesize));
180
181 //-----'(os.)phasor'-----
182 // A simple phasor to be used with a 'rdtable'.
183 // 'phasor' is a standard Faust function.
184 //
185 //
186 // #### Usage
187 //
188 // '''
189 // phasor(tablesize,freq) : _
190 // '''
191 //
192 // Where:
193 //
194 // * 'tablesize': the table size
195 // * 'freq': the frequency in Hz
196 //
197 // Note that 'tablesize' is just a multiplier for the output of a unit-amp phasor
198 // so 'phasor(1.0, freq)' can be used to generate a phasor output in the range [0, 1].
199 //-----
200 phasor(tablesize, freq) = phasor_table(tablesize, freq, 0, 0);
201
202 //-----'(os.)hs_phasor'-----
203 // Hardsyncing phasor to be used with a 'rdtable'.
204 //
205 //
206 // #### Usage
207 //
208 // '''
209 // hs_phasor(tablesize,freq,reset) : _
210 // '''
211 //
212 // Where:
213 //
214 // * 'tablesize': the table size
215 // * 'freq': the frequency in Hz
216 // * 'reset': a reset signal, reset phase to 0 when equal to 1
217 //-----
218 declare hs_phasor author "Mike Olsen, revised by Stéphane Letz";
219
220 hs_phasor(tablesize, freq, reset) = phasor_table(tablesize, freq, reset, 0);
221
222 //-----'(os.)hsp_phasor'-----
223 // Hardsyncing phasor with selectable phase to be used with a 'rdtable'.
224 //
225 //
226 // #### Usage
227 //
228 // '''
229 // hsp_phasor(tablesize,freq,reset,phase)
230 // '''
231 //
232 // Where:
233 //

```

```

234 // * 'tablesize': the table size
235 // * 'freq': the frequency in Hz
236 // * 'reset': reset the oscillator to phase when equal to 1
237 // * 'phase': phase between 0 and 1
238 //-----
239 declare hsp_phasor author "Christophe Lebreton, revised by Stéphane Letz";
240
241 hsp_phasor(tablesize, freq, reset, phase) = phasor_table(tablesize, freq, reset, phase);
242
243
244 //-----'(os.)oscscin'-----
245 // Sine wave oscillator.
246 // 'oscscin' is a standard Faust function.
247 //
248 // #### Usage
249 //
250 // ' '
251 // oscscin(freq) : _
252 // ' '
253 //
254 // Where:
255 //
256 // * 'freq': the frequency in Hz
257 //-----
258 oscscin(freq) = rdtable(tablesize, sinwaveform(tablesize), int(phasor(tablesize,freq)))
259 with {
260     tablesize = pl.tablesize;
261 };
262
263
264 //-----'(os.)hs_oscscin'-----
265 // Sin lookup table with hardsyncing phase.
266 //
267 // #### Usage
268 //
269 // ' '
270 // hs_oscscin(freq,reset) : _
271 // ' '
272 //
273 // Where:
274 //
275 // * 'freq': the frequency in Hz
276 // * 'reset': reset the oscillator to 0 when equal to 1
277 //-----
278 declare hs_oscscin author "Mike Olsen";
279
280 hs_oscscin(freq,reset) = rdtable(tablesize, sinwaveform(tablesize), int(hs_phasor(tablesize,
281     freq,reset)))
282 with {
283     tablesize = pl.tablesize;
284 };
285
286
287 //-----'(os.)osccos'-----
288 // Cosine wave oscillator.
289 //
290 // #### Usage
291 //
292 // ' '
293 // osccos(freq) : _
294 // ' '
295 //
296 // Where:
297 //
298 // * 'freq': the frequency in Hz
299 //-----
300 osccos(freq) = rdtable(tablesize, coswaveform(tablesize), int(phasor(tablesize,freq)))
301 with {

```



```

301     tablesize = pl.tablesize;
302 };
303
304 //-----'(os.)hs_osccos'-----
305 // Cos lookup table with hardsyncing phase.
306 //
307 // #### Usage
308 //
309 // '
310 // hs_osccos(freq,reset) : _
311 // '
312 //
313 // Where:
314 //
315 // * 'freq': the frequency in Hz
316 // * 'reset': reset the oscillator to 0 when equal to 1
317 //-----
318 declare hs_osccos author "Stéphane Letz";
319
320 hs_osccos(freq,reset) = rdtable(tablesize, coswaveform(tablesize), int(hs_phasor(tablesize,
321     freq,reset)))
322 with {
323     tablesize = pl.tablesize;
324 };
325
326 //-----'(os.)oscp'-----
327 // A sine wave generator with controllable phase.
328 //
329 // #### Usage
330 //
331 // '
332 // oscp(freq,phase) : _
333 // '
334 //
335 // Where:
336 //
337 // * 'freq': the frequency in Hz
338 // * 'phase': the phase in radian
339 //-----
340 oscp(freq,phase) = oscsin(freq) * cos(phase) + osccos(freq) * sin(phase);
341
342 //-----'(os.)osci'-----
343 // Interpolated phase sine wave oscillator.
344 //
345 // #### Usage
346 //
347 // '
348 // osci(freq) : _
349 // '
350 //
351 // Where:
352 //
353 // * 'freq': the frequency in Hz
354 //-----
355 osci(freq) = s1 + d * (s2 - s1)
356 with {
357     tablesize = pl.tablesize;
358     i = int(phasor(tablesize,freq));
359     d = ma.decimal(phasor(tablesize,freq));
360     s1 = rdtable(tablesize+1,sinwaveform(tablesize),i);
361     s2 = rdtable(tablesize+1,sinwaveform(tablesize),i+1);
362 };
363
364 //-----'(os.)osc'-----
365
366

```

```

368 // Default sine wave oscillator (same as [oscsin](#oscsin)).
369 // 'osc' is a standard Faust function.
370 //
371 // #### Usage
372 //
373 // '''
374 // osc(freq) : _
375 // '''
376 //
377 // Where:
378 //
379 // * 'freq': the frequency in Hz
380 //-----
381 osc = oscsin;
382
383
384 //-----'(os.)m_oscsin'-----
385 // Sine wave oscillator based on the 'sin' mathematical function.
386 //
387 // #### Usage
388 //
389 // '''
390 // m_oscsin(freq) : _
391 // '''
392 //
393 // Where:
394 //
395 // * 'freq': the frequency in Hz
396 //-----
397 m_oscsin(freq) = lf_sawpos(freq) : *(2*ma.PI) : sin;
398
399
400 //-----'(os.)m_osccos'-----
401 // Sine wave oscillator based on the 'cos' mathematical function.
402 //
403 // #### Usage
404 //
405 // '''
406 // m_osccos(freq) : _
407 // '''
408 //
409 // Where:
410 //
411 // * 'freq': the frequency in Hz
412 //-----
413 m_osccos(freq) = lf_sawpos(freq) : *(2*ma.PI) : cos;
414
415
416 // end GRAME section
417 //#####
418 //#####
419 FAUST library file, jos section
420
421 Except where noted otherwise, The Faust functions below in this
422 section are Copyright (C) 2003-2022 by Julius O. Smith III <jos@ccrma.stanford.edu>
423 ([jos](http://ccrma.stanford.edu/~jos/)), and released under the
424 (MIT-style) [STK-4.3](#stk-4.3-license) license.
425
426 The MarkDown comments in this section are Copyright 2016-2017 by Romain
427 Michon and Julius O. Smith III, and are released under the
428 [CC4I](https://creativecommons.org/licenses/by/4.0/) license (TODO: if/when Romain agrees)
429
430 *****/
431
432 //=====Low Frequency Oscillators=====
433 // Low Frequency Oscillators (LFOs) have prefix 'lf_'
434 // (no aliasing suppression, since it is inaudible at LF).
435 // Use 'sawN' and its derivatives for audio oscillators with suppressed aliasing.

```

```

436 //=====
437
438 //-----'(os.)lf_imptrain'-----
439 // Unit-amplitude low-frequency impulse train.
440 // 'lf_imptrain' is a standard Faust function.
441
442 // #### Usage
443 //
444 // ' '
445 // lf_imptrain(freq) : _
446 // ' '
447
448 // Where:
449 //
450 // * 'freq': frequency in Hz
451 //-----
452 lf_imptrain(freq) = lf_sawpos(freq)<:-(mem)<0; // definition below
453
454 //-----'(os.)lf_pulsestrainpos'-----
455 // Unit-amplitude nonnegative LF pulse train, duty cycle between 0 and 1.
456 //
457 //
458 //
459 // #### Usage
460 //
461 // ' '
462 // lf_pulsestrainpos(freq, duty) : _
463 // ' '
464 //
465 // Where:
466 //
467 // * 'freq': frequency in Hz
468 // * 'duty': duty cycle between 0 and 1
469 //-----
470 lf_pulsestrainpos(freq,duty) = float(lf_sawpos(freq) <= duty);
471
472 //pulsestrainpos = lf_pulsestrainpos; // for backward compatibility
473
474 //-----'(os.)lf_pulsestrain'-----
475 // Unit-amplitude zero-mean LF pulse train, duty cycle between 0 and 1.
476 //
477 //
478 // #### Usage
479 //
480 // ' '
481 // lf_pulsestrain(freq,duty) : _
482 // ' '
483 //
484 // Where:
485 //
486 // * 'freq': frequency in Hz
487 // * 'duty': duty cycle between 0 and 1
488 //-----
489 lf_pulsestrain(freq,duty) = 2.0*lf_pulsestrainpos(freq,duty) - 1.0;
490
491 //-----'(os.)lf_squarewavepos'-----
492 // Positive LF square wave in [0,1]
493 //
494 //
495 // #### Usage
496 //
497 // ' '
498 // lf_squarewavepos(freq) : _
499 // ' '
500 //
501 // Where:
502 //
503 // * 'freq': frequency in Hz

```

```

504 //-----'lf_squarewavepos'-----
505 lf_squarewavepos(freq) = lf_pulsetrainpos(freq,0.5);
506 // squarewavepos = lf_squarewavepos; // for backward compatibility
507
508
509 //-----'lf_squarewave'-----
510 // Zero-mean unit-amplitude LF square wave.
511 // 'lf_squarewave' is a standard Faust function.
512 //
513 // #### Usage
514 //
515 // '
516 // lf_squarewave(freq) : _
517 // '
518 //
519 // Where:
520 //
521 // * 'freq': frequency in Hz
522 //-----
523 lf_squarewave(freq) = 2.0*lf_squarewavepos(freq) - 1.0;
524 // squarewave = lf_squarewave; // for backward compatibility
525
526
527 //-----'lf_trianglepos'-----
528 // Positive unit-amplitude LF positive triangle wave.
529 //
530 // #### Usage
531 //
532 // '
533 // lf_trianglepos(freq) : _
534 // '
535 //
536 // Where:
537 //
538 // * 'freq': frequency in Hz
539 //-----
540 lf_trianglepos(freq) = 1.0-abs(saw1(freq)); // saw1 defined below
541
542
543 //-----'lf_triangle'-----
544 // Zero-mean unit-amplitude LF triangle wave.
545 // 'lf_triangle' is a standard Faust function.
546 //
547 // #### Usage
548 //
549 // '
550 // lf_triangle(freq) : _
551 // '
552 //
553 // Where:
554 //
555 // * 'freq': frequency in Hz
556 //-----
557 declare lf_triangle author "Bart Brouns";
558 declare lf_triangle licence "STK-4.3";
559
560 lf_triangle(freq) = 2.0*lf_trianglepos(freq) - 1.0;
561
562
563 //===== Low Frequency Sawtooths =====
564 // Sawtooth waveform oscillators for virtual analog synthesis et al.
565 // The 'simple' versions ('lf_rawsaw', 'lf_sawpos' and 'saw1'), are mere samplings of
566 // the ideal continuous-time ("analog") waveforms. While simple, the
567 // aliasing due to sampling is quite audible. The differentiated
568 // polynomial waveform family ('saw2', 'sawN', and derived functions)
569 // do some extra processing to suppress aliasing (not audible for
570 // very low fundamental frequencies). According to Lehtonen et al.
571 // (JASA 2012), the aliasing of 'saw2' should be inaudible at fundamental

```

```

572 // frequencies below 2 kHz or so, for a 44.1 kHz sampling rate and 60 dB SPL
573 // presentation level; fundamentals 415 and below required no aliasing
574 // suppression (i.e., 'saw1' is ok).
575 //=====
576
577 //-----'(os.)lf_rawsaw'-----
578 // Simple sawtooth waveform oscillator between 0 and period in samples.
579 //
580 // #### Usage
581 //
582 // '
583 // lf_rawsaw(periodsamps) : _
584 // '
585 //
586 // Where:
587 //
588 // * 'periodsamps': number of periods per samples
589 //-----
590 lf_rawsaw(periodsamps) = (_,periodsamps : fmod) ~ +(1.0);
591
592
593 //-----'(os.)lf_sawpos'-----
594 // Simple sawtooth waveform oscillator between 0 and 1.
595 //
596 // #### Usage
597 //
598 // '
599 // lf_sawpos(freq) : _
600 // '
601 //
602 // Where:
603 //
604 // * 'freq': frequency in Hz
605 //
606 //-----
607 declare lf_sawpos author "Bart Brouns, revised by Stéphane Letz";
608 declare lf_sawpos licence "STK-4.3";
609
610 lf_sawpos(freq) = phasor_imp(freq, 0, 0);
611
612
613 //-----'(os.)lf_sawpos_phase'-----
614 // Simple sawtooth waveform oscillator between 0 and 1
615 // with phase control.
616 //
617 // #### Usage
618 //
619 // '
620 // lf_sawpos_phase(freq, phase) : _
621 // '
622 //
623 // Where:
624 //
625 // * 'freq': frequency in Hz
626 // * 'phase': phase between 0 and 1
627 //-----
628 declare lf_sawpos_phase author "Bart Brouns, revised by Stéphane Letz";
629 declare lf_sawpos_phase licence "STK-4.3";
630
631 lf_sawpos_phase(freq,phase) = phasor_imp(freq, 0, phase);
632
633
634 //-----'(os.)lf_sawpos_reset'-----
635 // Simple sawtooth waveform oscillator between 0 and 1
636 // with reset.
637 //
638 // #### Usage
639 //

```

