IMPLANTABLE SEIZURE DETECTOR & PREDICTOR

By

Ahmed Yasser Abo El-Makarem

Al Moataz Bellah Refaat Fouad

Taha Shawky Kamel

Kareem Ayman Mohamed

Mohamed Mahmoud Kamal El-Din

Mohamed Moustafa Abd El-Rahman

Under the Supervision of

Dr. Hassan Mostafa

A Graduation Project Report Submitted to
the Faculty of Engineering at Cairo University
In Partial Fulfillment of the Requirements for the
Degree of Bachelor of Science
in
Electronics and Communications Engineering

Faculty of Engineering, Cairo University

Giza, Egypt

July 2016

Table of Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

Please insert here any symbols and abbreviations that you commonly use in your report.

Example:

iEEG	Intracranial Electroencephalography
AEDs	Anti-Epileptic Drugs
ASIC	Application-Specific Integrated Circuit
SVM	Support Vector Machine
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
DADS	Detection Algorithm Design Space
FPGA	Field Programmable Gate Array

Acknowledgments

First, we are truly grateful to our great supervisor, Dr. Hassan Mostafa, who supported us a lot throughout the project course starting from proposing the idea, going through the medical background we needed in our research till reaching the achievements mentioned in this report that would have never been reached without his help and continuous support. He never failed to motivate us and set new challenges for each stage in the project to make sure we are on the right track.

We would like to thank the team of Cairo university research assistants who assisted us throughout the project specially Eng. George Maximus who helped us a lot in the early stages of the research.

We would like also to thank our colleagues and friends for their support and belief in what we are doing since the very first beginning till presenting our work.

Very warm thanks to our parents and extended families that raised us that way and exerted efforts for years doing so; we hope they are proud of their sons by now.

Abstract

Implantable Seizure Detector & Predictor

Neural disorders such as epilepsy are caused by malfunctioning nerve cell activity in the brain. These malfunctions cause episodes called seizures. Epilepsy hits more than 65 million people worldwide; nearly 80% of cases occur in developing countries. Over a lifetime, 1 in 26 people will be diagnosed with epilepsy. Seizures can cause a range of symptoms from momentarily staring blankly to loss of consciousness and uncontrollable twitching. Some seizures can be milder than others, but even minor seizures can be dangerous if they occur during activities like swimming or even driving. Epilepsy is not fully cured yet. In some cases, seizures do not respond to medication, also patients become drug resistive after five years maximum. Thus, neuro-stimulation should be considered.

Hereby, this project is addressing helping these people to live normal life again by predicting & detecting the seizures to overcome their effect.

The project outcome will be simulations of these prediction & detection algorithms on Matlab. Following that, a complete synthesizable behavioral design written with VHDL will be tested on FPGA Spartan-6 kit. The next phase should be a complete ASIC (Application Specific Integrated Circuit) design that will be conducted to produce a test chip that contains the implemented algorithms.

Chapter 1: Introduction

Epilepsy is a neurological disorder of the brain that more than 65 million people globally suffer from. According to the World Health Organization (WHO), epilepsy is characterized by recurrent seizures, which are physical reactions to sudden, usually brief, excessive electrical discharges in a group of brain cells. Uncontrolled attacks can put patients at risk of suffering oro-facial trauma.

The data flow diagram of seizure detection system consists of several blocks as shown in Figure 1. Data acquisition block is responsible for passing the input Intracranial Electroencephalography (iEEG) signal with fixed rate to the following blocks. The feature extraction block is responsible for calculating specific features for iEEG input signal on different domains. The classifier block is responsible of monitoring the features to make sure that the condition of the seizure is not valid. When it is valid, the decision making block take the final call as this block gets input from each feature and classifier.

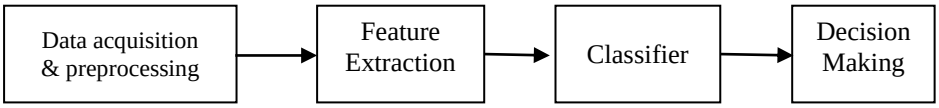


Fig. 1. Data flow diagram of the seizure detection system.

1.1 Historical Background

Thousands of years ago, ancient civilizations were mystified by the epileptic condition. Ancient Greeks thought that one got epilepsy by offending the moon goddess Selene. Ancient Romans believed that epilepsy came from the demons. Ancient Babylonians thought similarly, but that different spirits caused the different types of seizures. All came up with their own explanations as to who or what the cause was because none of them could comprehend it.

In 400 BC, Hippocrates, the "Father of Medicine", wrote *On the Sacred Disease*, where he refuted the idea that epilepsy was a curse or prophetic power: "*It is thus with regard to the disease called Sacred: It appears to me to*

be nowise more divine nor more sacred than other diseases, but has a natural cause like other affections."

The word "epilepsy" is derived from the Greek "*epilepsia*", which means, "to take hold of" or "to seize". The first documented incidence of epilepsy was more than 3,000 years ago in ancient Babylonia, where the condition was referred to as "*miqtu*". In both Ancient Greece and Babylonia, people saw epilepsy as a supernatural, but perhaps holy phenomenon.

Epilepsy has been viewed differently by various cultures. During World War II, the Nazi Eugenics Laws mandated that persons with epilepsy must be sterilized. Epileptics were highly discriminated against and treated very differently than the average person. In fact, this type of discrimination still exists in today's society.

In modern-day Africa, people with epilepsy are finding it difficult to cope with the social consequences of the disorder as well as the exclusion from education and employment. This is further compounded by the fact that a large proportion of people there with epilepsy are not taking Anti-Epileptic Drugs (AEDs). The lack of health services makes it very difficult for people with epilepsy to be a functional part of society.

Discrimination is equally present in western culture as well. Up until 1980, various states in America forbade epileptics to marry. The Americans with Disabilities Act of 1990 made it illegal to discriminate against people with epilepsy in the workplace. Those whose seizures can be effectively controlled are not considered disabled under the act. The ability to predict and control seizure activity would dramatically improve the lifestyle of people with epilepsy, hence the motivation of this research.

1.2 Epilepsy Today

Epilepsy is classified as the second most serious neurological condition known to man after stroke. It affects nearly 65 million people around the world, which is approximately 1% of the world's population. As many as one out of ten people will have a seizure sometime during their lives, but the majority will not have epilepsy as the underlying cause of the convulsions lie outside the brain. Approximately

one in 50 people will have some form of epilepsy at some point in their life. About 75 people are diagnosed with epilepsy every day. Only about 3-5% of people with epilepsy will be affected by flashing lights (photosensitive epilepsy). It's a common misconception that all epileptics are affected by quick visual stimuli; this is actually quite uncommon. Only about two out of 10,000 people in the general population have this condition.

At the present time, it is estimated that from 1.5 to 2 million people in the United States alone have an active form of epilepsy. Today, there are many people living with epilepsy whose lives have been greatly improved by modern science. In fact, many people who have had epilepsy have also been able to have successful careers or become very famous. Alfred Nobel, founder of the Nobel Prize and the inventor of dynamite, had epilepsy. Other famous/well-known people with epilepsy include Fyodor Dostoevsk, Neil Young, George Gerswhin, Philip K. Dick, Napoleon I, and Joan of Arc. These people, and many others, give the word 'disability' a different meaning.

1.3 Epileptic Seizure Prediction

One of the most debilitating aspects of epilepsy is the lack of warning before a seizure occurs. As a result, common day to day tasks such as driving or using a knife become much more hazardous. In some patients, seizures occur hundreds of times a day, but they can also be as infrequent as once every few years. These limitations can have a debilitating effect on quality of life, and hinder basic everyday activities.

The ability to predict seizures could lead the way to novel diagnostic and therapeutic methods for the treatment of patients with epilepsy. Real-time prediction of epileptic state transitions and early onset prediction provide time to administer preventive interventions possibly terminating the seizure before it happens. Also, a lead-time could give epileptic patients enough time to remove themselves from harms away. It would give them the ability to do things that people without epilepsy take for granted as they will already know that they are not facing any seizures for a while, which is the lead-time period.

1.4 Goal: Epileptic Seizure Control

Once a robust seizure prediction algorithm is in place, a device that could somehow control the seizure would be possible. An implanted system that activates a mini-AED delivery system to deliver the medications directly into the epileptogenic focus or activate a stimulator would be feasible. A so-called "brain pacemaker" which appropriately stimulates the vagus nerve at the predicted onset time could also reduce the frequency and ferocity of epileptic seizures. However, if the true prediction rate is too high and the false prediction rate is too low, these medications and therapeutic treatments would be administered too often. This would result in a whole slew of clinical side-effects, most of which would be neuropsychological. A careful balance of all the influential parameters is required for the optimal automated seizure prediction algorithm which would put us on the path to the ultimate goal: Seizure Control.

1.5 Organization

In *Chapter 1*, a brief introduction to the problem is given, and the goals of this thesis are provided.

Chapter 2 provides some information on the epileptic condition as well as some information on iEEG signals in relation to epilepsy. An overview of the time series analysis techniques is given. Also, a literature review of the current progress in the field of seizure detection is presented.

Chapter 3 outlines the proposed system for seizure state prediction. Also presented here are some new methods and algorithms that were created during the course of this research to contribute to the field of seizure prediction. *Chapter 4* discusses the results of the seizure prediction algorithms that were proposed in Chapter 3. The optimized feature set is presented for different types of iEEG signals, and the strength of the dynamic real-time classification system is discussed.

Chapter 5 summarizes the conclusions of this research and provides a discussion about the research methods. Possible ways to improve the results and algorithms are given as possible future work.

Chapter 2: Background

This chapter gives an overview of epilepsy and provides its relation to iEEG signals. The different states of a seizure are provided along with a discussion about the brain waves during each period. In the following section, all of the numerical methods that were implemented in the system are provided.

2.1 The Neurological Condition: Epilepsy

Epilepsy is a serious neurological disorder characterized by recurrent unprovoked seizures due to abnormal or excessive neuronal activity in the brain. This condition is characterized by chronic abnormal bursts of electrical spikes in the brain. The region of seizure generating tissue, known as the *epileptogenic focus*, can be the result of structural abnormalities in the brain which may or may not be genetic. In cases where the cause is known, the epileptic condition is referred to as *symptomatic epilepsy*. In cases where there is no identifiable cause, but by deduction a genetic basis is presumed, the condition is referred to as *idiopathic epilepsy*. Cases that don't fit into either of these two categories are considered to be *cryptogenic epilepsy*.

Epileptic seizures are characterized by uncontrollable movements such as shaking of the arms or legs, known as convulsions. Some may lose consciousness, which may consist of a complete collapse or the patient simply gazing into space. Fainting spells with incontinence, followed by excessive fatigue, is common in more serious types of seizures. Distorted perceptions, odd sounds, and sudden feelings of fear for no apparent reason are characteristic of the "aura", which is felt just prior to the seizure. These "auras" can happen anywhere from a day before the seizure to just a few seconds prior. A select few can predict that they will have a seizure by themselves just by understanding the "aura".

