DSG

Generated by Doxygen 1.8.8

Mon Nov 24 2014 22:55:28

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

  DSG::DSGSample - 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 28 of file DSGTypes.h.

6.1.2.2 typedef float DSG::DSGPhase

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

Definition at line 30 of file DSGTypes.h.

6.1.2.3 typedef float DSG::DSGSample

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

Definition at line 32 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 31 of file DSGMath.h.

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

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 37 of file BufferConversion.h.

```
                                                                            {
        int i=0;
        ring.Flush();
        while (!ring.Full()) {
            ring.Write(array[i]);
            ++i;
        }return true;
    }
```

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 44 of file Bounds.h.

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

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

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

Definition at line 76 of file Sine.h.

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

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

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

Definition at line 81 of file Sine.h.

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

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

DSG::CosineInterpolate - Cosine Interpolation.

Definition at line 39 of file Interpolate.h.

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

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 49 of file Interpolate.h.

```
    {
        decimal a0,a1,a2,a3,mu2;
        mu2 = mu*mu;
        a0 = y3 - y2 - y0 + y1;
        a1 = y0 - y1 - a0;
        a2 = y2 - y0;
        a3 = y1;
        return(a0*mu*mu2+a1*mu2+a2*mu+a3);
    }
```

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 30 of file DSF.h.

```
                                                                                             {
#ifdef __APPLE__
#warning Untested DSG::DSF()
#endif
        decimal denominator = 1 + DSG::Pow<2>(a) - (2.0*a*cos(beta));
        decimal numerator = sin(theta) - a * sin(theta-beta) - pow(a, N+1) * (sin(theta + (N+1)*beta) - a*
sin(theta + (N*beta)));
        return numerator/denominator;
    }
```

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 30 of file Bounds.h.

```
                                         {
        if (value<lower) {
            return lower;
        }else if(value> upper){
            return upper;
        }else return value;
    }
```

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 63 of file Interpolate.h.

```
    {
        /*
         Tension: 1 is high, 0 normal, -1 is low
         Bias: 0 is even,
         positive is towards first segment,
         negative towards the other
         */
        decimal m0,m1,mu2,mu3;
        decimal a0,a1,a2,a3;
        mu2 = mu * mu;
        mu3 = mu2 * mu;
        m0  = (y1-y0)*(1+bias)*(1-tension)/2.0;
        m0 += (y2-y1)*(1-bias)*(1-tension)/2.0;
        m1  = (y2-y1)*(1+bias)*(1-tension)/2.0;
        m1 += (y3-y2)*(1-bias)*(1-tension)/2.0;
        a0 =  2*mu3 - 3*mu2 + 1;
        a1 =    mu3 - 2*mu2 + mu;
        a2 =    mu3 -   mu2;
        a3 = -2*mu3 + 3*mu2;
        return(a0*y1+a1*m0+a2*m1+a3*y2);
    }
```

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 31 of file Denormal.h.

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

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 34 of file Interpolate.h.

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

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

Definition at line 56 of file SignalGenerator.h.

```
                                                                {
        double _s = DSG::SampleRate()*  20000.0/DSG::SampleRate();
        _s/=frequency;
        return _s;
    }
```

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 48 of file AudioSettings.h.

```
                                      {
        return DSG::AudioSettings::Nyquist();
    }
```

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 60 of file DSGMath.h.

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

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 29 of file BufferConversion.h.

```
                                                                                {
        for (int i=0; i<length; ++i) {
            if (!ring.Empty()) {
                ring.Read(array[i]);
            }
        }return true;
    }
```

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

DSG::SampleRate - Get Global Sample Rate.

Definition at line 40 of file AudioSettings.h.

```
                                                  {
        return DSG::AudioSettings::SampleRate();
    }
```

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

DSG::SampleRate - Set Global Sample Rate.

Definition at line 44 of file AudioSettings.h.

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

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

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

Definition at line 66 of file Sine.h.

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

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

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

Definition at line 71 of file Sine.h.

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

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 34 of file Sinc.h.

```
                                             {
        static_assert(std::is_floating_point<decimal>::value==true,"DSG::Sinc Function Requires Floating
 Point Type");
        decimal pix;
        if (DSG::IsDenormal(x)) {
            return 1.0;
        }else{
            pix = PI*x;
            return DSG::Sin(pix)/pix;
        }
    }
```

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

DSG::Sleep - Millisecond Sleep Function.

Definition at line 31 of file Sleep.h.

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

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 39 of file Bounds.h.

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

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 38 of file DPW.h.

```
00038                                                          {
00039             DSG::StaticAssertBounds<1,6,order>();//must be 1-6 order
00040             return value;
00041         }
```

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 44 of file DPW.h.

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

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 49 of file DPW.h.

```
00049                                                                    {
00050             return DSG::Pow<2>(value);
00051         }
```

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 54 of file DPW.h.

```
00054                                                                    {
00055             return DSG::Pow<3>(value)-value;
00056         }
```

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 59 of file DPW.h.

```
00059                                                                    {
00060             return DSG::Pow<2>(value) * (DSG::Pow<2>(value) - 2.0);
00061         }
```

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 64 of file DPW.h.

```
00064                                                                    {
00065             return DSG::Pow<5>(value) - DSG::Pow<3>(value) * 10.0/3.0 + value * 7.0/3.0;
00066         }
```

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 69 of file DPW.h.

```
00069                                                                    {
00070             return DSG::Pow<6>(value) - 5.0 * DSG::Pow<4>(value) + 7.0 *
      DPW_Polynomial<2>(value);
00071         }
```

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 35 of file Gaussian.h.

```
                                    {
            static decimal normalizer=1;//variable used to actively normalize the output
            //to enforce compatability with DSG::LUT a dummy parameter is applied
            //this parameter is useless except for compatability reasons
            decimal R1 = DSG::Noise::White();
            decimal R2 = DSG::Noise::White();
            decimal x= (decimal)sqrt(-2.0f * log(R1))*DSG::Cos(R2);
            if (DSG::Abs(x)>normalizer) {
                //store highest output
                normalizer=DSG::Abs(x);
            }
            x/=normalizer;//normalize
            return x;
        }
```

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 35 of file Pink.h.

```
                        {
            //routine: Get white or gaussian, filter, return
            static decimal b0,b1,b2,b3,b4,b5,b6;
            static decimal normalizer=1;//variable used to actively normalize the output
            static DSG::DCBlocker _block;
            decimal white = DSG::Noise::Gaussian();
            decimal pink;
            //pinking filter
            b0 = 0.99886 * b0 + white * 0.0555179;
            b1 = 0.99332 * b1 + white * 0.0750759;
            b2 = 0.96900 * b2 + white * 0.1538520;
            b3 = 0.86650 * b3 + white * 0.3104856;
            b4 = 0.55000 * b4 + white * 0.5329522;
            b5 = -0.7616 * b5 - white * 0.0168980;
            pink = b0 + b1 + b2 + b3 + b4 + b5 + b6 + white * 0.5362;
            b6 = white * 0.115926;
            if (DSG::Abs(pink)>normalizer) {
                //store highest output
                normalizer=DSG::Abs(pink);
            }
            pink/=normalizer;
            _block.Perform(pink);
            return pink;
        }
```

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 50 of file Random.h.

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

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 35 of file White.h.

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

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 35 of file Window.h.

```
                {
            decimal step = range/(decimal)lut.Size();
            decimal phs=0;
            for (int i=0; i<lut.Size(); ++i) {
                lut[i]*=windowFunction(phs);
                phs+=step;
            }
        }
```

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 45 of file Window.h.

```
         {
            decimal step = range/(decimal)lut.Size();
            decimal phs=0;
            for (int i=0; i<lut.Size(); ++i) {
                lut[i]*=windowFunction(phs);
                phs+=step;
            }
        }
```

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

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

Definition at line 36 of file Blackman.h.

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

Chapter 7

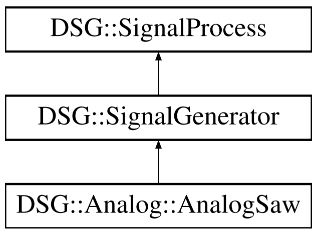
Class Documentation

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 34 of file AnalogSaw.h.

7.1.2 Constructor & Destructor Documentation

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

Definition at line 25 of file AnalogSaw.cpp.

```
00025 :DSG::SignalGenerator(){}
```

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

Definition at line 26 of file AnalogSaw.cpp.

```
00026 :DSG::SignalGenerator(frequency,offset){}
```

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

Definition at line 27 of file AnalogSaw.cpp.

```
00027 {}
```

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 44 of file AnalogSaw.h.

```
00044                                                                   {
00045             _stor=_phasor;
00046             _stor+=0.5;
00047             if (_stor>1.0) {
00048                 --_stor;
00049             }
00050             _stor-=0.5;
00051             _stor*=2.0;
00052             signal=_stor;
00053             step();
00054             return true;
00055         }
```

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

Reimplemented from DSG::SignalGenerator.

Definition at line 56 of file AnalogSaw.h.

```
00056                                                                    {
00057             signal.Flush();
00058             while (!signal.Full()) {
00059                 if (Perform(_storage)) {
00060                     if(signal.Write(_storage)){
00061                     }else return false;
00062                 }else return false;
00063             }return true;
00064         }
```

7.1.4 Member Data Documentation

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

Definition at line 42 of file AnalogSaw.h.

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

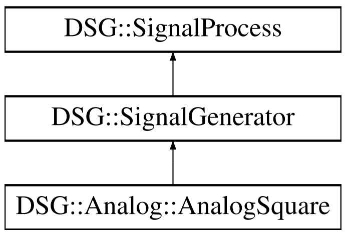
- AnalogSaw.h
- AnalogSaw.cpp

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 34 of file AnalogSquare.h.

7.2.2 Constructor & Destructor Documentation

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

Definition at line 25 of file AnalogSquare.cpp.

```
00025 :DSG::SignalGenerator(){}
```

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

Definition at line 26 of file AnalogSquare.cpp.

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

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

Definition at line 27 of file AnalogSquare.cpp.

```
{}
```

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 42 of file AnalogSquare.h.

```
                                                                       {
            signal=_phasor < 0.5 ? 1.0:-1.0;
            step();
            return true;
        }
```

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

Reimplemented from DSG::SignalGenerator.

Definition at line 47 of file AnalogSquare.h.

```
                                                                        {
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
```

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

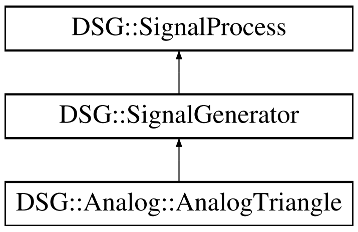
- AnalogSquare.h
- AnalogSquare.cpp

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 34 of file AnalogTriangle.h.

7.3.2 Constructor & Destructor Documentation

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

Definition at line 25 of file AnalogTriangle.cpp.

```
00025 :DSG::SignalGenerator(){}
```

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

Definition at line 26 of file AnalogTriangle.cpp.

```
00026 :DSG::SignalGenerator(frequency,offset){}
```

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

Definition at line 27 of file AnalogTriangle.cpp.

```
00027 {}
```

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 44 of file AnalogTriangle.h.

```
                                                                           {
            _stor = _phasor;
            _stor+=0.25;
            while (_stor>1.0) {
                _stor-=1.0;
            }
            _stor-=0.5;
            if (_stor<0) {
                _stor*=-1.0;
            }
            _stor-=0.25;
            _stor*=-4.0;
            signal = _stor;
            step();//always last
            return true;
        }
```

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

Reimplemented from DSG::SignalGenerator.

Definition at line 60 of file AnalogTriangle.h.

```
                                                                            {
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
```

7.3.4 Member Data Documentation

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

Definition at line 42 of file AnalogTriangle.h.

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

- AnalogTriangle.h
- 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 30 of file AudioSettings.h.

7.4.2 Member Function Documentation

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

Definition at line 35 of file AudioSettings.cpp.

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

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

Definition at line 27 of file AudioSettings.cpp.

```
00027                                                  {
00028     return _sampleRate;
00029 }
```

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

Definition at line 30 of file AudioSettings.cpp.

```
00030                                                                               {
00031     _sampleRate = value;
00032     _nyquist = _sampleRate*0.5;
00033     return _sampleRate;
00034 }
```

7.4.3 Member Data Documentation

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

Definition at line 37 of file AudioSettings.h.

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

Definition at line 36 of file AudioSettings.h.

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

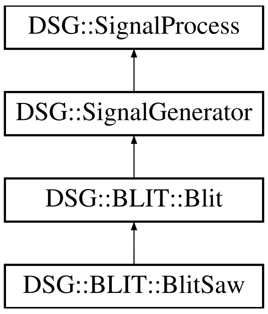
- AudioSettings.h
- AudioSettings.cpp

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 39 of file BLIT.h.

7.5.2 Constructor & Destructor Documentation

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

Definition at line 25 of file BLIT.cpp.

```
00025                     :DSG::SignalGenerator(){
00026     Frequency(0);
00027 }
```

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

Definition at line 28 of file BLIT.cpp.

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

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

Definition at line 31 of file BLIT.cpp.

```
{}
```

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 78 of file BLIT.h.

```
                                                                                       {
            this->SignalGenerator::Frequency(value);
            p_ = DSG::SampleRate()/_frequency;
            _h = (unsigned)floor(p_*0.5);
            m_ = 2 * (_h)+1;
            a_ = m_/(double)p_;
            return _frequency;
        }
```

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 55 of file BLIT.h.

```
                                                   {
            //found better results in this case with built in sine function. not performance wise but
      algorithmically
            denominator = m_ * sin(PI*_phasor);
            if (DSG::IsDenormal(denominator)) {
                signal = a_;
            }else{
                value = sin(PI*_phasor * m_);
                value/=denominator;
                value*=a_;
                signal = value;
            }
            step();
            return true;
        }
```

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 69 of file BLIT.h.

```
                                                           {
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
```

7.5.4 Member Data Documentation

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

Definition at line 50 of file BLIT.h.

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

Definition at line 51 of file BLIT.h.

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

Definition at line 52 of file BLIT.h.

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

Definition at line 49 of file BLIT.h.

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

Definition at line 48 of file BLIT.h.

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

Definition at line 53 of file BLIT.h.

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

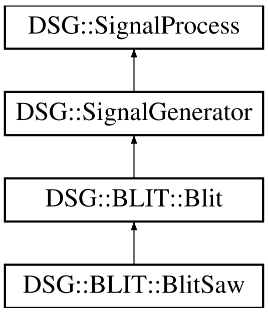
- BLIT.h
- BLIT.cpp

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 34 of file BLITSaw.h.

7.6.2 Constructor & Destructor Documentation

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

Definition at line 25 of file BLITSaw.cpp.

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

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

Definition at line 28 of file BLITSaw.cpp.

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

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

Definition at line 31 of file BLITSaw.cpp.

```
{}
```

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 72 of file BLITSaw.h.

```
                                                                                           {
              this->SignalGenerator::Frequency(value);
              p_ = DSG::SampleRate()/_frequency;
              _h = (unsigned)floor(p_*0.5);
              m_ = 2 * (_h)+1;
              a_ = m_/(double)p_;
              C2_ = 1.0/(double)p_;
              return _frequency;
          }
```

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

Reimplemented from DSG::BLIT::Blit.

Definition at line 46 of file BLITSaw.h.

```
                                                                {
              denominator = m_ * sin(PI*_phasor);
              if (DSG::IsDenormal(denominator)) {
                  signal = a_;
              }else{
                  value = sin(PI*_phasor * m_);
                  value/=denominator;
                  value*=a_;
                  signal = value;
              }
              step();
              signal += (Register_ - C2_);
              Register_ = signal * 0.995;
              C2_+=signal;
              C2_*=0.5;
              return true;
          }
```

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

Reimplemented from DSG::BLIT::Blit.

Definition at line 63 of file BLITSaw.h.

```
                                                                 {
              signal.Flush();
              while (!signal.Full()) {
                  if (Perform(_storage)) {
                      if(signal.Write(_storage)){
                      }else return false;
                  }else return false;
              }return true;
          }
```

7.6.4 Member Data Documentation

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

Definition at line 43 of file BLITSaw.h.

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

Definition at line 44 of file BLITSaw.h.

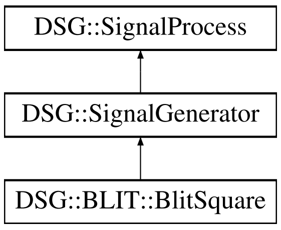
The documentation for this class was generated from the following files:

- BLITSaw.h
- 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 33 of file BLITSquare.h.

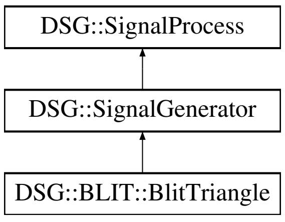
The documentation for this class was generated from the following file:

- 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 34 of file BLITTriangle.h.

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

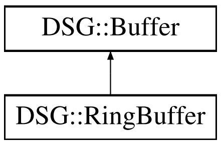
- BLITTriangle.h

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 34 of file Buffer.h.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 DSG::Buffer::Buffer ( )

Definition at line 25 of file Buffer.cpp.

```
00025 :_size(0),_buffer(nullptr){}
```

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

Definition at line 26 of file Buffer.cpp.

