Coursework 2

Auralisation

Sören Schreiber

K1549360

**Contents**

[**Introduction** 3](#_Toc449705238)

[**Analysis** 3](#_Toc449705239)

[**Design** 3](#_Toc449705240)

[**Implementation** 4](#_Toc449705241)

[**Auralisation** 4](#_Toc449705242)

[**Signal processing** 5](#_Toc449705243)

[**Peripheral control** 5](#_Toc449705244)

[**Testing** 5](#_Toc449705245)

# **Introduction**

The second coursework assignment poses the problem of an application which can auralise the frequency spectrum of a given signal.

The goal of the application is to specify, design and implement a C-solution using the “dsPIC StarterKit 1”, which will move away from using visual aids for displaying information and move to sound as the new delivery system.

The application will be able to identify and auralise an 8-bit signal sampled at 8kHz.

The proposed solution has been tested by means of a signal generator app on an android phone (using LINE-IN and MIC).

# **Analysis**

The assignment requires that the developed solution, which will be able to read a signal and playback the frequency spectrum as an audio signal.

This will be accomplished by transforming the original signal into the frequency spectrum, split each frequency into a separate signal and then transform each data set back into an audio signal by using an inverse FFT function.

The signal is specified as a real valued 8-bit signal. This means that the audio output will be a mirrored sound. This can be removed by playing only the first half of the audio array.

The signal is to be sampled at 8000 Hz (8 kHz). Since the signal is real valued half will be a negative frequency which is ignored in this approach. This means the maximum frequency that can be measured will be 4000Hz (4 kHz).

The application will take an audio input using the MIC or LINE-IN peripheral (jumper) and return the result as an audio output and using the provided LEDs for displaying the current action of the appliation.

# **Design**

The project is designed in an object orientated way, meaning most functions will be written as reusable header and source code files.

The first step is the implementation of an audio input reader, which allows transforming an audio signal into a data set.

To sample the audio signal, a frame size needs to be defined. This frame size will and must stay the same everywhere in the program to make sure that no overflow error occurs.

For this project the defined frame size is 16. Due to limited memory on the chip set this number gives the best resolution while complying with the restricted memory.

The audio handler data types and sizes are defined and implemented in accordance with the functions using them.

The data type for the read audio input will be an array defined as a “fractional” since the FFT requires the input to be in this particular format. It has the size equal to the selected frame size.

The auralisation also has variables for “state”. Since they contain only numeric values, the data type int has been chosen.

After the audio is sampled it is given to the FFT functions which will transform the data into the frequency domain and detect the pitch by finding the maximum of the power spectral density (PSD). It calculates the pitch by matching the result frequency pin to the value of each pin in the frequency spectrum and returning the result.

The variables given to the FFT are set to be a fractional and fractional complex. This is due to the requirements specified by the FFT functions.

For the auralisation a multidimensional array is needed to split up the FFT results into individual signals. Both array given to the function are defined as fractional complex due to the results provided by the FFT.

The last step is the inverse FFT which turns the individual signals back into and audio data format, this function requires the results to be of the type fractional while the input will be defined as fractional complex. Both are 2 dimensional arrays, as they contain multiple signals.

# **Implementation**

The implementation for this project consist out of a number of functions which are used in an object orientated structure. These function control the LEDs, input/output and DSP functions. The “Auralisation”-main will only call these functions.

The solution is implemented on a pre-existing project called “Pitch detector”, which already contains most of the functions needed for this application. This way only small changes to pre-existing functions will be necessary besides the implementation of newly required functions.

When a signal is inputted into the “audio in” peripheral, the signal is processed and the results played back before the next sample can be taken by the audio reader.

The following will explain the functionality of each separate file and explain what its purpose is.

## **Auralisation**

This is the main function of the application. It handles the input coming from and the output going back to the audio peripheral and calls functions according to its design.

The audio is read through an “ADCChannelbuffer”, which samples the input with the given frame size and transform it into a fractional data type. This data is then given to the FFT functions. The results are then played back by the “ocPWMBuffer” using the connected sound system.

## **Signal processing**

The FFT and inverse FFT are the major part of this application. They transforms the given signal into the frequency domain and vis versa. The FFT function provides a fractional complex dataset for the auralisation. Since the input signal is expected to be a real signal, the FFT will return an array of data which is set up as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | …. | N/2 | N/2+1 | … | N |
| 0 | …. | W | -w | … | -1 |

The results of the FFT are passed on to the generateAuralisation function which will separate the frequency pins into individual frequency spectrums. The input is a fractcomplex array while the output will be a 2d array which is also of the data type fractcomplex.

The last step will be the inverseFFT function. This will turn the auralised data back into playable audio data. It works opposite to the FFT.

## **Peripheral control**

The LED control focuses on the LED peripherals provided by the DSP kit. It is responsible for providing a ready state, error state and the control for individual LEDs.

The ready state is called at the start up of the application indicating that it is ready to perform its function.

The error state will only show when an error occurred within the program.

In addition the LEDs will show the progress of the auralisation by separating it into 3 steps.

Each step will turn on a different LED until all 3 are showing:

* Step 1: Red LED
* Step 2: Yellow LED
* Step 3: Green LED

After the process is finished only the yellow led will show indicating that the program is ready to continue.

During playback, the LEDs will cycle until all the information have been played. Roughly 1-2 cycles per individual signal.

# **Testing**

To test the application, a signal generator app was used to generate a sinus signal with a changeable frequency. This signal was fed into the audio input and the results were checked in debug mode, which showed that all the data is getting manipulated correctly and moved to their appropriate positions. In addition to this the sound output was used to identify changes in the audio with different frequencies.

## **Unsolved problems**

In the submitted version of the application, there are still bugs/problems that remained unsolved.

* The output of the auralisation application is seemingly not changing by much no matter what input frequency is given. In contrast to this, in debug mode the values of the output are being changed depending on the signal input.
* The volume of the output is very low.
* When the output array is cut in half to remove the mirrored part the playback stays silent.