```

640 // '''
641 // lf_sawpos_reset(freq,reset) : _
642 // '''
643 //
644 // Where:
645 //
646 // * 'freq': frequency in Hz
647 // * 'reset': reset the oscillator to 0 when equal to 1
648 //
649 //-----
650 declare lf_sawpos_reset author "Bart Brouns, revised by Stéphane Letz";
651 declare lf_sawpos_reset licence "STK-4.3";
652
653 lf_sawpos_reset(freq,reset) = phasor_imp(freq, reset, 0);
654
655
656 //-----'(os.)lf_sawpos_phase_reset'-----
657 // Simple sawtooth waveform oscillator between 0 and 1
658 // with phase control and reset.
659 //
660 // #### Usage
661 //
662 // '''
663 // lf_sawpos_phase_reset(freq,phase,reset) : _
664 // '''
665 //
666 // Where:
667 //
668 // * 'freq': frequency in Hz
669 // * 'phase': phase between 0 and 1
670 // * 'reset': reset the oscillator to phase when equal to 1
671 //
672 //-----
673 declare lf_sawpos_phase_reset author "Bart Brouns, revised by Stéphane Letz";
674 declare lf_sawpos_phase_reset licence "STK-4.3";
675
676 lf_sawpos_phase_reset(freq,phase,reset) = phasor_imp(freq, reset, phase);
677
678
679 //-----'(os.)lf_saw'-----
680 // Simple sawtooth waveform oscillator between -1 and 1.
681 // 'lf_saw' is a standard Faust function.
682 //
683 // #### Usage
684 //
685 // '''
686 // lf_saw(freq) : _
687 // '''
688 //
689 // Where:
690 //
691 // * 'freq': frequency in Hz
692 //-----
693 declare saw1 author "Bart Brouns";
694 declare saw1 licence "STK-4.3";
695
696 saw1(freq) = 2.0 * lf_sawpos(freq) - 1.0;
697 lf_saw(freq) = saw1(freq);
698
699 //===== Alias-Suppressed Sawtooth =====
700 //-----'(os.)sawN'-----
701 // Alias-Suppressed Sawtooth Audio-Frequency Oscillator using Nth-order polynomial
702 // transitions
703 // to reduce aliasing.
704 //
705 // 'sawN(N,freq)', 'sawNp(N,freq,phase)', 'saw2dpw(freq)', 'saw2(freq)', 'saw3(freq)',
706 // 'saw4(freq)', 'sawtooth(freq)', 'saw2f2(freq)', 'saw2f4(freq)'

```

```

707 // ##### Usage
708 //
709 // '''
710 // sawN(N,freq) : _ // Nth-order aliasing-suppressed sawtooth using DPW method (see
// below)
711 // sawNp(N,freq,phase) : _ // sawN with phase offset feature
712 // saw2dpw(freq) : _ // saw2 using DPW
713 // saw2ptr(freq) : _ // saw2 using the faster, stateless PTR method
714 // saw2(freq) : _ // DPW method, but subject to change if a better method emerges
715 // saw3(freq) : _ // sawN(3)
716 // saw4(freq) : _ // sawN(4)
717 // sawtooth(freq) : _ // saw2
718 // saw2f2(freq) : _ // saw2dpw with 2nd-order droop-correction filtering
719 // saw2f4(freq) : _ // saw2dpw with 4th-order droop-correction filtering
720 // '''
721 //
722 // Where:
723 //
724 // * 'N': polynomial order, a constant numerical expression between 1 and 4
725 // * 'freq': frequency in Hz
726 // * 'phase': phase between 0 and 1
727 //
728 // ##### Method
729 // Differentiated Polynomial Wave (DPW).
730 //
731 // ##### Reference
732 // "Alias-Suppressed Oscillators based on Differentiated Polynomial Waveforms",
733 // Vesa Valimäki, Juhan Nam, Julius Smith, and Jonathan Abel,
734 // IEEE Tr. Audio, Speech, and Language Processing (IEEE-ASLP),
735 // Vol. 18, no. 5, pp 786-798, May 2010.
736 // 10.1109/TASL.2009.2026507.
737 //
738 // ##### Notes
739 // The polynomial order 'N' is limited to 4 because noise has been
740 // observed at very low 'freq' values. (LFO sawtooths should of course
741 // be generated using 'lf_sawpos' instead.)
742 //-----
743 declare sawN author "Julius O. Smith III";
744 declare sawN license "STK-4.3";
745 // --- sawN for N = 1 to 4 ---
746 // Orders 5 and 6 have noise at low fundamentals: MAX_SAW_ORDER = 6; MAX_SAW_ORDER_NEXTPOW2
// = 8;
747 MAX_SAW_ORDER = 4;
748 MAX_SAW_ORDER_NEXTPOW2 = 8; // par cannot handle the case of 0 elements
749 sawN(N,freq) = saw11 : poly(Nc) : D(Nc-1) : gate(Nc-1)
750 with {
751 Nc = max(1,min(N,MAX_SAW_ORDER));
752 clippedFreq = max(20.0,abs(freq)); // use lf_sawpos(freq) for LFOs (freq < 20 Hz)
753 saw11 = 2*lf_sawpos(clippedFreq) - 1; // zero-mean, amplitude +/- 1
754 poly(1,x) = x;
755 poly(2,x) = x*x;
756 poly(3,x) = x*x*x - x;
757 poly(4,x) = x*x*(x*x - 2.0);
758 poly(5,x) = x*(7.0/3 + x*x*(-10.0/3.0 + x*x));
759 poly(6,x) = x*x*(7.0 + x*x*(-5.0 + x*x));
760 p0n = float(ma.SR)/clippedFreq; // period in samples
761 diff1(x) = (x - x')/(2.0/p0n);
762 diff(N) = seq(n,N,diff1); // N diff1s in series
763 factorial(0) = 1;
764 factorial(i) = i * factorial(i-1);
765 D(0) = _;
766 D(i) = diff(i)/factorial(i+1);
767 gate(N) = *(1@N); // delayed step for blanking startup glitch
768 };
769
770 //-----'(os.)sawNp'-----
771 // Same as '(os.)sawN' but with a controllable waveform phase.
772 //

```

```

773 // #### Usage
774 //
775 // '''
776 // sawNp(N,freq,phase) : _
777 // '''
778 //
779 // where
780 //
781 // * 'N': waveform interpolation polynomial order 1 to 4 (constant integer expression)
782 // * 'freq': frequency in Hz
783 // * 'phase': waveform phase as a fraction of one period (rounded to nearest sample)
784 //
785 // ##### Implementation Notes
786 //
787 // The phase offset is implemented by delaying 'sawN(N,freq)' by
788 // 'round(phase*ma.SR/freq)' samples, for up to 8191 samples.
789 // The minimum sawtooth frequency that can be delayed a whole period
790 // is therefore 'ma.SR/8191', which is well below audibility for normal
791 // audio sampling rates.
792 //
793 //-----
794 declare sawNp author "Julius O. Smith III";
795 declare sawNp license "STK-4.3";
796 // --- sawNp for N = 1 to 4 ---
797 // Phase offset = delay (max 8191 samples is more than one period of audio):
798 sawNp(N,freq,phase) = sawN(N,freq) : @(max(0,min(8191,int(0.5+phase*ma.SR/freq)))));
799
800 //-----'(os.)saw2, (os.)saw3, (os.)saw4'-----
801 // Alias-Suppressed Sawtooth Audio-Frequency Oscillators of order 2, 3, 4.
802 //
803 // #### Usage
804 //
805 // '''
806 // saw2(freq) : _
807 // saw3(freq) : _
808 // saw4(freq) : _
809 // '''
810 //
811 // where
812 //
813 // * 'freq': frequency in Hz
814 //
815 // ##### References
816 // See 'sawN' above.
817 //
818 // ##### Implementation Notes
819 //
820 // Presently, only 'saw2' uses the PTR method, while 'saw3' and 'saw4' use DPW.
821 // This is because PTR has been implemented and tested for the 2nd-order case only.
822 //
823 //-----
824 saw2 = saw2ptr; // "faustlibraries choice"
825 saw3 = sawN(3); // only choice available right now
826 saw4 = sawN(4); // only choice available right now
827
828 //-----'(os.)saw2ptr'-----
829 // Alias-Suppressed Sawtooth Audio-Frequency Oscillator
830 // using Polynomial Transition Regions (PTR) for order 2.
831 //
832 // #### Usage
833 //
834 // '''
835 // saw2ptr(freq) : _
836 // '''
837 //
838 // where
839 //
840 // * 'freq': frequency in Hz

```



```

841 //
842 // ##### Implementation
843 //
844 // Polynomial Transition Regions (PTR) method for aliasing suppression.
845 //
846 // ##### References
847 //
848 // * Kleimola, J.; Valimäki, V., "Reducing Aliasing from Synthetic Audio
849 //   Signals Using Polynomial Transition Regions," in Signal Processing
850 //   Letters, IEEE , vol.19, no.2, pp.67-70, Feb. 2012
851 // * <https://aaltodoc.aalto.fi/bitstream/handle/123456789/7747/publication6.pdf?sequence=9>
852 // * <http://research.spa.aalto.fi/publications/papers/spl-ptr/>
853 //
854 // ##### Notes
855 //
856 // Method PTR may be preferred because it requires less
857 // computation and is stateless which means that the frequency 'freq'
858 // can be modulated arbitrarily fast over time without filtering
859 // artifacts. For this reason, 'saw2' is presently defined as 'saw2ptr'.
860 //
861 //-----
862 declare saw2ptr author "Julius O. Smith III";
863 declare saw2ptr license "STK-4.3";
864 // specialized reimplementation:
865 saw2ptr(freq) = y with { // newer PTR version (stateless - freq can vary at any speed)
866   p0 = float(ma.SR)/float(max(ma.EPSILON,abs(freq))); // period in samples
867   t0 = 1.0/p0; // phase increment
868   p = ((<:(-1)<:_,_)<:_<: selector1,selector2) ~(+t0))!:_;
869   selector1 = select2(<(0)); // for feedback
870   selector2 = select2(<(0), (<:_,(*(1-p0):+(1)):+), _); // for output
871   y = 2*p-1;
872 };
873
874 //-----'(os.)saw2dpw'-----
875 // Alias-Suppressed Sawtooth Audio-Frequency Oscillator
876 // using the Differentiated Polynomial Waveform (DPW) method.
877 //
878 // ##### Usage
879 //
880 // '''
881 // saw2dpw(freq) : _
882 // '''
883 //
884 // where
885 //
886 // * 'freq': frequency in Hz
887 //
888 // This is the original Faust 'saw2' function using the DPW method.
889 // Since 'saw2' is now defined as 'saw2ptr', the DPW version
890 // is now available as 'saw2dwp'.
891 //-----
892 declare saw2dwp author "Julius O. Smith III";
893 declare saw2dwp license "STK-4.3";
894 saw2dwp(freq) = saw1(freq) <: * <: -(mem) : *(0.25*ma.SR/freq);
895
896 //-----'(os.)sawtooth'-----
897 // Alias-suppressed aliasing-suppressed sawtooth oscillator, presently defined as 'saw2'.
898 // 'sawtooth' is a standard Faust function.
899 //
900 // ##### Usage
901 //
902 // '''
903 // sawtooth(freq) : _
904 // '''
905 //
906 // with
907 //
908 // * 'freq': frequency in Hz

```

```

909 //-----
910 sawtooth = saw2; // default choice for sawtooth signal - see also sawN
911
912 //-----'(os.)saw2f2, (os.)saw2f4'-----
913 // Alias-Suppressed Sawtooth Audio-Frequency Oscillator with Order 2 or 4 Droop Correction
    Filtering.
914 //
915 // #### Usage
916 //
917 // ''
918 // saw2f2(freq) : _
919 // saw2f4(freq) : _
920 // ''
921 //
922 // with
923 //
924 // * 'freq': frequency in Hz
925 //
926 // In return for aliasing suppression, there is some attenuation near half the sampling rate
    .
927 // This can be considered as beneficial, or it can be compensated with a high-frequency
    boost.
928 // The boost filter is second-order for 'saw2f2' and fourth-order for 'saw2f4', and both are
    designed
929 // for the DWP case and therefore use 'saw2dpw'.
930 // See Figure 4(b) in the DPW reference for a plot of the slight droop in the DPW case.
931 //-----
932 declare saw2f2 author "Julius O. Smith III";
933 declare saw2f2 license "STK-4.3";
934 // --- Correction-filtered versions of saw2: saw2f2, saw2f4 ----
935 saw2f2 = saw2dpw : cf2 with {
936     cf2 = fi.tf2(1.155704605878911, 0.745184288225518, 0.040305967265900,
937         0.823765146386639, 0.117420665547108);
938 };
939 declare saw2f4 author "Julius O. Smith III";
940 declare saw2f4 license "STK-4.3";
941 saw2f4 = saw2dpw : cf4 with {
942     cf4 = fi.iir((1.155727435125014, 2.285861038554662,
943         1.430915027294021, 0.290713280893317, 0.008306401748854),
944         (2.156834679164532, 1.559532244409321, 0.423036498118354,
945         0.032080681130972));
946 };
947
948 //=====Alias-Suppressed Pulse, Square, and Impulse Trains=====
949 // Alias-Suppressed Pulse, Square and Impulse Trains.
950 //
951 // 'pulsetrainN', 'pulsetrain', 'squareN', 'square', 'imptrainN', 'imptrain',
952 // 'triangleN', 'triangle'
953 //
954 // All are zero-mean and meant to oscillate in the audio frequency range.
955 // Use simpler sample-rounded 'lf_*' versions above for LFOs.
956 //
957 // #### Usage
958 //
959 // ''
960 // pulsetrainN(N,freq,duty) : _
961 // pulsetrain(freq, duty) : _ // = pulsetrainN(2)
962 //
963 // squareN(N,freq) : _
964 // square : _ // = squareN(2)
965 //
966 // imptrainN(N,freq) : _
967 // imptrain : _ // = imptrainN(2)
968 //
969 // triangleN(N,freq) : _
970 // triangle : _ // = triangleN(2)
971 // ''
972 //

```

