Next-generation neuromodulator for epilepsy prevention

Athanasios Karapatis

Technische Universiteit Delft

TU Delft Delft University of Technology

Challenge the future

NEXT-GENERATION NEUROMODULATOR FOR EPILEPSY PREVENTION

by

Athanasios Karapatis

in partial fulfillment of the requirements for the degree of

Master of Science
in Computer Engineering

at the Delft University of Technology,
to be defended publicly on Monday July 13, 2015 at 13:30.

Thesis committee:

Advisor:	Dr. ir. C. Strydis,	Erasmus MC, Neuroscience Department
Advisor:	Dr. ir. W. A. Serdijn,	TU Delft, Section Bio-electronics
Member:	Dr. ir. G. Gaydadjiev,	TU Delft, Parallel and Distributed Systems
Member:	Dr. F. E. Hoebeek,	Erasmus MC, Neuroscience Department
Member:	Ir. R. M. Seepers,	Erasmus MC, Neuroscience Department

TU Delft Delft University of Technology

Abstract

Closed-loop neurostimulation systems have emerged as a prominent method for treating seizures. However, most of the proposed solutions do not consider the need for fast (real-time) seizure detection or their energy overheads, resulting in systems not suitable for wearable or implantable applications. This thesis describes the design and implementation of a novel closed-loop system that is capable of real-time seizure detection and suppression, while requiring minimal power and energy consumption. The proposed system utilizes a complex Morlet wavelet in combination with a thresholding mechanism to detect the presence of ictal-activity in ECoG signals. We evaluate our system in terms of detection performance (sensitivity, specificity and delay) considering various filter parameters, such as the filter order and various (static) detection thresholds. Additionally, we consider the system's suitability for implantable applications by evaluating its computational overheads (execution time, energy consumption) when executed on the SiMS low-power processor. We show that decreasing the filter order results in less accurate detection (sensitivity, specificity), a faster detection (delay), and less overheads. In addition, we show that we may further improve the detection accuracy and delay with minimal overheads by considering an input-dependent (adaptive) threshold mechanism. Furthermore, we show that we can effectively trade-off detection accuracy and energy consumption: For example, shrinking filter order by 70% results in a decrease in detection accuracy of only 1%, while allowing us to obtain an improvement in delay by 190 ms (from 710 ms to 520 ms) and in energy consumption by 70% (from 5.04μJ to 1.51μJ). Compared to related work, we show that we can detect seizures significantly faster (492 ms, compared to 970 ms) with the same sensitivity (94%) and at a minimal decrease in specificity of 4.6% (93.60% compared to 98.2%). A prototype implementation of the closed-loop system has successfully been applied in *in-vivo* experiments, demonstrating its potential for epilepsy treatment.

Dedicated to my family and friends who believed in me…

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

There are many people that contributed to completion of this thesis and for whom I wish to express my thanks.

First and foremost, I would like to express my very great appreciation to my supervisor, Christos Strydis, for his guidance, his invaluable advice, and even for our long disagreements, from which I learned a great deal. I would also like to offer thanks to Robert Seepers for his constructive feedback, his encouragement and the many hours he devoted to improve this report. From this point on, I will always have a little imaginary demon, called Robert, on my shoulder asking me "why is this?".I am also very grateful to Marijn van Dongen. Without him none of this would be possible, for it was his idea to use the wavelet filter and also created all the necessary analog boards for the prototypes. His infectious enthusiasm and positive attitude were a great motivation.

I also wish to thank Wouter Serdijn for allowing me to graduate in his group and handling all the necessary procedures. Thanks also go to Freek Hoebeek and his lab members for coming with the problem of seizure detection and letting me work at their experiments.

I am eternally grateful to all my friends, who continuously provided me with moral support throughout this thesis and never stop believing in me.

Last but certainly not least, I thank my family for constantly encouraging me and supporting me in every step. Needless to say, I would never have made it this far without them.

Thank you all.

Athanasios Karapatis
Delft, July 2015

1

INTRODUCTION

An epileptic seizure is a transient occurrence of signs and/or symptoms due to abnormal excessive or synchronous neuronal activity in the brain [2]. Absence seizures are one form of epileptic seizures which typically last a few (<20) seconds and mostly affect children, although there are cases where adults are affected [3]. The typical manifestations of absence seizures are a transient impairment of consciousness (with abrupt onset and offset) accompanied by one or more other symptoms such as staring, behavioral arrest, eyelid fluttering, or hand/ face automatisms [4]. These problems can result in self-injury of the patients as well as in cognition issues, including attentional problems, anxiety, depression, social isolation, and low self-esteem [4, 5] .

Conventional methods of seizure treatment involve medications and resective surgery. However, the cost of developing these medications is enormously high and large pharmaceutical companies tend to stop their research efforts [6]. Resective surgery is used for medically intractable epilepsy cases, however, it comes with a high-risk of leading to complications, such as deficits memory, speech, visual, sensory, or motor control [7].

In the mean time, a new alternative of treating Central Nervous System (CNS) disorders, such as epilepsy, has emerged. These treatments focus on interpreting how the brain interacts and try to intervene by restoring its correct behavior using (electrical) stimulation. An example of such a system, that is actively aware of physiological changes and can stimulate in real-time, is called a closed-loop system. For epileptic seizures, the current neurostimulation solutions have a symptom's improvement rate of 40%-50% in patients and are only applicable to certain types of seizures [8, 9]. Consequently, there is still room for improvement in better understanding the brain's functions, creating more efficacious detection techniques, and considering various types of seizures.

As neurostimulation becomes an established therapy for drug-resistant seizures, a new challenge rises to build effective and efficient implantable devices that can detect and suppress them. Thus, the need is created to have accurate detections, avoiding over-stimulation that can cause implications, and to detect them fast enough before the seizure's symptoms manifest. In addition, low energy overheads are required to make such a device viable for implantable applications.

Part of this work also considered the implementation of a neurostimulator prototype for tinnitus treatment. However, it was not fully evaluated and is briefly presented in Appendix D.

1.1. PROBLEM STATEMENT

A closed-loop system, for seizure treatment, must be capable of rapid detections in order to prevent the seizure symptoms from manifesting. Additionally, it must have a low implementation complexity

whilst maintaining reasonable accuracy, in order to be applicable for implantable neural prostheses.

Several other works propose closed-loop systems for epilepsy prevention, which employ various detection techniques and then use electrical or optogenetic stimulation to suppress the development of a seizure [10], [11], [12]. However, most solutions fail to consider the needs for low power and fast detection. As a result, these solutions either consume too much energy for implantable application [13], or high detection delays unfit for seizure treatment (e.g. [14]). As such, we need to propose a detection mechanism that meets these requirements.

1.2. THESIS OBJECTIVES

The main objective of this thesis is to propose a solution that detects seizures, considering the aspects of detection delay and applicability for ultra-low power (implantable) applications. While there many other studies that suggest solutions to the detection problem, few focus on the other two aspects. As such, we need to answer the following question:

"Can we detect seizures fast and reliably using a seizure-detection system which is suitable for implantable (ultra-low power) application?"

In order to answer this questions, the following steps are undertaken:

1. We propose a detection mechanism which uses a complex Morlet wavelet filter and one of two threshold mechanisms, static and adaptive;
2. A working prototype of the closed-loop system has been applied in *in-vivo* experiments, demonstrating its potential for epilepsy treatment;
3. The detection mechanism We evaluate the detection mechanism in terms of detection performance (sensitivity, specificity and delay) considering various filter parameters, such as the filter order and detection threshold. Additionally, we consider the system's suitability for implantable applications by evaluating its computational overheads (execution time, energy consumption) when executed on the SiMS low-power processor.

1.3. THESIS OUTLINE

The remainder of this thesis is structured as follows: Chapter 2 contains the necessary background details required to understand this work. This includes the definition of absence seizures. Additionally, it presents the development platform used to prototype the detection mechanism. Then, it introduces the SiMS architecture used to evaluate the algorithm's applicability for ultra-low power architectures. Finally, it introduces related work in the field of seizure detection. Chapter 3 presents the seizure detection and suppression system developed in this thesis, as well as the specification and implementation of the realized prototype. Chapter 4 contains the evaluation of the detection and suppression system. considering both its detection performance and computational overhead to assess its suitability for ultra-low power (implantable) application. Finally, Chapter 5 summarizes the conclusions of this work and lists out scientific contributions. In addition, it proposes several direction for future work.

2

BACKGROUND

This chapter will give a brief overview of the concepts and background knowledge necessary to understand the work in this thesis. First, we describe the characteristics of absence seizures, after which we discuss the specifications of the development platform used for our prototype. Continuing, we introduce the SiMS framework which is used to evaluate the proposed solution for its low-power applicability. Finally, we discuss the related work on seizure detection techniques.

2.1. ABSENCE SEIZURES

Absence seizures are a form of epileptic seizures characterized by temporary loss and return of consciousness and behavioral arrest. The Electrocorticogram (ECoG) of a typical absence is characterized by as spike-and-wave discharges (SWDs) that repeat themselves at a certain frequency. The repeat frequency of SWDs in mice is 5 Hz $< f_e <$ 8 Hz, where in humans this is 3 Hz $< f_e <$ 4.5 Hz. An ictal (seizure) period is defined as starting at the first ECoG spike of a SWD and ending at the last ECoG spike lasting for at least 1 second [15]. An interictal period(non-seizure period) is defined as the time in between SWDs starting 2 seconds after 1 SWD and ending 2 seconds before the next SWD [15]. In Fig. 2.1 we see an example of a ECoG signal containing ictal activity/seizure period and in the zoomed part we can see the form of spike and wave discharges.

In the rest of this thesis, we will refer to "absence seizures" by the term "seizures".

2.1.1. STIMULATION TECHNIQUES

In this work we consider a closed-loop system that detects absence seizure and reacts accordingly to prevent it from further developing. This early termination of a seizure can be achieved by applying appropriate stimulation techniques, such as electrical stimulation [16] and optogenetic stimulation [11, 17]. Electrical stimulation works by delivering electrical pulses to either the epileptogenic zone or to the vagus nerve[18]. Even though this method is effective in stopping seizures and FDA approved for use in humans, it is not optimal. The drawbacks of electrical stimulation include that it affects surrounding cells other than the target population and can cause neural damage [19–21]. On the other side, optogenetic stimulation functions by applying light to genetically modified neurons that are light-sensitive. While this method provides more localized and cell-specific stimulation without interfering with other cells, its limiting factor is the introduction of virus transgenes that sensitize neurons to light, which is not yet approved for use in humans [22]. In this work, we have developed a system that used optogenetic stimulation on mice models[1].

[1] Described in Section 4.1.1

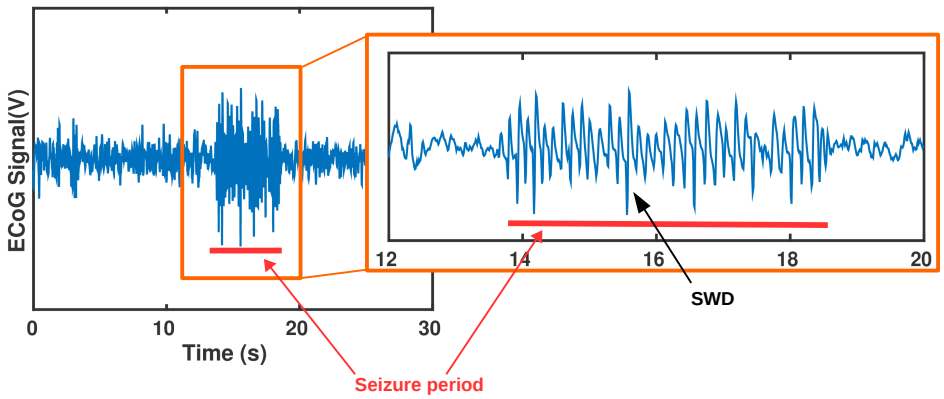


Figure 2.1: ElectroCorticoGram where part of the seizure period is zoomed in to illustrate the form Spike-Wave-Discharges(SWD)

2.2. DEVELOPMENT PLATFORM - BEAGLEBONE WHITE

For the requirements of the closed-loop system proposed in this work, we derived a set of constraints related to execution time, number of input channels, etc. (see Appendix C). Based on these constraints, we selected to prototype the system on the BeagleBone White [23], the details of which are presented in Table 2.1.

The BeagleBone White is a development board created by Texas Instruments[24] and is equipped with an ARM Cortex-A8 processor [25]. The board comes with a wide range of interfaces and is highly expandable through market available add-on boards (Capes). In order to utilize a certain set of interfaces (pins), the processor needs to be configured (pin multiplexing) at a software level.

For the prototype of this work, the pins are configured to provide access to certain general purpose input output(GPIO) pins and the ADC pins. The GPIO pins are used to trigger the stimulation mechanism and the ADC pins supply the input signals to the system (more details are presented in Section 3.2). From the rest of the board's specifications, the UART is used for monitoring and configuring the parameters of the system and the microSD card to store the software executable, which is loaded in memory during boot-up of the board. In addition, two (2) of the board's timers are employed, one to monitor the performance of system and one to trigger the ADC unit at the sampling rate set by the closed-loop system (see Section 3.2). At a software level, in order to use the BeagleBone bare-metal to meet the real-time constraints, the Starterware [26] package is utilized. This package includes necessary boot-up code and most of the drivers and libraries for board's peripherals.

2.3. SINS PROJECT

The work presented in this thesis is part of the SINs (Smart Implantable Neurostimulators) project [27]. The project is a collaboration between the Dutch Neuroscience Department of the Erasmus Medical Center, the Delft University of Technology and the Belgian BRAI2N group (Brain Research center Antwerp for Innovative & Interdisciplinary Neuromodulation). The project's goals are the improvement of the current generation of neurostimulators, by - among other means - the use of significantly smaller microchips, unbreakable wiring, miniature batteries that are fed by the human body, wireless monitoring and steering by smartphones. These improvements should result in the creation of new appliances and solutions that are much more patient-friendly.

In particular, our work address the design of ultra-low power consuming application for seizure detection and suppression, as explored by SINs-subproject SiMS (Smart Implantable Medical Systems

Board requirements	Details
Performance	720MHz DC-powered, 500MHz USB-powered
ADC inputs	4x
ADC resolution	12bit
ADC sample rate	200K samples per second
ADC voltage range	0-1.8V
DAC outputs	*-Not available-*
Data storage	microSD card 3.3 V
Wireless	*-Not available-*
UART	4x (1 via USB)
Bare-metal	Starterware drivers
Expansion	two 46-pin connectors
Support	Large Open community
Small size	86.40 mm × 53.3 mm
Power modes	Dynamic Voltage Frequency Scaling
Other H/W specifications	
SOC	TI SITARA AM3358, ARM Cortex-A8
SPI	2x , 48MHz max bitrate
Timers	4x , 32bit with min 40ns resolution
GPIO	65 @ 3.3V
Memory	256MB DDR2-400MHZ
H/W Revision	A6b

Table 2.1: BeagleBone white specifications

[27]).

2.3.1. IMD Processor Architecture (SiMS)

Apart from proposing a new algorithm to detect seizures, this work considers the suitability of the detection algorithm for low-power implantable applications with patient-specific configuration. In order to accommodate the different patient characteristics, such as different detection levels, different stimulation duration needs and different filter parameters, a flexible and customizable implementation needs to be considered. Therefore, we are investigating a software implementations running on a low-power processor rather than hardware specific implementations (ASIC). For this purpose, we select the SiMS processor architecture [1] which targets implantable medical devices(IMD) and incorporates all the necessary features for such a device, including dependability, miniature size and ultra-low power (ULP) consumption. The SiMS processor specifications consist of:

- A 5-stage RISC architecture
- A stall-detection unit
- 16-bit instruction size

- 16x 32-bit registers
- 16-kB (16-bit wide) Instruction Memory (IMEM)
- 16-kB (32-bit wide) Data Memory (DMEM)
- 1078.93 μW average power consumption
- 20Mhz clock frequency

See Appendix B for instruction set.

By executing the seizure detection algorithm on SiMS we can assess various performance aspects, include software execution statistics (instruction mix, memory usage, execution cycles) and power consumption, and how these scale with respect to different algorithm parameters. Based on these results, we estimate on whether it is viable to realize such an algorithm on a IMD.

2.4. RELATED WORK

Numerous studies have proposed various detection algorithms for seizures. Some of these techniques have been incorporated in hardware implementations and closed-loop systems. Below we present these studies and present the detection delay where available.

One of the first methods is based on the *energy* of the EcoG signal as described in Zaveri er al. [28]. In addition to the standard method of energy calculation, this study also evaluates a modified version of energy calculation called Taegers's algorithm or non-linear energy [29], where the input values are not weighted equally as in the conventional method. Both methods, start by calculating a baseline average energy based on an interictal interval and then calculate the average energy every 1 second. A seizure is detected when the average energy becomes greater than the baseline value. Using the Energy method they obtained a sensitivity of 58% and a specificity[2] of 78%, while using Teager's algorithm they obtained a sensitivity of 53% and a specificity of 100%. The advantage of the *energy* based method is its low complexity, however the performance results are relatively low.

A *template matching* technique is employed in Qu et al. [30], where the method tries to detect a new seizure that is similar to a given input template. The template is defined by a seizure recording and a set of background EEG which are given as initial inputs to train the system. Their method yields a sensitivity of 100%, an average of 0.2 false positive every hour and a detection delay of 9.6s. While this method has a high detection rate, its detection delay is very high and it will only detect seizures similar to the template.

The *line-length* method, which is described in [31], calculates the average running sum of "distances" between successive values within a fixed window and a seizure is detected when the sum crosses a pre-defined threshold. The line-length study reports a detection delay of 4.1s, an average false positive rate per hour of 0.051 and false negatives of zero (0). Similar techniques to the *line-length* are the *coastline* method found in [32] and [33], and the *steepness* method from [34], where both techniques compute the difference of successive values in a fixed window. Compared to the *line-method*, the *coastline* differs in that it does not average the distance results, while the *steepness* method uses the maximum "distance" value over a period of time and then calculate the average of consecutive sums of "distances". The benefit of these methods is the low complexity of implementing them and the downside is that they are susceptible to large amplitude noise which can trigger false detections.

The research in White et al. [33] presents an *autocorrelation* method in combination with a *spike frequency* method. In this method the difference of the maximum and minimum values in fixed size groups is computed, then the sum of those for consecutive groups form the autocorrelation metric and together with the number of spikes found in the current interval a decision is made. This method

[2] specificity and sensitivity are described in Section 4.1.1

results in a sensitivity of 100% and a specificity of 99.98%. The high detection performance is the advantage of the method, however the study does not consider its detection delay.

In van Hese et al. [35], a method based on the *spectrogram* is used to detect absence seizures. The method consists of the following steps: 1) the spectrogram is computed using a short-time Fourier transform; 2) an estimation of the background spectrum is made and artifacts are removed with a median filter, 3) harmonic analysis with continuity analysis is performed to estimate the fundamental frequency; and 4) seizure classification is based on the percentage of power in the harmonics to the total power of the spectrum. As a result, the method achieves a maximum sensitivity and specificity of 96% and 97% respectively. While yielding high detection performance results, the study does not report on the detection delay and it fails to yield the same performance when EEG signals are more irregular.

Furthermore, *wavelet based* methods have also been proposed as a tool to detect seizures. Wavelets based analysis is localized both in time and frequency and compared to classical analysis (e.g. Fourier analysis) it provides better information on signals that change substantially [36]. Additionally, wavelets outperform other types of time-frequency analysis, as they provide better time-frequency decomposition by selecting an appropriate wavelet function. A real time detection algorithm using the level-3 Daubechies D4 tap (or db2) wavelet is presented in [36]. The algorithm consists of a FIR filter approximation of the wavelet of order 22 followed by a median filter on the square values of the FIR output over a window of 2s. A seizure is detected when the calculated value remains above the a specified threshold for a certain amount of time. The algorithm achieves a detection delay of 1.5, with 100% Sensitivity and 0.04 False-Positive-per-seizure rate. The method's main advantages are its high detection performance and straightforward implementation, however the need to store a 2s window of previous FIR output might increase its computational complexity. In another study, Berdakh et al. [37], wavelets are used to filter the input data which are then fed to a *Artificial Neural Network (ANN)* classification algorithm. This study investigates different wavelets, Daubechies (db2, db5) and Biorthogonal (bior1.3, bior1.5), with the goal to find the optimal wavelets that produces the highest detection efficacy of the ANN. The study concludes in the Biorthogonal (bior1.3, bior1.5) wavelets as the best option for which the seizure detection system achieves a detection accuracy of 95.8%. Its main advantage is the high detection performance, however the use of ANN increase its implementation complexity. A similar study, Buteneers et al. [13], also uses *Artificial Neural Networks based on Reservoir Computing* as classification method where its EEG input is pre-filtered using wavelets ([36]). This study focuses on real-time absence seizure detection and achieves a detection delay of 0.97s with a Sensitivity of 96.2% and a Specificity of 98.2%. Moreover, the study states that detection delay can be improved by reducing detection accuracy. Despite the low detection delay and high detection rates, the main drawback of this implementation is that it requires a large set of samples to train the ANN so that it can correctly detect seizures.

Any of the previous detection systems that can provide real-time seizure detection could be incorporated in a seizure prevention close-loop systems. A closed-loop system actively monitors the EEG/ECoG signal and upon seizure detection triggers a stimulation mechanism (electrical or optogenetic) with the purpose to suppress the seizure's activity. The studies that have incorporated some of these methods in a closed-loop system are presented below.

The system presented in Fanselow et al.[12] uses a simple thresholding method to detect the seizures and uses electrical stimulation to suppress seizures. The detection is relatively fast as it detects upon the first EEG spike that crosses the threshold, which in most cases also defines the onset of the seizure. However, this method is susceptible to artifacts in the EEG as it will trigger to anything that will cross the threshold. This artifact problem is improved in Berenyi et al.[10], where a band-pass filter is applied to the input signal cutting off low frequencies being caused by movement. Besides the fact that the band-pass filter increases detection delay by approximately 200ms, the artifact problem is not fully resolved. This is because noise with frequency within the range of the band-pass filter can still gen-

erate false triggering. The closed-loop system in Paz et al. [11] employs the line length method from [31] in combination with optogenetic stimulation. The algorithm is implemented on a programmable real-time digital signal processor [38] which upon detection triggers the optogenetic stimulation.

