Loopy Tunes

AUTHOR
Version v1.0

Table of Contents

Table of contents

LoopyTunes

Author

Description

Namespace Index

Namespace List

Here is a list of all namespaces with brief descriptions:

Class Index

Class List

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

File Index

File List

Here is a list of all files with brief descriptions:

Namespace Documentation

Buffers Namespace Reference

Variables

- float DSY_SDRAM_BSS **track1** [2][**SAMPLERATE** ***DURATION**]
- float DSY_SDRAM_BSS **track2** [2][**SAMPLERATE** ***DURATION**]
- float DSY_SDRAM_BSS **track3** [2][**SAMPLERATE** ***DURATION**]
- float DSY_SDRAM_BSS **track4** [2][**SAMPLERATE** ***DURATION**]
- float * **track1Ptr** [2] = {**track1**[**L**], **track1**[**R**]}
- float * **track2Ptr** [2] = {**track2**[**L**], **track2**[**R**]}
- float * **track3Ptr** [2] = {**track3**[**L**], **track3**[**R**]}
- float * **track4Ptr** [2] = {**track4**[**L**], **track4**[**R**]}
- float DSY_SDRAM_BSS **mix** [2][**BLOCKLENGTH**]
- float DSY_SDRAM_BSS **t1m** [2][**BLOCKLENGTH**]
- float DSY_SDRAM_BSS **t2m** [2][**BLOCKLENGTH**]
- float DSY_SDRAM_BSS **t3m** [2][**BLOCKLENGTH**]
- float DSY_SDRAM_BSS **t4m** [2][**BLOCKLENGTH**]
- float * **mixPtr** [2] = {**mix**[**L**], **mix**[**R**]}
- float * **t1mPtr** [2] = {**t1m**[**L**], **t1m**[**R**]}
- float * **t2mPtr** [2] = {**t2m**[**L**], **t2m**[**R**]}
- float * **t3mPtr** [2] = {**t3m**[**L**], **t3m**[**R**]}
- float * **t4mPtr** [2] = {**t4m**[**L**], **t4m**[**R**]}
- DelayLine< float, **MAXDELAY** > DSY_SDRAM_BSS **t1delay** [2]
- DelayLine< float, **MAXDELAY** > DSY_SDRAM_BSS **t2delay** [2]
- DelayLine< float, **MAXDELAY** > DSY_SDRAM_BSS **t3delay** [2]
- DelayLine< float, **MAXDELAY** > DSY_SDRAM_BSS **t4delay** [2]
- DelayLine< float, **MAXDELAY** > * **t1delayPtr** [2] = {&**t1delay**[**L**], &**t1delay**[**R**]}
- DelayLine< float, **MAXDELAY** > * **t2delayPtr** [2] = {&**t2delay**[**L**], &**t2delay**[**R**]}
- DelayLine< float, **MAXDELAY** > * **t3delayPtr** [2] = {&**t3delay**[**L**], &**t3delay**[**R**]}
- DelayLine< float, **MAXDELAY** > * **t4delayPtr** [2] = {&**t4delay**[**L**], &**t4delay**[**R**]}

Variable Documentation

float DSY_SDRAM_BSS Buffers::mix[2][BLOCKLENGTH]

float* Buffers::mixPtr[2] = {mix[L], mix[R]}

DelayLine<float, MAXDELAY> DSY_SDRAM_BSS Buffers::t1delay[2]

DelayLine<float, MAXDELAY>* Buffers::t1delayPtr[2] = {&t1delay[L], &t1delay[R]}

float DSY_SDRAM_BSS Buffers::t1m[2][BLOCKLENGTH]

float* Buffers::t1mPtr[2] = {t1m[L], t1m[R]}

DelayLine<float, MAXDELAY> DSY_SDRAM_BSS Buffers::t2delay[2]

DelayLine<float, MAXDELAY>* Buffers::t2delayPtr[2] = {&t2delay[L], &t2delay[R]}

float DSY_SDRAM_BSS Buffers::t2m[2][BLOCKLENGTH]

float* Buffers::t2mPtr[2] = {t2m[L], t2m[R]}

DelayLine<float, MAXDELAY> DSY_SDRAM_BSS Buffers::t3delay[2]

DelayLine<float, MAXDELAY>* Buffers::t3delayPtr[2] = {&t3delay[L], &t3delay[R]}

float DSY_SDRAM_BSS Buffers::t3m[2][BLOCKLENGTH]

float* Buffers::t3mPtr[2] = {t3m[L], t3m[R]}

DelayLine<float, MAXDELAY> DSY_SDRAM_BSS Buffers::t4delay[2]

DelayLine<float, MAXDELAY>* Buffers::t4delayPtr[2] = {&t4delay[L], &t4delay[R]}

float DSY_SDRAM_BSS Buffers::t4m[2][BLOCKLENGTH]

float* Buffers::t4mPtr[2] = {t4m[L], t4m[R]}

float DSY_SDRAM_BSS Buffers::track1[2][SAMPLERATE *DURATION]

float* Buffers::track1Ptr[2] = {track1[L], track1[R]}

float DSY_SDRAM_BSS Buffers::track2[2][SAMPLERATE *DURATION]

float* Buffers::track2Ptr[2] = {track2[L], track2[R]}

float DSY_SDRAM_BSS Buffers::track3[2][SAMPLERATE *DURATION]

float* Buffers::track3Ptr[2] = {track3[L], track3[R]}

float DSY_SDRAM_BSS Buffers::track4[2][SAMPLERATE *DURATION]

float* Buffers::track4Ptr[2] = {track4[L], track4[R]}

ParameterIDs Namespace Reference

Namespaces

- namespace **Delay**
- namespace **Filter**
- namespace **PitchShifter**
- namespace **Reverb**
- namespace **Tracks**
- namespace **Waveshaper**

ParameterIDs::Delay Namespace Reference

Variables

- const int **effect** = 40
- const int **amount** = **effect** + 1
- const int **size** = **effect** + 2
- const int **feedback** = **effect** + 3

Variable Documentation

const int ParameterIDs::Delay::amount = effect + 1

const int ParameterIDs::Delay::effect = 40

const int ParameterIDs::Delay::feedback = effect + 3

const int ParameterIDs::Delay::size = effect + 2

ParameterIDs::Filter Namespace Reference

Variables

- const int **effect** = 30
- const int **mode** = **effect** + 1
- const int **frequency** = **effect** + 2
- const int **resonance** = **effect** + 3

Variable Documentation

const int ParameterIDs::Filter::effect = 30

const int ParameterIDs::Filter::frequency = effect + 2

const int ParameterIDs::Filter::mode = effect + 1

const int ParameterIDs::Filter::resonance = effect + 3

ParameterIDs::PitchShifter Namespace Reference

Variables

- const int **effect** = 10
- const int **amount** = **effect** + 1
- const int **semitones** = **effect** + 2
- const int **random** = **effect** + 3

Variable Documentation

const int ParameterIDs::PitchShifter::amount = effect + 1

const int ParameterIDs::PitchShifter::effect = 10

const int ParameterIDs::PitchShifter::random = effect + 3

const int ParameterIDs::PitchShifter::semitones = effect + 2

ParameterIDs::Reverb Namespace Reference

Variables

- const int **effect** = 50
- const int **amount** = **effect** + 1
- const int **mode** = **effect** + 2
- const int **size** = **effect** + 3
- const int **damp** = **effect** + 4
- const int **width** = **effect** + 5

Variable Documentation

const int ParameterIDs::Reverb::amount = effect + 1

const int ParameterIDs::Reverb::damp = effect + 4

const int ParameterIDs::Reverb::effect = 50

const int ParameterIDs::Reverb::mode = effect + 2

const int ParameterIDs::Reverb::size = effect + 3

const int ParameterIDs::Reverb::width = effect + 5

ParameterIDs::Tracks Namespace Reference

Variables

- const int **Track1** = 100
- const int **Track2** = 200
- const int **Track3** = 300
- const int **Track4** = 400

Variable Documentation

const int ParameterIDs::Tracks::Track1 = 100

const int ParameterIDs::Tracks::Track2 = 200

const int ParameterIDs::Tracks::Track3 = 300

const int ParameterIDs::Tracks::Track4 = 400

ParameterIDs::Waveshaper Namespace Reference

Variables

- const int **effect** = 20
- const int **amount** = **effect** + 1
- const int **funcControl** = **effect** + 2
- const int **mode** = **effect** + 3

Variable Documentation

const int ParameterIDs::Waveshaper::amount = effect + 1

const int ParameterIDs::Waveshaper::effect = 20

const int ParameterIDs::Waveshaper::funcControl = effect + 2

const int ParameterIDs::Waveshaper::mode = effect + 3

StyleSheet Namespace Reference

Namespaces

- namespace **Effects**
- namespace **Mixer**
- namespace **Tracks**

StyleSheet::Effects Namespace Reference

StyleSheet::Mixer Namespace Reference

StyleSheet::Tracks Namespace Reference

Class Documentation

allpass Class Reference

`#include <allpass.hpp>`

Public Member Functions

- **allpass** ()
- void **setbuffer** (float *buf, int size)
- float **process** (float inp)
- void **mute** ()
- void **setfeedback** (float val)
- float **getfeedback** ()

Public Attributes

- float **feedback**
- float * **buffer**
- int **bufsize**
- int **bufidx**

Constructor & Destructor Documentation

allpass::allpass ()

Member Function Documentation

float allpass::getfeedback ()

void allpass::mute ()

float allpass::process (float *inp*) `[inline]`

void allpass::setbuffer (float * *buf*, int *size*)

void allpass::setfeedback (float *val*)

Member Data Documentation

float* allpass::buffer

int allpass::bufidx

int allpass::bufsize

float allpass::feedback

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

- DSP/FX/Reverb/**allpass.hpp**
- DSP/FX/Reverb/**allpass.cpp**

AudioParameter< type > Class Template Reference

`#include <AudioParameter.h>`

Public Member Functions

- void **init** (DaisySeed *seed, type mi, type ma, **CurveType** c, int ID, std::function< void(type)> cb)
  Initialises an instance of the class.

- void **tick** ()
  Handles the polling of the assigned ADC channel.

- void **processCurve** ()
  Scales the input value according to the assigned curve.

- type **getValue** ()
  Fetches the current value of the instance.

Member Function Documentation

template<class type > type AudioParameter< type >::getValue () [inline]

Fetches the current value of the instance.

Returns

the current value

template<class type > void AudioParameter< type >::init (DaisySeed * *seed*, type *mi*, type *ma*, CurveType *c*, int *ID*, std::function< void(type)> *cb*) [inline]

Initialises an instance of the class.

Parameters

seed	A pointer to the program's instance of the hardware
mi	The minimum value the parameter's input should be scaled too
ma	The maximum value the parameter's input should be scaled too
c	The type of curve that should be used to scale the input
ID	The channel ID for the ADC channel the instance is assigned too
cb	The callback function that should be executed when the input values changes

template<class type > void AudioParameter< type >::processCurve () [inline]

Scales the input value according to the assigned curve.

template<class type > void AudioParameter< type >::tick () [inline]

Handles the polling of the assigned ADC channel.

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

- Parameters/**AudioParameter.h**

AudioParameterWrapper< type > Struct Template Reference

`#include <AudioParameter.h>`

Public Attributes

- **AudioParameter**< type > **param**
- type **value**

Member Data Documentation

template<class type > AudioParameter<type> AudioParameterWrapper< type >::param

template<class type > type AudioParameterWrapper< type >::value

---

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

- Parameters/**AudioParameter.h**

AudioSlider Class Reference

`#include <AudioSlider.h>`

Public Member Functions

- void **init** (int ID, DaisySeed *seed)
  Initialises the instance.

- void **tick** ()
  Handles the updating of the slider.

- void **repaint** (int index, bool selected)
  Handles the repainting of the slider on the screen.

Member Function Documentation

void AudioSlider::init (int *ID*, DaisySeed * *seed*)

Initialises the instance.

Parameters

ID	The channel ID of the ADC channel the slider is assigned to
seed	A pointer to the hardware instance

void AudioSlider::repaint (int *index*, bool *selected*)

Handles the repainting of the slider on the screen.

Parameters

Index	The channel index of the slider
selected	If the channel is selected or not

void AudioSlider::tick ()

Handles the updating of the slider.

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

- GUI/**AudioSlider.h**
- GUI/**AudioSlider.cpp**

BinaryParameter Class Reference

`#include <BinaryParameter.h>`

Public Member Functions

- void **init** (dsy_gpio_pin pin, float updateRate, std::function< void()> cb)
  Initialises an instance of the class.

- void **tick** ()
  Checks to see if the button has been pressed.

- bool **isPressed** ()
  Checks the state of the button.

Member Function Documentation

void BinaryParameter::init (dsy_gpio_pin *pin*, float *updateRate*, std::function< void()> *cb*) `[inline]`

Initialises an instance of the class.

Parameters

pin	The Daisy Seed pin the instance is assigned to
updateRate	The rate at which the button is updated
cb	The callback function executed when the button is pressed

bool BinaryParameter::isPressed () `[inline]`

Checks the state of the button.

Returns

If the button is pressed or not

void BinaryParameter::tick () `[inline]`

Checks to see if the button has been pressed.

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

- Parameters/**BinaryParameter.h**

BinaryParameterWrapper Struct Reference

`#include <BinaryParameter.h>`

Public Attributes

- **BinaryParameter param**
- bool **value**

Member Data Documentation

BinaryParameter BinaryParameterWrapper::param

bool BinaryParameterWrapper::value

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

- Parameters/**BinaryParameter.h**

BypassButton Class Reference

`#include <BypassButton.h>`

Public Member Functions

- void **init** (**EncoderDriver** *ed, std::function< void()> bypassCallback)
  Initialises the instance.

- void **tick** ()
  Checks if the bypass state needs to be changed.

- void **repaint** ()
  Handles the repainting of the button.

- void **setIsSelected** (bool state)
  Sets if the button is selected on the interface.

Detailed Description

Class name: **BypassButton** Function: Button used for bypassing an effect

---

Member Function Documentation

void BypassButton::init (EncoderDriver * *ed*, std::function< void()> *bypassCallback*) `[inline]`

Initialises the instance.

Parameters

ed	A pointer to the encoder driver
bypassCallback	The function to be called when the bypass is set

void BypassButton::repaint () `[inline]`

Handles the repainting of the button.

void BypassButton::setIsSelected (bool *state*) `[inline]`

Sets if the button is selected on the interface.

Parameters

state	Sets the bypass state of the instance

void BypassButton::tick () `[inline]`

Checks if the bypass state needs to be changed.

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

- GUI/FX/**BypassButton.h**

comb Class Reference

`#include <comb.hpp>`

Public Member Functions

- **comb** ()
- void **setbuffer** (float *buf, int size)
- float **process** (float inp)
- void **mute** ()
- void **setdamp** (float val)
- float **getdamp** ()
- void **setfeedback** (float val)
- float **getfeedback** ()

Constructor & Destructor Documentation

comb::comb ()

Member Function Documentation

float comb::getdamp ()

float comb::getfeedback ()

void comb::mute ()

float comb::process (float *inp*) `[inline]`

void comb::setbuffer (float * *buf*, int *size*)

void comb::setdamp (float *val*)

void comb::setfeedback (float *val*)

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

- DSP/FX/Reverb/**comb.hpp**
- DSP/FX/Reverb/**comb.cpp**

Delay Class Reference

`#include <Delay.h>`

Public Member Functions

- void **init** (**EncoderDriver** *driver, int trackID, DelayLine< float, **MAXDELAY** > *dl[2])
  Initialises the instance.

- void **setDefaultValues** ()
  Sets tthe default parameter values.

- void **setBypass** ()
  Sets the bypass state of the instance.

- void **setAmount** (float a)
  Sets the amount of the effect in the output.

- void **setDelay** (size_t s)
  Sets the delay time of the delay line.

- void **setFeedback** (float f)
  Sets the feedback value used in the delay algorithm.

- void **processBlock** (float *input[2], size_t size)
  Processes a block of samples through the delay and mixes the output.

Member Function Documentation

void Delay::init (EncoderDriver * *driver*, int *trackID*, DelayLine< float, MAXDELAY > * *dl*[2])

Initialises the instance.

Parameters

driver	A pointer to the encoder driver used to initialise the parameters
size	The size of the block of samples
trackID	The ID of the track the instance belongs to
dl	A pointer to an array of delay lines to be used by the instance

void Delay::processBlock (float * *input*[2], size_t *size*)

Processes a block of samples through the delay and mixes the output.

Parameters

input	An array of pointers pointing to the input buffer
size	The size of the block of samples

void Delay::setAmount (float *a*) [inline]

Sets the amount of the effect in the output.

void Delay::setBypass () [inline]

Sets the bypass state of the instance.

void Delay::setDefaultValues ()

Sets tthe default parameter values.

void Delay::setDelay (size_t *s*) [inline]

Sets the delay time of the delay line.

Parameters

s	The new delay time

void Delay::setFeedback (float *f*) [inline]

Sets the feedback value used in the delay algorithm.

Parameters

f	The new feedback value

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

- DSP/FX/**Delay.h**
- DSP/FX/**Delay.cpp**

DelayView Class Reference

`#include <DelayView.h>`

Public Member Functions

- void **init** (int ID, **EncoderDriver** *driver, **KeypadDriver** *kpd)
  Initialises the instance.

- void **tick** ()
  Handles the updating of the view.

- void **repaint** ()
  Handles the repainting of the view.

- void **clear** ()
  Handles the clearing of the view.

- void **setIsOpen** (bool state)
  Sets if the view is currently open.

- void **setIsPainted** (bool state)
  Sets if the view has been painted or not.

- void **setCurrentParam** (int newParam)
  Sets the currently selected parameter on the LCD.

Detailed Description

Class name: **DelayView** Function: FX level GUI class for the delay

---

Member Function Documentation

void DelayView::clear ()

Handles the clearing of the view.

void DelayView::init (int *ID*, EncoderDriver * *driver*, KeypadDriver * *kpd*)

Initialises the instance.

Parameters

ID	The ID of the track view the instance belongs to
encoder	A pointer to the encoder driver
kpd	A pointer to the keypad driver

void DelayView::repaint ()

Handles the repainting of the view.

void DelayView::setCurrentParam (int *newParam*) `[inline]`

Sets the currently selected parameter on the LCD.

void DelayView::setIsOpen (bool *state*) `[inline]`

Sets if the view is currently open.

void DelayView::setIsPainted (bool *state*) `[inline]`

Sets if the view has been painted or not.

void DelayView::tick ()

Handles the updating of the view.

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

- GUI/FX/**DelayView.h**
- GUI/FX/**DelayView.cpp**

EncoderDriver Class Reference

`#include <EncoderDriver.h>`

Public Member Functions

- void **init** (dsy_gpio_pin button, dsy_gpio_pin a, dsy_gpio_pin b, std::function< void()> navCb)
  Initialises the driver.

- void **tick** ()
  Polls the inputs to see if a change has occured.

- void **buttonCallback** ()
  The callback function executed when the centre button.

- bool **getButtonState** ()
  Gets the state of the encoder's centre button.

- std::function< void()> **getBypassCallback** (size_t index)
  Gets the bypass callback for a given index from the bypassCallbacks vector.

- void **setCurrentParam** (int newID)
  Sets the current parameter the encoder is assigned to.

- void **addParameter** (**SteppedParameter** *newParam)
  Adds a parameter to the parameter vector.

- void **addBypassCallback** (std::function< void()> newCallback)
  Adds a bypass callback to the bypassCallbacks vector.

- **SteppedParameter** * **getParameter** (int paramID)
  Gets a pointer to the parameter with the given index.

Detailed Description

Class name: **EncoderDriver** Function: Driver for the encoder input, controls the value of FX parameters

Based off of Encoder class from libDaisy (Stephen Hensley, 2019) `https://github.com/electro-smith/libDaisy/blob/master/src/hid/encoder.h`

Member Function Documentation

void EncoderDriver::addBypassCallback (std::function< void()> *newCallback*) `[inline]`

Adds a bypass callback to the bypassCallbacks vector.

Parameters

newCallback	The callback function to be added

void EncoderDriver::addParameter (SteppedParameter * *newParam*) `[inline]`

Adds a parameter to the parameter vector.

Parameters

newParam	A pointer to the parameter to be added to the vector

void EncoderDriver::buttonCallback () `[inline]`

The callback function executed when the centre button.

bool EncoderDriver::getButtonState () `[inline]`

Gets the state of the encoder's centre button.

Returns

If the button is pressed or not

std::function< void()> EncoderDriver::getBypassCallback (size_t *index*) `[inline]`

Gets the bypass callback for a given index from the bypassCallbacks vector.

SteppedParameter * EncoderDriver::getParameter (int *paramID*) `[inline]`

Gets a pointer to the parameter with the given index.

Parameters

paramID	The ID of the parameter to be fetched

void EncoderDriver::init (dsy_gpio_pin *button*, dsy_gpio_pin *a*, dsy_gpio_pin *b*, std::function< void()> *navCb*) `[inline]`

Initialises the driver.

Parameters

button	The pin used for the centre push button
a	The pin used for channel A of the encoder
b	The pin used for channel B of the encoder
navCb	The function used for navigation

void EncoderDriver::setCurrentParam (int *newID*) `[inline]`

Sets the current parameter the encoder is assigned to.

Parameters

newID	The ID of the parameter the encoder is to be assigned to

void EncoderDriver::tick () `[inline]`

Polls the inputs to see if a change has occured.

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

- Drivers/**EncoderDriver.h**

Filter Class Reference

`#include <Filter.h>`

Public Member Functions

- void **init** (**EncoderDriver** *driver, int trackID)
  Initialises the instance.

- void **setDefaultValues** ()
  Sets tthe default parameter values.

- void **setBypass** ()
  Sets the bypass state of the instance.

- void **setFreq** (float f)
  Sets the cutoff frequency of the filter.

- void **setMode** (float m)
  Sets the mode of the filter, either high pass or low pass.

- void **processBlock** (float *input[2], size_t size)
  Processes a block of samples through the filter.

Member Function Documentation

void Filter::init (EncoderDriver * *driver*, int *trackID*)

Initialises the instance.

