

Costa Rica Institute of Technology

Electronics Engineering School



Karlsruhe Institute of Technology

Chair for Embedded Systems



# **Evaluation of Feature Extraction Techniques for an Internet of Things Electroencephalogram**

Thesis in fulfillment of the requirements to obtain the academic degree of  
Licentiate in Electronics Engineering

David Barahona Pereira

Karlsruhe. November 7, 2016



**INSTITUTO TECNOLÓGICO DE COSTA RICA**

**ESCUELA DE INGENIERÍA ELECTRÓNICA**

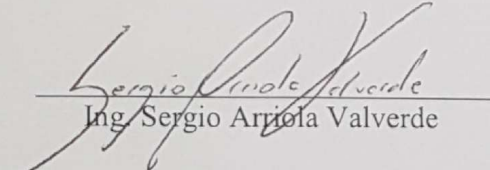
**PROYECTO DE GRADUACIÓN**

**ACTA DE APROBACIÓN**

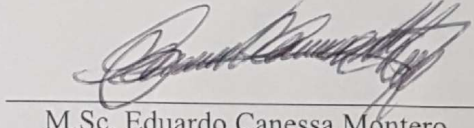
**Defensa de Proyecto de Graduación  
Requisito para optar por el título de Ingeniero en Electrónica  
Grado Académico de Licenciatura  
Instituto Tecnológico de Costa Rica**

El Tribunal Evaluador aprueba la defensa del proyecto de graduación denominado "Evaluation of Feature Extraction Techniques for an Internet of Things Electroencephalogram", realizado por el señor David Barahona Pereira y, hace constar que cumple con las normas establecidas por la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica.

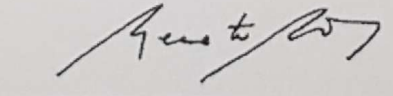
Miembros del Tribunal Evaluador

  
Ing. Sergio Arriola Valverde

Profesor lector

  
M.Sc. Eduardo Canessa Montero

Profesor lector

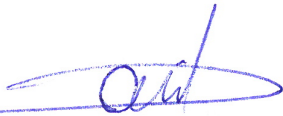
  
Dr. Renato Rímolo Donadio

Profesor asesor

Cartago, 16 de Noviembre, 2016



I declare that this thesis document has been made entirely by my person, using and applying literature on the subject, and introducing my own knowledge and experimental results. In the cases I have used literature, I proceeded to indicate the sources by the respective references. Accordingly, I assume full responsibility for this thesis work and the content of this document.



David Barahona Pereira

ID: 3 0474 0703

Karlsruhe. November 7, 2016



# Abstract

The emerging paradigm of Internet of Things (IoT) is revolutionizing our life with the introduction of new services and the improvement of existing applications. IoT is covering an ever-increasing number of applications in different domains including healthcare. One specific application in personal healthcare is the monitoring of the electrical activity in the brain using Electroencephalogram (EEG) with portable IoT devices. Due to portability and size constraints, most IoT devices are battery-powered which calls for energy-efficient implementation in both hardware and software along with an efficient use of the often limited resources.

This work evaluates three different feature extraction techniques for an IoT EEG in terms of execution time, memory usage and power consumption. The techniques under study were explored and simulated leading to select FIR, Welch's method and DWT as the ones to be evaluated. The techniques were implemented on a MSP432P401R LaunchPad platform, where an evaluation procedure was developed to assess the code performance. The implementations were validated against simulated references and also optimized for speed, code size and power consumption. The result of the performed evaluation provides a valuable comparison between the techniques which can help any designer in choosing the right technique based on design objectives and resource constraints.

**Keywords:** IoT, EEG, feature extraction, FIR, Welch's method, DWT.





*to my mother...*



# Acknowledgments

First, I would like to thank my thesis advisor Farzad Samie at the Karlsruhe Institute of Technology (KIT) for letting me be part of his research and for being always attentive and supportive during the development of the project. Also, I would like to thank my thesis advisor Renato Rímolo at the Costa Rica Institute of Technology (ITCR) for all his recommendations and for always being pending of my progress despite the distance.

I would like to thank all the people that in one way or another made possible for me to develop my project not only in Germany but also in such a remarkable institution as the KIT. I want to thank my friend Moises Araya and my teacher Jorge Castro for helping me find a project but specially for all their support, trust and recommendations. Also, I would like to thank the ITCR for promoting this kind of experiences and specially to Gustavo Rojas from the Construction Engineering School for all his help.

Thanks to all my friends who have accompanied me until this point, specially to Esteban and Adolfo who more than friends have become brothers to me. After all these years of working together I can be pretty sure that we make a great team together, thank you for all those endless nights of study and projects but specially for always being there.

Last but not least, I must express my very profound gratitude to my family. I want to thank them for providing me with unconditional love, unfailing support and continuous encouragement throughout all my life and years of study. Thank you for being always my everyday motivation. I would especially like to thank my mother for all her love and dedication throughout all these years. Thank you for being my best friend and the person I admire the most, there are no words that can describe how much I love you and how thankful I feel. This accomplishment would not have been possible without you.

David Barahona Pereira

Karlsruhe. November 7, 2016



# Contents

<b>List of Figures</b>	<b>iii</b>
<b>List of Tables</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Document Structure . . . . .	3
<b>2 Feature Extraction Techniques in EEG</b>	<b>5</b>
2.1 EEG and Applications . . . . .	5
2.2 Signal Processing Path . . . . .	7
2.2.1 Preprocessing . . . . .	8
2.2.2 Feature Extraction . . . . .	8
2.2.3 Classification . . . . .	8
2.3 Feature Extraction Techniques . . . . .	8
2.3.1 Filter Based Feature Extraction . . . . .	9
2.3.2 FFT Based Feature Extraction . . . . .	10
2.3.3 DWT Based Feature Extraction . . . . .	12
2.3.4 Feature Extraction Techniques Overview . . . . .	15
<b>3 Simulation and Exploration of Feature Extraction Techniques</b>	<b>17</b>
3.1 Test Signal . . . . .	17
3.2 Filter Based Techniques . . . . .	18
3.2.1 FIR . . . . .	18
3.2.2 IIR . . . . .	19
3.2.3 Comparison . . . . .	20
3.3 FFT Based Techniques . . . . .	21
3.3.1 Periodogram . . . . .	21
3.3.2 Welch's Method . . . . .	21
3.3.3 Comparison . . . . .	23
3.4 DWT Based Techniques . . . . .	23
3.4.1 DWT . . . . .	25
3.4.2 WPT . . . . .	25
3.4.3 Comparison . . . . .	27
3.5 Comparison between techniques . . . . .	27

---

<b>4</b>	<b>Hardware Platform and Measurement Procedure</b>	<b>31</b>
4.1	Platform Overview . . . . .	31
4.2	Measurement Procedure . . . . .	32
4.2.1	Execution Time . . . . .	32
4.2.2	Memory Usage . . . . .	33
4.2.3	Power Consumption . . . . .	34
<b>5</b>	<b>Implementation of Feature Extraction Techniques</b>	<b>39</b>
5.1	CMSIS DSP Software Library . . . . .	39
5.2	Microcontroller Implementation . . . . .	41
5.2.1	FIR Feature Extraction . . . . .	41
5.2.2	Welch's Method Feature Extraction . . . . .	43
5.2.3	DWT Feature Extraction . . . . .	46
5.3	Optimizations . . . . .	50
5.3.1	Execution Time and Memory Usage . . . . .	50
5.3.2	Power Consumption . . . . .	52
<b>6</b>	<b>Evaluation of Feature Extraction Techniques</b>	<b>55</b>
6.1	Execution Time . . . . .	55
6.2	Memory Usage . . . . .	56
6.3	Power Consumption . . . . .	58
6.4	Overall Evaluation . . . . .	60
<b>7</b>	<b>Conclusions</b>	<b>61</b>
	<b>Bibliography</b>	<b>63</b>
<b>A</b>	<b>Acronyms and Abbreviations</b>	<b>67</b>
<b>B</b>	<b>Feature Extraction Codes</b>	<b>69</b>
B.1	FIR . . . . .	69
B.2	Welch's method . . . . .	72
B.3	DWT . . . . .	75

# List of Figures

1.1	Stages of the project . . . . .	2
2.1	EEG device . . . . .	6
2.2	EEG frequency bands . . . . .	6
2.3	EEG signal processing path . . . . .	7
2.4	Welch's method procedure . . . . .	12
2.5	Single level DWT . . . . .	13
2.6	Multilevel DWT . . . . .	13
2.7	WPT decomposition . . . . .	14
3.1	EEG test signal . . . . .	17
3.2	FIR filter response for the Alpha band . . . . .	18
3.3	Power for the Alpha band using FIR filter . . . . .	19
3.4	IIR filter response for the Alpha band . . . . .	19
3.5	Power for the Alpha band using IIR filter . . . . .	20
3.6	Comparison of FIR and IIR . . . . .	20
3.7	Periodogram on the EEG signal . . . . .	22
3.8	Welch's method on the EEG signal . . . . .	22
3.9	Comparison of periodogram and Welch's method . . . . .	23
3.10	Low-pass decomposition filter response for DWT . . . . .	24
3.11	High-pass decomposition filter response for DWT . . . . .	24
3.12	DWT decomposition of the EEG signal . . . . .	25
3.13	WPT decomposition of the EEG signal . . . . .	26
3.14	Comparison of DWT and WPT . . . . .	27
3.15	Comparison of filter and DWT based techniques . . . . .	28
3.16	Comparison of filter, FFT and DWT based techniques . . . . .	28
4.1	CCS Breakpoints tab . . . . .	32
4.2	CCS Memory Allocation tab . . . . .	34
4.3	Profile tab in ET mode . . . . .	35
4.4	Power tab in ET mode . . . . .	36
4.5	Energy tab in ET mode . . . . .	36
4.6	Profile tab in ET+ mode . . . . .	37
4.7	States tab in ET+ mode . . . . .	37

---

5.1	FIR feature extraction algorithm sequence . . . . .	41
5.2	Simulated and microcontroller comparison for FIR . . . . .	42
5.3	Validation error for FIR . . . . .	42
5.4	Welch's method feature extraction algorithm sequence . . . . .	43
5.5	Rectangular window . . . . .	44
5.6	Simulated and microcontroller comparison for Welch's method . . . . .	45
5.7	Validation error for Welch's method . . . . .	45
5.8	DWT method feature extraction algorithm sequence . . . . .	46
5.9	Bookkeeping vector structure for DWT . . . . .	47
5.10	Rearrangement of coefficients process for DWT . . . . .	48
5.11	Simulated and microcontroller comparison for DWT . . . . .	48
5.12	Implemented bookkeeping vector for DWT . . . . .	49
5.13	Validation error for DWT . . . . .	49
5.14	Compiler optimization view . . . . .	50
5.15	Code size for different optimization level settings . . . . .	51
5.16	Code size for different code size and speed trade-off settings . . . . .	51
5.17	Power measurements for optimization . . . . .	53
5.18	Energy measurements for optimization . . . . .	54
6.1	Execution time evaluation . . . . .	56
6.2	Flash memory usage evaluation . . . . .	57
6.3	SRAM memory usage evaluation . . . . .	57
6.4	Power consumption evaluation . . . . .	59
6.5	Energy consumption evaluation . . . . .	59



# List of Tables

2.1	Cognitive states related to EEG frequency bands . . . . .	7
2.2	Feature extraction techniques comparison . . . . .	16
3.1	Bands obtained using DWT . . . . .	25
3.2	Bands obtained using WPT . . . . .	26
5.1	Validation percentage error for FIR . . . . .	43
5.2	Validation percentage error for Welch's method . . . . .	45
5.3	Validation percentage error for DWT . . . . .	49
5.4	Compiler optimization settings . . . . .	51
5.5	Speed optimization . . . . .	52
5.6	Code size optimization . . . . .	52
5.7	Power measurements for optimization . . . . .	53
5.8	Energy measurements for optimization . . . . .	54
6.1	Execution time evaluation . . . . .	55
6.2	Memory usage evaluation . . . . .	57
6.3	Power consumption evaluation . . . . .	58
6.4	Overall evaluation of feature extraction techniques . . . . .	60



# Chapter 1

## Introduction

The Internet of Things for Healthcare group of the Chair for Embedded Systems (CES) at the Karlsruhe Institute of Technology (KIT) is devoted to the research of solutions and novel techniques to address challenges in Internet of Things (IoT) and particularly, wearable healthcare monitoring systems. Wearable healthcare monitoring systems can be used to monitor patients who are out of the hospital, they can measure daily or sport activities, and they can even keep track of sleep patterns and stress levels. Part of the current research activities at Internet of Things for Healthcare group is the development of a wearable IoT Electroencephalogram (EEG) prototype.

An EEG is a device that is able to measure and record the electrical activity of the brain. EEG is commonly found in medical applications, however its use has been extended to other fields such as Brain Computer Interfaces (BCIs) in tasks like controlling a robotic arm by imagining hand movements [27]. An IoT EEG is intended to be a portable and hence battery-powered device, therefore it is important to find the best way to face inherent challenges in computation capability and energy capacity. One of the stages in the operation of an EEG is feature extraction, this stage uses several signal processing techniques to get relevant information about the brain activity usually by measuring power in different frequency bands. Since there are many techniques that can be used in feature extraction, it is important to compare them in order to develop a criteria about which can be better suited to face the previously mentioned challenges.

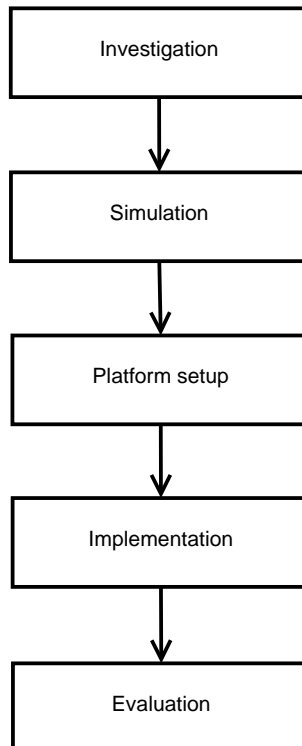
This work evaluates different feature extraction techniques from perspectives such as execution time, memory usage and power consumption in a microcontroller platform. In order to achieve that goal, the process depicted in Figure 1.1 can be followed. There are mainly three different approaches that can be used for feature extraction such as filtering, Fast Fourier Transform (FFT) and Discrete Wavelet Transform (DWT). However, there are plenty of feature extraction techniques that use one or even combinations of those approaches. Therefore, the first stage of the solution involves an extensive investigation about which are the most relevant techniques that use either filters, FFT or DWT for feature extraction, as well as their characteristics and possible optimization schemes.

The simulation stage uses the findings of the previous investigation to explore how each of the techniques under study behaves in the presence of a real EEG signal. The goal is to consider the characteristics of each technique and the results from the simulations to select three techniques for implementation, one using filters, one using FFT and one using DWT. A comparison between techniques is performed in order to make sure that they all provide similar information and therefore work correctly. The simulation results will also work as a reference to validate the implementations.

The selected platform for the implementation is a MSP432P401R LaunchPad from Texas Instruments which incorporates a microcontroller designed for low-power applications. In the platform setup stage, the goal is to find a procedure to measure the parameters that will be evaluated. A procedure to measure execution time, power consumption and memory usage is developed in this stage.

The implementation stage is the core of the solution and it consists in implementing each selected technique in the microcontroller platform. Here, the correct behavior of each technique needs to be validated and also since the main approach is to face computation and power challenges, possible optimizations must be carried out as well.

In the final stage, execution time, power consumption and memory usage are measured using the procedure developed in the setup stage. The results are documented and serve to compare each technique in order to evaluate how they fit in the final design of the device.



**Figure 1.1:** Stages of the project

## 1.1 Document Structure

This document is organized as follows:

Chapter 2 reviews EEG fundamentals, the commonly used signal processing path and delves into feature extraction techniques. Theoretical background for filters, FFT and DWT is provided and also techniques that use those approaches are explained emphasizing the most relevant characteristics, advantages and disadvantages.

Chapter 3 provides simulations for the techniques under study in order to show how they perform in the presence of a real EEG signal. It also takes those results and the ones of the investigation stage to decide which techniques are chosen for implementation.

Chapter 4 provides a brief introduction of the low-power microcontroller platform under use and its characteristics. It also states the procedure to follow for measuring execution time, power consumption and memory usage.

Chapter 5 shows the approach taken for each implementation along with results that validate that they behave as intended. Optimizations carried out for each of the metrics under test are also explained in this chapter.

Chapter 6 provides the results of the measurements for each technique and makes an analysis to evaluate the implementations and form a criteria about how each technique fits in the design of an IoT EEG.

Chapter 7 concludes the document and provides suggestions for future work.



# Chapter 2

## Feature Extraction Techniques in EEG

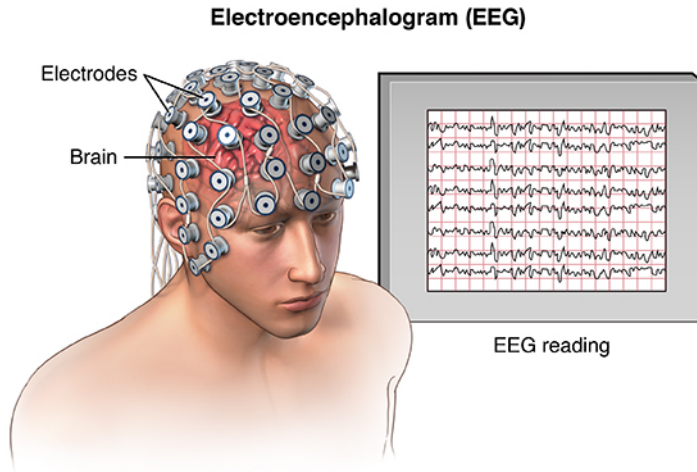
In order to understand feature extraction techniques it is important to set a background of what is an EEG and how it works. This chapter provides some basic notions of what is an EEG and which is the signal processing path commonly used to deal with brain activity measurements. It provides not only a theoretical background for filters, FFT and DWT approaches but also discusses the most common techniques that makes use of those approaches. Finally, it makes a comparison of the techniques by exploring characteristics, advantages and disadvantages.

### 2.1 EEG and Applications

Human body imaging techniques play a crucial role in modern medicine. Electrobiological measurements can be divided as electrocardiography (ECG), electromyography (EMG), electroencephalography (EEG), magnetoencephalography (MEG), electrogastrography (EGG) and electrooculography (EOG).

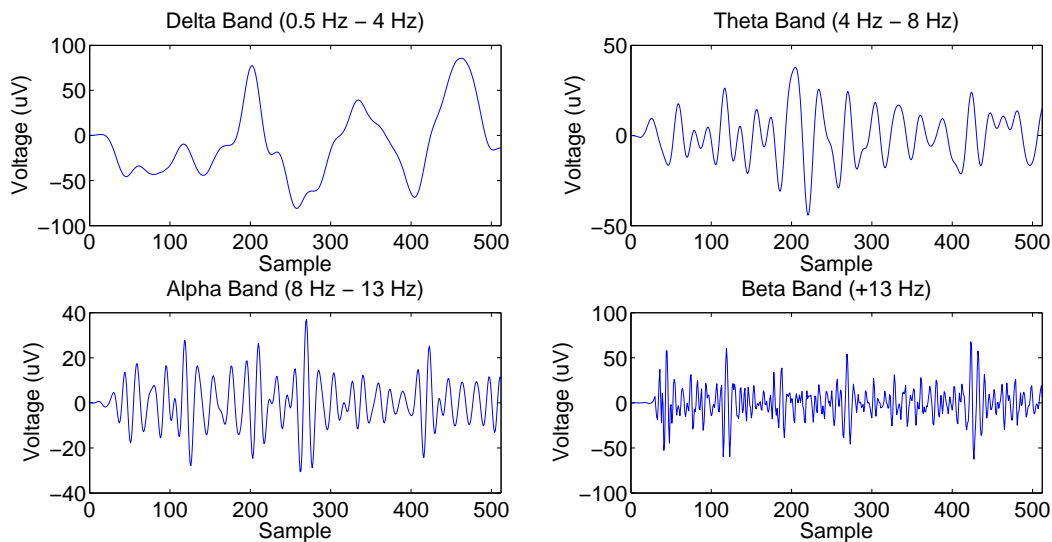
An EEG is a test device used to evaluate the electrical activity in the brain. Since brain cells communicate with each other through electrical impulses, EEG can be used to detect potential problems associated with this activity. The test is typically noninvasive, it tracks and records brain wave patterns through a set of electrodes attached to the scalp with wires as depicted in Figure 2.1.

Measurements given by an EEG can be used to rule out various conditions, including seizure disorders (such as epilepsy), a head injury, encephalitis (an inflammation of the brain), a brain tumor, encephalopathy (a disease that causes brain dysfunction), memory problems, sleep disorders, stroke, dementia and many others. The use of this device has been extended to other fields such as social interaction [4], marketing [5], psychology [28] or BCIs [27] [33].



**Figure 2.1:** EEG device. *Source: [www.saintlukeshhealthsystem.org](http://www.saintlukeshhealthsystem.org)*

Measurements taken from an EEG consist of an electrical wave that varies in time, much like a sound signal or a vibration. As such, it contains frequency components that can be measured and then analyzed, these frequency components have interesting and valuable properties. As shown in Figure 2.2, brain waves have been categorized according with their frequency range into four basic groups known as: Delta, Theta, Alpha and Beta.



**Figure 2.2:** EEG frequency bands obtained by band-pass filtering an EEG signal



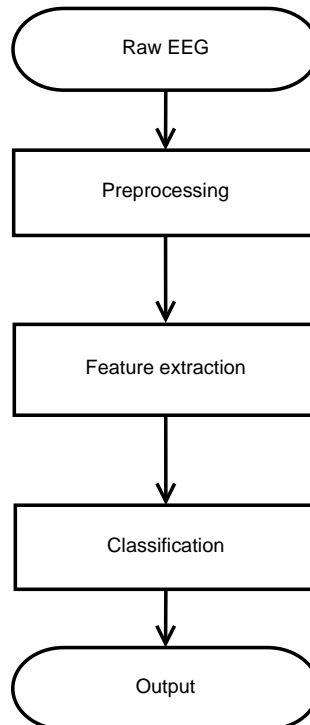
Most applications generally focus on the spectral content of EEG, that is, the type of neural oscillations that can be observed in EEG signals [7]. Table 2.1 shows the frequency range of each brain wave and also the specific brain activity associated with each band.

**Table 2.1:** Cognitive states related to EEG frequency bands

Band	Frequency range (Hz)	Brain activity
Delta	0.5-4	Deepest meditation and dreamless sleep
Theta	4-8	Light sleep
Alpha	8-13	Relaxation
Beta	+13	Consciousness

## 2.2 Signal Processing Path

Most modern applications follow a common path for EEG signal processing as depicted in Figure 2.3. Raw EEG signals go to a preprocessing stage mostly to deal with artifacts and noise. Then, relevant features about the brain activity are extracted and finally those features are classified to determine a mental state.



**Figure 2.3:** EEG signal processing path

### 2.2.1 Preprocessing

The preprocessing stage can include the acquisition of the signal, removal of artifacts, averaging, thresholding of the output, enhancement of the resulting signal, and finally, edge detection. The most critical step in this stage and in many others signal processing applications is the removal of artifacts. There are many sources of artifacts in recording raw EEG signals [15]. They can be defined as disturbances that may occur during the signal acquisition and that can alter the analysis of signals themselves. Useful features of the original signal can be severely affected if noise is not properly treated. Some sources of artifacts can be muscular activities, blinking of eyes during the signal acquisition procedure and power line electrical noise.

### 2.2.2 Feature Extraction

It is difficult to extract useful information from EEG signals just by observing them in the time domain. Therefore, there are many advanced signal processing techniques that can be used to extract relevant features from those signals [6] [12]. The choice of a particular technique is usually tied to the application under study and specific requirements.