Finally, there are various studies that propose low-power hardware implementations. The are two studies that present realizations and report their detection delay. First, in Raghunathan et al.[39], a digital circuit implementation is presented which uses a simple threshold algorithm combined with a crossing counter. This implementation is capable of detecting a seizure with an average delay 8.5s, which is relatively high compared to previous methods, a sensitivity of 95.3% and a specificity of 88.9%. Additionally, a power consumption of 350nW is reported. The second low-power system implementation, presented in Salam et al.[14], achieves a detection delay of 13.5s with a sensitivity of 100% (specificity results are not reported). The detection method used in this system identifies seizures that are characterized by their progressive increase of low-voltage fast activity. The work of this study also presents the fabrication of the system that yields an average power consumption of 44.58nW. The other two hardware implementations only focus on the power consumption and the detection rate of their solution. First in Markandeya et al. 2010 [40], a low power system based on the Daubencies D4 wavelet and a threshold value is presented. In this study different hardware optimizations are explored which minimize power consumption. The presented hardware implementation yields an average sensitivity of 98% and an average specificity of 95% for different subjects, while different power consumptions profiles are reported depending on the selected hardware optimization. Second, in Markandeya et al. 2012 [41], the study presents a seizure detection processor that incorporates multiple detection algorithms and allows the selection of a patient-specific algorithm. This hardware implementation has a power consumption of 360nW and yields average performance results, in terms of ADR, between 65% and 98% depending on the subject and the selected algorithm. The only commercially available and FDA approved closed-loop seizure detection and suppression implantable device to date is the Neuropace RNS system. The system provides three (3) methods to detect a seizure: 1) a *bandpass* method; 2) a *line-length* method; and 3) an *area* method which a similar method to the *energy* method. Upon seizure detection, the neurostimulator delivers current-controlled, charge-balanced biphasic pulses as programmed by the physician. The system has a life-expectancy of 2.5 to 3 years and is the size of a credit card. The system reports 600 to 2000 per day with a cumulative total of <5mins of stimulation. Clinical tests have shown a reduction of 50% in seizure frequency in patients in the 2nd year of use [18, 42, 43].

Concluding, most of the aforementioned detection techniques do not discuss extensively performance results and only few report the detection delay of the algorithm which is one of the objectives of our implementation. In this work, the solution presented is based on a wavelet filter (complex Morlet) and aims to achieve low detection delays. It differs from the aforementioned wavelet methods as it uses a low complexity threshold mechanism and it only requires a small sample set (4 seizures) to train the levels of the thresholds in the case of the adaptive mechanism. Furthermore, in comparison with the hardware implementations that mainly focus on low-power consumption and detection performance, our solution also focuses on flexibility and creating patient-specific solutions that maximize detection.

3

IMPLEMENTATION

In this chapter, we start by describing the conceptual model of a system that is capable of real-time detection and closed-loop suppression of absence seizures. For this system, we continue in presenting a prototype implementation, which is used to detect and suppress seizures in *in-vivo* measurement setup.

3.1. CLOSED-LOOP SYSTEM MODEL

An overview of the proposed system and its deployment is depicted in Fig. 3.1. The system consists of four modules: 1) the inputs of the system, which are Electrocorticogram (ECoG) signals recorded from live subjects (e.g. mice) using a *ECoG recorder*; 2) an analog module that filters these signal (*Input filter*); 3) a digital module that samples and converts the signal in the digital domain (*Digitization*), which then are passed through a Complex Morlet wavelet filter (*Feature extraction*) and finally, based on the result of the wavelet-filter, a decision is made on the existence of a seizure event (*Decision making*); 4) a *stimulator* (e.g. Optogenetic[1]) which is triggered by the decision making part upon a seizure detection and in turn stop the seizure.

In this thesis work, we have used off-the-shelf modules for the ECoG recorder and the optogenetic stimulator (modules *1* and *4*) and a custom PCB was developed for the analog module (*2*) to realize a prototype implementation, the details of which are described in section 3.2. The focus of our work was on the digital module (*3*), the implementation of which is described in the following section.

3.1.1. DIGITAL MODULE

The digital module is responsible for seizure detection. It achieves this by using a complex Morlet wavelet filter and a thresholding mechanism for decision making. We chose to implement the wavelet filter in the digital domain as this provides a higher level of flexibility and configurability. This allows us to adjust the parameters and configure the system according to subject specific characteristics.

As shown in Fig. 3.1, in order to perform theses functions the analog input signal is sampled at a frequency f_{sample} = 100 Hz and converted to the digital domain. The reason for choosing a sampling frequency of 100 Hz is based on the frequency range we are interested in, which is $f \in [0,50]Hz$ where most brain signal activity is observed [44]. Thus, from the Nyquist theorem a sufficient sampling frequency is 2 (or higher) times the highest frequency in our desired range.

In the following sections we describe the wavelet filter and the detection mechanism in more detail.

[1] Light-triggered stimulation of genetically sensitized neurons

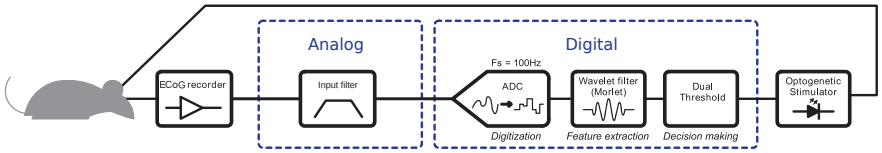


Figure 3.1: closed-loop epilepsy detection system overview

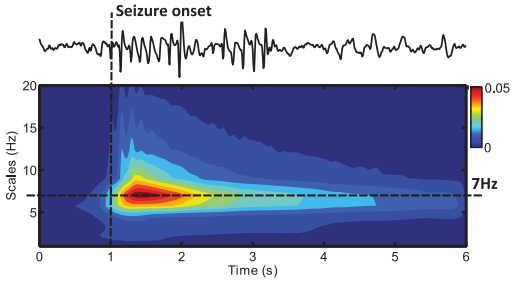


Figure 3.2: Average wavelet filter's response for all seizure events, where each seizure-start is aligned at 1 sec. For illustrative purposes a single seizure is presented.

WAVELET FILTER

The wavelet filter used in this work for distinguishing seizure behavior is a complex Morlet wavelet. The reason for selecting wavelet filters is because they are localized in both time and frequency, which provides frequency information at a certain time. The particular type is chosen because of its morphology resemblance with the ECoG spikes during a seizure [45].
The complex Morlet is defined as follows:

$$\Psi_\sigma(t) = c_\sigma \pi^{-\frac{1}{4}} e^{-\frac{1}{2}t^2} (e^{i\sigma t} - k_\sigma) \tag{3.1}$$

where,
$k_\sigma = e^{-\frac{1}{2}^2}$, $c_\sigma = (1 + e^{-\sigma^2} - 2e^{-\frac{3}{4}\sigma^2})^{-\frac{1}{2}}$
and σ =wavelet scale

In our work we select a single wavelet scale so that we can develop a suitable and realizable filter implementation for distinguishing seizures. In order to determine an optimum wavelet scale value, we obtain the continuous-time Morlet wavelet filter response over a range of frequency scales for each seizure event in a pre-recorded ECoG database (the details of which are described in section 4.1.1). The frequency scales are approximately related to the wavelet scale σ value by [46]:

$$f_\sigma = \frac{f_c}{\sigma f_s} \tag{3.2}$$

where,
f_c = wavelet center frequency , f_s = sampling frequency
and σ =wavelet scale

For each of these frequency scales we average its response to each seizure leading to the scalogram depicted in Fig. 3.2. The scalogram depicts the power of the filter response for each frequency scale with respect to time, where the onset of each seizure is aligned at 1 sec. This alignment allows us to

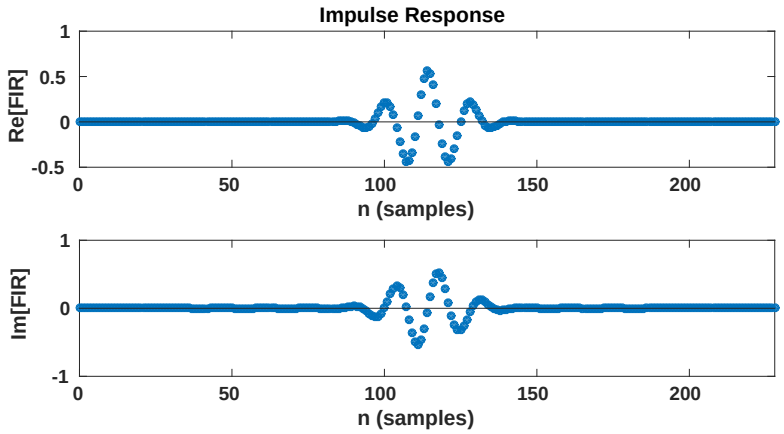


Figure 3.3: Discretely sampled impulse response of Complex Morlet at f_{scale} = 7 Hz for the max effective support of MATLAB®.

observe the average filter response before and after the onset of a seizure interval, where it is evident that the average filter response during seizures (after 1 sec) is higher than during non-seizure periods (before 1 second). Therefore, we estimate that the power of the filter output provides a good indicator for recognizing seizures. We discover that the maximum response is produced at f_σ = 7 Hz, which is therefore chosen to design a suitable filter implementation. The complex Morlet wavelet is defined on an infinite and continuous time span, making it unsuitable for practical applications. Thus, in order to create a practical (digital) implementation, we approximate the wavelet using an FIR filter, where we truncate and discretely sample the impulse response of the wavelet at the selected frequency scale [47]. The FIR is defined as follows:

$$y(n) = \sum_{i=0}^{N-1} c_i x(n-i) \tag{3.3}$$

where,

$x(n)$,	input signal
$y(n)$,	output signal
N,	order of the filter (i.e., number of coefficients)
c_i,	a complex coefficient value of the i-th of the filter-coefficient

The coefficient values of the FIR filter are obtained by sampling the wavelet filter (f_σ = 7 Hz) at the same frequency as our input sampling rate, i.e. f_s = 100 Hz. Using the effective support[2] in MATLAB® for the initial filter and sampling it at f_s = 100 Hz, this results in an accurate approximation of the continuous complex Morlet wavelet using 229 coefficients for each of the complex parts (both real and imaginary). In Fig. 3.3 we plot the impulse response of the FIR filter. We previously saw power can be used for distinguishing seizures. To calculate power from these impulse responses, we do:

$$Y(n) = P_{FIR}(n) = (\sum_{i=0}^{N-1} Re[c_i]x(n-i))^2 + (\sum_{i=0}^{N-1} Im[c_i]x(n-i))^2 \tag{3.4}$$

DETECTION MECHANISM

In the previous section, we defined a FIR filter which produces high output values during a seizure. To capitalize on this increased output, we may detect seizures using a threshold mechanism where a

[2]Region where the filter is non-zero

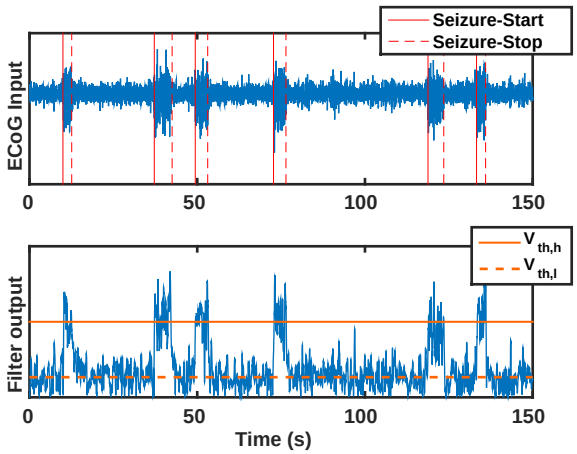


Figure 3.4: Example of an ECoG input (top) and its filter output (bottom) of an ECoG recording, which contains seizures. Vertical red lines in the **ECoG Input** mark the seizure periods. Horizontal orange lines in the **Filter Output** illustrate the dual thresholding mechanism

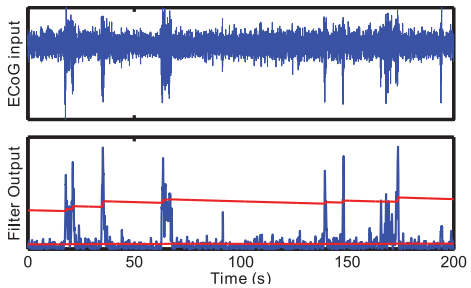


Figure 3.5: Generic of adaptive threshold mechanism presenting how the Upper threshold adapts in relation to the Filter Output and how the Lower threshold maintains a value close to the average of non-seizure periods.

seizure is detected when this threshold is crossed. To exemplify, we depict an both ECoG signal and its filter output in Fig. 3.4. Note the increased output during seizure intervals. Consequently, these seizure intervals can be detected using a threshold value (continuous orange line).

Based on the above observation, we define an upper threshold $V_{th,h}$ which detects this increased filter output $Y(n)$ and signifies the occurrence of a seizure. It may be observed from Fig. 3.4 that the filter output is reduced below $V_{th,h}$ during a seizure interval. Thus, to prevent multiple detections from occurring within the same seizure event, we employ a low threshold $V_{th,l}$ which approximates the end of the seizure interval when $Y(n)$ falls below that value. As such, we get a dual threshold mechanism.

In this work, we have considered the use of both a static and an adaptive threshold. The static threshold mechanism consists of fixed values for both $V_{th,h}$ and $V_{th,l}$ as depicted in Fig. 3.4. In contrast, the adaptive threshold mechanism tries to increase detection accuracy by defining the values of $V_{th,l}$ and $V_{th,h}$ as a function of the filter's output, an example of which is shown in Fig. 3.5. In this figure, we observe that the upper threshold ($V_{th,h}$) slopes up for high output value (seizure periods) and gradually decreases for low output values (non-seizure periods). By this means, it attempts to sustain a sufficiently high value that avoid mis-detections, while it tries to accommodate for seizures with lower filter output values. The adaptive upper threshold ($V_{th,h}$) aims to follow the average value

of the filter output during seizures, and it achieves this by low-pass filtering the output $Y(n)$ of the FIR filter. As such, the upper threshold is defined by:

$$V_{th,h}(n) = V_{th,h}(n-1) + \frac{Y(n) - V_{th,h}(n-1)}{\tau f_s} \tag{3.5}$$

where,

$$Y(n) > V_{th,h}(n-1) \rightarrow \tau = \tau_{up}$$

$$Y(n) < V_{th,h}(n-1) \rightarrow \tau = \tau_{down}$$

This low-pass filtering is only used to define $V_{th,h}$ and does not interfere with the input signal and the FIR filter output. The τ values define the rate with which the threshold $V_{th,h}$ is adapted. Specifically, small τ constant values will result in the threshold value to be more sensitive to changes in the filter output, while high values will cause the threshold value to change gradually. For instance, when $Y(n) > V_{th,h}$, a small τ_{up} will rapidly increase the upper threshold. On the other hand, when $Y(n) < V_{th,h}$, a large τ_{down} will ensure that the upper threshold will decrease at a slower rate (This trade-off between tau-values for τ_{up} and τ_{down} will be fully explored in section 4.2.2).

As for the adaptive lower threshold ($V_{th,l}$), we observe in Fig. 3.5 that it maintains a value close to the average value during non-seizure periods. This is achieved by calculating the (cumulative) running average of the filter output $Y(n)$ for a window of N samples. It is defined as follows:

$$V_{th,l}(n) = \frac{(N-1)V_{th,l}(n-1) + Y(n)}{N} \tag{3.6}$$

The size of the window has to be relatively large so that the duration of ictal periods does not significantly affect the average value of inter-ictal periods. Based on our experiments, a value of 10,000 seemed to be an appropriate choice for the sample window N. Effectively, as regular inter-ictal activity ensues after a seizure, $V_{th,l}$ may be used to approximate the end of a seizure.

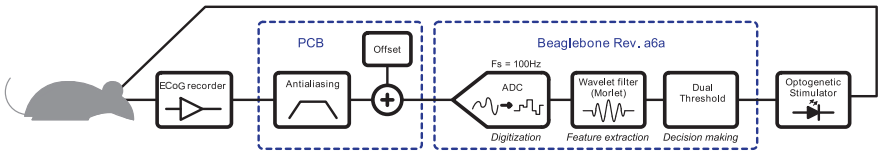
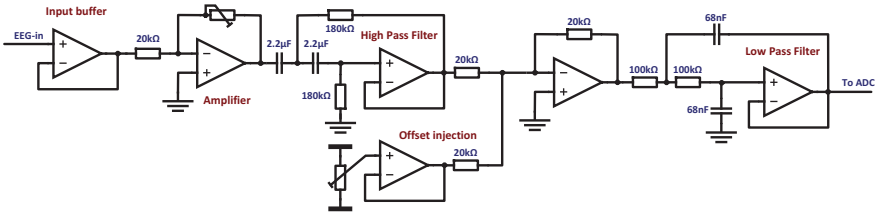


Figure 3.6: Prototype overview



(b)

Figure 3.7: Schematic of the analog input filter implementation

3.2. PROTOTYPE SPECIFICATIONS

In this section, we present the implementation details of a prototype setup that realizes the real-time seizure detection and suppression system, part of which has been published in [45]. Additionally, we briefly present the in-vino measurement results.

A detailed overview of the prototype (and its deployment in in-vivo experiments) is presented in Fig. 3.6. As the prototype fully implements the seizure detection and suppression system from the previous section, it consists of an ECoG recorder module, an analog module (implemented on a custom PCB), a digital module (executed on the BeagleBone development platform [23]), and an optogenetic stimulator. In this thesis, we have used off-the-self components for the ECoG recorder responsible for providing input values to the system and the optogenetic stimulator which suppresses seizure behavior. The ECoG recorder is a Cyberamp amplifier [48] and the optogenetic stimulator is an optical fiber with a LED source at one end [49].

Our work for the prototype focused on the realization of the digital module, which is extensively discussed in 3.2.2. In addition, as part of the prototype an analog module was also developed, which is briefly discussed in 3.2.1.

3.2.1. ANALOG MODULE

The ECoG signal that comes from the ECoG recorded needs to adjusted before it is fed to the digital module. Therefore, the analog module performs two main functions(signal conditioning, voltage adjustment) and is implemented on a custom PCB, the design of which has been originally presented in [45]. In Fig. 3.7 we depict the circuit implementation of the analog module.

The preconditioning function involves an input filter that removes unwanted artifacts from the ECoG input signal (e.g. equipment imperfections, external movement etc.) and performs signal anti-aliasing which restricts the bandwidth of the input signal. In detail, unwanted artifacts are removed by a second-order 0.4Hz high-pass filter cutting off all low-frequencies that are usually caused by movement. Signal anti-aliasing is performed by means of a second-order 23.4Hz low-pass filter cutting-off all frequencies above 23.4Hz and allowing only the desired frequency range to pass, as it was shown in section 3.1.1 most seizure activity was found in the frequency range of [0-20Hz].

The other function is responsible to adjust the voltage range of the ECoG signal in order to meet the

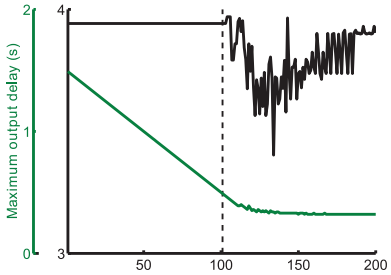


Figure 3.8: Filter precision and response delay trade-off for different truncations

requirements of the A/D converter (ADC) on the Beagleblone. So, the ECoG signal's voltage amplitude is regulated and a positive offset is injected to match the ADC's voltage accepted range of $V_{in} \in [0 - 1.8V]$.

3.2.2. DIGITAL MODULE

The digital module of the closed-loop system is implemented on the BeagleBone Rev.A6a development platform and consists of three components (as discussed in 3.1.1): 1) Digitization, 2) Feature Extraction and, 3) Decision making.

In the digitization component, the input signal that comes from the analog circuit is converted using the on-board 12bit ADC of the BeagleBone at a frequency f_s = 100 Hz. Then, the feature extraction component computes the FIR approximation of the complex Morlet wavelet. The FIR filter is responsible to produce increased output values during seizure intervals. Finally, the decision making component uses a the static thresholding mechanism to detect the increased output values (seizure) and trigger the optogenetic stimulator.

In the following sections we present the implementation details of our FIR filter and the software setup of the components on the Beaglebone White.

FIR FILTER

In this section we describe the design procedure to map the FIR filter on the Beaglebone White. First, we present how we truncate the FIR impulse response to minimize delay. Subsequently, we investigate the appropriate amount of scaling for the coefficients and the signal inputs.

The FIR filter implemented on the BeagleBone is a truncated version of the initial FIR filter presented in section 3.1.1. By truncating the FIR filter, we strive to decrease detection delay, which is caused by the large amount of near-zero values at its start (see Fig. 3.3). This section presents a preliminary evaluation on the best truncation point for the prototype implementation. In Chapter 4, we do a thorough investigation on different parameters and how these affect performance.

For the prototype implementation, in order to find an optimum truncation point we start by filtering all the seizure periods from the pre-recorded ECoG database[3] using different truncations. For a wavelet filter it is required to keep the DC value of the FIR impulse response to zero (see [47]). Therefore, for each truncation we add an offset on both the real and imaginary part to ensure a zero DC offset. The filter precision of each truncation is evaluated by the ratio of the averaged response during seizure to the average response of interictal activity. This ratio is represented by the black line in Fig. 3.8. Then, an approximation of the filter's response delay is defined as the time from the onset of a seizure event to the maximum filter's response during the seizure duration. The average response delay of all seizures in relation to the truncation value is represented by the green line in Fig. 3.8.

From Fig. 3.8 we can observe that for truncations > 100 the precision starts to deteriorate. This demonstrates that the average filters response for interictal activity becomes comparable to the value

[3]The database is described in section 4.1.1

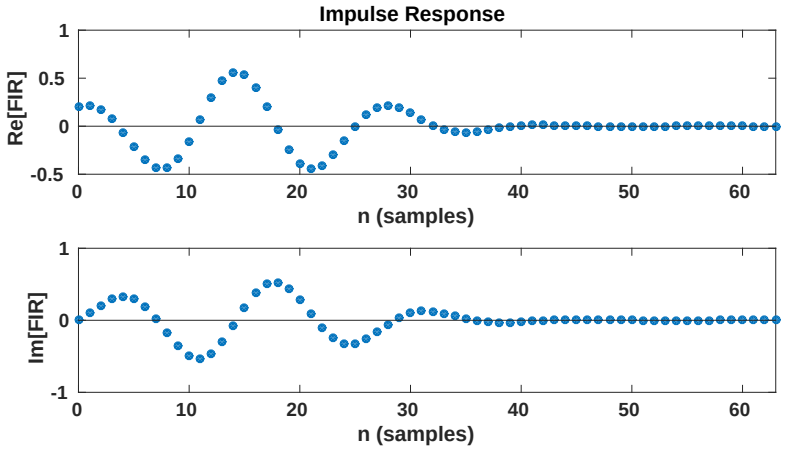


Figure 3.9: Truncated impulse response for the FIR filter implemented in the prototype (64th order filter)

of the average response of the seizures and in turn losing the advantage of using the filter. Based on the results presented in the figure we choose to truncate 101 samples from the filter and use a total of 64 coefficients (64-order FIR filter). The truncated version of the FIR impulse response is shown in Fig. 3.9. From the results we find that precision is close to the optimal value and the response delay (max response value) is 490 ms.

The ideal parameters of the FIR filter, i.e. the input and coefficients, have infinite precision. To make this filter suitable for the digital domain which operates using finite precision, we map the parameters of the FIR implementation to fixed-point arithmetic values. Compared to floating-point arithmetic, this ensures a fast execution on modern architectures and minimizes computational complexity.

The operations responsible for the filter response output are targeted to fit in 32-bit integers. One of the reasons this is done, is because the selected development platform has a 32-bit architecture (ARM cortex-A8), which means that basic 32-bit arithmetic integer operations are executed in one clock cycle. In addition, many other ultra low power architectures (such as the SiMS processor [1] or the ARM Cortex-M [50]) also provide 32-bit hardware and hence favoring a choice of 32-bit values.

