# An Inexpensive, Software-Defined IF Modulator

Generated by Doxygen 1.8.8

Wed Apr 13 2016 05:16:49

# **Contents**

| 1 | Bug  | List     |                                | 1  |
|---|------|----------|--------------------------------|----|
| 2 | Nam  | nespace  | Index                          | 3  |
|   | 2.1  | Names    | pace List                      | 3  |
| 3 | Hier | archica  | Index                          | 5  |
|   | 3.1  | Class I  | lierarchy                      | 5  |
| 4 | Clas | s Index  |                                | 7  |
|   | 4.1  | Class I  | ist                            | 7  |
| 5 | File | Index    |                                | 9  |
|   | 5.1  | File Lis | t                              | 9  |
| 6 | Nam  | nespace  | Documentation 1                | 11 |
|   | 6.1  | radio N  | amespace 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             | 13 |
|   |      |          | 6.1.2.4 ModulationType         | 13 |
|   |      | 6.1.3    | Function Documentation         | 13 |
|   |      |          | 6.1.3.1 aconj                  | 13 |
|   |      |          | 6.1.3.2 fft                    | 14 |
|   |      |          | 6.1.3.3 hilbert                | 15 |
|   |      |          | <b>6.1.3.4</b> ifft            | 16 |
|   |      |          | 6.1.3.5 makelQ                 | 17 |
|   |      |          | 6.1.3.6 ShowHelp               | 18 |
|   |      |          | 6.1.3.7 to_sint32              | 18 |
|   |      |          | 6.1.3.8 to_type                | 19 |
|   |      | 6.1.4    | Variable Documentation         | 20 |
|   |      |          | C141 E DACEDAND                | 20 |

iv CONTENTS

|   |      |          | 6.1.4.2 F_LOWERSIDEBAND                | 0 |
|---|------|----------|--|---|
|   |      |          | 6.1.4.3 F_UPPERSIDEBAND                | 1 |
|   |      |          | 6.1.4.4 FREQ_INTERMEDIATE              | 1 |
|   |      |          | 6.1.4.5 SAMPLING_RATE                  | 1 |
| 7 | Clas | s Docu   | mentation 2                            | 3 |
|   | 7.1  | radio::l | Filter Class Reference                 | 3 |
|   |      | 7.1.1    | Detailed Description                   | 3 |
|   |      | 7.1.2    | Constructor & Destructor Documentation | 3 |
|   |      |          | 7.1.2.1 Filter                         | 3 |
|   |      | 7.1.3    | Member Function Documentation          | 4 |
|   |      |          | 7.1.3.1 Pass                           | 4 |
|   |      | 7.1.4    | Member Data Documentation              | 5 |
|   |      |          | 7.1.4.1 data                           | 5 |
|   |      |          | 7.1.4.2 diffEq                         | 5 |
|   |      |          | 7.1.4.3 eqLength                       | 5 |
|   |      |          | 7.1.4.4 size                           | 5 |
|   | 7.2  | radio::0 | Gain Class Reference                   | 5 |
|   |      | 7.2.1    | Detailed Description                   | 6 |
|   |      | 7.2.2    | Constructor & Destructor Documentation | 6 |
|   |      |          | 7.2.2.1 Gain                           | 6 |
|   |      | 7.2.3    | Member Function Documentation          | 6 |
|   |      |          | 7.2.3.1 Apply                          | 6 |
|   | 7.3  | radio::l | Modulator Class Reference              | 7 |
|   |      | 7.3.1    | Detailed Description                   | 7 |
|   |      | 7.3.2    | Constructor & Destructor Documentation | 7 |
|   |      |          | 7.3.2.1 Modulator                      | 7 |
|   |      |          | 7.3.2.2 ~Modulator                     | 7 |
|   |      | 7.3.3    | Member Function Documentation          | 8 |
|   |      |          | 7.3.3.1 Mod                            | 8 |
|   | 7.4  | radio::l | PITone Class Reference                 | 9 |
|   |      | 7.4.1    | Detailed Description                   | 0 |
|   |      | 7.4.2    | Constructor & Destructor Documentation | 0 |
|   |      |          | 7.4.2.1 PITone                         | 0 |
|   |      | 7.4.3    | Member Function Documentation          | 0 |
|   |      |          | 7.4.3.1 Add                            | 1 |
|   | 7.5  | radio::  | Sinusoid Class Reference               | 1 |
|   |      | 7.5.1    | Detailed Description                   | 2 |
|   |      | 7.5.2    | Constructor & Destructor Documentation | 2 |
|   |      |          | 7.5.2.1 Sinusoid                       | 2 |
|   |      |          |  |   |

CONTENTS

|   |        |          | 7500 0' ''                    |                                       | ~~ |
|---|--------|----------|-------------------------------|---------------------------------------|----|
|   |        | 7.5.0    | 7.5.2.2 ~Sinusoid             |                                       | 33 |
|   |        | 7.5.3    | Member Function Documentation |                                       | 33 |
|   |        |          | 7.5.3.1 next                  |                                       | 33 |
|   |        |          | 7.5.3.2 nextShifted           |                                       | 33 |
|   |        | 7.5.4    | Member Data Documentation     |                                       | 34 |
|   |        |          | 7.5.4.1 frequency             |                                       | 34 |
|   |        |          | 7.5.4.2 samplingRate          |                                       | 34 |
|   |        |          | 7.5.4.3 sinIndex              |                                       | 34 |
|   |        |          | 7.5.4.4 sinIndexShifted       |                                       | 34 |
|   |        |          | 7.5.4.5 sinusoid              |                                       | 34 |
|   |        |          | 7.5.4.6 sinusoidShift90       |                                       | 34 |
| 8 | File I | Docume   | entation                      |                                       | 35 |
| • | 8.1    |          |                               |                                       | 35 |
|   | 0.1    |          | test File Reference           |                                       |    |
|   |        | 8.1.1    | Detailed Description          |                                       | 35 |
|   | 8.2    |          |                               |                                       | 35 |
|   | 8.3    |          | ftest File Reference          |                                       | 35 |
|   |        | 8.3.1    | Detailed Description          |                                       | 35 |
|   | 8.4    |          | t                             |                                       | 35 |
|   | 8.5    | bin/mo   | dtest File Reference          |                                       | 35 |
|   |        | 8.5.1    | Detailed Description          |                                       | 35 |
|   | 8.6    | modtes   | st                            |                                       | 36 |
|   | 8.7    | bin/msi  | intest File Reference         |                                       | 36 |
|   |        | 8.7.1    | Detailed Description          |                                       | 36 |
|   | 8.8    | msintes  | st                            |                                       | 36 |
|   | 8.9    | bin/plte | est File Reference            |                                       | 36 |
|   |        | 8.9.1    | Detailed Description          |                                       | 36 |
|   | 8.10   | pltest . |                               |                                       | 36 |
|   | 8.11   | bin/rad  | io File Reference             |                                       | 36 |
|   |        | 8.11.1   | Detailed Description          |                                       | 36 |
|   | 8.12   | radio .  |                               |                                       | 36 |
|   | 8.13   | bin/sint | test File Reference           |                                       | 37 |
|   |        | 8.13.1   | Detailed Description          |                                       | 37 |
|   | 8.14   | sintest  |                               |                                       | 37 |
|   | 8.15   | bin/usb  | oftest File Reference         |                                       | 37 |
|   |        |          | Detailed Description          |                                       | 37 |
|   | 8.16   |          | ·<br>st                       |                                       | 37 |
|   |        |          | xygen.config File Reference   |                                       | 37 |
|   | •      |          | Detailed Description          |                                       | 37 |
|   | 8.18   |          | en.config                     |                                       | 37 |
|   | 00     | ~~.,,90  |                               | · · · · · · · · · · · · · · · · · · · | ٠, |

vi CONTENTS

| 8.19 | makefile   | File Reference                      | 8  |
|------|------------|-------------------------------------|----|
|      | 8.19.1     | Detailed Description                | 8  |
| 8.20 | makefile   |                                     | 8  |
| 8.21 | src/alsa_  | test.cpp File Reference             | 19 |
|      | 8.21.1     | Detailed Description                | 39 |
|      | 8.21.2 F   | Function Documentation              | 10 |
|      | 8          | 3.21.2.1 main                       | 10 |
| 8.22 | alsa_test  | .cpp                                | 1  |
| 8.23 | src/auxili | ary.hpp File Reference              | 2  |
|      | 8.23.1     | Detailed Description                | 13 |
| 8.24 | auxiliary. | hpp                                 | 13 |
| 8.25 | src/basel  | pand_filter_test.cpp File Reference | 4  |
|      | 8.25.1     | Detailed Description                | 15 |
|      | 8.25.2 F   | Function Documentation              | 15 |
|      | 8          | 3.25.2.1 main                       | 15 |
| 8.26 | baseban    | d_filter_test.cpp                   | 16 |
| 8.27 | src/defini | tions.hpp File Reference            | 16 |
|      | 8.27.1     | Detailed Description                | 8  |
|      | 8.27.2 N   | Macro Definition Documentation      | 8  |
|      | 8          | 3.27.2.1 ENUM                       | 8  |
|      | 8          | 3.27.2.2 ERROR                      | 8  |
|      | 8.27.3     |                                     | 8  |
|      | 8          | 3.27.3.1 byte                       | 8  |
|      | 8          | 3.27.3.2 cfloat32                   | 8  |
|      | 8          | 3.27.3.3 float32                    | 8  |
|      | 8          | 3.27.3.4 float64                    | 8  |
|      | 8          | 3.27.3.5 fparams                    | 19 |
|      | 8          |                                     | 19 |
|      | 8          | 3.27.3.7 sint32                     | 19 |
|      | 8          | 3.27.3.8 sint64                     | 19 |
|      | 8          | 3.27.3.9 sint8                      | 19 |
|      |            |                                     | 19 |
|      | 8          | 3.27.3.11 uint32                    | 19 |
|      |            |                                     | 19 |
|      | 8          | 3.27.3.13 uint8                     | 19 |
| 8.28 | definition | s.hpp                               | 19 |
| 8.29 |            |                                     | 0  |
|      |            | ·                                   | 1  |
|      |            |                                     | 1  |
|      | 8          | 3.29.2.1 CArray                     | 1  |

CONTENTS vii

|      | 8.29.3 Function Documentation             | 51 |
|------|---|----|
|      | 8.29.3.1 fft                              | 51 |
|      | 8.29.3.2 hilbert                          | 52 |
|      | 8.29.3.3 ifft                             | 52 |
|      | 8.29.3.4 main                             | 53 |
|      | 8.29.4 Variable Documentation             | 54 |
|      | 8.29.4.1 Pl                               | 54 |
| 8.30 | fft_test.cpp                              | 54 |
| 8.31 | src/fft_test2.cpp File Reference          | 55 |
|      | 8.31.1 Detailed Description               | 56 |
|      | 8.31.2 Function Documentation             | 56 |
|      | 8.31.2.1 main                             | 56 |
| 8.32 | fft_test2.cpp                             | 57 |
| 8.33 | src/Filter.hpp File Reference             | 58 |
|      | 8.33.1 Detailed Description               | 59 |
| 8.34 | Filter.hpp                                | 59 |
| 8.35 | src/fvectors.hpp File Reference           | 60 |
|      | 8.35.1 Detailed Description               | 61 |
| 8.36 | fvectors.hpp                              | 61 |
| 8.37 | src/Gain.hpp File Reference               | 62 |
|      | 8.37.1 Detailed Description               | 63 |
| 8.38 | Gain.hpp                                  | 64 |
| 8.39 | src/iq_test.cpp File Reference            | 64 |
|      | 8.39.1 Detailed Description               | 65 |
|      | 8.39.2 Function Documentation             | 65 |
|      | 8.39.2.1 main                             | 65 |
| 8.40 | iq_test.cpp                               | 66 |
| 8.41 | src/lsb_filter_test.cpp    File Reference | 66 |
|      | 8.41.1 Function Documentation             | 67 |
|      | 8.41.1.1 main                             | 67 |
| 8.42 | lsb_filter_test.cpp                       | 68 |
| 8.43 | src/main.cpp File Reference               | 69 |
|      | 8.43.1 Detailed Description               | 69 |
|      | 8.43.2 Function Documentation             | 69 |
|      | 8.43.2.1 main                             | 69 |
| 8.44 | main.cpp                                  | 71 |
| 8.45 | src/mic_test.cpp File Reference           | 72 |
|      | 8.45.1 Detailed Description               | 73 |
|      | 8.45.2 Function Documentation             | 73 |
|      | 8.45.2.1 main                             | 73 |
|      |   |    |