Parameters

driver	A pointer to the encoder driver used to initialise the parameters
size	The size of the block of samples
trackID	The ID of the track the instance belongs to

void Filter::processBlock (float * *input*[2], size_t *size*)

Processes a block of samples through the filter.

Parameters

input	An array of pointers pointing to the input buffer
size	The size of the block of samples

void Filter::setBypass () `[inline]`

Sets the bypass state of the instance.

void Filter::setDefaultValues ()

Sets tthe default parameter values.

void Filter::setFreq (float *f*) `[inline]`

Sets the cutoff frequency of the filter.

Parameters

f	The new cutoff frequency

void Filter::setMode (float *m*)

Sets the mode of the filter, either high pass or low pass.

Parameters

m	the new mode

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

- DSP/FX/**Filter.h**
- DSP/FX/**Filter.cpp**

FilterView Class Reference

`#include <FilterView.h>`

Public Member Functions

- void **init** (int ID, **EncoderDriver** *driver, **KeypadDriver** *kpd)
  Initialises the instance.

- void **tick** ()
  Handles the updating of the view.

- void **repaint** ()
  Handles the repainting of the view.

- void **clear** ()
  Handles the clearing of the view.

- void **setIsOpen** (bool state)
  Sets if the view is currently open.

- void **setIsPainted** (bool state)
  Sets if the view has been painted or not.

- void **setCurrentParam** (int newParam)
  Sets the currently selected parameter on the LCD.

Detailed Description

Class name: **FilterView** Function: FX level GUI class for the filter

---

Member Function Documentation

void FilterView::clear ()

Handles the clearing of the view.

void FilterView::init (int *ID*, EncoderDriver * *driver*, KeypadDriver * *kpd*)

Initialises the instance.

Parameters

ID	The ID of the track view the instance belongs to
encoder	A pointer to the encoder driver
kpd	A pointer to the keypad driver

void FilterView::repaint ()

Handles the repainting of the view.

void FilterView::setCurrentParam (int *newParam*) `[inline]`

Sets the currently selected parameter on the LCD.

void FilterView::setIsOpen (bool *state*) `[inline]`

Sets if the view is currently open.

void FilterView::setIsPainted (bool *state*) `[inline]`

Sets if the view has been painted or not.

void FilterView::tick ()

Handles the updating of the view.

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

- GUI/FX/**FilterView.h**
- GUI/FX/**FilterView.cpp**

ILI9341SpiTransport Class Reference

`#include <daisy_ILI9341.hpp>`

Public Member Functions

- void **Init** ()
- void **Reset** ()
- SpiHandle::Result **SendDataDMA** ()
- SpiHandle::Result **SendDataDMA** (uint8_t *buff, size_t size)
- uint32_t **GetTransferSize** () const
- SpiHandle::Result **SendCommand** (uint8_t cmd)
- SpiHandle::Result **SendData** (uint8_t *buff, size_t size)
- void **SetAddressWindow** (uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1)
- void **PaintPixel** (uint32_t id, uint8_t color_id, uint8_t alpha=255) const
- uint16_t **GetPixel** (uint32_t id)

Static Public Member Functions

- static void **TxCompleteCallback** (void *context, SpiHandle::Result result)

Public Attributes

- bool **dma_busy** = false
- uint32_t **remaining_buff** = 0
- const uint16_t **buf_chunk_size** = **buffer_size** / 3
- SpiHandle **spi_**
- uint16_t **tftPalette** [**NUMBER_OF_TFT_COLORS**]

Static Public Attributes

- static uint32_t const **buffer_size** = 153600

Detailed Description

SPI Transport for ILI9341 TFT display devices

Member Function Documentation

uint16_t ILI9341SpiTransport::GetPixel (uint32_t *id*) `[inline]`

uint32_t ILI9341SpiTransport::GetTransferSize () const `[inline]`

void ILI9341SpiTransport::Init () `[inline]`

void ILI9341SpiTransport::PaintPixel (uint32_t *id*, uint8_t *color_id*, uint8_t *alpha* = 255) const `[inline]`

void ILI9341SpiTransport::Reset () `[inline]`

SpiHandle::Result ILI9341SpiTransport::SendCommand (uint8_t *cmd*) `[inline]`

SpiHandle::Result ILI9341SpiTransport::SendData (uint8_t * *buff*, size_t *size*) `[inline]`

SpiHandle::Result ILI9341SpiTransport::SendDataDMA () `[inline]`

SpiHandle::Result ILI9341SpiTransport::SendDataDMA (uint8_t * *buff*, size_t *size*) `[inline]`

void ILI9341SpiTransport::SetAddressWindow (uint16_t *x0*, uint16_t *y0*, uint16_t *x1*, uint16_t *y1*) `[inline]`

static void ILI9341SpiTransport::TxCompleteCallback (void * *context*, SpiHandle::Result *result*) `[inline]`, `[static]`

Member Data Documentation

const uint16_t ILI9341SpiTransport::buf_chunk_size = buffer_size / 3

uint32_t const ILI9341SpiTransport::buffer_size = 153600 `[static]`

bool ILI9341SpiTransport::dma_busy = false

uint32_t ILI9341SpiTransport::remaining_buff = 0

SpiHandle ILI9341SpiTransport::spi_

uint16_t ILI9341SpiTransport::tftPalette[NUMBER_OF_TFT_COLORS]

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

- Drivers/**daisy_ILI9341.hpp**

KeypadDriver::Index Struct Reference

`#include <KeypadDriver.h>`

Public Attributes

- int **row**
- int **col**

Member Data Documentation

int KeypadDriver::Index::col

int KeypadDriver::Index::row

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

- Drivers/**KeypadDriver.h**

KeypadDriver Class Reference

`#include <KeypadDriver.h>`

Classes

struct Index**Public Member Functions**

- **KeypadDriver** ()
- void **init** (dsy_gpio_pin t, dsy_gpio_pin b, dsy_gpio_pin l, dsy_gpio_pin r)
- void **tick** ()
- **Index getIndex** () const
- bool **isTopPressed** ()
- bool **isBottomPressed** ()
- bool **isRightPressed** ()
- bool **isLeftPressed** ()

Detailed Description

Class name: **KeypadDriver** Function: Driver for the keypad input, controls the movement between views

---

Constructor & Destructor Documentation

KeypadDriver::KeypadDriver () `[inline]`

Member Function Documentation

Index KeypadDriver::getIndex () const `[inline]`

void KeypadDriver::init (dsy_gpio_pin *t*, dsy_gpio_pin *b*, dsy_gpio_pin *l*, dsy_gpio_pin *r*) `[inline]`

bool KeypadDriver::isBottomPressed () `[inline]`

bool KeypadDriver::isLeftPressed () `[inline]`

bool KeypadDriver::isRightPressed () `[inline]`

bool KeypadDriver::isTopPressed () `[inline]`

void KeypadDriver::tick () `[inline]`

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

- Drivers/**KeypadDriver.h**

Mixer Class Reference

`#include <Mixer.h>`

Public Member Functions

- void **init** (DaisySeed *seed, float *t1[2], float *t2[2], float *t3[2], float *t4[2])
  Initialises the mixer class.

- void **initMixChannels** (float *m[2], float *t1[2], float *t2[2], float *t3[2], float *t4[2])
  Initialises the mixer channels.

- void **initFX** (**EncoderDriver** *driver, DelayLine< float, **MAXDELAY** > *t1[2], DelayLine< float, **MAXDELAY** > *t2[2], DelayLine< float, **MAXDELAY** > *t3[2], DelayLine< float, **MAXDELAY** > *t4[2])
  Initialises all the effects.

- void **tick** ()
  Handles the updating of the backend parameters.

- void **processInputBlock** (const float *left, const float *right, size_t size)
  Processes a block of samples from the Seed's input.

- void **panChannels** (size_t size)
  Handles the panning of the mixer channels.

- void **mixOutput** (size_t size)
  Handles the mixing of the output.

- void **processOutputBlock** (float *left, float *right, size_t size)
  Processes a block of samples and sends them to the Seed's output.

- void **panMixBuffer** (float *buffer[2], float pan, size_t size)
  Applies a -6dB linear taper pan to a mix buffer.

- void **setMixDiv** ()
  Sets the mix divisor used to mix the output.

- void **setTrack1Pan** (float p)
  Sets the pan value for track 1.

- void **setTrack2Pan** (float p)
  Sets the pan value for track 2.

- void **setTrack3Pan** (float p)
  Sets the pan value for track 3.

- void **setTrack4Pan** (float p)

Sets the pan value for track 4.

- void **setTrack1Gain** (float g)
  Sets the gain value for track 1.

- void **setTrack2Gain** (float g)
  Sets the gain value for track 2.

- void **setTrack3Gain** (float g)
  Sets the gain value for track 3.

- void **setTrack4Gain** (float g)
  Sets the gain value for track 4.

- void **setMasterVolume** (float m)
  Sets the master gain value.

Detailed Description

Class name: **Mixer** Function: Mixes outputs of track classes

---

Member Function Documentation

void Mixer::init (DaisySeed * *seed*, float * *t1*[2], float * *t2*[2], float * *t3*[2], float * *t4*[2])

Initialises the mixer class.

Parameters

t1	An array of pointers pointing to the buffer for track 1
t2	An array of pointers pointing to the buffer for track 2
t3	An array of pointers pointing to the buffer for track 3
t4	An array of pointers pointing to the buffer for track 4

void Mixer::initFX (EncoderDriver * *driver*, DelayLine< float, MAXDELAY > * *t1*[2], DelayLine< float, MAXDELAY > * *t2*[2], DelayLine< float, MAXDELAY > * *t3*[2], DelayLine< float, MAXDELAY > * *t4*[2])

Initialises all the effects.

Parameters

driver	A pointer to the LCD driver
t1	An array of pointers pointing to the delay line for track 1
t2	An array of pointers pointing to the delay line for track 2
t3	An array of pointers pointing to the delay line for track 3
t4	An array of pointers pointing to the delay line for track 4

void Mixer::initMixChannels (float * *m*[2], float * *t1*[2], float * *t2*[2], float * *t3*[2], float * *t4*[2])

Initialises the mixer channels.

Parameters

t1	An array of pointers pointing to the mix buffer for track 1
t2	An array of pointers pointing to the mix buffer for track 2
t3	An array of pointers pointing to the mix buffer for track 3
t4	An array of pointers pointing to the mix buffer for track 4

void Mixer::mixOutput (size_t *size*)

Handles the mixing of the output.

void Mixer::panChannels (size_t *size*)

Handles the panning of the mixer channels.

void Mixer::panMixBuffer (float * *buffer*[2], float *pan*, size_t *size*)

Applies a -6dB linear taper pan to a mix buffer.

Parameters

buffer	The mix buffer the panning should be applied to
pan	The pan value for that mix buffer
size	The size of the mix buffer

void Mixer::processInputBlock (const float * *left*, const float * *right*, size_t *size*)

Processes a block of samples from the Seed's input.

Parameters

left	A pointer to the Seed's left input buffer
right	A pointer to the Seed's right input buffer
size	

void Mixer::processOutputBlock (float * *left*, float * *right*, size_t *size*)

Processes a block of samples and sends them to the Seed's output.

Parameters

left	A pointer to the Seed's left output buffer
right	A pointer to the Seed's right output buffer

void Mixer::setMasterVolume (float *m*) `[inline]`

Sets the master gain value.

void Mixer::setMixDiv ()

Sets the mix divisor used to mix the output.

void Mixer::setTrack1Gain (float *g*) [inline]

Sets the gain value for track 1.

void Mixer::setTrack1Pan (float *p*) [inline]

Sets the pan value for track 1.

void Mixer::setTrack2Gain (float *g*) [inline]

Sets the gain value for track 2.

void Mixer::setTrack2Pan (float *p*) [inline]

Sets the pan value for track 2.

void Mixer::setTrack3Gain (float *g*) [inline]

Sets the gain value for track 3.

void Mixer::setTrack3Pan (float *p*) [inline]

Sets the pan value for track 3.

void Mixer::setTrack4Gain (float *g*) [inline]

Sets the gain value for track 4.

void Mixer::setTrack4Pan (float *p*) [inline]

Sets the pan value for track 4.

void Mixer::tick ()

Handles the updating of the backend parameters.

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

- DSP/**Mixer.h**
- DSP/**Mixer.cpp**

MixerView Class Reference

`#include <MixerView.h>`

Public Member Functions

- void **init** (DaisySeed *seed, **EncoderDriver** ***encoder**, **KeypadDriver** *kpd)
  Initialises the mixer view and all other GUI classes.

- void **tick** ()
  Handles the updating of the view.

- void **repaint** ()
  Handles the repainting of the view.

- void **clear** ()
  Handles the clearing of the view.

- void **setIsOpen** (bool state)
  Sets if the view is currently open.

- void **setIsPainted** (bool state)
  Sets if the view has been painted or not.

- void **setSelectedChannel** (int channel)
  Sets which channel is currently selected.

Detailed Description

Class name: **MixerView** Function: Top level GUI class

Member Function Documentation

void MixerView::clear ()

Handles the clearing of the view.

void MixerView::init (DaisySeed * *seed*, EncoderDriver * *encoder*, KeypadDriver * *kpd*)

Initialises the mixer view and all other GUI classes.

Parameters

seed	A pointer to the hardware instance
encoder	A pointer to the encoder driver

kpd	A pointer to the keypad driver

void MixerView::repaint ()

Handles the repainting of the view.

void MixerView::setIsOpen (bool *state*) `[inline]`

Sets if the view is currently open.

void MixerView::setIsPainted (bool *state*) `[inline]`

Sets if the view has been painted or not.

void MixerView::setSelectedChannel (int *channel*) `[inline]`

Sets which channel is currently selected.

Parameters

channel	The channel to be selected

void MixerView::tick ()

Handles the updating of the view.

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

- GUI/**MixerView.h**
- GUI/**MixerView.cpp**

PitchShift Class Reference

`#include <PitchShift.h>`

Public Member Functions

- void **init** (**EncoderDriver** *driver, int trackID)
  Initialises the instance.

- void **setDefaultValues** ()
  Sets tthe default parameter values.

- void **setBypass** ()
  Sets the bypass state of the instance.

- void **setAmount** (float a)
  Sets the amount of the effect in the output.

- void **processBlock** (float *input[2], size_t size)
  Pitch shifts a block of samples and mixes it with the input.

Member Function Documentation

void PitchShift::init (EncoderDriver * *driver*, int *trackID*)

Initialises the instance.

Parameters

driver	A pointer to the encoder driver used to initialise the parameters
size	The size of the block of samples
trackID	The ID of the track the instance belongs to

void PitchShift::processBlock (float * *input*[2], size_t *size*)

Pitch shifts a block of samples and mixes it with the input.

Parameters

input	An array of pointers pointing to the input buffer
size	The size of the block of samples

void PitchShift::setAmount (float *a*) `[inline]`

Sets the amount of the effect in the output.

void PitchShift::setBypass () `[inline]`

Sets the bypass state of the instance.

void PitchShift::setDefaultValues ()

Sets tthe default parameter values.

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

- DSP/FX/**PitchShift.h**
- DSP/FX/**PitchShift.cpp**

PitchShiftView Class Reference

`#include <PitchShiftView.h>`

Public Member Functions

- void **init** (int ID, **EncoderDriver** *driver, **KeypadDriver** *kpd)
  Initialises the instance.

- void **tick** ()
  Handles the updating of the view.

- void **repaint** ()
  Handles the repainting of the view.

- void **clear** ()
  Handles the clearing of the view.

- void **setIsOpen** (bool state)
  Sets if the view is currently open.

- void **setIsPainted** (bool state)
  Sets if the view has been painted or not.

- void **setCurrentParam** (int newParam)
  Sets the currently selected parameter on the LCD.

Detailed Description

Class name: **PitchShiftView** Function: FX level GUI class for the pitch shift

---

Member Function Documentation

void PitchShiftView::clear ()

Handles the clearing of the view.

void PitchShiftView::init (int *ID*, EncoderDriver * *driver*, KeypadDriver * *kpd*)

Initialises the instance.

Parameters

ID	The ID of the track view the instance belongs to
encoder	A pointer to the encoder driver
kpd	A pointer to the keypad driver

void PitchShiftView::repaint ()

Handles the repainting of the view.

void PitchShiftView::setCurrentParam (int *newParam*) `[inline]`

Sets the currently selected parameter on the LCD.

void PitchShiftView::setIsOpen (bool *state*) `[inline]`

Sets if the view is currently open.

void PitchShiftView::setIsPainted (bool *state*) `[inline]`

Sets if the view has been painted or not.

void PitchShiftView::tick ()

Handles the updating of the view.

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

- GUI/FX/**PitchShiftView.h**
- GUI/FX/**PitchShiftView.cpp**

Reverb Class Reference

`#include <Reverb.h>`

Public Member Functions

- void **init** (**EncoderDriver** *driver, int trackID)
  Initialises the instance.

- void **setDefaultValues** ()
  Sets tthe default parameter values.

- void **setBypass** ()
  Sets the bypass state of the instance.

- void **setAmount** (float mix)
  Sets the amount of the effect in the output.

- void **processBlock** (float *input[2], long size)
  Processes a block of samples through the reverb and mixes the output.

Member Function Documentation

void Reverb::init (EncoderDriver * *driver*, int *trackID*)

Initialises the instance.

Parameters

driver	A pointer to the encoder driver used to initialise the parameters
size	The size of the block of samples
trackID	The ID of the track the instance belongs to

void Reverb::processBlock (float * *input*[2], long *size*)

Processes a block of samples through the reverb and mixes the output.

Parameters

input	An array of pointers pointing to the input buffer
size	The size of the block of samples

void Reverb::setAmount (float *mix*)

Sets the amount of the effect in the output.

void Reverb::setBypass () `[inline]`

Sets the bypass state of the instance.

void Reverb::setDefaultValues ()

Sets tthe default parameter values.

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

- DSP/FX/Reverb/**Reverb.h**
- DSP/FX/Reverb/**Reverb.cpp**

ReverbView Class Reference

`#include <ReverbView.h>`

Public Member Functions

- void **init** (int ID, **EncoderDriver** *driver, **KeypadDriver** *kpd)
  Initialises the instance.

- void **tick** ()
  Handles the updating of the view.

- void **repaint** ()
  Handles the repainting of the view.

- void **clear** ()
  Handles the clearing of the view.

- void **setIsOpen** (bool state)
  Sets if the view is currently open.

- void **setIsPainted** (bool state)
  Sets if the view has been painted or not.

- void **setCurrentParam** (int newParam)
  Sets the currently selected parameter on the LCD.

Detailed Description

Class name: **ReverbView** Function: FX level GUI class for the reverb

---

Member Function Documentation

void ReverbView::clear ()

Handles the clearing of the view.

void ReverbView::init (int *ID*, EncoderDriver * *driver*, KeypadDriver * *kpd*)

Initialises the instance.

Parameters

ID	The ID of the track view the instance belongs to
encoder	A pointer to the encoder driver
kpd	A pointer to the keypad driver

void ReverbView::repaint ()

Handles the repainting of the view.

void ReverbView::setCurrentParam (int *newParam*) `[inline]`

Sets the currently selected parameter on the LCD.

void ReverbView::setIsOpen (bool *state*) `[inline]`

Sets if the view is currently open.

void ReverbView::setIsPainted (bool *state*) `[inline]`

Sets if the view has been painted or not.

void ReverbView::tick ()

Handles the updating of the view.

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

- GUI/FX/**ReverbView.h**
- GUI/FX/**ReverbView.cpp**

revmodel Class Reference

`#include <revmodel.hpp>`

Public Member Functions

- **revmodel** ()
- void **mute** ()
- void **processmix** (float *inputL, float *inputR, float *outputL, float *outputR, long numsamples, int skip)
- void **processreplace** (float *inputL, float *inputR, float *outputL, float *outputR, long numsamples, int skip)
- void **setroomsize** (float value)
- float **getroomsize** ()
- void **setdamp** (float value)
- float **getdamp** ()
- void **setwet** (float value)
- float **getwet** ()
- void **setdry** (float value)
- float **getdry** ()
- void **setwidth** (float value)
- float **getwidth** ()
- void **setmode** (float value)
- float **getmode** ()

Constructor & Destructor Documentation

revmodel::revmodel ()

Member Function Documentation

float revmodel::getdamp ()

float revmodel::getdry ()

float revmodel::getmode ()

float revmodel::getroomsize ()

float revmodel::getwet ()

float revmodel::getwidth ()

void revmodel::mute ()

void revmodel::processmix (float * ***inputL*****, float *** ***inputR*****, float *** ***outputL*****, float *** ***outputR*****, long** ***numsamples*****, int** ***skip*****)**

void revmodel::processreplace (float * ***inputL*****, float *** ***inputR*****, float *** ***outputL*****, float *** ***outputR*****, long** ***numsamples*****, int** ***skip*****)**

void revmodel::setdamp (float ***value*****)**

void revmodel::setdry (float ***value*****)**

void revmodel::setmode (float ***value*****)**

void revmodel::setroomsize (float ***value*****)**

void revmodel::setwet (float ***value*****)**

void revmodel::setwidth (float ***value*****)**

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

- DSP/FX/Reverb/**revmodel.hpp**
- DSP/FX/Reverb/**revmodel.cpp**

SteppedParameter Class Reference

`#include <SteppedParameter.h>`

Public Member Functions

- void **init** (float mi, float ma, float st, int param, int track, std::function< void(float)> cb)
  Initialises an instance of the class.

- void **increment** ()
  Increments the instance's value and executes the callback.

- void **decrement** ()
  Decrements the instance's value and executes the callback.

- int **getID** ()
  Fetches the instance's full ID.

- float **getMin** ()
  Fetches the lowest value the instance decrements to.

- float **getMax** ()
  Fetches the highest value the instance increments to.

- float **getCurVal** ()
  Fetches the current value of the instance.

Member Function Documentation

void SteppedParameter::decrement () `[inline]`

Decrements the instance's value and executes the callback.

float SteppedParameter::getCurVal () `[inline]`

Fetches the current value of the instance.