Seizures are usually treated with medications known as Anti-Epileptic Drugs (AEDs). AEDs attempt to stop the occurrence of seizures, but do not serve as a cure for epilepsy. With the correct AEDs, approximately 70% of people with epilepsy could have their seizures controlled or stopped for maximum time of five years. For people who cannot control their seizures with AEDs, epileptic surgery is quite

common. This type of surgery involves the resection of sections of the brain that comprise the epileptogenic focus. It has been shown to greatly reduce the severity of epileptic seizures in patients, or in some cases, stop them completely. Vagus Nerve Stimulation (VNS) is a treatment of epilepsy which attenuates seizure frequency, severity, and duration by chronic intermittent stimulation of the vagus nerve. It is intended for use as an adjunctive treatment with AED medications. The patient is implanted with a VNS therapy system which directly stimulates the vagus nerve at predetermined time intervals.

2.2 Epilepsy & iEEG signals

iEEG is a recording of the electrical activity in the brain at different frequencies. It was first developed in 1924 by Hans Berger, a German psychiatrist, who revealed the practical and diagnostic use of this test. Special sensors are placed strategically around the head that are connected to a machine which records the electrical impulses, either on screen or on paper. Trained neurologists are able to look at the different frequencies in the iEEG and recognize patterns in it which provide information about the epileptic condition.

2.2.1 Brain Waves

Raw iEEG signals are usually described in terms of the four basic brain waves: Alpha [7.5-13] Hz, Beta [13-30] Hz, Delta [0-3.5] Hz, and Theta [3.5-7.5] Hz. These bands represent the most prominent activity in the brain.

Alpha Waves

Alpha waves are comprised of brain signals of the frequency range [7.5-13] Hz. Healthy alpha waves promote mental resourcefulness, aid in the ability to mentally coordinate, and enhance the overall sense of relaxation and fatigue. With healthy alpha waves, one can move quickly and efficiently to accomplish tasks at hand.

Alpha waves appear to bridge the conscious and subconscious mind and are quite prominent in normal relaxed adults. It is present throughout most of an individual’s life, but specially beyond the 13th year when it dominates the resting tracing (normal iEEG tracings).

The most important recorded wave in a normal adult iEEG is the occipital alpha waves, which are best obtained from the back of the head when the subject is resting quietly with the eyes closed, but not asleep. Interference in this band can be caused by opening the eyes or excitement. These waves are blocked by both excitement or by opening the eyes.

Beta Waves

Beta waves are considered "fast" brain activity (above 13Hz). These waves are seen on both sides of the brain in a symmetrical distribution and are most evident in the frontal region of the brain. This band may be absent or reduced in areas with cortical damage. It is generally the normal rhythm in those who are alert, anxious, or who have their eyes open.

Delta Waves

Delta waves are comprised of the lowest frequencies in the brain, specifically from [0-3.5] Hz. They usually occur in the deep sleep state and in some abnormal processes in the brain. These waves are the dominant rhythm in infants up to one year of age and are present in stages III and IV of sleep. The delta band tends to be the highest in amplitude and also the slowest of the waves. Increasing delta waves mean a decrease in our awareness of the physical world, which is a characteristic property of seizure activity. Our unconscious mind is also represented through delta waves. Peak performers decrease their Delta waves when high focus and peak performance are required.

Theta Waves

Theta activity is classified as rhythmic, slow waves from the frequency range of [3.5-7.5] Hz. It has connection with creativity, intuition, learning, daydreaming, fantasizing, and is a repository for memories, emotions, and sensations. The presence of these waves reflects the state between wakefulness and sleep. Theta waves are abnormal in awaken adults, but are perfectly normal in children up to 13 years old and during sleep. Theta waves are also observed during anxiety, pain, behavioral activation, and behavioral inhibition. The appearance of excess Theta waves is an indicator of abnormalities in the brain.

2.2.2 Epileptic States

The different stages of an epileptic seizure are referred to as *ictal* states. These states represent the different stages of an epileptic seizure in its most general sense.

1) Interictal State

The *interictal* state refers to the normal resting state containing no seizure activity, but the iEEG is still characterized by the epileptic condition (irregular neuronal activity). Given the possibility of seizures, the chronic interictal period is interesting because of the presence of natural homeostatic mechanisms that prevent seizure generation. It is unsure which factors or mechanisms try to maintain homeostasis in the brain, and it is also unsure if these mechanisms differ for various types of seizures and epileptic syndromes. This period comprises more than 99% of patients' lives. In this way, the interictal period can be used by neurologists to diagnose an epileptic condition. The iEEG tracings would normally exhibit small spikes and other abnormalities known by neurologists as subclinical seizures. These are not real seizures, but rather little hints from the brain that something is abnormal.

2) Preictal State [Seizure Onset]

The *preictal* state refers to a period of time occurring before a seizure, but does not refer to the normal state of the brain. This state defines that a seizure is going to occur within a certain period of time. The presence of a preictal period is still being debated by several researchers. Lehnertz and Litt state that in some conditions, the transition can take a considerable amount of time, opening the potential for the application of electrophysiological techniques to predict seizure onset anywhere from minutes to hours before occurrence.

Onset of a clinical seizure is defined as the time at which the transition between the interictal and preictal state occurs. It is characterized by a sudden change in the frequency characteristics of the iEEG. The alpha band has a tendency to decrease in frequency and increase in amplitude. The transition from the preictal to the ictal state, for a focal epileptic seizure, consists of a gradual change from chaotic to ordered waveforms. In each type of epilepsy, the transitional period may differ, and the transition can have significantly different characteristics.

3) Ictal State to Seizure Termination

The period of time during which the seizure is in its activation period is referred to as the *ictal* state. It is characterized by an iEEG tracing that exhibits significantly higher amplitudes and frequencies. There is an immediate alteration in synchronization and rhythmicity that takes place across widespread areas of the cerebral cortex. The patterns that are normally seen throughout the resting tracing suddenly become extremely erratic and unpredictable. Loss of consciousness and involuntary muscle twitching during this period is very common. Other symptoms such as incontinence are also common during this state. The patient typically has no control over their body at this point and convulsions tend to be prominent.

4) Postictal State

The end of an epileptic seizure represents a transition from the ictal state back to an individual's normal or interictal state. This is referred to as the *postictal* state and signifies the recovery period of the brain. Focal or generalized neurological deficit, ranging from postictal depression to aphasia or paralysis is prevalent during this state. This period is associated with a difficulty in thinking clearly and a variety of other cognitive defects. The postictal state could last from seconds to hours depending on the severity of the seizure and the efficacy of the AEDs. Disturbances or aftershocks are seen in the iEEG, which may just be the presence of natural mechanisms acting to terminate the seizure and restore homeostasis.

Often postictal deficits are a consequence of the natural mechanisms that act to terminate a seizure suggesting that interventions designed to exploit these same homeostatic events could exacerbate postictal dysfunction.

Attention and concentration is generally very difficult during this period. Poor short term memory and decreased verbal and interactive skills are noticeable. Postictal migraine headaches are very common due to the pressure resulting from cerebral edema. At this point, patients are unaware that they have had a seizure, but these symptoms are evidence enough for an experienced epileptic.

2.3 Time Series Analysis

A time series is an ordered sequence of values of a variable at equally spaced time intervals. Time series analysis is comprised of methods that attempt to identify the nature of a phenomenon represented by a sequence of observations. This requires that patterns of observed time series data are properly identified and mathematically described.

2.3.1 Sampling

Sampling is the operation of transforming a time signal from the continuous to the discrete time domain. Let $x_a(t)$ be a continuous-time signal that is sampled uniformly at $t = nT$, generating the sequence $x[n]$ where

$$x[n] = x_a(nT),\ n = 0, 1, 2, 3, \ldots \qquad (2.1)$$

with T being the *sampling period*. The reciprocal of T is referred to as the sampling frequency fs. In order to reconstruct the original signal completely and accurately, the *Nyquist-Shannon Sampling theorem* must be adhered to. This theorem states that a band-limited signal can be reconstructed perfectly from the discrete sampled signal given that the sampling rate is more than twice the frequency. Therefore, the frequency that is half of the sampling rate fs, also known as the *folding frequency* or the *Nyquist frequency*, is the highest frequency that can be represented in a sampled signal without being affected by aliasing or under-sampling. In equation (2.2), also known as the *Nyquist Condition*, the *Nyquist frequency* is denoted as Ω_m.

$$\Omega_T \geq 2\Omega_m, \textit{ where } \Omega_T = 2\pi/T \qquad (2.2)$$

"*Aliasing*" is the phenomenon in which high-frequency components of a function of time can translate into low frequencies if the sampling rate is too low. Thus, any component in $x(t)$ higher than the folding frequency is aliased (or folded) into the frequencies that are below the folding frequency. Figure 2.1 illustrates the effects of aliasing.

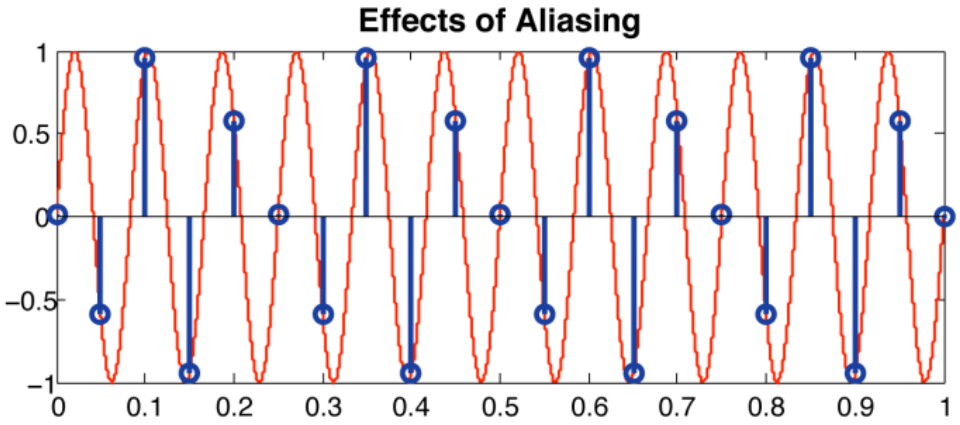


Figure 2.1: Effects of aliasing on a discretely sampled signal.

As seen in Figure 2.1, the reconstructed signal contains frequencies that are lower than those in the original signal due to aliasing. The sampled signal does not satisfy the *Nyquist Condition*.