viii CONTENTS

| 8.46 | mic_tes   | t.cpp                               | 74 |
|------|-----------|-------------------------------------|----|
| 8.47 | src/Mod   | dulator.hpp File Reference          | 76 |
| 8.48 | Modula    | tor.hpp                             | 77 |
| 8.49 | src/mod   | lulator_test.cpp File Reference     | 78 |
|      | 8.49.1    | Detailed Description                | 79 |
|      | 8.49.2    | Function Documentation              | 79 |
|      |           | 8.49.2.1 main                       | 79 |
| 8.50 | modula    | tor_test.cpp                        | 80 |
| 8.51 | src/mult  | ti_sinusoid_test.cpp File Reference | 81 |
|      | 8.51.1    | Detailed Description                | 82 |
|      | 8.51.2    | Function Documentation              | 82 |
|      |           | 8.51.2.1 main                       | 82 |
| 8.52 | multi_si  | nusoid_test.cpp                     | 83 |
| 8.53 | src/pipe  | d_test.cpp File Reference           | 84 |
|      | 8.53.1    | Detailed Description                | 84 |
|      | 8.53.2    | Function Documentation              | 84 |
|      |           | 8.53.2.1 main                       | 84 |
| 8.54 | piped_t   | est.cpp                             | 85 |
| 8.55 | src/pl_te | one_test.cpp File Reference         | 85 |
|      | 8.55.1    | Detailed Description                | 86 |
|      | 8.55.2    | Function Documentation              | 86 |
|      |           | 8.55.2.1 main                       | 86 |
| 8.56 | pl_tone   | _test.cpp                           | 87 |
| 8.57 | src/PITo  | one.hpp File Reference              | 87 |
|      | 8.57.1    | Detailed Description                | 88 |
| 8.58 | PITone.   | hpp                                 | 89 |
| 8.59 | src/Sinu  | usoid.hpp File Reference            | 89 |
|      | 8.59.1    | Detailed Description                | 90 |
| 8.60 | Sinusoi   | d.hpp                               | 90 |
| 8.61 | src/sinu  | soid_test.cpp File Reference        | 91 |
|      | 8.61.1    | Detailed Description                | 92 |
|      | 8.61.2    | Function Documentation              | 92 |
|      |           | 8.61.2.1 main                       | 92 |
| 8.62 | sinusoid  | d_test.cpp                          | 93 |
| 8.63 | src/usb   | _filter_test.cpp File Reference     | 94 |
|      | 8.63.1    | Function Documentation              | 94 |
|      |           | 8.63.1.1 main                       | 94 |
| 8.64 | usb_filte | er_test.cpp                         | 95 |
| 8.65 | src/zdoi  | main.hpp File Reference             | 96 |
|      | 8.65.1    | Detailed Description                | 97 |
|      |           |                                     |    |

| CONTENTS         | IX |
|------------------|----|
| 8.66 zdomain.hpp | 97 |
| Index            | 99 |

# **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

2 **Bug List** 

# Namespace Index

|  | 2.1 | Name | espace | List |
|--|-----|------|--------|------|
|--|-----|------|--------|------|

Here is a list of all namespaces with brief descriptions:

radio

Namespace Index

# **Hierarchical Index**

# 3.1 Class Hierarchy

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

| radio::Filter    | 23 | 3 |
|------------------|----|---|
| radio::Gain      | 2! | 5 |
| radio::Modulator | 2  | 7 |
| radio::Sinusoid  | 3  | 1 |
| radio::PITone    | 20 | q |

6 **Hierarchical Index** 

# **Class Index**

# 4.1 Class List

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

| radio::Filter    | 23 |
|------------------|----|
| radio::Gain      | 25 |
| radio::Modulator | 27 |
| radio::PITone    | 29 |
| radio::Sinusoid  | 31 |

8 Class Index

# 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 docu-          |   |
| mentation and counting total lines of code in src/   | ( |
| bin/bbftest  |   |
| bin/lsbftest   | ( |
| bin/modtest  | ; |
| bin/msintest   | ; |
| bin/pltest   | ; |
| bin/radio  | , |
| bin/sintest  | ; |
| bin/usbftest   | ; |
| etc/doxygen.config   |   |
| Contains doxygen configuration   | ( |
| src/alsa_test.cpp  |   |
| Tests sinusoidal tone generation   | ; |
| src/auxiliary.hpp  |   |
| Contains helper-functions for main()   | 4 |
| src/baseband_filter_test.cpp   |   |
| Contains a program to demonstrate the the baseband/AF filter   | 4 |
| src/definitions.hpp  |   |
| Contains declarations of system-independant (universal size) integers and float types, shortened     |   |
| type names for some commonly used types, and enumerations  | 4 |
| src/fft_test.cpp   |   |
| Tests FFT, IFFT, and Hilbert implementations   |   |
| src/fft_test2.cpp  |   |
| Tests FFT, IFFT, and Hilbert implementations in zdomain.hpp  | , |
| src/Filter.hpp   |   |
| Defines the Filter class   | , |
| src/fvectors.hpp   |   |
| Defines the transfer function coefficients used in the instances of the Filter class in this program | ( |
| src/Gain.hpp   |   |
| Contains the Gain class  | ( |
| src/iq_test.cpp  |   |
| Generates test IQ signal   | ( |
| src/lsb_filter_test.cpp  | ( |
| src/main.cpp   |   |
| "brains" of the entire project   | ( |

10 File Index

| src/mic_test.cpp  |    |
|---|----|
| Tests getting mic input via ALSA May not even compile at the moment                           | 72 |
| src/Modulator.hpp   | 76 |
| src/modulator_test.cpp  |    |
| Test program to test the Modulator class  | 78 |
| 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  | 81 |
| src/piped_test.cpp  |    |
| Containts the original program used to test the piping-in idea                                | 84 |
| src/pl_tone_test.cpp  |    |
| Test program to test the PITone class   | 85 |
| src/PlTone.hpp  |    |
| PITone class  | 87 |
| src/Sinusoid.hpp  |    |
| Sinusoid class  | 89 |
| src/sinusoid_test.cpp   |    |
| Test program to test the Sinusoid class   | 91 |
| src/usb_filter_test.cpp   | 94 |
| src/zdomain.hpp   |    |
| Contains the functions to manipulate sequential data in the frequency (z) domain              | 96 |

# **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, awisner 940 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.

```
00052 { OLD, NEW };
```

### 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.

```
00062 { FREQ = 1, MODE, PL_TONE };
```

#### 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.

```
00057 { NUM, DEN };
```

#### **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

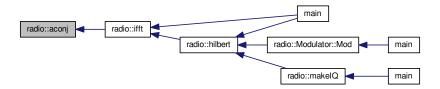
```
6.1.3.1 void radio::aconj ( cfloat32 * data, uint32 size )
```

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

#### **Parameters**

| data | the array whose values should be replaced with their respective conjugates |
|------|--|
| size | the number of elements in the data array                                   |

Definition at line 84 of file zdomain.hpp.



#### 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.

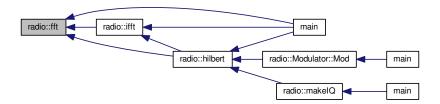
This code is based on code from http://rosettacode.org/wiki/Fast\_Fourier\_transform $\#C. \leftarrow 2B.2B.$ 

#### **Parameters**

| data | the array whose values should be replaced with its DFT |
|------|--|
| size | the number of elements in the data array               |

Definition at line 90 of file zdomain.hpp.

```
00090
00091
                // DFT
00092
                uint32 k = size;
                uint32 n;
00093
                float32 thetaT = M_PI / size;
cfloat32 phiT(cos(thetaT), sin(thetaT));
cfloat32 T;
00094
00095
00096
00097
00098
                while(k > 1) {
00099
                     n = k;
00100
                     k >>= 1;
00101
                     phiT = phiT * phiT;
                     T = 1.0L;
00102
00103
00104
                     for (uint32 1 = 0; 1 < k; 1++) {
00105
                          for(uint32 a = 1; a < size; a += n) {</pre>
                              uint32 b = a + k;
00106
00107
                              cfloat32 t = data[a] -data[b];
00108
                              data[a] +=data[b];
data[b] = t * T;
00109
00110
00111
00112
                          T \star = phiT;
00113
                     }
00114
                }
00115
00116
                // Decimate
                uint32 m = (uint32)log2(size);
00117
00118
00119
                for (uint32 a = 0; a < size; a++) {</pre>
00120
                     uint32 b = a;
00121
00122
                     // Reverse bits
00123
                     b = (((b \& 0xaaaaaaaa) >> 1) | ((b \& 0x55555555) << 1));
00124
                     b = (((b \& 0xcccccc) >> 2) | ((b \& 0x33333333) << 2));
                     b = (((b & 0xf0f0f0f00) >> 4) | ((b & 0x0f0f0f0f) << 4));
b = (((b & 0xff00ff00) >> 8) | ((b & 0x00ff00ff) << 8));
00125
00126
                     b = ((b >> 16) | (b << 16)) >> (32 - m);
00127
00128
00129
                     if (b > a)
00130
                     {
00131
                          cfloat32 t = data[a];
00132
                          data[a] =data[b];
                          data[b] = t;
00133
00134
                     }
00135
                }
00136
           }
```



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

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

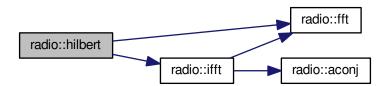
#### **Parameters**

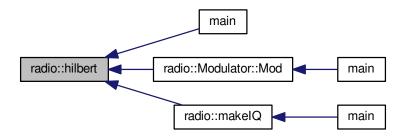
| data | the source array of the REAL numbers of which to take the Hilbert transform    |
|------|--|
| dest | the destination array of REAL numbers for the results of the Hilbert transform |
| size | the number of elements in the data and dest arrays                             |

Definition at line 138 of file zdomain.hpp.

```
00138
00139
                cfloat32* temp = (cfloat32*)std::malloc(sizeof(cfloat32) * size);
00140
00141
                for(int i = 0; i < size; i++) {</pre>
00142
                    temp[i] = data[i];
00143
00144
00145
                fft(temp, size);
00146
00147
                for(int i = size/2; i < size; i++) {</pre>
00148
                    temp[i] = 0;
00149
00150
00151
                ifft(temp, size);
00152
00153
                for(int i = 0; i < size; i++) {</pre>
                    // parentheses around temp prevent free() error
dest[i] = -2 * (temp[i].imag());
00154
00155
00156
00157
00158
                free(temp);
00159
```

Here is the call 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.

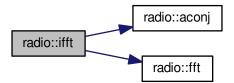
This code is based on code from http://rosettacode.org/wiki/Fast\_Fourier\_transform $\#C. \leftarrow 2B.2B.$ 

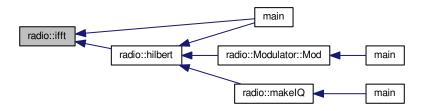
#### **Parameters**

| data | the array whose values should be replaced with its inverse DFT |
|------|--|
| size | the number of elements in the data array                       |

Definition at line 161 of file zdomain.hpp.

Here is the call graph for this function:





# 6.1.3.5 void radio::makelQ ( float32 \* data, float32 \* dest, uint32 size )

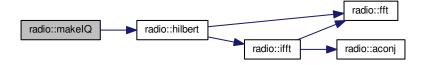
Produces an interleaved array of first an element from an original array of data and then an element from the original data's Hilbert transform. This function is intended to generate a two-channel output (I/Q output) for mixing applications.

#### **Parameters**

| data | the original data (left channel)   |
|------|--|
| dest | the interleaved data (left channel original data, right channel transformed data) twice the size |
|      | of the original data array   |
| size | 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:





#### 6.1.3.6 void radio::ShowHelp()

Displays the help information.

Definition at line 22 of file auxiliary.hpp.

```
std::cerr << std::endl << "Usage: radio [MODE] [MIC GAIN] "
   "[PL TONE]" << std::endl << std::endl
   << "MODE: one of the following types "
   "of modulation" << std::endl << std::endl;
00023
00024
00025
00026
00027
              00028
00029
                   << "dsbsc\t\tDouble sideband, suppressed carrier" << std::endl
00030
                   << "lsbhil\ttLower sideband created via Hilbert transform"
00031
00032
                   << std::endl
                   << \verb"lsbfilt\t\t\bwer sideband created via digital low-pass filter"
00033
00034
00035
                   << "usbhil\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
                   << "fm/t/talias for wfm" << std::endl << std::endl;
00042 //
00043
00044
              std::cerr << "MIC GAIN: Microphone power gain expressed in decibels"</pre>
00045
              << std::endl << std::endl;
00046
00047
               std::cerr << "PL TONE: Optional specification for CTCSS tone from "</pre>
00048
                   "60-260 Hz" << std::endl << std::endl;
00049
00050
               std::exit(ERROR);
00051
           }
```

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**

| data | the array containing the float32 samples that are directly replaced by their respective sint32 |
|------|--|
|      | representations  |
| size | 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**

| ctr | tune of modulation in tuned form |
|-----|----------------------------------|
| Str | type of modulation in typed form |
|     | Aber and Aber a                  |

#### Returns

enum value of the type of modulation

Definition at line 80 of file auxiliary.hpp.

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



### 6.1.4 Variable Documentation

### 6.1.4.1 fparams radio::F\_BASEBAND

#### Initial value:

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> {
             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.

| Namespace | D | ocur | nen | tat | ior |
|-----------|---|------|-----|-----|-----|
|           |   |      |     |     |     |

# **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 arameters

24 Class Documentation

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

Definition at line 80 of file Filter.hpp.