Returns

The current value

int SteppedParameter::getID () `[inline]`

Fetches the instance's full ID.

Returns

The ID of the instance

float SteppedParameter::getMax () `[inline]`

Fetches the highest value the instance increments to.

Returns

The maximum value

float SteppedParameter::getMin () `[inline]`

Fetches the lowest value the instance decrements to.

Returns

The miminum value

void SteppedParameter::increment () `[inline]`

Increments the instance's value and executes the callback.

void SteppedParameter::init (float *mi*, float *ma*, float *st*, int *param*, int *track*, std::function< void(float)> *cb*) `[inline]`

Initialises an instance of the class.

Parameters

mi	The lowest value the instance should decrement to
ma	The highest value the instance should increment to
st	The value the instance should increment/decrement by
param	The ID of the parameters the instance is assigned to
track	The ID of the track the instance is assigned to
cb	The callback function executed when the value changes

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

- Parameters/**SteppedParameter.h**

SteppedParameterWrapper Struct Reference

`#include <SteppedParameter.h>`

Public Attributes

- **SteppedParameter param**
- float **value**

Member Data Documentation

SteppedParameter SteppedParameterWrapper::param

float SteppedParameterWrapper::value

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

- Parameters/**SteppedParameter.h**

SteppedSlider Class Reference

`#include <SteppedSlider.h>`

Public Member Functions

- void **init** (int ID, **EncoderDriver** *ed)
  Initialises the instance.

- void **tick** ()
  Handles the updating of the slider to reflect any backend changes.

- void **setSelected** ()
  Changes the selected state to of the slider.

- void **repaint** ()
  Handles the repainting of the slider.

Detailed Description

Class name: **SteppedSlider** Function: Slider for representing stepped parameters

---

Member Function Documentation

void SteppedSlider::init (int *ID*, EncoderDriver * *ed*)

Initialises the instance.

Parameters

ID	The ID of the parameter the instance is assigned to
ed	A pointer to the encoder driver

void SteppedSlider::repaint ()

Handles the repainting of the slider.

void SteppedSlider::setSelected ()

Changes the selected state to of the slider.

void SteppedSlider::tick ()

Handles the updating of the slider to reflect any backend changes.

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

- GUI/FX/**SteppedSlider.h**
- GUI/FX/**SteppedSlider.cpp**

Track Class Reference

`#include <Track.h>`

Public Member Functions

- void **init** (float *mem[2], int ID, dsy_gpio_pin r, dsy_gpio_pin p)
  Initialises the instance.

- void **initFX** (**EncoderDriver** *driver, DelayLine< float, **MAXDELAY** > *dl[2])
  Initialise the track's effects.

- void **tick** ()
  Handles the updating of the instance's branch in the hierarchy.

- void **resetBuffer** ()
  Sets all values in the track's assigned buffer to 0.

- void **setIsRecording** ()
  Sets the track's recording state.

- void **setIsPlaying** ()
  Sets the track's playing state.

- **TrackState getState** ()
  Gets the current state of the track.

- void **incrementWritePos** ()
  Increments the track's playhead write position for recording.

- void **incrementReadPos** ()
  Increments the track's playhead's read position for playback.

- size_t **getReadPos** ()
  Gets the current value of the track's playhead's read position.

- void **processInputBlock** (const float *left, const float *right, size_t size)
  Processes a block of samples from the Seed's input.

- void **processOutputBlock** (float *output[2], size_t size)
  Fills the track's mix buffer with samples and processes it through the effects.

Detailed Description

Class name: **Track** Function: Processes audio buffer and FX for each track

---

Member Function Documentation

size_t Track::getReadPos () [inline]

Gets the current value of the track's playhead's read position.

TrackState Track::getState () [inline]

Gets the current state of the track.

Returns

The state of the track

void Track::incrementReadPos ()

Increments the track's playhead's read position for playback.

void Track::incrementWritePos ()

Increments the track's playhead write position for recording.

void Track::init (float * *mem*[2], int *ID*, dsy_gpio_pin *r*, dsy_gpio_pin *p*)

Initialises the instance.

Parameters

mem	An array of pointers pointing to the track's buffers in the **Buffers** namespace
ID	The track ID used to initialise its parameters
r	The pin number of the track's record button used to initialise the record **BinaryParameter**
p	The pin number of the track's play button used to initialise the play **BinaryParameter**

void Track::initFX (EncoderDriver * *driver*, DelayLine< float, MAXDELAY > * *dl*[2])

Initialise the track's effects.

Parameters

driver	A pointer to the LCD driver that lives in LoopyTunes
dl	An array of points pointing to the track's delay lines in the **Buffers** namespace

void Track::processInputBlock (const float * *left*, const float * *right*, size_t *size*)

Processes a block of samples from the Seed's input.

Parameters

left	A pointer to the Seed's left input buffer

right	A pointer to the Seed's right input buffer
size	the Size of the block of samples that need to be processed

void Track::processOutputBlock (float * *output*[2], size_t *size*)

Fills the track's mix buffer with samples and processes it through the effects.

Parameters

output	An array of pointers to the track's mix buffer
size	The number of samples to process

void Track::resetBuffer ()

Sets all values in the track's assigned buffer to 0.

void Track::setIsPlaying ()

Sets the track's playing state.

void Track::setIsRecording ()

Sets the track's recording state.

void Track::tick ()

Handles the updating of the instance's branch in the hierarchy.

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

- DSP/**Track.h**
- DSP/**Track.cpp**

TrackInformation Struct Reference

Struct definition for storing track information.
`#include <Helpers.h>`

Public Attributes

- bool **isEmpty**
- size_t **loopLength**

Detailed Description

Struct definition for storing track information.

Member Data Documentation

bool TrackInformation::isEmpty

size_t TrackInformation::loopLength

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

- Utils/**Helpers.h**

TrackView Class Reference

`#include <TrackView.h>`

Public Member Functions

- void **init** (int ID, **EncoderDriver** *driver, **KeypadDriver** *kpd)
  Initialises the instance.

- void **tick** ()
  Handles the updating of the view.

- void **repaint** ()
  Handles the repainting of the view.

- void **clear** ()
  Handles the clearing of the view.

- void **setIsOpen** (bool state)
  Sets if the view is currently open.

Detailed Description

Class name: **TrackView** Function: **Track** level GUI class

---

Member Function Documentation

void TrackView::clear ()

Handles the clearing of the view.

void TrackView::init (int *ID*, EncoderDriver * *driver*, KeypadDriver * *kpd*)

Initialises the instance.

Parameters

seed	A pointer to the hardware instance
encoder	A pointer to the encoder driver
kpd	A pointer to the keypad driver

void TrackView::repaint ()

Handles the repainting of the view.

void TrackView::setIsOpen (bool *state*) `[inline]`

Sets if the view is currently open.

void TrackView::tick ()

Handles the updating of the view.

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

- GUI/**TrackView.h**
- GUI/**TrackView.cpp**

UiDriver Class Reference

`#include <daisy_ILI9341.hpp>`

Public Types

- enum class **Orientation** { **Default** = 0, **RRight**, **RLeft**, **UpsideDown** }

Public Member Functions

- void **Init** ()
- void **InitDriver** ()
- void **SetOrientation** (**Orientation** ori)
- Rectangle **GetDrawableFrame** () const
- Rectangle **GetBounds** () const
- void **Fill** (uint8_t color)
- void **DrawLine** (uint_fast16_t x1, uint_fast16_t y1, uint_fast16_t x2, uint_fast16_t y2, uint8_t color, uint8_t alpha=255)
- void **FillArea** (uint_fast16_t x, uint_fast16_t y, uint_fast16_t w, uint_fast16_t h, uint8_t color, uint8_t alpha=255)
- void **DrawRect** (uint16_t x, uint16_t y, uint16_t w, uint16_t h, uint8_t color, uint8_t alpha=255)
- void **DrawRect** (const Rectangle &rect, uint8_t color, uint8_t alpha=255)
- void **FillRect** (const Rectangle &rect, uint8_t color, uint8_t alpha=255)
- void **DrawTriangle** (int16_t x0, int16_t y0, int16_t x1, int16_t y1, int16_t x2, int16_t y2, uint8_t color, uint8_t alpha=255)
- void **FillTriangle** (int16_t x0, int16_t y0, int16_t x1, int16_t y1, int16_t x2, int16_t y2, uint8_t color, uint8_t alpha=255)
- Rectangle **WriteStringAligned** (const char *str, const FontDef &font, Rectangle boundingBox, Alignment alignment, uint8_t color)
- Rectangle **GetTextRect** (const char *text, const FontDef &font) const
- char **WriteChar** (char ch, FontDef font, uint8_t color)
- void **WriteString** (const char *str, uint16_t x, uint16_t y, FontDef font)
- void **WriteString** (const char *str, uint16_t x, uint16_t y, FontDef font, uint8_t color)
- uint16_t **GetStringWidth** (const char *str, FontDef font) const
- void **DrawCircle** (int16_t x0, int16_t y0, int16_t r, uint8_t color)
- void **FillCircle** (int16_t x0, int16_t y0, int16_t r, uint8_t color)
- void **FillCircleHelper** (int16_t x0, int16_t y0, int16_t r, uint8_t cornername, int16_t delta, uint8_t color)
- void **SetCursor** (uint16_t x, uint16_t y)
  Moves the 'Cursor' position used for WriteChar, and WriteStr to the specified coordinate.

- void **Update** ()
- bool **IsRender** ()
- void **UpdateFrameRate** ()
- uint16_t **Fps** () const
- void **TrimString** (char *str, char *str_trimmed, uint16_t str_len, uint16_t str_width, FontDef font) const
- void **Start** ()
- void **DrawPixel** (uint_fast16_t x, uint_fast16_t y, uint8_t color, uint8_t alpha=255)
- void **DrawVLine** (int16_t x, int16_t y, int16_t h, uint8_t color, uint8_t alpha=255)
- void **DrawHLine** (int16_t x, int16_t y, int16_t w, uint8_t color, uint8_t alpha=255)
- uint16_t **Color565** (uint8_t red, uint8_t green, uint8_t blue) const
  Given 8-bit red, green and blue values, return a 'packed' 16-bit color value in '565' RGB format (5 bits red, 6 bits green, 5 bits blue). This is just a mathematical operation, no hardware is touched.

Public Attributes

- uint32_t **screen_update_last_**
- uint32_t **screen_update_period_**
- uint32_t **fps_update_last_**
- **ILI9341SpiTransport transport_**
- uint16_t **width**
- uint16_t **height**
- uint8_t **rotation**
- const uint8_t **header** = 20
- const uint8_t **footer** = 13
- uint32_t **diff**
- uint16_t **frames** = 0
- uint16_t **currentX_**
- uint16_t **currentY_**
- uint16_t **fps** = 0

Detailed Description

A driver implementation for the ILI9341

Member Enumeration Documentation

enum class UiDriver::Orientation `[strong]`

Enumerator:

Default	
RRight	
RLeft	
UpsideDown	

Member Function Documentation

uint16_t UiDriver::Color565 (uint8_t *red*, uint8_t *green*, uint8_t *blue*) const `[inline]`

Given 8-bit red, green and blue values, return a 'packed' 16-bit color value in '565' RGB format (5 bits red, 6 bits green, 5 bits blue). This is just a mathematical operation, no hardware is touched.

Parameters

red	8-bit red brightnesss (0 = off, 255 = max).
green	8-bit green brightnesss (0 = off, 255 = max).
blue	8-bit blue brightnesss (0 = off, 255 = max).

Returns

'Packed' 16-bit color value (565 format).

void UiDriver::DrawCircle (int16_t *x0*, int16_t *y0*, int16_t *r*, uint8_t *color*) [inline]

void UiDriver::DrawHLine (int16_t *x*, int16_t *y*, int16_t *w*, uint8_t *color*, uint8_t *alpha* = 255) [inline]

void UiDriver::DrawLine (uint_fast16_t *x1*, uint_fast16_t *y1*, uint_fast16_t *x2*, uint_fast16_t *y2*, uint8_t *color*, uint8_t *alpha* = 255) [inline]

void UiDriver::DrawPixel (uint_fast16_t *x*, uint_fast16_t *y*, uint8_t *color*, uint8_t *alpha* = 255) [inline]

void UiDriver::DrawRect (const Rectangle & *rect*, uint8_t *color*, uint8_t *alpha* = 255) [inline]

void UiDriver::DrawRect (uint16_t *x*, uint16_t *y*, uint16_t *w*, uint16_t *h*, uint8_t *color*, uint8_t *alpha* = 255) [inline]

void UiDriver::DrawTriangle (int16_t *x0*, int16_t *y0*, int16_t *x1*, int16_t *y1*, int16_t *x2*, int16_t *y2*, uint8_t *color*, uint8_t *alpha* = 255) [inline]

void UiDriver::DrawVLine (int16_t *x*, int16_t *y*, int16_t *h*, uint8_t *color*, uint8_t *alpha* = 255) [inline]

void UiDriver::Fill (uint8_t *color*) [inline]

void UiDriver::FillArea (uint_fast16_t *x*, uint_fast16_t *y*, uint_fast16_t *w*, uint_fast16_t *h*, uint8_t *color*, uint8_t *alpha* = 255) [inline]

void UiDriver::FillCircle (int16_t *x0*, int16_t *y0*, int16_t *r*, uint8_t *color*) [inline]

void UiDriver::FillCircleHelper (int16_t *x0*, int16_t *y0*, int16_t *r*, uint8_t *cornername*, int16_t *delta*, uint8_t *color*) [inline]

void UiDriver::FillRect (const Rectangle & *rect*, uint8_t *color*, uint8_t *alpha* = 255) [inline]

void UiDriver::FillTriangle (int16_t *x0*, int16_t *y0*, int16_t *x1*, int16_t *y1*, int16_t *x2*, int16_t *y2*, uint8_t *color*, uint8_t *alpha* = 255) [inline]

uint16_t UiDriver::Fps () const [inline]

Rectangle UiDriver::GetBounds () const [inline]

Rectangle UiDriver::GetDrawableFrame () const [inline]

uint16_t UiDriver::GetStringWidth (const char * *str*, FontDef *font*) const [inline]

Rectangle UiDriver::GetTextRect (const char * *text*, const FontDef & *font*) const [inline]

void UiDriver::Init () [inline]

void UiDriver::InitDriver () `[inline]`

bool UiDriver::IsRender () `[inline]`

void UiDriver::SetCursor (uint16_t *x*, uint16_t *y*) `[inline]`

Moves the 'Cursor' position used for WriteChar, and WriteStr to the specified coordinate.

Parameters

x	x pos
y	y pos

void UiDriver::SetOrientation (Orientation *ori*) `[inline]`

void UiDriver::Start () `[inline]`

void UiDriver::TrimString (char * *str*, char * *str_trimmed*, uint16_t *str_len*, uint16_t *str_width*, FontDef *font*) const `[inline]`

void UiDriver::Update () `[inline]`

void UiDriver::UpdateFrameRate () `[inline]`

char UiDriver::WriteChar (char *ch*, FontDef *font*, uint8_t *color*) `[inline]`

void UiDriver::WriteString (const char * *str*, uint16_t *x*, uint16_t *y*, FontDef *font*) `[inline]`

void UiDriver::WriteString (const char * *str*, uint16_t *x*, uint16_t *y*, FontDef *font*, uint8_t *color*) `[inline]`

Rectangle UiDriver::WriteStringAligned (const char * *str*, const FontDef & *font*, Rectangle *boundingBox*, Alignment *alignment*, uint8_t *color*) `[inline]`

Member Data Documentation

uint16_t UiDriver::currentX_

uint16_t UiDriver::currentY_

uint32_t UiDriver::diff

const uint8_t UiDriver::footer = 13

uint16_t UiDriver::fps = 0

uint32_t UiDriver::fps_update_last_

uint16_t UiDriver::frames = 0

const uint8_t UiDriver::header = 20

uint16_t UiDriver::height

uint8_t UiDriver::rotation

uint32_t UiDriver::screen_update_last_

uint32_t UiDriver::screen_update_period_

ILI9341SpiTransport UiDriver::transport_

uint16_t UiDriver::width

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

- Drivers/**daisy_ILI9341.hpp**

Waveshaper Class Reference

`#include <Waveshaper.h>`

Public Member Functions

- void **init** (**EncoderDriver** *driver, int trackID)
  Initialises the instance.

- void **setDefaultValues** ()
  Sets tthe default parameter values.

- void **setBypass** ()
  Sets the bypass state of the instance.

- void **setAmount** (float a)
  Sets the amount of the effect in the output.

- void **setFuncControl** (float fc)
  Sets the function control value of the instance.

- void **setMode** (float m)
  Sets the mode of the instance.

- void **scaleControlParam** ()
  Scales the value of the function control value for use with the different modes.

- void **setInputAG** (float *buffer[2], size_t size)
  Sets the input values to be used with the auto gain system.

- void **setOutputAG** (float *buffer[2], size_t size)
  Sets the output values to be used with the auto gain system.

- void **calculateAutoGain** (size_t size)
  Sets the auto gain value for each sample in the block.

- void **applyAutoGain** (float *buffer[2], size_t size)
  Applies the calculated auto gain values to a block of samples.

- void **processBlock** (float *input[2], size_t size)
  Processes a block of samples.

- void **processClipper** (float *input[2], size_t size)
  Processes a block of samples using the clipper algorithm.

- void **processFolder** (float *input[2], size_t size)
  Processes a block of samples using the wavefolder.

- void **processLFO** (float *input[2], size_t size)
  Modulates each sample in the block using an LFO.

- void **processBitReducer** (float *input[2], size_t size)
  Reduces the bit depth of each sample in a block.

Member Function Documentation

void Waveshaper::applyAutoGain (float * *buffer*[2], size_t *size*) `[inline]`

Applies the calculated auto gain values to a block of samples.

Parameters

buffer	The buffer the auto gain is to be applied to
size	The size of the buffer in samples

void Waveshaper::calculateAutoGain (size_t *size*) `[inline]`

Sets the auto gain value for each sample in the block.

Parameters

size	The size of the block of samples

void Waveshaper::init (EncoderDriver * *driver*, int *trackID*)

Initialises the instance.

Parameters

driver	A pointer to the encoder driver used to initialise the parameters
trackID	The ID of the track the instance belongs to

void Waveshaper::processBitReducer (float * *input*[2], size_t *size*)

Reduces the bit depth of each sample in a block.

Parameters

input	The block of samples
size	The size of the block of samples

void Waveshaper::processBlock (float * *input*[2], size_t *size*)

Processes a block of samples.

Parameters

input	The block of samples
size	The size of the block of samples

void Waveshaper::processClipper (float * *input*[2], size_t *size*)

Processes a block of samples using the clipper algorithm.

Parameters

input	The block of samples
size	The size of the block of samples

void Waveshaper::processFolder (float * *input*[2], size_t *size*)

Processes a block of samples using the wavefolder.

Parameters

input	The block of samples
size	The size of the block of samples

void Waveshaper::processLFO (float * *input*[2], size_t *size*)

Modulates each sample in the block using an LFO.

Parameters

input	The block of samples
size	The size of the block of samples

void Waveshaper::scaleControlParam () `[inline]`

Scales the value of the function control value for use with the different modes.

void Waveshaper::setAmount (float *a*) `[inline]`

Sets the amount of the effect in the output.

void Waveshaper::setBypass () `[inline]`

Sets the bypass state of the instance.

void Waveshaper::setDefaultValues ()

Sets tthe default parameter values.

void Waveshaper::setFuncControl (float *fc*) `[inline]`

Sets the function control value of the instance.

void Waveshaper::setInputAG (float * *buffer*[2], size_t *size*) `[inline]`

Sets the input values to be used with the auto gain system.

void Waveshaper::setMode (float *m*) `[inline]`

Sets the mode of the instance.

void Waveshaper::setOutputAG (float * *buffer*[2], size_t *size*) `[inline]`

Sets the output values to be used with the auto gain system.

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

- DSP/FX/**Waveshaper.h**
- DSP/FX/**Waveshaper.cpp**

WaveshaperView Class Reference

`#include <WaveshaperView.h>`

Public Member Functions

- void **init** (int ID, **EncoderDriver** *driver, **KeypadDriver** *kpd)
  Initialises the instance.

- void **tick** ()
  Handles the updating of the view.

- void **repaint** ()
  Handles the repainting of the view.

- void **clear** ()
  Handles the clearing of the view.

- void **setIsOpen** (bool state)
  Sets if the view is currently open.

- void **setIsPainted** (bool state)
  Sets if the view has been painted or not.

- void **setCurrentParam** (int newParam)
  Sets the currently selected parameter on the LCD.

Detailed Description

Class name: **WaveshaperView** Function: FX level GUI class for the **Waveshaper**

Member Function Documentation

void WaveshaperView::clear ()

Handles the clearing of the view.

void WaveshaperView::init (int *ID*, EncoderDriver * *driver*, KeypadDriver * *kpd*)

Initialises the instance.

Parameters

ID	The ID of the track view the instance belongs to
encoder	A pointer to the encoder driver
kpd	A pointer to the keypad driver

void WaveshaperView::repaint ()

Handles the repainting of the view.

void WaveshaperView::setCurrentParam (int *newParam*) `[inline]`

Sets the currently selected parameter on the LCD.

void WaveshaperView::setIsOpen (bool *state*) `[inline]`

Sets if the view is currently open.

void WaveshaperView::setIsPainted (bool *state*) `[inline]`

Sets if the view has been painted or not.

void WaveshaperView::tick ()

Handles the updating of the view.