```

973 // Where:
974 //
975 // * 'N': polynomial order, a constant numerical expression
976 // * 'freq': frequency in Hz
977 //=====
978
979
980 //-----'(os.)impulse'-----
981 // One-time impulse generated when the Faust process is started.
982 // 'impulse' is a standard Faust function.
983 //
984 // #### Usage
985 //
986 // '''
987 // impulse : _
988 // '''
989 //-----
990 impulse = 1-1';
991
992
993 //-----'(os.)pulsetrainN'-----
994 // Alias-suppressed pulse train oscillator.
995 //
996 // #### Usage
997 //
998 // '''
999 // pulsetrainN(N,freq,duty) : _
1000 // '''
1001 //
1002 // Where:
1003 //
1004 // * 'N': order, as a constant numerical expression
1005 // * 'freq': frequency in Hz
1006 // * 'duty': duty cycle between 0 and 1
1007
1008 //-----
1009 pulsetrainN(N,freq,duty) = diffdel(sawN(N,freqC),del) with {
1010 // non-interpolated-delay version: diffdel(x,del) = x - x@int(del+0.5);
1011 // linearly interpolated delay version (sounds good to me):
1012 // diffdel(x,del) = x-x@int(del)*(1-ma.frac(del))-x@(int(del)+1)*ma.frac(del);
1013 // Third-order Lagrange interpolated-delay version (see filters.lib):
1014 // diffdel(x,del) = x - fdelay3(DELPWR2,max(1,min(DELPWR2-2,ddel)));
1015 DELPWR2 = 2048; // Needs to be a power of 2 when fdelay*() used above.
1016 delmax = DELPWR2-1; // arbitrary upper limit on diff delay (duty=0.5)
1017 SRmax = 96000.0; // assumed upper limit on sampling rate
1018 fmin = SRmax / float(2.0*delmax); // 23.4 Hz (audio freqs only)
1019 freqC = max(freq,fmin); // clip frequency at lower limit
1020 period = (float(ma.SR) / freqC); // actual period
1021 ddel = duty * period; // desired delay
1022 del = max(0,min(delmax,ddel));
1023 };
1024
1025
1026 //-----'(os.)pulsetrain'-----
1027 // Alias-suppressed pulse train oscillator. Based on 'pulsetrainN(2)'.
1028 // 'pulsetrain' is a standard Faust function.
1029 //
1030 // #### Usage
1031 //
1032 // '''
1033 // pulsetrain(freq,duty) : _
1034 // '''
1035 //
1036 // Where:
1037 //
1038 // * 'freq': frequency in Hz
1039 // * 'duty': duty cycle between 0 and 1
1040 //-----

```

```

1041 pulsetrain = pulsetrainN(2);
1042
1043
1044 //-----'(os.)squareN'-----
1045 // Alias-suppressed square wave oscillator.
1046 //
1047 // #### Usage
1048 //
1049 // '''
1050 // squareN(N,freq) : _
1051 // '''
1052 //
1053 // Where:
1054 //
1055 // * 'N': order, as a constant numerical expression
1056 // * 'freq': frequency in Hz
1057 //-----
1058 squareN(N,freq) = pulsetrainN(N,freq,0.5);
1059
1060
1061 //-----'(os.)square'-----
1062 // Alias-suppressed square wave oscillator. Based on 'squareN(2)'.
1063 // 'square' is a standard Faust function.
1064 //
1065 // #### Usage
1066 //
1067 // '''
1068 // square(freq) : _
1069 // '''
1070 //
1071 // Where:
1072 //
1073 // * 'freq': frequency in Hz
1074 //-----
1075 square = squareN(2);
1076
1077
1078 //-----'(os.)imptrainN'-----
1079 // Alias-suppressed impulse train generator.
1080 //
1081 // #### Usage
1082 //
1083 // '''
1084 // imptrainN(N,freq) : _
1085 // '''
1086 //
1087 // Where:
1088 //
1089 // * 'N': order, as a constant numerical expression
1090 // * 'freq': frequency in Hz
1091 //-----
1092 imptrainN(N,freq) = impulse + 0.5*ma.diffn(sawN(N,freq));
1093
1094
1095 //-----'(os.)imptrain'-----
1096 // Alias-suppressed impulse train generator. Based on 'imptrainN(2)'.
1097 // 'imptrain' is a standard Faust function.
1098 //
1099 // #### Usage
1100 //
1101 // '''
1102 // imptrain(freq) : _
1103 // '''
1104 //
1105 // Where:
1106 //
1107 // * 'freq': frequency in Hz
1108 //-----

```

```

1109 imptrain = imptrainN(2); // default based on saw2
1110
1111
1112 //-----'(os.)triangleN'-----
1113 // Alias-suppressed triangle wave oscillator.
1114 //
1115 // #### Usage
1116 //
1117 // '''
1118 // triangleN(N,freq) : _
1119 // '''
1120 //
1121 // Where:
1122 //
1123 // * 'N': order, as a constant numerical expression
1124 // * 'freq': frequency in Hz
1125 //-----
1126 triangleN(N,freq) = squareN(N,freq) : fi.pole(p) : *(gain) with {
1127     gain = 4.0*freq/ma.SR; // for aproximate unit peak amplitude
1128     p = 0.999;
1129 };
1130
1131
1132 //-----'(os.)triangle'-----
1133 // Alias-suppressed triangle wave oscillator. Based on 'triangleN(2)'.
1134 // 'triangle' is a standard Faust function.
1135 //
1136 // #### Usage
1137 //
1138 // '''
1139 // triangle(freq) : _
1140 // '''
1141 //
1142 // Where:
1143 //
1144 // * 'freq': frequency in Hz
1145 //-----
1146 triangle = triangleN(2); // default based on saw2
1147
1148
1149 //=====Filter-Based Oscillators=====
1150 // Filter-Based Oscillators.
1151 //
1152 // #### Usage
1153 //
1154 // '''
1155 // osc[b|rq|rs|rc|s](freq), where freq = frequency in Hz.
1156 // '''
1157 //
1158 // #### References
1159 //
1160 // * <http://lac.linuxaudio.org/2012/download/lac12-slides-jos.pdf>
1161 // * <https://ccrma.stanford.edu/~jos/pdf/lac12-paper-jos.pdf>
1162 //=====
1163
1164 //-----'(os.)oscb'-----
1165 // Sinusoidal oscillator based on the biquad.
1166 //
1167 // #### Usage
1168 //
1169 // '''
1170 // oscb(freq) : _
1171 // '''
1172 //
1173 // Where:
1174 //
1175 // * 'freq': frequency in Hz
1176 //-----

```

```

1177 oscb(f) = impulse : fi.tf2(1,0,0,a1,1)
1178 with {
1179     a1 = -2*cos(2*ma.PI*f/ma.SR);
1180 };
1181
1182
1183 //-----'(os.)oscrq'-----
1184 // Sinusoidal (sine and cosine) oscillator based on 2D vector rotation,
1185 // = undamped "coupled-form" resonator
1186 // = lossless 2nd-order normalized ladder filter.
1187 //
1188 // #### Usage
1189 //
1190 // '''
1191 // oscrq(freq) : _,_
1192 // '''
1193 //
1194 // Where:
1195 //
1196 // * 'freq': frequency in Hz
1197 //
1198 // #### Reference
1199 //
1200 // * <https://ccrma.stanford.edu/~jos/pasp/Normalized\_Scattering\_Junctions.html>
1201 //-----
1202 oscrq(f) = impulse : fi.nlf2(f,1); // sine and cosine outputs
1203
1204 //-----'(os.)oscrc'-----
1205 // Sinusoidal (sine) oscillator based on 2D vector rotation,
1206 // = undamped "coupled-form" resonator
1207 // = lossless 2nd-order normalized ladder filter.
1208 //
1209 // #### Usage
1210 //
1211 // '''
1212 // oscrc(freq) : _
1213 // '''
1214 //
1215 // Where:
1216 //
1217 // * 'freq': frequency in Hz
1218 //
1219 // #### Reference
1220 //
1221 // * <https://ccrma.stanford.edu/~jos/pasp/Normalized\_Scattering\_Junctions.html>
1222 //-----
1223 oscrc(f) = impulse : fi.nlf2(f,1) : _,!; // sine
1224
1225 //-----'(os.)oscrc'-----
1226 // Sinusoidal (cosine) oscillator based on 2D vector rotation,
1227 // = undamped "coupled-form" resonator
1228 // = lossless 2nd-order normalized ladder filter.
1229 //
1230 // #### Usage
1231 //
1232 // '''
1233 // oscrc(freq) : _
1234 // '''
1235 //
1236 // Where:
1237 //
1238 // * 'freq': frequency in Hz
1239 //
1240 // #### Reference
1241 //
1242 // * <https://ccrma.stanford.edu/~jos/pasp/Normalized\_Scattering\_Junctions.html>
1243 //-----
1244 oscrc(f) = impulse : fi.nlf2(f,1) : !,_; // cosine

```

```

1245 oscrp(f,p) = oscrq(f) : *(cos(p)), *(sin(p)) : + ; // p=0 for sine, p=PI/2 for cosine, etc.
1246
1247
1248 oscr = oscrs; // default = sine (starts without a pop)
1249
1250 //-----'(os.)oscs'-----
1251 // Sinusoidal oscillator based on the state variable filter
1252 // = undamped "modified-coupled-form" resonator
1253 // = "magic circle" algorithm used in graphics.
1254 //
1255 // #### Usage
1256 //
1257 // '''
1258 // oscs(freq) : _
1259 // '''
1260 //
1261 // Where:
1262 //
1263 // * 'freq': frequency in Hz
1264 //-----
1265 oscs(f) = *(-1) : sint(wn) : sintp(wn,impulse)) ~ _
1266 with {
1267   wn = 2*ma.PI*f/ma.SR; // approximate
1268   // wn = 2*sin(PI*f/SR); // exact
1269   sint(x) = *(x) : + ~ _ ; // frequency-scaled integrator
1270   sintp(x,y) = *(x) : +(y) : + ~ _ ; // same + state input
1271 };
1272
1273 //-----'(os.)quadosc'-----
1274 // Quadrature (cosine and sine) oscillator based on QuadOsc by Martin Vicanek.
1275 //
1276 // #### Usage
1277 //
1278 // '''
1279 // quadosc(freq) : _,_
1280 // '''
1281 //
1282 // where
1283 //
1284 // * 'freq': frequency in Hz
1285 //
1286 // #### Reference
1287 // * <https://vicanek.de/articles/QuadOsc.pdf>
1288 //-----
1289 // Authors:
1290 // Dario Sanfilippo <sanfilippo.dario@gmail.com>
1291 // and Oleg Nesterov (jos ed.)
1292 quadosc(f) = tick ~ (_,_)
1293 with {
1294   k1 = tan(f * ma.PI / ma.SR);
1295   k2 = 2 * k1 / (1 + k1 * k1);
1296   tick(u_0,v_0) = u_1,v_1
1297   with {
1298     tmp = u_0 - k1 * v_0;
1299     v_1 = v_0 + k2 * tmp;
1300     u_1 = tmp - k1 * v_1 : select2(1',1);
1301   };
1302 };
1303
1304 //-----'(os.)sidebands'-----
1305 // Adds harmonics to quad oscillator.
1306 //
1307 // #### Usage
1308 //
1309 // '''
1310 // cos(x),sin(x) : sidebands(vs) : _,_
1311 // '''
1312 //

```

```

1313 // Where:
1314 //
1315 // * 'vs' : list of amplitudes
1316 //
1317 // #### Example test program
1318 //
1319 // '''
1320 //   cos(x),sin(x) : sidebands((10,20,30))
1321 //   '''
1322 //
1323 // outputs:
1324 //
1325 // '''
1326 //   10*cos(x) + 20*cos(2*x) + 30*cos(3*x),
1327 //   10*sin(x) + 20*sin(2*x) + 30*sin(3*x);
1328 //   '''
1329 //
1330 // The following:
1331 //
1332 // '''
1333 //   process = os.quadosc(F) : sidebands((10,20,30))
1334 //   '''
1335 //
1336 // is (modulo floating point issues) the same as:
1337 //
1338 // '''
1339 //   c = os.quadosc : _,!;
1340 //   s = os.quadosc : !,_;
1341 //   process =
1342 //       10*c(F) + 20*c(2*F) + 30*c(F),
1343 //       10*s(F) + 20*s(2*F) + 30*s(F);
1344 //   '''
1345 //
1346 // but much more efficient.
1347 //
1348 // #### Implementation Notes
1349 //
1350 // This is based on the trivial trigonometric identities:
1351 //
1352 // '''
1353 //   cos((n + 1) x) = 2 cos(x) cos(n x) - cos((n - 1) x)
1354 //   sin((n + 1) x) = 2 cos(x) sin(n x) - sin((n - 1) x)
1355 //   '''
1356 //
1357 // Note that the calculation of the cosine/sine parts do not depend
1358 // on each other, so if you only need the sine part you can do:
1359 //
1360 // '''
1361 //   process = os.quadosc(F) : sidebands(vs) : !,_;
1362 //   '''
1363 //
1364 // and the compiler will discard the half of the calculations.
1365 //-----
1366 sidebands(vs, c0,s0)
1367   = c0*vn(0),s0*vn(0), 1,c0, 0,s0
1368   : seq(n, outputs(vs)-1, add(vn(n+1)))
1369   : _,_, !,!, !,!
1370 with {
1371   // ba.take(n+1, vs)
1372   vn(n) = vs : route(outputs(vs),1, n+1,1);
1373
1374   add(vn, co,so, cn_2,cn_1, sn_2,sn_1) =
1375   co+cn*vn, so+sn*vn, cn_1,cn, sn_1,sn
1376   with {
1377     cn = 2*c0*cn_1 - cn_2;
1378     sn = 2*c0*sn_1 - sn_2;
1379   };
1380 };

```