Feature extraction aims at describing relevant information about the brain activity by an ideally small number of relevant values. All extracted features are usually arranged into a vector, known as a feature vector, which is used later for the brain activity classification. There are three main sources of information that can be extracted from EEG readings: spatial information (for multichannel EEG), spectral information (power in frequency bands) and temporal information (time windows based analysis) [9].

### 2.2.3 Classification

The last stage is denoted as classification and it consists of assigning a class to a feature vector corresponding to the mental state. Just as for the feature extraction stage, there are many classification methods that may suit a specific implementation better than others. In [31], a comparison of performance for different classification methods for EEG based BCIs is proposed.

## 2.3 Feature Extraction Techniques

Filtering, FFT and DWT are commonly used approaches in feature extraction. These approaches are discussed including a theoretical background, advantages, disadvantages as well as possible variations.

### 2.3.1 Filter Based Feature Extraction

Filters are devices that allow some signal frequencies applied at their input terminals to pass through to their output terminals with little or no reduction in the signal level. There are analog and digital implementations of filters and the use of one or another will depend of the nature of the signals in the system whether there are continuous or digital signals. Digital filters are systems commonly used in signal processing to deal with discrete time signals. They often consist of an analog to digital converter, a processing stage and a digital to analog converter.

One popular way to compute band power features from an EEG signal is to use band-pass filters to extract each desired band to later estimate energy. The process to follow is to first take an interval of the signal, e.g. 250 ms [23], then band-pass the EEG signal to each band of interest, square each sample and finally average the signal over several consecutive samples. This approach is commonly found in BCI research, e.g. in OpenViBE software [26] this is the method performed by default.

There are two types of digital filters, each with advantages and disadvantages: the Finite Impulse Response (FIR) filters and the Infinite Impulse Response (IIR) filters [32]. FIR filters have no feedback so their impulse response is of finite duration because it settles to 0 after some time. For an FIR filter of order  $N$ , the output sequence consists of a weighted sum of past input values:

$$y[n] = \sum_{i=0}^N b_i x[n-i] \quad (2.1)$$

where:

$y[n]$  = output signal

$x[n]$  = input signal

$N$  = filter order

$b_i$  = filter coefficients

FIR filters are not complex and they are used in applications in which phase characteristics are very important. These filters are always stable and show a linear phase response. They exhibit a delay that could be critical in certain applications and also they usually have a high order since they use a lot of inputs to calculate the output. A sharp fall-off can be attained if a high order filter is used.

In contrast to FIR filters, IIR filters have feedback and their impulse response does not become exactly 0 past a certain point, but continues indefinitely. The output depends not only of past input values but also from past output values:

$$y[n] = \sum_{i=0}^N b_i x[n-i] + \sum_{j=0}^M a_j y[n-j] \quad (2.2)$$

where:

$y[n]$  = output signal

$x[n]$  = input signal

$N$  = feedforward filter order

$M$  = feedback filter order

$b_i$  = feedforward filter coefficients

$a_i$  = feedback filter coefficients

IIR filters are more complex and are well suited in applications where a sharp fall-off or phase is not critical but when flat pass-bands and stop-bands are important. They exhibit a smaller delay than FIR and use less elements to calculate the output. However, they show a non-linear phase response and can become unstable. For both types of filters, the desired response is given by the set of coefficients that multiply each past input or output.

### 2.3.2 FFT Based Feature Extraction

Fourier analysis is a popular signal processing approach to go from time domain signals to frequency domain signals or vice versa. This analysis can be applied to both continuous and discrete time signals. It relays on the principle that every signal can be represented or approximated by sums of trigonometric functions [22]. FFT is an algorithm that computes the Discrete Fourier Transform (DFT) of a sequence, or its inverse. It produces the exact same result as evaluating the DFT definition directly with the difference that is much faster.

The DFT is defined by the formula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi i k n}{N}} \quad k=0, \dots, N-1 \quad (2.3)$$

where:

$X_k$  = DFT of  $x_n$

$x_n$  = input sequence

$N$  = elements in input sequence

FFT is commonly used in EEG to estimate Power Spectral Density (PSD). PSD refers to the spectral energy distribution that would be found per unit frequency. It can be computed by applying FFT directly on the signal or also by transforming the estimated autocorrelation sequence. In [20] a comparison of PSD estimation methods for EEG is shown. Among the techniques that use FFT for feature extraction, the periodogram and Welch's method are two of the most popular and commonly exploited.

The easiest approach to compute PSD is the periodogram. It consists of a frequency decomposition and is given by the modulus squared of the Fourier transform of the signal:

$$S(f) = \frac{\Delta t}{N} \left| \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi i k n}{N}} \right|^2 \quad (2.4)$$

where:

$S(f)$  = PSD of  $x_n$

$\Delta t$  = space between samples

$x_n$  = input sequence

$N$  = elements in input sequence

Given that EEG signals are usually finite and non-periodic, the periodogram can produce spurious artifacts in undesired bands. One workaround to deal with this issue is to apply a window function on the whole signal considering that there is some information loss as the window would smoothly damp the signal to zero on each end [6].

On the other hand, Welch's method is an improvement on the standard periodogram [10]. It is based on the use of overlapping windows to the signal in which a periodogram is calculated for each window and then those periodograms are averaged between them to compute PSD. This procedure is depicted in Figure 2.4. This method reduces variance and hence noise in the estimated power spectra in exchange for reducing the frequency resolution [1].

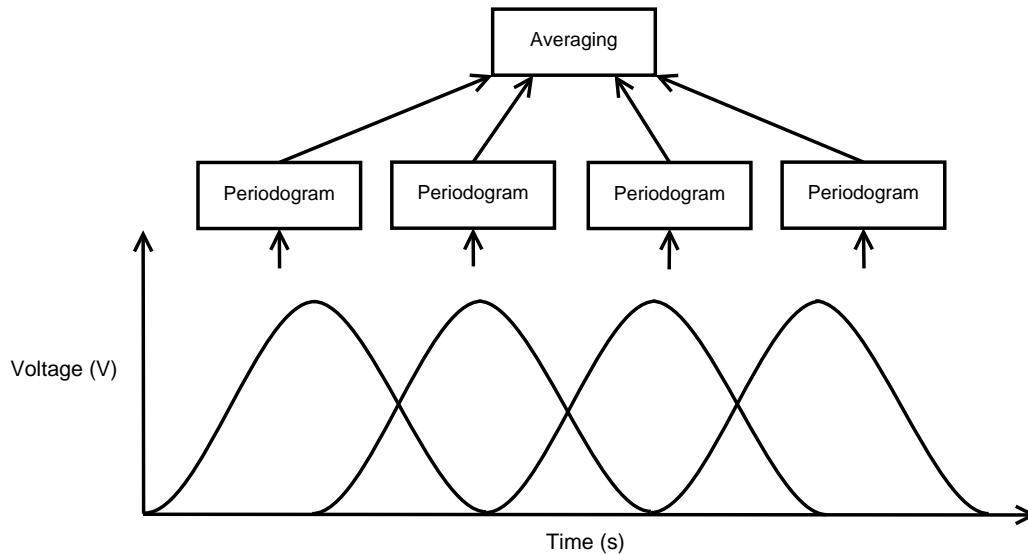


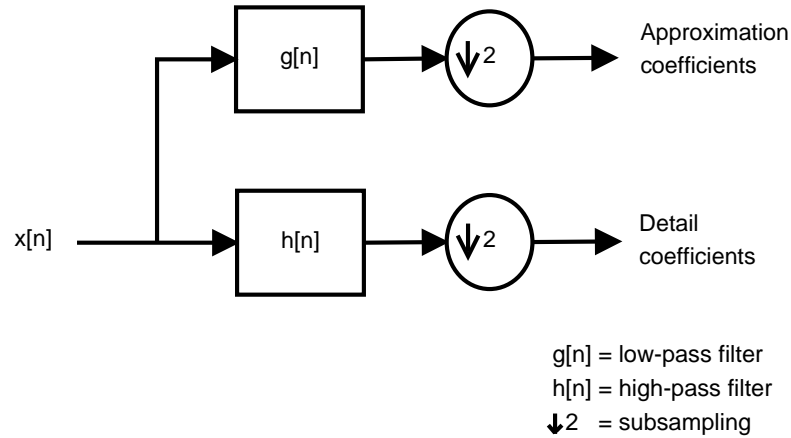
Figure 2.4: Welch's method procedure

### 2.3.3 DWT Based Feature Extraction

A wavelet is mathematical function that shows an oscillation with an amplitude that begins at 0, increases, and then decreases back to 0. Those functions have interesting features that make them useful for signal processing tasks. A wavelet transform is just a representation of a function by a certain orthonormal series generated by a wavelet [2].

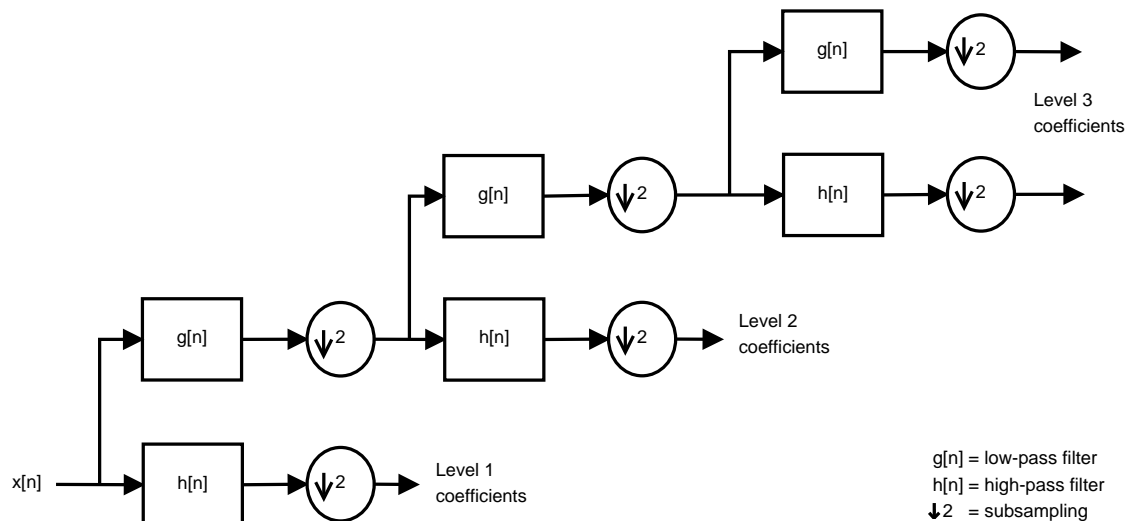
Wavelet approach is very common in EEG signal analysis [17] [18] [25]. This analysis aims to decompose EEG signals into a certain level and then sub-band energies contained at the last or previous levels to use them as features. The level of decomposition is selected based on the dominant frequency components of the signal. That means that certain frequency ranges for the sub-bands can be achieved depending on the sampling rate. Several mother wavelets can be used [2], however Daubechies 4 wavelet (db4) is a popular choice [19] [21] due to its near optimal time-frequency localization properties and smoothing features that make suitable detecting changes on EEG signals. Wavelet analysis can be performed using DWT or Wavelet Packet Transform (WPT).

DWT is a decomposition that uses discrete sampled wavelets and is usually implemented using filters. Figure 2.5 shows a single level DWT implementation. The signal goes through a high-pass filter and a low-pass filter that are related as quadrature mirror filters. The high-pass filter provides details coefficients and the low-pass filter provides approximation coefficients. After filtering, since half of the frequencies have been removed, a subsampling is applied to reduce the amount of data because only half of the samples are needed to represent the new signal according to Nyquist's sampling theorem [22].



**Figure 2.5:** Single level DWT decomposition of a signal

There are also multilevel approaches that aim a better frequency resolution as depicted in Figure 2.6. In these implementations another DWT is applied to the approximation coefficients [34].

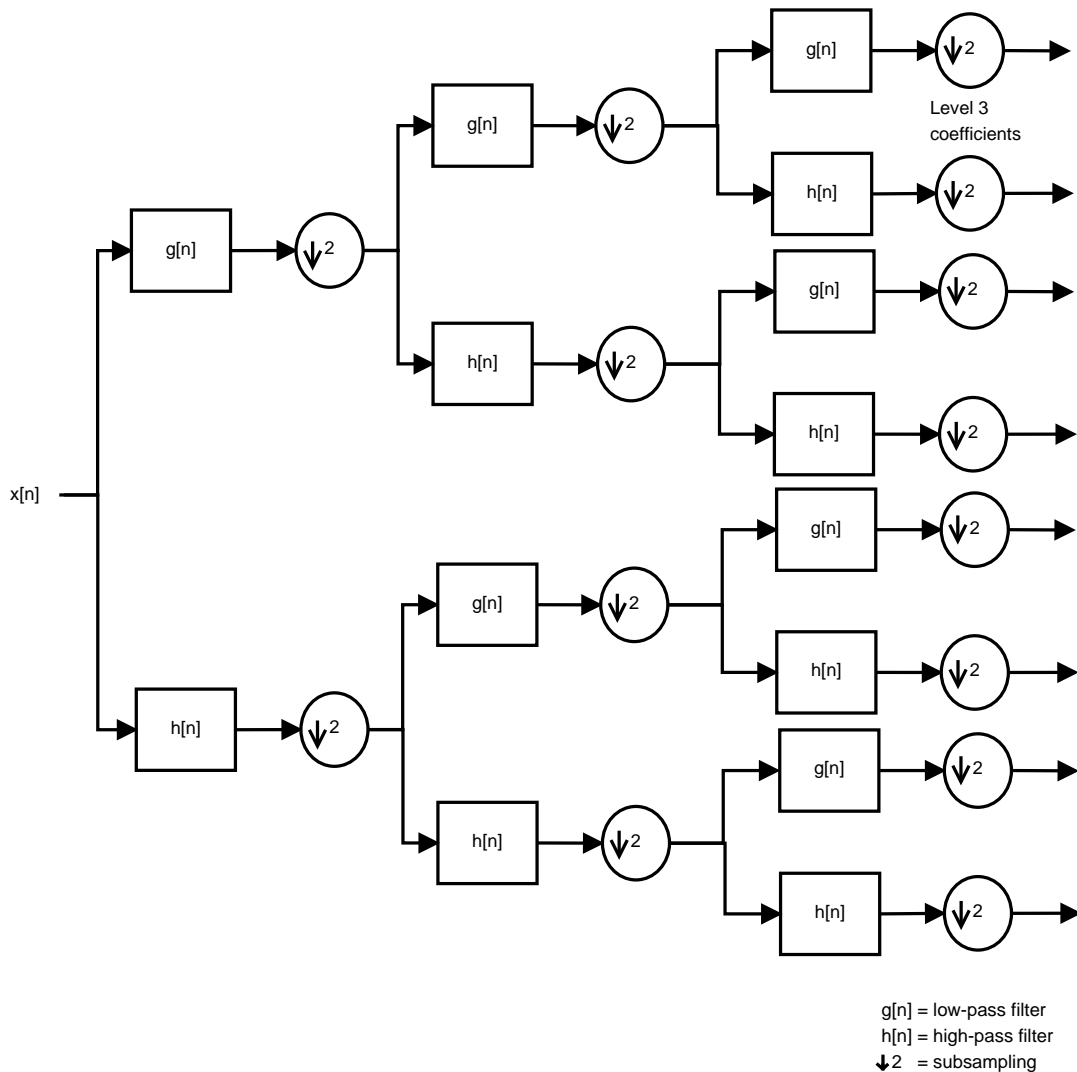


**Figure 2.6:** A multilevel DWT decomposition of a signal

Since DWT is a time-frequency domain method, it is suitable to study non-stationary signals like the ones obtained from EEG. Time domain and frequency domain methods like filters and FFT approaches are not usually well suited to handle this type of signals.

Wavelets have some slight benefits over Fourier transforms in reducing computations when examining specific frequencies. This method uses varying window sizes so it does not suffer the time-frequency resolution trade-off inherent to other time-frequency approaches [35].

WPT is a variation of the classic DWT that uses more filters in the implementation. The difference with the simple DWT is that this approach computes a DWT not only in the approximation coefficients but also in the details coefficients for each level, building a binary tree. A WPT decomposition is shown in Figure 2.7.



**Figure 2.7:** A multilevel WPT decomposition of a signal

WPT offers the possibility of decomposing the signal into frequency bands of various sizes, which is useful in EEG to select bands for individual subjects. This method improves frequency resolution in exchange of increasing complexity when compared to DWT. For simple applications it can provide more information than is actually needed. Sometimes the decomposed sub-bands frequencies might not be the best, therefore automated and adaptive WPT approaches have been studied [34] for complex and modern systems.



### 2.3.4 Feature Extraction Techniques Overview

The performed analysis can lead to have a better picture of how filters, FFT and DWT can be used as feature extraction techniques and how they possess certain characteristics that can be exploited depending on a specific application.

Filtering analysis is performed in the time domain and may not be the best to deal with non-stationary signals. The performance of a filter is tied its order and the type of filter. FIR filters are fairly simple, always stable and show a linear phase response, however they have a delay that could be critical in certain applications and also they are not very efficient since they use a lot of inputs to calculate the output. On the other side, IIR filters are more efficient since they use less inputs to calculate the output and therefore they have only a small delay, however they are more complex, they can become unstable and they modify the output because of the non-linearity of the phase response.

FFT analysis is performed in the frequency domain and just like filtering is not well suited for non-stationary signals. The periodogram is a low complex mechanism of computing PSD with a good frequency resolution, however it comes to a cost of producing spurious artifacts that could add a lot of noise to the signal. Welch's method is a variation of the periodogram that manages to reduce noise in the computation while sacrificing frequency resolution.

Finally, DWT is a technique that performs in both time and frequency domain and is well suited for non-stationary signals. There must be a careful selection of the mother wavelet function and decomposition level. DWT is less complex and has less frequency resolution than the WPT that can achieve a better decomposition and therefore a better resolution by using more filters but adding complexity to the design.

Table 2.2 summarizes characteristics, the main advantages and disadvantages of each of the techniques under study.

**Table 2.2:** Feature extraction techniques comparison

Technique	Characteristics	Variation	Advantages	Disadvantages
Filtering	Time domain analysis	FIR	Low complexity	Not very efficient
			Stability	Large delay
			Linear phase response	
	Not well suited for non-stationary signals	IIR	Efficient	High complexity
			Small delay	May become unstable
				Non-linear phase response
FFT	Frequency domain analysis	Periodogram	Low complexity	Can produce spurious artifacts
			Good frequency resolution	
	Not well suited for non-stationary signals	Welch's method	Noise reduction	Sacrifices frequency resolution
				Medium complexity
DWT	Time-frequency domain analysis	DWT	Low complexity	Variable frequency resolution
	Well suited for non-stationary signals			
	Needs to select a mother wavelet	WPT	Good frequency resolution	High complexity

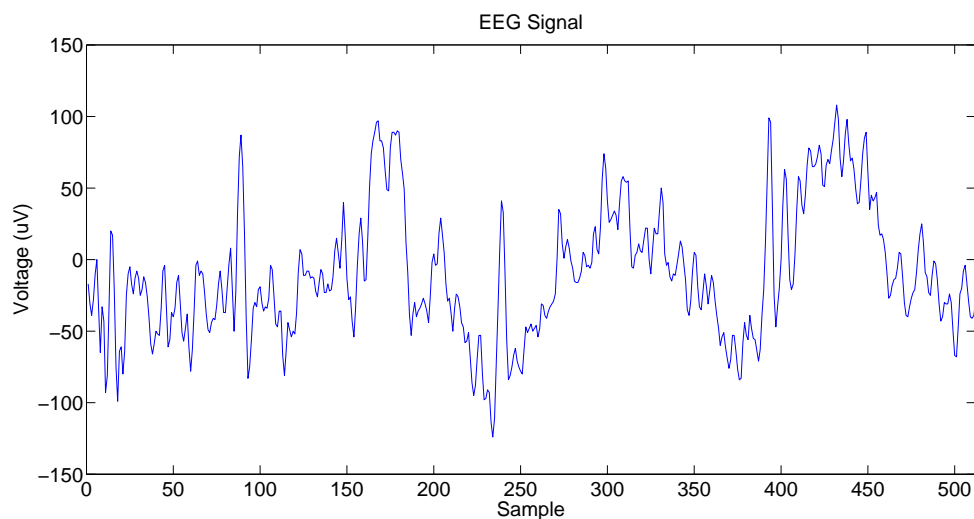
# Chapter 3

## Simulation and Exploration of Feature Extraction Techniques

In this chapter a set of MATLAB [16] simulations are presented in order to explore the behavior of the techniques studied in Chapter 2. For each technique two approaches were individually tested and then compared to select one to be implemented and evaluated in a microcontroller. Even though each technique provides information about the bands in different manners, they are compared in order to verify that they all provide similar and correct information. Lastly, the results of the simulations will be used to later validate the implementations. Every power or power density measurement for the frequency bands is normalized to a  $1 \Omega$  resistor for simplicity.

### 3.1 Test Signal

The signal in Figure 3.1 was selected in order get more representative simulations.



**Figure 3.1:** EEG test signal

The signal was taken from a dataset created and contributed to PhysioNet [11] by the developers of the BCI2000 [29] [30] instrumentation system that was used to make these recordings. The experimental procedure for the measurements can be consulted in [24]. The selected input signal was sampled at 160 Hz, with 512 samples and hence a duration of 3.2 s.

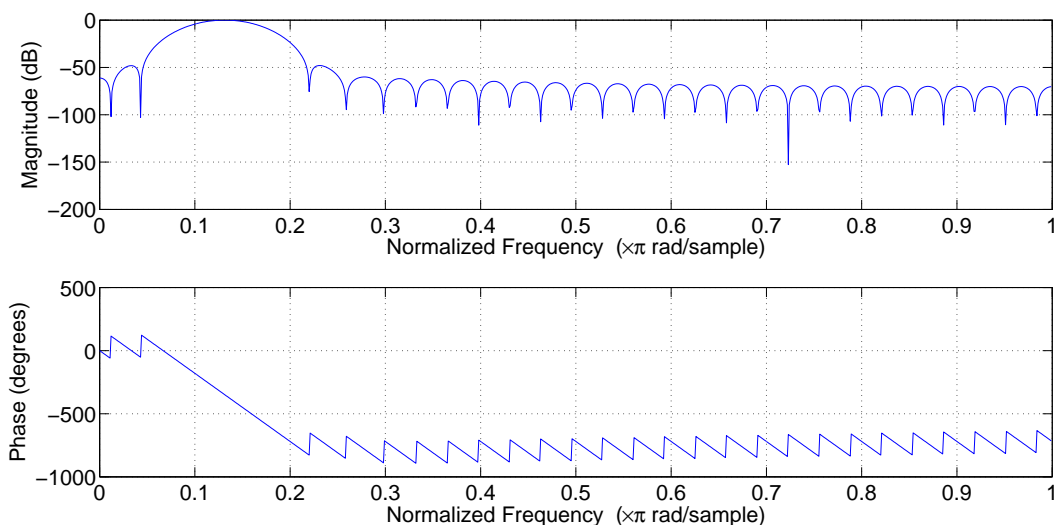
## 3.2 Filter Based Techniques

FIR and IIR approaches are individually simulated and then compared. The procedure to use filters as feature extraction technique is to pass the EEG signal through a set of band-pass filters to later compute power and finally to average it over time to reduce noise. The result is a time domain representation of power for each band.

Each filter must be designed separately since all of them have different cut-off frequencies depending on the band of interest. The order of the filter must be selected in order to get a good trade-off between complexity and performance. The criteria to design the filters was to get the lowest order that ensured an attenuation equal or less than 3 dB in the pass-band. Power was computed just by squaring the voltage reading and a moving average filter was applied to the power signal in order to reduce noise.