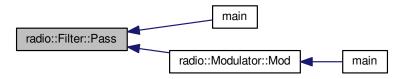
### 7.1.3 Member Function Documentation

#### 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.

```
00087
                   float64 temp[size];
00088
00089
00090
                   // create first values in filtered data
                   for(int i = 0; i < eqLength; i++) {</pre>
00091
                        temp[i] = 0;
00092
00093
                        for(int j = 0; j < eqLength; j++) {
   temp[i] += diffEq[NUM][j] * (j > i ? 0 : data[i - j]);
00094
00095
00096
00097
                        for(int j = 1; j < eqLength; j++) {
   temp[i] -= diffEq[DEN][j] * (j > i ? 0 : temp[i - j]);
00098
00099
00100
                         }
00101
                   }
00102
                   \ensuremath{//} create the REST of the values in filtered data
00103
                   for(int i = eqLength; i < size; i++) {
   temp[i] = 0;</pre>
00104
00105
00106
                        for(int j = 0; j < eqLength; j++) {
   temp[i] += diffEq[NUM][j] * data[i - j];</pre>
00107
00108
00109
00110
00111
                        for(int j = 1; j < eqLength; j++) {
   temp[i] -= diffEq[DEN][j] * temp[i - j];</pre>
00112
00113
00114
00115
              // save final values of data and filtered data
for(int i = 0; i < size; i++) {
    data[i] = temp[i];</pre>
00116
00117
00118
00119
00120
             }
```



#### 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 tranfer function of the filter in decending 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 denomenator) 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 ()

26 Class Documentation

# 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**

| data   | the signal to which the gain is applied  |
|--------|--|
| size   | the number of elements in the data array |
| gaindB | 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.

```
00067
00068
                for(uint32 i = 0; i < size; i++) {</pre>
00069
                    data[i] *= gainCoeff;
00070
                    if((data[i] > 1 \mid | data[i] < -1) && !hasClipped) {
00071
00072
                        hasClipped = true;
std::cerr << "Baseband clipping has occurred!"
00073
00074
                             << std::endl;
00075
00076
               }
00077
```

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**

| freaInter | the frequency of the IF carrier sinusoid |
|-----------|--|
| rate      |  |
| data      |  |
| <br>      |  |
| size      | the number of elements in data           |
| type      | form of modulation to use                |

Definition at line 103 of file Modulator.hpp.

```
00104
                freqCarrier = freqInter;
00105
00106
                this->rate = rate;
                this->data = data;
00107
                this->size = size;
00108
00109
               this->type = type;
00110
00111
                if(type == ModulationType::USB_HILBERT
                    || type == ModulationType::LSB_HILBERT) {
hilData = (float32*)malloc(size*sizeof(float32));
00112
00113
00114
00115
```

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

Definition at line 117 of file Modulator.hpp.

28 Class Documentation

#### 7.3.3 Member Function Documentation

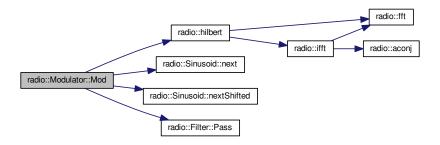
#### 7.3.3.1 void radio::Modulator::Mod ( )

Modulates the audio currently in the data array.

Definition at line 121 of file Modulator.hpp.

```
00121
00122
               \ensuremath{//} these variables should only ever be created once
               static float32 fmArg = 2 * M_PI * freqCarrier / (float32)rate;
static float32 fmK = 2 * M_PI / rate;
static float32 fmSum = 0; // cummulative sum used in FM modulation
00123
00124
00125
               static Filter lsbFilter(data, size, F_LOWERSIDEBAND);
00126
               static Sinusoid sinusoid(freqCarrier, rate); // IF c static Filter usbFilter(data, size, F_UPPERSIDEBAND);
                                                                    // IF carrier sinusoid
00128
00129
00130
                // take hilbert transform if necessary
               00131
00132
00133
                    hilbert (data, hilData, size);
00134
               } else if(type == ModulationType::FM_NARROW) {
00135
                   fmK *= 2.5;
               } else if(type == ModulationType::FM_WIDE) {
00136
                    fmK *= 5;
00137
00138
               }
00139
00140
                // perform main modulation
00141
                for(uint32 i = 0; i < size; i++) {</pre>
                   switch(type) {
    case ModulationType::DSB_LC:
00142
00143
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:
                             data[i] = data[i] * sinusoid.next()
  - hilData[i] * sinusoid.nextShifted();
00154
00155
00156
                             break:
00157
00158
                         case ModulationType::LSB_HILBERT:
00159
                           data[i] = data[i] * sinusoid.next()
00160
                                 + hilData[i] * sinusoid.nextShifted();
00161
00162
00163
                         case ModulationType::FM_NARROW:
                         case ModulationType::FM_WIDE:
    fmSum += fmK * data[i];
00164
00165
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) {
                    lsbFilter.Pass();
00173
00174
               } else if(type == ModulationType::USB_FILTERED) {
                    usbFilter.Pass();
00175
00176
00177
           }
```

Here is the call graph for this function:



Here is the caller graph for this function:



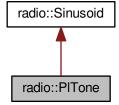
The documentation for this class was generated from the following file:

src/Modulator.hpp

# 7.4 radio::PITone Class Reference

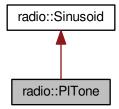
#include <PlTone.hpp>

Inheritance diagram for radio::PITone:



30 Class Documentation

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:( float32 amplitude, float32 \* data, uint32 size, float32 frequency, uint32 samplingRate )

Creates a PITone object.

### Parameters

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

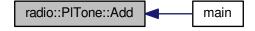
Adds the CTCSS tone to the baseband signal.

Definition at line 75 of file PlTone.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:

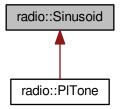
src/PlTone.hpp

# 7.5 radio::Sinusoid Class Reference

#include <Sinusoid.hpp>

32 Class Documentation

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

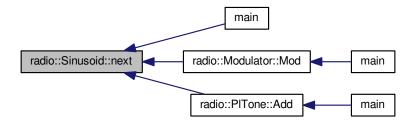
#### 7.5.3.1 float32 radio::Sinusoid::next()

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

Definition at line 97 of file Sinusoid.hpp.

```
00097
00098
if(sinIndex >= samplingRate) sinIndex = 0;
00099
return sinusoid[sinIndex++];
00100
}
```

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 consistant with a ring buffer.

Definition at line 102 of file Sinusoid.hpp.

34 Class Documentation

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/Isbftest File Reference

# 8.3.1 Detailed Description

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file Isbftest.

### 8.4 Isbftest

```
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 | \setminus 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 #USE_MDFILE_AS_MAINPAGE = README.md
00021 LATEX_SOURCE_CODE = YES
00022 STRIP_CODE_COMMENTS = YES
00023 INLINE_SOURCES = YES 00024 DISABLE_INDEX = NO
00025 GENERATE TREEVIEW = YES
00026 SEARCHENGINE = YES
00027 SERVER_BASED_SEARCH = NO
00028
00029 HAVE_DOT = YES
00030 CALL_GRAPH = YES
00031 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

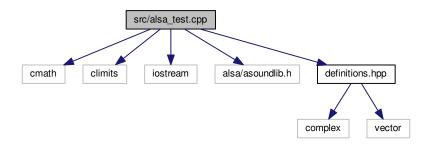
```
00001 GCC = g++-g-std=gnu++14
00002
00003 alsa-test:
00004
         $(GCC) src/alsa_test.cpp -o bin/alsatest -00 -lasound
00005
00006 baseband-filter-test:
00007
         GCC src/baseband_filter_test.cpp -o bin/basebandfiltertest
80000
00009 count:
00010
         grep -r "src/" -e "Samuel Andrew Wisner" -l | xargs wc -l
00011
00012 docs:
00013
         rm -r doc/
         doxygen etc/doxygen.config
00014
00015
         cd doc/latex; make pdf;
00016
         git reset
00017
         git add doc/.
00018
         git --no-pager log > etc/log.txt
         git add etc/log.txt
git commit -m "Updated documentation."
00019
00020
00021
         git push
00022
00023 fft-test:
00024
         $(GCC) src/fft_test.cpp -o bin/fft-test
00025
00026 fft-test2:
         $(GCC) src/fft_test2.cpp -o bin/fft-test2
00027
00028
00029 iq-test:
```

```
00030
         $(GCC) src/iq_test.cpp -o bin/iqtest
00031
00032 multi-sinusoid-test:
         GCC src/multi_sinusoid_test.cpp -o bin/multisinusoidtest
00033
00034
00035 modulator-test:
        $(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:
        $(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.

```
00023
           int ret;
00024
           snd_pcm_t* pcm_handle; // device handle
snd_pcm_stream_t stream = SND_PCM_STREAM_PLAYBACK;
00025
00026
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;
           long unsigned int bufferSize = 4096*4; // anything >8192 causes seg fault
00032
           const uint32 len = bufferSize*100;
00033
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) {
               vals[i] = (sint16)(SHRT_MAX * cos(arg * i/2) + 0.5);
00040
00041
                vals[i+1] = vals[i];
00042
00043
           ret = snd_pcm_open(&pcm_handle, pcm_name, stream, 0);
cout << "Opening: " << snd_strerror(ret) << endl;</pre>
00044
00045
00046
00047
           ret = snd_pcm_hw_params_any(pcm_handle, hwparams);
00048
           cout << "Initializing hwparams structure: " << snd_strerror(ret) << endl;</pre>
00049
00050
           ret = snd_pcm_hw_params_set_access(pcm_handle, hwparams,
           SND_PCM_ACCESS_RW_INTERLEAVED);
cout << "Setting access: " << snd_strerror(ret) << endl;
00051
00052
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;</pre>
00057
00058
           ret = snd_pcm_hw_params_set_rate(pcm_handle, hwparams,
00059
                    rate, (int)0);
00060
           cout << "Setting rate: " << snd_strerror(ret) << endl;</pre>
00061
           ret = snd_pcm_hw_params_set_channels(pcm_handle, hwparams, 2);
cout << "Setting channels: " << snd_strerror(ret) << endl;</pre>
00062
00063
00064
00065
           ret = snd_pcm_hw_params_set_periods(pcm_handle, hwparams, 2, 0);
00066
           cout << "Setting periods: " << snd_strerror(ret) << endl;</pre>
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;</pre>
00071
00072
           ret = snd_pcm_hw_params(pcm_handle, hwparams);
00073
           cout << "Applying parameters: " << snd_strerror(ret) << endl;</pre>
00074
           ret = snd_pcm_hw_params_get_period_size(hwparams, &count, 0);
cout << "Actual period size: " << count << endl;</pre>
00075 //
00076
           cout << "Returned: " << snd_strerror(ret) << endl;</pre>
00077
00078
00079
00080
00081
           cout << endl << endl;
00082
00083
00084
           const void* ptr[100];
00085
00086
           for(int i = 0; i < 100; i++) {</pre>
00087
               ptr[i] = (const void*)&vals + bufferSize*i;
00088
00089
00090
           int err;
00091
```

8.22 alsa\_test.cpp 41

```
for (int i = 0; i < 100; i++) {
00093
00094
                   ret = snd_pcm_writei(pcm_handle,
00095
                            ptr[i], count);
00096
00097
                   if(ret < 0) {
                       err = snd_pcm_prepare(pcm_handle);
00099
                       cout << "Preparing: " << snd_strerror(err)</pre>
00100
                            << endl;
00101
              } while (ret < 0);</pre>
00102
00103
00104
               cout << "Writing data: " << ret << endl;</pre>
00105
          }
00106 }
```

# 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
           int rate = 48000;
00029
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) {</pre>
00040
                vals[i] = (sint16)(SHRT_MAX * cos(arg * i/2) + 0.5);
00041
                vals[i+1] = vals[i];
00042
00043
           ret = snd_pcm_open(&pcm_handle, pcm_name, stream, 0);
cout << "Opening: " << snd_strerror(ret) << endl;</pre>
00044
00045
00046
00047
           ret = snd_pcm_hw_params_any(pcm_handle, hwparams);
00048
           cout << "Initializing hwparams structure: " << snd_strerror(ret) << endl;</pre>
00049
00050
           ret = snd_pcm_hw_params_set_access(pcm_handle, hwparams,
           SND_PCM_ACCESS_RW_INTERLEAVED);
cout << "Setting access: " << snd_strerror(ret) << endl;
00051
00052
00053
           00054
00055
00056
00057
00058
           ret = snd_pcm_hw_params_set_rate(pcm_handle, hwparams,
00059
                    rate, (int)0);
           cout << "Setting rate: " << snd_strerror(ret) << endl;</pre>
00060
00061
           ret = snd_pcm_hw_params_set_channels(pcm_handle, hwparams, 2);
cout << "Setting channels: " << snd_strerror(ret) << endl;</pre>
00062
00063
00064
           ret = snd_pcm_hw_params_set_periods(pcm_handle, hwparams, 2, 0);
cout << "Setting periods: " << snd_strerror(ret) << endl;</pre>
00065
00066
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;</pre>
00071
           ret = snd_pcm_hw_params(pcm_handle, hwparams);
cout << "Applying parameters: " << snd_strerror(ret) << endl;</pre>
00072
00073
00074
00075 // ret = snd_pcm_hw_params_get_period_size(hwparams, &count, 0);
           cout << "Actual period size: " << count << endl;</pre>
```