Brain waves are an example of a continuous time signal which, when sampled with an iEEG machine, becomes a discrete time series in the form of equation (2.1). In general, when any signal is discretely sampled, 'information' is lost from the resulting time series. The frequency of the sampled signal must be high enough to preserve the frequency information from within the signal. The problem with iEEG signals is that it is difficult to tell what the appropriate sampling frequency is because of its unpredictable nature. According to Castellaro, most of the prominent frequencies in iEEG signals lie between 0 and 46Hz. This means that the lowest feasible sampling rate would be *fs* = 92Hz. To satisfy the *Nyquist condition*, all iEEG signals used in testing the algorithms introduced in this paper were collected at a minimum of 250Hz –average of 256 Hz-.

2.3.2 Windowing

A window function is a function that defines a signal within a defined interval and is zero everywhere outside. It is multiplied by the original signal to give a clearer perspective of the signal in the frequency domain. The most basic window function is the *Rectangular Window*, also known as the *Dirichlet Window*, stated as:

$$w_R[n] = \begin{cases} 1, & -M \geq n \geq M, \\ 0, & \text{otherwise.} \end{cases} \tag{2.3}$$

iEEG signals cannot be classified at every discrete point collected in an iEEG signal. There is just not enough information at each time point. The system must be given a window of the signal with enough information to make a sensible judgement about the state of the signal at each moment in time. The length of the window must also be at least the Nyquist rate or else it will not be able to recognize the relevant frequencies that are present. But it is also unclear how much of the signal is necessary to give the system enough information to properly identify classes within the signal. iEEG signals tend to be arbitrary by nature, and with some epileptic states, the frequency component of the signal can vary with time depending on the severity of the condition. During the ictal period, the frequency components of an iEEG signal become extremely erratic and unpredictable. Due to this irregularity, a rectangular window is more appropriate for frequency analysis of these signals.

Chapter 3: Seizure Detection Features

In this chapter, we are discussing the different features that can be used to detect an epileptic seizure. The features mentioned in this chapter belong either to time domain or wavelet domain.

Detection algorithms employ mathematical operations on recorded data to demarcate them into baseline and non-baseline (seizure) states. These mathematical operators maybe linear or non-linear features extracted from the raw data that help enhance the boundary of demarcation.

3.1 Time Domain Features

3.1.1 Energy

This feature reflects the instantaneous power of the signal. A sliding rectangular window is used to calculate the instantaneous energy as described in equations (3.1) and (3.2).

$$E[i] = x^2(i) \tag{3.1}$$

$$E_{avg}[k] = \frac{1}{N}\sum_{i=1}^{N} E(i + (k-1) \times N) \tag{3.2}$$

In equations (3.1) and (3.2) and from this point forward, x(i) refers to the i^{th} sample of data given by the vector ‘x’ ,and ‘k’ represents the window number of size ‘N’. An average of the instantaneous energy is obtained using a rectangular window to sum the energies as shown in equation (3.2).

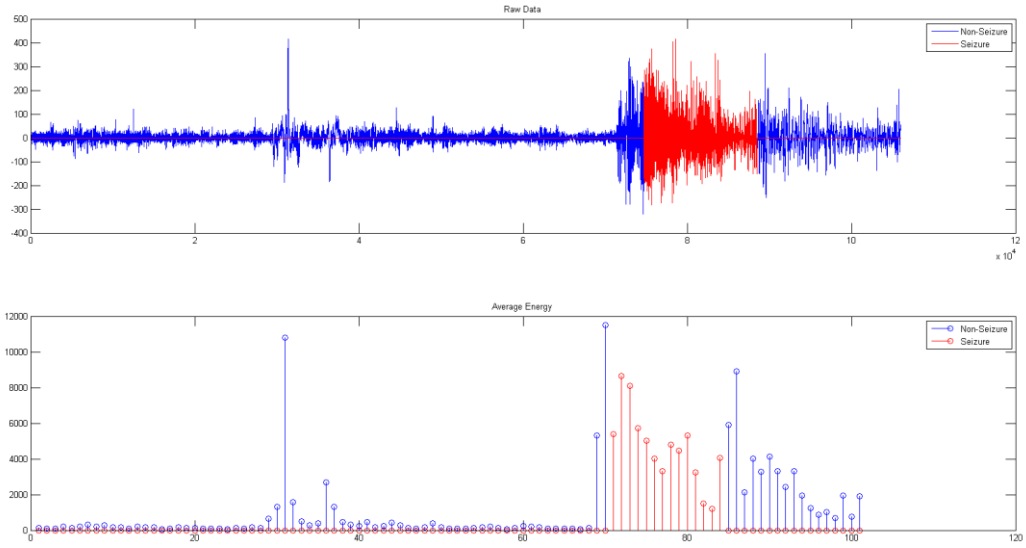


Figure 3.1: Seizure representation using energy feature

3.1.2 Coastline

The coastline feature represents the sum of the absolute value of the distance between consecutive data points; it can be described by equation (3.3).

$$CL(k) = \sum_{i=1}^{N} abs|x[i + (k-1) \times N] - x[i - 1 + (k-1) \times N]| \qquad (3.3)$$

It is to be noted that the line length feature is an adaptation of the original fractal dimensionality index that proposed by Michael J. Katz at 1988 and it is analogous to the described feature. The Neuropace responsive stimulation device employs line length as one of the tools in its detection algorithm.

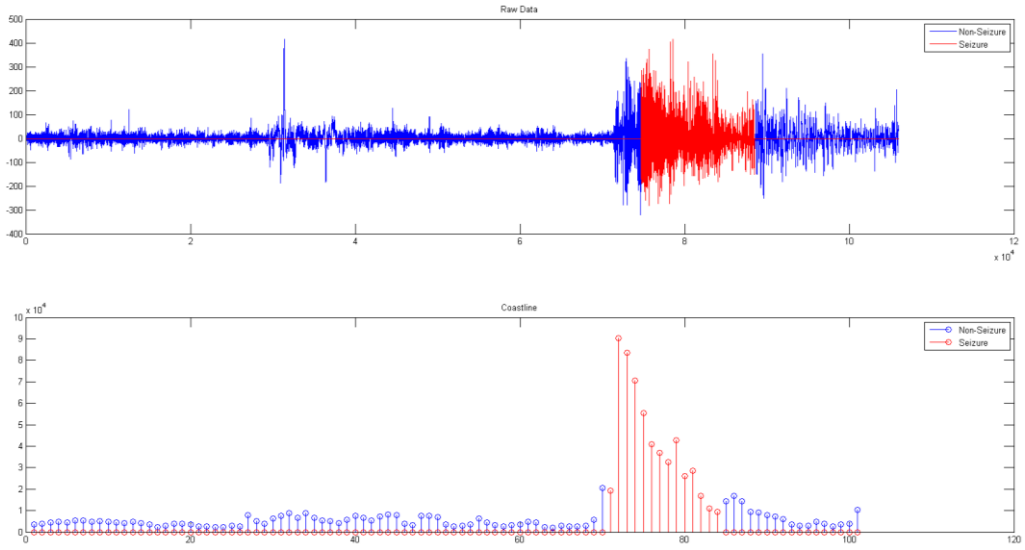


Figure 3.2: Seizure representation using coastline feature

3.1.3 Hjorth Variance Parameter

Hjorth parameters have been used extensively on iEEG based statistical calculations. The first parameter is termed as 'activity' and is equal to the variance of the signal amplitude.

$$Var[k] = \frac{1}{N} \times \sum_{i=1}^{N} (x[i + (k-1) \times N] - \mu_k)^2$$
$$= \frac{1}{N} \times \sum_{i=1}^{N} x^2[i + (k-1) \times N] - \mu_k^2 \qquad (3.4)$$

$$\mu_k = \frac{1}{N} \times \sum_{i=1}^{N} x[i + (k-1) \times N] \qquad (3.5)$$

In equations (3.4) and (3.5), the variance of a window of N samples is calculated and averaged to obtain the mean variance. 'k' represents the mean of the k^{th} window of data.

3.1.4 RMS Amplitude

The root mean squared amplitude represents the average instantaneous power of the signal. The cerebral function monitor is a widely used tool in detection of seizures. The RMS amplitude is analogous to the amplitude integrated iEEG used by the CFM.

$$A[k] = \sqrt{\frac{1}{N} \times \sum_{i=1}^{N} x^2[i + (k-1) \times N]} \qquad (3.6)$$

By definition, the RMS amplitude is similar to the energy estimate and can be defined by equation (3.6), and we picked these features to study the effect of combining mathematically similar detection features on the overall algorithm efficacy.

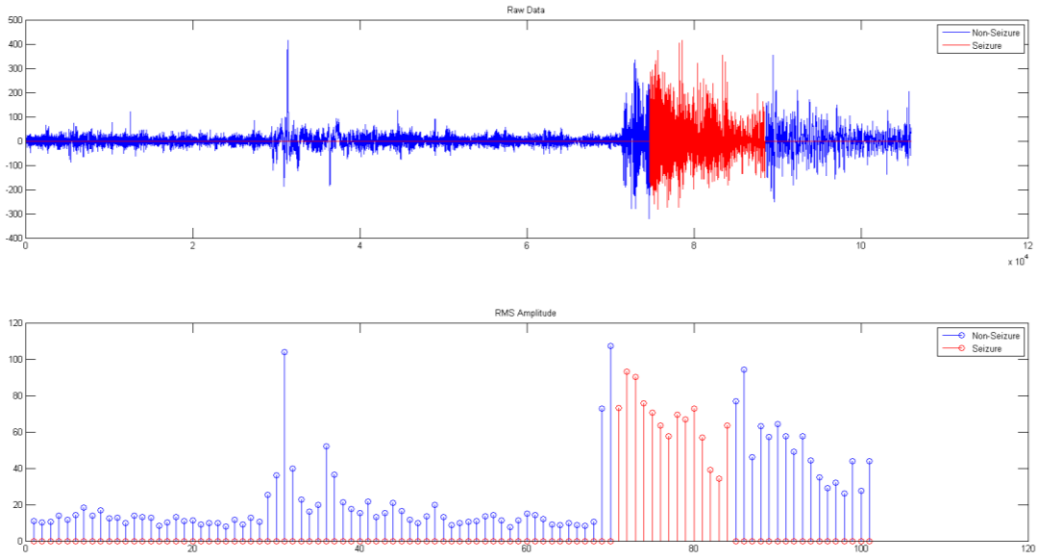


Figure 3.3: Seizure representation using RMS amplitude feature

3.1.5 Non-linear Autocorrelation

The autocorrelation definition is a value that represents the similarity between a signal and a shifted version of itself. This feature is used in an algorithm that is based on the

observation that all seizures can be identified by a repetitive spiking pattern with similar maxima and minima.

