PhaseNoiseDataProcessorLib

Generated by Doxygen 1.13.2

1 Namespace Index 1

1.1 Namespace List . . . 1

2 Hierarchical Index 3

2.1 Class Hierarchy . . . 3

3 Class Index 5

3.1 Class List . . . 5

4 File Index 7

4.1 File List . . . 7

5 Namespace Documentation 9

5.1 PhaseNoiseDataProcessor Namespace Reference . . . 9

5.1.1 Function Documentation . . . 11

5.1.1.1 calc_x_values() . . . 11

5.1.1.2 calc_y_values() [1/2] . . . 11

5.1.1.3 calc_y_values() [2/2] . . . 11

5.1.1.4 clear_meas_buffer() . . . 11

5.1.1.5 clear_output_buffer() . . . 11

5.1.1.6 convert_samples() . . . 11

5.1.1.7 dbv_to_dbc() . . . 12

5.1.1.8 filter_samples() . . . 12

5.1.1.9 get_x_values() . . . 12

5.1.1.10 get_y_values() . . . 12

5.1.1.11 overwrite_samples() . . . 12

5.1.1.12 process_fixed_smpl_rate() . . . 12

5.1.1.13 process_fixed_window_size() . . . 13

5.1.1.14 push_data_fixed_smpl_rate() . . . 13

5.1.1.15 push_data_fixed_window_size() . . . 13

5.1.1.16 read_meas_from_file() . . . 13

5.1.1.17 retrieve_samples() . . . 13

5.1.1.18 setup() [1/2] . . . 14

5.1.1.19 setup() [2/2] . . . 14

5.1.1.20 start_fixed_smpl_rate() . . . 14

5.1.1.21 start_fixed_window_size() . . . 14

5.1.1.22 stop() . . . 14

5.1.1.23 write_output_to_file() . . . 14

5.1.2 Variable Documentation . . . 15

5.1.2.1 band_data . . . 15

5.1.2.2 config . . . 15

5.1.2.3 fft_im_buffer . . . 15

5.1.2.4 fft_re_buffer . . . 15

5.1.2.5 meas_buffer . . . 15

5.1.2.6 meas_count 15
5.1.2.7 output_im_buffer 15
5.1.2.8 output_re_buffer 16
5.1.2.9 output_x_buffer 16
5.1.2.10 output_y_buffer 16
5.1.2.11 smpl_buffer 16

6 Class Documentation **17**

6.1 AntiAliasingFilter Class Reference 17
6.1.1 Detailed Description 17
6.1.2 Constructor & Destructor Documentation 18
6.1.2.1 AntiAliasingFilter() 18
6.1.3 Member Function Documentation 18
6.1.3.1 output() 18
6.1.3.2 pass() 18
6.1.3.3 reset() 18
6.1.4 Member Data Documentation 18
6.1.4.1 filt1 18
6.1.4.2 filt2 18
6.1.4.3 filt3 18
6.1.4.4 output_value 19
6.1.4.5 Q1 19
6.1.4.6 Q2 19
6.1.4.7 Q3 19
6.2 Band Class Reference 19
6.2.1 Detailed Description 20
6.2.2 Member Data Documentation 20
6.2.2.1 dft_end_pos 20
6.2.2.2 dft_size 20
6.2.2.3 dft_start_pos 20
6.2.2.4 num_sets 20
6.2.2.5 output_start_pos 20
6.2.2.6 smpl_rate_div 20
6.2.2.7 window_size 21
6.3 BandData Class Reference 21
6.3.1 Detailed Description 21
6.3.2 Constructor & Destructor Documentation 22
6.3.2.1 BandData() [1/2] 22
6.3.2.2 BandData() [2/2] 22
6.3.3 Member Data Documentation 22
6.3.3.1 band_mult 22
6.3.3.2 band_shift 22

6.3.3.3 bands 22
6.3.3.4 fixed_smpl_rate 22
6.3.3.5 min_window_size 23
6.3.3.6 num_bands 23
6.3.3.7 output_size 23
6.3.3.8 smpl_data_size 23
6.4 BiquadFilter Class Reference 23
6.4.1 Detailed Description 24
6.4.2 Constructor & Destructor Documentation 24
6.4.2.1 BiquadFilter() [1/2] 24
6.4.2.2 BiquadFilter() [2/2] 24
6.4.3 Member Function Documentation 24
6.4.3.1 generateCoefficients() 24
6.4.3.2 output() 24
6.4.3.3 pass() 25
6.4.3.4 reset() 25
6.4.4 Member Data Documentation 25
6.4.4.1 feedback_buffer 25
6.4.4.2 feedback_coeff 25
6.4.4.3 input_buffer 25
6.4.4.4 input_coeff 25
6.4.4.5 output_value 25
6.5 PhaseNoiseDataProcessor::Config Struct Reference 25
6.5.1 Member Data Documentation 26
6.5.1.1 band_mult 26
6.5.1.2 band_shift 26
6.5.1.3 conversion_fac 26
6.5.1.4 cross_corr 26
6.5.1.5 data_size 27
6.5.1.6 num_bands 27
6.5.1.7 num_meas 27
6.5.1.8 packet_size 27
6.5.1.9 smpl_rate 27
6.5.1.10 window_en 27
6.6 FlatTop Class Reference 28
6.6.1 Detailed Description 28
6.6.2 Constructor & Destructor Documentation 28
6.6.2.1 FlatTop() 28
6.6.3 Member Function Documentation 28
6.6.3.1 generate() 28
6.6.3.2 rbwFactor() 29
6.7 TrigTable Class Reference 29

6.7.1 Detailed Description 29
6.7.2 Constructor & Destructor Documentation 29
6.7.2.1 TrigTable() [1/2] 29
6.7.2.2 TrigTable() [2/2] 29
6.7.3 Member Function Documentation 30
6.7.3.1 cos_lookup() 30
6.7.3.2 sin_lookup() 30
6.8 Window Class Reference 30
6.8.1 Detailed Description 31
6.8.2 Constructor & Destructor Documentation 31
6.8.2.1 Window() 31
6.8.3 Member Function Documentation 31
6.8.3.1 generate() 31
6.8.3.2 rbwFactor() 31
6.8.4 Member Data Documentation 31
6.8.4.1 data 31
6.8.4.2 rbw_factor 32
6.8.4.3 size 32

7 File Documentation **33**
7.1 band_data.h File Reference 33
7.2 band_data.h 33
7.3 fft.cpp File Reference 34
7.3.1 Function Documentation 35
7.3.1.1 auto_corr_real() [1/2] 35
7.3.1.2 auto_corr_real() [2/2] 35
7.3.1.3 cross_corr() [1/2] 36
7.3.1.4 cross_corr() [2/2] 36
7.3.1.5 cross_corr_dual_real() [1/2] 36
7.3.1.6 cross_corr_dual_real() [2/2] 36
7.3.1.7 dft() 37
7.3.1.8 fft() [1/2] 37
7.3.1.9 fft() [2/2] 37
7.3.1.10 pfft() [1/2] 37
7.3.1.11 pfft() [2/2] 38
7.3.1.12 pfft_dual_real() [1/2] 38
7.3.1.13 pfft_dual_real() [2/2] 38
7.3.1.14 rdft() 38
7.3.1.15 rfft() [1/2] 39
7.3.1.16 rfft() [2/2] 39
7.3.1.17 rpfft() [1/2] 39
7.3.1.18 rpfft() [2/2] 39

7.3.1.19 set_data_size() . . . 39
7.3.2 Variable Documentation . . . 40
7.3.2.1 g_data_size . . . 40
7.3.2.2 g_trig . . . 40
7.4 fft.h File Reference . . . 40
7.4.1 Function Documentation . . . 41
7.4.1.1 auto_corr_real() [1/2] . . . 41
7.4.1.2 auto_corr_real() [2/2] . . . 41
7.4.1.3 cross_corr() [1/2] . . . 41
7.4.1.4 cross_corr() [2/2] . . . 41
7.4.1.5 cross_corr_dual_real() [1/2] . . . 42
7.4.1.6 cross_corr_dual_real() [2/2] . . . 42
7.4.1.7 dft() . . . 42
7.4.1.8 fft() [1/2] . . . 42
7.4.1.9 fft() [2/2] . . . 43
7.4.1.10 pfft() [1/2] . . . 43
7.4.1.11 pfft() [2/2] . . . 43
7.4.1.12 pfft_dual_real() [1/2] . . . 43
7.4.1.13 pfft_dual_real() [2/2] . . . 44
7.4.1.14 rdft() . . . 44
7.4.1.15 rfft() [1/2] . . . 44
7.4.1.16 rfft() [2/2] . . . 44
7.4.1.17 rpfft() [1/2] . . . 44
7.4.1.18 rpfft() [2/2] . . . 45
7.4.1.19 set_data_size() . . . 45
7.4.2 Variable Documentation . . . 45
7.4.2.1 g_data_size . . . 45
7.4.2.2 g_trig . . . 45
7.5 fft.h . . . 45
7.6 filter.h File Reference . . . 46
7.6.1 Macro Definition Documentation . . . 46
7.6.1.1 _USE_MATH_DEFINES . . . 46
7.7 filter.h . . . 47
7.8 PhaseNoiseDataProcessorLib.cpp File Reference . . . 48
7.9 PhaseNoiseDataProcessorLib.h File Reference . . . 50
7.9.1 Macro Definition Documentation . . . 51
7.9.1.1 _USE_MATH_DEFINES . . . 51
7.9.1.2 MAX_DATA_SIZE . . . 51
7.9.1.3 MAX_MEAS . . . 52
7.9.1.4 MAX_PACKET_SIZE . . . 52
7.9.1.5 OUTPUT_FILENAME . . . 52
7.9.1.6 SMPL_FILENAME . . . 52

7.10 PhaseNoiseDataProcessorLib.h 52
7.11 trig_table.h File Reference 53
7.11.1 Macro Definition Documentation 54
7.11.1.1 _USE_MATH_DEFINES 54
7.12 trig_table.h 54
7.13 window.h File Reference 54
7.13.1 Macro Definition Documentation 55
7.13.1.1 _USE_MATH_DEFINES 55
7.14 window.h 55

Index **57**

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

PhaseNoiseDataProcessor 9

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

AntiAliasingFilter 17
Band 19
BandData 21
BiquadFilter 23
PhaseNoiseDataProcessor::Config 25
TrigTable 29
Window 30
    FlatTop 28

Chapter 3

Class Index

3.1 Class List

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

AntiAliasingFilter
6th order Butterworth digital anti-aliasing filter 17
Band
Data structure for a single phase noise measurement band 19
BandData
Generates data for a set of phase noise measurement bands based on the supplied parameters 21
BiquadFilter
Digital biquadratic filter 23
PhaseNoiseDataProcessor::Config 25
FlatTop
Flat top window for minimal scalloping loss 28
TrigTable
Generates a lookup table of precalculated sine/cosine values 29
Window
Abstract class for a sampling window of given size, which defines the shape and resolution bandwidth scaling factor 30

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

band_data.h . . . 33
fft.cpp . . . 34
fft.h . . . 40
filter.h . . . 46
PhaseNoiseDataProcessorLib.cpp . . . 48
PhaseNoiseDataProcessorLib.h . . . 50
trig_table.h . . . 53
window.h . . . 54

