DSG

Generated by Doxygen 1.8.8

Sun Nov 23 2014 21:27:48

Contents

Chapter 1

Todo List

Class DSG::BLIT::Blit
Re-write DSG::BLIT::Blit algorithm

Class DSG::BLIT::BlitSaw
Re-write DSG::BLIT::BlitSaw algorithm

Class DSG::BLIT::BlitSquare
Write DSG::BLIT::BlitSquare algorithm

Class DSG::BLIT::BlitTriangle
Write DSG::BLIT::BlitTriangle algorithm

Class DSG::DPW::DPW_Differentiator< order >
Fix DSG::DPW::DPW_Differentiator algorithms for orders 3-6

Class DSG::EPTR::EPTRSaw
Test and Possibly Re-Write DSG::EPTR::EPTRSaw algorithm

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 DSG Namespace Reference

DSG - A Collection of tools for Digital Signal Generation.

Namespaces

- Analog

  DSG::Analog - Namespace Containing Analog Style Oscillators.
- BLIT

  DSG::BLIT - Namespace Containing BLIT Based Oscillators.
- DPW

  DSG::DPW - Generators using the DPW method.
- EPTR

  DSG::EPTR - Generators Based On The Efficienct Polynomial Transfer Region Algorithm.
- Filter

  DSG::Filter - Filters.
- Fourier

  DSG::Fourier - Namespace Containing Fourier Series Based Oscillators.
- Noise

  DSG::Noise - Noise Generators.
- Window

  DSG::Window - Window functions and utilities.

Classes

- class AudioSettings

  DSG::AudioSettings - Global Storage For Audio Settings Such As Sample Rate.
- class Buffer

  DSG::Buffer - Base Class For DSG::RingBuffer. Not For Direct Use.
- class Delay

  DSG::Delay - General purpose delay line.
- struct Factorial

  DSG::Factorial - Compute integer factorial.
- struct Factorial< 0 >

  DSG::Factorial - Compute integer factorial.
- class GenericGenerator

DSG::GenericGenerator - Generator designed to use a stateless generator function such as DSG::Sin()

- class LUT

  DSG::LUT - Look Up Table.
- class NoiseGenerator

  DSG::NoiseGenerator - Generator that uses noise functions such as DSG::White() to generate signal.
- class RingBuffer

  DSG::RingBuffer - Circular Buffer of Audio.
- class SignalGenerator

  DSG::SignalGenerator - Extends DSG::Signal Process With Tools For Signal Generation.
- class SignalProcess

  DSG::SignalProcess - Defines Base Interface For Audio Processing.

Typedefs

- typedef float DSGFrequency

  DSG::DSGFrequency - Type for representing a frequency value.
- typedef float DSGPhase

  DSG::DSGFrequency - Type for representing a phase value.
- typedef float DSGSample

  DSG::DSGFrequency - Type for representing an audio sample.

Functions

- DSG::DSGFrequency const & SampleRate ()

  DSG::SampleRate - Get Global Sample Rate.
- DSG::DSGFrequency const & SampleRate (DSG::DSGFrequency const &value)

  DSG::SampleRate - Set Global Sample Rate.
- DSG::DSGFrequency Nyquist ()

  DSG::Nyquist() - Pre-Calculated Nyquist Limit. Use instead of calculating each time needed. This value will be updated whenever the sample rate changes.
- template<int lower, int upper, typename decimal >
  decimal EnforceBounds (decimal const &value)

  DSG::EnforceBounds - Clip value to set bounds.
- template<int lower, int upper, int value>
  void StaticAssertBounds ()

  DSG::StaticAssertBounds - Fails on compile time if value is not within bounds.
- template<int lower, int upper, typename T >
  void AssertBounds (T const &value)

  DSG::AssertBounds - Fails on runtime if value is not within bounds.
- bool RingToArray (DSG::RingBuffer &ring, DSG::DSGSample *array, unsigned long length)

  DSG::RingToArray - Move Ring Buffer data to an array.
- bool ArrayToRing (DSG::RingBuffer &ring, DSG::DSGSample *array, unsigned long length)

  DSG::ArrayToRing - Move array data to a Ring Buffer.
- template<typename T >
  bool IsDenormal (T const &value)

  DSG::IsDenormal - Returns True if number is Denormal.
- template<typename decimal = DSG::DSGSample>
  decimal DSF (decimal const &beta, decimal const &theta, decimal const &N, decimal const &a)
- template<typename T >
  T Abs (T const &value)

  DSG::Abs - Calculate absolute value.

- template<unsigned exponent, class T >
  T constexpr Pow (T const base)

  DSG::Pow - Any type to an integer power, i.e. N^ I.
- template<typename decimal >
  decimal LinearInterpolate (decimal const &y1, decimal const &y2, decimal const &mu)

  DSG::LinearInterpolate - Linear Interpolation.
- template<typename decimal >
  decimal CosineInterpolate (decimal y1, decimal y2, decimal mu)

  DSG::CosineInterpolate - Cosine Interpolation.
- template<typename decimal >
  decimal CubicInterpolate (decimal const &y0, decimal const &y1, decimal const &y2, decimal const &y3, decimal const &mu)

  DSG::CubicInterpolate - Cubic Interpolation.
- template<typename decimal >
  decimal HermiteInterpolate (decimal const &y0, decimal const &y1, decimal const &y2, decimal const &y3, decimal const &mu, decimal const &tension, decimal const &bias)

  DSG::HermiteInterpolate - Hermite Interpolation.
- unsigned long MaxHarms (DSG::DSGFrequency const &frequency)
- template<typename decimal >
  decimal Sinc (decimal const &x)

  DSG::Sinc - Implements the Sinc() function (sin(PI∗x)/PI∗x)
- double Sin (double const &x)

  DSG::Sin() - General Purpose Sin Function, double precision.
- float Sin (float const &x)

  DSG::Sin() - General Purpose Sin Function, single precision.
- double Cos (double const &x)

  DSG::Cos() - General Purpose Cos Function, double precision.
- float Cos (float const &x)

  DSG::Cos() - General Purpose Cos Function, single precision.
- template<typename integer >
  void Sleep (integer const &milliseconds)

  DSG::Sleep - Millisecond Sleep Function.

6.1.1 Detailed Description

DSG - A Collection of tools for Digital Signal Generation.

6.1.2 Typedef Documentation

6.1.2.1 typedef float DSG::DSGFrequency

DSG::DSGFrequency - Type for representing a frequency value.

Definition at line 12 of file DSGTypes.h.

6.1.2.2 typedef float DSG::DSGPhase

DSG::DSGFrequency - Type for representing a phase value.

Definition at line 14 of file DSGTypes.h.

6.1.2.3 typedef float DSG::DSGSample

DSG::DSGFrequency - Type for representing an audio sample.

Definition at line 16 of file DSGTypes.h.

6.1.3 Function Documentation

6.1.3.1 template<typename T > T DSG::Abs (T const & *value*) [inline]

DSG::Abs - Calculate absolute value.

Definition at line 15 of file DSGMath.h.

```
00015                                 {
00016         return value < 0.0 ? -1.0 * value : value;
00017     }
```

6.1.3.2 bool DSG::ArrayToRing (DSG::RingBuffer & *ring*, DSG::DSGSample * *array*, unsigned long *length*) [inline]

DSG::ArrayToRing - Move array data to a Ring Buffer.

Definition at line 21 of file BufferConversion.h.

```
00021                                                                                                 {
00022         int i=0;
00023         ring.Flush();
00024         while (!ring.Full()) {
00025             ring.Write(array[i]);
00026             ++i;
00027         }return true;
00028     }
```

6.1.3.3 template<int lower, int upper, typename T > void DSG::AssertBounds (T const & *value*)

DSG::AssertBounds - Fails on runtime if value is not within bounds.

Definition at line 28 of file Bounds.h.

```
00028                                     {
00029         assert(value>=lower && value<=upper);
00030     }
```

6.1.3.4 double DSG::Cos (double const & *x*) [inline]

DSG::Cos() - General Purpose Cos Function, double precision.

Definition at line 60 of file Sine.h.

```
00060                                      {
00061         return static_cast<double>(Cos<Sine_Default>(x));//wrap default implementation as non template
00062     }
```

6.1.3.5 float DSG::Cos (float const & *x*) [inline]

DSG::Cos() - General Purpose Cos Function, single precision.

Definition at line 65 of file Sine.h.

```
00065                                       {
00066         return static_cast<float>(Cos<Sine_Default>(x));
00067     }
```

6.1.3.6 template<typename decimal > decimal DSG::CosineInterpolate (decimal *y1,* decimal *y2,* decimal *mu*)

DSG::CosineInterpolate - Cosine Interpolation.

Definition at line 23 of file Interpolate.h.

```
00026     {
00027         decimal mu2;
00028         mu2 = (1-cos(mu*PI))/2.0;
00029         return(y1*(1-mu2)+y2*mu2);
00030     }
```

6.1.3.7 template<typename decimal > decimal DSG::CubicInterpolate (decimal const & *y0,* decimal const & *y1,* decimal const & *y2,* decimal const & *y3,* decimal const & *mu*)

DSG::CubicInterpolate - Cubic Interpolation.

Definition at line 33 of file Interpolate.h.

```
00036     {
00037         decimal a0,a1,a2,a3,mu2;
00038         mu2 = mu*mu;
00039         a0 = y3 - y2 - y0 + y1;
00040         a1 = y0 - y1 - a0;
00041         a2 = y2 - y0;
00042         a3 = y1;
00043         return(a0*mu*mu2+a1*mu2+a2*mu+a3);
00044     }
```

6.1.3.8 template<typename decimal = DSG::DSGSample> decimal DSG::DSF (decimal const & *beta,* decimal const & *theta,* decimal const & *N,* decimal const & *a*)

Definition at line 14 of file DSF.h.

```
00014                                                                                                {
00015 #warning Untested DSG::DSF()
00016         decimal denominator = 1 + DSG::Pow<2>(a) - (2.0*a*cos(beta));
00017         decimal numerator = sin(theta) - a * sin(theta-beta) - pow(a, N+1) * (sin(theta + (N+1)*beta) - a*
      sin(theta + (N*beta)));
00018         return numerator/denominator;
00019     }
```

6.1.3.9 template<int lower, int upper, typename decimal > decimal DSG::EnforceBounds (decimal const & *value*)

DSG::EnforceBounds - Clip value to set bounds.

Definition at line 14 of file Bounds.h.

```
00014                                                {
00015         if (value<lower) {
00016             return lower;
00017         }else if(value> upper){
00018             return upper;
00019         }else return value;
00020     }
```

6.1.3.10 template<typename decimal > decimal DSG::HermiteInterpolate (decimal const & *y0*, decimal const & *y1*, decimal const & *y2*, decimal const & *y3*, decimal const & *mu*, decimal const & *tension*, decimal const & *bias*)

DSG::HermiteInterpolate - Hermite Interpolation.

Definition at line 47 of file Interpolate.h.

```
00052     {
00053         /*
00054          Tension: 1 is high, 0 normal, -1 is low
00055          Bias: 0 is even,
00056          positive is towards first segment,
00057          negative towards the other
00058          */
00059         decimal m0,m1,mu2,mu3;
00060         decimal a0,a1,a2,a3;
00061         mu2 = mu * mu;
00062         mu3 = mu2 * mu;
00063         m0  = (y1-y0)*(1+bias)*(1-tension)/2.0;
00064         m0 += (y2-y1)*(1-bias)*(1-tension)/2.0;
00065         m1  = (y2-y1)*(1+bias)*(1-tension)/2.0;
00066         m1 += (y3-y2)*(1-bias)*(1-tension)/2.0;
00067         a0 =  2*mu3 - 3*mu2 + 1;
00068         a1 =    mu3 - 2*mu2 + mu;
00069         a2 =    mu3 -   mu2;
00070         a3 = -2*mu3 + 3*mu2;
00071         return(a0*y1+a1*m0+a2*m1+a3*y2);
00072     }
```

6.1.3.11 template<typename T > bool DSG::IsDenormal (T const & *value*) [inline]

DSG::IsDenormal - Returns True if number is Denormal.

Definition at line 15 of file Denormal.h.

```
00015                                         {
00016         return DSG::Abs(value)<=std::numeric_limits<T>::epsilon();//return true if number is
      denormal
00017     }
```

6.1.3.12 template<typename decimal > decimal DSG::LinearInterpolate (decimal const & *y1*, decimal const & *y2*, decimal const & *mu*)

DSG::LinearInterpolate - Linear Interpolation.

Definition at line 18 of file Interpolate.h.

```
00018                                                                                   {
00019         return(y1*(1-mu)+y2*mu);
00020     }
```

6.1.3.13 unsigned long DSG::MaxHarms (DSG::DSGFrequency const & *frequency*) [inline]

Definition at line 40 of file SignalGenerator.h.

```
00040                                                                {
00041         double _s = DSG::SampleRate()*  20000.0/DSG::SampleRate();
00042         _s/=frequency;
00043         return _s;
00044     }
```

6.1.3.14 DSG::DSGFrequency DSG::Nyquist ( ) [inline]

DSG::Nyquist() - Pre-Calculated Nyquist Limit. Use instead of calculating each time needed. This value will be updated whenever the sample rate changes.

Definition at line 32 of file AudioSettings.h.

```
00032                                    {
00033         return DSG::AudioSettings::Nyquist();
00034     }
```

6.1.3.15 template<unsigned exponent, class T > T constexpr DSG::Pow (T const *base*)

DSG::Pow - Any type to an integer power, i.e. N ^ I.

Definition at line 44 of file DSGMath.h.

```
00044                                   {
00045         return power<T, exponent>::value(base);
00046     }
```

6.1.3.16 bool DSG::RingToArray (DSG::RingBuffer & *ring*, DSG::DSGSample * *array*, unsigned long *length*) [inline]

DSG::RingToArray - Move Ring Buffer data to an array.

Definition at line 13 of file BufferConversion.h.

```
00013                                                                                                {
00014         for (int i=0; i<length; ++i) {
00015             if (!ring.Empty()) {
00016                 ring.Read(array[i]);
00017             }
00018         }return true;
00019     }
```

6.1.3.17 DSG::DSGFrequency const& DSG::SampleRate ( ) [inline]

DSG::SampleRate - Get Global Sample Rate.

Definition at line 24 of file AudioSettings.h.

```
00024                                           {
00025         return DSG::AudioSettings::SampleRate();
00026     }
```

6.1.3.18 DSG::DSGFrequency const& DSG::SampleRate (DSG::DSGFrequency const & *value*) [inline]

DSG::SampleRate - Set Global Sample Rate.

Definition at line 28 of file AudioSettings.h.

```
00028                                                                                                {
00029         return DSG::AudioSettings::SampleRate(value);
00030     }
```

6.1.3.19 double DSG::Sin (double const & *x*) [inline]

DSG::Sin() - General Purpose Sin Function, double precision.

Definition at line 50 of file Sine.h.

```
00050                                       {
00051         return static_cast<double>(Sin<Sine_Default>(x));//wrap default implementation as non template
00052     }
```

6.1.3.20 float DSG::Sin (float const & *x*) [inline]

DSG::Sin() - General Purpose Sin Function, single precision.

Definition at line 55 of file Sine.h.

```
00055                                      {
00056         return static_cast<float>(Sin<Sine_Default>(x));
00057     }
```

6.1.3.21 template<typename decimal > decimal DSG::Sinc (decimal const & *x*) [inline]

DSG::Sinc - Implements the Sinc() function (sin(PI∗x)/PI∗x)

Definition at line 18 of file Sinc.h.

```
00018                                          {
00019         static_assert(std::is_floating_point<decimal>::value==true,"DSG::Sinc Function Requires Floating
      Point Type");
00020         decimal pix;
00021         if (DSG::IsDenormal(x)) {
00022             return 1.0;
00023         }else{
00024             pix = PI*x;
00025             return DSG::Sin(pix)/pix;
00026         }
00027     }
```

6.1.3.22 template<typename integer > void DSG::Sleep (integer const & *milliseconds*)

DSG::Sleep - Millisecond Sleep Function.

Definition at line 15 of file Sleep.h.

```
00015                                           {
00016         std::this_thread::sleep_for(std::chrono::milliseconds(milliseconds));
00017     }
```

6.1.3.23 template<int lower, int upper, int value> void DSG::StaticAssertBounds ( )

DSG::StaticAssertBounds - Fails on compile time if value is not within bounds.

Definition at line 23 of file Bounds.h.

```
00023                                {
00024         static_assert(value>=lower && value<=upper,"Failed Static Bounds Assert");
00025     }
```

6.2 DSG::Analog Namespace Reference

DSG::Analog - Namespace Containing Analog Style Oscillators.

Classes

- class AnalogSaw

  DSG::Analog::AnalogSaw - Analog Syle Saw Wave Generator.
- class AnalogSquare

  DSG::Analog::AnalogSquare - Analog Syle Square Wave Generator.
- class AnalogTriangle

  DSG::Analog::AnalogTriangle - Analog Syle Triangle Wave Generator.

6.2.1 Detailed Description

DSG::Analog - Namespace Containing Analog Style Oscillators.

6.3 DSG::BLIT Namespace Reference

DSG::BLIT - Namespace Containing BLIT Based Oscillators.

Classes

- class Blit

  DSG::BLIT::Blit - Band-Limited Impulse Train Generator.
- class BlitSaw

  DSG::BLIT::BlitSaw - Saw Wave Generator Based on BLIT Algorithm.
- class BlitSquare
- class BlitTriangle

6.3.1 Detailed Description

DSG::BLIT - Namespace Containing BLIT Based Oscillators.

6.4 DSG::DPW Namespace Reference

DSG::DPW - Generators using the DPW method.

Classes

- class DPW_Differentiator

  DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the DPW Algorithm.
- class DPW_Differentiator< 1 >

  DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 1st order DPW Algorithm.
- class DPW_Differentiator< 2 >

  DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 2nd order DPW Algorithm.
- class DPW_Differentiator< 3 >

  DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 3rd order DPW Algorithm.
- class DPW_Differentiator< 4 >

  DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 4th order DPW Algorithm.
- class DPW_Differentiator< 5 >

  DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 5th order DPW Algorithm.

- class DPW_Differentiator< 6 >

  DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 6th order DPW Algorithm.
- class DPWSaw

  DSG::DPW::DPWSaw - Sawtooth Generator using the Nth Order DPW algorithm.

Functions

- template<unsigned order>
  DSG::DSGSample DPW_Polynomial (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DPW_Polynomial< 1 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 1st Order Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DPW_Polynomial< 2 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 2nd order Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DPW_Polynomial< 3 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 3rd order Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DPW_Polynomial< 4 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 4th order Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DPW_Polynomial< 5 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 5th order Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DPW_Polynomial< 6 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 6th order Polynoimal used in DPW Algorithm.

6.4.1 Detailed Description

DSG::DPW - Generators using the DPW method.

6.4.2 Function Documentation

6.4.2.1 template<unsigned order> DSG::DSGSample DSG::DPW::DPW_Polynomial (DSG::DSGSample const & *value*) [inline]

DSG::DPW::DPW_Polynomial - Polynoimal used in DPW Algorithm.

Definition at line 22 of file DPW.h.

```
00022                                                                   {
00023             DSG::StaticAssertBounds<1,6,order>();//must be 1-6 order
00024             return value;
00025         }
```

6.4.2.2 template<> DSG::DSGSample DSG::DPW::DPW_Polynomial< 1 > (DSG::DSGSample const & *value*) [inline]

DSG::DPW::DPW_Polynomial - 1st Order Polynoimal used in DPW Algorithm.

Definition at line 28 of file DPW.h.

```
00028                                                                   {
00029             return value;
00030         }
```

6.4.2.3 template<> DSG::DSGSample DSG::DPW::DPW_Polynomial< 2 >(DSG::DSGSample const & *value*) [inline]

DSG::DPW::DPW_Polynomial - 2nd order Polynoimal used in DPW Algorithm.

Definition at line 33 of file DPW.h.

```
00033                                                                   {
00034             return DSG::Pow<2>(value);
00035         }
```

6.4.2.4 template<> DSG::DSGSample DSG::DPW::DPW_Polynomial< 3 >(DSG::DSGSample const & *value*) [inline]

DSG::DPW::DPW_Polynomial - 3rd order Polynoimal used in DPW Algorithm.

Definition at line 38 of file DPW.h.

```
00038                                                                   {
00039             return DSG::Pow<3>(value)-value;
00040         }
```

6.4.2.5 template<> DSG::DSGSample DSG::DPW::DPW_Polynomial< 4 >(DSG::DSGSample const & *value*) [inline]

DSG::DPW::DPW_Polynomial - 4th order Polynoimal used in DPW Algorithm.

Definition at line 43 of file DPW.h.

```
00043                                                                   {
00044             return DSG::Pow<2>(value) * (DSG::Pow<2>(value) - 2.0);
00045         }
```

6.4.2.6 template<> DSG::DSGSample DSG::DPW::DPW_Polynomial< 5 >(DSG::DSGSample const & *value*) [inline]

DSG::DPW::DPW_Polynomial - 5th order Polynoimal used in DPW Algorithm.

Definition at line 48 of file DPW.h.

```
00048                                                                   {
00049             return DSG::Pow<5>(value) - DSG::Pow<3>(value) * 10.0/3.0 + value * 7.0/3.0;
00050         }
```

6.4.2.7 template<> DSG::DSGSample DSG::DPW::DPW_Polynomial< 6 >(DSG::DSGSample const & *value*) [inline]

DSG::DPW::DPW_Polynomial - 6th order Polynoimal used in DPW Algorithm.

Definition at line 53 of file DPW.h.

```
00053                                                                   {
00054             return DSG::Pow<6>(value) - 5.0 * DSG::Pow<4>(value) + 7.0 *
      DPW_Polynomial<2>(value);
00055         }
```

6.5 DSG::EPTR Namespace Reference

DSG::EPTR - Generators Based On The Efficienct Polynomial Transfer Region Algorithm.

Classes

- class EPTRSaw

  DSG::EPTR::EPTRSaw-Sawtooth Wave Generator Using The Efficienct Polynomial Transfer Region Algorithm.

6.5.1 Detailed Description

DSG::EPTR - Generators Based On The Efficienct Polynomial Transfer Region Algorithm.

6.6 DSG::Filter Namespace Reference

DSG::Filter - Filters.

Classes

- class DCBlocker

  DSG::Filter::DCBlocker - DC blocking filter.
- class FilterBase

  DSG::Filter::FilterBase - Filter Base Class, implements interface for cutoff frequency.
- class LeakyIntegrator

  DSG::Filter::LeakyIntegrator - Leaky integrator.

6.6.1 Detailed Description

DSG::Filter - Filters.

6.7 DSG::Fourier Namespace Reference

DSG::Fourier - Namespace Containing Fourier Series Based Oscillators.

Classes

- class FourierSaw

  DSG::Fourier::FourierSaw - Fourier Series Sawtooth Wave Generator.
- class FourierSeriesGenerator

  DSG::Fourier::FourierSeriesGenerator - Generates a wave form using a user specified Fourier Series.
- class FourierSquare

  DSG::Fourier::FourierSquare - Fourier Series Square Wave Generator.
- class FourierTriangle

  DSG::Fourier::FourierTriangle - Fourier Series Triangle Wave Generator.
- class Harmonic

  DSG::Fourier::Harmonic - Represents a single harmonic in a Fourier Series.

6.7.1 Detailed Description

DSG::Fourier - Namespace Containing Fourier Series Based Oscillators.

6.8 DSG::Noise Namespace Reference

DSG::Noise - Noise Generators.

Functions

- template<typename decimal = DSG::DSGSample>
  decimal Gaussian (decimal=0.0)

  DSG::Noise::Gaussian - Gaussian Noise Generator Function.
- template<typename decimal = DSG::DSGSample>
  decimal Pink (decimal=0.0)

  DSG::Noise::Pink - Pink Noise Generator Function.
- template<typename decimal = DSG::DSGSample>
  decimal Random (decimal=0.0)

  DSG::Noise::Random - Random Number Function.
- template<typename decimal = DSG::DSGSample>
  decimal White (decimal=0.0)

  DSG::Noise::White - White Noise Generator Function.

6.8.1 Detailed Description

DSG::Noise - Noise Generators.

6.8.2 Function Documentation

6.8.2.1 template<typename decimal = DSG::DSGSample> decimal DSG::Noise::Gaussian (decimal = 0.0)

DSG::Noise::Gaussian - Gaussian Noise Generator Function.

Definition at line 19 of file Gaussian.h.

```
00019                                   {
00020             static decimal normalizer=1;//variable used to actively normalize the output
00021             //to enforce compatability with DSG::LUT a dummy parameter is applied
00022             //this parameter is useless except for compatability reasons
00023             decimal R1 = DSG::Noise::White();
00024             decimal R2 = DSG::Noise::White();
00025             decimal x= (decimal)sqrt(-2.0f * log(R1))*DSG::Cos(R2);
00026             if (DSG::Abs(x)>normalizer) {
00027                 //store highest output
00028                 normalizer=DSG::Abs(x);
00029             }
00030             x/=normalizer;//normalize
00031             return x;
00032         }
```

6.8.2.2 template<typename decimal = DSG::DSGSample> decimal DSG::Noise::Pink (decimal = 0.0)

DSG::Noise::Pink - Pink Noise Generator Function.

Definition at line 19 of file Pink.h.

```
00019                                {
00020             //routine: Get white or gaussian, filter, return
00021             static decimal b0,b1,b2,b3,b4,b5,b6;
00022             static decimal normalizer=1;//variable used to actively normalize the output
00023             static DSG::DCBlocker _block;
00024             decimal white = DSG::Noise::Gaussian();
00025             decimal pink;
00026             //pinking filter
00027             b0 = 0.99886 * b0 + white * 0.0555179;
00028             b1 = 0.99332 * b1 + white * 0.0750759;
00029             b2 = 0.96900 * b2 + white * 0.1538520;
00030             b3 = 0.86650 * b3 + white * 0.3104856;
00031             b4 = 0.55000 * b4 + white * 0.5329522;
00032             b5 = -0.7616 * b5 - white * 0.0168980;
00033             pink = b0 + b1 + b2 + b3 + b4 + b5 + b6 + white * 0.5362;
00034             b6 = white * 0.115926;
00035             if (DSG::Abs(pink)>normalizer) {
00036                 //store highest output
00037                 normalizer=DSG::Abs(pink);
00038             }
00039             pink/=normalizer;
00040             _block.Perform(pink);
00041             return pink;
00042         }
```

6.8.2.3 template<typename decimal = DSG::DSGSample> decimal DSG::Noise::Random (decimal = 0.0) [inline]

DSG::Noise::Random - Random Number Function.

Definition at line 34 of file Random.h.

```
00034                                      {
00035             static DSG::Noise::random_helper<decimal> _rand{};
00036             return _rand.next();
00037         }
```

6.8.2.4 template<typename decimal = DSG::DSGSample> decimal DSG::Noise::White (decimal = 0.0) [inline]

DSG::Noise::White - White Noise Generator Function.

Definition at line 19 of file White.h.

```
00019                                      {
00020             return DSG::Random<decimal>();
00021         }
```

6.9 DSG::Window Namespace Reference

DSG::Window - Window functions and utilities.

Functions

- template<typename decimal >
  decimal Blackman (decimal const &x)

  DSG::Window::Blackman - Blackman Window Function.
- template<typename decimal , unsigned long lutsize>
  void ApplyWindow (DSG::LUT< decimal, lutsize > &lut, decimal(&windowFunction)(decimal const &), decimal range=1.0)

  DSG::Window::ApplyWindow - Apply a window function to a LUT.
- template<typename decimal , unsigned long lutsize>
  void ApplyWindow (DSG::LUT< decimal, lutsize > &lut, decimal(&windowFunction)(decimal), decimal range=1.0)

  DSG::Window::ApplyWindow - Apply a window function to a LUT.

6.9.1 Detailed Description

DSG::Window - Window functions and utilities.

6.9.2 Function Documentation

6.9.2.1 template<typename decimal , unsigned long lutsize> void DSG::Window::ApplyWindow (DSG::LUT< decimal, lutsize > & *lut*, decimal(&)(decimal const &) *windowFunction*, decimal *range* = 1.0)

DSG::Window::ApplyWindow - Apply a window function to a LUT.

Definition at line 19 of file Window.h.

```
00019
            {
00020            decimal step = range/(decimal)lut.Size();
00021            decimal phs=0;
00022            for (int i=0; i<lut.Size(); ++i) {
00023                lut[i]*=windowFunction(phs);
00024                phs+=step;
00025            }
00026        }
```

6.9.2.2 template<typename decimal , unsigned long lutsize> void DSG::Window::ApplyWindow (DSG::LUT< decimal, lutsize > & *lut*, decimal(&)(decimal) *windowFunction*, decimal *range* = 1.0)

DSG::Window::ApplyWindow - Apply a window function to a LUT.

Definition at line 29 of file Window.h.

```
00029
       {
00030            decimal step = range/(decimal)lut.Size();
00031            decimal phs=0;
00032            for (int i=0; i<lut.Size(); ++i) {
00033                lut[i]*=windowFunction(phs);
00034                phs+=step;
00035            }
00036        }
```

6.9.2.3 template<typename decimal > decimal DSG::Window::Blackman (decimal const & *x*) [inline]

DSG::Window::Blackman - Blackman Window Function.

Definition at line 20 of file Blackman.h.

```
00020                                                {
00021            // Generate Blackman Window
00022            /*
00023             Blackman(x) = 0.42f - (0.5f * cos(2pi*x)) + (0.08f * cos(2pi*2.0*x));
00024             }*/
00025            static_assert(std::is_floating_point<decimal>::value==true,"DSG::Blackman Function Requires
     Floating Point Type");
00026            //we will implement the blackman window as a function as if it were sin(x)
00027            //cos input domain 0-1 not 0-2pi
00028            //range checking is handles within DSG::Cos
00029            decimal phs=x;
00030            while (phs>1.0) {
00031                phs-=1.0;
00032            }
00033            return 0.42 - (0.5 * DSG::Cos(phs))+(0.08 * DSG::Cos(2.0*phs));
00034        }
```

Chapter 7

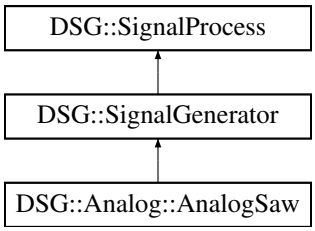
Class Documentation

7.1 DSG::Analog::AnalogSaw Class Reference

DSG::Analog::AnalogSaw - Analog Syle Saw Wave Generator.

`#include <AnalogSaw.h>`

Inheritance diagram for DSG::Analog::AnalogSaw:



Public Member Functions

- AnalogSaw ()
- AnalogSaw (DSG::DSGFrequency const &frequency, DSG::DSGPhase const &offset)
- virtual ~AnalogSaw ()
- virtual bool Perform (DSG::DSGSample &signal)
- virtual bool Perform (DSG::RingBuffer &signal)

Protected Attributes

- DSG::DSGSample _stor

Additional Inherited Members

7.1.1 Detailed Description

DSG::Analog::AnalogSaw - Analog Syle Saw Wave Generator.

Definition at line 18 of file AnalogSaw.h.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 DSG::Analog::AnalogSaw::AnalogSaw ( )

Definition at line 9 of file AnalogSaw.cpp.

```
00009 :DSG::SignalGenerator(){}
```

7.1.2.2 DSG::Analog::AnalogSaw::AnalogSaw (DSG::DSGFrequency const & *frequency,* DSG::DSGPhase const & *offset*)

Definition at line 10 of file AnalogSaw.cpp.

```
00010 :DSG::SignalGenerator(frequency,offset){}
```

7.1.2.3 DSG::Analog::AnalogSaw::~AnalogSaw ( ) [virtual]

Definition at line 11 of file AnalogSaw.cpp.

```
00011 {}
```

7.1.3 Member Function Documentation

7.1.3.1 bool DSG::Analog::AnalogSaw::Perform (DSG::DSGSample & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 28 of file AnalogSaw.h.

```
00028                                                                   {
00029             _stor=_phasor;
00030             _stor+=0.5;
00031             if (_stor>1.0) {
00032                 --_stor;
00033             }
00034             _stor-=0.5;
00035             _stor*=2.0;
00036             signal=_stor;
00037             step();
00038             return true;
00039         }
```

7.1.3.2 bool DSG::Analog::AnalogSaw::Perform (DSG::RingBuffer & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 40 of file AnalogSaw.h.

```
00040                                                                    {
00041             signal.Flush();
00042             while (!signal.Full()) {
00043                 if (Perform(_storage)) {
00044                     if(signal.Write(_storage)){
00045                     }else return false;
00046                 }else return false;
00047             }return true;
00048         }
```

7.1.4 Member Data Documentation

7.1.4.1 DSG::DSGSample DSG::Analog::AnalogSaw::_stor [protected]

Definition at line 26 of file AnalogSaw.h.

The documentation for this class was generated from the following files:

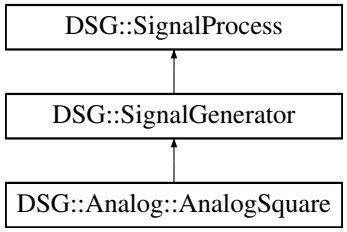
- /Users/alexanderzywicki/Documents/DSG/src/AnalogSaw.h
- /Users/alexanderzywicki/Documents/DSG/src/AnalogSaw.cpp

7.2 DSG::Analog::AnalogSquare Class Reference

DSG::Analog::AnalogSquare - Analog Syle Square Wave Generator.

