DaisySP

Generated by Doxygen 1.8.18

Chapter 1

Main Page

DSP Library for the Daisy product family...and elsewhere!

DaisySP is an open source DSP library written in C++ and specifically tailored to embedded audio applications.

1.0.0.1 Getting Started

- Browse the reference documentation at `/doc/`
- Check out our `How to Build` Wiki page.
- Make some sound!

1.0.0.2 Contributing

We'd love to have you become a contributor!

Here are ways that you can get involved:

- Make new DSP modules. See issues labeled "feature".
- Port existing DSP modules from other open source projects (MIT). See issues labeled "port".
- Fix problems with existing modules. See issues labeled "bug" and/or "polish".
- Test existing functionality and make issues.

Before working on code, please check out our `Contribution Guidelines` and `/doc/Style↩ Guide.pdf`

1.0.0.3 License

DaisySP is licensed with the permissive MIT open source license.

This allows for modification and reuse in both commercial and personal projects. It does not provide a warranty of any kind.

For the full license, read the `LICENSE` file in the root directory.

Chapter 2

Todo List

Class daisysp::AdEnv

- Add Cycling

- Implement Curve (its only linear for now).

- Maybe make this an ADsr_ that has AD/AR/Asr_ modes.

Class daisysp::Compressor

With fixed controls this is relatively quick, but changing controls now costs a lot more

Still pretty expensive

Add soft/hard knee settings

Maybe make stereo possible? (needing two for stereo is a bit silly, and their gain shouldn't be totally unique.

Class daisysp::NlFilt

make this work on a single sample instead of just on blocks at a time.

Class daisysp::Phasor

Selecting which channels should be initialized/included in the sequence conversion.

Setup a similar start function for an external mux, but that seems outside the scope of this file.

" move this to dsp.h

move this to dsp.h and name more appropriately

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

Class Documentation

4.1 daisysp::AdEnv Class Reference

```
#include <adenv.h>
```

Public Member Functions

- void Init (float sample_rate)
- float Process ()
- void Trigger ()
- void SetTime (uint8_t seg, float time)
- void SetCurve (float scalar)
- void SetMin (float min)
- void SetMax (float max)
- float GetValue () const
- uint8_t GetCurrentSegment ()
- bool IsRunning () const

4.1.1 Detailed Description

Trigger-able envelope with adjustable min/max, and independent per-segment time control.

Author

shensley

Todo

- Add Cycling

- Implement Curve (its only linear for now).

- Maybe make this an ADsr_ that has AD/AR/Asr_ modes.

4.1.2 Member Function Documentation

4.1.2.1 GetCurrentSegment()

```
uint8_t daisysp::AdEnv::GetCurrentSegment ( ) [inline]
```

Returns the segment of the envelope that the phase is currently located in.

4.1.2.2 GetValue()

```
float daisysp::AdEnv::GetValue ( ) const [inline]
```

Returns the current output value without processing the next sample

4.1.2.3 Init()

```
void AdEnv::Init (
            float sample_rate )
```

Initializes the ad envelope.

Defaults:

- current segment = idle
- curve = linear
- phase = 0
- min = 0
- max = 1

Parameters

sample_rate	sample rate of the audio engine being run

4.1.2.4 IsRunning()

```
bool daisysp::AdEnv::IsRunning ( ) const [inline]
```

Returns true if the envelope is currently in any stage apart from idle.

4.1.2.5 Process()

```
float AdEnv::Process ( )
```

Processes the current sample of the envelope. This should be called once per sample period.

Returns

the current envelope value.

4.1.2.6 SetCurve()

```
void daisysp::AdEnv::SetCurve (
             float scalar ) [inline]
```

Sets the amount of curve applied. A positve value will create a log curve. Input range: -100 to 100. (or more)

4.1.2.7 SetMax()

```
void daisysp::AdEnv::SetMax (
             float max ) [inline]
```

Sets the maximum value of the envelope output. Input range: -FLTmax_, to FLTmax_

4.1.2.8 SetMin()

```
void daisysp::AdEnv::SetMin (
             float min ) [inline]
```

Sets the minimum value of the envelope output. Input range: -FLTmax_, to FLTmax_

4.1.2.9 SetTime()

```
void daisysp::AdEnv::SetTime (
             uint8_t seg,
             float time ) [inline]
```

Sets the length of time (in seconds) for a specific segment.

4.1.2.10 Trigger()

```
void daisysp::AdEnv::Trigger ( ) [inline]
```

Starts or retriggers the envelope.

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

- modules/adenv.h
- modules/adenv.cpp

4.2 daisysp::Adsr Class Reference

```
#include <adsr.h>
```

Public Member Functions

- void Init (float sample_rate)
- float Process (bool gate)
- void SetTime (int seg, float time)
- void SetSustainLevel (float sus_level)
- uint8_t GetCurrentSegment ()
- bool IsRunning () const

4.2.1 Detailed Description

adsr envelope module Original author(s) : Paul Batchelor Ported from Soundpipe by Ben Sergentanis, May 2020

4.2.2 Member Function Documentation

4.2.2.1 GetCurrentSegment()

```
uint8_t daisysp::Adsr::GetCurrentSegment ( )  [inline]
```

Returns the segment of the envelope that the phase is currently located in.

4.2.2.2 Init()

```
void Adsr::Init (
            float sample_rate )
```

Initializes the ATone module. sample_rate - The sample rate of the audio engine being run.

4.2.2.3 IsRunning()

```
bool daisysp::Adsr::IsRunning ( ) const  [inline]
```

Returns true if the envelope is currently in any stage apart from idle.