To map floating precision values to fixed precision, we scale input values and coefficients using k_x and k_c respectively. To derive these values, we start from the initial FIR description (Eq. 3.4):

$$Y[n] = (\sum_{i=0}^{N-1} Re[c_i]x[n-i])^2 + (\sum_{i=0}^{N-1} Im[c_i]x[n-i])^2 \tag{3.7}$$

Then we introduce the scaling factors:

$$Y[n] = (\frac{k_c}{k_x})^2 \left\{ (\sum_{i=0}^{N-1} k_c Re[c_i] \frac{1}{k_x} x[n-i])^2 + (\sum_{i=0}^{N-1} k_c Im[c_i] \frac{1}{k_x} x[n-i])^2 \right\} \tag{3.8}$$

k_c = Coefficient scaling factor
k_x = Input scaling factor

In order to avoid costly multiplication and division operations, only powers of 2 were considered for the scaling factors. Hence, parameters scaling can be implemented with arithmetic shift operations.

Subsequently, we truncate the parameters to integer values (floor function on each value), which introduces an error to the output:

$$Y[n] = K(\sum_{i=0}^{N-1} Re[c_i^*]x^*[n-i])^2 + (\sum_{i=0}^{N-1} Im[c_i^*]x^*[n-i]))^2 + \epsilon \tag{3.9}$$

$$c^* = \lfloor k_c c_i \rfloor, \text{ scaled coefficient value}$$
$$x^* = \lfloor \frac{1}{k_x} x[n-i] \rfloor, \text{ scaled input value}$$
$$\epsilon = \text{Truncation error, } K(\frac{k_x}{k_c})^2$$

The FIR value $Y'[n]$ calculated with the scaled values is the following

$$Y'[n] = (\sum_{i=0}^{N-1} Re[c_i^*]x^*[n-i])^2 + (\sum_{i=0}^{N-1} Im[c_i^*]x^*[n-i]))^2 \tag{3.10}$$

which is related to the initial description as follows,

$$Y[n] = KY'[n] + \epsilon$$

The criterion to determine the k_c, k_x values is that the $Y'[n]$ value fits in 31 bits for any input with the minimum rounding error ϵ.

$$\lceil log_2(Y'[n]) \rceil \leq 31$$

31 bits are considered as we have incorporated 32-bit signed integers in our software implementation, therefore the max positive integer has a length of 31 bits. Using signed integers is done as a safety measure to detect potential overflows in the calculation of $Y'[n]$ (a negative $Y'[n]$ indicates the resulting sum did not fit in 31bits)
Using samples from the pre-recorded ECoG database (described in section 4.1.1) and testing different values for the scaling factor, we found the values $k_c = 64$ and $k_x = 16$ to best match our criteria. This means that coefficients become integers in the range [-64,64], thus needing 7-bits to represent them. The input values, from an initial range of [0-4096], can scaled down to [0-255] and fit in a 8-bit value.

SOFTWARE SETUP

The software setup consists of an overview of the software implementation and the software tools that were used to realize the system on the BeagleBone White. The flow diagram of Fig. 3.10 depicts the main components of the software implementation, these include: 1) the startup code and variable initialization component (*Initialization*); 2) the ADC value storage component (*Store sample*); 3) the FIR filter calculation component (*FIR kernel*) computation; 4) the decision making component (*Detection*) mechanism;and 5) a component responsible for the device control via a host-PC (*PC-comm*)
.

Besides the *Initialization* component, which is executed one time, the software implementation is an infinite loop which every time for a new sample value to arrive. The main body of the infinite loop is composed by the *Store sample*, the *FIR kernel*, and the *Detection* components. These components are responsible for the implementation of the detection mechanism and their computation have a real-time constrain of 10 ms given the input data sampling frequency. The loop also contains a *PC comm* component, which is used to monitor and configure the device's parameters at run-time.

Initialization: This component is responsible to configure the BeagleBone hardware peripherals which are used by the software implementation and initialize software variables. These peripherals include

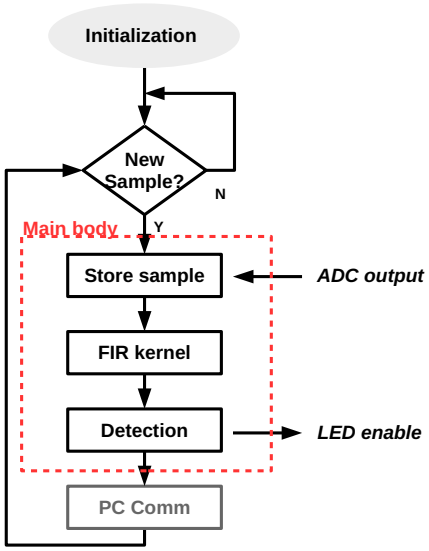


Figure 3.10: Seizure Detection SW flow chart

the ADC unit responsible for sampling the input signals and hardware timers that monitor the performance.

Store sample: A circular history buffer responsible for storing the current and maintaining previous samples. The point where the new sample is inserted is point zero of the FIR filter and going backwards in the buffer for the previous samples. The size of each buffer is 64 elements which correspond to number of coefficients and are required for the FIR calculation. Each element's size is defined as a 16-bit signed integer and stores the 12-bit values that arrive from the ADC peripheral which in turn are shifted right to 8-bit values, as defined in 3.2.2.

FIR kernel: This part computes a 64 order FIR filter output, as presented in 3.2.2.

Detection: The dual threshold mechanism described in section 3.1.1 is applied to the value calculated in the FIR kernel. Upon seizure detection, the optogenetic stimulator is enabled for a specified duration.

PC comm: This component is responsible for the communication of a host-PC with the BeagleBone. This communication interface is used to monitor and configure various parameters of the system. Configurating the system includes setting threshold levels, adjusting stimulation duration time, and enabling monitoring channels. Monitoring the system involves the BeagleBone sending telemetry data back to the host-PC to monitor, such as filter input and output information. This component is mainly used for debugging, setting parameters at run-time and monitoring the system and can be omitted if the system is proven to work. A screenshot of the host-PC graphical user interface is presented in Appendix A.

The software implementation is written in C software language and is compiled using the GNU bare-metal toolchain of version 4.7-2013-q3. The executable is running without the support of an Operating System (OS), i.e. bare-metal. The use of an OS was avoided because of the real-time requirements of the implementation. Additionally relying on OS operations, it would the porting ability

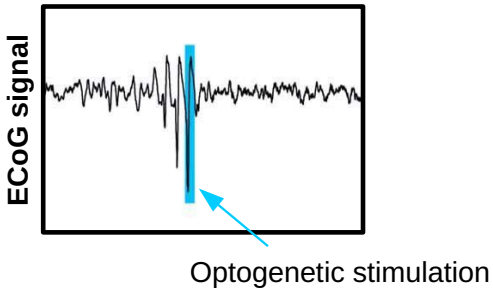


Figure 3.11: ECoG behavior during optogenetic stimulation during ictal-activity using the close-loop prototype and suppressing ictal-activity

to other platforms. Therefore, the only software components used is the StarterWare[4], which provided basic hardware drivers support and bootup code.

3.2.3. IN-VIVO RESULTS

The closed-loop prototype has been deployed on live subjects (mice), where its correct functionality in detecting and suppressing seizures was validated [15]. *In-vivo* measurement showed that When ictal-activity was detected, the optogenetic stimulator was triggered which in turn suppressed the seizure. On average, seizures were detected and stopped within 500 ms. An illustration of the optogenetic stimulation moment (blue line) and its effect on the ECoG (termination of ictal-activity) is depicted in Fig. 3.11.

[4] See details in section 2.2

4

EVALUATION

In this chapter, we evaluate the seizure-detection mechanism in terms of detection performance and software execution costs. We start with presenting the setup of our evaluation in section 4.1. Continuing in section 4.2, we analyze the results of the detection performance quantified in terms of accurate detections (sensitivity, specificity) and the detection delay. Then in section 4.3, we analyze the results for the computational costs of implementing the filter on a ultra-low power processor in order to assess the complexity of the mechanism. Finally in section 4.4, we present the trade-offs between seizure-detection performance and computational costs.

4.1. EVALUATION METHOD

Before we present the results of the evaluation, we need to explain the methods of evaluating the Detection mechanism. First, we introduce the parameters and the evaluation procedure of the detection performance. Next, we present the evaluation procedure for a software implementation of the detection filter on *SiMS* (ultra-low power processor). Finally, we show how we are combining these two performance aspects.

4.1.1. DETECTION PERFORMANCE

The Detection performance specifies how apt is the detection mechanism in detecting seizures and is determined by the FIR filter and decision mechanism. In this section we explain the setup of our evaluation. First, we define the metrics that quantify the performance, after which we list the parameters of the FIR filter and decision mechanism that affect it. Then, we describe the input values we varied to evaluate the detection performance. In the end, we present the procedure of evaluating the detection performance of the two decision mechanisms, one that is based on a static thresholding mechanism and one on a adaptive thresholding mechanism.

PERFORMANCE METRICS

The performance metrics of the detection mechanism include the Specificity, Sensitivity and Detection Delay as detailed next.

Sensitivity is a value that signifies the percentage of correctly classified seizures in a given input (ECoG database). It quantifies how sensitive is the detection-filter in detecting seizures. A high sensitivity indicates that the closed-loop system prevents a high amount of seizures from manifesting in the subject. Sensitivity is defined as:

$$Sensitivity = \frac{TP}{FN + TP}$$

where,

TP = number of True Positives. A true positive is defined as a detection during an ictal timestamp from the annotations in the ECoG database.

FN = number of False Negatives. A false negative is classified as the absence of detection during an ictal timestamp from the annotations in the ECoG database.

Specificity is a value that signifies the percentage of correctly classified inter-ictal intervals in a given input. It quantifies how accurate is the detection-filter in only detecting seizures and not other events. A high specificity indicates that the closed-loop system will not stimulate when not necessary, minimizing the adverse effects of stimulation. It is be calculated by the following definition:

$$Specificity = \frac{TN}{FP + TN}$$

where,

TN = number of True Negatives. A true negative is defined as the absence of a detection during an inter-ictal timestamp from the annotations in the ECoG database (correct behavior).

FP = number of False Positives. A false positive is defined as a detection during an inter-ictal timestamp from the annotations in the ECoG database (wrong detection).

ADR is a compound metric and describes the *A*verage *D*etection *R*ate performance of the detection filter. It is the mean value of Sensitivity and Specificity and provides a single value to classify the detection performance

$$ADR = \frac{Sensitivity + Specificity}{2}$$

Detection Delay is the average detection delay of all detected seizures from the ECoG database. The detection delay of a seizure is defined as the time between its onset, annotated time-stamp in the ECoG database, and the detection time of the detection filter. An example illustration of a seizure's Detection Delay is shown in Fig. 4.1. A small detection delay signifies that the closed-loop system is able to detect and suppress ictal-activity fast enough before it develops into a seizure that would affect the subject.

INPUT PARAMETERS/INPUT SPACE

The detection mechanism's input space comprises the coefficient window, which is divided in the coefficient window size and coefficient window offset, and the threshold values. These parameters are varied with the purpose of quantifying the various performance results as described in the previous section.

Coefficient Window In order to create a low complexity and efficacious FIR filter in terms of detection accuracy and delay, we consider different truncations of the initial impulse response (described in section 3.2.2). Therefore, we define the *Coefficient Window* parameter as the set of coefficients (both imaginary and real) that remain after truncating the impulse response on both sides. The Coefficient Window is characterized by its length in coefficient values and its location within the initial impulse response. An example of coefficient window, as projected on the real component of the impulse response, is depicted In Fig. 4.2. The first attribute, the *Coefficient Window Size*, describes the total number of coefficient found in the window and represents the order of the newly formed filter. We vary different size values to evaluate different performance results. While smaller sizes will reduce computational complexity, we expect that they will reduce detection performance. As for the second

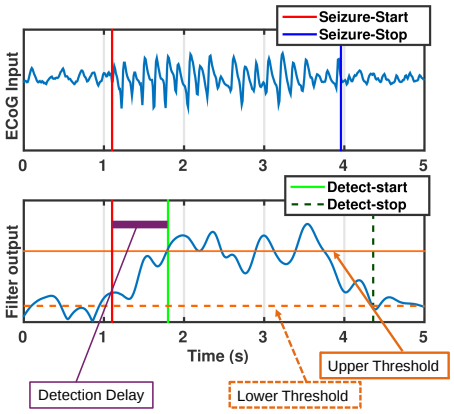


Figure 4.1: Detailed example of static threshold mechanism presenting the detection moment, the re-enabling of the mechanism and the Detection Delay of a thresholding mechanism.

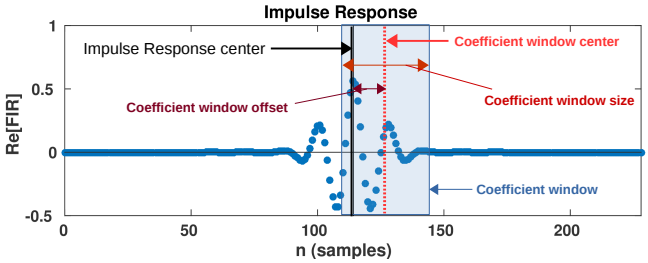


Figure 4.2: Coefficient Window Size and Offset illustration on real component of FIR impulse response

attribute , the *Coefficient Window Offset*, it quantifies the shifting amount of the coefficient window center with respect to the middle of the initial FIR impulse response. An zero offset means that the coefficient window is centered to the initial FIR impulse response. A positive offset, as seen in Fig. 4.2, means that the coefficient window is shifted to right in relation to the center, and a negative offset that it is shifted to left. We vary the window offset as to see whether different parts of the impulse response can produce faster responses and affect detection accuracy of seizure intervals. We chose to select window offset values that always include the center of the initial impulse response in the coefficient window. This center contains the most significant component of the impulse response from which we expect the best output values.

Threshold values The threshold values are involved in determining filter output values that belong to seizure intervals. We vary these values because they affect detection performance. It may be expected that higher values will prevent from false detections in inter-ictal intervals and lower values will allow more ictal intervals detections. In this work, the threshold values are specified with two methods, static and adaptive as discussed in section 3.1.1.

For the static mechanism, which uses fixed values for its thresholds, we vary the values for the upper threshold $V_{th,h}$. However, it is unfair to use the same absolute values to compare between filters, as we expect that different filter instances (i.e different Coefficient Window) have different maximum absolute outputs, . Therefore, we evaluate the upper threshold value as relative to the maximum filter output during seizure intervals, to do so, we consider 70% as least-required detected seizures. So we want to vary a range of $V_{th,h}$ values between a minimum and maximum level. The maximum $V_{th,h}$ is

set as the value that detects 70% of all annotated seizure intervals. The minimum $V_{th,h}$ is set at a value that is 75% of the highest $V_{th,h}$ value that detects 100% of all annotated seizure interval level. In this threshold range we consider 64 steps to test different $V_{th,h}$ values. Because the lower threshold Vth,l is only used as safety mechanism to prevent multiple detections in the same seizure interval, we used a single value close to the average value of non-seizure interval.

On the other hand, the adaptive mechanism uses $V_{th,h}$ values that are dependent on the filter's output. Specifically, two τ values, τ_{up} and τ_{down}, determine how the threshold value is adapted, increasing and decreasing respectively (as described in section 3.1.1), and in turn they affect detection performance. Therefore, we vary the τ values as follows:

$$\tau_{up} = j \ , \ j \in [1,30]$$
$$\tau_{down} = i * 10 \ , \ i \in [1,150]$$

Our evaluation showed that larger τ values did not further improve detection performance, so we consider the above set representative. The Vth,l, which is defined by the running average, is again not varied for the same reason as in the static mechanism. The window size of the of the running average is set to N=10,000. In our evaluation, we observed that this window size allowed the Vth,l to sustain a value close to the values of inter-ictal periods.

ECoG Database

The ECoG Database used in this thesis' work was obtained from 24 awake tottering mutant mice that exhibit absence seizures. The mice were connected with Teflon coated silver ball tip electrodes (diameter 0.2 mm, [51]) which were bilaterally placed above the primary motor cortices and primary sensory cortices. The ECoG signals were amplified using a Cyberamp amplifier [48] and recorded with Multiclamp 700A [48]. These recordings were made by neuroscientists at Erasmus MC.

The total duration length of the database is 29.75 hours and was divided in sets of 15 minutes. The database contains 1914 annotated seizures time-stamps which were created by an offline algorithm. The algorithm uses peak detection to find SWDs that exhibit a repetition rate of at least 5Hz and the interval of those that last more than 1 second are marked as seizures. The outcome of the algorithm was validated visually by a neuroscientist. In the following sections these annotations will be referred as golden standard annotations.

However, neuroscientists remain uncertain whether SWDs intervals of less than 1 second are not a seizure. Therefore, the initial annotations were extended manually by trained neuroscientists to include ictal-activity that contained at least 3 SWDs. In the following sections, these additional annotations will be referred as SWD annotations.

Evaluation procedure

The evaluation of the detection-filter performance is conducted using MATLAB®[52]. As our interest lies primarily with digital domain, we emulate the analog module functionality the parameters of which are fixed and sample the input values at the sampling frequency of the closed-loop system. We subsequently vary the aforementioned input parameters and evaluate the detection mechanism on the performance metrics described above in order to derive appropriate solutions for an efficacious system implementation. In our evaluation we consider the static and adaptive mechanism and evaluate each with different parameters as described below.

For the *Static threshold mechanism* case, we assess the detection performance varying the Upper Threshold $V_{th,h}$ and the Coefficient Window input parameters. The performance results are computed using both golden standard annotations and SWD annotations. We conclude the mechanism's evaluation by considering all the performance metrics derived from the golden annotations as optimization objectives and compute which are Pareto optimal. Subsequently, we approximate the Pareto front that is formed by these points.

For the *Adaptive threshold mechanism* case, we evaluate the detection performance for different values of τ_{up} and τ_{down}. In order to do so, we select the FIR filter implementation that derived from our preliminary study for the prototype implementation (section 3.2). The performance results were computed using *SWD annotations* as these results are close to the performance results of a closed-loop system. Additionally, in order to obtain sufficiently stable values for the thresholds ($V_{th,h}$ and $V_{th,l}$), the adaptive mechanism needs to be initialized. In the evaluation process, we allowed the thresholds to settle with ECoG input values that contained a set of 4 seizures per subject.

Finally, we select single representative instances from both mechanisms and compare their performance results with other existing closed-loop implementations.

4.1.2. COMPUTATIONAL OVERHEADS

In this section we present the evaluation setup for the computational overheads. First, we describe the platform that is used to calculate the overheads and the details of the ported software implementation. Then, we define the profiling statistics that quantify the overheads and finally we analyze the procedure we follow to simulate and evaluate these costs.

PLATFORM AND SOFTWARE IMPLEMENTATION

The computational overheads for the software implementation of the closed-loop system are measured by executing it on the SiMS architecture processor [1], which is an ultra low power architecture. The architecture details of this processor are described in section 2.3.1, runs at a frequency of 20 MHz, and has an average power consumption of 1078.93 μW ([53],[1]). The implemented processor targets implantable biomedical implants, which is also the intended target of our closed-loop system.

In order to port the software implementation to SiMS, we created a version by stripping the Beaglebone software implementation discussed in section 3.2.2. More importantly, the ported version is more representative of an actual application, as it contains no extra functionality other than the detection and triggering mechanism. In detail, the simulated components involve waiting for a new sample to arrive, computing the FIR filter's response, and deciding on the existence of a seizure (detection mechanism).

To emulate sensory input, ECoG sample values are stored in the source code in global arrays and are given as input during execution. All data read and data writes to peripherals are modeled by performing a memory operation of 1 clock cycle. The stimulator activation is modeled by writing the value of 1 to a pre-defined memory address. Also, the circular history buffer was initialized with input samples in order to get a correct filter outputs at the first loop iteration without having to wait for the buffer to fill by incoming values.

The software application was compiled using the SiMS core compiler with O2 level optimization enabled. Then the binary's execution was simulated with ModelSim [54]

PROFILING STATISTICS

The profiling results generated from simulating the code on SiMS architecture in ModelSim, include the following metrics:

The *Instruction mix* specifies the percentages of the different types of instructions that are executed. The instructions of the ISA are grouped in 4 categories:1) ALU- arithmetic instructions and instructions, 2) BRA- conditional instructions, 3) MEM- load/store instructions. JUMP- jumping instructions. We calculate the instruction mix by analyzing the instruction trace of the simulated execution. The percentage identifies which of the instruction types are mostly used in our implementation and allows to identify the most prominent candidates for improvements.

The *Total instructions* describe the number of instructions committed during a single main loop iteration of the software application. It is computed from the instruction trace generated by simulating the software implementation and quantify the execution size of the application.

The *Execution cycles* are the number of cycles required to execute a single main loop iteration for the software implementation. This value is computed from the binary simulation on Modelsim. Using this value we can compute its execution time for a given clock period. The execution time allows us to estimate on Energy consumption (discussed below) and timing requirements of the implementation, that is in this case 10 ms per loop iteration.

The *IPC* metric specifies the average Instruction Per Cycle and is one of the performance metrics used for quantifying processor architectures performance. A higher IPC value means that the processor architecture is more efficient in executing instructions.

The *Multiplication overhead* is the overhead introduced by the emulation code for a multiplication as the SiMS processor architecture lacks a dedicated hardware multiplier unit. We compute the overheads by detecting the emulation blocks in the instruction trace. The overhead shows how it affects total performance and allows us to estimate the improvement which could be obtained, in terms of execution cycles and total instructions, by using a dedicated multiplier.

Memory usage comprises Instruction Memory (IMEM) and Data Memory (DMEM) usage by the software implementation. IMEM is determined by the executable binary size of the software implementation. DMEM usage is determined by the size of static data, global arrays, and maximal stack size during execution. Memory usage is important as the size of the memory unit greatly affects total power consumption [1]. Thus, by defining the minimum memory requirements of the implementation we can keep power consumption spent on memory to the minimum.

The *Energy consumption* is the energy required to execute a single iteration of the main loop on the SiMS processor. In order to calculate the Energy consumption, we first compute the execution time by multiplying the execution cycles with the cycle period of ($\frac{1}{20}ns$). Then, we multiply the average power consumption(1078.93 μW) with the execution time. Energy consumption enable us to estimate the costs required for a system realization that deploys this algorithm and how it will scale in relation to various parameters.

The *Battery-powered system lifespan* involves the estimation on the durability of a system that uses a battery with a capacity of 0.1Wh. The battery capacity of 0.1Wh was chosen as a reference because it is close to the capacity of a battery used in a wristwatch [55] which is relatively small in size. In order to calculate the lifespan, we first calculate the total number of loop iterations that can executed for this capacity with the pre-calculated energy consumption per loop. Then, using the main loop's execution frequency(sampling period f_{sample}=10 ms), we calculate the expected battery lifetime. We assume that after the main loop completes its execution, the system is powered down with a zero power consumption until the next iteration.