`#include <AnalogSquare.h>`

Inheritance diagram for DSG::Analog::AnalogSquare:



Public Member Functions

- AnalogSquare ()
- AnalogSquare (DSG::DSGFrequency const &frequency, DSG::DSGPhase const &offset)
- virtual ~AnalogSquare ()
- virtual bool Perform (DSG::DSGSample &signal)
- virtual bool Perform (DSG::RingBuffer &signal)

Additional Inherited Members

7.2.1 Detailed Description

DSG::Analog::AnalogSquare - Analog Syle Square Wave Generator.

Definition at line 18 of file AnalogSquare.h.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 DSG::Analog::AnalogSquare::AnalogSquare ( )

Definition at line 9 of file AnalogSquare.cpp.

```
00009 :DSG::SignalGenerator(){}
```

7.2.2.2 DSG::Analog::AnalogSquare::AnalogSquare (DSG::DSGFrequency const & *frequency,* DSG::DSGPhase const & *offset*)

Definition at line 10 of file AnalogSquare.cpp.

```
00010 :DSG::SignalGenerator(frequency,offset){}
```

7.2.2.3 DSG::Analog::AnalogSquare::~AnalogSquare ( ) [virtual]

Definition at line 11 of file AnalogSquare.cpp.

```
00011 {}
```

7.2.3 Member Function Documentation

7.2.3.1 bool DSG::Analog::AnalogSquare::Perform (DSG::DSGSample & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 26 of file AnalogSquare.h.

```
00026                                                                   {
00027             signal=_phasor < 0.5 ? 1.0:-1.0;
00028             step();
00029             return true;
00030         }
```

7.2.3.2 bool DSG::Analog::AnalogSquare::Perform (DSG::RingBuffer & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 31 of file AnalogSquare.h.

```
00031                                                                    {
00032             signal.Flush();
00033             while (!signal.Full()) {
00034                 if (Perform(_storage)) {
00035                     if(signal.Write(_storage)){
00036                     }else return false;
00037                 }else return false;
00038             }return true;
00039         }
```

The documentation for this class was generated from the following files:

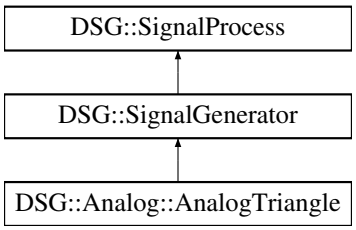
- /Users/alexanderzywicki/Documents/DSG/src/AnalogSquare.h
- /Users/alexanderzywicki/Documents/DSG/src/AnalogSquare.cpp

7.3 DSG::Analog::AnalogTriangle Class Reference

DSG::Analog::AnalogTriangle - Analog Syle Triangle Wave Generator.

`#include <AnalogTriangle.h>`

Inheritance diagram for DSG::Analog::AnalogTriangle:



Public Member Functions

- AnalogTriangle ()
- AnalogTriangle (DSG::DSGFrequency const &frequency, DSG::DSGPhase const &offset)
- virtual ~AnalogTriangle ()
- virtual bool Perform (DSG::DSGSample &signal)
- virtual bool Perform (DSG::RingBuffer &signal)

Protected Attributes

- DSG::DSGSample _stor

Additional Inherited Members

7.3.1 Detailed Description

DSG::Analog::AnalogTriangle - Analog Syle Triangle Wave Generator.

Definition at line 18 of file AnalogTriangle.h.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 DSG::Analog::AnalogTriangle::AnalogTriangle ( )

Definition at line 9 of file AnalogTriangle.cpp.

```
00009 :DSG::SignalGenerator(){}
```

7.3.2.2 DSG::Analog::AnalogTriangle::AnalogTriangle (DSG::DSGFrequency const & *frequency*, DSG::DSGPhase const & *offset*)

Definition at line 10 of file AnalogTriangle.cpp.

```
00010 :DSG::SignalGenerator(frequency,offset){}
```

7.3.2.3 DSG::Analog::AnalogTriangle::~AnalogTriangle ( ) [virtual]

Definition at line 11 of file AnalogTriangle.cpp.

```
00011 {}
```

7.3.3 Member Function Documentation

7.3.3.1 bool DSG::Analog::AnalogTriangle::Perform (DSG::DSGSample & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 28 of file AnalogTriangle.h.

```
00028                                                                   {
00029             _stor = _phasor;
00030             _stor+=0.25;
00031             while (_stor>1.0) {
00032                 _stor-=1.0;
00033             }
00034             _stor-=0.5;
00035             if (_stor<0) {
00036                 _stor*=-1.0;
00037             }
00038             _stor-=0.25;
00039             _stor*=-4.0;
00040             signal = _stor;
00041             step();//always last
00042             return true;
00043         }
```

7.3.3.2 bool DSG::Analog::AnalogTriangle::Perform (DSG::RingBuffer & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 44 of file AnalogTriangle.h.

```
00044                                                                    {
00045             signal.Flush();
00046             while (!signal.Full()) {
00047                 if (Perform(_storage)) {
00048                     if(signal.Write(_storage)){
00049                     }else return false;
00050                 }else return false;
00051             }return true;
00052         }
```

7.3.4 Member Data Documentation

7.3.4.1 DSG::DSGSample DSG::Analog::AnalogTriangle::_stor [protected]

Definition at line 26 of file AnalogTriangle.h.

The documentation for this class was generated from the following files:

- /Users/alexanderzywicki/Documents/DSG/src/AnalogTriangle.h
- /Users/alexanderzywicki/Documents/DSG/src/AnalogTriangle.cpp

7.4 DSG::AudioSettings Class Reference

DSG::AudioSettings - Global Storage For Audio Settings Such As Sample Rate.

`#include <AudioSettings.h>`

Static Public Member Functions

- static DSG::DSGFrequency const & SampleRate ()
- static DSG::DSGFrequency const & SampleRate (DSG::DSGFrequency const &value)
- static DSG::DSGFrequency const & Nyquist ()

Static Protected Attributes

- static DSG::DSGFrequency _sampleRate
- static DSG::DSGFrequency _nyquist

7.4.1 Detailed Description

DSG::AudioSettings - Global Storage For Audio Settings Such As Sample Rate.

Definition at line 14 of file AudioSettings.h.

7.4.2 Member Function Documentation

7.4.2.1 DSG::DSGFrequency const & DSG::AudioSettings::Nyquist ( ) [static]

Definition at line 19 of file AudioSettings.cpp.

```
00019                                                {
00020     return _nyquist;
00021 }
```

7.4.2.2 DSG::DSGFrequency const & DSG::AudioSettings::SampleRate ( ) [static]

Definition at line 11 of file AudioSettings.cpp.

```
00011                                                    {
00012     return _sampleRate;
00013 }
```

7.4.2.3 DSG::DSGFrequency const & DSG::AudioSettings::SampleRate (DSG::DSGFrequency const & *value*) [static]

Definition at line 14 of file AudioSettings.cpp.

```
00014                                                                                {
00015     _sampleRate = value;
00016     _nyquist = _sampleRate*0.5;
00017     return _sampleRate;
00018 }
```

7.4.3 Member Data Documentation

7.4.3.1 DSG::DSGFrequency DSG::AudioSettings::_nyquist [static], [protected]

Definition at line 21 of file AudioSettings.h.

7.4.3.2 DSG::DSGFrequency DSG::AudioSettings::_sampleRate [static], [protected]

Definition at line 20 of file AudioSettings.h.

The documentation for this class was generated from the following files:

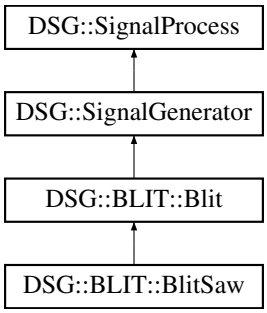
- /Users/alexanderzywicki/Documents/DSG/src/AudioSettings.h
- /Users/alexanderzywicki/Documents/DSG/src/AudioSettings.cpp

7.5 DSG::BLIT::Blit Class Reference

DSG::BLIT::Blit - Band-Limited Impulse Train Generator.

`#include <BLIT.h>`

Inheritance diagram for DSG::BLIT::Blit:



Public Member Functions

- Blit ()
- Blit (DSG::DSGFrequency const &frequency, DSG::DSGPhase const &offset)
- virtual ~Blit ()
- virtual bool Perform (DSG::DSGSample &signal)
- virtual bool Perform (DSG::RingBuffer &signal)
- virtual DSG::DSGFrequency const & Frequency (DSG::DSGFrequency const &value)

Protected Attributes

- unsigned long p_
- unsigned long m_
- unsigned long _h
- double a_
- DSG::DSGSample denominator
- DSG::DSGSample value

Additional Inherited Members

7.5.1 Detailed Description

DSG::BLIT::Blit - Band-Limited Impulse Train Generator.

Todo Re-write DSG::BLIT::Blit algorithm

Definition at line 23 of file BLIT.h.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 DSG::BLIT::Blit::Blit ( )

Definition at line 9 of file BLIT.cpp.

```
00009                   :DSG::SignalGenerator(){
00010     Frequency(0);
00011 }
```

7.5.2.2 DSG::BLIT::Blit::Blit (DSG::DSGFrequency const & *frequency,* DSG::DSGPhase const & *offset*)

Definition at line 12 of file BLIT.cpp.

```
00012                                                                                :
      DSG::SignalGenerator(frequency,offset){
00013     Frequency(frequency);
00014 }
```

7.5.2.3 DSG::BLIT::Blit::~Blit ( ) [virtual]

Definition at line 15 of file BLIT.cpp.

```
00015 {}
```

7.5.3 Member Function Documentation

7.5.3.1 DSG::DSGFrequency const & DSG::BLIT::Blit::Frequency (DSG::DSGFrequency const & *value*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Reimplemented in DSG::BLIT::BlitSaw.

Definition at line 62 of file BLIT.h.

```
00062                                                                                {
00063             this->SignalGenerator::Frequency(value);
00064             p_ = DSG::SampleRate()/_frequency;
00065             _h = (unsigned)floor(p_*0.5);
00066             m_ = 2 * (_h)+1;
00067             a_ = m_/(double)p_;
00068             return _frequency;
00069         }
```

7.5.3.2 bool DSG::BLIT::Blit::Perform (DSG::DSGSample & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Reimplemented in DSG::BLIT::BlitSaw.

Definition at line 39 of file BLIT.h.

```
00039                                                    {
00040             //found better results in this case with built in sine function. not performance wise but
      algorithmically
00041             denominator = m_ * sin(PI*_phasor);
00042             if (DSG::IsDenormal(denominator)) {
00043                 signal = a_;
00044             }else{
00045                 value = sin(PI*_phasor * m_);
00046                 value/=denominator;
00047                 value*=a_;
00048                 signal = value;
00049             }
00050             step();
00051             return true;
00052         }
```

7.5.3.3 bool DSG::BLIT::Blit::Perform (DSG::RingBuffer & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Reimplemented in DSG::BLIT::BlitSaw.

Definition at line 53 of file BLIT.h.

```
00053                                                                   {
00054             signal.Flush();
00055             while (!signal.Full()) {
00056                 if (Perform(_storage)) {
00057                     if(signal.Write(_storage)){
00058                     }else return false;
00059                 }else return false;
00060             }return true;
00061         }
```

7.5.4 Member Data Documentation

7.5.4.1 unsigned long DSG::BLIT::Blit::_h [protected]

Definition at line 34 of file BLIT.h.

7.5.4.2 double DSG::BLIT::Blit::a_ [protected]

Definition at line 35 of file BLIT.h.

7.5.4.3 DSG::DSGSample DSG::BLIT::Blit::denominator [protected]

Definition at line 36 of file BLIT.h.

7.5.4.4 unsigned long DSG::BLIT::Blit::m_ [protected]

Definition at line 33 of file BLIT.h.

7.5.4.5 unsigned long DSG::BLIT::Blit::p_ [protected]

Definition at line 32 of file BLIT.h.

7.5.4.6 DSG::DSGSample DSG::BLIT::Blit::value [protected]

Definition at line 37 of file BLIT.h.

The documentation for this class was generated from the following files:

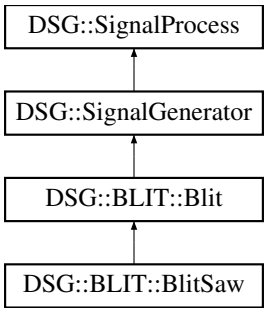
- /Users/alexanderzywicki/Documents/DSG/src/BLIT.h
- /Users/alexanderzywicki/Documents/DSG/src/BLIT.cpp

7.6 DSG::BLIT::BlitSaw Class Reference

DSG::BLIT::BlitSaw - Saw Wave Generator Based on BLIT Algorithm.

`#include <BLITSaw.h>`

Inheritance diagram for DSG::BLIT::BlitSaw:



Public Member Functions

- BlitSaw ()
- BlitSaw (DSG::DSGFrequency const &frequency, DSG::DSGPhase const &offset)
- virtual ~BlitSaw ()
- virtual bool Perform (DSG::DSGSample &signal)
- virtual bool Perform (DSG::RingBuffer &signal)
- virtual DSG::DSGFrequency const & Frequency (DSG::DSGFrequency const &value)

Protected Attributes

- DSG::DSGSample C2_
- DSG::DSGSample Register_

Additional Inherited Members

7.6.1 Detailed Description

DSG::BLIT::BlitSaw - Saw Wave Generator Based on BLIT Algorithm.

Todo Re-write DSG::BLIT::BlitSaw algorithm

Definition at line 18 of file BLITSaw.h.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 DSG::BLIT::BlitSaw::BlitSaw ( )

Definition at line 9 of file BLITSaw.cpp.

```
00009                       :DSG::BLIT::Blit(),Register_(0){
00010     Frequency(0);
00011 }
```

7.6.2.2 DSG::BLIT::BlitSaw::BlitSaw (DSG::DSGFrequency const & *frequency*, DSG::DSGPhase const & *offset*)

Definition at line 12 of file BLITSaw.cpp.

```
00012                                                                   :
      DSG::BLIT::Blit(frequency,offset),Register_(0){
00013     Frequency(frequency);
00014 }
```

7.6.2.3 DSG::BLIT::BlitSaw::~BlitSaw ( ) [virtual]

Definition at line 15 of file BLITSaw.cpp.

```
00015 {}
```

7.6.3 Member Function Documentation

7.6.3.1 DSG::DSGFrequency const & DSG::BLIT::BlitSaw::Frequency (DSG::DSGFrequency const & *value*) [inline], [virtual]

Reimplemented from DSG::BLIT::Blit.

Definition at line 56 of file BLITSaw.h.

```
00056                                                                                {
00057             this->SignalGenerator::Frequency(value);
00058             p_ = DSG::SampleRate()/_frequency;
00059             _h = (unsigned)floor(p_*0.5);
00060             m_ = 2 * (_h)+1;
00061             a_ = m_/(double)p_;
00062             C2_ = 1.0/(double)p_;
00063             return _frequency;
00064         }
```

7.6.3.2 bool DSG::BLIT::BlitSaw::Perform (DSG::DSGSample & *signal*) [inline], [virtual]

Reimplemented from DSG::BLIT::Blit.

Definition at line 30 of file BLITSaw.h.

```
00030                                                                  {
00031             denominator = m_ * sin(PI*_phasor);
00032             if (DSG::IsDenormal(denominator)) {
00033                 signal = a_;
00034             }else{
00035                 value = sin(PI*_phasor * m_);
00036                 value/=denominator;
00037                 value*=a_;
00038                 signal = value;
00039             }
00040             step();
00041             signal += (Register_ - C2_);
00042             Register_ = signal * 0.995;
00043             C2_+=signal;
00044             C2_*=0.5;
00045             return true;
00046         }
```

7.6.3.3 bool DSG::BLIT::BlitSaw::Perform (DSG::RingBuffer & *signal*) [inline], [virtual]

Reimplemented from DSG::BLIT::Blit.

Definition at line 47 of file BLITSaw.h.

```
00047                                                                   {
00048             signal.Flush();
00049             while (!signal.Full()) {
00050                 if (Perform(_storage)) {
00051                     if(signal.Write(_storage)){
00052                     }else return false;
00053                 }else return false;
00054             }return true;
00055         }
```

7.6.4 Member Data Documentation

7.6.4.1 DSG::DSGSample DSG::BLIT::BlitSaw::C2_ [protected]

Definition at line 27 of file BLITSaw.h.

7.6.4.2 DSG::DSGSample DSG::BLIT::BlitSaw::Register_ [protected]

Definition at line 28 of file BLITSaw.h.

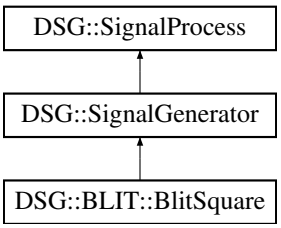
The documentation for this class was generated from the following files:

- /Users/alexanderzywicki/Documents/DSG/src/BLITSaw.h
- /Users/alexanderzywicki/Documents/DSG/src/BLITSaw.cpp

7.7 DSG::BLIT::BlitSquare Class Reference

`#include <BLITSquare.h>`

Inheritance diagram for DSG::BLIT::BlitSquare:



Additional Inherited Members

7.7.1 Detailed Description

Todo Write DSG::BLIT::BlitSquare algorithm

Definition at line 17 of file BLITSquare.h.

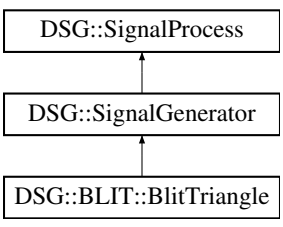
The documentation for this class was generated from the following file:

- /Users/alexanderzywicki/Documents/DSG/src/BLITSquare.h

7.8 DSG::BLIT::BlitTriangle Class Reference

`#include <BLITTriangle.h>`

Inheritance diagram for DSG::BLIT::BlitTriangle:



Additional Inherited Members

7.8.1 Detailed Description

Todo Write DSG::BLIT::BlitTriangle algorithm

Definition at line 19 of file BLITTriangle.h.

The documentation for this class was generated from the following file:

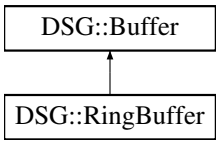
- /Users/alexanderzywicki/Documents/DSG/src/BLITTriangle.h

7.9 DSG::Buffer Class Reference

DSG::Buffer - Base Class For DSG::RingBuffer. Not For Direct Use.

`#include <Buffer.h>`

Inheritance diagram for DSG::Buffer:



Public Member Functions

- Buffer ()
- Buffer (size_t size)
- Buffer (Buffer const &other)
- Buffer & operator= (Buffer const &other)
- virtual ~Buffer ()
- DSG::DSGSample & operator[] (size_t const &index)
- size_t const & Size () const

Protected Attributes

- DSG::DSGSample * _buffer
- size_t _size

7.9.1 Detailed Description

DSG::Buffer - Base Class For DSG::RingBuffer. Not For Direct Use.

Definition at line 18 of file Buffer.h.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 DSG::Buffer::Buffer ( )

Definition at line 9 of file Buffer.cpp.

```
00009 :_size(0),_buffer(nullptr){}
```

7.9.2.2 DSG::Buffer::Buffer (size_t *size*)

Definition at line 10 of file Buffer.cpp.

```
00010 :_size(size),_buffer(new DSG::DSGSample[size]){}
```

7.9.2.3 DSG::Buffer::Buffer (Buffer const & *other*)

Definition at line 11 of file Buffer.cpp.

```
00011                                       {
00012     _buffer = new  DSG::DSGSample[_size];
00013     _size = other._size;
00014     *this = other;
00015 }
```

7.9.2.4 DSG::Buffer::~Buffer ( ) [virtual]

Definition at line 29 of file Buffer.cpp.

```
00029                    {
00030     if (_buffer!=nullptr) {
00031         delete [] _buffer;
00032     }
00033 }
```

7.9.3 Member Function Documentation

7.9.3.1 DSG::Buffer & DSG::Buffer::operator= (Buffer const & *other*)

Definition at line 16 of file Buffer.cpp.

```
00016                                                  {
00017     if (_size!=other._size) {
00018         if (_buffer!=nullptr) {
00019             delete [] _buffer;
00020         }
00021         _size = other._size;
00022         _buffer = new  DSG::DSGSample[_size];
00023     }
00024     for (int i=0; i<_size; ++i) {
00025         _buffer[i] = other._buffer[i];
00026     }
00027     return *this;
00028 }
```

7.9.3.2 DSG::DSGSample & DSG::Buffer::operator[] (size_t const & *index*)

Definition at line 34 of file Buffer.cpp.

```
00034                                                    {
00035 #ifdef DEBUG
00036     assert(index<_size);
00037 #endif
00038     return _buffer[index];
00039 }
```

7.9.3.3 size_t const & DSG::Buffer::Size ( ) const [inline]

Definition at line 31 of file Buffer.h.

```
00031                                             {
00032         return _size;
00033     }
```

7.9.4 Member Data Documentation

7.9.4.1 DSG::DSGSample* DSG::Buffer::_buffer [protected]

Definition at line 28 of file Buffer.h.

7.9.4.2 size_t DSG::Buffer::_size [protected]

Definition at line 29 of file Buffer.h.

The documentation for this class was generated from the following files:

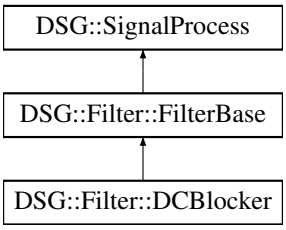
- /Users/alexanderzywicki/Documents/DSG/src/Buffer.h
- /Users/alexanderzywicki/Documents/DSG/src/Buffer.cpp

7.10 DSG::Filter::DCBlocker Class Reference

DSG::Filter::DCBlocker - DC blocking filter.

`#include <DCBlocker.h>`

Inheritance diagram for DSG::Filter::DCBlocker:



Public Member Functions

- DCBlocker ()
- virtual ~DCBlocker ()
- virtual bool Perform (DSG::DSGSample &signal)
- virtual bool Perform (DSG::RingBuffer &signal)

Protected Attributes

- unsigned long count
- DSG::DSGSample _temp
- DSG::DSGSample xm1
- DSG::DSGSample ym1
- DSG::DSGSample x
- DSG::DSGSample _a

7.10.1 Detailed Description

DSG::Filter::DCBlocker - DC blocking filter.

Definition at line 17 of file DCBlocker.h.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 DSG::Filter::DCBlocker::DCBlocker ( )

Definition at line 9 of file DCBlocker.cpp.

```
00009 :DSG::Filter::FilterBase(),_a(0.995),xm1(0),ym1(0),
      x(0),_temp(0){}
```

7.10.2.2 DSG::Filter::DCBlocker::~DCBlocker ( ) [virtual]

Definition at line 10 of file DCBlocker.cpp.

```
00010 {}
```

7.10.3 Member Function Documentation

7.10.3.1 bool DSG::Filter::DCBlocker::Perform (DSG::DSGSample & *signal*) [inline], [virtual]

Reimplemented from DSG::Filter::FilterBase.

Definition at line 31 of file DCBlocker.h.

```
00031                                                                    {
00032             x = signal;
00033             signal= x - xm1+ (_a * ym1);
00034             xm1 = x;
00035             ym1=signal;
00036             return true;
00037         }
```

7.10.3.2 bool DSG::Filter::DCBlocker::Perform (DSG::RingBuffer & *signal*) [inline], [virtual]

Reimplemented from DSG::Filter::FilterBase.

Definition at line 38 of file DCBlocker.h.

```
00038                                                                     {
00039             if (!signal.Empty()) {
00040                 count = signal.Count();
00041                 while (count-- > 0) {
00042                     if(signal.Read(_temp)){
00043                         if (Perform(_temp)) {
00044                             signal.Write(_temp);
00045                         }else return false;
00046                     }else return false;
00047                 }return true;
00048             }else return false;
00049         }
```

7.10.4 Member Data Documentation

7.10.4.1 DSG::DSGSample DSG::Filter::DCBlocker::_a [protected]

Definition at line 29 of file DCBlocker.h.

7.10.4.2 DSG::DSGSample DSG::Filter::DCBlocker::_temp [protected]

Definition at line 25 of file DCBlocker.h.

7.10.4.3 unsigned long DSG::Filter::DCBlocker::count [protected]

Definition at line 24 of file DCBlocker.h.

7.10.4.4 DSG::DSGSample DSG::Filter::DCBlocker::x [protected]

Definition at line 28 of file DCBlocker.h.

7.10.4.5 DSG::DSGSample DSG::Filter::DCBlocker::xm1 [protected]

Definition at line 26 of file DCBlocker.h.

7.10.4.6 DSG::DSGSample DSG::Filter::DCBlocker::ym1 [protected]

Definition at line 27 of file DCBlocker.h.

The documentation for this class was generated from the following files:

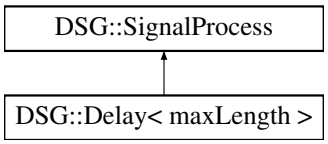
- /Users/alexanderzywicki/Documents/DSG/src/DCBlocker.h
- /Users/alexanderzywicki/Documents/DSG/src/DCBlocker.cpp

7.11 DSG::Delay< maxLength > Class Template Reference

DSG::Delay - General purpose delay line.

`#include <Delay.h>`

Inheritance diagram for DSG::Delay< maxLength >:



Public Member Functions

- Delay ()
- Delay (double const &samples)
- virtual ~Delay ()
- virtual unsigned long const & Length () const
- virtual unsigned long const & Length (unsigned long const &samples)
- virtual bool Perform (DSG::DSGSample &signal)
- virtual bool Perform (DSG::RingBuffer &signal)

Protected Member Functions

- virtual void increment ()

Protected Attributes

- unsigned long count
- unsigned long _delay
- unsigned long _index
- const unsigned long _max
- DSG::DSGSample _buffer [maxLength]
- DSG::DSGSample _swap
- DSG::DSGSample _temp

7.11.1 Detailed Description

template<unsigned long maxLength>class DSG::Delay< maxLength >

DSG::Delay - General purpose delay line.

Definition at line 17 of file Delay.h.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 template<unsigned long maxLength> DSG::Delay< maxLength >::Delay ( ) [inline]

Definition at line 19 of file Delay.h.

```
00019                 :DSG::SignalProcess(),_max(maxLength),_swap(0),
      _temp(0),count(0),_index(0),_delay(0){
00020             for (int i=0; i<_max; ++i) {
00021                 _buffer[i]=0;
00022             }
00023         }
```

7.11.2.2 template<unsigned long maxLength> DSG::Delay< maxLength >::Delay (double const & *samples*) [inline]

Definition at line 24 of file Delay.h.

```
00024                                     :DSG::SignalProcess(),_max(maxLength),
      _swap(0),_temp(0),count(0),_index(0),_delay(0){
00025             for (int i=0; i<_max; ++i) {
00026                 _buffer[i]=0;
00027             }
00028             if (samples>maxLength) {
00029                 _delay = maxLength;
00030             }else{
00031                 _delay = samples;
00032             }
00033         }
```

7.11.2.3 template<unsigned long maxLength> virtual DSG::Delay< maxLength >::~Delay ( ) [inline], [virtual]

Definition at line 34 of file Delay.h.

```
00034 {}
```

7.11.3 Member Function Documentation

7.11.3.1 template<unsigned long maxLength> virtual void DSG::Delay< maxLength >::increment ( ) [inline], [protected], [virtual]

Definition at line 56 of file Delay.h.

```
00056                                       {
00057             ++_index;
00058             if (_index>_delay) {
00059                 _index-=_delay;
00060             }
00061         }
```

7.11.3.2 template<unsigned long maxLength> virtual unsigned long const& DSG::Delay< maxLength >::Length ( ) const [inline], [virtual]

Definition at line 35 of file Delay.h.

```
00035                                                       {
00036             return _delay;
00037         }
```

7.11.3.3 template<unsigned long maxLength> virtual unsigned long const& DSG::Delay< maxLength >::Length (unsigned long const & *samples*) [inline], [virtual]

Definition at line 38 of file Delay.h.

```
00038                                                                                 {
00039             if (samples>maxLength) {
00040                 _delay = maxLength;
00041             }else{
00042                 _delay = samples;
00043             }
00044             return _delay;
00045         }
```

7.11.3.4 template<unsigned long maxLength> bool DSG::Delay< maxLength >::Perform (DSG::DSGSample & *signal*) [inline], [virtual]

Implements DSG::SignalProcess.

Definition at line 64 of file Delay.h.

```
00064                                                                  {
00065         _swap = _buffer[_index-1];
00066         _buffer[_index-1]=signal;
00067         signal = _swap;
00068         increment();
00069         return true;
00070     }
```

7.11.3.5 template<unsigned long maxLength> bool DSG::Delay< maxLength >::Perform (DSG::RingBuffer & *signal*) [inline], [virtual]

Implements DSG::SignalProcess.

Definition at line 72 of file Delay.h.

```
00072                                                                   {
00073         if (!signal.Empty()) {
00074             count = signal.Count();
00075             while (count-- > 0) {
00076                 if(signal.Read(_temp)){
00077                     if (Perform(_temp)) {
00078                         signal.Write(_temp);
00079                     }else return false;
00080                 }else return false;
00081             }return true;
00082         }else return false;
00083     }
```

7.11.4 Member Data Documentation

7.11.4.1 template<unsigned long maxLength> DSG::DSGSample DSG::Delay< maxLength >::_buffer[maxLength] [protected]

Definition at line 53 of file Delay.h.

7.11.4.2 template<unsigned long maxLength> unsigned long DSG::Delay< maxLength >::_delay [protected]

Definition at line 50 of file Delay.h.

7.11.4.3 template<unsigned long maxLength> unsigned long DSG::Delay< maxLength >::_index [protected]

Definition at line 51 of file Delay.h.

7.11.4.4 template<unsigned long maxLength> const unsigned long DSG::Delay< maxLength >::_max [protected]

Definition at line 52 of file Delay.h.

7.11.4.5 template<unsigned long maxLength> DSG::DSGSample DSG::Delay< maxLength >::_swap [protected]

Definition at line 54 of file Delay.h.

7.11.4.6 template<unsigned long maxLength> DSG::DSGSample DSG::Delay< maxLength >::_temp [protected]

Definition at line 55 of file Delay.h.

7.11.4.7 template<unsigned long maxLength> unsigned long DSG::Delay< maxLength >::count [protected]

Definition at line 49 of file Delay.h.

The documentation for this class was generated from the following file:

- /Users/alexanderzywicki/Documents/DSG/src/Delay.h

7.12 DSG::DPW::DPW_Differentiator< order > Class Template Reference

DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the DPW Algorithm.

`#include <DPW.h>`

Public Member Functions

- DPW_Differentiator ()

7.12.1 Detailed Description

template<unsigned order>class DSG::DPW::DPW_Differentiator< order >

DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the DPW Algorithm.

Todo Fix DSG::DPW::DPW_Differentiator algorithms for orders 3-6

Definition at line 61 of file DPW.h.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 template<unsigned order> DSG::DPW::DPW_Differentiator< order >::DPW_Differentiator ( ) [inline]

Definition at line 63 of file DPW.h.

```
00063                                {
00064                    DSG::StaticAssertBounds<1, 6,order>();//order must be 1-6
00065                }
```

The documentation for this class was generated from the following file:

- /Users/alexanderzywicki/Documents/DSG/src/DPW.h

7.13 DSG::DPW::DPW_Differentiator< 1 > Class Template Reference

DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 1st order DPW Algorithm.

`#include <DPW.h>`

Public Member Functions

- DSG::DSGSample operator() (DSG::DSGSample const &signal, DSG::DSGSample const &dt)

7.13.1 Detailed Description

template<>class DSG::DPW::DPW_Differentiator< 1 >

DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 1st order DPW Algorithm.

Definition at line 69 of file DPW.h.

7.13.2 Member Function Documentation

7.13.2.1 DSG::DSGSample DSG::DPW::DPW_Differentiator< 1 >::operator() (DSG::DSGSample const & *signal*, DSG::DSGSample const & *dt*) [inline]

Definition at line 71 of file DPW.h.

```
00071                                                                                                                {
00072                 return signal;
00073             }
```

The documentation for this class was generated from the following file:

- /Users/alexanderzywicki/Documents/DSG/src/DPW.h

7.14 DSG::DPW::DPW_Differentiator< 2 > Class Template Reference

DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 2nd order DPW Algorithm.

`#include <DPW.h>`

Public Member Functions

- DSG::DSGSample operator() (DSG::DSGSample const &signal, DSG::DSGSample const &dt)

Protected Attributes

- DSG::DSGSample output
- DSG::DSGSample _delay

7.14.1 Detailed Description

template<>class DSG::DPW::DPW_Differentiator< 2 >

DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 2nd order DPW Algorithm.

Definition at line 77 of file DPW.h.

7.14.2 Member Function Documentation

7.14.2.1 DSG::DSGSample DSG::DPW::DPW_Differentiator< 2 >::operator() (DSG::DSGSample const & *signal*, DSG::DSGSample const & *dt*) [inline]

Definition at line 79 of file DPW.h.

```
00079                                                                                                                {
00080                 output = (signal - _delay)/(4.0 * dt);
00081                 _delay = signal;
00082                 return output;
00083             }
```

7.14.3 Member Data Documentation

7.14.3.1 DSG::DSGSample DSG::DPW::DPW_Differentiator< 2 >::_delay [protected]

Definition at line 86 of file DPW.h.

7.14.3.2 DSG::DSGSample DSG::DPW::DPW_Differentiator< 2 >::output [protected]

Definition at line 85 of file DPW.h.

The documentation for this class was generated from the following file:

- /Users/alexanderzywicki/Documents/DSG/src/DPW.h

7.15 DSG::DPW::DPW_Differentiator< 3 > Class Template Reference

DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 3rd order DPW Algorithm.