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

- GUI/FX/**WaveshaperView.h**
- GUI/FX/**WaveshaperView.cpp**

File Documentation

build/allpass.d File Reference

build/AudioSlider.d File Reference

build/comb.d File Reference

build/Delay.d File Reference

build/DelayView.d File Reference

build/Filter.d File Reference

build/FilterView.d File Reference

build/ILI9341_ui_driver.d File Reference

build/LoopyTunes.d File Reference

build/Mixer.d File Reference

build/MixerView.d File Reference

build/PitchShift.d File Reference

build/PitchShiftView.d File Reference

build/Reverb.d File Reference

build/ReverbView.d File Reference

build/revmodel.d File Reference

build/startup_stm32h750xx.d File Reference

build/SteppedSlider.d File Reference

build/Track.d File Reference

build/TrackView.d File Reference

build/Waveshaper.d File Reference

build/WaveshaperView.d File Reference

Drivers/daisy_ILI9341.hpp File Reference

```
#include <cstring>
#include "daisy_seed.h"
#include <util/oled_fonts.h>
#include <hid/disp/graphics_common.h>
```

Classes

- class **ILI9341SpiTransport**class **UiDriver**

Macros

- #define **MASK_RB** 63519
- #define **MASK_G** 2016
- #define **MASK_MUL_RB** 4065216
- #define **MASK_MUL_G** 129024
- #define **MAX_ALPHA** 64

Enumerations

- enum **TFT_COLOR** { **COLOR_BLACK** = 0, **COLOR_WHITE**, **COLOR_BLUE**, **COLOR_DARK_BLUE**, **COLOR_CYAN**, **COLOR_YELLOW**, **COLOR_DARK_YELLOW**, **COLOR_ORANGE**, **COLOR_RED**, **COLOR_DARK_RED**, **COLOR_GREEN**, **COLOR_DARK_GREEN**, **COLOR_LIGHT_GREEN**, **COLOR_GRAY**, **COLOR_DARK_GRAY**, **COLOR_LIGHT_GRAY**, **COLOR_MEDIUM_GRAY**, **COLOR_ABL_BG**, **COLOR_ABL_LINE**, **COLOR_ABL_D_LINE**, **COLOR_ABL_L_GRAY**, **COLOR_ABL_M_GRAY**, **NUMBER_OF_TFT_COLORS** }

Variables

- uint8_t DMA_BUFFER_MEM_SECTION **frame_buffer** [153600]
- uint8_t DSY_SDRAM_BSS **color_mem** [153600/2]
- **UiDriver lcd**

Macro Definition Documentation

#define MASK_G 2016

#define MASK_MUL_G 129024

#define MASK_MUL_RB 4065216

#define MASK_RB 63519

#define MAX_ALPHA 64

Enumeration Type Documentation

enum TFT_COLOR

Enumerator:

COLOR_BLACK	
COLOR_WHITE	

COLOR_BLUE	
COLOR_DARK_BLUE	
COLOR_CYAN	
COLOR_YELLOW	
COLOR_DARK_YELLOW	
COLOR_ORANGE	
COLOR_RED	
COLOR_DARK_RED	
COLOR_GREEN	
COLOR_DARK_GREEN	
COLOR_LIGHT_GREEN	
COLOR_GRAY	
COLOR_DARK_GRAY	
COLOR_LIGHT_GRAY	
COLOR_MEDIUM_GRAY	
COLOR_ABL_BG	
COLOR_ABL_LINE	
COLOR_ABL_D_LINE	
COLOR_ABL_L_GRAY	
COLOR_ABL_M_GRAY	
NUMBER_OF_TFT_COLORS	

Variable Documentation

uint8_t DSY_SDRAM_BSS color_mem[153600/2] `[extern]`

uint8_t DMA_BUFFER_MEM_SECTION frame_buffer[153600] `[extern]`

UiDriver lcd `[extern]`

daisy_ILI9341.hpp

Go to the documentation of this file.

```
1 #ifndef DAISY_ILI9341_HPP
2 #define DAISY_ILI9341_HPP
3
4 #include <cstring>
5
6 #include "daisy_seed.h"
7 #include <util/oled_fonts.h>
8 #include <hid/disp/graphics_common.h>
9
10 using namespace daisy;
11
12 // #define SPI1_NSS 7    // LCD CS 3
13 // #define SPI1_SCK 8    // LCD SCK 7
14 // #define SPI1_MOSI 10  // LCD MOSI 6
15 // #define ADC1_INP15 16 // PIN 23 - LCD RST 4
16 // #define DAC1_OUT2 22  //PIN 29 - LCD DC 5
17
18 // #define SPI1_MISO 9 // Not used
19 // #define I2C1_SCL 11
20 // #define I2C1_SDA 12
21
22 enum TFT_COLOR
23 {
24     COLOR_BLACK = 0,
25     COLOR_WHITE,
26     COLOR_BLUE,
27     COLOR_DARK_BLUE,
28     COLOR_CYAN,
29     COLOR_YELLOW,
30     COLOR_DARK_YELLOW,
31     COLOR_ORANGE,
32     COLOR_RED,
33     COLOR_DARK_RED,
34     COLOR_GREEN,
35     COLOR_DARK_GREEN,
36     COLOR_LIGHT_GREEN,
37     COLOR_GRAY,
38     COLOR_DARK_GRAY,
39     COLOR_LIGHT_GRAY,
40     COLOR_MEDIUM_GRAY,
41     COLOR_ABL_BG,
42     COLOR_ABL_LINE,
43     COLOR_ABL_D_LINE,
44     COLOR_ABL_L_GRAY,
45     COLOR_ABL_M_GRAY,
46     NUMBER_OF_TFT_COLORS,
47 };
48
49 extern uint8_t DMA_BUFFER_MEM_SECTION frame_buffer[153600];
50 extern uint8_t DSY_SDRAM_BSS          color_mem[153600 / 2];
51
55 class ILI9341SpiTransport
56 {
57   public:
58     void Init()
59     {
60         // Initialize SPI
61         SpiHandle::Config spi_config;
62         spi_config.periph    = SpiHandle::Config::Peripheral::SPI_1;
63         spi_config.mode      = SpiHandle::Config::Mode::MASTER;
64         spi_config.direction = SpiHandle::Config::Direction::TWO_LINES_TX_ONLY;
65         spi_config.clock_polarity  = SpiHandle::Config::ClockPolarity::LOW;
66         spi_config.baud_prescaler  = SpiHandle::Config::BaudPrescaler::PS_2;
67         spi_config.clock_phase     = SpiHandle::Config::ClockPhase::ONE_EDGE;
68         spi_config.nss             = SpiHandle::Config::NSS::HARD_OUTPUT;
69         spi_config.datasize        = 8;
70         spi_config.pin_config.sclk = {DSY_GPIOG, 11};
71         spi_config.pin_config.mosi = {DSY_GPIOB, 5};
72         spi_config.pin_config.nss  = {DSY_GPIOG, 10};
73         // spi_config.pin_config.miso = {DSY_GPIOX, 0}; // not used
74
75         // v0.1 mix up
```

```
76          //uint8_t dc_pin    = 16;
77          //uint8_t reset_pin = 22;
78
79          // DC pin
80          pin_dc_.mode = DSY_GPIO_MODE_OUTPUT_PP;
81          pin_dc_.pin  = seed::D12;
82          dsy_gpio_init(&pin_dc_);
83          // Reset pin
84          pin_reset_.mode = DSY_GPIO_MODE_OUTPUT_PP;
85          pin_reset_.pin  = seed::D11;
86          dsy_gpio_init(&pin_reset_);
87          // CS pin
88          pin_cs_.mode = DSY_GPIO_MODE_OUTPUT_PP;
89          pin_cs_.pin  = spi_config.pin_config.nss;
90          dsy_gpio_init(&pin_cs_);
91
92          spi_.Init(spi_config);
93
94          InitPalette();
95      };
96
97      void Reset()
98      {
99          dsy_gpio_write(&pin_reset_, 0);
100          System::Delay(10);
101          dsy_gpio_write(&pin_reset_, 1);
102          System::Delay(120);
103      }
104
105      /*void ClearBuffer(uint16_t color) {
106          for (size_t i = 0; i < buffer_size / 2; i++) {
107              frame_buffer[i * 2] = color >> 8;
108              frame_buffer[i * 2 + 1] = color & 0xFF; // new function
109          }
110      }*/
111
112      // an internal function to handle SPI DMA callbacks
113      // called when an DMA transmission completes and the next driver must be updated
114      static void TxCompleteCallback(void* context, SpiHandle::Result result)
115      {
116          if(result == SpiHandle::Result::OK)
117          {
118              auto transport     = static_cast<ILI9341SpiTransport*>(context);
119              auto transfer_size = transport->GetTransferSize();
120              transport->remaining_buff -= transfer_size;
121              if(transport->remaining_buff > 0)
122              {
123                  uint8_t* next_buffer_ptr = frame_buffer + (transport->buffer_size -
transport->remaining_buff);  // potential issue
124                  transport->SendDataDMA(
125                     &frame_buffer[buffer_size - transport->remaining_buff],
126                      transfer_size);
127
128
129                  // 16bit transfer             uncommented
130                  // transport->SendDataDMA(
131                    //   &frame_buffer[2 * transport->buf_chunk_size],
132                      // transport->buf_chunk_size);
133              }
134              else
135              {
136                  //System::Delay(120);
137                  transport->dma_busy = false;
138
139
140                  // 8 bit transfer
141                  // auto* spi_h = transport->spi .GetHandle();
142                  // spi_h->Instance->CFG1 &= ~SPI_CFG1_DSIZE_3;
143                  // spi_h->Instance->CFG2 &= ~SPI_CFG2_LSBFRST;
144                  // spi_h->Init.DataSize = SPI_DATASIZE_8BIT;
145              }
146          }
147      }
148
149      SpiHandle::Result SendDataDMA()
150      {
151          remaining_buff = buffer_size;
```

```
152         //System::Delay(120);
153         dma_busy       = true;
154 
155 
156         // auto* spi_h = spi_.GetHandle();
157         // spi_h->Instance->CFG1 |= SPI_CFG1_DSIZE_3;
158         // spi_h->Instance->CFG2 |= SPI_CFG2_LSBFRST;
159         // spi_h->Init.DataSize = SPI_DATASIZE_16BIT; // 15;
160 
161         return SendDataDMA(frame_buffer, buf_chunk_size);
162     };
163 
164     SpiHandle::Result SendDataDMA(uint8_t* buff, size_t size)
165     {
166         dsy_gpio_write(&pin_dc_, 1);
167         return spi_.DmaTransmit(buff, size, nullptr, &TxCompleteCallback, this);
168     };
169 
170     uint32_t GetTransferSize() const
171     {
172         return remaining_buff < buf_chunk_size ? remaining_buff
173                                                : buf_chunk_size;
174         // return remaining_buff < 2 * buf_chunk_size ? remaining_buff // uncommented
these two lines
175           //                                         : 2 * buf_chunk_size;
176     }
177 
178     SpiHandle::Result SendCommand(uint8_t cmd)
179     {
180         dsy_gpio_write(&pin_dc_, 0);
181         return spi_.BlockingTransmit(&cmd, 1);
182     };
183 
184     SpiHandle::Result SendData(uint8_t* buff, size_t size)
185     {
186         dsy_gpio_write(&pin_dc_, 1);
187         return spi_.BlockingTransmit(buff, size);
188     };
189 
190     void SetAddressWindow(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1)
191     {
192         // column address set
193         SendCommand(0x2A); // CASET
194         {
195             uint8_t data[4] = {static_cast<uint8_t>((x0 >> 8) & 0xFF),
196                                static_cast<uint8_t>(x0 & 0xFF),
197                                static_cast<uint8_t>((x1 >> 8) & 0xFF),
198                                static_cast<uint8_t>(x1 & 0xFF)};
199             SendData(data, 4);
200         }
201 
202         // row address set
203         SendCommand(0x2B); // RASET
204         {
205             uint8_t data[4] = {static_cast<uint8_t>((y0 >> 8) & 0xFF),
206                                static_cast<uint8_t>(y0 & 0xFF),
207                                static_cast<uint8_t>((y1 >> 8) & 0xFF),
208                                static_cast<uint8_t>(y1 & 0xFF)};
209             SendData(data, 4);
210         }
211 
212         // write to RAM
213         SendCommand(0x2C); // RAMWR
214     }
215 
216     void PaintPixel(uint32_t id, uint8_t color_id, uint8_t alpha = 255) const
217     {
218 
219 
220         auto color = tftPalette[color_id];
221 
222         if(alpha != 255)
223         {
224 
225 
226             auto bg_color = tftPalette[color_mem[id]];
227             color         = Blend565(color, bg_color, alpha);
```

```
228         }
229
230         color_mem[id] = color_id;
231
232         frame_buffer[id]     = color >> 8;
233         frame_buffer[id + 1] = color & 0xFF;
234     }
235
236     uint16_t GetPixel(uint32_t id) { return color_mem[id]; }
237
238     bool     dma_busy       = false;
239     uint32_t remaining_buff = 0;
240
241
242     static uint32_t const buffer_size    = 153600;          // 320 * 240 * 2   = 153600
243     const uint16_t        buf_chunk_size = buffer_size / 3; // 8bit data
244     //const uint16_t buf_chunk_size = buffer_size / 4; // 16bit data
245
246     SpiHandle                                spi_;
247
248     uint16_t tftPalette[NUMBER_OF_TFT_COLORS];
249
250   private:
251     dsy_gpio pin_reset_;
252     dsy_gpio pin_dc_;
253     dsy_gpio pin_cs_;
254
255
256     //   rrrrrgggggggbbbbb
257 #define MASK_RB 63519       // 0b1111100000011111
258 #define MASK_G 2016         // 0b0000011111100000
259 #define MASK_MUL_RB 4065216 // 0b11111000000111111000000
260 #define MASK_MUL_G 129024   // 0b00000111111000000000000
261 #define MAX_ALPHA 64        // 6bits+1 with rounding
262
263     uint16_t Blend565(uint16_t fg, uint16_t bg, uint8_t alpha) const
264     {
265         // alpha for foreground multiplication
266         // convert from 8bit to (6bit+1) with rounding
267         // will be in [0..64] inclusive
268         alpha = (alpha + 2) >> 2;
269         // "beta" for background multiplication; (6bit+1);
270         // will be in [0..64] inclusive
271         uint8_t beta = MAX_ALPHA - alpha;
272         // so (0..64)*alpha + (0..64)*beta always in 0..64
273
274         return (uint16_t)((((alpha * (uint32_t)(fg & MASK_RB)
275                              + beta * (uint32_t)(bg & MASK_RB))
276                             & MASK_MUL_RB)
277                            | ((alpha * (fg & MASK_G) + beta * (bg & MASK_G))
278                               & MASK_MUL_G))
279                           >> 6);
280     }
281
282     /*
283   result masks of multiplications
284   uppercase: usable bits of multiplications
285   RRRRRrrrrrrBBBBBbbbbbb // 5-5 bits of red+blue
286         1111100000011111 // from MASK_RB * 1
287   11111000000111111000000 //   to MASK_RB * MAX_ALPHA // 22 bits!
288
289
290   -----GGGGGGgggggg----- // 6 bits of green
291         0000011111100000 // from MASK_G * 1
292   00000111111000000000000 //   to MASK_G * MAX_ALPHA
293 */
294
295
296     void InitPalette()
297     {
298         // HEX to RBG565 converter:
https://trolsoft.ru/en/articles/rgb565-color-picker
299         tftPalette[COLOR_BLACK]       = 0x0000;
300         tftPalette[COLOR_WHITE]       = 0xffff;
301         tftPalette[COLOR_BLUE]        = 0x5AFF;
302         tftPalette[COLOR_DARK_BLUE]   = 0x18EB;
303         tftPalette[COLOR_YELLOW]      = 0xFFE0;
```

```
304         tftPalette[COLOR_DARK_YELLOW] = 0x49E1;
305         tftPalette[COLOR_RED]         = 0xF9E1; // 0xff4010
306         tftPalette[COLOR_DARK_RED]    = 0x4880; // 0x401000
307         tftPalette[COLOR_GREEN]       = 0x3FE7; // 0x40ff40
308         tftPalette[COLOR_DARK_GREEN]  = 0x01E0; // 0x004000
309         tftPalette[COLOR_LIGHT_GRAY]  = 0xAD75; // 0xb0b0b0
310         tftPalette[COLOR_MEDIUM_GRAY] = 0x8C71; // 0x909090
311         tftPalette[COLOR_GRAY]        = 0x5AEB; // 0x606060
312         tftPalette[COLOR_DARK_GRAY]   = 0x2965; // 0x303030
313         tftPalette[COLOR_CYAN]        = 0x76FD; // 0x76dfef
314         tftPalette[COLOR_ORANGE]      = 0xFBE0; // 0xff7f00
315         tftPalette[COLOR_LIGHT_GREEN] = 0x6FED; // 0x70ff70
316         // tftPalette[COLOR_ABL_BG]      = (0x2104); // 0x212121
317         tftPalette[COLOR_ABL_BG]     = 0x4A69; // #4D4D4D
318         tftPalette[COLOR_ABL_LINE]   = 0x39E7; // 0x3d3d3d
319         tftPalette[COLOR_ABL_D_LINE] = 0x31A6; // 0x363636
320         tftPalette[COLOR_ABL_L_GRAY] = 0x52AA; // 0x555555
321         tftPalette[COLOR_ABL_M_GRAY] = 0x4228; // 0x454545
322         //System::Delay(1000);
323     }
324 };
325
326
330 // class ILI9341Driver
331 class UiDriver
332 {
333   public:
334
335     enum class Orientation
336     {
337         Default = 0,
338         RRight,
339         RLeft,
340         UpsideDown,
341     };
342
343     void Init()
344     {
345         screen_update_period_ = 34; // 17 is roughly 60Hz
346         screen_update_last_   = System::GetNow();
347
348         InitDriver();
349     }
350
351     void InitDriver()
352     {
353         transport_.Init();
354
355         SetOrientation(Orientation::RLeft);
356
357         transport_.Reset();
358
359         //Software Reset
360         transport_.SendCommand(0x01);
361         System::Delay(100); // TODO: maybe less?
362
363         // command list is based on https://github.com/martnak/STM32-ILI9341
364
365         // POWER CONTROL A
366         transport_.SendCommand(0xCB);
367         {
368             uint8_t data[5] = {0x39, 0x2C, 0x00, 0x34, 0x02};
369             transport_.SendData(data, 5);
370         }
371
372         // POWER CONTROL B
373         transport .SendCommand(0xCF);
374         {
375             uint8_t data[3] = {0x00, 0xC1, 0x30};
376             transport_.SendData(data, 3);
377         }
378
379         // DRIVER TIMING CONTROL A
380         transport_.SendCommand(0xE8);
381         {
382             uint8_t data[3] = {0x85, 0x00, 0x78};
383             transport_.SendData(data, 3);
```

```
384         }
385
386         // DRIVER TIMING CONTROL B
387         transport_.SendCommand(0xEA);
388         {
389             uint8_t data[2] = {0x00, 0x00};
390             transport_.SendData(data, 2);
391         }
392
393         // POWER ON SEQUENCE CONTROL
394         transport_.SendCommand(0xED);
395         {
396             uint8_t data[4] = {0x64, 0x03, 0x12, 0x81};
397             transport_.SendData(data, 4);
398         }
399
400         // PUMP RATIO CONTROL
401         transport_.SendCommand(0xF7);
402         {
403             uint8_t data[1] = {0x20};
404             transport_.SendData(data, 1);
405         }
406
407         // POWER CONTROL,VRH[5:0]
408         transport_.SendCommand(0xC0);
409         {
410             uint8_t data[1] = {0x23};
411             transport_.SendData(data, 1);
412         }
413
414         // POWER CONTROL,SAP[2:0];BT[3:0]
415         transport_.SendCommand(0xC1);
416         {
417             uint8_t data[1] = {0x10};
418             transport_.SendData(data, 1);
419         }
420
421         // VCM CONTROL
422         transport_.SendCommand(0xC5);
423         {
424             uint8_t data[2] = {0x3E, 0x28};
425             transport_.SendData(data, 2);
426         }
427
428         // VCM CONTROL 2
429         transport_.SendCommand(0xC7);
430         {
431             uint8_t data[1] = {0x86};
432             transport_.SendData(data, 1);
433         }
434
435         // MEMORY ACCESS CONTROL
436         transport_.SendCommand(0x36);
437         {
438             uint8_t data[1] = {0x48};
439             transport_.SendData(data, 1);
440         }
441
442         // PIXEL FORMAT
443         transport_.SendCommand(0x3A);
444         {
445             uint8_t data[1] = {0x55};
446             transport_.SendData(data, 1);
447         }
448
449         // FRAME RATIO CONTROL, STANDARD RGB COLOR
450         transport .SendCommand(0xB1);
451         {
452             uint8_t data[2] = {0x00, 0x18};
453             transport_.SendData(data, 2);
454         }
455
456         // DISPLAY FUNCTION CONTROL
457         transport_.SendCommand(0xB6);
458         {
459             uint8_t data[3] = {0x08, 0x82, 0x27};
460             transport_.SendData(data, 3);
```

```
461         }
462
463         // 3GAMMA FUNCTION DISABLE
464         transport_.SendCommand(0xF2);
465         {
466             uint8_t data[1] = {0x00};
467             transport_.SendData(data, 1);
468         }
469
470         // GAMMA CURVE SELECTED
471         transport_.SendCommand(0x26);
472         {
473             uint8_t data[1] = {0x01};
474             transport_.SendData(data, 1);
475         }
476
477         // POSITIVE GAMMA CORRECTION
478         transport_.SendCommand(0xE0);
479         {
480             uint8_t data[15] = {0x0F,
481                                 0x31,
482                                 0x2B,
483                                 0x0C,
484                                 0x0E,
485                                 0x08,
486                                 0x4E,
487                                 0xF1,
488                                 0x37,
489                                 0x07,
490                                 0x10,
491                                 0x03,
492                                 0x0E,
493                                 0x09,
494                                 0x00};
495             transport_.SendData(data, 15);
496         }
497
498         // NEGATIVE GAMMA CORRECTION
499         transport_.SendCommand(0xE1);
500         {
501             uint8_t data[15] = {0x00,
502                                 0x0E,
503                                 0x14,
504                                 0x03,
505                                 0x11,
506                                 0x07,
507                                 0x31,
508                                 0xC1,
509                                 0x48,
510                                 0x08,
511                                 0x0F,
512                                 0x0C,
513                                 0x31,
514                                 0x36,
515                                 0x0F};
516             transport_.SendData(data, 15);
517         }
518
519         // EXIT SLEEP
520         transport_.SendCommand(0x11);
521         System::Delay(120);
522
523         // TURN ON DISPLAY
524         transport_.SendCommand(0x29);
525
526         // MADCTL
527         transport .SendCommand(0x36);
528         System::Delay(10);
529
530         {
531             uint8_t data[1] = {rotation};
532             transport .SendData(data, 1);
533         }
534     };
535
536     void SetOrientation(Orientation ori)
537     {
```

```
538         uint8_t ili_bgr = 0x08;
539         uint8_t ili_mx  = 0x40;
540         uint8_t ili_my  = 0x80;
541         uint8_t ili_mv  = 0x20;
542         switch(ori)
543         {
544             case Orientation::RRight:
545             {
546                 width    = 320;
547                 height   = 240;
548                 rotation = ili_mx | ili_my | ili_mv | ili_bgr;
549                 return;
550             }
551             case Orientation::RLeft:
552             {
553                 width    = 320;
554                 height   = 240;
555                 rotation = ili_mv | ili_bgr;
556                 return;
557             }
558             case Orientation::UpsideDown:
559             {
560                 width    = 240;
561                 height   = 320;
562                 rotation = ili_my | ili_bgr;
563                 return;
564             }
565             default:
566             {
567                 width    = 240;
568                 height   = 320;
569                 rotation = ili_mx | ili_bgr;
570             };
571         }
572     }
573
574     Rectangle GetDrawableFrame() const
575     {
576         return Rectangle(int16_t(width), int16_t(height))
577             .WithTrimmedTop(header)
578             .WithTrimmedBottom(footer);
579     }
580
581     Rectangle GetBounds() const
582     {
583         return Rectangle(int16_t(width), int16_t(height));
584     }
585
586     void Fill(uint8_t color)
587     {
588         for(size_t i = 0; i < transport_.buffer_size / 2; i++)
589         {
590             transport_.PaintPixel(i * 2, color);
591         }
592     };
593
594     void DrawLine(uint_fast16_t x1,
595                   uint_fast16_t y1,
596                   uint_fast16_t x2,
597                   uint_fast16_t y2,
598                   uint8_t t       color,
599                   uint8_t       alpha = 255)
600     {
601         if(x1 == x2)
602         {
603             return DrawVLine(x1, y1, y2 - y1 + 1, color, alpha);
604         }
605         else if(y1 == y2)
606         {
607             return DrawHLine(x1, y1, x2 - x1 + 1, color, alpha);
608         }
609
610         auto deltaX = abs((int_fast16_t)x2 - (int_fast16_t)x1);
611         auto deltaY = abs((int_fast16_t)y2 - (int_fast16_t)y1);
612         auto signX  = ((x1 < x2) ? 1 : -1);
613         auto signY  = ((y1 < y2) ? 1 : -1);
614         auto error  = deltaX - deltaY;
```

```
615
616         DrawPixel(x2, y2, color, alpha);
617
618         while((x1 != x2) || (y1 != y2))
619         {
620             DrawPixel(x1, y1, color, alpha);
621             auto error2 = error * 2;
622             if(error2 > -deltaY)
623             {
624                 error -= deltaY;
625                 x1 += signX;
626             }
627
628             if(error2 < deltaX)
629             {
630                 error += deltaX;
631                 y1 += signY;
632             }
633         }
634     }
635
636     void FillArea(uint_fast16_t x,
637                   uint_fast16_t y,
638                   uint_fast16_t w,
639                   uint_fast16_t h,
640                   uint8_t       color,
641                   uint8_t       alpha = 255)
642     {
643         // Loop through every Y sector
644         for(size_t i = 0; i < h; i++)
645         {
646             for(size_t j = 0; j < w; j++)
647             {
648                 DrawPixel(x + j, y + i, color, alpha);
649             }
650         }
651     };
652
653     void DrawRect(uint16_t x,
654                   uint16_t y,
655                   uint16_t w,
656                   uint16_t h,
657                   uint8_t  color,
658                   uint8_t  alpha = 255)
659     {
660         auto x2 = x + w;
661         auto y2 = y + h;
662         DrawLine(x, y, x, y2, color, alpha);
663         DrawLine(x, y, x2, y, color, alpha);
664         DrawLine(x, y2, x2, y2, color, alpha);
665         DrawLine(x2, y, x2, y2, color, alpha);
666     }
667
668     void DrawRect(const Rectangle& rect, uint8_t color, uint8_t alpha = 255)
669     {
670         DrawRect(rect.GetX(),
671                  rect.GetY(),
672                  rect.GetWidth(),
673                  rect.GetHeight(),
674                  color,
675                  alpha);
676     }
677
678     void FillRect(const Rectangle& rect, uint8_t color, uint8_t alpha = 255)
679     {
680         FillArea(rect.GetX(),
681                  rect.GetY(),
682                  rect.GetWidth(),
683                  rect.GetHeight(),
684                  color,
685                  alpha);
686     };
687
688     void DrawTriangle(int16_t x0,
689                       int16_t y0,
690                       int16_t x1,
691                       int16_t y1,
```

```
692                     int16_t x2,
693                     int16_t y2,
694                     uint8_t color,
695                     uint8_t alpha = 255)
696     {
697         DrawLine(x0, y0, x1, y1, color, alpha);
698         DrawLine(x1, y1, x2, y2, color, alpha);
699         DrawLine(x2, y2, x0, y0, color, alpha);
700     }
701
702     void FillTriangle(int16_t x0,
703                     int16_t y0,
704                     int16_t x1,
705                     int16_t y1,
706                     int16_t x2,
707                     int16_t y2,
708                     uint8_t color,
709                     uint8_t alpha = 255)
710     {
711         int16_t a, b, y, last;
712
713         // Sort coordinates by Y order (y2 >= y1 >= y0)
714         if(y0 > y1)
715         {
716             std::swap(y0, y1);
717             std::swap(x0, x1);
718         }
719         if(y1 > y2)
720         {
721             std::swap(y2, y1);
722             std::swap(x2, x1);
723         }
724         if(y0 > y1)
725         {
726             std::swap(y0, y1);
727             std::swap(x0, x1);
728         }
729
730         if(y0 == y2)
731         { // Handle awkward all-on-same-line case as its own thing
732             a = b = x0;
733             if(x1 < a)
734                 a = x1;
735             else if(x1 > b)
736                 b = x1;
737             if(x2 < a)
738                 a = x2;
739             else if(x2 > b)
740                 b = x2;
741             DrawHLine(a, y0, b - a + 1, color, alpha);
742             return;
743         }
744
745         int16_t dx01 = x1 - x0, dy01 = y1 - y0, dx02 = x2 - x0, dy02 = y2 - y0,
746                 dx12 = x2 - x1, dy12 = y2 - y1;
747         int32_t sa = 0, sb = 0;
748
749         // For upper part of triangle, find scanline crossings for segments
750         // 0-1 and 0-2.  If y1=y2 (flat-bottomed triangle), the scanline y1
751         // is included here (and second loop will be skipped, avoiding a /0
752         // error there), otherwise scanline y1 is skipped here and handled
753         // in the second loop...which also avoids a /0 error here if y0=y1
754         // (flat-topped triangle).
755         if(y1 == y2)
756         {
757             last = y1; // Include y1 scanline
758         }
759         else
760         {
761             last = y1 - 1; // Skip it
762         }
763
764         for(y = y0; y <= last; y++)
765         {
766             a = x0 + sa / dy01;
767             b = x0 + sb / dy02;
768             sa += dx01;
```

```
769             sb += dx02;
770             /* longhand:
771             a = x0 + (x1 - x0) * (y - y0) / (y1 - y0);
772             b = x0 + (x2 - x0) * (y - y0) / (y2 - y0);
773             */
774             if(a > b)
775             {
776                 std::swap(a, b);
777             }
778             DrawHLine(a, y, b - a + 1, color, alpha);
779         }
780 
781         // For lower part of triangle, find scanline crossings for segments
782         // 0-2 and 1-2.  This loop is skipped if y1=y2.
783         sa = (int32_t)dx12 * (y - y1);
784         sb = (int32_t)dx02 * (y - y0);
785         for(; y <= y2; y++)
786         {
787             a = x1 + sa / dy12;
788             b = x0 + sb / dy02;
789             sa += dx12;
790             sb += dx02;
791             /* longhand:
792                 a = x1 + (x2 - x1) * (y - y1) / (y2 - y1);
793                 b = x0 + (x2 - x0) * (y - y0) / (y2 - y0);
794             */
795             if(a > b)
796             {
797                 std::swap(a, b);
798             }
799             DrawHLine(a, y, b - a + 1, color, alpha);
800         }
801     }
802 
803 
804     Rectangle WriteStringAligned(const char*    str,
805                                  const FontDef& font,
806                                  Rectangle      boundingBox,
807                                  Alignment      alignment,
808                                  uint8_t        color)
809     {
810         const auto alignedRect
811             = GetTextRect(str, font).AlignedWithin(boundingBox, alignment);
812         WriteString(str, alignedRect.GetX(), alignedRect.GetY(), font, color);
813         return alignedRect;
814     }
815 
816     Rectangle GetTextRect(const char* text, const FontDef& font) const
817     {
818         return {int16_t(strlen(text) * font.FontWidth), font.FontHeight};
819     }
820 
821     char WriteChar(char ch, FontDef font, uint8_t color)
822     {
823         // Check if character is valid
824         if(ch < 32 || ch > 126)
825             return 0;
826 
827         // Check remaining space on current line
828         if(width < (currentX_ + font.FontWidth)
829            || height < (currentY  + font.FontHeight))
830         {
831             return 0;
832         }
833 
834         // Use the font to write
835         for(auto i = 0; i < font.FontHeight; i++)
836         {
837             auto b = font.data[(ch - 32) * font.FontHeight + i];
838             for(auto j = 0; j < font.FontWidth; j++)
839             {
840                 if((b << j) & 0x8000)
841                 {
842                     DrawPixel(currentX_ + j, (currentY_ + i), color);
843                 }
844             }
845         }
```

```
846
847         // The current space is now taken
848         SetCursor(currentX_ + font.FontWidth, currentY_);
849
850         return ch;
851     }
852
853     void WriteString(const char* str, uint16_t x, uint16_t y, FontDef font)
854     {
855         WriteString(str, x, y, font, COLOR_WHITE);
856     }
857
858     void WriteString(const char* str,
859                      uint16_t    x,
860                      uint16_t    y,
861                      FontDef     font,
862                      uint8_t     color)
863     {
864         SetCursor(x, y);
865         while(*str) // Write until null-byte
866         {
867             if(WriteChar(*str, font, color) != *str)
868             {
869                 return; // Char could not be written
870             }
871             str++; // Next char
872         }
873     }
874
875     uint16_t GetStringWidth(const char* str, FontDef font) const
876     {
877         uint16_t font_width = 0;
878         // Loop until null-byte
879         while(*str)
880         {
881             font_width += font.FontWidth;
882             str++;
883         }
884
885         return font_width;
886     }
887
888     void DrawCircle(int16_t x0, int16_t y0, int16_t r, uint8_t color)
889     {
890         int16_t f     = 1 - r;
891         int16_t ddF_x = 1;
892         int16_t ddF_y = -2 * r;
893         int16_t x     = 0;
894         int16_t y     = r;
895
896         DrawPixel(x0, y0 + r, color);
897         DrawPixel(x0, y0 - r, color);
898         DrawPixel(x0 + r, y0, color);
899         DrawPixel(x0 - r, y0, color);
900
901         while(x < y)
902         {
903             if(f >= 0)
904             {
905                 y--;
906                 ddF y += 2;
907                 f += ddF_y;
908             }
909             x++;
910             ddF_x += 2;
911             f += ddF_x;
912
913             DrawPixel(x0 + x, y0 + y, color);
914             DrawPixel(x0 - x, y0 + y, color);
915             DrawPixel(x0 + x, y0 - y, color);
916             DrawPixel(x0 - x, y0 - y, color);
917             DrawPixel(x0 + y, y0 + x, color);
918             DrawPixel(x0 - y, y0 + x, color);
919             DrawPixel(x0 + y, y0 - x, color);
920             DrawPixel(x0 - y, y0 - x, color);
921         }
922     }
```

```
923
924     void FillCircle(int16_t x0, int16_t y0, int16_t r, uint8_t color)
925     {
926         DrawLine(x0, y0, x0, y0 + 2 * r + 1, color);
927         FillCircleHelper(x0, y0, r, 3, 0, color);
928     }
929
930     void FillCircleHelper(int16_t x0,
931                           int16_t y0,
932                           int16_t r,
933                           uint8_t cornername,
934                           int16_t delta,
935                           uint8_t color)
936     {
937         int16_t f     = 1 - r;
938         int16_t ddF_x = 1;
939         int16_t ddF_y = -2 * r;
940         int16_t x     = 0;
941         int16_t y     = r;
942
943         delta++;
944
945         while(x < y)
946         {
947             if(f >= 0)
948             {
949                 y--;
950                 ddF_y += 2;
951                 f += ddF_y;
952             }
953             x++;
954             ddF_x += 2;
955             f += ddF_x;
956
957             if(cornername & 0x1)
958             {
959                 DrawLine(
960                     x0 + x, y0 - y, x0 + x, y0 - y + 2 * y + delta - 1, color);
961                 DrawLine(
962                     x0 + y, y0 - x, x0 + y, y0 - x + 2 * x + delta - 1, color);
963             }
964             if(cornername & 0x2)
965             {
966                 DrawLine(
967                     x0 - x, y0 - y, x0 - x, y0 - y + 2 * y + delta - 1, color);
968                 DrawLine(
969                     x0 - y, y0 - x, x0 - y, y0 - x + 2 * x + delta - 1, color);
970             }
971         }
972     }
973
981     void SetCursor(uint16_t x, uint16_t y)
982     {
983         currentX_ = (x >= width) ? width - 1 : x;
984         currentY_ = (y >= height) ? height - 1 : y;
985     }
986
987     void Update()
988     {
989         Start();
990         transport .SendDataDMA();
991     }
992
993     bool IsRender()
994     {
995         if(transport_.dma_busy == false)
996         {
997             diff = System::GetNow() - screen_update_last_;
998             if(diff > screen_update_period_)
999             {
1000                 UpdateFrameRate();
1001                 screen update last  = System::GetNow();
1002                 return true;
1003              }
1004          }
1005
1006          return false;
```

```
1007    }
1008
1009    void UpdateFrameRate()
1010    {
1011        ++frames;
1012        if(System::GetNow() - fps_update_last_ > 1000)
1013        {
1014            fps              = frames;
1015            frames           = 0;
1016            fps_update_last_ = System::GetNow();
1017        }
1018    }
1019
1020    uint16_t Fps() const { return fps; }
1021
1022    void TrimString(char*    str,
1023                    char*    str_trimmed,
1024                    uint16_t str_len,
1025                    uint16_t str_width,
1026                    FontDef  font) const
1027    {
1028        if(str_len <= str_width)
1029        {
1030            return;
1031        }
1032
1033        uint16_t max_chars = str_width / font.FontWidth;
1034        strncpy(str_trimmed, str, max_chars);
1035        str_trimmed[max_chars] = '\0';
1036    }
1037
1038  public:
1039    void Start() { transport_.SetAddressWindow(0, 0, width - 1, height - 1); }
1040
1041    void DrawPixel(uint_fast16_t x,
1042                   uint_fast16_t y,
1043                   uint8_t       color,
1044                   uint8_t       alpha = 255)
1045    {
1046        if(x >= width || y >= height)
1047            return;
1048
1049        auto id = 2 * (x + y * width);
1050
1051        // NOTE: Probably we should check the color id before accessing the array
1052        transport_.PaintPixel(id, color, alpha);
1053
1054        // Lets divide the whole screen in 10 sectors, 32 pixel high each
1055        //uint8_t screen_sector     = y / 32;
1056        // dirty_buff[screen_sector] = 1;
1057    }
1058
1059    void DrawVLine(int16_t x,
1060                   int16_t y,
1061                   int16_t h,
1062                   uint8_t color,
1063                   uint8_t alpha = 255)
1064    {
1065        for(int16_t i = y; i < y + h; i++)
1066        {
1067            DrawPixel(x, i, color, alpha);
1068        }
1069    }
1070
1071    void DrawHLine(int16_t x,
1072                   int16_t y,
1073                   int16_t w,
1074                   uint8_t color,
1075                   uint8_t alpha = 255)
1076    {
1077        for(int16_t i = x; i < x + w; i++)
1078        {
1079            DrawPixel(i, y, color, alpha);
1080        }
1081    }
1082
1083
```

```
1094     uint16_t Color565(uint8_t red, uint8_t green, uint8_t blue) const
1095     {
1096         return ((red & 0xF8) << 8) | ((green & 0xFC) << 3) | (blue >> 3);
1097     }
1098 
1099     uint32_t screen_update_last_, screen_update_period_, fps_update_last_;
1100 
1101     ILI9341SpiTransport transport_;
1102 
1103     uint16_t      width;
1104     uint16_t      height;
1105     uint8_t       rotation;
1106     const uint8_t header = 20;
1107     const uint8_t footer = 13;
1108     uint32_t      diff;
1109     uint16_t      frames = 0;
1110 
1111     uint16_t currentX_;
1112     uint16_t currentY_;
1113     // 2 * width * 32; // 2 bits per pixel, 32 rows
1114     // static uint16_t const num_sectors     = 10;
1115     // static uint16_t const sector_size     = buffer_size / num_sectors;
1116     // static uint16_t       dirty_buff[num_sectors]; // = {0};
1117     // = {0}; // DMA max (?) 65536 // full screen - 153600
1118 
1119     uint16_t fps = 0;
1120 };
1121 
1122 extern UiDriver lcd;
1123 
1124 #endif
```