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

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

Definition at line 27 of file Buffer.cpp.

```
                                    {
    _buffer = new  DSG::DSGSample[_size];
    _size = other._size;
    *this = other;
}
```

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

Definition at line 45 of file Buffer.cpp.

```
                 {
    if (_buffer!=nullptr) {
        delete [] _buffer;
    }
}
```

7.9.3 Member Function Documentation

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

Definition at line 32 of file Buffer.cpp.

```
                                                 {
    if (_size!=other._size) {
        if (_buffer!=nullptr) {
            delete [] _buffer;
        }
        _size = other._size;
        _buffer = new  DSG::DSGSample[_size];
    }
    for (int i=0; i<_size; ++i) {
        _buffer[i] = other._buffer[i];
    }
    return *this;
}
```

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

Definition at line 50 of file Buffer.cpp.

```
                                                     {
#ifdef DEBUG
    assert(index<_size);
#endif
    return _buffer[index];
}
```

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

Definition at line 47 of file Buffer.h.

```
                                             {
        return _size;
    }
```

7.9.4 Member Data Documentation

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

Definition at line 44 of file Buffer.h.

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

Definition at line 45 of file Buffer.h.

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

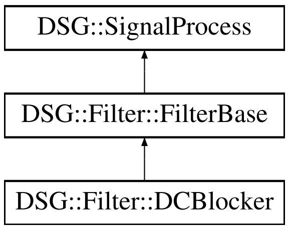
- Buffer.h
- Buffer.cpp

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 33 of file DCBlocker.h.

7.10.2 Constructor & Destructor Documentation

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

Definition at line 25 of file DCBlocker.cpp.

```
00025 :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 26 of file DCBlocker.cpp.

```
00026 {}
```

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 47 of file DCBlocker.h.

```
00047                                                                  {
00048             x = signal;
00049             signal= x - xm1+ (_a * ym1);
00050             xm1 = x;
00051             ym1=signal;
00052             return true;
00053         }
```

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

Reimplemented from DSG::Filter::FilterBase.

Definition at line 54 of file DCBlocker.h.

```
00054                                                                   {
00055             if (!signal.Empty()) {
00056                 count = signal.Count();
00057                 while (count-- > 0) {
00058                     if(signal.Read(_temp)){
00059                         if (Perform(_temp)) {
00060                             signal.Write(_temp);
00061                         }else return false;
00062                     }else return false;
00063                 }return true;
00064             }else return false;
00065         }
```

7.10.4 Member Data Documentation

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

Definition at line 45 of file DCBlocker.h.

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

Definition at line 41 of file DCBlocker.h.

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

Definition at line 40 of file DCBlocker.h.

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

Definition at line 44 of file DCBlocker.h.

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

Definition at line 42 of file DCBlocker.h.

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

Definition at line 43 of file DCBlocker.h.

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

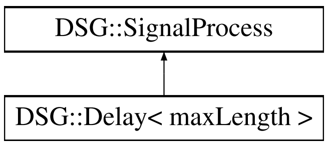
- DCBlocker.h
- DCBlocker.cpp

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 33 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 35 of file Delay.h.

```
00035                 :DSG::SignalProcess(),_max(maxLength),_swap(0),
      _temp(0),count(0),_index(0),_delay(0){
00036             for (int i=0; i<_max; ++i) {
00037                 _buffer[i]=0;
00038             }
00039         }
```

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

Definition at line 40 of file Delay.h.

```
00040                                     :DSG::SignalProcess(),_max(maxLength),
      _swap(0),_temp(0),count(0),_index(0),_delay(0){
00041             for (int i=0; i<_max; ++i) {
00042                 _buffer[i]=0;
00043             }
00044             if (samples>maxLength) {
00045                 _delay = maxLength;
00046             }else{
00047                 _delay = samples;
00048             }
00049         }
```

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

Definition at line 50 of file Delay.h.

```
00050 {}
```

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 72 of file Delay.h.

```
                                        {
            ++_index;
            if (_index>_delay) {
                _index-=_delay;
            }
        }
```

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

Definition at line 51 of file Delay.h.

```
                                                        {
            return _delay;
        }
```

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 54 of file Delay.h.

```
                                                                            {
            if (samples>maxLength) {
                _delay = maxLength;
            }else{
                _delay = samples;
            }
            return _delay;
        }
```

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

Implements DSG::SignalProcess.

Definition at line 80 of file Delay.h.

```
                                                                {
        _swap = _buffer[_index-1];
        _buffer[_index-1]=signal;
        signal = _swap;
        increment();
        return true;
    }
```

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

Implements DSG::SignalProcess.

Definition at line 88 of file Delay.h.

```
                                                                   {
        if (!signal.Empty()) {
            count = signal.Count();
            while (count-- > 0) {
                if(signal.Read(_temp)){
                    if (Perform(_temp)) {
                        signal.Write(_temp);
                    }else return false;
                }else return false;
            }return true;
        }else return false;
    }
```

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 69 of file Delay.h.

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

Definition at line 66 of file Delay.h.

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

Definition at line 67 of file Delay.h.

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

Definition at line 68 of file Delay.h.

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

Definition at line 70 of file Delay.h.

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

Definition at line 71 of file Delay.h.

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

Definition at line 65 of file Delay.h.

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

- 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 79 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 81 of file DPW.h.

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

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

- 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 87 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 89 of file DPW.h.

```
                                                                                           {
                return signal;
            }
```

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

- 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 95 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 97 of file DPW.h.

```
                                                                                           {
                output = (signal - _delay)/(4.0 * dt);
                _delay = signal;
                return output;
            }
```

7.14.3 Member Data Documentation

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

Definition at line 104 of file DPW.h.

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

Definition at line 103 of file DPW.h.

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

- 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 108 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 110 of file DPW.h.

```
                                                                                   {
                output  = (signal - _delay[0]);
                output -= (_delay[0] - _delay[1]);
                output /= (24.*DSG::Pow<2>(dt));
                _delay[1]=_delay[0];
                _delay[0]=signal;
                return output;
            }
```

7.15.3 Member Data Documentation

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

Definition at line 120 of file DPW.h.

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

Definition at line 119 of file DPW.h.

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

- 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 124 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 126 of file DPW.h.

```
                                                                                                     {
                output  = (signal - _delay[0]);
                output -= (_delay[0] - _delay[1]);
                output -= (_delay[1] - _delay[2]);
                output /= 144*DSG::Pow<3>(dt);
                _delay[2]=_delay[1];
                _delay[1]=_delay[0];
                _delay[0]=signal;
                return output;
            }
```

7.16.3 Member Data Documentation

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

Definition at line 138 of file DPW.h.

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

Definition at line 137 of file DPW.h.

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

- 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 142 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 144 of file DPW.h.

```
                                                                                                 {
            output  = (signal - _delay[0]);
            output -= (_delay[0] - _delay[1]);
            output -= (_delay[1] - _delay[2]);
            output -= (_delay[2] - _delay[3]);
            output /= 960*DSG::Pow<4>(dt);
            _delay[3]=_delay[2];
            _delay[2]=_delay[1];
            _delay[1]=_delay[0];
            _delay[0]=signal;
            return output;
        }
```

7.17.3 Member Data Documentation

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

Definition at line 158 of file DPW.h.

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

Definition at line 157 of file DPW.h.

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

- 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 162 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 164 of file DPW.h.

```
                                                                                                          {
                output  = (signal - _delay[0]);
                output -= (_delay[0] - _delay[1]);
                output -= (_delay[1] - _delay[2]);
                output -= (_delay[2] - _delay[3]);
                output -= (_delay[3] - _delay[4]);
                output /= 7200*DSG::Pow<5>(dt);
                _delay[4]=_delay[3];
                _delay[3]=_delay[2];
                _delay[2]=_delay[1];
                _delay[1]=_delay[0];
                _delay[0]=signal;
                return output;
            }
```

7.18.3 Member Data Documentation

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

Definition at line 180 of file DPW.h.

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

Definition at line 179 of file DPW.h.

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

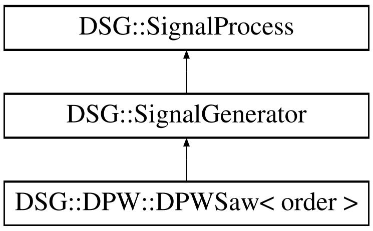
- DPW.h

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 34 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 36 of file DPWSaw.h.

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

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

Definition at line 39 of file DPWSaw.h.

```
00039 :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 40 of file DPWSaw.h.

```
00040 {}
```

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 41 of file DPWSaw.h.

```
00041                                                              {
00042                 //trivial saw ramping from -1 to 1
00043                 _register = _phasor;
00044                 _register-=0.5;
00045                 _register*=2.0;
00046                 /*-------------------------*/
00047                 //DPW algorithm
00048                 //polynomial shaping
00049                 _register=DSG::DPW::DPW_Polynomial<order>(_register);
00050                 //differentiating
00051                 signal = _diff(_register,_dt);
00052                 /*-------------------------*/
00053                 //signal = DSG::EnforceBounds<-1, 1>(signal);
00054                 //advance phase
00055                 step();
00056                 return true;
00057             }
```

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 58 of file DPWSaw.h.

```
00058                                                               {
00059                 signal.Flush();
00060                 while (!signal.Full()) {
00061                     if (Perform(_storage)) {
00062                         if(signal.Write(_storage)){
00063                         }else return false;
00064                     }else return false;
00065                 }return true;
00066             }
```

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 69 of file DPWSaw.h.

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

Definition at line 68 of file DPWSaw.h.

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

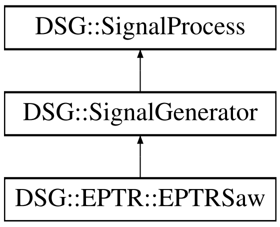
- DPWSaw.h

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 35 of file EPTRSaw.h.

7.20.2 Constructor & Destructor Documentation

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

Definition at line 25 of file EPTRSaw.cpp.

```
00025 :DSG::SignalGenerator(){}
```

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

Definition at line 26 of file EPTRSaw.cpp.

```
00026 :DSG::SignalGenerator(frequency,offset){}
```

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

Definition at line 27 of file EPTRSaw.cpp.

```
00027 {}
```

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 45 of file EPTRSaw.h.

```
00045                                                                 {
00046 #ifdef __APPLE__
00047 #warning Untested For Aliasing DSG::EPTR::EPTRSaw::Perform()
00048 #endif
00049             //generate trivial saw
00050             _register = _phasor;
00051             _register+=0.5;
00052             if (_register>1.0) {
00053                 --_register;
00054             }
00055             _register-=0.5;
00056             _register*=2.0;
00057             if (_register > 1.0-_dt) {
00058                 //transition region detected
00059                 //apply eptr correction
00060                 signal = _register - (_register/_dt) + (1.0/
      _dt) -1;
00061             }else{
00062                 signal = _register;
00063             }
00064             step();//avance phase
00065             return true;
00066         }
```

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

Reimplemented from DSG::SignalGenerator.

Definition at line 67 of file EPTRSaw.h.

```
00067                                                                  {
00068             signal.Flush();
00069             while (!signal.Full()) {
00070                 if (Perform(_storage)) {
00071                     if(signal.Write(_storage)){
00072                     }else return false;
00073                 }else return false;
00074             }return true;
00075         }
```

7.20.4 Member Data Documentation

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

Definition at line 43 of file EPTRSaw.h.

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

- EPTRSaw.h
- 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 36 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 37 of file DSGMath.h.

```
00037 {value = N * Factorial<N-1>::value};
```

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

- 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 41 of file DSGMath.h.

7.22.2 Member Enumeration Documentation

7.22.2.1 anonymous enum

Enumerator

value

Definition at line 42 of file DSGMath.h.

```
00042 { value = 1 };
```

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

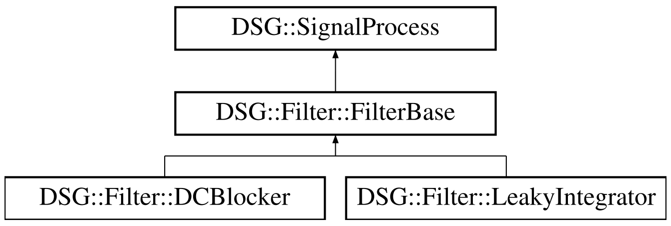
- DSGMath.h

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 34 of file Filter.h.

7.23.2 Constructor & Destructor Documentation

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

Definition at line 25 of file Filter.cpp.

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

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

Definition at line 26 of file Filter.cpp.

```
{}
```

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 60 of file Filter.h.

```
                                                                               {
                return false;
            }
```

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 45 of file Filter.h.

```
                                                                       {
                return true;
            }
```

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 48 of file Filter.h.

```
                                                                        {
                if (!signal.Empty()) {
                    count = signal.Count();
                    while (count-- > 0) {
                        if(signal.Read(_temp)){
                            if (Perform(_temp)) {
                                signal.Write(_temp);
                            }else return false;
                        }else return false;
                    }return true;
                }else return false;
            }
```

7.23.4 Member Data Documentation

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

Definition at line 42 of file Filter.h.

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

Definition at line 43 of file Filter.h.

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

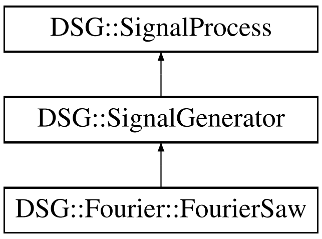
- Filter.h
- Filter.cpp

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 34 of file FourierSaw.h.

7.24.2 Constructor & Destructor Documentation

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

Definition at line 25 of file FourierSaw.cpp.

```
00025 :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 26 of file FourierSaw.cpp.

```
00026                                                                                            :
      DSG::SignalGenerator(frequency,offset),_a(1.7/PI),phs(0),
      value(0),i(0){
00027     _h = MaxHarms(_frequency)+1;
00028 }
```

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

Definition at line 29 of file FourierSaw.cpp.

```
00029 {}
```

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 69 of file FourierSaw.h.

```
00069                                                                                                    {
00070             _frequency = value;
00071             _dt = _frequency/DSG::SampleRate();
00072             _h = MaxHarms(_frequency);
00073             return _frequency;
00074         }
```

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

Reimplemented from DSG::SignalGenerator.

Definition at line 49 of file FourierSaw.h.

```
00049                                                                         {
00050             //_h Sine Calls Per Sample where _h  is theoretically nyquist / frequency
00051             value=DSG::Sin(_phasor);
00052             for (i=2; i<_h; ++i) {
00053                 value += (1.0/i) * DSG::Sin(_phasor*i);
00054             }
00055             value*=_a;
00056             signal = value;
00057             step();
00058             return true;
00059         }
```

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

Reimplemented from DSG::SignalGenerator.

Definition at line 60 of file FourierSaw.h.

```
                                                                                {
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
```

7.24.4 Member Data Documentation

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

Definition at line 44 of file FourierSaw.h.

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

Definition at line 43 of file FourierSaw.h.

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

Definition at line 47 of file FourierSaw.h.

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

Definition at line 45 of file FourierSaw.h.

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

Definition at line 46 of file FourierSaw.h.

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

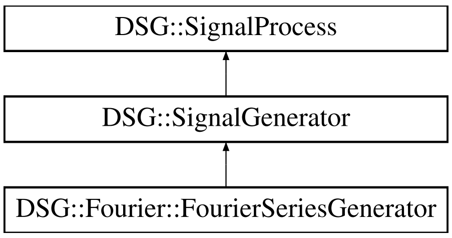
- FourierSaw.h
- FourierSaw.cpp

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 48 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 50 of file FourierSeries.h.

7.25.3 Constructor & Destructor Documentation

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

Definition at line 45 of file FourierSeries.cpp.

```
00045 :DSG::SignalGenerator(){}
```

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

Definition at line 46 of file FourierSeries.cpp.

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

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

Definition at line 47 of file FourierSeries.cpp.

```
{}
```

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 62 of file FourierSeries.h.

```
                                                                                {
            value = _phasor;
            signal=0;
            for (auto i = _series.begin(); i!=_series.end(); ++i) {
                signal += DSG::Sin(_phasor * i->Ratio())*i->Amplitude();
            }
            step();
            return true;
        }
```

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

Reimplemented from DSG::SignalGenerator.

Definition at line 71 of file FourierSeries.h.

```
                                                                                 {
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
```

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

Definition at line 80 of file FourierSeries.h.

```
            {
            _series = series;
        }
```

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

Definition at line 83 of file FourierSeries.h.

```
                                                                                    {
                return _series;
        }
```

7.25.5 Member Data Documentation

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

Definition at line 59 of file FourierSeries.h.

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

Definition at line 60 of file FourierSeries.h.

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

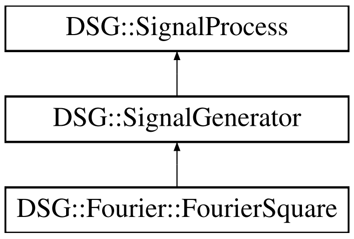
- FourierSeries.h
- FourierSeries.cpp

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 34 of file FourierSquare.h.

7.26.2 Constructor & Destructor Documentation

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

Definition at line 25 of file FourierSquare.cpp.