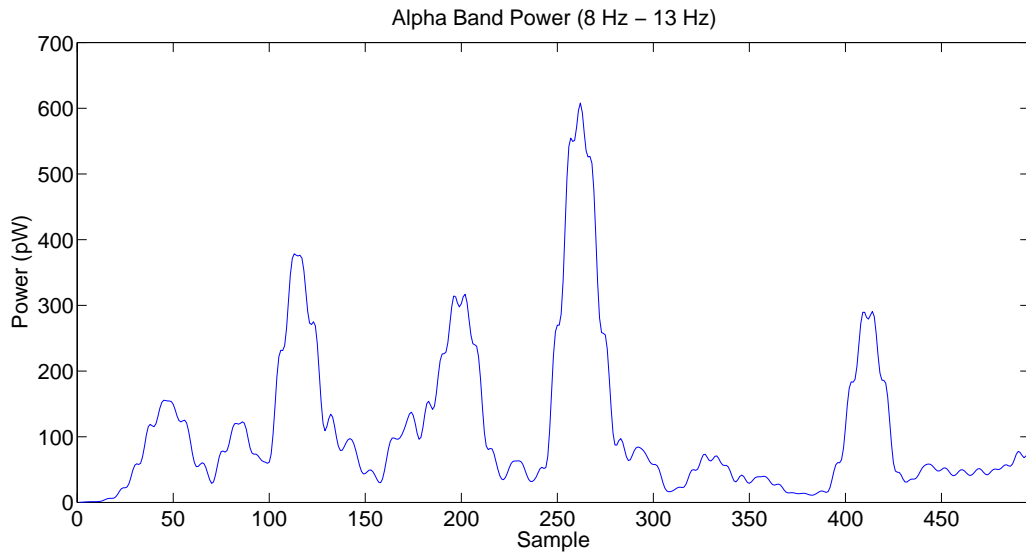
### 3.2.1 FIR

The designed FIR filter is of order 60, so it has 61 taps. The magnitude and phase response for the filter designed for the Alpha band can be observed in Figure 3.2, the same approach was taken with the other bands.



**Figure 3.2:** FIR filter response for the Alpha band

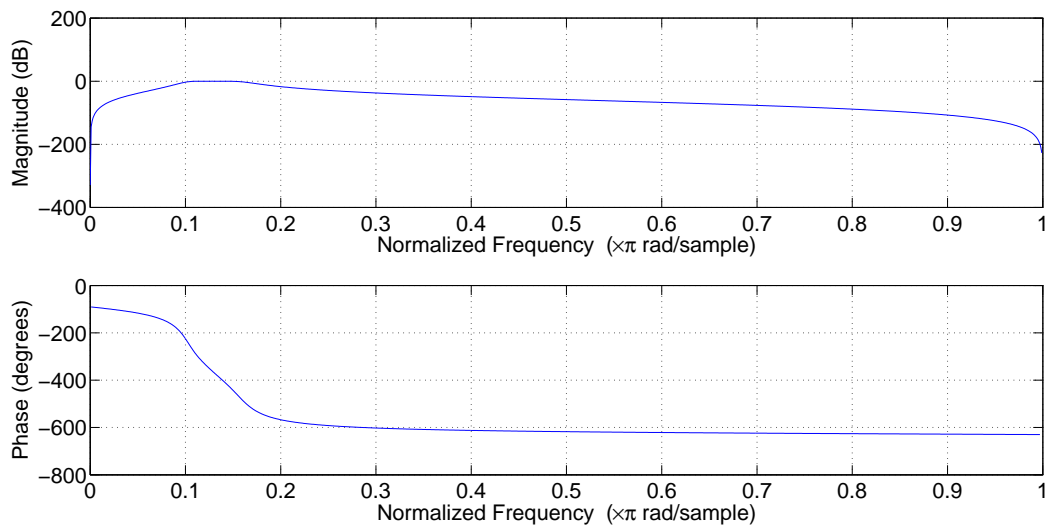
After passing the input signal through the filter, power is computed and averaged over time to get the power reading shown in Figure 3.3



**Figure 3.3:** Power for the Alpha band using FIR filter

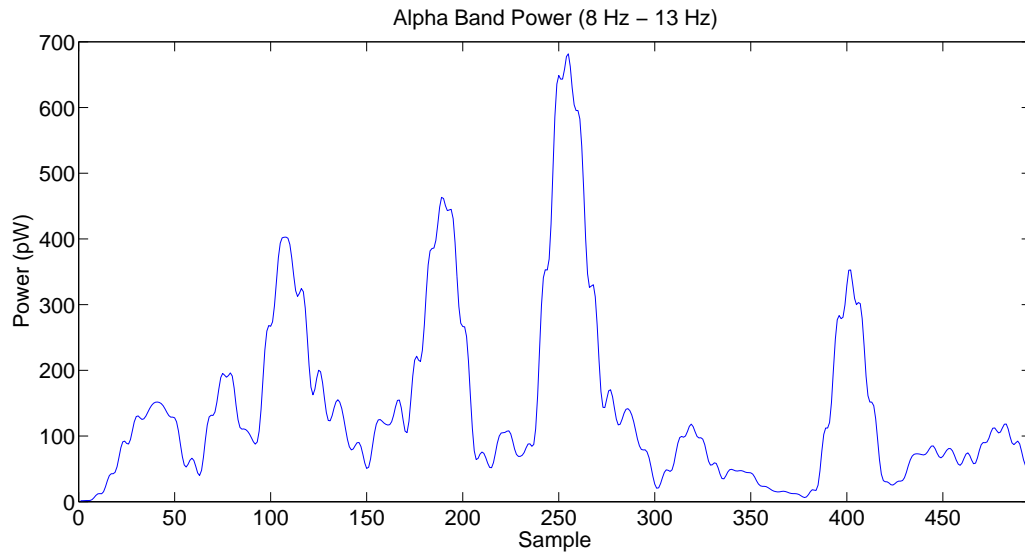
### 3.2.2 IIR

The same design approach was used for IIR filters. In this case, the filter is of order 3 with a Butterworth response that is intended to have a frequency response as flat as possible in the pass-band. The magnitude and phase response of the filter is shown in Figure 3.4



**Figure 3.4:** IIR filter response for the Alpha band

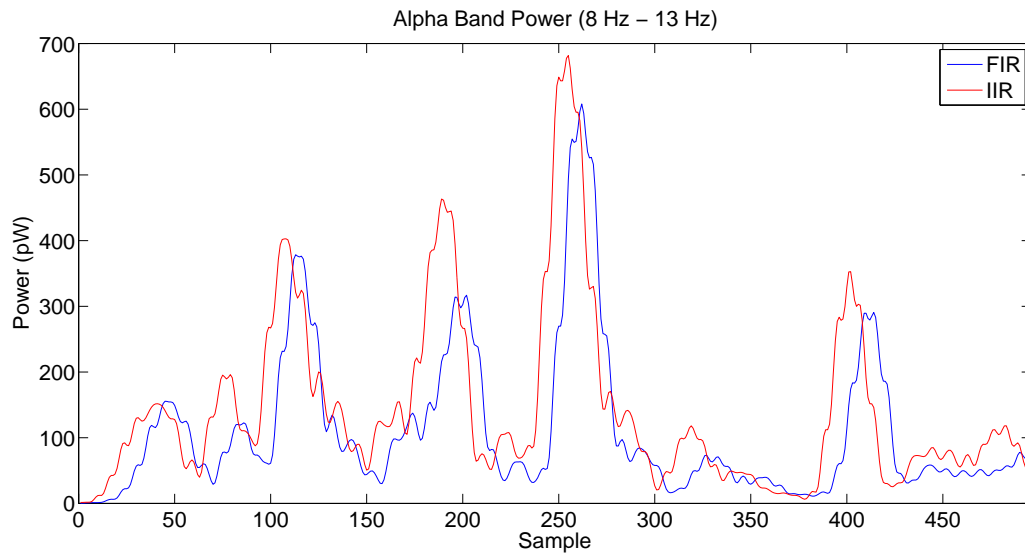
The power signal for the Alpha band is depicted in Figure 3.5



**Figure 3.5:** Power for the Alpha band using IIR filter

### 3.2.3 Comparison

A comparison of the two filters is presented. Figure 3.6 shows power for FIR and IIR filters working in the Alpha band.



**Figure 3.6:** Comparison of FIR and IIR

The power plot shows that the shape of both curves is almost identical with slightly differences. There is a small difference in amplitude that is associated to a small variation between the gain of the filters in the pass-band, the magnitude response of the IIR filter is not as sharp as the FIR counterpart so it means there is more gain in certain frequencies. There are also small differences in shape due to the linearity of the phase response, however those differences are not critical. Finally, there is a small time shift as expected since IIR has a smaller delay than FIR. Since both approaches showed similar results and the observed differences are not critical for the application, FIR is the algorithm to be implemented since it is able to get satisfactory results with much less complexity in implementation than IIR.

### 3.3 FFT Based Techniques

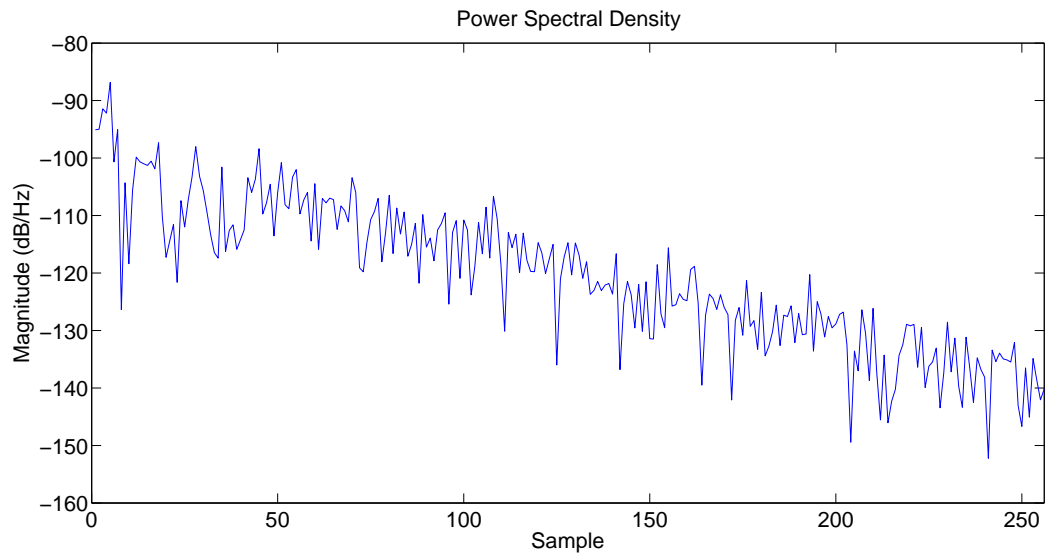
Periodogram and Welch's method are also simulated and compared as feature extraction techniques. Both approaches aim to compute PSD for different frequencies. The result is a frequency domain representation of power in which each frequency band is contained in a finite amount of samples depending on the frequency resolution. Frequency resolution will depend of the number of points used to compute the FFT to the whole signal or to a time window in the signal. Power after the FFT is normalized so the units of the PSD are W/Hz, however it is usually represented in dB/Hz by taking the base 10 logarithm of the power.

#### 3.3.1 Periodogram

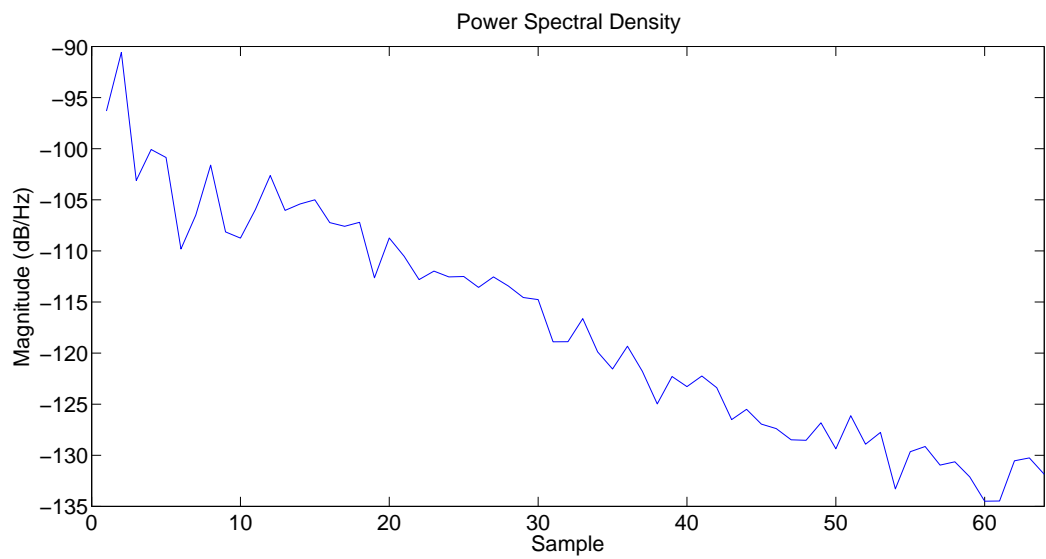
A simple periodogram can be applied to the EEG signal in order to compute PSD. In this case the periodogram definition given in equation 2.4 was applied to the input signal, using a 512 points FFT. Therefore, for this approach the output has 256 samples with a frequency resolution of 0.3125 Hz/sample. The results of the simulation can be observed in Figure 3.7.

#### 3.3.2 Welch's Method

Welch's method is also tested for PSD computation and in this case a rectangular window of 128 samples and a 50% overlap is used in order to reduce complexity. A 128 points FFT is computed for each window resulting in an output with 64 samples and therefore a frequency resolution of 1.25 Hz/sample. The results can be observed in Figure 3.8.



**Figure 3.7:** Periodogram on the EEG signal

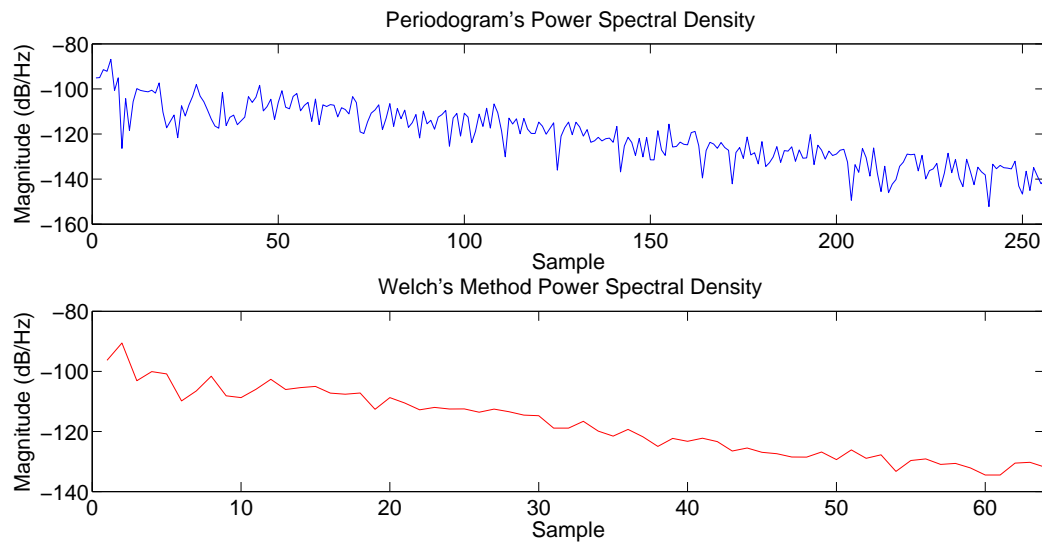


**Figure 3.8:** Welch's method on the EEG signal



### 3.3.3 Comparison

A comparison of the periodogram and Welch's method is depicted in Figure 3.9.

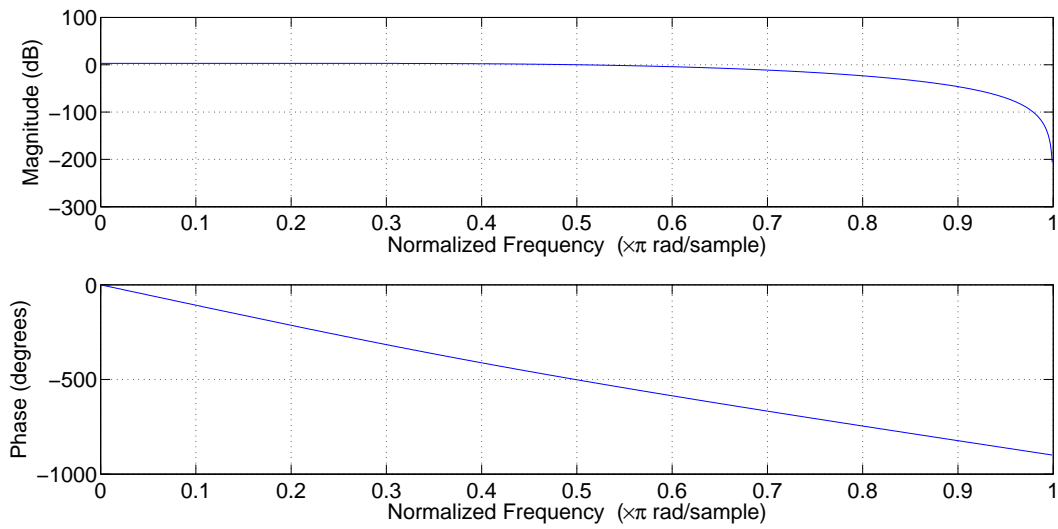


**Figure 3.9:** Comparison of periodogram and Welch's method

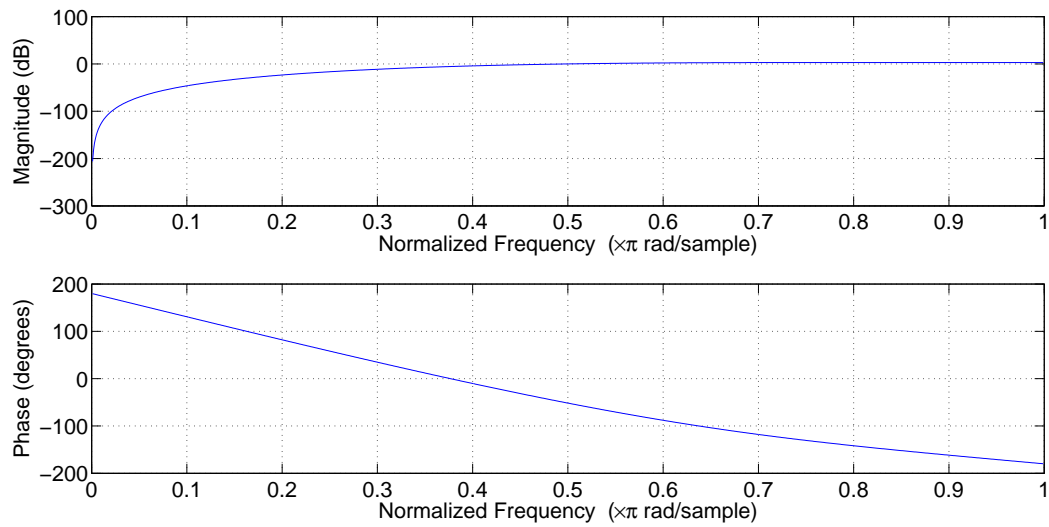
Both plots depict a similar shape with some differences. As expected, the periodogram provides a good frequency resolution in trade of a noisy reading. On the other hand, Welch's method frequency resolution is smaller but the plot has reduced variance and the power is not as noisy as the one from the periodogram. The noise present in the periodogram reading can provide erroneous information about the brain activity, Welch's method is able to provide more accurate readings and can also supply information about each band using less points. Frequency resolution can be adjusted only by changing the window size and since it is not very complex when compared to the periodogram as it only includes some extra windowing and averaging, it is the selected method to be implemented.

## 3.4 DWT Based Techniques

Lastly, DWT and WPT are compared as wavelet based feature extraction techniques. Wavelet approaches lay on the principle of performing a decomposition through a set of high-pass and low-pass filters in order to get signals that relate to certain frequency ranges, if those ranges are calculated to be similar to the EEG frequency bands then power can be computed as well. The result is a time-frequency domain representation of power in which each band is contained in a finite amount of samples depending on the method and decomposition level. The mother wavelet is directly related to the filter coefficients and response. The mother wavelet for simulations is db4, the magnitude and phase response for the decomposition filters are shown in Figures 3.10 and 3.11 respectively.



**Figure 3.10:** Low-pass decomposition filter response for DWT



**Figure 3.11:** High-pass decomposition filter response for DWT

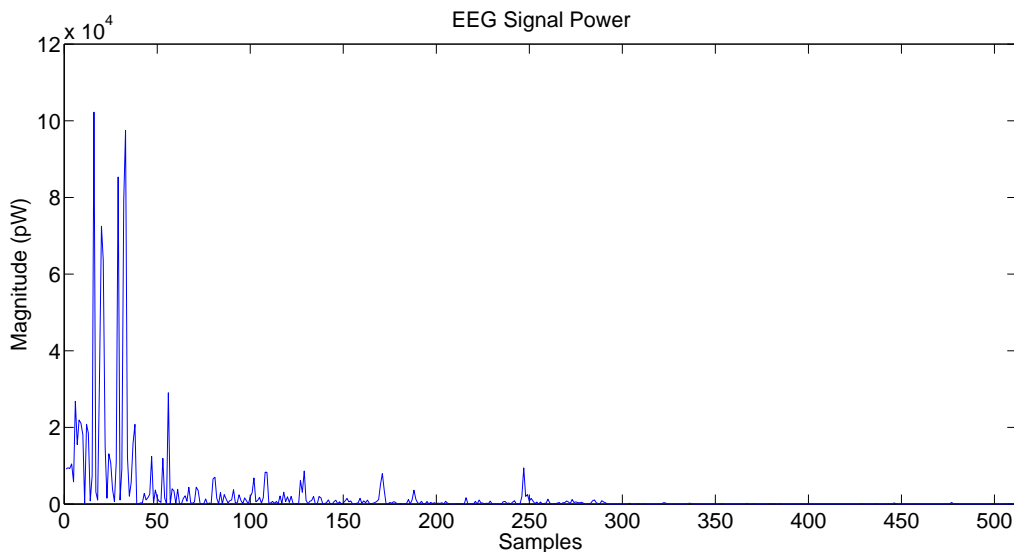
### 3.4.1 DWT

In order to use DWT, a decomposition level must be selected depending on the sampling frequency. Since the signal is sampled at 160 Hz the decomposition level suited to extract the desired bands is 4. Table 3.1 shows which frequency bands can be obtained in each level and the amount of samples that relate to each band.

**Table 3.1:** Bands obtained using DWT

Level	Frequency (Hz)	Samples	Related band
4	0-5	38	Delta
	5-10	38	Theta
3	10-20	70	Alpha
2	20-40	133	Beta
1	40-80	259	

The resulting power signal from the decomposition is shown in Figure 3.12 and it is formed by the approximation and details coefficients of the last level followed by the details coefficients of the rest of the levels.