A set of 1024 samples (window of 4 seconds with sampling rate equals 256 samples/second) is taken. Then these points are reshaped into 16 columns and 64 rows, the maximum and minimum values for each row are computed. Autocorrelation is the difference between the maximum and the minimum value of each row with its shifted version. So, autocorrelation matrix consists of 64 (16*4) points.

By plotting this matrix, each column represents the point and its shifted versions in time. If there are similarities between the autocorrelation graphs this means the seizure could be located on the similarity sample.

$$HV_i = min\big[max(S_i), max\big(max(S_i + 1), max(S_i + 2)\big)\big] \qquad (3.7)$$

$$LV_i = max\big[min(S_i), min\big(min(S_i + 1), min(S_i + 2)\big)\big] \qquad (3.8)$$

Where HV_i is the ith highest value, LV_i is the ith lowest value, and $max(S_i)$ and $min(S_i)$ are the max and min values of each row in the data matrix and stored in S_i matrix.

$$Autocorr(k) = \sum_{i=1}^{N} \big(HV(i + (k-1) \times N) - LV(i + (k-1) \times N)\big) \qquad (3.9)$$

3.2 Wavelet Domain Features

A wavelet is a mathematical function useful in digital signal processing and image compression. The use of wavelets for these purposes is a recent development, although the theory is not new. The principles are similar to those of Fourier analysis, which was first developed in the early part of the 19th century.

In signal processing, wavelets make it possible to recover weak signals from noise. This has proven useful specially in the processing of X-ray and magnetic-resonance images in medical applications. Images processed in this way can be "cleaned up" without blurring or muddling the details.

Wavelet transform employs long time windows for more precise low frequency information, and short time intervals for high frequency information. The wavelet transform had better resolution and high performance for representation and visualization of the epilepsy activity than the short time Fourier transform. We use Discrete Wave transformations in our tests as it could analyze the signal at different frequency bands with different resolutions. The signals are decomposed into five different scales; we use two built-in functions in MATLAB to do this which are "*wavedec*" and "*detcoef*". We extract three features using this domain which are: average energy, fluctuation index and coefficient of variation.

3.2.1 Relative Energy

The relative energy indicates the strength of the signal as it gives the area under the curve of power at any interval of time. For the Daubechies wavelet, the sum of square of coefficients of the wavelet series is the energy of the iEEG signal. The energy of iEEG signal with limited length is given by:

$$E_r(l) = \sum_{i=1}^{N} D_i^2 \times \frac{\tau}{N} \qquad (3.10)$$

, where τ is the sampling interval and N is the number of DWT coefficients D_i presented at scale l. The relative energy of the scale is computed as:

$$E_r(l) = \frac{E(l)}{\sum_{i=1}^{S} E(i)} \qquad (3.11)$$

, where S is the number of the wavelet scales.

3.2.2 Coefficient of Variation

The standard deviation σ shows how the features' values vary compared to the mean value μ. We use mean value to measure the mean amplitude. The coefficient of variation V_c can measure the variations of the signal amplitude. The coefficient of

variation gives smaller values in pre-seizure period and increases gradually as the seizure comes closer. The corresponding coefficient of variance can be expressed as:

$$V_c = \left(\frac{\sigma(l)^2}{\mu(l)^2}\right) \qquad (3.12)$$

where

$$\mu(l) = \left(\frac{1}{N}\right) * \sum_{i=1}^{N} D_i \qquad (3.13)$$

and

$$\sigma(l) = \sqrt{\left(\frac{1}{N}\right) * \sum_{i=1}^{N} (D_i - \mu(l))^2} \qquad (3.14)$$

, where N is the number of DWT coefficients $D_i$presented at scale l.

3.2.3 Fluctuation Index

The fluctuation index is proposed to measure the intensity of iEEG signal's changes. It could be found that the fluctuation index of the iEEG during seizures usually becomes greater than that during the non-seizure periods. The mathematical description is as shown in equation (3.15)

$$FI(l) = \frac{1}{N}\sum_{i=1}^{N} |D_{i+1} - D_i| \qquad (3.15)$$

, where N is the number of DWT coefficients $D_i$presented at scale l.

3.3 Results of Time Domain Features Analysis

As there are five time domain features being tested, we needed to avoid some of them to get the most efficient combination of features. Thus the following survey is done. The results illustrated in this section are obtained from a study done on five female Long Evans rats (250–350 g). The results are normalized to the worst performing feature in order to set the boundaries of each feature's performance in comparison to the reference; allowing for fair comparison. Moreover, the comparison is done between five selected features from the previously demonstrated features in this chapter.

3.3.1 Feature Efficacy Results

Keeping in mind the definition of false positive (FP) and false negative (FN), that is false positive is a false warning of a seizure and false negative is a missed seizure that is not detected.

The percentage of time spent under false positives is reported instead of the absolute number of false positives or the false positive rate. This is in accordance with suggestions made with regards to the utility of reporting false positive rates, sensitivity and specificity for evaluating prediction and detection algorithms. On each testing dataset, the detection efficacy is calculated using equation (3.16).

$$Detection\ Efficacy = \frac{\%FP_{Time} + \%FN_{Time}}{2} \qquad (3.16)$$

Figure (3.4) shows a bar plot of the detection efficacy of the compared features. The detection efficacy recorded for each animal was averaged across the five different animals used in this study to obtain an estimate of the detection efficacy for the feature under study. Figure (3.4) (Left) uses a box plot to illustrate the distribution of detection efficacy across animals for each of the features compared in this study. The median and distribution of the data around it are captured using the box plot. The false positive and negative rates were also normalized to the worst performing feature and do not represent the absolute numbers for each feature. The normalized plots shown in Figure (3.4) capture the trend between the features under comparison in this study. In

all the plots in Figure (3.4), a lower value indicates a better performance in the bar and box plots, as noted earlier.

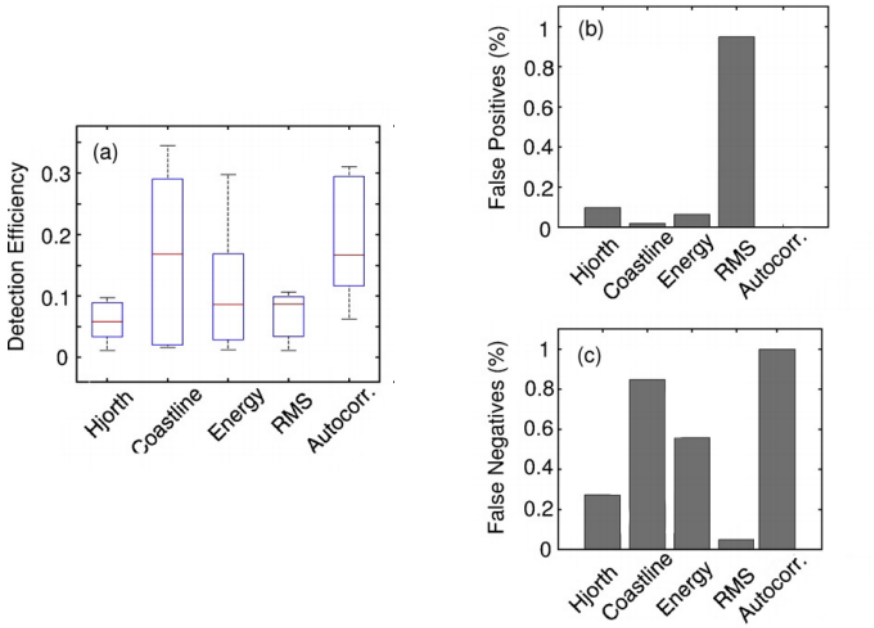


Figure 3.4: (Left) Box plot showing the distribution of detection efficacy for all the features under comparison and (Right) (Top) normalized plot of percentage time spent in false positives for the same features and (Bottom) normalized plot of percentage time spent in false negatives.

3.3.2 Hardware Analysis Results

The features under comparison in the prospective study were implemented using standard CMOS digital library cells. The total power consumption and area on silicon were normalized and averaged to calculate the hardware cost function in each case. Figure (3.5) compares the normalized hardware cost function for each of the compared features and also plots out the individual components contributing to the same. The hardware cost function was calculated by averaging power and area on chip which were already normalized with the largest power and area obtained from the individual features (Equation (3.17)).

$$Norm\ Hardware\ Cost = 0.5 \times (total\ power + area\ on\ chip) \qquad (3.17)$$

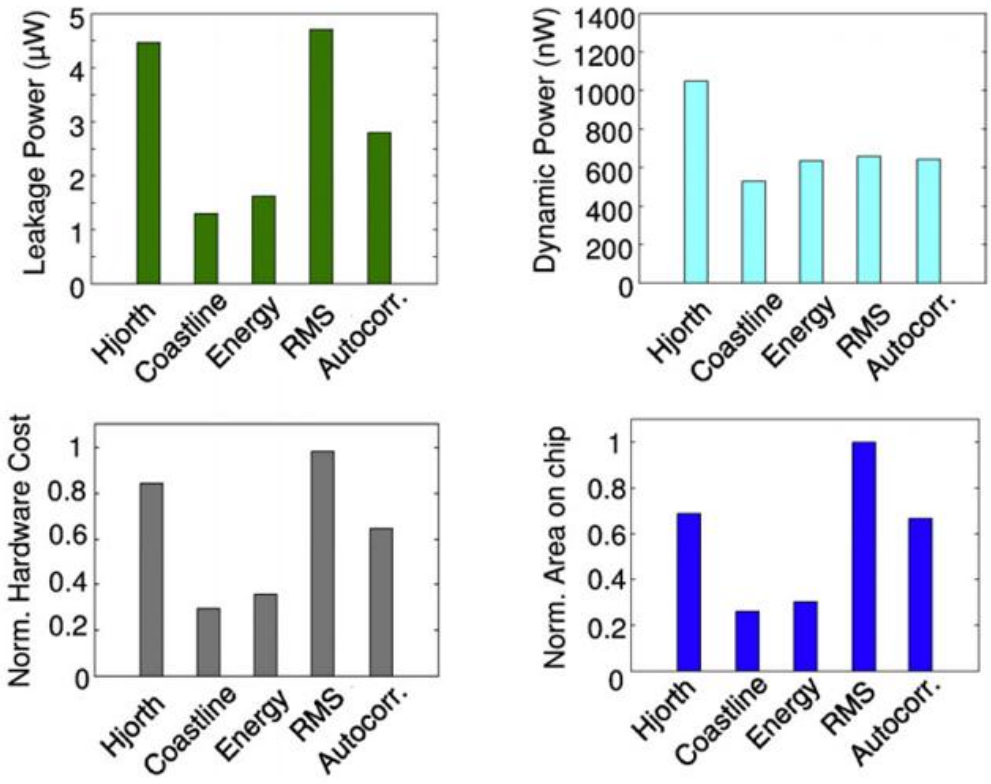


Figure 3.5: (Clockwise from top, left) Bar graphs comparing cell leakage power, average dynamic power, total area on chip and normalized hardware metric for each of the compared features in the study.