Chapter 5

Namespace Documentation

5.1 PhaseNoiseDataProcessor Namespace Reference

Classes

- struct Config

Functions

- void convert_samples (int channel, const void ∗data_bytes)

    Convert raw input sample byte data to voltage values and store to meas buffer for the selected channel.
- void calc_x_values ()

    Calculate and store x-axis frequency output values.
- void calc_y_values (uint32_t num_meas)

    Calculate and store y-axis power spectral density output values.
- void calc_y_values ()
- void dbv_to_dbc (double rf_input_level, double cal_input_level, double cal_output_level)

    Convert y-axis power spectral density output values from dBV/Hz to dBc/Hz based on the specified calibration settings.
- bool read_meas_from_file (char ∗filename, uint32_t meas_num)

    Read sample data from file for a single measurement and store to data buffers.
- bool write_output_to_file (char ∗filename)

    Write output data points to file.
- void retrieve_samples (uint32_t set_num, uint32_t smpl_rate_div)

    Retrieve a sample packet containing a single data set from meaurement data buffers and store in sample buffers (only used for fixed window size processing)
- void filter_samples (AntiAliasingFilter &filter_0, AntiAliasingFilter &filter_1)

    Apply a digital filter to sample buffers (only used for fixed window size processing)
- void overwrite_samples (uint32_t packet_num)

    Overwrite selected packet in measurement data buffers with the contents of sample buffers (only used for fixed window size processing)
- void clear_meas_buffer ()

    Set all values in measurement data buffers to 0.
- void clear_output_buffer ()

    Set all values in output buffers to 0.
- void process_fixed_window_size ()

Process sample data from file to produce a set of output data points using a fixed window size.
- void setup (bool cross_corr, double smpl_rate, uint32_t num_meas, uint32_t data_size, uint32_t min_window_size, uint32_t num_bands, uint32_t band_mult, uint32_t band_shift, double conversion_fac)

  Setup processing functions based on the specified parameters.
- void setup (Config new_config)
- const std::vector< double > & get_x_values ()

  Return vector containing x-axis frequency output values.
- const std::vector< double > & get_y_values ()

  Return vector containing y-axis power spectral density output values.
- void process_fixed_smpl_rate ()

  Process sample data from file to produce a set of output data points using a fixed sampling rate.
- void start_fixed_window_size ()

  Start fixed window size processing where sample data is pushed in real time.
- bool push_data_fixed_window_size (double *data_0, double *data_1)

  Push sample data for a single measurement to be processed with a fixed window size.
- void start_fixed_smpl_rate ()

  Start fixed sample rate processing where sample data is pushed in real time.
- bool push_data_fixed_smpl_rate (double *data_0, double *data_1)

  Push sample data for a single measurement to be processed with a fixed sample rate.
- void stop ()

  Stop real time processing and release data buffers.

Variables

- std::vector< double > meas_buffer [2]

  Sample buffers for captured data from a single measurement.
- std::vector< double > smpl_buffer [2]

  Sample buffers for a single sample window.
- std::vector< double > fft_re_buffer

  FFT real output buffer.
- std::vector< double > fft_im_buffer

  FFT imaginary output buffer.
- std::vector< double > output_re_buffer

  Buffer to store the sum of real values of cross correlations / auto correlations.
- std::vector< double > output_im_buffer

  Buffer to store the sum of imaginary values of cross correlations / auto correlations.
- std::vector< double > output_x_buffer

  Buffer to store x-axis frequency output values.
- std::vector< double > output_y_buffer

  Buffer to store y-axis power spectral density output values.
- uint32_t meas_count = 0

  Number of measurements that have been processed since processing was started.
- BandData band_data

  Generated data descibing the properties of each measurement band.
- Config config

  Measurement configuration settings.

5.1.1 Function Documentation

5.1.1.1 calc_x_values()

```
void PhaseNoiseDataProcessor::calc_x_values ()
```

Calculate and store x-axis frequency output values.

Values in Hz are calculated based on configuration settings and generated band data. Should be called after selected processing has been started.

5.1.1.2 calc_y_values() [1/2]

```
void PhaseNoiseDataProcessor::calc_y_values ()
```

5.1.1.3 calc_y_values() [2/2]

```
void PhaseNoiseDataProcessor::calc_y_values (
            uint32_t num_meas)
```

Calculate and store y-axis power spectral density output values.

Values in dBV/Hz are calculated based on the combined data of all processed measurements since processing was started. Should be called after measurement data are pushed for real-time monitoring of phase noise.

5.1.1.4 clear_meas_buffer()

```
void PhaseNoiseDataProcessor::clear_meas_buffer ()
```

Set all values in measurement data buffers to 0.

5.1.1.5 clear_output_buffer()

```
void PhaseNoiseDataProcessor::clear_output_buffer ()
```

Set all values in output buffers to 0.

5.1.1.6 convert_samples()

```
void PhaseNoiseDataProcessor::convert_samples (
            int channel,
            const void * data_bytes)
```

Convert raw input sample byte data to voltage values and store to meas buffer for the selected channel.

Byte data should be in the format of consecutive 16-bit half-words each representing one sample. Samples are converted to a voltage in the range of -(*conversion_fac*) to +(*conversion_fac*) using an offset binary representation (not 2's complement).

5.1.1.7 dbv_to_dbc()

```
void PhaseNoiseDataProcessor::dbv_to_dbc (
            double rf_input_level,
            double cal_input_level,
            double cal_output_level)
```

Convert y-axis power spectral density output values from dBV/Hz to dBc/Hz based on the specified calibration settings.

rf_input_level is the level of the carrier signal at the output of the device under test in dBV. *cal_input_level* is the level of the calibration tone applied at the output of the device under test in dBV. *cal_output_level* is the level of the calibration tone measured at the input of the sample capture device in dBV.

5.1.1.8 filter_samples()

```
void PhaseNoiseDataProcessor::filter_samples (
            AntiAliasingFilter & filter_0,
            AntiAliasingFilter & filter_1)
```

Apply a digital filter to sample buffers (only used for fixed window size processing)

This is a 6th-order digital Butterworth filter with a cutoff at (*sampl_rate* / *band_mult*). It allows for under-sampling of the data (such as with the retrieve_samples() function) with reduced aliasing artefacts.

5.1.1.9 get_x_values()

```
const std::vector< double > & PhaseNoiseDataProcessor::get_x_values ()
```

Return vector containing x-axis frequency output values.

5.1.1.10 get_y_values()

```
const std::vector< double > & PhaseNoiseDataProcessor::get_y_values ()
```

Return vector containing y-axis power spectral density output values.

5.1.1.11 overwrite_samples()

```
void PhaseNoiseDataProcessor::overwrite_samples (
            uint32_t packet_num)
```

Overwrite selected packet in measurement data buffers with the contents of sample buffers (only used for fixed window size processing)

5.1.1.12 process_fixed_smpl_rate()

```
void PhaseNoiseDataProcessor::process_fixed_smpl_rate ()
```

Process sample data from file to produce a set of output data points using a fixed sampling rate.

Frequency bands are generated with a fixed sampling rate and variable window size (more accurate).

5.1.1.13 process_fixed_window_size()

```
void PhaseNoiseDataProcessor::process_fixed_window_size ()
```

Process sample data from file to produce a set of output data points using a fixed window size.

Frequency bands are generated with a fixed window size and variable sampling rate (faster).

5.1.1.14 push_data_fixed_smpl_rate()

```
bool PhaseNoiseDataProcessor::push_data_fixed_smpl_rate (
            double * data_0,
            double * data_1)
```

Push sample data for a single measurement to be processed with a fixed sample rate.

5.1.1.15 push_data_fixed_window_size()

```
bool PhaseNoiseDataProcessor::push_data_fixed_window_size (
            double * data_0,
            double * data_1)
```

Push sample data for a single measurement to be processed with a fixed window size.

5.1.1.16 read_meas_from_file()

```
bool PhaseNoiseDataProcessor::read_meas_from_file (
            char * filename,
            uint32_t meas_num)
```

Read sample data from file for a single measurement and store to data buffers.

Byte data should be in the format of consecutive 16-bit half-words each representing one sample. Samples are converted to a voltage in the range of -(*conversion_fac*) to +(*conversion_fac*) using an offset binary representation (not 2's complement).

5.1.1.17 retrieve_samples()

```
void PhaseNoiseDataProcessor::retrieve_samples (
            uint32_t set_num,
            uint32_t smpl_rate_div)
```

Retrieve a sample packet containing a single data set from meaurement data buffers and store in sample buffers (only used for fixed window size processing)

Setting *smpl_rate_div* to a number greater than 1 allows the sample rate to be effectively reduced by skipping samples. (e.g. setting *smpl_rate_div* to 8 will retrieve the first in every 8 samples)

5.1.1.18 setup() [1/2]

```
void PhaseNoiseDataProcessor::setup (
            bool cross_corr,
            double smpl_rate,
            uint32_t num_meas,
            uint32_t data_size,
            uint32_t min_window_size,
            uint32_t num_bands,
            uint32_t band_mult,
            uint32_t band_shift,
            double conversion_fac)
```

Setup processing functions based on the specified parameters.

5.1.1.19 setup() [2/2]

```
void PhaseNoiseDataProcessor::setup (
            Config new_config)
```

5.1.1.20 start_fixed_smpl_rate()

```
void PhaseNoiseDataProcessor::start_fixed_smpl_rate ()
```

Start fixed sample rate processing where sample data is pushed in real time.

5.1.1.21 start_fixed_window_size()

```
void PhaseNoiseDataProcessor::start_fixed_window_size ()
```

Start fixed window size processing where sample data is pushed in real time.

5.1.1.22 stop()

```
void PhaseNoiseDataProcessor::stop ()
```

Stop real time processing and release data buffers.

5.1.1.23 write_output_to_file()

```
bool PhaseNoiseDataProcessor::write_output_to_file (
            char * filename)
```

Write output data points to file.

The file is in in CSV format and contains the properties of each frequency band followed by the list of power spectral density output values against frequency.

5.1.2 Variable Documentation

5.1.2.1 band_data

`BandData PhaseNoiseDataProcessor::band_data`

Generated data descibing the properties of each measurement band.

5.1.2.2 config

`Config PhaseNoiseDataProcessor::config`

Measurement configuration settings.

5.1.2.3 fft_im_buffer

`std::vector< double > PhaseNoiseDataProcessor::fft_im_buffer`

FFT imaginary output buffer.

5.1.2.4 fft_re_buffer

`std::vector< double > PhaseNoiseDataProcessor::fft_re_buffer`

FFT real output buffer.

5.1.2.5 meas_buffer

`std::vector< double > PhaseNoiseDataProcessor::meas_buffer`

Sample buffers for captured data from a single measurement.

5.1.2.6 meas_count

`uint32_t PhaseNoiseDataProcessor::meas_count = 0`

Number of measurements that have been processed since processing was started.

5.1.2.7 output_im_buffer

`std::vector< double > PhaseNoiseDataProcessor::output_im_buffer`

Buffer to store the sum of imaginary values of cross correlations / auto correlations.