```

1381 //-----'(os.)sidebands_list'-----
1382 // Creates the list of complex harmonics from quad oscillator.
1383 //
1384 // Similar to 'sidebands' but doesn't sum the harmonics, so it is more
1385 // generic but less convenient for immediate usage.
1386 //
1387 // #### Usage
1388 //
1389 // '''
1390 //     cos(x),sin(x) : sidebands_list(N) : si.bus(2*N)
1391 // '''
1392 //
1393 // Where:
1394 //
1395 // * 'N' : number of harmonics, compile time constant > 1
1396 //
1397 // #### Example test program
1398 //
1399 // '''
1400 //     cos(x),sin(x) : sidebands_list(3)
1401 // '''
1402 //
1403 // outputs:
1404 //
1405 // '''
1406 //     cos(x),sin(x), cos(2*x),sin(2*x), cos(3*x),sin(3*x);
1407 // '''
1408 //
1409 // The following:
1410 //
1411 // '''
1412 //     process = os.quadosc(F) : sidebands_list(3)
1413 // '''
1414 //
1415 // is (modulo floating point issues) the same as:
1416 //
1417 // '''
1418 //     process = os.quadosc(F), os.quadosc(2*F), os.quadosc(3*F);
1419 // '''
1420 //
1421 // but much more efficient.
1422 //-----
1423 sidebands_list(N, c0,s0)
1424     = c0,s0, 1,c0, 0,s0
1425     : seq(n, N-1, si.bus(2*(n+1)), add)
1426     : si.bus(2*N), !,!, !,!
1427 with {
1428     add(cn_2,cn_1, sn_2,sn_1) =
1429         cn,sn, cn_1,cn, sn_1,sn
1430     with {
1431         cn = 2*c0*cn_1 - cn_2;
1432         sn = 2*c0*sn_1 - sn_2;
1433     };
1434 };
1435
1436 //===== Waveguide-Resonator-Based Oscillators =====
1437 // Sinusoidal oscillator based on the waveguide resonator 'wgr'.
1438 //=====
1439
1440 //-----'(os.)oscwc'-----
1441 // Sinusoidal oscillator based on the waveguide resonator 'wgr'. Unit-amplitude
1442 // cosine oscillator.
1443 //
1444 // #### Usage
1445 //
1446 // '''
1447 //     oscwc(freq) : _
1448

```

```

1449 // ''
1450 //
1451 // Where:
1452 //
1453 // * 'freq': frequency in Hz
1454 //
1455 // #### Reference
1456 //
1457 // * <https://ccrma.stanford.edu/~jos/pasp/Digital\_Waveguide\_Oscillator.html>
1458 //-----
1459 oscwc(fr) = impulse : fi.wgr(fr,1) : _,!; // cosine (cheapest at 1 mpy/sample)
1460
1461 //-----'(os.)oscws'-----
1462 // Sinusoidal oscillator based on the waveguide resonator 'wgr'. Unit-amplitude
1463 // sine oscillator.
1464 //
1465 // #### Usage
1466 //
1467 // ''
1468 // oscws(freq) : _
1469 // ''
1470 //
1471 // Where:
1472 //
1473 // * 'freq': frequency in Hz
1474 //
1475 // #### Reference
1476 //
1477 // * <https://ccrma.stanford.edu/~jos/pasp/Digital\_Waveguide\_Oscillator.html>
1478 //-----
1479 oscws(fr) = impulse : fi.wgr(fr,1) : !,_, // sine (needs a 2nd scaling mpy)
1480
1481 //-----'(os.)oscq'-----
1482 // Sinusoidal oscillator based on the waveguide resonator 'wgr'.
1483 // Unit-amplitude cosine and sine (quadrature) oscillator.
1484 //
1485 // #### Usage
1486 //
1487 // ''
1488 // oscq(freq) : _,_
1489 // ''
1490 //
1491 // Where:
1492 //
1493 // * 'freq': frequency in Hz
1494 //
1495 // #### Reference
1496 //
1497 // * <https://ccrma.stanford.edu/~jos/pasp/Digital\_Waveguide\_Oscillator.html>
1498 //-----
1499 oscq(fr) = impulse : fi.wgr(fr,1); // phase quadrature outputs
1500
1501 //-----'(os.)oscw'-----
1502 // Sinusoidal oscillator based on the waveguide resonator 'wgr'.
1503 // Unit-amplitude cosine oscillator (default).
1504 //
1505 // #### Usage
1506 //
1507 // ''
1508 // oscw(freq) : _
1509 // ''
1510 //
1511 // Where:
1512 //
1513 // * 'freq': frequency in Hz
1514 //
1515 // #### Reference
1516 //

```

```

1517 // * <https://ccrma.stanford.edu/~jos/pasp/Digital_Waveguide_Oscillator.html>
1518 //-----
1519 oscw = oscwc;
1520
1521 // end jos section
1522 //#####
1523 /*****
1524 FAUST library file, further contributions section
1525
1526 All contributions below should indicate both the contributor and terms
1527 of license. If no such indication is found, "git blame" will say who
1528 last edited each line, and that person can be emailed to inquire about
1529 license disposition, if their license choice is not already indicated
1530 elsewhere among the libraries. It is expected that all software will be
1531 released under LGPL, STK-4.3, MIT, BSD, or a similar FOSS license.
1532 *****/
1533
1534 //===== Casio CZ Oscillators =====
1535 // Oscillators that mimic some of the Casio CZ oscillators.
1536 //
1537 // There are two sets:
1538 //
1539 // * a set with an index parameter
1540 //
1541 // * a set with a res parameter
1542 //
1543 // The "index oscillators" outputs a sine wave at index=0 and gets brighter with a higher
1544 // index.
1545 // There are two versions of the "index oscillators":
1546 //
1547 // * with P appended to the name: is phase aligned with 'fund:sin'
1548 //
1549 // * without P appended to the name: has the phase of the original CZ oscillators
1550 //
1551 // The "res oscillators" have a resonant frequency.
1552 // "res" is the frequency of resonance as a factor of the fundamental pitch.
1553 //
1554 // For the 'fund' waveform, use a low-frequency oscillator without anti-aliasing such as 'os.
1555 // lf_saw'.
1556 //=====
1557 //-----'(os.)CZsaw'-----
1558 // Oscillator that mimics the Casio CZ saw oscillator.
1559 // 'CZsaw' is a standard Faust function.
1560 //
1561 // ### Usage
1562 //
1563 // CZsaw(fund,index) : _
1564 //
1565 //
1566 // Where:
1567 //
1568 // * 'fund': a saw-tooth waveform between 0 and 1 that the oscillator slaves to
1569 // * 'index': the brightness of the oscillator, 0 to 1. 0 = sine-wave, 1 = saw-wave
1570 //-----
1571 declare CZsaw author "Bart Brouns";
1572 declare CZsaw licence "STK-4.3";
1573
1574 // CZ oscillators by Mike Moser-Booth:
1575 // <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
1576 // Ported from pd to Faust by Bart Brouns
1577
1578 CZsaw(fund, index) = CZ.sawChooseP(fund, index, 0);
1579
1580 //-----'(os.)CZsawP'-----
1581 // Oscillator that mimics the Casio CZ saw oscillator,
1582 // with it's phase aligned to 'fund:sin'.

```

```

1583 // 'CZsawP' is a standard Faust function.
1584 //
1585 // #### Usage
1586 //
1587 // '
1588 // CZsawP(fund,index) : _
1589 // '
1590 //
1591 // Where:
1592 //
1593 // * 'fund': a saw-tooth waveform between 0 and 1 that the oscillator slaves to
1594 // * 'index': the brightness of the oscillator, 0 to 1. 0 = sine-wave, 1 = saw-wave
1595 //-----
1596 declare CZsawP author "Bart Brouns";
1597 declare CZsawP licence "STK-4.3";
1598
1599 // CZ oscillators by Mike Moser-Booth:
1600 // <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
1601 // Ported from pd to Faust by Bart Brouns
1602
1603 CZsawP(fund, index) = CZ.sawChooseP(fund, index, 1);
1604
1605 //-----'(os.)CZsquare'-----
1606 // Oscillator that mimics the Casio CZ square oscillator
1607 // 'CZsquare' is a standard Faust function.
1608 //
1609 // #### Usage
1610 //
1611 // '
1612 // CZsquare(fund,index) : _
1613 // '
1614 //
1615 // Where:
1616 //
1617 // * 'fund': a saw-tooth waveform between 0 and 1 that the oscillator slaves to
1618 // * 'index': the brightness of the oscillator, 0 to 1. 0 = sine-wave, 1 = square-wave
1619 //-----
1620 declare CZsquare author "Bart Brouns";
1621 declare CZsquare licence "STK-4.3";
1622
1623 // CZ oscillators by Mike Moser-Booth:
1624 // <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
1625 // Ported from pd to Faust by Bart Brouns
1626
1627 CZsquare(fund, index) = CZ.squareChooseP(fund, index, 0);
1628
1629 //-----'(os.)CZsquareP'-----
1630 // Oscillator that mimics the Casio CZ square oscillator,
1631 // with it's phase aligned to 'fund:sin'.
1632 // 'CZsquareP' is a standard Faust function.
1633 //
1634 // #### Usage
1635 //
1636 // '
1637 // CZsquareP(fund,index) : _
1638 // '
1639 //
1640 // Where:
1641 //
1642 // * 'fund': a saw-tooth waveform between 0 and 1 that the oscillator slaves to
1643 // * 'index': the brightness of the oscillator, 0 to 1. 0 = sine-wave, 1 = square-wave
1644 //-----
1645 declare CZsquareP author "Bart Brouns";
1646 declare CZsquareP licence "STK-4.3";
1647
1648 // CZ oscillators by Mike Moser-Booth:
1649 // <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
1650 // Ported from pd to Faust by Bart Brouns

```

```

1651 CZsquareP(fund, index) = CZ.squareChooseP(fund, index, 1);
1652
1653
1654 //-----'(os.)CZpulse'-----
1655 // Oscillator that mimics the Casio CZ pulse oscillator.
1656 // 'CZpulse' is a standard Faust function.
1657 //
1658 // #### Usage
1659 //
1660 // '''
1661 // CZpulse(fund,index) : _
1662 // '''
1663 //
1664 // Where:
1665 //
1666 // * 'fund': a saw-tooth waveform between 0 and 1 that the oscillator slaves to
1667 // * 'index': the brightness of the oscillator, 0 gives a sine-wave, 1 is closer to a pulse
1668 //-----
1669 declare CZpulse author "Bart Brouns";
1670 declare CZpulse licence "STK-4.3";
1671
1672 // CZ oscillators by Mike Moser-Booth:
1673 // <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
1674 // Ported from pd to Faust by Bart Brouns
1675
1676 CZpulse(fund, index) = CZ.pulseChooseP(fund, index, 0);
1677
1678 //-----'(os.)CZpulseP'-----
1679 // Oscillator that mimics the Casio CZ pulse oscillator,
1680 // with it's phase aligned to 'fund:sin'.
1681 // 'CZpulseP' is a standard Faust function.
1682 //
1683 // #### Usage
1684 //
1685 // '''
1686 // CZpulseP(fund,index) : _
1687 // '''
1688 //
1689 // Where:
1690 //
1691 // * 'fund': a saw-tooth waveform between 0 and 1 that the oscillator slaves to
1692 // * 'index': the brightness of the oscillator, 0 gives a sine-wave, 1 is closer to a pulse
1693 //-----
1694 declare CZpulseP author "Bart Brouns";
1695 declare CZpulseP licence "STK-4.3";
1696
1697 // CZ oscillators by Mike Moser-Booth:
1698 // <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
1699 // Ported from pd to Faust by Bart Brouns
1700
1701 CZpulseP(fund, index) = CZ.pulseChooseP(fund, index, 1);
1702
1703 //-----'(os.)CZsinePulse'-----
1704 // Oscillator that mimics the Casio CZ sine/pulse oscillator.
1705 // 'CZsinePulse' is a standard Faust function.
1706 //
1707 // #### Usage
1708 //
1709 // '''
1710 // CZsinePulse(fund,index) : _
1711 // '''
1712 //
1713 // Where:
1714 //
1715 // * 'fund': a saw-tooth waveform between 0 and 1 that the oscillator slaves to
1716 // * 'index': the brightness of the oscillator, 0 gives a sine-wave, 1 is a sine minus a
1717 // pulse
1718 //-----

```

