**KINGSTON UNIVERSITY.**



**MSC. EMBEDDED SYSTEMS.**

**Digital Signal Processing.**

**COURSEWORK 1:**

**Pitch Detector.**

**Repository:** <https://drive.google.com/open?id=0B8TTFXVZqgFSWElESGdwdm9lV2s>

1. **INTRODUCTION**

This application has been designed to fulfill a simple task. Receive an input signal and through seer comparison get the pitch at every given moment.

**Code and demo:** <https://drive.google.com/open?id=0B8TTFXVZqgFSWElESGdwdm9lV2s>

1. **ANALYSIS**

The application has been optimized for working under a specific context. Given an input signal based on a sound emission, the application will determine the pitch of this signal. The user can understand the output with a visual representation; this will be achieved through LED lights. Depending on which LED is active, the highest frequency will be located in an interval or another. The original idea is to split the frequency spectrum in three different intervals: low, medium and high frequency.

To create the previous application we require a programmable device. Since we are interested in using a portable machine we will be using a particular Peripheral Interface Controller; additionally, Kingston´s University has kindly provided us with a DSPic Starter Kit. This programmable peripheral can be accessed using the MPLAB software and modified to accomplish different works.

We will learn to use this device by doing some research in the previously created libraries, Source\_Tree in particular. This will help understanding how to access the PIC´s hardware such as LEDs, input and output. Secondly we will use this software as the skeleton of the application, taking many functions that will provide different individual functionalities.

Our application will take any audio input through its respective entrance and will reflect the results using the LEDs included in the board.

.

