

An Inexpensive, Software-Defined IF Modulator

Generated by Doxygen 1.8.8

Wed Apr 13 2016 05:19:31

Contents

1	Bug List	1
2	Namespace Index	3
2.1	Namespace List	3
3	Hierarchical Index	5
3.1	Class Hierarchy	5
4	Class Index	7
4.1	Class List	7
5	File Index	9
5.1	File List	9
6	Namespace Documentation	11
6.1	radio Namespace Reference	11
6.1.1	Detailed Description	12
6.1.2	Enumeration Type Documentation	12
6.1.2.1	Age	12
6.1.2.2	Argument	12
6.1.2.3	Fractional	12
6.1.2.4	ModulationType	13
6.1.3	Function Documentation	13
6.1.3.1	aconj	13
6.1.3.2	fft	13
6.1.3.3	hilbert	14
6.1.3.4	ifft	15
6.1.3.5	makeIQ	16
6.1.3.6	ShowHelp	17
6.1.3.7	to_sint32	17
6.1.3.8	to_type	17
6.1.4	Variable Documentation	18
6.1.4.1	F_BASEBAND	18

6.1.4.2	F_LOWERSIDEBAND	18
6.1.4.3	F_UPPERSIDEBAND	19
6.1.4.4	FREQ_INTERMEDIATE	19
6.1.4.5	SAMPLING_RATE	19
7	Class Documentation	21
7.1	radio::Filter Class Reference	21
7.1.1	Detailed Description	21
7.1.2	Constructor & Destructor Documentation	21
7.1.2.1	Filter	21
7.1.3	Member Function Documentation	22
7.1.3.1	Pass	22
7.1.4	Member Data Documentation	22
7.1.4.1	data	22
7.1.4.2	diffEq	22
7.1.4.3	eqLength	22
7.1.4.4	size	22
7.2	radio::Gain Class Reference	23
7.2.1	Detailed Description	23
7.2.2	Constructor & Destructor Documentation	23
7.2.2.1	Gain	23
7.2.3	Member Function Documentation	23
7.2.3.1	Apply	23
7.3	radio::Modulator Class Reference	24
7.3.1	Detailed Description	24
7.3.2	Constructor & Destructor Documentation	24
7.3.2.1	Modulator	24
7.3.2.2	~Modulator	24
7.3.3	Member Function Documentation	24
7.3.3.1	Mod	24
7.4	radio::PITone Class Reference	25
7.4.1	Detailed Description	26
7.4.2	Constructor & Destructor Documentation	26
7.4.2.1	PITone	26
7.4.3	Member Function Documentation	26
7.4.3.1	Add	26
7.5	radio::Sinusoid Class Reference	27
7.5.1	Detailed Description	28
7.5.2	Constructor & Destructor Documentation	28
7.5.2.1	Sinusoid	28

7.5.2.2	~Sinusoid	28
7.5.3	Member Function Documentation	28
7.5.3.1	next	28
7.5.3.2	nextShifted	29
7.5.4	Member Data Documentation	29
7.5.4.1	frequency	29
7.5.4.2	samplingRate	29
7.5.4.3	sinIndex	29
7.5.4.4	sinIndexShifted	29
7.5.4.5	sinusoid	29
7.5.4.6	sinusoidShift90	29
8	File Documentation	31
8.1	bin/bbftest File Reference	31
8.1.1	Detailed Description	31
8.2	bbftest	31
8.3	bin/lbftest File Reference	31
8.3.1	Detailed Description	31
8.4	lbftest	31
8.5	bin/modtest File Reference	31
8.5.1	Detailed Description	31
8.6	modtest	32
8.7	bin/msintest File Reference	32
8.7.1	Detailed Description	32
8.8	msintest	32
8.9	bin/pltest File Reference	32
8.9.1	Detailed Description	32
8.10	pltest	32
8.11	bin/radio File Reference	32
8.11.1	Detailed Description	32
8.12	radio	32
8.13	bin/sintest File Reference	33
8.13.1	Detailed Description	33
8.14	sintest	33
8.15	bin/usbftest File Reference	33
8.15.1	Detailed Description	33
8.16	usbftest	33
8.17	etc/doxygen.config File Reference	33
8.17.1	Detailed Description	33
8.18	doxygen.config	33

8.19	makefile File Reference	34
8.19.1	Detailed Description	34
8.20	makefile	34
8.21	src/alsa_test.cpp File Reference	35
8.21.1	Detailed Description	35
8.21.2	Function Documentation	36
8.21.2.1	main	36
8.22	alsa_test.cpp	36
8.23	src/auxiliary.hpp File Reference	37
8.23.1	Detailed Description	38
8.24	auxiliary.hpp	38
8.25	src/baseband_filter_test.cpp File Reference	39
8.25.1	Detailed Description	40
8.25.2	Function Documentation	40
8.25.2.1	main	40
8.26	baseband_filter_test.cpp	40
8.27	src/definitions.hpp File Reference	41
8.27.1	Detailed Description	42
8.27.2	Macro Definition Documentation	42
8.27.2.1	ENUM	42
8.27.2.2	ERROR	42
8.27.3	Typedef Documentation	43
8.27.3.1	byte	43
8.27.3.2	cfloat32	43
8.27.3.3	float32	43
8.27.3.4	float64	43
8.27.3.5	fparams	43
8.27.3.6	sint16	43
8.27.3.7	sint32	43
8.27.3.8	sint64	43
8.27.3.9	sint8	43
8.27.3.10	uint16	43
8.27.3.11	uint32	43
8.27.3.12	uint64	44
8.27.3.13	uint8	44
8.28	definitions.hpp	44
8.29	src/fft_test.cpp File Reference	44
8.29.1	Detailed Description	45
8.29.2	Typedef Documentation	45
8.29.2.1	CArray	45

8.29.3	Function Documentation	45
8.29.3.1	fft	45
8.29.3.2	hilbert	46
8.29.3.3	ifft	46
8.29.3.4	main	47
8.29.4	Variable Documentation	47
8.29.4.1	PI	47
8.30	fft_test.cpp	47
8.31	src/fft_test2.cpp File Reference	49
8.31.1	Detailed Description	49
8.31.2	Function Documentation	50
8.31.2.1	main	50
8.32	fft_test2.cpp	50
8.33	src/Filter.hpp File Reference	51
8.33.1	Detailed Description	52
8.34	Filter.hpp	52
8.35	src/fvectors.hpp File Reference	53
8.35.1	Detailed Description	54
8.36	fvectors.hpp	54
8.37	src/Gain.hpp File Reference	55
8.37.1	Detailed Description	56
8.38	Gain.hpp	56
8.39	src/iq_test.cpp File Reference	57
8.39.1	Detailed Description	57
8.39.2	Function Documentation	58
8.39.2.1	main	58
8.40	iq_test.cpp	58
8.41	src/lb_filter_test.cpp File Reference	58
8.41.1	Function Documentation	59
8.41.1.1	main	59
8.42	lb_filter_test.cpp	59
8.43	src/main.cpp File Reference	60
8.43.1	Detailed Description	61
8.43.2	Function Documentation	61
8.43.2.1	main	61
8.44	main.cpp	62
8.45	src/mic_test.cpp File Reference	63
8.45.1	Detailed Description	63
8.45.2	Function Documentation	64
8.45.2.1	main	64

8.46 mic_test.cpp	64
8.47 src/Modulator.hpp File Reference	65
8.48 Modulator.hpp	66
8.49 src/modulator_test.cpp File Reference	68
8.49.1 Detailed Description	68
8.49.2 Function Documentation	68
8.49.2.1 main	68
8.50 modulator_test.cpp	69
8.51 src/multi_sinusoid_test.cpp File Reference	70
8.51.1 Detailed Description	70
8.51.2 Function Documentation	70
8.51.2.1 main	70
8.52 multi_sinusoid_test.cpp	71
8.53 src/piped_test.cpp File Reference	72
8.53.1 Detailed Description	72
8.53.2 Function Documentation	72
8.53.2.1 main	72
8.54 piped_test.cpp	73
8.55 src/pl_tone_test.cpp File Reference	73
8.55.1 Detailed Description	74
8.55.2 Function Documentation	74
8.55.2.1 main	74
8.56 pl_tone_test.cpp	74
8.57 src/PITone.hpp File Reference	75
8.57.1 Detailed Description	76
8.58 PITone.hpp	77
8.59 src/Sinusoid.hpp File Reference	77
8.59.1 Detailed Description	78
8.60 Sinusoid.hpp	78
8.61 src/sinusoid_test.cpp File Reference	79
8.61.1 Detailed Description	80
8.61.2 Function Documentation	80
8.61.2.1 main	80
8.62 sinusoid_test.cpp	81
8.63 src/usb_filter_test.cpp File Reference	81
8.63.1 Function Documentation	82
8.63.1.1 main	82
8.64 usb_filter_test.cpp	82
8.65 src/zdomain.hpp File Reference	83
8.65.1 Detailed Description	84

8.66 zdomain.hpp	84
Index	87

Chapter 1

Bug List

File [alsa_test.cpp](#)

Clicking noise from sinusoidal discontinuity

File [Filter.hpp](#)

discontinuities created at the beginning of each pass

File [modulator_test.cpp](#)

filtered SSB clicking

Namespace [radio](#)

both FM modulations don't work

clicking on the filtered SSB

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

radio	Contains the classes for the various types of modulation supported by the program	11
-----------------------	---	--------------------

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

radio::Filter	21
radio::Gain	23
radio::Modulator	24
radio::Sinusoid	27
radio::PITone	25

Chapter 4

Class Index

4.1 Class List

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

radio::Filter	21
radio::Gain	23
radio::Modulator	24
radio::PITone	25
radio::Sinusoid	27

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

makefile	Contains recipes to compile the main program and the tests programs as well as making documentation and counting total lines of code in src/	34
bin/bbftest	31
bin/lbftest	31
bin/modtest	31
bin/msintest	32
bin/pltest	32
bin/radio	32
bin/sintest	33
bin/usbftest	33
etc/doxygen.config	Contains doxygen configuration	33
src/alsa_test.cpp	Tests sinusoidal tone generation	35
src/auxiliary.hpp	Contains helper-functions for main()	37
src/baseband_filter_test.cpp	Contains a program to demonstrate the the baseband/AF filter	39
src/definitions.hpp	Contains declarations of system-independant (universal size) integers and float types, shortened type names for some commonly used types, and enumerations	41
src/fft_test.cpp	Tests FFT, IFFT, and Hilbert implementations	44
src/fft_test2.cpp	Tests FFT, IFFT, and Hilbert implementations in zdomain.hpp	49
src/Filter.hpp	Defines the Filter class	51
src/fvectors.hpp	Defines the transfer function coefficients used in the instances of the Filter class in this program	53
src/Gain.hpp	Contains the Gain class	55
src/iq_test.cpp	Generates test IQ signal	57
src/lb_filter_test.cpp	58
src/main.cpp	"brains" of the entire project	60

src/ mic_test.cpp	
Tests getting mic input via ALSA May not even compile at the moment	63
src/ Modulator.hpp	65
src/ modulator_test.cpp	
Test program to test the Modulator class	68
src/ multi_sinusoid_test.cpp	
Program to demonstrate the ability of the Sinusoid class and the sound card to generate sinu- soids accross the spectrum	70
src/ piped_test.cpp	
Contains the original program used to test the piping-in idea	72
src/ pl_tone_test.cpp	
Test program to test the PITone class	73
src/ PITone.hpp	
PITone class	75
src/ Sinusoid.hpp	
Sinusoid class	77
src/ sinusoid_test.cpp	
Test program to test the Sinusoid class	79
src/ usb_filter_test.cpp	81
src/ zdomain.hpp	
Contains the functions to manipulate sequential data in the frequency (z) domain	83

Chapter 6

Namespace Documentation

6.1 radio Namespace Reference

Contains the classes for the various types of modulation supported by the program.

Classes

- class [Filter](#)
- class [Gain](#)
- class [Modulator](#)
- class [PITone](#)
- class [Sinusoid](#)

Enumerations

- enum [Age](#) { [OLD](#), [NEW](#) }
- enum [Fractional](#) { [NUM](#), [DEN](#) }
- enum [Argument](#) { [FREQ](#) = 1, [MODE](#), [PL_TONE](#) }
- enum [ModulationType](#) {
 [ModulationType::DSB_LC](#), [ModulationType::DSB_SC](#), [ModulationType::USB_FILTERED](#), [ModulationType::USB_HILBERT](#),
 [ModulationType::LSB_FILTERED](#), [ModulationType::LSB_HILBERT](#), [ModulationType::FM_NARROW](#),
 [ModulationType::FM_WIDE](#) }

Functions

- void [ShowHelp](#) ()
- void [to_sint32](#) ([float32](#) *data, [uint32](#) size)
- [ModulationType](#) [to_type](#) (std::string str)
- void [aconj](#) ([cfloat32](#) *data, [uint32](#) size)
- void [fft](#) ([cfloat32](#) *data, [uint32](#) size)
- void [hilbert](#) ([float32](#) *data, [float32](#) *dest, [uint32](#) size)
- void [ifft](#) ([cfloat32](#) *data, [uint32](#) size)
- void [makeIQ](#) ([float32](#) *data, [float32](#) *dest, [uint32](#) size)

Variables

- [fparams F_BASEBAND](#)
- [fparams F_LOWERSIDEBAND](#)
- [fparams F_UPPERSIDEBAND](#)
- const [uint32](#) [FREQ_INTERMEDIATE](#) = 20000
- const [uint32](#) [SAMPLING_RATE](#) = 48000

6.1.1 Detailed Description

Contains the classes for the various types of modulation supported by the program.

This namespace contains all the classes, functions, and enumerations used in the application.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Bug both FM modulations don't work
clicking on the filtered SSB

6.1.2 Enumeration Type Documentation

6.1.2.1 enum [radio::Age](#)

Describes the age of a filter (from last [Pass\(\)](#) or in this [Pass\(\)](#))

Enumerator

OLD

NEW

Definition at line 52 of file [definitions.hpp](#).

6.1.2.2 enum [radio::Argument](#)

Describes the arguments in [argv](#). Never actually used.

Enumerator

FREQ

MODE

PL_TONE

Definition at line 62 of file [definitions.hpp](#).

6.1.2.3 enum [radio::Fractional](#)

Describes the numerator and denominator of a z-domain transfer function

Enumerator

NUM

DEN

Definition at line 57 of file [definitions.hpp](#).

6.1.2.4 enum `radio::ModulationType` `[strong]`

Describes a form of modulation.

Enumerator

DSB_LC
DSB_SC
USB_FILTERED
USB_HILBERT
LSB_FILTERED
LSB_HILBERT
FM_NARROW
FM_WIDE

Definition at line 67 of file [definitions.hpp](#).

6.1.3 Function Documentation

6.1.3.1 void `radio::aconj` (`cfloat32 * data`, `uint32 size`)

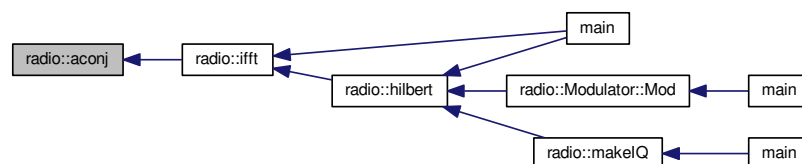
Replaces the values in an array of complex float32's with their respective conjugates.

Parameters

<i>data</i>	the array whose values should be replaced with their respective conjugates
<i>size</i>	the number of elements in the data array

Definition at line 84 of file [zdomain.hpp](#).

Here is the caller graph for this function:

6.1.3.2 void `radio::fft` (`cfloat32 * data`, `uint32 size`)

Replaces the values of an array of cfloat32's with the array's DFT using a decimation-in-frequency algorithm.

This code is based on code from http://rosettacode.org/wiki/Fast_Fourier_transform#C.↵2B.2B.

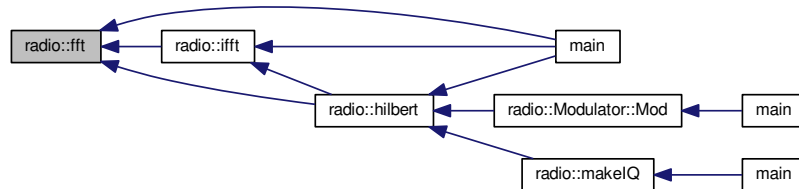
Parameters

<i>data</i>	the array whose values should be replaced with its DFT
-------------	--

<i>size</i>	the number of elements in the data array
-------------	--

Definition at line 90 of file [zdomain.hpp](#).

Here is the caller graph for this function:



6.1.3.3 void radio::hilbert (float32 * data, float32 * dest, uint32 size)

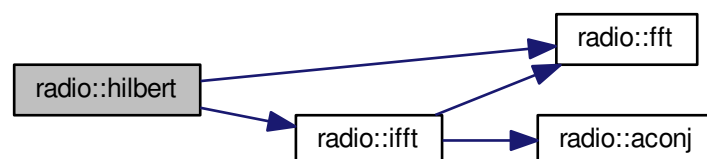
Performs the hilbert tranfor of an array of float32's.