Drivers/EncoderDriver.h File Reference

```
#include "../Parameters/SteppedParameter.h"
```

Classes

class **EncoderDriver**

EncoderDriver.h

Go to the documentation of this file.

```
1 #ifndef ENCODERDRIVER_H
2 #define ENCODERDRIVER_H
3
4 #include "../Parameters/SteppedParameter.h"
5
6 /*****************************************************************************/
14 class EncoderDriver
15 {
16 public:
17
18     /*************************************************************************/
25     void init(dsy_gpio_pin button, dsy_gpio_pin a, dsy_gpio_pin b,
std::function<void()> navCb)
26     {
27         parameters.reserve(70);
28
29         prevUpdate = 0;
30         isUpdated = false;
31
32         isNavigation = true;
33         navCallback = navCb;
34
35         currentParam = 0;
36         valueA = 0xFF;
37         valueB = 0xFF;
38
39         btn.Init(button);
40
41         channelA.pin = a;
42         channelB.pin = b;
43         channelA.mode = DSY_GPIO_MODE_INPUT;
44         channelB.mode = DSY_GPIO_MODE_INPUT;
45         channelA.pull = DSY_GPIO_PULLUP;
46         channelB.pull = DSY_GPIO_PULLUP;
47         dsy_gpio_init(&channelA);
48         dsy_gpio_init(&channelB);
49
50         // FOR TESTING
51         currentParam = 3;
52     }
53
54     /*************************************************************************/
57     void tick()
58     {
59         u_int32_t now = System::GetNow();
60         if(now - prevUpdate >= 1) // adjust to change update rate, 1 = 1000Hz, 2 = 2000Hz
etc.
61         {
62             prevUpdate = now;
63
64             // Shift Button states to debounce
65             valueA = (valueA << 1) | dsy_gpio_read(&channelA);
66             valueB = (valueB << 1) | dsy_gpio_read(&channelB);
67
68             if(!isNavigation)
69             {
70                 if((valueA & 0x03) == 0x02 && (valueB & 0x03) == 0x00)
71                     parameters[currentParam]->decrement();
72                 else if((valueB & 0x03) == 0x02 && (valueA & 0x03) == 0x00)
73                     parameters[currentParam]->increment();
74             }
75         }
76
77         btn.Debounce();
78         if(btn.Pressed())
79             buttonCallback();
80     }
81
82     /*************************************************************************/
85     void buttonCallback()
86     {
87         if(isNavigation)
```

```
88              navCallback();
89      }
90
91      /*************************************************************************/
95      bool getButtonState() { return btn.Pressed(); }
96
97
/*****************************************************************************/
100     std::function<void()> getBypassCallback(size_t index) { return
bypassCallbacks.at(index); }
101
102     /*************************************************************************/
106     void setCurrentParam(int newID)
107     {
108         for(unsigned int i = 0 ; i < parameters.size() ; i++)
109         {
110             if(parameters[i]->getID() == newID)
111             {
112                 currentParam = i;
113                 break;
114             }
115         }
116     }
117
118     /*************************************************************************/
122     void addParameter(SteppedParameter* newParam)
123     {
124         parameters.push_back(newParam);
125     }
126
127     /*************************************************************************/
131     void addBypassCallback(std::function<void()> newCallback)
132     {
133         bypassCallbacks.push_back(newCallback);
134     }
135
136     /*************************************************************************/
140     SteppedParameter* getParameter(int paramID)
141     {
142         for(unsigned int i = 0 ; i < parameters.size() ; i++)
143         {
144             if(parameters[i]->getID() == paramID)
145                 return parameters[i];
146             else
147                 return nullptr;
148         }
149     }
150
151 private:
152
153     bool isUpdated;
154     uint32_t prevUpdate;
155
156     bool isNavigation;
157     std::function<void()> navCallback;
158
159     int currentParam;
160     std::vector<SteppedParameter*> parameters;
161     std::vector<std::function<void()>> bypassCallbacks;
162
163     Switch btn;
164     dsy_gpio channelA;
165     dsy_gpio channelB;
166
167     uint8_t valueA;
168     uint8_t valueB;
169 };
170
171 #endif
```

Drivers/ILI9341_ui_driver.cpp File Reference

```
#include "daisy_ILI9341.hpp"
```

Variables

- uint8_t DMA_BUFFER_MEM_SECTION **frame_buffer** [**ILI9341SpiTransport::buffer_size**] = {0}
- uint8_t DSY_SDRAM_BSS **color_mem** [**ILI9341SpiTransport::buffer_size**/2] = {0}
- **UiDriver lcd**

