Sinewaveoscillator

October 15, 2014

math.lib/name	Math Library
math.lib/author	GRAME
math.lib/copyright	GRAME
math.lib/version	1.0
math.lib/license	LGPL

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

1 Mathematical definition of `process`

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

1. Output signal y such that

$$y(t) = r_1(t)$$

2. Input signal (none)

3. Intermediate signals r_i for $i \in [1, 2]$ such that

$$r_2(t) = r_1(t-1) \cdot (r_1(t-1) \neq 1)$$
$$r_1(t) = k_1 + r_2(t-1)$$

4. Constant k_1 such that

$$k_1 = \min\left(0.05 \cdot \min\left(192000, \max\left(1, f_S\right)\right), 1\right)$$

2 Block diagram of `process`

The block diagram of `process` is shown on Figure **??** (page **??**).

Figure 1: Block diagram of `process`

3 Notice

- This document was generated using Faust version 0.9.46 on October 15, 2014.
- 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 mapping[1] 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.
- The `Sinewaveoscillator-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.

4 Faust code listings

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

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

Listing 1: `Sinewaveoscillator.dsp`

```
import("math.lib");
loop(begin, eind, tijdinms)=
(_+((eind - begin)/tijdinms:max(1):min(SR/20)))~(((_<:(begin*(_==eind),_*(_!=eind):_+_)))@
     (1)):

_;

BitDepth=_/2^(8*checkbox("8 or 16 bit")+ 1):_;
sine=_*(2*PI):sin:_;
process = loop(0, 1, 100);//BitDepth:sine;

//((eind - begin)/tijdinms:max(1):min(SR/20))
```

Listing 2: `math.lib`

```
/************************************************************************
 ************************************************************************
    FAUST library file
    Copyright (C) 2003-2011 GRAME, Centre National de Creation Musicale
    --------------------------------------------------------------------
    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as
    published by the Free Software Foundation; either version 2.1 of the
    License, or (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with the GNU C Library; if not, write to the Free
    Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
    02111-1307 USA.
 ************************************************************************
 ************************************************************************/

declare name "Math Library";
declare author "GRAME";
declare copyright "GRAME";
declare version "1.0";
declare license "LGPL";

//-----------------------------------------------------------------------------
//                    Mathematic library for Faust

// Implementation of the math.h file as Faust foreign functions
//
// History
// ----------
// 28/06/2005  [YO]   postfixed functions with 'f' to force float version
//                    instead of double
//             [YO]   removed 'modf' because it requires a pointer as argument
//-----------------------------------------------------------------------------

// -- Utilities and constants

SR         = min(192000, max(1, fconstant(int fSamplingFreq, <math.h>)));
BS         = fvariable(int count, <math.h>);

PI         = 3.1415926535897932385;