4.2.2.4 Process()

```
float Adsr::Process (
            bool gate )
```

Processes one sample through the filter and returns one sample. gate - trigger the envelope, hold it to sustain

4.2.2.5 SetSustainLevel()

```
void daisysp::Adsr::SetSustainLevel (
            float sus_level )  [inline]
```

Arguments float sus_level, sets sustain level

4.2.2.6 SetTime()

```
void daisysp::Adsr::SetTime (
            int seg,
            float time )  [inline]
```

Set time per segment in seconds

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

- modules/adsr.h
- modules/adsr.cpp

4.3 daisysp::ATone Class Reference

```
#include <atone.h>
```

Public Member Functions

- void Init (float sample_rate)
- float Process (float &in)
- void SetFreq (float &freq)
- float GetFreq ()

4.3.1 Detailed Description

A first-order recursive high-pass filter with variable frequency response. Original Author(s): Barry Vercoe, John FFitch, Gabriel Maldonado Year: 1991 Original Location: Csound – OOps/ugens5.c Ported from soundpipe by Ben Sergentanis, May 2020

4.3.2 Member Function Documentation

4.3.2.1 GetFreq()

```
float daisysp::ATone::GetFreq ( )  [inline]
```

Returns the current value for the cutoff frequency or half-way point of the filter.

4.3.2.2 Init()

```
void ATone::Init (
            float sample_rate )
```

Initializes the ATone module. sample_rate - The sample rate of the audio engine being run.

4.3.2.3 Process()

```
float ATone::Process (
            float & in )
```

Processes one sample through the filter and returns one sample. in - input signal

4.3.2.4 SetFreq()

```
void daisysp::ATone::SetFreq (
            float & freq )  [inline]
```

Sets the cutoff frequency or half-way point of the filter. Arguments

- freq - frequency value in Hz. Range: Any positive value.

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

- modules/atone.h
- modules/atone.cpp

4.4 daisysp::Autowah Class Reference

```
#include <autowah.h>
```

Public Member Functions

- void Init (float sample_rate)
- float **Process** (float in)
- void SetWah (float wah)
- void SetDryWet (float drywet)
- void SetLevel (float level)

4.4.1 Detailed Description

Autowah module Original author(s) : Ported from soundpipe by Ben Sergentanis, May 2020

4.4.2 Member Function Documentation

4.4.2.1 Init()

```
void Autowah::Init (
             float sample_rate )
```

Initializes the Autowah module. sample_rate - The sample rate of the audio engine being run.

4.4.2.2 SetDryWet()

```
void daisysp::Autowah::SetDryWet (
             float drywet )  [inline]
```

- float drywet : set effect dry/wet

4.4.2.3 SetLevel()

```
void daisysp::Autowah::SetLevel (
             float level )  [inline]
```

- float level : set wah level

4.4.2.4 SetWah()

```
void daisysp::Autowah::SetWah (
             float wah )  [inline]
```

- float wah : set wah amount

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

- modules/autowah.h
- modules/autowah.cpp

4.5 daisysp::Balance Class Reference

```
#include <balance.h>
```

Public Member Functions

- void Init (float sample_rate)
- float **Process** (float sig, float comp)
- void SetCutoff (float cutoff)

4.5.1 Detailed Description

Balances two sound sources. Sig is boosted to the level of comp. Original author(s) : Barry Vercoe, john ffitch, Gabriel Maldonado Year: 1991 Ported from soundpipe by Ben Sergentanis, May 2020

4.5.2 Member Function Documentation

4.5.2.1 Init()

```
void Balance::Init (
            float sample_rate )
```

Initializes the balance module. sample_rate - The sample rate of the audio engine being run.

4.5.2.2 SetCutoff()

```
void daisysp::Balance::SetCutoff (
            float cutoff )  [inline]
```

- float cutoff : Sets half power point of special internal cutoff filter.

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

- modules/balance.h
- modules/balance.cpp

4.6 daisysp::Biquad Class Reference

```
#include <biquad.h>
```

Public Member Functions

- void Init (float sample_rate)
- float **Process** (float in)
- void SetRes (float res)
- void SetCutoff (float cutoff)

4.6.1 Detailed Description

Two pole recursive filter Original author(s) : Hans Mikelson Year: 1998 Ported from soundpipe by Ben Sergentanis, May 2020

4.6.2 Member Function Documentation

4.6.2.1 Init()

```
void Biquad::Init (
            float sample_rate )
```

Initializes the biquad module. sample_rate - The sample rate of the audio engine being run.

4.6.2.2 SetCutoff()

```
void daisysp::Biquad::SetCutoff (
            float cutoff )  [inline]
```

- float cutoff : Set filter cutoff.

4.6.2.3 SetRes()

```
void daisysp::Biquad::SetRes (
            float res )  [inline]
```

- float res : Set filter resonance.

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

- modules/biquad.h
- modules/biquad.cpp

4.7 daisysp::Bitcrush Class Reference

```
#include <bitcrush.h>
```

Public Member Functions

- void Init (float sample_rate)
- float **Process** (float in)
- void SetBitDepth (int bitdepth)
- void SetCrushRate (float crushrate)

4.7.1 Detailed Description

bitcrush module Original author(s) : Paul Batchelor, Ported from soundpipe by Ben Sergentanis, May 2020

4.7.2 Member Function Documentation

4.7.2.1 Init()

```
void Bitcrush::Init (
            float sample_rate )
```

Initializes the bitcrush module. sample_rate - The sample rate of the audio engine being run.

4.7.2.2 SetBitDepth()

```
void daisysp::Bitcrush::SetBitDepth (
            int bitdepth )  [inline]
```

- int bitdepth : Sets bit depth.