EVALUATION PROCEDURE

In this section, first we present the different parameters used to calculated the profiling statistics, after which, we explain how we obtain our profiling statistics. Finally, we present how we are going to assess the software implementation of the two detection mechanism (i.e. static and adaptive).

The detection mechanism's input parameters used to evaluate the computational cost is the coefficient window size (filter order) as the rest of the parameters do not affect the execution. The coefficient window size, determines how many multiplications and additions of the FIR filter need to be calculated. Therefore, we experiment with window sizes in a logarithmic scale from 1 to 64. In future implementations, we might require to analyze ECoG signals from multiple locations with the purpose of increasing detection accuracy (e.g stimulate only when seizure detected in all locations). Therefore, we consider how different numbers of input channels affect software execution and computational costs. The number of channels affect the total number of FIR kernels and the total number of buffers required to store all necessary data. In the evaluation process, we experiment with different number of channels in a logarithmic scale from 1 to 32. For each of the parameters a new binary was compiled and simulated.

The software code section that is profiled is only the main loop; initialization and boot-up code are excluded. The reason these parts are not included, is that they contain code only required for simulation which does not contribute to the actual runtime execution. Multiple loop iterations are executed to eliminate the effects of startup code and the results for the computational overheads represent the average values for a single iteration. Additionally, in order to measure IMEM usage we profiled a software implementation where emulated input is excluded. The software implementations for both the static and adaptive threshold mechanism are evaluated as follows:

In the case of the static threshold, we evaluate all the above metrics as a function of different coefficient window sizes and number of channels. First, we present how the selected parameters affect the average instruction profiling statistics per loop iteration. These include instruction-mix results, total instructions committed number, execution cycles and multiplication overheads. As for the multiplication overheads, these are computed for a software implementation with 1 (one) monitoring channel and a coefficient window size of 64. Then, we report on the memory requirements for each parameter, after which we evaluate which of the software implementations, in relation to the coefficient window size, meet the real-time constraint of 10 ms (sampling period of the ECoG input signals). In the end, we report on the Energy consumption requirements of the software implementation in relation to its input parameters where we also discuss the lifespan of a battery powered system.

In the case of the adaptive threshold mechanism, we evaluate the extra computational overheads that are introduced in comparison to the static threshold mechanism. The additional costs of the adaptive mechanism derive from the extra instructions and the additional variables required to store intermediate results and threshold values per channel. Since each channel requires its own threshold mechanism and the threshold calculation is not affected by the coefficient window size, the only parameter that affects the computational overheads is the number of monitoring channels. Therefore, in the evaluation results we present the additional clock cycles caused by the extra instructions, and the memory usage increase caused by the additional variables, in relation to the number of different channels.

4.1.3. DETECTION PERFORMANCE VS. COMPUTATIONAL OVERHEADS

In order to evaluate the overall performance of a low power architecture utilizing the presented closed-loop system, we first want to find the pseudo-optimal solutions and then see the trade-offs between detection performance and energy consumption. In both cases, we consider the results produced by the static threshold mechanism.

At first, we consider the all the detection performance metrics (i.e. Sensitivity, Specificity, and Detection Delay) together with energy consumption as optimization objectives. From these objectives we compute which are Pareto-optimal and construct the Pareto front where we can assess the performance of the system. Next, we compare in terms of ADR, energy consumption, and Detection Delay for various coefficient window sizes. This is done to estimate the costs of various solutions and how we trade-off detection accuracy for extended lifespan.

4.2. DETECTION PERFORMANCE RESULTS

In this section we present the performance results of the detection filter using the methods described in section 4.1. First, we describe the results of a full exploration on the selected input parameters of the detection system using a static threshold, after which we present the performance results of the adaptive threshold mechanism. Finally, we compare our work to other existing solutions.

4.2.1. STATIC THRESHOLD

In this section, we first present the results of each metric as a function of the input parameters. We do this for both golden standard and SWD annotations. Following, we describe the Pareto front defined by the Pareto optimal performance metric values.

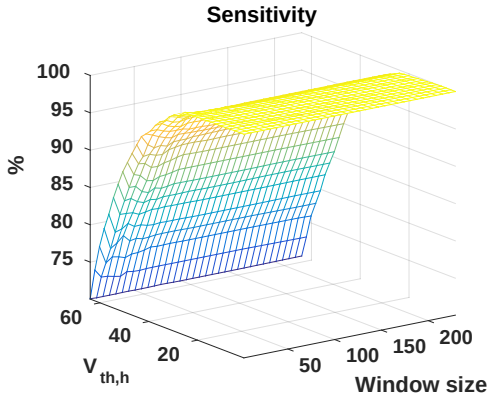
PERFORMANCE METRICS

The performance metrics consists of the Sensitivity, Specificity, and Detection Delay as functions of the input parameters, i.e. coefficient window(size and offset) and threshold value $V_{th,h}$. Given the complexity to study the effects of individual parameters on detection performance (3 inputs,3 outputs), we select a single value for one of the inputs and describe the effects of the remaining inputs. We start by selecting coefficient window offset 0 (symmetric impulse response) and demonstrate the effects of the threshold $V_{th,h}$ and coefficient window size. Subsequently, we select a coefficient window size of 40 and present the effects of the coefficient window offset.

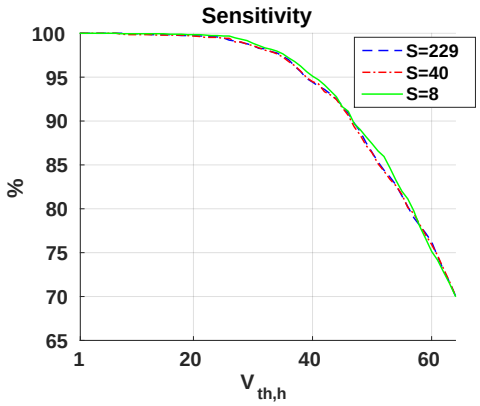
Threshold $V_{th,h}$ & Window size In this section, we present the effects of different *threshold* $V_{th,h}$ and *window size* values on the performance metrics by setting the coefficient *window offset* fixed at 0 and using the golden annotations (seizure>1sec). We present each of the performance metric separately starting from Sensitivity, then Specificity and moving on to the combination of two in the ADR metric. In the end we explain the results of Detection Delay.

Sensitivity results are presented in Fig. 4.3, where Fig 4.3a shows the 3D view of the Sensitivity as function of $V_{th,h}$ and Window size, Fig 4.3b/4.3c depict the 2D aspects as a function of one parameter ($V_{th,h}$ and Window size respectively) while using representative values for the other. First, let us see the effect of $V_{th,h}$ on Sensitivity for a given window size, as depicted in 4.3b. Starting at $V_{th,h} = 1$ and up to $V_{th,h} = 25$, we find that Sensitivity is 100% for a given window size, which means all seizures periods in the ECoG database are detected. This is also seen in 4.3b where all indicative window sizes have a Sensitivity of 100% and in 4.3c where for $V_{th,h} = 25$ Sensitivity is fixed at 100%. For these threshold values all seizures are detected, as we select the starting point of the threshold levels ($V_{th,h} = 1$) at a value that is guaranteed to contain no False Negatives (FN). It is evident that we remain within that range up to $V_{th,h} = 25$. For increasing values of threshold level above 25, Sensitivity starts dropping, which means that the threshold level exceed filter outputs of certain seizure intervals leading to False Negatives (FN). Finally, at the maximum $V_{th,h} = 64$ we see that sensitivity gets as low as 70% for all window sizes, Fig. 4.3c. This is because our max threshold level is such that will miss 30% of all seizures, recall from section 4.1.1. Finally, we observe that different values for window size have no effect on Sensitivity, as is evident from the monotonous lines in Fig. 4.3c. Although different windows sizes do affect the filter's output range, Sensitivity is not affected as a function of the threshold levels relative to this filter's output.

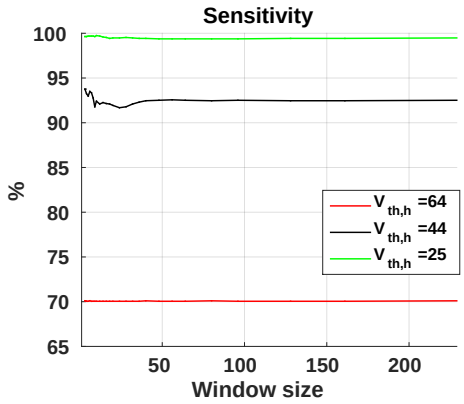
Specificity results are presented in Fig. 4.4, where Fig 4.4a shows the 3D view of the Specificity as function of $V_{th,h}$ and Window size, Fig 4.4b/4.4c depict the 2D aspects as a function of one parameter ($V_{th,h}$ and Window size respectively) while using representative values for the other. First, let us consider the effect of $V_{th,h}$ on Specificity as a function of the threshold level for a given window size (Fig. 4.4b). Starting at $V_{th,h} = 1$, we notice that Specificity varies between 0% and 10% for window sizes 1 and 229 respectively. This can be seen in 4.4a by the increasing Specificity for different window sizes



(a) 3D plot of Sensitivity as a function of Threshold-$V_{th,h}$ and Window size



(b) Sensitivity as a function of Threshold-$V_{th,h}$ for selected window sizes

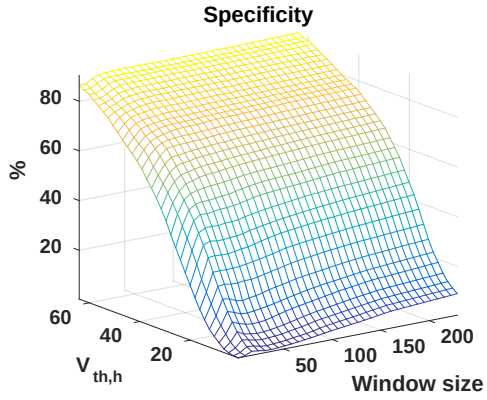


(c) Sensitivity as function of Window Size for selected Thresholds-$V_{th,h}$

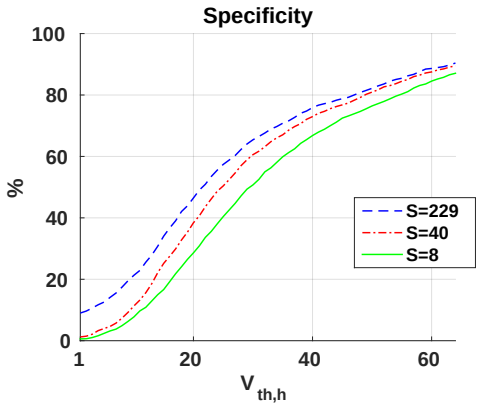
Figure 4.3: Sensitivity as a function of Upper Threshold $V_{th,h}$ and Coefficient Window Size

at $V_{th,h} = 1$ and in 4.4b by the Specificity values for the 3 window sizes. This low Specificity is caused by the fact that low threshold values are very close to the average filter response during inter-ictal periods and even small increases in the filter response can trigger the detection mechanism, i.e. high False Positive (FP) and low True Negative (TN) numbers. As the filter more accurately approximates the complex Morlet wavelet for larger windows sizes, the filter's output is higher for seizures periods than inter-ictal periods. As a result, Specificity is increased for window size 229. For increasing values of $V_{th,h}$, we observe in 4.4b that Specificity increases for all 3 window sizes, which results from reduced False Positives as the threshold values are high enough and don't detect at low filter outputs. Finally, for max $V_{th,h} = 64$, we find a Specificity over 87% for the all 3 window sizes, as seen in Fig. 4.4b. Even though we have set a significantly high threshold value, the results are affected by ECoG events that cause the filter's response to cross the max threshold level. These events may include SWD, SWD-like behavior and various artifacts (ECoG problems during recording).

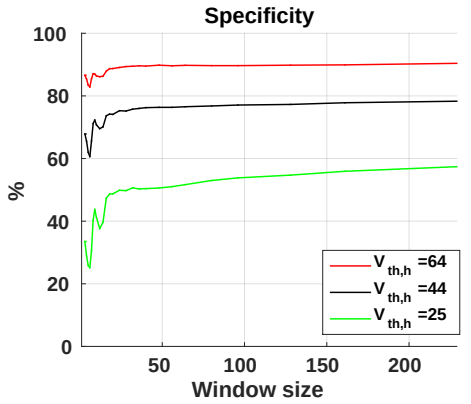
Let us now consider Specificity as a function of window size for a given threshold value $V_{th,h}$ (Fig. 4.4c). We find that different window sizes slightly affect Specificity, most noticeably for smaller window sizes (s<16), Fig. 4.3c. This is because the filter's seizure-detection capabilities are greatly reduced resulting in low True Negatives and high False Positives, when truncating the initial FIR impulse re-



(a) 3D plot of Specificity as a function of Threshold-$V_{th,h}$ and Window size



(b) Specificity as a function of Thresholds-$V_{th,h}$ for selected Window Sizes



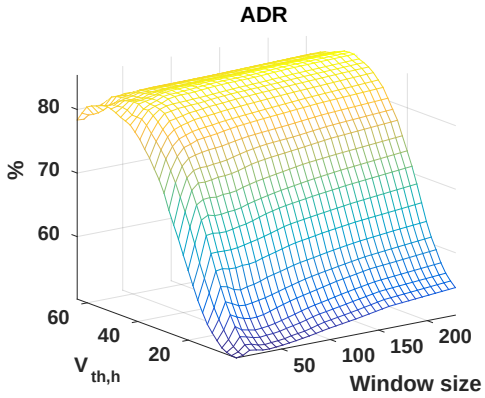
(c) Specificity as a function of Window sizes for selected Thresholds-$V_{th,h}$

Figure 4.4: Specificity as a function of Upper Threshold $V_{th,h}$ and Coefficient Window Size

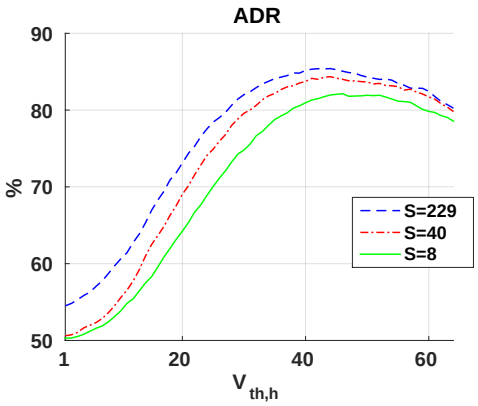
sponse. For increasing the window sizes (>16), we find that Specificity is improving. Finally, for max window size (S=229) we find that the maximum Specificity is reached for each of the threshold values.

We observe from the previous results that Sensitivity and Specificity have opposite behavior in relation to threshold levels, while window size only affects Specificity. While Sensitivity decreases when threshold levels are raised, Specificity is increasing. In Fig. 4.5 we present the results for *ADR* which quantifies the average of the two metrics. We find that for $V_{th,h} = 1$ ADR is at its lowest values between 50% and 55% , as seen in Fig. 4.5b and attributed to the Sensitivity of 100% and Specificity of 0%. Increasing threshold levels also increases ADR, this is because Specificity improves at a higher rate than the decrease rate of Sensitivity. Between threshold levels $V_{th,h} = 42$ and $V_{th,h} = 46$ we have the maximum ADR for each of the selected windows, yielding ADR values up to 85%. Further increasing threshold values results in decreasing the ADR as a result of the decreasing values of Sensitivity for higher threshold values while Specificity increases at smaller rate .

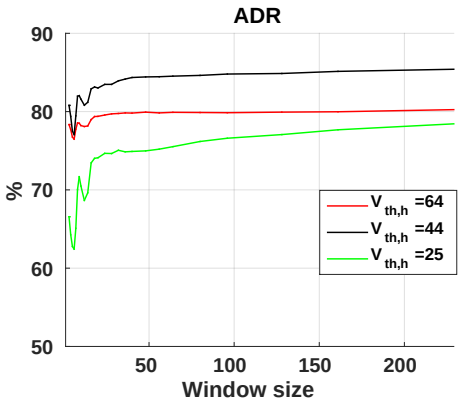
For different window sizes (Fig. 4.5c), we find that ADR follows the same trends as Specificity which is expected as Sensitivity remains constant for different window sizes. In this sub-figure (Fig. 4.5c), we find again that the in-between threshold of $V_{th,h} = 44$ has better ADR results than the other $V_{th,h} = 64$



(a) 3D plot of ADR as a function of Threshold-$V_{th,h}$ and Window size



(b) ADR as a function of Threshold-$V_{th,h}$ for selected Window Size



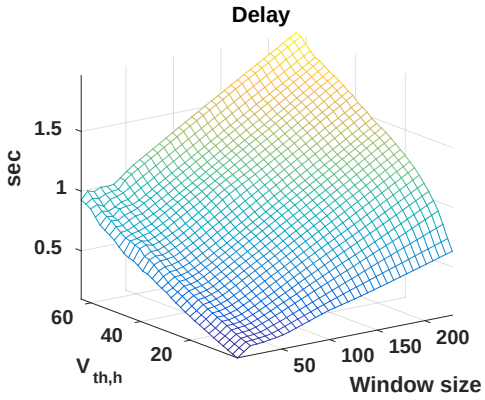
(c) ADR as a function of Window sizes for selected Threshold-$V_{th,h}$

Figure 4.5: ADR as a function of Upper Threshold $V_{th,h}$ and Coefficient Window Size

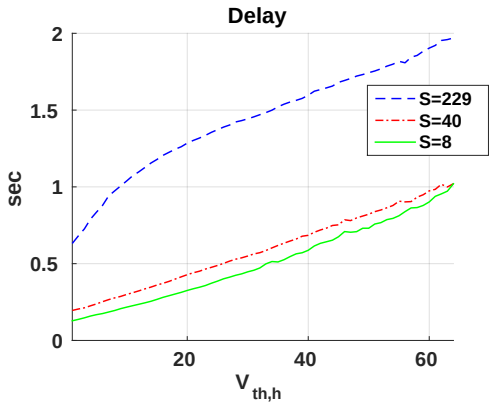
and $V_{th,h} = 25$, something which is also seen above where maximum levels of ADR are found around thresholds of $V_{th,h} = 42$.

We now present *Detection Delay* results as seen in Fig. 4.6, where Fig 4.6a shows the 3D view of the Detection Delay as function of $V_{th,h}$ and Window size and Fig 4.6b/4.6c depict the 2D aspects as a function of one parameter ($V_{th,h}$ and Window size respectively) while using representative values for the other.

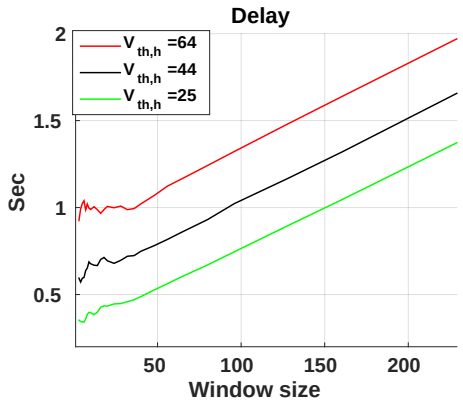
First, let us consider the effect of $V_{th,h}$ on Detection Delay as a function of threshold for a given window size (Fig. 4.6b). Starting from $V_{th,h} = 1$, we find that detection delay varies from 198 ms to 729 ms for window sizes S=8 and S=229 respectively. This can be seen in Fig. 4.6a by the increasing Detection Delay for various window sizes at $V_{th,h} = 1$ and in Fig. 4.6b for the depicted windows. This is because a low threshold value triggers a detection at filter outputs that are closer to the onset of a seizure period (recall from Fig. 4.1). For increasing values of threshold levels detection delay increases, which is a result of detecting at a higher filter response which happens is at a later time than the seizure's onset. Finally, at the maximum $V_{th,h} = 64$, we find the maximum detection delay for each of the selected window. The large difference in detection delay for S=229 and other window sizes at maximum $V_{th,h}$, is because the maximum window size contains many near-zero values in its start that delay high filter output values (see Fig. 3.3). While smaller window sizes include less, to none,



(a) 3D plot of Delay as a function of Threshold-$V_{th,h}$ and Window size



(b) Detection Delay as a function of Window Size for selected Thresholds-$V_{th,h}$



(c) Detection Delay as a function of Threshold-$V_{th,h}$ for selected Window Sizes

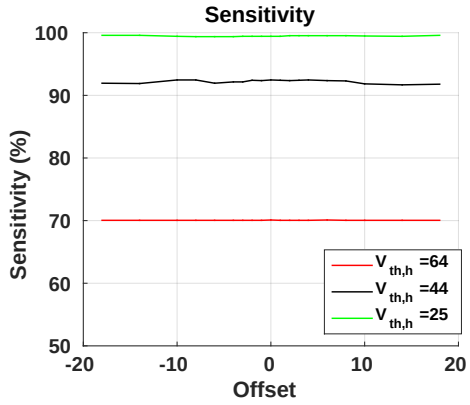
Figure 4.6: Detection Delay as a function of Upper Threshold $V_{th,h}$ and Coefficient Window Size

near-zero coefficient values the Detection delay results are relatively similar.

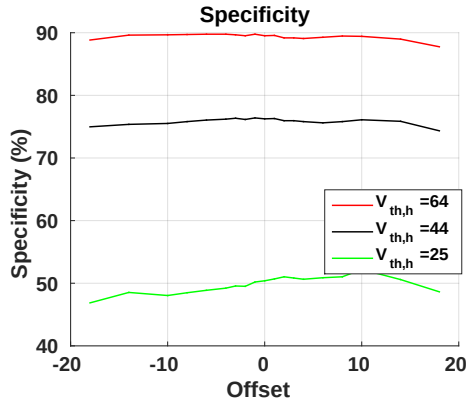
Let us now consider Detection Delay as a function of window size for a given threshold value $V_{th,h}$ (Fig. 4.6c) We observe that different window sizes affect detection delay results in a linear manner for each of the selected threshold values, as seen in Fig. 4.6c. Smaller window sizes contain less near-zero coefficients and produce faster a high response value compared to larger window sizes. Subsequently, increasing values of window size increase Detection Delay, where for all selected thresholds we observe an increase of 1 second in the Detection Delay when increasing from window size 1 to window size 229.

Summing up the trends for different values of threshold levels and window sizes that the biggest effect is caused by the threshold levels. Specifically, we find that increasing the threshold levels improves Specificity, lowers Sensitivity and increases the Detection delay. On the other hand, we observe that increasing window sizes mainly affect and worsen the Detection Delay results. Additionally increasing window sizes slightly improve Specificity, while having a minimal impact on Sensitivity.