```
: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 26 of file FourierSquare.cpp.

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

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

Definition at line 29 of file FourierSquare.cpp.

```
{}
```

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 69 of file FourierSquare.h.

```
                                                                                              {
            _frequency = value;
            _dt = _frequency/DSG::SampleRate();
            _h = MaxHarms(_frequency);
            return _frequency;
        }
```

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

Reimplemented from DSG::SignalGenerator.

Definition at line 49 of file FourierSquare.h.

```
                                                                          {
            //(_h/2)+1 Sine Calls Per Sample
            value=DSG::Sin(_phasor);//i=1
            for (i=3; i<_h; i+=2) {//i=3..5..7..
                value += (1.0/i) * DSG::Sin(_phasor*i);
            }
            value*=_a;
            signal = value;
            step();
            return true;
        }
```

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

Reimplemented from DSG::SignalGenerator.

Definition at line 60 of file FourierSquare.h.

```
                                                                           {
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
```

7.26.4 Member Data Documentation

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

Definition at line 44 of file FourierSquare.h.

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

Definition at line 43 of file FourierSquare.h.

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

Definition at line 47 of file FourierSquare.h.

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

Definition at line 45 of file FourierSquare.h.

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

Definition at line 46 of file FourierSquare.h.

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

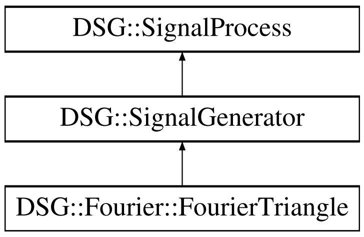
- FourierSquare.h
- FourierSquare.cpp

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 34 of file FourierTriangle.h.

7.27.2 Constructor & Destructor Documentation

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

Definition at line 25 of file FourierTriangle.cpp.

```
00025 :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 26 of file FourierTriangle.cpp.

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

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

Definition at line 29 of file FourierTriangle.cpp.

```
{}
```

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 71 of file FourierTriangle.h.

```
                                                                                            {
                _frequency = value;
                _dt = _frequency/DSG::SampleRate();
                _h = MaxHarms(_frequency);
                return _frequency;
            }
```

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

Reimplemented from DSG::SignalGenerator.

Definition at line 49 of file FourierTriangle.h.

```
                                                                            {
                //(_h/2)+1 Sine Calls Per Sample
                value=DSG::Sin(_phasor);//i=1
                double sgn = -1;
                for (i=3; i<_h; i+=2) {//i=3..5..7..
                    value += sgn * (1.0/(i*i)) * DSG::Sin(_phasor*
     i);
                    sgn*=-1;
                }
                value*=_a;
                signal = value;
                step();
                return true;
            }
```

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

Reimplemented from DSG::SignalGenerator.

Definition at line 62 of file FourierTriangle.h.

```
                                                                                {
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
```

7.27.4 Member Data Documentation

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

Definition at line 44 of file FourierTriangle.h.

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

Definition at line 43 of file FourierTriangle.h.

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

Definition at line 47 of file FourierTriangle.h.

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

Definition at line 45 of file FourierTriangle.h.

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

Definition at line 46 of file FourierTriangle.h.

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

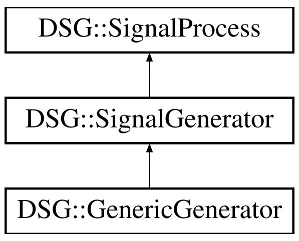
- FourierTriangle.h
- FourierTriangle.cpp

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 29 of file GenericGenerator.h.

7.28.2 Constructor & Destructor Documentation

7.28.2.1 DSG::GenericGenerator::GenericGenerator ( )

Definition at line 25 of file GenericGenerator.cpp.

```
00025 :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 26 of file GenericGenerator.cpp.

```
00026 :DSG::SignalGenerator(frequency,offset),_callback(signalFunction){}
```

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

Definition at line 27 of file GenericGenerator.cpp.

```
00027 {}
```

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 39 of file GenericGenerator.h.

```
00039                                                                  {
00040         if (_callback!=nullptr) {
00041             signal = _callback(_phasor);
00042         }else signal = 0;
00043         step();
00044         return true;
00045     }
```

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

Reimplemented from DSG::SignalGenerator.

Definition at line 46 of file GenericGenerator.h.

```
                                                                        {
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
```

7.28.4 Member Data Documentation

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

Definition at line 37 of file GenericGenerator.h.

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

- GenericGenerator.h
- 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 34 of file FourierSeries.h.

7.29.2 Constructor & Destructor Documentation

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

Definition at line 25 of file FourierSeries.cpp.

```
00025 :_ratio(0),_amplitude(0){}
```

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

Definition at line 26 of file FourierSeries.cpp.

```
00026 :_ratio(ratio),_amplitude(amplitude){}
```

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

Definition at line 27 of file FourierSeries.cpp.

```
00027                             {
00028     _ratio=0;
00029     _amplitude=0;
00030 }
```

7.29.3 Member Function Documentation

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

Definition at line 38 of file FourierSeries.cpp.

```
00038                                                  {
00039     return _amplitude;
00040 }
```

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

Definition at line 41 of file FourierSeries.cpp.

```
00041                                                                          {
00042     _amplitude=value;
00043     return _amplitude;
00044 }
```

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

Definition at line 31 of file FourierSeries.cpp.

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

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

Definition at line 34 of file FourierSeries.cpp.

```
00034                                                                      {
00035     _ratio = value;
00036     return _ratio;
00037 }
```

7.29.4 Member Data Documentation

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

Definition at line 45 of file FourierSeries.h.

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

Definition at line 44 of file FourierSeries.h.

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

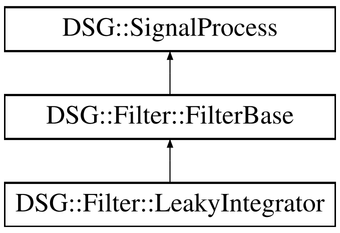
- FourierSeries.h
- FourierSeries.cpp

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 34 of file Leaky.h.

7.30.2 Constructor & Destructor Documentation

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

Definition at line 25 of file Leaky.cpp.

```
                                          :DSG::Filter::FilterBase(){
    x1=0;
    y1=0;
    a=0;
    b=0;
    y=0;
}
```

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

Definition at line 32 of file Leaky.cpp.

```
                                                                             :
     DSG::Filter::FilterBase(){
    x1=0;
    y1=0;
    a=0;
    b=0;
    y=0;
    Cutoff(cutoff);
}
```

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

Definition at line 40 of file Leaky.cpp.

```
                                           {
    x1=0;
    y1=0;
    a=0;
    b=0;
    y=0;
}
```

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 65 of file Leaky.h.

```
                                                                                   {
            double Omega;
            x1 = y1 = 0.0;
            Omega = atan(PI * cutoff);
            a = -(1.0 - Omega) / (1.0 + Omega);
            b = (1.0 - b) / 2.0;
            return true;
        }
```

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

Reimplemented from DSG::Filter::FilterBase.

Definition at line 46 of file Leaky.h.

```
                                                                          {
            y = b * (signal + x1) - a * y1;
            x1=signal;
            y1=y;
            signal=y;
            return true;
        }
```

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

Reimplemented from DSG::Filter::FilterBase.

Definition at line 53 of file Leaky.h.

```
                                                                           {
            if (!signal.Empty()) {
                count = signal.Count();
                while (count-- > 0) {
                    if(signal.Read(_temp)){
                        if (Perform(_temp)) {
                            signal.Write(_temp);
                        }else return false;
                    }else return false;
                }return true;
            }else return false;
        }
```

7.30.4 Member Data Documentation

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

Definition at line 43 of file Leaky.h.

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

Definition at line 43 of file Leaky.h.

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

Definition at line 43 of file Leaky.h.

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

Definition at line 44 of file Leaky.h.

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

Definition at line 43 of file Leaky.h.

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

- Leaky.h
- 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 33 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 35 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 36 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 37 of file LUT.h.

```
00037 :_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 38 of file LUT.h.

```
00038                                                          :_size(size){
00039             //range is the expected input range for the function
00040             //example would  be 0-2pi or 0-1
00041             //would be provided a 2pi or 1
00042             //defaults to 1
00043             double step = range/(double)_size;
00044             phs = 0;
00045             for (int i=0; i<_size; ++i) {
00046                 _table[i] = fill(phs);
00047                 phs+=step;
00048             }
00049         }
```

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 50 of file LUT.h.

```
00050                                                                  :_size(size){
00051             //range is the expected input range for the function
00052             //example would  be 0-2pi or 0-1
00053             //would be provided a 2pi or 1
00054             //defaults to 1
00055             double step = range/_size;
00056             phs = 0;
00057             for (int i=0; i<_size; ++i) {
00058                 _table[i] = fill(phs);
00059                 phs+=step;
00060             }
00061         }
```

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

Definition at line 62 of file LUT.h.

```
00062 {}
```

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 75 of file LUT.h.

```
00075                                                             {
00076             phs=x;
00077             //need range checking on x to ensure 0-1 range
00078             phs<0 ? phs = 1-(phs*-1):0;
00079             phs-=((int)phs);
00080             return this->_table[(unsigned)(phs*(this->_size-1))];
00081         }
```

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 63 of file LUT.h.

```
                                                                        {
#ifdef DEBUG
            assert(index<_size);
#endif
            return _table[index];
        }
```

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

Definition at line 69 of file LUT.h.

```
                                                             {
#ifdef DEBUG
            assert(index<_size);
#endif
            return _table[index];
        }
```

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

Definition at line 82 of file LUT.h.

```
                                              {
            return _size;
        }
```

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 87 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 86 of file LUT.h.

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

Definition at line 88 of file LUT.h.

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

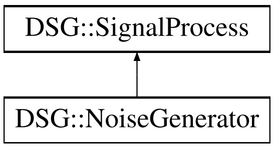
- LUT.h

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 29 of file NoiseGenerator.h.

7.32.2 Constructor & Destructor Documentation

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

Definition at line 25 of file NoiseGenerator.cpp.

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

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

Definition at line 28 of file NoiseGenerator.cpp.

```
{}
```

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 39 of file NoiseGenerator.h.

```
                                                          {
        signal = _function(0);
        return true;
    }
```

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

Implements DSG::SignalProcess.

Definition at line 43 of file NoiseGenerator.h.

```
                                                                  {
        signal.Flush();
        while (!signal.Full()) {
            if (Perform(_storage)) {
                if(signal.Write(_storage)){
                }else return false;
            }else return false;
        }return true;
    }
```

7.32.4 Member Data Documentation

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

Definition at line 36 of file NoiseGenerator.h.

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

Definition at line 37 of file NoiseGenerator.h.

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

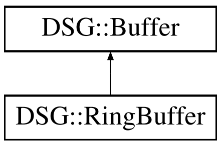
- NoiseGenerator.h
- NoiseGenerator.cpp

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 35 of file RingBuffer.h.

7.33.2 Constructor & Destructor Documentation

7.33.2.1 DSG::RingBuffer::RingBuffer ( )

Definition at line 25 of file RingBuffer.cpp.

```
00025 :Buffer(0),_read(0),_write(0),_count(0),MASK(0){}
```

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

Definition at line 26 of file RingBuffer.cpp.

```
00026                                            :Buffer(make_pow_2(size)),
      _read(0),_write(0),_count(0){
00027     MASK = this->_size-1;
00028 }
```

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

Definition at line 29 of file RingBuffer.cpp.

```
00029                                             :Buffer(buffer){
00030     _write.store(buffer._write.load(std::memory_order_acquire));
00031     _read.store(buffer._read.load(std::memory_order_acquire));
00032     _count = buffer._count;
00033     MASK = buffer._size-1;
00034 }
```

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

Definition at line 43 of file RingBuffer.cpp.

```
00043 {Flush();}
```

7.33.3 Member Function Documentation

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

Definition at line 106 of file RingBuffer.h.

```
00106                                                    {
00107         return _count;
00108     }
```

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

Definition at line 80 of file RingBuffer.h.

```
00080                                           {
00081         return _count==0;
00082     }
```

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

Definition at line 83 of file RingBuffer.h.

```
00083                                      {
00084         _write.store(0,std::memory_order_relaxed);
00085         _read.store(0,std::memory_order_relaxed);
00086         _count=0;
00087     }
```

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

Definition at line 77 of file RingBuffer.h.

```
00077                                          {
00078         return _count==this->_size;
00079     }
```

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

Definition at line 111 of file RingBuffer.h.

```
00111                                                          {
00112         return pow(2, ceil(log(number)/log(2)));
00113     }
```

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

Definition at line 110 of file RingBuffer.h.

```
00110 {return (current+1) & MASK;}
```

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

Definition at line 35 of file RingBuffer.cpp.

```
                                                           {
    Buffer::operator=(buffer);
    _write.store(buffer._write.load(std::memory_order_acquire));
    _read.store(buffer._read.load(std::memory_order_acquire));
    _count = buffer._count;
    MASK = buffer._size-1;
    return *this;
}
```

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

Definition at line 97 of file RingBuffer.h.

```
                                              {
        if (!Empty()) {
            read = _read.load(std::memory_order_acquire);
            _read.store(next(read),std::memory_order_release);
            elem = this->_buffer[read];
            --_count;
            return true;
        }else return false;
    }
```

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

Definition at line 88 of file RingBuffer.h.

```
                                                     {
        if (!Full()) {
            write = _write.load(std::memory_order_acquire);
            _write.store(next(write),std::memory_order_release);
            this->_buffer[write] = elem;
            ++_count;
            return true;
        }else return false;
    }
```

7.33.4 Friends And Related Function Documentation

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

Definition at line 60 of file RingBuffer.h.

```
                                                                          {
            return buffer.Read(signal);
        }
```

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

Definition at line 57 of file RingBuffer.h.

```
                                                                                {
            return buffer.Write(signal);
        }
```

7.33.5 Member Data Documentation

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

Definition at line 39 of file RingBuffer.h.

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

Definition at line 38 of file RingBuffer.h.

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

Definition at line 37 of file RingBuffer.h.

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

Definition at line 40 of file RingBuffer.h.

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

Definition at line 42 of file RingBuffer.h.

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

Definition at line 41 of file RingBuffer.h.

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

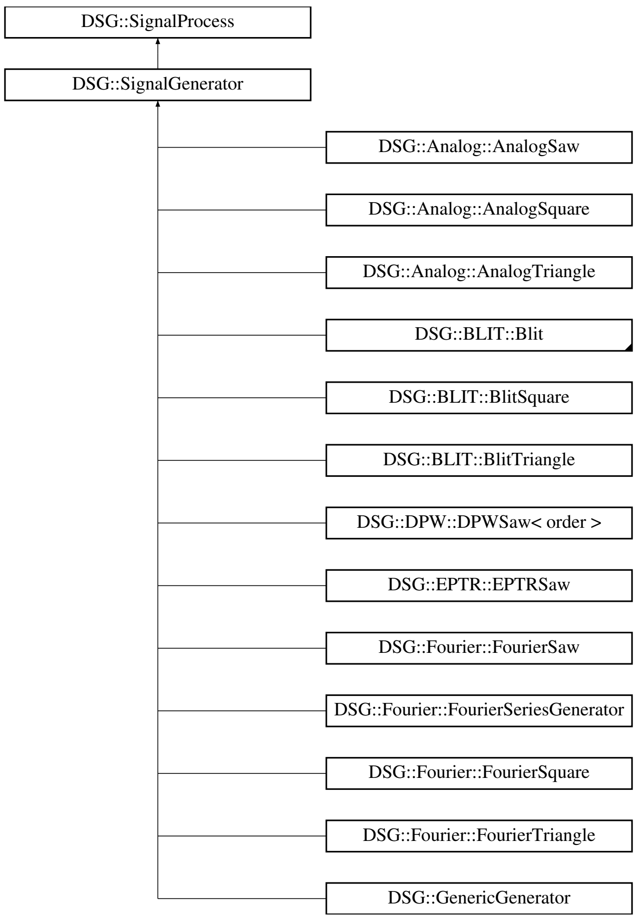
- RingBuffer.h
- RingBuffer.cpp

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 33 of file SignalGenerator.h.

7.34.2 Constructor & Destructor Documentation

7.34.2.1 DSG::SignalGenerator::SignalGenerator ( )

Definition at line 25 of file SignalGenerator.cpp.

```
00025 :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 26 of file SignalGenerator.cpp.

```
00026                                                                                                   :
      _phasor(0),_frequency(frequency),_dt(0),_offset(offset){
00027     Frequency(frequency);
00028     Phase(offset);
00029 }
```

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

Definition at line 30 of file SignalGenerator.cpp.

```
00030 {}
```

7.34.3 Member Function Documentation

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

Definition at line 70 of file SignalGenerator.h.

```
00070                                                             {
00071     return _frequency;
00072 }
```

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 73 of file SignalGenerator.h.

```
00073                                                                                      {
00074     _frequency = DSG::EnforceBounds<0, 20000,DSG::DSGSample>(value);
00075     _dt = _frequency/DSG::SampleRate();
00076     return _frequency;
00077 }
```

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 >, DS↩G::EPTR::EPTRSaw, DSG::Analog::AnalogSaw, DSG::Analog::AnalogSquare, DSG::Analog::AnalogTriangle, D↩SG::BLIT::BlitSaw, DSG::Fourier::FourierSaw, DSG::Fourier::FourierSquare, DSG::Fourier::FourierTriangle, and DSG::GenericGenerator.