`#include <DPW.h>`

Public Member Functions

- DSG::DSGSample operator() (DSG::DSGSample const &signal, DSG::DSGSample const &dt)

Protected Attributes

- DSG::DSGSample output
- DSG::DSGSample _delay [2]

7.15.1 Detailed Description

template<>class DSG::DPW::DPW_Differentiator< 3 >

DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 3rd order DPW Algorithm.

Definition at line 90 of file DPW.h.

7.15.2 Member Function Documentation

7.15.2.1 DSG::DSGSample DSG::DPW::DPW_Differentiator< 3 >::operator() (DSG::DSGSample const & *signal*, DSG::DSGSample const & *dt*) [inline]

Definition at line 92 of file DPW.h.

```
00092                                                                                {
00093                 output  = (signal - _delay[0]);
00094                 output -= (_delay[0] - _delay[1]);
00095                 output /= (24.*DSG::Pow<2>(dt));
00096                 _delay[1]=_delay[0];
00097                 _delay[0]=signal;
00098                 return output;
00099             }
```

7.15.3 Member Data Documentation

7.15.3.1 DSG::DSGSample DSG::DPW::DPW_Differentiator< 3 >::_delay[2] [protected]

Definition at line 102 of file DPW.h.

7.15.3.2 DSG::DSGSample DSG::DPW::DPW_Differentiator< 3 >::output [protected]

Definition at line 101 of file DPW.h.

The documentation for this class was generated from the following file:

- /Users/alexanderzywicki/Documents/DSG/src/DPW.h

7.16 DSG::DPW::DPW_Differentiator< 4 > Class Template Reference

DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 4th order DPW Algorithm.

`#include <DPW.h>`

Public Member Functions

- DSG::DSGSample operator() (DSG::DSGSample const &signal, DSG::DSGSample const &dt)

Protected Attributes

- DSG::DSGSample output
- DSG::DSGSample _delay [3]

7.16.1 Detailed Description

template<>class DSG::DPW::DPW_Differentiator< 4 >

DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 4th order DPW Algorithm.

Definition at line 106 of file DPW.h.

7.16.2 Member Function Documentation

7.16.2.1 DSG::DSGSample DSG::DPW::DPW_Differentiator< 4 >::operator() (DSG::DSGSample const & *signal*, DSG::DSGSample const & *dt*) [inline]

Definition at line 108 of file DPW.h.

```
00108                                                                                {
00109                 output  = (signal - _delay[0]);
00110                 output -= (_delay[0] - _delay[1]);
00111                 output -= (_delay[1] - _delay[2]);
00112                 output /= 144*DSG::Pow<3>(dt);
00113                 _delay[2]=_delay[1];
00114                 _delay[1]=_delay[0];
00115                 _delay[0]=signal;
00116                 return output;
00117             }
```

7.16.3 Member Data Documentation

7.16.3.1 DSG::DSGSample DSG::DPW::DPW_Differentiator< 4 >::_delay[3] [protected]

Definition at line 120 of file DPW.h.

7.16.3.2 DSG::DSGSample DSG::DPW::DPW_Differentiator< 4 >::output [protected]

Definition at line 119 of file DPW.h.

The documentation for this class was generated from the following file:

- /Users/alexanderzywicki/Documents/DSG/src/DPW.h

7.17 DSG::DPW::DPW_Differentiator< 5 > Class Template Reference

DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 5th order DPW Algorithm.

`#include <DPW.h>`

Public Member Functions

- DSG::DSGSample operator() (DSG::DSGSample const &signal, DSG::DSGSample const &dt)

Protected Attributes

- DSG::DSGSample output
- DSG::DSGSample _delay [4]

7.17.1 Detailed Description

template<>class DSG::DPW::DPW_Differentiator< 5 >

DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 5th order DPW Algorithm.

Definition at line 124 of file DPW.h.

7.17.2 Member Function Documentation

7.17.2.1 DSG::DSGSample DSG::DPW::DPW_Differentiator< 5 >::operator() (DSG::DSGSample const & *signal*, DSG::DSGSample const & *dt*) [inline]

Definition at line 126 of file DPW.h.

```
00126                                                                                                {
00127                 output  = (signal - _delay[0]);
00128                 output -= (_delay[0] - _delay[1]);
00129                 output -= (_delay[1] - _delay[2]);
00130                 output -= (_delay[2] - _delay[3]);
00131                 output /= 960*DSG::Pow<4>(dt);
00132                 _delay[3]=_delay[2];
00133                 _delay[2]=_delay[1];
00134                 _delay[1]=_delay[0];
00135                 _delay[0]=signal;
00136                 return output;
00137             }
```

7.17.3 Member Data Documentation

7.17.3.1 DSG::DSGSample DSG::DPW::DPW_Differentiator< 5 >::_delay[4] [protected]

Definition at line 140 of file DPW.h.

7.17.3.2 DSG::DSGSample DSG::DPW::DPW_Differentiator< 5 >::output [protected]

Definition at line 139 of file DPW.h.

The documentation for this class was generated from the following file:

- /Users/alexanderzywicki/Documents/DSG/src/DPW.h

7.18 DSG::DPW::DPW_Differentiator< 6 > Class Template Reference

DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 6th order DPW Algorithm.

`#include <DPW.h>`

Public Member Functions

- DSG::DSGSample operator() (DSG::DSGSample const &signal, DSG::DSGSample const &dt)

Protected Attributes

- DSG::DSGSample output
- DSG::DSGSample _delay [5]

7.18.1 Detailed Description

template<>class DSG::DPW::DPW_Differentiator< 6 >

DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 6th order DPW Algorithm.

Definition at line 144 of file DPW.h.

7.18.2 Member Function Documentation

7.18.2.1 DSG::DSGSample DSG::DPW::DPW_Differentiator< 6 >::operator() (DSG::DSGSample const & *signal*, DSG::DSGSample const & *dt*) [inline]

Definition at line 146 of file DPW.h.

```
00146                                                                                                {
00147                 output  = (signal - _delay[0]);
00148                 output -= (_delay[0] - _delay[1]);
00149                 output -= (_delay[1] - _delay[2]);
00150                 output -= (_delay[2] - _delay[3]);
00151                 output -= (_delay[3] - _delay[4]);
00152                 output /= 7200*DSG::Pow<5>(dt);
00153                 _delay[4]=_delay[3];
00154                 _delay[3]=_delay[2];
00155                 _delay[2]=_delay[1];
00156                 _delay[1]=_delay[0];
00157                 _delay[0]=signal;
00158                 return output;
00159             }
```

7.18.3 Member Data Documentation

7.18.3.1 DSG::DSGSample DSG::DPW::DPW_Differentiator< 6 >::_delay[5] [protected]

Definition at line 162 of file DPW.h.

7.18.3.2 DSG::DSGSample DSG::DPW::DPW_Differentiator< 6 >::output [protected]

Definition at line 161 of file DPW.h.

The documentation for this class was generated from the following file:

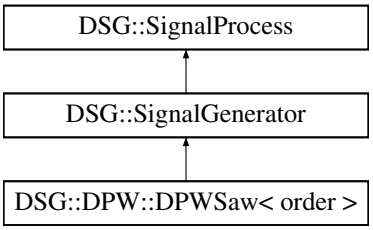
- /Users/alexanderzywicki/Documents/DSG/src/DPW.h

7.19 DSG::DPW::DPWSaw< order > Class Template Reference

DSG::DPW::DPWSaw - Sawtooth Generator using the Nth Order DPW algorithm.

`#include <DPWSaw.h>`

Inheritance diagram for DSG::DPW::DPWSaw< order >:



Public Member Functions

- DPWSaw ()
- DPWSaw (DSG::DSGFrequency const &frequency, DSG::DSGPhase const &offset)
- virtual ~DPWSaw ()
- virtual bool Perform (DSG::DSGSample &signal)
- virtual bool Perform (DSG::RingBuffer &signal)

Protected Attributes

- DSG::DSGSample _register
- DSG::DPW::DPW_Differentiator < order > _diff

Additional Inherited Members

7.19.1 Detailed Description

template<unsigned order>class DSG::DPW::DPWSaw< order >

DSG::DPW::DPWSaw - Sawtooth Generator using the Nth Order DPW algorithm.

Definition at line 18 of file DPWSaw.h.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 template<unsigned order> DSG::DPW::DPWSaw< order >::DPWSaw ( ) [inline]

Definition at line 20 of file DPWSaw.h.

```
00020                     :DSG::SignalGenerator(),_register(0){
00021                 DSG::StaticAssertBounds<1, 6,order>();
00022             }
```

7.19.2.2 template<unsigned order> DSG::DPW::DPWSaw< order >::DPWSaw (DSG::DSGFrequency const & *frequency*, DSG::DSGPhase const & *offset*) [inline]

Definition at line 23 of file DPWSaw.h.

```
00023 :DSG::SignalGenerator(frequency,offset),_register(0){
      DSG::StaticAssertBounds<1, 6,order>();}
```

7.19.2.3 template<unsigned order> virtual DSG::DPW::DPWSaw< order >::~DPWSaw ( ) [inline], [virtual]

Definition at line 24 of file DPWSaw.h.

```
00024 {}
```

7.19.3 Member Function Documentation

7.19.3.1 template<unsigned order> virtual bool DSG::DPW::DPWSaw< order >::Perform (DSG::DSGSample & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 25 of file DPWSaw.h.

```
00025                                                       {
00026                 //trivial saw ramping from -1 to 1
00027                 _register = _phasor;
00028                 _register-=0.5;
00029                 _register*=2.0;
00030                 /*-------------------------*/
00031                 //DPW algorithm
00032                 //polynomial shaping
00033                 _register=DSG::DPW::DPW_Polynomial<order>(_register);
00034                 //differentiating
00035                 signal = _diff(_register,_dt);
00036                 /*-------------------------*/
00037                 //signal = DSG::EnforceBounds<-1, 1>(signal);
00038                 //advance phase
00039                 step();
00040                 return true;
00041             }
```

7.19.3.2 template<unsigned order> virtual bool DSG::DPW::DPWSaw< order >::Perform (DSG::RingBuffer & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 42 of file DPWSaw.h.

```
00042                                                       {
00043                 signal.Flush();
00044                 while (!signal.Full()) {
00045                     if (Perform(_storage)) {
00046                         if(signal.Write(_storage)){
00047                         }else return false;
00048                     }else return false;
00049                 }return true;
00050             }
```

7.19.4 Member Data Documentation

7.19.4.1 template<unsigned order> DSG::DPW::DPW_Differentiator<order> DSG::DPW::DPWSaw< order >::_diff [protected]

Definition at line 53 of file DPWSaw.h.

7.19.4.2 template<unsigned order> DSG::DSGSample DSG::DPW::DPWSaw< order >::_register [protected]

Definition at line 52 of file DPWSaw.h.

The documentation for this class was generated from the following file:

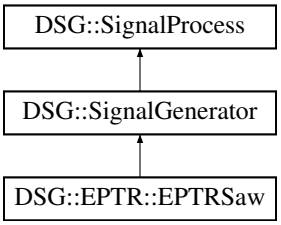
- /Users/alexanderzywicki/Documents/DSG/src/DPWSaw.h

7.20 DSG::EPTR::EPTRSaw Class Reference

DSG::EPTR::EPTRSaw-Sawtooth Wave Generator Using The Efficienct Polynomial Transfer Region Algorithm.

`#include <EPTRSaw.h>`

Inheritance diagram for DSG::EPTR::EPTRSaw:



Public Member Functions

- EPTRSaw ()
- EPTRSaw (DSG::DSGFrequency const &frequency, DSG::DSGPhase const &offset)
- virtual ~EPTRSaw ()
- virtual bool Perform (DSG::DSGSample &signal)
- virtual bool Perform (DSG::RingBuffer &signal)

Protected Attributes

- DSG::DSGSample _register

Additional Inherited Members

7.20.1 Detailed Description

DSG::EPTR::EPTRSaw-Sawtooth Wave Generator Using The Efficienct Polynomial Transfer Region Algorithm.

Todo Test and Possibly Re-Write DSG::EPTR::EPTRSaw algorithm

Definition at line 19 of file EPTRSaw.h.

7.20.2 Constructor & Destructor Documentation

7.20.2.1 DSG::EPTR::EPTRSaw::EPTRSaw ( )

Definition at line 9 of file EPTRSaw.cpp.

```
00009 :DSG::SignalGenerator(){}
```

7.20.2.2 DSG::EPTR::EPTRSaw::EPTRSaw (DSG::DSGFrequency const & *frequency*, DSG::DSGPhase const & *offset*)

Definition at line 10 of file EPTRSaw.cpp.

```
00010 :DSG::SignalGenerator(frequency,offset){}
```

7.20.2.3 DSG::EPTR::EPTRSaw::~EPTRSaw ( ) [virtual]

Definition at line 11 of file EPTRSaw.cpp.

```
00011 {}
```

7.20.3 Member Function Documentation

7.20.3.1 bool DSG::EPTR::EPTRSaw::Perform (DSG::DSGSample & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 29 of file EPTRSaw.h.

```
00029                                                          {
00030 #warning Untested For Aliasing DSG::EPTR::EPTRSaw::Perform()
00031             //generate trivial saw
00032             _register = _phasor;
00033             _register+=0.5;
00034             if (_register>1.0) {
00035                 --_register;
00036             }
00037             _register-=0.5;
00038             _register*=2.0;
00039             if (_register > 1.0-_dt) {
00040                 //transition region detected
00041                 //apply eptr correction
00042                 signal = _register - (_register/_dt) + (1.0/
      _dt) -1;
00043             }else{
00044                 signal = _register;
00045             }
00046             step();//avance phase
00047             return true;
00048         }
```

7.20.3.2 bool DSG::EPTR::EPTRSaw::Perform (DSG::RingBuffer & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 49 of file EPTRSaw.h.

```
00049                                                          {
00050             signal.Flush();
00051             while (!signal.Full()) {
00052                 if (Perform(_storage)) {
00053                     if(signal.Write(_storage)){
00054                     }else return false;
00055                 }else return false;
00056             }return true;
00057         }
```

7.20.4 Member Data Documentation

7.20.4.1 DSG::DSGSample DSG::EPTR::EPTRSaw::_register [protected]

Definition at line 27 of file EPTRSaw.h.

The documentation for this class was generated from the following files:

- /Users/alexanderzywicki/Documents/DSG/src/EPTRSaw.h
- /Users/alexanderzywicki/Documents/DSG/src/EPTRSaw.cpp

7.21 DSG::Factorial< N > Struct Template Reference

DSG::Factorial - Compute integer factorial.

`#include <DSGMath.h>`

Public Types

- enum { value = N * Factorial<N-1>::value }

7.21.1 Detailed Description

template<unsigned long N>struct DSG::Factorial< N >

DSG::Factorial - Compute integer factorial.

Definition at line 20 of file DSGMath.h.

7.21.2 Member Enumeration Documentation

7.21.2.1 template<unsigned long N> anonymous enum

Enumerator

value

Definition at line 21 of file DSGMath.h.

```
00021 {value = N * Factorial<N-1>::value};
```

The documentation for this struct was generated from the following file:

- /Users/alexanderzywicki/Documents/DSG/src/DSGMath.h

7.22 DSG::Factorial< 0 > Struct Template Reference

DSG::Factorial - Compute integer factorial.

`#include <DSGMath.h>`

Public Types

- enum { value = 1 }

7.22.1 Detailed Description

template<>struct DSG::Factorial< 0 >

DSG::Factorial - Compute integer factorial.

Definition at line 25 of file DSGMath.h.

7.22.2 Member Enumeration Documentation

7.22.2.1 anonymous enum

Enumerator

value

Definition at line 26 of file DSGMath.h.

```
00026 { value = 1 };
```

The documentation for this struct was generated from the following file:

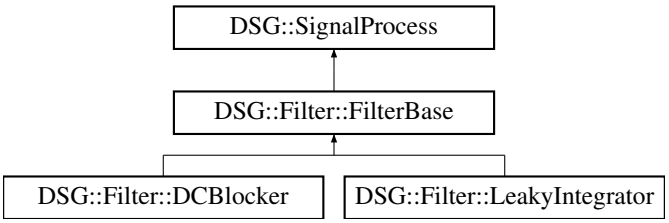
- /Users/alexanderzywicki/Documents/DSG/src/DSGMath.h

7.23 DSG::Filter::FilterBase Class Reference

DSG::Filter::FilterBase - Filter Base Class, implements interface for cutoff frequency.

`#include <Filter.h>`

Inheritance diagram for DSG::Filter::FilterBase:



Public Member Functions

- FilterBase ()
- virtual ~FilterBase ()
- virtual bool Perform (DSG::DSGSample &signal)
- virtual bool Perform (DSG::RingBuffer &signal)
- virtual bool Cutoff (DSG::DSGFrequency const &cutoff)

Protected Attributes

- DSG::DSGSample _temp
- unsigned long count

7.23.1 Detailed Description

DSG::Filter::FilterBase - Filter Base Class, implements interface for cutoff frequency.

Definition at line 18 of file Filter.h.

7.23.2 Constructor & Destructor Documentation

7.23.2.1 DSG::Filter::FilterBase::FilterBase ( )

Definition at line 9 of file Filter.cpp.

```
00009 :_temp(0),count(0){}
```

7.23.2.2 DSG::Filter::FilterBase::~FilterBase ( ) [virtual]

Definition at line 10 of file Filter.cpp.

```
00010 {}
```

7.23.3 Member Function Documentation

7.23.3.1 bool DSG::Filter::FilterBase::Cutoff (DSG::DSGFrequency const & *cutoff*) [inline], [virtual]

Reimplemented in DSG::Filter::LeakyIntegrator.

Definition at line 44 of file Filter.h.

```
00044                                                                   {
00045             return false;
00046         }
```

7.23.3.2 bool DSG::Filter::FilterBase::Perform (DSG::DSGSample & *signal*) [inline], [virtual]

Implements DSG::SignalProcess.

Reimplemented in DSG::Filter::LeakyIntegrator, and DSG::Filter::DCBlocker.

Definition at line 29 of file Filter.h.

```
00029                                                                {
00030             return true;
00031         }
```

7.23.3.3 bool DSG::Filter::FilterBase::Perform (DSG::RingBuffer & *signal*) [inline], [virtual]

Implements DSG::SignalProcess.

Reimplemented in DSG::Filter::LeakyIntegrator, and DSG::Filter::DCBlocker.

Definition at line 32 of file Filter.h.

```
00032                                                                 {
00033             if (!signal.Empty()) {
00034                 count = signal.Count();
00035                 while (count-- > 0) {
00036                     if(signal.Read(_temp)){
00037                         if (Perform(_temp)) {
00038                             signal.Write(_temp);
00039                         }else return false;
00040                     }else return false;
00041                 }return true;
00042             }else return false;
00043         }
```

7.23.4 Member Data Documentation

7.23.4.1 DSG::DSGSample DSG::Filter::FilterBase::_temp [protected]

Definition at line 26 of file Filter.h.

7.23.4.2 unsigned long DSG::Filter::FilterBase::count [protected]

Definition at line 27 of file Filter.h.

The documentation for this class was generated from the following files:

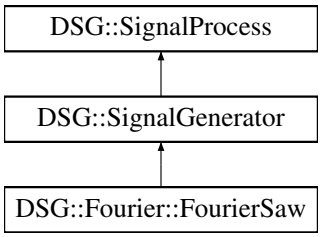
- /Users/alexanderzywicki/Documents/DSG/src/Filter.h
- /Users/alexanderzywicki/Documents/DSG/src/Filter.cpp

7.24 DSG::Fourier::FourierSaw Class Reference

DSG::Fourier::FourierSaw - Fourier Series Sawtooth Wave Generator.

`#include <FourierSaw.h>`

Inheritance diagram for DSG::Fourier::FourierSaw:



Public Member Functions

- FourierSaw ()
- FourierSaw (DSG::DSGFrequency const &frequency, DSG::DSGPhase const &offset)
- virtual ~FourierSaw ()
- virtual bool Perform (DSG::DSGSample &signal)
- virtual bool Perform (DSG::RingBuffer &signal)
- virtual DSG::DSGFrequency const & Frequency (DSG::DSGFrequency const &value)

Protected Attributes

- unsigned long _h
- const double _a
- double phs
- double value
- int i

Additional Inherited Members

7.24.1 Detailed Description

DSG::Fourier::FourierSaw - Fourier Series Sawtooth Wave Generator.

Definition at line 18 of file FourierSaw.h.

7.24.2 Constructor & Destructor Documentation

7.24.2.1 DSG::Fourier::FourierSaw::FourierSaw ( )

Definition at line 9 of file FourierSaw.cpp.

```
00009 :DSG::SignalGenerator(),_a(1.7/PI),phs(0),value(0),
      i(0){}
```

7.24.2.2 DSG::Fourier::FourierSaw::FourierSaw (DSG::DSGFrequency const & *frequency,* DSG::DSGPhase const & *offset*)

Definition at line 10 of file FourierSaw.cpp.

```
00010                                                                                                  :
      DSG::SignalGenerator(frequency,offset),_a(1.7/PI),phs(0),
      value(0),i(0){
00011     _h = MaxHarms(_frequency)+1;
00012 }
```

7.24.2.3 DSG::Fourier::FourierSaw::~FourierSaw ( ) [virtual]

Definition at line 13 of file FourierSaw.cpp.

```
00013 {}
```

7.24.3 Member Function Documentation

7.24.3.1 DSG::DSGFrequency const & DSG::Fourier::FourierSaw::Frequency (DSG::DSGFrequency const & *value*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 53 of file FourierSaw.h.

```
00053                                                                                                  {
00054             _frequency = value;
00055             _dt = _frequency/DSG::SampleRate();
00056             _h = MaxHarms(_frequency);
00057             return _frequency;
00058         }
```

7.24.3.2 bool DSG::Fourier::FourierSaw::Perform (DSG::DSGSample & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 33 of file FourierSaw.h.

```
00033                                                                  {
00034             //_h Sine Calls Per Sample where _h  is theoretically nyquist / frequency
00035             value=DSG::Sin(_phasor);
00036             for (i=2; i<_h; ++i) {
00037                 value += (1.0/i) * DSG::Sin(_phasor*i);
00038             }
00039             value*=_a;
00040             signal = value;
00041             step();
00042             return true;
00043         }
```

7.24.3.3 bool DSG::Fourier::FourierSaw::Perform (DSG::RingBuffer & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 44 of file FourierSaw.h.

```
00044                                                                   {
00045             signal.Flush();
00046             while (!signal.Full()) {
00047                 if (Perform(_storage)) {
00048                     if(signal.Write(_storage)){
00049                     }else return false;
00050                 }else return false;
00051             }return true;
00052         }
```

7.24.4 Member Data Documentation

7.24.4.1 const double DSG::Fourier::FourierSaw::_a [protected]

Definition at line 28 of file FourierSaw.h.

7.24.4.2 unsigned long DSG::Fourier::FourierSaw::_h [protected]

Definition at line 27 of file FourierSaw.h.

7.24.4.3 int DSG::Fourier::FourierSaw::i [protected]

Definition at line 31 of file FourierSaw.h.

7.24.4.4 double DSG::Fourier::FourierSaw::phs [protected]

Definition at line 29 of file FourierSaw.h.

7.24.4.5 double DSG::Fourier::FourierSaw::value [protected]

Definition at line 30 of file FourierSaw.h.

The documentation for this class was generated from the following files:

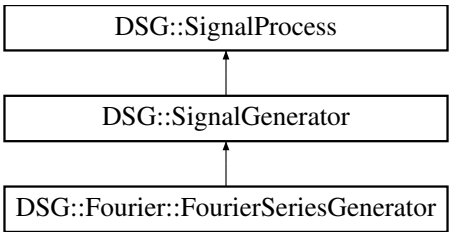
- /Users/alexanderzywicki/Documents/DSG/src/FourierSaw.h
- /Users/alexanderzywicki/Documents/DSG/src/FourierSaw.cpp

7.25 DSG::Fourier::FourierSeriesGenerator Class Reference

DSG::Fourier::FourierSeriesGenerator - Generates a wave form using a user specified Fourier Series.

`#include <FourierSeries.h>`

Inheritance diagram for DSG::Fourier::FourierSeriesGenerator:



Public Types

- typedef std::vector< Harmonic > FourierSeries

Public Member Functions

- FourierSeriesGenerator ()
- FourierSeriesGenerator (DSG::DSGFrequency const &frequency, DSG::DSGPhase const &offset)
- virtual ~FourierSeriesGenerator ()
- virtual bool Perform (DSG::DSGSample &signal)
- virtual bool Perform (DSG::RingBuffer &signal)
- void Series (FourierSeries const &series)
- FourierSeries & Series ()

Protected Attributes

- FourierSeries _series
- DSG::DSGSample value

Additional Inherited Members

7.25.1 Detailed Description

DSG::Fourier::FourierSeriesGenerator - Generates a wave form using a user specified Fourier Series.

Definition at line 32 of file FourierSeries.h.

7.25.2 Member Typedef Documentation

7.25.2.1 typedef std::vector<Harmonic> DSG::Fourier::FourierSeriesGenerator::FourierSeries

Definition at line 34 of file FourierSeries.h.

7.25.3 Constructor & Destructor Documentation

7.25.3.1 DSG::Fourier::FourierSeriesGenerator::FourierSeriesGenerator ( )

Definition at line 29 of file FourierSeries.cpp.

```
00029 :DSG::SignalGenerator(){}
```

7.25.3.2 DSG::Fourier::FourierSeriesGenerator::FourierSeriesGenerator (DSG::DSGFrequency const & *frequency*, DSG::DSGPhase const & *offset*)

Definition at line 30 of file FourierSeries.cpp.

```
00030 :DSG::SignalGenerator(frequency,offset){}
```

7.25.3.3 DSG::Fourier::FourierSeriesGenerator::~FourierSeriesGenerator ( ) [virtual]

Definition at line 31 of file FourierSeries.cpp.

```
00031 {}
```

7.25.4 Member Function Documentation

7.25.4.1 bool DSG::Fourier::FourierSeriesGenerator::Perform (DSG::DSGSample & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 46 of file FourierSeries.h.

```
00046                                                                    {
00047             value = _phasor;
00048             signal=0;
00049             for (auto i = _series.begin(); i!=_series.end(); ++i) {
00050                 signal += DSG::Sin(_phasor * i->Ratio())*i->Amplitude();
00051             }
00052             step();
00053             return true;
00054         }
```

7.25.4.2 bool DSG::Fourier::FourierSeriesGenerator::Perform (DSG::RingBuffer & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 55 of file FourierSeries.h.

```
00055                                                                     {
00056             signal.Flush();
00057             while (!signal.Full()) {
00058                 if (Perform(_storage)) {
00059                     if(signal.Write(_storage)){
00060                     }else return false;
00061                 }else return false;
00062             }return true;
00063         }
```

7.25.4.3 void DSG::Fourier::FourierSeriesGenerator::Series (FourierSeries const & *series*) [inline]

Definition at line 64 of file FourierSeries.h.

```
00064
00065             {
00066             _series = series;
00067         }
```

7.25.4.4 DSG::Fourier::FourierSeriesGenerator::FourierSeries & DSG::Fourier::FourierSeriesGenerator::Series ( ) [inline]

Definition at line 67 of file FourierSeries.h.

```
00067                                                                                {
00068             return _series;
00069         }
```

7.25.5 Member Data Documentation

7.25.5.1 FourierSeries DSG::Fourier::FourierSeriesGenerator::_series [protected]

Definition at line 43 of file FourierSeries.h.

7.25.5.2 DSG::DSGSample DSG::Fourier::FourierSeriesGenerator::value [protected]

Definition at line 44 of file FourierSeries.h.

The documentation for this class was generated from the following files:

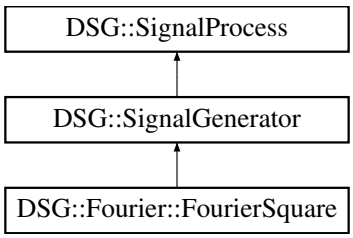
- /Users/alexanderzywicki/Documents/DSG/src/FourierSeries.h
- /Users/alexanderzywicki/Documents/DSG/src/FourierSeries.cpp

7.26 DSG::Fourier::FourierSquare Class Reference

DSG::Fourier::FourierSquare - Fourier Series Square Wave Generator.

`#include <FourierSquare.h>`

Inheritance diagram for DSG::Fourier::FourierSquare:



Public Member Functions

- FourierSquare ()
- FourierSquare (DSG::DSGFrequency const &frequency, DSG::DSGPhase const &offset)
- virtual ~FourierSquare ()
- virtual bool Perform (DSG::DSGSample &signal)
- virtual bool Perform (DSG::RingBuffer &signal)
- virtual DSG::DSGFrequency const & Frequency (DSG::DSGFrequency const &value)

Protected Attributes

- unsigned long _h
- const double _a
- double phs

- double value
- int i

Additional Inherited Members

7.26.1 Detailed Description

DSG::Fourier::FourierSquare - Fourier Series Square Wave Generator.

Definition at line 18 of file FourierSquare.h.

7.26.2 Constructor & Destructor Documentation

7.26.2.1 DSG::Fourier::FourierSquare::FourierSquare ( )

Definition at line 9 of file FourierSquare.cpp.

```
00009 :DSG::SignalGenerator(),_a(3.6/PI),phs(0),value(0),
      i(0){}
```

7.26.2.2 DSG::Fourier::FourierSquare::FourierSquare (DSG::DSGFrequency const & *frequency*, DSG::DSGPhase const & *offset*)

Definition at line 10 of file FourierSquare.cpp.

```
00010                                                                                :
      DSG::SignalGenerator(frequency,offset),_a(3.6/PI),phs(0),
      value(0),i(0){
00011     _h = MaxHarms(_frequency)+1;
00012 }
```

7.26.2.3 DSG::Fourier::FourierSquare::~FourierSquare ( ) [virtual]

Definition at line 13 of file FourierSquare.cpp.

```
00013 {}
```

7.26.3 Member Function Documentation

7.26.3.1 DSG::DSGFrequency const & DSG::Fourier::FourierSquare::Frequency (DSG::DSGFrequency const & *value*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 53 of file FourierSquare.h.

```
00053                                                                                {
00054             _frequency = value;
00055             _dt = _frequency/DSG::SampleRate();
00056             _h = MaxHarms(_frequency);
00057             return _frequency;
00058         }
```

7.26.3.2 bool DSG::Fourier::FourierSquare::Perform (DSG::DSGSample & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 33 of file FourierSquare.h.

```
00033                                                                   {
00034             //(_h/2)+1 Sine Calls Per Sample
00035             value=DSG::Sin(_phasor);//i=1
00036             for (i=3; i<_h; i+=2) {//i=3..5..7..
00037                 value += (1.0/i) * DSG::Sin(_phasor*i);
00038             }
00039             value*=_a;
00040             signal = value;
00041             step();
00042             return true;
00043         }
```

7.26.3.3 bool DSG::Fourier::FourierSquare::Perform (DSG::RingBuffer & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 44 of file FourierSquare.h.

```
00044                                                                    {
00045             signal.Flush();
00046             while (!signal.Full()) {
00047                 if (Perform(_storage)) {
00048                     if(signal.Write(_storage)){
00049                     }else return false;
00050                 }else return false;
00051             }return true;
00052         }
```

7.26.4 Member Data Documentation

7.26.4.1 const double DSG::Fourier::FourierSquare::_a [protected]

Definition at line 28 of file FourierSquare.h.

7.26.4.2 unsigned long DSG::Fourier::FourierSquare::_h [protected]

Definition at line 27 of file FourierSquare.h.

7.26.4.3 int DSG::Fourier::FourierSquare::i [protected]

Definition at line 31 of file FourierSquare.h.

7.26.4.4 double DSG::Fourier::FourierSquare::phs [protected]

Definition at line 29 of file FourierSquare.h.

7.26.4.5 double DSG::Fourier::FourierSquare::value [protected]

Definition at line 30 of file FourierSquare.h.

The documentation for this class was generated from the following files:

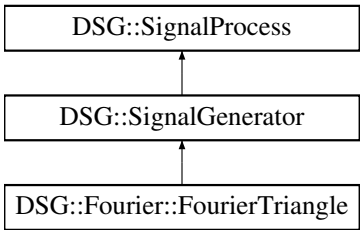
- /Users/alexanderzywicki/Documents/DSG/src/FourierSquare.h
- /Users/alexanderzywicki/Documents/DSG/src/FourierSquare.cpp

7.27 DSG::Fourier::FourierTriangle Class Reference

DSG::Fourier::FourierTriangle - Fourier Series Triangle Wave Generator.

`#include <FourierTriangle.h>`

Inheritance diagram for DSG::Fourier::FourierTriangle:



Public Member Functions

- FourierTriangle ()
- FourierTriangle (DSG::DSGFrequency const &frequency, DSG::DSGPhase const &offset)
- virtual ~FourierTriangle ()
- virtual bool Perform (DSG::DSGSample &signal)
- virtual bool Perform (DSG::RingBuffer &signal)
- virtual DSG::DSGFrequency const & Frequency (DSG::DSGFrequency const &value)

Protected Attributes

- unsigned long _h
- const double _a
- double phs
- double value
- int i

Additional Inherited Members

7.27.1 Detailed Description

DSG::Fourier::FourierTriangle - Fourier Series Triangle Wave Generator.

Definition at line 18 of file FourierTriangle.h.