**Figure 3.12:** DWT decomposition of the EEG signal

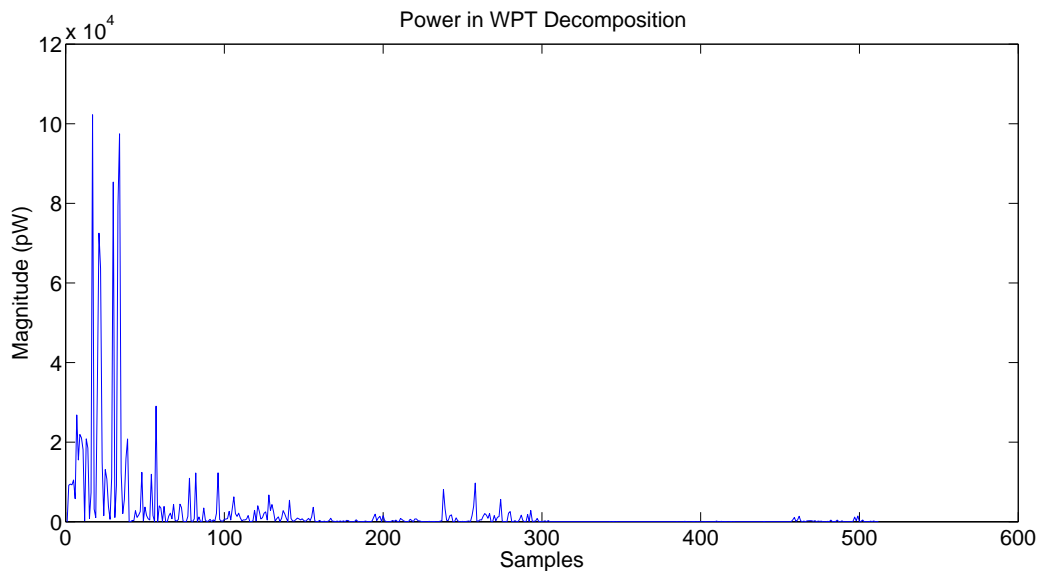
### 3.4.2 WPT

In the WPT, since approximation and details coefficients are decomposed in each level, the amount of nodes in the last level is  $2^N$  where  $N$  is the decomposition level. In this case a decomposition of level 4 is well suited to extract the desired frequency bands, Table 3.2 shows the frequency bands that can be obtained using WPT.

**Table 3.2:** Bands obtained using WPT

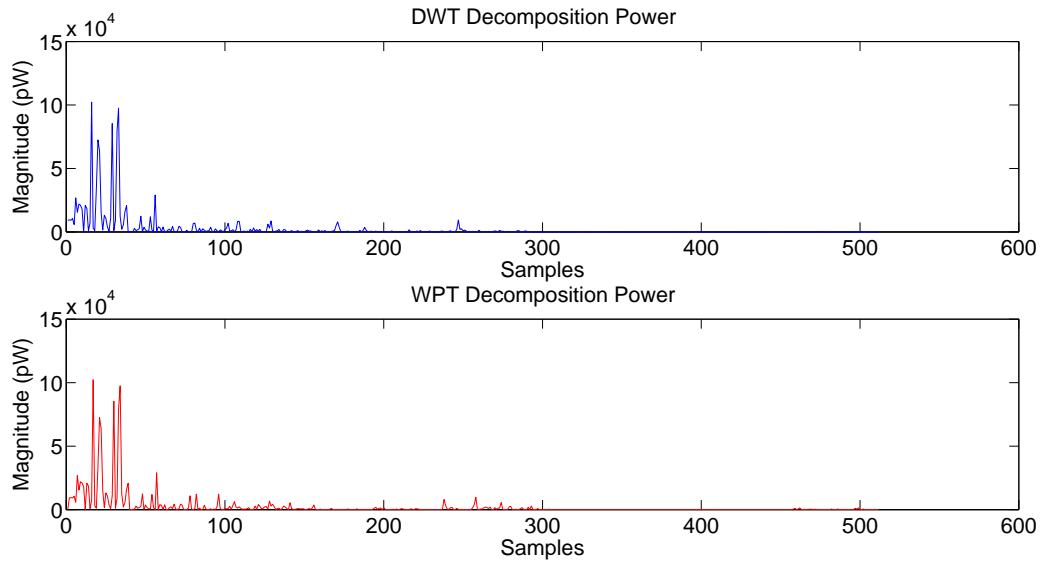
Frequency (Hz)	Related and
0-5	Delta
5-10	Theta
10-15	Alpha
15-20	Beta
20-25	
25-30	
30-35	
35-40	
40-45	
45-50	
50-55	
55-60	
60-65	
65-70	
70-75	
75-80	

The amount of samples for each band is the same and is equal to the number of samples in the last decomposition level of a DWT of the same level. The resulting power signal from the decomposition is shown in Figure 3.13.

**Figure 3.13:** WPT decomposition of the EEG signal

### 3.4.3 Comparison

Figure 3.14 shows a comparison between DWT and WPT.

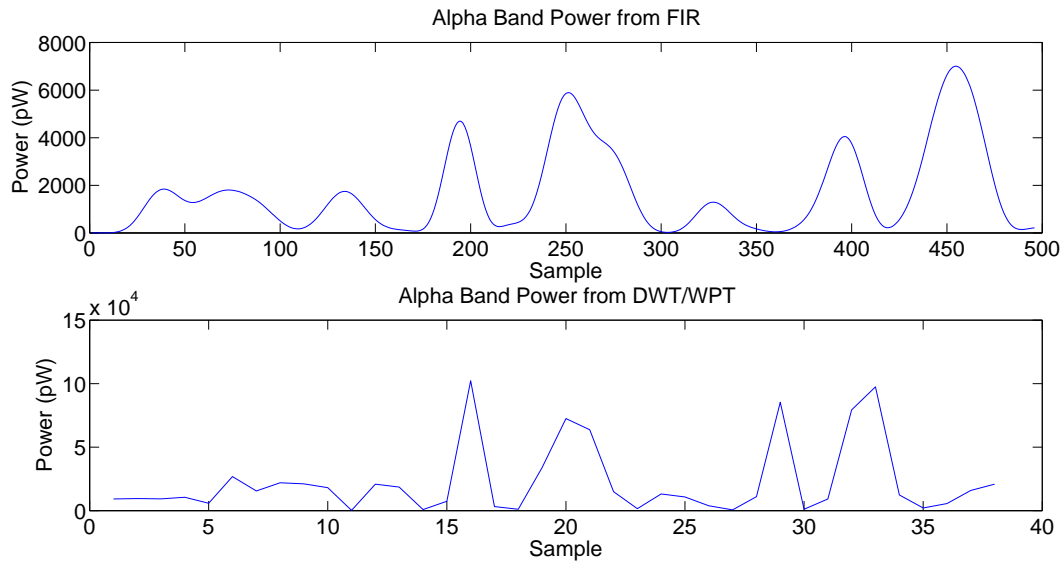


**Figure 3.14:** Comparison of DWT and WPT

Both plots show very similar results with small differences in the two upper bands. Just as expected, the two lower bands look identical since the signal goes through the exact same filters, the approximation and detail coefficients for the last level are common for DWT and WPT as it can be noticed by looking at Figures 2.6 and 2.7. The two upper bands do not differ much, they provide the same information but differently distributed because of differences in frequency resolution between techniques as it can be appreciated in Tables 3.1 and 3.2. Since both approaches provided similar results and because DWT is considerably less complex to implement than WPT, it is the last selected technique.

## 3.5 Comparison between techniques

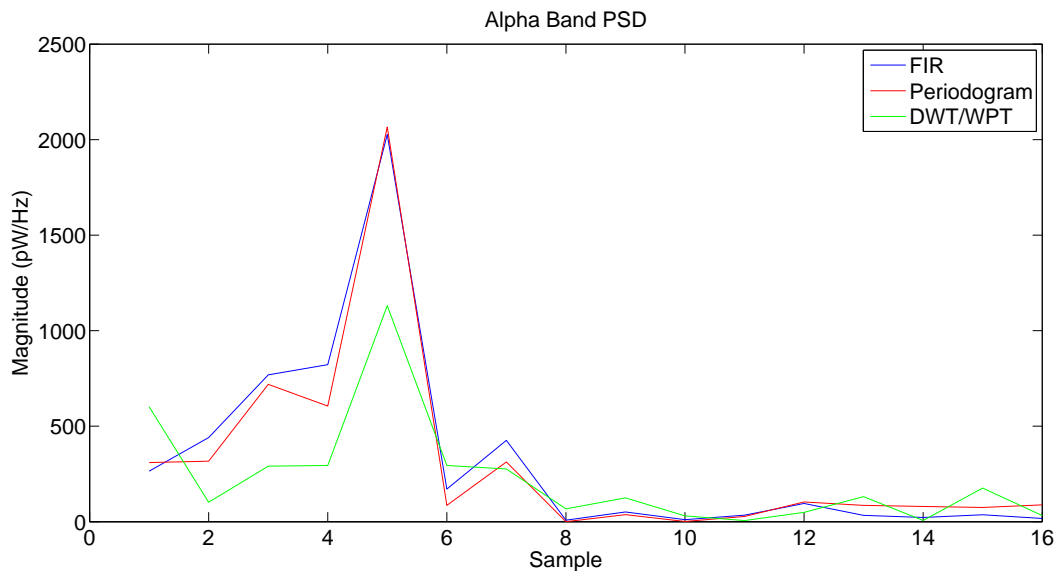
Since filter, FFT and DWT based techniques provide information about the frequency bands in different manners, it is useful to find a way compare them in order to verify that they provide similar and therefore correct information. Filter and DWT based approaches work in the time and time-frequency domain respectively. However, they both end up providing a representation of power over time with different resolutions. A comparison of both approaches can be observed in Figure 3.15.



**Figure 3.15:** Comparison of filter and DWT based techniques

The power computed for the FIR is made up of 512 samples and it includes frequencies from 0.5 to 4 Hz. The DWT/WPT approach is made up of the first 38 samples of the output array corresponding to the Alpha band and it includes frequencies from 0 to 5 Hz because of the resolution of the method. Even though there are differences in resolution and frequency ranges, both plots appear to provide similar information. Scale differences are due to variations in the gain of the filters.

FFT approaches work in the frequency domain and they can not be compared directly to the rest. One way to compare this approach with the other two is to move the time signals computed with the filters and DWT to the frequency domain by calculating PSD. Figure 3.16 shows a comparison of PSD for each approach.



**Figure 3.16:** Comparison of filter, FFT and DWT based techniques

In this case, a periodogram was computed to the time signals obtained with the filters and DWT decomposition for the Alpha band, while for the periodogram the first 16 samples were taken corresponding to that band as well. It can be noticed that all the plots follow a not identical but similar shape too. Scale and shape differences are due to amplitude variations in time signals because of filter gains and also because of frequency resolution. This results evidence that even though each approach provide information about the bands in different manners, they all provide similar and therefore correct information. This comparison turns the results a reliable tool to later validate the implementations.





# Chapter 4

## Hardware Platform and Measurement Procedure

Before running into implementations it is important to set a procedure to ensure that the techniques can be properly evaluated. This chapter provides a brief introduction about the platform under use and its main characteristics. It also states the procedure to follow in order to measure execution time, memory usage and power consumption.

### 4.1 Platform Overview

The platform in which the selected feature extraction techniques will be implemented is a MSP432P401R LaunchPad from Texas Instruments. This development kit incorporates a MSP432P401R microcontroller oriented to develop high performance applications that benefit from low-power operation. The main features of this microcontroller are:

- **Processor:** 48 MHz 32-bit ARM Cortex M4F with FPU and DSP acceleration.
- **Power consumption:** 80 uA/MHz active and 660 nA RTC standby operation.
- **Analog:** 24 Ch 14-bit differential 1MSPS SAR ADC, 2 comparators.
- **Digital:** Advanced encryption standard accelerator, CRC, DMA, HW MPY32.
- **Memory:** 256 KB Flash, 64 KB RAM.
- **Timers:** 4 x 16-bit, 2 x 32-bit.
- **Communication:** up to 4 I2C, 8 SPI, 4 UART.

The LaunchPad also includes an on-board emulator which means the user can program and debug the projects without the need for additional tools. Free software development tools are available to work with this platform such as the Code Composer Studio™ (CCS) IDE, IAR Embedded Workbench™ IDE and Keil®  $\mu$ Vision® IDE.

The use of this platform together with the mentioned development tools provides the developer many optimization tools and possibilities to improve any application in terms of code size, speed and mostly low-power consumption. More information about the platform can be found in [14].

## 4.2 Measurement Procedure

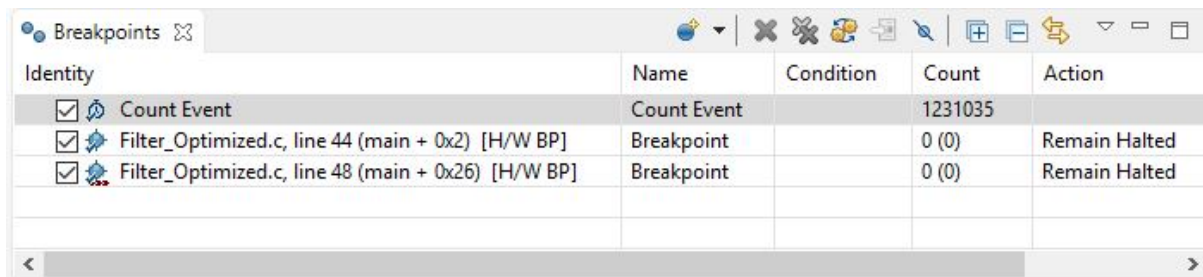
Here, a procedure to measure execution time, memory usage and power consumption is presented. This procedure is not exclusive for the feature extraction techniques and can be used in any application that runs in the MSP432P401R LaunchPad. Each procedure was developed using CCS.




### 4.2.1 Execution Time

Execution time is, as its name implies, the time it takes to run a piece of code. Measuring execution time with CCS is fairly simple. The measurement can be performed using the debugger by setting breakpoints between the piece of code that wants to be measured and one count event breakpoint, then running the code and finally looking at a counter with the clock cycles that it took the code to run. Then, it is possible to get the execution time if the operating frequency of the system is known by using:

$$\text{execution time} = \frac{\text{clock cycles}}{\text{clock frequency}} \quad (4.1)$$

Figure 4.1 depicts an example of the Breakpoints tab showing the clock cycles that it took to run a piece of code and also the two breakpoints that delimit the measured code. The clock cycles for the same piece of code do not vary and therefore the execution time is strictly tied to the operating frequency as shown in Equation 4.1.



Identity	Name	Condition	Count	Action
<input checked="" type="checkbox"/>  Count Event	Count Event		1231035	
<input checked="" type="checkbox"/>  Filter_Optimized.c, line 44 (main + 0x2) [H/W BP]	Breakpoint		0 (0)	Remain Halted
<input checked="" type="checkbox"/>  Filter_Optimized.c, line 48 (main + 0x26) [H/W BP]	Breakpoint		0 (0)	Remain Halted

**Figure 4.1:** CCS Breakpoints tab

The procedure to measure execution time for a piece of code is:

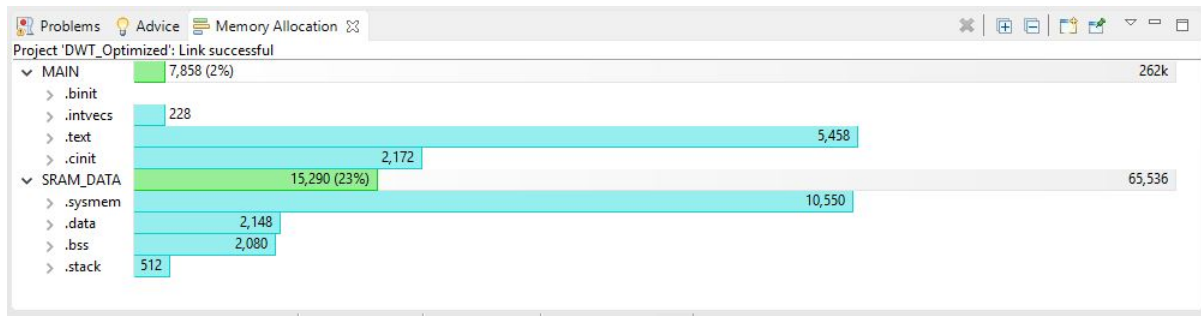
1. In CCS enter the Debug Mode.
2. Open the Breakpoints tab and create a new count event breakpoint configured to count clock cycles.
3. In the breakpoint properties set reset count on run as true.
4. Place two breakpoint between the piece of code to be measured.
5. Run the program until reaching the last breakpoint.
6. Get the clock cycles from the count event breakpoint.
7. Calculate the execution time.

### 4.2.2 Memory Usage

As mentioned before, the MP432P401R microcontroller has 256 KB of Flash and 64 KB of RAM, both types of memory have different purposes. Flash memory is non-volatile and it normally stores data that does not change, it is the program memory. RAM memory is volatile and stores data needed at runtime, this memory is very fast but its size is often limited. Two important components of the RAM are the stack and the heap. The stack is the region in RAM in which variables created inside the functions called by the program are stored and the heap is the region of RAM managed by the programmer.

CCS counts with a Memory Allocation tool that makes measuring Flash memory usage fairly simple and is also able to measure RAM memory usage partially. The tool is only able to perform a static analysis on the code and therefore is able to measure static but not dynamically allocated memory.

Figure 4.2 depicts an example of the Memory Allocation tab. The MAIN measurement shows the Flash memory usage, the percentage used over the whole memory available and also specifies how it is distributed. The SRAM\_DATA measurement shows the RAM memory usage. It also shows the percentage used over the whole memory available and specifies how it is distributed. The static data is comprised in the .data and .bss sections while the stack is in the .stack section and the heap in the .system section. The minimum default values are 512 bytes for the stack and 1024 bytes for the heap. The programmer can specify the space assigned for the stack and heap in the build settings. Depending on the application those values should increase if the code requires more space for any of those memory regions. Sometimes during debugging or in a real application a code can crash or not work as intended because it can run out of memory or even because the heap and the stack can crash during runtime.



**Figure 4.2:** CCS Memory Allocation tab

If the total RAM memory usage wants to be calculated the programmer needs to include the dynamic allocated peak memory that could be reached by the code and therefore must specify in the build settings the space provided for the heap and the stack. The programmer is the responsible of taking and releasing memory commonly using functions like *malloc* and *free*. To estimate the dynamic peak memory, the programmer should have a clear idea about what the code does and needs to find the scenario in which the biggest amount of memory is requested. This should not be an issue since every time memory is requested the size of the memory block needs to be specified, so this amount of memory can be approximated by looking at memory allocations inside the code and their respective size.

The procedure to measure memory usage is:

1. In CCS open the Memory Allocation tab.
2. Get Flash memory usage from the MAIN measurement.
3. Get RAM memory usage from the SRAM\_DATA measurement.
4. Add the amount of dynamically allocated peak memory to the RAM measurement if necessary.

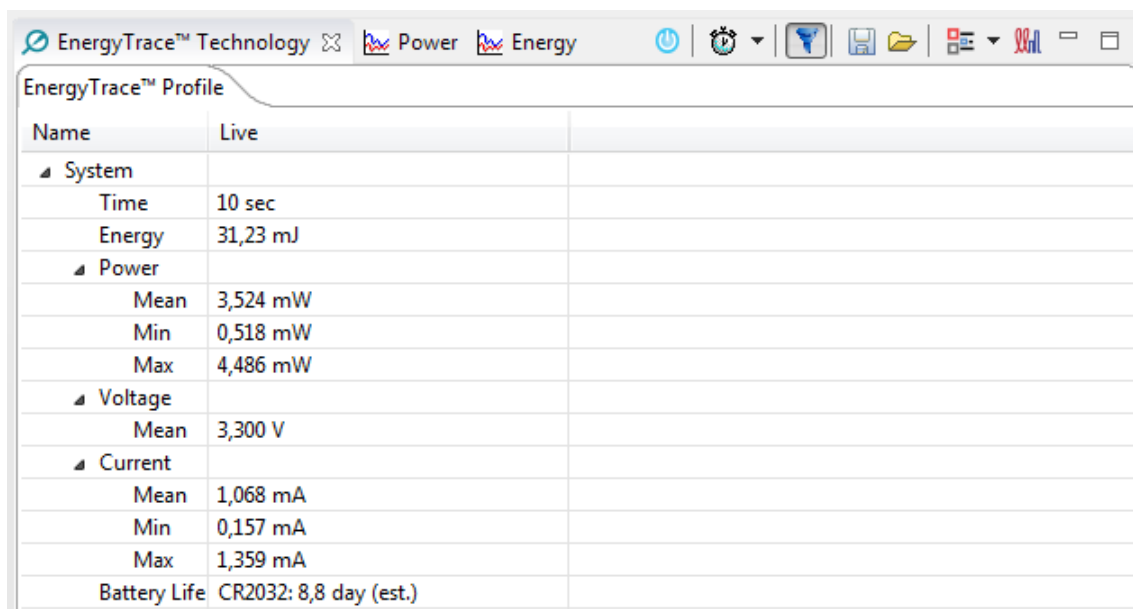
### 4.2.3 Power Consumption

Given that battery life is often an issue in IoT applications, power consumption is a critical parameter to take into account during the design process. There are many factors that can influence power consumption like operating voltage, clock frequency, sleep times, transition times or even intermodule dependencies. It is important then to find a method to accurately measure power to evaluate different scenarios aiming to achieve the lowest power consumption.

EnergyTrace™ is an energy based code analysis tool included in CCS that measures and displays the application's energy profile and helps to optimize it for ultra-low-power consumption [13]. For MSP432 devices Energy Trace supports two modes of operation: ET (energy profiling only) and ET+ (energy profiling + program counter trace). ET enables analog energy measurements to determine the consumption of an application. ET+ in addition supports a tool useful for measuring and viewing the application's energy profile and correlating it with the state of the CPU.

Figures 4.3, 4.4 and 4.5 show the different views available for the Energy Trace working in ET mode. For every measurement the user can specify the length of the measurement and is also able to save data for future comparison. The Profile tab provides information about the consumed energy, power, voltage, current and also gives an estimate of the battery life for the current application using a specific type of battery that can also be configured. Power and energy tabs show a plot of how power and energy behave over time for the current application. Sometimes, if the code runs only once and very fast, Energy Trace may not be able to measure power. Then, it is useful to include the code that wants to be measured inside a loop just for measuring purposes. By doing that the mean energy consumption for one iteration can be found using:

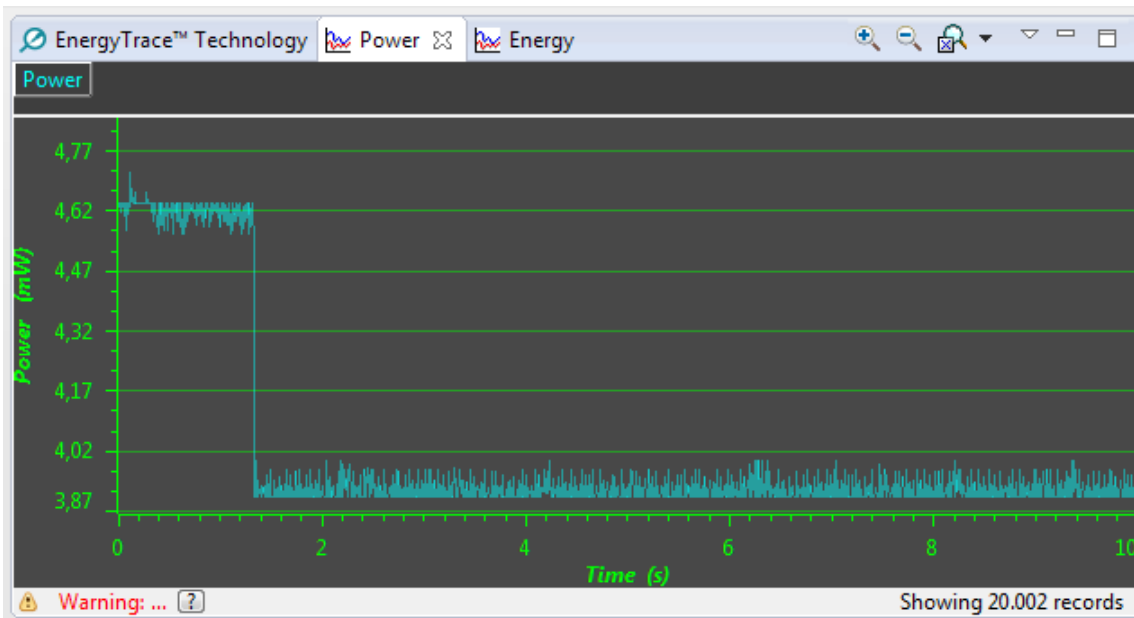
$$\text{mean energy} = \text{mean power} \times \text{execution time} \quad (4.2)$$



The screenshot shows the EnergyTrace™ Profile window with the following data:

Name	Live
▲ System	
Time	10 sec
Energy	31,23 mJ
▲ Power	
Mean	3,524 mW
Min	0,518 mW
Max	4,486 mW
▲ Voltage	
Mean	3,300 V
▲ Current	
Mean	1,068 mA
Min	0,157 mA
Max	1,359 mA
Battery Life	CR2032: 8,8 day (est.)