Definition at line 62 of file SignalGenerator.h.

```
                                                     {
    signal=0;
    return false;
}
```

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, DS↩G::EPTR::EPTRSaw, DSG::Analog::AnalogSaw, DSG::Analog::AnalogSquare, DSG::Analog::AnalogTriangle, D↩SG::BLIT::BlitSaw, DSG::Fourier::FourierSaw, DSG::Fourier::FourierSquare, DSG::Fourier::FourierTriangle, and DSG::GenericGenerator.

Definition at line 66 of file SignalGenerator.h.

```
                                                      {
    signal.Flush();
    return false;
}
```

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

Definition at line 78 of file SignalGenerator.h.

```
                                              {
    return _offset;
}
```

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

Definition at line 81 of file SignalGenerator.h.

```
                                                                      {
    _offset-=value;
    _phasor-=_offset;
    _offset=value;
    return _offset;
}
```

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

Definition at line 87 of file SignalGenerator.h.

```
                               {
    _phasor+=_dt;
    _phasor>1.0 ? --_phasor:0;
}
```

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

Definition at line 91 of file SignalGenerator.h.

```
00091                                    {
00092     _phasor=_offset;
00093 }
```

7.34.4 Member Data Documentation

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

Definition at line 51 of file SignalGenerator.h.

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

Definition at line 50 of file SignalGenerator.h.

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

Definition at line 52 of file SignalGenerator.h.

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

Definition at line 53 of file SignalGenerator.h.

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

Definition at line 54 of file SignalGenerator.h.

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

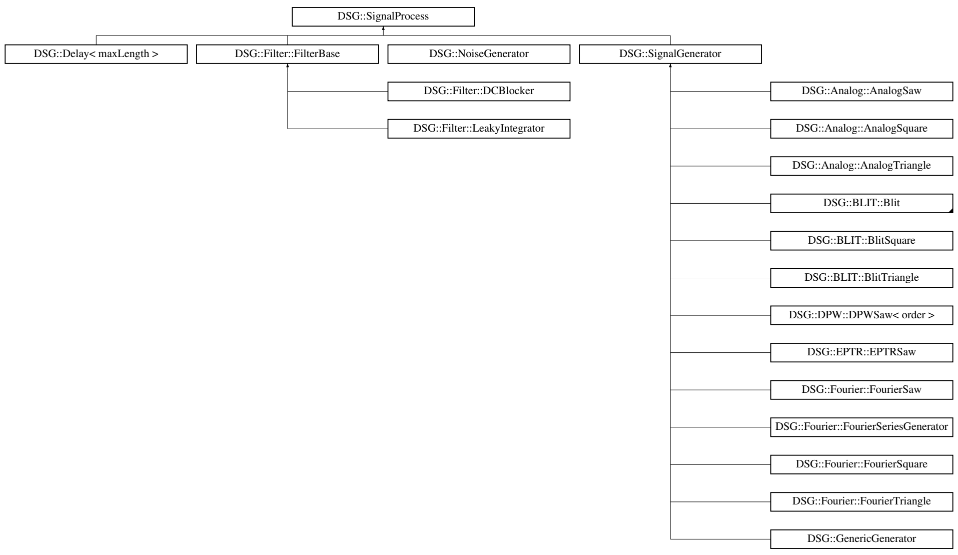
- SignalGenerator.h
- SignalGenerator.cpp

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 31 of file SignalProcess.h.

7.35.2 Constructor & Destructor Documentation

7.35.2.1 DSG::SignalProcess::SignalProcess ( )

Definition at line 25 of file SignalProcess.cpp.

```
00025 {}
```

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

Definition at line 26 of file SignalProcess.cpp.

```
00026 {}
```

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::D↩
PW::DPWSaw< order >, DSG::EPTR::EPTRSaw, DSG::Analog::AnalogSaw, DSG::Analog::AnalogSquare, D↩
SG::Analog::AnalogTriangle, DSG::BLIT::BlitSaw, DSG::Fourier::FourierSaw, DSG::Fourier::FourierSquare, DS↩
G::Fourier::FourierTriangle, DSG::Filter::LeakyIntegrator, 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::FourierSeries↩ Generator, DSG::BLIT::Blit, DSG::EPTR::EPTRSaw, DSG::Analog::AnalogSaw, DSG::Analog::AnalogSquare, D↩ SG::Analog::AnalogTriangle, DSG::BLIT::BlitSaw, DSG::Fourier::FourierSaw, DSG::Fourier::FourierSquare, DS↩ G::Fourier::FourierTriangle, DSG::Filter::LeakyIntegrator, DSG::Filter::FilterBase, DSG::SignalGenerator, DSG::↩ Filter::DCBlocker, DSG::GenericGenerator, and DSG::NoiseGenerator.

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

- SignalProcess.h
- SignalProcess.cpp

Chapter 8

File Documentation

8.1 AnalogSaw.cpp File Reference

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

8.2 AnalogSaw.cpp

```
//
//  AnalogSaw.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "AnalogSaw.h"
DSG::Analog::AnalogSaw::AnalogSaw():DSG::
SignalGenerator(){}
DSG::Analog::AnalogSaw::AnalogSaw(
DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):
DSG::SignalGenerator(frequency,offset){}
DSG::Analog::AnalogSaw::~AnalogSaw(){}
```

8.3 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

```
//
//  AnalogSaw.h
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__AnalogSaw__
#define __DSG__AnalogSaw__
#include "SignalGenerator.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //! DSG::Analog - Namespace Containing Analog Style Oscillators
    namespace Analog{
        //!\brief DSG::Analog::AnalogSaw - Analog Syle Saw Wave Generator
        class AnalogSaw : public DSG::SignalGenerator {
        public:
            AnalogSaw();
            AnalogSaw(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
            virtual ~AnalogSaw();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
        protected:
            DSG::DSGSample _stor;
        };
        inline bool DSG::Analog::AnalogSaw::Perform(
DSG::DSGSample& signal){
            _stor=_phasor;
            _stor+=0.5;
            if (_stor>1.0) {
                --_stor;
            }
            _stor-=0.5;
            _stor*=2.0;
            signal=_stor;
            step();
            return true;
        }
        inline bool DSG::Analog::AnalogSaw::Perform(
DSG::RingBuffer& signal){
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
    }
}
#endif /* defined(__DSG__AnalogSaw__) */
```

8.5 AnalogSquare.cpp File Reference

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

8.6 AnalogSquare.cpp

```
//
//  AnalogSquare.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "AnalogSquare.h"
DSG::Analog::AnalogSquare::AnalogSquare():
DSG::SignalGenerator(){}
DSG::Analog::AnalogSquare::AnalogSquare(
DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):
DSG::SignalGenerator(frequency,offset){}
DSG::Analog::AnalogSquare::~AnalogSquare(){}
```

8.7 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

```
//
//  AnalogSquare.h
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__AnalogSquare__
#define __DSG__AnalogSquare__
#include "SignalGenerator.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //! DSG::Analog - Namespace Containing Analog Style Oscillators
    namespace Analog{
        //!\brief DSG::Analog::AnalogSquare - Analog Syle Square Wave Generator
        class AnalogSquare : public DSG::SignalGenerator {
        public:
            AnalogSquare();
            AnalogSquare(DSG::DSGFrequency const& frequency, DSG::DSGPhase const& offset);
            virtual ~AnalogSquare();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
        };
        inline bool DSG::Analog::AnalogSquare::Perform(DSG::DSGSample& signal){
            signal=_phasor < 0.5 ? 1.0:-1.0;
            step();
            return true;
        }
        inline bool DSG::Analog::AnalogSquare::Perform(DSG::RingBuffer& signal){
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
    }
}
#endif /* defined(__DSG__AnalogSquare__) */
```

8.9 AnalogTriangle.cpp File Reference

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

8.10 AnalogTriangle.cpp

```
//
//  AnalogTriangle.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "AnalogTriangle.h"
DSG::Analog::AnalogTriangle::AnalogTriangle():
DSG::SignalGenerator(){}
DSG::Analog::AnalogTriangle::AnalogTriangle(
DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):
DSG::SignalGenerator(frequency,offset){}
DSG::Analog::AnalogTriangle::~AnalogTriangle(){}
```

8.11 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

```
//
//  AnalogTriangle.h
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__AnalogTriangle__
#define __DSG__AnalogTriangle__
#include "SignalGenerator.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //! DSG::Analog - Namespace Containing Analog Style Oscillators
    namespace Analog{
        //!\brief DSG::Analog::AnalogTriangle - Analog Syle Triangle Wave Generator
        class AnalogTriangle : public DSG::SignalGenerator {
        public:
            AnalogTriangle();
            AnalogTriangle(DSG::DSGFrequency const& frequency, DSG::DSGPhase const& offset);
            virtual ~AnalogTriangle();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
        protected:
            DSG::DSGSample _stor;
        };
        inline bool DSG::Analog::AnalogTriangle::Perform(DSG::DSGSample& signal){
            _stor = _phasor;
            _stor+=0.25;
            while (_stor>1.0) {
                _stor-=1.0;
            }
            _stor-=0.5;
            if (_stor<0) {
                _stor*=-1.0;
            }
            _stor-=0.25;
            _stor*=-4.0;
            signal = _stor;
            step();//always last
            return true;
        }
        inline bool DSG::Analog::AnalogTriangle::Perform(DSG::RingBuffer& signal){
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
    }
}
#endif /* defined(__DSG__AnalogTriangle__) */
```

8.13 AudioSettings.cpp File Reference

```
#include "AudioSettings.h"
```

8.14 AudioSettings.cpp

```
//
//  AudioSettings.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "AudioSettings.h"
DSG::DSGFrequency DSG::AudioSettings::_sampleRate;
DSG::DSGFrequency DSG::AudioSettings::_nyquist;
DSG::DSGFrequency const& DSG::AudioSettings::SampleRate(){
    return _sampleRate;
}
DSG::DSGFrequency const& DSG::AudioSettings::SampleRate(
DSG::DSGFrequency const& value){
    _sampleRate = value;
    _nyquist = _sampleRate*0.5;
    return _sampleRate;
}
DSG::DSGFrequency const& DSG::AudioSettings::Nyquist(){
    return _nyquist;
}
```

8.15 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

```
//
//  AudioSettings.h
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__AudioSettings__
```

```
#define __DSG__AudioSettings__
#include "DSGTypes.h"
namespace DSG {
    /*!\brief DSG::AudioSettings - Global Storage For Audio Settings Such As Sample Rate
     */
    class AudioSettings{
    public:
        static DSG::DSGFrequency const& SampleRate();
        static DSG::DSGFrequency const& SampleRate(
DSG::DSGFrequency const& value);
        static DSG::DSGFrequency const& Nyquist();
    protected:
        static DSG::DSGFrequency _sampleRate;
        static DSG::DSGFrequency _nyquist;
    };
    //!\brief DSG::SampleRate - Get Global Sample Rate
    inline DSG::DSGFrequency const& SampleRate(){
        return DSG::AudioSettings::SampleRate();
    }
    //!\brief DSG::SampleRate - Set Global Sample Rate
    inline DSG::DSGFrequency const& SampleRate(
DSG::DSGFrequency const& value){
        return DSG::AudioSettings::SampleRate(value);
    }
    //!\brief DSG::Nyquist() - Pre-Calculated Nyquist Limit. Use instead of calculating each time needed.
 This value will be updated whenever the sample rate changes.
    inline DSG::DSGFrequency Nyquist(){
        return DSG::AudioSettings::Nyquist();
    }
}
#endif /* defined(__DSG__AudioSettings__) */
```

8.17 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

```
//
//  Blackman.h
//  DSG
//
//  Created by Alexander Zywicki on 9/24/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef DSG_Blackman_h
#define DSG_Blackman_h
#include "PI.h"
#include "LUT.h"
#include "Sine.h"
namespace DSG {
#ifdef DSG_Short_Names
    inline
#endif
    namespace Window{
        //!\brief DSG::Window::Blackman - Blackman Window Function
        template<typename decimal>
        inline decimal Blackman(decimal const& x){
            // Generate Blackman Window
            /*
             Blackman(x) = 0.42f - (0.5f * cos(2pi*x)) + (0.08f * cos(2pi*2.0*x));
             }*/
            static_assert(std::is_floating_point<decimal>::value==true,"DSG::Blackman Function Requires
 Floating Point Type");
            //we will implement the blackman window as a function as if it were sin(x)
            //cos input domain 0-1 not 0-2pi
            //range checking is handles within DSG::Cos
            decimal phs=x;
            while (phs>1.0) {
                phs-=1.0;
            }
            return 0.42 - (0.5 * DSG::Cos(phs))+(0.08 * DSG::Cos(2.0*phs));
        }
    }
}
#endif
```

8.19 BLIT.cpp File Reference

```
#include "BLIT.h"
```

8.20 BLIT.cpp

```
//
//  BLIT.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "BLIT.h"
DSG::BLIT::Blit::Blit():DSG::SignalGenerator(){
    Frequency(0);
}
DSG::BLIT::Blit::Blit(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset):DSG::SignalGenerator(frequency,offset){
    Frequency(frequency);
}
DSG::BLIT::Blit::~Blit(){}
```

8.21 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

```
//
//  BLIT.h
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__BLIT__
#define __DSG__BLIT__
#include "SignalGenerator.h"
#include "Denormal.h"
#include "Sinc.h"
#include "DSGMath.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //!DSG::BLIT - Namespace Containing BLIT Based Oscillators
    namespace BLIT{
        /*!\brief DSG::BLIT::Blit - Band-Limited Impulse Train Generator
         */
        //!\todo Re-write DSG::BLIT::Blit algorithm
        class Blit:public DSG::SignalGenerator{
        public:
            Blit();
            Blit(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
            virtual ~Blit();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
            virtual inline DSG::DSGFrequency const& Frequency(
DSG::DSGFrequency const& value);
        protected:
            unsigned long p_;
            unsigned long m_;
            unsigned long _h;
            double a_;
            DSG::DSGSample denominator;
            DSG::DSGSample value;
        };
        inline bool DSG::BLIT::Blit::Perform(
DSG::DSGSample& signal){
            //found better results in this case with built in sine function. not performance wise but
 algorithmically
            denominator = m_ * sin(PI*_phasor);
            if (DSG::IsDenormal(denominator)) {
                signal = a_;
            }else{
                value = sin(PI*_phasor * m_);
                value/=denominator;
                value*=a_;
                signal = value;
            }
            step();
            return true;
        }
        inline bool DSG::BLIT::Blit::Perform(
DSG::RingBuffer& signal){
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
        inline DSG::DSGFrequency const&
DSG::BLIT::Blit::Frequency(DSG::DSGFrequency const& value){
            this->SignalGenerator::Frequency(value);
            p_ = DSG::SampleRate()/_frequency;
            _h = (unsigned)floor(p_*0.5);
            m_ = 2 * (_h)+1;
            a_ = m_/(double)p_;
            return _frequency;
        }
    }
}
#endif /* defined(__DSG__BLIT__) */
```

8.23 BLITSaw.cpp File Reference

```
#include "BLITSaw.h"
```

8.24 BLITSaw.cpp

```
//
//  BLITSaw.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or "DSG".

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "BLITSaw.h"
DSG::BLIT::BlitSaw::BlitSaw():DSG::BLIT::Blit(),Register_(0){
    Frequency(0);
}
DSG::BLIT::BlitSaw::BlitSaw(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset):DSG::BLIT::Blit(frequency,offset),Register_(0){
    Frequency(frequency);
}
DSG::BLIT::BlitSaw::~BlitSaw(){}
```

8.25 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

```
//
//  BLITSaw.h
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__BLITSaw__
#define __DSG__BLITSaw__
#include "BLIT.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    namespace BLIT{
        //!\brief DSG::BLIT::BlitSaw - Saw Wave Generator Based on BLIT Algorithm
        //!\todo Re-write DSG::BLIT::BlitSaw algorithm
        class BlitSaw : public Blit{
        public:
            BlitSaw();
            BlitSaw(DSG::DSGFrequency const& frequency,
 DSG::DSGPhase const& offset);
            virtual ~BlitSaw();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
            virtual inline DSG::DSGFrequency const& Frequency(
 DSG::DSGFrequency const& value);
        protected:
            DSG::DSGSample C2_;
            DSG::DSGSample Register_;
        };
        inline bool DSG::BLIT::BlitSaw::Perform(
 DSG::DSGSample& signal){
            denominator = m_ * sin(PI*_phasor);
            if (DSG::IsDenormal(denominator)) {
                signal = a_;
            }else{
                value = sin(PI*_phasor * m_);
                value/=denominator;
                value*=a_;
                signal = value;
            }
            step();
            signal += (Register_ - C2_);
            Register_ = signal * 0.995;
            C2_+=signal;
            C2_*=0.5;
            return true;
        }
        inline bool DSG::BLIT::BlitSaw::Perform(
 DSG::RingBuffer& signal){
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
        inline DSG::DSGFrequency const&
 DSG::BLIT::BlitSaw::Frequency(DSG::DSGFrequency const& value)
 {
            this->SignalGenerator::Frequency(value);
            p_ = DSG::SampleRate()/_frequency;
            _h = (unsigned)floor(p_*0.5);
            m_ = 2 * (_h)+1;
            a_ = m_/(double)p_;
            C2_ = 1.0/(double)p_;
            return _frequency;
        }
    }
}
#endif /* defined(__DSG__BLITSaw__) */
```