```
cout << "Returned: " << snd_strerror(ret) << endl;</pre>
00078
00079
08000
00081
           cout << endl << endl;
00082
00083
00084
           const void* ptr[100];
00085
            for(int i = 0; i < 100; i++) {</pre>
00086
                ptr[i] = (const void*)&vals + bufferSize*i;
00087
00088
00089
00090
00091
00092
           for (int i = 0; i < 100; i++) {
00093
00094
                    ret = snd_pcm_writei(pcm_handle,
00095
                             ptr[i], count);
00096
00097
                    if(ret < 0) {</pre>
                         err = snd_pcm_prepare(pcm_handle);
cout << "Preparing: " << snd_strerror(err)
00098
00099
00100
                              << endl;
00101
00102
                } while (ret < 0);</pre>
00103
00104
                cout << "Writing data: " << ret << endl;</pre>
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:
```

src/auxiliary.hpp

climits iostream stdexcept string definitions.hpp

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



complex

vector

8.24 auxiliary.hpp 43

#### **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
00010 #include <climits>
00011 #include <iostream>
00012 #include <stdexcept>
00013 #include <string>
00014
00015 #include "definitions.hpp"
00016
00017 namespace radio {
00018
         void ShowHelp() {
00022
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
             00029
00030
                 << "dsbsc\t\tDouble sideband, suppressed carrier" << std::endl
00031
                 << "lsbhil\t\tLower sideband created via Hilbert transform"
00032
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
                 << "usbfilt\t\tUpper sideband created via digital high-pass filter"
00038
                 << std::endl
00039 //
                 << "nfm\t\tFrequency modulation, 2.5 kHz bandwidth"</pre>
00040
                 << std::endl;
                 << "wfm\t\tFrequency modulation, 5 kHz bandwidth" << std::endl</pre>
00041 //
00042 //
                 << "fm\t\talias for wfm" << std::endl << std::endl;
00043
00044
             std::cerr << "MIC GAIN: Microphone power gain expressed in decibels"</pre>
00045
             << std::endl << std::endl;
00046
00047
             std::cerr << "PL TONE: Optional specification for CTCSS tone from "</pre>
                 "60-260 Hz" << std::endl << std::endl;
00048
00049
00050
             std::exit(ERROR);
00051
         }
00052
         void to_sint32(float32* data, uint32 size) {
00062
             for (uint32 i = 0; i < size; i++) {</pre>
00063
00064
                 ((sint32*)data)[i] = (sint32)(data[i] * INT_MAX + 0.5);
00065
```

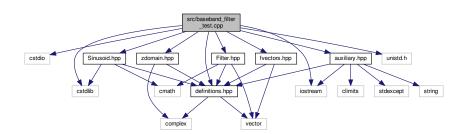
```
00066
00067
00080
           ModulationType to_type(std::string str) {
00081
                ModulationType type;
00082
                if(str == "dsblc" || str == "am") {
    type = ModulationType::DSB_LC;
00083
00084
00085
                 } else if(str == "dsbsc") {
                type = ModulationType::DSB_SC;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

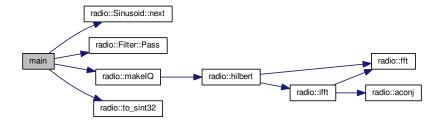
### 8.25.2.1 int main ( int argc, char \* argv[] )

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

Definition at line 25 of file baseband\_filter\_test.cpp.

```
00025
00026
00027
           // Constants
           const uint16 BUFFER_SIZE = 48000;
00028
00029
00030
           // Declare primative Variables
00031
           uint8 i = 0;
00032
           uint8 size = 0;
           uint16 delta = 250;
float32 dataBuffer[BUFFER_SIZE];
00033
00034
00035
           float32 iqBuffer[2 * BUFFER_SIZE];
00036
           // create 1 sec of audio
for(uint16 f = delta; f <= 3000; f += delta, i++) {</pre>
00037
00038
00039
                Sinusoid sinusoid(f);
00040
                for(uint16 i = 0; i < BUFFER_SIZE; i++) {
   dataBuffer[i] += sinusoid.next();</pre>
00041
00042
00043
00044
           }
00045
00046
           size = i;
00047
           // adjust dataBuffer so values are between -1 and 1
00048
00049
           for (uint16 i = 0; i < BUFFER_SIZE; i++) {</pre>
00050
                dataBuffer[i] /= size;
00051
00052
00053
           Filter filter(dataBuffer, BUFFER_SIZE, F_BASEBAND);
00054
           filter.Pass();
00055
           makeIQ(dataBuffer, iqBuffer, BUFFER_SIZE);
00056
           to_sint32(iqBuffer, 2 * BUFFER_SIZE);
00057
00058
           while(true) {
00059
               write(STDOUT FILENO, &igBuffer, 2 * BUFFER SIZE * sizeof(sint32));
00060
00061 }
```

Here is the call graph for this function:



# 8.26 baseband\_filter\_test.cpp

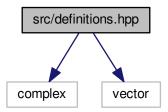
```
00001
00007 #include <cstdio>
00008 #include <cstdlib>
00009 #include <iostream>
00010 #include <unistd.h>
00011
00012 #include "auxiliary.hpp"
00013 #include "definitions.hpp"
00014 #include "Filter.hpp"
00015 #include "fvectors.hpp"
00016 #include "Sinusoid.hpp"
00017 #include "zdomain.hpp"
00019 using namespace std;
00020 using namespace radio;
00021
00025 int main(int argc, char* argv[]) {
00026
00028
          const uint16 BUFFER_SIZE = 48000;
00029
00030
           // Declare primative Variables
          uint8 i = 0;
uint8 size = 0;
00031
00032
00033
           uint16 delta = 250;
00034
           float32 dataBuffer[BUFFER_SIZE];
00035
           float32 iqBuffer[2 * BUFFER_SIZE];
00036
          // create 1 sec of audio
for(uint16 f = delta; f <= 3000; f += delta, i++) {</pre>
00037
00038
00039
               Sinusoid sinusoid(f);
00040
00041
               for(uint16 i = 0; i < BUFFER_SIZE; i++) {</pre>
00042
                    dataBuffer[i] += sinusoid.next();
00043
00044
           }
00045
00046
           size = i;
00047
00048
           // adjust dataBuffer so values are between -1 and 1 \,
           for (uint16 i = 0; i < BUFFER_SIZE; i++) {</pre>
00049
               dataBuffer[i] /= size;
00050
00051
00052
00053
           Filter filter(dataBuffer, BUFFER_SIZE, F_BASEBAND);
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 }
```

# 8.27 src/definitions.hpp File Reference

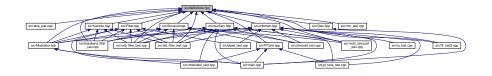
Contains declarations of system-independant (universal size) integers and float types, shortened type names for some commonly used types, and enumerations.

```
#include <complex>
#include <vector>
```

Include dependency graph for definitions.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.

### **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::F
   M\_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.

8.28 definitions.hpp 49

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;
```

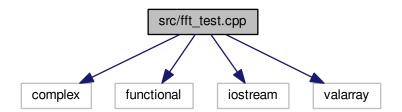
```
00024
00025 typedef unsigned int uint32;
00026 typedef signed int sint32;
00027
00028 typedef unsigned long long uint64; 00029 typedef signed long long sint64;
00031 typedef float float32;
00032 typedef double float64;
00033
00037 typedef std::complex<float32> cfloat32;
00038
00042 typedef std::vector<std::vector<float64>> fparams;
00043
00048 namespace radio {
00052
          enum Age { OLD, NEW };
00053
          enum Fractional { NUM, DEN };
00057
00058
00062
          enum Argument { FREQ = 1, MODE, PL_TONE };
00063
00067
          enum class ModulationType { DSB_LC, DSB_SC,
      USB_FILTERED, USB_HILBERT,
LSB_FILTERED, LSB_HILBERT, FM_NARROW,
00068
      FM_WIDE };
00069 }
00070
00071 #endif
00072
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::valarraystd::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.

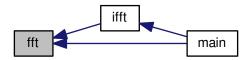
#### 8.29.3 Function Documentation

### 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.

```
00024 {
00025
           // DFT
00026
           unsigned int N = x.size(), k = N, n;
           double thetaT = 3.14159265358979323846264338328L / N;
00027
00028
           std::complex<double> phiT(cos(thetaT), sin(thetaT)), T;
00029
00030
00031
                n = k;
               n - k;
k >>= 1;
phiT = phiT * phiT;
00032
00033
00034
                T = 1.0L;
00035
                for (unsigned int 1 = 0; 1 < k; 1++)
00036
00037
                     for (unsigned int a = 1; 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
                    T *= phiT;
00044
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 \& 0xccccccc) >> 2) | ((b \& 0x33333333) << 2));
               b = (((b & 0xf0f0f0f0f0) >> 4) | ((b & 0x0f0f0f0f) << 4));
b = (((b & 0xff00ff00) >> 8) | ((b & 0x00ff00ff) << 8));
b = ((b >> 16) | (b << 16)) >> (32 - m);
00055
00056
00057
00058
                if (b > a)
00059
```

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.

```
00073 {
00074
00075
           \ensuremath{//} conjugate the complex numbers
           x = x.apply(std::conj);
00076
00077
           // forward fft
           fft(x);
00078
00079
08000
           // conjugate the complex numbers again
00081
00082
           x = x.apply(std::conj);
00083
           // scale the numbers
00084
           x /= x.size();
00085 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

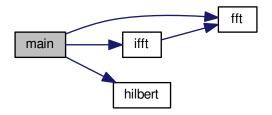


### 8.29.3.4 int main ( )

Definition at line 91 of file fft\_test.cpp.

```
00092 {
00093
           const std::complex<double> test[] = { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 };
00094
           CArray data(test, 16);
00095
00096
           // forward fft
00097
           fft (data);
00098
           std::cout << "fft" << std::endl;
for (int i = 0; i < 16; ++i)</pre>
00099
00100
00101
00102
           // std::cout << data[i] << std::endl;
00103
00104
           for(int i = 8; i < 16; i++) {
    data[i] = 0;</pre>
00105
00106
00107
00108
00109
           // inverse fft
00110
           ifft(data);
00111
00112
           std::cout << std::endl << "ifft" << std::endl;</pre>
00113
           for (int i = 0; i < 16; ++i)
00114
00115
               std::cout << data[i] << std::endl;</pre>
00116
00117
00118
           data = data.apply(hilbert);
00119
00120
           std::cout << std::endl;
00121
00122
           for (int i = 0; i < 16; i++) {
               std::cout << data[i].real() << std::endl;</pre>
00123
00124
00125
00126
           return 0;
00127 }
```

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
           unsigned int N = x.size(), k = N, n; double thetaT = 3.14159265358979323846264338328L / N;
00026
00027
00028
            std::complex<double> phiT(cos(thetaT), sin(thetaT)), T;
00029
            while (k > 1)
00030
            {
00031
                 n = k;
                k >>= 1;
phiT = phiT * phiT;
00032
00033
                 T = 1.0L;
00034
                 for (unsigned int 1 = 0; 1 < k; 1++)
00035
00036
                 {
00037
                      for (unsigned int a = 1; a < N; a += n)
00038
00039
                           unsigned int b = a + k;
                           std::complex<double> t = x[a] - x[b];
00040
00041
                          x[a] += x[b];
x[b] = t * T;
00042
00043
00044
                      T *= phiT;
00045
                }
00046
00047
            // Decimate
            unsigned int m = (unsigned int)log2(N);
00048
00049
            for (unsigned int a = 0; a < N; a++)
00050
00051
                 unsigned int b = a;
00052
                 // Reverse bits
                b = (((b & 0xaaaaaaaa) >> 1) | ((b & 0x555555555) << 1));
b = (((b & 0xccccccc) >> 2) | ((b & 0x333333333) << 2));
b = (((b & 0xf0f0f0f0) >> 4) | ((b & 0x0f0f0f0f) << 4));
00053
00054
00055
00056
                 b = (((b \& 0xff00ff00) >> 8) | ((b \& 0x00ff00ff) << 8));
```

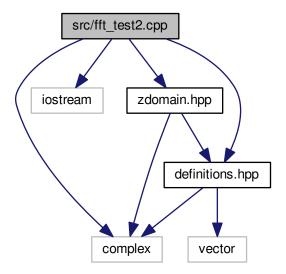
```
b = ((b >> 16) | (b << 16)) >> (32 - m);
00058
               if (b > a)
00059
00060
                   std::complex < double > t = x[a];
00061
                   x[a] = x[b];
x[b] = t;
00062
00063
00064
           //std::complex<double> f = 1.0 / sqrt(N);
//for (unsigned int i = 0; i < N; i++)</pre>
00066
00067
00068
           // x[i] *= f;
00069 }
00070
00071 // inverse fft (in-place)
00072 void ifft (CArray& x)
00073 {
00074
           // conjugate the complex numbers
00075
          x = x.apply(std::conj);
00077
           // forward fft
00078
          fft(x);
00079
          \ensuremath{//} conjugate the complex numbers again
08000
00081
          x = x.apply(std::conj);
00082
          // scale the numbers
00084
          x /= x.size();
00085 }
00086
00087 std::complex<double> hilbert(std::complex<double> n) {
00088
          return std::complex<double>(-2 * n.imag(), 0);
00089 }
00090
00091 int main()
00092 {
           const std::complex<double> test[] = { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 };
00093
00094
          CArray data(test, 16);
00096
           // forward fft
00097
          fft (data);
00098
          std::cout << "fft" << std::endl;
for (int i = 0; i < 16; ++i)</pre>
00099
00100
00101
00102
           // std::cout << data[i] << std::endl;
00103
00104
           for(int i = 8; i < 16; i++) {
00105
00106
               data[i] = 0;
00107
00108
00109
           // inverse fft
00110
           ifft(data);
00111
          std::cout << std::endl << "ifft" << std::endl;</pre>
00112
00113
           for (int i = 0; i < 16; ++i)
00114
00115
           // std::cout << data[i] << std::endl;
00116
00117
00118
          data = data.apply(hilbert);
00119
00120
          std::cout << std::endl;
00121
00122
           for (int i = 0; i < 16; i++) {
             std::cout << data[i].real() << std::endl;</pre>
00123
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.