3.3.3 Detection Algorithm Design Space (DADS)

The normalized hardware cost function was plotted with the normalized detection efficacy to obtain the two-dimensional detection algorithm design space for the features under comparison.

Every feature under evaluation is represented by these two coordinates in the described design space. Features lying in the lower left corner closer to the origin would be ideally desired as this would imply high detection efficacy combined with low hardware cost.

Figure 3.6 plots the design space for the current study based on the results obtained above. The presented design space is a visualization of a classical design trade-off seen most engineering problems, and its application to adapting seizure detection algorithms for hardware implementation.

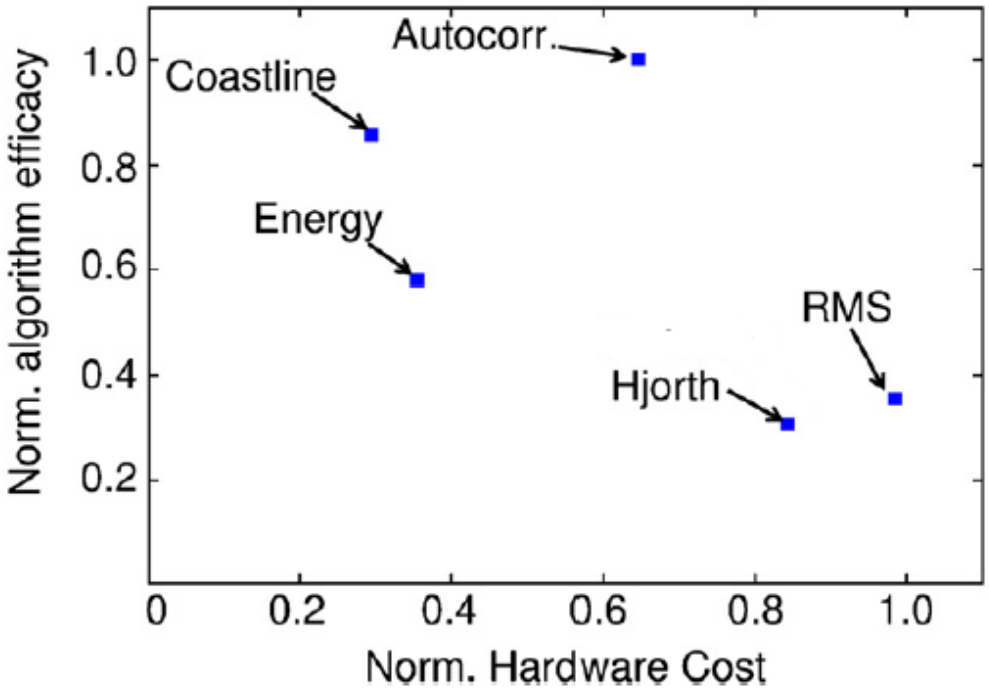


Figure 3.6: Design space illustrating hardware and detection efficacies for all the compared features in the study.

3.3.4 Discussion of Results

The definition of false positives and false negatives used, which are mentioned in Section 3.3.1, led to inherently assigning more importance to the false negatives during the training procedure. The detection efficacy numbers reported were highly sensitive to even a single missed seizure. The absolute numbers of percent time spent in false warnings were therefore much lower than the corresponding false negative numbers. The results summarized in Figure (3.4) reflect the expected penalty towards missed seizures. The Hjorth variance parameter and the RMS amplitude parameters turned out to be the best performing features from those compared in detecting seizures based on their average detection rates. Detection features that were mathematically similar understandably showed a high similarity in detection efficacy as well. The detection efficacies reported in Figure (3.4) are absolute values, obtained prior to normalization whereas the reported percent time spent in false positives and false negatives are normalized.

From the summary of hardware performance in Figure (3.5), the coastline feature consumed the lowest average power, obtained by calculating the mean of leakage and dynamic power consumption. The RMS amplitude feature was the most costly feature, as can be explained by the implementation of the square root function on chip.

The detection algorithm design space (DADS) shown in Figure (3.6) helps in prioritizing the choice of detection algorithm based on the requirements of the application. For example, an implantable monitoring device implementing an algorithm to aid in screening for seizures may be able to tolerate a much higher false positive rate in exchange for extended battery life, whereas an implanted stimulator would want to minimize the number of false positives that trigger therapeutic intervention, even if it meant a higher hardware cost.

Typically, a combination of features is used to increase the detection efficacy. In choosing features to combine, it is important to consider their relative locations in this space. We illustrate the design of one such optimized combination with an example in the next section. We also demonstrate how the code sign of hardware and algorithm results in a more optimal location in the DADS as compared to a blind combination. Of the compared features in this study, the coastline method was the most hardware efficient and the RMS feature was the least. In terms of detection efficacy, the Hjorth variance parameter outperformed all the compared features and the non-linear autocorrelation feature was the least effective in detection per the definitions used for efficacy.

Chapter 4: Seizure Classification

In this chapter, we are going to introduce the concept of classifying the results of certain feature examination into seizure or non-seizure, going through the classification techniques used for this purpose and reaching to the technique we are implementing in our project.

4.1 Introduction

Features by themselves cannot tell us if there is a seizure in a specific time or not. Of course you could see an anomaly in the feature itself but you need an algorithm that detects the seizure buried inside the extracted feature.

Classification is the stage where we classify the feature into two groups: Seizure and Non-Seizure. So, it is more likely to be an algorithm that is applied on a feature to make a decision based on its results.

For example, Figure (4.1) shows raw data from iEEG signal of a patient, the red part of the signal marks occurrence of a seizure. Figures (4.2) and (4.3) show the output from feature extraction when two different features are examined, which are coastline and energy respectively. Figure (4.2) can clearly approve the presence of seizure represented by the intensity of the feature around the sample number 100, but it is not the case in Figure (4.3), where other there are more bulks of feature intensities found at other samples along the signal.

This confirms the need for an algorithm to operate on the output from the feature extraction from the raw data; hereby we introduce the classifier. The classifier is needed to precisely make the decision whether the output describes a seizure or non-seizure.

We have used two classification techniques which are the Support Vector Machine (SVM) and Multi-window Event Based Counting Classifier in this purpose as discussed in the upcoming sections.

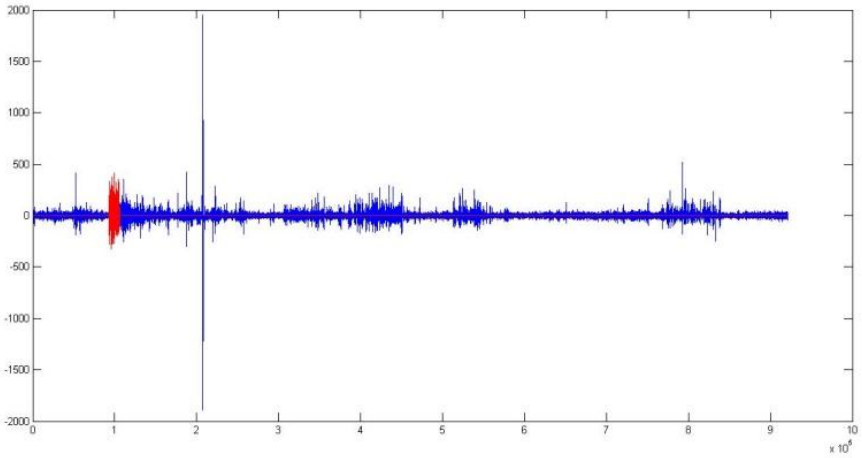


Figure 4.1: Raw data with the red part marking occurrence of a seizure

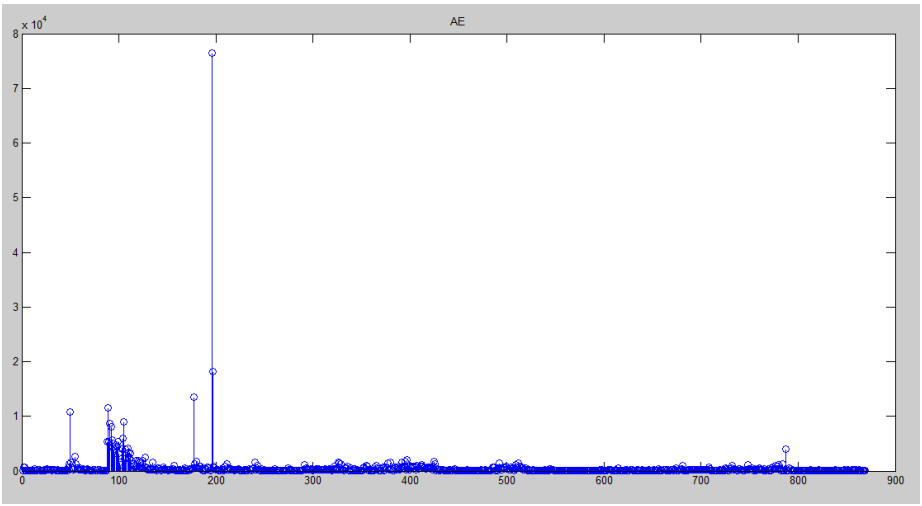


Figure 4.2: Output of feature extraction using coastline feature

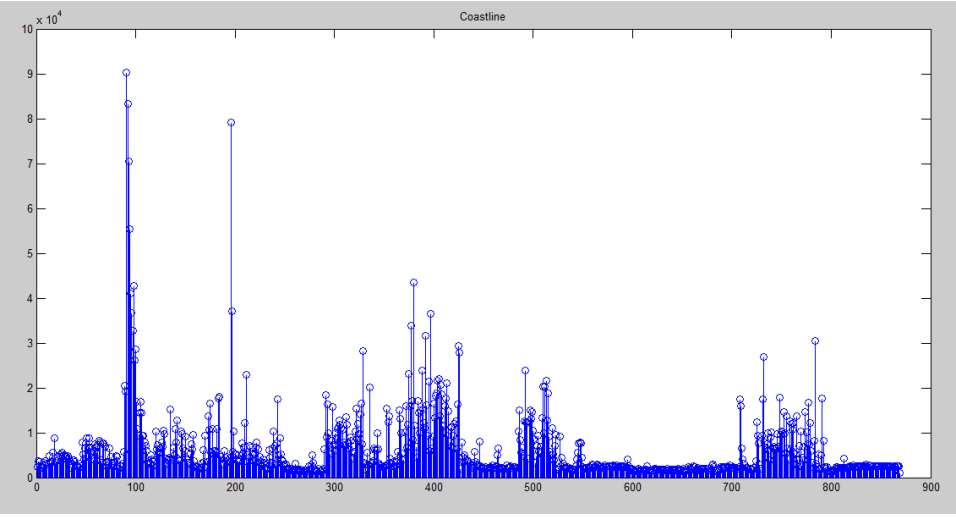


Figure 4.3: Output of feature extraction using energy feature

4.2 Support Vector Machine (SVM) Classifier

In machine learning, SVM is a learning model with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training features, each marked for belonging to one of two categories (in our case it's either seizure or non-seizure), an SVM training algorithm builds a model that assigns new examples into one category or the other.