5.1.2.8 output_re_buffer

`std::vector< double > PhaseNoiseDataProcessor::output_re_buffer`

Buffer to store the sum of real values of cross correlations / auto correlations.

5.1.2.9 output_x_buffer

`std::vector< double > PhaseNoiseDataProcessor::output_x_buffer`

Buffer to store x-axis frequency output values.

5.1.2.10 output_y_buffer

`std::vector< double > PhaseNoiseDataProcessor::output_y_buffer`

Buffer to store y-axis power spectral density output values.

5.1.2.11 smpl_buffer

`std::vector< double > PhaseNoiseDataProcessor::smpl_buffer`

Sample buffers for a single sample window.

Chapter 6

Class Documentation

6.1 AntiAliasingFilter Class Reference

6th order Butterworth digital anti-aliasing filter

```
#include <filter.h>
```

Public Member Functions

- AntiAliasingFilter (int cutoff_div)

  Instatiate the anti-aliasing filter and generate coefficients for the selected cutoff division.
- float pass (double input)

  Passe the next sample into the filter and return the corresponding output sample.
- void reset ()

  Reset all of the internal sample buffers of the filter to 0.
- double output ()

  Return the current output sample.

Protected Attributes

- const double Q1 = 1.931851989228
- const double Q2 = 0.707106562373
- const double Q3 = 0.517637997114
- BiquadFilter filt1
- BiquadFilter filt2
- BiquadFilter filt3
- double output_value = 0

6.1.1 Detailed Description

6th order Butterworth digital anti-aliasing filter

This filter is used to prepare data for under-sampling with reduced aliasing artefacts.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 AntiAliasingFilter()

```
AntiAliasingFilter::AntiAliasingFilter (
            int cutoff_div) [inline]
```

Instatiate the anti-aliasing filter and generate coefficients for the selected cutoff division.

6.1.3 Member Function Documentation

6.1.3.1 output()

```
double AntiAliasingFilter::output () [inline]
```

Return the current output sample.

6.1.3.2 pass()

```
float AntiAliasingFilter::pass (
            double input) [inline]
```

Passe the next sample into the filter and return the corresponding output sample.

6.1.3.3 reset()

```
void AntiAliasingFilter::reset () [inline]
```

Reset all of the internal sample buffers of the filter to 0.

6.1.4 Member Data Documentation

6.1.4.1 filt1

```
BiquadFilter AntiAliasingFilter::filt1 [protected]
```

6.1.4.2 filt2

```
BiquadFilter AntiAliasingFilter::filt2 [protected]
```

6.1.4.3 filt3

```
BiquadFilter AntiAliasingFilter::filt3 [protected]
```

6.1.4.4 output_value

```
double AntiAliasingFilter::output_value = 0  [protected]
```

6.1.4.5 Q1

```
const double AntiAliasingFilter::Q1 = 1.931851989228  [protected]
```

6.1.4.6 Q2

```
const double AntiAliasingFilter::Q2 = 0.707106562373  [protected]
```

6.1.4.7 Q3

```
const double AntiAliasingFilter::Q3 = 0.517637997114  [protected]
```

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

- filter.h

6.2 Band Class Reference

Data structure for a single phase noise measurement band.

```
#include <band_data.h>
```

Public Attributes

- uint32_t num_sets = 0

  Number of separate data sets that a single measurement is split into (equal to the number of correlation/autocorrelations performed)
- uint16_t smpl_rate_div = 0

  Ratio of under-sampling relative to the sampling rate of the captured data.
- uint32_t output_start_pos = 0

  Start position in the output data buffers of where output data for this band should be written.
- uint32_t window_size = 0

  The sampling window size for a single data set.
- uint32_t dft_size = 0

  Size of the output of the Fourier transform (only positive frequencies)
- uint32_t dft_start_pos = 0

  Start position of useful output data from Fourier transforms calculated for this band.
- uint32_t dft_end_pos = 0

  End position of useful output data from Fourier transforms calculated for this band.

6.2.1 Detailed Description

Data structure for a single phase noise measurement band.

6.2.2 Member Data Documentation

6.2.2.1 dft_end_pos

```
uint32_t Band::dft_end_pos = 0
```

End position of useful output data from Fourier transforms calculated for this band.

6.2.2.2 dft_size

```
uint32_t Band::dft_size = 0
```

Size of the output of the Fourier transform (only positive frequencies)

6.2.2.3 dft_start_pos

```
uint32_t Band::dft_start_pos = 0
```

Start position of useful output data from Fourier transforms calculated for this band.

6.2.2.4 num_sets

```
uint32_t Band::num_sets = 0
```

Number of separate data sets that a single measurement is split into (equal to the number of correlation/autocorrelations performed)

6.2.2.5 output_start_pos

```
uint32_t Band::output_start_pos = 0
```

Start position in the output data buffers of where output data for this band should be written.

6.2.2.6 smpl_rate_div

```
uint16_t Band::smpl_rate_div = 0
```

Ratio of under-sampling relative to the sampling rate of the captured data.

6.2.2.7 window_size

```
uint32_t Band::window_size = 0
```

The sampling window size for a single data set.

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

- band_data.h

6.3 BandData Class Reference

Generates data for a set of phase noise measurement bands based on the supplied parameters.

```
#include <band_data.h>
```

Public Member Functions

- BandData ()
- BandData (uint16_t num_bands, uint16_t band_mult, uint32_t smpl_data_size, uint32_t min_window_size, uint16_t band_shift, bool fixed_smpl_rate)

    Instatiates the BandData object and generates the Bands.

Public Attributes

- uint32_t smpl_data_size = 0

    Number of samples per channel in a single measurement data capture (must be a power of 2)
- uint16_t band_mult = 1

    Frequency multiplier for subsequent bands (must be a power of 2)
- uint16_t num_bands = 0

    Number of frequency bands to use.
- uint32_t min_window_size = 0

    FFT window size to use for the highest frequency band (must be a power of 2)
- uint16_t band_shift = 0

    Shifts each band down in multiples of band_mult by the specified number (prioritises more correlations over smaller RBW for higher frequencies)
- bool fixed_smpl_rate = true

    True if each band uses a fixed sample rate (more accurate), false if each band uses a fixed window size (faster)
- uint32_t output_size = 0

    Number of output data points that will be generated based on current settings.
- std::vector< Band > bands

    Vector array of band data structures.

6.3.1 Detailed Description

Generates data for a set of phase noise measurement bands based on the supplied parameters.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 BandData() [1/2]

```
BandData::BandData ()  [inline]
```

6.3.2.2 BandData() [2/2]

```
BandData::BandData (
            uint16_t num_bands,
            uint16_t band_mult,
            uint32_t smpl_data_size,
            uint32_t min_window_size,
            uint16_t band_shift,
            bool fixed_smpl_rate)  [inline]
```

Instatiates the BandData object and generates the Bands.

6.3.3 Member Data Documentation

6.3.3.1 band_mult

```
uint16_t BandData::band_mult = 1
```

Frequency multiplier for subsequent bands (must be a power of 2)

6.3.3.2 band_shift

```
uint16_t BandData::band_shift = 0
```

Shifts each band down in multiples of band_mult by the specified number (prioritises more correlations over smaller RBW for higher frequencies)

6.3.3.3 bands

```
std::vector<Band> BandData::bands
```

Vector array of band data structures.

6.3.3.4 fixed_smpl_rate

```
bool BandData::fixed_smpl_rate = true
```

True if each band uses a fixed sample rate (more accurate), false if each band uses a fixed window size (faster)

6.3.3.5 min_window_size

```
uint32_t BandData::min_window_size = 0
```

FFT window size to use for the highest frequency band (must be a power of 2)

6.3.3.6 num_bands

```
uint16_t BandData::num_bands = 0
```

Number of frequency bands to use.

6.3.3.7 output_size

```
uint32_t BandData::output_size = 0
```

Number of output data points that will be generated based on current settings.

6.3.3.8 smpl_data_size

```
uint32_t BandData::smpl_data_size = 0
```

Number of samples per channel in a single measurement data capture (must be a power of 2)

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

- band_data.h

6.4 BiquadFilter Class Reference

Digital biquadratic filter.

```
#include <filter.h>
```

Public Member Functions

- BiquadFilter ()
- BiquadFilter (int cutoff_div, double q)

  Instatiate the biquad filter and generate coefficients for the selected cutoff division and Q factor.
- void generateCoefficients (int cutoff_div, double q)

  Generate coefficients for the selected cutoff division and Q factor.
- double pass (double input)

  Passe the next sample into the filter and return the corresponding output sample.
- void reset ()

  Reset all of the internal sample buffers of the filter to 0.
- double output ()

  Return the current output sample.

Protected Attributes

- double input_coeff [3] = { 0, 0, 0 }
- double feedback_coeff [2] = { 0, 0 }
- double input_buffer [3] = { 0, 0, 0 }
- double feedback_buffer [2] = { 0, 0 }
- double output_value = 0

6.4.1 Detailed Description

Digital biquadratic filter.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 BiquadFilter() [1/2]

```
BiquadFilter::BiquadFilter ()  [inline]
```

6.4.2.2 BiquadFilter() [2/2]

```
BiquadFilter::BiquadFilter (
            int cutoff_div,
            double q)  [inline]
```

Instatiate the biquad filter and generate coefficients for the selected cutoff division and Q factor.

6.4.3 Member Function Documentation

6.4.3.1 generateCoefficients()

```
void BiquadFilter::generateCoefficients (
            int cutoff_div,
            double q)  [inline]
```

Generate coefficients for the selected cutoff division and Q factor.

cutoff_div sets the cutoff frequency of the filter as an integer division relative to the sampling frequency. q sets the quality factor of the resonant peak.

6.4.3.2 output()

```
double BiquadFilter::output ()  [inline]
```

Return the current output sample.

6.4.3.3 pass()

```
double BiquadFilter::pass (
            double input)  [inline]
```

Passe the next sample into the filter and return the corresponding output sample.

6.4.3.4 reset()

```
void BiquadFilter::reset ()  [inline]
```

Reset all of the internal sample buffers of the filter to 0.