Parameters

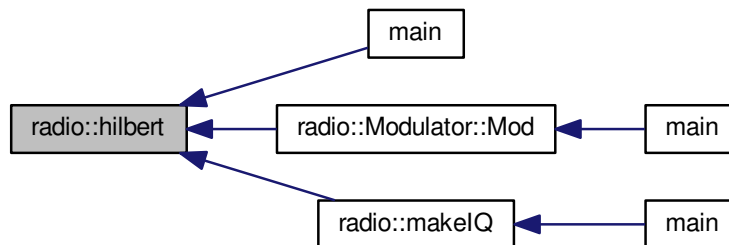
<i>data</i>	the source array of the REAL numbers of which to take the Hilbert transform
<i>dest</i>	the destination array of REAL numbers for the results of the Hilbert transform
<i>size</i>	the number of elements in the data and dest arrays

Definition at line 138 of file [zdomain.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.3.4 void radio::ifft (cfloat32 * data, uint32 size)

Replaces the values of an array of `cfloat32`'s with the array's inverse DFT.

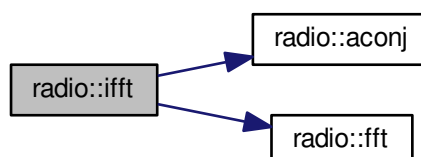
This code is based on code from http://rosettacode.org/wiki/Fast_Fourier_transform#C.↔2B.2B.

Parameters

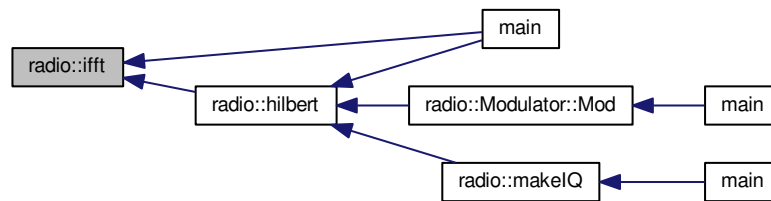
<i>data</i>	the array whose values should be replaced with its inverse DFT
<i>size</i>	the number of elements in the data array

Definition at line 161 of file `zdomain.hpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.3.5 void radio::makeIQ (float32 * data, float32 * dest, uint32 size)

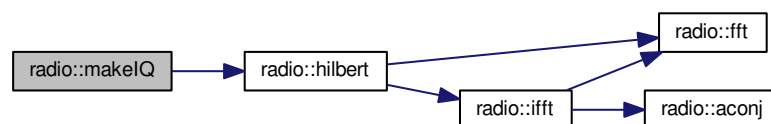
Produces an interleaved array of first an element from an original array of data and then an element from the original data's Hilbert transform. This function is intended to generate a two-channel output (I/Q output) for mixing applications.

Parameters

<i>data</i>	the original data (left channel)
<i>dest</i>	the interleaved data (left channel original data, right channel transformed data) twice the size of the original data array
<i>size</i>	the number of elements in the data array (NOT in the destination array)

Definition at line 171 of file [zdomain.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.3.6 void radio::ShowHelp ()

Displays the help information.

Definition at line 22 of file [auxiliary.hpp](#).

Here is the caller graph for this function:



6.1.3.7 void radio::to_sint32 (float32 * data, uint32 size)

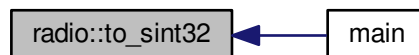
Converts float32 samples to sint32 samples. Rounds conversion to nearest integer.

Parameters

<i>data</i>	the array containing the float32 samples that are directly replaced by their respective sint32 representations
<i>size</i>	the number of elements in the data array

Definition at line 62 of file [auxiliary.hpp](#).

Here is the caller graph for this function:



6.1.3.8 ModulationType radio::to_type (std::string str)

Converts a string representation of the supported modulation types (see [ShowHelp\(\)](#) documentation) to the enum ModulationType value.

This function is not as elegant as it could be. Ideally, I would have used a `std::map<string, ModulationType>` rather than a long series of if-else's.

Parameters

<i>str</i>	type of modulation in typed form
------------	----------------------------------

Returns

enum value of the type of modulation

Definition at line 80 of file [auxiliary.hpp](#).

Here is the caller graph for this function:



6.1.4 Variable Documentation

6.1.4.1 fparams radio::F_BASEBAND

Initial value:

```

= { std::vector<float64> {
    0.0008977019461,
    -0.002215694636,
    0.001372192986,
    0.001372192986,
    -0.002215694636,
    0.0008977019461
  }, std::vector<float64> {
    1,
    -4.678616047,
    8.822912216,
    -8.379911423,
    4.007629871,
    -0.7719064355
  } }

```

Baseband filter coefficients. Generated with MATLAB 2015A.

Definition at line 19 of file [fvectors.hpp](#).

6.1.4.2 fparams radio::F_LOWERSIDEBAND

Initial value:

```

= { std::vector<float64> {
    0.2758039069174,
    2.763578787693,
    12.83915022756,
    36.47584850651,
    70.37084637368,
    96.76893503179,
    96.76893503179,
    70.37084637368,
    36.47584850651,
    12.83915022756,
    2.763578787693,
    0.2758039069174
  }, std::vector<float64> {
    1,
    7.605497780083,
    27.34180552438,
    60.83375457605,
    92.60908886875,
    100.8363857,
    79.74796574736,

```

```

        45.4982252145,
        18.13566776308,
        4.690036472717,
        0.6617552879305,
        0.0281427334611
    } }

```

Lower-sideband filter coefficients. Generated with MATLAB 2015A.

Definition at line 38 of file [fvectors.hpp](#).

6.1.4.3 fparams radio::F_UPPERSIDEBAND

Initial value:

```

= { std::vector<float64> {
    0.001690387681463,
    0.01145271586989,
    0.03591799189724,
    0.06576926098562,
    0.07119343282702,
    0.03156377419766,
    -0.03156377419766,
    -0.07119343282702,
    -0.06576926098562,
    -0.03591799189724,
    -0.01145271586989,
    -0.001690387681463
}, std::vector<float64> {
    1,
    9.465175013624,
    41.62402815905,
    112.0971027069,
    205.2097686473,
    267.9378582311,
    254.486805213,
    175.7772755115,
    86.51619894548,
    28.89988093561,
    5.89781461091,
    0.5572910543053
} }

```

Upper-sideband filter coefficients. Generated with MATLAB 2015A.

Definition at line 69 of file [fvectors.hpp](#).

6.1.4.4 const uint32 radio::FREQ_INTERMEDIATE = 20000

The default intermediate carrier frequency

Definition at line 28 of file [Modulator.hpp](#).

6.1.4.5 const uint32 radio::SAMPLING_RATE = 48000

The default sampling rate (frequency)

Definition at line 33 of file [Modulator.hpp](#).

Chapter 7

Class Documentation

7.1 radio::Filter Class Reference

```
#include <Filter.hpp>
```

Public Member Functions

- [Filter](#) ([float32](#) *[data](#), [uint32](#) [size](#), [fparams](#) &[diffEq](#))
- void [Pass](#) ()

Protected Attributes

- [uint8](#) [eqLength](#)
- [uint32](#) [size](#)
- [float32](#) * [data](#)
- [fparams](#) [diffEq](#)

7.1.1 Detailed Description

This class implements a z-domain filter on a specified array of float32's (a.k.a. singles, floats). It requires the transfer function coefficients already be calculated (i.e., it does not generate the coefficients based on desired filter characteristics). MATLAB and its Signal Processing Toolbox can be used to generate the coefficients.

While this class is designed to implement a single-section filter, several instances of the class can be created and run over the data array sequentially to effectively implement a multi-section filter.

Definition at line 28 of file [Filter.hpp](#).

7.1.2 Constructor & Destructor Documentation

7.1.2.1 radio::Filter::Filter ([float32](#) * [data](#), [uint32](#) [size](#), [fparams](#) & [diffEq](#))

Initializes [Filter](#) based on a difference equation.

Parameters

<i>data</i>	array to be filtered. The filtered data will be placed here.
<i>size</i>	number of elements in the data array
<i>diffEq</i>	a vector containing two vectors of float32's (a.k.a. singles, floats), containing the numerator and denominator coefficients, respectively, of the z-domain transfer function of the filter in descending order (z^0 , z^{-1} , z^{-2} , etc.).

Definition at line 80 of file [Filter.hpp](#).

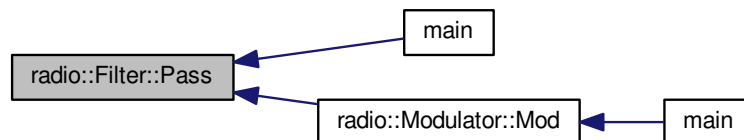
7.1.3 Member Function Documentation

7.1.3.1 void radio::Filter::Pass ()

Passes the data array through the digital filter but does not account for previous $x[n]$ and $y[n]$ values from the previous call to [Pass\(\)](#).

Definition at line 87 of file [Filter.hpp](#).

Here is the caller graph for this function:



7.1.4 Member Data Documentation

7.1.4.1 float32* radio::Filter::data [protected]

A pointer to the data array that should be filtered when [Pass\(\)](#) is called.

Definition at line 69 of file [Filter.hpp](#).

7.1.4.2 fparams radio::Filter::diffEq [protected]

A vector containing two vectors of float32's (a.k.a. singles, floats), containing the numerator and denominator coefficients, respectively, of the z-domain transfer function of the filter in descending order (z^0 , z^{-1} , z^{-2} , etc.).

Definition at line 77 of file [Filter.hpp](#).

7.1.4.3 uint8 radio::Filter::eqLength [protected]

The number of terms in the numerator (or denominator) of the transfer function.

Definition at line 58 of file [Filter.hpp](#).

7.1.4.4 uint32 radio::Filter::size [protected]

The number of elements in the data array.

Definition at line 63 of file [Filter.hpp](#).

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

- [src/Filter.hpp](#)

7.2 radio::Gain Class Reference

```
#include <Gain.hpp>
```

Public Member Functions

- [Gain](#) ([float32](#) *data, [uint32](#) size, [float32](#) gaindB)
- void [Apply](#) ()

7.2.1 Detailed Description

Applies a gain to a (baseband) signal.

Definition at line 18 of file [Gain.hpp](#).

7.2.2 Constructor & Destructor Documentation

7.2.2.1 radio::Gain::Gain ([float32](#) * data, [uint32](#) size, [float32](#) gaindB)

Initializes a [Gain](#) object and converts gain from decibels to a standard value.

Parameters

<i>data</i>	the signal to which the gain is applied
<i>size</i>	the number of elements in the data array
<i>gaindB</i>	the desired gain in decibels (of power)

Definition at line 61 of file [Gain.hpp](#).

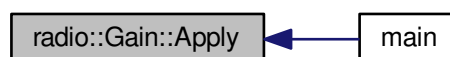
7.2.3 Member Function Documentation

7.2.3.1 void radio::Gain::Apply ()

Applies the gain to the signal contained in the data array

Definition at line 67 of file [Gain.hpp](#).

Here is the caller graph for this function:



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

- [src/Gain.hpp](#)

7.3 radio::Modulator Class Reference

```
#include <Modulator.hpp>
```

Public Member Functions

- [Modulator](#) (float32 data[], uint32 size, ModulationType type, float32 freqInter=FREQ_INTERMEDIATE, uint32 rate=SAMPLING_RATE)
- [~Modulator](#) ()
- void [Mod](#) ()

7.3.1 Detailed Description

This class, while not intended to be called directly, is a superclass for the classes of the modulation forms used in this project.

Definition at line 39 of file [Modulator.hpp](#).

7.3.2 Constructor & Destructor Documentation

7.3.2.1 [radio::Modulator::Modulator](#) (float32 data[], uint32 size, ModulationType type, float32 freqInter = FREQ_INTERMEDIATE, uint32 rate = SAMPLING_RATE)

Creates a [Modulator](#) with the specified parameters. Intended to be called only by subclasses.

Parameters

<i>freqInter</i>	the frequency of the IF carrier sinusoid
<i>rate</i>	the sampling rate of the baseband and IF signals
<i>data</i>	the array holding initially the baseband signal
<i>size</i>	the number of elements in data
<i>type</i>	form of modulation to use

Definition at line 103 of file [Modulator.hpp](#).

7.3.2.2 [radio::Modulator::~~Modulator](#) ()

Definition at line 117 of file [Modulator.hpp](#).

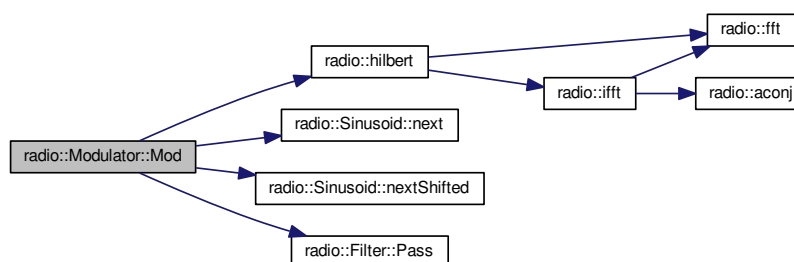
7.3.3 Member Function Documentation

7.3.3.1 [void radio::Modulator::Mod](#) ()

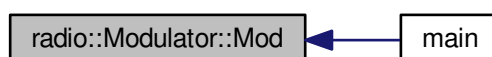
Modulates the audio currently in the data array.

Definition at line 121 of file [Modulator.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



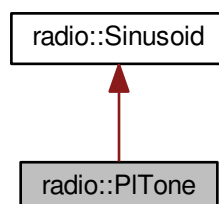
The documentation for this class was generated from the following file:

- [src/Modulator.hpp](#)

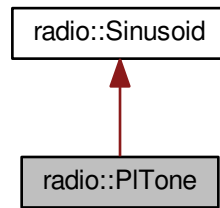
7.4 radio::PITone Class Reference

```
#include <PlTone.hpp>
```

Inheritance diagram for `radio::PITone`:



Collaboration diagram for `radio::PITone`:



Public Member Functions

- `PITone` (`float32` amplitude, `float32 *`data, `uint32` size, `float32` frequency, `uint32` samplingRate)
- void `Add` ()

7.4.1 Detailed Description

This class creates a CTCSS subcarrier (PL tone) at a specified frequency in a baseband signal.

Definition at line 18 of file [PITone.hpp](#).

7.4.2 Constructor & Destructor Documentation

7.4.2.1 `radio::PITone::PITone (float32 amplitude, float32 * data, uint32 size, float32 frequency, uint32 samplingRate)`

Creates a `PITone` object.

Parameters

<i>amplitude</i>	the amplitude (0-1) of the subcarrier. Assumes baseband signal has a peak-to-peak range of -1 to 1.
<i>data</i>	an array containing a portion of the discrete baseband signal
<i>size</i>	the number of elements in the data array
<i>frequency</i>	the frequency of the CTCSS tone in the baseband (not in the IF or RF signals)
<i>samplingRate</i>	the sampling frequency of the baseband signal

Definition at line 63 of file [PITone.hpp](#).

7.4.3 Member Function Documentation

7.4.3.1 void `radio::PITone::Add` ()

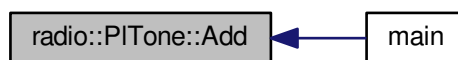
Adds the CTCSS tone to the baseband signal.

Definition at line 75 of file [PITone.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



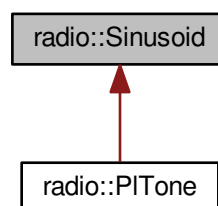
The documentation for this class was generated from the following file:

- [src/PITone.hpp](#)

7.5 radio::Sinusoid Class Reference

```
#include <Sinusoid.hpp>
```

Inheritance diagram for `radio::Sinusoid`:



Public Member Functions

- [Sinusoid](#) ([float32 frequency](#), [uint32 samplingRate=48000](#))
- [~Sinusoid](#) ()
- [float32 next](#) ()
- [float32 nextShifted](#) ()

Protected Attributes

- `float32 frequency`
- `uint32 sinIndex = 0`
- `uint32 sinIndexShifted = 0`
- `uint32 samplingRate`
- `float32 * sinusoid`
- `float32 * sinusoidShift90`

7.5.1 Detailed Description

This class creates an easy-to-call sinusoid that will preserve its phase throughout its lifespan. Essentially, it is a ring buffer.

Definition at line 20 of file [Sinusoid.hpp](#).

7.5.2 Constructor & Destructor Documentation

7.5.2.1 `radio::Sinusoid::Sinusoid (float32 frequency, uint32 samplingRate = 48000)`

Creates a ring-buffer sinusoid.

Definition at line 77 of file [Sinusoid.hpp](#).

7.5.2.2 `radio::Sinusoid::~Sinusoid ()`

Free arrays malloc'd in the constructor.

Definition at line 92 of file [Sinusoid.hpp](#).

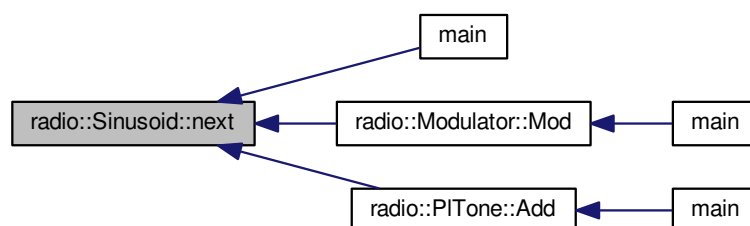
7.5.3 Member Function Documentation

7.5.3.1 `float32 radio::Sinusoid::next ()`

Provides the next value of the sinusoid in a manner consistent with a ring buffer.

Definition at line 97 of file [Sinusoid.hpp](#).