7.27.2 Constructor & Destructor Documentation

7.27.2.1 DSG::Fourier::FourierTriangle::FourierTriangle ( )

Definition at line 9 of file FourierTriangle.cpp.

```
00009 :DSG::SignalGenerator(),_a(8.0/(PI*PI)),phs(0),
      value(0),i(0){}
```

7.27.2.2 DSG::Fourier::FourierTriangle::FourierTriangle (DSG::DSGFrequency const & *frequency*, DSG::DSGPhase const & *offset*)

Definition at line 10 of file FourierTriangle.cpp.

```
00010                                                                         :
      DSG::SignalGenerator(frequency,offset),_a(8.0/(PI*PI)),
      phs(0),value(0),i(0){
00011     _h = MaxHarms(_frequency)+1;
00012 }
```

7.27.2.3 DSG::Fourier::FourierTriangle::∼FourierTriangle ( ) [virtual]

Definition at line 13 of file FourierTriangle.cpp.

```
00013 {}
```

7.27.3 Member Function Documentation

7.27.3.1 DSG::DSGFrequency const & DSG::Fourier::FourierTriangle::Frequency (DSG::DSGFrequency const & *value*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 55 of file FourierTriangle.h.

```
00055                                                                           {
00056             _frequency = value;
00057             _dt = _frequency/DSG::SampleRate();
00058             _h = MaxHarms(_frequency);
00059             return _frequency;
00060         }
```

7.27.3.2 bool DSG::Fourier::FourierTriangle::Perform (DSG::DSGSample & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 33 of file FourierTriangle.h.

```
00033                                                         {
00034             //(_h/2)+1 Sine Calls Per Sample
00035             value=DSG::Sin(_phasor);//i=1
00036             double sgn = -1;
00037             for (i=3; i<_h; i+=2) {//i=3..5..7..
00038                 value += sgn * (1.0/(i*i)) * DSG::Sin(_phasor*
      i);
00039                 sgn*=-1;
00040             }
00041             value*=_a;
00042             signal = value;
00043             step();
00044             return true;
00045         }
```

7.27.3.3 bool DSG::Fourier::FourierTriangle::Perform (DSG::RingBuffer & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 46 of file FourierTriangle.h.

```
00046                                                                   {
00047             signal.Flush();
00048             while (!signal.Full()) {
00049                 if (Perform(_storage)) {
00050                     if(signal.Write(_storage)){
00051                     }else return false;
00052                 }else return false;
00053             }return true;
00054         }
```

7.27.4 Member Data Documentation

7.27.4.1 const double DSG::Fourier::FourierTriangle::_a [protected]

Definition at line 28 of file FourierTriangle.h.

7.27.4.2 unsigned long DSG::Fourier::FourierTriangle::_h [protected]

Definition at line 27 of file FourierTriangle.h.

7.27.4.3 int DSG::Fourier::FourierTriangle::i [protected]

Definition at line 31 of file FourierTriangle.h.

7.27.4.4 double DSG::Fourier::FourierTriangle::phs [protected]

Definition at line 29 of file FourierTriangle.h.

7.27.4.5 double DSG::Fourier::FourierTriangle::value [protected]

Definition at line 30 of file FourierTriangle.h.

The documentation for this class was generated from the following files:

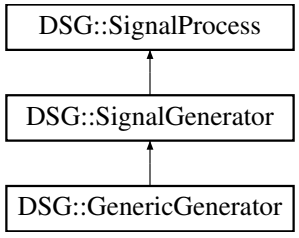
- /Users/alexanderzywicki/Documents/DSG/src/FourierTriangle.h
- /Users/alexanderzywicki/Documents/DSG/src/FourierTriangle.cpp

7.28 DSG::GenericGenerator Class Reference

DSG::GenericGenerator - Generator designed to use a stateless generator function such as DSG::Sin()

`#include <GenericGenerator.h>`

Inheritance diagram for DSG::GenericGenerator:



Public Member Functions

- GenericGenerator ()
- GenericGenerator (DSG::DSGFrequency const &frequency, DSG::DSGPhase const &offset, DSG::DSG↩ Sample(∗signalFunction)(DSG::DSGSample const &))
- virtual ∼GenericGenerator ()
- virtual bool Perform (DSG::DSGSample &signal)
- virtual bool Perform (DSG::RingBuffer &signal)

Protected Attributes

- DSG::DSGSample(∗ _callback)(DSG::DSGSample const &)

Additional Inherited Members

7.28.1 Detailed Description

DSG::GenericGenerator - Generator designed to use a stateless generator function such as DSG::Sin()

Definition at line 13 of file GenericGenerator.h.

7.28.2 Constructor & Destructor Documentation

7.28.2.1 DSG::GenericGenerator::GenericGenerator ( )

Definition at line 9 of file GenericGenerator.cpp.

```
00009 :DSG::SignalGenerator(){}
```

7.28.2.2 DSG::GenericGenerator::GenericGenerator (DSG::DSGFrequency const & *frequency,* DSG::DSGPhase const & *offset,* DSG::DSGSample(∗)(DSG::DSGSample const &) *signalFunction*)

Definition at line 10 of file GenericGenerator.cpp.

```
00010 :DSG::SignalGenerator(frequency,offset),_callback(signalFunction){}
```

7.28.2.3 DSG::GenericGenerator::∼GenericGenerator ( ) [virtual]

Definition at line 11 of file GenericGenerator.cpp.

```
00011 {}
```

7.28.3 Member Function Documentation

7.28.3.1 bool DSG::GenericGenerator::Perform (DSG::DSGSample & *signal*) [inline], [virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 23 of file GenericGenerator.h.

```
00023                                                       {
00024         if (_callback!=nullptr) {
00025             signal = _callback(_phasor);
00026         }else signal = 0;
00027         step();
00028         return true;
00029     }
```

7.28.3.2 bool DSG::GenericGenerator::Perform (DSG::RingBuffer & *signal*) [inline],[virtual]

Reimplemented from DSG::SignalGenerator.

Definition at line 30 of file GenericGenerator.h.

```
00030                                                                   {
00031         signal.Flush();
00032         while (!signal.Full()) {
00033             if (Perform(_storage)) {
00034                 if(signal.Write(_storage)){
00035                 }else return false;
00036             }else return false;
00037         }return true;
00038     }
```

7.28.4 Member Data Documentation

7.28.4.1 DSG::DSGSample(* DSG::GenericGenerator::_callback)(DSG::DSGSample const &) [protected]

Definition at line 21 of file GenericGenerator.h.

The documentation for this class was generated from the following files:

- /Users/alexanderzywicki/Documents/DSG/src/GenericGenerator.h
- /Users/alexanderzywicki/Documents/DSG/src/GenericGenerator.cpp

7.29 DSG::Fourier::Harmonic Class Reference

DSG::Fourier::Harmonic - Represents a single harmonic in a Fourier Series.

`#include <FourierSeries.h>`

Public Member Functions

- Harmonic ()
- Harmonic (DSG::DSGSample const &ratio, DSG::DSGSample const &amplitude)
- virtual ~Harmonic ()
- DSG::DSGSample const & Ratio () const
- DSG::DSGSample const & Ratio (DSG::DSGSample const &value)
- DSG::DSGSample const & Amplitude () const
- DSG::DSGSample const & Amplitude (DSG::DSGSample const &value)

Protected Attributes

- DSG::DSGSample _ratio
- DSG::DSGSample _amplitude

7.29.1 Detailed Description

DSG::Fourier::Harmonic - Represents a single harmonic in a Fourier Series.

Definition at line 18 of file FourierSeries.h.

7.29.2 Constructor & Destructor Documentation

7.29.2.1 DSG::Fourier::Harmonic::Harmonic ( )

Definition at line 9 of file FourierSeries.cpp.

```
00009 :_ratio(0),_amplitude(0){}
```

7.29.2.2 DSG::Fourier::Harmonic::Harmonic (DSG::DSGSample const & *ratio*, DSG::DSGSample const & *amplitude*)

Definition at line 10 of file FourierSeries.cpp.

```
00010 :_ratio(ratio),_amplitude(amplitude){}
```

7.29.2.3 DSG::Fourier::Harmonic::~Harmonic ( ) [virtual]

Definition at line 11 of file FourierSeries.cpp.

```
00011                                {
00012     _ratio=0;
00013     _amplitude=0;
00014 }
```

7.29.3 Member Function Documentation

7.29.3.1 DSG::DSGSample const & DSG::Fourier::Harmonic::Amplitude ( ) const

Definition at line 22 of file FourierSeries.cpp.

```
00022                                                    {
00023     return _amplitude;
00024 }
```

7.29.3.2 DSG::DSGSample const & DSG::Fourier::Harmonic::Amplitude (DSG::DSGSample const & *value*)

Definition at line 25 of file FourierSeries.cpp.

```
00025                                                                    {
00026     _amplitude=value;
00027     return _amplitude;
00028 }
```

7.29.3.3 DSG::DSGSample const & DSG::Fourier::Harmonic::Ratio ( ) const

Definition at line 15 of file FourierSeries.cpp.

```
00015                                                {
00016     return _ratio;
00017 }
```

7.29.3.4 DSG::DSGSample const & DSG::Fourier::Harmonic::Ratio (DSG::DSGSample const & *value*)

Definition at line 18 of file FourierSeries.cpp.

```
00018                                                                {
00019     _ratio = value;
00020     return _ratio;
00021 }
```

7.29.4 Member Data Documentation

7.29.4.1 DSG::DSGSample DSG::Fourier::Harmonic::_amplitude [protected]

Definition at line 29 of file FourierSeries.h.

7.29.4.2 DSG::DSGSample DSG::Fourier::Harmonic::_ratio [protected]

Definition at line 28 of file FourierSeries.h.

The documentation for this class was generated from the following files:

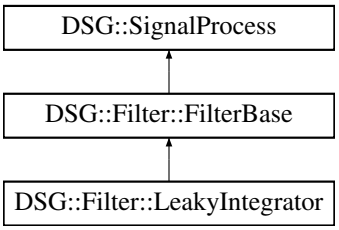
- /Users/alexanderzywicki/Documents/DSG/src/FourierSeries.h
- /Users/alexanderzywicki/Documents/DSG/src/FourierSeries.cpp

7.30 DSG::Filter::LeakyIntegrator Class Reference

DSG::Filter::LeakyIntegrator - Leaky integrator.

`#include <Leaky.h>`

Inheritance diagram for DSG::Filter::LeakyIntegrator:



Public Member Functions

- LeakyIntegrator ()
- LeakyIntegrator (DSG::DSGFrequency const &cutoff)
- virtual ~LeakyIntegrator ()
- virtual bool Perform (DSG::DSGSample &signal)
- virtual bool Perform (DSG::RingBuffer &signal)
- virtual bool Cutoff (DSG::DSGFrequency const &cutoff)

Protected Attributes

- double x1
- double y1
- double a
- double b
- double y

7.30.1 Detailed Description

DSG::Filter::LeakyIntegrator - Leaky integrator.

Definition at line 19 of file Leaky.h.

7.30.2 Constructor & Destructor Documentation

7.30.2.1 DSG::Filter::LeakyIntegrator::LeakyIntegrator ( )

Definition at line 9 of file Leaky.cpp.

```
00009                                              :DSG::Filter::FilterBase(){
00010     x1=0;
00011     y1=0;
00012     a=0;
00013     b=0;
00014     y=0;
00015 }
```

7.30.2.2 DSG::Filter::LeakyIntegrator::LeakyIntegrator (DSG::DSGFrequency const & *cutoff*)

Definition at line 16 of file Leaky.cpp.

```
00016                                                                          :
      DSG::Filter::FilterBase(){
00017     x1=0;
00018     y1=0;
00019     a=0;
00020     b=0;
00021     y=0;
00022     Cutoff(cutoff);
00023 }
```

7.30.2.3 DSG::Filter::LeakyIntegrator::~LeakyIntegrator ( ) [virtual]

Definition at line 24 of file Leaky.cpp.

```
00024                                              {
00025     x1=0;
00026     y1=0;
00027     a=0;
00028     b=0;
00029     y=0;
00030 }
```

7.30.3 Member Function Documentation

7.30.3.1 bool DSG::Filter::LeakyIntegrator::Cutoff (DSG::DSGFrequency const & *cutoff*) [inline], [virtual]

Reimplemented from DSG::Filter::FilterBase.

Definition at line 50 of file Leaky.h.

```
00050                                                                          {
00051             double Omega;
00052             x1 = y1 = 0.0;
00053             Omega = atan(PI * cutoff);
00054             a = -(1.0 - Omega) / (1.0 + Omega);
00055             b = (1.0 - b) / 2.0;
00056             return true;
00057         }
```

7.30.3.2 bool DSG::Filter::LeakyIntegrator::Perform (DSG::DSGSample & *signal*) [inline], [virtual]

Reimplemented from DSG::Filter::FilterBase.

Definition at line 31 of file Leaky.h.

```
00031                                                                   {
00032             y = b * (signal + x1) - a * y1;
00033             x1=signal;
00034             y1=y;
00035             signal=y;
00036             return true;
00037         }
```

7.30.3.3 bool DSG::Filter::LeakyIntegrator::Perform (DSG::RingBuffer & *signal*) [inline], [virtual]

Reimplemented from DSG::Filter::FilterBase.

Definition at line 38 of file Leaky.h.

```
00038                                                                    {
00039             if (!signal.Empty()) {
00040                 count = signal.Count();
00041                 while (count-- > 0) {
00042                     if(signal.Read(_temp)){
00043                         if (Perform(_temp)) {
00044                             signal.Write(_temp);
00045                         }else return false;
00046                     }else return false;
00047                 }return true;
00048             }else return false;
00049         }
```

7.30.4 Member Data Documentation

7.30.4.1 double DSG::Filter::LeakyIntegrator::a [protected]

Definition at line 28 of file Leaky.h.

7.30.4.2 double DSG::Filter::LeakyIntegrator::b [protected]

Definition at line 28 of file Leaky.h.

7.30.4.3 double DSG::Filter::LeakyIntegrator::x1 [protected]

Definition at line 28 of file Leaky.h.

7.30.4.4 double DSG::Filter::LeakyIntegrator::y [protected]

Definition at line 29 of file Leaky.h.

7.30.4.5 double DSG::Filter::LeakyIntegrator::y1 [protected]

Definition at line 28 of file Leaky.h.

The documentation for this class was generated from the following files:

- /Users/alexanderzywicki/Documents/DSG/src/Leaky.h
- /Users/alexanderzywicki/Documents/DSG/src/Leaky.cpp

7.31 DSG::LUT< element, size > Class Template Reference

DSG::LUT - Look Up Table.

`#include <LUT.h>`

Public Types

- typedef element(∗ FillFunction)(element)
- typedef element(∗ FillFunctionConstRef)(element const &)

Public Member Functions

- LUT ()
- LUT (FillFunction fill, double const &range=1.0)
- LUT (FillFunctionConstRef fill, double const &range=1.0)
- ∼LUT ()
- element const & operator[] (unsigned long const &index) const
- element & operator[] (unsigned long const &index)
- element const & operator() (double const &x)
- unsigned long const & Size () const

Protected Attributes

- element _table [size]
- const unsigned long _size
- double phs

7.31.1 Detailed Description

template<typename element, unsigned long size>class DSG::LUT< element, size >

DSG::LUT - Look Up Table.

Definition at line 17 of file LUT.h.

7.31.2 Member Typedef Documentation

7.31.2.1 template<typename element, unsigned long size> typedef element(∗ DSG::LUT< element, size >::FillFunction)(element)

Definition at line 19 of file LUT.h.

7.31.2.2 template<typename element, unsigned long size> typedef element(∗ DSG::LUT< element, size >::FillFunctionConstRef)(element const &)

Definition at line 20 of file LUT.h.

7.31.3 Constructor & Destructor Documentation

7.31.3.1 template<typename element, unsigned long size> DSG::LUT< element, size >::LUT ( ) [inline]

Definition at line 21 of file LUT.h.

```
00021 :_size(size){}
```

7.31.3.2 template<typename element, unsigned long size> DSG::LUT< element, size >::LUT (FillFunction *fill,* double const & *range* = 1.0) [inline]

Definition at line 22 of file LUT.h.

```
00022                                                   :_size(size){
00023             //range is the expected input range for the function
00024             //example would  be 0-2pi or 0-1
00025             //would be provided a 2pi or 1
00026             //defaults to 1
00027             double step = range/(double)_size;
00028             phs = 0;
00029             for (int i=0; i<_size; ++i) {
00030                 _table[i] = fill(phs);
00031                 phs+=step;
00032             }
00033         }
```

7.31.3.3 template<typename element, unsigned long size> DSG::LUT< element, size >::LUT (FillFunctionConstRef *fill,* double const & *range* = 1.0) [inline]

Definition at line 34 of file LUT.h.

```
00034                                                           :_size(size){
00035             //range is the expected input range for the function
00036             //example would  be 0-2pi or 0-1
00037             //would be provided a 2pi or 1
00038             //defaults to 1
00039             double step = range/_size;
00040             phs = 0;
00041             for (int i=0; i<_size; ++i) {
00042                 _table[i] = fill(phs);
00043                 phs+=step;
00044             }
00045         }
```

7.31.3.4 template<typename element, unsigned long size> DSG::LUT< element, size >::~LUT ( ) [inline]

Definition at line 46 of file LUT.h.

```
00046 {}
```

7.31.4 Member Function Documentation

7.31.4.1 template<typename element, unsigned long size> element const& DSG::LUT< element, size >::operator() (double const & *x*) [inline]

Definition at line 59 of file LUT.h.

```
00059                                                   {
00060             phs=x;
00061             //need range checking on x to ensure 0-1 range
00062             phs<0 ? phs = 1-(phs*-1):0;
00063             phs-=((int)phs);
00064             return this->_table[(unsigned)(phs*(this->_size-1))];
00065         }
```

7.31.4.2 template<typename element, unsigned long size> element const& DSG::LUT< element, size >::operator[] (unsigned long const & *index*) const [inline]

Definition at line 47 of file LUT.h.

```
00047                                                                   {
00048 #ifdef DEBUG
00049             assert(index<_size);
00050 #endif
00051             return _table[index];
00052         }
```

7.31.4.3 template<typename element, unsigned long size> element& DSG::LUT< element, size >::operator[] (unsigned long const & *index*) [inline]

Definition at line 53 of file LUT.h.

```
00053                                                       {
00054 #ifdef DEBUG
00055             assert(index<_size);
00056 #endif
00057             return _table[index];
00058         }
```

7.31.4.4 template<typename element, unsigned long size> unsigned long const& DSG::LUT< element, size >::Size ( ) const [inline]

Definition at line 66 of file LUT.h.

```
00066                                       {
00067             return _size;
00068         }
```

7.31.5 Member Data Documentation

7.31.5.1 template<typename element, unsigned long size> const unsigned long DSG::LUT< element, size >::_size [protected]

Definition at line 71 of file LUT.h.

7.31.5.2 template<typename element, unsigned long size> element DSG::LUT< element, size >::_table[size] [protected]

Definition at line 70 of file LUT.h.

7.31.5.3 template<typename element, unsigned long size> double DSG::LUT< element, size >::phs [protected]

Definition at line 72 of file LUT.h.

The documentation for this class was generated from the following file:

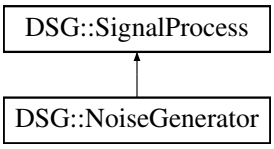
- /Users/alexanderzywicki/Documents/DSG/src/LUT.h

7.32 DSG::NoiseGenerator Class Reference

DSG::NoiseGenerator - Generator that uses noise functions such as DSG::White() to generate signal.

`#include <NoiseGenerator.h>`

Inheritance diagram for DSG::NoiseGenerator:



Public Member Functions

- NoiseGenerator (DSGSample(∗StatelessFunction)(DSGSample))
- virtual ∼NoiseGenerator ()
- virtual bool Perform (DSG::DSGSample &signal)
- virtual bool Perform (DSG::RingBuffer &signal)

Protected Attributes

- DSGSample(∗ _function)(DSGSample)
- DSG::DSGSample _storage

7.32.1 Detailed Description

DSG::NoiseGenerator - Generator that uses noise functions such as DSG::White() to generate signal.

Definition at line 13 of file NoiseGenerator.h.

7.32.2 Constructor & Destructor Documentation

7.32.2.1 DSG::NoiseGenerator::NoiseGenerator (DSGSample(∗)(DSGSample) *StatelessFunction*)

Definition at line 9 of file NoiseGenerator.cpp.

```
00009                                                                    :
      DSG::SignalProcess(){
00010     _function = StatelessFunction;
00011 }
```

7.32.2.2 DSG::NoiseGenerator::∼NoiseGenerator ( ) [virtual]

Definition at line 12 of file NoiseGenerator.cpp.

```
00012 {}
```

7.32.3 Member Function Documentation

7.32.3.1 bool DSG::NoiseGenerator::Perform (DSG::DSGSample & *signal*) [inline], [virtual]

Implements DSG::SignalProcess.

Definition at line 23 of file NoiseGenerator.h.

```
00023                                                    {
00024         signal = _function(0);
00025         return true;
00026     }
```

7.32.3.2 bool DSG::NoiseGenerator::Perform (DSG::RingBuffer & *signal*) [inline], [virtual]

Implements DSG::SignalProcess.

Definition at line 27 of file NoiseGenerator.h.

```
00027                                                                {
00028         signal.Flush();
00029         while (!signal.Full()) {
00030             if (Perform(_storage)) {
00031                 if(signal.Write(_storage)){
00032                 }else return false;
00033             }else return false;
00034         }return true;
00035     }
```

7.32.4 Member Data Documentation

7.32.4.1 DSGSample(∗ DSG::NoiseGenerator::_function)(DSGSample) [protected]

Definition at line 20 of file NoiseGenerator.h.

7.32.4.2 DSG::DSGSample DSG::NoiseGenerator::_storage [protected]

Definition at line 21 of file NoiseGenerator.h.

The documentation for this class was generated from the following files:

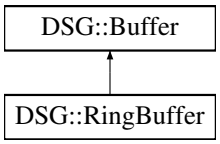
- /Users/alexanderzywicki/Documents/DSG/src/NoiseGenerator.h
- /Users/alexanderzywicki/Documents/DSG/src/NoiseGenerator.cpp

7.33 DSG::RingBuffer Class Reference

DSG::RingBuffer - Circular Buffer of Audio.

`#include <RingBuffer.h>`

Inheritance diagram for DSG::RingBuffer:



Public Member Functions

- RingBuffer ()
- RingBuffer (const size_t size)
- RingBuffer (RingBuffer &buffer)
- RingBuffer & operator= (RingBuffer &buffer)
- virtual ∼RingBuffer ()
- bool Write (const DSGSample &elem)
- bool Read (DSG::DSGSample &elem)
- size_t const & Count () const
- bool Full () const
- bool Empty () const
- void Flush ()

Protected Member Functions

- size_t next (size_t current)
- size_t make_pow_2 (size_t number)

Protected Attributes

- std::atomic< size_t > _write
- std::atomic< size_t > _read
- size_t _count
- size_t MASK
- size_t write
- size_t read

Friends

- bool operator>> (DSG::DSGSample const &signal, DSG::RingBuffer &buffer)
- bool operator<< (DSG::DSGSample &signal, DSG::RingBuffer &buffer)

7.33.1 Detailed Description

DSG::RingBuffer - Circular Buffer of Audio.

Definition at line 19 of file RingBuffer.h.

7.33.2 Constructor & Destructor Documentation

7.33.2.1 DSG::RingBuffer::RingBuffer ( )

Definition at line 9 of file RingBuffer.cpp.

```
00009 :Buffer(0),_read(0),_write(0),_count(0),MASK(0){}
```

7.33.2.2 DSG::RingBuffer::RingBuffer (const size_t *size*)

Definition at line 10 of file RingBuffer.cpp.

```
00010                                           :Buffer(make_pow_2(size)),
      _read(0),_write(0),_count(0){
00011     MASK = this->_size-1;
00012 }
```

7.33.2.3 DSG::RingBuffer::RingBuffer (RingBuffer & *buffer*)

Definition at line 13 of file RingBuffer.cpp.

```
00013                                             :Buffer(buffer){
00014     _write.store(buffer._write.load(std::memory_order_acquire));
00015     _read.store(buffer._read.load(std::memory_order_acquire));
00016     _count = buffer._count;
00017     MASK = buffer._size-1;
00018 }
```

7.33.2.4 DSG::RingBuffer::~RingBuffer ( ) [virtual]

Definition at line 27 of file RingBuffer.cpp.

```
00027 {Flush();}
```

7.33.3 Member Function Documentation

7.33.3.1 size_t const & DSG::RingBuffer::Count ( ) const [inline]

Definition at line 90 of file RingBuffer.h.

```
00090                                                 {
00091         return _count;
00092     }
```

7.33.3.2 bool DSG::RingBuffer::Empty ( ) const [inline]

Definition at line 64 of file RingBuffer.h.

```
00064                                           {
00065         return _count==0;
00066     }
```

7.33.3.3 void DSG::RingBuffer::Flush ( ) [inline]

Definition at line 67 of file RingBuffer.h.

```
00067                                        {
00068         _write.store(0,std::memory_order_relaxed);
00069         _read.store(0,std::memory_order_relaxed);
00070         _count=0;
00071     }
```

7.33.3.4 bool DSG::RingBuffer::Full ( ) const [inline]

Definition at line 61 of file RingBuffer.h.

```
00061                                           {
00062         return _count==this->_size;
00063     }
```

7.33.3.5 size_t DSG::RingBuffer::make_pow_2 (size_t *number*) [inline], [protected]

Definition at line 95 of file RingBuffer.h.

```
00095                                                       {
00096         return pow(2, ceil(log(number)/log(2)));
00097     }
```

7.33.3.6 size_t DSG::RingBuffer::next (size_t *current*) [inline], [protected]

Definition at line 94 of file RingBuffer.h.

```
00094 {return (current+1) & MASK;}
```

7.33.3.7 DSG::RingBuffer & DSG::RingBuffer::operator= (RingBuffer & *buffer*)

Definition at line 19 of file RingBuffer.cpp.

```
00019                                                          {
00020     Buffer::operator=(buffer);
00021     _write.store(buffer._write.load(std::memory_order_acquire));
00022     _read.store(buffer._read.load(std::memory_order_acquire));
00023     _count = buffer._count;
00024     MASK = buffer._size-1;
00025     return *this;
00026 }
```

7.33.3.8 bool DSG::RingBuffer::Read (DSG::DSGSample & *elem*) [inline]

Definition at line 81 of file RingBuffer.h.

```
00081                                                 {
00082         if (!Empty()) {
00083             read = _read.load(std::memory_order_acquire);
00084             _read.store(next(read),std::memory_order_release);
00085             elem = this->_buffer[read];
00086             --_count;
00087             return true;
00088         }else return false;
00089     }
```

7.33.3.9 bool DSG::RingBuffer::Write (const DSGSample & *elem*) [inline]

Definition at line 72 of file RingBuffer.h.

```
00072                                                    {
00073         if (!Full()) {
00074             write = _write.load(std::memory_order_acquire);
00075             _write.store(next(write),std::memory_order_release);
00076             this->_buffer[write] = elem;
00077             ++_count;
00078             return true;
00079         }else return false;
00080     }
```

7.33.4 Friends And Related Function Documentation

7.33.4.1 bool operator<< (DSG::DSGSample & *signal,* DSG::RingBuffer & *buffer*) [friend]

Definition at line 44 of file RingBuffer.h.

```
00044                                                                  {
00045             return buffer.Read(signal);
00046         }
```

7.33.4.2 bool operator>> (DSG::DSGSample const & *signal,* DSG::RingBuffer & *buffer*) [friend]

Definition at line 41 of file RingBuffer.h.

```
00041                                                                        {
00042             return buffer.Write(signal);
00043         }
```

7.33.5 Member Data Documentation

7.33.5.1 size_t DSG::RingBuffer::_count [protected]

Definition at line 23 of file RingBuffer.h.

7.33.5.2 std::atomic<size_t> DSG::RingBuffer::_read [protected]

Definition at line 22 of file RingBuffer.h.

7.33.5.3 std::atomic<size_t> DSG::RingBuffer::_write [protected]

Definition at line 21 of file RingBuffer.h.

7.33.5.4 size_t DSG::RingBuffer::MASK [protected]

Definition at line 24 of file RingBuffer.h.

7.33.5.5 size_t DSG::RingBuffer::read [protected]

Definition at line 26 of file RingBuffer.h.

7.33.5.6 size_t DSG::RingBuffer::write [protected]

Definition at line 25 of file RingBuffer.h.

The documentation for this class was generated from the following files:

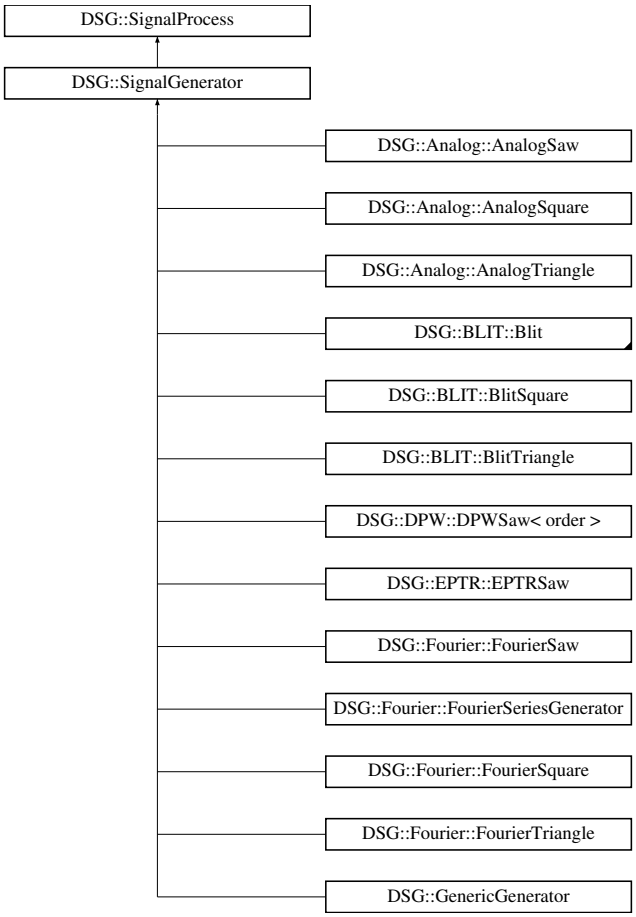
- /Users/alexanderzywicki/Documents/DSG/src/RingBuffer.h
- /Users/alexanderzywicki/Documents/DSG/src/RingBuffer.cpp

7.34 DSG::SignalGenerator Class Reference

DSG::SignalGenerator - Extends DSG::Signal Process With Tools For Signal Generation.

`#include <SignalGenerator.h>`

Inheritance diagram for DSG::SignalGenerator:



Public Member Functions

- SignalGenerator ()
- SignalGenerator (DSG::DSGFrequency const &frequency, DSG::DSGPhase const &offset)
- virtual ~SignalGenerator ()
- virtual bool Perform (DSG::DSGSample &signal)
- virtual bool Perform (DSG::RingBuffer &signal)
- virtual DSG::DSGFrequency const & Frequency ()
- virtual DSG::DSGFrequency const & Frequency (DSG::DSGFrequency const &value)
- virtual DSG::DSGPhase const & Phase ()
- virtual DSG::DSGPhase const & Phase (DSG::DSGPhase const &value)

Protected Member Functions

- void step ()
- void sync ()

Protected Attributes

- DSG::DSGFrequency _frequency
- DSG::DSGPhase _dt
- DSG::DSGPhase _offset
- DSG::DSGPhase _phasor
- DSG::DSGSample _storage

7.34.1 Detailed Description

DSG::SignalGenerator - Extends DSG::Signal Process With Tools For Signal Generation.

Definition at line 17 of file SignalGenerator.h.

7.34.2 Constructor & Destructor Documentation

7.34.2.1 DSG::SignalGenerator::SignalGenerator ( )

Definition at line 9 of file SignalGenerator.cpp.

```
00009 :DSG::SignalProcess(),_phasor(0),_frequency(0),
      _dt(0),_offset(0){}
```

7.34.2.2 DSG::SignalGenerator::SignalGenerator (DSG::DSGFrequency const & *frequency,* DSG::DSGPhase const & *offset*)

Definition at line 10 of file SignalGenerator.cpp.

```
00010                                                                                                :
      _phasor(0),_frequency(frequency),_dt(0),_offset(offset){
00011     Frequency(frequency);
00012     Phase(offset);
00013 }
```

7.34.2.3 DSG::SignalGenerator::~SignalGenerator ( ) [virtual]

Definition at line 14 of file SignalGenerator.cpp.

```
00014 {}
```

7.34.3 Member Function Documentation

7.34.3.1 DSG::DSGFrequency const & DSG::SignalGenerator::Frequency ( ) [inline], [virtual]

Definition at line 54 of file SignalGenerator.h.

```
00054                                                {
00055     return _frequency;
00056 }
```

7.34.3.2 DSG::DSGFrequency const & DSG::SignalGenerator::Frequency (DSG::DSGFrequency const & *value*) [inline], [virtual]

Reimplemented in DSG::BLIT::Blit, DSG::BLIT::BlitSaw, DSG::Fourier::FourierSaw, DSG::Fourier::FourierSquare, and DSG::Fourier::FourierTriangle.

Definition at line 57 of file SignalGenerator.h.

```
00057                                                                                                {
00058     _frequency = DSG::EnforceBounds<0, 20000,DSG::DSGSample>(value);
00059     _dt = _frequency/DSG::SampleRate();
00060     return _frequency;
00061 }
```

7.34.3.3 bool DSG::SignalGenerator::Perform (DSG::DSGSample & *signal*) [inline], [virtual]

Implements DSG::SignalProcess.

Reimplemented in DSG::Fourier::FourierSeriesGenerator, DSG::BLIT::Blit, DSG::DPW::DPWSaw< order >, DSG::EPTR::EPTRSaw, DSG::Analog::AnalogSaw, DSG::Analog::AnalogSquare, DSG::Analog::AnalogTriangle, DSG::BLIT::BlitSaw, DSG::Fourier::FourierSaw, DSG::Fourier::FourierSquare, DSG::Fourier::FourierTriangle, and DSG::GenericGenerator.

