# My Project

Generated by Doxygen 1.9.1

# Chapter 1

# Walkthrough

## 1.1 explain_sec1 Introduction

This document adds to the intergration of how to build and run the project.

## 1.2 explain_sec2 Building the project

To run the project, the cmake file is needed along with the fftw library. Currently running cmake will generate the makefile needed to build the project.

**Note**

> Currently the code is set on a loop such that it will grab 10 frames from the IQ file and turn them to hdf5 files. The files are stored under the database folder.

## 1.3 Functions explained

The IQ file reading will not be covered here since the plan is to replace the IQ file with a buffer providing active feedback through a UDP server. The app will run and capture samples when given the command from the xApp to do so.

### 1.3.1 SFFT

The SFFT function is used to transform the IQ data into the frequency domain. The function is called with the IQ data and the size of the data. The first portion of the code calculates the proper window size to maintain a resoultion of equal measure across both frequency and time domain. The frequency resolution can be given by the formula

$$\frac{f_s}{M}$$

where N is the length of the window and $f_s$ is the sample rate. Within a 5G signal it is cut into discreate chunks of SCS (Subcarrier Spacing) and since the signal is cyclic prefix it will repeat to space each signal since it using OFDM (Orthogonal Frequency Division Multiplexing). A table was made for ease of reference.

A single slot contains 14 OFDM symbols meaning that the depending on SCS the formula to determine the smallest unit in time domain is $\frac{slot_{time}}{14}$ with the inverse being the smallest possible dicreate. in the frequency domain.

The window length to make claculation faster is all powers of 2 which to save on calculation time, a LUT is created with

```
const int windowSizeLUT[7] = {64, 128, 256, 512, 1024, 2048, 4096};
```

Since the maximum frequency resolution is dependent on the bandwidth an algorigthmic method is needed to determine the

For 40 MHz

| SCS | Slot Time | Max Frequency Resolution | Max Time Resolution |
|---|---|---|---|
| 15 | 1ms | 14kHz | $\frac{1}{14k}$ |
| 30 | 0.5ms | 28Khz | $\frac{1}{28k}$ |

The max number of samples within a resource block will vary greatly depending on the SCS to avoid the max resolution in a single domain the window length should be to capture within at least 30 subframes. The window length is calculated by the following formula $\frac{f_s}{M}$ where $f_s$ is the sample rate and $M$ is the window length.

```
if (signalIndex < iq.size())
            {
                std::complex<double> windowSample = iq[signalIndex] * window[j];
                in[i * windowSize + j][0] = std::real(windowSample);
                in[i * windowSize + j][1] = std::imag(windowSample);
            }
```

This portion of the code is used to create the windowed samples to be used in the FFT. The windowed samples are then stored in the in array, which allows for the FFT to be performed. The fftw function used is

```
p = fftw_plan_dft_1d(windowSize, in, out, FFTW_FORWARD, FFTW_ESTIMATE);
```

The output of the FFT is then stored in the out array. Only half of the out array is stored to avoid redundancy. The output is then stored in the fftData array. Since we take the absolute value of the complex number, the struct contains two extra vectors which are the stft structure.

The way the data is stored within the first array as a absolute value with the power in dB which creates ease of use and reference as a comparsion for debugging purposes.

**Note**

Later editions will remove this to save on memory and go straight to the CIR

## 1.4 Running the application

The executable will convert all the named binary file from the IQ folder to the database folder as a hdf5 file. The hdf5 file is simple in structure as a 2D array with the rows and coloumns serving as the window and frequency bins respectivly.

1. Run the cmake file to generate the makefile
```
mkdir build
cd build
cmake ..
```
2. Run the makefile to build the project
3. Run the executable to convert the binary files to hdf5 files.

# Chapter 2

# Module Index

## 2.1 Modules

Here is a list of all modules:

# Chapter 3

# Class Index

## 3.1 Class List

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

# Chapter 4

# Module Documentation

## 4.1 Functions

### Functions

- void hamming (int lo, std::vector< double > &c)

    *Creates the hamming window for the STFT.*
- stft stftCalc (std::vector< double > &c, std::vector< std::complex< double >> &iq)

    *This function calculates the Short Time Fourier Transform (STFT) of the input signal.*

### 4.1.1 Detailed Description

### 4.1.2 Definitions

### 4.1.3 Function Documentation

#### 4.1.3.1 hamming()

```
void hamming (
            int lo,
            std::vector< double > & c )
```

Creates the hamming window for the STFT.

The hamming window utlizes the hamming function and applies it over a set length the function applies is thus :

$$w[n] = 0.54 - 0.46\cos(\frac{2\pi n}{(w-1)})$$

**Parameters**

| | |
|---|---|
| *w* | window size |
| *window* | reference to the window vector |

### 4.1.3.2 stftCalc()

```
stft stftCalc (
            std::vector< double > & c,
            std::vector< std::complex< double >> & iq )
```

This function calculates the Short Time Fourier Transform (STFT) of the input signal.

The function calculates the STFT of the input signal using the Fast Fourier Transform (FFT) algorithm. The function uses the FFTW library to perform the FFT calculations. Based off the formula for the stft which is

$$x[m,n] = \sum_{k=0}^{m+(N-1)} x[n]w[k]e^{-j2\pi km/N}$$

**Parameters**

| | |
|---|---|
| *window* | The window vector used for the STFT |
| *iq* | The input signal in the time domain |

**Returns**

The STFT structure containing the STFT values

# Chapter 5

# Class Documentation

## 5.1 sig Struct Reference

Short Time Fourier Transform (STFT) structure to store the STFT values.

```
#include <stored.h>
```

### 5.1.1 Detailed Description

Short Time Fourier Transform (STFT) structure to store the STFT values.

Performs the portion of storing the STFT values in a buffer.Along with the window size and the total number of windows processed.

**Parameters**

| | |
|---|---|
| *stftBuff* | Buffer to store the complex values of the STFT |
| *stftAbs* | Buffer to store the absolute values of the STFT |
| *stftCIR* | Buffer to store the independent CIR values of the STFT |
| *windowSize* | Size of the window used for STFT |
| *windowCount* | Total number of windows |
| *hopSize* | Overlap size of the windows |

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

- /home/migue/Desktop/programs/spectrogram/source/headers/stored.h

## 5.2 stft Struct Reference

### Public Attributes

- std::vector< std::vector< std::complex< double > > > **stftBuff**

- std::vector< std::vector< double > > stftAbs

    *Buffer to store the absolute values of the STFT \label stftAbs.*
- std::vector< std::vector< double > > stftCIR

    *Buffer to store the independent signal values of the STFT.*
- int **windowSize**
- int **windowCount**
- int **hopSize**

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

- /home/migue/Desktop/programs/spectrogram/source/headers/stored.h

# Index