Here is the caller graph for this function:



7.5.3.2 float32 radio::Sinusoid::nextShifted ()

Provides the next value of the sinusoid shifted 90 degrees in a manner constant with a ring buffer.

Definition at line 102 of file [Sinusoid.hpp](#).

Here is the caller graph for this function:



7.5.4 Member Data Documentation

7.5.4.1 float32 radio::Sinusoid::frequency [protected]

The frequency of the sinusoid

Definition at line 48 of file [Sinusoid.hpp](#).

7.5.4.2 uint32 radio::Sinusoid::samplingRate [protected]

The sampling rate

Definition at line 63 of file [Sinusoid.hpp](#).

7.5.4.3 uint32 radio::Sinusoid::sinIndex = 0 [protected]

The current index of the sinusoid's unshifted array

Definition at line 53 of file [Sinusoid.hpp](#).

7.5.4.4 uint32 radio::Sinusoid::sinIndexShifted = 0 [protected]

The current index of the shifted sinusoid's array

Definition at line 58 of file [Sinusoid.hpp](#).

7.5.4.5 float32* radio::Sinusoid::sinusoid [protected]

Initialized as an array of the sinusoid values

Definition at line 68 of file [Sinusoid.hpp](#).

7.5.4.6 float32* radio::Sinusoid::sinusoidShift90 [protected]

Initialized as an array of the sinusoid values shifted 90 degrees

Definition at line 74 of file [Sinusoid.hpp](#).

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

- [src/Sinusoid.hpp](#)

Chapter 8

File Documentation

8.1 bin/bbftest File Reference

8.1.1 Detailed Description

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [bbftest](#).

8.2 bbftest

```
00001 basebandfiltertest | aplay -c 2 -r 48000 -t raw -f S32_LE
```

8.3 bin/lbftest File Reference

8.3.1 Detailed Description

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [lbftest](#).

8.4 lbftest

```
00001 lowersidebandftest | aplay -c 2 -r 48000 -t raw -f S32_LE
```

8.5 bin/modtest File Reference

8.5.1 Detailed Description

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [modtest](#).

8.6 modtest

```
00001 OPTIONS="-c 2 -r 48000 -t raw -f S32_LE -q"  
00002 modulatortest $1 $2 $3 | aplay $OPTIONS -D plughw:0,0
```

8.7 bin/msintest File Reference

8.7.1 Detailed Description

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [msintest](#).

8.8 msintest

```
00001 multisinusoidtest | aplay -c 2 -r 48000 -t raw -f S32_LE
```

8.9 bin/pltest File Reference

8.9.1 Detailed Description

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [pltest](#).

8.10 pltest

```
00001 OPTIONS="-c 2 -r 48000 -t raw -f S32_LE"  
00002 pltonetest $1 | aplay $OPTIONS
```

8.11 bin/radio File Reference

8.11.1 Detailed Description

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [radio](#).

8.12 radio

```
00001 OPTIONS="-r 48000 -t raw -q"  
00002 arecord $OPTIONS -c 1 -D plughw:1,0 -f FLOAT_LE | sdr $1 $2 $3 | \  
00003 aplay $OPTIONS -c 2 -f S32_LE -D plughw:0,0
```

8.13 bin/sintest File Reference

8.13.1 Detailed Description

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [sintest](#).

8.14 sintest

```
00001 OPTIONS="-c 2 -r 48000 -t raw -f S32_LE"
00002 sinusoidtest $1 | aplay $OPTIONS
```

8.15 bin/usbftest File Reference

8.15.1 Detailed Description

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [usbftest](#).

8.16 usbftest

```
00001 uppersidebandftest | aplay -c 2 -r 48000 -t raw -f S32_LE
```

8.17 etc/doxygen.config File Reference

Contains doxygen configuration.

8.17.1 Detailed Description

Contains doxygen configuration.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [doxygen.config](#).

8.18 doxygen.config

```
00001 PROJECT_NAME = "An Inexpensive, Software-Defined IF Modulator"
00002
00003 INPUT = makefile src/ etc/doxygen.config bin/bbftest bin/modtest bin/msintest bin/lsbftest bin/pltest
        bin/radio bin/sintest bin/usbftest
00004 OUTPUT_DIRECTORY = doc/
00005
00006 GENERATE_HTML = YES
00007 GENERATE_RTF = YES
```

```

00008 GENERATE_LATEX = YES
00009 GENERATE_MAN = YES
00010 GENERATE_XML = NO
00011 GENERATE_DOCBOOK = NO
00012
00013 USE_PDF_LATEX = YES
00014 USE_PDF_HYPERLINKS = YES
00015
00016 RECURSIVE = YES
00017 SOURCE_BROWSER = YES
00018 SOURCE_TOOLTIPS = YES
00019 EXTRACT_ALL = YES
00020 LATEX_SOURCE_CODE = YES
00021 DISABLE_INDEX = NO
00022 GENERATE_TREEVIEW = YES
00023 SEARCHENGINE = YES
00024 SERVER_BASED_SEARCH = NO
00025
00026 HAVE_DOT = YES
00027 CALL_GRAPH = YES
00028 CALLER_GRAPH = YES

```

8.19 makefile File Reference

Contains recipes to compile the main program and the tests programs as well as making documentation and counting total lines of code in src/.

8.19.1 Detailed Description

Contains recipes to compile the main program and the tests programs as well as making documentation and counting total lines of code in src/.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [makefile](#).

8.20 makefile

```

00001 GCC = g++ -g -std=gnu++14
00002
00003 alsa-test:
00004     $(GCC) src/alsa_test.cpp -o bin/alsatest -O0 -lasound
00005
00006 baseband-filter-test:
00007     $(GCC) src/baseband_filter_test.cpp -o bin/basebandfiltertest
00008
00009 count:
00010     grep -r "src/" -e "Samuel Andrew Wisner" -l | xargs wc -l
00011
00012 docs:
00013     rm -r doc/
00014     doxygen etc/doxygen.config
00015     cd doc/latex; make pdf;
00016     git reset
00017     git add doc/.
00018     git --no-pager log > etc/log.txt
00019     git add etc/log.txt
00020     git commit -m "Updated documentation."
00021     git push
00022
00023 fft-test:
00024     $(GCC) src/fft_test.cpp -o bin/fft-test
00025
00026 fft-test2:
00027     $(GCC) src/fft_test2.cpp -o bin/fft-test2
00028
00029 iq-test:
00030     $(GCC) src/iq_test.cpp -o bin/iqtest
00031
00032 multi-sinusoid-test:

```

```

00033 $(GCC) src/multi_sinusoid_test.cpp -o bin/multisinusoidtest
00034
00035 modulator-test:
00036 $(GCC) src/modulator_test.cpp -o bin/modulatortest
00037
00038 lsb-filter-test:
00039 $(GCC) src/lsb_filter_test.cpp -o bin/lowersidebandftest
00040
00041 pl-tone-test:
00042 $(GCC) src/pl_tone_test.cpp -o bin/pltonetest
00043
00044 radio:
00045 $(GCC) src/main.cpp -o bin/sdr
00046
00047 sinusoid-test:
00048 $(GCC) src/sinusoid_test.cpp -o bin/sinusoidtest
00049
00050 usb-filter-test:
00051 $(GCC) src/usb_filter_test.cpp -o bin/uppersidebandftest
00052
00053

```

8.21 src/alsa_test.cpp File Reference

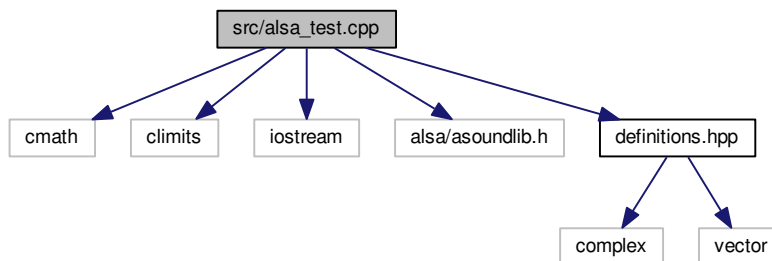
Tests sinusoidal tone generation.

```

#include <cmath>
#include <climits>
#include <iostream>
#include <alsa/asoundlib.h>
#include "definitions.hpp"

```

Include dependency graph for alsa_test.cpp:



Functions

- int [main](#) ()

8.21.1 Detailed Description

Tests sinusoidal tone generation.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Bug Clicking noise from sinusoidal discontinuity

Definition in file [alsa_test.cpp](#).

8.21.2 Function Documentation

8.21.2.1 int main ()

This program tests sinusoidal speaker output through the ALSA API. Not sure if it works. When it did at least compile and run, it produced a sinusoid with an approximately twice-per-second clicking noise.

Definition at line 22 of file [alsa_test.cpp](#).

8.22 alsa_test.cpp

```

00001
00008 #include <cmath>
00009 #include <climits>
00010 #include <iostream>
00011 #include <alsa/asoundlib.h>
00012
00013 #include "definitions.hpp"
00014
00015 using namespace std;
00016
00022 int main() {
00023     int ret;
00024
00025     snd_pcm_t* pcm_handle; // device handle
00026     snd_pcm_stream_t stream = SND_PCM_STREAM_PLAYBACK;
00027     snd_pcm_hw_params_t* hwparams; // hardware information
00028     char* pcm_name = strdup("plughw:1,0"); // on-board audio jack
00029     int rate = 48000;
00030
00031     const uint16 freq = 440;
00032     long unsigned int bufferSize = 4096*4; // anything >8192 causes seg fault
00033     const uint32 len = bufferSize*100;
00034     const float32 arg = 2 * 3.141592 * freq / rate;
00035     sint16 vals[len];
00036
00037     long unsigned int count = 0;
00038
00039     for(uint32 i = 0; i < len; i = i + 2) {
00040         vals[i] = (sint16)(SHRT_MAX * cos(arg * i/2) + 0.5);
00041         vals[i+1] = vals[i];
00042     }
00043
00044     ret = snd_pcm_open(&pcm_handle, pcm_name, stream, 0);
00045     cout << "Opening: " << snd_strerror(ret) << endl;
00046
00047     ret = snd_pcm_hw_params_any(pcm_handle, hwparams);
00048     cout << "Initializing hwparams structure: " << snd_strerror(ret) << endl;
00049
00050     ret = snd_pcm_hw_params_set_access(pcm_handle, hwparams,
00051         SND_PCM_ACCESS_RW_INTERLEAVED);
00052     cout << "Setting access: " << snd_strerror(ret) << endl;
00053
00054     ret = snd_pcm_hw_params_set_format(pcm_handle, hwparams,
00055         SND_PCM_FORMAT_S16_LE);
00056     cout << "Setting format: " << snd_strerror(ret) << endl;
00057
00058     ret = snd_pcm_hw_params_set_rate(pcm_handle, hwparams,
00059         rate, (int)0);
00060     cout << "Setting rate: " << snd_strerror(ret) << endl;
00061
00062     ret = snd_pcm_hw_params_set_channels(pcm_handle, hwparams, 2);
00063     cout << "Setting channels: " << snd_strerror(ret) << endl;
00064
00065     ret = snd_pcm_hw_params_set_periods(pcm_handle, hwparams, 2, 0);
00066     cout << "Setting periods: " << snd_strerror(ret) << endl;
00067
00068     ret = snd_pcm_hw_params_set_buffer_size_near(pcm_handle, hwparams,
00069         &bufferSize);
00070     cout << "Setting buffer size: " << snd_strerror(ret) << endl;
00071
00072     ret = snd_pcm_hw_params(pcm_handle, hwparams);
00073     cout << "Applying parameters: " << snd_strerror(ret) << endl;
00074
00075     // ret = snd_pcm_hw_params_get_period_size(hwparams, &count, 0);
00076     cout << "Actual period size: " << count << endl;
00077     cout << "Returned: " << snd_strerror(ret) << endl;
00078
00079
00080

```

```

00081     cout << endl << endl;
00082
00083
00084     const void* ptr[100];
00085
00086     for(int i = 0; i < 100; i++) {
00087         ptr[i] = (const void*)&vals + bufferSize*i;
00088     }
00089
00090     int err;
00091
00092     for(int i = 0; i < 100; i++) {
00093         do {
00094             ret = snd_pcm_writei(pcm_handle,
00095                                 ptr[i], count);
00096
00097             if(ret < 0) {
00098                 err = snd_pcm_prepare(pcm_handle);
00099                 cout << "Preparing: " << snd_strerror(err)
00100                     << endl;
00101             }
00102         } while(ret < 0);
00103
00104         cout << "Writing data: " << ret << endl;
00105     }
00106 }

```

8.23 src/auxiliary.hpp File Reference

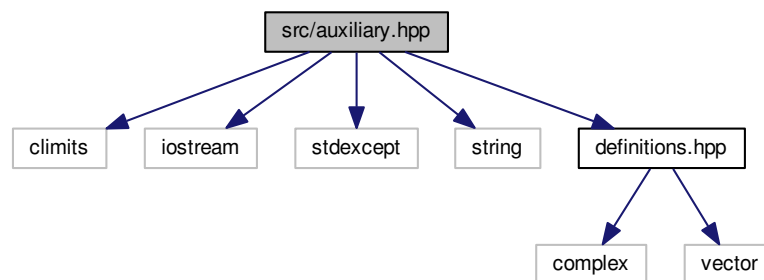
Contains helper-functions for `main()`.

```

#include <climits>
#include <iostream>
#include <stdexcept>
#include <string>
#include "definitions.hpp"

```

Include dependency graph for auxiliary.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- [radio](#)

Contains the classes for the various types of modulation supported by the program.

Functions

- void [radio::ShowHelp](#) ()
- void [radio::to_sint32](#) (float32 *data, uint32 size)
- ModulationType [radio::to_type](#) (std::string str)

8.23.1 Detailed Description

Contains helper-functions for [main\(\)](#).

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [auxiliary.hpp](#).

8.24 auxiliary.hpp

```

00001
00007 #ifndef auxiliary_H
00008 #define auxiliary_H
00009
00010 #include <climits>
00011 #include <iostream>
00012 #include <stdexcept>
00013 #include <string>
00014
00015 #include "definitions.hpp"
00016
00017 namespace radio {
00018
00022     void ShowHelp() {
00023         std::cerr << std::endl << "Usage: radio [MODE] [MIC GAIN] "
00024             "[PL TONE]" << std::endl << std::endl
00025             << "MODE: one of the following types "
00026             "of modulation" << std::endl << std::endl;
00027
00028         std::cerr << "dsbfc\t\tDouble sideband, large carrier" << std::endl
00029             << "am\t\tAlias for dsbfc" << std::endl
00030             << "dsbsc\t\tDouble sideband, suppressed carrier" << std::endl
00031             << "lsbhil\t\tLower sideband created via Hilbert transform"
00032             << std::endl
00033             << "lsbfilt\t\tLower sideband created via digital low-pass filter"
00034             << std::endl
00035             << "usbhil\t\tUpper sideband created via Hilbert transform"
00036             << std::endl
00037             << "usbfilt\t\tUpper sideband created via digital high-pass filter"
00038             << std::endl
00039             // << "nfm\t\tFrequency modulation, 2.5 kHz bandwidth"
00040             << std::endl;
00041             // << "wfm\t\tFrequency modulation, 5 kHz bandwidth" << std::endl
00042             // << "fm\t\tAlias for wfm" << std::endl << std::endl;
00043
00044         std::cerr << "MIC GAIN: Microphone power gain expressed in decibels"
00045             << std::endl << std::endl;
00046
00047         std::cerr << "PL TONE: Optional specification for CTCSS tone from "
00048             "60-260 Hz" << std::endl << std::endl;
00049
00050         std::exit(ERROR);
00051     }
00052
00062     void to_sint32(float32* data, uint32 size) {
00063         for(uint32 i = 0; i < size; i++) {
00064             ((sint32*)data)[i] = (sint32)(data[i] * INT_MAX + 0.5);
00065         }

```



```

00066     }
00067
00080     ModulationType to_type(std::string str) {
00081         ModulationType type;
00082
00083         if(str == "dsblc" || str == "am") {
00084             type = ModulationType::DSB_LC;
00085         } else if(str == "dsbsc") {
00086             type = ModulationType::DSB_SC;
00087         } else if(str == "lsbhil") {
00088             type = ModulationType::LSB_HILBERT;
00089         } else if(str == "lsbfilt") {
00090             type = ModulationType::LSB_FILTERED;
00091         } else if(str == "usbhil") {
00092             type = ModulationType::USB_HILBERT;
00093         } else if(str == "usbfilt") {
00094             type = ModulationType::USB_FILTERED;
00095         } else if(str == "wfm" || str == "fm") {
00096             type = ModulationType::FM_NARROW;
00097         } else if(str == "nfm") {
00098             type = ModulationType::FM_WIDE;
00099         } else {
00100             throw std::logic_error("The given modulation type is invalid!");
00101         }
00102
00103         return type;
00104     }
00105 }
00106
00107 #endif

```

8.25 src/baseband_filter_test.cpp File Reference

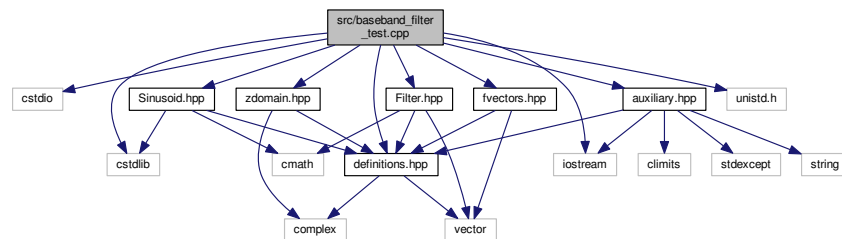
Contains a program to demonstrate the the baseband/AF filter.

```

#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <unistd.h>
#include "auxiliary.hpp"
#include "definitions.hpp"
#include "Filter.hpp"
#include "fvectors.hpp"
#include "Sinusoid.hpp"
#include "zdomain.hpp"

```

Include dependency graph for baseband_filter_test.cpp:



Functions

- int [main](#) (int argc, char *argv[])

8.25.1 Detailed Description

Contains a program to demonstrate the the baseband/AF filter.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [baseband_filter_test.cpp](#).