Variable Documentation

uint8_t DSY_SDRAM_BSS color_mem[ILI9341SpiTransport::buffer_size/2] = {0}

uint8_t DMA_BUFFER_MEM_SECTION frame_buffer[ILI9341SpiTransport::buffer_size] = {0}

UiDriver lcd

Drivers/KeypadDriver.h File Reference

```
#include "../Parameters/BinaryParameter.h"
```

Classes

- class **KeypadDriver**struct **KeypadDriver::Index**

KeypadDriver.h

Go to the documentation of this file.

```
1 #ifndef KEYPADDRIVER_H
2 #define KEYPADDRIVER_H
3
4 #include "../Parameters/BinaryParameter.h"
5
6 /*****************************************************************************/
11 class KeypadDriver
12 {
13 public:
14     KeypadDriver() { index.row = index.col = 0; }
15
16     void init(dsy_gpio_pin t, dsy_gpio_pin b, dsy_gpio_pin l, dsy_gpio_pin r)
17     {
18         top.init(t, 1000, [this] { topPressed(); });
19         bottom.init(b, 1000, [this] { bottomPressed(); });
20         left.init(l, 1000, [this] { leftPressed(); });
21         right.init(r, 1000, [this] { rightPressed(); });
22     }
23
24     void tick()
25     {
26         top.tick();
27         bottom.tick();
28         left.tick();
29         right.tick();
30     }
31
32     struct Index
33     {
34         int row;
35         int col;
36     };
37
38     Index getIndex() const
39     {
40         return index;
41     }
42
43     bool isTopPressed() { return top.isPressed();}
44     bool isBottomPressed() { return bottom.isPressed();}
45     bool isRightPressed() { return right.isPressed();}
46     bool isLeftPressed() { return left.isPressed();}
47
48
49
50 private:
51     void topPressed()
52     {
53         index.col--; // Move up through the effects
54         wrapIndex();
55     }
56
57     void bottomPressed()
58     {
59         index.col++; // Move down through the effects
60         wrapIndex();
61     }
62
63     void leftPressed()
64     {
65         index.row++;
66         wrapIndex();
67     }
68
69     void rightPressed()
70     {
71         index.row--;
72         wrapIndex();
73     }
74
75     void wrapIndex()
76     {
```

```
77          // Wrap index.row for number of tracks
78          if(index.row < 0) index.row += 4;
79          else if(index.row >= 4) index.row -= 4;
80
81          // Wrap index.col for the number of effects
82          if(index.col < 0) index.col += 5;
83          else if(index.col >= 5) index.col -= 5;
84      }
85
86      Index index;
87      BinaryParameter top;
88      BinaryParameter bottom;
89      BinaryParameter left;
90      BinaryParameter right;
91  };
92
93  #endif
```

DSP/FX/Delay.cpp File Reference

```
#include "Delay.h"
```

DSP/FX/Delay.h File Reference

```
#include "../../Parameters/DefaultValues.h"
#include "../../Drivers/EncoderDriver.h"
```

Classes

class DelayMacros

- #define **MAXDELAY** 240000

Macro Definition Documentation

#define MAXDELAY 240000

Delay.h

Go to the documentation of this file.

```
1 #ifndef DELAY_H
2 #define DELAY_H
3
4 #include "../../Parameters/DefaultValues.h"
5 #include "../../Drivers/EncoderDriver.h"
6
7 #define MAXDELAY 240000 // 5 second max delay
8
9 /*************************************************************/
13 using namespace daisysp;
14
15 class Delay
16 {
17 public:
18
19
/*****************************************************************************/
26     void init(EncoderDriver* driver, int trackID, DelayLine<float, MAXDELAY>* dl[2]);
27
28     /*****************************************************************************/
31     void setDefaultValues();
32
33     /*****************************************************************************/
36     inline void setBypass() { isBypass = !isBypass; }
37
38     /*****************************************************************************/
41     inline void setAmount(float a) { amount.value = a; }
42
43      /*****************************************************************************/
47     inline void setDelay(size_t s) { delayLine[0]->SetDelay(s),
delayLine[1]->SetDelay(s); }
48
49      /*****************************************************************************/
53     inline void setFeedback(float f) { feedback.value = f; }
54
55     /*****************************************************************************/
60     void processBlock(float* input[2], size_t size);
61
62 private:
63
64     template<typename type>
65     type toSize(type toConvert)
66     {
67         return round((size_t)toConvert);
68     }
69
70     bool isBypass;
71     SteppedParameterWrapper amount;
72     SteppedParameterWrapper size;
73     SteppedParameterWrapper feedback;
74
75     DelayLine<float, MAXDELAY>* delayLine[2];
76 };
77
78 #endif
```

DSP/FX/Filter.cpp File Reference

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

DSP/FX/Filter.h File Reference

```
#include "../../Parameters/DefaultValues.h"
#include "../../Drivers/EncoderDriver.h"
#include "../../Utils/Constants.h"
```

Classes

class **Filter**

Filter.h

Go to the documentation of this file.

```
1 #ifndef FILTER_H
2 #define FILTER_H
3
4 #include "../../Parameters/DefaultValues.h"
5 #include "../../Drivers/EncoderDriver.h"
6 #include "../../Utils/Constants.h"
7
8
/*****************************************************************************************
*********/
12 using namespace daisysp;
13
14 class Filter
15 {
16 public:
17
18
/*************************************************************************/
24     void init(EncoderDriver* driver, int trackID);
25
26     /*************************************************************************/
29     void setDefaultValues();
30
31     /*************************************************************************/
34     void setBypass() { isBypass = !isBypass; }
35
36     /*************************************************************************/
40     void setFreq(float f) { filter.SetFrequency(f); }
41
42     /*************************************************************************/
46     void setMode(float m);
47
48     /*************************************************************************/
53     void processBlock(float* input[2], size_t size);
54
55 private:
56
57     OnePole filter;
58
59     bool isBypass;
60     SteppedParameter mode;
61     SteppedParameter freq;
62 };
63
64 #endif
```

DSP/FX/PitchShift.cpp File Reference

```
#include "PitchShift.h"
```

DSP/FX/PitchShift.h File Reference

```
#include "../../Parameters/DefaultValues.h"
#include "../../Drivers/EncoderDriver.h"
#include "../../Utils/Constants.h"
```

Classes

class **PitchShift**

PitchShift.h

Go to the documentation of this file.

```
1 #ifndef PITCHSHIFTER_H
2 #define PITCHSHIFTER_H
3
4 #include "../../Parameters/DefaultValues.h"
5 #include "../../Drivers/EncoderDriver.h"
6 #include "../../Utils/Constants.h"
7
8
/*****************************************************************************************
**/
12 using namespace daisysp;
13
14 class PitchShift
15 {
16 public:
17
18
/*************************************************************************/
24     void init(EncoderDriver* driver, int trackID);
25
26      /*************************************************************************/
29     void setDefaultValues();
30
31     /*************************************************************************/
34     void setBypass() { isBypass = !isBypass; }
35
36     /*************************************************************************/
39     void setAmount(float a) { amount.value = a; }
40
41     /*************************************************************************/
46     void processBlock(float* input[2], size_t size);
47
48 private:
49
50     float buffer[2][BLOCKLENGTH];
51     daisysp::PitchShifter shifter;
52
53     bool isBypass;
54     SteppedParameterWrapper amount;
55     SteppedParameterWrapper semitones;
56 };
57
58 #endif
```

DSP/FX/Reverb/allpass.cpp File Reference

```
#include "allpass.hpp"
```

DSP/FX/Reverb/allpass.hpp File Reference

```
#include "denormals.h"
```

Classes

class **allpass**

allpass.hpp

Go to the documentation of this file.

```
1 // Allpass filter declaration
2 //
3 // Written by Jezar at Dreampoint, June 2000
4 // http://www.dreampoint.co.uk
5 // This code is public domain
6
7 #ifndef _allpass_
8 #define _allpass_
9 #include "denormals.h"
10
11 class allpass
12 {
13 public:
14                     allpass();
15             void    setbuffer(float *buf, int size);
16     inline  float   process(float inp);
17             void    mute();
18             void    setfeedback(float val);
19             float   getfeedback();
20 // private:
21     float   feedback;
22     float   *buffer;
23     int     bufsize;
24     int     bufidx;
25 };
26
27
28 // Big to inline - but crucial for speed
29
30 inline float allpass::process(float input)
31 {
32     float output;
33     float bufout;
34
35     bufout = buffer[bufidx];
36     undenormalise(bufout);
37
38     output = -input + bufout;
39     buffer[bufidx] = input + (bufout*feedback);
40
41     if(++bufidx>=bufsize) bufidx = 0;
42
43     return output;
44 }
45
46 #endif//_allpass
47
48 //ends
```

DSP/FX/Reverb/comb.cpp File Reference

```
#include "comb.hpp"
```

DSP/FX/Reverb/comb.hpp File Reference

```
#include "denormals.h"
```

Classes

class **comb**

comb.hpp

Go to the documentation of this file.

```
1 // Comb filter class declaration
2 //
3 // Written by Jezar at Dreampoint, June 2000
4 // http://www.dreampoint.co.uk
5 // This code is public domain
6
7 #ifndef _comb_
8 #define _comb_
9
10 #include "denormals.h"
11
12 class comb
13 {
14 public:
15                     comb();
16             void    setbuffer(float *buf, int size);
17     inline  float   process(float inp);
18             void    mute();
19             void    setdamp(float val);
20             float   getdamp();
21             void    setfeedback(float val);
22             float   getfeedback();
23 private:
24     float   feedback;
25     float   filterstore;
26     float   damp1;
27     float   damp2;
28     float   *buffer;
29     int     bufsize;
30     int     bufidx;
31 };
32
33
34 // Big to inline - but crucial for speed
35
36 inline float comb::process(float input)
37 {
38     float output;
39
40     output = buffer[bufidx];
41     undenormalise(output);
42
43     filterstore = (output*damp2) + (filterstore*damp1);
44     undenormalise(filterstore);
45
46     buffer[bufidx] = input + (filterstore*feedback);
47
48     if(++bufidx>=bufsize) bufidx = 0;
49
50     return output;
51 }
52
53 #endif //_comb_
54
55 //ends
```

DSP/FX/Reverb/denormals.h File Reference

Macros

- #define **undenormalise**(**sample**) if(((*(unsigned int*)&**sample**)&0x7f800000)==0) **sample**=0.0f

---

Macro Definition Documentation

#define undenormalise(sample) if(((*(unsigned int*)&sample)&0x7f800000)==0) sample=0.0f

denormals.h

Go to the documentation of this file.

```
1 // Macro for killing denormalled numbers
2 //
3 // Written by Jezar at Dreampoint, June 2000
4 // http://www.dreampoint.co.uk
5 // Based on IS_DENORMAL macro by Jon Watte
6 // This code is public domain
7
8 #ifndef _denormals_
9 #define _denormals_
10
11 #define undenormalise(sample) if(((*(unsigned int*)&sample)&0x7f800000)==0)
sample=0.0f
12
13 #endif//_denormals_
14
15 //ends
```

DSP/FX/Reverb/Reverb.cpp File Reference

```
#include "Reverb.h"
```

DSP/FX/Reverb/Reverb.h File Reference

```
#include "../../../Parameters/DefaultValues.h"
#include "../../../Drivers/EncoderDriver.h"
#include "../../../Utils/Constants.h"
#include "revmodel.hpp"
```

Classes

class **Reverb**

Reverb.h

Go to the documentation of this file.

```
1 #ifndef REVERB_H
2 #define REVERB_H
3
4 #include "../../../Parameters/DefaultValues.h"
5 #include "../../../Drivers/EncoderDriver.h"
6 #include "../../../Utils/Constants.h"
7 #include "revmodel.hpp"
8
9 /************************************************************************/
16 using namespace daisysp;
17
18 class Reverb
19 {
20 public:
21
22
/*************************************************************************/
28     void init(EncoderDriver* driver, int trackID);
29
30     /************************************************************************/
33     void setDefaultValues();
34
35     /************************************************************************/
38     void setBypass() { isBypass = !isBypass; }
39
40     /************************************************************************/
43     void setAmount(float mix);
44
45     /*************************************************************************/
50     void processBlock(float* input[2], long size);
51
52 private:
53
54     float output[2][BLOCKLENGTH];
55     revmodel model;
56
57     bool isBypass;
58     SteppedParameterWrapper amount;
59     SteppedParameterWrapper mode;
60     SteppedParameterWrapper size;
61     SteppedParameterWrapper damp;
62     SteppedParameterWrapper width;
63 };
64
65 #endif
```

DSP/FX/Reverb/revmodel.cpp File Reference

```
#include "revmodel.hpp"
```

DSP/FX/Reverb/revmodel.hpp File Reference

```
#include "comb.hpp"
#include "allpass.hpp"
#include "tuning.h"
```

Classes

class **revmodel**

revmodel.hpp

Go to the documentation of this file.

```
1 // Reverb model declaration
2 //
3 // Written by Jezar at Dreampoint, June 2000
4 // http://www.dreampoint.co.uk
5 // This code is public domain
6
7 #ifndef _revmodel_
8 #define _revmodel_
9
10 #include "comb.hpp"
11 #include "allpass.hpp"
12 #include "tuning.h"
13
14 class revmodel
15 {
16 public:
17                     revmodel();
18             void    mute();
19             void    processmix(float *inputL, float *inputR, float *outputL, float
*outputR, long numsamples, int skip);
20             void    processreplace(float *inputL, float *inputR, float *outputL, float
*outputR, long numsamples, int skip);
21             void    setroomsize(float value);
22             float   getroomsize();
23             void    setdamp(float value);
24             float   getdamp();
25             void    setwet(float value);
26             float   getwet();
27             void    setdry(float value);
28             float   getdry();
29             void    setwidth(float value);
30             float   getwidth();
31             void    setmode(float value);
32             float   getmode();
33 private:
34             void    update();
35 private:
36     float   gain;
37     float   roomsize,roomsize1;
38     float   damp,damp1;
39     float   wet,wet1,wet2;
40     float   dry;
41     float   width;
42     float   mode;
43
44     // The following are all declared inline
45     // to remove the need for dynamic allocation
46     // with its subsequent error-checking messiness
47
48     // Comb filters
49     comb    combL[numcombs];
50     comb    combR[numcombs];
51
52     // Allpass filters
53     allpass allpassL[numallpasses];
54     allpass allpassR[numallpasses];
55
56     // Buffers for the combs
57     float   bufcombL1[combtuningL1];
58     float   bufcombR1[combtuningR1];
59     float   bufcombL2[combtuningL2];
60     float   bufcombR2[combtuningR2];
61     float   bufcombL3[combtuningL3];
62     float   bufcombR3[combtuningR3];
63     float   bufcombL4[combtuningL4];
64     float   bufcombR4[combtuningR4];
65     float   bufcombL5[combtuningL5];
66     float   bufcombR5[combtuningR5];
67     float   bufcombL6[combtuningL6];
68     float   bufcombR6[combtuningR6];
69     float   bufcombL7[combtuningL7];
70     float   bufcombR7[combtuningR7];
```

```
71      float   bufcombL8[combtuningL8];
72      float   bufcombR8[combtuningR8];
73
74      // Buffers for the allpasses
75      float   bufallpassL1[allpasstuningL1];
76      float   bufallpassR1[allpasstuningR1];
77      float   bufallpassL2[allpasstuningL2];
78      float   bufallpassR2[allpasstuningR2];
79      float   bufallpassL3[allpasstuningL3];
80      float   bufallpassR3[allpasstuningR3];
81      float   bufallpassL4[allpasstuningL4];
82      float   bufallpassR4[allpasstuningR4];
83  };
84
85  #endif//_revmodel_
86
87  //ends
```

DSP/FX/Reverb/tuning.h File Reference

Variables

- const int **numcombs** = 8
- const int **numallpasses** = 4
- const float **muted** = 0
- const float **fixedgain** = 0.015f
- const float **scalewet** = 3
- const float **scaledry** = 2
- const float **scaledamp** = 0.4f
- const float **scaleroom** = 0.28f
- const float **offsetroom** = 0.7f
- const float **initialroom** = 0.5f
- const float **initialdamp** = 0.5f
- const float **initialwet** = 1/**scalewet**
- const float **initialdry** = 0
- const float **initialwidth** = 1
- const float **initialmode** = 0
- const float **freezemode** = 0.5f
- const int **stereospread** = 23
- const int **combtuningL1** = 1116
- const int **combtuningR1** = 1116+**stereospread**
- const int **combtuningL2** = 1188
- const int **combtuningR2** = 1188+**stereospread**
- const int **combtuningL3** = 1277
- const int **combtuningR3** = 1277+**stereospread**
- const int **combtuningL4** = 1356
- const int **combtuningR4** = 1356+**stereospread**
- const int **combtuningL5** = 1422
- const int **combtuningR5** = 1422+**stereospread**
- const int **combtuningL6** = 1491
- const int **combtuningR6** = 1491+**stereospread**
- const int **combtuningL7** = 1557
- const int **combtuningR7** = 1557+**stereospread**
- const int **combtuningL8** = 1617
- const int **combtuningR8** = 1617+**stereospread**
- const int **allpasstuningL1** = 556
- const int **allpasstuningR1** = 556+**stereospread**
- const int **allpasstuningL2** = 441
- const int **allpasstuningR2** = 441+**stereospread**
- const int **allpasstuningL3** = 341
- const int **allpasstuningR3** = 341+**stereospread**
- const int **allpasstuningL4** = 225
- const int **allpasstuningR4** = 225+**stereospread**

Variable Documentation

const int allpasstuningL1 = 556

const int allpasstuningL2 = 441

const int allpasstuningL3 = 341

const int allpasstuningL4 = 225

const int allpasstuningR1 = 556+stereospread

const int allpasstuningR2 = 441+stereospread

const int allpasstuningR3 = 341+stereospread

const int allpasstuningR4 = 225+stereospread

const int combtuningL1 = 1116

const int combtuningL2 = 1188

const int combtuningL3 = 1277

const int combtuningL4 = 1356

const int combtuningL5 = 1422

const int combtuningL6 = 1491

const int combtuningL7 = 1557

const int combtuningL8 = 1617

const int combtuningR1 = 1116+stereospread

const int combtuningR2 = 1188+stereospread

const int combtuningR3 = 1277+stereospread

const int combtuningR4 = 1356+stereospread

const int combtuningR5 = 1422+stereospread

const int combtuningR6 = 1491+stereospread

const int combtuningR7 = 1557+stereospread

const int combtuningR8 = 1617+stereospread

const float fixedgain = 0.015f

const float freezemode = 0.5f

const float initialdamp = 0.5f

const float initialdry = 0

const float initialmode = 0

const float initialroom = 0.5f

const float initialwet = 1/scalewet

const float initialwidth = 1

const float muted = 0

const int numallpasses = 4

const int numcombs = 8

const float offsetroom = 0.7f

const float scaledamp = 0.4f

const float scaledry = 2

const float scaleroom = 0.28f

const float scalewet = 3

const int stereospread = 23

tuning.h

Go to the documentation of this file.

```
1 // Reverb model tuning values
2 //
3 // Written by Jezar at Dreampoint, June 2000
4 // http://www.dreampoint.co.uk
5 // This code is public domain
6
7 #ifndef _tuning_
8 #define _tuning_
9
10 const int   numcombs        = 8;
11 const int   numallpasses    = 4;
12 const float muted           = 0;
13 const float fixedgain       = 0.015f;
14 const float scalewet        = 3;
15 const float scaledry        = 2;
16 const float scaledamp       = 0.4f;
17 const float scaleroom       = 0.28f;
18 const float offsetroom      = 0.7f;
19 const float initialroom     = 0.5f;
20 const float initialdamp     = 0.5f;
21 const float initialwet      = 1/scalewet;
22 const float initialdry      = 0;
23 const float initialwidth    = 1;
24 const float initialmode     = 0;
25 const float freezemode      = 0.5f;
26 const int   stereospread    = 23;
27
28 // These values assume 44.1KHz sample rate
29 // they will probably be OK for 48KHz sample rate
30 // but would need scaling for 96KHz (or other) sample rates.
31 // The values were obtained by listening tests.
32 const int combtuningL1      = 1116;
33 const int combtuningR1      = 1116+stereospread;
34 const int combtuningL2      = 1188;
35 const int combtuningR2      = 1188+stereospread;
36 const int combtuningL3      = 1277;
37 const int combtuningR3      = 1277+stereospread;
38 const int combtuningL4      = 1356;
39 const int combtuningR4      = 1356+stereospread;
40 const int combtuningL5      = 1422;
41 const int combtuningR5      = 1422+stereospread;
42 const int combtuningL6      = 1491;
43 const int combtuningR6      = 1491+stereospread;
44 const int combtuningL7      = 1557;
45 const int combtuningR7      = 1557+stereospread;
46 const int combtuningL8      = 1617;
47 const int combtuningR8      = 1617+stereospread;
48 const int allpasstuningL1   = 556;
49 const int allpasstuningR1   = 556+stereospread;
50 const int allpasstuningL2   = 441;
51 const int allpasstuningR2   = 441+stereospread;
52 const int allpasstuningL3   = 341;
53 const int allpasstuningR3   = 341+stereospread;
54 const int allpasstuningL4   = 225;
55 const int allpasstuningR4   = 225+stereospread;
56
57 #endif// tuning
58
59 //ends
60
```

DSP/FX/Waveshaper.cpp File Reference

```
#include "Waveshaper.h"
```

DSP/FX/Waveshaper.h File Reference

```
#include "../../Parameters/DefaultValues.h"
#include "../../Drivers/EncoderDriver.h"
#include "../../Utils/Constants.h"
#include <cmath>
```