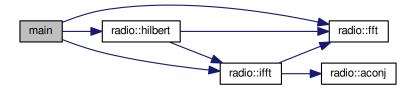
This code is based on code from http://rosettacode.org/wiki/Fast\_Fourier\_transform $\#C. \leftarrow 2B.2B.$ 

Definition at line 22 of file fft\_test2.cpp.

8.32 fft\_test2.cpp 57

```
00031
00032
            // forward fft
00033
           fft(test, 16);
00034
           std::cout << "fft" << std::endl;
00035
00036
00037
            for (int i = 0; i < 16; ++i)
00038
00039
            // std::cout << test[i] << std::endl;
00040
00041
           // inverse fft
00042
           ifft(test, 16);
std::cout << std::endl << "ifft" << std::endl;</pre>
00043
00044
00045
00046
           for (int i = 0; i < 16; ++i)
00047
00048
                std::cout << test[i] << std::endl;</pre>
00049
00050
00051
           hilbert(ftest, dest, 16);
           std::cout << std::endl << "hilbert" << std::endl;
00052
00053
           for(int i = 0; i < 16; i++) {
    std::cout << dest[i] << std::endl;</pre>
00054
00055
00056
00057
00058
           return 0;
00059 }
```

Here is the call graph for this function:



# 8.32 fft\_test2.cpp

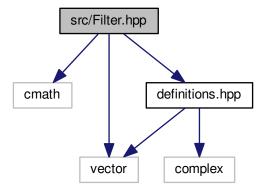
```
00001
00007 #include <complex>
00008 #include <iostream>
00009
00010 #include "definitions.hpp"
00011 #include "zdomain.hpp"
00012
00013 using namespace radio;
00014
00022 int main()
00023 {
00024
           std::complex<float32> test[] = { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 };
           float32 ftest[16];
float32 dest[16];
00025
00026
00027
           for (int i = 0; i < 16; i++) {</pre>
00028
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
            ifft(test, 16);
std::cout << std::endl << "ifft" << std::endl;</pre>
00043
00044
00045
00046
             for (int i = 0; i < 16; ++i)
00047
00048
                  std::cout << test[i] << std::endl;</pre>
00049
00050
            hilbert(ftest, dest, 16);
std::cout << std::endl << "hilbert" << std::endl;</pre>
00051
00052
00053
            for(int i = 0; i < 16; i++) {
    std::cout << dest[i] << std::endl;</pre>
00054
00055
00056
00057
00058
             return 0;
00059 }
```

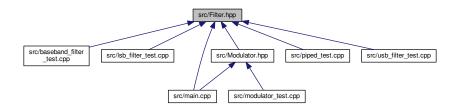
# 8.33 src/Filter.hpp File Reference

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:



8.34 Filter.hpp 59

### 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
00011 #include <cmath>
00012 #include <vector>
00013
00014 #include "definitions.hpp"
00015
00016 namespace radio {
00028 class Filter {
           public:
00029
00044
                    Filter(float32* data, uint32 size,
      fparams& diffEq);
00045
00051
                    void Pass();
00052
00053
                protected:
00058
                     uint8 eqLength;
00059
00063
                    uint32 size;
00064
00069
                    float32* data;
00070
00077
00078
                    fparams diffEq;
           };
00079
           Filter::Filter(float32* data, uint32 size,
00080
      fparams& diffEq) {
00081
              this->data = data;
00082
                this->size = size;
                this->diffEq = diffEq;
eqLength = this->diffEq[DEN].size();
00083
00084
00085
           }
00086
00087
           void Filter::Pass() {
00088
               float64 temp[size];
00089
                // create first values in filtered data
for(int i = 0; i< eqLength; i++) {
   temp[i] = 0;</pre>
00090
00091
00092
00093
                     for(int j = 0; j < eqLength; j++) {
   temp[i] += diffEq[NUM][j] * (j > i ? 0 : data[i - j]);
00094
00095
00096
00097
                     for(int j = 1; j < eqLength; j++) {
   temp[i] -= diffEq[DEN][j] * (j > i ? 0 : temp[i - j]);
00098
00099
```

```
}
00101
00102
                  \ensuremath{//} create the REST of the values in filtered data
00103
                  for(int i = eqLength; i < size; i++) {
   temp[i] = 0;</pre>
00104
00105
00106
00107
                        for(int j = 0; j < eqLength; j++) {</pre>
                           temp[i] += diffEq[NUM][j] * data[i - j];
00108
00109
00110
                        for(int j = 1; j < eqLength; j++) {
   temp[i] -= diffEq[DEN][j] * temp[i - j];</pre>
00111
00112
00113
00114
                 }
00115
             // save final values of data and filtered data
for(int i = 0; i < size; i++) {
    data[i] = temp[i];</pre>
00116
00117
00118
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:
```

src/fvectors.hpp

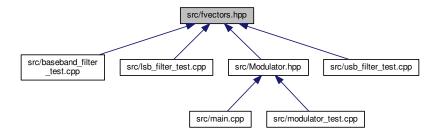
definitions.hpp

complex

vector

8.36 fvectors.hpp 61

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
00011 #include <vector>
00012
00013 #include "definitions.hpp"
00014
00015 namespace radio {
         fparams F_BASEBAND = { std::vector<float64> {
00020
             0.0008977019461,
00021
                  -0.002215694636,
                  0.001372192986,
00022
00023
                  0.001372192986,
                  -0.002215694636,
00024
00025
                  0.0008977019461
00026
         }, std::vector<float64> {
00027
              1,
                  -4.678616047,
00028
00029
                  8.822912216,
00030
                  -8.379911423,
00031
                  4.007629871,
00032
                  -0.7719064355
```

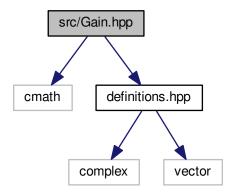
```
00033
          } };
00034
          fparams F_LOWERSIDEBAND = { std::vector<float64> {
00038
00039
              0.2758039069174,
00040
                  2.763578787693.
                  12.83915022756,
00041
00042
                  36.47584850651,
00043
                  70.37084637368,
00044
                  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,
                  27.34180552438,
00054
00055
                  60.83375457605,
00056
                  92.60908886875,
00057
                  100.8363857,
                  79.74796574736,
00058
00059
                  45.4982252145,
00060
                  18.13566776308,
00061
                   4.690036472717,
00062
                  0.6617552879305,
00063
                  0.0281427334611
00064
          } };
00065
          fparams F_UPPERSIDEBAND = { std::vector<float64> {
00069
00070
              0.001690387681463,
00071
                 0.01145271586989,
00072
                  0.03591799189724,
00073
                  0.06576926098562,
00074
                  0.07119343282702,
00075
                  0.03156377419766,
                  -0.03156377419766,
00077
                  -0.07119343282702,
00078
                  -0.06576926098562,
                  -0.03591799189724,
00079
                  -0.01145271586989,
08000
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

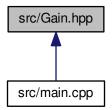
### Contains the Gain class.

```
#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

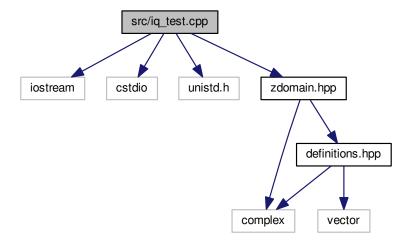
```
00001
00007 #ifndef Gain_H
00008 #define Gain_H
00009
00010 #include <cmath>
00011
00012 #include "definitions.hpp"
00014 namespace radio {
00018
        class Gain {
00019
             public:
00030
                   Gain(float32* data, uint32 size, float32 gaindB);
00031
00035
                   void Apply();
00036
00037
               private:
00041
                   float32* data;
00042
00046
                   float32 gainCoeff;
00047
00052
                   bool hasClipped = false;
00053
                    uint32 size;
00057
00058
00059
          };
00060
          Gain::Gain(float32* data, uint32 size, float32 gaindB) {
             this->data = data;
this->size = size;
00062
00063
               gainCoeff = pow(10, gaindB / 20);
00064
00065
          }
00066
          void Gain::Apply() {
    for(uint32 i = 0; i < size; i++) {
        data[i] *= gainCoeff;
}</pre>
00068
00069
00070
00071
                    if((data[i] > 1 \mid | data[i] < -1) && !hasClipped) {
                        hasClipped = true;
std::cerr << "Baseband clipping has occurred!"
00072
00073
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 <cstdio>
#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 makelQ() function.

Definition at line 20 of file iq\_test.cpp.

```
00020
             const uint16 len = 16384;
float32 data[len];
00021
00022
00023
             float32 iqData[2*len];
00024
            for(int i = 0; i < len; i++) {
   data[i] = sin(2*3.141592*170*i/len);</pre>
00025
00026
00027
00028
00029
             while(true) {
             read(STDIN_FILENO, &data, len * sizeof(float32));
00030
00031
                 makeIQ(data, iqData, len);
write(STDOUT_FILENO, &iqData, 2 * len * sizeof(float32));
00032
00033
             }
00034 }
```

Here is the call graph for this function:



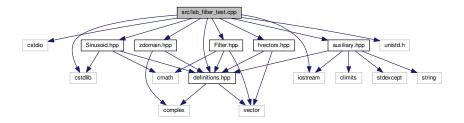
# 8.40 iq\_test.cpp

```
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() {
          const uint16 len = 16384;
float32 data[len];
00021
00022
00023
          float32 iqData[2*len];
00024
00025
          for (int i = 0; i < len; i++) {</pre>
00026
              data[i] = sin(2*3.141592*170*i/len);
00027
00028
00029
          while(true) {
              read(STDIN_FILENO, &data, len * sizeof(float32));
00030
00031
               makeIQ(data, iqData, len);
00032
               write(STDOUT_FILENO, &iqData, 2 * len * sizeof(float32));
00033
00034 }
```

# 8.41 src/lsb\_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

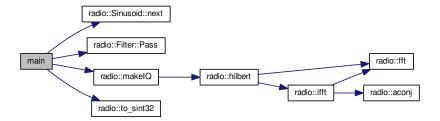
#### 8.41.1.1 int main ( int argc, char \* argv[] )

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

Definition at line 25 of file lsb\_filter\_test.cpp.

```
00025
00026
00027
           // Constants
00028
           const uint16 BUFFER_SIZE = 48000;
00029
00030
           // Declare primative Variables
00031
           uint8 i = 0;
00032
           uint8 size = 0;
           uint16 delta = 250;
float32 dataBuffer[BUFFER_SIZE];
float32 iqBuffer[2 * BUFFER_SIZE];
00033
00034
00035
00036
           // create 1 sec of audio
for(uint16 f = 17000; f <= 23000; f += delta, i++) {</pre>
00037
00038
00039
               Sinusoid sinusoid(f);
00040
               for (uint16 i = 0; i < BUFFER_SIZE; i++) {</pre>
00041
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++) {</pre>
               dataBuffer[i] /= size;
00050
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, &igBuffer, 2 * BUFFER_SIZE * sizeof(sint32));
00060
00061 }
```

Here is the call graph for this function:



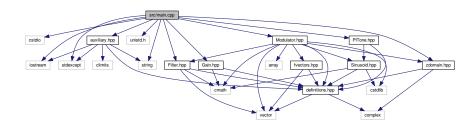
# 8.42 lsb\_filter\_test.cpp

```
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 "Filter.hpp"
00016 #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
           const uint16 BUFFER_SIZE = 48000;
00028
00029
00030
            // Declare primative Variables
00031
           uint8 i = 0;
           uint8 size = 0;
uint16 delta = 250;
float32 dataBuffer[BUFFER_SIZE];
00032
00033
00034
           float32 iqBuffer[2 * BUFFER_SIZE];
00035
00036
00037
            // create 1 sec of audio
            for(uint16 f = 17000; f <= 23000; f += delta, i++) {</pre>
00038
00039
                Sinusoid sinusoid(f);
00040
00041
                for(uint16 i = 0; i < BUFFER_SIZE; i++) {</pre>
00042
                     dataBuffer[i] += sinusoid.next();
00043
0\,0\,0\,4\,4
            }
00045
00046
           size = i;
00047
00048
            // adjust dataBuffer so values are between -1 and 1
00049
            for (uint16 i = 0; i < BUFFER_SIZE; i++) {</pre>
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, &igBuffer, 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!