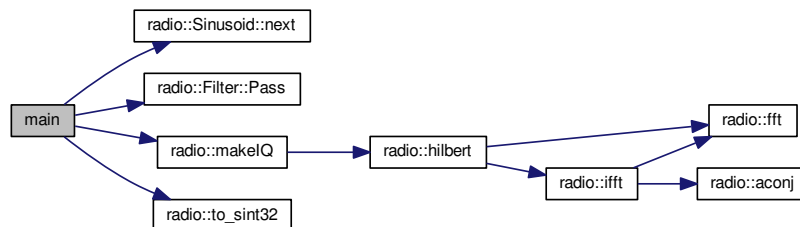
8.25.2 Function Documentation

8.25.2.1 `int main (int argc, char * argv[])`

Program to test the Filter class and the baseband filter coefficients.

Definition at line 25 of file [baseband_filter_test.cpp](#).

Here is the call graph for this function:



8.26 baseband_filter_test.cpp

```

00001
00007 #include <cstdio>
00008 #include <cstdlib>
00009 #include <iostream>
00010 #include <unistd.h>
00011
00012 #include "auxiliary.hpp"
00013 #include "definitions.hpp"
00014 #include "Filter.hpp"
00015 #include "fvectors.hpp"
00016 #include "Sinusoid.hpp"
00017 #include "zdomain.hpp"
00018
00019 using namespace std;
00020 using namespace radio;
00021
00025 int main(int argc, char* argv[]) {
00026
00027     // Constants
00028     const uint16 BUFFER_SIZE = 48000;
00029
00030     // Declare primitive Variables
00031     uint8 i = 0;
00032     uint8 size = 0;
00033     uint16 delta = 250;
00034     float32 dataBuffer[BUFFER_SIZE];
00035     float32 iqBuffer[2 * BUFFER_SIZE];
00036
00037     // create 1 sec of audio
00038     for(uint16 f = delta; f <= 3000; f += delta, i++) {
00039         Sinusoid sinusoid(f);
00040
00041         for(uint16 i = 0; i < BUFFER_SIZE; i++) {

```

8.27 src/definitions.hpp File Reference

Include dependency graph for definitions.hpp:



- Contains the classes for the various types of modulation supported by the program.*

Macros

- `#define ENUM` signed char
- `#define ERROR` -1

Typedefs

- `typedef unsigned char` [byte](#)
- `typedef unsigned char` [uint8](#)
- `typedef signed char` [sint8](#)
- `typedef unsigned short` [uint16](#)
- `typedef signed short` [sint16](#)
- `typedef unsigned int` [uint32](#)
- `typedef signed int` [sint32](#)
- `typedef unsigned long long` [uint64](#)
- `typedef signed long long` [sint64](#)
- `typedef float` [float32](#)
- `typedef double` [float64](#)
- `typedef std::complex< float32 >` [cfloat32](#)
- `typedef std::vector`
`< std::vector< float64 > >` [fparams](#)

Enumerations

- `enum radio::Age` { [radio::OLD](#), [radio::NEW](#) }
- `enum radio::Fractional` { [radio::NUM](#), [radio::DEN](#) }
- `enum radio::Argument` { [radio::FREQ](#) = 1, [radio::MODE](#), [radio::PL_TONE](#) }
- `enum radio::ModulationType` {
[radio::ModulationType::DSB_LC](#), [radio::ModulationType::DSB_SC](#), [radio::ModulationType::USB_FILTERED](#),
[radio::ModulationType::USB_HILBERT](#),
[radio::ModulationType::LSB_FILTERED](#), [radio::ModulationType::LSB_HILBERT](#), [radio::ModulationType::FM_NARROW](#), [radio::ModulationType::FM_WIDE](#) }

8.27.1 Detailed Description

Contains declarations of system-independant (universal size) integers and float types, shortened type names for some commonly used types, and enumerations.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [definitions.hpp](#).

8.27.2 Macro Definition Documentation

8.27.2.1 `#define ENUM` signed char

Definition at line 15 of file [definitions.hpp](#).

8.27.2.2 `#define ERROR` -1

Definition at line 16 of file [definitions.hpp](#).

8.27.3 Typedef Documentation

8.27.3.1 typedef unsigned char byte

Definition at line 18 of file [definitions.hpp](#).

8.27.3.2 typedef std::complex<float32> cfloat32

Defines a type for complex float32's.

Definition at line 37 of file [definitions.hpp](#).

8.27.3.3 typedef float float32

Definition at line 31 of file [definitions.hpp](#).

8.27.3.4 typedef double float64

Definition at line 32 of file [definitions.hpp](#).

8.27.3.5 typedef std::vector<std::vector<float64> > fparams

Defines a type for the filter coefficients.

Definition at line 42 of file [definitions.hpp](#).

8.27.3.6 typedef signed short sint16

Definition at line 23 of file [definitions.hpp](#).

8.27.3.7 typedef signed int sint32

Definition at line 26 of file [definitions.hpp](#).

8.27.3.8 typedef signed long long sint64

Definition at line 29 of file [definitions.hpp](#).

8.27.3.9 typedef signed char sint8

Definition at line 20 of file [definitions.hpp](#).

8.27.3.10 typedef unsigned short uint16

Definition at line 22 of file [definitions.hpp](#).

8.27.3.11 typedef unsigned int uint32

Definition at line 25 of file [definitions.hpp](#).

8.27.3.12 typedef unsigned long long uint64

Definition at line 28 of file [definitions.hpp](#).

8.27.3.13 typedef unsigned char uint8

Definition at line 19 of file [definitions.hpp](#).

8.28 definitions.hpp

```

00001
00009 #ifndef definitions_H
00010 #define definitions_H
00011
00012 #include <complex>
00013 #include <vector>
00014
00015 #define ENUM signed char
00016 #define ERROR -1
00017
00018 typedef unsigned char byte;
00019 typedef unsigned char uint8;
00020 typedef signed char sint8;
00021
00022 typedef unsigned short uint16;
00023 typedef signed short sint16;
00024
00025 typedef unsigned int uint32;
00026 typedef signed int sint32;
00027
00028 typedef unsigned long long uint64;
00029 typedef signed long long sint64;
00030
00031 typedef float float32;
00032 typedef double float64;
00033
00037 typedef std::complex<float32> cfloat32;
00038
00042 typedef std::vector<std::vector<float64>> fparams;
00043
00048 namespace radio {
00052     enum Age { OLD, NEW };
00053
00057     enum Fractional { NUM, DEN };
00058
00062     enum Argument { FREQ = 1, MODE, PL_TONE };
00063
00067     enum class ModulationType { DSB_LC, DSB_SC,
00068         USB_FILTERED, USB_HILBERT,
00069         LSB_FILTERED, LSB_HILBERT, FM_NARROW,
00070         FM_WIDE };
00071 }
00072 #endif
00073 // Doxygen descriptions for non-code files
00074

```

8.29 src/fft_test.cpp File Reference

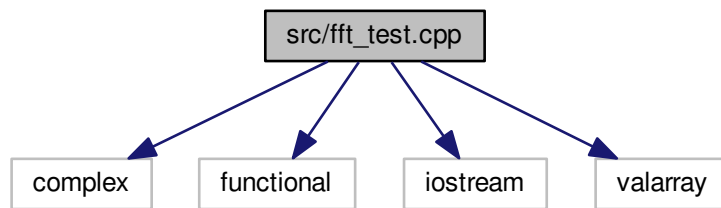
Tests FFT, IFFT, and Hilbert implementations.

```

#include <complex>
#include <functional>
#include <iostream>
#include <valarray>

```

Include dependency graph for fft_test.cpp:



Typedefs

- typedef std::valarray
< std::complex< double > > [CArray](#)

Functions

- void [fft](#) ([CArray](#) &x)
- void [ifft](#) ([CArray](#) &x)
- std::complex< double > [hilbert](#) (std::complex< double > n)
- int [main](#) ()

Variables

- const double [PI](#) = 3.141592653589793238460

8.29.1 Detailed Description

Tests FFT, IFFT, and Hilbert implementations.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [fft_test.cpp](#).

8.29.2 Typedef Documentation

8.29.2.1 typedef std::valarray<std::complex<double> > [CArray](#)

Definition at line 14 of file [fft_test.cpp](#).

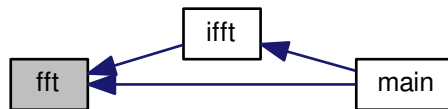
8.29.3 Function Documentation

8.29.3.1 void [fft](#) ([CArray](#) & x)

This code was taken from http://rosettacode.org/wiki/Fast_Fourier_transform#C.2B.2B.

Definition at line 23 of file [fft_test.cpp](#).

Here is the caller graph for this function:



8.29.3.2 `std::complex<double> hilbert (std::complex< double > n)`

Definition at line 87 of file [fft_test.cpp](#).

Here is the caller graph for this function:



8.29.3.3 `void ifft (CArray & x)`

Definition at line 72 of file [fft_test.cpp](#).

Here is the call graph for this function:



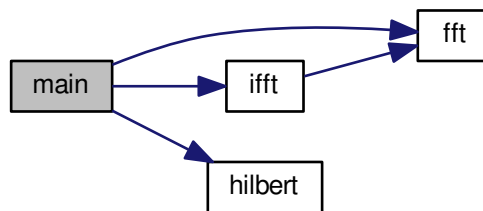
Here is the caller graph for this function:



8.29.3.4 int main ()

Definition at line 91 of file [fft_test.cpp](#).

Here is the call graph for this function:



8.29.4 Variable Documentation

8.29.4.1 const double PI = 3.141592653589793238460

Definition at line 12 of file [fft_test.cpp](#).

8.30 fft_test.cpp

```

00001
00007 #include <complex>
00008 #include <functional>
00009 #include <iostream>
00010 #include <valarray>
00011
00012 const double PI = 3.141592653589793238460;
00013
00014 typedef std::valarray<std::complex<double>> CArray;
00015
00021 // Cooley-Tukey FFT (in-place, breadth-first, decimation-in-frequency)
00022 // Better optimized but less intuitive
00023 void fft(CArray &x)
00024 {
00025     // DFT
00026     unsigned int N = x.size(), k = N, n;
00027     double thetaT = 3.14159265358979323846264338328L / N;
00028     std::complex<double> phiT(cos(thetaT), sin(thetaT)), T;
00029     while (k > 1)
  
```

```

00030     {
00031         n = k;
00032         k >>= 1;
00033         phiT = phiT * phiT;
00034         T = 1.0L;
00035         for (unsigned int l = 0; l < k; l++)
00036         {
00037             for (unsigned int a = l; a < N; a += n)
00038             {
00039                 unsigned int b = a + k;
00040                 std::complex<double> t = x[a] - x[b];
00041                 x[a] += x[b];
00042                 x[b] = t * T;
00043             }
00044             T *= phiT;
00045         }
00046     }
00047     // Decimate
00048     unsigned int m = (unsigned int)log2(N);
00049     for (unsigned int a = 0; a < N; a++)
00050     {
00051         unsigned int b = a;
00052         // Reverse bits
00053         b = ((b & 0xaaaaaaaa) >> 1) | ((b & 0x55555555) << 1);
00054         b = ((b & 0xcccccccc) >> 2) | ((b & 0x33333333) << 2);
00055         b = ((b & 0xf0f0f0f0) >> 4) | ((b & 0x0f0f0f0f) << 4);
00056         b = ((b & 0xff00ff00) >> 8) | ((b & 0x00ff00ff) << 8);
00057         b = ((b >> 16) | (b << 16)) >> (32 - m);
00058         if (b > a)
00059         {
00060             std::complex<double> t = x[a];
00061             x[a] = x[b];
00062             x[b] = t;
00063         }
00064     }
00065     //std::complex<double> f = 1.0 / sqrt(N);
00066     //for (unsigned int i = 0; i < N; i++)
00067     //    x[i] *= f;
00068 }
00069 // inverse fft (in-place)
00070 void ifft(CArray& x)
00071 {
00072     // conjugate the complex numbers
00073     x = x.apply(std::conj);
00074     // forward fft
00075     fft(x);
00076     // conjugate the complex numbers again
00077     x = x.apply(std::conj);
00078     // scale the numbers
00079     x /= x.size();
00080 }
00081 std::complex<double> hilbert(std::complex<double> n) {
00082     return std::complex<double>(-2 * n.imag(), 0);
00083 }
00084 int main()
00085 {
00086     const std::complex<double> test[] = { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 };
00087     CArray data(test, 16);
00088     // forward fft
00089     fft(data);
00090     std::cout << "fft" << std::endl;
00091     for (int i = 0; i < 16; ++i)
00092     {
00093         // std::cout << data[i] << std::endl;
00094     }
00095     for(int i = 8; i < 16; i++) {
00096         data[i] = 0;
00097     }
00098     // inverse fft
00099     ifft(data);
00100     std::cout << std::endl << "ifft" << std::endl;
00101     for (int i = 0; i < 16; ++i)
00102     {
00103         // std::cout << data[i] << std::endl;
00104     }
00105 }
00106
00107
00108
00109
00110
00111
00112
00113
00114
00115
00116
00117

```

```

00118     data = data.apply(hilbert);
00119
00120     std::cout << std::endl;
00121
00122     for(int i = 0; i < 16; i++) {
00123         std::cout << data[i].real() << std::endl;
00124     }
00125
00126     return 0;
00127 }

```

8.31 src/fft_test2.cpp File Reference

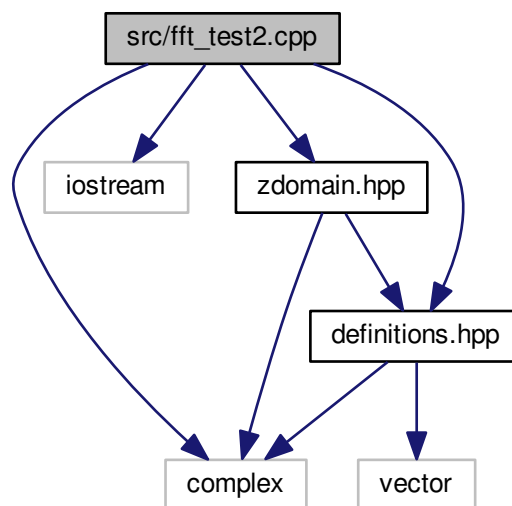
Tests FFT, IFFT, and Hilbert implementations in [zdomain.hpp](#).

```

#include <complex>
#include <iostream>
#include "definitions.hpp"
#include "zdomain.hpp"

```

Include dependency graph for `fft_test2.cpp`:



Functions

- `int main ()`

8.31.1 Detailed Description

Tests FFT, IFFT, and Hilbert implementations in [zdomain.hpp](#).

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [fft_test2.cpp](#).

8.31.2 Function Documentation

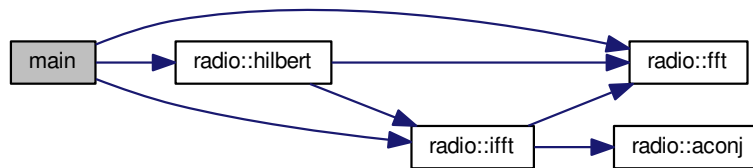
8.31.2.1 int main ()

This program tests the `fft()`, `ifft()`, and `hilbert()` functions in the `zdomain.hpp` file.

This code is based on code from http://rosettacode.org/wiki/Fast_Fourier_transform#C.↔2B.2B.

Definition at line 22 of file `fft_test2.cpp`.