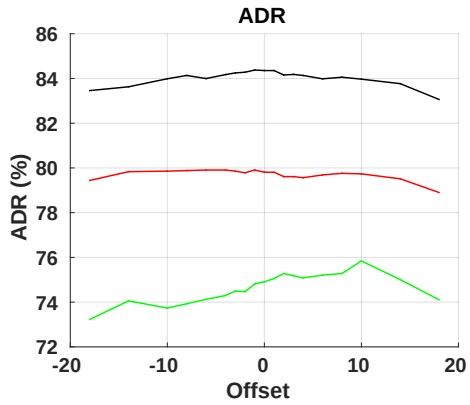
Window Offset In the previous section, we kept the window offset at 0, while varying the other two parameters (i.e. window size and threshold). Let us now consider shifting the window offset, where



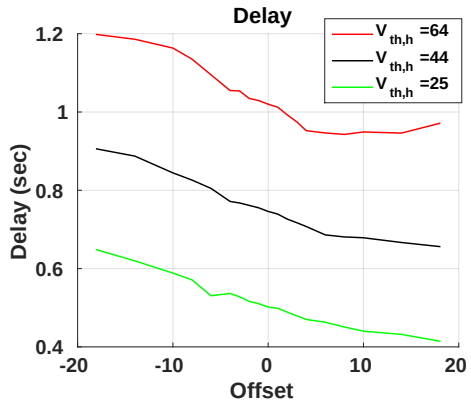
(a) Sensitivity as function of Offset with selected Thresholds-$V_{th,h}$



(b) Specificity as function of Offset with selected Thresholds-$V_{th,h}$



(c) ADR as function of Offset with selected Thresholds-$V_{th,h}$



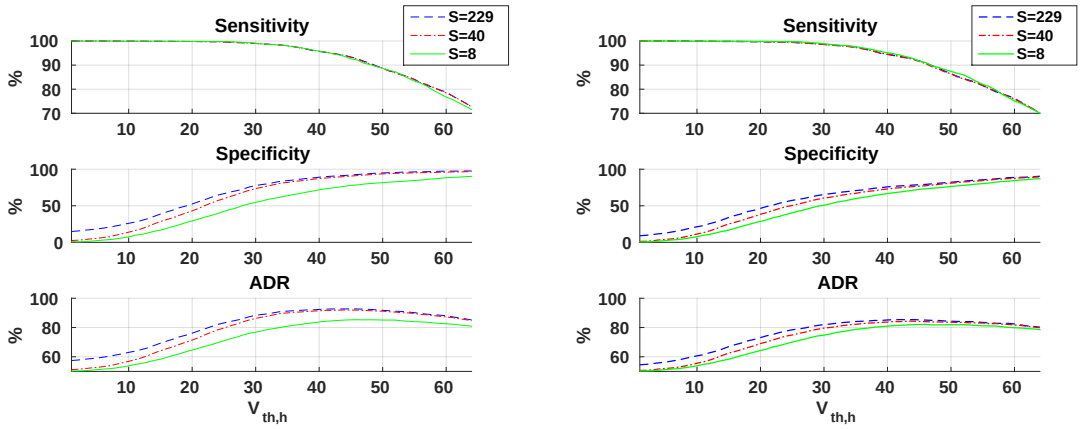
(d) Detection Delay as function of Offset with selected Thresholds-$V_{th,h}$

Figure 4.7: All performance metrics as function of Offset for a fixed window size and selected Thresholds-$V_{th,h}$

a negative offset means that the coefficient window is shifted to the left and a positive offset that it is shifted to the right (recall from figure 4.2). In doing so, we hope to achieve better Detection Delay by shifting the "significant peak" closer to the start of the FIR filter. In Fig. 4.7 we present performance metrics as a function of the offset for 3 indicative threshold levels (25,44,64) and a coefficient window size of 40. We consider a window size of 40 representative as we observed similar trends for other window sizes and it was also highlighted in our previous results.

We find that Sensitivity (4.7a), Specificity (Fig. 4.7b), and ADR Fig. 4.7c are not substantially affected by various offset values. This results from the fact that we maintain the center of the initial impulse response, which contains most significant coefficients values. Hence, by shifting the coefficient window the FIR filter output range is slightly affected.

In Fig. 4.7d, we present the *Detection Delay* performance as a function of the window offset. Starting at an Offset O=-18 we find the highest Detection Delay for each of the 3 threshold levels. This is because shifting the coefficient window to the left the appearance "singificant peak" of the impulse response is delayed. Shifting the window to the right, towards positive values, we find that Detection delay is decreasing, which results from bringing the significant peak closer to the start of the FIR filter. At the max offset considered O=18, we find the smallest Detection Delay and the improvement gained in Detection Delay by shifting from the leftmost to rightmost offset is approximately 200 ms for a given threshold.



(a) SWD annotations

(b) Golden standard annotations

Figure 4.8: Detection performance using golden standard an SWD annotations

SWD annotations So far, we have considered the performance metrics using the golden standard annotations. In this section we discuss the results with SWD annotations in place. As mentioned in the evaluation setup, with these annotations SWD intervals that trigger a detection are classified as True Positives which in turn reduces the number of False Positives. In Fig. 4.8 we present the performance results for SWD annotations as a function of the threshold level (Fig. 4.8a) and as a comparison we include the previous results next to them (Fig. 4.8b).

We find that *Sensitivity* results for SWD annotations compared to the results for golden annotations are relatively similar, as seen in In Fig. 4.8. This is because SWD intervals have on average the same filter output range(max value) of filter output values as seizure intervals. In the example Fig. 4.9 we observe that both filter output within the intervals is roughly the same, seizure interval in Fig. 4.9a and SWD interval in Fig. 4.9b. Hence, the number of detected intervals (seizure and SWD) depends on the threshold values, where we find that the average Sensitivity results do not change.

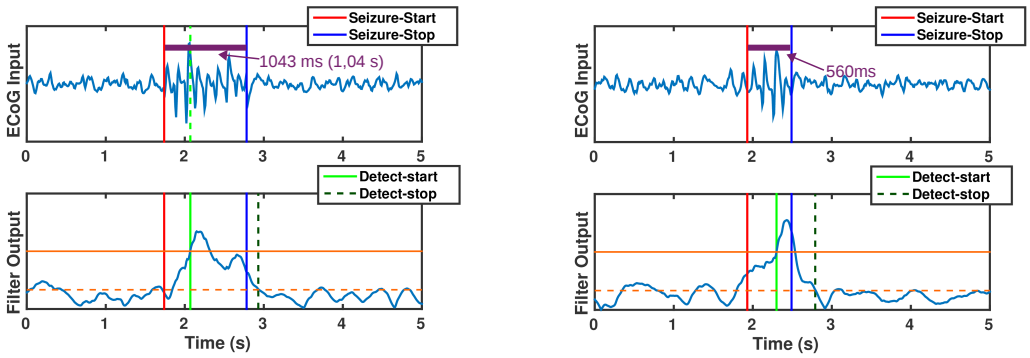
On the other hand, *Specificity* results are increased compared to the golden annotation results, as it is noticed in Fig. 4.8. The improved Specificity results from not counting detections during SWD intervals as False Positives but as True Positives. Starting $V_{th,h}$=1 we find that Specificity varies from 0% to 14.9% for SWD annotations compared to 0% and 10% for golden annotations. For increasing values of threshold levels, we observe that Specificity increases in the same manner, where the values obtained using the SWD annotations are slightly improved. For max $V_{th,h}$=64 we see that Specificity for reaches values up to 96.5% compared to 90% of the golden standard annotations.

Since Specificity is increased, subsequently this also affects *ADR* values. In the bottom graph of Fig. 4.8a, we find that the maximum *ADR* values reach 92%, whereas in the golden annotations4.8b we have values up 85%, an increase of 7% for the same threshold region.

Finally Detection delay results do not change from the new annotations. This is because in the golden annotation, we have found that a detection is triggered after a few SWDs, usually after 3 SWDs (e.g. Fig. 4.9a) depending on the threshold level, and therefore, also SWD intervals containing 3 or more SWDs and have similar filter output values will also be detected at the same. An example of such an SWD interval is depicted in Fig.4.9b.

PARETO FRONT

Carrying out an evaluation where we vary 3 input parameters and obtain 3 output/metric values, we may derive a set of pseudo-optimal solutions (Pareto points), as depicted in Fig. 4.10a.



(a) SWD-interval>1sec

(b) SWD-interval <1sec

Figure 4.9: Detection triggering example for a seizure interval vs. SWD interval(<1sec)

As was done in the previous sections, we highlight Pareto points that correspond to a certain window size (S) from the input parameters (green asterisks S=8, red asterisks S=40 and blue asterisks S=229) to illustrate the various trends in each output-value (objective). In the rest of the subfigures of Fig. 4.10 we present the relations between each combination of metrics.

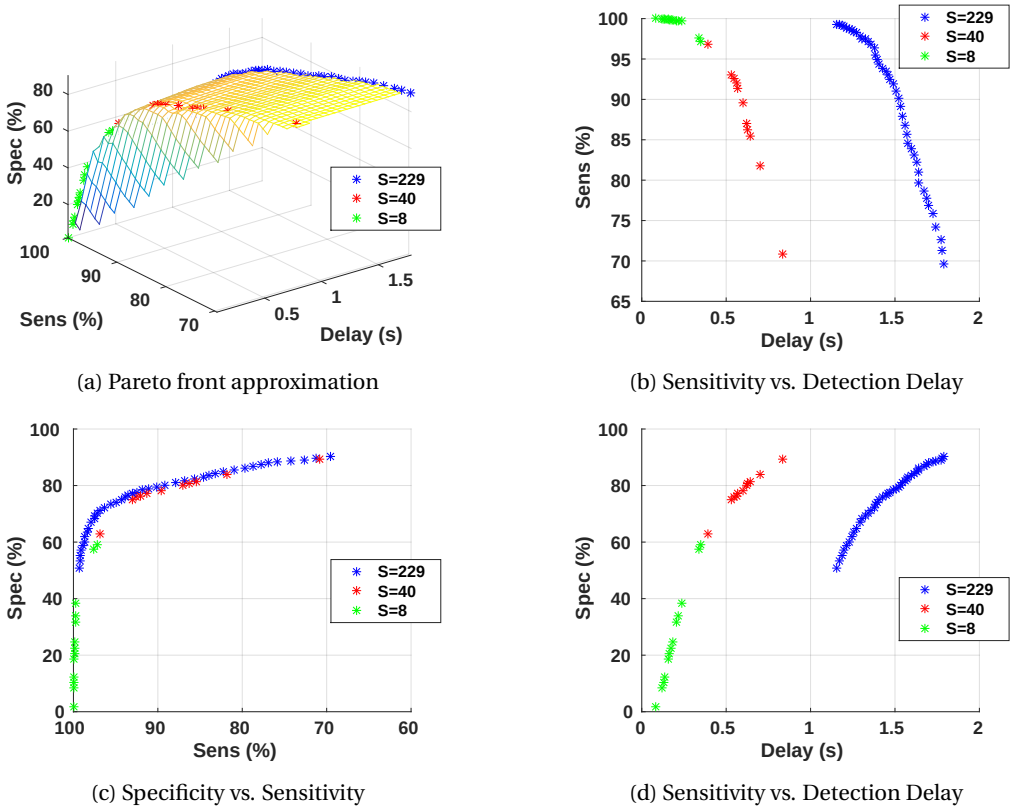
First, we see the relation between Sensitivity and Detection Delay in Fig. 4.10b, where we observe that for each represented window as Sensitivity decreases, the Detection Delay increases, that is worse. This is because of the effect of increasing threshold levels which deteriorates both metrics. Comparing between different window size we can get "free" improvement in delay as for similar Sensitivity values we can chose smaller window size that produce better Detection delay results.

Next, we see the relation of Specificity and Sensitivity in Fig. 4.10c, where we observe the counteractive behavior of the two metrics, while one improves the other worsens. This is because where Sensitivity is favored by low threshold values, Specificity is reduced. Nevertheless, we discover that for different window sizes there are Pareto points that have similar performance for Sensitivity and Specificity, which results from the small impact of different window sizes on the results of the 2 (two) metrics. This allows us to choose smaller windows without significant reduction in detection accuracy.

Lastly, in Fig. 4.10d, we see the relation of Specificity with Detection Delay, where we observe that as Specificity improves(increases), Detection Delay worsens(increases). We note, as in the Sensitivity case, that for same Specificity we can obtain a Pareto point that has a significant smaller Detection Delay by using a smaller window size. Even though for a smaller window size S=40 we achieve a max Specificity of 89.23%, the difference is very small compared to the maximum Specificity 90.25% (1% difference) obtained for a window size S=229.

SUMMARY - STATIC THRESHOLD MECHANISM

In this section we evaluated the role of window size, window offset, and threshold level in the seizure-detection performance of the filter. We have shown that large window sizes increase detection delay but improve Specificity results, while Sensitivity remains unaffected. Low threshold provide better Detection Delay and Sensitivity, but worse Specificity. An average best solution (based on ADR) may be found for threshold levels $V_{th,h}$=44 irrespective of window size. In addition, we saw that positive window offsets decrease detection delay while having minimal impact on Sensitivity and Specificity. Finally, from all the output parameters we calculated the Pareto optimal points which provide different options for creating an optimal filter. This study has revealed that even for small window sizes (S=8) we can achieve comparable detection accuracy to the max attainable accuracy (S=229) with a difference of 3% in the ADR values. What is more, for smaller window sizes we achieve better detection delay



(a) Pareto front approximation

(b) Sensitivity vs. Detection Delay

(c) Specificity vs. Sensitivity

(d) Sensitivity vs. Detection Delay

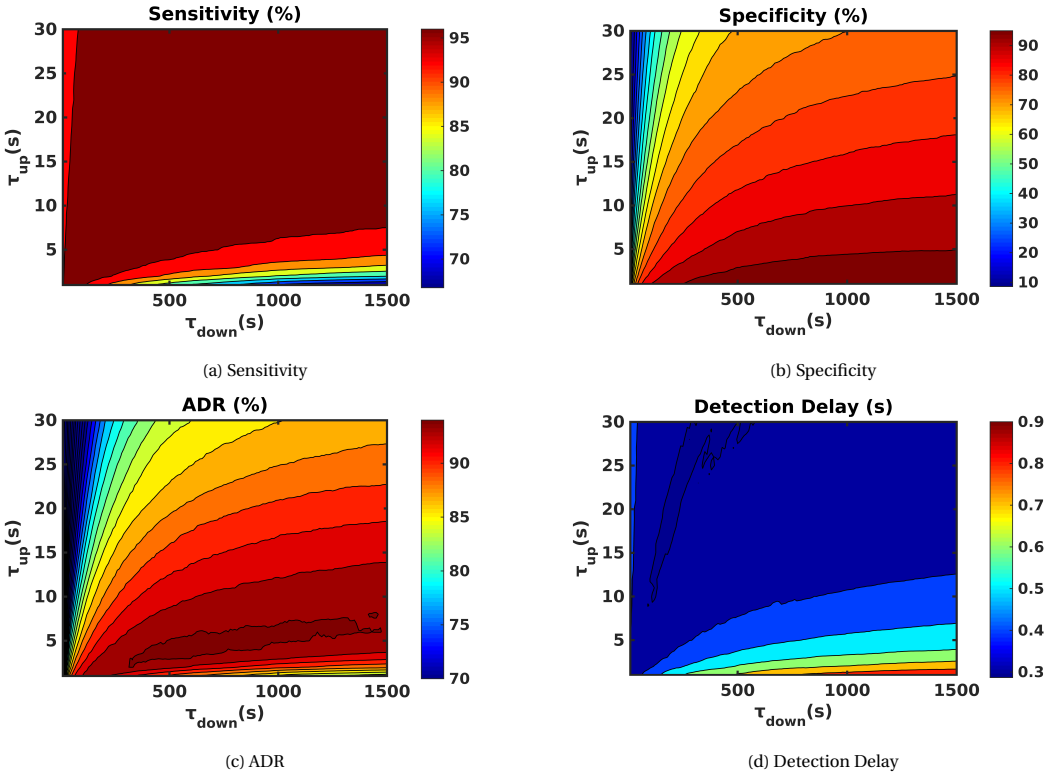
Figure 4.10: Pareto front for Sensitivity, Specificity, and Detection Delay. Highlighted are points that correspond to various window sizes(S=8,40,229)

results that are approximately 600ms for maximum ADR values.

4.2.2. ADAPTIVE THRESHOLD

In this section we evaluate the filter's performance on various τ values (τ_{up} and τ_{down}) for the adaptive threshold mechanism. Recall from section 3.1.1), the τ_{up} parameter defines the increase rate of the threshold value in response to high filter output values ($Y(n) > V_{th,h}$), and τ_{down} defines the drop rate of the threshold when low filter output occur. In this evaluation process we selected a single FIR instance (window size=64, window offset=18), which derived from our preliminary study for the prototype implementation (Section 3.2.2) .

Sensitivity results are presented in Fig. 4.11a. First, we examine the effect of τ_{up} for a given τ_{down} value. Starting at $\tau_{up} = 1$, we find that Sensitivity drops from 95% to 66% for τ_{down} values of 1 and 1500 respectively. This drop is explained by the the small τ_{up} value rapidly which rapidly ramps up that the threshold value and in combination with a large τ_{down} the threshold maintains high level resulting in many False Negatives. For increasing values of τ_{up}, we observe that for large τ_{down}(>200s) values Sensitivity also increases, while for small τ_{down}(<200s) values a maximum Sensitivity of 95% is obtained. Large τ_{up} and τ_{down} value signify that the threshold levels adapt at a smaller rate and thus maintain a relatively low level to detect more seizure intervals. Finally, for τ_{up} above 7 we find that Sensitivity attains values above 95%. Next, we consider Sensitivity as a function of τ_{down} for a given



(a) Sensitivity

(b) Specificity

(c) ADR

(d) Detection Delay

Figure 4.11: Detection performance as a function of τ_{up} and τ_{down}

τ_{up} value. At $\tau_{down} = 1$, we find that Sensitivity has values above 95% for all τ_{up} [1]. This is because small τ_{down} values ensure the threshold drops quickly for low filter outputs and thus creating low threshold values that detect most increased filter outputs. For increasing values of τ_{down}, we observe that Sensitivity drops when τ_{up}<7. This is because larger τ_{down} values only gradually decrease the threshold value and subsequently not reaching a level that will allow it to detect seizure intervals with low filter outputs.

Similarly, Specificity results are presented in Fig. 4.11b. We start by discussing the effect of τ_{up} for a given τ_{down} value. At $\tau_{up} = 1$, we find that Specificity varies from 66% to 98% for τ_{down} values of 1 and 1500 respectively. The increase for different τ_{down} is due slower drop rate of threshold which help it maintain a high value and subsequently avoiding False Positives. Increasing values of τ_{up} reduce Specificity for specific τ_{down} values. This is because at high τ_{up} the threshold follows at lesser magnitude high filter outputs and hence remains relatively low producing high False Positives. In the following step, we consider Sensitivity as a function of τ_{down} for a given τ_{up} value. At $\tau_{down} = 1$, we find the lowest Specificity for all τ_{up}, which results from the small τ_{down} that allow the threshold to rapidly drop at low filter outputs and generate high False Positives. For increasing τ_{down}, we observe that Sensitivity improves for a selected τ_{up}, as higher τ_{down} values prevent the threshold from dropping too quickly which would lead to incorrect detections. At maximum τ_{down} we find the highest Specificity ranging from 98% to 76% for τ_{up} of 1 to 30 respectively.

The mean value of *Sensitivity* and *Specificity* is depicted in the ADR plot of Fig. 4.11c. We start again

[1] The small reduction in Sensitivity for τ_{up}>15 is an artifact of our analysis scripts for low thresholds. Pre-onset seizure detections together with the safety mechanism of the low threshold cause certain seizures to be considered as False Negatives.

References	Line-length [31]	Daubechies wavelet [36]	Wavelet with ANN [13]	**Adaptive**	**Static**
Detection Delay	4.1s	1.5s	0.97s	0.492s	0.670s
FPPS	1.44	0.040	0.091	0.090	0.210
FNPS	0.0023	0	0.065	0.040	0.046
Specificity	N/A	N/A	98.2%	93.60%	88.60%
Sensitivity	N/A	100%	96.2%	96.03%	95.54%
ADR	N/A	N/A	97.2%	94.81%	92.07%

Table 4.1: Performance Comparison of various real-time seizure detection systems

by looking at the effect of τ_{up} for a given τ_{down} value. At $\tau_{up} = 1$, we find that ADR varies between 71% and 93% with its maximum at τ_{down}=170s, this is because Sensitivity drops for larger τ_{down} while Specificity increases. For increasing τ_{up} values we notice that ADR peaks at 94% around τ_{up}=7 for various τ_{down} values and then deteriorates following the same trends as in Specificity. This is because Sensitivity reaches its maximum value around τ_{up}=7 while Specificity continues to decrease.

Next, we consider ADR as a function of τ_{down} for a given τ_{up} value. At $\tau_{down} = 1$, we find the lowest ADR values (around 70%) as dictated by the Specifity value. For increasing τ_{down}, we notice that ADR also improves at a given window except for $\tau_{up} < 2$, which is again a result of the Specificity values whereas Sensitivity remains approximately constant for most values.

Finally, we present the results for *Detection Delay* in Fig. 4.11d. We first consider the effect of τ_{up} for a given τ_{down} value. At $\tau_{up} = 1$, we find that Detection Delay varies from 380ms to 900ms for τ_{down} of 1 to 1500 respectively. As seen in previous metrics, a small τ_{up} and τ_{down} values make the threshold follow the filter output keeping at relatively low values hence the low detection delay, for large τ_{down} the increased threshold values drop slowly keeping the threshold high which results in a high Detection Delay. For increasing τ_{up} values we notice that Detection delay is improves, which is a result of the slower increase rate of the threshold which keeps it at low levels and thus shorten the Detection Delay. For values of τ_{up} above 13 for all the τ_{down} values, we observe a Detection Delay of 300 ms. Next, we consider Detection Delay as a function of τ_{down} for a given τ_{up} value. At $\tau_{down} = 1$, we find a Detection Delay of 300ms irrespectively of the τ_{up} value and is caused by the high downwards rate which keep the threshold at low values. For increasing τ_{down}, we notice that Detection Delay deteriorates in the cases of τ_{up}<13. This is because higher values of τ_{down} keep the threshold relatively high and high threshold values increase Detection Delay. At maximum τ_{down} Detection Delay various between 900ms for the lowest τ_{up} and 320ms for the highest τ_{up}.

SUMMARY - ADAPTIVE THRESHOLD MECHANISM

In conclusion, small τ values make the threshold level more sensitive to changes, while large values will cause the threshold value to change more gradually. We observed that the two constants (τ_{up} and τ_{down}) differ significantly in magnitude, as was proved that it is preferred for threshold levels to increase rapidly at high filter outputs and gradually decrease when filter's outputs are low. This study has revealed the efficacy of the adaptive mechanism, which is capable of obtaining high performance results (up to ADR=94%) along with low Detection Delay (down to 300ms)

4.2.3. COMPARISON WITH RELATED WORK

In this section, we want to compare our detection system with other existing implementations of real-time seizure detection systems(Section 2.4). While the results in the previous sections for our detec-

tion system have shown many possibilities, we restrict our output by requiring a minimum Sensitivity of 95% while obtaining the maximum ADR. These values were selected as we want to ensure that we detect at least 95% of the all the seizures while keeping False positives to a minimum. Using these constraints we derive the performance results for a the detection system using the adaptive and static threshold mechanism and are compared to related work in Table. 4.1. In this table we only present systems that achieve an average delay of less than 5 seconds. Adding to the previous metrics, two extra are also calculated, FPPS and FNPS:

- *FPPS*, False-Positive-Per-Seizure rate is the number of False positives divided by the total number of seizures.
- *FNPS*, False-Negative-Per-Seizure rate is the number of False negatives divided by the total number of seizures.