Definition at line 46 of file SignalGenerator.h.

```
00046                                                    {
00047     signal=0;
00048     return false;
00049 }
```

7.34.3.4 bool DSG::SignalGenerator::Perform (DSG::RingBuffer & *signal*) [inline], [virtual]

Implements DSG::SignalProcess.

Reimplemented in DSG::DPW::DPWSaw< order >, DSG::Fourier::FourierSeriesGenerator, DSG::BLIT::Blit, DSG::EPTR::EPTRSaw, DSG::Analog::AnalogSaw, DSG::Analog::AnalogSquare, DSG::Analog::AnalogTriangle, DSG::BLIT::BlitSaw, DSG::Fourier::FourierSaw, DSG::Fourier::FourierSquare, DSG::Fourier::FourierTriangle, and DSG::GenericGenerator.

Definition at line 50 of file SignalGenerator.h.

```
00050                                                    {
00051     signal.Flush();
00052     return false;
00053 }
```

7.34.3.5 DSG::DSGPhase const & DSG::SignalGenerator::Phase ( ) [inline], [virtual]

Definition at line 62 of file SignalGenerator.h.

```
00062                                                {
00063     return _offset;
00064 }
```

7.34.3.6 DSG::DSGPhase const & DSG::SignalGenerator::Phase (DSG::DSGPhase const & *value*) [inline], [virtual]

Definition at line 65 of file SignalGenerator.h.

```
00065                                                                    {
00066     _offset-=value;
00067     _phasor-=_offset;
00068     _offset=value;
00069     return _offset;
00070 }
```

7.34.3.7 void DSG::SignalGenerator::step ( ) [inline], [protected]

Definition at line 71 of file SignalGenerator.h.

```
00071                                    {
00072     _phasor+=_dt;
00073     _phasor>1.0 ? --_phasor:0;
00074 }
```

7.34.3.8 void DSG::SignalGenerator::sync ( ) [inline], [protected]

Definition at line 75 of file SignalGenerator.h.

```
00075                                {
00076     _phasor=_offset;
00077 }
```

7.34.4 Member Data Documentation

7.34.4.1 DSG::DSGPhase DSG::SignalGenerator::_dt [protected]

Definition at line 35 of file SignalGenerator.h.

7.34.4.2 DSG::DSGFrequency DSG::SignalGenerator::_frequency [protected]

Definition at line 34 of file SignalGenerator.h.

7.34.4.3 DSG::DSGPhase DSG::SignalGenerator::_offset [protected]

Definition at line 36 of file SignalGenerator.h.

7.34.4.4 DSG::DSGPhase DSG::SignalGenerator::_phasor [protected]

Definition at line 37 of file SignalGenerator.h.

7.34.4.5 DSG::DSGSample DSG::SignalGenerator::_storage [protected]

Definition at line 38 of file SignalGenerator.h.

The documentation for this class was generated from the following files:

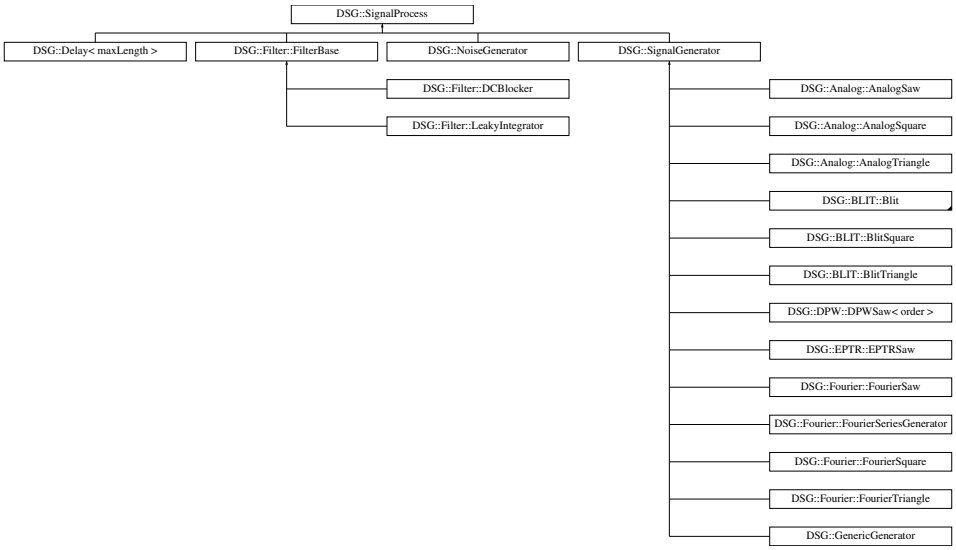
- /Users/alexanderzywicki/Documents/DSG/src/SignalGenerator.h
- /Users/alexanderzywicki/Documents/DSG/src/SignalGenerator.cpp

7.35 DSG::SignalProcess Class Reference

DSG::SignalProcess - Defines Base Interface For Audio Processing.

`#include <SignalProcess.h>`

Inheritance diagram for DSG::SignalProcess:



Public Member Functions

- SignalProcess ()
- virtual ~SignalProcess ()
- virtual bool Perform (DSG::DSGSample &signal)=0
- virtual bool Perform (DSG::RingBuffer &signal)=0

7.35.1 Detailed Description

DSG::SignalProcess - Defines Base Interface For Audio Processing.

Definition at line 15 of file SignalProcess.h.

7.35.2 Constructor & Destructor Documentation

7.35.2.1 DSG::SignalProcess::SignalProcess ( )

Definition at line 9 of file SignalProcess.cpp.

```
00009 {}
```

7.35.2.2 DSG::SignalProcess::~SignalProcess ( ) [virtual]

Definition at line 10 of file SignalProcess.cpp.

```
00010 {}
```

7.35.3 Member Function Documentation

7.35.3.1 virtual bool DSG::SignalProcess::Perform (DSG::DSGSample & *signal*) [inline], [pure virtual]

Implemented in DSG::Delay< maxLength >, DSG::Fourier::FourierSeriesGenerator, DSG::BLIT::Blit, DSG::DPW::DPWSaw< order >, DSG::EPTR::EPTRSaw, DSG::Filter::LeakyIntegrator, DSG::Analog::AnalogSaw, DSG::Analog::AnalogSquare, DSG::Analog::AnalogTriangle, DSG::BLIT::BlitSaw, DSG::Fourier::FourierSaw, DSG::Fourier::FourierSquare, DSG::Fourier::FourierTriangle, DSG::Filter::FilterBase, DSG::SignalGenerator, DSG::Filter::DCBlocker, DSG::GenericGenerator, and DSG::NoiseGenerator.

7.35.3.2 virtual bool DSG::SignalProcess::Perform (DSG::RingBuffer & *signal*) [inline], [pure virtual]

Implemented in DSG::Delay< maxLength >, DSG::DPW::DPWSaw< order >, DSG::Fourier::FourierSeriesGenerator, DSG::BLIT::Blit, DSG::EPTR::EPTRSaw, DSG::Filter::LeakyIntegrator, DSG::Analog::AnalogSaw, DSG::Analog::AnalogSquare, DSG::Analog::AnalogTriangle, DSG::BLIT::BlitSaw, DSG::Fourier::FourierSaw, DSG::Fourier::FourierSquare, DSG::Fourier::FourierTriangle, DSG::Filter::FilterBase, DSG::SignalGenerator, DSG::Filter::DCBlocker, DSG::GenericGenerator, and DSG::NoiseGenerator.

The documentation for this class was generated from the following files:

- /Users/alexanderzywicki/Documents/DSG/src/SignalProcess.h
- /Users/alexanderzywicki/Documents/DSG/src/SignalProcess.cpp

Chapter 8

File Documentation

8.1 /Users/alexanderzywicki/Documents/DSG/src/AnalogSaw.cpp File Reference

```
#include "AnalogSaw.h"
```

8.2 AnalogSaw.cpp

```
00001 //
00002 //  AnalogSaw.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/17/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "AnalogSaw.h"
00009 DSG::Analog::AnalogSaw::AnalogSaw():DSG::
      SignalGenerator(){}
00010 DSG::Analog::AnalogSaw::AnalogSaw(
      DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):
      DSG::SignalGenerator(frequency,offset){}
00011 DSG::Analog::AnalogSaw::~AnalogSaw(){}
```

8.3 /Users/alexanderzywicki/Documents/DSG/src/AnalogSaw.h File Reference

```
#include "SignalGenerator.h"
```

Classes

- class DSG::Analog::AnalogSaw

  DSG::Analog::AnalogSaw - Analog Syle Saw Wave Generator.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::Analog

  DSG::Analog - Namespace Containing Analog Style Oscillators.

8.4 AnalogSaw.h

```
00001 //
00002 //  AnalogSaw.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/17/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__AnalogSaw__
00009 #define __DSG__AnalogSaw__
00010 #include "SignalGenerator.h"
00011 namespace DSG{
00012 #ifdef DSG_Short_Names
00013     inline
00014 #endif
00015     //! DSG::Analog - Namespace Containing Analog Style Oscillators
00016     namespace Analog{
00017         //!\brief DSG::Analog::AnalogSaw - Analog Syle Saw Wave Generator
00018         class AnalogSaw : public DSG::SignalGenerator {
00019         public:
00020             AnalogSaw();
00021             AnalogSaw(DSG::DSGFrequency const& frequency,
      DSG::DSGPhase const& offset);
00022             virtual ~AnalogSaw();
00023             virtual inline bool Perform(DSG::DSGSample& signal);
00024             virtual inline bool Perform(DSG::RingBuffer& signal);
00025         protected:
00026             DSG::DSGSample _stor;
00027         };
00028         inline bool DSG::Analog::AnalogSaw::Perform(
      DSG::DSGSample& signal){
00029             _stor=_phasor;
00030             _stor+=0.5;
00031             if (_stor>1.0) {
00032                 --_stor;
00033             }
00034             _stor-=0.5;
00035             _stor*=2.0;
00036             signal=_stor;
00037             step();
00038             return true;
00039         }
00040         inline bool DSG::Analog::AnalogSaw::Perform(
      DSG::RingBuffer& signal){
00041             signal.Flush();
00042             while (!signal.Full()) {
00043                 if (Perform(_storage)) {
00044                     if(signal.Write(_storage)){
00045                     }else return false;
00046                 }else return false;
00047             }return true;
00048         }
00049     }
00050 }
00051 #endif /* defined(__DSG__AnalogSaw__) */
```

8.5 /Users/alexanderzywicki/Documents/DSG/src/AnalogSquare.cpp File Reference

```
#include "AnalogSquare.h"
```

8.6 AnalogSquare.cpp

```
00001 //
00002 //  AnalogSquare.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/17/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "AnalogSquare.h"
00009 DSG::Analog::AnalogSquare::AnalogSquare():
      DSG::SignalGenerator(){}
00010 DSG::Analog::AnalogSquare::AnalogSquare(
      DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):
      DSG::SignalGenerator(frequency,offset){}
```

```
00011 DSG::Analog::AnalogSquare::~AnalogSquare(){}
```

8.7 /Users/alexanderzywicki/Documents/DSG/src/AnalogSquare.h File Reference

```
#include "SignalGenerator.h"
```

Classes

- class DSG::Analog::AnalogSquare

  DSG::Analog::AnalogSquare - Analog Syle Square Wave Generator.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::Analog

  DSG::Analog - Namespace Containing Analog Style Oscillators.

8.8 AnalogSquare.h

```
00001 //
00002 //  AnalogSquare.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/17/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__AnalogSquare__
00009 #define __DSG__AnalogSquare__
00010 #include "SignalGenerator.h"
00011 namespace DSG{
00012 #ifdef DSG_Short_Names
00013     inline
00014 #endif
00015     //! DSG::Analog - Namespace Containing Analog Style Oscillators
00016     namespace Analog{
00017         //!\brief DSG::Analog::AnalogSquare - Analog Syle Square Wave Generator
00018         class AnalogSquare : public DSG::SignalGenerator {
00019         public:
00020             AnalogSquare();
00021             AnalogSquare(DSG::DSGFrequency const& frequency,
      DSG::DSGPhase const& offset);
00022             virtual ~AnalogSquare();
00023             virtual inline bool Perform(DSG::DSGSample& signal);
00024             virtual inline bool Perform(DSG::RingBuffer& signal);
00025         };
00026         inline bool DSG::Analog::AnalogSquare::Perform(
      DSG::DSGSample& signal){
00027             signal=_phasor < 0.5 ? 1.0:-1.0;
00028             step();
00029             return true;
00030         }
00031         inline bool DSG::Analog::AnalogSquare::Perform(
      DSG::RingBuffer& signal){
00032             signal.Flush();
00033             while (!signal.Full()) {
00034                 if (Perform(_storage)) {
00035                     if(signal.Write(_storage)){
00036                     }else return false;
00037                 }else return false;
00038             }return true;
00039         }
00040     }
00041 }
00042 #endif /* defined(__DSG__AnalogSquare__) */
```

8.9 /Users/alexanderzywicki/Documents/DSG/src/AnalogTriangle.cpp File Reference

```
#include "AnalogTriangle.h"
```

8.10 AnalogTriangle.cpp

```
00001 //
00002 //  AnalogTriangle.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/17/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "AnalogTriangle.h"
00009 DSG::Analog::AnalogTriangle::AnalogTriangle():
      DSG::SignalGenerator(){}
00010 DSG::Analog::AnalogTriangle::AnalogTriangle(
      DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):
      DSG::SignalGenerator(frequency,offset){}
00011 DSG::Analog::AnalogTriangle::~AnalogTriangle(){}
```

8.11 /Users/alexanderzywicki/Documents/DSG/src/AnalogTriangle.h File Reference

```
#include "SignalGenerator.h"
```

Classes

- class DSG::Analog::AnalogTriangle

  DSG::Analog::AnalogTriangle - Analog Syle Triangle Wave Generator.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::Analog

  DSG::Analog - Namespace Containing Analog Style Oscillators.

8.12 AnalogTriangle.h

```
00001 //
00002 //  AnalogTriangle.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/17/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__AnalogTriangle__
00009 #define __DSG__AnalogTriangle__
00010 #include "SignalGenerator.h"
00011 namespace DSG{
00012 #ifdef DSG_Short_Names
00013     inline
00014 #endif
00015     //! DSG::Analog - Namespace Containing Analog Style Oscillators
00016     namespace Analog{
00017         //!\brief DSG::Analog::AnalogTriangle - Analog Syle Triangle Wave Generator
00018         class AnalogTriangle : public DSG::SignalGenerator {
00019         public:
00020             AnalogTriangle();
00021             AnalogTriangle(DSG::DSGFrequency const& frequency,
```

```
      DSG::DSGPhase const& offset);
00022             virtual ~AnalogTriangle();
00023             virtual inline bool Perform(DSG::DSGSample& signal);
00024             virtual inline bool Perform(DSG::RingBuffer& signal);
00025         protected:
00026             DSG::DSGSample _stor;
00027         };
00028         inline bool DSG::Analog::AnalogTriangle::Perform(
      DSG::DSGSample& signal){
00029             _stor = _phasor;
00030             _stor+=0.25;
00031             while (_stor>1.0) {
00032                 _stor-=1.0;
00033             }
00034             _stor-=0.5;
00035             if (_stor<0) {
00036                 _stor*=-1.0;
00037             }
00038             _stor-=0.25;
00039             _stor*=-4.0;
00040             signal = _stor;
00041             step();//always last
00042             return true;
00043         }
00044         inline bool DSG::Analog::AnalogTriangle::Perform(
      DSG::RingBuffer& signal){
00045             signal.Flush();
00046             while (!signal.Full()) {
00047                 if (Perform(_storage)) {
00048                     if(signal.Write(_storage)){
00049                     }else return false;
00050                 }else return false;
00051             }return true;
00052         }
00053     }
00054 }
00055 #endif /* defined(__DSG__AnalogTriangle__) */
```

8.13 /Users/alexanderzywicki/Documents/DSG/src/AudioSettings.cpp File Reference

```
#include "AudioSettings.h"
```

8.14 AudioSettings.cpp

```
00001 //
00002 //  AudioSettings.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "AudioSettings.h"
00009 DSG::DSGFrequency DSG::AudioSettings::_sampleRate;
00010 DSG::DSGFrequency DSG::AudioSettings::_nyquist;
00011 DSG::DSGFrequency const& DSG::AudioSettings::SampleRate(){
00012     return _sampleRate;
00013 }
00014 DSG::DSGFrequency const& DSG::AudioSettings::SampleRate(
      DSG::DSGFrequency const& value){
00015     _sampleRate = value;
00016     _nyquist = _sampleRate*0.5;
00017     return _sampleRate;
00018 }
00019 DSG::DSGFrequency const& DSG::AudioSettings::Nyquist(){
00020     return _nyquist;
00021 }
```

8.15 /Users/alexanderzywicki/Documents/DSG/src/AudioSettings.h File Reference

```
#include "DSGTypes.h"
```

Classes

- class DSG::AudioSettings

  DSG::AudioSettings - Global Storage For Audio Settings Such As Sample Rate.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.

Functions

- DSG::DSGFrequency const & DSG::SampleRate ()

  DSG::SampleRate - Get Global Sample Rate.
- DSG::DSGFrequency const & DSG::SampleRate (DSG::DSGFrequency const &value)

  DSG::SampleRate - Set Global Sample Rate.
- DSG::DSGFrequency DSG::Nyquist ()

  DSG::Nyquist() - Pre-Calculated Nyquist Limit. Use instead of calculating each time needed. This value will be updated whenever the sample rate changes.

8.16 AudioSettings.h

```
00001 //
00002 //  AudioSettings.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__AudioSettings__
00009 #define __DSG__AudioSettings__
00010 #include "DSGTypes.h"
00011 namespace DSG {
00012     /*!\brief DSG::AudioSettings - Global Storage For Audio Settings Such As Sample Rate
00013      */
00014     class AudioSettings{
00015     public:
00016         static DSG::DSGFrequency const& SampleRate();
00017         static DSG::DSGFrequency const& SampleRate(
      DSG::DSGFrequency const& value);
00018         static DSG::DSGFrequency const& Nyquist();
00019     protected:
00020         static DSG::DSGFrequency _sampleRate;
00021         static DSG::DSGFrequency _nyquist;
00022     };
00023     //!\brief DSG::SampleRate - Get Global Sample Rate
00024     inline DSG::DSGFrequency const& SampleRate(){
00025         return DSG::AudioSettings::SampleRate();
00026     }
00027     //!\brief DSG::SampleRate - Set Global Sample Rate
00028     inline DSG::DSGFrequency const& SampleRate(
      DSG::DSGFrequency const& value){
00029         return DSG::AudioSettings::SampleRate(value);
00030     }
00031     //!\brief DSG::Nyquist() - Pre-Calculated Nyquist Limit. Use instead of calculating each time needed.
       This value will be updated whenever the sample rate changes.
00032     inline DSG::DSGFrequency Nyquist(){
00033         return DSG::AudioSettings::Nyquist();
00034     }
00035 }
00036 #endif /* defined(__DSG__AudioSettings__) */
```

8.17 /Users/alexanderzywicki/Documents/DSG/src/Blackman.h File Reference

```
#include "PI.h"
```

```
#include "LUT.h"
#include "Sine.h"
```

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::Window

  DSG::Window - Window functions and utilities.

Functions

- template<typename decimal >
  decimal DSG::Window::Blackman (decimal const &x)

  DSG::Window::Blackman - Blackman Window Function.

8.18 Blackman.h

```
00001     //
00002 //  Blackman.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/24/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef DSG_Blackman_h
00009 #define DSG_Blackman_h
00010 #include "PI.h"
00011 #include "LUT.h"
00012 #include "Sine.h"
00013 namespace DSG {
00014 #ifdef DSG_Short_Names
00015     inline
00016 #endif
00017     namespace Window{
00018         //!\brief DSG::Window::Blackman - Blackman Window Function
00019         template<typename decimal>
00020         inline decimal Blackman(decimal const& x){
00021             // Generate Blackman Window
00022             /*
00023              Blackman(x) = 0.42f - (0.5f * cos(2pi*x)) + (0.08f * cos(2pi*2.0*x));
00024              }*/
00025             static_assert(std::is_floating_point<decimal>::value==true,"DSG::Blackman Function Requires
       Floating Point Type");
00026             //we will implement the blackman window as a function as if it were sin(x)
00027             //cos input domain 0-1 not 0-2pi
00028             //range checking is handles within DSG::Cos
00029             decimal phs=x;
00030             while (phs>1.0) {
00031                 phs-=1.0;
00032             }
00033             return 0.42 - (0.5 * DSG::Cos(phs))+(0.08 * DSG::Cos(2.0*phs));
00034         }
00035     }
00036 }
00037 #endif
```

8.19 /Users/alexanderzywicki/Documents/DSG/src/BLIT.cpp File Reference

```
#include "BLIT.h"
```

8.20 BLIT.cpp

```
00001 //
00002 //  BLIT.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/17/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "BLIT.h"
00009 DSG::BLIT::Blit::Blit():DSG::SignalGenerator(){
00010     Frequency(0);
00011 }
00012 DSG::BLIT::Blit::Blit(DSG::DSGFrequency const& frequency,
      DSG::DSGPhase const& offset):DSG::SignalGenerator(frequency,offset){
00013     Frequency(frequency);
00014 }
00015 DSG::BLIT::Blit::~Blit(){}
```

8.21 /Users/alexanderzywicki/Documents/DSG/src/BLIT.h File Reference

```
#include "SignalGenerator.h"
#include "Denormal.h"
#include "Sinc.h"
#include "DSGMath.h"
```

Classes

- class DSG::BLIT::Blit

  DSG::BLIT::Blit - Band-Limited Impulse Train Generator.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::BLIT

  DSG::BLIT - Namespace Containing BLIT Based Oscillators.

8.22 BLIT.h

```
00001 //
00002 //  BLIT.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/17/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__BLIT__
00009 #define __DSG__BLIT__
00010 #include "SignalGenerator.h"
00011 #include "Denormal.h"
00012 #include "Sinc.h"
00013 #include "DSGMath.h"
00014 namespace DSG{
00015 #ifdef DSG_Short_Names
00016     inline
00017 #endif
00018     //!DSG::BLIT - Namespace Containing BLIT Based Oscillators
00019     namespace BLIT{
00020         /*!\brief DSG::BLIT::Blit - Band-Limited Impulse Train Generator
00021          */
00022         //!\todo Re-write DSG::BLIT::Blit algorithm
00023         class Blit:public DSG::SignalGenerator{
00024         public:
00025             Blit();
```

```
00026             Blit(DSG::DSGFrequency const& frequency,
      DSG::DSGPhase const& offset);
00027             virtual ~Blit();
00028             virtual inline bool Perform(DSG::DSGSample& signal);
00029             virtual inline bool Perform(DSG::RingBuffer& signal);
00030             virtual inline DSG::DSGFrequency const& Frequency(
      DSG::DSGFrequency const& value);
00031         protected:
00032             unsigned long p_;
00033             unsigned long m_;
00034             unsigned long _h;
00035             double a_;
00036             DSG::DSGSample denominator;
00037             DSG::DSGSample value;
00038         };
00039         inline bool DSG::BLIT::Blit::Perform(
      DSG::DSGSample& signal){
00040             //found better results in this case with built in sine function. not performance wise but
       algorithmically
00041             denominator = m_ * sin(PI*_phasor);
00042             if (DSG::IsDenormal(denominator)) {
00043                 signal = a_;
00044             }else{
00045                 value = sin(PI*_phasor * m_);
00046                 value/=denominator;
00047                 value*=a_;
00048                 signal = value;
00049             }
00050             step();
00051             return true;
00052         }
00053         inline bool DSG::BLIT::Blit::Perform(
      DSG::RingBuffer& signal){
00054             signal.Flush();
00055             while (!signal.Full()) {
00056                 if (Perform(_storage)) {
00057                     if(signal.Write(_storage)){
00058                     }else return false;
00059                 }else return false;
00060             }return true;
00061         }
00062         inline DSG::DSGFrequency const&
      DSG::BLIT::Blit::Frequency(DSG::DSGFrequency const& value){
00063             this->SignalGenerator::Frequency(value);
00064             p_ = DSG::SampleRate()/_frequency;
00065             _h = (unsigned)floor(p_*0.5);
00066             m_ = 2 * (_h)+1;
00067             a_ = m_/(double)p_;
00068             return _frequency;
00069         }
00070     }
00071 }
00072 #endif /* defined(__DSG__BLIT__) */
```

8.23 /Users/alexanderzywicki/Documents/DSG/src/BLITSaw.cpp File Reference

```
#include "BLITSaw.h"
```

8.24 BLITSaw.cpp

```
00001 //
00002 //  BLITSaw.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/17/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "BLITSaw.h"
00009 DSG::BLIT::BlitSaw::BlitSaw():DSG::BLIT::Blit(),Register_(0){
00010     Frequency(0);
00011 }
00012 DSG::BLIT::BlitSaw::BlitSaw(DSG::DSGFrequency const& frequency,
      DSG::DSGPhase const& offset):DSG::BLIT::Blit(frequency,offset),Register_(0){
00013     Frequency(frequency);
00014 }
00015 DSG::BLIT::BlitSaw::~BlitSaw(){}
```

8.25 /Users/alexanderzywicki/Documents/DSG/src/BLITSaw.h File Reference

```
#include "BLIT.h"
```

Classes

- class DSG::BLIT::BlitSaw

  DSG::BLIT::BlitSaw - Saw Wave Generator Based on BLIT Algorithm.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::BLIT

  DSG::BLIT - Namespace Containing BLIT Based Oscillators.

8.26 BLITSaw.h

```
00001 //
00002 //  BLITSaw.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/17/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__BLITSaw__
00009 #define __DSG__BLITSaw__
00010 #include "BLIT.h"
00011 namespace DSG{
00012 #ifdef DSG_Short_Names
00013     inline
00014 #endif
00015     namespace BLIT{
00016         //!\brief DSG::BLIT::BlitSaw - Saw Wave Generator Based on BLIT Algorithm
00017         //!\todo Re-write DSG::BLIT::BlitSaw algorithm
00018         class BlitSaw : public Blit{
00019         public:
00020             BlitSaw();
00021             BlitSaw(DSG::DSGFrequency const& frequency,
      DSG::DSGPhase const& offset);
00022             virtual ~BlitSaw();
00023             virtual inline bool Perform(DSG::DSGSample& signal);
00024             virtual inline bool Perform(DSG::RingBuffer& signal);
00025             virtual inline DSG::DSGFrequency const& Frequency(
      DSG::DSGFrequency const& value);
00026         protected:
00027             DSG::DSGSample C2_;
00028             DSG::DSGSample Register_;
00029         };
00030         inline bool DSG::BLIT::BlitSaw::Perform(
      DSG::DSGSample& signal){
00031             denominator = m_ * sin(PI*_phasor);
00032             if (DSG::IsDenormal(denominator)) {
00033                 signal = a_;
00034             }else{
00035                 value = sin(PI*_phasor * m_);
00036                 value/=denominator;
00037                 value*=a_;
00038                 signal = value;
00039             }
00040             step();
00041             signal += (Register_ - C2_);
00042             Register_ = signal * 0.995;
00043             C2_+=signal;
00044             C2_*=0.5;
00045             return true;
00046         }
00047         inline bool DSG::BLIT::BlitSaw::Perform(
      DSG::RingBuffer& signal){
00048             signal.Flush();
```

```
00049             while (!signal.Full()) {
00050                 if (Perform(_storage)) {
00051                     if(signal.Write(_storage)){
00052                     }else return false;
00053                 }else return false;
00054             }return true;
00055         }
00056         inline DSG::DSGFrequency const&
      DSG::BLIT::BlitSaw::Frequency(DSG::DSGFrequency const& value)
      {
00057             this->SignalGenerator::Frequency(value);
00058             p_ = DSG::SampleRate()/_frequency;
00059             _h = (unsigned)floor(p_*0.5);
00060             m_ = 2 * (_h)+1;
00061             a_ = m_/(double)p_;
00062             C2_ = 1.0/(double)p_;
00063             return _frequency;
00064         }
00065     }
00066 }
00067 #endif /* defined(__DSG__BLITSaw__) */
```

8.27 /Users/alexanderzywicki/Documents/DSG/src/BLITSquare.cpp File Reference

```
#include "BLITSquare.h"
```

8.28 BLITSquare.cpp

```
00001 //
00002 //  BLITSquare.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/17/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "BLITSquare.h"
```

8.29 /Users/alexanderzywicki/Documents/DSG/src/BLITSquare.h File Reference

```
#include "SignalGenerator.h"
```

Classes

- class DSG::BLIT::BlitSquare

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::BLIT

  DSG::BLIT - Namespace Containing BLIT Based Oscillators.

8.30 BLITSquare.h

```
00001 //
00002 //  BLITSquare.h
00003 //  DSG
00004 //
```

```
00005 //  Created by Alexander Zywicki on 9/17/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__BLITSquare__
00009 #define __DSG__BLITSquare__
00010 #include "SignalGenerator.h"
00011 namespace DSG {
00012 #ifdef DSG_Short_Names
00013     inline
00014 #endif
00015     namespace BLIT{
00016         //!\todo Write DSG::BLIT::BlitSquare algorithm
00017         class BlitSquare:public DSG::SignalGenerator{
00018
00019         };
00020     }
00021 }
00022 #endif /* defined(__DSG__BLITSquare__) */
```

8.31 /Users/alexanderzywicki/Documents/DSG/src/BLITTriangle.cpp File Reference

```
#include "BLITTriangle.h"
```

8.32 BLITTriangle.cpp

```
00001 //
00002 //  BLITTriangle.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/17/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "BLITTriangle.h"
```

8.33 /Users/alexanderzywicki/Documents/DSG/src/BLITTriangle.h File Reference

```
#include "SignalGenerator.h"
```

Classes

- class DSG::BLIT::BlitTriangle

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::BLIT

  DSG::BLIT - Namespace Containing BLIT Based Oscillators.

8.34 BLITTriangle.h

```
00001 //
00002 //  BLITTriangle.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/17/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
```

```
00008
00009 #ifndef __DSG__BLITTriangle__
00010 #define __DSG__BLITTriangle__
00011
00012 #include "SignalGenerator.h"
00013 namespace DSG {
00014 #ifdef DSG_Short_Names
00015     inline
00016 #endif
00017     namespace BLIT{
00018         //!\todo Write DSG::BLIT::BlitTriangle algorithm
00019         class BlitTriangle:public DSG::SignalGenerator{
00020
00021         };
00022     }
00023 }
00024 #endif /* defined(__DSG__BLITTriangle__) */
```

8.35 /Users/alexanderzywicki/Documents/DSG/src/Bounds.h File Reference

`#include <assert.h>`

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.

Functions

- template<int lower, int upper, typename decimal >
  decimal DSG::EnforceBounds (decimal const &value)

  DSG::EnforceBounds - Clip value to set bounds.
- template<int lower, int upper, int value>
  void DSG::StaticAssertBounds ()

  DSG::StaticAssertBounds - Fails on compile time if value is not within bounds.
- template<int lower, int upper, typename T >
  void DSG::AssertBounds (T const &value)

  DSG::AssertBounds - Fails on runtime if value is not within bounds.

8.36 Bounds.h

```
00001 //
00002 //  Bounds.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 11/11/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef DSG_Bounds_h
00009 #define DSG_Bounds_h
00010 #include <assert.h>
00011 namespace DSG{
00012     //!\brief DSG::EnforceBounds - Clip value to set bounds
00013     template<int lower,int upper,typename decimal>
00014     decimal EnforceBounds(decimal const& value){
00015         if (value<lower) {
00016             return lower;
00017         }else if(value> upper){
00018             return upper;
00019         }else return value;
00020     }
00021     //!\brief DSG::StaticAssertBounds - Fails on compile time if value is not within bounds
00022     template<int lower,int upper,int value>
00023     void StaticAssertBounds(){
```

```
00024         static_assert(value>=lower && value<=upper,"Failed Static Bounds Assert");
00025     }
00026     //!\brief DSG::AssertBounds - Fails on runtime if value is not within bounds
00027     template<int lower,int upper,typename T>
00028     void AssertBounds(T const& value){
00029         assert(value>=lower && value<=upper);
00030     }
00031 }
00032 #endif
```

8.37 /Users/alexanderzywicki/Documents/DSG/src/Buffer.cpp File Reference

```
#include "Buffer.h"
```

8.38 Buffer.cpp

```
00001 //
00002 //  Buffer.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "Buffer.h"
00009 DSG::Buffer::Buffer():_size(0),_buffer(nullptr){}
00010 DSG::Buffer::Buffer(size_t size):_size(size),_buffer(new
      DSG::DSGSample[size]){}
00011 DSG::Buffer::Buffer(Buffer const& other) {
00012     _buffer = new  DSG::DSGSample[_size];
00013     _size = other._size;
00014     *this = other;
00015 }
00016 DSG::Buffer& DSG::Buffer::operator=(Buffer const& other){
00017     if (_size!=other._size) {
00018         if (_buffer!=nullptr) {
00019             delete [] _buffer;
00020         }
00021         _size = other._size;
00022         _buffer = new  DSG::DSGSample[_size];
00023     }
00024     for (int i=0; i<_size; ++i) {
00025         _buffer[i] = other._buffer[i];
00026     }
00027     return *this;
00028 }
00029 DSG::Buffer::~Buffer(){
00030     if (_buffer!=nullptr) {
00031         delete [] _buffer;
00032     }
00033 }
00034 DSG::DSGSample& DSG::Buffer::operator[](size_t const& index){
00035 #ifdef DEBUG
00036     assert(index<_size);
00037 #endif
00038     return _buffer[index];
00039 }
```

8.39 /Users/alexanderzywicki/Documents/DSG/src/Buffer.h File Reference

```
#include <stddef.h>
#include "DSGTypes.h"
```

Classes

- class DSG::Buffer

  DSG::Buffer - Base Class For DSG::RingBuffer. Not For Direct Use.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.