8.27 BLITSquare.cpp File Reference

```
#include "BLITSquare.h"
```

8.28 BLITSquare.cpp

```
//
//  BLITSquare.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "BLITSquare.h"
```

8.29 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

```
//
//  BLITSquare.h
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__BLITSquare__
#define __DSG__BLITSquare__
#include "SignalGenerator.h"
namespace DSG {
#ifdef DSG_Short_Names
    inline
#endif
    namespace BLIT{
        //!\todo Write DSG::BLIT::BlitSquare algorithm
        class BlitSquare:public DSG::SignalGenerator{};
    }
}
#endif /* defined(__DSG__BLITSquare__) */
```

8.31 BLITTriangle.cpp File Reference

```
#include "BLITTriangle.h"
```

8.32 BLITTriangle.cpp

```
//
//  BLITTriangle.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "BLITTriangle.h"
```

8.33 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

```
//
//  BLITTriangle.h
//  DSG
//
//  Created by Alexander Zywicki on 9/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__BLITTriangle__
#define __DSG__BLITTriangle__

#include "SignalGenerator.h"
namespace DSG {
#ifdef DSG_Short_Names
    inline
#endif
    namespace BLIT{
        //!\todo Write DSG::BLIT::BlitTriangle algorithm
        class BlitTriangle:public DSG::SignalGenerator{};
    }
}
#endif /* defined(__DSG__BLITTriangle__) */
```

8.35 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

```
//
//  Bounds.h
//  DSG
//
//  Created by Alexander Zywicki on 11/11/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef DSG_Bounds_h
#define DSG_Bounds_h
#include <assert.h>
namespace DSG{
    //!\brief DSG::EnforceBounds - Clip value to set bounds
    template<int lower,int upper,typename decimal>
    decimal EnforceBounds(decimal const& value){
        if (value<lower) {
            return lower;
        }else if(value> upper){
            return upper;
        }else return value;
    }
    //!\brief DSG::StaticAssertBounds - Fails on compile time if value is not within bounds
    template<int lower,int upper,int value>
    void StaticAssertBounds(){
        static_assert(value>=lower && value<=upper,"Failed Static Bounds Assert");
    }
    //!\brief DSG::AssertBounds - Fails on runtime if value is not within bounds
    template<int lower,int upper,typename T>
    void AssertBounds(T const& value){
        assert(value>=lower && value<=upper);
    }
}
#endif
```

8.37 Buffer.cpp File Reference

```
#include "Buffer.h"
```

8.38 Buffer.cpp

```
//
//  Buffer.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "Buffer.h"
DSG::Buffer::Buffer():_size(0),_buffer(nullptr){}
DSG::Buffer::Buffer(size_t size):_size(size),_buffer(new
      DSG::DSGSample[size]){}
DSG::Buffer::Buffer(Buffer const& other) {
    _buffer = new  DSG::DSGSample[_size];
    _size = other._size;
    *this = other;
}
DSG::Buffer& DSG::Buffer::operator=(Buffer const& other){
    if (_size!=other._size) {
        if (_buffer!=nullptr) {
            delete [] _buffer;
        }
        _size = other._size;
        _buffer = new  DSG::DSGSample[_size];
    }
    for (int i=0; i<_size; ++i) {
        _buffer[i] = other._buffer[i];
    }
    return *this;
}
DSG::Buffer::~Buffer(){
    if (_buffer!=nullptr) {
        delete [] _buffer;
    }
}
DSG::DSGSample& DSG::Buffer::operator[](size_t const& index){
#ifdef DEBUG
    assert(index<_size);
#endif
    return _buffer[index];
}
```

8.39 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

```
//
//  Buffer.h
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__Buffer__
#define __DSG__Buffer__
#include <stddef.h>
#include "DSGTypes.h"
#ifdef DEBUG
#include <assert.h>
#endif
namespace DSG{
    /*!\brief DSG::Buffer - Base Class For DSG::RingBuffer. Not For Direct Use
    */
    class Buffer {
    public:
        Buffer();
        Buffer(size_t size);
        Buffer(Buffer const& other);
        Buffer& operator=(Buffer const& other);
        virtual ~Buffer();
        DSG::DSGSample& operator[](size_t const& index);
        inline size_t const& Size()const;
    protected:
        DSG::DSGSample* _buffer;
        size_t _size;
    };
    inline size_t const& DSG::Buffer::Size()const{
        return _size;
    }
}
#endif /* defined(__DSG__Buffer__) */
```

8.41 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

```
//
//  BufferConversion.h
//  DSG
//
//  Created by Alexander Zywicki on 10/14/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__BufferConversion__
#define __DSG__BufferConversion__
#include "RingBuffer.h"
namespace DSG {
    //!\brief DSG::RingToArray - Move Ring Buffer data to an array
    inline bool RingToArray(DSG::RingBuffer& ring,
DSG::DSGSample* array,unsigned long length){
        for (int i=0; i<length; ++i) {
            if (!ring.Empty()) {
                ring.Read(array[i]);
            }
        }return true;
    }
    //!\brief DSG::ArrayToRing - Move array data to a Ring Buffer
    inline bool ArrayToRing(DSG::RingBuffer& ring,
DSG::DSGSample* array, unsigned long length){
        int i=0;
        ring.Flush();
        while (!ring.Full()) {
            ring.Write(array[i]);
            ++i;
        }return true;
    }
}
#endif /* defined(__DSG__BufferConversion__) */
```

8.43 DCBlocker.cpp File Reference

```
#include "DCBlocker.h"
```

8.44 DCBlocker.cpp

```
//
//  DCBlocker.cpp
//  DSG
//
//  Created by Alexander Zywicki on 10/13/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "DCBlocker.h"
DSG::Filter::DCBlocker::DCBlocker():DSG::Filter::
FilterBase(),_a(0.995),xm1(0),ym1(0),x(0),_temp(0){}
DSG::Filter::DCBlocker::~DCBlocker(){}
```

8.45 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

```
//
//  DCBlocker.h
//  DSG
//
//  Created by Alexander Zywicki on 10/13/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__DCBlocker__
#define __DSG__DCBlocker__
#include "Filter.h"
namespace DSG {
#ifdef DSG_Short_Names
    inline
#endif
    namespace Filter{
        //!\brief DSG::Filter::DCBlocker - DC blocking filter
        class DCBlocker:public DSG::Filter::FilterBase {
        public:
            DCBlocker();
            virtual ~DCBlocker();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
        protected:
            unsigned long count;
            DSG::DSGSample _temp;
            DSG::DSGSample xm1;
            DSG::DSGSample ym1;
            DSG::DSGSample x;
            DSG::DSGSample _a;
        };
        inline bool DSG::Filter::DCBlocker::Perform(
      DSG::DSGSample& signal){
            x = signal;
            signal= x - xm1+ (_a * ym1);
            xm1 = x;
            ym1=signal;
            return true;
        }
        inline bool DSG::Filter::DCBlocker::Perform(
      DSG::RingBuffer& signal){
            if (!signal.Empty()) {
                count = signal.Count();
                while (count-- > 0) {
                    if(signal.Read(_temp)){
                        if (Perform(_temp)) {
                            signal.Write(_temp);
                        }else return false;
                    }else return false;
                }return true;
            }else return false;
        }
    }
}
#endif /* defined(__DSG__DCBlocker__) */
```

8.47 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

```
//
//  Delay.h
//  DSG
//
//  Created by Alexander Zywicki on 10/23/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__Delay__
#define __DSG__Delay__
#include "DSGTypes.h"
#include "SignalProcess.h"
#include "Interpolate.h"
#include "AudioSettings.h"
namespace DSG{
    //!\brief DSG::Delay - General purpose delay line
    template<unsigned long maxLength>
    class Delay:public DSG::SignalProcess{
    public:
        Delay():DSG::SignalProcess(),_max(maxLength),
_swap(0),_temp(0),count(0),_index(0),_delay(0){
            for (int i=0; i<_max; ++i) {
                _buffer[i]=0;
            }
        }
        Delay(double const& samples):DSG::SignalProcess(),
_max(maxLength),_swap(0),_temp(0),count(0),_index(0),
_delay(0){
            for (int i=0; i<_max; ++i) {
                _buffer[i]=0;
            }
            if (samples>maxLength) {
                _delay = maxLength;
            }else{
                _delay = samples;
            }
        }
        virtual ~Delay(){}
        virtual inline unsigned long const& Length()const{
            return _delay;
        }
        virtual inline unsigned long const& Length(unsigned long const& samples){
            if (samples>maxLength) {
                _delay = maxLength;
            }else{
                _delay = samples;
            }
            return _delay;
        }
        virtual inline bool Perform(DSG::DSGSample& signal);
        virtual inline bool Perform(DSG::RingBuffer& signal);
    protected:
        unsigned long count;
        unsigned long  _delay;
        unsigned long _index;
        const unsigned long _max;
        DSG::DSGSample _buffer[maxLength];
        DSG::DSGSample _swap;
        DSG::DSGSample _temp;
        virtual inline void increment(){
            ++_index;
            if (_index>_delay) {
                _index-=_delay;
            }
        }
    };
    template<unsigned long maxLength>
    inline bool DSG::Delay<maxLength>::Perform(
DSG::DSGSample& signal){
        _swap = _buffer[_index-1];
        _buffer[_index-1]=signal;
        signal = _swap;
        increment();
        return true;
    }
    template<unsigned long maxLength>
    inline bool DSG::Delay<maxLength>::Perform(
DSG::RingBuffer& signal){
        if (!signal.Empty()) {
            count = signal.Count();
            while (count-- > 0) {
                if(signal.Read(_temp)){
                    if (Perform(_temp)) {
                        signal.Write(_temp);
                    }else return false;
                }else return false;
            }return true;
        }else return false;
    }
}
#endif /* defined(__DSG__Delay__) */
```

8.49 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

```
//
//  Denormal.h
//  DSG
//
//  Created by Alexander Zywicki on 9/23/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef DSG_Denormal_h
#define DSG_Denormal_h
#include <limits>
#include "DSGMath.h"
namespace DSG{
    //!\brief DSG::IsDenormal - Returns True if number is Denormal
    template<typename T>
    inline bool IsDenormal(T const& value){
        return DSG::Abs(value)<=std::numeric_limits<T>::epsilon();//return true if number is
 denormal
    }
}
#endif
```

8.51 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

```
//
//  DPW.h
//  DSG
//
//  Created by Alexander Zywicki on 11/11/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef DSG_DPW_h
#define DSG_DPW_h
#include "DSGTypes.h"
#include "DSGMath.h"
#include "SignalGenerator.h"
#include "Bounds.h"
namespace DSG{
#ifdef DSG_Short_Names
```

```
    inline
#endif
    //!\brief DSG::DPW - Generators using the DPW method
    namespace DPW{
        //!\brief DSG::DPW::DPW_Polynomial - Polynoimal used in DPW Algorithm
        template<unsigned order>
        inline DSG::DSGSample DPW_Polynomial(DSG::DSGSample const& value){
            DSG::StaticAssertBounds<1,6,order>();//must be 1-6 order
            return value;
        }
        //!\brief DSG::DPW::DPW_Polynomial - 1st Order Polynoimal used in DPW Algorithm
        template<>
        inline DSG::DSGSample DPW_Polynomial<1>(DSG::DSGSample const& value){
            return value;
        }
        //!\brief DSG::DPW::DPW_Polynomial - 2nd order Polynoimal used in DPW Algorithm
        template<>
        inline DSG::DSGSample DPW_Polynomial<2>(DSG::DSGSample const& value){
            return DSG::Pow<2>(value);
        }
        //!\brief DSG::DPW::DPW_Polynomial - 3rd order Polynoimal used in DPW Algorithm
        template<>
        inline DSG::DSGSample DPW_Polynomial<3>(DSG::DSGSample const& value){
            return DSG::Pow<3>(value)-value;
        }
        //!\brief DSG::DPW::DPW_Polynomial - 4th order Polynoimal used in DPW Algorithm
        template<>
        inline DSG::DSGSample DPW_Polynomial<4>(DSG::DSGSample const& value){
            return DSG::Pow<2>(value) * (DSG::Pow<2>(value) - 2.0);
        }
        //!\brief DSG::DPW::DPW_Polynomial - 5th order Polynoimal used in DPW Algorithm
        template<>
        inline DSG::DSGSample DPW_Polynomial<5>(DSG::DSGSample const& value){
            return DSG::Pow<5>(value) - DSG::Pow<3>(value) * 10.0/3.0 + value * 7.0/3.0;
        }
        //!\brief DSG::DPW::DPW_Polynomial - 6th order Polynoimal used in DPW Algorithm
        template<>
        inline DSG::DSGSample DPW_Polynomial<6>(DSG::DSGSample const& value){
            return DSG::Pow<6>(value) - 5.0 * DSG::Pow<4>(value) + 7.0 * DPW_Polynomial<2>(value);
        }
#ifdef __APPLE__
#warning DSG::DPW - differentiators order 3-6 need verification. they cause major clipping
#endif
        //!\todo Fix DSG::DPW::DPW_Differentiator algorithms for orders 3-6
        //differentiators
        //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the DPW Algorithm
        template<unsigned order>
        class DPW_Differentiator{
        public:
            DPW_Differentiator(){
                DSG::StaticAssertBounds<1, 6,order>();//order must be 1-6
            }
        };
        //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 1st order DPW Algorithm
        template<>
        class DPW_Differentiator<1>{
        public:
            inline DSG::DSGSample operator()(DSG::DSGSample const& signal,DSG::DSGSample const& dt){
                return signal;
            }
        };
        //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 2nd order DPW Algorithm
        template<>
        class DPW_Differentiator<2>{
        public:
            inline DSG::DSGSample operator()(DSG::DSGSample const& signal,DSG::DSGSample const& dt){
                output = (signal - _delay)/(4.0 * dt);
                _delay = signal;
                return output;
            }
        protected:
            DSG::DSGSample output;
            DSG::DSGSample _delay;
        };
        //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 3rd order DPW
```

```
 Algorithm
        template<>
        class DPW_Differentiator<3>{
        public:
            inline DSG::DSGSample operator()(
DSG::DSGSample const& signal,DSG::DSGSample const& dt){
                output  = (signal - _delay[0]);
                output -= (_delay[0] - _delay[1]);
                output /= (24.*DSG::Pow<2>(dt));
                _delay[1]=_delay[0];
                _delay[0]=signal;
                return output;
            }
        protected:
            DSG::DSGSample output;
            DSG::DSGSample _delay[2];
        };
        //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 4th order DPW
 Algorithm
        template<>
        class DPW_Differentiator<4>{
        public:
            inline DSG::DSGSample operator()(
DSG::DSGSample const& signal,DSG::DSGSample const& dt){
                output  = (signal - _delay[0]);
                output -= (_delay[0] - _delay[1]);
                output -= (_delay[1] - _delay[2]);
                output /= 144*DSG::Pow<3>(dt);
                _delay[2]=_delay[1];
                _delay[1]=_delay[0];
                _delay[0]=signal;
                return output;
            }
        protected:
            DSG::DSGSample output;
            DSG::DSGSample _delay[3];
        };
        //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 5th order DPW
 Algorithm
        template<>
        class DPW_Differentiator<5>{
        public:
            inline DSG::DSGSample operator()(
DSG::DSGSample const& signal,DSG::DSGSample const& dt){
                output  = (signal - _delay[0]);
                output -= (_delay[0] - _delay[1]);
                output -= (_delay[1] - _delay[2]);
                output -= (_delay[2] - _delay[3]);
                output /= 960*DSG::Pow<4>(dt);
                _delay[3]=_delay[2];
                _delay[2]=_delay[1];
                _delay[1]=_delay[0];
                _delay[0]=signal;
                return output;
            }
        protected:
            DSG::DSGSample output;
            DSG::DSGSample _delay[4];
        };
        //!\brief DSG::DPW::DPW_Differentiator - Class Performing Differentiation for the 6th order DPW
 Algorithm
        template<>
        class DPW_Differentiator<6>{
        public:
            inline DSG::DSGSample operator()(
DSG::DSGSample const& signal,DSG::DSGSample const& dt){
                output  = (signal - _delay[0]);
                output -= (_delay[0] - _delay[1]);
                output -= (_delay[1] - _delay[2]);
                output -= (_delay[2] - _delay[3]);
                output -= (_delay[3] - _delay[4]);
                output /= 7200*DSG::Pow<5>(dt);
                _delay[4]=_delay[3];
                _delay[3]=_delay[2];
                _delay[2]=_delay[1];
                _delay[1]=_delay[0];
                _delay[0]=signal;
                return output;
            }
        protected:
            DSG::DSGSample output;
            DSG::DSGSample _delay[5];
        };
    }
}
#endif
```

8.53 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

```
//
//  DPWSaw.h
//  DSG
//
//  Created by Alexander Zywicki on 9/27/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__DPWSaw__
#define __DSG__DPWSaw__
#include "DPW.h"
namespace DSG {
#ifdef DSG_Short_Names
    inline
#endif
    namespace DPW{
        //!\brief DSG::DPW::DPWSaw - Sawtooth Generator using the Nth Order DPW algorithm
        template<unsigned order>
        class DPWSaw:public DSG::SignalGenerator{
        public:
            DPWSaw():DSG::SignalGenerator(),_register(0){
                DSG::StaticAssertBounds<1, 6,order>();
            }
            DPWSaw(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset):DSG::SignalGenerator(frequency,offset),
_register(0){DSG::StaticAssertBounds<1, 6,order>();}
            virtual ~DPWSaw(){}
            virtual inline bool Perform(DSG::DSGSample& signal){
                //trivial saw ramping from -1 to 1
                _register = _phasor;
                _register-=0.5;
                _register*=2.0;
                /*------------------------*/
                //DPW algorithm
                //polynomial shaping
                _register=DSG::DPW::DPW_Polynomial<order>(_register);
                //differentiating
                signal = _diff(_register,_dt);
                /*-------------------------*/
                //signal = DSG::EnforceBounds<-1, 1>(signal);
                //advance phase
                step();
                return true;
            }
            virtual inline bool Perform(DSG::RingBuffer& signal){
                signal.Flush();
                while (!signal.Full()) {
                    if (Perform(_storage)) {
                        if(signal.Write(_storage)){
                        }else return false;
                    }else return false;
                }return true;
            }
        protected:
            DSG::DSGSample _register;
            DSG::DPW::DPW_Differentiator<order> _diff;
        };
    }
}
#endif /* defined(__DSG__DPWSaw__) */
```

8.55 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;
```

```
    if (_out!=nullptr) {
        _buffer.Flush();
        _osc->Perform(_buffer);
        for (int i=0; i<frameCount; ++i) {
            _buffer.Read(_sample);
            *_out++ = _sample;
            *_out++ = _sample;
        }
    }
    return 0;
}
```

8.57 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.