Comparing the two thresholding mechanisms presented in this work, we find that the adaptive threshold outperforms the static threshold in all aspects. This is because the adaptive threshold is able to approximate the average values occurring in seizure intervals and provide better detections. On the other hand, the inability of the static mechanism to accommodate for filter output values, reduces its detection performance. In detail, the adaptive mechanism's higher sensitivity, compared to the static mechanism, is attributed to its capability to drop $V_{th,h}$ for lower filter outputs, hence reducing the number of false negatives. As for the increased specificity of the adaptive mechanism, this is because it slopes up when higher values are encountered. By this means, it is able to yield higher levels avoiding many false detections. Furthermore, detection delay is lower in the adaptive mechanism, as it gradually drops between detections that allow it to detect at a lower levels, hence detecting faster. Adding to the adaptive mechanism's performance advantage is its ability to tune its levels automatically if the signals levels change over time, whereas the static mechanism would require re-adjustment.

In Table 4.1, we find that both the adaptive and static mechanisms have better detection delays compared to other solutions. For instance, delay is lower by 500 ms (adaptive) and 320 ms(static) compared to the next best solution [13]. Looking at each solution separately, the overall improved results compared to the *line-length* method [31] derive from the finely tuned wavelet filter, which enables us to better distinguish seizure intervals. Concerning the other two methods [13, 36], which are also based on wavelets, our solution is faster as it utilizes a simple thresholding mechanism. Specifically, in [13] the use of a complex Artificial Neural Network increases the detection delay, but it provides better specificity. However, the main drawback of this solution is that the Artificial Neural Network requires an extensive training set (several hours), while the adaptive mechanism only needs a small training set (e.g. set containing 4 seizures). In the other wavelet based solution, Daubechies wavelet [36], the detection mechanism is based on the average value of successive filter outputs to improve detection accuracy (sensitivity of 100%). However, calculating an average delays the occurrence of high values, that trigger a detection, and in turn increases detection delay. Overall, the main benefit of our detection mechanism is its low implementation complexity.

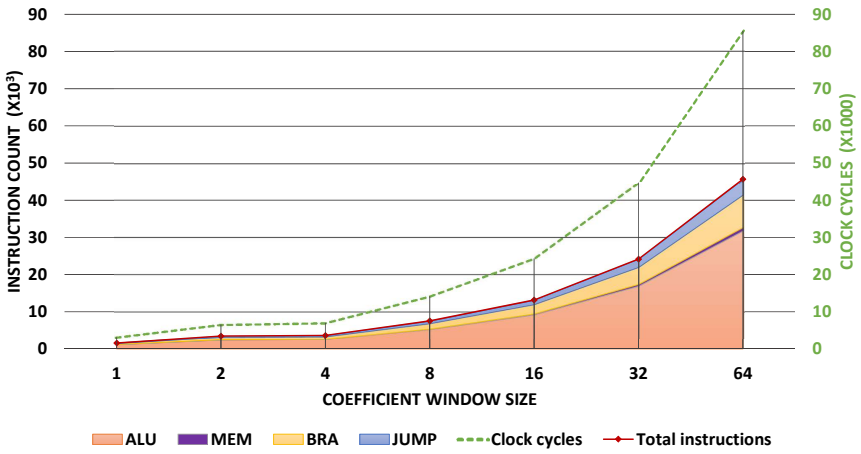


Figure 4.12: Instruction mix and execution time as a function of Coefficient Window Size (1 loop iteration)

4.3. COMPUTATIONAL OVERHEADS

In this section we present the profiling results of executing the software implementation on the SiMS architecture. First, we demonstrate the profiling results for the implementation with a static threshold mechanism discussing the instruction mix results, the real-time constraints, the memory requirements, and energy requirements. Secondly, we present the additional overheads which are introduced by using the adaptive threshold mechanism.

4.3.1. STATIC THRESHOLD

For the static threshold mechanism, we present all the execution costs of the software implementation as a function of different coefficient window sizes and different numbers of channels. We start by presenting the instruction-mix results and analyze the impacts of the multiplication emulation code, after which we discuss whether the software implementation meets the real-time constraints. Finally, we evaluate the memory requirements followed by the energy consumption estimation. In the following section instructions, we refer to the assembly instructions executed on the SiMS processor architecture as instructions.

INSTRUCTION-MIX RESULTS

Fig. 4.12 depicts the instruction mix and clock cycles as a function of different coefficient window sizes. Starting at a coefficient window size of 1, we find a total of 1,577 instructions being executed in 3,100 clock cycles. For increasing numbers of window size, we observe that the total number of instructions and execution cycles roughly double (2x) with the coefficient window size[2] Finally, at the max window size of 64 the total instruction count is 45,640 which are executed in 85,524 clock cycles. Based on the clock cycles of the execution and the instruction count, we calculate an average performance of 0.54 instruction per cycle for the SiMS architecture for a given window size.

Looking at the instruction mix of the execution trace, we notice that all different types of instructions follow the same linear increase as the total instructions. The percentage of each type remain, on average, the same for different window sizes. *ALU* instructions dictate the types of executed instructions at an average percentage of 70%. This is expected as the FIR algorithm consists mainly of arithmetic calculations. *BRA* instructions type with an average percentage 19%. Even though we observe a relatively high percentage of branching instructions, this is a result of the emulation code for

[2]Increasing window size from 2 to 4 does not double the results as a result of compiler optimizations.

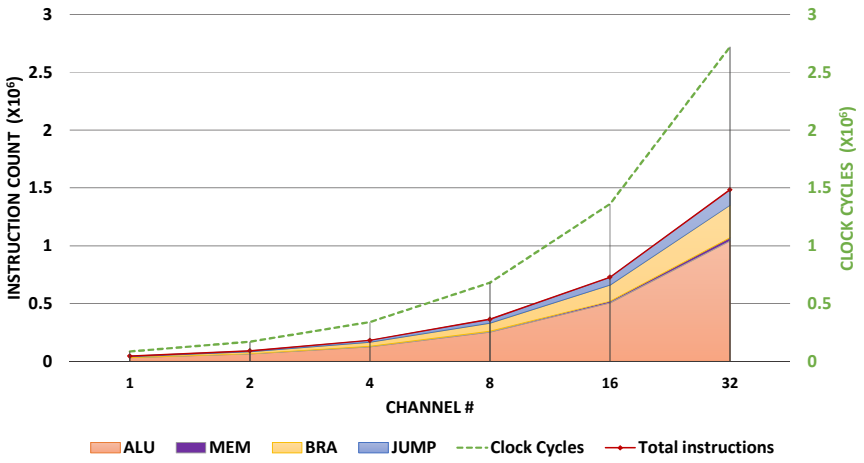
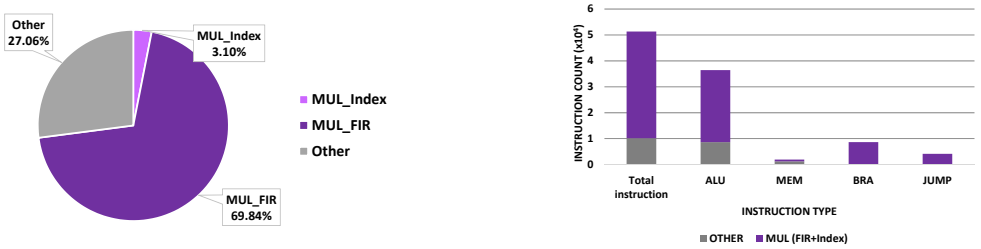


Figure 4.13: Instruction mix and execution time as a function of number of channels (1 loop iteration)



(a) Percentage of total clock cycles

(b) Percentage in instruction mix

Figure 4.14: Multiplication overhead in clock cycles and in executed instructions

multiplication and will be examined in a following section. The emulation code also affects *JUMP* instructions where in average they take 9% of the executed instruction. Finally, *MEM* instructions are in average 2% of the executed code. For larger coefficient window sizes (>64), we expect that the linear increase will continue in the same manner.

Next we consider the instruction mix and clock cycles as a function of different numbers of channels, the results of which are depicted in Fig. 4.13. Starting for 1 channel we find a total instruction count of 45,640 which are executed in 85,524 clock cycles. For increasing values of channels we observe a linear increase of 2 in the executed instructions for every doubling of channel numbers. For the maximum number of channels (32) we observe a total instruction count of $1.48*10^6$ which are executed in $2.72*10^6$ clock cycles. These results demonstrate how execution is affected if more channels are required to read inputs from multiple electrodes. Subsequently, we can estimate on the capabilities and limitations of a given platform.

Let us now see how instruction mix and execution cycles could behave with a dedicated multiplier. Assuming a software implementation with 1 (one) monitoring channel and a coefficient window size of 64. In Fig. 4.14 we depict the percentage of clock cycles (Fig. 4.14a) and the percentage of each instruction type (Fig. 4.14b) that are occupied by the multiplication emulation code. In these results, we have split the multiplication types in MUL_{index} and MUL_{FIR} to distinguish the operations between the ones that are instantiated indirectly for array indexing and the ones for the filter implementation. Array indexing multiplication MUL_{index} are used to compute the memory addresses of array elements and can be avoided by optimizing the software implementation. Fig. 4.14a clearly shows

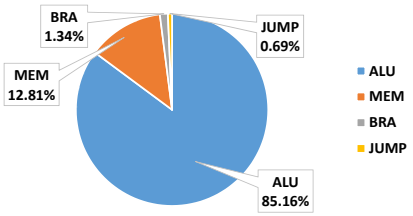
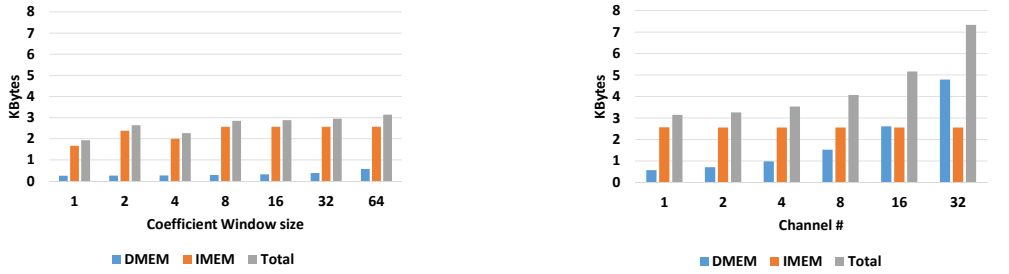


Figure 4.15: Instruction mix without emulation code



(a) Memory usage as a function of Coefficient Window Size

(b) Memory usage as a function of the number of Channels

Figure 4.16: Memory Requirements

that multiplication-related instructions occupy a large percentage of the execution time (73% of total), while Fig. 4.14b shows that over 80% of the total instructions are used for multiplication emulation. Moreover, we find that the majority of the BRA and JUMP instruction types are only instantiated by the multiplication emulation code.

Execution-time wise, these results motivate the use of a dedicated multiplier instead of emulating the multiplication. In Fig. 4.15 we estimate the resulting instruction mix if such a multiplier were employed. In these results we see that from a total of 10,222 instruction committed, 85.25% are ALU operations , 12.73% are memory operations (*MEM*), 1.33% are branch operations (*BRA*), and 0.69% are jump operations (*JUMP*). Note that branch and jump types now occupy only a small percentage in the instructions executed.

Based on this change in instruction mix we estimate execution time would also improve. In terms of timing performance, introducing a multiplier we could yield a reduction of 73% in execution time; assuming that the multiplier unit has a latency of 1 cycle and no data hazards occur. But even if we assume a shift and accumulate multiplier of 32 cycles and no data hazards, this would improve the number of cycles by 68.6% (down to 32 clock cycles from 530). In addition, the use of a dedicated multiplier unit will reduce the size of the executable binary and the amount of total executed instructions.

REAL-TIME CONSTRAINT

In the closed-loop system, ECoG input signals are sampled every 10 ms. This event introduces a real-time constraint where all calculations are required to finish within a time window of 10ms. Using the number of clock cycles presented in the previous section and dividing these values with the frequency of 20MHz (frequency of the low power implementation of SIMS as presented in [1]), we calculate the execution time for different Coefficient window sizes. These values are depicted in Fig. 4.17 along with the execution upper-limit (dashed red line). From the figure, we find that for coefficient window size up to 149 the execution time remains within the window of 10ms.

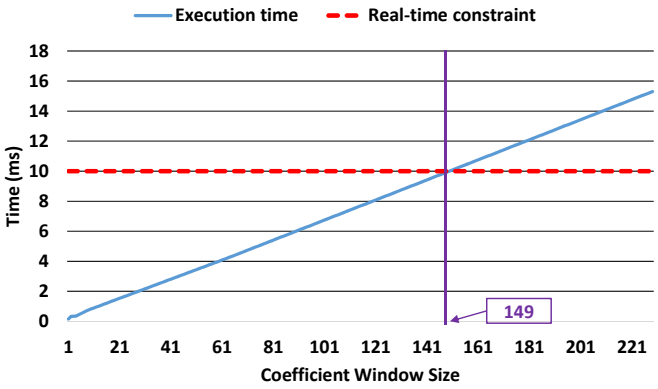


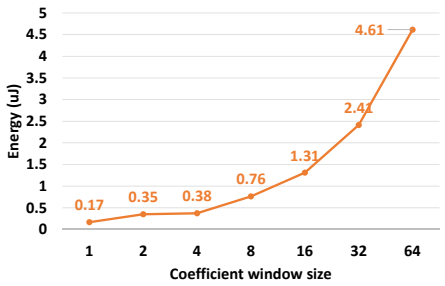
Figure 4.17: SiMS execution time for different window sizes for 20MHz clock frequency

MEMORY REQUIREMENTS

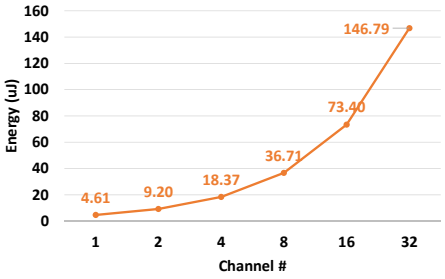
In Fig. 4.16 we present memory usage statistics, IMEM and DMEM, for different coefficient window sizes and different number of monitoring channels.

For the IMEM usage, we present how the it scales for different window sizes in Fig. 4.16a. Starting at window size 1 we find the smallest value, this is because no actual iteration occur for the filter calculation, hence we estimate loop instructions are eliminated and hence the number of the instructions in the executable are reduced. For increasing window sizes the IMEM memory usage remains constant, except for window size 4 where compiler optimizations take place and reduce the size of the executable. As for different number of channels we find that IMEM is not affected, as shown in Fig. 4.16b. In both cases, we observe that it maintains the same value 2,5KBytes across different parameters (Figures 4.16a and 4.16b). The reason it remains the same is because the parameters are hard-coded values and only determine the amount of loop iterations executed in various parts of the software implementation which do not affect executable size. The size of the executable would only change if loop unrolling optimization was used.

Next, DMEM memory usage comprises static data, global data, and the stack, hence it is more dynamically defined. In the global data region we have the array structures that store the FIR filter's coefficient values and they scale linearly with respect to coefficient window size. In the stack section, we find all the function call stack allocations which include the data structures responsible for each channel's information, as they are declared in the body of the main function which scale with the number of channels. In detail, in Fig. 4.16a we depict the effects of different window sizes on DMEM usage. Starting at 260 bytes for window size 1 it increases to 576 bytes for window size 64. This is mainly attributed to global data that contain the arrays for the coefficient values and hence increase as the window size increases. As for the stack size we observe that its size remains constant up to window size 32, this is probably because of a minimum stack allocation policy of 256 bytes. In Fig. 4.16b we observe a significant impact of different numbers of channels on DMEM usage. For a single monitoring channel we have a DMEM usage of 572 bytes which increases to 4,532 bytes for 32 monitoring channels. Breaking down DMEM usage we have: 1) global data which are constant at 256bytes (coefficient arrays), 2) a "dynamic" part of the stack data which contains the data structures for each monitoring channel, 136 bytes per channel, and 3) a constant part of the stack data of 184 bytes which contains the rest of the variables. Therefore, only the "dynamic" part changes which is linear to the increase of the channels size.

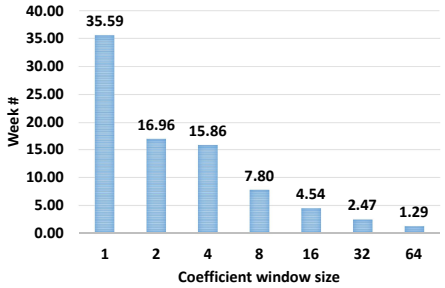


(a) Energy consumption as a function of the Coefficient Window Size

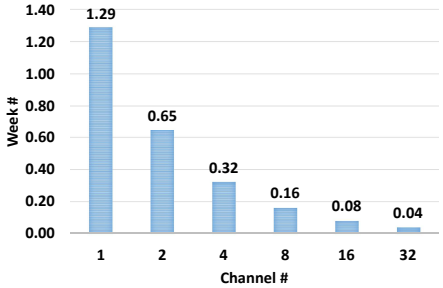


(b) Energy consumption as a function of the number of Channels

Figure 4.18: Energy consumption in uJ per single loop iteration as function of the Coefficient window size and the number of Channels



(a) Battey lifetime as a function of Coefficient Window Size



(b) Battey lifetime as a function of number of Channels

Figure 4.19: Battery lifetime in weeks as function of the Coefficient window size and the number of Channels

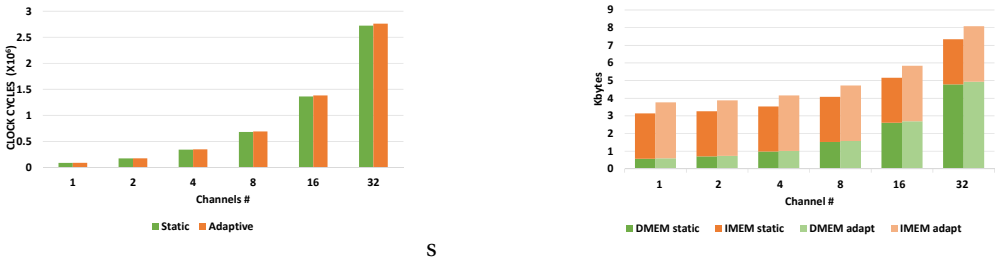
ENERGY REQUIREMENTS

In this section, we present the energy requirements of the software implementation and the expected lifetime of such a system on a battery of 0.1Wh, as described in section 4.1.2. In Fig. 4.18 we depict the Energy consumption of a single single loop iteration for different values of the parameters. Energy consumption is calculated by the product of the executed clock cycles and a constant power consumption. Accordingly, we observe the same increasing behavior as in the instruction mix section. The Energy consumption for a single channel varies between of 0.167μJ and 4.613μJ for window size 1 and 64 respectively. For the maximum number of channels 32, with 64 coefficients for each channels, we find an Energy consumption of 146.79μJ.

Given the energy consumption for a single iteration, in Fig. 4.19 we depict the expected battery lifetime in weeks for different values of the parameters, where we assume no energy is consumed when not calculating. For a coefficient window size 1 we find a lifespan of 35.59 weeks which decreases to 1.29 weeks for 64 coefficient window size. Obviously lifetime decreases when energy consumption is increased for different window sizes. This also applies to battery lifetime in relation to the number of channels, where the lifetime decreases by 2 for every doubling in the number of channels. For the maximum number of channels 32, we find a life expectancy of 0.04 weeks.

4.3.2. ADAPTIVE THRESHOLD

So far we have considered the computational overheads of the static threshold mechanism. We now briefly consider the additional overhead imposed using the adaptive threshold mechanism in Fig. 4.20. First, in the clock cycles overheads graph (Fig. 4.20a), it can be noticed that the overhead is relatively insignificant compared to the execution time of the static mechanism. For all the different



S

(a) Clock cycles overheads

(b) Memory usage overhead

Figure 4.20: Computational overheads of adaptive threshold mechanism in comparison with static threshold mechanism for various numbers of Channels

numbers of channels we observe in the adaptive case an additional 1.5% clock cycles to the static case. For example, for the static case we find $2.72*10^6$ clock cycles for the maximum number of channels and for the adaptive case we have $2.76*10^6$ clock cycles. This is expected as the Detection(3.10) is only a minor component relative to the FIR kernel.

In the memory usage overhead graph (Fig. 4.20b), we present the increase in DMEM and IMEM usage for the adaptive mechanism. We find the largest impact is seen in IMEM usage. While the execution time is hardly increased, code for the adaptive does have to be stored, resulting in an increase in IMEM of 23.2%.. However, IMEM remains constant across different numbers of channels. In contrast, DMEM memory usage portrays the same usage behavior as the static mechanism; due to requiring only 2 extra variables per channel that store the upper and lower threshold values.

4.4. DETECTION PERFORMANCE VS. COMPUTATIONAL OVERHEADS

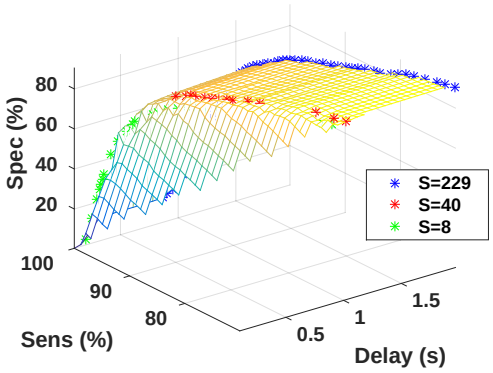
In this section, we combine the results of the filter's performance and the computational cost to evaluate the overall performance of the implementation on a low power processor. First, we consider a new Pareto front which includes both Energy and Pareto points from the Detection Performance, after which we look at particular optimizations (in terms of lifetime, ADR, and delay) that may be made in relation to the coefficient window size.

4.4.1. PARETO FRONT

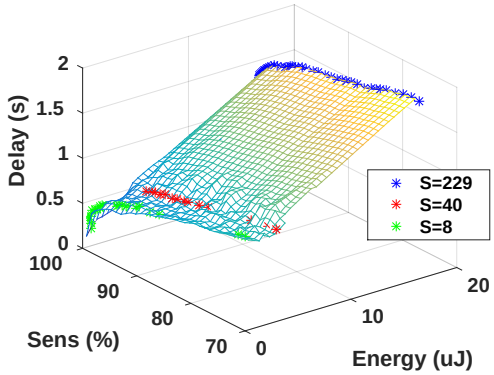
In Fig. 4.21 we present the Pareto front defined by the Pareto point for each combination of the optimization objective, that is: 1) maximum Specificity, 2) maximum Sensitivity, 3) minimum Delay, and 4) minimum Energy consumption. In each of the perspectives we highlight the Pareto points that correspond to window sizes S=8,40 and 229.

To represent 4-dimensions in 3-dimensions we take four (4) 3d plots. In Fig. 4.21a we depict the Pareto front of Sensitivity, Specifity and Detection Delay. We find that front is the same as the Pareto front in the filter performance Fig. 4.10a. This is expected, as the Pareto points will increase only by including points that have worse performance in these metrics but better Energy consumption. Thus the optimal points defined by these 3 will not change.