8.40 Buffer.h

```
00001 //
00002 //  Buffer.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__Buffer__
00009 #define __DSG__Buffer__
00010 #include <stddef.h>
00011 #include "DSGTypes.h"
00012 #ifdef DEBUG
00013 #include <assert.h>
00014 #endif
00015 namespace DSG{
00016     /*!\brief DSG::Buffer - Base Class For DSG::RingBuffer. Not For Direct Use
00017      */
00018     class Buffer {
00019     public:
00020         Buffer();
00021         Buffer(size_t size);
00022         Buffer(Buffer const& other);
00023         Buffer& operator=(Buffer const& other);
00024         virtual ~Buffer();
00025         DSG::DSGSample& operator[](size_t const& index);
00026         inline size_t const& Size()const;
00027     protected:
00028         DSG::DSGSample* _buffer;
00029         size_t _size;
00030     };
00031     inline size_t const& DSG::Buffer::Size()const{
00032         return _size;
00033     }
00034 }
00035 #endif /* defined(__DSG__Buffer__) */
```

8.41 /Users/alexanderzywicki/Documents/DSG/src/BufferConversion.h File Reference

```
#include "RingBuffer.h"
```

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.

Functions

- bool DSG::RingToArray (DSG::RingBuffer &ring, DSG::DSGSample *array, unsigned long length)

  DSG::RingToArray - Move Ring Buffer data to an array.
- bool DSG::ArrayToRing (DSG::RingBuffer &ring, DSG::DSGSample *array, unsigned long length)

  DSG::ArrayToRing - Move array data to a Ring Buffer.

8.42 BufferConversion.h

```
00001 //
```

```
00002 //  BufferConversion.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/14/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__BufferConversion__
00009 #define __DSG__BufferConversion__
00010 #include "RingBuffer.h"
00011 namespace DSG {
00012     //!\brief DSG::RingToArray - Move Ring Buffer data to an array
00013     inline bool RingToArray(DSG::RingBuffer& ring,
      DSG::DSGSample* array,unsigned long length){
00014         for (int i=0; i<length; ++i) {
00015             if (!ring.Empty()) {
00016                 ring.Read(array[i]);
00017             }
00018         }return true;
00019     }
00020     //!\brief DSG::ArrayToRing - Move array data to a Ring Buffer
00021     inline bool ArrayToRing(DSG::RingBuffer& ring,
      DSG::DSGSample* array, unsigned long length){
00022         int i=0;
00023         ring.Flush();
00024         while (!ring.Full()) {
00025             ring.Write(array[i]);
00026             ++i;
00027         }return true;
00028     }
00029 }
00030 #endif /* defined(__DSG__BufferConversion__) */
```

8.43 /Users/alexanderzywicki/Documents/DSG/src/DCBlocker.cpp File Reference

```
#include "DCBlocker.h"
```

8.44 DCBlocker.cpp

```
00001 //
00002 //  DCBlocker.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/13/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "DCBlocker.h"
00009 DSG::Filter::DCBlocker::DCBlocker():DSG::Filter::
      FilterBase(),_a(0.995),xm1(0),ym1(0),x(0),_temp(0){}
00010 DSG::Filter::DCBlocker::~DCBlocker(){}
```

8.45 /Users/alexanderzywicki/Documents/DSG/src/DCBlocker.h File Reference

```
#include "Filter.h"
```

Classes

- class DSG::Filter::DCBlocker

  DSG::Filter::DCBlocker - DC blocking filter.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.

- DSG::Filter

  DSG::Filter - Filters.

8.46 DCBlocker.h

```
00001 //
00002 //  DCBlocker.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/13/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__DCBlocker__
00009 #define __DSG__DCBlocker__
00010 #include "Filter.h"
00011 namespace DSG {
00012 #ifdef DSG_Short_Names
00013     inline
00014 #endif
00015     namespace Filter{
00016         //!\brief DSG::Filter::DCBlocker - DC blocking filter
00017         class DCBlocker:public DSG::Filter::FilterBase {
00018         public:
00019             DCBlocker();
00020             virtual ~DCBlocker();
00021             virtual inline bool Perform(DSG::DSGSample& signal);
00022             virtual inline bool Perform(DSG::RingBuffer& signal);
00023         protected:
00024             unsigned long count;
00025             DSG::DSGSample _temp;
00026             DSG::DSGSample xm1;
00027             DSG::DSGSample ym1;
00028             DSG::DSGSample x;
00029             DSG::DSGSample _a;
00030         };
00031         inline bool DSG::Filter::DCBlocker::Perform(
      DSG::DSGSample& signal){
00032             x = signal;
00033             signal= x - xm1+ (_a * ym1);
00034             xm1 = x;
00035             ym1=signal;
00036             return true;
00037         }
00038         inline bool DSG::Filter::DCBlocker::Perform(
      DSG::RingBuffer& signal){
00039             if (!signal.Empty()) {
00040                 count = signal.Count();
00041                 while (count-- > 0) {
00042                     if(signal.Read(_temp)){
00043                         if (Perform(_temp)) {
00044                             signal.Write(_temp);
00045                         }else return false;
00046                     }else return false;
00047                 }return true;
00048             }else return false;
00049         }
00050     }
00051 }
00052 #endif /* defined(__DSG__DCBlocker__) */
```

8.47 /Users/alexanderzywicki/Documents/DSG/src/Delay.h File Reference

```
#include "DSGTypes.h"
#include "SignalProcess.h"
#include "Interpolate.h"
#include "AudioSettings.h"
```

Classes

- class DSG::Delay< maxLength >

  DSG::Delay - General purpose delay line.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.

8.48 Delay.h

```
00001 //
00002 //  Delay.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/23/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__Delay__
00009 #define __DSG__Delay__
00010 #include "DSGTypes.h"
00011 #include "SignalProcess.h"
00012 #include "Interpolate.h"
00013 #include "AudioSettings.h"
00014 namespace DSG{
00015     //!\brief DSG::Delay - General purpose delay line
00016     template<unsigned long maxLength>
00017     class Delay:public DSG::SignalProcess{
00018     public:
00019         Delay():DSG::SignalProcess(),_max(maxLength),
      _swap(0),_temp(0),count(0),_index(0),_delay(0){
00020             for (int i=0; i<_max; ++i) {
00021                 _buffer[i]=0;
00022             }
00023         }
00024         Delay(double const& samples):DSG::SignalProcess(),
      _max(maxLength),_swap(0),_temp(0),count(0),_index(0),
      _delay(0){
00025             for (int i=0; i<_max; ++i) {
00026                 _buffer[i]=0;
00027             }
00028             if (samples>maxLength) {
00029                 _delay = maxLength;
00030             }else{
00031                 _delay = samples;
00032             }
00033         }
00034         virtual ~Delay(){}
00035         virtual inline unsigned long const& Length()const{
00036             return _delay;
00037         }
00038         virtual inline unsigned long const& Length(unsigned long const& samples){
00039             if (samples>maxLength) {
00040                 _delay = maxLength;
00041             }else{
00042                 _delay = samples;
00043             }
00044             return _delay;
00045         }
00046         virtual inline bool Perform(DSG::DSGSample& signal);
00047         virtual inline bool Perform(DSG::RingBuffer& signal);
00048     protected:
00049         unsigned long count;
00050         unsigned long  _delay;
00051         unsigned long _index;
00052         const unsigned long _max;
00053         DSG::DSGSample _buffer[maxLength];
00054         DSG::DSGSample _swap;
00055         DSG::DSGSample _temp;
00056         virtual inline void increment(){
00057             ++_index;
00058             if (_index>_delay) {
00059                 _index-=_delay;
00060             }
00061         }
00062     };
00063     template<unsigned long maxLength>
00064     inline bool DSG::Delay<maxLength>::Perform(
      DSG::DSGSample& signal){
00065         _swap = _buffer[_index-1];
00066         _buffer[_index-1]=signal;
00067         signal = _swap;
00068         increment();
00069         return true;
00070     }
```

```
00071     template<unsigned long maxLength>
00072     inline bool DSG::Delay<maxLength>::Perform(
      DSG::RingBuffer& signal){
00073         if (!signal.Empty()) {
00074             count = signal.Count();
00075             while (count-- > 0) {
00076                 if(signal.Read(_temp)){
00077                     if (Perform(_temp)) {
00078                         signal.Write(_temp);
00079                     }else return false;
00080                 }else return false;
00081             }return true;
00082         }else return false;
00083     }
00084 }
00085 #endif /* defined(__DSG__Delay__) */
```

8.49 /Users/alexanderzywicki/Documents/DSG/src/Denormal.h File Reference

```
#include <limits>
#include "DSGMath.h"
```

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.

Functions

- template<typename T >
  bool DSG::IsDenormal (T const &value)

  DSG::IsDenormal - Returns True if number is Denormal.

8.50 Denormal.h

```
00001 //
00002 //  Denormal.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/23/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef DSG_Denormal_h
00009 #define DSG_Denormal_h
00010 #include <limits>
00011 #include "DSGMath.h"
00012 namespace DSG{
00013     //!\brief DSG::IsDenormal - Returns True if number is Denormal
00014     template<typename T>
00015     inline bool IsDenormal(T const& value){
00016         return DSG::Abs(value)<=std::numeric_limits<T>::epsilon();//return true if number is
      denormal
00017     }
00018 }
00019 #endif
```

8.51 /Users/alexanderzywicki/Documents/DSG/src/DPW.h File Reference

```
#include "DSGTypes.h"
#include "DSGMath.h"
#include "SignalGenerator.h"
#include "Bounds.h"
```

Classes

- class DSG::DPW::DPW_Differentiator< order >

  DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the DPW Algorithm.
- class DSG::DPW::DPW_Differentiator< 1 >

  DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 1st order DPW Algorithm.
- class DSG::DPW::DPW_Differentiator< 2 >

  DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 2nd order DPW Algorithm.
- class DSG::DPW::DPW_Differentiator< 3 >

  DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 3rd order DPW Algorithm.
- class DSG::DPW::DPW_Differentiator< 4 >

  DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 4th order DPW Algorithm.
- class DSG::DPW::DPW_Differentiator< 5 >

  DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 5th order DPW Algorithm.
- class DSG::DPW::DPW_Differentiator< 6 >

  DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 6th order DPW Algorithm.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::DPW

  DSG::DPW - Generators using the DPW method.

Functions

- template<unsigned order>
  DSG::DSGSample DSG::DPW::DPW_Polynomial (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DSG::DPW::DPW_Polynomial< 1 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 1st Order Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DSG::DPW::DPW_Polynomial< 2 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 2nd order Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DSG::DPW::DPW_Polynomial< 3 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 3rd order Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DSG::DPW::DPW_Polynomial< 4 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 4th order Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DSG::DPW::DPW_Polynomial< 5 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 5th order Polynoimal used in DPW Algorithm.
- template<>
  DSG::DSGSample DSG::DPW::DPW_Polynomial< 6 > (DSG::DSGSample const &value)

  DSG::DPW::DPW_Polynomial - 6th order Polynoimal used in DPW Algorithm.

8.52 DPW.h

```
00001 //
00002 //  DPW.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 11/11/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef DSG_DPW_h
00009 #define DSG_DPW_h
00010 #include "DSGTypes.h"
00011 #include "DSGMath.h"
00012 #include "SignalGenerator.h"
00013 #include "Bounds.h"
00014 namespace DSG{
00015 #ifdef DSG_Short_Names
00016     inline
00017 #endif
00018     //!\brief DSG::DPW - Generators using the DPW method
00019     namespace DPW{
00020         //!\brief DSG::DPW::DPW_Polynomial - Polynoimal used in DPW Algorithm
00021         template<unsigned order>
00022         inline DSG::DSGSample DPW_Polynomial(
    DSG::DSGSample const& value){
00023             DSG::StaticAssertBounds<1,6,order>();//must be 1-6 order
00024             return value;
00025         }
00026         //!\brief DSG::DPW::DPW_Polynomial - 1st Order Polynoimal used in DPW Algorithm
00027         template<>
00028         inline DSG::DSGSample DPW_Polynomial<1>(
    DSG::DSGSample const& value){
00029             return value;
00030         }
00031         //!\brief DSG::DPW::DPW_Polynomial - 2nd order Polynoimal used in DPW Algorithm
00032         template<>
00033         inline DSG::DSGSample DPW_Polynomial<2>(
    DSG::DSGSample const& value){
00034             return DSG::Pow<2>(value);
00035         }
00036         //!\brief DSG::DPW::DPW_Polynomial - 3rd order Polynoimal used in DPW Algorithm
00037         template<>
00038         inline DSG::DSGSample DPW_Polynomial<3>(
    DSG::DSGSample const& value){
00039             return DSG::Pow<3>(value)-value;
00040         }
00041         //!\brief DSG::DPW::DPW_Polynomial - 4th order Polynoimal used in DPW Algorithm
00042         template<>
00043         inline DSG::DSGSample DPW_Polynomial<4>(
    DSG::DSGSample const& value){
00044             return DSG::Pow<2>(value) * (DSG::Pow<2>(value) - 2.0);
00045         }
00046         //!\brief DSG::DPW::DPW_Polynomial - 5th order Polynoimal used in DPW Algorithm
00047         template<>
00048         inline DSG::DSGSample DPW_Polynomial<5>(
    DSG::DSGSample const& value){
00049             return DSG::Pow<5>(value) - DSG::Pow<3>(value) * 10.0/3.0 + value * 7.0/3.0;
00050         }
00051         //!\brief DSG::DPW::DPW_Polynomial - 6th order Polynoimal used in DPW Algorithm
00052         template<>
00053         inline DSG::DSGSample DPW_Polynomial<6>(
    DSG::DSGSample const& value){
00054             return DSG::Pow<6>(value) - 5.0 * DSG::Pow<4>(value) + 7.0 *
    DPW_Polynomial<2>(value);
00055         }
00056 #warning DSG::DPW - differentiators order 3-6 need verification. they cause major clipping
00057         //!\todo Fix DSG::DPW::DPW_Differentiator algorithms for orders 3-6
00058         //differentiators
00059         //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the DPW Algorithm
00060         template<unsigned order>
00061         class DPW_Differentiator{
00062         public:
00063             DPW_Differentiator(){
00064                 DSG::StaticAssertBounds<1, 6,order>();//order must be 1-6
00065             }
00066         };
00067         //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 1st order DPW
     Algorithm
00068         template<>
00069         class DPW_Differentiator<1>{
00070         public:
00071             inline DSG::DSGSample operator()(
    DSG::DSGSample const& signal,DSG::DSGSample const& dt){
00072                 return signal;
00073             }
00074         };
```

```
00075         //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 2nd order DPW
       Algorithm
00076         template<>
00077         class DPW_Differentiator<2>{
00078         public:
00079             inline DSG::DSGSample operator()(
       DSG::DSGSample const& signal,DSG::DSGSample const& dt){
00080                 output = (signal - _delay)/(4.0 * dt);
00081                 _delay = signal;
00082                 return output;
00083             }
00084         protected:
00085             DSG::DSGSample output;
00086             DSG::DSGSample _delay;
00087         };
00088         //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 3rd order DPW
       Algorithm
00089         template<>
00090         class DPW_Differentiator<3>{
00091         public:
00092             inline DSG::DSGSample operator()(
       DSG::DSGSample const& signal,DSG::DSGSample const& dt){
00093                 output  = (signal - _delay[0]);
00094                 output -= (_delay[0] - _delay[1]);
00095                 output /= (24.*DSG::Pow<2>(dt));
00096                 _delay[1]=_delay[0];
00097                 _delay[0]=signal;
00098                 return output;
00099             }
00100         protected:
00101             DSG::DSGSample output;
00102             DSG::DSGSample _delay[2];
00103         };
00104         //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 4th order DPW
       Algorithm
00105         template<>
00106         class DPW_Differentiator<4>{
00107         public:
00108             inline DSG::DSGSample operator()(
       DSG::DSGSample const& signal,DSG::DSGSample const& dt){
00109                 output  = (signal - _delay[0]);
00110                 output -= (_delay[0] - _delay[1]);
00111                 output -= (_delay[1] - _delay[2]);
00112                 output /= 144*DSG::Pow<3>(dt);
00113                 _delay[2]=_delay[1];
00114                 _delay[1]=_delay[0];
00115                 _delay[0]=signal;
00116                 return output;
00117             }
00118         protected:
00119             DSG::DSGSample output;
00120             DSG::DSGSample _delay[3];
00121         };
00122         //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 5th order DPW
       Algorithm
00123         template<>
00124         class DPW_Differentiator<5>{
00125         public:
00126             inline DSG::DSGSample operator()(
       DSG::DSGSample const& signal,DSG::DSGSample const& dt){
00127                 output  = (signal - _delay[0]);
00128                 output -= (_delay[0] - _delay[1]);
00129                 output -= (_delay[1] - _delay[2]);
00130                 output -= (_delay[2] - _delay[3]);
00131                 output /= 960*DSG::Pow<4>(dt);
00132                 _delay[3]=_delay[2];
00133                 _delay[2]=_delay[1];
00134                 _delay[1]=_delay[0];
00135                 _delay[0]=signal;
00136                 return output;
00137             }
00138         protected:
00139             DSG::DSGSample output;
00140             DSG::DSGSample _delay[4];
00141         };
00142         //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 6th order DPW
       Algorithm
00143         template<>
00144         class DPW_Differentiator<6>{
00145         public:
00146             inline DSG::DSGSample operator()(
       DSG::DSGSample const& signal,DSG::DSGSample const& dt){
00147                 output  = (signal - _delay[0]);
00148                 output -= (_delay[0] - _delay[1]);
00149                 output -= (_delay[1] - _delay[2]);
00150                 output -= (_delay[2] - _delay[3]);
00151                 output -= (_delay[3] - _delay[4]);
```

```
00152                 output /= 7200*DSG::Pow<5>(dt);
00153                 _delay[4]=_delay[3];
00154                 _delay[3]=_delay[2];
00155                 _delay[2]=_delay[1];
00156                 _delay[1]=_delay[0];
00157                 _delay[0]=signal;
00158                 return output;
00159             }
00160         protected:
00161             DSG::DSGSample output;
00162             DSG::DSGSample _delay[5];
00163         };
00164     }
00165 }
00166 #endif
```

8.53 /Users/alexanderzywicki/Documents/DSG/src/DPWSaw.h File Reference

```
#include "DPW.h"
```

Classes

- class DSG::DPW::DPWSaw< order >

  DSG::DPW::DPWSaw - Sawtooth Generator using the Nth Order DPW algorithm.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::DPW

  DSG::DPW - Generators using the DPW method.

8.54 DPWSaw.h

```
00001 //
00002 //  DPWSaw.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/27/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__DPWSaw__
00009 #define __DSG__DPWSaw__
00010 #include "DPW.h"
00011 namespace DSG {
00012 #ifdef DSG_Short_Names
00013     inline
00014 #endif
00015     namespace DPW{
00016         //!\brief DSG::DPW::DPWSaw - Sawtooth Generator using the Nth Order DPW algorithm
00017         template<unsigned order>
00018         class DPWSaw:public DSG::SignalGenerator{
00019         public:
00020             DPWSaw():DSG::SignalGenerator(),_register(0){
00021                 DSG::StaticAssertBounds<1, 6,order>();
00022             }
00023             DPWSaw(DSG::DSGFrequency const& frequency,
      DSG::DSGPhase const& offset):DSG::SignalGenerator(frequency,offset),
      _register(0){DSG::StaticAssertBounds<1, 6,order>();}
00024             virtual ~DPWSaw(){}
00025             virtual inline bool Perform(DSG::DSGSample& signal){
00026                 //trivial saw ramping from -1 to 1
00027                 _register = _phasor;
00028                 _register-=0.5;
00029                 _register*=2.0;
00030                 /*-------------------------*/
00031                 //DPW algorithm
```

```
00032                 //polynomial shaping
00033                 _register=DSG::DPW::DPW_Polynomial<order>(_register);
00034                 //differentiating
00035                 signal = _diff(_register,_dt);
00036                 /*------------------------*/
00037                 //signal = DSG::EnforceBounds<-1, 1>(signal);
00038                 //advance phase
00039                 step();
00040                 return true;
00041             }
00042             virtual inline bool Perform(DSG::RingBuffer& signal){
00043                 signal.Flush();
00044                 while (!signal.Full()) {
00045                     if (Perform(_storage)) {
00046                         if(signal.Write(_storage)){
00047                         }else return false;
00048                     }else return false;
00049                 }return true;
00050             }
00051         protected:
00052             DSG::DSGSample _register;
00053             DSG::DPW::DPW_Differentiator<order>
      _diff;
00054         };
00055     }
00056 }
00057 #endif /* defined(__DSG__DPWSaw__) */
```

8.55 /Users/alexanderzywicki/Documents/DSG/src/Driver.cpp File Reference

```
#include "Driver.h"
```

Macros

- #define BufferSize 512

Functions

- int DriverInit (void *data)
- int DriverExit ()
- int Callback (const void *input, void *output, unsigned long frameCount, const PaStreamCallbackTimeInfo *timeInfo, PaStreamCallbackFlags statusFlags, void *userData)

Variables

- PaStream * stream
- DSG::RingBuffer _buffer (BufferSize)

8.55.1 Macro Definition Documentation

8.55.1.1 #define BufferSize 512

Definition at line 10 of file Driver.cpp.

8.55.2 Function Documentation

8.55.2.1 int Callback (const void * *input*, void * *output*, unsigned long *frameCount*, const PaStreamCallbackTimeInfo * *timeInfo*, PaStreamCallbackFlags *statusFlags*, void * *userData*)

Definition at line 61 of file Driver.cpp.

```
00066                               {
00067     DSG::DSGSample* _out = (DSG::DSGSample*)output;
00068     DSG:: DSGSample _sample;
00069     DSG::SignalGenerator* _osc = (DSG::SignalGenerator*)userData;
00070     if (_out!=nullptr) {
00071         _buffer.Flush();
00072         _osc->Perform(_buffer);
00073         for (int i=0; i<frameCount; ++i) {
00074             _buffer.Read(_sample);
00075             *_out++ = _sample;
00076             *_out++ = _sample;
00077         }
00078     }
00079     return 0;
00080 }
```

8.55.2.2 int DriverExit ( )

Definition at line 38 of file Driver.cpp.

```
00038                 {
00039     PaError err=0;
00040     err = Pa_StopStream(stream);
00041     if (err!=paNoError) {
00042 #ifdef DEBUG
00043         printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00044 #endif
00045         return 1;
00046     }
00047     err = Pa_CloseStream( stream );
00048     if( err != paNoError ){
00049 #ifdef DEBUG
00050         printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00051 #endif
00052     }
00053     err = Pa_Terminate();
00054     if( err != paNoError ){
00055 #ifdef DEBUG
00056         printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00057 #endif
00058     }
00059     return 0;
00060 }
```

8.55.2.3 int DriverInit (void * *data*)

Definition at line 12 of file Driver.cpp.

```
00012                             {
00013     PaError err=0;
00014
00015     err=Pa_Initialize();
00016     if (err!=paNoError) {
00017 #ifdef DEBUG
00018         printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00019 #endif
00020         return 1;
00021     }
00022     err = Pa_OpenDefaultStream(&stream, 0, 2, paFloat32,DSG::SampleRate(),
      BufferSize, Callback, data);
00023     if (err!=paNoError) {
00024 #ifdef DEBUG
00025         printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00026 #endif
00027         return 1;
00028     }
00029     err = Pa_StartStream(stream);
00030     if (err!=paNoError) {
00031 #ifdef DEBUG
00032         printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00033 #endif
00034         return 1;
00035     }
00036     return 0;
00037 }
```

8.55.3 Variable Documentation

8.55.3.1 DSG:: RingBuffer _buffer(BufferSize)

8.55.3.2 PaStream* stream

Definition at line 9 of file Driver.cpp.

8.56 Driver.cpp

```
00001 //
00002 //  Driver.cpp
00003 //  Waveform
00004 //
00005 //  Created by Alexander Zywicki on 8/25/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "Driver.h"
00009 PaStream* stream;
00010 #define BufferSize 512
00011 DSG:: RingBuffer _buffer(BufferSize);
00012 int DriverInit(void * data){
00013     PaError err=0;
00014
00015     err=Pa_Initialize();
00016     if (err!=paNoError) {
00017 #ifdef DEBUG
00018         printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00019 #endif
00020         return 1;
00021     }
00022     err = Pa_OpenDefaultStream(&stream, 0, 2, paFloat32,DSG::SampleRate(),
      BufferSize, Callback, data);
00023     if (err!=paNoError) {
00024 #ifdef DEBUG
00025         printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00026 #endif
00027         return 1;
00028     }
00029     err = Pa_StartStream(stream);
00030     if (err!=paNoError) {
00031 #ifdef DEBUG
00032         printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00033 #endif
00034         return 1;
00035     }
00036     return 0;
00037 }
00038 int DriverExit(){
00039     PaError err=0;
00040     err = Pa_StopStream(stream);
00041     if (err!=paNoError) {
00042 #ifdef DEBUG
00043         printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00044 #endif
00045         return 1;
00046     }
00047     err = Pa_CloseStream( stream );
00048     if( err != paNoError ){
00049 #ifdef DEBUG
00050         printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00051 #endif
00052     }
00053     err = Pa_Terminate();
00054     if( err != paNoError ){
00055 #ifdef DEBUG
00056         printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00057 #endif
00058     }
00059     return 0;
00060 }
00061 int Callback(const void *input,
00062              void *output,
00063              unsigned long frameCount,
00064              const PaStreamCallbackTimeInfo* timeInfo,
00065              PaStreamCallbackFlags statusFlags,
00066              void *userData) {
00067     DSG::DSGSample* _out = (DSG::DSGSample*)output;
00068     DSG:: DSGSample _sample;
00069     DSG::SignalGenerator* _osc = (DSG::SignalGenerator*)userData;
```

```
00070     if (_out!=nullptr) {
00071         _buffer.Flush();
00072         _osc->Perform(_buffer);
00073         for (int i=0; i<frameCount; ++i) {
00074             _buffer.Read(_sample);
00075             *_out++ = _sample;
00076             *_out++ = _sample;
00077         }
00078     }
00079     return 0;
00080 }
```

8.57 /Users/alexanderzywicki/Documents/DSG/src/Driver.h File Reference

```
#include <portaudio.h>
#include "DSG.h"
```

Functions

- int DriverInit (void *data)
- int DriverExit ()
- int Callback (const void *input, void *output, unsigned long frameCount, const PaStreamCallbackTimeInfo *timeInfo, PaStreamCallbackFlags statusFlags, void *userData)

8.57.1 Function Documentation

8.57.1.1 int Callback (const void * *input*, void * *output*, unsigned long *frameCount*, const PaStreamCallbackTimeInfo * *timeInfo*, PaStreamCallbackFlags *statusFlags*, void * *userData*)

Definition at line 61 of file Driver.cpp.

```
00066                                {
00067     DSG::DSGSample* _out = (DSG::DSGSample*)output;
00068     DSG:: DSGSample _sample;
00069     DSG::SignalGenerator* _osc = (DSG::SignalGenerator*)userData;
00070     if (_out!=nullptr) {
00071         _buffer.Flush();
00072         _osc->Perform(_buffer);
00073         for (int i=0; i<frameCount; ++i) {
00074             _buffer.Read(_sample);
00075             *_out++ = _sample;
00076             *_out++ = _sample;
00077         }
00078     }
00079     return 0;
00080 }
```

8.57.1.2 int DriverExit ( )

Definition at line 38 of file Driver.cpp.

```
00038                {
00039     PaError err=0;
00040     err = Pa_StopStream(stream);
00041     if (err!=paNoError) {
00042 #ifdef DEBUG
00043         printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00044 #endif
00045         return 1;
00046     }
00047     err = Pa_CloseStream( stream );
00048     if( err != paNoError ){
00049 #ifdef DEBUG
00050         printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00051 #endif
```

```
00052     }
00053     err = Pa_Terminate();
00054     if( err != paNoError ){
00055 #ifdef DEBUG
00056         printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00057 #endif
00058     }
00059     return 0;
00060 }
```

8.57.1.3 int DriverInit (void * *data*)

Definition at line 12 of file Driver.cpp.

```
00012                           {
00013     PaError err=0;
00014
00015     err=Pa_Initialize();
00016     if (err!=paNoError) {
00017 #ifdef DEBUG
00018         printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00019 #endif
00020         return 1;
00021     }
00022     err = Pa_OpenDefaultStream(&stream, 0, 2, paFloat32,DSG::SampleRate(),
      BufferSize, Callback, data);
00023     if (err!=paNoError) {
00024 #ifdef DEBUG
00025         printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00026 #endif
00027         return 1;
00028     }
00029     err = Pa_StartStream(stream);
00030     if (err!=paNoError) {
00031 #ifdef DEBUG
00032         printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
00033 #endif
00034         return 1;
00035     }
00036     return 0;
00037 }
```

8.58 Driver.h

```
00001 //
00002 //  Driver.h
00003 //  Waveform
00004 //
00005 //  Created by Alexander Zywicki on 8/25/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __Waveform__Driver__
00009 #define __Waveform__Driver__
00010 #ifdef DEBUG
00011 #include <iostream>
00012 #endif
00013 #include <portaudio.h>
00014 #include "DSG.h"
00015 int DriverInit(void * data);
00016 int DriverExit();
00017 int Callback( const void *input,
00018               void *output,
00019               unsigned long frameCount,
00020               const PaStreamCallbackTimeInfo* timeInfo,
00021               PaStreamCallbackFlags statusFlags,
00022               void *userData );
00023 #endif /* defined(__Waveform__Driver__) */
```

8.59 /Users/alexanderzywicki/Documents/DSG/src/DSF.h File Reference

```
#include "DSGMath.h"
#include "DSGTypes.h"
```

Namespaces

- DSG

DSG - A Collection of tools for Digital Signal Generation.

Functions

- template<typename decimal = DSG::DSGSample>
  decimal DSG::DSF (decimal const &beta, decimal const &theta, decimal const &N, decimal const &a)

8.60 DSF.h

```
00001 //
00002 //  DSF.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 11/5/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__DSF__
00009 #define __DSG__DSF__
00010 #include "DSGMath.h"
00011 #include "DSGTypes.h"
00012 namespace DSG{
00013     template<typename decimal=DSG::DSGSample>
00014     decimal DSF(decimal const& beta,decimal const& theta,decimal const& N,decimal const& a){
00015 #warning Untested DSG::DSF()
00016         decimal denominator = 1 + DSG::Pow<2>(a) - (2.0*a*cos(beta));
00017         decimal numerator = sin(theta) - a * sin(theta-beta) - pow(a, N+1) * (sin(theta + (N+1)*beta) - a*
      sin(theta + (N*beta)));
00018         return numerator/denominator;
00019     }
00020 }
00021 #endif /* defined(__DSG__DSF__) */
```

8.61 /Users/alexanderzywicki/Documents/DSG/src/DSG.h File Reference

```
#include "AudioSettings.h"
#include "SignalProcess.h"
#include "Buffer.h"
#include "RingBuffer.h"
#include "SignalGenerator.h"
#include "Sine.h"
#include "Sinc.h"
#include "Denormal.h"
#include "Math.h"
#include "Blackman.h"
#include "LUT.h"
#include "Window.h"
#include "Bounds.h"
#include "GenericGenerator.h"
#include "Delay.h"
#include "Sleep.h"
#include "BufferConversion.h"
#include "FourierSeries.h"
#include "FourierSaw.h"
#include "FourierSquare.h"
#include "FourierTriangle.h"
#include "AnalogSaw.h"
#include "AnalogSquare.h"
#include "AnalogTriangle.h"
#include "BLIT.h"
#include "BLITSaw.h"
#include "DSF.h"
#include "DPW.h"
#include "DPWSaw.h"
#include "EPTRSaw.h"
#include "Noise.h"
#include "DCBlocker.h"
#include "Filter.h"
#include "Leaky.h"
```

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.

Macros

- #define DSG_Short_Names

8.61.1 Macro Definition Documentation

8.61.1.1 #define DSG_Short_Names

Definition at line 10 of file DSG.h.

8.62 DSG.h

```
00001 //
00002 //  DSG.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef DSG_DSG_h
00009 #define DSG_DSG_h
00010 #define DSG_Short_Names // enables inlining of nested namespaces to allow shorter explicit typenames
00011 //Example: DSG::Analog::AnalogSaw (Long Name)...DSG::AnalogSaw (Short Name)(only available with this macro
       enabled
00012 //!\brief DSG - A Collection of tools for Digital Signal Generation
00013 namespace DSG {}
00014 #include "AudioSettings.h"
00015 #include "SignalProcess.h"
00016 #include "Buffer.h"
00017 #include "RingBuffer.h"
00018 #include "SignalGenerator.h"
00019 #include "Sine.h"
00020 #include "Sinc.h"
00021 #include "Denormal.h"
00022 #include "Math.h"
00023 #include "Blackman.h"
00024 #include "LUT.h"
00025 #include "Window.h"
00026 #include "Bounds.h"
00027
00028 #include "GenericGenerator.h"
00029
00030 #include "Delay.h"
00031
00032
00033 #include "Sleep.h"
00034 #include "BufferConversion.h"
00035
00036 #include "FourierSeries.h"
00037 #include "FourierSaw.h"
00038 #include "FourierSquare.h"
00039 #include "FourierTriangle.h"
00040
00041 #include "AnalogSaw.h"
00042 #include "AnalogSquare.h"
00043 #include "AnalogTriangle.h"
00044
00045 #include "BLIT.h"
00046 #include "BLITSaw.h"
00047
00048 #include "DSF.h"
00049
00050 #include "DPW.h"
00051 #include "DPWSaw.h"
00052
00053 #include "EPTRSaw.h"
00054
00055 #include "Noise.h"
00056
00057 #include "DCBlocker.h"
00058
00059 #include "Filter.h"
00060 #include "Leaky.h"
00061
00062 #endif
```

8.63 /Users/alexanderzywicki/Documents/DSG/src/DSGMath.h File Reference

```
#include <math.h>
#include <type_traits>
```

Classes

- struct DSG::Factorial< N >

  DSG::Factorial - Compute integer factorial.