```
                              {
    DSG::DSGSample* _out = (DSG::DSGSample*)output;
    DSG:: DSGSample _sample;
    DSG::SignalGenerator* _osc = (DSG::SignalGenerator*)userData;
    if (_out!=nullptr) {
        _buffer.Flush();
        _osc->Perform(_buffer);
        for (int i=0; i<frameCount; ++i) {
            _buffer.Read(_sample);
            *_out++ = _sample;
            *_out++ = _sample;
        }
    }
    return 0;
}
```

8.57.1.2 int DriverExit ( )

Definition at line 38 of file Driver.cpp.

```
                  {
    PaError err=0;
    err = Pa_StopStream(stream);
    if (err!=paNoError) {
#ifdef DEBUG
        printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
#endif
        return 1;
    }
    err = Pa_CloseStream( stream );
    if( err != paNoError ){
#ifdef DEBUG
        printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
#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 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

```
//
//  DSF.h
//  DSG
//
//  Created by Alexander Zywicki on 11/5/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__DSF__
#define __DSG__DSF__
#include "DSGMath.h"
#include "DSGTypes.h"
namespace DSG{
    template<typename decimal=DSG::DSGSample>
    decimal DSF(decimal const& beta,decimal const& theta,decimal const& N,decimal const& a){
#ifdef __APPLE__
#warning Untested DSG::DSF()
#endif
        decimal denominator = 1 + DSG::Pow<2>(a) - (2.0*a*cos(beta));
        decimal numerator = sin(theta) - a * sin(theta-beta) - pow(a, N+1) * (sin(theta + (N+1)*beta) - a*
sin(theta + (N*beta)));
        return numerator/denominator;
    }
}
#endif /* defined(__DSG__DSF__) */
```

8.61 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 26 of file DSG.h.

8.62 DSG.h

```
//
//  DSG.h
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef DSG_DSG_h
#define DSG_DSG_h
#define DSG_Short_Names // enables inlining of nested namespaces to allow shorter explicit typenames
//Example: DSG::Analog::AnalogSaw (Long Name)...DSG::AnalogSaw (Short Name)(only available with this macro
 enabled
//!\brief DSG - A Collection of tools for Digital Signal Generation
namespace DSG {}
#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"

#endif
```

8.63 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

```
//
//  Math.h
//  DSG
//
//  Created by Alexander Zywicki on 9/23/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef DSG_Math_h
#define DSG_Math_h
#include <math.h>
#include <type_traits>
namespace DSG {
    //!\brief DSG::Abs - Calculate absolute value
    template<typename T>
    inline T Abs(T const& value){
        return value < 0.0 ? -1.0 * value : value;
    }
    //!\brief DSG::Factorial - Compute integer factorial
    template<unsigned long N>
    struct Factorial{
        enum {value = N * Factorial<N-1>::value};
    };
    //!\brief DSG::Factorial - Compute integer factorial
    template<>
    struct Factorial<0>{
        enum{ value = 1 };
    };
    namespace{
        template<class T, unsigned N>
        struct power{
            static constexpr T value(const T x){
                return power<T, N-1>::value(x) * x;
            }
        };
        template<class T>
        struct power<T, 0>{
            static constexpr T value(const T x){
                return 1;
            }
        };
    }
    //!\brief DSG::Pow - Any type to an integer power, i.e. N ^ I
    template<unsigned exponent, class T>
    T constexpr Pow(T const base){
        return power<T, exponent>::value(base);
    }
}
#endif
```

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

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

8.66 DSGTypes.h

```
//
//  DSGTypes.h
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef DSG_DSGTypes_h
#define DSG_DSGTypes_h
namespace DSG {
    //!\brief DSG::DSGFrequency - Type for representing a frequency value
    typedef float DSGFrequency;
    //!\brief DSG::DSGPhase - Type for representing a phase value
    typedef float DSGPhase;
    //!\brief DSG::DSGSample - Type for representing an audio sample
    typedef float DSGSample;
}
#endif
```

8.67 EPTRSaw.cpp File Reference

```
#include "EPTRSaw.h"
```

8.68 EPTRSaw.cpp

```
//
//  EPTRSaw.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/29/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "EPTRSaw.h"
DSG::EPTR::EPTRSaw::EPTRSaw():DSG::SignalGenerator(){}
DSG::EPTR::EPTRSaw::EPTRSaw(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset):DSG::SignalGenerator(frequency,offset){}
DSG::EPTR::EPTRSaw::~EPTRSaw(){}
```

8.69 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

```
//
//  EPTRSaw.h
//  DSG
//
//  Created by Alexander Zywicki on 9/29/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__EPTRSaw__
#define __DSG__EPTRSaw__
#include "SignalGenerator.h"
namespace DSG {
#ifdef DSG_Short_Names
    inline
#endif
    //!DSG::EPTR - Generators Based On The Efficienct Polynomial Transfer Region Algorithm
    namespace EPTR{
        //!\brief DSG::EPTR::EPTRSaw-Sawtooth Wave Generator Using The Efficienct Polynomial Transfer Region Algorithm
        //!\todo Test and Possibly Re-Write DSG::EPTR::EPTRSaw algorithm
        class EPTRSaw : public DSG::SignalGenerator{
        public:
            EPTRSaw();
            EPTRSaw(DSG::DSGFrequency const& frequency, DSG::DSGPhase const& offset);
            virtual ~EPTRSaw();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
        protected:
            DSG::DSGSample _register;
        };
        inline bool DSG::EPTR::EPTRSaw::Perform( DSG::DSGSample& signal){
#ifdef __APPLE__
#warning Untested For Aliasing DSG::EPTR::EPTRSaw::Perform()
#endif
            //generate trivial saw
            _register = _phasor;
            _register+=0.5;
            if (_register>1.0) {
                --_register;
            }
            _register-=0.5;
            _register*=2.0;
            if (_register > 1.0-_dt) {
                //transition region detected
                //apply eptr correction
                signal = _register - (_register/_dt) + (1.0/_dt) -1;
            }else{
                signal = _register;
            }
            step();//avance phase
            return true;
        }
        inline bool DSG::EPTR::EPTRSaw::Perform( DSG::RingBuffer& signal){
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
    }
}
#endif /* defined(__DSG__EPTRSaw__) */
```

8.71 Filter.cpp File Reference

```
#include "Filter.h"
```

8.72 Filter.cpp

```
//
//  Filter.cpp
//  DSG
//
//  Created by Alexander Zywicki on 10/27/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "Filter.h"
DSG::Filter::FilterBase::FilterBase():_temp(0),count(0){}
DSG::Filter::FilterBase::~FilterBase(){}
```

8.73 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

```
//
//  Filter.h
//  DSG
//
//  Created by Alexander Zywicki on 10/27/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__Filter__
#define __DSG__Filter__
#include "SignalProcess.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //!\brief DSG::Filter - Filters
    namespace Filter{
        //!\brief DSG::Filter::FilterBase - Filter Base Class, implements interface for cutoff frequency
        class FilterBase:public DSG::SignalProcess{
        public:
            FilterBase();
            virtual ~FilterBase();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
            virtual inline bool Cutoff(DSG::DSGFrequency const& cutoff);
        protected:
            DSG::DSGSample _temp;
            unsigned long count;
        };
        inline bool DSG::Filter::FilterBase::Perform(
DSG::DSGSample& signal){
            return true;
        }
        inline bool DSG::Filter::FilterBase::Perform(
DSG::RingBuffer& signal){
            if (!signal.Empty()) {
                count = signal.Count();
                while (count-- > 0) {
                    if(signal.Read(_temp)){
                        if (Perform(_temp)) {
                            signal.Write(_temp);
                        }else return false;
                    }else return false;
                }return true;
            }else return false;
        }
        inline bool DSG::Filter::FilterBase::Cutoff(
DSG::DSGFrequency const& cutoff){
            return false;
        }
    }
}
#endif /* defined(__DSG__Filter__) */
```

8.75 FourierSaw.cpp File Reference

`#include "FourierSaw.h"`

8.76 FourierSaw.cpp

```
//
//  FourierSaw.cpp
//  DSG
//
```

```
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "FourierSaw.h"
DSG::Fourier::FourierSaw::FourierSaw():DSG::
SignalGenerator(),_a(1.7/PI),phs(0),value(0),i(0){}
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){
    _h = MaxHarms(_frequency)+1;
}
DSG::Fourier::FourierSaw::~FourierSaw(){}
```

8.77 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

```
//
//  FourierSaw.h
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__FourierSaw__
#define __DSG__FourierSaw__
#include "SignalGenerator.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //!DSG::Fourier - Namespace Containing Fourier Series Based Oscillators
    namespace Fourier{
        //!\brief DSG::Fourier::FourierSaw - Fourier Series Sawtooth Wave Generator
        class FourierSaw : public DSG::SignalGenerator {
        public:
            FourierSaw();
            FourierSaw(DSG::DSGFrequency const& frequency, DSG::DSGPhase const& offset);
            virtual ~FourierSaw();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
            virtual inline DSG::DSGFrequency const& Frequency(DSG::DSGFrequency const& value);
        protected:
            unsigned long _h;
            const double _a;
            double phs;
            double value;
            int i;
        };
        inline bool DSG::Fourier::FourierSaw::Perform(DSG::DSGSample& signal){
            //_h Sine Calls Per Sample where _h  is theoretically nyquist / frequency
            value=DSG::Sin(_phasor);
            for (i=2; i<_h; ++i) {
                value += (1.0/i) * DSG::Sin(_phasor*i);
            }
            value*=_a;
            signal = value;
            step();
            return true;
        }
        inline bool DSG::Fourier::FourierSaw::Perform(DSG::RingBuffer& signal){
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
        inline DSG::DSGFrequency const& DSG::Fourier::FourierSaw::Frequency(DSG::DSGFrequency const& value){
            _frequency = value;
            _dt = _frequency/DSG::SampleRate();
            _h = MaxHarms(_frequency);
            return _frequency;
        }
    }
}
#endif /* defined(__DSG__FourierSaw__) */
```

8.79 FourierSeries.cpp File Reference

```
#include "FourierSeries.h"
```

8.80 FourierSeries.cpp

```
//
//  FourierSeries.cpp
//  DSG
//
//  Created by Alexander Zywicki on 11/18/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
```

```
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "FourierSeries.h"
DSG::Fourier::Harmonic::Harmonic():_ratio(0),_amplitude(0){}
DSG::Fourier::Harmonic::Harmonic(DSG::DSGSample const& ratio,
DSG::DSGSample const& amplitude):_ratio(ratio),_amplitude(amplitude){}
DSG::Fourier::Harmonic::~Harmonic(){
    _ratio=0;
    _amplitude=0;
}
DSG::DSGSample const& DSG::Fourier::Harmonic::Ratio()const{
    return _ratio;
}
DSG::DSGSample const& DSG::Fourier::Harmonic::Ratio(
DSG::DSGSample const& value){
    _ratio = value;
    return _ratio;
}
DSG::DSGSample const& DSG::Fourier::Harmonic::Amplitude()
const{
    return _amplitude;
}
DSG::DSGSample const& DSG::Fourier::Harmonic::Amplitude(
DSG::DSGSample const& value){
    _amplitude=value;
    return _amplitude;
}
DSG::Fourier::FourierSeriesGenerator::FourierSeriesGenerator
():DSG::SignalGenerator(){}
DSG::Fourier::FourierSeriesGenerator::FourierSeriesGenerator
(DSG::DSGFrequency const& frequency, DSG::DSGPhase const& offset):
DSG::SignalGenerator(frequency,offset){}
DSG::Fourier::FourierSeriesGenerator::~FourierSeriesGenerator
(){}
```

8.81 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

```
//
//  FourierSeries.h
//  DSG
//
//  Created by Alexander Zywicki on 11/18/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__FourierSeries__
#define __DSG__FourierSeries__
#include "SignalGenerator.h"
#include <vector>
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    namespace Fourier{
        //!\brief DSG::Fourier::Harmonic - Represents a single harmonic in a Fourier Series.
        class Harmonic{
        public:
            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:
            DSG::DSGSample _ratio;
            DSG::DSGSample _amplitude;
        };
        //!\brief DSG::Fourier::FourierSeriesGenerator - Generates a wave form using a user specified
 Fourier Series
        class FourierSeriesGenerator: public
DSG::SignalGenerator{
        public:
            typedef std::vector<Harmonic> FourierSeries;
            FourierSeriesGenerator();
            FourierSeriesGenerator(DSG::DSGFrequency const&
frequency, DSG::DSGPhase const& offset);
            virtual ~FourierSeriesGenerator();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
            inline void Series(FourierSeries const& series);
            inline FourierSeries& Series();
        protected:
            FourierSeries _series;
            DSG::DSGSample value;
        };
        inline bool DSG::Fourier::FourierSeriesGenerator::Perform
(DSG::DSGSample& signal){
            value = _phasor;
            signal=0;
            for (auto i = _series.begin(); i!=_series.end(); ++i) {
                signal += DSG::Sin(_phasor * i->Ratio())*i->Amplitude();
            }
            step();
            return true;
        }
        inline bool DSG::Fourier::FourierSeriesGenerator::Perform
(DSG::RingBuffer& signal){
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
        inline void DSG::Fourier::FourierSeriesGenerator::Series
    (DSG::Fourier::FourierSeriesGenerator::FourierSeries
    const& series){
            _series = series;
        }
        inline DSG::Fourier::FourierSeriesGenerator::FourierSeries
    & DSG::Fourier::FourierSeriesGenerator::Series(){
            return _series;
        }
    }
}
#endif /* defined(__DSG__FourierSeries__) */
```

8.83 FourierSquare.cpp File Reference

```
#include "FourierSquare.h"
```

8.84 FourierSquare.cpp

```
//
//  FourierSquare.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "FourierSquare.h"
DSG::Fourier::FourierSquare::FourierSquare():
DSG::SignalGenerator(),_a(3.6/PI),phs(0),value(0),i(0){}
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){
    _h = MaxHarms(_frequency)+1;
}
DSG::Fourier::FourierSquare::~FourierSquare(){}
```

8.85 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

```
//
//  FourierSquare.h
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__FourierSquare__
#define __DSG__FourierSquare__
#include "SignalGenerator.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //!DSG::Fourier - Namespace Containing Fourier Series Based Oscillators
    namespace Fourier{
        //!\brief DSG::Fourier::FourierSquare - Fourier Series Square Wave Generator
        class FourierSquare : public DSG::SignalGenerator {
        public:
            FourierSquare();
            FourierSquare(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
            virtual ~FourierSquare();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
            virtual inline DSG::DSGFrequency const& Frequency(
DSG::DSGFrequency const& value);
        protected:
            unsigned long _h;
            const double _a;
            double phs;
            double value;
            int i;
        };
        inline bool DSG::Fourier::FourierSquare::Perform(
DSG::DSGSample& signal){
            //(_h/2)+1 Sine Calls Per Sample
            value=DSG::Sin(_phasor);//i=1
            for (i=3; i<_h; i+=2) {//i=3..5..7..
                value += (1.0/i) * DSG::Sin(_phasor*i);
            }
            value*=_a;
            signal = value;
            step();
            return true;
        }
        inline bool DSG::Fourier::FourierSquare::Perform(
DSG::RingBuffer& signal){
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
        inline DSG::DSGFrequency const&
DSG::Fourier::FourierSquare::Frequency(
DSG::DSGFrequency const& value){
            _frequency = value;
            _dt = _frequency/DSG::SampleRate();
            _h = MaxHarms(_frequency);
            return _frequency;
        }
    }
}
#endif /* defined(__DSG__FourierSquare__) */
```