6.4.4 Member Data Documentation

6.4.4.1 feedback_buffer

```
double BiquadFilter::feedback_buffer[2] = { 0, 0 }  [protected]
```

6.4.4.2 feedback_coeff

```
double BiquadFilter::feedback_coeff[2] = { 0, 0 }  [protected]
```

6.4.4.3 input_buffer

```
double BiquadFilter::input_buffer[3] = { 0, 0, 0 }  [protected]
```

6.4.4.4 input_coeff

```
double BiquadFilter::input_coeff[3] = { 0, 0, 0 }  [protected]
```

6.4.4.5 output_value

```
double BiquadFilter::output_value = 0  [protected]
```

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

- filter.h

6.5 PhaseNoiseDataProcessor::Config Struct Reference

```
#include <PhaseNoiseDataProcessorLib.h>
```

Public Attributes

- bool cross_corr = true

  True for cross correlation, false for single channel.
- double smpl_rate = 524288.0

  Sample rate of the captured data.
- uint32_t num_meas = 1

  Number of measurements to process.
- uint32_t data_size = 4194304

  Number of samples captured per channel for a single measurement.
- uint32_t packet_size = 524288

  Minimum size of the FFT window.
- uint32_t num_bands = 1

  Number of frequency bands to use.
- uint32_t band_mult = 8

  Frequency multiplier for subsequent bands (must be a power of 2)
- uint32_t band_shift = 2

  Shifts each band down in multiples of band_mult by the specified number (prioritises more correlations over smaller RBW for higher frequencies)
- double conversion_fac = 0.0025

  Maximum input voltage reference (bipolar)
- bool window_en = false

  True to apply flat top window to FFT input data (NOT YET IMPLEMENTED), false for no windowing (rectangular)

6.5.1 Member Data Documentation

6.5.1.1 band_mult

```
uint32_t PhaseNoiseDataProcessor::Config::band_mult = 8
```

Frequency multiplier for subsequent bands (must be a power of 2)

6.5.1.2 band_shift

```
uint32_t PhaseNoiseDataProcessor::Config::band_shift = 2
```

Shifts each band down in multiples of band_mult by the specified number (prioritises more correlations over smaller RBW for higher frequencies)

6.5.1.3 conversion_fac

```
double PhaseNoiseDataProcessor::Config::conversion_fac = 0.0025
```

Maximum input voltage reference (bipolar)

6.5.1.4 cross_corr

```
bool PhaseNoiseDataProcessor::Config::cross_corr = true
```

True for cross correlation, false for single channel.

6.5.1.5 data_size

```
uint32_t PhaseNoiseDataProcessor::Config::data_size = 4194304
```

Number of samples captured per channel for a single measurement.

6.5.1.6 num_bands

```
uint32_t PhaseNoiseDataProcessor::Config::num_bands = 1
```

Number of frequency bands to use.

6.5.1.7 num_meas

```
uint32_t PhaseNoiseDataProcessor::Config::num_meas = 1
```

Number of measurements to process.

6.5.1.8 packet_size

```
uint32_t PhaseNoiseDataProcessor::Config::packet_size = 524288
```

Minimum size of the FFT window.

6.5.1.9 smpl_rate

```
double PhaseNoiseDataProcessor::Config::smpl_rate = 524288.0
```

Sample rate of the captured data.

6.5.1.10 window_en

```
bool PhaseNoiseDataProcessor::Config::window_en = false
```

True to apply flat top window to FFT input data (NOT YET IMPLEMENTED), false for no windowing (rectangular)

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

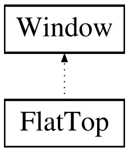
- PhaseNoiseDataProcessorLib.h

6.6 FlatTop Class Reference

Flat top window for minimal scalloping loss.

`#include <window.h>`

Inheritance diagram for FlatTop:



Public Member Functions

- FlatTop (size_t size)

Protected Member Functions

- double rbwFactor () override

  Calculate and return the resolution bandwidth scaling factor.
- void generate ()

  Generate the data points.

6.6.1 Detailed Description

Flat top window for minimal scalloping loss.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 FlatTop()

```
FlatTop::FlatTop (
          size_t size)  [inline]
```

6.6.3 Member Function Documentation

6.6.3.1 generate()

```
void FlatTop::generate ()  [inline], [protected], [virtual]
```

Generate the data points.

Implements Window.

6.6.3.2 rbwFactor()

```
double FlatTop::rbwFactor ()  [inline], [override], [protected], [virtual]
```

Calculate and return the resolution bandwidth scaling factor.

Implements Window.

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

- window.h

6.7 TrigTable Class Reference

Generates a lookup table of precalculated sine/cosine values.

```
#include <trig_table.h>
```

Public Member Functions

- TrigTable (uint32_t size)
- TrigTable ()
- double sin_lookup (uint64_t pos)

    Look up sine value at the given position.
- double cos_lookup (uint64_t pos)

    Look up cosine value at the given position.

6.7.1 Detailed Description

Generates a lookup table of precalculated sine/cosine values.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 TrigTable() [1/2]

```
TrigTable::TrigTable (
            uint32_t size)  [inline]
```

6.7.2.2 TrigTable() [2/2]

```
TrigTable::TrigTable ()  [inline]
```

6.7.3 Member Function Documentation

6.7.3.1 cos_lookup()

```
double TrigTable::cos_lookup (
            uint64_t pos) [inline]
```

Look up cosine value at the given position.

6.7.3.2 sin_lookup()

```
double TrigTable::sin_lookup (
            uint64_t pos) [inline]
```

Look up sine value at the given position.

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

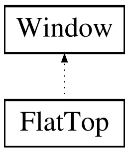
- trig_table.h

6.8 Window Class Reference

Abstract class for a sampling window of given size, which defines the shape and resolution bandwidth scaling factor.

```
#include <window.h>
```

Inheritance diagram for Window:



Public Member Functions

- Window (size_t size)

Protected Member Functions

- virtual double rbwFactor ()=0

  Calculate and return the resolution bandwidth scaling factor.
- virtual void generate ()=0

  Generate the data points.

Protected Attributes

- size_t size = 0

  Size of the window in samples.
- double rbw_factor

  Resolution bandwidth scaling factor (equal to 1.0 for a rectangular window)
- std::vector< double > data

  Data points that define the shape of of the window (all 1.0s for a rectangular window)

6.8.1 Detailed Description

Abstract class for a sampling window of given size, which defines the shape and resolution bandwidth scaling factor.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 Window()

```
Window::Window (
        size_t size)  [inline]
```

6.8.3 Member Function Documentation

6.8.3.1 generate()

```
virtual void Window::generate ()  [protected], [pure virtual]
```

Generate the data points.

Implemented in FlatTop.

6.8.3.2 rbwFactor()

```
virtual double Window::rbwFactor ()  [protected], [pure virtual]
```

Calculate and return the resolution bandwidth scaling factor.

Implemented in FlatTop.

6.8.4 Member Data Documentation

6.8.4.1 data

```
std::vector<double> Window::data  [protected]
```

Data points that define the shape of of the window (all 1.0s for a rectangular window)

6.8.4.2 rbw_factor

```
double Window::rbw_factor  [protected]
```

Resolution bandwidth scaling factor (equal to 1.0 for a rectangular window)

6.8.4.3 size

```
size_t Window::size = 0  [protected]
```

Size of the window in samples.

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

- window.h

Chapter 7

File Documentation

7.1 band_data.h File Reference

```
#include <vector>
```

Classes

- class Band

  Data structure for a single phase noise measurement band.
- class BandData

  Generates data for a set of phase noise measurement bands based on the supplied parameters.

7.2 band_data.h

Go to the documentation of this file.

```
#pragma once

#include <vector>

class Band
{
public:

    uint32_t num_sets = 0;
    uint16_t smpl_rate_div = 0;
    uint32_t output_start_pos = 0;
    uint32_t window_size = 0;
    uint32_t dft_size = 0;
    uint32_t dft_start_pos = 0;
    uint32_t dft_end_pos = 0;
};


class BandData
{
public:

    uint32_t smpl_data_size = 0;
    uint16_t band_mult = 1;
    uint16_t num_bands = 0;
    uint32_t min_window_size = 0;
    uint16_t band_shift = 0;
    bool fixed_smpl_rate = true;

    uint32_t output_size = 0;
```

```
    std::vector<Band> bands;

    BandData() {};

    BandData(uint16_t num_bands, uint16_t band_mult, uint32_t smpl_data_size, uint32_t min_window_size, uint16_t band_shift, bool fixed_smpl_rate)
    {
        this->smpl_data_size = smpl_data_size;
        this->band_mult = band_mult;
        this->num_bands = num_bands;
        this->min_window_size = min_window_size;
        this->band_shift = band_shift;
        this->fixed_smpl_rate = fixed_smpl_rate;


        uint16_t smpl_rate_div = 1;
        uint32_t num_sets = smpl_data_size / min_window_size;

        this->bands.resize(num_bands);

        for (int b = (this->num_bands - 1); b >= 0; b--)
        {
            this->bands[b].smpl_rate_div = smpl_rate_div;
            this->bands[b].num_sets = num_sets;

            if(!fixed_smpl_rate) smpl_rate_div *= this->band_mult;
            num_sets /= this->band_mult;
        }

        uint32_t pos_count = 0;
        uint16_t shift = 1;
        for (int i = 0; i < band_shift; i++) shift *= this->band_mult;

        for (int b = 0; b < this->num_bands; b++)
        {
            this->bands[b].window_size = fixed_smpl_rate ? (smpl_data_size / this->bands[b].num_sets) : min_window_size;
            this->bands[b].dft_size = (this->bands[b].window_size / 2) + 1;

            this->bands[b].output_start_pos = pos_count;
            this->bands[b].dft_start_pos = (b == 0) ? 0 : (min_window_size / (2 * band_mult * shift));
            this->bands[b].dft_end_pos = (b == (this->num_bands - 1)) ? (min_window_size / 2) : ((min_window_size / (2 * shift)) - 1);
            pos_count += (this->bands[b].dft_end_pos - this->bands[b].dft_start_pos + 1);
        }

        this->output_size = pos_count;
    }
};
```

7.3 fft.cpp File Reference

```
#include "fft.h"
```

Functions

- void set_data_size (uint32_t size)

  Sets the size of the FFT window to be used and precalculates sine/cosine values.
- void dft (double ∗x_re, double ∗x_im, double ∗out_re, double ∗out_im, uint32_t k_start, uint32_t k_stop)

  Calculate the DFT for an array of real and imaginary data points.
- void rdft (double ∗x_re, double ∗out_re, double ∗out_im, uint32_t k_start, uint32_t k_stop)

  Calculate the DFT for an array of real data points.
- void fft (double ∗x_re, double ∗x_im, double ∗out_re, double ∗out_im, uint32_t data_size, uint32_t div)
- void fft (double ∗x_re, double ∗x_im, double ∗out_re, double ∗out_im)

  Calculate the FFT for an array of real and imaginary data points.
- void pfft (double ∗x_re, double ∗x_im, double ∗out_re, double ∗out_im, uint32_t k_num, uint32_t data_size, uint32_t div)
- void pfft (double ∗x_re, double ∗x_im, double ∗out_re, double ∗out_im, uint32_t k_num)

Calculate the partial FFT for an array of real and imaginary data points.