4.7.2.3 SetCrushRate()

```
void daisysp::Bitcrush::SetCrushRate (
            float crushrate )  [inline]
```

- float crushrate : Sets rate to downsample to.

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

- modules/bitcrush.h
- modules/bitcrush.cpp

4.8 daisysp::BlOsc Class Reference

```
#include <blosc.h>
```

Public Types

- enum Waveforms { **WAVE_TRIANGLE**, **WAVE_SAW**, **WAVE_SQUARE**, **WAVE_OFF** }

Public Member Functions

- void Init (float sample_rate)
- float Process ()
- void SetFreq (float freq)
- void SetAmp (float amp)
- void SetPw (float pw)
- void SetWaveform (uint8_t waveform)

4.8.1 Detailed Description

Band Limited Oscillator Based on bltriangle, blsaw, blsquare from soundpipe Original Author(s): Paul Batchelor, saw2 Faust by Julius Smith Ported by Ben Sergentanis, May 2020

4.8.2 Member Enumeration Documentation

4.8.2.1 Waveforms

```
enum daisysp::BlOsc::Waveforms
```

Bl Waveforms

4.8.3 Member Function Documentation

4.8.3.1 Init()

```
void BlOsc::Init (
            float sample_rate )
```

-Initialize oscillator. -Defaults to: 440Hz, .5 amplitude, .5 pw, Triangle.

4.8.3.2 Process()

```
float BlOsc::Process ( )
```

- Get next floating point oscillator sample.

4.8.3.3 SetAmp()

```
void daisysp::BlOsc::SetAmp (
            float amp )  [inline]
```

- Float amp: Set oscillator amplitude, 0 to 1.

4.8.3.4 SetFreq()

```
void daisysp::BlOsc::SetFreq (
            float freq )  [inline]
```

- Float freq: Set oscillator frequency in Hz.

4.8.3.5 SetPw()

```
void daisysp::BlOsc::SetPw (
            float pw )  [inline]
```

- Float pw: Set square osc pulsewidth, 0 to 1. (no thru 0 at the moment)

4.8.3.6 SetWaveform()

```
void daisysp::BlOsc::SetWaveform (
            uint8_t waveform )  [inline]
```

- uint8_t waveform: select between waveforms from enum above.
- i.e. SetWaveform(BL_WAVEFORM_SAW); to set waveform to saw

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

- modules/blosc.h
- modules/blosc.cpp

4.9 daisysp::Comb Class Reference

```
#include <comb.h>
```

Public Member Functions

- void Init (float sample_rate, float ∗buff, size_t size)
- float **Process** (float in)
- void **SetFreq** (float looptime)
- void **SetRevTime** (float revtime)

4.9.1 Detailed Description

Comb filter module Original author(s) : Ported from soundpipe by Ben Sergentanis, May 2020

4.9.2 Member Function Documentation

4.9.2.1 Init()

```
void Comb::Init (
            float sample_rate,
            float * buff,
            size_t size )
```

Initializes the Comb module. sample_rate - The sample rate of the audio engine being run.

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

- modules/comb.h
- modules/comb.cpp

4.10 daisysp::Compressor Class Reference

```
#include <compressor.h>
```

Public Member Functions

- void Init (float sample_rate)
- float Process (float in, float key)
- float **Process** (float in)
- void SetRatio (const float &ratio)
- void SetThreshold (const float &thresh)
- void SetAttack (const float &atk)
- void SetRelease (const float &rel)

4.10.1 Detailed Description

influenced by compressor in soundpipe (from faust). Modifications made to do:

- Less calculations during each process loop (coefficients recalculated on parameter change).
- C++-ified
- added sidechain support by: shensley

Todo With fixed controls this is relatively quick, but changing controls now costs a lot more

Still pretty expensive

Add soft/hard knee settings

Maybe make stereo possible? (needing two for stereo is a bit silly, and their gain shouldn't be totally unique.

4.10.2 Member Function Documentation

4.10.2.1 Init()

```
void Compressor::Init (
            float sample_rate )
```

Initializes compressor sample_rate - rate at which samples will be produced by the audio engine.

4.10.2.2 Process()

```
float Compressor::Process (
            float in,
            float key )
```

compresses the audio input signal, either keyed by itself, or a secondary input. in - audio input signal (to be compressed) (optional) key - audio input that will be used to side-chain the compressor.

4.10.2.3 SetAttack()

```
void daisysp::Compressor::SetAttack (
            const float & atk )  [inline]
```

envelope time for onset of compression for signals above the threshold. Expects 0.001 -> 10

4.10.2.4 SetRatio()

```
void daisysp::Compressor::SetRatio (
            const float & ratio )  [inline]
```

amount of gain reduction applied to compressed signals Expects 1.0 -> 40. (untested with values < 1.0)

4.10.2.5 SetRelease()

```
void daisysp::Compressor::SetRelease (
            const float & rel )  [inline]
```

envelope time for release of compression as input signal falls below threshold. Expects 0.001 -> 10

4.10.2.6 SetThreshold()

```
void daisysp::Compressor::SetThreshold (
            const float & thresh )  [inline]
```

threshold in dB at which compression will be applied Expects 0.0 -> -80.

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

- modules/compressor.h
- modules/compressor.cpp

4.11 daisysp::CrossFade Class Reference

```
#include <crossfade.h>
```

Public Member Functions

- void Init (int curve)
- void **Init** ()
- float Process (float &in1, float &in2)
- void SetPos (float pos)
- void SetCurve (uint8_t curve)
- float GetPos (float pos)
- uint8_t GetCurve (uint8_t curve)

4.11.1 Detailed Description

Performs a CrossFade between two signals Original author: Paul Batchelor Ported from Soundpipe by Andrew Ikenberry added curve option for constant power, etc.