8.87 FourierTriangle.cpp File Reference

```
#include "FourierTriangle.h"
```

8.88 FourierTriangle.cpp

```
//
//  FourierTriangle.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "FourierTriangle.h"
DSG::Fourier::FourierTriangle::FourierTriangle():
DSG::SignalGenerator(),_a(8.0/(PI*PI)),phs(0),value(0),i(0){}
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){
    _h = MaxHarms(_frequency)+1;
}
DSG::Fourier::FourierTriangle::~FourierTriangle(){}
```

8.89 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

```
//
//  FourierTriangle.h
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__FourierTriangle__
#define __DSG__FourierTriangle__
#include "SignalGenerator.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //!DSG::Fourier - Namespace Containing Fourier Series Based Oscillators
    namespace Fourier{
        //!\brief DSG::Fourier::FourierTriangle - Fourier Series Triangle Wave Generator
        class FourierTriangle : public DSG::SignalGenerator {
        public:
            FourierTriangle();
            FourierTriangle(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
            virtual ~FourierTriangle();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
            virtual inline DSG::DSGFrequency const& Frequency(
DSG::DSGFrequency const& value);
        protected:
            unsigned long _h;
            const double _a;
            double phs;
            double value;
            int i;
        };
        inline bool DSG::Fourier::FourierTriangle::Perform(
DSG::DSGSample& signal){
            //(_h/2)+1 Sine Calls Per Sample
            value=DSG::Sin(_phasor);//i=1
            double sgn = -1;
            for (i=3; i<_h; i+=2) {//i=3..5..7..
                value += sgn * (1.0/(i*i)) * DSG::Sin(_phasor*i);
                sgn*=-1;
            }
            value*=_a;
            signal = value;
            step();
            return true;
        }
        inline bool DSG::Fourier::FourierTriangle::Perform(
DSG::RingBuffer& signal){
            signal.Flush();
            while (!signal.Full()) {
                if (Perform(_storage)) {
                    if(signal.Write(_storage)){
                    }else return false;
                }else return false;
            }return true;
        }
        inline DSG::DSGFrequency const&
DSG::Fourier::FourierTriangle::Frequency(
    DSG::DSGFrequency const& value){
            _frequency = value;
            _dt = _frequency/DSG::SampleRate();
            _h = MaxHarms(_frequency);
            return _frequency;
        }
    }
}
#endif /* defined(__DSG__FourierTriangle__) */
```

8.91 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

```
//
//  Gaussian.h
//  DSG
//
//  Created by Alexander Zywicki on 10/6/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef DSG_Gaussian_h
#define DSG_Gaussian_h
#include "Sine.h"
#include "White.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    namespace Noise{
        //!\brief DSG::Noise::Gaussian - Gaussian Noise Generator Function
        template<typename decimal=DSG::DSGSample>
        decimal Gaussian(decimal=0.0){
            static decimal normalizer=1;//variable used to actively normalize the output
            //to enforce compatability with DSG::LUT a dummy parameter is applied
            //this parameter is useless except for compatability reasons
            decimal R1 = DSG::Noise::White();
            decimal R2 = DSG::Noise::White();
            decimal x= (decimal)sqrt(-2.0f * log(R1))*DSG::Cos(R2);
            if (DSG::Abs(x)>normalizer) {
                //store highest output
                normalizer=DSG::Abs(x);
            }
            x/=normalizer;//normalize
            return x;
        }
    }
}
#endif
```

8.93 GenericGenerator.cpp File Reference

`#include "GenericGenerator.h"`

8.94 GenericGenerator.cpp

```
//
//  GenericGenerator.cpp
//  DSG
//
//  Created by Alexander Zywicki on 10/21/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "GenericGenerator.h"
DSG::GenericGenerator::GenericGenerator():
DSG::SignalGenerator(){}
DSG::GenericGenerator::GenericGenerator(
DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset,
DSG::DSGSample (*signalFunction)(DSG::DSGSample const&)):
DSG::SignalGenerator(frequency,offset),_callback(signalFunction){}
DSG::GenericGenerator::~GenericGenerator(){}
```

8.95 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

```
//
//  GenericGenerator.h
//  DSG
//
//  Created by Alexander Zywicki on 10/21/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__GenericGenerator__
#define __DSG__GenericGenerator__
#include "SignalGenerator.h"
namespace DSG{
    //!\brief DSG::GenericGenerator - Generator designed to use a stateless generator function such as
 DSG::Sin()
    class GenericGenerator:public DSG::SignalGenerator{
    public:
        GenericGenerator();
        GenericGenerator(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset,DSG::DSGSample (*signalFunction)(
DSG::DSGSample const&));
        virtual ~GenericGenerator();
        virtual inline bool Perform(DSG::DSGSample& signal);
        virtual inline bool Perform(DSG::RingBuffer& signal);
    protected:
        DSG::DSGSample (*_callback)(DSG::DSGSample const&);
    };
    inline bool DSG::GenericGenerator::Perform(
DSG::DSGSample& signal){
        if (_callback!=nullptr) {
            signal = _callback(_phasor);
        }else signal = 0;
        step();
        return true;
    }
    inline bool DSG::GenericGenerator::Perform(
DSG::RingBuffer& signal){
        signal.Flush();
        while (!signal.Full()) {
            if (Perform(_storage)) {
                if(signal.Write(_storage)){
                }else return false;
            }else return false;
        }return true;
    }
}
#endif /* defined(__DSG__GenericGenerator__) */
```

8.97 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

```
//
//  Interpolate.h
//  DSG
//
//  Created by Alexander Zywicki on 10/21/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
//Code In this file was adapted from the code provided on this website
//http://paulbourke.net/miscellaneous/interpolation/
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef DSG_Interpolate_h
#define DSG_Interpolate_h
#include "DSGMath.h"
#include "PI.h"
namespace DSG{
    //!\brief DSG::LinearInterpolate - Linear Interpolation
    template<typename decimal>
    decimal LinearInterpolate(decimal const& y1,decimal const& y2,decimal const& mu){
        return(y1*(1-mu)+y2*mu);
    }
    //!\brief DSG::CosineInterpolate - Cosine Interpolation
    template<typename decimal>
    decimal CosineInterpolate(
                              decimal y1,decimal y2,
                              decimal mu)
    {
        decimal mu2;
        mu2 = (1-cos(mu*PI))/2.0;
        return(y1*(1-mu2)+y2*mu2);
    }
    //!\brief DSG::CubicInterpolate - Cubic Interpolation
    template<typename decimal>
    decimal CubicInterpolate(decimal const& y0,decimal const& y1,
                             decimal const& y2,decimal const& y3,
                             decimal const& mu)
    {
        decimal a0,a1,a2,a3,mu2;
        mu2 = mu*mu;
        a0 = y3 - y2 - y0 + y1;
        a1 = y0 - y1 - a0;
        a2 = y2 - y0;
        a3 = y1;
        return(a0*mu*mu2+a1*mu2+a2*mu+a3);
    }
    //!\brief DSG::HermiteInterpolate - Hermite 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)
    {
        /*
         Tension: 1 is high, 0 normal, -1 is low
         Bias: 0 is even,
         positive is towards first segment,
         negative towards the other
         */
        decimal m0,m1,mu2,mu3;
        decimal a0,a1,a2,a3;
        mu2 = mu * mu;
        mu3 = mu2 * mu;
        m0  = (y1-y0)*(1+bias)*(1-tension)/2.0;
        m0 += (y2-y1)*(1-bias)*(1-tension)/2.0;
        m1  = (y2-y1)*(1+bias)*(1-tension)/2.0;
        m1 += (y3-y2)*(1-bias)*(1-tension)/2.0;
        a0 =  2*mu3 - 3*mu2 + 1;
        a1 =    mu3 - 2*mu2 + mu;
        a2 =    mu3 -   mu2;
        a3 = -2*mu3 + 3*mu2;
        return(a0*y1+a1*m0+a2*m1+a3*y2);
    }
}
#endif
```

8.99 Leaky.cpp File Reference

```
#include "Leaky.h"
```

8.100 Leaky.cpp

```
//
//  Leaky.cpp
//  DSG
//
//  Created by Alexander Zywicki on 10/27/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "Leaky.h"
DSG::Filter::LeakyIntegrator::LeakyIntegrator():
      DSG::Filter::FilterBase(){
```

```
    x1=0;
    y1=0;
    a=0;
    b=0;
    y=0;
}
DSG::Filter::LeakyIntegrator::LeakyIntegrator(DSG::DSGFrequency const& cutoff):DSG::Filter::FilterBase(){
    x1=0;
    y1=0;
    a=0;
    b=0;
    y=0;
    Cutoff(cutoff);
}
DSG::Filter::LeakyIntegrator::~LeakyIntegrator(){
    x1=0;
    y1=0;
    a=0;
    b=0;
    y=0;
}
```

8.101 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

```
//
//  Leaky.h
//  DSG
//
//  Created by Alexander Zywicki on 10/27/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__Leaky__
#define __DSG__Leaky__
```

```
#include "Filter.h"
#include "PI.h"
namespace DSG {
#ifdef DSG_Short_Names
    inline
#endif
    namespace Filter{
        //!\brief DSG::Filter::LeakyIntegrator - Leaky integrator
        class LeakyIntegrator:public DSG::Filter::FilterBase{
        public:
            LeakyIntegrator();
            LeakyIntegrator(DSG::DSGFrequency const& cutoff);
            virtual ~LeakyIntegrator();
            virtual inline bool Perform(DSG::DSGSample& signal);
            virtual inline bool Perform(DSG::RingBuffer& signal);
            virtual inline bool Cutoff(DSG::DSGFrequency const& cutoff);
        protected:
            double x1,y1,a,b;
            double y;
        };
        inline bool DSG::Filter::LeakyIntegrator::Perform(
DSG::DSGSample& signal){
            y = b * (signal + x1) - a * y1;
            x1=signal;
            y1=y;
            signal=y;
            return true;
        }
        inline bool DSG::Filter::LeakyIntegrator::Perform(
DSG::RingBuffer& signal){
            if (!signal.Empty()) {
                count = signal.Count();
                while (count-- > 0) {
                    if(signal.Read(_temp)){
                        if (Perform(_temp)) {
                            signal.Write(_temp);
                        }else return false;
                    }else return false;
                }return true;
            }else return false;
        }
        inline bool DSG::Filter::LeakyIntegrator::Cutoff(
DSG::DSGFrequency const& cutoff){
            double Omega;
            x1 = y1 = 0.0;
            Omega = atan(PI * cutoff);
            a = -(1.0 - Omega) / (1.0 + Omega);
            b = (1.0 - b) / 2.0;
            return true;
        }
    }
}
#endif /* defined(__DSG__Leaky__) */
```

8.103 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

```
//
//  LUT.h
//  Waveform
//
//  Created by Alexander Zywicki on 8/25/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef Waveform_LUT_h
#define Waveform_LUT_h
#ifdef DEBUG
#include <assert.h>
#endif
#include "Interpolate.h"
namespace DSG{
    //!\brief DSG::LUT - Look Up Table
    template <typename element,unsigned long size>
    class LUT {
    public:
        typedef element (*FillFunction)(element);
        typedef element (*FillFunctionConstRef)(element const&);
        LUT():_size(size){}
        LUT(FillFunction fill,double const& range = 1.0):_size(size){
            //range is the expected input range for the function
            //example would  be 0-2pi or 0-1
            //would be provided a 2pi or 1
            //defaults to 1
            double step = range/(double)_size;
            phs = 0;
            for (int i=0; i<_size; ++i) {
                _table[i] = fill(phs);
                phs+=step;
            }
        }
        LUT(FillFunctionConstRef fill,double const& range = 1.0):
_size(size){
            //range is the expected input range for the function
            //example would  be 0-2pi or 0-1
            //would be provided a 2pi or 1
            //defaults to 1
            double step = range/_size;
            phs = 0;
            for (int i=0; i<_size; ++i) {
                _table[i] = fill(phs);
                phs+=step;
            }
        }
        ~LUT(){}
        element const& operator[](unsigned long const& index)const{
#ifdef DEBUG
            assert(index<_size);
#endif
            return _table[index];
        }
        element& operator[](unsigned long const& index){
#ifdef DEBUG
            assert(index<_size);
#endif
            return _table[index];
        }
        inline element const&  operator()(double const& x){
            phs=x;
            //need range checking on x to ensure 0-1 range
            phs<0 ? phs = 1-(phs*-1):0;
            phs-=((int)phs);
            return this->_table[(unsigned)(phs*(this->_size-1))];
        }
        unsigned long const& Size()const{
            return _size;
        }
    protected:
        element _table[size];
        const unsigned long _size;
        double phs;
    };
}
#endif
```

8.105 Noise.h File Reference

```
#include "Random.h"
#include "Gaussian.h"
#include "White.h"
#include "Pink.h"
#include "NoiseGenerator.h"
```

8.106 Noise.h

```
//
//  Noise.h
//  DSG
//
//  Created by Alexander Zywicki on 10/20/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef DSG_Noise_h
#define DSG_Noise_h
#include "Random.h"
#include "Gaussian.h"
#include "White.h"
#include "Pink.h"
#include "NoiseGenerator.h"
#endif
```

8.107 NoiseGenerator.cpp File Reference

```
#include "NoiseGenerator.h"
```

8.108 NoiseGenerator.cpp

```
//
//  NoiseGenerator.cpp
//  DSG
//
//  Created by Alexander Zywicki on 10/20/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
```

```
This file is part of the Digital Signal Generation Project or “DSG”.

DSG is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

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

You should have received a copy of the GNU General Public License
along with DSG.  If not, see <http://www.gnu.org/licenses/>.
*/
#include "NoiseGenerator.h"
DSG::NoiseGenerator::NoiseGenerator(DSGSample (*
StatelessFunction)(DSGSample)):DSG::SignalProcess(){
    _function = StatelessFunction;
}
DSG::NoiseGenerator::~NoiseGenerator(){}
```

8.109 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

```
//
//  NoiseGenerator.h
//  DSG
//
//  Created by Alexander Zywicki on 10/20/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
This file is part of the Digital Signal Generation Project or “DSG”.

DSG is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

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

You should have received a copy of the GNU General Public License
along with DSG.  If not, see <http://www.gnu.org/licenses/>.
*/
#ifndef __DSG__NoiseGenerator__
#define __DSG__NoiseGenerator__
#include "SignalGenerator.h"
namespace DSG{
    //!\brief DSG::NoiseGenerator - Generator that uses noise functions such as DSG::White() to generate
 signal
    class NoiseGenerator:public SignalProcess{
    public:
        NoiseGenerator(DSGSample (*StatelessFunction)(DSGSample));
        virtual ~NoiseGenerator();
        virtual inline bool Perform(DSG::DSGSample& signal);
        virtual inline bool Perform(DSG::RingBuffer& signal);
    protected:
        DSGSample (*_function)(DSGSample);
        DSG::DSGSample _storage;
    };
    inline bool DSG::NoiseGenerator::Perform(DSG::DSGSample& signal){
        signal = _function(0);
        return true;
    }
    inline bool DSG::NoiseGenerator::Perform(DSG::RingBuffer& signal){
        signal.Flush();
        while (!signal.Full()) {
            if (Perform(_storage)) {
                if(signal.Write(_storage)){
                }else return false;
            }else return false;
        }return true;
    }
}
#endif /* defined(__DSG__NoiseGenerator__) */
```

8.111 PI.h File Reference

Namespaces

- DSG

    DSG - A Collection of tools for Digital Signal Generation.

Macros

- #define PI 3.141592653589793238462643383278
- #define TWOPI 6.283185307179586476925286766559

8.111.1 Macro Definition Documentation

8.111.1.1 #define PI 3.141592653589793238462643383278

Definition at line 27 of file PI.h.