Here is the call graph for this function:



8.32 fft_test2.cpp

```

00001
00007 #include <complex>
00008 #include <iostream>
00009
00010 #include "definitions.hpp"
00011 #include "zdomain.hpp"
00012
00013 using namespace radio;
00014
00022 int main()
00023 {
00024     std::complex<float32> test[] = { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 };
00025     float32 ftest[16];
00026     float32 dest[16];
00027
00028     for(int i = 0; i < 16; i++) {
00029         ftest[i] = test[i].real();
00030     }
00031
00032     // forward fft
00033     fft(test, 16);
00034
00035     std::cout << "fft" << std::endl;
00036
00037     for (int i = 0; i < 16; ++i)
00038     {
00039         // std::cout << test[i] << std::endl;
00040     }
00041
00042     // inverse fft
00043     ifft(test, 16);
00044     std::cout << std::endl << "ifft" << std::endl;
00045
00046     for (int i = 0; i < 16; ++i)
00047     {
00048         std::cout << test[i] << std::endl;
00049     }
00050
00051     hilbert(ftest, dest, 16);
00052     std::cout << std::endl << "hilbert" << std::endl;
00053
00054     for(int i = 0; i < 16; i++) {
00055         std::cout << dest[i] << std::endl;
00056     }
00057
00058     return 0;

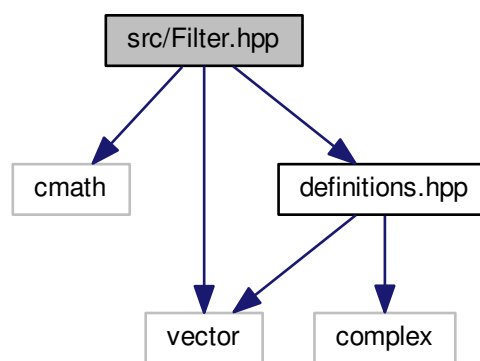
```

```
00059 }
```

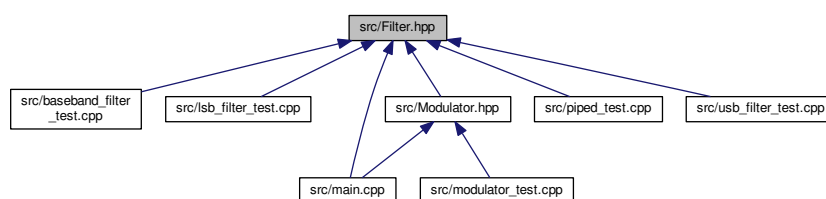
8.33 src/Filter.hpp File Reference

Defines the Filter class.

```
#include <cmath>
#include <vector>
#include "definitions.hpp"
Include dependency graph for Filter.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [radio::Filter](#)

Namespaces

- [radio](#)

Contains the classes for the various types of modulation supported by the program.

8.33.1 Detailed Description

Defines the Filter class.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Bug discontinuities created at the beginning of each pass

Definition in file [Filter.hpp](#).

8.34 Filter.hpp

```

00001
00008 #ifndef Filter_H
00009 #define Filter_H
00010
00011 #include <cmath>
00012 #include <vector>
00013
00014 #include "definitions.hpp"
00015
00016 namespace radio {
00028     class Filter {
00029     public:
00044         Filter(float32* data, uint32 size,
00045                fparams& diffEq);
00051
00052         void Pass();
00053
00054     protected:
00058         uint8 eqLength;
00059
00063         uint32 size;
00064
00069         float32* data;
00070
00077         fparams diffEq;
00078     };
00079
00080     Filter::Filter(float32* data, uint32 size,
00081                   fparams& diffEq) {
00082         this->data = data;
00083         this->size = size;
00084         this->diffEq = diffEq;
00085         eqLength = this->diffEq[DEN].size();
00086     }
00087
00088     void Filter::Pass() {
00089         float64 temp[size];
00090
00091         // create first values in filtered data
00092         for(int i = 0; i < eqLength; i++) {
00093             temp[i] = 0;
00094
00095             for(int j = 0; j < eqLength; j++) {
00096                 temp[i] += diffEq[NUM][j] * (j > i ? 0 : data[i - j]);
00097             }
00098
00099             for(int j = 1; j < eqLength; j++) {
00100                 temp[i] -= diffEq[DEN][j] * (j > i ? 0 : temp[i - j]);
00101             }
00102
00103         // create the REST of the values in filtered data
00104         for(int i = eqLength; i < size; i++) {
00105             temp[i] = 0;
00106
00107             for(int j = 0; j < eqLength; j++) {
00108                 temp[i] += diffEq[NUM][j] * data[i - j];
00109             }
00110
00111             for(int j = 1; j < eqLength; j++) {
00112                 temp[i] -= diffEq[DEN][j] * temp[i - j];
00113             }
00114         }
00115     }

```

```

00116         // save final values of data and filtered data
00117         for(int i = 0; i < size; i++) {
00118             data[i] = temp[i];
00119         }
00120     }
00121 }
00122
00123 #endif

```

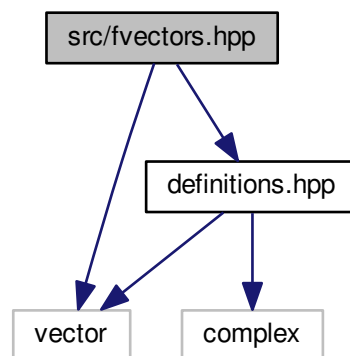
8.35 src/fvectors.hpp File Reference

Defines the transfer function coefficients used in the instances of the Filter class in this program.

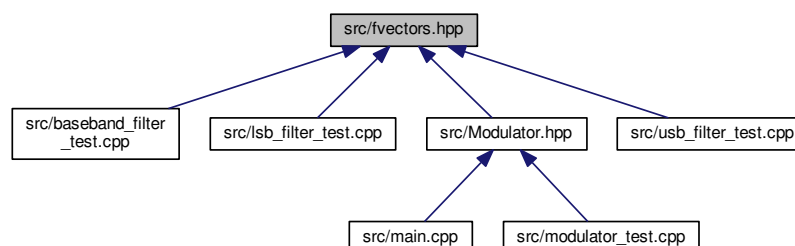
```
#include <vector>
```

```
#include "definitions.hpp"
```

Include dependency graph for fvectors.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- [radio](#)

Contains the classes for the various types of modulation supported by the program.

Variables

- [fparams radio::F_BASEBAND](#)
- [fparams radio::F_LOWERSIDEBAND](#)
- [fparams radio::F_UPPERSIDEBAND](#)

8.35.1 Detailed Description

Defines the transfer function coefficients used in the instances of the Filter class in this program.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [fvectors.hpp](#).

8.36 fvectors.hpp

```

00001
00008 #ifndef fvectors_H
00009 #define fvectors_H
00010
00011 #include <vector>
00012
00013 #include "definitions.hpp"
00014
00015 namespace radio {
00019     fparams F_BASEBAND = { std::vector<float64> {
00020         0.0008977019461,
00021         -0.002215694636,
00022         0.001372192986,
00023         0.001372192986,
00024         -0.002215694636,
00025         0.0008977019461
00026     }, std::vector<float64> {
00027         1,
00028         -4.678616047,
00029         8.822912216,
00030         -8.379911423,
00031         4.007629871,
00032         -0.7719064355
00033     } };
00034
00038     fparams F_LOWERSIDEBAND = { std::vector<float64> {
00039         0.2758039069174,
00040         2.763578787693,
00041         12.83915022756,
00042         36.47584850651,
00043         70.37084637368,
00044         96.76893503179,
00045         96.76893503179,
00046         70.37084637368,
00047         36.47584850651,
00048         12.83915022756,
00049         2.763578787693,
00050         0.2758039069174
00051     }, std::vector<float64> {
00052         1,
00053         7.605497780083,
00054         27.34180552438,
00055         60.83375457605,
00056         92.60908886875,
00057         100.8363857,
00058         79.74796574736,
00059         45.4982252145,
00060         18.13566776308,
00061         4.690036472717,
00062         0.6617552879305,
00063         0.0281427334611
00064     } };
00065
00069     fparams F_UPPERSIDEBAND = { std::vector<float64> {
00070         0.001690387681463,
00071         0.01145271586989,
00072         0.03591799189724,

```



```

00073         0.06576926098562,
00074         0.07119343282702,
00075         0.03156377419766,
00076         -0.03156377419766,
00077         -0.07119343282702,
00078         -0.06576926098562,
00079         -0.03591799189724,
00080         -0.01145271586989,
00081         -0.001690387681463
00082     }, std::vector<float64> {
00083         1,
00084         9.465175013624,
00085         41.62402815905,
00086         112.0971027069,
00087         205.2097686473,
00088         267.9378582311,
00089         254.486805213,
00090         175.7772755115,
00091         86.51619894548,
00092         28.89988093561,
00093         5.89781461091,
00094         0.5572910543053
00095     } };
00096
00097
00098 }
00099
00100 #endif

```

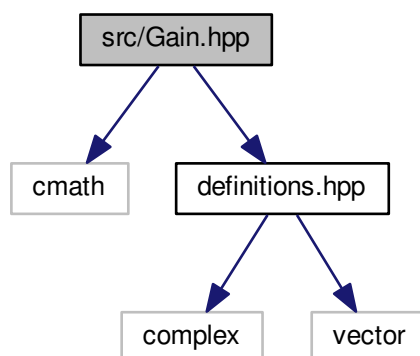
8.37 src/Gain.hpp File Reference

Contains the Gain class.

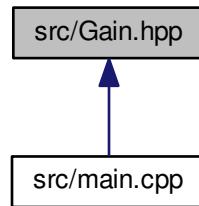
```
#include <cmath>
```

```
#include "definitions.hpp"
```

Include dependency graph for Gain.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [radio::Gain](#)

Namespaces

- [radio](#)

Contains the classes for the various types of modulation supported by the program.

8.37.1 Detailed Description

Contains the Gain class.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [Gain.hpp](#).

8.38 Gain.hpp

```

00001
00007 #ifndef Gain_H
00008 #define Gain_H
00009
00010 #include <cmath>
00011
00012 #include "definitions.hpp"
00013
00014 namespace radio {
00018     class Gain {
00019     public:
00030         Gain(float32* data, uint32 size, float32 gaindB);
00031
00035         void Apply();
00036
00037     private:
00041         float32* data;
00042
00046         float32 gainCoeff;
00047
00052         bool hasClipped = false;
00053
00057         uint32 size;
00058

```

```

00059     };
00060
00061     Gain::Gain(float32* data, uint32 size, float32 gaindB) {
00062         this->data = data;
00063         this->size = size;
00064         gainCoeff = pow(10, gaindB / 20);
00065     }
00066
00067     void Gain::Apply() {
00068         for(uint32 i = 0; i < size; i++) {
00069             data[i] *= gainCoeff;
00070
00071             if((data[i] > 1 || data[i] < -1) && !hasClipped) {
00072                 hasClipped = true;
00073                 std::cerr << "Baseband clipping has occurred!"
00074                     << std::endl;
00075             }
00076         }
00077     }
00078 }
00079
00080 #endif

```

8.39 src/iq_test.cpp File Reference

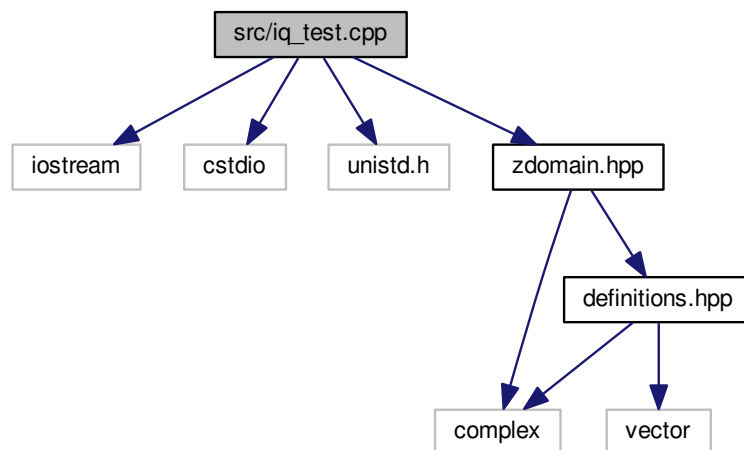
Generates test IQ signal.

```

#include <iostream>
#include <stdio>
#include <unistd.h>
#include "zdomain.hpp"

```

Include dependency graph for iq_test.cpp:



Functions

- int `main` ()

8.39.1 Detailed Description

Generates test IQ signal.

Author

Samuel Andrew Wisner, awisner94@gmail.com

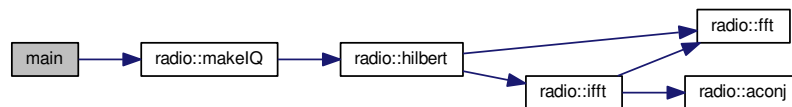
Definition in file [iq_test.cpp](#).

8.39.2 Function Documentation**8.39.2.1 int main ()**

This small program demonstrates the IQ generation abilities of the [makeIQ\(\)](#) function.

Definition at line 20 of file [iq_test.cpp](#).

Here is the call graph for this function:

**8.40 iq_test.cpp**

```

00001
00007 #include <iostream>
00008 #include <cstdio>
00009 #include <unistd.h>
00010
00011 #include "zdomain.hpp"
00012
00013 using namespace std;
00014 using namespace radio;
00015
00020 int main() {
00021     const uint16 len = 16384;
00022     float32 data[len];
00023     float32 iqData[2*len];
00024
00025     for(int i = 0; i < len; i++) {
00026         data[i] = sin(2*3.141592*170*i/len);
00027     }
00028
00029     while(true) {
00030         read(STDIN_FILENO, &data, len * sizeof(float32));
00031         makeIQ(data, iqData, len);
00032         write(STDOUT_FILENO, &iqData, 2 * len * sizeof(float32));
00033     }
00034 }

```

8.41 src/lb_filter_test.cpp File Reference

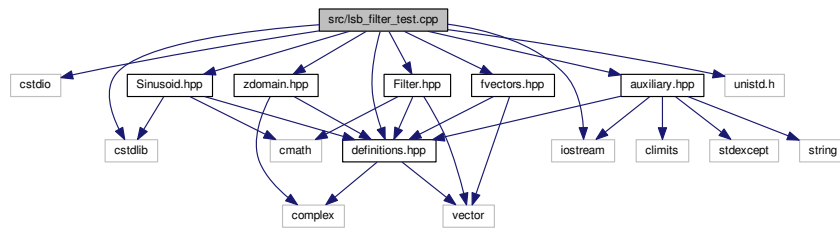
```
#include <cstdio>
```

```

#include <cstdlib>
#include <iostream>
#include <unistd.h>
#include "auxiliary.hpp"
#include "definitions.hpp"
#include "Filter.hpp"
#include "fvectors.hpp"
#include "Sinusoid.hpp"
#include "zdomain.hpp"

```

Include dependency graph for lsb_filter_test.cpp:



Functions

- int [main](#) (int argc, char *argv[])

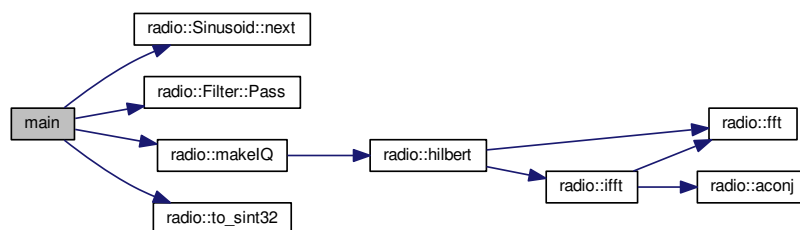
8.41.1 Function Documentation

8.41.1.1 int main (int argc, char * argv[])

Program to test the Filter class and the LSB filter coefficients.

Definition at line 25 of file [lsb_filter_test.cpp](#).

Here is the call graph for this function:



8.42 lsb_filter_test.cpp

```

00001
00007 #include <cstdio>
00008 #include <cstdlib>
00009 #include <iostream>
00010 #include <unistd.h>
00011

```

```

00012 #include "auxiliary.hpp"
00013 #include "definitions.hpp"
00014 #include "Filter.hpp"
00015 #include "fvectors.hpp"
00016 #include "Sinusoid.hpp"
00017 #include "zdomain.hpp"
00018
00019 using namespace std;
00020 using namespace radio;
00021
00025 int main(int argc, char* argv[]) {
00026
00027     // Constants
00028     const uint16 BUFFER_SIZE = 48000;
00029
00030     // Declare primitive Variables
00031     uint8 i = 0;
00032     uint8 size = 0;
00033     uint16 delta = 250;
00034     float32 dataBuffer[BUFFER_SIZE];
00035     float32 iqBuffer[2 * BUFFER_SIZE];
00036
00037     // create 1 sec of audio
00038     for(uint16 f = 17000; f <= 23000; f += delta, i++) {
00039         Sinusoid sinusoid(f);
00040
00041         for(uint16 i = 0; i < BUFFER_SIZE; i++) {
00042             dataBuffer[i] += sinusoid.next();
00043         }
00044     }
00045
00046     size = i;
00047
00048     // adjust dataBuffer so values are between -1 and 1
00049     for(uint16 i = 0; i < BUFFER_SIZE; i++) {
00050         dataBuffer[i] /= size;
00051     }
00052
00053     Filter filter(dataBuffer, BUFFER_SIZE, F_LOWERSIDEBAND);
00054     filter.Pass();
00055     makeIQ(dataBuffer, iqBuffer, BUFFER_SIZE);
00056     to_sint32(iqBuffer, 2 * BUFFER_SIZE);
00057
00058     while(true) {
00059         write(STDOUT_FILENO, &iqBuffer, 2 * BUFFER_SIZE * sizeof(sint32));
00060     }
00061 }

```

8.43 src/main.cpp File Reference

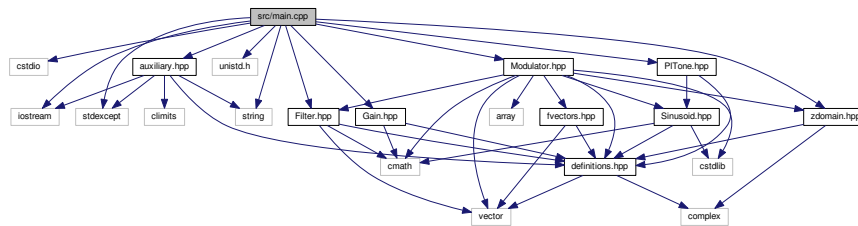
contains the "brains" of the entire project