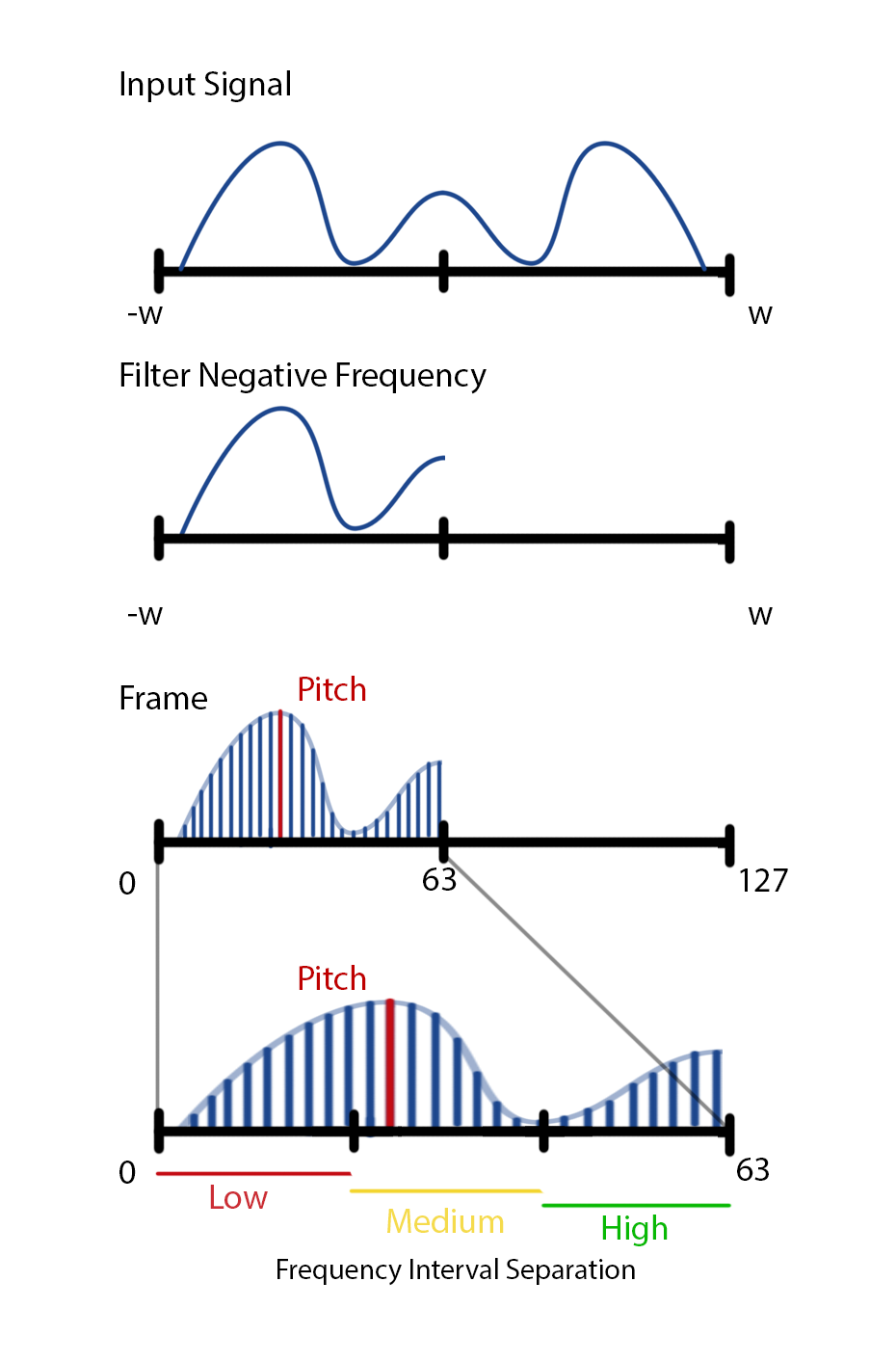
1. **DESIGN**

After a little research we have gathered some understanding of the process we need to create. First we will initialize a couple of buffers (the handlers), one will be for storing the input and another will be part of a future update to store an audio output. We gain access to these elements with the peripheral´s libraries “ADCChannelDrv” and “OCPWMDrv”.

After we get the data we have to proceed and transform the signal into the frequency domain. This will be done by using the Complex Fourier Transform. For this purpose there are also libraries in the Source Tree.

As a final step we will use another library, sacks, to communicate with the LEDs. Lighting these lights is an easy matter, they respond to a logical input, one for ON and zero for OFF.

These are the steps to make after researching about the project, now we narrate the implementation process in the upcoming section.



1. **IMPLEMENTATION: PROBLEM SOLVING AND FURTHER DISCUSSION**

**Starting the Project, external FFT.**

In order to avoid the initial problems and safe some time, we have chosen to start from an already made program. In this case we have elected to use one called “AudioAjuster” which was a filtering and processing a voice input application. Futhermore, it already contained its own implementation of the FFT, which was one of the issues foreseeing during design. Additionally this project provided us with the inputs and outputs declarations along with the length of the frame we will be working with.

As a note, we kept the frame length as 128 which is the number of samples in which we will split the audio from the input. This seemed like a fair number although we end up using just half of it as it is explain in the next step.

**Separating the Negative Frequency**

One unnecessary issue is to work with the whole frequency spectrum, since the positive and negative part are the same. We can filter one of the parts but we will be restricting the working frequency from an interval between 0 and 8000Hz to a reduced range between 0 and 4000Hz. This implies that half our frame of samples is actually not being used. It is important to avoid reading the whole length to avoid affecting the real time performance.

**Separating Frequency Interval**

Since we want to split the bandwidth into three different regions, we must decide where to set the decisive thresholds. This decision must be done according to the required demands. For this project we decided to divide the samples into more less equal sample spaces. Remember we are not using the whole set of samples but rather half of it (64 instead of 128). The low frequency will cover from 0 to 20. The medium range will take from 20 to 40. And high will take 40 and upper.

According to our calculations we have split the spectrum into the following intervals:

* **LOW:** 0 – 1250 Hz
* **MEDIUM:** 1250 - 2500 Hz
* **HIGH:** 2500 Hz and higher

Finally a call to the corresponding LED will be done, lighting a single light when the frequency is located in an interval.