8.111.1.2 #define TWOPI 6.283185307179586476925286766559

Definition at line 28 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 /*
00009  This file is part of the Digital Signal Generation Project or “DSG”.
00010
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
```

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef DSG_PI_h
#define DSG_PI_h
namespace DSG{
#define PI    3.14159265358979323846264338327
#define TWOPI 6.28318530717958647692528676656
}
#endif
```

8.113 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

```
//
//  Pink.h
//  DSG
//
//  Created by Alexander Zywicki on 10/8/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef DSG_Pink_h
#define DSG_Pink_h
#include "Gaussian.h"
#include "DCBlocker.h"
namespace DSG{
#ifdef DSG_Short_Names
```

```
    inline
#endif
    namespace Noise{
        //!\brief DSG::Noise::Pink - Pink Noise Generator Function
        template<typename decimal=DSG::DSGSample>
        decimal Pink(decimal=0.0){
            //routine: Get white or gaussian, filter, return
            static decimal b0,b1,b2,b3,b4,b5,b6;
            static decimal normalizer=1;//variable used to actively normalize the output
            static DSG::DCBlocker _block;
            decimal white = DSG::Noise::Gaussian();
            decimal pink;
            //pinking filter
            b0 = 0.99886 * b0 + white * 0.0555179;
            b1 = 0.99332 * b1 + white * 0.0750759;
            b2 = 0.96900 * b2 + white * 0.1538520;
            b3 = 0.86650 * b3 + white * 0.3104856;
            b4 = 0.55000 * b4 + white * 0.5329522;
            b5 = -0.7616 * b5 - white * 0.0168980;
            pink = b0 + b1 + b2 + b3 + b4 + b5 + b6 + white * 0.5362;
            b6 = white * 0.115926;
            if (DSG::Abs(pink)>normalizer) {
                //store highest output
                normalizer=DSG::Abs(pink);
            }
            pink/=normalizer;
            _block.Perform(pink);
            return pink;
        }
    }
}
#endif
```

8.115 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 45 of file Random.h.

8.116 Random.h

```
//
//  Random.h
//  DSG
```

```
//
//  Created by Alexander Zywicki on 10/28/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef DSG_Random_h
#define DSG_Random_h
#include "DSGTypes.h"
#include <random>
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //!\brief DSG::Noise - Noise Generators
    namespace Noise{
        namespace{
            template<typename decimal>
            class random_helper{
            public:
                random_helper(){
                    srand(static_cast<unsigned>(time(NULL)));
                }
                inline decimal next(){
                    return static_cast<decimal>(rand()/max);
                }
            protected:
                const decimal max = static_cast<decimal>(RAND_MAX);
            };
        }
        //!\brief DSG::Noise::Random - Random Number Function
        template<typename decimal = DSG::DSGSample>
        inline decimal Random(decimal=0.0){
            static DSG::Noise::random_helper<decimal> _rand{};
            return _rand.next();
        }
    }
}
#endif
```

8.117 RingBuffer.cpp File Reference

`#include "RingBuffer.h"`

8.118 RingBuffer.cpp

```
//
//  RingBuffer.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "RingBuffer.h"
DSG:: RingBuffer::RingBuffer():Buffer(0),_read(0),_write(0),_count(0),
MASK(0){}
DSG:: RingBuffer::RingBuffer(const size_t size):
Buffer(make_pow_2(size)),_read(0),_write(0),_count(0){
    MASK = this->_size-1;
}
DSG:: RingBuffer::RingBuffer(RingBuffer& buffer):
Buffer(buffer){
    _write.store(buffer._write.load(std::memory_order_acquire));
    _read.store(buffer._read.load(std::memory_order_acquire));
    _count = buffer._count;
    MASK = buffer._size-1;
}
DSG:: RingBuffer& DSG:: RingBuffer::operator=(
RingBuffer& buffer){
    Buffer::operator=(buffer);
    _write.store(buffer._write.load(std::memory_order_acquire));
    _read.store(buffer._read.load(std::memory_order_acquire));
    _count = buffer._count;
    MASK = buffer._size-1;
    return *this;
}
DSG:: RingBuffer::~RingBuffer(){Flush();}

```

8.119 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

```
//
//  RingBuffer.h
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__RingBuffer__
#define __DSG__RingBuffer__
#ifdef DEBUG
#include <iostream>
#endif
#include <atomic>
#include "DSGMath.h"
#include "Buffer.h"
namespace DSG {
    /*!\brief DSG::RingBuffer - Circular Buffer of Audio
     */
    class RingBuffer:public DSG::Buffer {
    protected:
        std::atomic<size_t> _write;
        std::atomic<size_t> _read;
        size_t _count;
        size_t MASK;
        size_t write;
        size_t read;
        inline size_t next(size_t current);
        inline size_t make_pow_2(size_t number);
    public:
        RingBuffer();
        RingBuffer(const size_t size);
        RingBuffer(RingBuffer& buffer);
        RingBuffer& operator=(RingBuffer& buffer);
        virtual ~RingBuffer();
        inline bool Write(const DSGSample& elem);
        inline bool Read(DSG::DSGSample& elem);
        inline size_t const& Count()const;
        inline bool Full()const;
        inline bool Empty()const;
        inline void Flush();
        friend bool operator>>(DSG::DSGSample const& signal, DSG::RingBuffer& buffer){
            return buffer.Write(signal);
        }
        friend bool operator<<(DSG::DSGSample& signal, DSG::RingBuffer& buffer){
            return buffer.Read(signal);
        }
#ifdef DEBUG
        friend std::ostream& operator<<(std::ostream& os, DSG:: RingBuffer const& buffer){
            if (!buffer.Empty()) {
                size_t index= buffer._read;
                size_t count=buffer.Count();
                size_t size = buffer.Size();
                for (int i=0; i<count; ++i) {
                    os<<index<<": "<<buffer._buffer[index]<<std::endl;
                    index = ((index+1)%size);
                }
            }return os;
        }
#endif
    };
    inline bool DSG::RingBuffer::Full()const{
        return _count==this->_size;
    }
    inline bool DSG::RingBuffer::Empty()const{
        return _count==0;
    }
    inline void DSG::RingBuffer::Flush(){
        _write.store(0,std::memory_order_relaxed);
        _read.store(0,std::memory_order_relaxed);
        _count=0;
    }
    inline bool DSG::RingBuffer::Write(const DSGSample& elem){
        if (!Full()) {
            write = _write.load(std::memory_order_acquire);
            _write.store(next(write),std::memory_order_release);
            this->_buffer[write] = elem;
            ++_count;
            return true;
        }else return false;
    }
    inline bool DSG::RingBuffer::Read(DSGSample& elem){
        if (!Empty()) {
            read = _read.load(std::memory_order_acquire);
            _read.store(next(read),std::memory_order_release);
            elem = this->_buffer[read];
            --_count;
            return true;
        }else return false;
    }
    inline size_t const& DSG::RingBuffer::Count()const{
        return _count;
    }
    //note: RingBuffer implementation will force a power of 2 size to allow use of bitwise increment.
    inline size_t DSG::RingBuffer::next(size_t current){return (current+1) & MASK;}
    inline size_t DSG::RingBuffer::make_pow_2(size_t number){
        return pow(2, ceil(log(number)/log(2)));
    }
}
#endif /* defined(__DSG__RingBuffer__) */
```

8.121 SignalGenerator.cpp File Reference

#include "SignalGenerator.h"

8.122 SignalGenerator.cpp

```
//
//  SignalGenerator.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "SignalGenerator.h"
DSG::SignalGenerator::SignalGenerator():DSG::
SignalProcess(),_phasor(0),_frequency(0),_dt(0),_offset(0){}
DSG::SignalGenerator::SignalGenerator(
DSG::DSGFrequency const& frequency,DSG::DSGPhase const& offset):_phasor(0),
_frequency(frequency),_dt(0),_offset(offset){
    Frequency(frequency);
    Phase(offset);
}
DSG::SignalGenerator::~SignalGenerator(){}
```

8.123 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

```
//
//  SignalGenerator.h
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__SignalGenerator__
#define __DSG__SignalGenerator__
#include "SignalProcess.h"
#include "AudioSettings.h"
#include "Sine.h"
#include "Bounds.h"
namespace DSG{
    /*!\brief DSG::SignalGenerator - Extends DSG::Signal Process With Tools For Signal Generation
     */
    class SignalGenerator:public DSG::SignalProcess{
    public:
        SignalGenerator();
        SignalGenerator(DSG::DSGFrequency const& frequency,
DSG::DSGPhase const& offset);
        virtual ~SignalGenerator();
        virtual inline bool Perform(DSG::DSGSample& signal);
        virtual inline bool Perform(DSG::RingBuffer& signal);
        //Adds interface for control rate processing
        virtual inline DSG::DSGFrequency const& Frequency();
        virtual inline DSG::DSGFrequency const& Frequency(
DSG::DSGFrequency const& value);
        virtual inline DSG::DSGPhase const& Phase();
        virtual inline DSG::DSGPhase const& Phase(
DSG::DSGPhase const& value);
    protected:
        //extends sample rate interface
        inline void step();
        inline void sync();
        //-----------------------------
        DSG::DSGFrequency _frequency;//frequency in Hz
        DSG::DSGPhase _dt;//delta time (change in phase per sample) unit: phase 0-1
        DSG::DSGPhase _offset;//phase shift
        DSG::DSGPhase _phasor;//phase counter
        DSG::DSGSample _storage;//storage variable for calculations
    };
    inline unsigned long MaxHarms(DSG::DSGFrequency const& frequency){
        double _s = DSG::SampleRate()*  20000.0/DSG::SampleRate();
        _s/=frequency;
        return _s;
    }
}
inline bool DSG::SignalGenerator::Perform(
DSG::DSGSample& signal){
    signal=0;
    return false;
}
inline bool DSG::SignalGenerator::Perform(
DSG::RingBuffer& signal){
    signal.Flush();
    return false;
}
inline DSG::DSGFrequency const& DSG::SignalGenerator::Frequency
(){
    return _frequency;
}
inline DSG::DSGFrequency const& DSG::SignalGenerator::Frequency
(DSG::DSGFrequency const& value){
    _frequency = DSG::EnforceBounds<0, 20000,DSG::DSGSample>(value);
    _dt = _frequency/DSG::SampleRate();
    return _frequency;
}
inline DSG::DSGPhase const& DSG::SignalGenerator::Phase(){
    return _offset;
}
inline DSG::DSGPhase const& DSG::SignalGenerator::Phase(
DSG::DSGPhase const& value){
    _offset-=value;
    _phasor-=_offset;
    _offset=value;
    return _offset;
}
inline void DSG::SignalGenerator::step(){
    _phasor+=_dt;
    _phasor>1.0 ? --_phasor:0;
}
inline void DSG::SignalGenerator::sync(){
    _phasor=_offset;
}
#endif /* defined(__DSG__SignalGenerator__) */
```

8.125 SignalProcess.cpp File Reference

`#include "SignalProcess.h"`

8.126 SignalProcess.cpp

```
//
//  SignalProcess.cpp
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#include "SignalProcess.h"
DSG::SignalProcess::SignalProcess(){}
DSG::SignalProcess::~SignalProcess(){}
```

8.127 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

```
//
//  SignalProcess.h
//  DSG
//
//  Created by Alexander Zywicki on 9/16/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__SignalProcess__
#define __DSG__SignalProcess__
#include "DSGTypes.h"
#include "RingBuffer.h"
namespace DSG {
    /*!\brief DSG::SignalProcess - Defines Base Interface For Audio Processing
     */
    class SignalProcess{
    public:
        SignalProcess();
        virtual ~SignalProcess();
        //Defines Interface for sample rate processing
        virtual inline bool Perform(DSG::DSGSample& signal)=0;
        virtual inline bool Perform(DSG::RingBuffer& signal)=0;
    };
}
#endif /* defined(__DSG__SignalProcess__) */
```

8.129 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

```
//
//  Sinc.h
//  DSG
//
//  Created by Alexander Zywicki on 9/23/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__Sinc__
#define __DSG__Sinc__
#include "PI.h"
#include "Sine.h"
#include "Denormal.h"
#include <type_traits>
#include "DSGMath.h"
namespace DSG{
    //!\brief DSG::Sinc - Implements the Sinc() function (sin(PI*x)/PI*x)
    template<typename decimal>
    inline decimal Sinc(decimal const& x) {
        static_assert(std::is_floating_point<decimal>::value==true,"DSG::Sinc Function Requires Floating
 Point Type");
        decimal pix;
        if (DSG::IsDenormal(x)) {
            return 1.0;
        }else{
            pix = PI*x;
            return DSG::Sin(pix)/pix;
        }
    }
}
#endif /* defined(__DSG__Sinc__) */
```

8.131 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 30 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 /*
00009  This file is part of the Digital Signal Generation Project or “DSG”.
00010 
00011  DSG is free software: you can redistribute it and/or modify
00012  it under the terms of the GNU General Public License as published by
00013  the Free Software Foundation, either version 3 of the License, or
00014  (at your option) any later version.
00015 
00016  DSG is distributed in the hope that it will be useful,
00017  but WITHOUT ANY WARRANTY; without even the implied warranty of
00018  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00019  GNU General Public License for more details.
00020 
00021  You should have received a copy of the GNU General Public License
00022  along with DSG.  If not, see <http://www.gnu.org/licenses/>.
00023  */
00024 #ifndef __DSG__Sine__
00025 #define __DSG__Sine__
00026 #include "LUT.h"
00027 #include "PI.h"
00028 namespace DSG {
00029     namespace{
00030             #define LUT_SIZE 16384
```

```
        typedef enum Sine_Implementations{
            /*!\brief DSG::Sine_Implementations - Specifies The Implementation Option For DSG::Sin<>()*/
            Sine_Taylor =1,
            Sine_LUT =2,
            Sine_Default = Sine_LUT
        }Sine_Implementations;
        /*!\brief DSG::Sin() - Templated Sin Function With Optional Implementation
         */
        template<unsigned implementation> inline double Sin(double const& x){
            return 0;
        }
        /*!\brief DSG::Sin() - Templated Cos Function With Optional Implementation
         */
        template<unsigned implementation> inline double Cos(double const& x){
            return 0;
        }
        template<> inline double Sin<Sine_LUT>(double const& x){
            static DSG::LUT<double, LUT_SIZE> _lut(&sin,
TWOPI);
            return _lut(x);
        }
        template<> inline double Cos<Sine_LUT>(double const& x){
            static DSG::LUT<double, LUT_SIZE> _lut(&cos,
TWOPI);
            return _lut(x);
        }
        template<> inline double Sin<Sine_Taylor>(double const& x){
            //taylor serie version here
            return 0;
        }
        template<> inline double Cos<Sine_Taylor>(double const& x){
            //taylor series version here
            return 0;
        }
    }
    /*!\brief DSG::Sin() - General Purpose Sin Function, double precision
     */
    inline double Sin(double const& x){
        return static_cast<double>(Sin<Sine_Default>(x));//wrap default implementation as non template
    }
    /*!\brief DSG::Sin() - General Purpose Sin Function, single precision
     */
    inline float Sin(float const& x){
        return static_cast<float>(Sin<Sine_Default>(x));
    }
    /*!\brief DSG::Cos() - General Purpose Cos Function, double precision
     */
    inline double Cos(double const& x){
        return static_cast<double>(Cos<Sine_Default>(x));//wrap default implementation as non template
    }
    /*!\brief DSG::Cos() - General Purpose Cos Function, single precision
     */
    inline float Cos(float const& x){
        return static_cast<float>(Cos<Sine_Default>(x));
    }
}
#endif /* defined(__DSG__Sine__) */
```

8.133 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

```
//
//  Sleep.h
//  DSG
//
//  Created by Alexander Zywicki on 10/5/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef __DSG__Sleep__
#define __DSG__Sleep__
#include <chrono>
#include <thread>
namespace DSG{
    //!\brief DSG::Sleep - Millisecond Sleep Function
    template<typename integer>
    void Sleep(integer const& milliseconds){
        std::this_thread::sleep_for(std::chrono::milliseconds(milliseconds));
    }
}
#endif /* defined(__DSG__Sleep__) */
```

8.135 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

```
//
//  White.h
//  DSG
//
//  Created by Alexander Zywicki on 10/14/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef DSG_White_h
#define DSG_White_h
#include "DSGTypes.h"
#include "Random.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    namespace Noise{
        //!\brief DSG::Noise::White - White Noise Generator Function
        template<typename decimal = DSG::DSGSample>
        inline decimal White(decimal=0.0){
            return DSG::Random<decimal>();
        }
    }
}
#endif
```

8.137 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

```
//
```

```
//  Window.h
//  DSG
//
//  Created by Alexander Zywicki on 10/17/14.
//  Copyright (c) 2014 Alexander Zywicki. All rights reserved.
//
/*
 This file is part of the Digital Signal Generation Project or “DSG”.

 DSG is free software: you can redistribute it and/or modify
 it under the terms of the GNU General Public License as published by
 the Free Software Foundation, either version 3 of the License, or
 (at your option) any later version.

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

 You should have received a copy of the GNU General Public License
 along with DSG.  If not, see <http://www.gnu.org/licenses/>.
 */
#ifndef DSG_Window_h
#define DSG_Window_h
#include "LUT.h"
namespace DSG{
#ifdef DSG_Short_Names
    inline
#endif
    //!\brief DSG::Window - Window functions and utilities
    namespace Window{
        //!\brief 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 const&),decimal range = 1.0){
            decimal step = range/(decimal)lut.Size();
            decimal phs=0;
            for (int i=0; i<lut.Size(); ++i) {
                lut[i]*=windowFunction(phs);
                phs+=step;
            }
        }
        //!\brief 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){
            decimal step = range/(decimal)lut.Size();
            decimal phs=0;
            for (int i=0; i<lut.Size(); ++i) {
                lut[i]*=windowFunction(phs);
                phs+=step;
            }
        }
    }
}
#endif
```

Index