```

#include <cstdio>
#include <iostream>
#include <stdexcept>
#include <string>
#include <unistd.h>
#include "auxiliary.hpp"
#include "Filter.hpp"
#include "Gain.hpp"
#include "Modulator.hpp"
#include "PlTone.hpp"
#include "zdomain.hpp"

```

Include dependency graph for main.cpp:



Functions

- `int main (int argc, char *argv[])`

8.43.1 Detailed Description

contains the "brains" of the entire project

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [main.cpp](#).

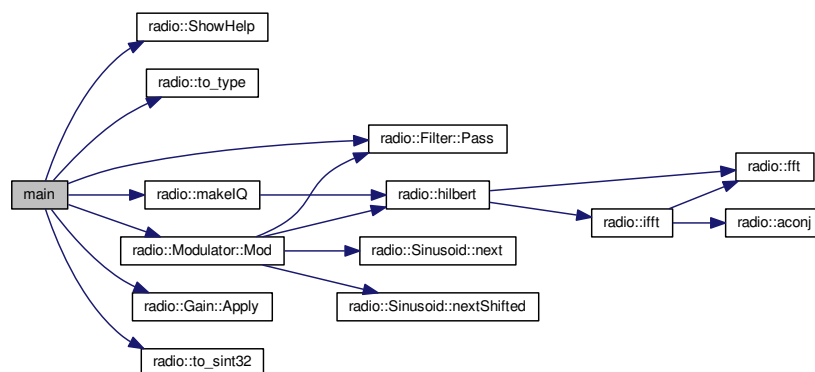
8.43.2 Function Documentation

8.43.2.1 `int main (int argc, char * argv[])`

Final result of the entire project. Completes all goals and more!

Definition at line 26 of file [main.cpp](#).

Here is the call graph for this function:



8.44 main.cpp

```

00001
00007 #include <cstdio>
00008 #include <iostream>
00009 #include <stdexcept>
00010 #include <string>
00011 #include <unistd.h>
00012
00013 #include "auxiliary.hpp"
00014 #include "Filter.hpp"
00015 #include "Gain.hpp"
00016 #include "Modulator.hpp"
00017 #include "PlTone.hpp"
00018 #include "zdomain.hpp"
00019
00020 using namespace std;
00021 using namespace radio;
00022
00026 int main(int argc, char* argv[]) {
00027
00028     // Constants
00029     const uint8 NUM_TYPES = 8;
00030     const uint16 BUFFER_SIZE = 16384;
00031     const uint32 BUFFER_BYTE_COUNT = BUFFER_SIZE * sizeof(sint32);
00032     const uint32 IQ_BUFFER_SIZE = 2 * BUFFER_SIZE;
00033     const uint32 IQ_BUFFER_BYTE_COUNT = BUFFER_BYTE_COUNT * 2;
00034     const uint32 SAMPLING_RATE = 48000;
00035
00036     // Ensure 1 or 2 arguments given
00037     if(argc > 4) {
00038         std::cerr << "Error: too many arguments!" << std::endl;
00039         ShowHelp();
00040         return ERROR;
00041     } else if(argc < 2) {
00042         std::cerr << "Error: too few arguments!" << std::endl;
00043         ShowHelp();
00044         return ERROR;
00045     }
00046
00047     // Declare primitive Variables
00048     float32 micGain = 0;
00049     float32 toneFreq = 0;
00050     float32 dataBuffer[BUFFER_SIZE];
00051     float32 iqBuffer[IQ_BUFFER_SIZE];
00052     ModulationType type;
00053
00054     // validate modulation type
00055     try{
00056         type = to_type(string(argv[1]));
00057     } catch(std::exception ex) {
00058         std::cerr << "The given modulation type is invalid!" << std::endl;
00059         ShowHelp();
00060     }
00061
00062     // process mic gain
00063     if(argc >= 3) {
00064         try {
00065             micGain = std::stof(argv[2]);
00066         } catch(std::invalid_argument ex) {
00067             std::cerr << "The specified microphone gain is not a number."
00068                 << std::endl;
00069             ShowHelp();
00070         }
00071     }
00072
00073     // validate CTCSS tone
00074     if(argc == 4) {
00075         try {
00076             toneFreq = std::stof(argv[3]);
00077
00078             if(toneFreq < 60 || toneFreq > 260) {
00079                 throw std::out_of_range("");
00080             }
00081         } catch(std::out_of_range ex) {
00082             std::cerr << "The specified CTCSS frequency is outside of the "
00083                 << "standard PL tone range." << std::endl;
00084             ShowHelp();
00085         } catch(std::invalid_argument ex) {
00086             std::cerr << "The specified CTCSS frequency is not a number."
00087                 << std::endl;
00088             ShowHelp();
00089         }
00090     }
00091
00092     // Declare objects

```



```

00093     Filter baseFilter(dataBuffer, BUFFER_SIZE, F_BASEBAND);
00094     Gain gain(dataBuffer, BUFFER_SIZE, micGain);
00095     PlTone pltone(0.15, dataBuffer, BUFFER_SIZE, toneFreq, SAMPLING_RATE);
00096     Modulator modulator(dataBuffer, BUFFER_SIZE, type, 20000);
00097
00098     // SDR guts of the program
00099     while(true) {
00100         // get next samples
00101         read(STDIN_FILENO, &dataBuffer, BUFFER_BYTE_COUNT);
00102
00103         // process/modulate samples
00104         baseFilter.Pass();
00105         // pltone.Add();
00106         gain.Apply();
00107         modulator.Mod();
00108         makeIQ(dataBuffer, iqBuffer, BUFFER_SIZE);
00109         to_sint32(iqBuffer, IQ_BUFFER_SIZE);
00110
00111         // write samples
00112         write(STDOUT_FILENO, &iqBuffer, IQ_BUFFER_BYTE_COUNT);
00113     }
00114 }

```

8.45 src/mic_test.cpp File Reference

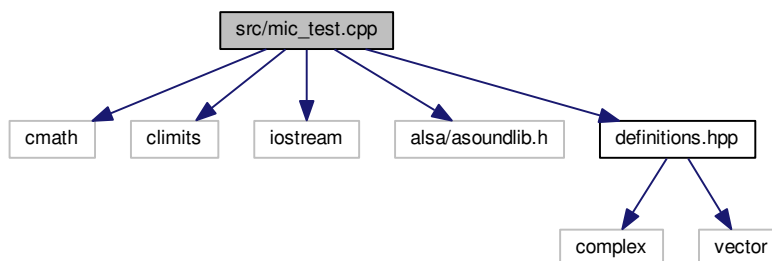
Tests getting mic input via ALSA May not even compile at the moment.

```

#include <cmath>
#include <climits>
#include <iostream>
#include <alsa/asoundlib.h>
#include "definitions.hpp"

```

Include dependency graph for mic_test.cpp:



Functions

- int [main](#) ()

8.45.1 Detailed Description

Tests getting mic input via ALSA May not even compile at the moment.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [mic_test.cpp](#).

8.45.2 Function Documentation

8.45.2.1 int main ()

This program tests taking information from the microphone via the ALSA API. Not sure if it works.

Definition at line 21 of file [mic_test.cpp](#).

8.46 mic_test.cpp

```

00001
00008 #include <cmath>
00009 #include <climits>
00010 #include <iostream>
00011 #include <alsa/asoundlib.h>
00012
00013 #include "definitions.hpp"
00014
00015 using namespace std;
00016
00021 int main() {
00022     int ret;
00023
00024     snd_pcm_t* pcm_handle; // device handle
00025     // snd_pcm_stream_t stream = SND_PCM_STREAM_PLAYBACK;
00026     snd_pcm_stream_t stream = SND_PCM_STREAM_CAPTURE;
00027     snd_pcm_hw_params_t* hwparams; // hardware information
00028     char* pcm_name = strdup("plughw:1,0"); // on-board audio jack
00029     //char* pcm_name = strdup("plughw:0,0"); // on-board audio jack
00030     int rate = 48000;
00031
00032     const uint16 freq = 440;
00033     long unsigned int bufferSize = 8192*4;
00034     const uint32 len = bufferSize*100;
00035     const float32 arg = 2 * 3.141592 * freq / rate;
00036     sint16 vals[len];
00037
00038     float test;
00039     float last = 0;
00040     long unsigned int count = 0;
00041     int count2 = 0;
00042
00043     for(int i = 0; i < len; i = i + 2) {
00044         bool lastWas = abs(sin(last)) < 0.01;
00045
00046         last += arg;
00047         if(last > 2 * M_PI) last -= 2 * M_PI;
00048
00049         test = 32000 * sin(last);
00050
00051         if(abs(sin(last)) < 0.01 && lastWas) count++;
00052
00053         vals[i] = (sint16)(test + 0.5);
00054         vals[i+1] = vals[i];
00055     }
00056
00057     cout << "COUNT: " << count << endl;
00058     snd_pcm_hw_params_alloca(&hwparams);
00059
00060     ret = snd_pcm_open(&pcm_handle, pcm_name, stream, 0);
00061     cout << "Opening: " << snd_strerror(ret) << endl;
00062
00063     ret = snd_pcm_hw_params_any(pcm_handle, hwparams);
00064     cout << "Initializing hwparams structure: " << snd_strerror(ret) << endl;
00065
00066     ret = snd_pcm_hw_params_set_access(pcm_handle, hwparams,
00067         SND_PCM_ACCESS_RW_INTERLEAVED);
00068     cout << "Setting access: " << snd_strerror(ret) << endl;
00069
00070     ret = snd_pcm_hw_params_set_format(pcm_handle, hwparams,
00071         SND_PCM_FORMAT_S16_LE);
00072     cout << "Setting format: " << snd_strerror(ret) << endl;
00073
00074     ret = snd_pcm_hw_params_set_rate(pcm_handle, hwparams,
00075         rate, (int)0);
00076     cout << "Setting rate: " << snd_strerror(ret) << endl;
00077
00078     ret = snd_pcm_hw_params_set_channels(pcm_handle, hwparams, 2);
00079     cout << "Setting channels: " << snd_strerror(ret) << endl;
00080
00081     ret = snd_pcm_hw_params_set_periods(pcm_handle, hwparams, 2, 0);

```

```

00082     cout << "Setting periods: " << snd_strerror(ret) << endl;
00083
00084     ret = snd_pcm_hw_params_set_buffer_size_near(pcm_handle, hwparams,
00085         &bufferSize);
00086     cout << "Setting buffer size: " << snd_strerror(ret) << endl;
00087
00088     ret = snd_pcm_hw_params(pcm_handle, hwparams);
00089     cout << "Applying parameters: " << snd_strerror(ret) << endl;
00090
00091     /* ret = snd_pcm_hw_params_get_period_size(hwparams, &count, &count2);
00092     cout << "Actual period size: " << count << endl;
00093     cout << "Returned: " << snd_strerror(ret) << endl;*/
00094
00095
00096
00097     cout << endl << endl;
00098
00099
00100     //const void* ptr = (const void*)&vals;
00101     void* ptr = (void*)&vals;
00102     int err;
00103
00104     for(int i = 0; i < 100; i++) {
00105         do {
00106             ret = snd_pcm_readi(pcm_handle,
00107                 ptr, bufferSize);
00108
00109             if(ret < 0) {
00110                 err = snd_pcm_prepare(pcm_handle);
00111                 cout << "Preparing: " << snd_strerror(err)
00112                     << endl;
00113             }
00114         } while(ret < 0);
00115
00116         cout << "Writing data: " << ret << endl;
00117     }
00118 }

```

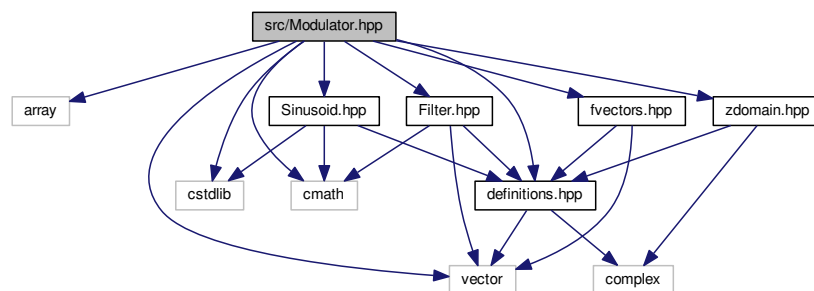
8.47 src/Modulator.hpp File Reference

```

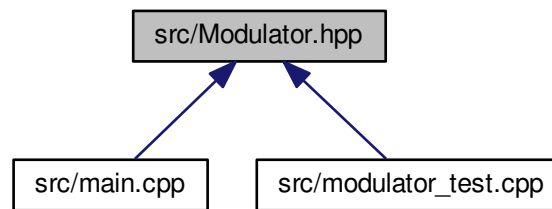
#include <array>
#include <cmath>
#include <cstdlib>
#include <vector>
#include "definitions.hpp"
#include "Filter.hpp"
#include "fvectors.hpp"
#include "Sinusoid.hpp"
#include "zdomain.hpp"

```

Include dependency graph for Modulator.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class `radio::Modulator`

Namespaces

- `radio`

Contains the classes for the various types of modulation supported by the program.

Variables

- `const uint32 radio::FREQ_INTERMEDIATE = 20000`
- `const uint32 radio::SAMPLING_RATE = 48000`

8.48 Modulator.hpp

```

00001
00009 #ifndef modulation_H
00010 #define modulation_H
00011
00012 #include <array>
00013 #include <cmath>
00014 #include <cstdlib>
00015 #include <vector>
00016
00017 #include "definitions.hpp"
00018 #include "Filter.hpp"
00019 #include "fvectors.hpp"
00020 #include "Sinusoid.hpp"
00021 #include "zdomain.hpp"
00022
00023 namespace radio {
00024
00028     const uint32 FREQ_INTERMEDIATE = 20000;
00029
00033     const uint32 SAMPLING_RATE = 48000;
00034
00039     class Modulator {
00040     public:
00055         Modulator(float32 data[], uint32 size,
ModulationType type,
00056                     float32 freqInter = FREQ_INTERMEDIATE,
00057                     uint32 rate = SAMPLING_RATE);
00058
00062         ~Modulator();
00063
00067         void Mod();
  
```

```

00068
00069     private:
00074         float32* data;
00075
00079         float32 freqCarrier;
00080
00081
00085         float32* hilData = nullptr;
00086
00090         float32 rate;
00091
00095         uint32 size;
00096
00100         ModulationType type;
00101     };
00102
00103     Modulator::Modulator(float32 data[], uint32 size,
00104                          ModulationType type,
00105                          float32 freqInter, uint32 rate) {
00106         freqCarrier = freqInter;
00107         this->rate = rate;
00108         this->data = data;
00109         this->size = size;
00110         this->type = type;
00111
00112         if(type == ModulationType::USB_HILBERT
00113            || type == ModulationType::LSB_HILBERT) {
00114             hilData = (float32*)malloc(size*sizeof(float32));
00115         }
00116
00117     Modulator::~Modulator() {
00118         if(hilData != nullptr) free(hilData);
00119     }
00120
00121     void Modulator::Mod() {
00122         // these variables should only ever be created once
00123         static float32 fmArg = 2 * M_PI * freqCarrier / (float32)rate;
00124         static float32 fmK = 2 * M_PI / rate;
00125         static float32 fmSum = 0; // cumulative sum used in FM modulation
00126         static Filter lsbFilter(data, size, F_LOWERSIDEBAND);
00127         static Sinusoid sinusoid(freqCarrier, rate); // IF carrier sinusoid
00128         static Filter usbFilter(data, size, F_UPPERSIDEBAND);
00129
00130         // take hilbert transform if necessary
00131         if(type == ModulationType::USB_HILBERT
00132            || type == ModulationType::LSB_HILBERT) {
00133             hilbert(data, hilData, size);
00134         } else if(type == ModulationType::FM_NARROW) {
00135             fmK *= 2.5;
00136         } else if(type == ModulationType::FM_WIDE) {
00137             fmK *= 5;
00138         }
00139
00140         // perform main modulation
00141         for(uint32 i = 0; i < size; i++) {
00142             switch(type) {
00143                 case ModulationType::DSB_LC:
00144                     data[i] = ((data[i] + 1) * sinusoid.next()) / 2;
00145                     break;
00146
00147                 case ModulationType::DSB_SC:
00148                 case ModulationType::USB_FILTERED:
00149                 case ModulationType::LSB_FILTERED:
00150                     data[i] = data[i] * sinusoid.next();
00151                     break;
00152
00153                 case ModulationType::USB_HILBERT:
00154                     data[i] = data[i] * sinusoid.next()
00155                         - hilData[i] * sinusoid.nextShifted();
00156                     break;
00157
00158                 case ModulationType::LSB_HILBERT:
00159                     data[i] = data[i] * sinusoid.next()
00160                         + hilData[i] * sinusoid.nextShifted();
00161                     break;
00162
00163                 case ModulationType::FM_NARROW:
00164                 case ModulationType::FM_WIDE:
00165                     fmSum += fmK * data[i];
00166                     data[i] = cos(fmArg * i + fmSum);
00167                     break;
00168             }
00169         }
00170
00171         // filter out a sideband if using filtered SSB modulation
00172         if(type == ModulationType::LSB_FILTERED) {

```