4.11.2 Member Function Documentation

4.11.2.1 GetCurve()

```
uint8_t daisysp::CrossFade::GetCurve (
            uint8_t curve )  [inline]
```

Returns current curve

4.11.2.2 GetPos()

```
float daisysp::CrossFade::GetPos (
            float pos ) [inline]
```

Returns current position

4.11.2.3 Init()

```
void daisysp::CrossFade::Init (
            int curve ) [inline]
```

Initializes CrossFade module Defaults

- current position = .5
- curve = linear

4.11.2.4 Process()

```
float CrossFade::Process (
            float & in1,
            float & in2 )
```

processes CrossFade and returns single sample

4.11.2.5 SetCurve()

```
void daisysp::CrossFade::SetCurve (
            uint8_t curve ) [inline]
```

Sets current curve applied to CrossFade Expected input: See `Curve Options`

4.11.2.6 SetPos()

```
void daisysp::CrossFade::SetPos (
            float pos ) [inline]
```

Sets position of CrossFade between two input signals Input range: 0 to 1

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

- modules/crossfade.h
- modules/crossfade.cpp

4.12 daisysp::DcBlock Class Reference

```
#include <dcblock.h>
```

Public Member Functions

- void Init (float sample_rate)
- float Process (float in)

4.12.1 Detailed Description

Removes DC component of a signal

4.12.2 Member Function Documentation

4.12.2.1 Init()

```
void DcBlock::Init (
            float sample_rate )
```

Initializes DcBlock module

4.12.2.2 Process()

```
float DcBlock::Process (
            float in )
```

performs DcBlock Process

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

- modules/dcblock.h
- modules/dcblock.cpp

4.13 daisysp::Decimator Class Reference

```
#include <decimator.h>
```

Public Member Functions

- void Init ()
- float Process (float input)
- void SetDownsampleFactor (float downsample_factor)
- void SetBitcrushFactor (float bitcrush_factor)
- void SetBitsToCrush (const uint8_t &bits)
- float GetDownsampleFactor ()
- float GetBitcrushFactor ()

4.13.1 Detailed Description

Performs downsampling and bitcrush effects

4.13.2 Member Function Documentation

4.13.2.1 GetBitcrushFactor()

```
float daisysp::Decimator::GetBitcrushFactor ( )  [inline]
```

Returns current setting of bitcrush

4.13.2.2 GetDownsampleFactor()

```
float daisysp::Decimator::GetDownsampleFactor ( )  [inline]
```

Returns current setting of downsample

4.13.2.3 Init()

```
void Decimator::Init ( )
```

Initializes downsample module

4.13.2.4 Process()

```
float Decimator::Process (
             float input )
```

Applies downsample and bitcrush effects to input signal. Returns one sample. This should be called once per sample period.

4.13.2.5 SetBitcrushFactor()

```
void daisysp::Decimator::SetBitcrushFactor (
            float bitcrush_factor )  [inline]
```

Sets amount of bitcrushing Input range:

4.13.2.6 SetBitsToCrush()

```
void daisysp::Decimator::SetBitsToCrush (
            const uint8_t & bits )  [inline]
```

Sets the exact number of bits to crush 0-16 bits

4.13.2.7 SetDownsampleFactor()

```
void daisysp::Decimator::SetDownsampleFactor (
            float downsample_factor )  [inline]
```

Sets amount of downsample Input range:

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

- modules/decimator.h
- modules/decimator.cpp

4.14 daisysp::Fold Class Reference

```
#include <fold.h>
```

Public Member Functions

- void Init ()
- float **Process** (float in)
- void SetIncrement (float incr)

4.14.1 Detailed Description

fold module Original author(s) : John FFitch, Gabriel Maldonado Year : 1998 Ported from soundpipe by Ben Sergentanis, May 2020

4.14.2 Member Function Documentation

4.14.2.1 Init()

```
void Fold::Init ( )
```

Initializes the fold module.

4.14.2.2 SetIncrement()

```
void daisysp::Fold::SetIncrement (
            float incr )  [inline]
```

- float incr : set fold increment

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

- modules/fold.h
- modules/fold.cpp

4.15 daisysp::Limiter Class Reference

Public Member Functions

- void Init ()
- void ProcessBlock (float ∗in, size_t size, float pre_gain)

4.15.1 Member Function Documentation

4.15.1.1 Init()

```
void Limiter::Init ( )
```

Initializes the Limiter instance.

4.15.1.2 ProcessBlock()

```
void Limiter::ProcessBlock (
            float * in,
            size_t size,
            float pre_gain )
```

Processes a block of audio through the limiter. in - pointer to a block of audio samples to be processed. The buffer is operated on directly. size - size of the buffer "in" pre_gain - amount of pre_gain applied to the signal.

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

- modules/limiter.h
- modules/limiter.cpp

4.16 daisysp::Line Class Reference

```
#include <line.h>
```

Public Member Functions

- void Init (float sample_rate)
- float Process (uint8_t *finished)
- void Start (float start, float end, float dur)

4.16.1 Detailed Description

creates a Line segment signal

4.16.2 Member Function Documentation

4.16.2.1 Init()

```
void Line::Init (
            float sample_rate )
```

Initializes Line module.

4.16.2.2 Process()

```
float Line::Process (
            uint8_t * finished )
```

Processes Line segment. Returns one sample. value of finished will be updated to a 1, upon completion of the Line's trajectory.