```

```
// -- neg and inv functions

neg(x)     = -x;
inv(x)     = 1/x;

// -- Trigonometric Functions

//acos     = ffunction(float acosf (float), <math.h>, "");
//asin     = ffunction(float asinf (float), <math.h>, "");
//atan     = ffunction(float atanf (float), <math.h>, "");
//atan2    = ffunction(float atan2f (float, float), <math.h>, "");

//sin          = ffunction(float sinf (float), <math.h>, "");
//cos          = ffunction(float cosf (float), <math.h>, "");
//tan          = ffunction(float tanf (float), <math.h>,"");

// -- Exponential Functions

//exp      = ffunction(float expf (float), <math.h>,"");
//log      = ffunction(float logf (float), <math.h>,"");
//log10    = ffunction(float log10f (float), <math.h>,"");
//pow      = ffunction(float powf (float, float), <math.h>,"");
//sqrt     = ffunction(float sqrtf (float), <math.h>,"");
cbrt       = ffunction(float cbrtf (float), <math.h>,"");
hypot      = ffunction(float hypotf (float, float), <math.h>,"");
ldexp      = ffunction(float ldexpf (float, int), <math.h>,"");
scalb      = ffunction(float scalbf (float, float), <math.h>,"");
log1p      = ffunction(float log1pf (float), <math.h>,"");
logb       = ffunction(float logbf (float), <math.h>,"");
ilogb      = ffunction(int ilogbf (float), <math.h>,"");
expm1      = ffunction(float expm1f (float), <math.h>,"");

// -- Hyperbolic Functions

acosh      = ffunction(float acoshf (float), <math.h>, "");
asinh      = ffunction(float asinhf (float), <math.h>, "");
atanh      = ffunction(float atanhf (float), <math.h>, "");

sinh       = ffunction(float sinhf (float), <math.h>, "");
cosh       = ffunction(float coshf (float), <math.h>, "");
tanh       = ffunction(float tanhf (float), <math.h>,"");

// -- Remainder Functions

//fmod     = ffunction(float fmodf (float, float),<math.h>,"");
//remainder    = ffunction(float remainderf (float, float),<math.h>,"");

// -- Nearest Integer Functions

//floor        = ffunction(float floorf (float), <math.h>,"");
//ceil     = ffunction(float ceilf (float), <math.h>,"");
//rint     = ffunction(float rintf (float), <math.h>,"");

// -- Special Functions

erf        = ffunction(float erff(float), <math.h>,"");
erfc       = ffunction(float erfcf(float), <math.h>,"");
gamma      = ffunction(float gammaf(float), <math.h>,"");
J0         = ffunction(float j0f(float), <math.h>,"");
J1         = ffunction(float j1f(float), <math.h>,"");
Jn         = ffunction(float jnf(int, float), <math.h>,"");
lgamma     = ffunction(float lgammaf(float), <math.h>,"");
Y0         = ffunction(float y0f(float), <math.h>,"");
Y1         = ffunction(float y1f(float), <math.h>,"");
Yn         = ffunction(float ynf(int, float), <math.h>,"");


// -- Miscellaneous Functions
```

```
116
117 //fabs     = ffunction(float fabsf (float), <math.h>,"");
118 //fmax     = ffunction(float max (float, float),<math.h>,"");
119 //fmin     = ffunction(float min (float, float),<math.h>,"");
120
121 fabs = abs;
122 fmax = max;
123 fmin = min;
124
125 isnan      = ffunction(int isnan (float),<math.h>,"");
126 nextafter  = ffunction(float nextafter(float, float),<math.h>,"");
127
128 // Pattern matching functions to count and access the elements of a list
129 // USAGE :  count ((10,20,30,40))  -> 4
130 //          take  (3,(10,20,30,40)) -> 30
131 //
132
133 count ((xs, xxs)) = 1 + count(xxs);
134 count (xx) = 1;
135
136 take (1, (xs, xxs))   = xs;
137 take (1, xs)          = xs;
138 take (nn, (xs, xxs))  = take (nn-1, xxs);
139
140 // linear interpolation between two signals
141 interpolate(i) = *(1.0-i),*(i) : +;
142
143 // if-then-else implemented with a select2.
144 if(cond,thn,els) = select2(cond,els,thn);
145
146
147 //-----------------------------------------------------------------
148 // countdown(count,trig)
149 // start counting down from count, count-1,...,0 when trig > 0
150 //-----------------------------------------------------------------
151 countdown(count, trig) = \(c).(if(trig>0, count, max(0, c-1))) ~_;
152
153 //-----------------------------------------------------------------
154 // countup(count,trig)
155 // start counting down from 0, 1, ... count-1, count when trig > 0
156 //-----------------------------------------------------------------
157 countup(count, trig)  = \(c).(if(trig>0, 0, min(count, c+1))) ~_;
158
159 /***************************************************************
160  *  Hadamard matrix function
161  *  Implementation contributed by Remy Muller
162  ***************************************************************/
163
164 // bus(n) : n parallel cables
165 bus(2) = _,_; // avoids a lot of "bus(1)" labels in block diagrams
166 bus(n) = par(i, n, _);
167
168 // selector(i,n) : select ith cable among n
169 selector(i,n) = par(j, n, S(i, j)) with { S(i,i) = _; S(i,j) = !; };
170
171 // interleave(m,n) : interleave m*n cables : x(0), x(m), x(2m), ..., x(1),x(1+m), x(1+2m)...
172 //interleave(m,n) = bus(m*n) <: par(i, m, par(j, n, selector(i+j*m,m*n)));
173
174 // interleave(row,col) : interleave row*col cables from column order to row order.
175 // input : x(0), x(1), x(2) ..., x(row*col-1)
176 // output: x(0+0*row), x(0+1*row), x(0+2*row), ..., x(1+0*row), x(1+1*row), x(1+2*row), ...
177 interleave(row,col) = bus(row*col) <: par(r, row, par(c, col, selector(r+c*row,row*col)));
178
179 // butterfly(n) : addition then substraction of interleaved signals :
180 butterfly(n) = bus(n) <: interleave(n/2,2), interleave(n/2,2) : par(i, n/2, +), par(i, n/2,
      -);
181
182 // hadamard(n) : hadamard matrix function of size n = 2^k
```

```
183 hadamard(2) = butterfly(2);
184 hadamard(n) = butterfly(n) : (hadamard(n/2) , hadamard(n/2));
```