Classes

class **Waveshaper**

Waveshaper.h

Go to the documentation of this file.

```
1 #ifndef WAVESHAPER_H
2 #define WAVESHAPER_H
3
4 #include "../../Parameters/DefaultValues.h"
5 #include "../../Drivers/EncoderDriver.h"
6 #include "../../Utils/Constants.h"
7 #include <cmath>
8
9 /*****************************************************************/
19 using namespace daisysp;
20
21 class Waveshaper
22 {
23 public:
24
25
/*************************************************************************/
30     void init(EncoderDriver* driver, int trackID);
31
32     /*************************************************************************/
35     void setDefaultValues();
36
37     /*************************************************************************/
40     void setBypass() { isBypass = !isBypass; }
41
42     /*************************************************************************/
45     inline void setAmount(float a) { amount.value = a; }
46
47     /*************************************************************************/
50     inline void setFuncControl(float fc);
51
52     /*************************************************************************/
55     inline void setMode(float m) { mode.value = m; }
56
57
/*************************************************************************************
/
60     inline void scaleControlParam();
61
62     /*************************************************************************/
65     inline void setInputAG(float* buffer[2], size_t size);
66
67     /*************************************************************************/
70     inline void setOutputAG(float* buffer[2], size_t size);
71
72     /*************************************************************************/
76     inline void calculateAutoGain(size_t size);
77
78     /*************************************************************************/
83     inline void applyAutoGain(float* buffer[2], size_t size);
84
85     /*************************************************************************/
90     void processBlock(float* input[2], size_t size);
91
92     /*************************************************************************/
97     void processClipper(float* input[2], size_t size);
98
99     /*************************************************************************/
104     void processFolder(float* input[2], size_t size);
105
106     /*************************************************************************/
111     void processLFO(float* input[2], size_t size);
112
113     /*************************************************************************/
118     void processBitReducer(float* input[2], size_t size);
119
120 private:
121
122     enum Funcs
123     {
124         CLIPPER = 0,
125         FOLDER,
```

```
126         LFO,
127         BITREDUCER
128     };
129 
130     Wavefolder folder;
131     Oscillator lfo;
132 
133     float lfoMin, lfoMax;
134     float lfoFreq;
135     int bitsMin, bitsMax;
136     float bitRate, bitCount;
137     int bits;
138 
139     float buffer[2][BLOCKLENGTH];
140     float inputAG[2][BLOCKLENGTH];
141     float outputAG[2][BLOCKLENGTH];
142     float diffAG[2][BLOCKLENGTH];
143 
144     bool isBypass;
145     SteppedParameterWrapper amount;
146     SteppedParameterWrapper funcControl;
147     SteppedParameterWrapper mode;
148 };
149 
150 #endif
```

DSP/Mixer.cpp File Reference

```
#include "Mixer.h"
```

DSP/Mixer.h File Reference

```
#include "Track.h"
```

Classes

class **Mixer**

Mixer.h

Go to the documentation of this file.

```
1 #ifndef MIXER_H
2 #define MIXER_H
3
4 #include "Track.h"
5
6 /*****************************************/
11 class Mixer
12 {
13 public:
14
15     /*************************************************************/
22     void init(DaisySeed* seed, float* t1[2], float* t2[2], float* t3[2], float* t4[2]);
23
24     /*************************************************************/
31     void initMixChannels(float* m[2], float* t1[2], float* t2[2], float* t3[2], float*
t4[2]);
32
33     /*************************************************************/
41     void initFX(EncoderDriver* driver, DelayLine<float, MAXDELAY>* t1[2],
DelayLine<float, MAXDELAY>* t2[2],
42                                        DelayLine<float, MAXDELAY>* t3[2],
DelayLine<float, MAXDELAY>* t4[2]);
43     /*************************************************************/
46     void tick();
47     /*************************************************************/
53     void processInputBlock(const float* left, const float* right, size_t size);
54     /*************************************************************/
57     void panChannels(size_t size);
58     /*************************************************************/
61     void mixOutput(size_t size);
62     /*************************************************************/
67     void processOutputBlock(float* left, float* right, size_t size);
68     /*************************************************************/
74     void panMixBuffer(float* buffer[2], float pan, size t size);
75     /*************************************************************/
78     void setMixDiv();
79     /*************************************************************/
82     void setTrack1Pan(float p){ track1.pan.value = p; }
83     /*************************************************************/
86     void setTrack2Pan(float p){ track2.pan.value = p; }
87     /*************************************************************/
90     void setTrack3Pan(float p){ track3.pan.value = p; }
91     /*************************************************************/
94     void setTrack4Pan(float p){ track4.pan.value = p; }
95     /*************************************************************/
98     void setTrack1Gain(float g){ track1.gain.value = g; }
99     /*************************************************************/
102     void setTrack2Gain(float g){ track2.gain.value = g; }
103     /*************************************************************/
106     void setTrack3Gain(float g){ track3.gain.value = g; }
107     /*************************************************************/
110     void setTrack4Gain(float g){ track4.gain.value = g; }
111     /*************************************************************/
114     void setMasterVolume(float m){ master.value = m; }
115
116 private:
117
118     struct MixerChannel
119     {
120         Track track;
121         float* buffer[2];
122         AudioParameterWrapper<float> pan;
123         AudioParameterWrapper<float> gain;
124
125         inline float getCurVal(int chan, size_t index)
126         {
127             if(gain.value < 0.005 || track.getState() == STOPPED)
128                 return 0.0f;
129             else
130                 return buffer[chan][index] * gain.value;
131         }
132     };
```

```
133
134     MixerChannel track1;
135     MixerChannel track2;
136     MixerChannel track3;
137     MixerChannel track4;
138
139     int mixDiv;
140     AudioParameterWrapper<float> master;
141     float* mix[2];
142     size_t bufferSize;
143 };
144
145 #endif
```

DSP/Track.cpp File Reference

```
#include "Track.h"
```

DSP/Track.h File Reference

```
#include "FX/PitchShift.h"
#include "FX/Waveshaper.h"
#include "FX/Filter.h"
#include "FX/Delay.h"
#include "FX/Reverb/Reverb.h"
#include "../Parameters/AudioParameter.h"
#include "../Parameters/BinaryParameter.h"
#include <utility>
```

Classes

class **Track**

Track.h

Go to the documentation of this file.

```
1 #ifndef TRACK_H
2 #define TRACK_H
3
4 #include "FX/PitchShift.h"
5 #include "FX/Waveshaper.h"
6 #include "FX/Filter.h"
7 #include "FX/Delay.h"
8 #include "FX/Reverb/Reverb.h"
9 #include "../Parameters/AudioParameter.h"
10 #include "../Parameters/BinaryParameter.h"
11 #include <utility>
12
13 /*************************************************************/
18 class Track
19 {
20 public:
21
22
/*****************************************************************************************
************/
29     void init(float* mem[2], int ID, dsy_gpio_pin r, dsy_gpio_pin p);
30
31
/*****************************************************************************************
**/
36     void initFX(EncoderDriver* driver, DelayLine<float, MAXDELAY>* dl[2]);
37
38     /*********************************************************************/
41     void tick();
42
43     /*********************************************************************/
46     void resetBuffer();
47
48     /*********************************************************************/
51     void setIsRecording();
52
53     /*********************************************************************/
56     void setIsPlaying();
57
58     /*********************************************************************/
62     TrackState getState() { return state; }
63
64     /*********************************************************************/
67     void incrementWritePos();
68
69     /*********************************************************************/
72     void incrementReadPos();
73
74     /*********************************************************************/
77     size_t getReadPos() { return ph.readPos; }
78
79     /*********************************************************************/
85     void processInputBlock(const float* left, const float* right, size_t size);
86
87
/*****************************************************************************************/
92     void processOutputBlock(float* output[2], size t size);
93
94 private:
95
96     /*********************************************************************/
99     void setLoopStart()
100     {
101         startPos = bufferSize - ((startPos * SAMPLERATE) / 1000);
102     }
103     /*********************************************************************/
109     inline float calculateLoop(float input, int channel)
110     {
111         float d1 = bufferSize - ph.readPos;
112         float d2 = startPos + ph.readPos;
113         float sample = ((input * d1) + (buffer[channel][0] * d2)) / (d1 + d2);
114
```

```
115         return sample;
116     }
117 
118     int trackID;
119 
120     TrackState state;
121     struct Playhead
122     {
123         size_t writePos;
124         size_t readPos;
125 
126         void reset()
127         {
128             writePos = 0;
129             readPos = 0;
130         }
131     } ph;
132     TrackInformation ti;
133 
134     BinaryParameter record;
135     BinaryParameter play;
136 
137     float* buffer[2];
138     size_t bufferSize;
139 
140     float loopStart;
141     float startPos;
142 
143     PitchShift pitchShift;
144     Waveshaper shaper;
145     Filter filter;
146     Delay delay;
147     Reverb reverb;
148 };
149 
150 #endif
```

GUI/AudioSlider.cpp File Reference

```
#include "AudioSlider.h"
#include "../Drivers/daisy_ILI9341.hpp"
```

GUI/AudioSlider.h File Reference

```
#include "daisy_seed.h"
#include "FX/StyleSheet.h"
#include <string>
```

Classes

class **AudioSlider**

AudioSlider.h

Go to the documentation of this file.

```
1 #ifndef AUDIOSLIDER_H
2 #define AUDIOSLIDER_H
3
4 #include "daisy_seed.h"
5 #include "FX/StyleSheet.h"
6 #include <string>
7
8 /**********************************************************************/
13 using namespace daisy;
14
15 class AudioSlider
16 {
17 public:
18
19     /**********************************************************************/
24     void init(int ID, DaisySeed* seed);
25
26     /**********************************************************************/
29     void tick();
30
31     /**********************************************************************/
36     void repaint(int index, bool selected);
37
38 private:
39
40     DaisySeed* hw;
41
42     float input;
43     const float jitter = 0.01f;
44
45     int channelID;
46     bool isUpdated;
47 };
48
49 #endif
```

GUI/FX/BypassButton.h File Reference

```
#include "../../Drivers/EncoderDriver.h"
#include "StyleSheet.h"
```

Classes

class **BypassButton**

BypassButton.h

Go to the documentation of this file.

```
1 #ifndef BYPASSBUTTON_H
2 #define BYPASSBUTTON_H
3
4 #include "../../Drivers/EncoderDriver.h"
5 #include "StyleSheet.h"
6
7 /*****************************************************/
12 class BypassButton
13 {
14 public:
15
16     /*************************************************************************/
21     void init(EncoderDriver* ed, std::function<void()> bypassCallback)
22     {
23         encoder = ed;
24
25         callback = bypassCallback;
26         isSelected = false;
27         isBypassed = true;
28     }
29
30     /*************************************************************************/
33     void tick()
34     {
35         if(isSelected && encoder->getButtonState())
36         {
37             callback();
38             isBypassed = !isBypassed;
39         }
40     }
41
42     /*************************************************************************/
45     void repaint()
46     {
47         // Draw on/off button for Bypass
48         uint16_t bypassButtonY = StyleSheet::Effects::headerHeight + 15;
49         //uint16_t buttonColor = isBypassed? COLOR_GREEN : COLOR_RED;
50         Rectangle bypassButtonRect(10, bypassButtonY,
StyleSheet::Effects::bypassButtonWidth, StyleSheet::Effects::bypassButtonHeight);
51         //lcd->FillRect(bypassButtonRect, buttonColor);
52         //lcd->DrawRect(bypassButtonRect, StyleSheet::Effects::borderColor);
53         //lcd->WriteString(isBypassed ? "On" : "Off", 20,
StyleSheet::Effects::headerHeight + 20, Font_11x18, StyleSheet::Effects::textColor);
54     }
55
56     /*************************************************************************/
60     void setIsSelected(bool state) { isSelected = state; } // called when the bypass
button is selected
61
62 private:
63
64     EncoderDriver* encoder;
65     //UiDriver* lcd;
66
67     std::function<void()> callback;
68     bool isSelected;
69     bool isBypassed;
70 };
71
72 #endif
```

GUI/FX/DelayView.cpp File Reference

```
#include "DelayView.h"
#include "../../Drivers/daisy_ILI9341.hpp"
```

GUI/FX/DelayView.h File Reference

```
#include "SteppedSlider.h"
```

Classes

class **DelayView**

DelayView.h

Go to the documentation of this file.

```
1 #ifndef DELAYVIEW_H
2 #define DELAYVIEW_H
3
4 #include "SteppedSlider.h"
5
6 /*************************************************/
11 class DelayView
12 {
13 public:
14
15     /*************************************************************************/
21     void init(int ID, EncoderDriver* driver, KeypadDriver* kpd);
22
23     /*************************************************************************/
26     void tick();
27
28     /*************************************************************************/
31     void repaint();
32
33     /*************************************************************************/
36     void clear();
37
38     /*************************************************************************/
41     void setIsOpen(bool state) { isOpen = state; }
42
43     /*************************************************************************/
46     void setIsPainted(bool state) { isPainted = state; }
47
48     /*************************************************************************/
51     void setCurrentParam(int newParam) { currentParam = newParam; }
52
53 private:
54
55     int trackID;
56
57     KeypadDriver* keypad;
58
59     BypassButton bypass;
60     SteppedSlider amount;
61     SteppedSlider size;
62     SteppedSlider feedback;
63
64     int numberOfParameters = 3;
65     const char* parameterNames[3] = {"Amount", "Size", "Feedback"};
66     int currentParam = 0;
67
68     bool isOpen;
69     bool isPainted;
70     int bypassIndexes[4] = {2, 7, 12, 17};
71 };
72
73 #endif
```

GUI/FX/FilterView.cpp File Reference

```
#include "FilterView.h"
#include "../../Drivers/daisy_ILI9341.hpp"
```

GUI/FX/FilterView.h File Reference

```
#include "SteppedSlider.h"
```

Classes

class **FilterView**

FilterView.h

Go to the documentation of this file.

```
1 #ifndef FILTERVIEW_H
2 #define FILTERVIEW_H
3
4 #include "SteppedSlider.h"
5
6 /*************************************************/
11 class FilterView
12 {
13 public:
14
15
16     /*************************************************************************/
22     void init(int ID, EncoderDriver* driver, KeypadDriver* kpd);
23
24     /*************************************************************************/
27     void tick();
28
29     /*************************************************************************/
32     void repaint();
33
34     /*************************************************************************/
37     void clear();
38
39     /*************************************************************************/
42     void setIsOpen(bool state) { isOpen = state; }
43
44     /*************************************************************************/
47     void setIsPainted(bool state) { isPainted = state; }
48
49     /*************************************************************************/
52     void setCurrentParam(int newParam) { currentParam = newParam; }
53
54 private:
55
56     int trackID;
57
58     KeypadDriver* keypad;
59
60     BypassButton bypass;
61     SteppedSlider mode;
62     SteppedSlider frequency;
63     SteppedSlider resonance;
64
65     int numberOfParameters = 3;
66     const char* parameterNames[3] = {"Mode", "Frequency", "Resonance"};
67     int currentParam = 0;
68
69     bool isOpen;
70     bool isPainted;
71     int bypassIndexes[4] = {2, 7, 12, 17};
72 };
73
74 #endif
```

GUI/FX/PitchShiftView.cpp File Reference

```
#include "PitchShiftView.h"
#include "../../Drivers/daisy_ILI9341.hpp"
```

GUI/FX/PitchShiftView.h File Reference

```
#include "SteppedSlider.h"
```

Classes

class PitchShiftView**Macros**

- #define **PITCHSHIFTVIEW_H**

Macro Definition Documentation

#define PITCHSHIFTVIEW_H

PitchShiftView.h

Go to the documentation of this file.

```
1 #ifndef PITCHSHIFTIEW_H
2 #define PITCHSHIFTVIEW_H
3
4 #include "SteppedSlider.h"
5
6 /*************************************************/
11 class PitchShiftView
12 {
13 public:
14
15
16     /*************************************************************************/
22     void init(int ID, EncoderDriver* driver, KeypadDriver* kpd);
23
24     /*************************************************************************/
27     void tick();
28
29     /*************************************************************************/
32     void repaint();
33
34     /*************************************************************************/
37     void clear();
38
39     /*************************************************************************/
42     void setIsOpen(bool state) { isOpen = state; }
43
44     /*************************************************************************/
47     void setIsPainted(bool state) { isPainted = state; }
48
49     /*************************************************************************/
52     void setCurrentParam(int newParam) { currentParam = newParam; }
53
54 private:
55
56     int trackID;
57
58     KeypadDriver* keypad;
59
60     BypassButton bypass;
61     SteppedSlider amount;
62     SteppedSlider semitones;
63
64     int numberOfParameters = 2;
65     const char* parameterNames[2] = {"Amount", "Semitones"};
66     int currentParam = 0;
67
68     bool isOpen;
69     bool isPainted;
70     int bypassIndexes[4] = {0, 5, 10, 15};
71 };
72
73 #endif
```

GUI/FX/ReverbView.cpp File Reference

```
#include "ReverbView.h"
#include "../../Drivers/daisy_ILI9341.hpp"
```

GUI/FX/ReverbView.h File Reference

```
#include "SteppedSlider.h"
```

Classes

class **ReverbView**

ReverbView.h

Go to the documentation of this file.

```
1 #ifndef REVERBVIEW_H
2 #define REVERBVIEW_H
3
4 #include "SteppedSlider.h"
5
6 /************************************************/
11 class ReverbView
12 {
13 public:
14
15     /*************************************************************************/
21     void init(int ID, EncoderDriver* driver, KeypadDriver* kpd);
22
23     /*************************************************************************/
26     void tick();
27
28     /*************************************************************************/
31     void repaint();
32
33     /*************************************************************************/
36     void clear();
37
38     /*************************************************************************/
41     void setIsOpen(bool state) { isOpen = state; }
42
43     /*************************************************************************/
46     void setIsPainted(bool state) { isPainted = state; }
47
48     /*************************************************************************/
51     void setCurrentParam(int newParam) { currentParam = newParam; }
52
53 private:
54
55     int trackID;
56
57     KeypadDriver* keypad;
58
59     BypassButton bypass;
60     SteppedSlider amount;
61     SteppedSlider size;
62
63     int numberOfParameters = 2;
64     const char* parameterNames[2] = {"Amount", "Size"};
65     int currentParam = 0;
66
67     bool isOpen;
68     bool isPainted;
69     int bypassIndexes[4] = {4, 9, 14, 19};
70 };
71
72 #endif
```

GUI/FX/SteppedSlider.cpp File Reference

```
#include "SteppedSlider.h"
#include "../../Drivers/daisy_ILI9341.hpp"
```

GUI/FX/SteppedSlider.h File Reference

```
#include "../../Drivers/KeypadDriver.h"
#include "BypassButton.h"
```

Classes

class **SteppedSlider**

SteppedSlider.h

Go to the documentation of this file.

```
1 #ifndef STEPPEDSLIDER_H
2 #define STEPPEDSLIDER_H
3
4 #include "../../Drivers/KeypadDriver.h"
5 #include "BypassButton.h"
6
7 /*****************************************************/
12 class SteppedSlider
13 {
14 public:
15
16     /*************************************************************************/
21     void init(int ID, EncoderDriver* ed);
22
23     /*************************************************************************/
26     void tick();
27
28     /*************************************************************************/
31     void setSelected();
32
33     /*************************************************************************/
36     void repaint();
37
38 private:
39
40     EncoderDriver* encoder;
41
42     int paramID;
43     SteppedParameter* param;
44     float curVal;
45 };
46
47 #endif
```

GUI/FX/StyleSheet.h File Reference

Namespaces

- namespace **StyleSheet**
- namespace **StyleSheet::Mixer**
- namespace **StyleSheet::Tracks**
- namespace **StyleSheet::Effects**

StyleSheet.h

Go to the documentation of this file.

```
1 #ifndef STYLESHEET_H
2 #define STYLESHEET_H
3
4 namespace StyleSheet
5 {
6     namespace Mixer
7     {
8         const static int screenWidth = 320;
9         const static int screenHeight = 240;
10         const static int faderWidth = 30;
11         const static int faderHeight = 100;
12         const static int faderSpacing = 35;
13         const static int firstFaderX = (screenWidth - (faderWidth * 4 + faderSpacing
* 3)) / 2;
14         const static int mixerBottom = 180;
15         const static uint16_t frameColor = 5; // CYAN
16         const static uint16_t textColor = 2; // WHITE
17         const static int frameThickness = 2;
18     }
19
20     namespace Tracks
21     {
22         static uint16_t borderColor = 5; // CYAN
23         static uint16_t text_color = 2; // WHITE
24         static uint16_t effectBoxColor = 1; // BLACK
25         static uint16_t selectedEffectColor = 11;
26
27         // Define fonts
28         static const FontDef& smallerFont = Font_7x10;
29         static const FontDef& largerFont = Font_16x26;
30         static const int smallerFontHeight = 10;
31
32         // Define the display dimensions
33         static const int displayWidth = 320;
34         static const int displayHeight = 240;
35
36         // Top position for effect boxes
37         static int effectsYStart = 96;
38     }
39
40     namespace Effects
41     {
42         const static uint16_t screenWidth = 320;
43         const static uint16_t screenHeight = 240;
44         const static uint16_t borderColor = 5;
45         const static uint16_t textColor = 5;
46         const static uint16_t sliderBackColor = 1;
47         const static uint16_t headerHeight = 50;
48         const static uint16_t paramSpacing = 35; // Adjust spacing as needed
49         const static uint16_t sliderHeight = 20; // Adjust height as needed
50         const static uint16_t sliderWidth = 180; // Slider width
51         const static uint16_t frameThickness = 2;
52         const static uint16_t sliderOffsetX = 130; // X position for sliders
53         const static uint16_t bypassButtonWidth = 60;
54         const static uint16_t bypassButtonHeight = 30;
55     }
56 }
57
58 #endif
```

GUI/FX/WaveshaperView.cpp File Reference

```
#include "WaveshaperView.h"
#include "../../Drivers/daisy_ILI9341.hpp"
```