- void pfft_dual_real (double ∗x_re1, double ∗x_re2, double ∗out_re, double ∗out_im, uint32_t k_num, uint32←_t data_size, uint32_t div)
- void pfft_dual_real (double ∗x_re1, double ∗x_re2, double ∗out_re, double ∗out_im, uint32_t k_num)

  Calculate the partial FFT for two arrays of real data points to be used as the input to a dual-real FFT cross-correlation function.
- void rfft (double ∗x_re, double ∗out_re, double ∗out_im, uint32_t data_size, uint32_t div)
- void rfft (double ∗x_re, double ∗out_re, double ∗out_im)

  Calculate the FFT for an array of real data points.
- void rpfft (double ∗x_re, double ∗out_re, double ∗out_im, uint32_t k_num, uint32_t data_size, uint32_t div)
- void rpfft (double ∗x_re, double ∗out_re, double ∗out_im, uint32_t k_num)

  Calculate the partial FFT for an array of real data points.
- void cross_corr_dual_real (double ∗x_re, double ∗x_im, double ∗out_re, double ∗out_im, uint32_t k_start, uint32_t k_stop)

  Performs the cross correlation of an FFT of two sets of real data.
- void cross_corr_dual_real (double ∗x_re, double ∗x_im, double ∗out_re, double ∗out_im)
- void cross_corr (double ∗f_re, double ∗f_im, double ∗g_re, double ∗g_im, double ∗out_re, double ∗out_im, uint32_t k_start, uint32_t k_stop)

  Performs the cross correlation of complex data sets f and g
- void cross_corr (double ∗f_re, double ∗f_im, double ∗g_re, double ∗g_im, double ∗out_re, double ∗out_im)
- void auto_corr_real (double ∗x_re, double ∗x_im, double ∗out_re, uint32_t k_start, uint32_t k_stop)

  Performs the auto correlation of the result of an FFT of real data.
- void auto_corr_real (double ∗x_re, double ∗x_im, double ∗out_re)

Variables

- uint32_t g_data_size = 1
- TrigTable g_trig = TrigTable()

7.3.1 Function Documentation

7.3.1.1 auto_corr_real() [1/2]

```
void auto_corr_real (
            double * x_re,
            double * x_im,
            double * out_re)
```

7.3.1.2 auto_corr_real() [2/2]

```
void auto_corr_real (
            double * x_re,
            double * x_im,
            double * out_re,
            uint32_t k_start,
            uint32_t k_stop)
```

Performs the auto correlation of the result of an FFT of real data.

Only output values in the range *k_start* to *k_stop* are calculated. The output values are the sum of the positive and negative frequency components (i.e. multiplied by 2).

7.3.1.3 cross_corr() [1/2]

```
void cross_corr (
            double * f_re,
            double * f_im,
            double * g_re,
            double * g_im,
            double * out_re,
            double * out_im)
```

7.3.1.4 cross_corr() [2/2]

```
void cross_corr (
            double * f_re,
            double * f_im,
            double * g_re,
            double * g_im,
            double * out_re,
            double * out_im,
            uint32_t k_start,
            uint32_t k_stop)
```

Performs the cross correlation of complex data sets *f* and *g*

Only output values in the range *k_start* to *k_stop* are calculated.

7.3.1.5 cross_corr_dual_real() [1/2]

```
void cross_corr_dual_real (
            double * x_re,
            double * x_im,
            double * out_re,
            double * out_im)
```

7.3.1.6 cross_corr_dual_real() [2/2]

```
void cross_corr_dual_real (
            double * x_re,
            double * x_im,
            double * out_re,
            double * out_im,
            uint32_t k_start,
            uint32_t k_stop)
```

Performs the cross correlation of an FFT of two sets of real data.

The input data is the FFT generated where one of the real data sets is used as the real part of the FFT input, and the other real data set is used as the imaginary part of the FFT input. Only output values in the range *k_start* to *k_stop* are calculated. Unlike the standard cross_corr() function, the output values are the sum of the positive and negative frequency components (i.e. multiplied by 2).

7.3.1.7 dft()

```
void dft (
            double * x_re,
            double * x_im,
            double * out_re,
            double * out_im,
            uint32_t k_start,
            uint32_t k_stop)
```

Calculate the DFT for an array of real and imaginary data points.

Only output values in the range *k_start* to *k_stop* are calculated.

7.3.1.8 fft() [1/2]

```
void fft (
            double * x_re,
            double * x_im,
            double * out_re,
            double * out_im)
```

Calculate the FFT for an array of real and imaginary data points.

7.3.1.9 fft() [2/2]

```
void fft (
            double * x_re,
            double * x_im,
            double * out_re,
            double * out_im,
            uint32_t data_size,
            uint32_t div)
```

7.3.1.10 pfft() [1/2]

```
void pfft (
            double * x_re,
            double * x_im,
            double * out_re,
            double * out_im,
            uint32_t k_num)
```

Calculate the partial FFT for an array of real and imaginary data points.

Only the first *k_num* output values are calculated.

7.3.1.11 pfft() [2/2]

```
void pfft (
            double * x_re,
            double * x_im,
            double * out_re,
            double * out_im,
            uint32_t k_num,
            uint32_t data_size,
            uint32_t div)
```

7.3.1.12 pfft_dual_real() [1/2]

```
void pfft_dual_real (
            double * x_re1,
            double * x_re2,
            double * out_re,
            double * out_im,
            uint32_t k_num)
```

Calculate the partial FFT for two arrays of real data points to be used as the input to a dual-real FFT cross-correlation function.

//! Only the first *k_num* output values are calculated.

7.3.1.13 pfft_dual_real() [2/2]

```
void pfft_dual_real (
            double * x_re1,
            double * x_re2,
            double * out_re,
            double * out_im,
            uint32_t k_num,
            uint32_t data_size,
            uint32_t div)
```

7.3.1.14 rdft()

```
void rdft (
            double * x_re,
            double * out_re,
            double * out_im,
            uint32_t k_start,
            uint32_t k_stop)
```

Calculate the DFT for an array of real data points.

Only output values in the range *k_start* to *k_stop* are calculated.

7.3.1.15 rfft() [1/2]

```
void rfft (
            double * x_re,
            double * out_re,
            double * out_im)
```

Calculate the FFT for an array of real data points.

7.3.1.16 rfft() [2/2]

```
void rfft (
            double * x_re,
            double * out_re,
            double * out_im,
            uint32_t data_size,
            uint32_t div)
```

7.3.1.17 rpfft() [1/2]

```
void rpfft (
            double * x_re,
            double * out_re,
            double * out_im,
            uint32_t k_num)
```

Calculate the partial FFT for an array of real data points.

Only the first *k_num* output values are calculated.

7.3.1.18 rpfft() [2/2]

```
void rpfft (
            double * x_re,
            double * out_re,
            double * out_im,
            uint32_t k_num,
            uint32_t data_size,
            uint32_t div)
```

7.3.1.19 set_data_size()

```
void set_data_size (
            uint32_t size)
```

Sets the size of the FFT window to be used and precalculates sine/cosine values.

Must be set before any transform functions are performed.

7.3.2 Variable Documentation

7.3.2.1 g_data_size

```
uint32_t g_data_size = 1
```

7.3.2.2 g_trig

```
TrigTable g_trig = TrigTable()
```

7.4 fft.h File Reference

```
#include <stdint.h>
#include "trig_table.h"
```

Functions

- void set_data_size (uint32_t size)

    Sets the size of the FFT window to be used and precalculates sine/cosine values.
- void dft (double ∗x_re, double ∗x_im, double ∗out_re, double ∗out_im, uint32_t k_start, uint32_t k_stop)

    Calculate the DFT for an array of real and imaginary data points.
- void rdft (double ∗x_re, double ∗out_re, double ∗out_im, uint32_t k_start, uint32_t k_stop)

    Calculate the DFT for an array of real data points.
- void fft (double ∗x_re, double ∗x_im, double ∗out_re, double ∗out_im)

    Calculate the FFT for an array of real and imaginary data points.
- void fft (double ∗x_re, double ∗x_im, double ∗out_re, double ∗out_im, uint32_t data_size, uint32_t div)
- void pfft (double ∗x_re, double ∗x_im, double ∗out_re, double ∗out_im, uint32_t k_num)

    Calculate the partial FFT for an array of real and imaginary data points.
- void pfft (double ∗x_re, double ∗x_im, double ∗out_re, double ∗out_im, uint32_t k_num, uint32_t data_size, uint32_t div)
- void pfft_dual_real (double ∗x_re1, double ∗x_re2, double ∗out_re, double ∗out_im, uint32_t k_num)

    Calculate the partial FFT for two arrays of real data points to be used as the input to a dual-real FFT cross-correlation function.
- void pfft_dual_real (double ∗x_re1, double ∗x_re2, double ∗out_re, double ∗out_im, uint32_t k_num, uint32←_t data_size, uint32_t div)
- void rfft (double ∗x_re, double ∗out_re, double ∗out_im)

    Calculate the FFT for an array of real data points.
- void rfft (double ∗x_re, double ∗out_re, double ∗out_im, uint32_t data_size, uint32_t div)
- void rpfft (double ∗x_re, double ∗out_re, double ∗out_im, uint32_t k_num)

    Calculate the partial FFT for an array of real data points.
- void rpfft (double ∗x_re, double ∗out_re, double ∗out_im, uint32_t k_num, uint32_t data_size, uint32_t div)
- void cross_corr (double ∗f_re, double ∗f_im, double ∗g_re, double ∗g_im, double ∗out_re, double ∗out_im, uint32_t k_start, uint32_t k_stop)

    Performs the cross correlation of complex data sets f and g
- void cross_corr (double ∗f_re, double ∗f_im, double ∗g_re, double ∗g_im, double ∗out_re, double ∗out_im)
- void cross_corr_dual_real (double ∗x_re, double ∗x_im, double ∗out_re, double ∗out_im, uint32_t k_start, uint32_t k_stop)

    Performs the cross correlation of an FFT of two sets of real data.
- void cross_corr_dual_real (double ∗x_re, double ∗x_im, double ∗out_re, double ∗out_im)
- void auto_corr_real (double ∗x_re, double ∗x_im, double ∗out_re, uint32_t k_start, uint32_t k_stop)

    Performs the auto correlation of the result of an FFT of real data.
- void auto_corr_real (double ∗x_re, double ∗x_im, double ∗out_re)

Variables

- uint32_t g_data_size
- TrigTable g_trig

7.4.1 Function Documentation

7.4.1.1 auto_corr_real() [1/2]

```
void auto_corr_real (
            double * x_re,
            double * x_im,
            double * out_re)
```

7.4.1.2 auto_corr_real() [2/2]

```
void auto_corr_real (
            double * x_re,
            double * x_im,
            double * out_re,
            uint32_t k_start,
            uint32_t k_stop)
```

Performs the auto correlation of the result of an FFT of real data.

Only output values in the range *k_start* to *k_stop* are calculated. The output values are the sum of the positive and negative frequency components (i.e. multiplied by 2).

7.4.1.3 cross_corr() [1/2]

```
void cross_corr (
            double * f_re,
            double * f_im,
            double * g_re,
            double * g_im,
            double * out_re,
            double * out_im)
```

7.4.1.4 cross_corr() [2/2]

```
void cross_corr (
            double * f_re,
            double * f_im,
            double * g_re,
            double * g_im,
            double * out_re,
            double * out_im,
            uint32_t k_start,
            uint32_t k_stop)
```

Performs the cross correlation of complex data sets *f* and *g*

Only output values in the range *k_start* to *k_stop* are calculated.

7.4.1.5 cross_corr_dual_real() [1/2]

```
void cross_corr_dual_real (
            double * x_re,
            double * x_im,
            double * out_re,
            double * out_im)
```

7.4.1.6 cross_corr_dual_real() [2/2]

```
void cross_corr_dual_real (
            double * x_re,
            double * x_im,
            double * out_re,
            double * out_im,
            uint32_t k_start,
            uint32_t k_stop)
```

Performs the cross correlation of an FFT of two sets of real data.

The input data is the FFT generated where one of the real data sets is used as the real part of the FFT input, and the other real data set is used as the imaginary part of the FFT input. Only output values in the range *k_start* to *k_stop* are calculated. Unlike the standard cross_corr() function, the output values are the sum of the positive and negative frequency components (i.e. multiplied by 2).