```

1718 declare CZsinePulse author "Bart Brouns";
1719 declare CZsinePulse licence "STK-4.3";
1720
1721 // CZ oscillators by Mike Moser-Booth:
1722 // <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
1723 // Ported from pd to Faust by Bart Brouns
1724
1725 CZsinePulse(fund, index) = CZ.sinePulseChooseP(fund, index, 0);
1726
1727 //-----'(os.)CZsinePulseP'-----
1728 // Oscillator that mimics the Casio CZ sine/pulse oscillator,
1729 // with it's phase aligned to 'fund:sin'.
1730 // 'CZsinePulseP' is a standard Faust function.
1731 //
1732 // #### Usage
1733 //
1734 // ' ' '
1735 // CZsinePulseP(fund,index) : _
1736 // ' ' '
1737 //
1738 // Where:
1739 //
1740 // * 'fund': a saw-tooth waveform between 0 and 1 that the oscillator slaves to
1741 // * 'index': the brightness of the oscillator, 0 gives a sine-wave, 1 is a sine minus a
1742 // pulse
1743 //-----
1744 declare CZsinePulseP author "Bart Brouns";
1745 declare CZsinePulseP licence "STK-4.3";
1746
1747 // CZ oscillators by Mike Moser-Booth:
1748 // <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
1749 // Ported from pd to Faust by Bart Brouns
1750
1751 CZsinePulseP(fund, index) = CZ.sinePulseChooseP(fund, index, 1);
1752
1753 //-----'(os.)CZhalfSine'-----
1754 // Oscillator that mimics the Casio CZ half sine oscillator.
1755 // 'CZhalfSine' is a standard Faust function.
1756 //
1757 // #### Usage
1758 //
1759 // ' ' '
1760 // CZhalfSine(fund,index) : _
1761 // ' ' '
1762 //
1763 // Where:
1764 //
1765 // * 'fund': a saw-tooth waveform between 0 and 1 that the oscillator slaves to
1766 // * 'index': the brightness of the oscillator, 0 gives a sine-wave, 1 is somewhere between a
1767 // saw and a square
1768 //-----
1769 declare CZhalfSine author "Bart Brouns";
1770 declare CZhalfSine licence "STK-4.3";
1771
1772 // CZ oscillators by Mike Moser-Booth:
1773 // <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
1774 // Ported from pd to Faust by Bart Brouns
1775
1776 CZhalfSine(fund, index) = CZ.halfSineChooseP(fund, index, 0);
1777
1778 //-----'(os.)CZhalfSineP'-----
1779 // Oscillator that mimics the Casio CZ half sine oscillator,
1780 // with it's phase aligned to 'fund:sin'.
1781 // 'CZhalfSineP' is a standard Faust function.
1782 //
1783 // #### Usage
1784 //
1785 // ' ' '

```

```

1784 // CZhalfSineP(fund,index) : _
1785 // ''
1786 //
1787 // Where:
1788 //
1789 // * 'fund': a saw-tooth waveform between 0 and 1 that the oscillator slaves to
1790 // * 'index': the brightness of the oscillator, 0 gives a sine-wave, 1 is somewhere between a
           saw and a square
1791 //-----
1792 declare CZhalfSineP author "Bart Brouns";
1793 declare CZhalfSineP licence "STK-4.3";
1794
1795 // CZ oscillators by Mike Moser-Booth:
1796 // <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
1797 // Ported from pd to Faust by Bart Brouns
1798
1799 CZhalfSineP(fund, index) = CZ.halfSineChooseP(fund, index, 1);
1800
1801 //-----'(os.)CZresSaw'-----
1802 // Oscillator that mimics the Casio CZ resonant sawtooth oscillator.
1803 // 'CZresSaw' is a standard Faust function.
1804 //
1805 // #### Usage
1806 //
1807 // ''
1808 // CZresSaw(fund,res) : _
1809 // ''
1810 //
1811 // Where:
1812 //
1813 // * 'fund': a saw-tooth waveform between 0 and 1 that the oscillator slaves to
1814 // * 'res': the frequency of resonance as a factor of the fundamental pitch.
1815 //-----
1816 declare CZresSaw author "Bart Brouns";
1817 declare CZresSaw licence "STK-4.3";
1818
1819 // CZ oscillators by Mike Moser-Booth:
1820 // <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
1821 // Ported from pd to Faust by Bart Brouns
1822
1823 CZresSaw(fund,res) = CZ.resSaw(fund,res);
1824
1825 //-----'(os.)CZresTriangle'-----
1826 // Oscillator that mimics the Casio CZ resonant triangle oscillator.
1827 // 'CZresTriangle' is a standard Faust function.
1828 //
1829 // #### Usage
1830 //
1831 // ''
1832 // CZresTriangle(fund,res) : _
1833 // ''
1834 //
1835 // Where:
1836 //
1837 // * 'fund': a saw-tooth waveform between 0 and 1 that the oscillator slaves to
1838 // * 'res': the frequency of resonance as a factor of the fundamental pitch.
1839 //-----
1840 declare CZresTriangle author "Bart Brouns";
1841 declare CZresTriangle licence "STK-4.3";
1842
1843 // CZ oscillators by Mike Moser-Booth:
1844 // <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
1845 // Ported from pd to Faust by Bart Brouns
1846
1847 CZresTriangle(fund,res) = CZ.resTriangle(fund,res);
1848
1849 //-----'(os.)CZresTrap'-----
1850 // Oscillator that mimics the Casio CZ resonant trapeze oscillator

```

```

1851 // 'CZresTrap' is a standard Faust function.
1852 //
1853 // ### Usage
1854 //
1855 // '
1856 // CZresTrap(fund,res) : _
1857 // '
1858 //
1859 // Where:
1860 //
1861 // * 'fund': a saw-tooth waveform between 0 and 1 that the oscillator slaves to
1862 // * 'res': the frequency of resonance as a factor of the fundamental pitch.
1863 //-----
1864 declare CZresTrap author "Bart Brouns";
1865 declare CZresTrap licence "STK-4.3";
1866
1867 // CZ oscillators by Mike Moser-Booth:
1868 // <https://forum.pdpatchrepo.info/topic/5992/casio-cz-oscillators>
1869 // Ported from pd to Faust by Bart Brouns
1870
1871 CZresTrap(fund, res) = CZ.resTrap(fund, res);
1872
1873 CZ = environment {
1874
1875     saw(fund, index) = sawChooseP(fund, index, 0);
1876     sawP(fund, index) = sawChooseP(fund, index, 1);
1877     sawChooseP(fund, index, p) =
1878         (((FUND(fund,align,p)*((.5-INDEX)/INDEX)),(-1*FUND(fund,align,p)+1)*((.5-INDEX)/(1-
1879             INDEX)))):min+FUND(fund,align,p))*2*ma.PI:cos
1880
1881     with {
1882         INDEX = (.5-(index*.5)):max(0.01):min(0.5);
1883         align = si.interpolate(index, 0.75, 0.5);
1884     };
1885
1886     square(fund, index) = squareChooseP(fund, index, 0);
1887     squareP(fund, index) = squareChooseP(fund, index, 1);
1888     squareChooseP(fund, index, p) = (FUND(fund,align,p)>=0.5), (ma.decimal((FUND(fund,align,p
1889         )*2)+1):_-min(_,-1*_+1)*((INDEX)/(1-INDEX)))) :+ *ma.PI:cos
1890
1891     with {
1892         INDEX = (index:pow(0.25)):max(0):min(1);
1893         align = si.interpolate(INDEX, -0.25, 0);
1894     };
1895
1896     pulse(fund, index) = pulseChooseP(fund, index, 0);
1897     pulseP(fund, index) = pulseChooseP(fund, index, 1);
1898     pulseChooseP(fund, index, p) = ((FUND(fund,align,p)-min(FUND(fund,align,p),(-1*FUND(fund
1899         ,align,p)+1)*(INDEX/(1-INDEX)))))*2*ma.PI:cos
1900
1901     with {
1902         INDEX = index:min(0.99):max(0);
1903         align = si.interpolate(index, -0.25, 0.0);
1904     };
1905
1906     sinePulse(fund, index) = sinePulseChooseP(fund, index, 0);
1907     sinePulseP(fund, index) = sinePulseChooseP(fund, index, 1);
1908     sinePulseChooseP(fund, index, p) = (min(FUND(fund,align,p)*((0.5-INDEX)/INDEX),(-1*FUND(
1909         fund,align,p)+1)*((.5-INDEX)/(1-INDEX)))+FUND(fund,align,p))*4*ma.PI:cos
1910
1911     with {
1912         INDEX = ((index*-0.49)+0.5);
1913         align = si.interpolate(index, -0.125, -0.25);
1914     };
1915
1916     halfSine(fund, index) = halfSineChooseP(fund, index, 0);
1917     halfSineP(fund, index) = halfSineChooseP(fund, index, 1);
1918     halfSineChooseP(fund, index, p) = (select2(FUND(fund,align,p)<.5, .5*(FUND(fund,align,p)
1919         -.5)/INDEX+.5, FUND(fund,align,p)):min(1))*2*ma.PI:cos
1920
1921     with {
1922         INDEX = (.5-(index*0.5)):min(.5):max(.01);
1923         align = si.interpolate(index:min(0.975), -0.25, -0.5);
1924     };
1925

```



```

1914 };
1915
1916 FUND =
1917   case {
1918     (fund,align,0) => fund;
1919     (fund,align,1) => (fund+align) : ma.frac; // align phase with fund
1920   };
1921 resSaw(fund,res) = (((-1*(1-fund))*((cos((ma.decimal((max(1,res)*fund)+1))*2*ma.PI)*-.5)
1922   +.5))*2)+1;
1923 resTriangle(fund,res) = select2(fund<.5, 2-(fund*2), fund*2)*INDEX*2-1
1924 with {
1925   INDEX = ((fund*(res:max(1)))+1:ma.decimal)*2*ma.PI*cos*.5+.5;
1926 };
1927 resTrap(fund, res) = (((1-fund)*2):min(1)*sin(ma.decimal(fund*(res:max(1)))*2*ma.PI));
1928 };
1929 //=====PolyBLEP-Based Oscillators=====
1930
1931 //-----'(os.)polyblep'-----
1932 // PolyBLEP residual function, used for smoothing steps in the audio signal.
1933 //
1934 // #### Usage
1935 //
1936 // '''
1937 // polyblep(Q,phase) : _
1938 // '''
1939 //
1940 // Where:
1941 //
1942 // * 'Q': smoothing factor between 0 and 0.5. Determines how far from the ends of the phase
1943 // * 'phase': normalised phase (between 0 and 1)
1944 //-----
1945 declare polyblep author "Jacek Wiecezorek";
1946
1947 polyblep(Q, phase) = (0, L(phase / Q), R((phase - 1) / Q)) : select3(sel)
1948 with {
1949   sel = (phase < Q) + 2*(phase > 1 - Q);
1950   L(x) = 2*x - x*x - 1; // Used near the left end of the interval
1951   R(x) = 2*x + x*x + 1; // Used near the right end of the interval
1952 };
1953
1954 //-----'(os.)polyblep_saw'-----
1955 // Sawtooth oscillator with suppressed aliasing (using 'polyblep').
1956 //
1957 // #### Usage
1958 //
1959 // '''
1960 // polyblep_saw(freq) : _
1961 // '''
1962 //
1963 // Where:
1964 //
1965 // * 'freq': frequency in Hz
1966 //-----
1967 declare polyblep_saw author "Jacek Wiecezorek";
1968
1969 polyblep_saw(freq) = naive - polyblep(Q , phase)
1970 with {
1971   phase = phasor(1, freq);
1972   naive = 2 * phase - 1;
1973   Q = freq / ma.SR;
1974 };
1975
1976 //-----'(os.)polyblep_square'-----
1977 // Square wave oscillator with suppressed aliasing (using 'polyblep').
1978 //
1979 // #### Usage

```

```

1980 //
1981 // '''
1982 // polyblep_square(freq) : _
1983 // '''
1984 //
1985 // Where:
1986 //
1987 // * 'freq': frequency in Hz
1988 //-----
1989 declare polyblep_square author "Jacek Wiecezorek";
1990
1991 polyblep_square(freq) = naive - polyblep(Q, phase) + polyblep(Q, ma.modulo(phase + 0.5, 1))
1992 with {
1993     phase = phasor(1, freq);
1994     naive = 2 * (phase * 2 : int) - 1;
1995     Q = freq / ma.SR;
1996 };
1997
1998 //-----'(os.)polyblep_triangle'-----
1999 // Triangle wave oscillator with suppressed aliasing (using 'polyblep').
2000 //
2001 // ### Usage
2002 //
2003 // '''
2004 // polyblep_triangle(freq) : _
2005 // '''
2006 //
2007 // Where:
2008 //
2009 // * 'freq': frequency in Hz
2010 //-----
2011 declare polyblep_triangle author "Jacek Wiecezorek";
2012
2013 polyblep_triangle(freq) = polyblep_square(freq) : fi.pole(0.999) : *(4 * freq / ma.SR);
2014
2015 // end further contributions section
2016 //#####

```

Listing 4: maths.lib

```

1 //##### maths.lib #####
2 // Mathematic library for Faust. Its official prefix is 'ma'.
3 //
4 // ### References
5 // * <https://github.com/grame-cncm/faustlibraries/blob/master/maths.lib>
6 //#####
7 // Some functions are implemented as Faust foreign functions of 'math.h' functions
8 // that are not part of Faust's primitives. Defines also various constants and several
9 // utilities.
10 //#####
11
12 // ## History
13 // * 06/13/2016 [RM] normalizing and integrating to new libraries
14 // * 07/08/2015 [Y0] documentation comments
15 // * 20/06/2014 [SL] added FTZ function
16 // * 22/06/2013 [Y0] added float/double/quad variants of some foreign functions
17 // * 28/06/2005 [Y0] postfix functions with 'f' to force float version instead of double
18 // * 28/06/2005 [Y0] removed 'modf' because it requires a pointer as argument
19
20 /*****
21 *****/
22 FAUST library file
23 Copyright (C) 2003-2016 GRAME, Centre National de Creation Musicale
24 -----

```