**Figure 4.3:** Profile tab in ET mode



**Figure 4.4:** Power tab in ET mode



**Figure 4.5:** Energy tab in ET mode

On the other hand, ET+ mode is able to provide information about the time that the application spends in active or in one of the low-power modes of the device, it is also able to give information about which routine or routines consume more or less power. Figure 4.6 shows how the Profile tab looks in the ET+ with an example that breaks down power for every function inside the code. Figure 4.7 shows the states tab that is not present in ET mode, this tab shows the distribution of power modes over time.

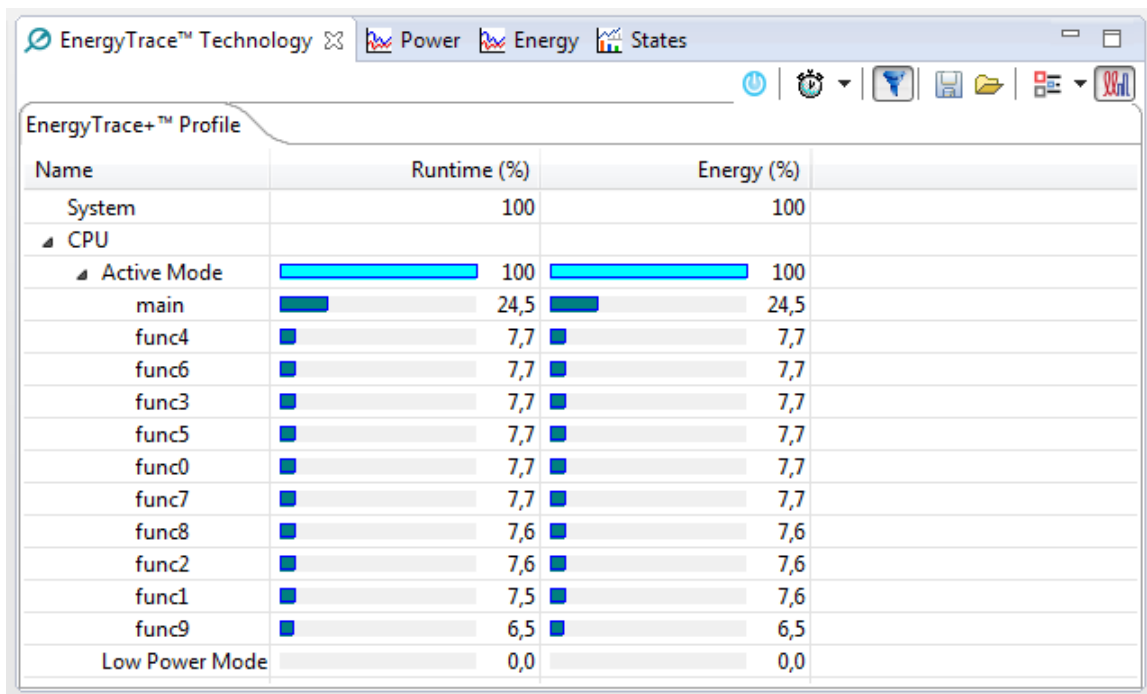


Figure 4.6: Profile tab in ET+ mode

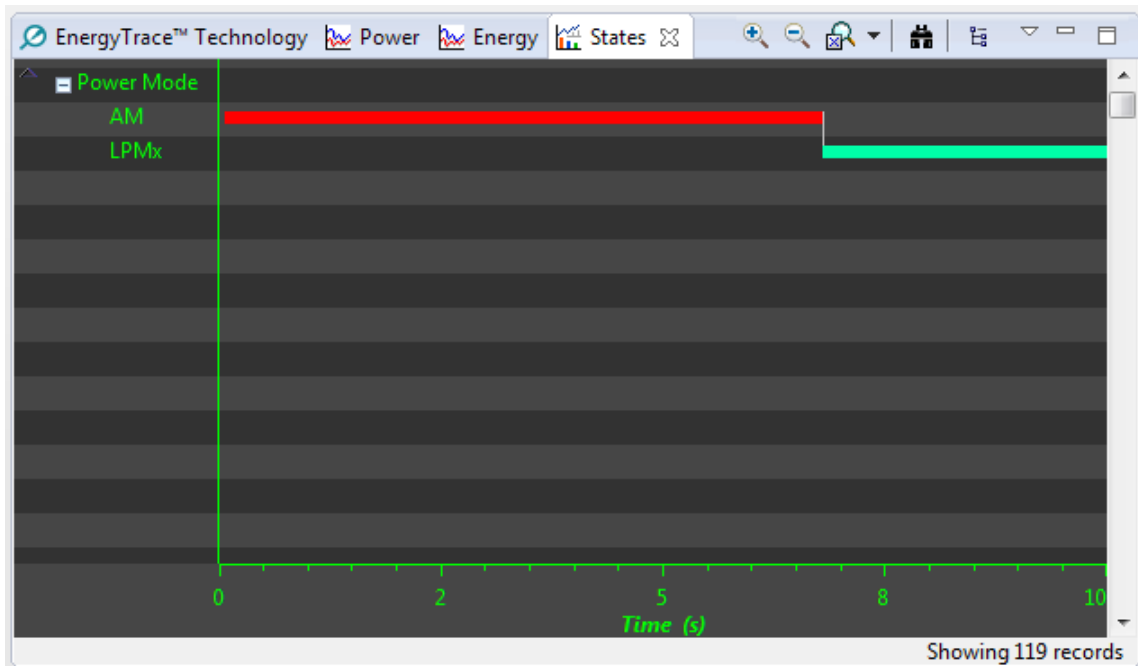


Figure 4.7: States tab in ET+ mode

For measurement purposes, only ET mode will be used in order to measure power and energy for each technique. Also, each technique was included inside a loop and therefore to measure energy Equation 4.2 was used. ET+ mode can be useful in the final stages of the IoT EEG device when not only feature extraction is ready but the preprocessing and classification stage are too. It could give the chance then to incorporate low-power modes between the acquisition of the signal and the whole processing and can also provide information about the power consumption of each individual stage.

The procedure to measure power consumption is:

1. In CCS a open debug session and open Energy Trace.
2. Configure the time for the measurement.
3. Run the application and Energy Trace simultaneously.
4. Get the power measurement from the Profile tab.



# Chapter 5

## Implementation of Feature Extraction Techniques

This chapter explains the approach taken for the microcontroller implementation of each feature extraction technique. It introduces the CMSIS-DSP library and describes some of the functions that were borrowed from this library into the implementation. It also brings a detailed explanation of the procedure followed by each algorithm and it presents validations for each technique against the simulated references developed in MATLAB.

### 5.1 CMSIS DSP Software Library

Before describing the implementation of each feature extraction technique, it is necessary to briefly introduce the CMSIS-DSP Software Library [3]. CMSIS stands for Cortex Microcontroller Software Interface Standard and it is a vendor-independent hardware abstraction layer for the Cortex-M processor series. The CMSIS enables consistent device support and simple software interfaces to the processor and the peripherals. Hence, it is intended to simplify software re-use, reduce the learning curve for microcontroller developers and also the time to market for new devices.

One component of the CMSIS is the CMSIS-DSP. DSP is a library collection with over 60 functions for various data types like fix-point (fractional q7, q15, q31) and single precision floating-point (32-bit). The library is available for all Cortex-M cores. The Cortex-M4 and Cortex-M7 implementations are optimized for the SIMD instruction set. The library covers functions for the following operations:

- Basic math functions.
- Fast math functions.
- Complex math functions.

- Filters.
- Matrix functions.
- Transforms.
- Motor control functions.
- Statistical functions.
- Support functions.
- Interpolation functions.

Some functions from filters, complex math functions and transforms were exploited during the implementation of the feature extraction techniques. A short description each of them is given next:

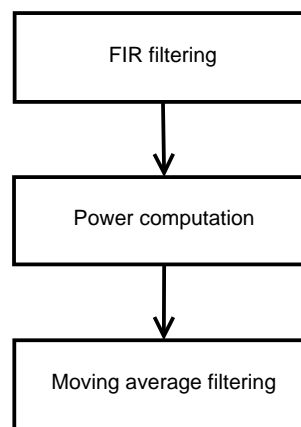
- *arm\_fir\_init\_f32*: this function is used to initialize a floating-point FIR filter. Here a floating-point FIR filter structure is referenced, also information about the filter coefficients and the size of the block that is processed per call is provided.
- *arm\_fir\_f32*: this function performs a floating-point FIR filtering. It must be called after the initialization. A FIR instance is provided, a pointer to the input and output vector, and the block size.
- *arm\_rfft\_fast\_init\_f32*: this function initializes the floating-point real FFT. Here a real FFT instance structure is referenced as well as the length of the FFT. It verifies that the length of the FFT is a supported value. Supported FFT lengths are 32, 64, 128, 256, 512, 1024, 2048 and 4096.
- *arm\_rfft\_fast\_f32*: this function performs a floating-point real FFT. It must be called after the initialization. A FFT instance is provided, also a pointer to the input and output array, and a flag to indicate if a direct or an inverse transform is performed.
- *arm\_cmplx\_mag\_f32*: this function can compute the magnitude of a floating-point complex vector. A pointer to the input and output vectors must be provided as well as the number of complex samples in the input.
- *arm\_conv\_f32*: this function computes convolution of floating-point sequences. A pointer to two input sequences, the length of each of them and a pointer to the output vector must be provided. The length of the output will be equal to the sum of the lengths of the two input sequences plus 1.

## 5.2 Microcontroller Implementation

FIR, Welch's method and DWT were implemented in the MSP432P401R LaunchPad. For each technique, a general description, implementation details and a performance validation is given. To validate that each technique works as intended, the test signal in Figure 3.1 was also provided to each code as input. The output array was extracted from the microcontroller and then imported to MATLAB in order to be compared to the reference obtained during simulations.

### 5.2.1 FIR Feature Extraction

The FIR feature extraction implemented code is able to take an EEG input signal to compute power in any of the frequency bands. A single function was developed in which the user must provide an array with the EEG reading, an array containing the filter coefficients for the desired band and finally an output array to store power. The filter coefficients for each band are fixed for a specific sampling frequency and can be easily obtained using MATLAB. Figure 5.1 shows the approach taken to extract features from an EEG reading using FIR.



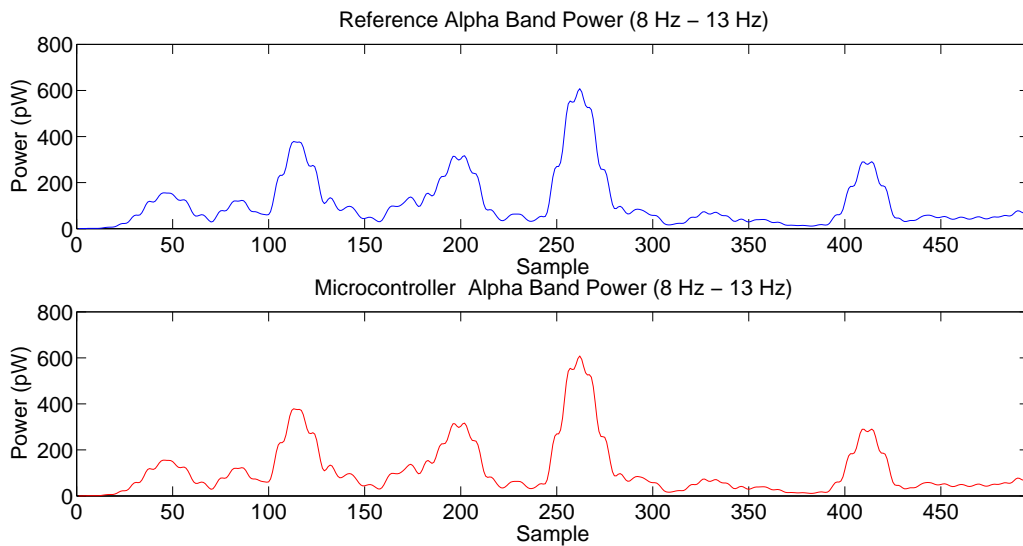
**Figure 5.1:** FIR feature extraction algorithm sequence

The first step involves filtering the EEG signal. Here, two functions from the CMSIS-DSP are used together. The *arm\_fir\_init\_f32* is used to initialize the filter with the provided band coefficients and later the *arm\_fir\_f32* function computes the filtered signal.

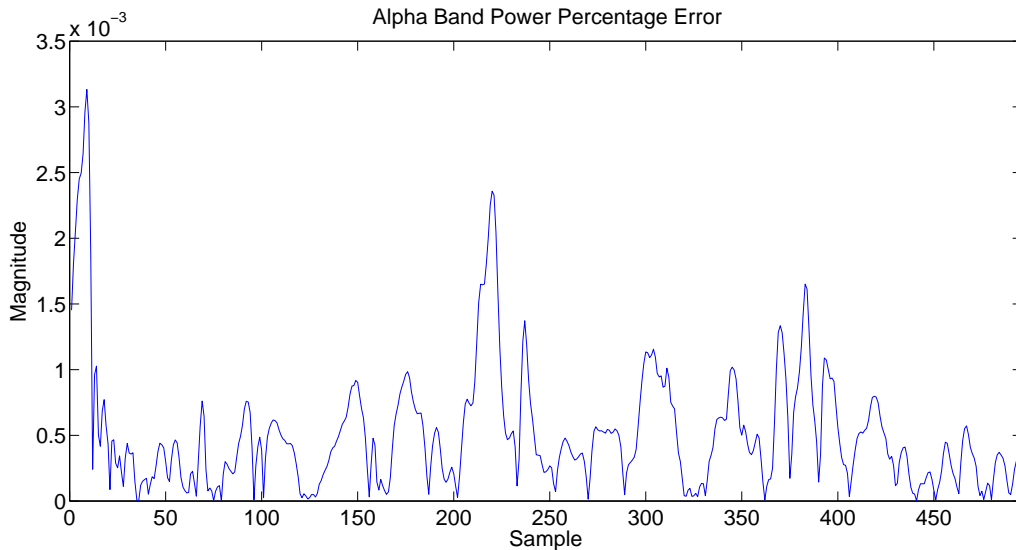
After the signal filtering stage comes power computation. Since power is normalized to a  $1 \Omega$  resistor, it can be computed just by squaring the filtered time signal. It is achieved by using a loop to multiply each element stored in the filtered signal array by itself.

Finally, a moving average filter is applied to the power measurement in order to get a less noisy reading. This filter takes each element in the power array and replaces it with a value obtained by averaging a certain number of consecutive samples.

Figure 5.2 shows a comparison of the reference simulated signal from the one obtained in the microcontroller. Since both plots look very similar a plot of the percentage error is provided in Figure 5.3 and also Table 5.1 shows the minimum, average and maximum error in the measurement for the Alpha band and the rest. This measurements show that the algorithm performs well and is able to extract power from each of the bands in a proper and accurate way.



**Figure 5.2:** Simulated and microcontroller implementation comparison for FIR



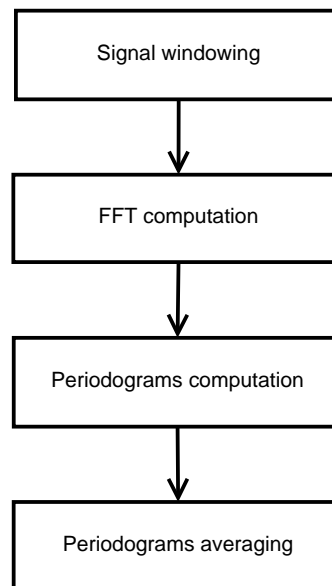
**Figure 5.3:** Validation error for FIR

**Table 5.1:** Validation percentage error for FIR

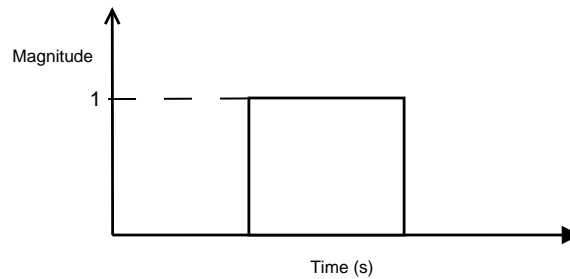
Band	Minimum	Average	Maximum
Delta	0.000057911	0.00046846	0.0012616
Theta	0.0000061622	0.001232	0.0060379
Alpha	0.00000046394	0.00051055	0.0031324
Beta	0.000010121	0.028568	1.6774

### 5.2.2 Welch's Method Feature Extraction

The implemented Welch's method algorithm takes an EEG input signal and computes PSD. For this technique a single function was created too, the user must provide an input array containing the EEG reading and an output array that will hold power. For this algorithm the sampling frequency must be declared since it is used for computing the periodogram. The window length, the FFT length and the number of windows are set by the code based on the input size but they can also be tuned for other specific implementations. The code is restricted to work with inputs and windows containing a power of 2 number of elements because of a restriction of the FFT functions from the CMSIS-DSP library. The implemented code creates 7 windows with sizes that are equal to the input size divided by 4, therefore the output will have a size equal to the input size divided by 8 as a result of the periodogram calculation. Figure 5.4 shows the approach taken to extract features from an EEG reading using Welch's method.

**Figure 5.4:** Welch's method feature extraction algorithm sequence

The first thing to do when using Welch's method is to apply a window function to the input signal. In this case, a rectangular window function was chosen in order to reduce complexity. A rectangular function window is equal to 1 inside an interval and equal to 0 elsewhere as shown in Figure 5.5, therefore it only implies dividing the signal into intervals. A 50% overlap was selected for the time windows, this makes easier to window the signal and also ensures that every windows will contain a power of 2 number of elements if the input signal does too. This part of the algorithm fills arrays with signal windows to compute FFT later.



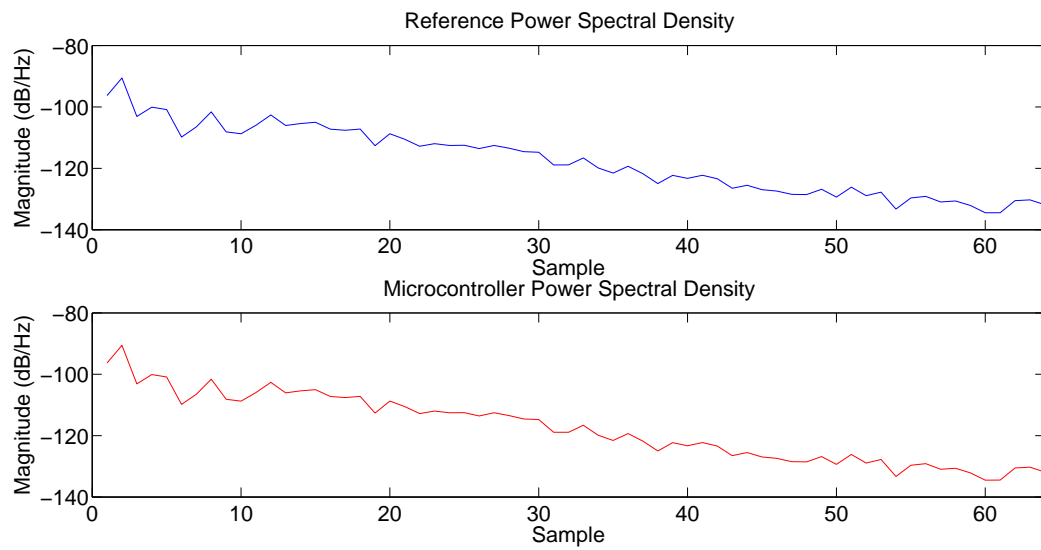
**Figure 5.5:** Rectangular window

For the FFT computation stage the algorithm takes the windows in which the signal was divided and computes a FFT in each of them. In order to do that, three functions from the CMSIS-DSP library were used. The *arm\_rfft\_fast\_init\_f32* function is used to ensure that the length of the transforms that will be calculated are supported values, if the value is correct the code can execute but if not it will get in an infinite loop. Then, the *arm\_rfft\_fast\_f32* function performs a real floating-point FFT creating an array with complex values representing each window in the frequency domain. Finally, since the FFT returns a frequency domain representation array with complex values, the magnitude is calculated using the function *arm\_cmplx\_mag\_f32*.

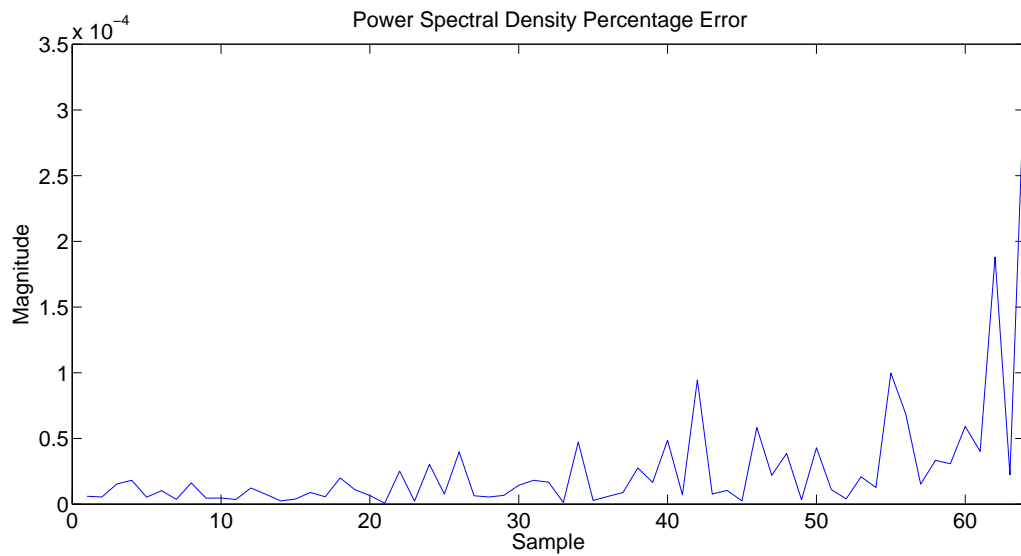
Having calculated the FFT of each window section it is possible then to compute the periodogram for each block. This is achieved by using Equation 2.4 where the FFT is already computed, the space between samples is the inverse of the sampling frequency and the number in samples is equal to the length of the window.

The final step in Welch's method is to average all the periodograms. In order to do that, the algorithm adds the same element for each window and divides the result by the number of windows. The output will then be a representation of the PSD in the EEG signal in which all the different bands can be extracted and analyzed.

A comparison between the simulated signal and the microcontroller implementation is depicted in Figure 5.6. Both plots look to provide the same output and in this case a plot of the percentage error is provided in Figure 5.7 too. Table 5.2 provides information about the the minimum, average and maximum error in the measurements. The error looks to grow as frequency does, however the maximum value is extremely low. By looking at the results it is possible to confirm that the microcontroller implementation performs as intended and is able to estimate PSD accurately.



**Figure 5.6:** Simulated and microcontroller implementation comparison for Welch's method



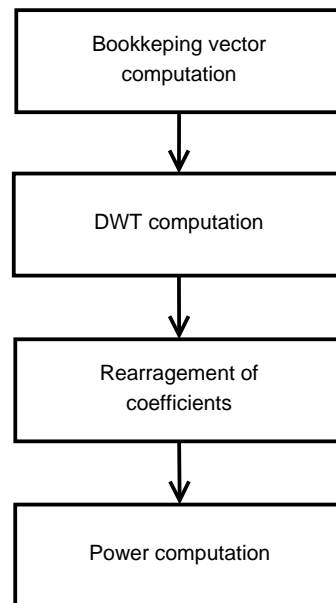
**Figure 5.7:** Validation error for Welch's method

**Table 5.2:** Validation percentage error for Welch's method

Minimum	Average	Maximum
0.0000006253	0.00002702	0.00033331

### 5.2.3 DWT Feature Extraction

The algorithm developed to extract features using DWT is able to decompose an EEG signal through a set of high-pass and low-pass filters in order to extract the frequency bands to compute their power. It also provides a bookkeeping vector that indicates the size of the decompositions in order to make easier to extract the bands later. The use of the technique was reduced to a simple function in which the user must provide the input array containing the EEG signal, an output array to store the decomposition and another one to store the bookkeeping vector. The level of decomposition must be declared by the user depending on the specific application. The algorithm uses a db4 mother wavelet by default, the coefficients of both the high-pass and the low-pass filter were obtained using MATLAB. The user can also declare another mother wavelet by changing the arrays holding the coefficients if necessary. Figure 5.8 shows the approach taken to extract features from an EEG reading using DWT.