- struct DSG::Factorial< 0 >

  DSG::Factorial - Compute integer factorial.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.

Functions

- template<typename T >
  T DSG::Abs (T const &value)

  DSG::Abs - Calculate absolute value.

- template<unsigned exponent, class T >
  T constexpr DSG::Pow (T const base)

  DSG::Pow - Any type to an integer power, i.e. N^ I.

8.64 DSGMath.h

```
00001 //
00002 //  Math.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/23/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef DSG_Math_h
00009 #define DSG_Math_h
00010 #include <math.h>
00011 #include <type_traits>
00012 namespace DSG {
00013     //!\brief DSG::Abs - Calculate absolute value
00014     template<typename T>
00015     inline T Abs(T const& value){
00016         return value < 0.0 ? -1.0 * value : value;
00017     }
00018     //!\brief DSG::Factorial - Compute integer factorial
00019     template<unsigned long N>
00020     struct Factorial{
00021         enum {value = N * Factorial<N-1>::value};
00022     };
00023     //!\brief DSG::Factorial - Compute integer factorial
00024     template<>
00025     struct Factorial<0>{
00026         enum{ value = 1 };
00027     };
00028     namespace{
00029         template<class T, unsigned N>
00030         struct power{
00031             static constexpr T value(const T x){
00032                 return power<T, N-1>::value(x) * x;
00033             }
00034         };
00035         template<class T>
00036         struct power<T, 0>{
00037             static constexpr T value(const T x){
00038                 return 1;
00039             }
00040         };
00041     }
00042     //!\brief DSG::Pow - Any type to an integer power, i.e. N ^ I
00043     template<unsigned exponent, class T>
00044     T constexpr Pow(T const base){
00045         return power<T, exponent>::value(base);
00046     }
00047 }
00048 #endif
```

8.65 /Users/alexanderzywicki/Documents/DSG/src/DSGTypes.h File Reference

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.

Typedefs

- typedef float DSG::DSGFrequency

  DSG::DSGFrequency - Type for representing a frequency value.
- typedef float DSG::DSGPhase

  DSG::DSGFrequency - Type for representing a phase value.
- typedef float DSG::DSGSample

  DSG::DSGFrequency - Type for representing an audio sample.

8.66 DSGTypes.h

```
00001 //
00002 //  DSGTypes.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef DSG_DSGTypes_h
00009 #define DSG_DSGTypes_h
00010 namespace DSG {
00011     //!\brief DSG::DSGFrequency - Type for representing a frequency value
00012     typedef float DSGFrequency;
00013     //!\brief DSG::DSGFrequency - Type for representing a phase value
00014     typedef float DSGPhase;
00015     //!\brief DSG::DSGFrequency - Type for representing an audio sample
00016     typedef float DSGSample;
00017 }
00018 #endif
```

8.67 /Users/alexanderzywicki/Documents/DSG/src/EPTRSaw.cpp File Reference

```
#include "EPTRSaw.h"
```

8.68 EPTRSaw.cpp

```
00001 //
00002 //  EPTRSaw.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/29/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "EPTRSaw.h"
00009 DSG::EPTR::EPTRSaw::EPTRSaw():DSG::SignalGenerator(){}
00010 DSG::EPTR::EPTRSaw::EPTRSaw(DSG::DSGFrequency const& frequency,
      DSG::DSGPhase const& offset):DSG::SignalGenerator(frequency,offset){}
00011 DSG::EPTR::EPTRSaw::~EPTRSaw(){}
```

8.69 /Users/alexanderzywicki/Documents/DSG/src/EPTRSaw.h File Reference

```
#include "SignalGenerator.h"
```

Classes

- class DSG::EPTR::EPTRSaw

  DSG::EPTR::EPTRSaw-Sawtooth Wave Generator Using The Efficienct Polynomial Transfer Region Algorithm.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::EPTR

  DSG::EPTR - Generators Based On The Efficienct Polynomial Transfer Region Algorithm.

8.70 EPTRSaw.h

```
00001 //
00002 //  EPTRSaw.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/29/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__EPTRSaw__
00009 #define __DSG__EPTRSaw__
00010 #include "SignalGenerator.h"
00011 namespace DSG {
00012 #ifdef DSG_Short_Names
00013     inline
00014 #endif
00015     //!DSG::EPTR - Generators Based On The Efficienct Polynomial Transfer Region Algorithm
00016     namespace EPTR{
00017         //!\brief DSG::EPTR::EPTRSaw-Sawtooth Wave Generator Using The Efficienct Polynomial Transfer
     Region Algorithm
00018         //!\todo Test and Possibly Re-Write DSG::EPTR::EPTRSaw algorithm
00019         class EPTRSaw : public DSG::SignalGenerator{
00020         public:
00021             EPTRSaw();
00022             EPTRSaw(DSG::DSGFrequency const& frequency,
      DSG::DSGPhase const& offset);
00023             virtual ~EPTRSaw();
00024             virtual inline bool Perform(DSG::DSGSample& signal);
00025             virtual inline bool Perform(DSG::RingBuffer& signal);
00026         protected:
00027             DSG::DSGSample _register;
00028         };
00029         inline bool DSG::EPTR::EPTRSaw::Perform(
      DSG::DSGSample& signal){
00030 #warning Untested For Aliasing DSG::EPTR::EPTRSaw::Perform()
00031             //generate trivial saw
00032             _register = _phasor;
00033             _register+=0.5;
00034             if (_register>1.0) {
00035                 --_register;
00036             }
00037             _register-=0.5;
00038             _register*=2.0;
00039             if (_register > 1.0-_dt) {
00040                 //transition region detected
00041                 //apply eptr correction
00042                 signal = _register - (_register/_dt) + (1.0/_dt) -1;
00043             }else{
00044                 signal = _register;
00045             }
00046             step();//avance phase
00047             return true;
00048         }
00049         inline bool DSG::EPTR::EPTRSaw::Perform(
```

```
    DSG::RingBuffer& signal){
00050             signal.Flush();
00051             while (!signal.Full()) {
00052                 if (Perform(_storage)) {
00053                     if(signal.Write(_storage)){
00054                     }else return false;
00055                 }else return false;
00056             }return true;
00057         }
00058     }
00059 }
00060 #endif /* defined(__DSG__EPTRSaw__) */
```

8.71 /Users/alexanderzywicki/Documents/DSG/src/Filter.cpp File Reference

```
#include "Filter.h"
```

8.72 Filter.cpp

```
00001 //
00002 //  Filter.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/27/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "Filter.h"
00009 DSG::Filter::FilterBase::FilterBase():_temp(0),count(0){}
00010 DSG::Filter::FilterBase::~FilterBase(){}
```

8.73 /Users/alexanderzywicki/Documents/DSG/src/Filter.h File Reference

```
#include "SignalProcess.h"
```

Classes

- class DSG::Filter::FilterBase

  DSG::Filter::FilterBase - Filter Base Class, implements interface for cutoff frequency.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::Filter

  DSG::Filter - Filters.

8.74 Filter.h

```
00001 //
00002 //  Filter.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/27/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__Filter__
00009 #define __DSG__Filter__
```

```
00010 #include "SignalProcess.h"
00011 namespace DSG{
00012 #ifdef DSG_Short_Names
00013     inline
00014 #endif
00015     //!\brief DSG::Filter - Filters
00016     namespace Filter{
00017         //!\brief DSG::Filter::FilterBase - Filter Base Class, implements interface for cutoff frequency
00018         class FilterBase:public DSG::SignalProcess{
00019         public:
00020             FilterBase();
00021             virtual ~FilterBase();
00022             virtual inline bool Perform(DSG::DSGSample& signal);
00023             virtual inline bool Perform(DSG::RingBuffer& signal);
00024             virtual inline bool Cutoff(DSG::DSGFrequency const& cutoff);
00025         protected:
00026             DSG::DSGSample _temp;
00027             unsigned long count;
00028         };
00029         inline bool DSG::Filter::FilterBase::Perform(
      DSG::DSGSample& signal){
00030             return true;
00031         }
00032         inline bool DSG::Filter::FilterBase::Perform(
      DSG::RingBuffer& signal){
00033             if (!signal.Empty()) {
00034                 count = signal.Count();
00035                 while (count-- > 0) {
00036                     if(signal.Read(_temp)){
00037                         if (Perform(_temp)) {
00038                             signal.Write(_temp);
00039                         }else return false;
00040                     }else return false;
00041                 }return true;
00042             }else return false;
00043         }
00044         inline bool DSG::Filter::FilterBase::Cutoff(
      DSG::DSGFrequency const& cutoff){
00045             return false;
00046         }
00047     }
00048 }
00049 #endif /* defined(__DSG__Filter__) */
```

8.75 /Users/alexanderzywicki/Documents/DSG/src/FourierSaw.cpp File Reference

```
#include "FourierSaw.h"
```

8.76 FourierSaw.cpp

```
00001 //
00002 //  FourierSaw.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "FourierSaw.h"
00009 DSG::Fourier::FourierSaw::FourierSaw():DSG::
      SignalGenerator(),_a(1.7/PI),phs(0),value(0),i(0){}
00010 DSG::Fourier::FourierSaw::FourierSaw(
      DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):
      DSG::SignalGenerator(frequency,offset),_a(1.7/PI),phs(0),value(0),i(0){
00011     _h = MaxHarms(_frequency)+1;
00012 }
00013 DSG::Fourier::FourierSaw::~FourierSaw(){}
```

8.77 /Users/alexanderzywicki/Documents/DSG/src/FourierSaw.h File Reference

```
#include "SignalGenerator.h"
```

Classes

- class DSG::Fourier::FourierSaw

  DSG::Fourier::FourierSaw - Fourier Series Sawtooth Wave Generator.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::Fourier

  DSG::Fourier - Namespace Containing Fourier Series Based Oscillators.

8.78 FourierSaw.h

```
00001 //
00002 //  FourierSaw.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__FourierSaw__
00009 #define __DSG__FourierSaw__
00010 #include "SignalGenerator.h"
00011 namespace DSG{
00012 #ifdef DSG_Short_Names
00013     inline
00014 #endif
00015     //!DSG::Fourier - Namespace Containing Fourier Series Based Oscillators
00016     namespace Fourier{
00017         //!\brief DSG::Fourier::FourierSaw - Fourier Series Sawtooth Wave Generator
00018         class FourierSaw : public DSG::SignalGenerator {
00019         public:
00020             FourierSaw();
00021             FourierSaw(DSG::DSGFrequency const& frequency,
      DSG::DSGPhase const& offset);
00022             virtual ~FourierSaw();
00023             virtual inline bool Perform(DSG::DSGSample& signal);
00024             virtual inline bool Perform(DSG::RingBuffer& signal);
00025             virtual inline DSG::DSGFrequency const& Frequency(
      DSG::DSGFrequency const& value);
00026         protected:
00027             unsigned long _h;
00028             const double _a;
00029             double phs;
00030             double value;
00031             int i;
00032         };
00033         inline bool DSG::Fourier::FourierSaw::Perform(
      DSG::DSGSample& signal){
00034             //_h Sine Calls Per Sample where _h  is theoretically nyquist / frequency
00035             value=DSG::Sin(_phasor);
00036             for (i=2; i<_h; ++i) {
00037                 value += (1.0/i) * DSG::Sin(_phasor*i);
00038             }
00039             value*=_a;
00040             signal = value;
00041             step();
00042             return true;
00043         }
00044         inline bool DSG::Fourier::FourierSaw::Perform(
      DSG::RingBuffer& signal){
00045             signal.Flush();
00046             while (!signal.Full()) {
00047                 if (Perform(_storage)) {
00048                     if(signal.Write(_storage)){
00049                     }else return false;
00050                 }else return false;
00051             }return true;
00052         }
00053         inline DSG::DSGFrequency const&
      DSG::Fourier::FourierSaw::Frequency(
      DSG::DSGFrequency const& value){
00054             _frequency = value;
00055             _dt = _frequency/DSG::SampleRate();
00056             _h = MaxHarms(_frequency);
```

```
00057             return _frequency;
00058         }
00059     }
00060 }
00061 #endif /* defined(__DSG__FourierSaw__) */
```

8.79 /Users/alexanderzywicki/Documents/DSG/src/FourierSeries.cpp File Reference

```
#include "FourierSeries.h"
```

8.80 FourierSeries.cpp

```
00001 //
00002 //  FourierSeries.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 11/18/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "FourierSeries.h"
00009 DSG::Fourier::Harmonic::Harmonic():_ratio(0),_amplitude(0){}
00010 DSG::Fourier::Harmonic::Harmonic(DSG::DSGSample const& ratio,
      DSG::DSGSample const& amplitude):_ratio(ratio),_amplitude(amplitude){}
00011 DSG::Fourier::Harmonic::~Harmonic(){
00012     _ratio=0;
00013     _amplitude=0;
00014 }
00015 DSG::DSGSample const& DSG::Fourier::Harmonic::Ratio()const{
00016     return _ratio;
00017 }
00018 DSG::DSGSample const& DSG::Fourier::Harmonic::Ratio(
      DSG::DSGSample const& value){
00019     _ratio = value;
00020     return _ratio;
00021 }
00022 DSG::DSGSample const& DSG::Fourier::Harmonic::Amplitude()
      const{
00023     return _amplitude;
00024 }
00025 DSG::DSGSample const& DSG::Fourier::Harmonic::Amplitude(
      DSG::DSGSample const& value){
00026     _amplitude=value;
00027     return _amplitude;
00028 }
00029 DSG::Fourier::FourierSeriesGenerator::FourierSeriesGenerator
      ():DSG::SignalGenerator(){}
00030 DSG::Fourier::FourierSeriesGenerator::FourierSeriesGenerator
      (DSG::DSGFrequency const& frequency, DSG::DSGPhase const& offset):
      DSG::SignalGenerator(frequency,offset){}
00031 DSG::Fourier::FourierSeriesGenerator::~FourierSeriesGenerator
      (){}
```

8.81 /Users/alexanderzywicki/Documents/DSG/src/FourierSeries.h File Reference

```
#include "SignalGenerator.h"
#include <vector>
```

Classes

- class DSG::Fourier::Harmonic

  DSG::Fourier::Harmonic - Represents a single harmonic in a Fourier Series.

- class DSG::Fourier::FourierSeriesGenerator

  DSG::Fourier::FourierSeriesGenerator - Generates a wave form using a user specified Fourier Series.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::Fourier

  DSG::Fourier - Namespace Containing Fourier Series Based Oscillators.

8.82 FourierSeries.h

```
00001 //
00002 //  FourierSeries.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 11/18/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__FourierSeries__
00009 #define __DSG__FourierSeries__
00010 #include "SignalGenerator.h"
00011 #include <vector>
00012 namespace DSG{
00013 #ifdef DSG_Short_Names
00014     inline
00015 #endif
00016     namespace Fourier{
00017         //!\brief DSG::Fourier::Harmonic - Represents a single harmonic in a Fourier Series.
00018         class Harmonic{
00019         public:
00020             Harmonic();
00021             Harmonic(DSG::DSGSample const& ratio,
      DSG::DSGSample const& amplitude);
00022             virtual ~Harmonic();
00023             DSG::DSGSample const& Ratio()const;
00024             DSG::DSGSample const& Ratio(DSG::DSGSample const& value);
00025             DSG::DSGSample const& Amplitude()const;
00026             DSG::DSGSample const& Amplitude(
      DSG::DSGSample const& value);
00027         protected:
00028             DSG::DSGSample _ratio;
00029             DSG::DSGSample _amplitude;
00030         };
00031         //!\brief DSG::Fourier::FourierSeriesGenerator - Generates a wave form using a user specified
       Fourier Series
00032         class FourierSeriesGenerator: public
      DSG::SignalGenerator{
00033         public:
00034             typedef std::vector<Harmonic> FourierSeries;
00035             FourierSeriesGenerator();
00036             FourierSeriesGenerator(DSG::DSGFrequency const&
      frequency, DSG::DSGPhase const& offset);
00037             virtual ~FourierSeriesGenerator();
00038             virtual inline bool Perform(DSG::DSGSample& signal);
00039             virtual inline bool Perform(DSG::RingBuffer& signal);
00040             inline void Series(FourierSeries const& series);
00041             inline FourierSeries& Series();
00042         protected:
00043             FourierSeries _series;
00044             DSG::DSGSample value;
00045         };
00046         inline bool DSG::Fourier::FourierSeriesGenerator::Perform
      (DSG::DSGSample& signal){
00047             value = _phasor;
00048             signal=0;
00049             for (auto i = _series.begin(); i!=_series.end(); ++i) {
00050                 signal += DSG::Sin(_phasor * i->Ratio())*i->Amplitude();
00051             }
00052             step();
00053             return true;
00054         }
00055         inline bool DSG::Fourier::FourierSeriesGenerator::Perform
      (DSG::RingBuffer& signal){
00056             signal.Flush();
00057             while (!signal.Full()) {
00058                 if (Perform(_storage)) {
00059                     if(signal.Write(_storage)){
00060                     }else return false;
00061                 }else return false;
00062             }return true;
00063         }
00064         inline void DSG::Fourier::FourierSeriesGenerator::Series
```

```
      (DSG::Fourier::FourierSeriesGenerator::FourierSeries
      const& series){
00065             _series = series;
00066         }
00067         inline DSG::Fourier::FourierSeriesGenerator::FourierSeries
      & DSG::Fourier::FourierSeriesGenerator::Series(){
00068             return _series;
00069         }
00070     }
00071 }
00072 #endif /* defined(__DSG__FourierSeries__) */
```

8.83 /Users/alexanderzywicki/Documents/DSG/src/FourierSquare.cpp File Reference

```
#include "FourierSquare.h"
```

8.84 FourierSquare.cpp

```
00001 //
00002 //  FourierSquare.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "FourierSquare.h"
00009 DSG::Fourier::FourierSquare::FourierSquare():
      DSG::SignalGenerator(),_a(3.6/PI),phs(0),value(0),i(0){}
00010 DSG::Fourier::FourierSquare::FourierSquare(
      DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):
      DSG::SignalGenerator(frequency,offset),_a(3.6/PI),phs(0),value(0),i(0){
00011     _h = MaxHarms(_frequency)+1;
00012 }
00013 DSG::Fourier::FourierSquare::~FourierSquare(){}
```

8.85 /Users/alexanderzywicki/Documents/DSG/src/FourierSquare.h File Reference

```
#include "SignalGenerator.h"
```

Classes

- class DSG::Fourier::FourierSquare

  DSG::Fourier::FourierSquare - Fourier Series Square Wave Generator.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::Fourier

  DSG::Fourier - Namespace Containing Fourier Series Based Oscillators.

8.86 FourierSquare.h

```
00001 //
00002 //  FourierSquare.h
00003 //  DSG
00004 //
```

```
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__FourierSquare__
00009 #define __DSG__FourierSquare__
00010 #include "SignalGenerator.h"
00011 namespace DSG{
00012 #ifdef DSG_Short_Names
00013     inline
00014 #endif
00015     //!DSG::Fourier - Namespace Containing Fourier Series Based Oscillators
00016     namespace Fourier{
00017         //!\brief DSG::Fourier::FourierSquare - Fourier Series Square Wave Generator
00018         class FourierSquare : public DSG::SignalGenerator {
00019         public:
00020             FourierSquare();
00021             FourierSquare(DSG::DSGFrequency const& frequency,
      DSG::DSGPhase const& offset);
00022             virtual ~FourierSquare();
00023             virtual inline bool Perform(DSG::DSGSample& signal);
00024             virtual inline bool Perform(DSG::RingBuffer& signal);
00025             virtual inline DSG::DSGFrequency const& Frequency(
      DSG::DSGFrequency const& value);
00026         protected:
00027             unsigned long _h;
00028             const double _a;
00029             double phs;
00030             double value;
00031             int i;
00032         };
00033         inline bool DSG::Fourier::FourierSquare::Perform(
      DSG::DSGSample& signal){
00034             //(_h/2)+1 Sine Calls Per Sample
00035             value=DSG::Sin(_phasor);//i=1
00036             for (i=3; i<_h; i+=2) {//i=3..5..7..
00037                 value += (1.0/i) * DSG::Sin(_phasor*i);
00038             }
00039             value*=_a;
00040             signal = value;
00041             step();
00042             return true;
00043         }
00044         inline bool DSG::Fourier::FourierSquare::Perform(
      DSG::RingBuffer& signal){
00045             signal.Flush();
00046             while (!signal.Full()) {
00047                 if (Perform(_storage)) {
00048                     if(signal.Write(_storage)){
00049                     }else return false;
00050                 }else return false;
00051             }return true;
00052         }
00053         inline DSG::DSGFrequency const&
      DSG::Fourier::FourierSquare::Frequency(
      DSG::DSGFrequency const& value){
00054             _frequency = value;
00055             _dt = _frequency/DSG::SampleRate();
00056             _h = MaxHarms(_frequency);
00057             return _frequency;
00058         }
00059     }
00060 }
00061 #endif /* defined(__DSG__FourierSquare__) */
```

8.87 /Users/alexanderzywicki/Documents/DSG/src/FourierTriangle.cpp File Reference

```
#include "FourierTriangle.h"
```

8.88 FourierTriangle.cpp

```
00001 //
00002 //  FourierTriangle.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
```

```
00008 #include "FourierTriangle.h"
00009 DSG::Fourier::FourierTriangle::FourierTriangle():
      DSG::SignalGenerator(),_a(8.0/(PI*PI)),phs(0),value(0),i(0){}
00010 DSG::Fourier::FourierTriangle::FourierTriangle(
      DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):
      DSG::SignalGenerator(frequency,offset),_a(8.0/(PI*PI)),phs(0),value(0),i(0){
00011     _h = MaxHarms(_frequency)+1;
00012 }
00013 DSG::Fourier::FourierTriangle::~FourierTriangle(){}
```

8.89 /Users/alexanderzywicki/Documents/DSG/src/FourierTriangle.h File Reference

```
#include "SignalGenerator.h"
```

Classes

- class DSG::Fourier::FourierTriangle

  DSG::Fourier::FourierTriangle - Fourier Series Triangle Wave Generator.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::Fourier

  DSG::Fourier - Namespace Containing Fourier Series Based Oscillators.

8.90 FourierTriangle.h

```
00001 //
00002 //  FourierTriangle.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__FourierTriangle__
00009 #define __DSG__FourierTriangle__
00010 #include "SignalGenerator.h"
00011 namespace DSG{
00012 #ifdef DSG_Short_Names
00013     inline
00014 #endif
00015     //!DSG::Fourier - Namespace Containing Fourier Series Based Oscillators
00016     namespace Fourier{
00017         //!\brief DSG::Fourier::FourierTriangle - Fourier Series Triangle Wave Generator
00018         class FourierTriangle : public DSG::SignalGenerator {
00019         public:
00020             FourierTriangle();
00021             FourierTriangle(DSG::DSGFrequency const& frequency,
      DSG::DSGPhase const& offset);
00022             virtual ~FourierTriangle();
00023             virtual inline bool Perform(DSG::DSGSample& signal);
00024             virtual inline bool Perform(DSG::RingBuffer& signal);
00025             virtual inline DSG::DSGFrequency const& Frequency(
      DSG::DSGFrequency const& value);
00026         protected:
00027             unsigned long _h;
00028             const double _a;
00029             double phs;
00030             double value;
00031             int i;
00032         };
00033         inline bool DSG::Fourier::FourierTriangle::Perform(
      DSG::DSGSample& signal){
00034             //(_h/2)+1 Sine Calls Per Sample
00035             value=DSG::Sin(_phasor);//i=1
00036             double sgn = -1;
```

```
00037             for (i=3; i<_h; i+=2) {//i=3..5..7..
00038                 value += sgn * (1.0/(i*i)) * DSG::Sin(_phasor*i);
00039                 sgn*=-1;
00040             }
00041             value*=_a;
00042             signal = value;
00043             step();
00044             return true;
00045         }
00046         inline bool DSG::Fourier::FourierTriangle::Perform(
      DSG::RingBuffer& signal){
00047             signal.Flush();
00048             while (!signal.Full()) {
00049                 if (Perform(_storage)) {
00050                     if(signal.Write(_storage)){
00051                     }else return false;
00052                 }else return false;
00053             }return true;
00054         }
00055         inline DSG::DSGFrequency const&
      DSG::Fourier::FourierTriangle::Frequency(
      DSG::DSGFrequency const& value){
00056             _frequency = value;
00057             _dt = _frequency/DSG::SampleRate();
00058             _h = MaxHarms(_frequency);
00059             return _frequency;
00060         }
00061     }
00062 }
00063 #endif /* defined(__DSG__FourierTriangle__) */
```

8.91 /Users/alexanderzywicki/Documents/DSG/src/Gaussian.h File Reference

```
#include "Sine.h"
#include "White.h"
```

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::Noise

  DSG::Noise - Noise Generators.

Functions

- template<typename decimal = DSG::DSGSample>
  decimal DSG::Noise::Gaussian (decimal=0.0)

  DSG::Noise::Gaussian - Gaussian Noise Generator Function.

8.92 Gaussian.h

```
00001 //
00002 //  Gaussian.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/6/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef DSG_Gaussian_h
00009 #define DSG_Gaussian_h
00010 #include "Sine.h"
00011 #include "White.h"
00012 namespace DSG{
00013 #ifdef DSG_Short_Names
00014     inline
00015 #endif
```

```
00016     namespace Noise{
00017         //!\brief DSG::Noise::Gaussian - Gaussian Noise Generator Function
00018         template<typename decimal=DSG::DSGSample>
00019         decimal Gaussian(decimal=0.0){
00020             static decimal normalizer=1;//variable used to actively normalize the output
00021             //to enforce compatability with DSG::LUT a dummy parameter is applied
00022             //this parameter is useless except for compatability reasons
00023             decimal R1 = DSG::Noise::White();
00024             decimal R2 = DSG::Noise::White();
00025             decimal x= (decimal)sqrt(-2.0f * log(R1))*DSG::Cos(R2);
00026             if (DSG::Abs(x)>normalizer) {
00027                 //store highest output
00028                 normalizer=DSG::Abs(x);
00029             }
00030             x/=normalizer;//normalize
00031             return x;
00032         }
00033     }
00034 }
00035 #endif
```

8.93 /Users/alexanderzywicki/Documents/DSG/src/GenericGenerator.cpp File Reference

```
#include "GenericGenerator.h"
```

8.94 GenericGenerator.cpp

```
00001 //
00002 //  GenericGenerator.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/21/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "GenericGenerator.h"
00009 DSG::GenericGenerator::GenericGenerator():
      DSG::SignalGenerator(){}
00010 DSG::GenericGenerator::GenericGenerator(
      DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset,
      DSG::DSGSample (*signalFunction)(DSG::DSGSample const&)):
      DSG::SignalGenerator(frequency,offset),_callback(signalFunction){}
00011 DSG::GenericGenerator::~GenericGenerator(){}
```

8.95 /Users/alexanderzywicki/Documents/DSG/src/GenericGenerator.h File Reference

```
#include "SignalGenerator.h"
```

Classes

- class DSG::GenericGenerator

  DSG::GenericGenerator - Generator designed to use a stateless generator function such as DSG::Sin()

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.

8.96 GenericGenerator.h

```
00001 //
```

```
00002 //  GenericGenerator.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/21/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__GenericGenerator__
00009 #define __DSG__GenericGenerator__
00010 #include "SignalGenerator.h"
00011 namespace DSG{
00012     //!\brief DSG::GenericGenerator - Generator designed to use a stateless generator function such as
       DSG::Sin()
00013     class GenericGenerator:public DSG::SignalGenerator{
00014     public:
00015         GenericGenerator();
00016         GenericGenerator(DSG::DSGFrequency const& frequency,
      DSG::DSGPhase const& offset,DSG::DSGSample (*signalFunction)(
      DSG::DSGSample const&));
00017         virtual ~GenericGenerator();
00018         virtual inline bool Perform(DSG::DSGSample& signal);
00019         virtual inline bool Perform(DSG::RingBuffer& signal);
00020     protected:
00021         DSG::DSGSample (*_callback)(DSG::DSGSample const&);
00022     };
00023     inline bool DSG::GenericGenerator::Perform(
      DSG::DSGSample& signal){
00024         if (_callback!=nullptr) {
00025             signal = _callback(_phasor);
00026         }else signal = 0;
00027         step();
00028         return true;
00029     }
00030     inline bool DSG::GenericGenerator::Perform(
      DSG::RingBuffer& signal){
00031         signal.Flush();
00032         while (!signal.Full()) {
00033             if (Perform(_storage)) {
00034                 if(signal.Write(_storage)){
00035                 }else return false;
00036             }else return false;
00037         }return true;
00038     }
00039 }
00040 #endif /* defined(__DSG__GenericGenerator__) */
```

8.97 /Users/alexanderzywicki/Documents/DSG/src/Interpolate.h File Reference

```
#include "DSGMath.h"
#include "PI.h"
```

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.

Functions

- template<typename decimal >
  decimal DSG::LinearInterpolate (decimal const &y1, decimal const &y2, decimal const &mu)

  DSG::LinearInterpolate - Linear Interpolation.
- template<typename decimal >
  decimal DSG::CosineInterpolate (decimal y1, decimal y2, decimal mu)

  DSG::CosineInterpolate - Cosine Interpolation.
- template<typename decimal >
  decimal DSG::CubicInterpolate (decimal const &y0, decimal const &y1, decimal const &y2, decimal const &y3, decimal const &mu)

  DSG::CubicInterpolate - Cubic Interpolation.

- template<typename decimal >
  decimal DSG::HermiteInterpolate (decimal const &y0, decimal const &y1, decimal const &y2, decimal const &y3, decimal const &mu, decimal const &tension, decimal const &bias)

  DSG::HermiteInterpolate - Hermite Interpolation.

8.98 Interpolate.h

```
00001 //
00002 //  Interpolate.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/21/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 //Code In this file was adapted from the code provided on this website
00009 //http://paulbourke.net/miscellaneous/interpolation/
00010 //
00011 #ifndef DSG_Interpolate_h
00012 #define DSG_Interpolate_h
00013 #include "DSGMath.h"
00014 #include "PI.h"
00015 namespace DSG{
00016     //!\brief DSG::LinearInterpolate - Linear Interpolation
00017     template<typename decimal>
00018     decimal LinearInterpolate(decimal const& y1,decimal const& y2,decimal const& mu){
00019         return(y1*(1-mu)+y2*mu);
00020     }
00021     //!\brief DSG::CosineInterpolate - Cosine Interpolation
00022     template<typename decimal>
00023     decimal CosineInterpolate(
00024                               decimal y1,decimal y2,
00025                               decimal mu)
00026     {
00027         decimal mu2;
00028         mu2 = (1-cos(mu*PI))/2.0;
00029         return(y1*(1-mu2)+y2*mu2);
00030     }
00031     //!\brief DSG::CubicInterpolate - Cubic Interpolation
00032     template<typename decimal>
00033     decimal CubicInterpolate(decimal const& y0,decimal const& y1,
00034                              decimal const& y2,decimal const& y3,
00035                              decimal const& mu)
00036     {
00037         decimal a0,a1,a2,a3,mu2;
00038         mu2 = mu*mu;
00039         a0 = y3 - y2 - y0 + y1;
00040         a1 = y0 - y1 - a0;
00041         a2 = y2 - y0;
00042         a3 = y1;
00043         return(a0*mu*mu2+a1*mu2+a2*mu+a3);
00044     }
00045     //!\brief DSG::HermiteInterpolate - Hermite Interpolation
00046     template<typename decimal>
00047     decimal HermiteInterpolate(decimal const& y0,decimal const& y1,
00048                                decimal const& y2,decimal const& y3,
00049                                decimal const& mu,
00050                                decimal const& tension,
00051                                decimal const& bias)
00052     {
00053         /*
00054          Tension: 1 is high, 0 normal, -1 is low
00055          Bias: 0 is even,
00056          positive is towards first segment,
00057          negative towards the other
00058          */
00059         decimal m0,m1,mu2,mu3;
00060         decimal a0,a1,a2,a3;
00061         mu2 = mu * mu;
00062         mu3 = mu2 * mu;
00063         m0  = (y1-y0)*(1+bias)*(1-tension)/2.0;
00064         m0 += (y2-y1)*(1-bias)*(1-tension)/2.0;
00065         m1  = (y2-y1)*(1+bias)*(1-tension)/2.0;
00066         m1 += (y3-y2)*(1-bias)*(1-tension)/2.0;
00067         a0 =  2*mu3 - 3*mu2 + 1;
00068         a1 =    mu3 - 2*mu2 + mu;
00069         a2 =    mu3 -   mu2;
00070         a3 = -2*mu3 + 3*mu2;
00071         return(a0*y1+a1*m0+a2*m1+a3*y2);
00072     }
00073 }
00074 #endif
```

8.99 /Users/alexanderzywicki/Documents/DSG/src/Leaky.cpp File Reference

```
#include "Leaky.h"
```

8.100 Leaky.cpp

```
00001 //
00002 //  Leaky.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/27/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "Leaky.h"
00009 DSG::Filter::LeakyIntegrator::LeakyIntegrator():
      DSG::Filter::FilterBase(){
00010     x1=0;
00011     y1=0;
00012     a=0;
00013     b=0;
00014     y=0;
00015 }
00016 DSG::Filter::LeakyIntegrator::LeakyIntegrator(
      DSG::DSGFrequency const& cutoff):DSG::Filter::FilterBase(){
00017     x1=0;
00018     y1=0;
00019     a=0;
00020     b=0;
00021     y=0;
00022     Cutoff(cutoff);
00023 }
00024 DSG::Filter::LeakyIntegrator::~LeakyIntegrator(){
00025     x1=0;
00026     y1=0;
00027     a=0;
00028     b=0;
00029     y=0;
00030 }
```

8.101 /Users/alexanderzywicki/Documents/DSG/src/Leaky.h File Reference

```
#include "Filter.h"
#include "PI.h"
```

Classes

- class DSG::Filter::LeakyIntegrator

  DSG::Filter::LeakyIntegrator - Leaky integrator.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::Filter

  DSG::Filter - Filters.