7.4.1.7 dft()

```
void dft (
            double * x_re,
            double * x_im,
            double * out_re,
            double * out_im,
            uint32_t k_start,
            uint32_t k_stop)
```

Calculate the DFT for an array of real and imaginary data points.

Only output values in the range *k_start* to *k_stop* are calculated.

7.4.1.8 fft() [1/2]

```
void fft (
            double * x_re,
            double * x_im,
            double * out_re,
            double * out_im)
```

Calculate the FFT for an array of real and imaginary data points.

7.4.1.9 fft() [2/2]

```
void fft (
            double * x_re,
            double * x_im,
            double * out_re,
            double * out_im,
            uint32_t data_size,
            uint32_t div)
```

7.4.1.10 pfft() [1/2]

```
void pfft (
            double * x_re,
            double * x_im,
            double * out_re,
            double * out_im,
            uint32_t k_num)
```

Calculate the partial FFT for an array of real and imaginary data points.

Only the first *k_num* output values are calculated.

7.4.1.11 pfft() [2/2]

```
void pfft (
            double * x_re,
            double * x_im,
            double * out_re,
            double * out_im,
            uint32_t k_num,
            uint32_t data_size,
            uint32_t div)
```

7.4.1.12 pfft_dual_real() [1/2]

```
void pfft_dual_real (
            double * x_re1,
            double * x_re2,
            double * out_re,
            double * out_im,
            uint32_t k_num)
```

Calculate the partial FFT for two arrays of real data points to be used as the input to a dual-real FFT cross-correlation function.

//! Only the first *k_num* output values are calculated.

7.4.1.13 pfft_dual_real() [2/2]

```
void pfft_dual_real (
            double * x_re1,
            double * x_re2,
            double * out_re,
            double * out_im,
            uint32_t k_num,
            uint32_t data_size,
            uint32_t div)
```

7.4.1.14 rdft()

```
void rdft (
            double * x_re,
            double * out_re,
            double * out_im,
            uint32_t k_start,
            uint32_t k_stop)
```

Calculate the DFT for an array of real data points.

Only output values in the range *k_start* to *k_stop* are calculated.

7.4.1.15 rfft() [1/2]

```
void rfft (
            double * x_re,
            double * out_re,
            double * out_im)
```

Calculate the FFT for an array of real data points.

7.4.1.16 rfft() [2/2]

```
void rfft (
            double * x_re,
            double * out_re,
            double * out_im,
            uint32_t data_size,
            uint32_t div)
```

7.4.1.17 rpfft() [1/2]

```
void rpfft (
            double * x_re,
            double * out_re,
            double * out_im,
            uint32_t k_num)
```

Calculate the partial FFT for an array of real data points.

Only the first *k_num* output values are calculated.

7.4.1.18 rpfft() [2/2]

```
void rpfft (
            double * x_re,
            double * out_re,
            double * out_im,
            uint32_t k_num,
            uint32_t data_size,
            uint32_t div)
```

7.4.1.19 set_data_size()

```
void set_data_size (
            uint32_t size)
```

Sets the size of the FFT window to be used and precalculates sine/cosine values.

Must be set before any transform functions are performed.

7.4.2 Variable Documentation

7.4.2.1 g_data_size

```
uint32_t g_data_size [extern]
```

7.4.2.2 g_trig

```
TrigTable g_trig [extern]
```

7.5 fft.h

Go to the documentation of this file.

```
#pragma once

#include <stdint.h>
#include "trig_table.h"

extern uint32_t g_data_size;
extern TrigTable g_trig;


void set_data_size(uint32_t size);


void dft(double* x_re, double* x_im, double* out_re, double* out_im, uint32_t k_start, uint32_t k_stop);


void rdft(double* x_re, double* out_re, double* out_im, uint32_t k_start, uint32_t k_stop);

void fft(double* x_re, double* x_im, double* out_re, double* out_im);
void fft(double* x_re, double* x_im, double* out_re, double* out_im, uint32_t data_size, uint32_t div);


void pfft(double* x_re, double* x_im, double* out_re, double* out_im, uint32_t k_num);
```

```
void pfft(double* x_re, double* x_im, double* out_re, double* out_im, uint32_t k_num, uint32_t data_size, uint32_t div);

void pfft_dual_real(double* x_re1, double* x_re2, double* out_re, double* out_im, uint32_t k_num);
void pfft_dual_real(double* x_re1, double* x_re2, double* out_re, double* out_im, uint32_t k_num, uint32_t data_size, uint32_t div);

void rfft(double* x_re, double* out_re, double* out_im);
void rfft(double* x_re, double* out_re, double* out_im, uint32_t data_size, uint32_t div);

void rpfft(double* x_re, double* out_re, double* out_im, uint32_t k_num);
void rpfft(double* x_re, double* out_re, double* out_im, uint32_t k_num, uint32_t data_size, uint32_t div);

void cross_corr(double* f_re, double* f_im, double* g_re, double* g_im, double* out_re, double* out_im, uint32_t k_start, uint32_t k_stop);
void cross_corr(double* f_re, double* f_im, double* g_re, double* g_im, double* out_re, double* out_im);

void cross_corr_dual_real(double* x_re, double* x_im, double* out_re, double* out_im, uint32_t k_start, uint32_t k_stop);
void cross_corr_dual_real(double* x_re, double* x_im, double* out_re, double* out_im);

void auto_corr_real(double* x_re, double* x_im, double* out_re, uint32_t k_start, uint32_t k_stop);
void auto_corr_real(double* x_re, double* x_im, double* out_re);
```

7.6 filter.h File Reference

```
#include <cmath>
#include <stdint.h>
```

Classes

- class BiquadFilter

  Digital biquadratic filter.
- class AntiAliasingFilter

  6th order Butterworth digital anti-aliasing filter

Macros

- #define _USE_MATH_DEFINES

7.6.1 Macro Definition Documentation

7.6.1.1 _USE_MATH_DEFINES

```
#define _USE_MATH_DEFINES
```

7.7 filter.h

Go to the documentation of this file.

```
#pragma once

#define _USE_MATH_DEFINES
#include <cmath>
#include <stdint.h>

class BiquadFilter
{
    protected:

        double input_coeff[3] = { 0, 0, 0 };
        double feedback_coeff[2] = { 0, 0 };

        double input_buffer[3] = { 0, 0, 0 };
        double feedback_buffer[2] = { 0, 0 };

        double output_value = 0;

    public:

        BiquadFilter() {};

        BiquadFilter(int cutoff_div, double q)
        {
            generateCoefficients(cutoff_div, q);
        }


        void generateCoefficients(int cutoff_div, double q)
        {
            double k_wa = 1.0 / tanf(M_PI / cutoff_div);
            double k_wa_sqr = powf(k_wa, 2);

            double fac = 1.0 / (k_wa_sqr + (k_wa / q) + 1.0);

            input_coeff[0] = fac;
            input_coeff[1] = 2 * fac;
            input_coeff[2] = fac;

            feedback_coeff[0] = (2.0 * k_wa_sqr - 2.0) * fac;
            feedback_coeff[1] = -(k_wa_sqr - (k_wa / q) + 1.0) * fac;
        }

        double pass(double input)
        {
            input_buffer[2] = input_buffer[1];
            input_buffer[1] = input_buffer[0];
            input_buffer[0] = input;

            feedback_buffer[1] = feedback_buffer[0];
            feedback_buffer[0] = output_value;
            output_value = (input_buffer[0] * input_coeff[0]) + (input_buffer[1] * input_coeff[1]) + (input_buffer[2] * input_coeff[2]) +
                                    (feedback_buffer[0] * feedback_coeff[0]) + (feedback_buffer[1] * feedback_coeff[1]);

            return output();
        }

        void reset()
        {
            input_buffer[2] = 0;
            input_buffer[1] = 0;
            input_buffer[0] = 0;

            feedback_buffer[1] = 0;
            feedback_buffer[0] = 0;

            output_value = 0;
        }

        double output() { return output_value; };
};


class AntiAliasingFilter
{
    protected:
        const double Q1 = 1.931851989228;
        const double Q2 = 0.707106562373;
        const double Q3 = 0.517637997114;

```

```
        BiquadFilter filt1;
        BiquadFilter filt2;
        BiquadFilter filt3;

        double output_value = 0;

    public:

        AntiAliasingFilter(int cutoff_div)
        {
            filt1.generateCoefficients(cutoff_div, Q1);
            filt2.generateCoefficients(cutoff_div, Q2);
            filt3.generateCoefficients(cutoff_div, Q3);
        }

        float pass(double input)
        {
            output_value = filt3.pass(filt2.pass(filt1.pass(input)));
            return output();
        }

        void reset()
        {
            filt1.reset();
            filt2.reset();
            filt3.reset();

            output_value = 0;
        }

        double output() { return output_value; };
};
```

7.8 PhaseNoiseDataProcessorLib.cpp File Reference

```
#include "PhaseNoiseDataProcessorLib.h"
```

Namespaces

- namespace PhaseNoiseDataProcessor

Functions

- void PhaseNoiseDataProcessor::convert_samples (int channel, const void *data_bytes)

  Convert raw input sample byte data to voltage values and store to meas buffer for the selected channel.
- void PhaseNoiseDataProcessor::calc_x_values ()

  Calculate and store x-axis frequency output values.
- void PhaseNoiseDataProcessor::calc_y_values (uint32_t num_meas)

  Calculate and store y-axis power spectral density output values.
- void PhaseNoiseDataProcessor::calc_y_values ()
- void PhaseNoiseDataProcessor::dbv_to_dbc (double rf_input_level, double cal_input_level, double cal_↩ output_level)

  Convert y-axis power spectral density output values from dBV/Hz to dBc/Hz based on the specified calibration settings.
- bool PhaseNoiseDataProcessor::read_meas_from_file (char *filename, uint32_t meas_num)

  Read sample data from file for a single measurement and store to data buffers.
- bool PhaseNoiseDataProcessor::write_output_to_file (char *filename)

  Write output data points to file.
- void PhaseNoiseDataProcessor::retrieve_samples (uint32_t set_num, uint32_t smpl_rate_div)

  Retrieve a sample packet containing a single data set from meaurement data buffers and store in sample buffers (only used for fixed window size processing)

- void PhaseNoiseDataProcessor::filter_samples (AntiAliasingFilter &filter_0, AntiAliasingFilter &filter_1)

  Apply a digital filter to sample buffers (only used for fixed window size processing)
- void PhaseNoiseDataProcessor::overwrite_samples (uint32_t packet_num)

  Overwrite selected packet in measurement data buffers with the contents of sample buffers (only used for fixed window size processing)
- void PhaseNoiseDataProcessor::clear_meas_buffer ()

  Set all values in measurement data buffers to 0.
- void PhaseNoiseDataProcessor::clear_output_buffer ()

  Set all values in output buffers to 0.
- void PhaseNoiseDataProcessor::process_fixed_window_size ()

  Process sample data from file to produce a set of output data points using a fixed window size.
- void PhaseNoiseDataProcessor::setup (bool cross_corr, double smpl_rate, uint32_t num_meas, uint32_↩t data_size, uint32_t min_window_size, uint32_t num_bands, uint32_t band_mult, uint32_t band_shift, double conversion_fac)

  Setup processing functions based on the specified parameters.
- void PhaseNoiseDataProcessor::setup (Config new_config)
- const std::vector< double > & PhaseNoiseDataProcessor::get_x_values ()

  Return vector containing x-axis frequency output values.
- const std::vector< double > & PhaseNoiseDataProcessor::get_y_values ()

  Return vector containing y-axis power spectral density output values.
- void PhaseNoiseDataProcessor::process_fixed_smpl_rate ()

  Process sample data from file to produce a set of output data points using a fixed sampling rate.
- void PhaseNoiseDataProcessor::start_fixed_window_size ()

  Start fixed window size processing where sample data is pushed in real time.
- bool PhaseNoiseDataProcessor::push_data_fixed_window_size (double ∗data_0, double ∗data_1)

  Push sample data for a single measurement to be processed with a fixed window size.
- void PhaseNoiseDataProcessor::start_fixed_smpl_rate ()

  Start fixed sample rate processing where sample data is pushed in real time.
- bool PhaseNoiseDataProcessor::push_data_fixed_smpl_rate (double ∗data_0, double ∗data_1)

  Push sample data for a single measurement to be processed with a fixed sample rate.
- void PhaseNoiseDataProcessor::stop ()

  Stop real time processing and release data buffers.