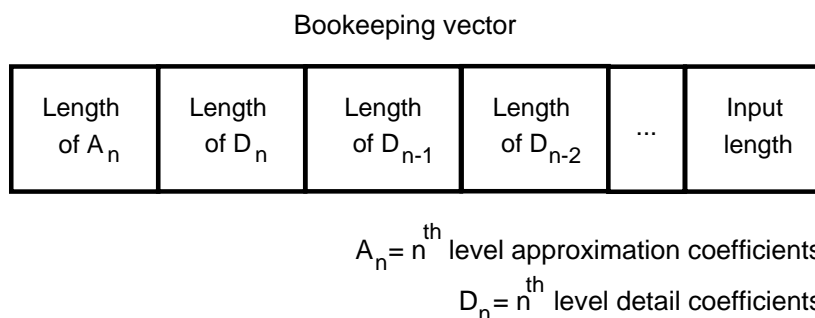
**Figure 5.8:** DWT method feature extraction algorithm sequence

Before calculating the DWT it is possible to get the bookkeeping vector if the input size and the decomposition level are known. The size of each decomposition level can be obtained taking into account the convolution and the down-sampling for the previous level as shown in Figure 2.6. Therefore the size for each level can be calculated using Equation 5.1. The numerator represents the convolution process and the denominator the effect of down-sampling.



$$\text{level size} = \frac{\text{previous level size} + \text{filter length} - 1}{2} \quad (5.1)$$

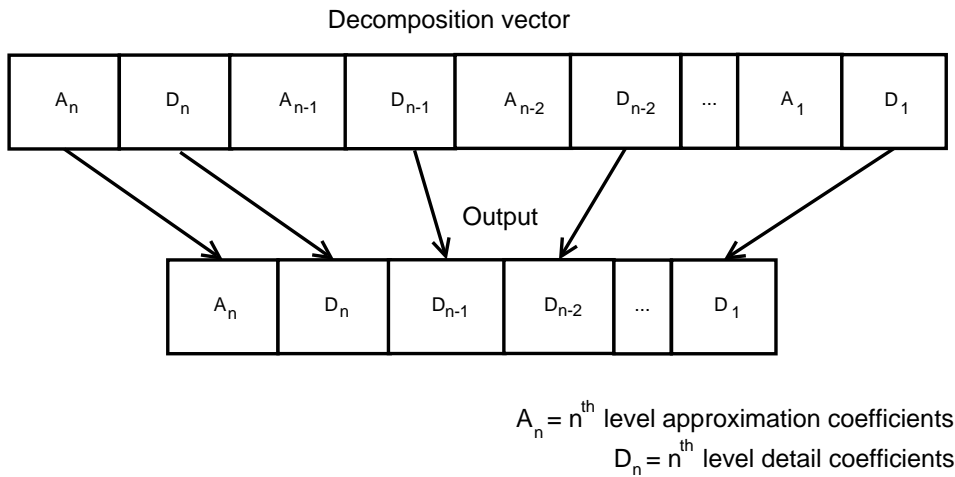
After the length of each decomposition level is computed, a small rearrangement is carried out in order to get the bookkeeping vector that contains information of the approximation and detail coefficients for the last level. The detail coefficients for the rest and finally the input size as shown in Figure 5.9.



**Figure 5.9:** Bookkeeping vector structure for DWT

The DWT computation stage is the most relevant in which the frequency bands are extracted. There is no function in CMSIS-DSP library to perform a DWT decomposition as there is for FIR filtering or FFT computation so most of the implementation was developed from scratch. However, the *arm\_conv\_f32* function from the library was used to make the convolution between the signals in the levels and the filter coefficients. The piece of code that is oriented to compute DWT is embedded inside a loop that runs as many times as level are in the decomposition. It takes the EEG reading as input for the first iteration and then the approximation coefficients of each level for the rest. The first step is to take the input array and pass it through the filters, it is achieved using the function *arm\_conv\_f32* between the input and the filters. The result of the convolution is down-sampled to get rid of the extra information since half of the frequencies are cut because of the filter, in order to do that two new arrays store only the odd index values from the approximation and detail coefficients. Both coefficients arrays are stored in one bigger array that stores the whole decomposition because the approximation coefficients need to be used in the next iteration and the details coefficients will be present in the output array. Then the process is repeated until reaching the last level.

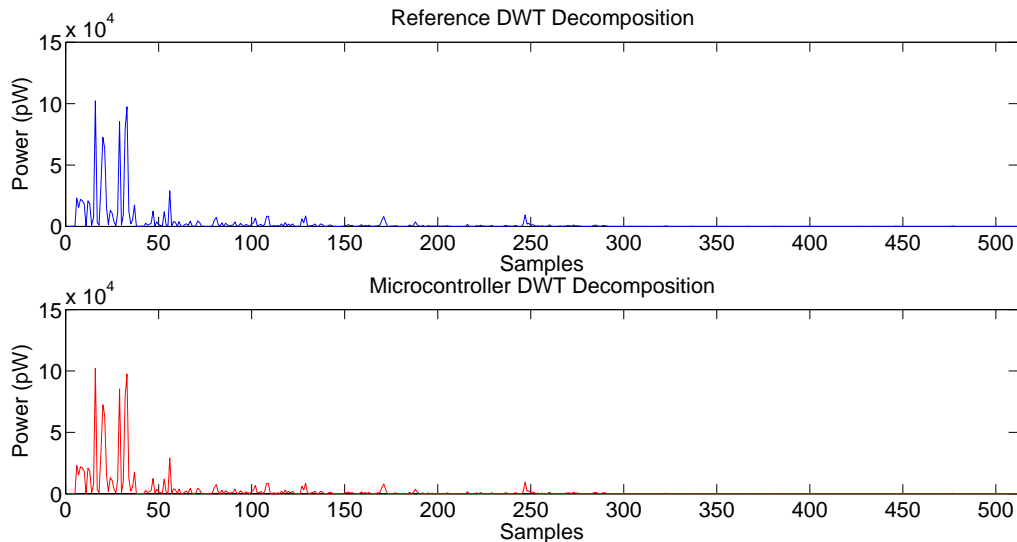
After computing DWT and getting an array holding both the approximation and details coefficients for each level, it is necessary to rearrange and get rid of some values to get the final representation. Based on the decomposition array the output is built containing the approximation and detail coefficients for the last level and the detail coefficients for the rest. The rearrangement is depicted in Figure 5.10.



**Figure 5.10:** Rearrangement of coefficients process for DWT

Finally, the last step is power computation. The output vector will hold a set of coefficients representing information about the signal in different bands and power can be extracted from them. In this case power is computed taking the square of the signal representing each band, therefore power is also normalized for this technique.

The simulated reference signal and the microcontroller implementation are compared in Figure 5.11, also the obtained bookkeeping vector is shown in Figure 5.12. Since it is hard to notice differences between the reference and the implementation, a plot of the percentage error is provided in Figure 5.13 and Table 5.3 provides information about the minimum, average and maximum error in the measurements. The error is bigger in samples that show less power but even though this technique shows bigger errors than the others, those errors are still very low. Therefore, from the experimental measurements, it is evident that the developed algorithm is able to decompose an EEG signal using DWT achieving the same results with the MATLAB analysis.



**Figure 5.11:** Simulated and microcontroller implementation comparison for DWT

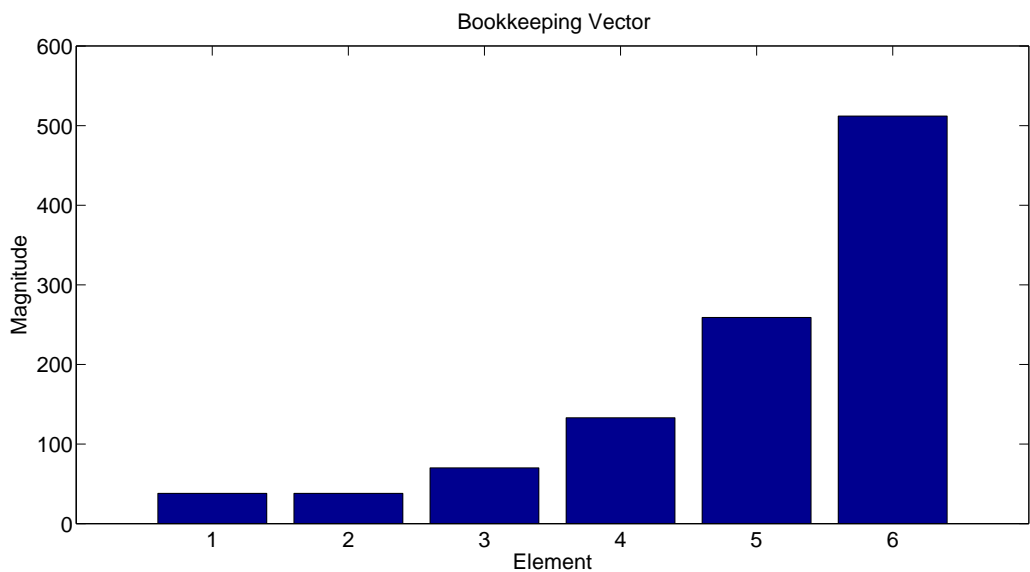


Figure 5.12: Implemented bookkeeping vector for DWT

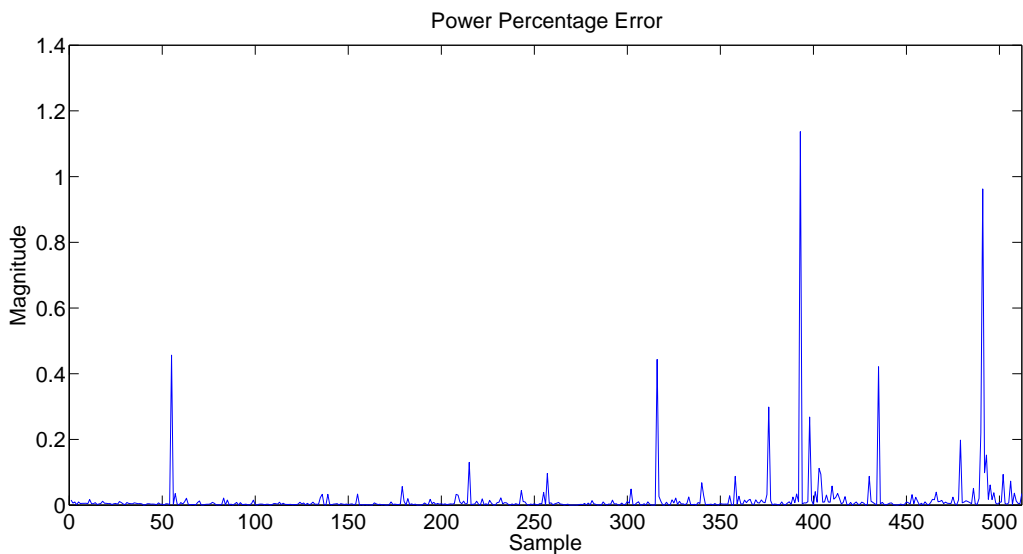


Figure 5.13: Validation error for DWT

Table 5.3: Validation percentage error for DWT

Minimum	Average	Maximum
0.00001162	0.017211	1.1367

## 5.3 Optimizations

Each technique was not only implemented but also optimized as much as possible in order to improve execution time, memory usage and power consumption before running the evaluations. The selected optimizations and also results that demonstrate their impact are shown for each technique. Every measurement was carried out using the procedures described in Chapter 4.

### 5.3.1 Execution Time and Memory Usage

CCS incorporates optimization tools and options to improve speed, code size and power consumption. Settings in the compiler can be adjusted to achieve different optimization levels for speed and code size. Figure 5.14 shows the optimization view in CCS. There are different optimization levels that can be selected from 0 to 4, these optimizations can include register, local, global, interprocedural and whole program optimizations. There are two floating point modes, strict is the one set by default and relaxed is another mode that results in smaller and faster code by sacrificing some accuracy. Finally, there is another option to select the trade-off between speed and code size from 0 to 5, being 0 oriented to size and 5 to speed.

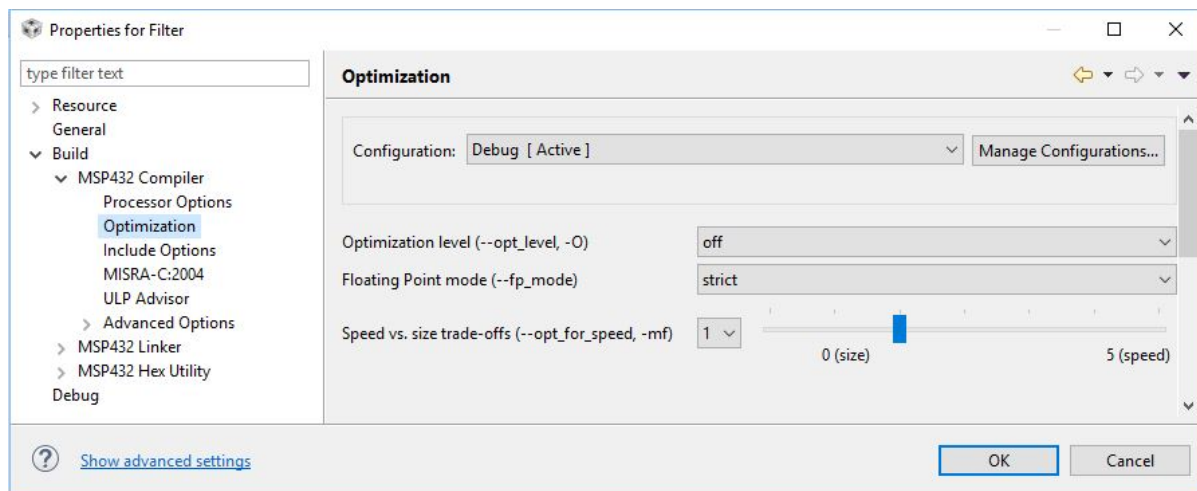
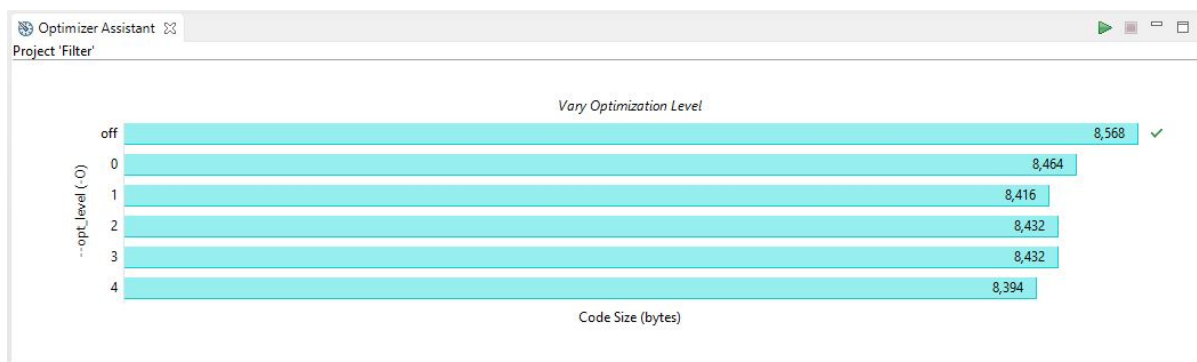
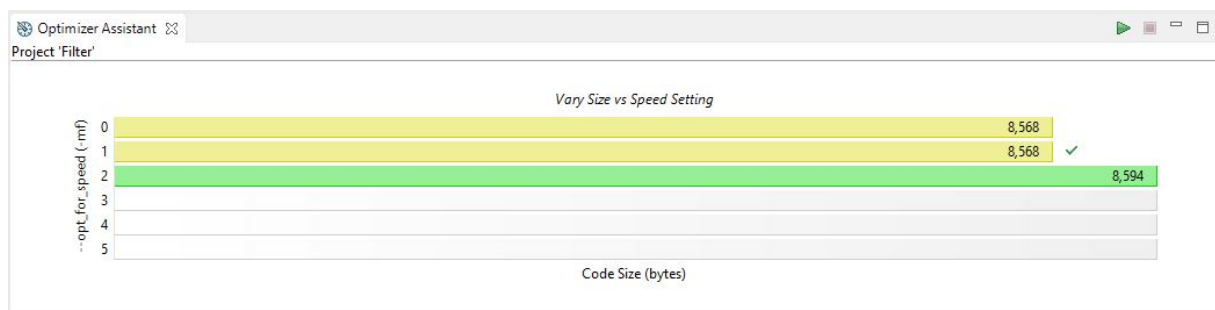


Figure 5.14: Compiler optimization view

Since experimenting with different compiler options can be sometimes hard, CCS also incorporates an Optimizer Assistant that is able to perform an exhaustive analysis in the code to decide which are the best suited optimization settings. Figures 5.15 and 5.16 show the Optimizer Assistant performing an analysis for optimization level and for the speed and code size trade-off.



**Figure 5.15:** Code size for different optimization level settings



**Figure 5.16:** Code size for different code size and speed trade-off settings

The Optimizer Assistant was used to determine which was the best combination of settings for each technique. Table 5.4 contains the optimization settings used for each technique using the Optimizer Assistant.

**Table 5.4:** Compiler optimization settings

Technique	Optimization level	Floating point mode	Speed vs size trade-off
FIR	4	Relaxed	2
Welch's method	4	Relaxed	2
DWT	3	Relaxed	2

Having defined the settings for optimizations, it is possible to test how much they improve each metric. Table 5.5 shows how execution time is improved by the tool. Since at this point no operating frequency is defined, the results are shown in clock cycles. Looking at the table it is evident that there is a considerable improvement in speed, in this case up to 36% of improvement was achieved just by changing the compiler settings. Table 5.6 shows how memory usage is improved. In this case, the tool can only improve the size of the code so it will affect only the Flash memory. Looking at the table it is evident an improvement for each technique even though is not as big as the one obtained for speed.

**Table 5.5:** Speed (clock cycles) optimization

Technique	Before optimization	After optimization	Improvement
FIR	2414827	1526264	36.80%
Welch's method	310863	232221	25.30%
DWT	352497	247968	29.65%

**Table 5.6:** Code size (bytes) optimization

Technique	Before optimization	After optimization	Improvement
FIR	5520	5362	2.86%
Welch's method	87432	87210	0.25%
DWT	6290	5882	6.48%

### 5.3.2 Power Consumption

Power consumption was also optimized for each feature extraction technique. Operating voltage and frequency are critical parameters in power consumption. In this case, both parameters were tuned in order to find the most suitable parameters combination.

Given that power is the product of voltage and current, if there is an application that consumes a certain amount of current, a lower supply voltage can help reduce the power consumption. The MSP432P01R requires a secondary core voltage (VCORE) for its internal digital operation in addition to the primary one applied to the device (VCC). The VCORE output is programmable with two predefined voltage levels. VCORE0 is approximately 1.2 V and is recommended for applications running in frequencies below 24 MHz while VCORE1 is close to 1.4 V and is recommended if the operating frequency is between 24 MHz and 48 MHz.

VCORE can be generated using two dedicated voltage regulators, there is a low-dropout voltage regulator (LDO) set by default and also an inductor-based DC-to-DC step-down switching regulator (DC-DC). The LDO has some advantages over the DC-DC as it is cost-effective, relatively noise free, faster to ramp up and down from low-power modes, among others. However, the LDO regulator might not be ideal for power savings, the DC-DC has a significant power improvement saving up to 45% compared to the LDO regulator.

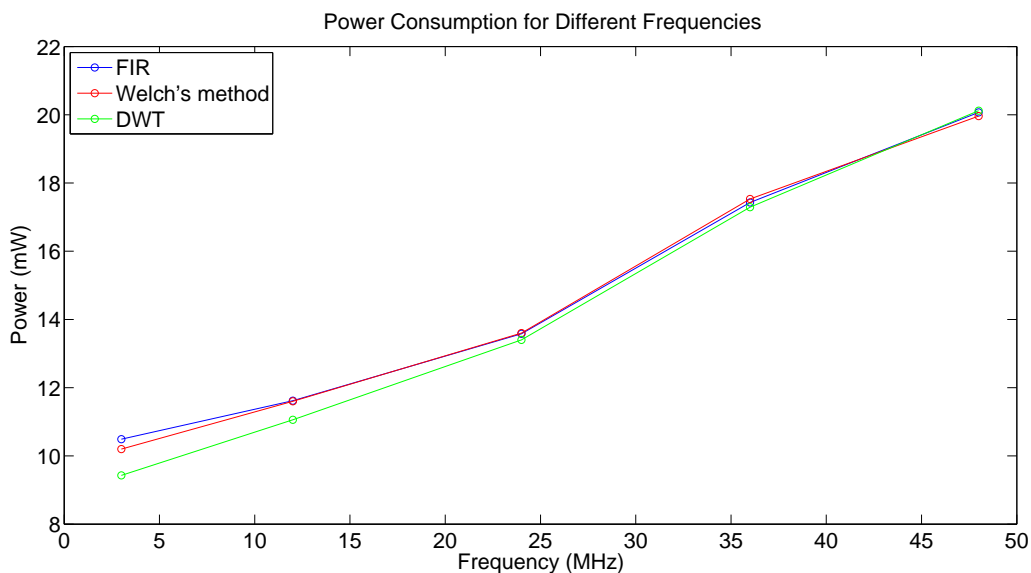
As it was also mentioned, the selection of the operating frequency of the system is an important element in low-power consumption applications. A reduction in the operating frequency will cause a reduction in current consumption. However, reducing the frequency could reduce overall system throughput or may increase energy consumption. Energy is the product between power and time, therefore there is a possibility that the same code can consume more power but less energy at higher frequencies because the power is consumed during a shorter period of time.

Since the selection of the VCORE is strictly tied to the operating frequency, for each of the feature extraction technique different frequencies and hence operating voltages were evaluated to get the power and energy consumption. Because the application requires to reduce energy consumption as much as possible, the DC-DC regulator was selected over the LDO.

Tables 5.7 and Figure 5.17 show the results of the measurements for power in five different frequencies, starting in the 3 MHz default frequency up to the maximum of 48 MHz. Just as expected it can be seen that power consumption rises when frequency does.