4.16.2.3 Start()

```
void Line::Start (
            float start,
            float end,
            float dur )
```

Begin creation of Line. Arguments:

- start - beginning value
- end - ending value
- dur - duration in seconds of Line segment

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

- modules/line.h
- modules/line.cpp

4.17 daisysp::Maytrig Class Reference

```
#include <maytrig.h>
```

Public Member Functions

- float Process (float prob)

4.17.1 Detailed Description

Probabilistic trigger module Original author(s) : Paul Batchelor Ported from soundpipe by Ben Sergentanis, May 2020

4.17.2 Member Function Documentation

4.17.2.1 Process()

```
float daisysp::Maytrig::Process (
            float prob ) [inline]
```

- Returns given a probability 0 to 1, returns true or false. (1 always returns true, 0 always false)

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

- modules/maytrig.h

4.18 daisysp::Metro Class Reference

```
#include <metro.h>
```

Public Member Functions

- void Init (float freq, float sample_rate)
- uint8_t Process ()
- void Reset ()
- void SetFreq (float freq)
- float GetFreq ()

4.18.1 Detailed Description

Creates a clock signal at a specific frequency.

4.18.2 Member Function Documentation

4.18.2.1 GetFreq()

```
float daisysp::Metro::GetFreq ( )  [inline]
```

Returns current value for frequency.

4.18.2.2 Init()

```
void Metro::Init (
            float freq,
            float sample_rate )
```

Initializes Metro module. Arguments:

- freq: frequency at which new clock signals will be generated Input Range:
- sample_rate: sample rate of audio engine Input range:

4.18.2.3 Process()

```
uint8_t Metro::Process ( )
```

checks current state of Metro object and updates state if necesary.

4.18.2.4 Reset()

```
void daisysp::Metro::Reset ( )  [inline]
```

resets phase to 0

4.18.2.5 SetFreq()

```
void Metro::SetFreq (
            float freq )
```

Sets frequency at which Metro module will run at.

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

- modules/metro.h
- modules/metro.cpp

4.19 daisysp::Mode Class Reference

```
#include <mode.h>
```

Public Member Functions

- void Init (float sample_rate)
- float Process (float in)
- void Clear ()
- void SetFreq (float freq)
- void SetQ (float q)

4.19.1 Detailed Description

Resonant Modal Filter Extracted from soundpipe to work as a Daisy Module, originally extracted from csound by Paul Batchelor. Original Author(s): Francois Blanc, Steven Yi Year: 2001 Location: Opcodes/biquad.c (csound)

4.19.2 Member Function Documentation

4.19.2.1 Clear()

```
void Mode::Clear ( )
```

Clears the filter, returning the output to 0.0

4.19.2.2 Init()

```
void Mode::Init (
            float sample_rate )
```

Initializes the instance of the module. sample_rate: frequency of the audio engine in Hz

4.19.2.3 Process()

```
float Mode::Process (
            float in )
```

Processes one input sample through the filter, and returns the output.

4.19.2.4 SetFreq()

```
void daisysp::Mode::SetFreq (
            float freq ) [inline]
```

Sets the resonant frequency of the modal filter. Range: Any frequency such that sample_rate / freq < PI (about 15.2kHz at 48kHz)

4.19.2.5 SetQ()

```
void daisysp::Mode::SetQ (
            float q )  [inline]
```

Sets the quality factor of the filter. Range: Positive Numbers (Good values range from 70 to 1400)

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

- modules/mode.h
- modules/mode.cpp

4.20 daisysp::MoogLadder Class Reference

```
#include <moogladder.h>
```

Public Member Functions

- void Init (float sample_rate)
- float **Process** (float in)
- void SetFreq (float freq)
- void SetRes (float res)

4.20.1 Detailed Description

Moog ladder filter module Original author(s) : Victor Lazzarini, John ffitch (fast tanh), Bob Moog

4.20.2 Member Function Documentation

4.20.2.1 Init()

```
void MoogLadder::Init (
            float sample_rate )
```

Initializes the MoogLadder module. sample_rate - The sample rate of the audio engine being run.

4.20.2.2 SetFreq()

```
void daisysp::MoogLadder::SetFreq (
            float freq )  [inline]
```

Sets the cutoff frequency or half-way point of the filter. Arguments

- freq - frequency value in Hz. Range: Any positive value.

4.20.2.3 SetRes()

```
void daisysp::MoogLadder::SetRes (
            float res ) [inline]
```

Sets the resonance of the filter.

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

- modules/moogladder.h
- modules/moogladder.cpp

4.21 daisysp::NlFilt Class Reference

```
#include <nlfilt.h>
```

Public Member Functions

- void Init ()
- void ProcessBlock (float *in, float *out, size_t size)
- void SetCoefficients (float a, float b, float d, float C, float L)
- void **SetA** (float a)
- void **SetB** (float b)
- void **SetD** (float d)
- void **SetC** (float C)
- void **SetL** (float L)

4.21.1 Detailed Description

port by: Stephen Hensley, December 2019 Non-linear filter. The four 5-coefficients: a, b, d, C, and L are used to configure different filter types. Structure for Dobson/Fitch nonlinear filter Revised Formula from Risto Holopainen 12 Mar 2004 `Y{n} =tanh(a Y{n-1} + b Y{n-2} + d Y^2{n-L} + X{n} - C)` Though traditional filter types can be made, the effect will always respond differently to different input. This Source is a heavily modified version of the original source from Csound.

Todo make this work on a single sample instead of just on blocks at a time.

4.21.2 Member Function Documentation

4.21.2.1 Init()

```
void NlFilt::Init ( )
```

Initializes the NlFilt object.

4.21.2.2 ProcessBlock()

```
void NlFilt::ProcessBlock (
            float * in,
            float * out,
            size_t size )
```

Process the array pointed to by ∗in and updates the output to ∗out; This works on a block of audio at once, the size of which is set with the size.