Variables

- std::vector< double > PhaseNoiseDataProcessor::meas_buffer [2]

  Sample buffers for captured data from a single measurement.
- std::vector< double > PhaseNoiseDataProcessor::smpl_buffer [2]

  Sample buffers for a single sample window.
- std::vector< double > PhaseNoiseDataProcessor::fft_re_buffer

  FFT real output buffer.
- std::vector< double > PhaseNoiseDataProcessor::fft_im_buffer

  FFT imaginary output buffer.
- std::vector< double > PhaseNoiseDataProcessor::output_re_buffer

  Buffer to store the sum of real values of cross correlations / auto correlations.
- std::vector< double > PhaseNoiseDataProcessor::output_im_buffer

  Buffer to store the sum of imaginary values of cross correlations / auto correlations.
- std::vector< double > PhaseNoiseDataProcessor::output_x_buffer

  Buffer to store x-axis frequency output values.
- std::vector< double > PhaseNoiseDataProcessor::output_y_buffer

  Buffer to store y-axis power spectral density output values.

- uint32_t PhaseNoiseDataProcessor::meas_count = 0

  Number of measurements that have been processed since processing was started.
- BandData PhaseNoiseDataProcessor::band_data

  Generated data descibing the properties of each measurement band.
- Config PhaseNoiseDataProcessor::config

  Measurement configuration settings.

7.9 PhaseNoiseDataProcessorLib.h File Reference

```
#include <cmath>
#include <stdio.h>
#include <stdint.h>
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include "band_data.h"
#include "fft.h"
#include "filter.h"
```

Classes

- struct PhaseNoiseDataProcessor::Config

Namespaces

- namespace PhaseNoiseDataProcessor

Macros

- #define _USE_MATH_DEFINES
- #define MAX_DATA_SIZE 268435456
- #define MAX_PACKET_SIZE 268435456
- #define MAX_MEAS 65536
- #define SMPL_FILENAME "samples.bin"
- #define OUTPUT_FILENAME "output.csv"

Functions

- void PhaseNoiseDataProcessor::convert_samples (int channel, const void ∗data_bytes)

  Convert raw input sample byte data to voltage values and store to meas buffer for the selected channel.
- void PhaseNoiseDataProcessor::calc_x_values ()

  Calculate and store x-axis frequency output values.
- void PhaseNoiseDataProcessor::calc_y_values (uint32_t num_meas)

  Calculate and store y-axis power spectral density output values.
- void PhaseNoiseDataProcessor::calc_y_values ()
- void PhaseNoiseDataProcessor::dbv_to_dbc (double rf_input_level, double cal_input_level, double cal_↩ output_level)

Convert y-axis power spectral density output values from dBV/Hz to dBc/Hz based on the specified calibration settings.

- bool PhaseNoiseDataProcessor::read_meas_from_file (char ∗filename, uint32_t meas_num)

  Read sample data from file for a single measurement and store to data buffers.
- void PhaseNoiseDataProcessor::retrieve_samples (uint32_t set_num, uint32_t smpl_rate_div)

  Retrieve a sample packet containing a single data set from meaurement data buffers and store in sample buffers (only used for fixed window size processing)
- void PhaseNoiseDataProcessor::filter_samples (AntiAliasingFilter &filter_0, AntiAliasingFilter &filter_1)

  Apply a digital filter to sample buffers (only used for fixed window size processing)
- void PhaseNoiseDataProcessor::overwrite_samples (uint32_t packet_num)

  Overwrite selected packet in measurement data buffers with the contents of sample buffers (only used for fixed window size processing)
- void PhaseNoiseDataProcessor::clear_meas_buffer ()

  Set all values in measurement data buffers to 0.
- void PhaseNoiseDataProcessor::clear_output_buffer ()

  Set all values in output buffers to 0.
- void PhaseNoiseDataProcessor::setup (bool cross_corr, double smpl_rate, uint32_t num_meas, uint32_↩ t data_size, uint32_t min_window_size, uint32_t num_bands, uint32_t band_mult, uint32_t band_shift, double conversion_fac)

  Setup processing functions based on the specified parameters.
- void PhaseNoiseDataProcessor::setup (Config new_config)
- const std::vector< double > & PhaseNoiseDataProcessor::get_x_values ()

  Return vector containing x-axis frequency output values.
- const std::vector< double > & PhaseNoiseDataProcessor::get_y_values ()

  Return vector containing y-axis power spectral density output values.
- bool PhaseNoiseDataProcessor::write_output_to_file (char ∗filename)

  Write output data points to file.
- void PhaseNoiseDataProcessor::process_fixed_window_size ()

  Process sample data from file to produce a set of output data points using a fixed window size.
- void PhaseNoiseDataProcessor::process_fixed_smpl_rate ()

  Process sample data from file to produce a set of output data points using a fixed sampling rate.
- void PhaseNoiseDataProcessor::start_fixed_window_size ()

  Start fixed window size processing where sample data is pushed in real time.
- bool PhaseNoiseDataProcessor::push_data_fixed_window_size (double ∗data_0, double ∗data_1)

  Push sample data for a single measurement to be processed with a fixed window size.
- void PhaseNoiseDataProcessor::start_fixed_smpl_rate ()

  Start fixed sample rate processing where sample data is pushed in real time.
- bool PhaseNoiseDataProcessor::push_data_fixed_smpl_rate (double ∗data_0, double ∗data_1)

  Push sample data for a single measurement to be processed with a fixed sample rate.
- void PhaseNoiseDataProcessor::stop ()

  Stop real time processing and release data buffers.

7.9.1 Macro Definition Documentation

7.9.1.1 _USE_MATH_DEFINES

```
#define _USE_MATH_DEFINES
```

7.9.1.2 MAX_DATA_SIZE

```
#define MAX_DATA_SIZE 268435456
```

7.9.1.3 MAX_MEAS

```
#define MAX_MEAS 65536
```

7.9.1.4 MAX_PACKET_SIZE

```
#define MAX_PACKET_SIZE 268435456
```

7.9.1.5 OUTPUT_FILENAME

```
#define OUTPUT_FILENAME "output.csv"
```

7.9.1.6 SMPL_FILENAME

```
#define SMPL_FILENAME "samples.bin"
```

7.10 PhaseNoiseDataProcessorLib.h

Go to the documentation of this file.

```
#pragma once

#define _USE_MATH_DEFINES
#include <cmath>

#include <stdio.h>
#include <stdint.h>
#include <iostream>
#include <fstream>
#include <string>
#include <vector>

#include "band_data.h"
#include "fft.h"
#include "filter.h"

using namespace std;

#define MAX_DATA_SIZE 268435456
#define MAX_PACKET_SIZE 268435456
#define MAX_MEAS 65536
#define SMPL_FILENAME "samples.bin"
#define OUTPUT_FILENAME "output.csv"

namespace PhaseNoiseDataProcessor
{
    extern std::vector<double> meas_buffer[2];
    extern std::vector<double> smpl_buffer[2];
    extern std::vector<double> fft_re_buffer;
    extern std::vector<double> fft_im_buffer;
    extern std::vector<double> output_re_buffer;
    extern std::vector<double> output_im_buffer;
    extern std::vector<double> output_x_buffer;
    extern std::vector<double> output_y_buffer;

    extern uint32_t meas_count;

    extern BandData band_data;

    struct Config
    {
        bool cross_corr = true;
        double smpl_rate = 524288.0;
        uint32_t num_meas = 1;
        uint32_t data_size = 4194304;
        uint32_t packet_size = 524288;
```

```
        uint32_t num_bands = 1;
        uint32_t band_mult = 8;
        uint32_t band_shift = 2;
        double conversion_fac = 0.0025;
        bool window_en = false;
    };

    extern Config config;


    void convert_samples(int channel, const void* data_bytes);


    void calc_x_values();


    void calc_y_values(uint32_t num_meas);
    void calc_y_values();


    void dbv_to_dbc(double rf_input_level, double cal_input_level, double cal_output_level);


    bool read_meas_from_file(char* filename, uint32_t meas_num);


    void retrieve_samples(uint32_t set_num, uint32_t smpl_rate_div);


    void filter_samples(AntiAliasingFilter& filter_0, AntiAliasingFilter& filter_1);

    void overwrite_samples(uint32_t packet_num);

    void clear_meas_buffer();

    void clear_output_buffer();

    void setup(bool cross_corr, double smpl_rate, uint32_t num_meas, uint32_t data_size, uint32_t
 min_window_size, uint32_t num_bands, uint32_t band_mult, uint32_t band_shift, double conversion_fac);
    void setup(Config new_config);

    const std::vector<double>& get_x_values();

    const std::vector<double>& get_y_values();


    bool write_output_to_file(char* filename);


    void process_fixed_window_size();


    void process_fixed_smpl_rate();

    void start_fixed_window_size();

    bool push_data_fixed_window_size(double* data_0, double* data_1);

    void start_fixed_smpl_rate();

    bool push_data_fixed_smpl_rate(double* data_0, double* data_1);

    void stop();
}
```

7.11 trig_table.h File Reference

```
#include <cmath>
#include <stdint.h>
#include <vector>
```

Classes

- class TrigTable

  Generates a lookup table of precalculated sine/cosine values.