**Table 5.7:** Power (mW) measurements for optimization

Technique	Frequency (MHz)				
	3	12	24	36	48
FIR	10.49	11.62	13.58	17.43	20.07
Welch's method	10.20	11.60	13.60	17.53	19.96
DWT	9.43	11.06	13.40	17.29	20.12

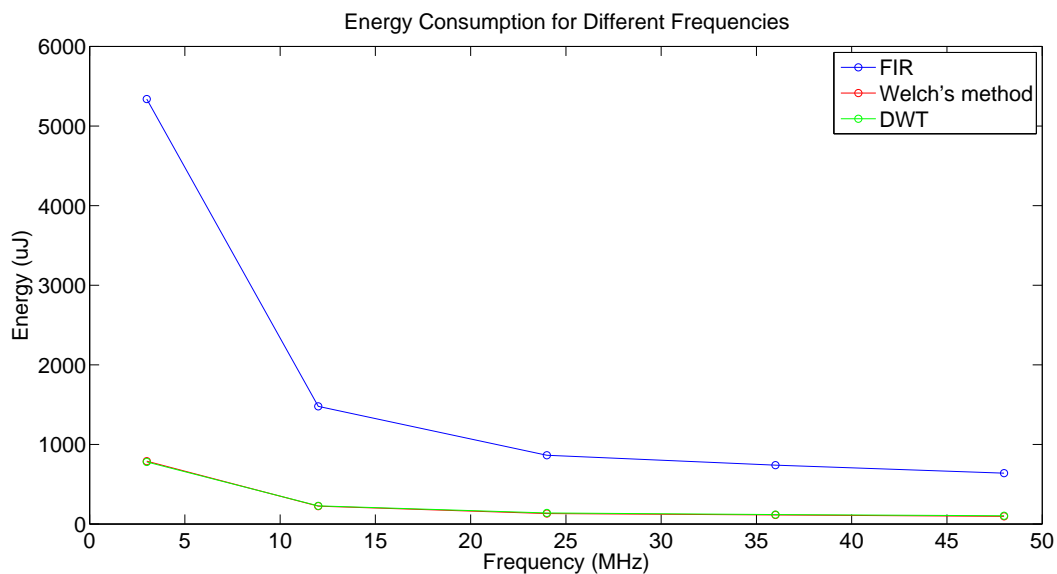


**Figure 5.17:** Power measurements for optimization

Based only on the power readings the ideal frequency to choose use would be the lowest, however as mentioned before, a low frequency produces a slower execution and therefore it consumes power for more time. It is necessary then to measure not only power but also the time for how long power is consumed to estimate energy. Looking at the energy plot in Figure 5.18 energy consumption goes down as frequency increases. It means that even though more power is consumed at higher frequencies, it is consumed for a shorter time period. Based on these experiments the selected operating frequency is 48 MHz and the core voltage setting VCORE1.

**Table 5.8:** Energy (uJ) measurements for optimization

Technique	Frequency (MHz)				
	3	12	24	36	48
FIR	5336.84	1477.93	863.61	738.97	638.17
Welch's method	789.55	224.48	131.59	113.08	96.57
DWT	779.85	228.54	138.45	119.09	103.94



**Figure 5.18:** Energy measurements for optimization

As it is explained in Chapter 2, feature extraction is just one stage in the operation of an EEG. Therefore, at this point it is only possible to perform a partial power optimization and not a whole application optimization because it is necessary to put all the stages together and also consider interdependencies between modules. In the whole application optimization of the EEG, power consumption can be considerably reduced by including low-power modes, minimizing transitions between modes, optimizing memory accesses, terminating and configuring I/Os or even enabling and disabling modules whether they are needed or not. In [8] an application report about design of Ultra-Low-Power (ULP) applications with MSP432<sup>TM</sup> microcontrollers can be consulted for further details.



# Chapter 6

## Evaluation of Feature Extraction Techniques

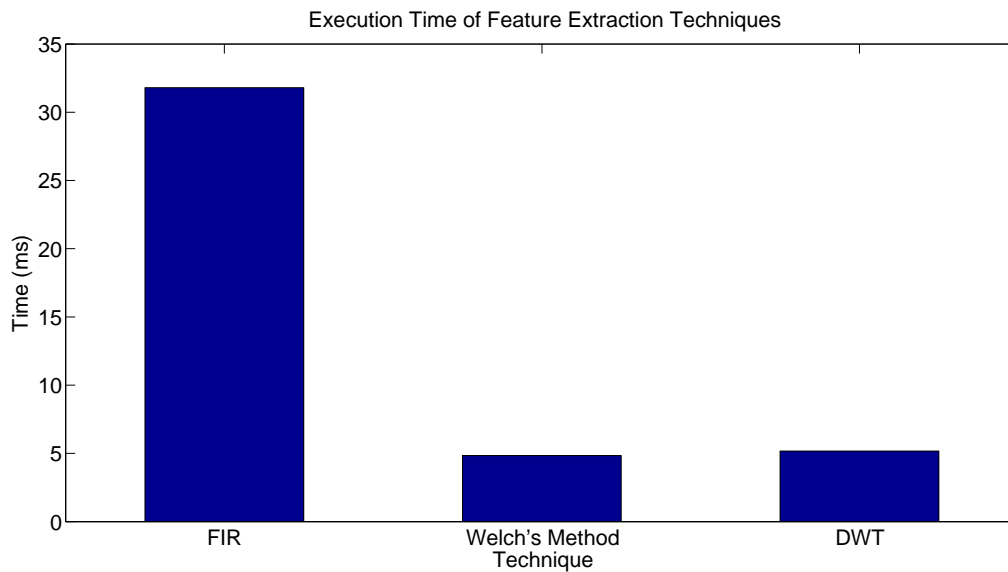
After exploring, simulating and implementing different feature extraction techniques, it is possible to evaluate them and analyze how each would fit in an IoT implementation of an EEG. Execution time, memory usage and power consumption are measured and compared in order to find how the selected techniques perform not only in each specific parameter but also taking them as a whole, keeping in mind that the EEG device should face computation capability and energy capacity in the best way possible. In order to evaluate the techniques in equal conditions, the test signal in Figure 3.1 was provided to each code as input. Every measurement was taken following the procedures explained in Chapter 4.

### 6.1 Execution Time

CPU cycles were measured and time was computed using Equation 4.1. The 48 MHz operating frequency defined during the optimization was also set. Table 6.1 and Figure 6.1 show the results of the execution time measurements for each algorithm.

**Table 6.1:** Execution time evaluation

Technique	CPU cycles	Time (ms)
FIR	1526264	31,80
Welch's method	232221	4,84
DWT	247968	5,17



**Figure 6.1:** Execution time evaluation

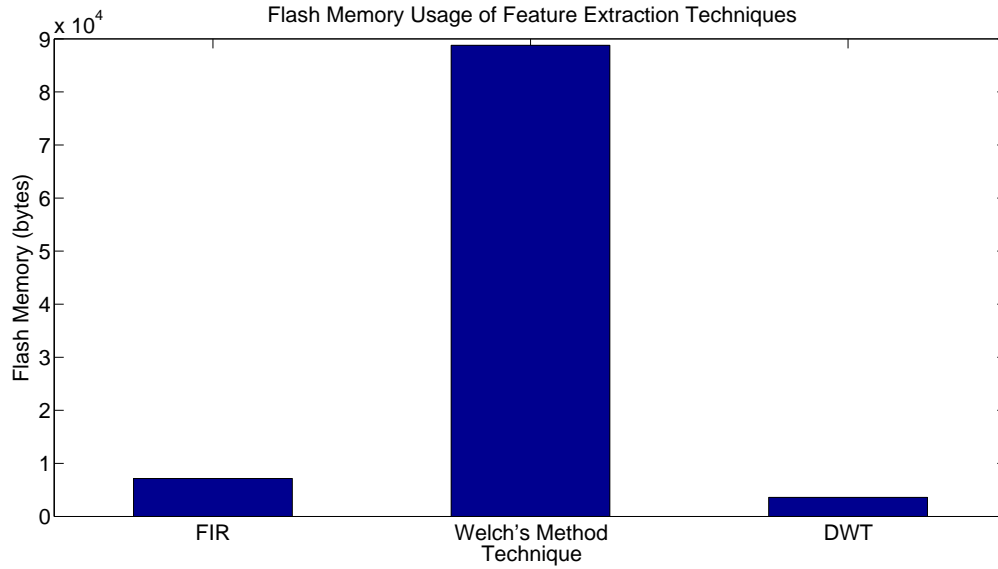
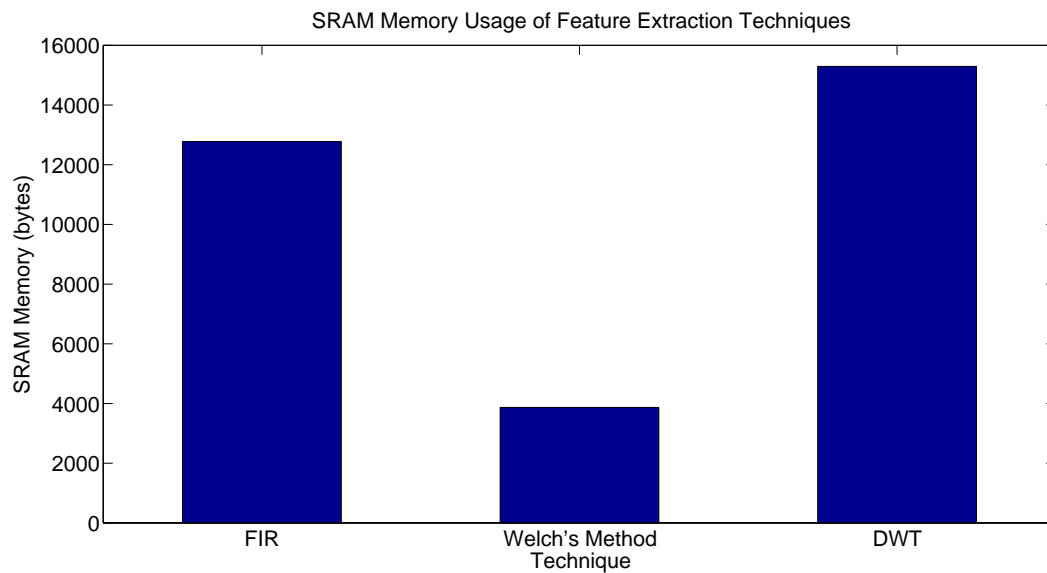
The execution time is desired to be as low as possible, a fast algorithm would let the application spend less time in active mode and therefore spend more time in passive mode saving energy. When comparing each technique, FIR takes considerably more time to be executed than Welch's method or DWT that last about the same. FIR execution time is approximately 6 times slower than its counterparts, this difference is mostly due to the dynamics of the algorithm. FIR is not able to provide information about all the frequency bands simultaneously, on the contrary it must extract information about each band one at the time. The process of filtering, power computation and averaging is carried out once for every band. Even if information about only one band wants to be computed, it is faster to use any of the other two approaches. When comparing Welch's method and DWT, no big difference is found since Welch's method is only 0.33 ms faster than DWT. From the execution time point of view either Welch's method or DWT are better suited for the IoT EEG implementation.

## 6.2 Memory Usage

Flash and SRAM memory measurements along with the usage over the available memory in the platform are shown in Table 6.2. For the SRAM the measurement is decomposed in static and dynamic memory usage. Figures 6.2 and 6.3 depict a comparison for the Flash and SRAM respectively.

**Table 6.2:** Memory usage (bytes) evaluation

Technique	Flash (bytes)		SRAM (bytes)			
	Total	Usage	Static	Dynamic	Total	Usage
FIR	7154	2.73%	11240	1536	12776	19.49%
Welch's method	88786	33.88%	2332	1536	3868	5.90%
DWT	3582	1.37%	4228	11062	15290	23.33%

**Figure 6.2:** Flash memory usage evaluation**Figure 6.3:** SRAM memory usage evaluation

Looking at the Flash memory measurements, there is a big difference between Welch's method and the rests of techniques. In this case, memory usage for Welch's method is around one third of the whole memory available in the platform. The size of the code for Welch's method is bigger because of the use of the FFT functions taken from the DSP library. In order to perform the FFT, this library needs to include a big amount of lookup tables that take 78936 bytes of memory. For the other two techniques Flash memory consumption is low if compared to Welch's method and is not bigger than 3%.

For the SRAM memory consumption Welch's method is no longer the most consuming technique but the lowest. For this technique SRAM memory usage is only around 5% of the available memory in the platform. Most memory is spent holding the input and output arrays. FIR and DWT posses similar total consumption but differently distributed. In the case of FIR most memory in SRAM is static allocated because the algorithm needs to hold the input array, four arrays for the filter coefficients and also four arrays to store the power in frequency bands. DWT consumes most memory dynamically, static memory is dedicated to store input, output and filter coefficients arrays while dynamic memory is exploited in this algorithm to create temporal arrays to store the results of the decomposition and sub-sampling for the approximation and details coefficients in each level before storing them to the decomposition array.

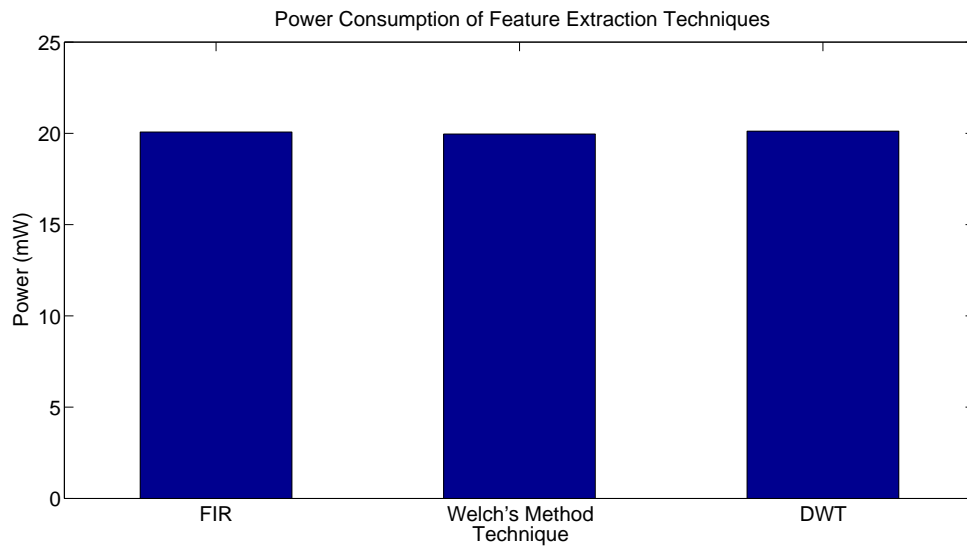
Looking at the results, if terms of Flash memory DWT and FIR are better suited for the application. Regarding SRAM memory consumption, Welch's method performs better than the others.

## 6.3 Power Consumption

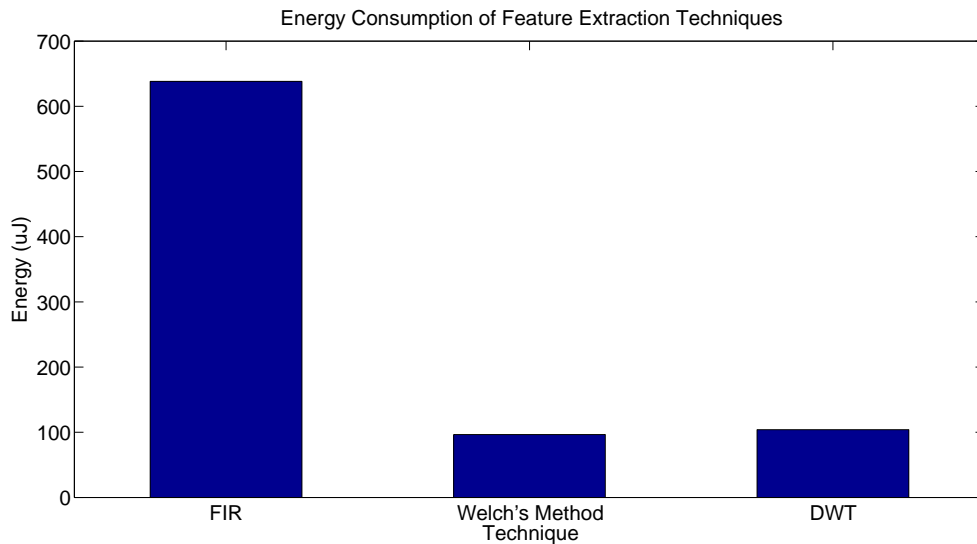
To evaluate power consumption both power and energy were measured for each of the techniques, the results can be observed in Table 6.3. Figures 6.4 and 6.5 show a graphical representation of power and energy respectively for each feature extraction technique.

**Table 6.3:** Power consumption evaluation

Technique	Power (mW)	Energy (uJ)
FIR	20.07	638.17
Welch's method	19.96	96.57
DWT	20.12	103.94



**Figure 6.4:** Power consumption evaluation



**Figure 6.5:** Energy consumption evaluation

Power consumption was very similar for each of the techniques and close to 20 mW. This similarity between measurements was expected and can be explained since each technique is running at the same frequency, operating voltage, using the same voltage regulator and making use of the same computation resources. Because the power measurements do not provide relevant information about the impact in energy capacity, the time for how long that power is consumed must be taken into account as well. The energy measurements show that since energy consumption is tied to execution time and power is almost the same for each technique, execution time and energy measurements show similar results. FIR consumes around 6 times more energy than the other algorithms because it is slower, while Welch's method and DWT perform in a similar way. Therefore, in terms of power and energy both Welch's method and DWT perform satisfactorily.

## 6.4 Overall Evaluation

After evaluating each of the metrics in a separate way, it is important then to carry out a general evaluation to put the results together. In terms of execution time, Welch's method and DWT performed similarly between them and considerably better than FIR. For the memory usage, Flash and SRAM were individually analyzed. In terms of Flash memory, DWT and FIR showed good performance while for the SRAM, it was Welch's method which performed better. Power consumption was almost the same for each technique and therefore energy was measured to show similar results to execution time where Welch's method and DWT perform better than FIR.

When all the metrics are seen as a whole, a bigger picture of the performance of the feature extraction techniques in an IoT implementation can be obtained. Table 6.4 gives a better picture of the overall evaluation by showing if the technique performs good, bad or even in the same way when compared to the rest of the techniques.

**Table 6.4:** Overall evaluation of feature extraction techniques

Technique	Execution time	Memory usage		Power consumption	
		Flash	SRAM	Power	Energy
FIR	✗	✓	✗	=	✗
Welch's method	✓	✗	✓	=	✓
DWT	✓	✓	✗	=	✓

# Chapter 7

## Conclusions

This work evaluated FIR, Welch's method and DWT as feature extraction techniques for an IoT EEG in terms of execution time, memory usage and power consumption on a microcontroller implementation. As a result of an extensive research, it was found that there are many feature extraction techniques that use filters, FFT or even DWT concepts during operation, and therefore several approaches were analyzed and compared. Using the findings of the investigation and the results from software simulations, those approaches were explored leading to the selection of FIR, Welch's method and DWT as the techniques to evaluate more closely. Those techniques were implemented, validated and optimized before running the measurements that would contrast their behavior.

The evaluation results showed that even though FIR can be used as a feature extraction technique, it may not be the best suited for an IoT implementation due to its poor speed and energy performance. On the other hand, since Welch's method and DWT performed similarly in those fields, the use of one or another will depend on the available resources and implementation. If Flash memory is critical, then DWT is preferred over Welch's method. If SRAM memory usage is critical, Welch's method can suit better the application. Specifically for the IoT EEG, the factor that will determine if Flash or SRAM is critical in the application is the total memory consumption of the rest of the EEG stages.

The study showed that if an application is focused on the content of one specific band, it could be faster and more energy-efficient to use certain techniques to extract features from all the bands at one time than using one technique like FIR that lets the user extract features from just one band if desired. In terms of memory, the results showed how the nature of each algorithm finds more suitable to take either memory from SRAM or Flash during their operation. In terms of power, the measurements suggest that algorithms like the evaluated techniques that use similar processing cores will consume almost the same average power, so the energy consumption will be tied to execution time.

The importance of this findings rely not only in providing three different implementations of processing cores that are commonly used to extract features from EEG but also in setting a reference of how they behave in different aspects like execution time, memory usage and power consumption. This knowledge is useful in the IoT EEG as well as in any other EEG implementation because any of this cores can be incorporated into the design knowing with certainty their impact in relevant areas. This makes easier the selection of one technique that may fit better in the design based on the final application requirements and available resources.

Future work should incorporate the rest of the EEG stages along with a full application optimization. It would also be interesting to evaluate each technique in terms their actual performance to determine mental states when combined with the other stages in the EEG. In that way, the results of the evaluation presented in this document and a evaluation of performance of the system can lead to a better criteria in order to chose the optimal feature extraction technique for the specific application in which the IoT EEG device will perform. Another suggestion for further work is to explore an alternative hardware implementation (FPGA, ASIC) of the techniques and compare it to the results of the microcontroller implementation presented in this document.



# Bibliography

- [1] A. S. Al-Fahoum and A. A. Al-Fraihat. Methods of EEG Signal Features Extraction Using Linear Analysis in Frequency and Time-Frequency Domains. *ISRN Neuroscience*, 2014(730218):7, Feb 2014.
- [2] N. K. Al-Qazzaz, S. Ali, S. A. Ahmad, M. S. Islam, and M. I. Ariff. Selection of mother wavelets thresholding methods in denoising multi-channel EEG signals during working memory task. In *Biomedical Engineering and Sciences (IECBES), 2014 IEEE Conference on*, pages 214–219, Dec 2014.
- [3] ARM. Cortex Microcontroller Software Interface Standard [online]. 2015. URL <http://www.keil.com/pack/doc/CMSIS/General/html/index.html>.
- [4] L. Astolfi, F. Cincotti, D. Mattia, F. De Vico Fallani, S. Salinari, G. Vecchiato, J. Toppi, C. Wilke, A. Doud, H. Yuan, B. He, and F. Babiloni. Imaging the social brain: multi-subjects EEG recordings during the "Chicken's game". In *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology*, pages 1734–1737, Aug 2010.
- [5] F. Babiloni, F. Cincotti, D. Mattia, M. Mattiocco, S. Bufalari, F. De Vico Fallani, A. Tocci, L. Bianchi, M. G. Marciani, V. Meroni, and L. Astolfi. Neural Basis For The Brain Responses To The Marketing Messages: an High Resolution EEG Study. In *Engineering in Medicine and Biology Society, 2006. EMBS '06. 28th Annual International Conference of the IEEE*, pages 3676–3679, Aug 2006.
- [6] N. Brodu, F. Lotte, and A. Lécuyer. Comparative study of band-power extraction techniques for Motor Imagery classification. In *Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB), 2011 IEEE Symposium on*, pages 1–6, Apr 2011.
- [7] C. Chesnutt. Feature Generation of EEG Data Using Wavelet Analysis. Master's thesis, Texas Tech University, 2012.
- [8] D. Dang and A. Lele. Designing an Ultra-Low-Power (ULP) Application With MSP432™ Microcontrollers, 2015. URL <http://www.ti.com/lit/an/slaa668/slaa668.pdf>.

- [9] Julien Castet Eduardo Reck Miranda. *Guide to Brain-Computer Music Interfacing*. Springer, 2014.
- [10] O. Faust, R. U. Acharya, A. R. Allen, and C. M. Lin. Analysis of EEG signals during epileptic and alcoholic states using AR modeling techniques. *IRBM*, 29, 2008.
- [11] Ary L. Goldberger, Luis A. N. Amaral, Leon Glass, Jeffrey M. Hausdorff, Plamen Ch. Ivanov, Roger G. Mark, Joseph E. Mietus, George B. Moody, Chung-Kang Peng, and H. Eugene Stanley. Physiobank, PhysioToolkit, and Physionet. *Circulation*, 101(23):e215–e220, 2000. URL <http://circ.ahajournals.org/content/101/23/e215>.
- [12] P. Herman, G. Prasad, T. M. McGinnity, and D. Coyle. Comparative Analysis of Spectral Approaches to Feature Extraction for EEG-Based Motor Imagery Classification. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 16(4):317–326, Aug 2008.
- [13] Texas Instruments. MSP EnergyTrace Technology [online]. 2016. URL <http://www.ti.com/tool/energytrace>.
- [14] Texas Instruments. MSP432P401R LaunchPad [online]. 2016. URL <http://www.ti.com/tool/msp-exp432p401r>.
- [15] M. R. N. Kousarrizi, A. A. Ghanbari, M. Teshnehlab, M. A. Shorehdeli, and A. Gharaviri. Feature Extraction and Classification of EEG Signals Using Wavelet Transform, SVM and Artificial Neural Networks for Brain Computer Interfaces. In *Bioinformatics, Systems Biology and Intelligent Computing, 2009. IJCBS '09. International Joint Conference on*, pages 352–355, Aug 2009.
- [16] MathWorks. Matlab [online]. 1994. URL <http://www.matlab.com>.
- [17] A. Medl. Time Frequency and Wavelets in Biomedical Signal Processing. *IEEE Engineering in Medicine and Biology Magazine*, 17(6):15–97, Nov 1998.
- [18] A. C. Merzagora, S. Bunce, M. Izzetoglu, and B. Onaral. Wavelet analysis for EEG feature extraction in deception detection. In *Engineering in Medicine and Biology Society, 2006. EMBS '06. 28th Annual International Conference of the IEEE*, pages 2434–2437, Aug 2006.
- [19] M. Murugappan, N. Ramachandran, and Y. Sazali. Classification of human emotion from EEG using discrete wavelet transform. *Biomedical Science and Engineering*, 2010.
- [20] J. Musson and J. Li. A Comparative Survey of PSD Estimation Methods for EEG Signal Analysis. Technical report, Old Dominion University, 2011.