8.102 Leaky.h

```
00001 //
00002 //  Leaky.h
```

```
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/27/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008
00009 #ifndef __DSG__Leaky__
00010 #define __DSG__Leaky__
00011 #include "Filter.h"
00012 #include "PI.h"
00013 namespace DSG {
00014 #ifdef DSG_Short_Names
00015     inline
00016 #endif
00017     namespace Filter{
00018         //!\brief DSG::Filter::LeakyIntegrator - Leaky integrator
00019         class LeakyIntegrator:public DSG::Filter::FilterBase{
00020         public:
00021             LeakyIntegrator();
00022             LeakyIntegrator(DSG::DSGFrequency const& cutoff);
00023             virtual ~LeakyIntegrator();
00024             virtual inline bool Perform(DSG::DSGSample& signal);
00025             virtual inline bool Perform(DSG::RingBuffer& signal);
00026             virtual inline bool Cutoff(DSG::DSGFrequency const& cutoff);
00027         protected:
00028             double x1,y1,a,b;
00029             double y;
00030         };
00031         inline bool DSG::Filter::LeakyIntegrator::Perform(
      DSG::DSGSample& signal){
00032             y = b * (signal + x1) - a * y1;
00033             x1=signal;
00034             y1=y;
00035             signal=y;
00036             return true;
00037         }
00038         inline bool DSG::Filter::LeakyIntegrator::Perform(
      DSG::RingBuffer& signal){
00039             if (!signal.Empty()) {
00040                 count = signal.Count();
00041                 while (count-- > 0) {
00042                     if(signal.Read(_temp)){
00043                         if (Perform(_temp)) {
00044                             signal.Write(_temp);
00045                         }else return false;
00046                     }else return false;
00047                 }return true;
00048             }else return false;
00049         }
00050         inline bool DSG::Filter::LeakyIntegrator::Cutoff(
      DSG::DSGFrequency const& cutoff){
00051             double Omega;
00052             x1 = y1 = 0.0;
00053             Omega = atan(PI * cutoff);
00054             a = -(1.0 - Omega) / (1.0 + Omega);
00055             b = (1.0 - b) / 2.0;
00056             return true;
00057         }
00058     }
00059 }
00060 #endif /* defined(__DSG__Leaky__) */
```

8.103 /Users/alexanderzywicki/Documents/DSG/src/LUT.h File Reference

```
#include "Interpolate.h"
```

Classes

- class DSG::LUT< element, size >

  DSG::LUT - Look Up Table.

Namespaces

- DSG

DSG - A Collection of tools for Digital Signal Generation.

8.104 LUT.h

```
00001 //
00002 //  LUT.h
00003 //  Waveform
00004 //
00005 //  Created by Alexander Zywicki on 8/25/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef Waveform_LUT_h
00009 #define Waveform_LUT_h
00010 #ifdef DEBUG
00011 #include <assert.h>
00012 #endif
00013 #include "Interpolate.h"
00014 namespace DSG{
00015     //!\brief DSG::LUT - Look Up Table
00016     template <typename element,unsigned long size>
00017     class LUT {
00018     public:
00019         typedef element (*FillFunction)(element);
00020         typedef element (*FillFunctionConstRef)(element const&);
00021         LUT():_size(size){}
00022         LUT(FillFunction fill,double const& range = 1.0):_size(size){
00023             //range is the expected input range for the function
00024             //example would  be 0-2pi or 0-1
00025             //would be provided a 2pi or 1
00026             //defaults to 1
00027             double step = range/(double)_size;
00028             phs = 0;
00029             for (int i=0; i<_size; ++i) {
00030                 _table[i] = fill(phs);
00031                 phs+=step;
00032             }
00033         }
00034         LUT(FillFunctionConstRef fill,double const& range = 1.0):
    _size(size){
00035             //range is the expected input range for the function
00036             //example would  be 0-2pi or 0-1
00037             //would be provided a 2pi or 1
00038             //defaults to 1
00039             double step = range/_size;
00040             phs = 0;
00041             for (int i=0; i<_size; ++i) {
00042                 _table[i] = fill(phs);
00043                 phs+=step;
00044             }
00045         }
00046         ~LUT(){}
00047         element const& operator[](unsigned long const& index)const{
00048 #ifdef DEBUG
00049             assert(index<_size);
00050 #endif
00051             return _table[index];
00052         }
00053         element& operator[](unsigned long const& index){
00054 #ifdef DEBUG
00055             assert(index<_size);
00056 #endif
00057             return _table[index];
00058         }
00059         inline element const&  operator()(double const& x){
00060             phs=x;
00061             //need range checking on x to ensure 0-1 range
00062             phs<0 ? phs = 1-(phs*-1):0;
00063             phs-=((int)phs);
00064             return this->_table[(unsigned)(phs*(this->_size-1))];
00065         }
00066         unsigned long const& Size()const{
00067             return _size;
00068         }
00069     protected:
00070         element _table[size];
00071         const unsigned long _size;
00072         double phs;
00073     };
00074 }
00075 #endif
```

8.105 /Users/alexanderzywicki/Documents/DSG/src/Noise.h File Reference

```
#include "Random.h"
#include "Gaussian.h"
#include "White.h"
#include "Pink.h"
#include "NoiseGenerator.h"
```

8.106 Noise.h

```
00001 //
00002 //  Noise.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/20/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef DSG_Noise_h
00009 #define DSG_Noise_h
00010 #include "Random.h"
00011 #include "Gaussian.h"
00012 #include "White.h"
00013 #include "Pink.h"
00014 #include "NoiseGenerator.h"
00015 #endif
```

8.107 /Users/alexanderzywicki/Documents/DSG/src/NoiseGenerator.cpp File Reference

```
#include "NoiseGenerator.h"
```

8.108 NoiseGenerator.cpp

```
00001 //
00002 //  NoiseGenerator.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/20/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "NoiseGenerator.h"
00009 DSG::NoiseGenerator::NoiseGenerator(DSGSample (*
      StatelessFunction)(DSGSample)):DSG::SignalProcess(){
00010     _function = StatelessFunction;
00011 }
00012 DSG::NoiseGenerator::~NoiseGenerator(){}
```

8.109 /Users/alexanderzywicki/Documents/DSG/src/NoiseGenerator.h File Reference

```
#include "SignalGenerator.h"
```

Classes

- class DSG::NoiseGenerator

  DSG::NoiseGenerator - Generator that uses noise functions such as DSG::White() to generate signal.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.

8.110 NoiseGenerator.h

```
00001 //
00002 //  NoiseGenerator.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/20/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__NoiseGenerator__
00009 #define __DSG__NoiseGenerator__
00010 #include "SignalGenerator.h"
00011 namespace DSG{
00012     //!\brief DSG::NoiseGenerator - Generator that uses noise functions such as DSG::White() to generate
       signal
00013     class NoiseGenerator:public SignalProcess{
00014     public:
00015         NoiseGenerator(DSGSample (*StatelessFunction)(
      DSGSample));
00016         virtual ~NoiseGenerator();
00017         virtual inline bool Perform(DSG::DSGSample& signal);
00018         virtual inline bool Perform(DSG::RingBuffer& signal);
00019     protected:
00020         DSGSample (*_function)(DSGSample);
00021         DSG::DSGSample _storage;
00022     };
00023     inline bool DSG::NoiseGenerator::Perform(
      DSG::DSGSample& signal){
00024         signal = _function(0);
00025         return true;
00026     }
00027     inline bool DSG::NoiseGenerator::Perform(
      DSG::RingBuffer& signal){
00028         signal.Flush();
00029         while (!signal.Full()) {
00030             if (Perform(_storage)) {
00031                 if(signal.Write(_storage)){
00032                 }else return false;
00033             }else return false;
00034         }return true;
00035     }
00036 }
00037 #endif /* defined(__DSG__NoiseGenerator__) */
```

8.111 /Users/alexanderzywicki/Documents/DSG/src/PI.h File Reference

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.

Macros

- #define PI 3.14159265358979323846264338327
- #define TWOPI 6.28318530717958647692528676656

8.111.1 Macro Definition Documentation

8.111.1.1 #define PI 3.14159265358979323846264338327

Definition at line 11 of file PI.h.

8.111.1.2 #define TWOPI 6.2831853071795864769252867656

Definition at line 12 of file PI.h.

8.112 PI.h

```
00001 //
00002 //  PI.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef DSG_PI_h
00009 #define DSG_PI_h
00010 namespace DSG{
00011 #define PI    3.14159265358979323846264338327
00012 #define TWOPI 6.28318530717958647692528676656
00013 }
00014 #endif
```

8.113 /Users/alexanderzywicki/Documents/DSG/src/Pink.h File Reference

```
#include "Gaussian.h"
#include "DCBlocker.h"
```

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::Noise

  DSG::Noise - Noise Generators.

Functions

- template<typename decimal = DSG::DSGSample>
  decimal DSG::Noise::Pink (decimal=0.0)

  DSG::Noise::Pink - Pink Noise Generator Function.

8.114 Pink.h

```
00001 //
00002 //  Pink.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/8/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef DSG_Pink_h
00009 #define DSG_Pink_h
00010 #include "Gaussian.h"
00011 #include "DCBlocker.h"
00012 namespace DSG{
00013 #ifdef DSG_Short_Names
00014     inline
00015 #endif
00016     namespace Noise{
00017         //!\brief DSG::Noise::Pink - Pink Noise Generator Function
00018         template<typename decimal=DSG::DSGSample>
00019         decimal Pink(decimal=0.0){
00020             //routine: Get white or gaussian, filter, return
```

```
00021             static decimal b0,b1,b2,b3,b4,b5,b6;
00022             static decimal normalizer=1;//variable used to actively normalize the output
00023             static DSG::DCBlocker _block;
00024             decimal white = DSG::Noise::Gaussian();
00025             decimal pink;
00026             //pinking filter
00027             b0 = 0.99886 * b0 + white * 0.0555179;
00028             b1 = 0.99332 * b1 + white * 0.0750759;
00029             b2 = 0.96900 * b2 + white * 0.1538520;
00030             b3 = 0.86650 * b3 + white * 0.3104856;
00031             b4 = 0.55000 * b4 + white * 0.5329522;
00032             b5 = -0.7616 * b5 - white * 0.0168980;
00033             pink = b0 + b1 + b2 + b3 + b4 + b5 + b6 + white * 0.5362;
00034             b6 = white * 0.115926;
00035             if (DSG::Abs(pink)>normalizer) {
00036                 //store highest output
00037                 normalizer=DSG::Abs(pink);
00038             }
00039             pink/=normalizer;
00040             _block.Perform(pink);
00041             return pink;
00042         }
00043     }
00044 }
00045 #endif
```

8.115 /Users/alexanderzywicki/Documents/DSG/src/Random.h File Reference

```
#include "DSGTypes.h"
#include <random>
```

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::Noise

  DSG::Noise - Noise Generators.

Functions

- template<typename decimal = DSG::DSGSample>
  decimal DSG::Noise::Random (decimal=0.0)

  DSG::Noise::Random - Random Number Function.

8.115.1 Variable Documentation

8.115.1.1 const decimal max = static_cast<decimal>(RAND_MAX)

Definition at line 29 of file Random.h.

8.116 Random.h

```
00001 //
00002 //  Random.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/28/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef DSG_Random_h
00009 #define DSG_Random_h
00010 #include "DSGTypes.h"
```

```
00011 #include <random>
00012 namespace DSG{
00013 #ifdef DSG_Short_Names
00014     inline
00015 #endif
00016     //!\brief DSG::Noise - Noise Generators
00017     namespace Noise{
00018         namespace{
00019             template<typename decimal>
00020             class random_helper{
00021             public:
00022                 random_helper(){
00023                     srand(static_cast<unsigned>(time(NULL)));
00024                 }
00025                 inline decimal next(){
00026                     return static_cast<decimal>(rand()/max);
00027                 }
00028             protected:
00029                 const decimal max = static_cast<decimal>(RAND_MAX);
00030             };
00031         }
00032         //!\brief DSG::Noise::Random - Random Number Function
00033         template<typename decimal = DSG::DSGSample>
00034         inline decimal Random(decimal=0.0){
00035             static DSG::Noise::random_helper<decimal> _rand{};
00036             return _rand.next();
00037         }
00038     }
00039 }
00040 #endif
```

8.117 /Users/alexanderzywicki/Documents/DSG/src/RingBuffer.cpp File Reference

```
#include "RingBuffer.h"
```

8.118 RingBuffer.cpp

```
00001 //
00002 //  RingBuffer.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "RingBuffer.h"
00009 DSG:: RingBuffer::RingBuffer():Buffer(0),_read(0),_write(0),_count(0),
      MASK(0){}
00010 DSG:: RingBuffer::RingBuffer(const size_t size):
      Buffer(make_pow_2(size)),_read(0),_write(0),_count(0){
00011     MASK = this->_size-1;
00012 }
00013 DSG:: RingBuffer::RingBuffer(RingBuffer& buffer):
      Buffer(buffer){
00014     _write.store(buffer._write.load(std::memory_order_acquire));
00015     _read.store(buffer._read.load(std::memory_order_acquire));
00016     _count = buffer._count;
00017     MASK = buffer._size-1;
00018 }
00019 DSG:: RingBuffer& DSG:: RingBuffer::operator=(
      RingBuffer& buffer){
00020     Buffer::operator=(buffer);
00021     _write.store(buffer._write.load(std::memory_order_acquire));
00022     _read.store(buffer._read.load(std::memory_order_acquire));
00023     _count = buffer._count;
00024     MASK = buffer._size-1;
00025     return *this;
00026 }
00027 DSG:: RingBuffer::~RingBuffer(){Flush();}
00028
```

8.119 /Users/alexanderzywicki/Documents/DSG/src/RingBuffer.h File Reference

```
#include <atomic>
#include "DSGMath.h"
#include "Buffer.h"
```

Classes

- class DSG::RingBuffer

  DSG::RingBuffer - Circular Buffer of Audio.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.

8.120 RingBuffer.h

```
00001 //
00002 //  RingBuffer.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__RingBuffer__
00009 #define __DSG__RingBuffer__
00010 #ifdef DEBUG
00011 #include <iostream>
00012 #endif
00013 #include <atomic>
00014 #include "DSGMath.h"
00015 #include "Buffer.h"
00016 namespace DSG {
00017     /*!\brief DSG::RingBuffer - Circular Buffer of Audio
00018      */
00019     class RingBuffer:public DSG::Buffer {
00020     protected:
00021         std::atomic<size_t> _write;
00022         std::atomic<size_t> _read;
00023         size_t _count;
00024         size_t MASK;
00025         size_t write;
00026         size_t read;
00027         inline size_t next(size_t current);
00028         inline size_t make_pow_2(size_t number);
00029     public:
00030         RingBuffer();
00031         RingBuffer(const size_t size);
00032         RingBuffer(RingBuffer& buffer);
00033         RingBuffer& operator=(RingBuffer& buffer);
00034         virtual ~RingBuffer();
00035         inline bool Write(const DSGSample& elem);
00036         inline bool Read(DSG::DSGSample& elem);
00037         inline size_t const& Count()const;
00038         inline bool Full()const;
00039         inline bool Empty()const;
00040         inline void Flush();
00041         friend bool operator>>(DSG::DSGSample const& signal,
      DSG::RingBuffer& buffer){
00042             return buffer.Write(signal);
00043         }
00044         friend bool operator<<(DSG::DSGSample& signal,
      DSG::RingBuffer& buffer){
00045             return buffer.Read(signal);
00046         }
00047 #ifdef DEBUG
00048         friend std::ostream& operator<<(std::ostream& os,
      DSG:: RingBuffer const& buffer){
00049             if (!buffer.Empty()) {
00050                 size_t index= buffer._read;
```

```
00051                 size_t count=buffer.Count();
00052                 size_t size = buffer.Size();
00053                 for (int i=0; i<count; ++i) {
00054                     os<<index<<": "<<buffer._buffer[index]<<std::endl;
00055                     index = ((index+1)%size);
00056                 }
00057             }return os;
00058         }
00059 #endif
00060     };
00061     inline bool DSG::RingBuffer::Full()const{
00062         return _count==this->_size;
00063     }
00064     inline bool DSG::RingBuffer::Empty()const{
00065         return _count==0;
00066     }
00067     inline void DSG::RingBuffer::Flush(){
00068         _write.store(0,std::memory_order_relaxed);
00069         _read.store(0,std::memory_order_relaxed);
00070         _count=0;
00071     }
00072     inline bool DSG::RingBuffer::Write(const DSGSample& elem){
00073         if (!Full()) {
00074             write = _write.load(std::memory_order_acquire);
00075             _write.store(next(write),std::memory_order_release);
00076             this->_buffer[write] = elem;
00077             ++_count;
00078             return true;
00079         }else return false;
00080     }
00081     inline bool DSG::RingBuffer::Read(DSGSample& elem){
00082         if (!Empty()) {
00083             read = _read.load(std::memory_order_acquire);
00084             _read.store(next(read),std::memory_order_release);
00085             elem = this->_buffer[read];
00086             --_count;
00087             return true;
00088         }else return false;
00089     }
00090     inline size_t const& DSG::RingBuffer::Count()const{
00091         return _count;
00092     }
00093     //note: RingBuffer implementation will force a power of 2 size to allow use of bitwise increment.
00094     inline size_t DSG::RingBuffer::next(size_t current){return (current+1) & MASK;}
00095     inline size_t DSG::RingBuffer::make_pow_2(size_t number){
00096         return pow(2, ceil(log(number)/log(2)));
00097     }
00098 }
00099 #endif /* defined(__DSG__RingBuffer__) */
```

8.121 /Users/alexanderzywicki/Documents/DSG/src/SignalGenerator.cpp File Reference

```
#include "SignalGenerator.h"
```

8.122 SignalGenerator.cpp

```
00001 //
00002 //  SignalGenerator.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "SignalGenerator.h"
00009 DSG::SignalGenerator::SignalGenerator():DSG::
      SignalProcess(),_phasor(0),_frequency(0),_dt(0),_offset(0){}
00010 DSG::SignalGenerator::SignalGenerator(
      DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):_phasor(0),
      _frequency(frequency),_dt(0),_offset(offset){
00011     Frequency(frequency);
00012     Phase(offset);
00013 }
00014 DSG::SignalGenerator::~SignalGenerator(){}
```

8.123 /Users/alexanderzywicki/Documents/DSG/src/SignalGenerator.h File Reference

```
#include "SignalProcess.h"
#include "AudioSettings.h"
#include "Sine.h"
#include "Bounds.h"
```

Classes

- class DSG::SignalGenerator

  DSG::SignalGenerator - Extends DSG::Signal Process With Tools For Signal Generation.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.

Functions

- unsigned long DSG::MaxHarms (DSG::DSGFrequency const &frequency)

8.124 SignalGenerator.h

```
00001 //
00002 //  SignalGenerator.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__SignalGenerator__
00009 #define __DSG__SignalGenerator__
00010 #include "SignalProcess.h"
00011 #include "AudioSettings.h"
00012 #include "Sine.h"
00013 #include "Bounds.h"
00014 namespace DSG{
00015     /*!\brief DSG::SignalGenerator - Extends DSG::Signal Process With Tools For Signal Generation
00016      */
00017     class SignalGenerator:public DSG::SignalProcess{
00018     public:
00019         SignalGenerator();
00020         SignalGenerator(DSG::DSGFrequency const& frequency,
      DSG::DSGPhase const& offset);
00021         virtual ~SignalGenerator();
00022         virtual inline bool Perform(DSG::DSGSample& signal);
00023         virtual inline bool Perform(DSG::RingBuffer& signal);
00024         //Adds interface for control rate processing
00025         virtual inline DSG::DSGFrequency const& Frequency();
00026         virtual inline DSG::DSGFrequency const& Frequency(
      DSG::DSGFrequency const& value);
00027         virtual inline DSG::DSGPhase const& Phase();
00028         virtual inline DSG::DSGPhase const& Phase(
      DSG::DSGPhase const& value);
00029     protected:
00030         //extends sample rate interface
00031         inline void step();
00032         inline void sync();
00033         //----------------------------
00034         DSG::DSGFrequency _frequency;//frequency in Hz
00035         DSG::DSGPhase _dt;//delta time (change in phase per sample) unit: phase 0-1
00036         DSG::DSGPhase _offset;//phase shift
00037         DSG::DSGPhase _phasor;//phase counter
00038         DSG::DSGSample _storage;//storage variable for calculations
00039     };
00040     inline unsigned long MaxHarms(DSG::DSGFrequency const& frequency){
00041         double _s = DSG::SampleRate()*  20000.0/DSG::SampleRate();
```

```
00042         _s/=frequency;
00043         return _s;
00044     }
00045 }
00046 inline bool DSG::SignalGenerator::Perform(
      DSG::DSGSample& signal){
00047     signal=0;
00048     return false;
00049 }
00050 inline bool DSG::SignalGenerator::Perform(
      DSG::RingBuffer& signal){
00051     signal.Flush();
00052     return false;
00053 }
00054 inline DSG::DSGFrequency const& DSG::SignalGenerator::Frequency
      (){
00055     return _frequency;
00056 }
00057 inline DSG::DSGFrequency const& DSG::SignalGenerator::Frequency
      (DSG::DSGFrequency const& value){
00058     _frequency = DSG::EnforceBounds<0, 20000,DSG::DSGSample>(value);
00059     _dt = _frequency/DSG::SampleRate();
00060     return _frequency;
00061 }
00062 inline DSG::DSGPhase const& DSG::SignalGenerator::Phase(){
00063     return _offset;
00064 }
00065 inline DSG::DSGPhase const& DSG::SignalGenerator::Phase(
      DSG::DSGPhase const& value){
00066     _offset-=value;
00067     _phasor-=_offset;
00068     _offset=value;
00069     return _offset;
00070 }
00071 inline void DSG::SignalGenerator::step(){
00072     _phasor+=_dt;
00073     _phasor>1.0 ? --_phasor:0;
00074 }
00075 inline void DSG::SignalGenerator::sync(){
00076     _phasor=_offset;
00077 }
00078 #endif /* defined(__DSG__SignalGenerator__) */
```

8.125 /Users/alexanderzywicki/Documents/DSG/src/SignalProcess.cpp File Reference

```
#include "SignalProcess.h"
```

8.126 SignalProcess.cpp

```
00001 //
00002 //  SignalProcess.cpp
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #include "SignalProcess.h"
00009 DSG::SignalProcess::SignalProcess(){}
00010 DSG::SignalProcess::~SignalProcess(){}
```

8.127 /Users/alexanderzywicki/Documents/DSG/src/SignalProcess.h File Reference

```
#include "DSGTypes.h"
#include "RingBuffer.h"
```

Classes

- class DSG::SignalProcess

DSG::SignalProcess - Defines Base Interface For Audio Processing.

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.

8.128 SignalProcess.h

```
00001 //
00002 //  SignalProcess.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__SignalProcess__
00009 #define __DSG__SignalProcess__
00010 #include "DSGTypes.h"
00011 #include "RingBuffer.h"
00012 namespace DSG {
00013     /*!\brief DSG::SignalProcess - Defines Base Interface For Audio Processing
00014      */
00015     class SignalProcess{
00016     public:
00017         SignalProcess();
00018         virtual ~SignalProcess();
00019         //Defines Interface for sample rate processing
00020         virtual inline bool Perform(DSG::DSGSample& signal)=0;
00021         virtual inline bool Perform(DSG::RingBuffer& signal)=0;
00022     };
00023 }
00024 #endif /* defined(__DSG__SignalProcess__) */
```

8.129 /Users/alexanderzywicki/Documents/DSG/src/Sinc.h File Reference

```
#include "PI.h"
#include "Sine.h"
#include "Denormal.h"
#include <type_traits>
#include "DSGMath.h"
```

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.

Functions

- template<typename decimal >
  decimal DSG::Sinc (decimal const &x)

  DSG::Sinc - Implements the Sinc() function (sin(PI∗x)/PI∗x)

8.130 Sinc.h

```
00001 //
00002 //  Sinc.h
00003 //  DSG
```

```
00004 //
00005 //  Created by Alexander Zywicki on 9/23/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__Sinc__
00009 #define __DSG__Sinc__
00010 #include "PI.h"
00011 #include "Sine.h"
00012 #include "Denormal.h"
00013 #include <type_traits>
00014 #include "DSGMath.h"
00015 namespace DSG{
00016     //!\brief DSG::Sinc - Implements the Sinc() function (sin(PI*x)/PI*x)
00017     template<typename decimal>
00018     inline decimal Sinc(decimal const& x) {
00019         static_assert(std::is_floating_point<decimal>::value==true,"DSG::Sinc Function Requires Floating
      Point Type");
00020         decimal pix;
00021         if (DSG::IsDenormal(x)) {
00022             return 1.0;
00023         }else{
00024             pix = PI*x;
00025             return DSG::Sin(pix)/pix;
00026         }
00027     }
00028 }
00029 #endif /* defined(__DSG__Sinc__) */
```

8.131 /Users/alexanderzywicki/Documents/DSG/src/Sine.h File Reference

```
#include "LUT.h"
#include "PI.h"
```

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.

Macros

- #define LUT_SIZE 16384

Enumerations

- enum **Sine_Implementations**

Functions

- double DSG::Sin (double const &x)

  DSG::Sin() - General Purpose Sin Function, double precision.
- float DSG::Sin (float const &x)

  DSG::Sin() - General Purpose Sin Function, single precision.
- double DSG::Cos (double const &x)

  DSG::Cos() - General Purpose Cos Function, double precision.
- float DSG::Cos (float const &x)

  DSG::Cos() - General Purpose Cos Function, single precision.

8.131.1 Macro Definition Documentation

8.131.1.1 #define LUT_SIZE 16384

Definition at line 14 of file Sine.h.

8.132 Sine.h

```
00001 //
00002 //  Sine.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 9/16/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__Sine__
00009 #define __DSG__Sine__
00010 #include "LUT.h"
00011 #include "PI.h"
00012 namespace DSG {
00013     namespace{
00014             #define LUT_SIZE 16384
00015         typedef enum Sine_Implementations{
00016             /*!\brief DSG::Sine_Implementations - Specifies The Implementation Option For DSG::Sin<>()*/
00017             Sine_Taylor =1,
00018             Sine_LUT =2,
00019             Sine_Default = Sine_LUT
00020         }Sine_Implementations;
00021         /*!\brief DSG::Sin() - Templated Sin Function With Optional Implementation
00022          */
00023         template<unsigned implementation> inline double Sin(double const& x){
00024             return 0;
00025         }
00026         /*!\brief DSG::Sin() - Templated Cos Function With Optional Implementation
00027          */
00028         template<unsigned implementation> inline double Cos(double const& x){
00029             return 0;
00030         }
00031         template<> inline double Sin<Sine_LUT>(double const& x){
00032             static DSG::LUT<double, LUT_SIZE> _lut(&sin,
      TWOPI);
00033             return _lut(x);
00034         }
00035         template<> inline double Cos<Sine_LUT>(double const& x){
00036             static DSG::LUT<double, LUT_SIZE> _lut(&cos,
      TWOPI);
00037             return _lut(x);
00038         }
00039         template<> inline double Sin<Sine_Taylor>(double const& x){
00040             //taylor serie version here
00041             return 0;
00042         }
00043         template<> inline double Cos<Sine_Taylor>(double const& x){
00044             //taylor series version here
00045             return 0;
00046         }
00047     }
00048     /*!\brief DSG::Sin() - General Purpose Sin Function, double precision
00049      */
00050     inline double Sin(double const& x){
00051         return static_cast<double>(Sin<Sine_Default>(x));//wrap default implementation as non template
00052     }
00053     /*!\brief DSG::Sin() - General Purpose Sin Function, single precision
00054      */
00055     inline float Sin(float const& x){
00056         return static_cast<float>(Sin<Sine_Default>(x));
00057     }
00058     /*!\brief DSG::Cos() - General Purpose Cos Function, double precision
00059      */
00060     inline double Cos(double const& x){
00061         return static_cast<double>(Cos<Sine_Default>(x));//wrap default implementation as non template
00062     }
00063     /*!\brief DSG::Cos() - General Purpose Cos Function, single precision
00064      */
00065     inline float Cos(float const& x){
00066         return static_cast<float>(Cos<Sine_Default>(x));
00067     }
00068 }
00069 #endif /* defined(__DSG__Sine__) */
```

8.133 /Users/alexanderzywicki/Documents/DSG/src/Sleep.h File Reference

```
#include <chrono>
#include <thread>
```

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.

Functions

- template<typename integer >
  void DSG::Sleep (integer const &milliseconds)

  DSG::Sleep - Millisecond Sleep Function.

8.134 Sleep.h

```
00001 //
00002 //  Sleep.h
00003 //  DSG_Tests
00004 //
00005 //  Created by Alexander Zywicki on 10/5/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef __DSG__Sleep__
00009 #define __DSG__Sleep__
00010 #include <chrono>
00011 #include <thread>
00012 namespace DSG{
00013     //!\brief DSG::Sleep - Millisecond Sleep Function
00014     template<typename integer>
00015     void Sleep(integer const& milliseconds){
00016         std::this_thread::sleep_for(std::chrono::milliseconds(milliseconds));
00017     }
00018 }
00019 #endif /* defined(__DSG__Sleep__) */
```

8.135 /Users/alexanderzywicki/Documents/DSG/src/White.h File Reference

```
#include "DSGTypes.h"
#include "Random.h"
```

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::Noise

  DSG::Noise - Noise Generators.

Functions

- template<typename decimal = DSG::DSGSample>
  decimal DSG::Noise::White (decimal=0.0)

  DSG::Noise::White - White Noise Generator Function.

8.136 White.h

```
00001 //
00002 //  White.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/14/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef DSG_White_h
00009 #define DSG_White_h
00010 #include "DSGTypes.h"
00011 #include "Random.h"
00012 namespace DSG{
00013 #ifdef DSG_Short_Names
00014     inline
00015 #endif
00016     namespace Noise{
00017         //!\brief DSG::Noise::White - White Noise Generator Function
00018         template<typename decimal = DSG::DSGSample>
00019         inline decimal White(decimal=0.0){
00020             return DSG::Random<decimal>();
00021         }
00022     }
00023 }
00024 #endif
```

8.137 /Users/alexanderzywicki/Documents/DSG/src/Window.h File Reference

```
#include "LUT.h"
```

Namespaces

- DSG

  DSG - A Collection of tools for Digital Signal Generation.
- DSG::Window

  DSG::Window - Window functions and utilities.

Functions

- template<typename decimal , unsigned long lutsize>
  void DSG::Window::ApplyWindow (DSG::LUT< decimal, lutsize > &lut, decimal(&windowFunction)(decimal const &), decimal range=1.0)

  DSG::Window::ApplyWindow - Apply a window function to a LUT.
- template<typename decimal , unsigned long lutsize>
  void DSG::Window::ApplyWindow (DSG::LUT< decimal, lutsize > &lut, decimal(&windowFunction)(decimal), decimal range=1.0)

  DSG::Window::ApplyWindow - Apply a window function to a LUT.

8.138 Window.h

```
00001 //
00002 //  Window.h
00003 //  DSG
00004 //
00005 //  Created by Alexander Zywicki on 10/17/14.
00006 //  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
00007 //
00008 #ifndef DSG_Window_h
00009 #define DSG_Window_h
00010 #include "LUT.h"
00011 namespace DSG{
00012 #ifdef DSG_Short_Names
```

```
00013     inline
00014 #endif
00015     //!\brief DSG::Window - Window functions and utilities
00016     namespace Window{
00017         //!\brief DSG::Window::ApplyWindow - Apply a window function to a LUT
00018         template<typename decimal,unsigned long lutsize>
00019         void ApplyWindow(DSG::LUT<decimal,lutsize>& lut,decimal (&
      windowFunction)(decimal const&),decimal range = 1.0){
00020             decimal step = range/(decimal)lut.Size();
00021             decimal phs=0;
00022             for (int i=0; i<lut.Size(); ++i) {
00023                 lut[i]*=windowFunction(phs);
00024                 phs+=step;
00025             }
00026         }
00027         //!\brief DSG::Window::ApplyWindow - Apply a window function to a LUT
00028         template<typename decimal,unsigned long lutsize>
00029         void ApplyWindow(DSG::LUT<decimal,lutsize>& lut,decimal (&
      windowFunction)(decimal),decimal range = 1.0){
00030             decimal step = range/(decimal)lut.Size();
00031             decimal phs=0;
00032             for (int i=0; i<lut.Size(); ++i) {
00033                 lut[i]*=windowFunction(phs);
00034                 phs+=step;
00035             }
00036         }
00037     }
00038 }
00039 #endif
```

Index