```

25 | This program is free software; you can redistribute it and/or modify
26 | it under the terms of the GNU Lesser General Public License as
27 | published by the Free Software Foundation; either version 2.1 of the
28 | License, or (at your option) any later version.
29 |
30 | This program is distributed in the hope that it will be useful,
31 | but WITHOUT ANY WARRANTY; without even the implied warranty of
32 | MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
33 | GNU Lesser General Public License for more details.
34 |
35 | You should have received a copy of the GNU Lesser General Public
36 | License along with the GNU C Library; if not, write to the Free
37 | Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
38 | 02111-1307 USA.
39 |
40 | EXCEPTION TO THE LGPL LICENSE : As a special exception, you may create a
41 | larger FAUST program which directly or indirectly imports this library
42 | file and still distribute the compiled code generated by the FAUST
43 | compiler, or a modified version of this compiled code, under your own
44 | copyright and license. This EXCEPTION TO THE LGPL LICENSE explicitly
45 | grants you the right to freely choose the license for the resulting
46 | compiled code. In particular the resulting compiled code has no obligation
47 | to be LGPL or GPL. For example you are free to choose a commercial or
48 | closed source license or any other license if you decide so.
49 | *****
50 | *****/
51 |
52 | // This library contains platform specific constants
53 | pl = library("platform.lib");
54 | ma = library("maths.lib"); // for compatible copy/paste out of this file
55 |
56 | declare name "Faust Math Library";
57 | declare version "2.8.0";
58 | declare author "GRAMME";
59 | declare copyright "GRAMME";
60 | declare license "LGPL with exception";
61 |
62 | //=====Functions Reference=====
63 | //=====
64 |
65 |
66 | //-----'(ma.)SR'-----
67 | // Current sampling rate given at init time. Constant during program execution.
68 | //
69 | // #### Usage
70 | //
71 | // ' ' '
72 | // SR : _
73 | // ' ' '
74 | //-----
75 | SR = pl.SR;
76 |
77 | //-----'(ma.)T'-----
78 | // Current sample duration in seconds computed from the sampling rate given at init time.
79 | // Constant during program execution.
80 | //
81 | // #### Usage
82 | //
83 | // ' ' '
84 | // T : _
85 | // ' ' '
86 | //-----
87 | T = 1.0 / SR;
88 |
89 | //-----'(ma.)BS'-----
90 | // Current block-size. Can change during the execution at each block.
91 | //
92 | // #### Usage

```

```

92 //
93 // '''
94 // BS : _
95 // '''
96 //-----
97 BS = pl.BS;
98
99
100 //-----'(ma.)PI'-----
101 // Constant PI in double precision.
102 //
103 // #### Usage
104 //
105 // '''
106 // PI : _
107 // '''
108 //-----
109 PI = 3.14159265358979323846;
110
111
112 //-----'(ma.)deg2rad'-----
113 // Convert degrees to radians.
114 //
115 // #### Usage
116 //
117 // '''
118 // 45. : deg2rad
119 // '''
120 //-----
121 deg2rad = _ * PI / 180.;
122
123
124 //-----'(ma.)rad2deg'-----
125 // Convert radians to degrees.
126 //
127 // #### Usage
128 //
129 // '''
130 // ma.PI : rad2deg
131 // '''
132 //-----
133 rad2deg = _ * 180. / PI;
134
135
136 //-----'(ma.)E'-----
137 // Constant e in double precision.
138 //
139 // #### Usage
140 //
141 // '''
142 // E : _
143 // '''
144 //-----
145 E = 2.71828182845904523536;
146
147
148 //-----'(ma.)EPSILON'-----
149 // Constant EPSILON available in simple/double/quad precision,
150 // as defined in the [floating-point standard](https://en.wikipedia.org/wiki/IEEE_754)
151 // and [machine epsilon](https://en.wikipedia.org/wiki/Machine_epsilon),
152 // that is smallest positive number such that '1.0 + EPSILON != 1.0'.
153 //
154 // #### Usage
155 //
156 // '''
157 // EPSILON : _
158 // '''
159 //-----

```

```

160 singleprecision EPSILON = 1.192092896e-07;
161 doubleprecision EPSILON = 2.2204460492503131e-016;
162 quadprecision EPSILON = 1.084202172485504434007452e-019;
163 fixedpointprecision EPSILON = 2.2204460492503131e-016;
164
165
166 //-----'(ma.)MIN'-----
167 // Constant MIN available in simple/double/quad precision (minimal positive value).
168 //
169 // #### Usage
170 //
171 // '
172 // MIN : _
173 // '
174 //-----
175 singleprecision MIN = 1.175494351e-38;
176 doubleprecision MIN = 2.2250738585072014e-308;
177 quadprecision MIN = 2.2250738585072014e-308;
178 fixedpointprecision MIN = 2.2250738585072014e-308;
179
180
181 //-----'(ma.)MAX'-----
182 // Constant MAX available in simple/double/quad precision (maximal positive value).
183 //
184 // #### Usage
185 //
186 // '
187 // MAX : _
188 // '
189 //-----
190 singleprecision MAX = 3.402823466e+38;
191 doubleprecision MAX = 1.7976931348623158e+308;
192 quadprecision MAX = 1.7976931348623158e+308;
193 fixedpointprecision MAX = 1.7976931348623158e+308;
194
195 // Obsolete, kept for compatibility reasons
196 INFINITY = MAX;
197
198 //-----'(ma.)FTZ'-----
199 // Flush to zero: force samples under the "maximum subnormal number"
200 // to be zero. Usually not needed in C++ because the architecture
201 // file take care of this, but can be useful in JavaScript for instance.
202 //
203 // #### Usage
204 //
205 // '
206 // _ : FTZ : _
207 // '
208 //
209 // #### Reference
210 //
211 // <http://docs.oracle.com/cd/E19957-01/806-3568/ncg_math.html>
212 //-----
213 FTZ(x) = x * (abs(x) > MIN);
214
215
216 //-----'(ma.)copysign'-----
217 // Changes the sign of x (first input) to that of y (second input).
218 //
219 // #### Usage
220 //
221 // '
222 // _ : copysign : _
223 // '
224 //-----
225 copysign = ffunction(float copysignf|copysign|copysignl (float, float), <math.h>,"");
226
227

```

```

228 //-----'(ma.)neg'-----
229 // Invert the sign (-x) of a signal.
230 //
231 // #### Usage
232 //
233 // '''
234 // _ : neg : _
235 // '''
236 //-----
237 neg(x) = -x;
238
239
240 //-----'(ma.)not'-----
241 // Bitwise 'not' implemented with [xor](https://faustdoc.grame.fr/manual/syntax/#xor-
    primitive) as 'not(x) = x xor -1;'.
242 // So working regardless of the size of the integer, assuming negative numbers in two's
    complement.
243 //
244 // #### Usage
245 //
246 // '''
247 // _ : not : _
248 // '''
249 //-----
250 not(x) = x xor -1;
251
252
253 //-----'(ma.)sub(x,y)'-----
254 // Subtract 'x' and 'y'.
255 //
256 // #### Usage
257 //
258 // '''
259 // _ , _ : sub : _
260 // '''
261 //-----
262 sub(x,y) = y-x;
263
264
265 //-----'(ma.)inv'-----
266 // Compute the inverse (1/x) of the input signal.
267 //
268 // #### Usage
269 //
270 // '''
271 // _ : inv : _
272 // '''
273 //-----
274 inv(x) = 1/x;
275
276
277 //-----'(ma.)cbrt'-----
278 // Computes the cube root of of the input signal.
279 //
280 // #### Usage
281 //
282 // '''
283 // _ : cbrt : _
284 // '''
285 //-----
286 cbrt = ffunction(float cbrtf|cbrt|cbrtl (float), <math.h>,"");
287
288
289 //-----'(ma.)hypot'-----
290 // Computes the euclidian distance of the two input signals
291 // sqrt(x*x+y*y) without undue overflow or underflow.
292 //
293 // #### Usage

```

```

294 //
295 // '''
296 // _ : hypot : _
297 // '''
298 //-----
299 hypot = ffunction(float hypotf|hypot|hypotl (float, float), <math.h>,"");
300
301
302 //-----'(ma.)ldexp'-----
303 // Takes two input signals: x and n, and multiplies x by 2 to the power n.
304 //
305 // #### Usage
306 //
307 // '''
308 // _ : ldexp : _
309 // '''
310 //-----
311 ldexp = ffunction(float ldexpf|ldexp|ldexpl (float, int), <math.h>,"");
312
313
314 //-----'(ma.)scalb'-----
315 // Takes two input signals: x and n, and multiplies x by 2 to the power n.
316 //
317 // #### Usage
318 //
319 // '''
320 // _ : scalb : _
321 // '''
322 //-----
323 scalb = ffunction(float scalbnf|scalbn|scalbnl (float, int), <math.h>,"");
324
325
326 //-----'(ma.)log1p'-----
327 // Computes log(1 + x) without undue loss of accuracy when x is nearly zero.
328 //
329 // #### Usage
330 //
331 // '''
332 // _ : log1p : _
333 // '''
334 //-----
335 log1p = ffunction(float log1pf|log1p|log1pl (float), <math.h>,"");
336
337
338 //-----'(ma.)logb'-----
339 // Return exponent of the input signal as a floating-point number.
340 //
341 // #### Usage
342 //
343 // '''
344 // _ : logb : _
345 // '''
346 //-----
347 logb = ffunction(float logbf|logb|logbl (float), <math.h>,"");
348
349
350 //-----'(ma.)ilogb'-----
351 // Return exponent of the input signal as an integer number.
352 //
353 // #### Usage
354 //
355 // '''
356 // _ : ilogb : _
357 // '''
358 //-----
359 ilogb = ffunction(int ilogbf|ilogb|ilogbl (float), <math.h>,"");
360
361

```

```

362 //-----'(ma.)log2'-----
363 // Returns the base 2 logarithm of x.
364 //
365 // #### Usage
366 //
367 // '''
368 // _ : log2 : _
369 // '''
370 //-----
371 log2(x) = log(x)/log(2.0);
372
373
374 //-----'(ma.)expm1'-----
375 // Return exponent of the input signal minus 1 with better precision.
376 //
377 // #### Usage
378 //
379 // '''
380 // _ : expm1 : _
381 // '''
382 //-----
383 expm1 = ffunction(float expm1f|expm1|expm1l (float), <math.h>,"");
384
385
386 //-----'(ma.)acosh'-----
387 // Computes the principle value of the inverse hyperbolic cosine
388 // of the input signal.
389 //
390 // #### Usage
391 //
392 // '''
393 // _ : acosh : _
394 // '''
395 //-----
396 acosh = ffunction(float acoshf|acosh|acoshl (float), <math.h>,"");
397
398
399 //-----'(ma.)asinh'-----
400 // Computes the inverse hyperbolic sine of the input signal.
401 //
402 // #### Usage
403 //
404 // '''
405 // _ : asinh : _
406 // '''
407 //-----
408 asinh = ffunction(float asinhf|asinh|asinh1 (float), <math.h>,"");
409
410
411 //-----'(ma.)atanh'-----
412 // Computes the inverse hyperbolic tangent of the input signal.
413 //
414 // #### Usage
415 //
416 // '''
417 // _ : atanh : _
418 // '''
419 //-----
420 atanh = ffunction(float atanhf|atanh|atanhl (float), <math.h>,"");
421
422
423 //-----'(ma.)sinh'-----
424 // Computes the hyperbolic sine of the input signal.
425 //
426 // #### Usage
427 //
428 // '''
429 // _ : sinh : _

```



```

430 // '''
431 //-----'ma.cosh'-----
432 sinh = ffunction(float sinhf|sinh|sinhl (float), <math.h>, "");
433
434
435 //-----'ma.cosh'-----
436 // Computes the hyperbolic cosine of the input signal.
437 //
438 // #### Usage
439 //
440 // '''
441 // _ : cosh : _
442 // '''
443 //-----
444 cosh = ffunction(float coshf|cosh|coshl (float), <math.h>, "");
445
446
447 //-----'ma.tanh'-----
448 // Computes the hyperbolic tangent of the input signal.
449 //
450 // #### Usage
451 //
452 // '''
453 // _ : tanh : _
454 // '''
455 //-----
456 tanh = ffunction(float tanhf|tanh|tanhl (float), <math.h>,"");
457
458
459 //-----'ma.erf'-----
460 // Computes the error function of the input signal.
461 //
462 // #### Usage
463 //
464 // '''
465 // _ : erf : _
466 // '''
467 //-----
468 erf = ffunction(float erff|erf|erfl(float), <math.h>,"");
469
470
471 //-----'ma.erfc'-----
472 // Computes the complementary error function of the input signal.
473 //
474 // #### Usage
475 //
476 // '''
477 // _ : erfc : _
478 // '''
479 //-----
480 erfc = ffunction(float erfcf|erfc|erfcl(float), <math.h>,"");
481
482
483 //-----'ma.gamma'-----
484 // Computes the gamma function of the input signal.
485 //
486 // #### Usage
487 //
488 // '''
489 // _ : gamma : _
490 // '''
491 //-----
492 gamma = ffunction(float tgammaf|tgamma|tgammal(float), <math.h>,"");
493
494
495 //-----'ma.lgamma'-----
496 // Calculates the natural logarithm of the absolute value of
497 // the gamma function of the input signal.

```