- [21] I. Omerhodzic, S. Avdakovic, A. Nuhanovic, and K. Dizdarevic. Energy Distribution of EEG Signals: EEG Signal Wavelet-Neural Network Classifier. *World Academy of Science, Engineering and Technology*, 2010.
- [22] A. Oppenheim, A. Willsky, and S. H. Nawab. *Signals and Systems*. Prentice Hall, 2nd edition, 1998.
- [23] G. Pfurtscheller, C. Neuper, C. Guger, W. Harkam, H. Ramoser, A. Schlogl, B. Obermaier, and M. Pregenzer. Current trends in Graz brain-computer interface (BCI) research. *IEEE Transactions on Rehabilitation Engineering*, 8(2):216–219, Jun 2000.
- [24] Physionet. EEG Motor Movement / Imagery Dataset [online]. 2009. URL <http://physionet.org/physiobank/database/eegmmidb/>.
- [25] A. Prochazka, J. Kukal, and O. Vysata. Wavelet transform use for feature extraction and EEG signal segments classification. In *Communications, Control and Signal Processing, 2008. ISCCSP 2008. 3rd International Symposium on*, pages 719–722, March 2008.
- [26] Y. Renard, F. Lotte, G. Gibert, M. Congedo, E. Maby, V. Delannoy, O. Bertrand, and A. Lécuyer. OpenViBE: An Open-Source Software Platform to Design, Test, and Use Brain 8211; Computer Interfaces in Real and Virtual Environments. *Presence*, 19(1):35–53, Feb 2010.
- [27] N. Robinson and A. P. Vinod. Bi-directional imagined hand movement classification using low cost eeg-based bci. In *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on*, pages 3134–3139, Oct 2015.
- [28] S. Sahoo, S. Mohanty, and T. Sahoo. Association between psychology and technical education by EEG. In *Advance Computing Conference (IACC), 2014 IEEE International*, pages 1315–1321, Feb 2014.
- [29] G. Schalk. BCI2000 [online]. 2016. URL [www.bci2000.org](http://www.bci2000.org).
- [30] G. Schalk, D. J. McFarland, T. Hinterberger, N. Birbaumer, and J. R. Wolpaw. BCI2000: a general-purpose brain-computer interface (BCI) system. *IEEE Transactions on Biomedical Engineering*, 51(6):1034–1043, Jun 2004.
- [31] B. Wang, C. M. Wong, F. Wan, P. U. Mak, P. I. Mak, and M. I. Vai. Comparison of different classification methods for EEG-based brain computer interfaces: A case study. In *Information and Automation, 2009. ICIA '09. International Conference on*, pages 1416–1421, Jun 2009.
- [32] S. Winder. *Analog and Digital Filter Design*. Elsevier Science, 2nd edition, 2002.
- [33] J. Wu, J. Zhang, and L. Yao. An automated detection and correction method of EOG artifacts in EEG-based BCI. In *Complex Medical Engineering, 2009. CME. ICME International Conference on*, pages 1–5, Apr 2009.

- 
- [34] G. z. Yan, B. h. Yang, and S. Chen. Automated and Adaptive Feature Extraction for Brain-Computer Interfaces by using Wavelet Packet. In *2006 International Conference on Machine Learning and Cybernetics*, pages 4248–4251, Aug 2006.
- [35] A. Zabidi, W. Mansor, Y. K. Lee, and C. W. N. F. Che Wan Fadzal. Short-time Fourier Transform analysis of EEG signal generated during imagined writing. In *System Engineering and Technology (ICSET), 2012 International Conference on*, pages 1–4, Sept 2012.

# Appendix A

## Acronyms and Abbreviations

<b>ASIC</b>	Application-Specific Integrated Circuit
<b>BCI</b>	Brain-Computer Interface
<b>CPU</b>	Central Processing Unit
<b>CMSIS</b>	Cortex Microcontroller Software Interface Standard
<b>CCS</b>	Code Composer Studio
<b>DFT</b>	Discrete Fourier Transform
<b>DSP</b>	Digital Signal Processing
<b>db4</b>	Daubechies 4 wavelet
<b>DC-DC</b>	Dc-to-DC step-down switching regulator
<b>DWT</b>	Discrete Wavelet Transform
<b>ECG</b>	Electrocardiography
<b>EEG</b>	Electroencephalography, Electroencephalogram
<b>EKG</b>	Electrogastrography
<b>EMG</b>	Electromyography
<b>EOG</b>	Electrooculography
<b>ET</b>	EnergyTrace
<b>ET+</b>	EnergyTrace+
<b>FIR</b>	Finite Impulse Response
<b>FFT</b>	Fast Fourier Transform
<b>FPGA</b>	Field Programmable Gate Array
<b>IDE</b>	Integrated Development Environment

**IIR** Infinite Impulse Response

**IoT** Internet of Things

**LDO** Low-dropout voltage regulator

**MEG** Magnetoencephalography

**RAM** Random Access Memory

**SRAM** Static Random Access Memory

**SIMD** Single Instruction Multiple Data

**PSD** Power Spectral Density

**ULP** Ultra-low Power

**VCC** Main core voltage

**VCORE** Secondary core voltage

**VCORE0** Low secondary core voltage

**VCORE1** High secondary core voltage

**WPT** Wavelet Packet Transform

# Appendix B

## Feature Extraction Codes

This appendix contains the codes that were implemented for each of the studied feature extraction techniques in the MSP432P401R LaunchPad. All of them are written in C and every computation stage is specified by comments to ease the understanding of the codes.

### B.1 FIR

```
/*
 * Filter EEG Feature Extraction
 * -----
 * This code takes an EEG input signal and computes
 * power for the Delta, Theta, Alpha and Beta
 * frequency bands.
 *
 * Author: David Barahona Pereira
 *
 * Email address: davidbp.13@gmail.com
 *
 * Institution: Costa Rica Institute of Technology
 *              Karlsruhe Institute of Technology
 *
 * Last revision: 16/09/2016
 */

#include <stdio.h>
#include "msp.h"
#include "arm_math.h"
#include "math_helper.h"
#include "driverlib.h"

// Elements in input array
#define TEST_LENGTH_SAMPLES 512
// Number of taps in FIR filter
#define NUM_TAPS 61
```

```

// Samples used for each computation for averaging
#define MOVING_AVERAGE_SAMPLES (TEST_LENGTH_SAMPLES / 32)
// Samples used for each FIR calculation
#define BLOCK_SIZE (TEST_LENGTH_SAMPLES / 8)

// EEG reading
extern float32_t EEG_Input[TEST_LENGTH_SAMPLES];
// Filter coefficients for Delta band
extern float32_t Delta_Coeffs[NUM_TAPS];
// Filter coefficients for Theta band
extern float32_t Theta_Coeffs[NUM_TAPS];
// Filter coefficients for Alpha band
extern float32_t Alpha_Coeffs[NUM_TAPS];
// Filter coefficients for Beta band
extern float32_t Beta_Coeffs[NUM_TAPS];
// Array to store power in Delta Band
float32_t Delta_Power[TEST_LENGTH_SAMPLES];
// Array to store power in Theta Band
float32_t Theta_Power[TEST_LENGTH_SAMPLES];
// Array to store power in Alpha Band
float32_t Alpha_Power[TEST_LENGTH_SAMPLES];
// Array to store power in Beta Band
float32_t Beta_Power[TEST_LENGTH_SAMPLES];

void Init(void);
void Band_Power(float32_t *inputF32, float32_t *outputF32,
float32_t *Coeffs);

int32_t main(void)
{
    Init();
    Band_Power(EEG_Input, Delta_Power, Delta_Coeffs);
    Band_Power(EEG_Input, Theta_Power, Theta_Coeffs);
    Band_Power(EEG_Input, Alpha_Power, Alpha_Coeffs);
    Band_Power(EEG_Input, Beta_Power, Beta_Coeffs);
}

/*
 * Function: Init
 * -----
 * Initialization routine for MSP432.
 *
 * returns: none
 */
void Init(void)
{
    // Stop watchdog timer
    WDCTL = WDTPW | WDTHOLD;
    // Core voltage selection
    PCM_setPowerState(PCM_AM_DCDC_VCORE1);
    // Operating frequency

```



```

        CS_setDC0Frequency(48000000);
        // Clock source selection
        CS_initClockSignal(CS_MCLK, CS_DCOCLK_SELECT,
        CS_CLOCK_DIVIDER_1);
    }

    /*
    * Function:  Band_Power
    * -----
    * Computes the power in a certain frequency band. The signal
    * is filtered using a FIR filter and then power is computed
    * and averaged. The frequency band is given by the filter
    * coefficients.
    *
    * *inputF32:  pointer to input signal array
    * *outputF32: pointer to the array that will store power
    * *Coeffs:    pointer to the filter coefficients array
    *
    * returns: none
    */
    void Band_Power(float32_t *inputF32, float32_t *outputF32,
    float32_t *Coeffs)
    {
        uint32_t i;
        uint32_t j;
        // Size of blocks to compute filtering
        uint32_t blockSize = BLOCK_SIZE;
        // Number of block that will be filtered
        uint32_t numBlocks = (TEST_LENGTH_SAMPLES / BLOCK_SIZE);
        // FIR state array
        float32_t firStateF32[BLOCK_SIZE + NUM_TAPS - 1];
        // Variable to calculate power
        float32_t Power_aux;
        // FIR instance
        arm_fir_instance_f32 S;

        // Call FIR init function to initialize the instance structure
        arm_fir_init_f32(&S, NUM_TAPS, (float32_t *)&Coeffs[0],
        &firStateF32[0], blockSize);

        // Call the FIR process function for every blockSize samples
        for(i=0; i < numBlocks; i++)
        {
            arm_fir_f32(&S, inputF32 + (i * blockSize),
            outputF32 + (i * blockSize), blockSize);
        }

        // Power computation
        for (i = 0; i < TEST_LENGTH_SAMPLES; i++)
        {
            outputF32[i] *= outputF32[i];
        }
    }

```

```

// Moving Average computation
for (i = 0; i < TEST_LENGTH_SAMPLES
- MOVING_AVERAGE_SAMPLES; i++)
{
    Power_aux = 0;
    for (j = 0; j < MOVING_AVERAGE_SAMPLES; j++)
    {
        Power_aux += outputF32[i + j];
    }
    outputF32[i] = Power_aux / MOVING_AVERAGE_SAMPLES;
}
for (i = TEST_LENGTH_SAMPLES - MOVING_AVERAGE_SAMPLES;
i < TEST_LENGTH_SAMPLES; i++)
{
    outputF32[i] = 0;
}
}

```

## B.2 Welch's method

```

/*
* FFT EEG Feature Extraction
* -----
* This code takes an EEG input signal and computes
* Power Spectral Density (PSD) using Welch's method.
*
* Author: David Barahona Pereira
*
* Email address: davidbp.13@gmail.com
*
* Institution: Costa Rica Institute of Technology
*              Karlsruhe Institute of Technology
*
* Last revision: 13/09/2016
*/

#include <stdio.h>
#include "msp.h"
#include "arm_math.h"
#include "arm_const_structs.h"
#include "driverlib.h"

// Elements in input array (Power of 2 and MINIMUM 256)
#define TEST_LENGTH_SAMPLES 512
// Sampling frequency of input data
#define SAMPLING_F 160
// Length for each window
#define WINDOW_LENGTH (TEST_LENGTH_SAMPLES / 4)
// Length of FFT array
#define FFT_LENGTH (WINDOW_LENGTH / 2)
// Number of overlapped windows

```

```

#define NUMBER_OF_WINDOWS ((2 * TEST_LENGTH_SAMPLES
/ WINDOW_LENGTH) - 1)

// EEG reading
extern float32_t EEG_Input[TEST_LENGTH_SAMPLES];
// Array to store PSD
float32_t PSD[FFT_LENGTH];

void Welch(float32_t *testInput, float32_t *testOutput);
void Init(void);

int32_t main(void)
{
    Init();
    Welch(EEG_Input, PSD);
}

/*
 * Function: Init
 * -----
 * Initialization routine for MSP432
 *
 * returns: none
 */
void Init(void)
{
    // Stop watchdog timer
    WDTCTL = WDTPW | WDTHOLD;
    // Core voltage selection
    PCM_setPowerState(PCM_AM_DCDC_VCORE1);
    // Operating frequency
    CS_setDCOFrequency(48000000);
    // Clock source selection
    CS_initClockSignal(CS_MCLK, CS_DCOCLK_SELECT,
    CS_CLOCK_DIVIDER_1);
}

/*
 * Function: Welch
 * -----
 * Computes the Power Spectral Density (PSD) of an input signal
 * using Welch's method. The algorithm performs an overlapping
 * windowing of the input signal, then computes the periodogram
 * for each window and finally calculates an average of all
 * periodograms to estimate PSD.
 *
 * *testInput: pointer to input signal array
 * *testOutput: pointer to the array that will store PSD
 *
 * returns: none
 */

```

```

void Welch(float32_t *testInput, float32_t *testOutput)
{
    uint32_t i;
    uint32_t j;
    // 0 for FFT / 1 for IFFT
    uint32_t ifftFlag = 0;
    // Size of FFT array
    uint32_t fftSize = FFT_LENGTH;
    // Variable for periodograms averaging
    float32_t testOutput_aux;
    // Bidimensional array to store overlapping windows
    float32_t windows[NUMBER_OF_WINDOWS][WINDOW_LENGTH];
    // Bidimensional array to store the one sided FFT of each window
    float32_t windows_fft[NUMBER_OF_WINDOWS][WINDOW_LENGTH];
    // Bidimensional array to store the magnitude of the FFT
    float32_t windows_mag[NUMBER_OF_WINDOWS][FFT_LENGTH];
    // FFT status
    arm_status status;
    // FFT instance
    arm_rfft_fast_instance_f32 S;

    // Windowing in input signal
    for (i = 0; i < NUMBER_OF_WINDOWS; i++)
    {
        for (j = 0; j < WINDOW_LENGTH; j++)
        {
            windows[i][j] = testInput[j +
                i * (WINDOW_LENGTH/2)];
        }
    }

    // FFT init routine
    status = arm_rfft_fast_init_f32 (&S, WINDOW_LENGTH);
    if(status != ARM_MATH_SUCCESS)
    {
        while(1);
    }

    // Computes the magnitude of the complex one sided FFT
    for (i = 0; i < NUMBER_OF_WINDOWS; i++)
    {
        arm_rfft_fast_f32(&S, windows[i], windows_fft[i],
            ifftFlag);
        arm_cmplx_mag_f32(windows_fft[i], windows_mag[i],
            fftSize);
    }

    // Periodogram calculation
    for (i = 0; i < NUMBER_OF_WINDOWS; i++)
    {
        for (j = 0; j < FFT_LENGTH; j++)

```

```

        {
            windows_mag[i][j] = (windows_mag[i][j] *
                                windows_mag[i][j]) / (FFT_LENGTH * SAMPLING_F);
        }
    }

    // Periodograms averaging
    for (i = 0; i < FFT_LENGTH; i++)
    {
        testOutput_aux = 0;
        for (j = 0; j < NUMBER_OF_WINDOWS; j++)
        {
            testOutput_aux += windows_mag[j][i];
        }
        testOutput[i] = testOutput_aux / NUMBER_OF_WINDOWS;
    }
    testOutput[0] /= 2;
}

```

## B.3 DWT

```

/*
 * DWT EEG Feature Extraction
 * -----
 * This code takes an EEG input signal and computes
 * a multilevel Discrete Wavelet Transform (DWT)
 * to extract power in Delta, Theta, Alpha and Beta
 * frequency bands.
 *
 * Author: David Barahona Pereira
 *
 * Email address: davidbp.13@gmail.com
 *
 * Institution: Costa Rica Institute of Technology
 *              Karlsruhe Institute of Technology
 *
 * Last revision: 16/09/2016
 */

#include <stdio.h>
#include "msp.h"
#include "arm_math.h"
#include "math_helper.h"
#include "driverlib.h"

// Elements in input array
#define INPUT_SIZE 512
// Decomposition level
#define LEVEL 4
// Coefficients in decomposition filters
#define FILTER_LENGTH 8

```

```

// EEG reading
extern float32_t EEG_Input[INPUT_SIZE];
// Low pass decomposition coefficients
extern float32_t Lo_D[FILTER_LENGTH];
// High pass decomposition coefficients
extern float32_t Hi_D[FILTER_LENGTH];
// Power calculated from the coefficients
float32_t Power[INPUT_SIZE];
// Array to keep track of each level in power array
float32_t Lengths[LEVEL + 2];

void Init(void);
void wavedec(float32_t *Input, float32_t *Coefficients,
float32_t *Lengths);

int32_t main(void)
{
    Init();
    wavedec(EEG_Input, Power, Lengths);
}

/*
 * Function: Init
 * -----
 * Initialization routine for MSP432
 *
 * returns: none
 */
void Init(void)
{
    // Stop watchdog timer
    WDTCTL = WDTPW | WDTHOLD;
    // Core voltage selection
    PCM_setPowerState(PCM_AM_DCDC_VCORE0);
    // Operating frequency
    CS_setDCOFrequency(3000000);
    // Clock source selection
    CS_initClockSignal(CS_MCLK, CS_DCOCLK_SELECT,
    CS_CLOCK_DIVIDER_1);
}

/* Function: wavedec
 * -----
 * Computes a multilevel discrete wavelet decomposition
 * of an input vector
 *
 * *Input: input signal array
 * Coefficients: output array that will store the coefficients
 * as [cAn, cDn, cDn-1, ..., cD1)
 * Lengths: output array that will store the length of each level
 * as [cAn_length, cDn_length, cDn-1_length, ..., cD1_length]

```

```
*
* returns: none
*/
void wavedec(float32_t *Input, float32_t *Coefficients,
float32_t *Lengths)
{
    uint32_t i;
    uint32_t j;
    // Size of array that will be decomposed
    uint32_t Samples;
    // Keeps track of where to place the result of each
    // decomposition
    uint32_t Index_Deco = 0;
    // Size of convolution of the input and filter
    uint32_t F_G_Size;

    // Size of approximation and details arrays
    uint32_t cA_cD_Size;
    // Number of taps in the filter
    uint32_t Taps = FILTER_LENGTH;
    // Size of intermediate decomposition array
    uint32_t Output_Size = 0;
    // Tracks position in coefficients array
    uint32_t Index_Coeff;
    // Pointers to filters, convolution and coefficients arrays
    float32_t *lo, *hi, *F, *G, *cA, *cD, *Decomposition;
    // Block sizes for Hi and Lo decomposition before
    // downsampling (after is the same /2)
    uint32_t Level_Size[LEVEL];

    // Initializes the array that will store the size for
    // the decomposition blocks
    Level_Size[0] = INPUT_SIZE + FILTER_LENGTH - 2;
    for(i = 1; i < LEVEL; i++)
    {
        Level_Size[i] = (Level_Size[i-1] / 2)
            + FILTER_LENGTH - 1;
    }

    // Calculates the size of the output decomposition array
    for(i = 0; i < LEVEL; i++)
    {
        Output_Size += Level_Size[i];
    }

    // Computes DWT for each level
    Decomposition = (float32_t *)malloc(sizeof(float32_t)
    * Output_Size);
    for (i = 0; i < LEVEL; i++)
    {
        // Takes input for the first level decomposition
        if(i == 0)
```

```

{
    Samples = INPUT_SIZE;
}
// Takes the low decomposition of the preceding level
else
{
    Samples = Level_Size[i - 1] / 2;
    Input = (float32_t *)malloc(sizeof(float32_t)
    * Samples);
    for(j = 0; j < Samples; j++)
    {
        Input[j] = Decomposition[j +
        Index_Deco];
    }
    Index_Deco += Level_Size[i - 1];
}

// Computes a one level decomposition
F_G_Size = Level_Size[i];
cA_cD_Size = F_G_Size / 2;

// Dynamic allocation of convolution and coefficients
// arrays
F = (float32_t *)malloc(sizeof(float32_t)*F_G_Size);
G = (float32_t *)malloc(sizeof(float32_t)*F_G_Size);
cA = (float32_t *)malloc(sizeof(float32_t)*cA_cD_Size);
cD = (float32_t *)malloc(sizeof(float32_t)*cA_cD_Size);

// Pointers to low and high pass decomposition filters
lo = &Lo_D[0];
hi = &Hi_D[0];

// Convolution between signal and the low and high pass
// filter
arm_conv_f32(Input, Samples, lo, Taps, F);
arm_conv_f32(Input, Samples, hi, Taps, G);

// Downsampling
for(j=0; j < cA_cD_Size; j++)
{
    cA[j] = F[2 * j + 1];
    cD[j] = G[2 * j + 1];
}

// Stores approximation and details coefficients in one
// array
for(j=0; j < cA_cD_Size; j++)
{
    Decomposition[j + Index_Deco] = cA[j];
    Decomposition[j + Index_Deco + cA_cD_Size]
    = cD[j];
}

```



```

    }

    // Memory release
    free(F); F = NULL;
    free(G); G = NULL;
    free(cA); cA = NULL;
    free(cD); cD = NULL;
    free(Input); Input = NULL;
}

// Computes coefficients as MATLAB
Index_Deco = Output_Size - Level_Size[LEVEL - 1];
Index_Coeff = 0;

for(i = 0; i < LEVEL; i++)
{
    // Moves decomposition details data to the coefficients
    // array
    for(j = 0; j < Level_Size[LEVEL - 1 - i] / 2; j++)
    {
        Coefficients[j + Index_Coeff] =
            Decomposition[j + Index_Deco];
    }

    // Stores decomposition approximation for the last level
    // and/or calculates indexes for next iteration
    if(i == 0)
    {
        Index_Deco += Level_Size[LEVEL - 1 - i] / 2;
        Index_Coeff += Level_Size[LEVEL - 1 - i] / 2;
        for(j = 0; j < Level_Size[LEVEL - 1 - i] / 2;
            j++)
        {
            Coefficients[j + Index_Coeff] =
                Decomposition[j + Index_Deco];
        }
        Index_Deco -= (Level_Size[LEVEL - 1 - i]
            + Level_Size[LEVEL - 2 - i]) / 2;
        Index_Coeff += Level_Size[LEVEL - 1 - i] / 2;
    }
    else
    {
        Index_Deco -= (Level_Size[LEVEL - 1 - i]
            + Level_Size[LEVEL - 2 - i]) / 2;
        Index_Coeff += Level_Size[LEVEL - 1 - i] / 2;
    }
}

// Builds an array containing wavelet sizes
for(i = 0; i < LEVEL + 1; i++)
{
    if(i >= 2)

```

```
        {
            Lengths[i] = Level_Size[LEVEL - i] / 2;
        }
        else
        {
            Lengths[0] = Level_Size[LEVEL - 1] / 2;
            Lengths[1] = Level_Size[LEVEL - 1] / 2;
            Lengths[LEVEL + 1] = INPUT_SIZE;
        }
    }

    // Power computation
    for (i = 0; i < INPUT_SIZE; i++)
    {
        Coefficients[i] *= Coefficients[i];
    }

    free(Decomposition); Decomposition = NULL;
}
```