```

00173         lsbFilter.Pass();
00174     } else if (type == ModulationType::USB_FILTERED) {
00175         usbFilter.Pass();
00176     }
00177 }
00178 }
00179
00180 #endif

```

8.49 src/modulator_test.cpp File Reference

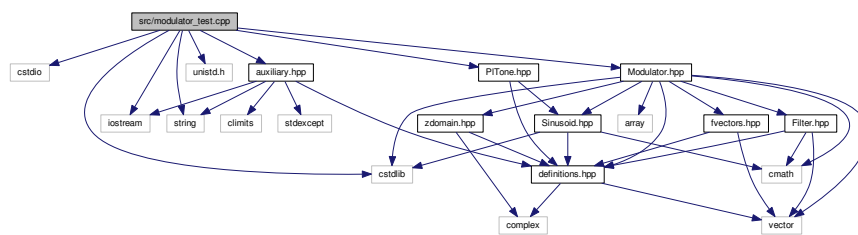
contains a test program to test the Modulator class

```

#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <string>
#include <unistd.h>
#include "auxiliary.hpp"
#include "Modulator.hpp"
#include "PlTone.hpp"

```

Include dependency graph for modulator_test.cpp:



Functions

- int [main](#) (int argc, char *argv[])

8.49.1 Detailed Description

contains a test program to test the Modulator class

Author

Samuel Andrew Wisner, awisner94@gmail.com

Bug filtered SSB clicking

Definition in file [modulator_test.cpp](#).

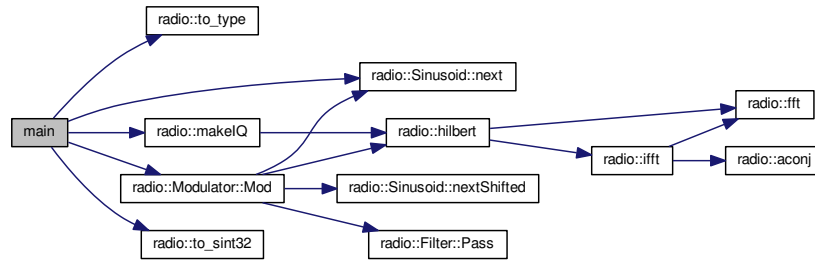
8.49.2 Function Documentation

8.49.2.1 int main (int argc, char * argv[])

Program to test the Modulator class with a self-generated sinusoidal input.

Definition at line 24 of file [modulator_test.cpp](#).

Here is the call graph for this function:



8.50 modulator_test.cpp

```

00001
00008 #include <cstdio>
00009 #include <cstdlib>
00010 #include <iostream>
00011 #include <string>
00012 #include <unistd.h>
00013
00014 #include "auxiliary.hpp"
00015 #include "Modulator.hpp"
00016 #include "PlTone.hpp"
00017
00018 using namespace std;
00019 using namespace radio;
00020
00024 int main(int argc, char* argv[]) {
00025
00026     // Constants
00027     const uint16 BUFFER_SIZE = 16384;
00028
00029     // Declare primitive Variables
00030     float32 dataBuffer[BUFFER_SIZE];
00031     float32 iqBuffer[2 * BUFFER_SIZE];
00032     ModulationType type;
00033     float32 freq = atof(argv[2]);
00034     float32 tone = 0;
00035
00036     if(argc >= 4) tone = atof(argv[3]);
00037
00038     try{
00039         type = to_type(string(argv[1]));
00040     } catch(std::exception ex) {
00041         std::cerr << ex.what() << std::endl << std::endl;
00042         return ERROR;
00043     }
00044
00045     if(freq < 0) {
00046         cerr << "The given tone was invalid." << endl;
00047         return ERROR;
00048     }
00049
00050     // Declare objects
00051     Modulator modulator(dataBuffer, BUFFER_SIZE, type, 20000);
00052     Sinusoid sinusoid(freq);
00053     PlTone(tone > 0 ? 0.15 : 0, dataBuffer, BUFFER_SIZE, tone, 48000);
00054
00055     while(true) {
00056         for(uint16 i = 0; i < BUFFER_SIZE; i++) {
00057             dataBuffer[i] = sinusoid.next();
00058         }
00059
00060         modulator.Mod();
00061         makeIQ(dataBuffer, iqBuffer, BUFFER_SIZE);
00062         to_sint32(iqBuffer, 2 * BUFFER_SIZE);
00063         write(STDOUT_FILENO, &iqBuffer, 2 * BUFFER_SIZE * sizeof(sint32));
00064     }
00065 }

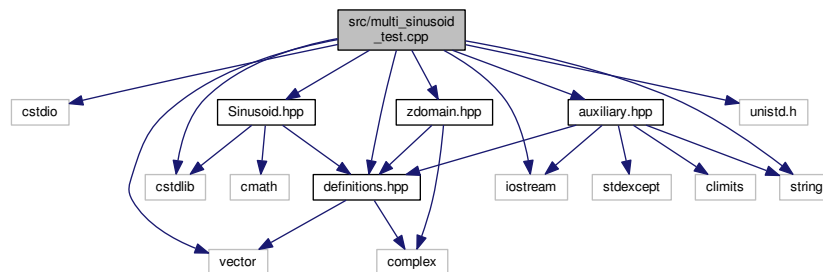
```

8.51 src/multi_sinusoid_test.cpp File Reference

contains a program to demonstrate the ability of the Sinusoid class and the sound card to generate sinusoids accross the spectrum.

```
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <string>
#include <unistd.h>
#include <vector>
#include "auxiliary.hpp"
#include "definitions.hpp"
#include "Sinusoid.hpp"
#include "zdomain.hpp"
```

Include dependency graph for multi_sinusoid_test.cpp:



Functions

- int [main](#) (int argc, char *argv[])

8.51.1 Detailed Description

contains a program to demonstrate the ability of the Sinusoid class and the sound card to generate sinusoids accross the spectrum.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [multi_sinusoid_test.cpp](#).

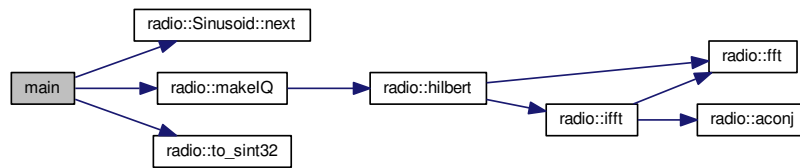
8.51.2 Function Documentation

8.51.2.1 int main (int argc, char * argv[])

Program to test the Sinusoid class and demonstrate the frequency range of the sound card.

Definition at line 27 of file [multi_sinusoid_test.cpp](#).

Here is the call graph for this function:



8.52 multi_sinusoid_test.cpp

```

00001
00008 #include <cstdio>
00009 #include <cstdlib>
00010 #include <iostream>
00011 #include <string>
00012 #include <unistd.h>
00013 #include <vector>
00014
00015 #include "auxiliary.hpp"
00016 #include "definitions.hpp"
00017 #include "Sinusoid.hpp"
00018 #include "zdomain.hpp"
00019
00020 using namespace std;
00021 using namespace radio;
00022
00027 int main(int argc, char* argv[]) {
00028
00029     // Constants
00030     const uint16 BUFFER_SIZE = 48000;
00031
00032     // Declare primitive Variables
00033     uint8 i = 0;
00034     uint8 size = 0;
00035     uint16 delta = 100;
00036     float32 dataBuffer[BUFFER_SIZE];
00037     float32 iqBuffer[2 * BUFFER_SIZE];
00038
00039     for(uint16 f = 100; f < 24000; f += delta, i++) {
00040         Sinusoid sinusoid(f);
00041
00042         for(uint16 i = 0; i < BUFFER_SIZE; i++) {
00043             dataBuffer[i] += sinusoid.next();
00044         }
00045
00046         switch(f) {
00047             case 500:
00048                 delta = 1000;
00049                 f = 1000;
00050                 break;
00051
00052             case 2000:
00053                 delta = 2000;
00054                 break;
00055         }
00056     }
00057
00058     size = i;
00059
00060     for(uint16 i = 0; i < BUFFER_SIZE; i++) {
00061         dataBuffer[i] /= size;
00062     }
00063
00064     makeIQ(dataBuffer, iqBuffer, BUFFER_SIZE);
00065     to_sint32(iqBuffer, 2 * BUFFER_SIZE);
00066
00067     while(true) {
00068         write(STDOUT_FILENO, &iqBuffer, 2 * BUFFER_SIZE * sizeof(sint32));
00069     }
00070 }

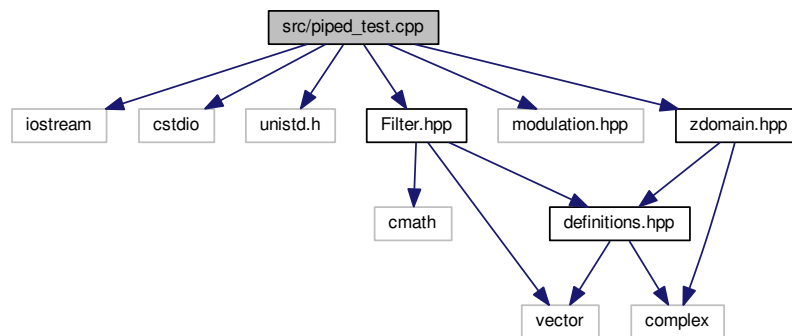
```

8.53 src/piped_test.cpp File Reference

contains the original program used to test the piping-in idea

```
#include <iostream>
#include <cstdio>
#include <unistd.h>
#include "Filter.hpp"
#include "modulation.hpp"
#include "zdomain.hpp"
```

Include dependency graph for `piped_test.cpp`:



Functions

- `int main ()`

8.53.1 Detailed Description

contains the original program used to test the piping-in idea

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [piped_test.cpp](#).

8.53.2 Function Documentation

8.53.2.1 `int main ()`

Program originally used to test whether baseband audio could be piped into the program in real time.

Definition at line 22 of file [piped_test.cpp](#).

Here is the call graph for this function:



8.54 piped_test.cpp

```

00001
00007 #include <iostream>
00008 #include <cstdio>
00009 #include <unistd.h>
00010
00011 #include "Filter.hpp"
00012 #include "modulation.hpp"
00013 #include "zdomain.hpp"
00014
00015 using namespace std;
00016 using namespace lolz;
00017
00022 int main() {
00023     const uint16 len = 16384;
00024     float32 data[len];
00025     float32 iqData[2*len];
00026
00027     while(true) {
00028         read(STDIN_FILENO, &data, len * sizeof(float32));
00029         makeIQ(data, iqData, len);
00030         write(STDOUT_FILENO, &iqData, 2 * len * sizeof(float32));
00031     }
00032 }
00033
  
```

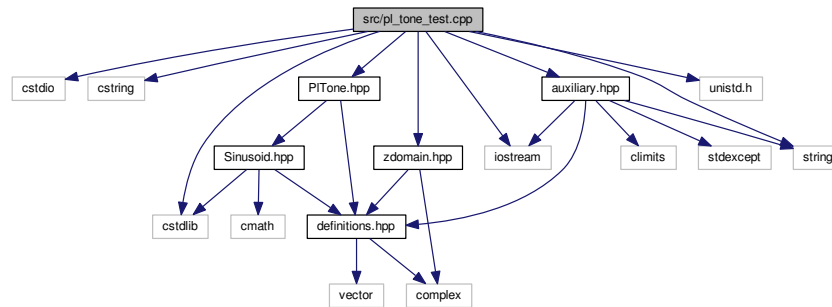
8.55 src/pl_tone_test.cpp File Reference

contains a test program to test the PITone class

```

#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <iostream>
#include <string>
#include <unistd.h>
#include "auxiliary.hpp"
#include "PlTone.hpp"
#include "zdomain.hpp"
  
```

Include dependency graph for `pl_tone_test.cpp`:



Functions

- `int main (int argc, char *argv[])`

8.55.1 Detailed Description

contains a test program to test the PITone class

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [pl_tone_test.cpp](#).

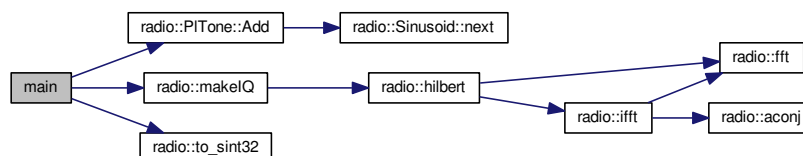
8.55.2 Function Documentation

8.55.2.1 `int main (int argc, char * argv[])`

Program to test the PITone class.

Definition at line 24 of file [pl_tone_test.cpp](#).

Here is the call graph for this function:



8.56 pl_tone_test.cpp

```

00001
00007 #include <cstdio>
00008 #include <cstring>

```

```

00009 #include <cstdlib>
00010 #include <iostream>
00011 #include <string>
00012 #include <unistd.h>
00013
00014 #include "auxiliary.hpp"
00015 #include "PlTone.hpp"
00016 #include "zdomain.hpp"
00017
00018 using namespace std;
00019 using namespace radio;
00020
00024 int main(int argc, char* argv[]) {
00025     // Constants
00026     const uint16 BUFFER_SIZE = 16384;
00027
00028     // Declare primitive Variables
00029     float32 dataBuffer[BUFFER_SIZE];
00030     float32 iqBuffer[2 * BUFFER_SIZE];
00031     float32 freq = atof(argv[1]);
00032
00033     if(freq < 0) {
00034         cerr << "The given tone was invalid." << endl;
00035         return ERROR;
00036     }
00037
00038     PlTone tone(0.15, dataBuffer, BUFFER_SIZE, freq, 48000);
00039
00040     while(true) {
00041         for(uint16 i = 0; i < BUFFER_SIZE; i++) {
00042             dataBuffer[i] = 1;
00043         }
00044
00045         tone.Add();
00046         makeIQ(dataBuffer, iqBuffer, BUFFER_SIZE);
00047         to_sint32(iqBuffer, 2 * BUFFER_SIZE);
00048         write(STDOUT_FILENO, &iqBuffer, 2 * BUFFER_SIZE * sizeof(sint32));
00049     }
00050 }

```

8.57 src/PITone.hpp File Reference

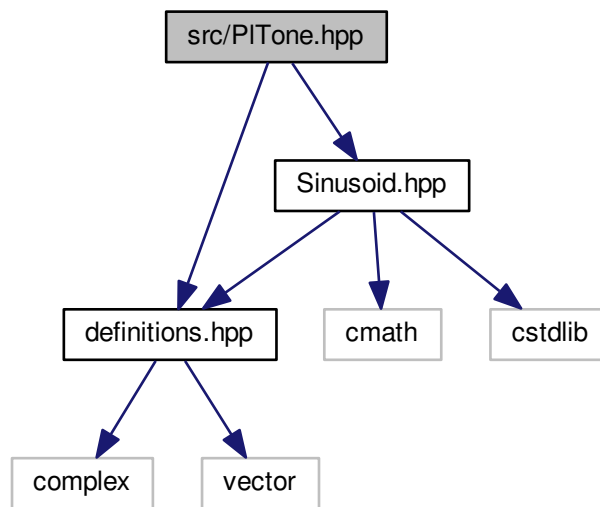
contains the PITone class

```

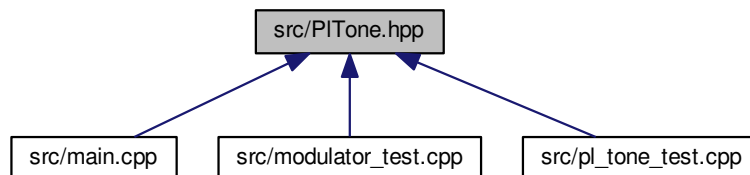
#include "definitions.hpp"
#include "Sinusoid.hpp"

```

Include dependency graph for PITone.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [radio::PITone](#)

Namespaces

- [radio](#)

Contains the classes for the various types of modulation supported by the program.

8.57.1 Detailed Description

contains the PITone class

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [PITone.hpp](#).

8.58 PITone.hpp

```

00001
00007 #ifndef PITone_H
00008 #define PITone_H
00009
00010 #include "definitions.hpp"
00011 #include "Sinusoid.hpp"
00012
00013 namespace radio {
00018     class PITone : Sinusoid {
00019     public:
00037         PITone(float32 amplitude, float32* data, uint32 size,
00038               float32 frequency, uint32 samplingRate);
00039
00043         void Add();
00044
00045     private:
00050         float32 amplitude;
00051
00055         float32* data;
00056
00060         uint32 size;
00061     };
00062
00063     PITone::PITone(float32 amplitude, float32* data,
00064                   uint32 size, float32 frequency, uint32 samplingRate)
00065         : Sinusoid(frequency, samplingRate) {
00066         this->data = data;
00067         this->amplitude = amplitude;
00068         this->size = size;
00069
00070         for(uint32 i = 0; i < samplingRate; i++) {
00071             sinusoid[i] *= amplitude;
00072         }
00073     }
00074
00075     void PITone::Add() {
00076         for(uint32 i = 0; i < size; i++) {
00077             data[i] += amplitude * next();
00078             data[i] /= (1 + amplitude); // ensures value <= 1
00079         }
00080     }
00081 }
00082
00083 #endif

```

8.59 src/Sinusoid.hpp File Reference

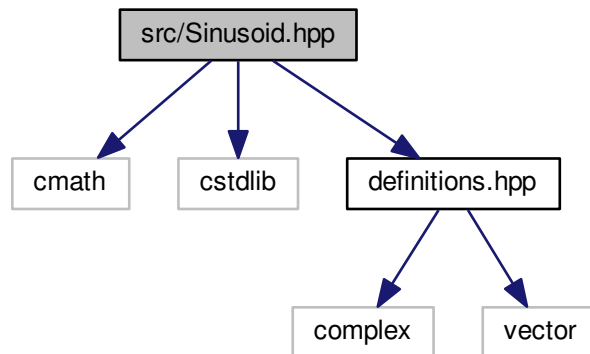
contains the Sinusoid class