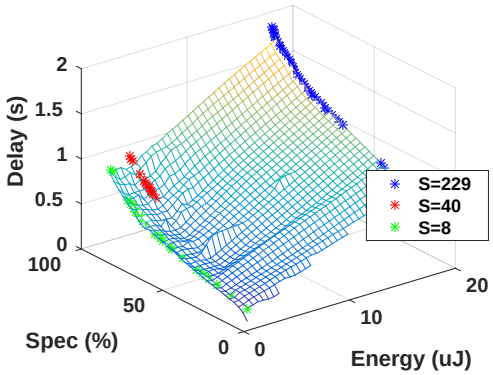
In Fig. 4.21b we depict the Pareto front of Sensitivity, Detection Delay and Energy consumption. We find that for smaller windows sizes we have Pareto points with equal Sensitivity for less Energy consumption and a smaller Detection Delay. This is because, as we have shown in previous sections, window size does not affect Sensitivity, but affects Detection Delay and Energy consumption. Detection delay is increased as larger coefficient window sizes cause a delay in the filter response. Additionally, Energy consumption is increased as larger coefficient window sizes require longer execution time.



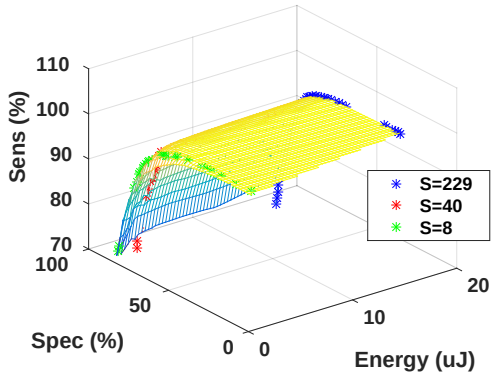
(a) Pareto front: Sensitivity-Specificity-Detection delay



(b) Pareto front: Sensitivity-Energy Consumption-Detection delay



(c) Pareto front: Energy Consumption-Specificity-Detection delay



(d) Pareto front: Sensitivity-Specificity-Energy Consumption

Figure 4.21: Pareto front from 4 objectives (Sensitivity, Specificity, Delay, Energy), highlighted are sample window sizes(S=8,40,229)

In Fig. 4.21c we depict the Pareto front in Specificity, Detection Delay and Energy consumption. We discover that for equal Specificity, there are Pareto points with a smaller Detection delay and smaller Energy consumption for smaller window sizes. As in the case for Sensitivity, Detection Delay and Energy consumption is directly affected by the window size. For Specificity, the Specificity might drop for smaller window size but only insignificantly. For example, the window size S=40 has a Specificity of 89.23% while a window size S=229 has a Specificity of 90.25%.

In Fig. 4.21d we depict the Pareto front in Sensitivity, Specificity and Energy consumption. In this last figure, we observe the same behavior in Sensitivity and Specificity for different windows, as seen in the previous figures where window size marginally impacts these values. Additionally, we see again the high Energy expenditure when selecting a large window size.

4.4.2. BATTERY LIFETIME VS. FILTER PERFORMANCE

One of the most interesting comparisons is that of the estimated battery lifetime and filter performance in relation to coefficient window size, as we can evaluate the trade-offs between the two values.

In Fig. 4.22 we illustrate how different coefficient window sizes affect battery lifetime and ADR. From the graph we observe that the maximum ADR performance achieved is 84.25% and for a close to maximum ADR value of 84% battery lifetime is 1.2 weeks (70 coefficients). Decreasing ADR per-

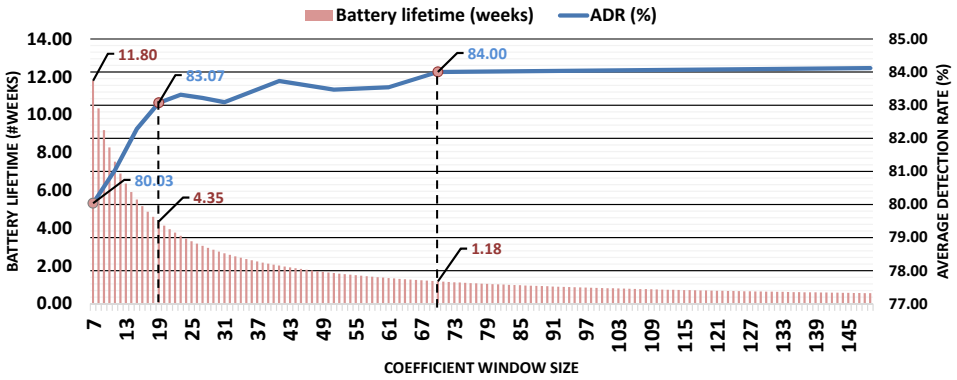


Figure 4.22: Battery lifetime and ADR for different Coefficient Window sizes

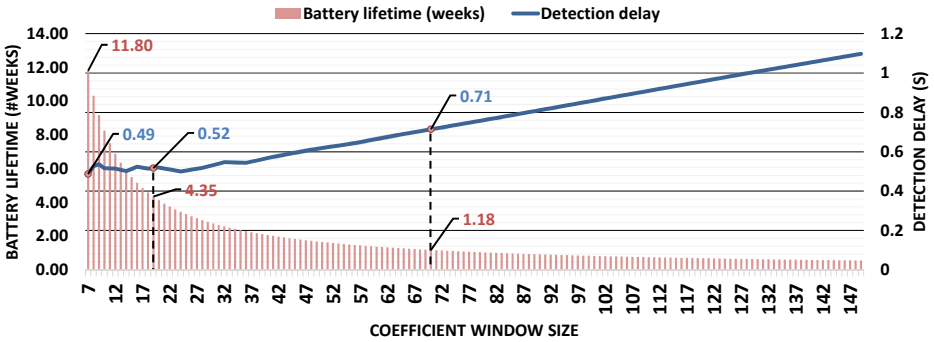


Figure 4.23: Battery and Detection Delay for different Coefficient Window sizes

formance improves battery lifetime significantly as a result of a smaller coefficient window size. More specifically, for decreasing ADR by ~1% (using 19 coefficients) , battery lifetime is extended 4 times (up to 4.4 weeks). Even further, reducing performance to 80% (7 coefficients), the lifespan of the battery reaches 11.8 weeks, increasing duration by a factor of 10.

In Fig. 4.23 we illustrate how different coefficient window sizes affect battery lifetime and Detection Delay. In the figure we find that both values are favored by small window size values, as it reduces execution time and filter's response delay. However while Detection Delay reaches a minimum of values around 500ms, for the Delay battery lifetime can still improve from 4.35 to 11.80 weeks. In comparison with Fig. 4.23 we find that by reducing ADR by 1% we also achieve an improvement in Detection delay of 200ms along with extended battery lifetime (increased by 4 times) .

In general, depending on the levels of tolerance for performance we can trade-off significantly the durability performance of the system. A performance reduction of 1% does not statistically increase by a significant amount the number of un-detected seizures, whereas it yields considerable increased lifespan of a battery powered system.

5

Conclusions and Future Work

In this Chapter we present the conclusion and a summary of this thesis' work. Continuing, we highlight our main scientific contributions and, finally, we propose various directions in which this work could be extended.

5.1. Summary

In this thesis, we presented and evaluated a novel approach in seizure detection and suppression. Our approach uses a complex Morlet wavelet filter with a thresholding mechanism to distinguish seizures from non-seizure intervals, for which we demonstrated a low detection delay, a high detection accuracy, and low computational overheads. Additionally, it was shown that various trade-offs can be made where by reducing detection accuracy the detection delay and the computational overheads are improved. Finally, a prototype has shown the feasibility of detecting and suppressing seizures in living subjects.

In Chapter 2 we presented the characteristics of absence seizures for which we designed the automatic detection system. Then, we listed the specifications of the BeagleBone White which was used as our prototyping platform. Furthermore, we described the SiMS processor architecture, which has served as a platform to quantify our seizure-detection algorithm for ultra-low power architectures. Finally, we discussed related work and implementations on seizure detection and specified the differences between this work and previous ones.

In Chapter 3, we defined the different modules that comprise the proposed detection system. Our work focused on the digital module, where we implemented a Complex Morlet wavelet and a threshold mechanism. We presented a practical FIR filter implementation of the Complex Morlet wavelet which produces high filter output values in seizure intervals. Subsequently, to decompose the input signal in seizure and non-seizure components, we use one of the two thresholding mechanisms (one static with fixed values and one adaptive based on the filter output). Both these mechanisms consist of an upper threshold to denote the detected event and a lower threshold to approximate the end of the seizure. Besides the conceptual model of the system, we presented our specific design choices in order to incorporate the system into a prototype, capable of real-time detection and stimulation triggering. We performed a preliminary analysis where we defined the amount of truncation for the FIR filter that minimizes the detection delay whilst maintaining relative high detection rate. In addition, a prototype implementation of the closed-loop system has successfully been applied in *in-vivo* experiments, demonstrating its potential for epilepsy treatment.

In Chapter 4, we analyzed different aspects of our solution in terms of detection performance

quantified by sensitivity, specificity, and detection delay, and computational overheads quantified by execution time and energy consumption. In our analysis, we studied the effects of different parameters of the system, including the coefficient window size, coefficient window offset and different threshold levels when using a static threshold mechanism. Specifically, we found that lowering the coefficient window size mostly reduces detection delay, while having a minor impact on sensitivity and specificity. In addition, shifting the coefficient window demonstrated again an improvement in detection delay, while causing minimal effects on sensitivity and specificity. For the last parameter, the threshold, we saw that larger values increase sensitivity, but worsen sensitivity and detection delay. Then, we evaluated the detection performance of the adaptive threshold mechanism, which is a function of the filter output based on τ_{up} and τ_{down} values. Specifically, we examined how different τ_{up} and τ_{down} values affect sensitivity, specificity, and detection delay. We found that smaller τ_{up} values favor specificity, but worsen sensitivity and detection delay. As for τ_{down}, we found the opposite behavior where larger values improve specificity and worsen the other two metrics. Using the adaptive threshold mechanism we obtain an improvement of 3% in average detection rate along with a 200ms reduction in detection delay. Regarding the computational overheads, the coefficient window size makes the execution time and energy consumption go up. Effectively, this allows us to trade-off between seizure detection accuracy and computational costs. For instance, we found that by reducing the average detection performance by just 1%, using a smaller coefficient window size, we can obtain an improvement of up to 3.7x times in the life-expectancy of a battery-powered detection system.

5.1.1. THESIS QUESTION

"Can we detect seizures fast and reliably using a seizure-detection system which is suitable for implantable (ultra-low power) application?"

In our evaluation results, we have presented a wide range of possible solutions for the seizure-detection system, which include both low detection delays, high detection rates and in between. For example, a solution that achieves a high sensitivity of 98% has a detection delay of 380ms and a specificity of 87%, and lowering sensitivity (down to 70%) yields a detection delay of 850ms and a specificity of 98%. In our solutions, we have observed that increasing the size of the coefficient window raises detection accuracy at the cost of a higher detection delay and higher energy consumption. In general, we have shown that we can trade-off between detection accuracy, detection delay and energy consumption, where for a small decrease in accuracy we substantial huge improvements in energy consumption and detection delay. As for implantable application suitability, we have proven that execution on a ultra-low power processor is capable of meeting the real-time constraints of the detection-system. Finally, comparing this system with other existing implementations, we saw that we provide improved detection delay (492ms over 970ms), while having similar sensitivity and comparable specificity results. As such, we consider that our solution achieves its purpose to detect seizures in a fast and reliable manner, while respecting the constraints set for an implantable application.

5.2. THESIS CONTRIBUTIONS

The main contributions of this thesis' work are the following:

- *Novel seizure detection and suppression system*
  We have designed a novel seizure detection and suppression system, which uses a complex Morlet Wavelet and a thresholding mechanism. This system was chosen because of morphology resemblance between the Spike-Wave-Discharges and the particular wavelet and because of the low complexity detection mechanism.

- *Prototype realization*
  An actual prototype realization of the detection and suppression system was developed with

cheap and readily available off-the-self components consisting of a custom PCB together with a software implementation on the Beaglebone development platform. In-vivo measurements proved that the system is capable of detecting real-time ictal activity and suppressing an ongoing seizure using an optogenetic stimulator.

- *Full exploration on seizure detection system's performance and complexity*
  The seizure detection system has been fully explored and evaluated in terms of detection performance and computational complexity. Detection performance, which is defined by the sensitivity, specificity, and detection delay, was considered by varying the coefficient window size, coefficient window offset and threshold. Computational complexity was evaluated, for various filter instances, considering the execution costs of the system running on an ultra-low power processor(SiMS).

  - *Implantable suitability*
    We have shown the feasibility of the SiMS implementation to meet the real-time constraints of system and hence its suitability for implantable (ultra-low power) application.
  - *Energy vs. Detection performance trade-off*
    Considering the overall performance of the seizure detection system and its SiMS implementation, we demonstrated several trade-offs between detection accuracy and energy consumption.

5.3. FUTURE WORK

In this thesis we have fully explored a novel solution for a seizure detection system. Both the detection performance and the suitability for Implantable Medical Devices of the solution were considered. Based on these results, we have found certain interesting areas which could be further extended. These are:

- Evaluating other types of Wavelets for seizure detection. The current selection of a Wavelet filter was based on the morphology resemblance, however a different type of Wavelet could produce better filter response values for seizure periods over non-seizure periods. This could lead to better detections and less false positives, hence avoiding unnecessary stimulation. A thorough evaluation could take place comparing many different types from which the optimum is chosen.

- Enhancing adaptive threshold mechanism so that it stops decreasing when a certain minimum is reached. The current mechanism will constantly decrease when no seizure events occur, which will possibly result in an increased number false detections if it gets too low. Therefore, a method to define this minimum value needs to be developed and evaluated.

- Extending SiMS architecture to accommodate the needs of the seizure detection algorithm with dedicated hardware units and evaluate the new energy costs. This includes introducing a multiplier unit to replace the emulation instructions. Additionally, a multiply and accumulate unit can also reduce the number of instructions. This is because the calculation of the FIR filter requires multiplication of input values with the coefficients and then added to a sum. Finally, SIMD (Single Instruction Multiple Data) can also benefit the instructions executed. In general, less executed instructions results in a smaller execution time and hence less energy consumption. However, the trade-offs between execution time and area/power overheads need to be evaluated.

- Developing a custom ASIC(Application-Specific Integrated Circuit) that implements the detection mechanism. Even though ASIC designs are not easily customizable, they are very power

efficient. Thus, by selecting a set parameters from the current evaluation, a digital design can be developed and compared with the implementation on SiMS. The design could be designed to be configurable in its threshold parameters and coefficient values. This implementation could also serve as a co-processor in collaboration with a microcontroller in order to create a complete IMD(implantable medical device). The microcontroller would then only be responsible for configuration, communication and telemetry functions.

A

GUI EPILEPSY

Figure A.1: Host-PC Graphical User Interface to control and control the closed-loop prototype

B

SiMS Instruction Set

name	format	assembly	action
jr	rr	jr rd	branch to addr in rd
and	rr	and rd,rs	rd ← rd and rs
lw	rr	lw rd,rs	rd ← mem[rs]
sw	rr	sw rd,rs	mem[rs] ← rd
mov	rr	mov rd,rs	rd ← rs
not	rr	not rd,rs	rd ← not rs
or	rr	or rd,rs	rd ← rd or rs
xor	rr	xor rd,rs	rd ← rd xor rs
sub	rr	sub rd,rs	rd ← rd-rs
add	rr	add rd,rs	rd ← rd+rs
lb	rr	lb rd,rs	rd ← mem[rs] (byte)
se	rr	se rd,rs	if(rd==rs) then rd← 1 else rd← 0
sgt	rr	sgt rd,rs	if(rd>rs) then rd← 1 else rd← 0(sgn)
sgtu	rr	sgtu rd,rs	if(rd>rs) then rd← 1 else rd← 0(uns)
beqz	rr	beqz rd,rs	if(rd==0) then branch to Imem[rs2]
subi	ri	subi rd,imm	rd←rd - imm
addi	ri	addi rd,imm	rd←rd + imm
li	ri	li rd,rs	rd ←imm
sftl	ri	sftl rd,rs	rd←rd < < imm
sftr	ri	sftr rd,rs	rd←rd > > imm(sign extension)
sftru	ri	sftru rd,rs	rd←rd > > imm(zero extension)
cb	rrr	cb rd,rs1,rs2	exchanges the rs2th - byte of rd by the LSB of rs1
j	jump	j imm	branch to Imem[imm]
jal	jump	jal imm	branch to Imem[imm] and r15 ← PC+1

Table B.1: SiMS ISA

Baseline Formats																
Formats:																
rrr_type	opcode				rd				rs1				rs2			
rr_type	opcode				rd				rs				funct			
ri_type	opcode				rd				imm							
jump_type	opcode				imm											
nop	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
rr_type:																
beqz	0	0	0	1	rd				rs				0	0	1	1
sgtu	0	0	0	1	rd				rs				1	1	1	1
sgt	0	0	0	1	rd				rs				1	1	1	0
se	0	0	0	1	rd				rs				1	1	0	1
lb	0	0	0	1	rd				rs				1	0	1	1
jr	0	0	0	1	rd				rs				1	0	1	0
and	0	0	0	1	rd				rs				0	0	1	0
lw	0	0	0	1	rd				rs				1	0	0	0
mov	0	0	0	1	rd				rs				0	1	1	1
not	0	0	0	1	rd				rs				0	1	0	0
or	0	0	0	1	rd				rs				0	1	0	1
sw	0	0	0	1	rd				rs				1	0	0	1
xor	0	0	0	1	rd				rs				0	1	1	0
sub	0	0	0	1	rd				rs				0	0	0	1
add	0	0	0	1	rd				rs				0	0	0	0
ri_type:																
subi	0	1	1	1	rd				imm							
addi	0	1	1	0	rd				imm							
li	0	1	0	0	rd				imm							
sftru	0	1	0	1	rd				imm							
sftl	0	0	1	1	rd				imm							
sftr	0	0	1	0	rd				imm							
rrr_type:																
cb	1	1	0	1	rd				rs1				rs2			
jump_type:																
jal	1	0	0	1	imm											
j	1	0	0	0	imm											

Table B.2: SiMS Instruction Formats

C

DEVELOPMENT PLATFORM SELECTION

In this appendix, first we define the criteria set for selecting an appropriate development platform to create the prototype presented in Chapter3, after which we present the various platforms that were considered. The prototype realizes the closed-loop seizure detection and suppression mechanism and will serve as proof of the mechanism's correct functionality. Additionally, in this work we also considered the development of a prototype for tinnitus treatment(briefly desribed in Appendix D), for which the the same platform was also used.

In order to select an appropriate platform for both prototypes, we have defined a number of requirements.

Perfomance - The performance of the platform must be greater of equal than that of ARDUINO Due (84Mhz)[56]. This requirement is based on the tinnitus treatment project(Appendix D) were older ARDUINO models were shown to be inadequate.

ADC inputs - In order to record various EEG/ECoG input signals from multiple, the platform should provide at least 4 ADC inputs, one for each of the 4 channels of the seizure closed-loop system. The ADC inputs should be able to sample data at a frequency of 1KHz and with a resolution of 8 bits. Although most brain signal information is found within the frequencies 1-50 Hz[44], a larger frequency range is defined to accommodate future implementations where we need to look into higher ranger that 1-50Hz. As for the ADC resolution, this is because in our analysis in Section 3.2.2 we concluded to a maximum value of 8bit for the inputs.

DAC outputs - The board should provide a minimum of 2 DAC outputs to be able to drive stimulation electrodes, one for the anode and one for the cathode. The resolution of the DAC should be 8-bit and have a sample frequency of 1KHz. This requirement is defined to accommodate future implementations where electrical stimulation is used. To date, most existing electrical stimulation implementations deliver pulses of frequencies up to 333Hz [18].

Local data storage - The platform should have non-volatile memory (i.e SD-card or flash memory) in order to be able to store EEG data for offline inspection on the efficacy of the implementation by physicians and researchers. In addition, various configurations settings should be able to be stored, such stimulation patterns for tinnitus treatment or coefficient values for the closed-loop seizure detection mechanism.

Wireless communication - Bluetooth (or ad-hoc wifi) support for future connectivity should be possible in order to send telemetry data or be able to control via smartphones, tablets etc.

Model	BeagleBone Black[57]	BeagleBone White[23]	OLIMEX STM32-H407[58]	Zedboard[59]	Arduino DUE[56]
Performance	Y	Y	Y	Y	Y
ADC inputs	Y	Y	Y	Y	Y
DAC outputs	N	N	Y	N	Y
Data storage	Y	Y	Y	Y	N
Wireless	N	N	N	N	N
UART	Y	Y	Y	Y	Y
Bare-metal	Y	Y	Y	Y	Y
Expansion	Y	Y	Y	Y	Y
Support	Y	Y	Y	Y	Y
Small size	Y	Y	Y	N	Y
Power modes	Y	Y	Y	Y	N

Y=Requirement is met, N=Requirement not met.

Table C.1: Candidate Development Boards

UART ports - 2 UART ports should be available on the platform, one for serial communication ports necessary for debugging and controlling the device, and one for future extensions, such as an external Bluetooth module if not available on board.

Bare-metal - The platform should be able to run without using an Operating System (OS). Thus the platform should provide drivers for its peripherals. The main reasons bare-metal is required are: 1)Easier to port on platforms without an OS; 2) no power overhead due to running OS and additional processes; 3) Inherently real-time as a single program is executed on the processor; and 4) no extra overhead from OS abstraction layers.

Expansion - A wide range of modules that extend the functionality of the platform should be available to minimize effort of implementing new features. Such modules should provide additional peripherals such as memories, DACs, ADCs, wireless connection, port expandability, LED display, sensors, actuators etc.

Support - The platform should have sufficient development support, either through a active manufacturer's forum or an Open community. This is required to minimize development time.

Small size - The size of the platform should be relatively small so that a portable system can be realized in the case where it is experimentally deployed on patients. Therefore, we require that it has the size of a credit-card which is small enough to be carried around.

Power operation modes - The CPU should have power operation modes such as Dynamic Frequency Voltage Scaling. This can enable us to explore energy aware features for our solution.

Based on the aforementioned requirements a number of candidates were considered. In the Table.C.1 we present the most prominent candidates and which of the requirements they meet.

D

TINNNITUS TREATMENT PROTOTYPE

This appendix provides a brief overview on the realized tinnitus treatment prototype and its deployment setup, which is also described in [60]. The prototype device is capable of producing arbitrary waveform stimuluation pulses (see example Fig. D.4) and can synchronize its stimulation with an audio signal, as depicted in Fig. D.1. In Fig .D.2 we present the interface that controls the device and generates audio signals. In Fig. D.3 we depict the software flow chart of the stimulation kernel that produces the waveforms, its components are:

INIT	Initialize hardware peripherals
PLAY	Updates voltage value to produce waveform. In *charge phase* it updates the voltage until complete pulse is played, in *cancellation phase* it updates voltage untill tissue is discharged or complete pulse is played (whichever comes first)
ISI	Inter-pulse delay, time between charge and cancellation phase
FIN	Finalize operation on hardware peripherals
DEL	Appropriate delay between states

In Fig. D.1 we present the profiling statistics of the software implementation for different waveform resolutions (number of voltage updates). The software implementation is presented in Listing D.1.