```

498 //
499 // #### Usage
500 //
501 // '''
502 // _ : lgamma : _
503 // '''
504 //-----
505 lgamma = ffunction(float lgammaf|lgamma|lgammal(float), <math.h>,"");
506
507 //-----'(ma.)J0'-----
508 // Computes the Bessel function of the first kind of order 0
509 // of the input signal.
510 //
511 //
512 // #### Usage
513 //
514 // '''
515 // _ : J0 : _
516 // '''
517 //-----
518 J0 = ffunction(float j0(float), <math.h>,"");
519
520 //-----'(ma.)J1'-----
521 // Computes the Bessel function of the first kind of order 1
522 // of the input signal.
523 //
524 //
525 // #### Usage
526 //
527 // '''
528 // _ : J1 : _
529 // '''
530 //-----
531 J1 = ffunction(float j1(float), <math.h>,"");
532
533 //-----'(ma.)Jn'-----
534 // Computes the Bessel function of the first kind of order n
535 // (first input signal) of the second input signal.
536 //
537 //
538 // #### Usage
539 //
540 // '''
541 // _,_ : Jn : _
542 // '''
543 //-----
544 Jn = ffunction(float jn(int, float), <math.h>,"");
545
546 //-----'(ma.)Y0'-----
547 // Computes the linearly independent Bessel function of the second kind
548 // of order 0 of the input signal.
549 //
550 //
551 // #### Usage
552 //
553 // '''
554 // _ : Y0 : _
555 // '''
556 //-----
557 Y0 = ffunction(float y0(float), <math.h>,"");
558
559 //-----'(ma.)Y1'-----
560 // Computes the linearly independent Bessel function of the second kind
561 // of order 1 of the input signal.
562 //
563 //
564 // #### Usage
565 //

```

```

566 // '''
567 // _ : Y0 : _
568 // '''
569 //-----
570 Y1 = ffunction(float y1(float), <math.h>,"");
571
572
573 //-----'(ma.)Yn'-----
574 // Computes the linearly independent Bessel function of the second kind
575 // of order n (first input signal) of the second input signal.
576 //
577 // #### Usage
578 //
579 // '''
580 // _ : Yn : _
581 // '''
582 //-----
583 Yn = ffunction(float yn(int, float), <math.h>,"");
584
585
586 //-----'(ma.)fabs', '(ma.)fmax', '(ma.)fmin
587 // Just for compatibility...
588 //
589 // '''
590 // fabs = abs
591 // fmax = max
592 // fmin = min
593 // '''
594 //-----
595 fabs = abs;
596 fmax = max;
597 fmin = min;
598
599 //-----'(ma.)np2'-----
600 // Gives the next power of 2 of x.
601 //
602 // #### Usage
603 //
604 // '''
605 // np2(n) : _
606 // '''
607 //
608 // Where:
609 //
610 // * 'n': an integer
611 //-----
612 np2 = -(1) <: >>(1)|_ <: >>(2)|_ <: >>(4)|_ <: >>(8)|_ <: >>(16)|_ : +(1);
613
614
615 //-----'(ma.)frac'-----
616 // Gives the fractional part of n.
617 //
618 // #### Usage
619 //
620 // '''
621 // frac(n) : _
622 // '''
623 //
624 // Where:
625 //
626 // * 'n': a decimal number
627 //-----
628 frac(n) = n - floor(n);
629 decimal = frac;
630
631 // NOTE: decimal does the same thing as frac but using floor instead. JOS uses frac a lot
632 // in filters.lib so we decided to keep that one... decimal is declared though for

```

```

633 // backward compatibility.
634 // decimal(n) = n - floor(n);
635
636
637 //-----'(ma.)modulo'-----
638 // Modulus operation using the '(x%y+y)%y' formula to ensures the result is always non-
        // negative, even if 'x' is negative.
639 //
640 // #### Usage
641 //
642 // '''
643 // modulo(x,y) : _
644 // '''
645 //
646 // Where:
647 //
648 // * 'x': the numerator
649 // * 'y': the denominator
650 //-----
651 modulo(x,y) = (x % y + y) % y;
652
653
654 //-----'(ma.)isnan'-----
655 // Return non-zero if x is a NaN.
656 //
657 // #### Usage
658 //
659 // '''
660 // isnan(x)
661 // _ : isnan : _
662 // '''
663 //
664 // Where:
665 //
666 // * 'x': signal to analyse
667 //-----
668 isnan = ffunction(int isnanf|isnan|isnanl (float),<math.h>,"");
669
670
671 //-----'(ma.)isinf'-----
672 // Return non-zero if x is a positive or negative infinity.
673 //
674 // #### Usage
675 //
676 // '''
677 // isinf(x)
678 // _ : isinf : _
679 // '''
680 //
681 // Where:
682 //
683 // * 'x': signal to analyse
684 //-----
685 isinf = ffunction(int isinff|isinf|isinfl (float),<math.h>,"");
686
687 nextafter = ffunction(float nextafter(float, float),<math.h>,"");
688
689
690 //-----'(ma.)chebychev'-----
691 // Chebychev transformation of order N.
692 //
693 // #### Usage
694 //
695 // '''
696 // _ : chebychev(N) : _
697 // '''
698 //
699 // Where:

```

```

700 //
701 // * 'N': the order of the polynomial, a constant numerical expression
702 //
703 // #### Semantics
704 //
705 // '''
706 // T[0](x) = 1,
707 // T[1](x) = x,
708 // T[n](x) = 2x*T[n-1](x) - T[n-2](x)
709 // '''
710 //
711 // #### Reference
712 //
713 // <http://en.wikipedia.org/wiki/Chebyshev\_polynomial>
714 //-----
715 chebychev(0,x) = 1;
716 chebychev(1,x) = x;
717 chebychev(n,x) = 2*x*chebychev(n-1, x) - chebychev(n-2, x);
718
719
720 //-----'(ma.)chebypoly'-----
721 // Linear combination of the first Chebyshev polynomials.
722 //
723 // #### Usage
724 //
725 // '''
726 // _ : chebypoly((c0,c1,...,cn)) : _
727 // '''
728 //
729 // Where:
730 //
731 // * 'cn': the different Chebyshev polynomials such that:
732 // chebypoly((c0,c1,...,cn)) = Sum of chebychev(i)*ci
733 //
734 // #### Reference
735 //
736 // <http://www.csounds.com/manual/html/chebypoly.html>
737 //-----
738 chebypoly(lcoef) = _ <: L(0,lcoef) :> _
739   with {
740     L(n,(c,cs)) = chebychev(n)*c, L(n+1,cs);
741     L(n,c)      = chebychev(n)*c;
742   };
743
744
745 //-----'(ma.)diffn'-----
746 // Negated first-order difference.
747 //
748 // #### Usage
749 //
750 // '''
751 // _ : diffn : _
752 // '''
753 //-----
754 diffn(x) = x' - x; // negated first-order difference
755
756
757 //-----'(ma.)signum'-----
758 // The signum function signum(x) is defined as
759 // -1 for x<0, 0 for x==0, and 1 for x>0.
760 //
761 // #### Usage
762 //
763 // '''
764 // _ : signum : _
765 // '''
766 //-----
767 signum(x) = (x>0)-(x<0);

```