4.21.2.3 SetCoefficients()

```
void daisysp::NlFilt::SetCoefficients (
            float a,
            float b,
            float d,
            float C,
            float L )  [inline]
```

inputs these are the five coefficients for the filter.

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

- modules/nlfilt.h
- modules/nlfilt.cpp

4.22 daisysp::Oscillator Class Reference

```
#include <oscillator.h>
```

Public Types

- enum {
  WAVE_SIN, **WAVE_TRI**, **WAVE_SAW**, **WAVE_RAMP**,
  WAVE_SQUARE, **WAVE_POLYBLEP_TRI**, **WAVE_POLYBLEP_SAW**, **WAVE_POLYBLEP_SQUARE**,
  WAVE_LAST }

Public Member Functions

- void Init (float sample_rate)
- void SetFreq (const float f)
- void SetAmp (const float a)
- void SetWaveform (const uint8_t wf)
- float Process ()
- void PhaseAdd (float _phase)
- void Reset (float _phase=0.0f)

4.22.1 Detailed Description

Synthesis of several waveforms, including polyBLEP bandlimited waveforms.

4.22.2 Member Enumeration Documentation

4.22.2.1 anonymous enum

```
anonymous enum
```

Choices for output waveforms, POLYBLEP are appropriately labeled. Others are naive forms.

4.22.3 Member Function Documentation

4.22.3.1 Init()

```
void daisysp::Oscillator::Init (
            float sample_rate )  [inline]
```

Initializes the Oscillator float sample_rate - sample rate of the audio engine being run, and the frequency that the Process function will be called. Defaults:

- freq_ = 100 Hz
- amp_ = 0.5
- waveform_ = sine wave.

4.22.3.2 PhaseAdd()

```
void daisysp::Oscillator::PhaseAdd (
            float _phase )  [inline]
```

Adds a value 0.0-1.0 (mapped to 0.0-TWO_PI) to the current phase. Useful for PM and "FM" synthesis.

4.22.3.3 Process()

```
float Oscillator::Process ( )
```

Processes the waveform to be generated, returning one sample. This should be called once per sample period.

4.22.3.4 Reset()

```
void daisysp::Oscillator::Reset (
            float _phase = 0.0f )  [inline]
```

Resets the phase to the input argument. If no argument is present, it will reset phase to 0.0;

4.22.3.5 SetAmp()

```
void daisysp::Oscillator::SetAmp (
            const float a )  [inline]
```

Sets the amplitude of the waveform.

4.22.3.6 SetFreq()

```
void daisysp::Oscillator::SetFreq (
            const float f )  [inline]
```

Changes the frequency of the Oscillator, and recalculates phase increment.

4.22.3.7 SetWaveform()

```
void daisysp::Oscillator::SetWaveform (
            const uint8_t wf )  [inline]
```

Sets the waveform to be synthesized by the Process() function.

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

- modules/oscillator.h
- modules/oscillator.cpp

4.23 daisysp::Phasor Class Reference

```
#include <phasor.h>
```

Public Member Functions

- void Init (float sample_rate, float freq, float initial_phase)
- void **Init** (float sample_rate, float freq)
- void **Init** (float sample_rate)
- float Process ()
- void SetFreq (float freq)
- float GetFreq ()

4.23.1 Detailed Description

Generates a normalized signal moving from 0-1 at the specified frequency.

Todo Selecting which channels should be initialized/included in the sequence conversion.

Setup a similar start function for an external mux, but that seems outside the scope of this file.

4.23.2 Member Function Documentation

4.23.2.1 GetFreq()

```
float daisysp::Phasor::GetFreq ( )  [inline]
```

Returns current frequency value in Hz

4.23.2.2 Init()

```
void daisysp::Phasor::Init (
            float sample_rate,
            float freq,
            float initial_phase )  [inline]
```

Initializes the Phasor module sample rate, and freq are in Hz initial phase is in radians Additional Init functions have defaults when arg is not specified:

- phs = 0.0f
- freq = 1.0f

4.23.2.3 Process()

```
float Phasor::Process ( )
```

processes Phasor and returns current value

4.23.2.4 SetFreq()

```
void Phasor::SetFreq (
            float freq )
```

Sets frequency of the Phasor in Hz

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

- modules/phasor.h
- modules/phasor.cpp

4.24 daisysp::PitchShifter Class Reference

```
#include <pitchshifter.h>
```

Public Member Functions

- void **Init** (float sr)
- float Process (float &in)
- void **SetTransposition** (const float &transpose)
- void **SetDelSize** (uint32_t size)
- void **SetFun** (float f)

4.24.1 Detailed Description

From ucsd.edu "Pitch Shifting" t = 1 - ((s ∗f) / R) where: s is the size of the delay f is the frequency of the lfo r is the sample_rate solving for t = 12.0 f = (12 - 1) ∗ 48000 / SHIFT_BUFFER_SIZE;

4.24.2 Member Function Documentation

4.24.2.1 Process()

```
float daisysp::PitchShifter::Process (
            float & in )  [inline]
```

First Process delay mod/crossfade

Handle Delay Writing

Modulate Delay Lines mod_a_amt = mod_b_amt = 0.0f; d_[0].SetDelay(mod_[0] + mod_a_amt_); d_[1].Set↵Delay(mod_[1] + mod_b_amt_);

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

- modules/pitchshifter.h

4.25 daisysp::Pluck Class Reference

```
#include <pluck.h>
```

Public Member Functions

- void Init (float sample_rate, float *buf, int32_t npt, int32_t mode)
- float Process (float &trig)
- void SetAmp (float amp)
- void SetFreq (float freq)
- void SetDecay (float decay)
- void SetDamp (float damp)
- void SetMode (int32_t mode)
- float GetAmp ()
- float GetFreq ()
- float GetDecay ()
- float GetDamp ()
- int32_t GetMode ()

4.25.1 Detailed Description

Produces a naturally decaying plucked string or drum sound based on the Karplus-Strong algorithms. This code has been extracted from the Csound opcode "pluck" It has been modified to work as a Daisy Soundpipe module. Original Author(s): Barry Vercoe, John ffitch Year: 1991 Location: OOps/ugens4.c

4.25.2 Member Function Documentation

4.25.2.1 GetAmp()

```
float daisysp::Pluck::GetAmp ( )  [inline]
```

Returns the current value for amp.