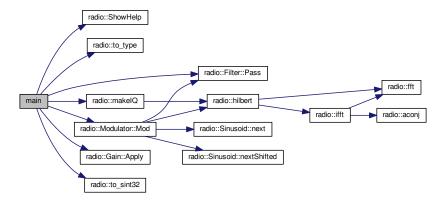
Definition at line 26 of file main.cpp.

```
00026 {
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 IO_BUFFER_SIZE = 2 * BUFFER_SIZE;
00033 const uint32 IO_BUFFER_BYTE_COUNT = BUFFER_BYTE_COUNT * 2;
00034 const uint32 SAMPLING_RATE = 48000;
```

```
00035
00036
           // Ensure 1 or 2 arguments given
           if(argc > 4) {
00037
               std::cerr << "Error: too many arguments!" << std::endl;</pre>
00038
00039
               ShowHelp();
00040
               return ERROR:
00041
           } else if(argc < 2) {</pre>
00042
               std::cerr << "Error: too few arguments!" << std::endl;</pre>
00043
               ShowHelp();
00044
               return ERROR;
00045
           }
00046
00047
           // Declare primative Variables
00048
           float32 micGain = 0;
00049
           float32 toneFreq = 0;
           float32 dataBuffer[BUFFER_SIZE];
00050
           float32 iqBuffer[IQ_BUFFER_SIZE];
00051
00052
           ModulationType type;
00053
00054
           // validate modulation type
00055
00056
               type = to_type(string(argv[1]));
           } catch(std::exception ex) {
   std::cerr << "The given modulation type is invalid!" << std::endl;</pre>
00057
00058
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
                   toneFreg = std::stof(argv[3]);
00077
00078
                    if(toneFreq < 60 || toneFreq > 260) {
00079
                        throw std::out_of_range("");
08000
                   }
00081
               } catch(std::out_of_range ex) {
                    std::cerr << "The specified CTCSS frequency is outside of the "</pre>
00082
                         "standard PL tone range." << std::endl;
00083
00084
                    ShowHelp();
00085
               } catch(std::invalid_argument ex) {
00086
                   std::cerr << "The specified CTCSS frequency is not a number."
00087
                        << std::endl;
                    ShowHelp();
00088
00089
               }
00090
           }
00091
00092
           // Declare objects
           // Decrare objects
Filter baseFilter(dataBuffer, BUFFER_SIZE, F_BASEBAND);
Gain gain(dataBuffer, BUFFER_SIZE, micGain);
PlTone pltone(0.15, dataBuffer, BUFFER_SIZE, toneFreq, SAMPLING_RATE);
Modulator modulator(dataBuffer, BUFFER_SIZE, type, 20000);
00093
00094
00095
00096
00097
00098
           // SDR guts of the program
00099
           while(true) {
00100
               // get next samples
               read(STDIN_FILENO, &dataBuffer, BUFFER_BYTE_COUNT);
00101
00102
00103
                // process/modulate samples
00104
               baseFilter.Pass();
00105 //
               pltone.Add();
00106
                gain.Apply();
00107
               modulator.Mod();
               makeIQ(dataBuffer, iqBuffer, BUFFER_SIZE);
00108
00109
               to sint32 (igBuffer, IO BUFFER SIZE);
00110
00111
00112
               write(STDOUT_FILENO, &iqBuffer, IQ_BUFFER_BYTE_COUNT);
00113
           }
00114 }
```

8.44 main.cpp 71

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;
           const uint32 BUFFER_BYTE_COUNT = BUFFER_SIZE * sizeof(sint32);
const uint32 IQ_BUFFER_SIZE = 2 * BUFFER_SIZE;
00031
00032
           const uint32 IQ_BUFFER_BYTE_COUNT = BUFFER_BYTE_COUNT * 2;
const uint32 SAMPLING_RATE = 48000;
00033
00034
00035
00036
           // Ensure 1 or 2 arguments given
00037
           if(argc > 4) {
00038
                std::cerr << "Error: too many arguments!" << std::endl;</pre>
00039
                ShowHelp();
00040
                return ERROR:
00041
           } else if(argc < 2) {</pre>
00042
               std::cerr << "Error: too few arguments!" << std::endl;
00043
                ShowHelp();
00044
                return ERROR;
00045
00046
00047
           // Declare primative Variables
           float32 micGain = 0;
00048
00049
            float32 toneFreq = 0;
00050
            float32 dataBuffer[BUFFER_SIZE];
            float32 iqBuffer[IQ_BUFFER_SIZE];
00051
           ModulationType type;
00052
00053
00054
            // validate modulation type
00055
           try{
00056
                type = to_type(string(argv[1]));
           } catch(std::exception ex) {
   std::cerr << "The given modulation type is invalid!" << std::endl;</pre>
00057
00058
00059
                ShowHelp();
00060
           }
00061
```

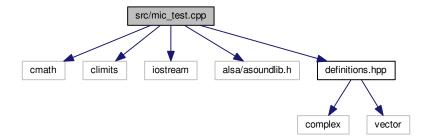
```
00062
           // process mic gain
00063
           if(argc >= 3) {
00064
00065
                  micGain = std::stof(argv[2]);
               } catch(std::invalid_argument ex) {
   std::cerr << "The specified microphone gain is not a number."</pre>
00066
00067
                        << std::endl;
00068
                   ShowHelp();
00069
00070
00071
          }
00072
00073
          // validate CTCSS tone
00074
          if(argc == 4) {
              try {
00075
00076
                   toneFreq = std::stof(argv[3]);
00077
                   if(toneFreq < 60 || toneFreq > 260) {
00078
00079
                        throw std::out_of_range("");
08000
                   }
               } catch(std::out_of_range ex) {
   std::cerr << "The specified CTCSS frequency is outside of the "</pre>
00081
00082
00083
                        "standard PL tone range." << std::endl;
                   ShowHelp();
00084
00085
               } catch(std::invalid_argument ex) {
   std::cerr << "The specified CTCSS frequency is not a number."
   << std::endl;</pre>
00086
00087
                   ShowHelp();
88000
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
               baseFilter.Pass();
00104
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
               write(STDOUT_FILENO, &iqBuffer, IQ_BUFFER_BYTE_COUNT);
00112
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.

```
00021
00022
             int ret;
00023
             snd_pcm_t* pcm_handle; // device handle
snd_pcm_stream_t stream = SND_PCM_STREAM_PLAYBACK;
snd_pcm_stream_t stream = SND_PCM_STREAM_CAPTURE;
00024
00025 //
00026
00027
             snd_pcm_hw_params_t* hwparams; // hardware information
             char* pcm_name = strdup("plughw:1,0"); // on-board audio jack
//char* pcm_name = strdup("plughw:0,0"); // on-board audio jack
00028
00029
00030
             int rate = 48000;
00031
             const uint16 freq = 440;
00032
             long unsigned int bufferSize = 8192*4;
00033
             const uint32 len = bufferSize*100;
const float32 arg = 2 * 3.141592 * freq / rate;
00034
00035
00036
             sint16 vals[len];
00037
00038
             float test;
float last = 0;
00039
00040
             long unsigned int count = 0;
00041
             int count2 = 0;
00042
             for(int i = 0; i < len; i = i + 2) {
00043
00044
                  bool lastWas = abs(sin(last)) < 0.01;
00045
00046
                  last += arg;
```

```
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
                vals[i] = (sint16)(test + 0.5);
00054
                vals[i+1] = vals[i];
00055
           }
00056
            cout << "COUNT: " << count << endl;</pre>
00057
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;</pre>
00062
           ret = snd_pcm_hw_params_any(pcm_handle, hwparams);
cout << "Initializing hwparams structure: " << snd_strerror(ret) << endl;</pre>
00063
00064
00065
00066
           ret = snd_pcm_hw_params_set_access(pcm_handle, hwparams,
           SND_PCM_ACCESS_RW_INTERLEAVED);
cout << "Setting access: " << snd_strerror(ret) << endl;
00067
00068
00069
           00070
00071
00072
            cout << "Setting format: " << snd_strerror(ret) << endl;</pre>
00073
00074
            ret = snd_pcm_hw_params_set_rate(pcm_handle, hwparams,
           rate, (int)0);
cout << "Setting rate: " << snd_strerror(ret) << endl;
00075
00076
00077
           ret = snd_pcm_hw_params_set_channels(pcm_handle, hwparams, 2);
cout << "Setting channels: " << snd_strerror(ret) << endl;</pre>
00078
00079
00080
           ret = snd_pcm_hw_params_set_periods(pcm_handle, hwparams, 2, 0);
cout << "Setting periods: " << snd_strerror(ret) << endl;</pre>
00081
00082
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;</pre>
00087
           ret = snd_pcm_hw_params(pcm_handle, hwparams);
cout << "Applying parameters: " << snd_strerror(ret) << endl;</pre>
00088
00089
00090
           ret = snd_pcm_hw_params_get_period_size(hwparams, &count, &count2);
cout << "Actual period size: " << count << endl;</pre>
00091 /*
00092
            cout << "Returned: " << snd_strerror(ret) << endl;*/</pre>
00093
00094
00095
00096
00097
           cout << endl << endl;
00098
00099
00100
            //const void* ptr = (const void*)&vals;
00101
            void* ptr = (void*)&vals;
00102
           int err;
00104
            for(int i = 0; i < 100; i++) {</pre>
00105
                     ret = snd_pcm_readi(pcm_handle,
00106
00107
                              ptr, bufferSize);
00108
00109
                     if(ret < 0) {
00110
                         err = snd_pcm_prepare(pcm_handle);
00111
                          cout << "Preparing: " << snd_strerror(err)</pre>
00112
                               << endl;
00113
00114
                } while(ret < 0);</pre>
00115
00116
                cout << "Writing data: " << ret << endl;</pre>
00117
           }
00118 }
```

# 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
```

8.46 mic test.cpp 75

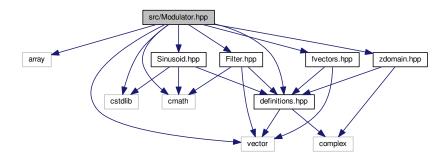
```
00015 using namespace std;
00021 int main() {
00022
          int ret;
00023
           snd_pcm_t* pcm_handle; // device handle
00024
00025 // snd_pcm_stream_t stream = SND_PCM_STREAM_PLAYBACK;
00026
           snd_pcm_stream_t stream = SND_PCM_STREAM_CAPTURE;
00027
           snd_pcm_hw_params_t* hwparams; // hardware information
           char* pcm_name = strdup("plughw:1,0"); // on-board audio jack
//char* pcm_name = strdup("plughw:0,0"); // on-board audio jack
00028
00029
00030
          int rate = 48000;
00031
00032
           const uint16 freq = 440;
00033
           long unsigned int bufferSize = 8192*4;
           const uint32 len = bufferSize*100;
const float32 arg = 2 * 3.141592 * freq / rate;
00034
00035
00036
           sint16 vals[len];
00037
00038
           float test;
00039
           float last = 0;
00040
           long unsigned int count = 0;
00041
           int count 2 = 0;
00042
00043
           for (int i = 0; i < len; i = i + 2) {
               bool lastWas = abs(sin(last)) < 0.01;
00044
00045
               last += arg;
00046
               if(last > 2 * M_PI) last -= 2 * M_PI;
00047
00048
00049
               test = 32000 * sin(last);
00050
00051
               if(abs(sin(last)) < 0.01 && lastWas) count++;</pre>
00052
               vals[i] = (sint16)(test + 0.5);
vals[i+1] = vals[i];
00053
00054
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);
cout << "Opening: " << snd_strerror(ret) << endl;</pre>
00061
00062
00063
           ret = snd_pcm_hw_params_any(pcm_handle, hwparams);
00064
           cout << "Initializing hwparams structure: " << snd_strerror(ret) << endl;</pre>
00065
          00066
00067
00068
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;</pre>
00073
00074
           ret = snd_pcm_hw_params_set_rate(pcm_handle, hwparams,
00075
                    rate, (int)0);
00076
           cout << "Setting rate: " << snd_strerror(ret) << endl;</pre>
00077
           ret = snd_pcm_hw_params_set_channels(pcm_handle, hwparams, 2);
cout << "Setting channels: " << snd_strerror(ret) << endl;</pre>
00078
00079
00080
           ret = snd_pcm_hw_params_set_periods(pcm_handle, hwparams, 2, 0);
cout << "Setting periods: " << snd_strerror(ret) << endl;</pre>
00081
00082
00083
00084
           ret = snd_pcm_hw_params_set_buffer_size_near(pcm_handle, hwparams,
00085
                    &bufferSize);
           cout << "Setting buffer size: " << snd_strerror(ret) << endl;</pre>
00086
00087
00088
           ret = snd_pcm_hw_params(pcm_handle, hwparams);
00089
           cout << "Applying parameters: " << snd_strerror(ret) << endl;</pre>
00090
00094
00095
00096
00097
           cout << endl << endl;
00098
00099
00100
           //const void* ptr = (const void*)&vals;
           void* ptr = (void*)&vals;
00101
00102
           int err;
00103
           for (int i = 0; i < 100; i++) {
00104
00105
               do {
```

```
ret = snd_pcm_readi(pcm_handle,
00107
                              ptr, bufferSize);
00108
00109
                     if(ret < 0) {
                          err = snd_pcm_prepare(pcm_handle);
cout << "Preparing: " << snd_strerror(err)</pre>
00110
00111
00112
                               << endl;
00113
00114
                } while(ret < 0);</pre>
00115
00116
                 cout << "Writing data: " << ret << endl;</pre>
00117
00118 }
```

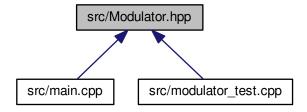
# 8.47 src/Modulator.hpp File Reference

```
#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:



8.48 Modulator.hpp 77

#### **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

```
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"
00017 #include definitions.1
00018 #include "Filter.hpp"
00019 #include "fvectors.hpp"
00020 #include "Sinusoid.hpp"
00021 #include "zdomain.hpp'
00022
00023 namespace radio {
00024
          const uint32 FREQ_INTERMEDIATE = 20000;
00028
00029
          const uint32 SAMPLING_RATE = 48000;
00034
00039
          class Modulator {
            public:
00040
                  Modulator(float32 data[], uint32 size,
00055
     ModulationType type,
                           float32 freqInter = FREQ_INTERMEDIATE,
00057
                            uint32 rate = SAMPLING_RATE);
00058
00062
                   ~Modulator();
00063
00067
                   void Mod();
00068
00069
              private:
00074
                   float32* data;
00075
                   float32 freqCarrier;
00079
00080
00081
00085
                   float32* hilData = nullptr;
00086
00090
                   float32 rate;
00091
00095
                   uint32 size;
00096
00100
                   ModulationType type;
00101
00102
         Modulator::Modulator(float32 data[], uint32 size,
00103
     ModulationType type,
float32 freqInter, uint32 rate) {
00104
               freqCarrier = freqInter;
00105
00106
               this->rate = rate;
               this->data = data;
00107
              this->size = size;
this->type = type;
00108
00109
00110
00111
               if(type == ModulationType::USB_HILBERT
00112
                       || type == ModulationType::LSB_HILBERT) {
```

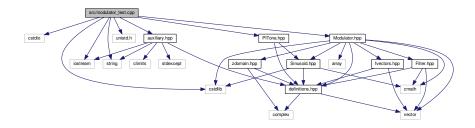
```
hilData = (float32*)malloc(size*sizeof(float32));
00114
00115
          }
00116
          Modulator::~Modulator() {
00117
            if(hilData != nullptr) free(hilData);
00118
00119
00120
          void Modulator::Mod() {
00121
00122
              // these variables should only ever be created once
              static float32 fmArg = 2 * M_PI * freqCarrier / (float32)rate;
static float32 fmK = 2 * M_PI / rate;
static float32 fmSum = 0; // cummulative sum used in FM modulation
00123
00124
00125
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
              if(type == ModulationType::USB_HILBERT
00131
00132
                       || type == ModulationType::LSB_HILBERT) {
00133
                  hilbert (data, hilData, size);
              } else if(type == ModulationType::FM_NARROW) {
00134
00135
                 fmK *= 2.5;
              } else if(type == ModulationType::FM_WIDE) {
   fmK *= 5;
00136
00137
00138
00139
00140
              \ensuremath{//} perform main modulation
00141
              for(uint32 i = 0; i < size; i++) {</pre>
00142
                  switch(type) {
00143
                      case ModulationType::DSB_LC:
00144
                          data[i] = ((data[i] + 1) * sinusoid.next()) / 2;
00145
00146
00147
                       case ModulationType::DSB_SC:
                       case ModulationType::USB_FILTERED:
case ModulationType::LSB_FILTERED:
00148
00149
00150
                          data[i] = data[i] * sinusoid.next();
00151
00152
00153
                       case ModulationType::USB_HILBERT:
                         00154
00155
00156
                           break;
00157
00158
                       case ModulationType::LSB_HILBERT:
                         00159
00160
                           break;
00161
00162
00163
                       case ModulationType::FM_NARROW:
00164
                       case ModulationType::FM_WIDE:
00165
                          fmSum += fmK * data[i];
                           data[i] = cos(fmArg * i + fmSum);
00166
00167
                           break;
00168
                  }
             }
00170
00171
              \ensuremath{//} filter out a sideband if using filtered SSB modulation
00172
              if(type == ModulationType::LSB_FILTERED) {
                  lsbFilter.Pass();
00173
              } else if(type == ModulationType::USB_FILTERED) {
00174
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[])
```

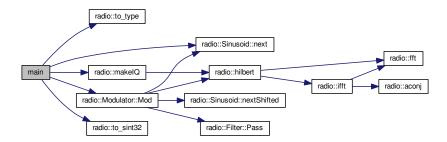
Program to test the Modulator class with a self-generated sinusoidal input.

Definition at line 24 of file modulator\_test.cpp.

```
00024
00025
00026
            // Constants
00027
            const uint16 BUFFER_SIZE = 16384;
00028
            // Declare primative Variables
00029
            float32 dataBuffer[BUFFER_SIZE];
float32 iqBuffer[2 * BUFFER_SIZE];
00030
00031
00032
            ModulationType type;
            float32 freq = atof(argv[2]);
float32 tone = 0;
00033
00034
00035
00036
            if(argc >= 4) tone = atof(argv[3]);
00037
00038
00039
                 type = to_type(string(argv[1]));
```

```
} catch(std::exception ex) {
00041
             std::cerr << ex.what() << std::endl << std::endl;</pre>
00042
              return ERROR;
00043
00044
          if(freq < 0) {
    cerr << "The given tone was invalid." << endl;</pre>
00045
00047
              return ERROR;
00048
00049
          // Declare objects
00050
00051
          Modulator modulator(dataBuffer, BUFFER_SIZE, type, 20000);
          Sinusoid sinusoid(freq);
00052
00053
          PlTone(tone > 0 ? 0.15 : 0, dataBuffer, BUFFER_SIZE, tone, 48000);
00054
00055
              for(uint16 i = 0; i < BUFFER_SIZE; i++) {
00056
00057
                  dataBuffer[i] = sinusoid.next();
00058
00059
00060
              modulator.Mod();
00061
              makeIQ(dataBuffer, iqBuffer, BUFFER_SIZE);
              to_sint32(iqBuffer, 2 * BUFFER_SIZE);
00062
              write(STDOUT_FILENO, &iqBuffer, 2 * BUFFER_SIZE * sizeof(sint32));
00063
00064
          }
00065 }
```

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"
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
           \ensuremath{//} Declare primative Variables
00029
           float32 dataBuffer[BUFFER_SIZE];
float32 iqBuffer[2 * BUFFER_SIZE];
00030
00031
00032
           ModulationType type;
           float32 freq = atof(argv[2]);
float32 tone = 0;
00033
00034
00035
00036
           if(argc >= 4) tone = atof(argv[3]);
00037
00038
00039
                type = to_type(string(argv[1]));
```

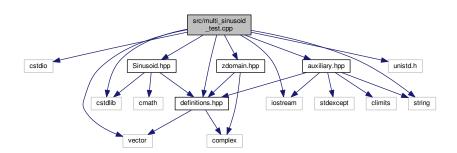
```
00040
          } catch(std::exception ex) {
00041
              std::cerr << ex.what() << std::endl << std::endl;</pre>
00042
               return ERROR;
00043
00044
          if(freq < 0) {
    cerr << "The given tone was invalid." << endl;</pre>
00045
00047
               return ERROR;
00048
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
               for (uint16 i = 0; i < BUFFER_SIZE; i++) {</pre>
00056
00057
                   dataBuffer[i] = sinusoid.next();
00058
00059
00060
               modulator.Mod();
00061
               makeIQ(dataBuffer, iqBuffer, BUFFER_SIZE);
00062
               to_sint32(iqBuffer, 2 * BUFFER_SIZE);
               write(STDOUT_FILENO, &iqBuffer, 2 * BUFFER_SIZE * sizeof(sint32));
00063
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

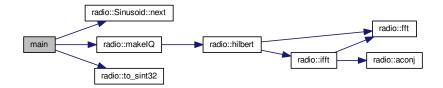
```
8.51.2.1 int main ( int argc, char * argv[] )
```

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.

```
00028
           // Constants
00029
           const uint16 BUFFER_SIZE = 48000;
00030
00031
00032
           // Declare primative Variables
00033
           uint8 i = 0;
00034
           uint8 size = 0;
           uint16 delta = 100;
float32 dataBuffer[BUFFER_SIZE];
float32 iqBuffer[2 * BUFFER_SIZE];
00035
00036
00037
00038
00039
           for (uint16 f = 100; f < 24000; f += delta, i++) {</pre>
00040
               Sinusoid sinusoid(f);
00041
               for(uint16 i = 0; i < BUFFER_SIZE; i++) {
    dataBuffer[i] += sinusoid.next();</pre>
00042
00043
00044
               }
00045
00046
               switch(f) {
                   case 500:
00047
                       delta = 1000;
00048
00049
                         f = 1000;
00050
                        break:
00051
00052
                    case 2000:
                        delta = 2000;
00053
00054
                        break;
00055
00056
           }
00057
00058
00059
           for(uint16 i = 0; i < BUFFER_SIZE; i++) {</pre>
00060
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, &igBuffer, 2 * BUFFER_SIZE * sizeof(sint32));
00069
00070 }
```

Here is the call graph for this function:



### 8.52 multi\_sinusoid\_test.cpp

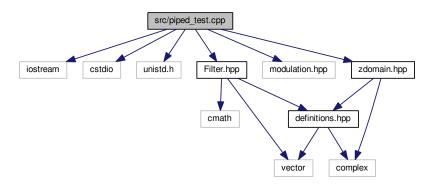
```
00001
00008 #include <cstdio>
00009 #include <cstdlib>
00010 #include <iostream>
00011 #include <string>
00012 #include <unistd.h>
00013 #include <vector>
00014
00015 #include "auxiliary.hpp"
00016 #include "definitions.hpp"
00017 #include "Sinusoid.hpp"
00018 #include "zdomain.hpp"
00019
00020 using 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 primative Variables
00033
           uint8 i = 0;
00034
           uint8 size = 0;
           uint16 delta = 100;
float32 dataBuffer[BUFFER_SIZE];
00035
00036
00037
           float32 iqBuffer[2 * BUFFER_SIZE];
00038
00039
           for (uint16 f = 100; f < 24000; f += delta, i++) {</pre>
00040
                Sinusoid sinusoid(f);
00041
00042
                for(uint16 i = 0; i < BUFFER_SIZE; i++) {</pre>
00043
                    dataBuffer[i] += sinusoid.next();
00044
00045
                switch(f) {
   case 500:
00046
00047
00048
                         delta = 1000;
00049
                         f = 1000;
                         break;
00050
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++) {</pre>
00061
                dataBuffer[i] /= size;
00062
00063
           makeIQ(dataBuffer, iqBuffer, BUFFER_SIZE);
to_sint32(iqBuffer, 2 * BUFFER_SIZE);
00064
00065
00066
00067
           while(true) {
00068
                write(STDOUT_FILENO, &iqBuffer, 2 * BUFFER_SIZE * sizeof(sint32));
00069
00070 }
```

### 8.53 src/piped\_test.cpp File Reference

containts 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

containts 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.

8.54 piped\_test.cpp 85

Here is the call graph for this function:



# 8.54 piped\_test.cpp

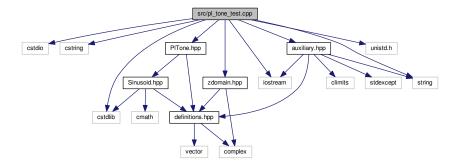
```
00001
00007 #include <iostream>
00008 #include <cstdio>
00009 #include <unistd.h>
00010
00011 #include "Filter.hpp"
00012 #include "modulation.hpp"
00013 #include "zdomain.hpp"
00014
00015 using namespace std;
00016 using namespace lolz;
00017
00022 int main() {
         const uint16 len = 16384;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

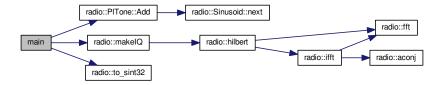
Program to test the PITone class.

Definition at line 24 of file pl\_tone\_test.cpp.

```
00024
00025
              // Constants
00026
             const uint16 BUFFER_SIZE = 16384;
00027
00028
             // Declare primative Variables
             float32 dataBuffer[BUFFER_SIZE];
float32 iqBuffer[2 * BUFFER_SIZE];
00029
00030
00031
             float32 freq = atof(argv[1]);
00032
             if(freq < 0) {
   cerr << "The given tone was invalid." << endl;
   return ERROR;</pre>
00033
00034
00035
00036
00037
00038
             PlTone tone(0.15, dataBuffer, BUFFER_SIZE, freq, 48000);
00039
00040
             while(true) {
                   for(uint16 i = 0; i < BUFFER_SIZE; i ++) {
    dataBuffer[i] = 1;</pre>
00041
00042
00043
00044
00045
                   tone.Add();
                   makeIQ(dataBuffer, iqBuffer, BUFFER_SIZE);
to_sint32(iqBuffer, 2 * BUFFER_SIZE);
write(STDOUT_FILENO, &iqBuffer, 2 * BUFFER_SIZE * sizeof(sint32));
00046
00047
00048
00049
             }
00050 }
```

8.56 pl\_tone\_test.cpp 87

Here is the call graph for this function:



# 8.56 pl\_tone\_test.cpp

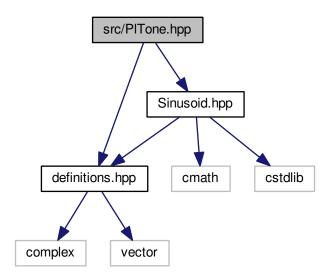
```
00001
00007 #include <cstdio>
00008 #include <cstring>
00009 #include <cstdlib>
00010 #include <iostream>
00011 #include <string>
00012 #include <unistd.h>
00013
00014 #include "auxiliary.hpp"
00015 #include "PlTone.hpp"
00016 #include "zdomain.hpp"
00017
00018 using namespace std;
00019 using namespace radio;
00020
00024 int main(int argc, char* argv[]) {
         // Constants
00025
          const uint16 BUFFER_SIZE = 16384;
00027
00028
          // Declare primative Variables
00029
          float32 dataBuffer[BUFFER_SIZE];
00030
          float32 iqBuffer[2 * BUFFER_SIZE];
00031
          float32 freq = atof(argv[1]);
00032
          if(freq < 0) {
   cerr << "The given tone was invalid." << endl;</pre>
00033
00034
00035
               return ERROR;
00036
00037
00038
          PlTone tone (0.15, dataBuffer, BUFFER_SIZE, freq, 48000);
00039
00040
00041
             for(uint16 i = 0; i < BUFFER_SIZE; i ++) {</pre>
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