```

768
769
770 //-----'(ma.)nextpow2'-----
771 // The nextpow2(x) returns the lowest integer m such that
772 // 2^m >= x.
773 //
774 // #### Usage
775 //
776 // '''
777 // 2^nextpow2(n) : _
778 // '''
779 // Useful for allocating delay lines, e.g.,
780 //
781 // delay(2^nextpow2(maxDelayNeeded), currentDelay);
782 // '''
783 //-----
784 nextpow2(x) = ceil(log(x)/log(2.0));
785
786
787 //-----'(ma.)zc'-----
788 // Indicator function for zero-crossing: it returns 1 if a zero-crossing
789 // occurs, 0 otherwise.
790 //
791 // #### Usage
792 //
793 // '''
794 // _ : zc : _
795 // '''
796 //-----
797 zc(x) = x * x' < 0;
798
799
800 //-----'(ma.)primes'-----
801 // Return the n-th prime using a waveform primitive. Note that primes(0) is 2,
802 // primes(1) is 3, and so on. The waveform is length 2048, so the largest
803 // precomputed prime is primes(2047) which is 17863.
804 //
805 // #### Usage
806 //
807 // '''
808 // _ : primes : _
809 // '''
810 //-----
811 primes(x) = rdtbl(waveform{
812 2,3,5,7,11,13,17,19,23,29,
813 31,37,41,43,47,53,59,61,67,71,
814 73,79,83,89,97,101,103,107,109,113,
815 127,131,137,139,149,151,157,163,167,173,
816 179,181,191,193,197,199,211,223,227,229,
817 233,239,241,251,257,263,269,271,277,281,
818 283,293,307,311,313,317,331,337,347,349,
819 353,359,367,373,379,383,389,397,401,409,
820 419,421,431,433,439,443,449,457,461,463,
821 467,479,487,491,499,503,509,521,523,541,
822 547,557,563,569,571,577,587,593,599,601,
823 607,613,617,619,631,641,643,647,653,659,
824 661,673,677,683,691,701,709,719,727,733,
825 739,743,751,757,761,769,773,787,797,809,
826 811,821,823,827,829,839,853,857,859,863,
827 877,881,883,887,907,911,919,929,937,941,
828 947,953,967,971,977,983,991,997,1009,1013,
829 1019,1021,1031,1033,1039,1049,1051,1061,1063,1069,
830 1087,1091,1093,1097,1103,1109,1117,1123,1129,1151,
831 1153,1163,1171,1181,1187,1193,1201,1213,1217,1223,
832 1229,1231,1237,1249,1259,1277,1279,1283,1289,1291,
833 1297,1301,1303,1307,1319,1321,1327,1361,1367,1373,
834 1381,1399,1409,1423,1427,1429,1433,1439,1447,1451,
835 1453,1459,1471,1481,1483,1487,1489,1493,1499,1511,

```

836	1523,1531,1543,1549,1553,1559,1567,1571,1579,1583,
837	1597,1601,1607,1609,1613,1619,1621,1627,1637,1657,
838	1663,1667,1669,1693,1697,1699,1709,1721,1723,1733,
839	1741,1747,1753,1759,1777,1783,1787,1789,1801,1811,
840	1823,1831,1847,1861,1867,1871,1873,1877,1879,1889,
841	1901,1907,1913,1931,1933,1949,1951,1973,1979,1987,
842	1993,1997,1999,2003,2011,2017,2027,2029,2039,2053,
843	2063,2069,2081,2083,2087,2089,2099,2111,2113,2129,
844	2131,2137,2141,2143,2153,2161,2179,2203,2207,2213,
845	2221,2237,2239,2243,2251,2267,2269,2273,2281,2287,
846	2293,2297,2309,2311,2333,2339,2341,2347,2351,2357,
847	2371,2377,2381,2383,2389,2393,2399,2411,2417,2423,
848	2437,2441,2447,2459,2467,2473,2477,2503,2521,2531,
849	2539,2543,2549,2551,2557,2579,2591,2593,2609,2617,
850	2621,2633,2647,2657,2659,2663,2671,2677,2683,2687,
851	2689,2693,2699,2707,2711,2713,2719,2729,2731,2741,
852	2749,2753,2767,2777,2789,2791,2797,2801,2803,2819,
853	2833,2837,2843,2851,2857,2861,2879,2887,2897,2903,
854	2909,2917,2927,2939,2953,2957,2963,2969,2971,2999,
855	3001,3011,3019,3023,3037,3041,3049,3061,3067,3079,
856	3083,3089,3109,3119,3121,3137,3163,3167,3169,3181,
857	3187,3191,3203,3209,3217,3221,3229,3251,3253,3257,
858	3259,3271,3299,3301,3307,3313,3319,3323,3329,3331,
859	3343,3347,3359,3361,3371,3373,3389,3391,3407,3413,
860	3433,3449,3457,3461,3463,3467,3469,3491,3499,3511,
861	3517,3527,3529,3533,3539,3541,3547,3557,3559,3571,
862	3581,3583,3593,3607,3613,3617,3623,3631,3637,3643,
863	3659,3671,3673,3677,3691,3697,3701,3709,3719,3727,
864	3733,3739,3761,3767,3769,3779,3793,3797,3803,3821,
865	3823,3833,3847,3851,3853,3863,3877,3881,3889,3907,
866	3911,3917,3919,3923,3929,3931,3943,3947,3967,3989,
867	4001,4003,4007,4013,4019,4021,4027,4049,4051,4057,
868	4073,4079,4091,4093,4099,4111,4127,4129,4133,4139,
869	4153,4157,4159,4177,4201,4211,4217,4219,4229,4231,
870	4241,4243,4253,4259,4261,4271,4273,4283,4289,4297,
871	4327,4337,4339,4349,4357,4363,4373,4391,4397,4409,
872	4421,4423,4441,4447,4451,4457,4463,4481,4483,4493,
873	4507,4513,4517,4519,4523,4547,4549,4561,4567,4583,
874	4591,4597,4603,4621,4637,4639,4643,4649,4651,4657,
875	4663,4673,4679,4691,4703,4721,4723,4729,4733,4751,
876	4759,4783,4787,4789,4793,4799,4801,4813,4817,4831,
877	4861,4871,4877,4889,4903,4909,4919,4931,4933,4937,
878	4943,4951,4957,4967,4969,4973,4987,4993,4999,5003,
879	5009,5011,5021,5023,5039,5051,5059,5077,5081,5087,
880	5099,5101,5107,5113,5119,5147,5153,5167,5171,5179,
881	5189,5197,5209,5227,5231,5233,5237,5261,5273,5279,
882	5281,5297,5303,5309,5323,5333,5347,5351,5381,5387,
883	5393,5399,5407,5413,5417,5419,5431,5437,5441,5443,
884	5449,5471,5477,5479,5483,5501,5503,5507,5519,5521,
885	5527,5531,5557,5563,5569,5573,5581,5591,5623,5639,
886	5641,5647,5651,5653,5657,5659,5669,5683,5689,5693,
887	5701,5711,5717,5737,5741,5743,5749,5779,5783,5791,
888	5801,5807,5813,5821,5827,5839,5843,5849,5851,5857,
889	5861,5867,5869,5879,5881,5897,5903,5923,5927,5939,
890	5953,5981,5987,6007,6011,6029,6037,6043,6047,6053,
891	6067,6073,6079,6089,6091,6101,6113,6121,6131,6133,
892	6143,6151,6163,6173,6197,6199,6203,6211,6217,6221,
893	6229,6247,6257,6263,6269,6271,6277,6287,6299,6301,
894	6311,6317,6323,6329,6337,6343,6353,6359,6361,6367,
895	6373,6379,6389,6397,6421,6427,6449,6451,6469,6473,
896	6481,6491,6521,6529,6547,6551,6553,6563,6569,6571,
897	6577,6581,6599,6607,6619,6637,6653,6659,6661,6673,
898	6679,6689,6691,6701,6703,6709,6719,6733,6737,6761,
899	6763,6779,6781,6791,6793,6803,6823,6827,6829,6833,
900	6841,6857,6863,6869,6871,6883,6899,6907,6911,6917,
901	6947,6949,6959,6961,6967,6971,6977,6983,6991,6997,
902	7001,7013,7019,7027,7039,7043,7057,7069,7079,7103,
903	7109,7121,7127,7129,7151,7159,7177,7187,7193,7207,

904 7211,7213,7219,7229,7237,7243,7247,7253,7283,7297,
 905 7307,7309,7321,7331,7333,7349,7351,7369,7393,7411,
 906 7417,7433,7451,7457,7459,7477,7481,7487,7489,7499,
 907 7507,7517,7523,7529,7537,7541,7547,7549,7559,7561,
 908 7573,7577,7583,7589,7591,7603,7607,7621,7639,7643,
 909 7649,7669,7673,7681,7687,7691,7699,7703,7717,7723,
 910 7727,7741,7753,7757,7759,7789,7793,7817,7823,7829,
 911 7841,7853,7867,7873,7877,7879,7883,7901,7907,7919,
 912 7927,7933,7937,7949,7951,7963,7993,8009,8011,8017,
 913 8039,8053,8059,8069,8081,8087,8089,8093,8101,8111,
 914 8117,8123,8147,8161,8167,8171,8179,8191,8209,8219,
 915 8221,8231,8233,8237,8243,8263,8269,8273,8287,8291,
 916 8293,8297,8311,8317,8329,8353,8363,8369,8377,8387,
 917 8389,8419,8423,8429,8431,8443,8447,8461,8467,8501,
 918 8513,8521,8527,8537,8539,8543,8563,8573,8581,8597,
 919 8599,8609,8623,8627,8629,8641,8647,8663,8669,8677,
 920 8681,8689,8693,8699,8707,8713,8719,8731,8737,8741,
 921 8747,8753,8761,8779,8783,8803,8807,8819,8821,8831,
 922 8837,8839,8849,8861,8863,8867,8887,8893,8923,8929,
 923 8933,8941,8951,8963,8969,8971,8999,9001,9007,9011,
 924 9013,9029,9041,9043,9049,9059,9067,9091,9103,9109,
 925 9127,9133,9137,9151,9157,9161,9173,9181,9187,9199,
 926 9203,9209,9221,9227,9239,9241,9257,9277,9281,9283,
 927 9293,9311,9319,9323,9337,9341,9343,9349,9371,9377,
 928 9391,9397,9403,9413,9419,9421,9431,9433,9437,9439,
 929 9461,9463,9467,9473,9479,9491,9497,9511,9521,9533,
 930 9539,9547,9551,9587,9601,9613,9619,9623,9629,9631,
 931 9643,9649,9661,9677,9679,9689,9697,9719,9721,9733,
 932 9739,9743,9749,9767,9769,9781,9787,9791,9803,9811,
 933 9817,9829,9833,9839,9851,9857,9859,9871,9883,9887,
 934 9901,9907,9923,9929,9931,9941,9949,9967,9973,10007,
 935 10009,10037,10039,10061,10067,10069,10079,10091,10093,10099,
 936 10103,10111,10133,10139,10141,10151,10159,10163,10169,10177,
 937 10181,10193,10211,10223,10243,10247,10253,10259,10267,10271,
 938 10273,10289,10301,10303,10313,10321,10331,10333,10337,10343,
 939 10357,10369,10391,10399,10427,10429,10433,10453,10457,10459,
 940 10463,10477,10487,10499,10501,10513,10529,10531,10559,10567,
 941 10589,10597,10601,10607,10613,10627,10631,10639,10651,10657,
 942 10663,10667,10687,10691,10709,10711,10723,10729,10733,10739,
 943 10753,10771,10781,10789,10799,10831,10837,10847,10853,10859,
 944 10861,10867,10883,10889,10891,10903,10909,10937,10939,10949,
 945 10957,10973,10979,10987,10993,11003,11027,11047,11057,11059,
 946 11069,11071,11083,11087,11093,11113,11117,11119,11131,11149,
 947 11159,11161,11171,11173,11177,11197,11213,11239,11243,11251,
 948 11257,11261,11273,11279,11287,11299,11311,11317,11321,11329,
 949 11351,11353,11369,11383,11393,11399,11411,11423,11437,11443,
 950 11447,11467,11471,11483,11489,11491,11497,11503,11519,11527,
 951 11549,11551,11579,11587,11593,11597,11617,11621,11633,11657,
 952 11677,11681,11689,11699,11701,11717,11719,11731,11743,11777,
 953 11779,11783,11789,11801,11807,11813,11821,11827,11831,11833,
 954 11839,11863,11867,11887,11897,11903,11909,11923,11927,11933,
 955 11939,11941,11953,11959,11969,11971,11981,11987,12007,12011,
 956 12037,12041,12043,12049,12071,12073,12097,12101,12107,12109,
 957 12113,12119,12143,12149,12157,12161,12163,12197,12203,12211,
 958 12227,12239,12241,12251,12253,12263,12269,12277,12281,12289,
 959 12301,12323,12329,12343,12347,12373,12377,12379,12391,12401,
 960 12409,12413,12421,12433,12437,12451,12457,12473,12479,12487,
 961 12491,12497,12503,12511,12517,12527,12539,12541,12547,12553,
 962 12569,12577,12583,12589,12601,12611,12613,12619,12637,12641,
 963 12647,12653,12659,12671,12689,12697,12703,12713,12721,12739,
 964 12743,12757,12763,12781,12791,12799,12809,12821,12823,12829,
 965 12841,12853,12889,12893,12899,12907,12911,12917,12919,12923,
 966 12941,12953,12959,12967,12973,12979,12983,13001,13003,13007,
 967 13009,13033,13037,13043,13049,13063,13093,13099,13103,13109,
 968 13121,13127,13147,13151,13159,13163,13171,13177,13183,13187,
 969 13217,13219,13229,13241,13249,13259,13267,13291,13297,13309,
 970 13313,13327,13331,13337,13339,13367,13381,13397,13399,13411,
 971 13417,13421,13441,13451,13457,13463,13469,13477,13487,13499,


```

972 | 13513,13523,13537,13553,13567,13577,13591,13597,13613,13619,
973 | 13627,13633,13649,13669,13679,13681,13687,13691,13693,13697,
974 | 13709,13711,13721,13723,13729,13751,13757,13759,13763,13781,
975 | 13789,13799,13807,13829,13831,13841,13859,13873,13877,13879,
976 | 13883,13901,13903,13907,13913,13921,13931,13933,13963,13967,
977 | 13997,13999,14009,14011,14029,14033,14051,14057,14071,14081,
978 | 14083,14087,14107,14143,14149,14153,14159,14173,14177,14197,
979 | 14207,14221,14243,14249,14251,14281,14293,14303,14321,14323,
980 | 14327,14341,14347,14369,14387,14389,14401,14407,14411,14419,
981 | 14423,14431,14437,14447,14449,14461,14479,14489,14503,14519,
982 | 14533,14537,14543,14549,14551,14557,14561,14563,14591,14593,
983 | 14621,14627,14629,14633,14639,14653,14657,14669,14683,14699,
984 | 14713,14717,14723,14731,14737,14741,14747,14753,14759,14767,
985 | 14771,14779,14783,14797,14813,14821,14827,14831,14843,14851,
986 | 14867,14869,14879,14887,14891,14897,14923,14929,14939,14947,
987 | 14951,14957,14969,14983,15013,15017,15031,15053,15061,15073,
988 | 15077,15083,15091,15101,15107,15121,15131,15137,15139,15149,
989 | 15161,15173,15187,15193,15199,15217,15227,15233,15241,15259,
990 | 15263,15269,15271,15277,15287,15289,15299,15307,15313,15319,
991 | 15329,15331,15349,15359,15361,15373,15377,15383,15391,15401,
992 | 15413,15427,15439,15443,15451,15461,15467,15473,15493,15497,
993 | 15511,15527,15541,15551,15559,15569,15581,15583,15601,15607,
994 | 15619,15629,15641,15643,15647,15649,15661,15667,15671,15679,
995 | 15683,15727,15731,15733,15737,15739,15749,15761,15767,15773,
996 | 15787,15791,15797,15803,15809,15817,15823,15859,15877,15881,
997 | 15887,15889,15901,15907,15913,15919,15923,15937,15959,15971,
998 | 15973,15991,16001,16007,16033,16057,16061,16063,16067,16069,
999 | 16073,16087,16091,16097,16103,16111,16127,16139,16141,16183,
1000 | 16187,16189,16193,16217,16223,16229,16231,16249,16253,16267,
1001 | 16273,16301,16319,16333,16339,16349,16361,16363,16369,16381,
1002 | 16411,16417,16421,16427,16433,16447,16451,16453,16477,16481,
1003 | 16487,16493,16519,16529,16547,16553,16561,16567,16573,16603,
1004 | 16607,16619,16631,16633,16649,16651,16657,16661,16673,16691,
1005 | 16693,16699,16703,16729,16741,16747,16759,16763,16787,16811,
1006 | 16823,16829,16831,16843,16871,16879,16883,16889,16901,16903,
1007 | 16921,16927,16931,16937,16943,16963,16979,16981,16987,16993,
1008 | 17011,17021,17027,17029,17033,17041,17047,17053,17077,17093,
1009 | 17099,17107,17117,17123,17137,17159,17167,17183,17189,17191,
1010 | 17203,17207,17209,17231,17239,17257,17291,17293,17299,17317,
1011 | 17321,17327,17333,17341,17351,17359,17377,17383,17387,17389,
1012 | 17393,17401,17417,17419,17431,17443,17449,17467,17471,17477,
1013 | 17483,17489,17491,17497,17509,17519,17539,17551,17569,17573,
1014 | 17579,17581,17597,17599,17609,17623,17627,17657,17659,17669,
1015 | 17681,17683,17707,17713,17729,17737,17747,17749,17761,17783,
1016 | 17789,17791,17807,17827,17837,17839,17851,17863
1017 | }, x);

```

Listing 5: platform.lib

```

1  //##### platform.lib #####
2  // A library to handle platform specific code in Faust. Its official prefix is 'pl'.
3  //
4  // #### References
5  // * <https://github.com/grame-cncm/faustlibraries/blob/master/platform.lib>
6  //#####
7  // It can be reimplemented to globally change the SR and the tablesize definitions
8
9
10 /*****
11 *****/
12 FAUST library file
13 Copyright (C) 2020 GRAME, Centre National de Creation Musicale
14 -----
15 This program is free software; you can redistribute it and/or modify

```

```

16  it under the terms of the GNU Lesser General Public License as
17  published by the Free Software Foundation; either version 2.1 of the
18  License, or (at your option) any later version.
19
20  This program is distributed in the hope that it will be useful,
21  but WITHOUT ANY WARRANTY; without even the implied warranty of
22  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
23  GNU Lesser General Public License for more details.
24
25  You should have received a copy of the GNU Lesser General Public
26  License along with the GNU C Library; if not, write to the Free
27  Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
28  02111-1307 USA.
29
30  EXCEPTION TO THE LGPL LICENSE : As a special exception, you may create a
31  larger FAUST program which directly or indirectly imports this library
32  file and still distribute the compiled code generated by the FAUST
33  compiler, or a modified version of this compiled code, under your own
34  copyright and license. This EXCEPTION TO THE LGPL LICENSE explicitly
35  grants you the right to freely choose the license for the resulting
36  compiled code. In particular the resulting compiled code has no obligation
37  to be LGPL or GPL. For example you are free to choose a commercial or
38  closed source license or any other license if you decide so.
39  *****/
40  *****/
41
42  declare name "Generic Platform Library";
43  declare version "1.3.0";
44
45  //-----'(pl.)SR'-----
46  // Current sampling rate (between 1 and 192000Hz). Constant during
47  // program execution. Setting this value to a constant will allow the
48  // compiler to optimize the code by computing constant expressions at
49  // compile time, and can be valuable for performance, especially on
50  // embedded systems.
51  //-----
52  SR = min(192000.0, max(1.0, fconstant(int fSamplingFreq, <math.h>)));
53
54  //-----'(pl.)BS'-----
55  // Current block-size (between 1 and 16384 frames). Can change during the execution.
56  //-----
57  BS = min(16384.0, max(1.0, fvariable(int count, <math.h>)));
58
59  //-----'(pl.)tablesize'-----
60  // Oscillator table size. This value is used to define the size of the
61  // table used by the oscillators. It is usually a power of 2 and can be lowered
62  // to save memory. The default value is 65536.
63  //-----
64  tablesize = 1 << 16;

```