Macros

- #define _USE_MATH_DEFINES

7.11.1 Macro Definition Documentation

7.11.1.1 _USE_MATH_DEFINES

```
#define _USE_MATH_DEFINES
```

7.12 trig_table.h

Go to the documentation of this file.

```
#pragma once

#define _USE_MATH_DEFINES
#include <cmath>

#include <stdint.h>
#include <vector>

class TrigTable
{
private:

    uint32_t mod_mask;
    uint32_t cos_shift;
    std::vector<double> sin_table;

public:
    TrigTable(uint32_t size)
    {
        mod_mask = size - 1;
        cos_shift = size » 2;
        sin_table.resize(size);
        for (int i = 0; i < size; i++) sin_table[i] = sin((((double) i / size)) * 2 * M_PI);
    }
    TrigTable()
    {
        mod_mask = 0x0;
        cos_shift = 0;
        sin_table.resize(1);
        sin_table[0] = 0.0;
    }
    double sin_lookup(uint64_t pos)
    {
        uint32_t pos_m = (uint32_t)(pos & mod_mask);
        return sin_table[pos_m];
    }

    double cos_lookup(uint64_t pos)
    {
        uint32_t pos_m = (uint32_t)((pos + cos_shift) & mod_mask);
        return sin_table[pos_m];
    }
};


```

7.13 window.h File Reference

```
#include <cmath>
#include <vector>
```

Classes

- class Window

  Abstract class for a sampling window of given size, which defines the shape and resolution bandwidth scaling factor.
- class FlatTop

  Flat top window for minimal scalloping loss.

Macros

- #define _USE_MATH_DEFINES

7.13.1 Macro Definition Documentation

7.13.1.1 _USE_MATH_DEFINES

```
#define _USE_MATH_DEFINES
```

7.14 window.h

Go to the documentation of this file.

```
#pragma once

#define _USE_MATH_DEFINES
#include <cmath>

#include <vector>

class Window
{
protected:

    size_t size = 0;
    double rbw_factor;
    std::vector<double> data;

    virtual double rbwFactor() = 0;
    virtual void generate() = 0;

public:

    Window(size_t size)
    {
        this->size = size;
        data.resize(size);
        this->rbw_factor = rbwFactor();
        generate();
    }
};

class FlatTop : Window
{
protected:

    double rbwFactor() override { return 3.8193596; };

    void generate()
    {
        double a0 = 0.21557895;
        double a1 = 0.41663158;
        double a2 = 0.277263158;
        double a3 = 0.083578947;
        double a4 = 0.006947368;
        double fac = M_PI / size;

        for (int i = 0; i < data.size(); i++)
        {
            data[i] = a0 - (a1 * cos(2 * i * fac)) + (a2 * cos(4 * i * fac)) - (a3 * cos(6 * i * fac))
 + (a4 * cos(8 * i * fac));
        }
    }

public:

    FlatTop(size_t size) : Window(size) {};
};
```

Index

_USE_MATH_DEFINES
filter.h, 46
PhaseNoiseDataProcessorLib.h, 51
trig_table.h, 54
window.h, 55

AntiAliasingFilter, 17
AntiAliasingFilter, 18
filt1, 18
filt2, 18
filt3, 18
output, 18
output_value, 18
pass, 18
Q1, 19
Q2, 19
Q3, 19
reset, 18
auto_corr_real
fft.cpp, 35
fft.h, 41

Band, 19
dft_end_pos, 20
dft_size, 20
dft_start_pos, 20
num_sets, 20
output_start_pos, 20
smpl_rate_div, 20
window_size, 20
band_data
PhaseNoiseDataProcessor, 15
band_data.h, 33
band_mult
BandData, 22
PhaseNoiseDataProcessor::Config, 26
band_shift
BandData, 22
PhaseNoiseDataProcessor::Config, 26
BandData, 21
band_mult, 22
band_shift, 22
BandData, 22
bands, 22
fixed_smpl_rate, 22
min_window_size, 22
num_bands, 23
output_size, 23
smpl_data_size, 23
bands
BandData, 22
BiquadFilter, 23
BiquadFilter, 24
feedback_buffer, 25
feedback_coeff, 25
generateCoefficients, 24
input_buffer, 25
input_coeff, 25
output, 24
output_value, 25
pass, 24
reset, 25

calc_x_values
PhaseNoiseDataProcessor, 11
calc_y_values
PhaseNoiseDataProcessor, 11
clear_meas_buffer
PhaseNoiseDataProcessor, 11
clear_output_buffer
PhaseNoiseDataProcessor, 11
config
PhaseNoiseDataProcessor, 15
conversion_fac
PhaseNoiseDataProcessor::Config, 26
convert_samples
PhaseNoiseDataProcessor, 11
cos_lookup
TrigTable, 30
cross_corr
fft.cpp, 35, 36
fft.h, 41
PhaseNoiseDataProcessor::Config, 26
cross_corr_dual_real
fft.cpp, 36
fft.h, 41, 42

data
Window, 31
data_size
PhaseNoiseDataProcessor::Config, 26
dbv_to_dbc
PhaseNoiseDataProcessor, 11
dft
fft.cpp, 36
fft.h, 42
dft_end_pos
Band, 20
dft_size
Band, 20

dft_start_pos
Band, 20

feedback_buffer
BiquadFilter, 25
feedback_coeff
BiquadFilter, 25
fft
fft.cpp, 37
fft.h, 42
fft.cpp, 34
auto_corr_real, 35
cross_corr, 35, 36
cross_corr_dual_real, 36
dft, 36
fft, 37
g_data_size, 40
g_trig, 40
pfft, 37
pfft_dual_real, 38
rdft, 38
rfft, 38, 39
rpfft, 39
set_data_size, 39
fft.h, 40
auto_corr_real, 41
cross_corr, 41
cross_corr_dual_real, 41, 42
dft, 42
fft, 42
g_data_size, 45
g_trig, 45
pfft, 43
pfft_dual_real, 43
rdft, 44
rfft, 44
rpfft, 44
set_data_size, 45
fft_im_buffer
PhaseNoiseDataProcessor, 15
fft_re_buffer
PhaseNoiseDataProcessor, 15
filt1
AntiAliasingFilter, 18
filt2
AntiAliasingFilter, 18
filt3
AntiAliasingFilter, 18
filter.h, 46
_USE_MATH_DEFINES, 46
filter_samples
PhaseNoiseDataProcessor, 12
fixed_smpl_rate
BandData, 22
FlatTop, 28
FlatTop, 28
generate, 28
rbwFactor, 28

g_data_size
fft.cpp, 40
fft.h, 45
g_trig
fft.cpp, 40
fft.h, 45
generate
FlatTop, 28
Window, 31
generateCoefficients
BiquadFilter, 24
get_x_values
PhaseNoiseDataProcessor, 12
get_y_values
PhaseNoiseDataProcessor, 12

input_buffer
BiquadFilter, 25
input_coeff
BiquadFilter, 25

MAX_DATA_SIZE
PhaseNoiseDataProcessorLib.h, 51
MAX_MEAS
PhaseNoiseDataProcessorLib.h, 51
MAX_PACKET_SIZE
PhaseNoiseDataProcessorLib.h, 52
meas_buffer
PhaseNoiseDataProcessor, 15
meas_count
PhaseNoiseDataProcessor, 15
min_window_size
BandData, 22

num_bands
BandData, 23
PhaseNoiseDataProcessor::Config, 27
num_meas
PhaseNoiseDataProcessor::Config, 27
num_sets
Band, 20

output
AntiAliasingFilter, 18
BiquadFilter, 24
OUTPUT_FILENAME
PhaseNoiseDataProcessorLib.h, 52
output_im_buffer
PhaseNoiseDataProcessor, 15
output_re_buffer
PhaseNoiseDataProcessor, 15
output_size
BandData, 23
output_start_pos
Band, 20
output_value
AntiAliasingFilter, 18
BiquadFilter, 25
output_x_buffer

PhaseNoiseDataProcessor, 16
output_y_buffer
PhaseNoiseDataProcessor, 16
overwrite_samples
PhaseNoiseDataProcessor, 12

packet_size
PhaseNoiseDataProcessor::Config, 27
pass
AntiAliasingFilter, 18
BiquadFilter, 24
pfft
fft.cpp, 37
fft.h, 43
pfft_dual_real
fft.cpp, 38
fft.h, 43
PhaseNoiseDataProcessor, 9
band_data, 15
calc_x_values, 11
calc_y_values, 11
clear_meas_buffer, 11
clear_output_buffer, 11
config, 15
convert_samples, 11
dbv_to_dbc, 11
fft_im_buffer, 15
fft_re_buffer, 15
filter_samples, 12
get_x_values, 12
get_y_values, 12
meas_buffer, 15
meas_count, 15
output_im_buffer, 15
output_re_buffer, 15
output_x_buffer, 16
output_y_buffer, 16
overwrite_samples, 12
process_fixed_smpl_rate, 12
process_fixed_window_size, 12
push_data_fixed_smpl_rate, 13
push_data_fixed_window_size, 13
read_meas_from_file, 13
retrieve_samples, 13
setup, 13, 14
smpl_buffer, 16
start_fixed_smpl_rate, 14
start_fixed_window_size, 14
stop, 14
write_output_to_file, 14
PhaseNoiseDataProcessor::Config, 25
band_mult, 26
band_shift, 26
conversion_fac, 26
cross_corr, 26
data_size, 26
num_bands, 27
num_meas, 27
packet_size, 27
smpl_rate, 27
window_en, 27
PhaseNoiseDataProcessorLib.cpp, 48
PhaseNoiseDataProcessorLib.h, 50
_USE_MATH_DEFINES, 51
MAX_DATA_SIZE, 51
MAX_MEAS, 51
MAX_PACKET_SIZE, 52
OUTPUT_FILENAME, 52
SMPL_FILENAME, 52
process_fixed_smpl_rate
PhaseNoiseDataProcessor, 12
process_fixed_window_size
PhaseNoiseDataProcessor, 12
push_data_fixed_smpl_rate
PhaseNoiseDataProcessor, 13
push_data_fixed_window_size
PhaseNoiseDataProcessor, 13

Q1
AntiAliasingFilter, 19
Q2
AntiAliasingFilter, 19
Q3
AntiAliasingFilter, 19

rbw_factor
Window, 31
rbwFactor
FlatTop, 28
Window, 31
rdft
fft.cpp, 38
fft.h, 44
read_meas_from_file
PhaseNoiseDataProcessor, 13
reset
AntiAliasingFilter, 18
BiquadFilter, 25
retrieve_samples
PhaseNoiseDataProcessor, 13
rfft
fft.cpp, 38, 39
fft.h, 44
rpfft
fft.cpp, 39
fft.h, 44

set_data_size
fft.cpp, 39
fft.h, 45
setup
PhaseNoiseDataProcessor, 13, 14
sin_lookup
TrigTable, 30
size
Window, 32
smpl_buffer
PhaseNoiseDataProcessor, 16

smpl_data_size
  BandData, 23
SMPL_FILENAME
  PhaseNoiseDataProcessorLib.h, 52
smpl_rate
  PhaseNoiseDataProcessor::Config, 27
smpl_rate_div
  Band, 20
start_fixed_smpl_rate
  PhaseNoiseDataProcessor, 14
start_fixed_window_size
  PhaseNoiseDataProcessor, 14
stop
  PhaseNoiseDataProcessor, 14

trig_table.h, 53
  _USE_MATH_DEFINES, 54
TrigTable, 29
  cos_lookup, 30
  sin_lookup, 30
  TrigTable, 29

Window, 30
  data, 31
  generate, 31
  rbw_factor, 31
  rbwFactor, 31
  size, 32
  Window, 31
window.h, 54
  _USE_MATH_DEFINES, 55
window_en
  PhaseNoiseDataProcessor::Config, 27
window_size
  Band, 20
write_output_to_file
  PhaseNoiseDataProcessor, 14