Another way to look at the SVM model is that it's a representation of the features as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New testing features (new epochs of features that were not used in training the SVM) are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

Therefore, A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, to detect the seizures after training, The SVM just compares the new data with the trained data and puts the new data in its best related area as shown in Figure (4.4) (Consider the red dots are seizures and the blue dots are non-seizures).

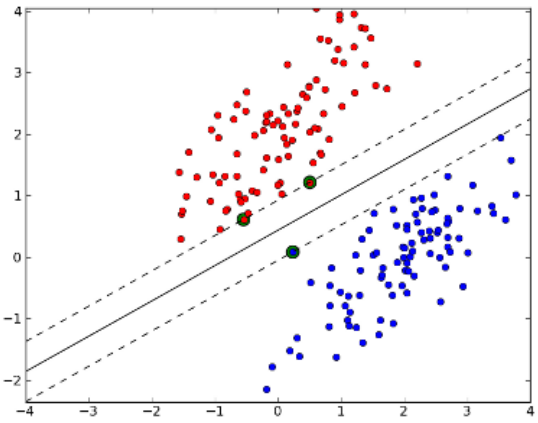


Figure 4.4: Example for a SVM model. The training of the model creates the threshold line which divides the space into two areas (Seizure area and not Seizure area).

Let us consider the following simple problem: For a linearly separable set of 2D points which belong to one of two classes, find a separating straight line.

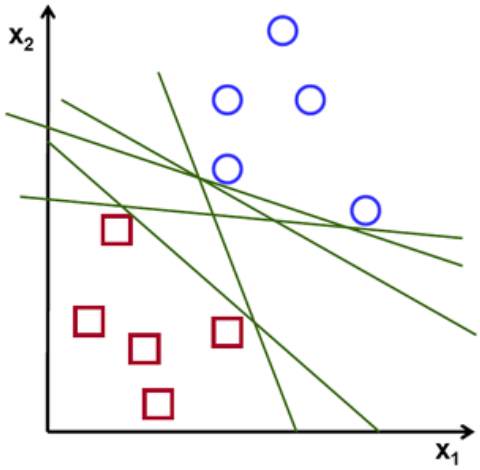


Figure 4.5: An SVM model with one feature presented in space.

Note: In this example we deal with lines and points in the Cartesian plane instead of hyperplanes and vectors in a high dimensional space. This is a simplification of the problem. However, the same concepts apply to tasks where the examples to classify lie in a space whose dimension is higher than two.

In Figure (4.5) you can see that there exist multiple lines that offer a solution to the problem. We can intuitively define a criterion to estimate the worth of the lines: A line is bad if it passes too close to the points because it will be noise sensitive and it will not generalize correctly.

Therefore, our goal should be to find the line passing as far as possible from all points. Then, the operation of the SVM algorithm is based on finding the hyperplane that gives the largest minimum distance to the training examples. Twice, this distance receives the important name of margin within SVM's theory. Therefore, the optimal separating hyperplane maximizes the margin of the training data.

How is the optimal hyperplane computed? Let's introduce the notation used to define formally a hyperplane:

$$f(x) = \beta_o + \beta^T x \qquad (4.1)$$

where β is known as the *weight vector* and β^T as the *bias*.

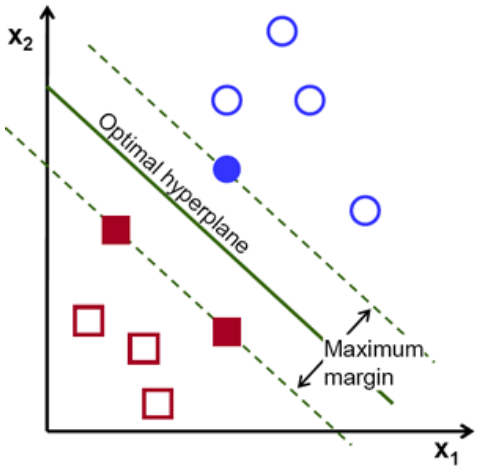


Figure 4.6: An SVM model with one feature presented in space where we can see the optimal hyperplane.

The optimal hyperplane can be represented in an infinite number of different ways by scaling of β and β^T in equation (4.1). As a matter of convention, among all the possible representations of the hyperplane, the one chosen is represented by equation (4.2):

$$|\beta_o + \beta^T x| = 1 \qquad (4.2)$$

where *x* symbolizes the training examples closest to the hyperplane. In general, the training examples that are closest to the hyperplane are called support vectors. This representation is known as the canonical hyperplane.

Now, we use the result of geometry that gives the distance between a point *x* and a hyperplane (β, β_o):

$$distance = \frac{|\beta_o + \beta^T x|}{||\beta||} \qquad (4.3)$$

In particular, for the canonical hyperplane, the numerator is equal to one and the distance to the support vectors is

$$distance_{\ support\ vectors} = \frac{|\beta_o + \beta^T x|}{||\beta||} = \frac{1}{||\beta||} \qquad (4.4)$$

Recall that the margin introduced in the previous section, here denoted as *M*, is twice the distance to the closest examples: $M = \frac{2}{||\beta||}$

Finally, the problem of maximizing *M* is equivalent to the problem of minimizing a function *L(β)* subject to some constraints. The constraints model the requirement for the hyperplane to classify correctly all the training examples x_i. Formally,

$$\min_{\beta,\beta_o} L(\beta) = \frac{1}{2}||\beta||^2 \ , \qquad y_i(\beta^T xi + \beta_o) \geq 1 \ \forall i \qquad (4.5)$$

where y_i in equation (4.5) represents each of the labels of the training examples.

This is a problem of Lagrangian optimization that can be solved using Lagrange multipliers to obtain the weight vector β and the bias β_o of the optimal hyperplane.

4.3 Multi-window Event Based Counting Classifier

This algorithm is much simpler than the SVM algorithm. We apply two thresholds α and β, where α is the threshold amplitude and β denotes the lower limit for passing the value α.

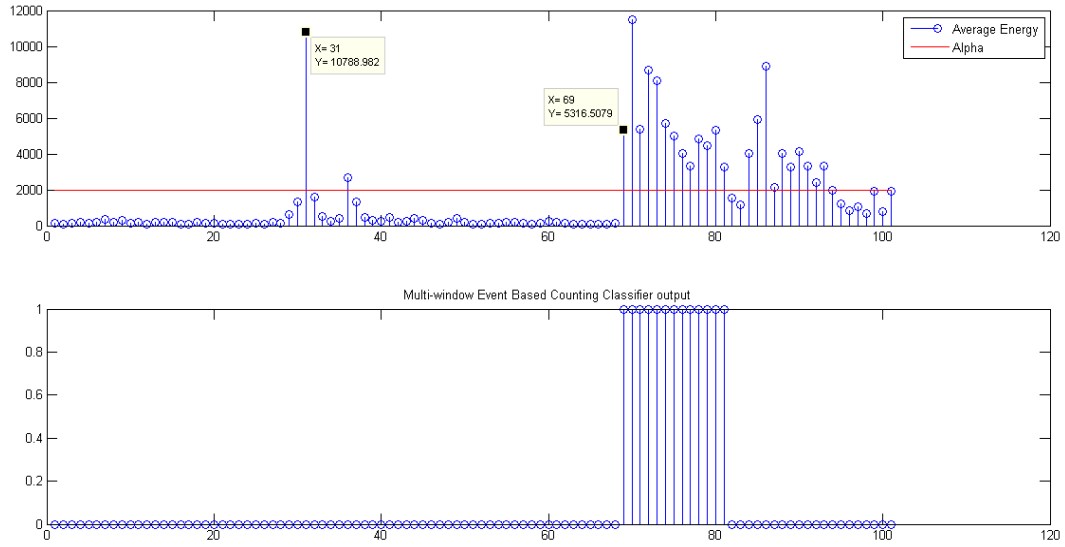


Figure 4.7: Illustration of the multi-window event based counting classifier.

To illustrate, in Figure (4.7) α=2000 *joules* (the red line) and β=12. At X=69 the Energy(Y)=5316.5 Joules, so starting from this point we have passed the threshold α a number of 13 consecutive times which is more than the value of threshold β which is totally fine, but it cannot be less than 12 consecutive times. When the two thresholds

are satisfied the classifier generates logic ‘1’s vector of length equal to or more than the value of β as shown in the lower graph in Figure (4.7).

If we don’t satisfy the two thresholds the classifier generates a vector of logic ‘0’s as shown in the flowchart illustrated below in Figure (4.8). To illustrate, at X=31 the Energy (Y) = 10788.982 which is more than the value of α, but we passed α only 1 time so we the classifier generates logic ‘0’s.

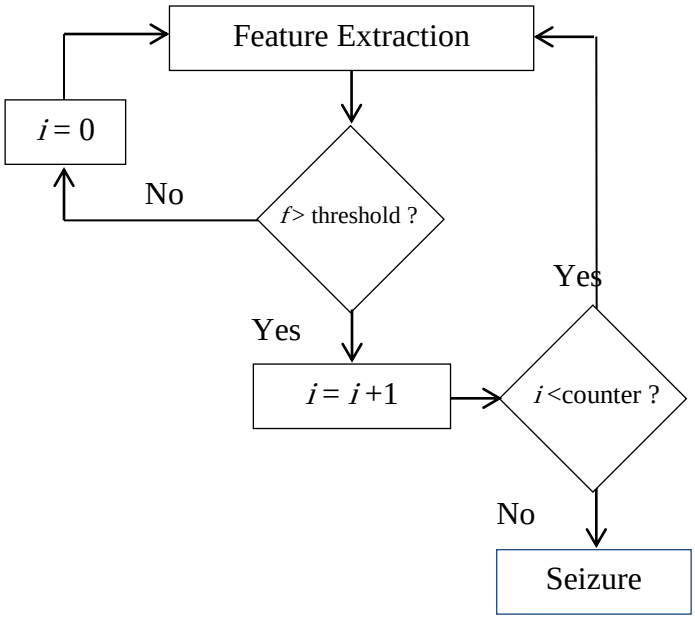


Figure 4.8: Flowchart shows the flow of the modified thresholding algorithm.