GUI/FX/WaveshaperView.h File Reference

```
#include "SteppedSlider.h"
```

Classes

class **WaveshaperView**

WaveshaperView.h

Go to the documentation of this file.

```
1 #ifndef WAVESHAPERVIEW_H
2 #define WAVESHAPERVIEW_H
3
4 #include "SteppedSlider.h"
5
6 /*************************************************/
11 class WaveshaperView
12 {
13 public:
14
15     /*************************************************************************/
21     void init(int ID, EncoderDriver* driver, KeypadDriver* kpd);
22
23     /*************************************************************************/
26     void tick();
27
28     /*************************************************************************/
31     void repaint();
32
33     /*************************************************************************/
36     void clear();
37
38     /*************************************************************************/
41     void setIsOpen(bool state) { isOpen = state; }
42
43     /*************************************************************************/
46     void setIsPainted(bool state) { isPainted = state; }
47
48     /*************************************************************************/
51     void setCurrentParam(int newParam) { currentParam = newParam; }
52
53 private:
54
55     int trackID;
56
57     KeypadDriver* keypad;
58
59     BypassButton bypass;
60     SteppedSlider amount;
61     SteppedSlider funcControl;
62     SteppedSlider mode;
63
64     int numberOfParameters = 3;
65     const char* parameterNames[3] = {"Amount", "Function Control", "Mode"};
66     int currentParam = 0;
67
68     bool isOpen;
69     bool isPainted;
70     int bypassIndexes[4] = {1, 6, 11, 16};
71 };
72
73 #endif
```

GUI/MixerView.cpp File Reference

```
#include "MixerView.h"
#include "../Drivers/daisy_ILI9341.hpp"
```

GUI/MixerView.h File Reference

```
#include "AudioSlider.h"
#include "TrackView.h"
```

Classes

class **MixerView**

MixerView.h

Go to the documentation of this file.

```
1 #ifndef MIXERVIEW_H
2 #define MIXERVIEW_H
3
4 #include "AudioSlider.h"
5 #include "TrackView.h"
6
7 /*****************************************/
12 class MixerView
13 {
14 public:
15
16     /*************************************************************************/
22     void init(DaisySeed* seed, EncoderDriver* encoder, KeypadDriver* kpd);
23
24     /*************************************************************************/
27     void tick();
28
29     /*************************************************************************/
32     void repaint();
33
34     /*************************************************************************/
37     void clear();
38
39     /*************************************************************************/
42     void setIsOpen(bool state) { isOpen = state; }
43
44     /*************************************************************************/
47     void setIsPainted(bool state) { isPainted = state; }
48
49     /*************************************************************************/
53     void setSelectedChannel(int channel) { selectedChannel = channel; }
54
55 private:
56
57     KeypadDriver* keypad;
58
59     AudioSlider amp1;
60     AudioSlider amp2;
61     AudioSlider amp3;
62     AudioSlider amp4;
63
64     int selectedChannel;
65     std::array<AudioSlider, 4> ampSliders = {amp1, amp2, amp3, amp4};
66
67     TrackView track1;
68     TrackView track2;
69     TrackView track3;
70     TrackView track4;
71
72     bool isOpen;
73     bool isPainted;
74 };
75
76 #endif
```

GUI/TrackView.cpp File Reference

```
#include "TrackView.h"
#include "../Drivers/daisy_ILI9341.hpp"
```

GUI/TrackView.h File Reference

```
#include "../Parameters/ParameterIDs.h"
#include "FX/PitchShiftView.h"
#include "FX/WaveshaperView.h"
#include "FX/FilterView.h"
#include "FX/ReverbView.h"
#include "FX/DelayView.h"
```

Classes

class **TrackView**

TrackView.h

Go to the documentation of this file.

```
1 #ifndef TRACKVIEW_H
2 #define TRACKVIEW_H
3
4 #include "../Parameters/ParameterIDs.h"
5 #include "FX/PitchShiftView.h"
6 #include "FX/WaveshaperView.h"
7 #include "FX/FilterView.h"
8 #include "FX/ReverbView.h"
9 #include "FX/DelayView.h"
10
11 /*****************************************/
16 class TrackView
17 {
18 public:
19
20     /*************************************************************************/
26     void init(int ID, EncoderDriver* driver,  KeypadDriver* kpd);
27
28     /*************************************************************************/
31     void tick();
32
33     /*************************************************************************/
36     void repaint();
37
38     /*************************************************************************/
41     void clear();
42
43     /*************************************************************************/
46     void setIsOpen(bool state) { isOpen = state; }
47
48 private:
49
50     int trackID;
51
52     //static UiDriver* lcd;
53     KeypadDriver* keypad;
54
55     PitchShiftView pitchShiftView;
56     WaveshaperView waveshaperView;
57     FilterView filterView;
58     ReverbView reverbView;
59     DelayView delayView;
60
61     int selectedEffectIndex;
62
63     bool isOpen;
64     bool isPainted;
65 };
66
67 #endif
```

LoopyTunes.cpp File Reference

```
#include "DSP/Mixer.h"
#include "GUI/MixerView.h"
#include "Drivers/daisy_ILI9341.hpp"
```

Namespaces

- namespace **Buffers**

Functions

- void **navCallback** ()
- void **init** ()
  Initialises the program and hardware.

- void **tick** (size_t size)
  Handles the processing of the inputs and updating of the audio processing and GUI branches.

- void **AudioCallback** (AudioHandle::InputBuffer in, AudioHandle::OutputBuffer out, size_t size)
  This is where audio data is inputted to and outputted from the software.

- int **main** (void)

Variables

- DaisySeed **hw**
  Function description.

- size_t **sample**
- **Mixer mixer**
- **MixerView mixerView**
- **EncoderDriver encoder**
- **KeypadDriver keypad**
- **UiDriver lcd**
- float DSY_SDRAM_BSS **Buffers::track1** [2][**SAMPLERATE** ***DURATION**]
- float DSY_SDRAM_BSS **Buffers::track2** [2][**SAMPLERATE** ***DURATION**]
- float DSY_SDRAM_BSS **Buffers::track3** [2][**SAMPLERATE** ***DURATION**]
- float DSY_SDRAM_BSS **Buffers::track4** [2][**SAMPLERATE** ***DURATION**]
- float * **Buffers::track1Ptr** [2] = {**track1**[**L**], **track1**[**R**]}
- float * **Buffers::track2Ptr** [2] = {**track2**[**L**], **track2**[**R**]}
- float * **Buffers::track3Ptr** [2] = {**track3**[**L**], **track3**[**R**]}
- float * **Buffers::track4Ptr** [2] = {**track4**[**L**], **track4**[**R**]}
- float DSY_SDRAM_BSS **Buffers::mix** [2][**BLOCKLENGTH**]
- float DSY_SDRAM_BSS **Buffers::t1m** [2][**BLOCKLENGTH**]
- float DSY_SDRAM_BSS **Buffers::t2m** [2][**BLOCKLENGTH**]
- float DSY_SDRAM_BSS **Buffers::t3m** [2][**BLOCKLENGTH**]
- float DSY_SDRAM_BSS **Buffers::t4m** [2][**BLOCKLENGTH**]
- float * **Buffers::mixPtr** [2] = {**mix**[**L**], **mix**[**R**]}
- float * **Buffers::t1mPtr** [2] = {**t1m**[**L**], **t1m**[**R**]}
- float * **Buffers::t2mPtr** [2] = {**t2m**[**L**], **t2m**[**R**]}
- float * **Buffers::t3mPtr** [2] = {**t3m**[**L**], **t3m**[**R**]}
- float * **Buffers::t4mPtr** [2] = {**t4m**[**L**], **t4m**[**R**]}
- DelayLine< float, **MAXDELAY** > DSY_SDRAM_BSS **Buffers::t1delay** [2]
- DelayLine< float, **MAXDELAY** > DSY_SDRAM_BSS **Buffers::t2delay** [2]

- DelayLine< float, **MAXDELAY** > DSY_SDRAM_BSS **Buffers::t3delay** [2]
- DelayLine< float, **MAXDELAY** > DSY_SDRAM_BSS **Buffers::t4delay** [2]
- DelayLine< float, **MAXDELAY** > * **Buffers::t1delayPtr** [2] = {&**t1delay**[**L**], &**t1delay**[**R**]}
- DelayLine< float, **MAXDELAY** > * **Buffers::t2delayPtr** [2] = {&**t2delay**[**L**], &**t2delay**[**R**]}
- DelayLine< float, **MAXDELAY** > * **Buffers::t3delayPtr** [2] = {&**t3delay**[**L**], &**t3delay**[**R**]}
- DelayLine< float, **MAXDELAY** > * **Buffers::t4delayPtr** [2] = {&**t4delay**[**L**], &**t4delay**[**R**]}

Function Documentation

void AudioCallback (AudioHandle::InputBuffer *in*, AudioHandle::OutputBuffer *out*, size_t *size*)

This is where audio data is inputted to and outputted from the software.

Parameters

in	The input to the Daisy Seed
out	The output to the Daisy Seed
size	The number of samples in the block

void init ()

Initialises the program and hardware.

int main (void)

void navCallback ()

void tick (size_t *size*) `[inline]`

Handles the processing of the inputs and updating of the audio processing and GUI branches.

Parameters

size	The size of the block of samples

Variable Documentation

EncoderDriver encoder

DaisySeed hw

Function description.

Parameters

arg1	
arg2	
arg3	

KeypadDriver keypad

UiDriver lcd

Mixer mixer

MixerView mixerView

size_t sample

Parameters/AudioParameter.h File Reference

```
#include "daisy_seed.h"
#include "daisysp.h"
#include "../Utils/Helpers.h"
#include "../Utils/Constants.h"
#include <cstdint>
#include <functional>
#include <memory>
#include <type_traits>
```

Classes

- class **AudioParameter< type >**struct **AudioParameterWrapper< type >**

AudioParameter.h

Go to the documentation of this file.

```
1 #ifndef AUDIOPARAMETER_H
2 #define AUDIOPARAMETER_H
3
4 #include "daisy_seed.h"
5 #include "daisysp.h"
6 #include "../Utils/Helpers.h"
7 #include "../Utils/Constants.h"
8 #include <cstdint>
9 #include <functional>
10 #include <memory>
11 #include <type_traits>
12
13 /*****************************************************************************/
18 using namespace daisy;
19
20 template <class type>
21 class AudioParameter
22 {
23 public:
24
25
/*****************************************************************************
*/
34     void init(DaisySeed* seed, type mi, type ma, CurveType c, int ID,
std::function<void(type)> cb)
35     {
36         hw = seed;
37         input = 0;
38
39         curVal = 0;
40         min = mi;
41         max = ma;
42
43         curve = c;
44         channelID = ID;
45         callback = cb;
46     }
47
48     /*****************************************************************/
51     inline void tick()
52     {
53         float newInput = hw->adc.GetFloat(channelID);
54         if(newInput > (input + jitter) || newInput < (input - jitter))
55         {
56             input = newInput;
57             processCurve();
58             callback(curVal);
59         }
60     }
61
62     /*****************************************************************************/
65     void processCurve()
66     {
67         switch(curve)
68         {
69             case LINEAR:
70                 curVal = ((input / 1.0f) * (max - min)) + min;
71             break;
72             case EXP:
73                 curVal = ((input * input) * (max - min)) + min;
74             break;
75         }
76     }
77
78     /*********************************************************/
82     inline type getValue() { return curVal; }
83
84 private:
85
86     DaisySeed* hw;
87     float input;
88     const float jitter = 0.01f;
```

```
89
90      type curVal, min, max;
91      int channelID;
92      CurveType curve;
93
94      std::function<void(type)> callback;
95  };
96
97  template <class type>
98  struct AudioParameterWrapper
99  {
100      AudioParameter<type> param;
101      type value;
102  };
103
104  #endif
```

Parameters/BinaryParameter.h File Reference

```
#include "daisy_seed.h"
#include <functional>
```

Classes

- class **BinaryParameter**struct **BinaryParameterWrapper**

BinaryParameter.h

Go to the documentation of this file.

```
1 #ifndef BINARYPARAMETER_H
2 #define BINARYPARAMETER_H
3
4 #include "daisy_seed.h"
5 #include <functional>
6
7 /*****************************************************************************/
12 using namespace daisy;
13
14 class BinaryParameter
15 {
16 public:
17
18     /*************************************************************/
24     inline void init(dsy_gpio_pin pin, float updateRate, std::function<void()> cb)
25     {
26         btn.Init(pin, updateRate);
27         callback = cb;
28     }
29
30     /*********************************************************/
33     inline void tick()
34     {
35         btn.Debounce();
36         if(btn.FallingEdge())
37             callback();
38     }
39
40     /*********************************************************/
44     bool isPressed() { return btn.Pressed(); }
45
46 private:
47
48     Switch btn;
49     std::function<void()> callback;
50 };
51
52 struct BinaryParameterWrapper
53 {
54     BinaryParameter param;
55     bool value;
56 };
57 #endif
```

Parameters/DefaultValues.h File Reference

```
#include <functional>
```

Classes

- struct **PitchShifterDefaults**
- struct **WaveshaperDefaults**
- struct **FilterDefaults**
- struct **ReverbDefaults**
- struct **DelayDefaults**

DefaultValues.h

Go to the documentation of this file.

```
1 #ifndef DEFAULTVALUES_H
2 #define DEFAULTVALUES_H
3
4 #include <functional>
5
6 static const struct PitchShifterDefaults
7 {
8     bool bypass = true;
9     float amount = 0.5;
10     float semitones = 12;
11     float rand = 0;
12 } pitchShifterDefs;
13
14 static const struct WaveshaperDefaults
15 {
16     bool bypass = true;
17     float amount = 0.5;
18     float funcControl = 0;
19     float mode = 0;
20 } waveshaperDefs;
21
22 static const struct FilterDefaults
23 {
24     bool bypass = true;
25     float mode = 0;
26     float frequency = 0.5;
27 } filterDefs;
28
29 static const struct ReverbDefaults
30 {
31     bool bypass = true;
32     float amount = 0.5;
33     float mode = 0;
34     float size = 0.5;
35     float damp = 0.5;
36     float width = 0.5;
37 } reverbDefs;
38
39 static const struct DelayDefaults
40 {
41     bool bypass = true;
42     float amount = 0.5;
43     size_t size = 5000;
44     float feedback = 0.5;
45 } delayDefs;
46
47 #endif
```

Parameters/ParameterIDs.h File Reference

Namespaces

- namespace **ParameterIDs**
- namespace **ParameterIDs::Tracks**
- namespace **ParameterIDs::PitchShifter**
- namespace **ParameterIDs::Waveshaper**
- namespace **ParameterIDs::Filter**
- namespace **ParameterIDs::Delay**
- namespace **ParameterIDs::Reverb**

Variables

- const int **ParameterIDs::Tracks::Track1** = 100
- const int **ParameterIDs::Tracks::Track2** = 200
- const int **ParameterIDs::Tracks::Track3** = 300
- const int **ParameterIDs::Tracks::Track4** = 400
- const int **ParameterIDs::PitchShifter::effect** = 10
- const int **ParameterIDs::PitchShifter::amount** = **effect** + 1
- const int **ParameterIDs::PitchShifter::semitones** = **effect** + 2
- const int **ParameterIDs::PitchShifter::random** = **effect** + 3
- const int **ParameterIDs::Waveshaper::effect** = 20
- const int **ParameterIDs::Waveshaper::amount** = **effect** + 1
- const int **ParameterIDs::Waveshaper::funcControl** = **effect** + 2
- const int **ParameterIDs::Waveshaper::mode** = **effect** + 3
- const int **ParameterIDs::Filter::effect** = 30
- const int **ParameterIDs::Filter::mode** = **effect** + 1
- const int **ParameterIDs::Filter::frequency** = **effect** + 2
- const int **ParameterIDs::Filter::resonance** = **effect** + 3
- const int **ParameterIDs::Delay::effect** = 40
- const int **ParameterIDs::Delay::amount** = **effect** + 1
- const int **ParameterIDs::Delay::size** = **effect** + 2
- const int **ParameterIDs::Delay::feedback** = **effect** + 3
- const int **ParameterIDs::Reverb::effect** = 50
- const int **ParameterIDs::Reverb::amount** = **effect** + 1
- const int **ParameterIDs::Reverb::mode** = **effect** + 2
- const int **ParameterIDs::Reverb::size** = **effect** + 3
- const int **ParameterIDs::Reverb::damp** = **effect** + 4
- const int **ParameterIDs::Reverb::width** = **effect** + 5

ParameterIDs.h

Go to the documentation of this file.

```
1 #ifndef PARAMETERIDS_H
2 #define PARAMETERIDS_H
3
4 namespace ParameterIDs
5 {
6     namespace Tracks
7     {
8         const int Track1 = 100;
9         const int Track2 = 200;
10         const int Track3 = 300;
11         const int Track4 = 400;
12     }
13
14     namespace PitchShifter
15     {
16         const int effect = 10;
17
18         const int amount = effect + 1;
19         const int semitones = effect + 2;
20         const int random = effect + 3;
21     }
22
23     namespace Waveshaper
24     {
25         const int effect = 20;
26
27         const int amount = effect + 1;
28         const int funcControl = effect + 2;
29         const int mode = effect + 3;
30     }
31
32     namespace Filter
33     {
34         const int effect = 30;
35
36         const int mode = effect + 1;
37         const int frequency = effect + 2;
38         const int resonance = effect + 3;
39     }
40
41     namespace Delay
42     {
43         const int effect = 40;
44
45         const int amount = effect + 1;
46         const int size = effect + 2;
47         const int feedback = effect + 3;
48     }
49
50     namespace Reverb
51     {
52         const int effect = 50;
53
54         const int amount = effect + 1;
55         const int mode = effect + 2;
56         const int size = effect + 3;
57         const int damp = effect + 4;
58         const int width = effect + 5;
59     }
60 }
61
62 #endif
```

Parameters/SteppedParameter.h File Reference

```
#include "daisy_seed.h"
#include "daisysp.h"
#include "ParameterIDs.h"
#include <string>
#include <functional>
```

Classes

- class **SteppedParameter**struct **SteppedParameterWrapper**

SteppedParameter.h

Go to the documentation of this file.

```
1 #ifndef STEPPEDPARAMETER_H
2 #define STEPPEDPARAMETER_H
3
4 #include "daisy_seed.h"
5 #include "daisysp.h"
6 #include "ParameterIDs.h"
7 #include <string>
8 #include <functional>
9
10
/*****************************************************************************/
15 using namespace daisy;
16
17 class SteppedParameter
18 {
19 public:
20
21     /*************************************************************/
30     void init(float mi, float ma, float st, int param, int track,
std::function<void(float)> cb)
31     {
32         paramID = param + track;
33
34         min = mi;
35         max = ma;
36         step = st;
37
38         callback = cb;
39     }
40
41     /*************************************************************/
44     void increment()
45     {
46         if(curVal >= max)
47             curVal = max;
48         else
49             curVal += step;
50
51         callback(curVal);
52     }
53
54     /*************************************************************/
57     void decrement()
58     {
59         if(curVal <= min)
60             curVal = min;
61         else
62             curVal -= step;
63
64         callback(curVal);
65     }
66
67     /*************************************************************/
71     int getID() { return paramID; }
72
73     /*************************************************************/
77     float getMin() { return min; }
78
79     /*************************************************************/
83     float getMax() { return max; }
84
85     /*************************************************************/
89     float getCurVal() { return curVal; }
90
91 private:
92
93     int paramID;
94     float min;
95     float max;
96     volatile float curVal;
97     float step;
98
```

```
99     std::function<void(float)> callback;
100 };
101
102 struct SteppedParameterWrapper
103 {
104     SteppedParameter param;
105     float value;
106 };
107
108 #endif
```

README.md File Reference

Utils/Constants.h File Reference

```
#include "stddef.h"
```

Macros

- #define **BLOCKLENGTH** 4
- #define **SAMPLERATE** 48000
- #define **DURATION** 10
- #define **TRACKCOUNT** 4
- #define **L** 0
- #define **R** 1
- #define **ADCINPUTS** 5
- #define **MACROBLOCK** 512
- #define **NO_OPT** __attribute__((optimize(0)))

Macro Definition Documentation

#define ADCINPUTS 5

#define BLOCKLENGTH 4

#define DURATION 10

#define L 0

#define MACROBLOCK 512

#define NO_OPT __attribute__((optimize(0)))

#define R 1

#define SAMPLERATE 48000

#define TRACKCOUNT 4

Constants.h

Go to the documentation of this file.

```
1 #include "stddef.h"
2
3 #define BLOCKLENGTH 4
4 #define SAMPLERATE 48000
5 #define DURATION 10
6
7 #define TRACKCOUNT 4
8
9 #define L 0
10 #define R 1
11
12 #define ADCINPUTS 5
13 #define MACROBLOCK 512
14
15 #define NO_OPT __attribute__((optimize(0)))
```

Utils/Helpers.h File Reference

```
#include "stddef.h"
#include <algorithm>
```

Classes

struct **TrackInformation***Struct definition for storing track information.*

Enumerations

- enum **ChannelIDs** { **AMP1** = 0, **AMP2**, **AMP3**, **AMP4**, **MASTER** }
  Represents the channel IDs for easy assignment and access.
- enum **TrackState** { **RECORDING** = 0, **PLAYING**, **STOPPED** }
  Struct definition for monitoring track state.
- enum **CurveType** { **LINEAR** = 0, **EXP** }
  Struct definition for types of curve used to process values.

Enumeration Type Documentation

enum ChannelIDs

Represents the channel IDs for easy assignment and access.

Enumerator:

AMP1	
AMP2	
AMP3	
AMP4	
MASTER	

enum CurveType

Struct definition for types of curve used to process values.

Enumerator:

LINEAR	
EXP	

enum TrackState

Struct definition for monitoring track state.

Enumerator:

RECORDING	
PLAYING	
STOPPED	

Helpers.h

Go to the documentation of this file.

```
1 #include "stddef.h"
2 #include <algorithm>
3
4 // ADC Channel IDs
5 /*****************************************************************************/
8 enum ChannelIDs
9 {
10     AMP1 = 0,
11     AMP2,
12     AMP3,
13     AMP4,
14     MASTER
15 };
16
17 // Track information
18 /*****************************************************************************/
21 struct TrackInformation
22 {
23     bool isEmpty;
24     size_t loopLength;
25 };
26
27 // Track state
28 /*****************************************************************************/
31 enum TrackState
32 {
33     RECORDING = 0,
34     PLAYING,
35     STOPPED
36 };
37
38 // Curve types
39 /*****************************************************************************/
42 enum CurveType
43 {
44     LINEAR = 0,
45     EXP
46 };
```

Index

INDEX