4.25.2.2 GetDamp()

```
float daisysp::Pluck::GetDamp ( )  [inline]
```

Returns the current value for damp.

4.25.2.3 GetDecay()

```
float daisysp::Pluck::GetDecay ( )  [inline]
```

Returns the current value for decay.

4.25.2.4 GetFreq()

```
float daisysp::Pluck::GetFreq ( )  [inline]
```

Returns the current value for freq.

4.25.2.5 GetMode()

```
int32_t daisysp::Pluck::GetMode ( ) [inline]
```

Returns the current value for mode.

4.25.2.6 Init()

```
void Pluck::Init (
            float sample_rate,
            float * buf,
            int32_t npt,
            int32_t mode )
```

Initializes the Pluck module. Arguments:

- sample_rate: Sample rate of the audio engine being run.
- buf: buffer used as an impulse when triggering the Pluck algorithm
- npt: number of elementes in buf.
- mode: Sets the mode of the algorithm.

4.25.2.7 Process()

```
float Pluck::Process (
            float & trig )
```

Processes the waveform to be generated, returning one sample. This should be called once per sample period.

4.25.2.8 SetAmp()

```
void daisysp::Pluck::SetAmp (
            float amp ) [inline]
```

Sets the amplitude of the output signal. Input range: 0-1?

4.25.2.9 SetDamp()

```
void daisysp::Pluck::SetDamp (
            float damp ) [inline]
```

Sets the dampening factor applied by the filter (based on PLUCK_MODE) Input range: 0-1

4.25.2.10 SetDecay()

```
void daisysp::Pluck::SetDecay (
            float decay )  [inline]
```

Sets the time it takes for a triggered note to end in seconds. Input range: 0-1

4.25.2.11 SetFreq()

```
void daisysp::Pluck::SetFreq (
            float freq )  [inline]
```

Sets the frequency of the output signal in Hz. Input range: Any positive value

4.25.2.12 SetMode()

```
void daisysp::Pluck::SetMode (
            int32_t mode )  [inline]
```

Sets the mode of the algorithm.

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

- modules/pluck.h
- modules/pluck.cpp

4.26 daisysp::PolyPluck< num_voices > Class Template Reference

```
#include <PolyPluck.h>
```

Public Member Functions

- void Init (float sample_rate)
- float Process (float &trig, float note)
- void SetDecay (float p)

4.26.1 Detailed Description

template<size_t num_voices>
class daisysp::PolyPluck< num_voices >

Simplified Pseudo-Polyphonic Pluck Voice Template Based Pluck Voice, with configurable number of voices and simple pseudo-polyphony. DC Blocking included to prevent biases from causing unwanted saturation distortion. Author∗∗: shensley Date Added∗∗: March 2020

4.26.2 Member Function Documentation

4.26.2.1 Init()

```
template<size_t num_voices>
void daisysp::PolyPluck< num_voices >::Init (
            float sample_rate )  [inline]
```

Initializes the PolyPluck instance. Arguments:

- sample_rate: rate in Hz that the Process() function will be called.

4.26.2.2 Process()

```
template<size_t num_voices>
float daisysp::PolyPluck< num_voices >::Process (
            float & trig,
            float note )  [inline]
```

Process function, synthesizes and sums the output of all voices, triggering a new voice with frequency of MIDI note number when trig > 0. Arguments:

- float &trig: value by reference of trig. When trig > 0 a the next voice will be triggered, and trig will be set to 0.
- float note: MIDI note number for the active_voice.

increment active voice

set new voice to new note

4.26.2.3 SetDecay()

```
template<size_t num_voices>
void daisysp::PolyPluck< num_voices >::SetDecay (
            float p )  [inline]
```

Sets the decay coefficients of the pluck voices. Expects 0.0-1.0 input.

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

- modules/PolyPluck.h

4.27 daisysp::Port Class Reference

```
#include <port.h>
```

Public Member Functions

- void Init (float sample_rate, float htime)
- float Process (float in)
- void SetHtime (float htime)
- float GetHtime ()

4.27.1 Detailed Description

Applies portamento to an input signal. At each new step value, the input is low-pass filtered to move towards that value at a rate determined by ihtim. ihtim is the half-time of the function (in seconds), during which the curve will traverse half the distance towards the new value, then half as much again, etc., theoretically never reaching its asymptote. This code has been ported from Soundpipe to DaisySP by Paul Batchelor. The Soundpipe module was extracted from the Csound opcode "portk". Original Author(s): Robbin Whittle, John ffitch Year: 1995, 1998 Location: Opcodes/biquad.c

4.27.2 Member Function Documentation

4.27.2.1 GetHtime()

```
float daisysp::Port::GetHtime ( )  [inline]
```

returns current value of htime

4.27.2.2 Init()

```
void Port::Init (
            float sample_rate,
            float htime )
```

Initializes Port module Arguments:

- sample_rate: sample rate of audio engine
- htime: half-time of the function, in seconds.

4.27.2.3 Process()

```
float Port::Process (
            float in )
```

Applies portamento to input signal and returns processed signal.