Chapter 5: Proposed Detection Algorithm

In this chapter, we will be proposing the detection algorithm that is finally implemented in our project, reached through examining many mixed algorithms, testing their results and comparing between them in terms of different design aspects.

5.1 Selected Features and Mixed Algorithms

To make the process of simulation smooth and domain-based, wavelet-domain features are tested alone with SVM classifier as well as time-domain features are tested alone with SVM classifier to compare between both types of features.

As mentioned in 3.3.4, processing the data on five features is power consuming and a combination of the best two features out of the time-domain ones is preferable. As shown in Figure (3.6), DADS of all time-domain features is plotted.

Based on the Figure mentioned, we illustrate the design of one such optimized combination with an example in the next section. We also demonstrate how the co-design of hardware and algorithm results in amore optimal location in the DADS as compared to a blind combination.

Of the compared features in this study, the coastline feature is the most hardware efficient and Hjorth variance parameter has the best detection efficiency. However, it has very complicated hardware. So, we had the idea of using two features with less detection accuracy than Hjorth parameter alone but have simpler hardware. Thus, we chose accumulated energy and coastline features to represent time-domain features.

5.2Algorithm-hardware co-optimization

In this section, we intend to combine selected features from the analysis to construct a highly efficient seizure detection algorithm for minimum possible hardware cost. The energy feature was found to be the second efficient detection feature. In order to construct an algorithm that outperforms this feature, we create a simple combination using the energy feature as a primary feature. The addition of a second feature in the combination comes at a hardware cost.

In order to find the most hardware efficient way of increasing the detection efficacy, we combine the primary feature with the lowest power consuming feature from the compared set. In this case, the coastline feature turned out to have the lowest hardware cost associated with it, as can be seen from the DADS in Figure (3.6).

It is to be noted that combining features located closely in the design space may not result in an optimized solution as such combinations double the hardware cost, decreasing the hardware efficacy by a factor of two. Any increase in detection efficacy of the combination would now have to justify the twofold increase in hardware. On the other hand, simple combination of a low hardware cost feature with a highly detection efficient feature would only increase the hardware by a small amount. In such a case, any increase in the detection efficiency of the combination would yield a much higher detection efficacy per unit hardware cost added. This is illustrated in the described example. The goal of the example combination is to improve the detection efficacy of the energy feature without significantly adding to the hardware cost. A simple combination technique was used with both 'AND' and 'OR' operations. Figure (5.1) captures the effect of this combination on the design space. The 'OR' combination improves the detection efficacy while the 'AND' combination improves the hardware cost.

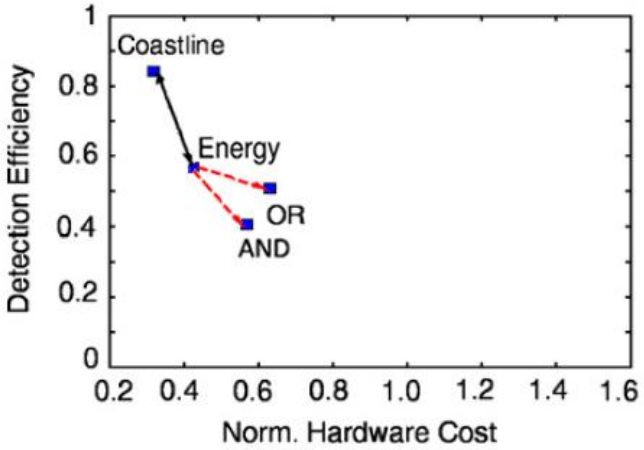


Figure 5.1: Design space reflecting changes in detection and hardware efficacy due to a simple OR and AND combination of two selected features.

The choice of which combination to use would depend on the specific needs of the application. Once the combination has been set and the target detection efficacy is

met, co-optimization of hardware and algorithm would allow for maximizing detection efficacy for minimal hardware cost. Thus, in our design we care about efficacy so we chose OR combination.

Now, we have two algorithms; the first one consists of average energy, fluctuation index and coefficient of variation features followed by modified SVM classifier representing wavelet-domain features as shown Figure (5.2).

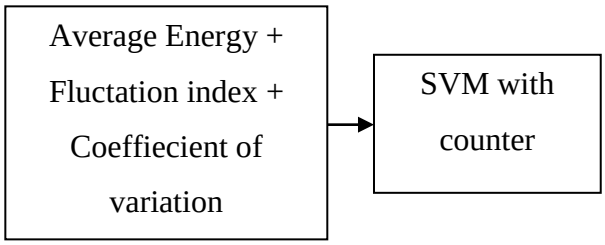


Figure 5.2: First algorithm which represents wavelet-domain features with modified SVM classifier.

The second algorithm is having accumulated energy, coastline features followed by modified SVM classifier representing time-domain features as shown in Figure (5.3).

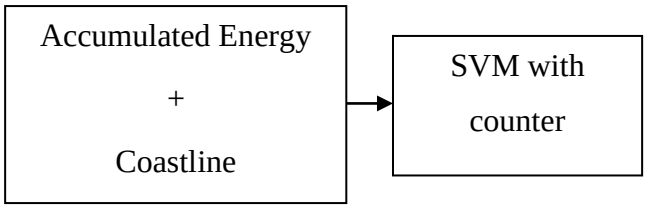


Figure 5.3: Second algorithm which represents time-domain features with modified SVM classifier.

Finally, as the time-domain features require no domain conversion, we derived a new algorithm; the proposed algorithm, which is the selected time-domain features – accumulated energy and coastline- followed by multi-window event-based counting threshold classifier without the SVM block as shown below in Figure (5.4). This is easier to implement that using SVM. However, the thresholds become more sensitive and they need very fine tuning.

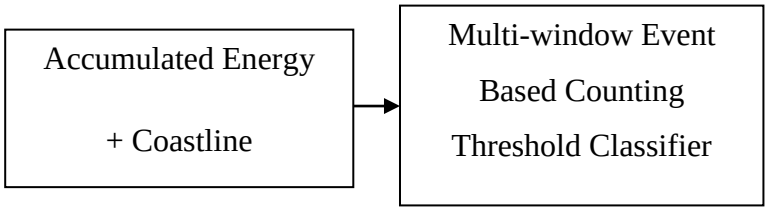


Figure 5.4: Third algorithm which represents time-domain features with multiwindow event-based counting threshold without SVM block.

5.3 Input Data

This database, collected at the Children's Hospital Boston, consists of iEEG recordings from pediatric subjects with intractable seizures. Subjects were monitored for up to several days following withdrawal of anti-seizure medication in order to characterize their seizures and assess their candidacy for surgical intervention. We tested all our algorithms on 5 patients of these dataset (Patients number 1, 3, 5, 19 and 21).

As some seizures could be in terms of tens of seconds, we needed the operating window to be less than 5 seconds in order for the classifier to work properly. However, we perform a division operation in the Energy feature to get the average energy value. To optimize the hardware implementation afterwards, we thought of choosing the window to be multiple of 2 seconds; as 2 seconds or 4 seconds. The both options are applicable and we chose our window to be 4 seconds.

Thus, the data is divided into epochs, each epoch is 4 seconds. And each second is represented by 256 points (the sampling frequency is 256 Hz) For training data, one hour is taken from each patient's data and classified as shown in Table 5.1.

Patient#	Seizure epochs	Non- Seizure epochs
1	8	712
3	11	709
5	23	697
19	16	704
21	10	710

Table 5.1: Seizure and non-seizure epochs in the training data for each patient.

For the testing data, they are classified as follows in Table 5.2.

Patient#	Number of Hours	Seizure epochs	Non- Seizure epochs
1	4 Hours	50	2830
3	3 Hours	28	2132
5	4 Hours	88	2792
19	2 Hours	31	1409
21	37 Hours	59	26581

Table 5.2: Seizure and non-seizure epochs in the testing data for each patient.

5.4 Comparison Metrics

To be able to compare different algorithms, we need to settle on fair metrics for all the algorithms. These metrics are sensitivity, specificity and accuracy. They are calculated using four other metrics; True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN). These four metrics are obtained after running the detection algorithm on the input iEEG data as described in equations 5.1, 5.2 and 5.3 below.

$$Sensitivity = \frac{TP}{TP + FN} \times 100 \qquad (5.1)$$

$$Specificity = \frac{TN}{TN + FP} \times 100 \qquad (5.2)$$

$$Accuracy = \frac{TN + TP}{TN + TP + FN + FP} \times 100 \qquad (5.3)$$

5.5 Results

Using MATLAB, we conduct simulations for the three algorithms using the input data stated above and obtain the following results in Tables (5.3), (5.4) and (5.5).

Patient#	TP	FP	TN	FN	Sensitivity	Specificity	Accuracy
1	38	0	2522	35	**52%**	**100%**	**98.66%**
3	8	328	26237	67	**55%**	**98.7%**	**98.51%**
5	52	34	2751	41	**55.9%**	**98.77%**	**97.39%**
19	7	0	2790	29	**19.44%**	**100%**	**98.973%**
21	5	66	1222	23	**17.85%**	**94.87%**	**93.23%**
Average					**40.03%**	**98.46%**	**97.35%**

Table 5.3: Results of first algorithm of wavelet-domain features with SVM classifier.

Patient#	TP	FP	TN	FN	Sensitivity	Specificity	Accuracy
1	50	5	2547	23	**68.49%**	**99.8%**	**98.93%**
3	59	305	26258	16	**78.66%**	**98.85%**	**98.79%**
5	79	116	2671	14	**84.94%**	**95.83%**	**95.48%**
19	21	8	2781	15	**58.33%**	**99.71%**	**99.18%**
21	29	3	2003	3	**90.62%**	**99.85%**	**99.7%**
Average					**76.2%**	**98.8%**	**98.41%**

Table 5.4: Results of second algorithm of time-domain features with SVM classifier.

Patient#	TP	FP	TN	FN	Sensitivity	Specificity	Accuracy
1	12	5	701	1	**92.64%**	**99.29%**	**99.16%**
3	12	10	695	1	**91.3%**	**98.47%**	**98.33%**
5	16	2	694	7	**68.75%**	**99.78%**	**98.75%**
19	7	0	677	9	**45.45%**	**100%**	**98.7%**
21	10	2	711	0	**100%**	**99.76%**	**99.76%**
Average					**79.62%**	**99.46%**	**98.94%**

Table 5.5: Results of third algorithm of time-domain features with multiwindow event-based counting threshold.

5.6 Conclusion and Proposed Algorithm

Based on the previous results, it is obvious that the third algorithm exhibits the best performance metrics; accuracy of 98.94%. Also, it requires no domain transformation –automatic data processing- in contrary to wavelet-domain algorithm.

This algorithm shows better performance due to the removal of all complex mathematical formulae. As both SVM & wavelet-domain features are mathematically complex blocks and need much more processing than our proposed algorithm.