```

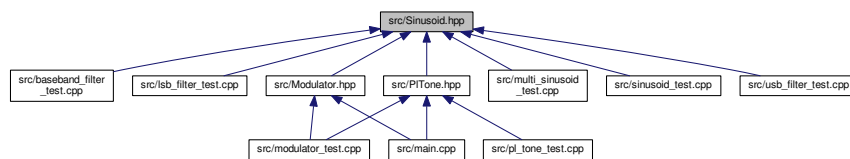
#include <cmath>
#include <cstdlib>
#include "definitions.hpp"

```

Include dependency graph for Sinusoid.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class [radio::Sinusoid](#)

Namespaces

- [radio](#)

Contains the classes for the various types of modulation supported by the program.

8.59.1 Detailed Description

contains the Sinusoid class

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [Sinusoid.hpp](#).

8.60 Sinusoid.hpp

00001


```

00007 #ifndef Sinusoid_H
00008 #define Sinusoid_H
00009
00010 #include <cmath>
00011 #include <cstdlib>
00012
00013 #include "definitions.hpp"
00014
00015 namespace radio {
00020     class Sinusoid {
00021     public:
00025         Sinusoid(float32 frequency, uint32
samplingRate = 48000);
00026
00030         ~Sinusoid();
00031
00036         float32 next();
00037
00042         float32 nextShifted();
00043
00044     protected:
00048         float32 frequency;
00049
00053         uint32 sinIndex = 0;
00054
00058         uint32 sinIndexShifted = 0;
00059
00063         uint32 samplingRate;
00064
00068         float32* sinusoid;
00069
00074         float32* sinusoidShift90;
00075     };
00076
00077     Sinusoid::Sinusoid(float32 frequency, uint32 samplingRate) {
00078         this->frequency = frequency;
00079         this->samplingRate = samplingRate;
00080         sinusoid = (float32*)std::malloc(samplingRate * sizeof(
float32));
00081         sinusoidShift90 = (float32*)std::malloc(samplingRate * sizeof(
float32));
00082
00083         float32 arg = 2 * M_PI * frequency / samplingRate;
00084
00085         for(uint32 i = 0; i < samplingRate; i++) {
00086             // cosine argument evaluates as float due to M_PI and frequency
00087             sinusoid[i] = cos(arg * i);
00088             sinusoidShift90[i] = sin(arg * i);
00089         }
00090     }
00091
00092     Sinusoid::~Sinusoid() {
00093         free(sinusoid);
00094         free(sinusoidShift90);
00095     }
00096
00097     float32 Sinusoid::next() {
00098         if(sinIndex >= samplingRate) sinIndex = 0;
00099         return sinusoid[sinIndex++];
00100     }
00101
00102     float32 Sinusoid::nextShifted() {
00103         if(sinIndexShifted >= samplingRate)
sinIndexShifted = 0;
00104         return sinusoidShift90[sinIndexShifted++];
00105     }
00106 }
00107
00108 #endif

```

8.61 src/sinusoid_test.cpp File Reference

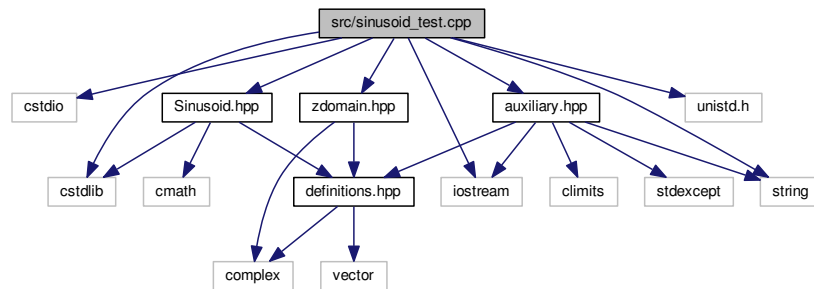
contains a test program to test the Sinusoid class

```

#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <string>
#include <unistd.h>
#include "auxiliary.hpp"
#include "Sinusoid.hpp"
#include "zdomain.hpp"

```

Include dependency graph for sinusoid_test.cpp:



Functions

- `int main (int argc, char *argv[])`

8.61.1 Detailed Description

contains a test program to test the Sinusoid class

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [sinusoid_test.cpp](#).

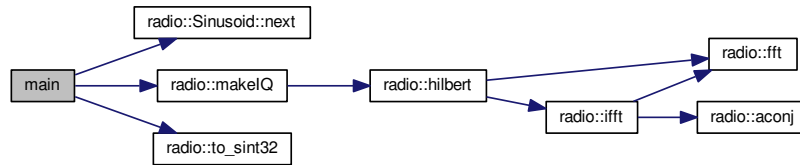
8.61.2 Function Documentation

8.61.2.1 `int main (int argc, char * argv[])`

Program to test the Sinusoid class.

Definition at line 23 of file [sinusoid_test.cpp](#).

Here is the call graph for this function:



8.62 sinusoid_test.cpp

```

00001
00007 #include <cstdio>
00008 #include <cstdlib>
00009 #include <iostream>
00010 #include <string>
00011 #include <unistd.h>
00012
00013 #include "auxiliary.hpp"
00014 #include "Sinusoid.hpp"
00015 #include "zdomain.hpp"
00016
00017 using namespace std;
00018 using namespace radio;
00019
00023 int main(int argc, char* argv[]) {
00024
00025     // Constants
00026     const uint16 BUFFER_SIZE = 16384;
00027
00028     // Declare primitive Variables
00029     float32 dataBuffer[BUFFER_SIZE];
00030     float32 iqBuffer[2 * BUFFER_SIZE];
00031     float32 freq = atof(argv[1]);
00032
00033     if(freq < 0) {
00034         cerr << "The given tone was invalid." << endl;
00035         return ERROR;
00036     }
00037
00038     Sinusoid sinusoid(freq, 48000);
00039
00040     while(true) {
00041         for(uint16 i = 0; i < BUFFER_SIZE; i++) {
00042             dataBuffer[i] = sinusoid.next();
00043         }
00044
00045         makeIQ(dataBuffer, iqBuffer, BUFFER_SIZE);
00046         to_sint32(iqBuffer, 2 * BUFFER_SIZE);
00047         write(STDOUT_FILENO, &iqBuffer, 2 * BUFFER_SIZE * sizeof(sint32));
00048     }
00049 }

```

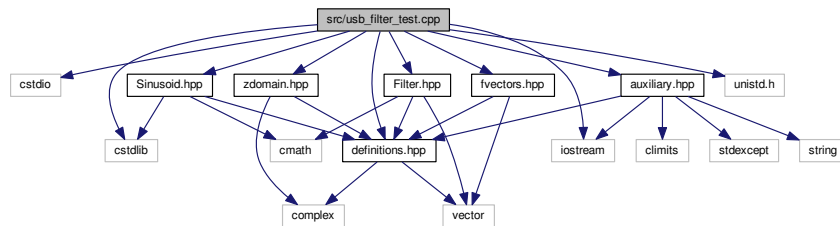
8.63 src/usb_filter_test.cpp File Reference

```

#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <unistd.h>
#include "auxiliary.hpp"
#include "definitions.hpp"
#include "Filter.hpp"
#include "fvectors.hpp"
#include "Sinusoid.hpp"
#include "zdomain.hpp"

```

Include dependency graph for `usb_filter_test.cpp`:



Functions

- `int main (int argc, char *argv[])`

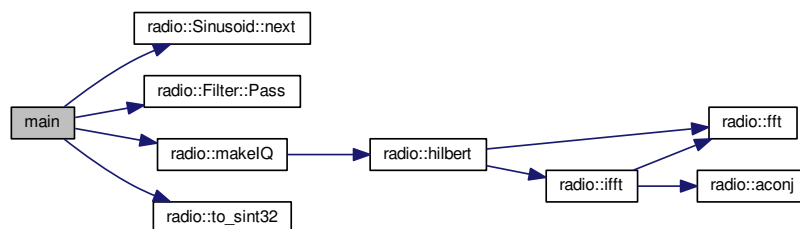
8.63.1 Function Documentation

8.63.1.1 `int main (int argc, char * argv[])`

Program to test the Filter class and the USB filter coefficients.

Definition at line 25 of file `usb_filter_test.cpp`.

Here is the call graph for this function:



8.64 usb_filter_test.cpp

```

00001
00007 #include <cstdio>
00008 #include <cstdlib>
00009 #include <iostream>
00010 #include <unistd.h>
00011
00012 #include "auxiliary.hpp"
00013 #include "definitions.hpp"
00014 #include "Filter.hpp"
00015 #include "fvecs.hpp"
00016 #include "Sinusoid.hpp"
00017 #include "zdomain.hpp"
00018
00019 using namespace std;
00020 using namespace radio;
00021
00025 int main(int argc, char* argv[]) {
00026
00027     // Constants
  
```

```

00028     const uint16 BUFFER_SIZE = 48000;
00029
00030     // Declare primitive Variables
00031     uint8 i = 0;
00032     uint8 size = 0;
00033     uint16 delta = 250;
00034     float32 dataBuffer[BUFFER_SIZE];
00035     float32 iqBuffer[2 * BUFFER_SIZE];
00036
00037     // create 1 sec of audio
00038     for(uint16 f = 17000; f <= 23000; f += delta, i++) {
00039         Sinusoid sinusoid(f);
00040
00041         for(uint16 i = 0; i < BUFFER_SIZE; i++) {
00042             dataBuffer[i] += sinusoid.next();
00043         }
00044     }
00045
00046     size = i;
00047
00048     // adjust dataBuffer so values are between -1 and 1
00049     for(uint16 i = 0; i < BUFFER_SIZE; i++) {
00050         dataBuffer[i] /= size;
00051     }
00052
00053     Filter filter(dataBuffer, BUFFER_SIZE, F_UPPERSIDE BAND);
00054     filter.Pass();
00055     makeIQ(dataBuffer, iqBuffer, BUFFER_SIZE);
00056     to_sint32(iqBuffer, 2 * BUFFER_SIZE);
00057
00058     while(true) {
00059         write(STDOUT_FILENO, &iqBuffer, 2 * BUFFER_SIZE * sizeof(sint32));
00060     }
00061 }

```

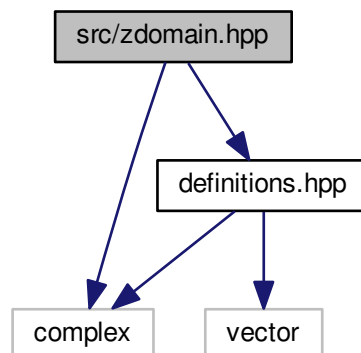
8.65 src/zdomain.hpp File Reference

Contains the functions to manipulate sequential data in the frequency (z) domain.

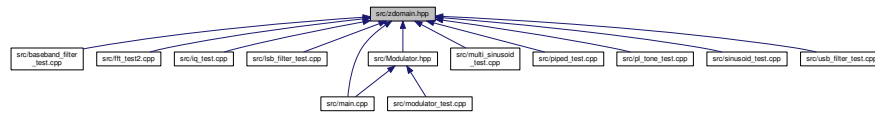
```
#include <complex>
```

```
#include "definitions.hpp"
```

Include dependency graph for zdomain.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- [radio](#)

Contains the classes for the various types of modulation supported by the program.

Functions

- void [radio::aconj](#) (cfloat32 *data, uint32 size)
- void [radio::fft](#) (cfloat32 *data, uint32 size)
- void [radio::hilbert](#) (float32 *data, float32 *dest, uint32 size)
- void [radio::ifft](#) (cfloat32 *data, uint32 size)
- void [radio::makeIQ](#) (float32 *data, float32 *dest, uint32 size)

8.65.1 Detailed Description

Contains the functions to manipulate sequential data in the frequency (z) domain.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file [zdomain.hpp](#).

8.66 zdomain.hpp

```

00001
00008 #ifndef zdomain_H
00009 #define zdomain_H
00010
00011 #include <complex>
00012
00013 #include "definitions.hpp"
00014
00015 namespace radio {
00016
00026     void aconj(cfloat32* data, uint32 size);
00027
00039     void fft(cfloat32* data, uint32 size);
00040
00052     void hilbert(float32* data, float32* dest, uint32 size);
00053
00066     void ifft(cfloat32* data, uint32 size);
00067
00082     void makeIQ(float32* data, float32* dest, uint32 size);
00083
00084     void aconj(cfloat32* data, uint32 size) {
00085         for(int i = 0; i < size; i++) {
00086             data[i] = std::conj(data[i]);
00087         }
00088     }
00089
00090     void fft(cfloat32* data, uint32 size) {
00091         // DFT
00092         uint32 k = size;
00093         uint32 n;
  
```

```

00094         float32 thetaT = M_PI / size;
00095         cfloat32 phiT(cos(thetaT), sin(thetaT));
00096         cfloat32 T;
00097
00098         while(k > 1) {
00099             n = k;
00100             k >>= 1;
00101             phiT = phiT * phiT;
00102             T = 1.0L;
00103
00104             for(uint32 l = 0; l < k; l++) {
00105                 for(uint32 a = l; a < size; a += n) {
00106                     uint32 b = a + k;
00107                     cfloat32 t = data[a] - data[b];
00108                     data[a] += data[b];
00109                     data[b] = t * T;
00110                 }
00111
00112                 T *= phiT;
00113             }
00114         }
00115
00116         // Decimate
00117         uint32 m = (uint32)log2(size);
00118
00119         for(uint32 a = 0; a < size; a++) {
00120             uint32 b = a;
00121
00122             // Reverse bits
00123             b = ((b & 0xaaaaaaaa) >> 1) | ((b & 0x55555555) << 1);
00124             b = (((b & 0xcccccccc) >> 2) | ((b & 0x33333333) << 2));
00125             b = (((b & 0xf0f0f0f0) >> 4) | ((b & 0x0f0f0f0f) << 4));
00126             b = (((b & 0xff00ff00) >> 8) | ((b & 0x00ff00ff) << 8));
00127             b = ((b >> 16) | (b << 16)) >> (32 - m);
00128
00129             if (b > a)
00130             {
00131                 cfloat32 t = data[a];
00132                 data[a] = data[b];
00133                 data[b] = t;
00134             }
00135         }
00136     }
00137
00138 void hilbert(float32* data, float32* dest, uint32 size) {
00139     cfloat32* temp = (cfloat32*)std::malloc(sizeof(cfloat32) * size);
00140
00141     for(int i = 0; i < size; i++) {
00142         temp[i] = data[i];
00143     }
00144
00145     fft(temp, size);
00146
00147     for(int i = size/2; i < size; i++) {
00148         temp[i] = 0;
00149     }
00150
00151     ifft(temp, size);
00152
00153     for(int i = 0; i < size; i++) {
00154         // parentheses around temp prevent free() error
00155         dest[i] = -2 * (temp[i].imag());
00156     }
00157
00158     free(temp);
00159 }
00160
00161 void ifft(cfloat32* data, uint32 size) {
00162     aconj(data, size);
00163     fft(data, size);
00164     aconj(data, size);
00165
00166     for(int i = 0; i < size; i++) {
00167         data[i] /= size;
00168     }
00169 }
00170
00171 void makeIQ(float32* data, float32* dest, uint32 size) {
00172     float32 quadData[size];
00173     hilbert(data, quadData, size);
00174
00175     for(int i = 0; i < 2 * size; i += 2) {
00176         dest[i] = quadData[i/2];
00177         dest[i+1] = data[i/2];
00178     }
00179 }
00180 }

```

```
00181  
00182 #endif
```


Index

aconj
radio, [13](#)

Age
radio, [12](#)

Argument
radio, [12](#)

bin/bbftest, [31](#)
bin/lsbftest, [31](#)
bin/modtest, [31](#), [32](#)
bin/msintest, [32](#)
bin/pltest, [32](#)
bin/radio, [32](#)
bin/sintest, [33](#)
bin/usbftest, [33](#)

DEN
radio, [12](#)

DSB_LC
radio, [13](#)

DSB_SC
radio, [13](#)

FM_NARROW
radio, [13](#)

FM_WIDE
radio, [13](#)

FREQ
radio, [12](#)

fft
radio, [13](#)

Fractional
radio, [12](#)

hilbert
radio, [14](#)

ifft
radio, [15](#)

LSB_FILTERED
radio, [13](#)

LSB_HILBERT
radio, [13](#)

MODE
radio, [12](#)

makefile, [34](#)

NEW
radio, [12](#)

NUM
radio, [12](#)

OLD
radio, [12](#)

PL_TONE
radio, [12](#)

radio, [11](#)
aconj, [13](#)
Age, [12](#)
Argument, [12](#)
DEN, [12](#)
DSB_LC, [13](#)
DSB_SC, [13](#)
FM_NARROW, [13](#)
FM_WIDE, [13](#)
FREQ, [12](#)
fft, [13](#)
Fractional, [12](#)
hilbert, [14](#)
ifft, [15](#)
LSB_FILTERED, [13](#)
LSB_HILBERT, [13](#)
MODE, [12](#)
NEW, [12](#)
NUM, [12](#)
OLD, [12](#)
PL_TONE, [12](#)
USB_FILTERED, [13](#)
USB_HILBERT, [13](#)

USB_FILTERED
radio, [13](#)

USB_HILBERT
radio, [13](#)