4.27.2.4 SetHtime()

```
void daisysp::Port::SetHtime (
            float htime )  [inline]
```

Sets htime

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

- modules/port.h
- modules/port.cpp

4.28 daisysp::ReverbSc Class Reference

```
#include <reverbsc.h>
```

Public Member Functions

- int Init (float sample_rate)
- int Process (const float &in1, const float &in2, float ∗out1, float ∗out2)
- void SetFeedback (const float &fb)
- void SetLpFreq (const float &freq)

4.28.1 Detailed Description

Stereo Reverb Reverb SC: Ported from csound/soundpipe Original author(s): Sean Costello, Istvan Varga Year: 1999, 2005 Ported to soundpipe by: Paul Batchelor Ported by: Stephen Hensley

4.28.2 Member Function Documentation

4.28.2.1 Init()

```
int ReverbSc::Init (
            float sample_rate )
```

Initializes the reverb module, and sets the sample_rate at which the Process function will be called. Returns 0 if all good, or 1 if it runs out of delay times exceed maximum allowed.

4.28.2.2 Process()

```
int ReverbSc::Process (
            const float & in1,
            const float & in2,
            float * out1,
            float * out2 )
```

Process the input through the reverb, and updates values of out1, and out2 with the new processed signal.

4.28.2.3 SetFeedback()

```
void daisysp::ReverbSc::SetFeedback (
            const float & fb )  [inline]
```

controls the reverb time. reverb tail becomes infinite when set to 1.0 range: 0.0 to 1.0

4.28.2.4 SetLpFreq()

```
void daisysp::ReverbSc::SetLpFreq (
            const float & freq )  [inline]
```

controls the internal dampening filter's cutoff frequency. range: 0.0 to sample_rate / 2

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

- modules/reverbsc.h
- modules/reverbsc.cpp

4.29 daisysp::ReverbScDl Struct Reference

Public Attributes

- int **write_pos**
- int **buffer_size**
- int **read_pos**
- int **read_pos_frac**
- int **read_pos_frac_inc**
- int **dummy**
- int **seed_val**
- int **rand_line_cnt**
- float **filter_state**
- float ∗ **buf**

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

- modules/reverbsc.h

4.30 daisysp::Svf Class Reference

```
#include <svf.h>
```

Public Member Functions

- void Init (float sample_rate)
- void Process (float in)
- void SetFreq (float f)
- void SetRes (float r)
- void SetDrive (float d)
- float **Low** ()
- float **High** ()
- float **Band** ()
- float **Notch** ()
- float **Peak** ()

4.30.1 Detailed Description

Double Sampled, Stable State Variable Filter Credit to Andrew Simper from musicdsp.org This is his "State Variable Filter (Double Sampled, Stable)" Additional thanks to Laurent de Soras for stability limit, and Stefan Diedrichsen for the correct notch output Ported by: Stephen Hensley example: daisysp/examples/Svf/

4.30.2 Member Function Documentation

4.30.2.1 Init()

```
void Svf::Init (
            float sample_rate )
```

Initializes the filter float sample_rate - sample rate of the audio engine being run, and the frequency that the Process function will be called.

4.30.2.2 Process()

```
void Svf::Process (
            float in )
```

Process the input signal, updating all of the outputs.

4.30.2.3 SetDrive()

```
void daisysp::Svf::SetDrive (
            float d )  [inline]
```

sets the drive of the filter, affecting the response of the resonance of the filter..

4.30.2.4 SetFreq()

```
void Svf::SetFreq (
            float f )
```

sets the frequency of the cutoff frequency. f must be between 0.0 and sample_rate / 2

4.30.2.5 SetRes()

```
void Svf::SetRes (
            float r )
```

sets the resonance of the filter. Must be between 0.0 and 1.0 to ensure stability.

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

- modules/svf.h
- modules/svf.cpp

4.31 daisysp::Tone Class Reference

```
#include <tone.h>
```

Public Member Functions

- void Init (float sample_rate)
- float Process (float &in)
- void SetFreq (float &freq)
- float GetFreq ()

4.31.1 Detailed Description

A first-order recursive low-pass filter with variable frequency response.

4.31.2 Member Function Documentation

4.31.2.1 GetFreq()

```
float daisysp::Tone::GetFreq ( ) [inline]
```

Returns the current value for the cutoff frequency or half-way point of the filter.

4.31.2.2 Init()

```
void Tone::Init (
            float sample_rate )
```

Initializes the Tone module. sample_rate - The sample rate of the audio engine being run.

4.31.2.3 Process()

```
float Tone::Process (
            float & in )
```

Processes one sample through the filter and returns one sample. in - input signal

4.31.2.4 SetFreq()

```
void daisysp::Tone::SetFreq (
            float & freq )  [inline]
```

Sets the cutoff frequency or half-way point of the filter. Arguments

- freq - frequency value in Hz. Range: Any positive value.

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

- modules/tone.h
- modules/tone.cpp

4.32 daisysp::WhiteNoise Class Reference

```
#include <whitenoise.h>
```

Public Member Functions

- void Init ()
- void SetAmp (float a)
- float Process ()

4.32.1 Detailed Description

fast white noise generator I think this came from musicdsp.org at some point

4.32.2 Member Function Documentation

4.32.2.1 Init()

```
void daisysp::WhiteNoise::Init ( )  [inline]
```

Initializes the WhiteNoise object

4.32.2.2 Process()

```
float daisysp::WhiteNoise::Process ( )  [inline]
```

returns a new sample of noise in the range of -amp_ to amp_

4.32.2.3 SetAmp()

```
void daisysp::WhiteNoise::SetAmp (
             float a )  [inline]
```

sets the amplitude of the noise output

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

- modules/whitenoise.h

Index