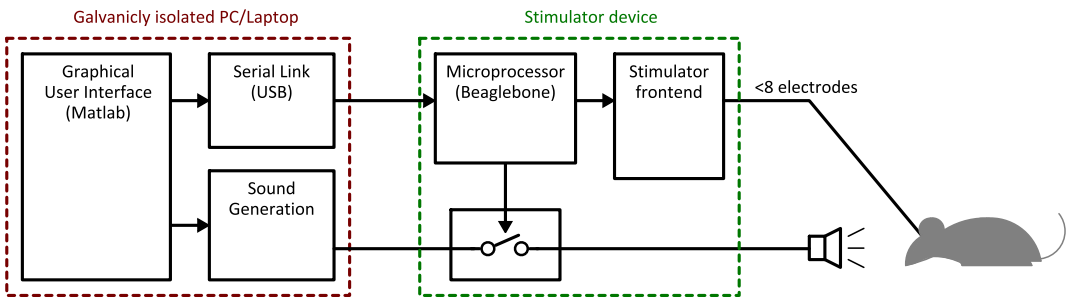


Figure D.1: Overview of the system topology used for the experiments: a computer is using MATLAB to generate the sound stimulus and to provide the user with a GUI. The computer subsequently connects to the stimulation device which synchronizes the electrical stimulation with the audio signal, which are both delivered to the subject.

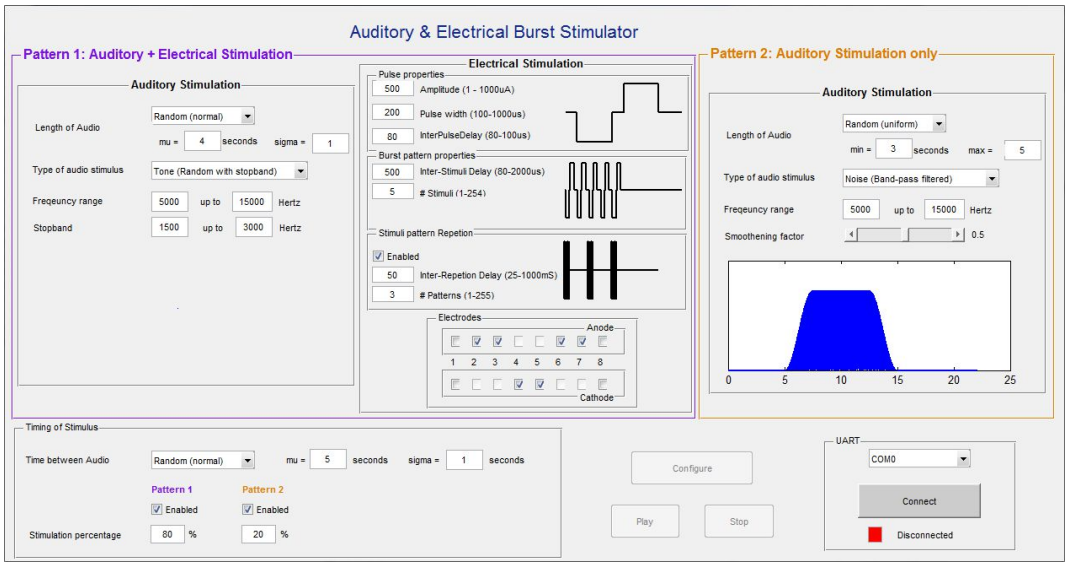


Figure D.2: Graphical User Interface used to control the synchronized audio/electrical stimulation system.

Waveform update-times	1	2	4	8	16	32	64
Instructions committed	1896.37	2378.37	3342.37	5270.37	9126.37	16838.37	32262.37
ALU	1523.45	1896.45	2642.45	4134.45	7118.45	13086.45	25022.45
MEM	134.97	179.97	269.97	449.97	809.97	1529.97	2969.97
BRA	54.99	68.99	96.99	152.99	264.99	488.99	936.99
JUMP	182.96	232.96	332.96	532.96	932.96	1732.96	3332.96
Execution cycles	4635.3	5827.3	8211.3	12979.3	22515.3	41587.3	79731.3

Table D.1: Profiling statistics of software implementation stimulation kernel on SiMS processor[1] with respect to different waveform resolutions (Waveform update-times)

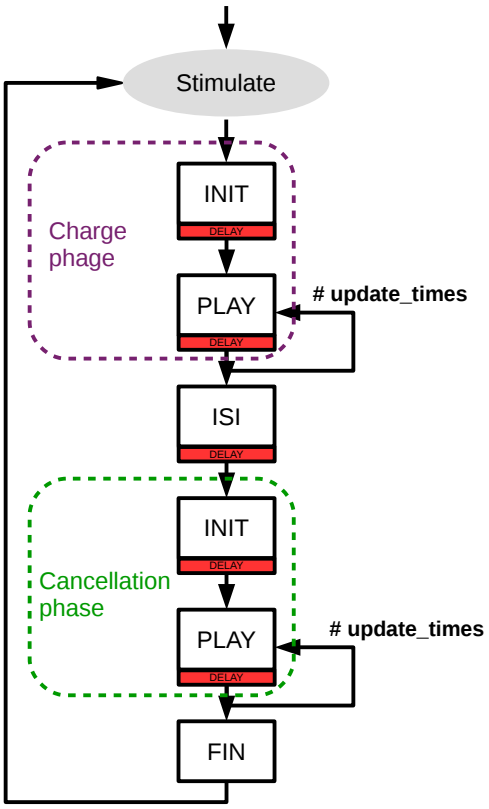


Figure D.3: Flow chart of the software implementation of the stimulation kernel

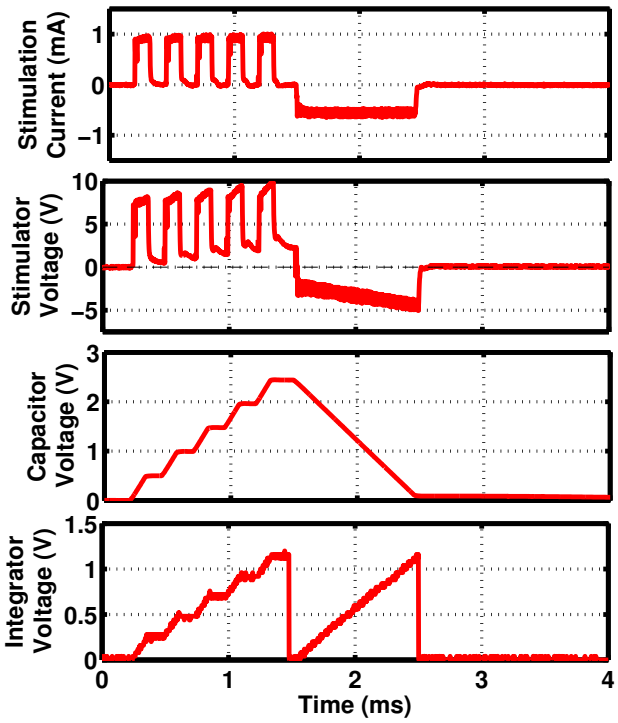


Figure D.4: Sample measurements of arbitrary waveform generator

```
void stimulate() {
  int done = 0;
  stim_st_t STIM_status = INIT;
  uint16_t playSample=0;
  volatile uint16_t SAMPLE_PERIOD=0;
  volatile uint16_t INIT_SAMPLE=0;
  uint32_t isiTime;

  // inter-Pulse Delay
  isiTime = uStim_getISI();

  // Initiliaze values
  INIT_SAMPLE = Waves_getSamplePeriod();
  if (INIT_SAMPLE < PLAY_OVERHEAD+DELAY_OVERHEAD) {
      SAMPLE_PERIOD = 0;
  } else {
      SAMPLE_PERIOD = INIT_SAMPLE - (PLAY_OVERHEAD + DELAY_OVERHEAD);
  }

  //Initialize H/W to 1st phase
  uStim_setPhase(CHARGE);

  while (done != 1) {
      switch (STIM_status) {

      case INIT:
      //=====================================================================

        // Get voltage value and send it to the DAC
```

```
30          playSample = Waves_nxtSample();
31          uStim_setPlayVoltage(playSample);
32
33          // Check state and set voltages accordingly
34          if (uStim_getPhase() == CHARGE) {
35            // [Charge phase] let stimulation play pulse in its whole
36            // set VREF as high as possible
37            uStim_setVref(VDD);
38            uStim_setChargeDIR();
39          } else {
40            // [Cancel phase] play pulse till voltage reaches Vcap
41            // set VREF to voltage at end of charge pulse
42            uStim_setVref( uStim_getVcap() );
43            uStim_setCancelationDIR();
44          }
45
46          // H/W operations
47          uStim_enableSTIM();
48
49          // Delay time
50          delayMicros(INIT_SAMPLE);
51
52          STIM_status = PLAY; //FSM next state
53
54          break;
55
56        case PLAY:
57        //=====================================================================
58        // In this state the consecutive values
59        // of the wave are loaded and played
60
61          if (uStim_getPhase() == CHARGE && Waves_completed() ) {
62            // ----------------------------------
63            // Check if play wave(pulse) is completed in charge phase
64            // ISI state
65            // ----------------------------------
66
67            // H/W operations
68            uStim_disableSTIM();
69            uStim_readVcap();
70            uStim_isolateTissue();
71            uStim_rstCAPS();
72            uStim_setPhase(CANCELATION);
73
74            STIM_status = INIT; // FSM next state
75
76            // ISI delay
77            delayMicros( isiTime );
78
79          } else if (uStim_getPhase() == CANCELATION && (uStim_chrgThreshReach() ||
80                Waves_completedXTimes())
81              ) {
82            // ----------------------------------
83            // Check if phase reached Voltage Threshold or
84            // wave(pulsed) has iterated for X times
85            // FINISH state
86            // ----------------------------------
87
88            // Used to denote the start period
89            uStim_setStimStartRestTm();
90
91            // H/W operations
```

```
92            uStim_shortTissue();
93            uStim_disableSTIM();
94            uStim_rstCAPS();
95            uStim_setVref(0);
96            uStim_setPlayVoltage(0);
97
98            STIM_status = INIT; // FSM next state
99
100           //Exit stimulation loop
101           done = 1;
102
103         } else {
104           // ---------------------------------
105           // Voltage of wave(pulse) is played
106           // ---------------------------------
107
108           uStim_setPlayVoltage( Waves_nxtSample() );
109           if( uStim_getPhase() == CHARGE ){
110             // Delay for update period
111             delayMicros(SAMPLE_PERIOD);
112           }
113           else{
114             // Just in case check Thres PIN
115             // if voltage has been reached
116             delayMicrosPIN(SAMPLE_PERIOD, THRESH_PIN);
117           }
118         }
119         break;
120
121       default:
122       //====================================================================
123       //reset STIM
124         uStim_setVref(0);
125         uStim_isolateTissue();
126         uStim_setPlayVoltage(0);
127         done = 1;
128         break;
129
130       }
131   }
132 }
```

Listing D.1: Software implementation of stimulation kernel

BIBLIOGRAPHY

[1] R. M. Seepers, C. Strydis, and G. N. Gaydadjiev, *Architecture-level fault-tolerance for biomedical implants,* in *Embedded Computer Systems (SAMOS), 2012 International Conference on* (IEEE, 2012) pp. 104–112.

[2] R. S. Fisher, W. v. E. Boas, W. Blume, C. Elger, P. Genton, P. Lee, and J. Engel, *Epileptic seizures and epilepsy: Definitions proposed by the international league against epilepsy (ILAE) and the international bureau for epilepsy (IBE),* Epilepsia **46**, 470 (2005).

[3] C. P. Panayiotopoulos, *Typical absence seizures and related epileptic syndromes: assessment of current state and directions for future research,* Epilepsia **49**, 2131 (2008).

[4] J. R. Tenney and T. A. Glauser, *The current state of absence epilepsy: can we have your attention?* Epilepsy Currents **13**, 135 (2013).

[5] L. M. Frank, T. Enlow, G. L. Holmes, P. Manasco, S. Concannon, C. Chen, G. Womble, and E. J. Casale, *Lamictal (lamotrigine) monotherapy for typical absence seizures in children,* Epilepsia **40**, 973 (1999).

[6] A. Abbott, *Novartis to shut brain research facility,* Nature **480**, 161 (2011).

[7] J. Engel, S. Wiebe, J. French, M. Sperling, P. Williamson, D. Spencer, R. Gumnit, C. Zahn, E. Westbrook, and B. Enos, *Practice parameter: Temporal lobe and localized neocortical resections for epilepsy report of the quality standards subcommittee of the american academy of neurology, in association with the american epilepsy society and the american association of neurological surgeons,* Neurology **60**, 538 (2003).

[8] D. J. Englot, E. F. Chang, and K. I. Auguste, *Vagus nerve stimulation for epilepsy: a meta-analysis of efficacy and predictors of response: a review,* Journal of neurosurgery **115**, 1248 (2011).

[9] M. Sprengers, K. Vonck, E. Carrette, A. G. Marson, and P. Boon, *Deep brain and cortical stimulation for epilepsy,* The Cochrane Library (2014).

[10] A. Berényi, M. Belluscio, D. Mao, and G. Buzsáki, *Closed-loop control of epilepsy by transcranial electrical stimulation,* Science **337**, 735 (2012).

[11] J. T. Paz, T. J. Davidson, E. S. Frechette, B. Delord, I. Parada, K. Peng, K. Deisseroth, and J. R. Huguenard, *Closed-loop optogenetic control of thalamus as a tool for interrupting seizures after cortical injury,* Nature neuroscience **16**, 64 (2013).

[12] E. E. Fanselow, A. P. Reid, and M. A. Nicolelis, *Reduction of pentylenetetrazole-induced seizure activity in awake rats by seizure-triggered trigeminal nerve stimulation,* The Journal of Neuroscience **20**, 8160 (2000).

[13] P. Buteneers, D. Verstraeten, B. Van Nieuwenhuyse, D. Stroobandt, R. Raedt, K. Vonck, P. Boon, and B. Schrauwen, *Real-time detection of epileptic seizures in animal models using reservoir computing,* Epilepsy research **103**, 124 (2013).

[14] M. T. Salam, M. Sawan, and D. K. Nguyen, *A novel low-power-implantable epileptic seizure-onset detector,* Biomedical Circuits and Systems, IEEE Transactions on **5**, 568 (2011).

[15] L. Kros, O. H. Eelkman Rooda, J. K. Spanke, P. Alva, M. N. Dongen, A. Karapatis, E. A. Tolner, C. Strydis, N. Davey, B. H. Winkelman, *et al.*, *Cerebellar output controls generalized spike-and-wave discharge occurrence,* Annals of neurology **77**, 1027 (2015).

[16] G. K. Bergey, *Neurostimulation in the treatment of epilepsy,* Experimental neurology **244**, 87 (2013).

[17] J. Tønnesen, A. T. Sørensen, K. Deisseroth, C. Lundberg, and M. Kokaia, *Optogenetic control of epileptiform activity,* Proceedings of the National Academy of Sciences **106**, 12162 (2009).

[18] F. T. Sun and M. J. Morrell, *Closed-loop neurostimulation: the clinical experience,* Neurotherapeutics **11**, 553 (2014).

[19] W. Agnew, D. McCreery, T. Yuen, and L. Bullara, *Local anaesthetic block protects against electrically-induced damage in peripheral nerve,* Journal of biomedical engineering **12**, 301 (1990).

[20] V. G. Udo Jonas, *New Perspectives in Sacral Nerve Stimulation: For Control of Lower Urinary Tract Dysfunction* (Martin Dunitz Ltd., 2002).

[21] M. N. Van Dongen, *A versatile output stage for implantable neural stimulators*, Master's thesis, Delft University of Technology (2009).

[22] J.-P. Yao, W.-S. Hou, and Z.-Q. Yin, *Optogenetics: a novel optical manipulation tool for medical investigation,* International journal of ophthalmology **5**, 517 (2012).

[23] *Beaglebone white: Microprocessor low-cost community-supported development platform based on am335x 720mhz (arm cortex-a8),* http://beagleboard.org/bone, accessed: 01-07-2015.

[24] *Texas instruments,* http://www.ti.com/, accessed: 01-07-2015.

[25] *ARM cortex-a8,* http://www.arm.com/products/processors/cortex-a/cortex-a8.php (), accessed: 01-07-2015.

[26] *Starterware,* http://www.ti.com/tool/starterware-sitara, accessed: 01-07-2015.

[27] *SINs (smart implantable neurostimulators) project,* http://braininnovations.nl/engels.php, accessed: 01-07-2015.

[28] H. P. Zaveri, W. J. Williams, and J. C. Sackellares, *Energy based detection of seizures,* in *Engineering in Medicine and Biology Society, 1993. Proceedings of the 15th Annual International Conference of the IEEE* (IEEE, 1993) pp. 363–364.

[29] J. F. Kaiser, *On a simple algorithm to calculate theenergy'of a signal,* in *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on* (1990) pp. 381–384.

[30] H. Qu and J. Gotman, *A seizure warning system for long-term epilepsy monitoring,* Neurology **45**, 2250 (1995).

[31] R. Esteller, J. Echauz, T. Tcheng, B. Litt, and B. Pless, *Line length: an efficient feature for seizure onset detection,* in *Engineering in Medicine and Biology Society, 2001. Proceedings of the 23rd Annual International Conference of the IEEE*, Vol. 2 (IEEE, 2001) pp. 1707–1710.

[32] S. J. Korn, J. L. Giacchino, N. L. Chamberlin, and R. Dingledine, *Epileptiform burst activity induced by potassium in the hippocampus and its regulation by gaba-mediated inhibition,* Journal of Neurophysiology **57**, 325 (1987).

[33] A. M. White, P. A. Williams, D. J. Ferraro, S. Clark, S. D. Kadam, F. E. Dudek, and K. J. Staley, *Efficient unsupervised algorithms for the detection of seizures in continuous eeg recordings from rats after brain injury,* Journal of neuroscience methods **152**, 255 (2006).

[34] F. Westerhuis, W. Van Schaijk, and G. Van Luijtelaar, *Automatic detection of spike-wave discharges in the cortical eeg of rats,* in *Measuring behavior*, Vol. 96 (Noldus Information Technology Wageningen, 1996) pp. 109–110.

[35] P. Van Hese, J.-P. Martens, L. Waterschoot, P. Boon, and I. Lemahieu, *Automatic detection of spike and wave discharges in the eeg of genetic absence epilepsy rats from strasbourg,* Biomedical Engineering, IEEE Transactions on **56**, 706 (2009).

[36] I. Osorio, M. G. Frei, and S. B. Wilkinson, *Real-time automated detection and quantitative analysis of seizures and short-term prediction of clinical onset,* Epilepsia **39**, 615 (1998).

[37] A. Berdakh and S. H. Don, *Epileptic seizures detection using continuous time wavelet based artificial neural networks,* in *Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on* (IEEE, 2009) pp. 1456–1461.

[38] *Rp2.1, tucker-davis technologies,* http://www.tdt.com/rp2.1-real-time-processor.html, accessed: 01-07-2015.

[39] S. Raghunathan, S. K. Gupta, M. P. Ward, R. M. Worth, K. Roy, and P. P. Irazoqui, *The design and hardware implementation of a low-power real-time seizure detection algorithm,* Journal of neural engineering **6**, 056005 (2009).

[40] H. Markandeya, G. Karakonstantis, S. Raghunathan, P. Irazoqui, and K. Roy, *Low-power dwt-based quasi-averaging algorithm and architecture for epileptic seizure detection,* in *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design* (ACM, 2010) pp. 301–306.

[41] H. S. Markandeya, S. Raghunathan, P. P. Irazoqui, and K. Roy, *A low-power near-threshold epileptic seizure detection processor with multiple algorithm programmability,* in *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design* (ACM, 2012) pp. 285–290.

[42] F. T. Sun, M. J. Morrell, and R. E. Wharen, *Responsive cortical stimulation for the treatment of epilepsy,* Neurotherapeutics **5**, 68 (2008).

[43] M. J. Morrell, *Responsive cortical stimulation for the treatment of medically intractable partial epilepsy,* Neurology **77**, 1295 (2011).

[44] W. O. Tatum, *Ellen r. grass lecture: Extraordinary eeg,* The Neurodiagnostic Journal **54**, 3 (2014).

[45] M. van Dongen, A. Karapatis, L. Kros, O. Eelkman Rooda, R. Seepers, C. Strydis, C. De Zeeuw, F. Hoebeek, and W. Serdijn, *An implementation of a wavelet-based seizure detection filter suitable for realtime closed-loop epileptic seizure suppression,* in *Biomedical Circuits and Systems Conference (BioCAS), 2014 IEEE* (IEEE, 2014) pp. 504–507.

[46] P. Abry, *Multirésolutions, algorithmes de décomposition, invariance d'échelles, diderot edition,* Ondelettes et turbulence. Paris.[Links] (1997).

[47] J. M. Karel, S. A. Haddad, S. Hiseni, R. L. Westra, W. Serdijn, R. L. Peeters, *et al.*, *Implementing wavelets in continuous-time analog circuits with dynamic range optimization,* Circuits and Systems I: Regular Papers, IEEE Transactions on **59**, 229 (2012).

[48] *Molecular devices llc., axon instruments, sunnyvale, ca, usa,* .

[49] *Thor labs (newton, nj, usa).* .

[50] *Arm cortex-m processor family,* http://www.arm.com/products/processors/cortex-m/ (), accessed: 01-07-2015.

[51] *Advent research materials, eynsham, oxford, uk,* .

[52] *Matlab,* http://mathworks.com/products/matlab/, accessed: 01-07-2015.

[53] R. M. Seepers, *Architecture-Level FaultTolerance Techniques for Biomedical Implants*, Master's thesis, Delft University of Technology (2011).

[54] *Modelsim,* http://www.mentor.com/products/fv/modelsim/, accessed: 01-07-2015.

[55] *Cr2032 battery, capacity 225mah, 3.0v,* http://cr2032.co/.

[56] *Arduino due board: Microcontroller low-cost community-supported development-board based on atmel sam3x8e (arm cortex-m3 cpu),* http://www.arduino.cc/en/Main/ArduinoBoardDue, accessed: 01-07-2015.

[57] *Beaglebone black: Microprocessor low-cost community-supported development platform based on am335x 1ghz (arm cortex-a8),* http://beagleboard.org/black, accessed: 01-07-2015.

[58] *STM32-H407: Microcontroller low-cost development board based on stm32f407zgt6 cortex-m4 microcontroller,* https://www.olimex.com/Products/ARM/ST/STM32-H407/open-source-hardware, accessed: 01-07-2015.

[59] *Zedboard: Microprocessor and fpga hybrid development board based on xilinx zynq-7000 soc,* http://zedboard.org/product/zedboard, accessed: 01-07-2015.

[60] M. N. Van Dongen, *Design of efficient and safe neural stimulators: a multidisciplinary approach*, Ph.D. thesis, Electrical Engineering, Mathematics and Computer Science (2015).