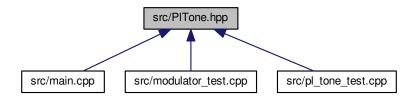
#### contains the PITone class

```
#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

8.58 PITone.hpp 89

**Author** 

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file PITone.hpp.

# 8.58 PITone.hpp

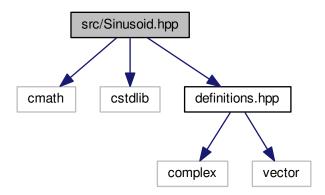
```
00001
00007 #ifndef PlTone_H
00008 #define PlTone_H
00009
00010 #include "definitions.hpp"
00011 #include "Sinusoid.hpp"
00012
00013 namespace radio {
00018 class PlTone : Sinusoid {
00019
             public:
00037
                  PlTone(float32 amplitude, float32* data, uint32 size,
00038
                             float32 frequency, uint32 samplingRate);
00039
00043
                   void Add();
00044
             private:
00045
                   float32 amplitude;
00050
00051
00055
                   float32* data;
00056
00060
                   uint32 size;
00061
           };
00062
          PlTone::PlTone(float32 amplitude, float32* data,
00063
                   uint32 size, float32 frequency, uint32 samplingRate)
00064
               : Sinusoid(frequency, samplingRate) {
00065
00066
               this->data = data;
00067
               this->amplitude = amplitude;
00068
               this->size = size;
00069
00070
               for(uint32 i = 0; i < samplingRate; i++) {
    sinusoid[i] *= amplitude;</pre>
00071
00072
00073
         }
00074
          void PlTone::Add() {
00075
            for (uint32 i = 0; i < size; i++) {
    data[i] += amplitude * next();</pre>
00076
00077
00078
                    data[i] /= (1 + amplitude); // ensures value <= 1</pre>
00079
08000
00081 }
00082
00083 #endif
```

# 8.59 src/Sinusoid.hpp File Reference

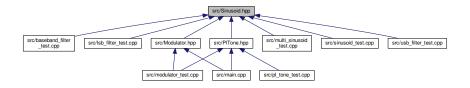
#### contains the Sinusoid class

```
#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

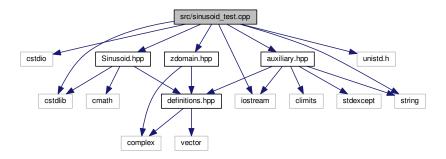
```
00007 #ifndef Sinusoid_H
00008 #define Sinusoid_H
00009
00010 #include <cmath>
00011 #include <cstdlib>
00012
00013 #include "definitions.hpp"
00014
00015 namespace radio {
      class Sinusoid {
00020
00021
           public:
                Sinusoid(float32 frequency, uint32
00025
     samplingRate = 48000);
00026
00030
                  ~Sinusoid();
00031
                 float32 next();
00036
00037
00042
                 float32 nextShifted();
00043
             protected:
00044
00048
                  float32 frequency;
00049
00053
                  uint32 sinIndex = 0;
00054
                 uint32 sinIndexShifted = 0;
00059
00063
                  uint32 samplingRate;
00064
00068
                 float32* sinusoid:
00069
                  float32* sinusoidShift90;
00075
00076
00077
          Sinusoid::Sinusoid(float32 frequency, uint32 samplingRate) {
00078
              this->frequency = frequency;
this->samplingRate = samplingRate;
00079
              sinusoid = (float32*)std::malloc(samplingRate * sizeof(
08000
     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++) {</pre>
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
          float32 Sinusoid::next() {
00098
              if(sinIndex >= samplingRate) sinIndex = 0;
00099
              return sinusoid[sinIndex++];
00100
00101
         float32 Sinusoid::nextShifted() {
00102
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[])
```

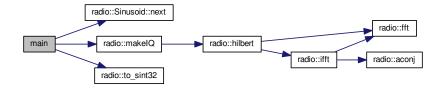
Program to test the Sinusoid class.

Definition at line 23 of file sinusoid\_test.cpp.

```
00024
00025
            // Constants
            const uint16 BUFFER_SIZE = 16384;
00026
00027
00028
            // Declare primative Variables
            float32 dataBuffer[BUFFER_SIZE];
float32 iqBuffer[2 * BUFFER_SIZE];
00029
00030
00031
            float32 freq = atof(argv[1]);
00032
            if(freq < 0) {
   cerr << "The given tone was invalid." << endl;</pre>
00033
00034
00035
                 return ERROR;
00036
```

```
00037
00038
           Sinusoid sinusoid(freq, 48000);
00039
00040
           while(true) {
               for (uint16 i = 0; i < BUFFER_SIZE; i++) {</pre>
00041
00042
                     dataBuffer[i] = sinusoid.next();
00044
00045
                makeIQ(dataBuffer, iqBuffer, BUFFER_SIZE);
                to_sint32(iqBuffer, 2 * BUFFER_SIZE);
write(STDOUT_FILENO, &iqBuffer, 2 * BUFFER_SIZE * sizeof(sint32));
00046
00047
00048
           }
00049 }
```

Here is the call graph for this function:

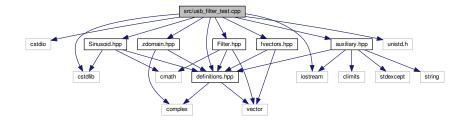


# 8.62 sinusoid\_test.cpp

```
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
           const uint16 BUFFER_SIZE = 16384;
00027
00028
           // Declare primative Variables
00029
           float32 dataBuffer[BUFFER_SIZE];
float32 iqBuffer[2 * BUFFER_SIZE];
00030
           float32 freq = atof(argv[1]);
00031
00032
           if(freq < 0) {
   cerr << "The given tone was invalid." << endl;</pre>
00033
00034
               return ERROR;
00035
00036
00037
00038
           Sinusoid sinusoid(freq, 48000);
00039
00040
               for(uint16 i = 0; i < BUFFER_SIZE; i++) {</pre>
00041
00042
                    dataBuffer[i] = sinusoid.next();
00043
00044
00045
               makeIQ(dataBuffer, iqBuffer, BUFFER_SIZE);
00046
               to_sint32(iqBuffer, 2 * BUFFER_SIZE);
               write(STDOUT_FILENO, &iqBuffer, 2 * BUFFER_SIZE * sizeof(sint32));
00047
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[] )
```

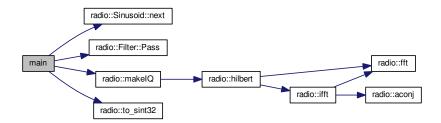
Program to test the Filter class and the USB filter coefficients.

Definition at line 25 of file usb\_filter\_test.cpp.

```
00025
00026
00027
            // Constants
00028
            const uint16 BUFFER_SIZE = 48000;
00029
            // Declare primative Variables
00030
00031
           uint8 i = 0;
uint8 size = 0;
00032
00033
            uint16 delta = 250;
00034
            float32 dataBuffer[BUFFER_SIZE];
00035
            float32 iqBuffer[2 * BUFFER_SIZE];
00036
00037
            // create 1 sec of audio
            for (uint16 f = 17000; f <= 23000; f += delta, i++) {</pre>
00038
                Sinusoid sinusoid(f);
00039
00040
00041
                 for(uint16 i = 0; i < BUFFER_SIZE; i++) {</pre>
                     dataBuffer[i] += sinusoid.next();
00042
00043
00044
            }
00045
00046
            size = i;
00047
            // adjust dataBuffer so values are between -1 and 1
for(uint16 i = 0; i < BUFFER_SIZE; i++) {
   dataBuffer[i] /= size;</pre>
00048
00049
00050
00051
00052
```

```
00053 Filter filter(dataBuffer, BUFFER_SIZE, F_UPPERSIDEBAND);
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 }
```

Here is the call graph for this function:



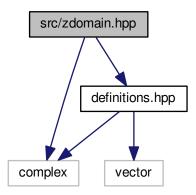
# 8.64 usb\_filter\_test.cpp

```
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 "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
           const uint16 BUFFER_SIZE = 48000;
00028
00029
00030
            // Declare primative Variables
00031
           uint8 i = 0;
00032
            uint8 size = 0;
           uint16 delta = 250;
float32 dataBuffer[BUFFER_SIZE];
00033
00034
            float32 iqBuffer[2 * BUFFER_SIZE];
00035
00036
            // create 1 sec of audio
for(uint16 f = 17000; f <= 23000; f += delta, i++) {</pre>
00037
00038
00039
                Sinusoid sinusoid(f);
00040
00041
                 for(uint16 i = 0; i < BUFFER_SIZE; i++) {</pre>
                     dataBuffer[i] += sinusoid.next();
00042
00043
00044
            }
00045
00046
            size = i;
00047
00048
            // adjust dataBuffer so values are between -1 and 1
            for(uint16 i = 0; i < BUFFER_SIZE; i++) {
   dataBuffer[i] /= size;</pre>
00049
00050
00051
00052
00053
           Filter filter(dataBuffer, BUFFER_SIZE, F_UPPERSIDEBAND);
00054
           filter.Pass();
00055
           makeIQ(dataBuffer, iqBuffer, BUFFER_SIZE);
           to_sint32(iqBuffer, 2 * BUFFER_SIZE);
```

# 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)

8.66 zdomain.hpp 97

#### 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>
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
          void hilbert(float32* data, float32* dest, uint32 size);
00052
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++)</pre>
                   data[i] = std::conj(data[i]);
00086
00087
00088
          }
00089
00090
           void fft(cfloat32* data, uint32 size) {
00091
               // DFT
00092
               uint32 k = size;
               uint32 n;
00093
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
                    for(uint32 1 = 0; 1 < k; 1++) {
   for(uint32 a = 1; a < size; a += n) {</pre>
00104
00105
00106
                          uint32 b = a + k;
00107
                             cfloat32 t = data[a] -data[b];
                            data[a] +=data[b];
data[b] = t * T;
00108
00109
00110
00111
00112
                        T \star = phiT;
00113
                   }
00114
               }
00115
               // Decimate
00116
               uint32 m = (uint32) log2 (size);
00117
00118
00119
               for(uint32 a = 0; a < size; a++) {</pre>
00120
                   uint32 b = a;
00121
00122
                    // Reverse bits
                   b = (((b \& 0xaaaaaaaaa) >> 1) | ((b \& 0x55555555) << 1));
00123
                   b = (((b & 0xccccccc) >> 2) | ((b & 0xof0f0f0f0f) << 4));
b = (((b & 0xf0f0f0f0f0) >> 4) | ((b & 0x0f0f0f0f0f) << 4));
00124
00125
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];
```

```
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
               for(int i = 0; i < size; i++) {
   temp[i] = data[i];
}</pre>
00141
00142
00143
00144
00145
               fft(temp, size);
00146
00147
               for(int i = size/2; i < size; i++) {</pre>
00148
                   temp[i] = 0;
00149
00150
00151
               ifft(temp, size);
00152
               for(int i = 0; i < size; i++) {</pre>
00153
                   // parentheses around temp prevent free() error
dest[i] = -2 * (temp[i].imag());
00154
00155
00156
               }
00157
00158
               free(temp);
00159
          }
00160
           void ifft(cfloat32* data, uint32 size) {
00161
00162
              aconj(data, size);
00163
               fft(data, size);
00164
               aconj(data, size);
00165
               for(int i = 0; i < size; i++) {
   data[i] /= size;</pre>
00166
00167
00168
00169
          }
00170
00171
          void makeIQ(float32* data, float32* dest, uint32 size) {
00172
               float32 quadData[size];
               hilbert (data, quadData, size);
00173
00174
00175
               for(int i = 0; i < 2 * size; i += 2) {</pre>
                  dest[i] = quadData[i/2];
dest[i+1] = data[i/2];
00176
00177
00178
00179
           }
00180 }
00181
00182 #endif
```

# Index

| aconj radio, 13  | NUM radio, 13   |
|--|---|
| Age radio, 12 Argument radio, 12   | OLD radio, 12 PL_TONE   |
| bin/bbftest, 35<br>bin/lsbftest, 35<br>bin/modtest, 35, 36<br>bin/msintest, 36<br>bin/pltest, 36<br>bin/radio, 36<br>bin/sintest, 37 | radio, 12  radio, 11  aconj, 13  Age, 12  Argument, 12  DEN, 13  DSB_LC, 13   |
| bin/usbftest, 37  DEN radio, 13  DSB_LC radio, 13  DSB_SC radio, 13  | DSB_SC, 13 FM_NARROW, 13 FM_WIDE, 13 FREQ, 12 fft, 14 Fractional, 12 hilbert, 15  |
| FM_NARROW radio, 13 FM_WIDE radio, 13 FREQ radio, 12 fft radio, 14 Fractional  | ifft, 16 LSB_FILTERED, 13 LSB_HILBERT, 13 MODE, 12 NEW, 12 NUM, 13 OLD, 12 PL_TONE, 12 USB_FILTERED, 13 USB_HILBERT, 13 |
| radio, 12 hilbert radio, 15  | USB_FILTERED<br>radio, 13<br>USB_HILBERT<br>radio, 13   |
| ifft radio, 16   | radio, 10   |
| LSB_FILTERED radio, 13 LSB_HILBERT radio, 13   |   |
| MODE<br>radio, 12<br>makefile, 38  |   |
| NEW radio, 12  |   |