The final block diagram of the proposed algorithm –the third one- is illustrated in Figure (5.5).

Decision making block is the responsible for deciding whether this is a seizure or not depending on the results of thresholds block and as the result of the threshold blocks are either '1' for seizure signal and '0' for non-seizure signal, so the decision making block is a logic combination between threshold blocks' outputs. This logic combination could be OR-gate or AND-gate. However, AND-gate is selective so the algorithm could miss seizures –as they could be detected by one feature not by the other- so the efficiency of the overall system decreases. So, the best option is OR-gate.

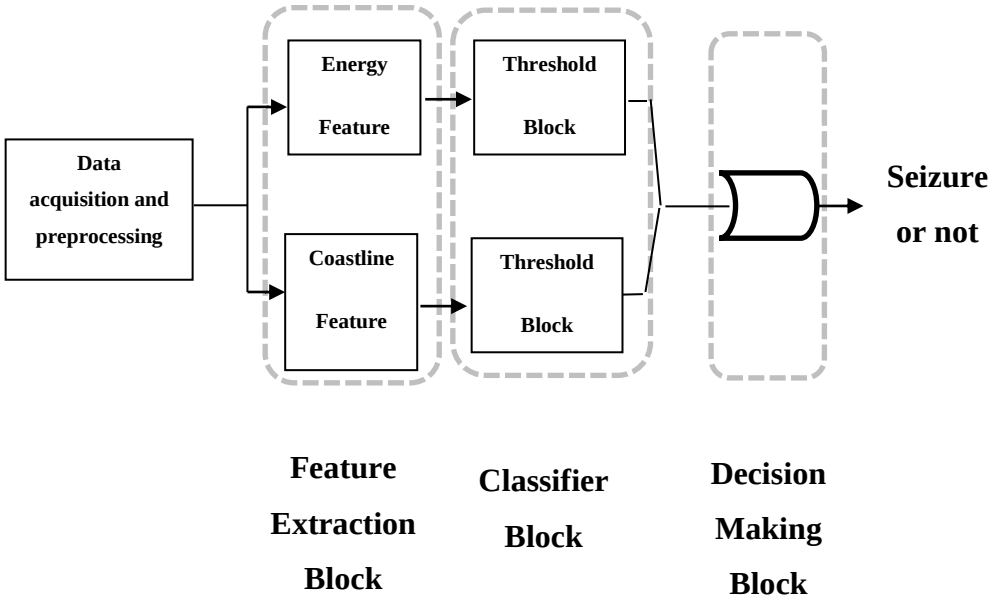


Figure 5.5: Block diagram of the proposed seizure detection algorithm.

Chapter 6: Seizure Prediction

Prediction was our idea from the beginning. The project was about seizure detection and stimulator only but we introduced prediction block before the detection and stimulator. In this chapter we are going to discuss the importance of prediction and techniques used for this purpose.

6.1 Introduction

What is the importance of prediction? What are the benefits of predicting a seizure? Before answering these questions, what will happen if the prediction block does not exist, the detection, which is the core of the project will work all the time trying to detect the seizure and can't be turned off as not to miss any seizure. This will consume much power as the features implemented in detection algorithm, which are coastline and average energy, have high complexity resulting from division, summation and square root blocks which cause relatively high computing power, and power consumption is one of the most important key parameters in digital design scheme.

We thought about reducing the consumed power by turning off the detector and turn it on only in case of receiving a trigger from the all-the-time powered on prediction block as shown in Figure 6.1. This is the importance of the predictor to trigger the detector.

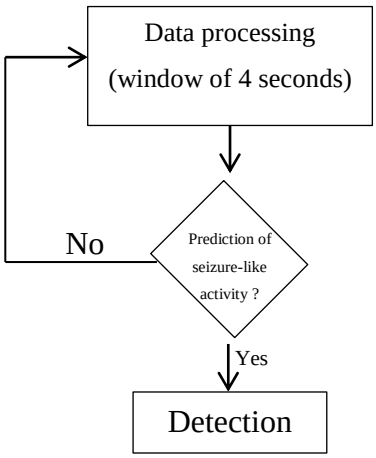


Figure 6.1: Flow chart showing the data flow starting from processing, passing by prediction to detection

Now, the predictor will work all the time, when it predicts a seizure, the detector will work and predictor turns off until the detecting block examine the signal predicted to contain seizure and decide whether to trigger the stimulator device in case of true detection or not to trigger it in case of false detection. After a time the predictor turns on again and the detector goes off and so on.

We must ensure in our idea that the predictor have to consume power less than that consumed by the detector otherwise the predictor will be no longer necessary.

There are a lot of features that can be used in prediction like non-linear features, wavelet domain features, time domain features and statistical values features (e.g.: mean, variance).

Predicting seizures potentially carries even greater advantages compared with seizure detection. Such devices might be useful both in preventing accidents and in improving outcomes, ultimately allowing early treatment or even prevention of seizures. A survey of 141 patients with epilepsy found that more than 90% of respondents believed that the development of means to predict seizures was important. These patients voiced a preference for sensitivity over specificity in seizure prediction.

Prediction systems must be able to identify preictal changes that – if present – occur within minutes, hours, or days prior to seizures. Note that the features used to predict seizures in advance may or may not be the same as those used to detect the presence of a seizure.

6.2 Prediction Techniques I

Throughout our research and survey phase of the project, we tested the following techniques:

6.2.1 The Information Theory Based Analysis and Entropy

There are many techniques used in signal analysis using information theory basis. In our research, we investigated Shannon entropy, spectral entropy, approximate entropy, sample entropy and Lempel-Ziv complexity techniques. In this section, we are going to investigate some of these techniques mentioning their mathematical representations.

6.2.1.1 Shannon Entropy

This technique is one of the most basic techniques in information theory, implying the Equation (6.1).

$$H = -\sum_{j=1}^{J} P_j \log_2(P_j) \qquad (6.1)$$

Where,

$$P_j = \frac{N(x_j)}{N} \qquad (6.2)$$

where $N(x_j)$ is the amount of samples that fall into bin j of total J bins to the total samples N. iEEG Shannon entropy has been correlated with Desflurane effect compartment concentrations. It has also been used in order analyze long term iEEG coming from patients with frontal lobe epilepsy.

6.2.1.2 Spectral Entropy

$$H = -\sum_{j=1}^{J} P_j \log_2(P_j) \qquad (6.3)$$

where,

$$P_j = \frac{S_j}{S} \qquad (6.4)$$

Where S is the total spectrum and S_j is the spectrum at frequency bin j of total J bins.

6.2.1.3 Lempel-Ziv Complexity

It has been used in a wide variety of applications including biomedical signal analysis, quantifying regularity of time series and genome data analysis and classification. The Lempel–Ziv measure estimates the rate of recurrence of patterns along a time series, reflecting a signal's complexity. Lempel– Ziv has been applied to epileptic iEEG signal showing increased values during ictal periods.

6.2.2 Non-linear Methods (Lyapunov Exponent)

This method along with the Lempel-Ziv complexity method are very complex, we tried them on MATLAB, but with no satisfying results, so we did not proceed using them further more in our research. Another reason for not using all the non-linear methods is that they consume huge power. In this case, the advantage of prediction is lost. As a result we investigated another approach which is the linear methods approach.

6.2.3 Linear Methods

Linear methods, in general, consume less power than the non-linear ones. We investigated some of these methods, including Hjorth parameters, accumulated energy and others that use statistical measures as mean and variance.

These linear techniques are used usually in detection, but we applied some modifications to them to make them usable in prediction, and this was of great advantage to us.

6.2.4 Seizure Prediction System Proposed by Araabi

It comprises three stages: preprocessing, feature extraction and thresholding, and rule-based decision making as illustrated in Figure (6.1).

6.2.4.1 Preprocessing

The purpose of this stage was first to remove both high frequency noise and low frequency activity and subsequently to divide the iEEG signal into quasi-stationary segments. For this two-fold purpose, the iEEG data were band-pass filtered between 0.5 and 100 Hz using a 4th order digital Butterworth filter, and notch filtered to remove 50 Hz power line noise. Then, the filtered iEEG data were partitioned into non over lapping 10-second segments.

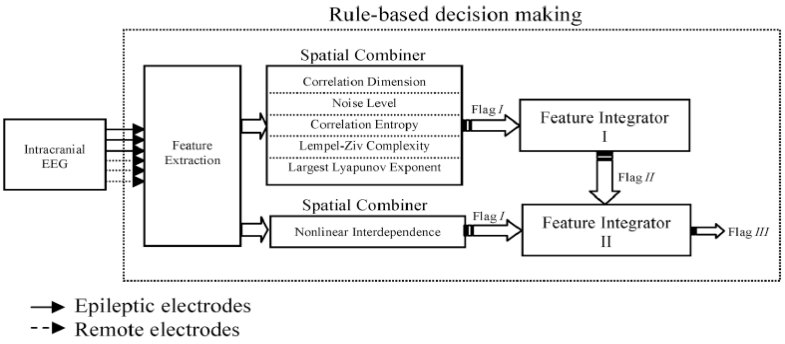


Figure 6.2: Seizure prediction system proposed by Araabi

6.2.4.2 Feature Extraction and Thresholding

This stage aimed at extracting relevant features, which contained specific characteristic properties of iEEG signal, and were suitable for the seizure prediction task.

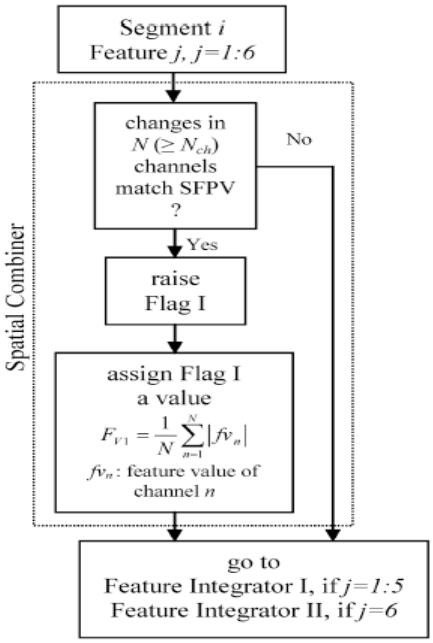


Figure 6.3: Spatial combiner block in the rule-based decision making stage

6.2.4.3 Rule-based Decision Making

The rule-based decision making stage included a spatial combiner to integrate the spatial information obtained from the multichannel iEEG data, and feature integrators to combine the information embedded in different features in a way to obtain maximum sensitivity and specificity for seizure prediction.

6.2.4.3.1 Spatial Combiner

The spatial combiner included the criteria for spatially combining the information embedded in features values extracted from the iEEG of different channels, as illustrated in Figure (6.2). The spatial combiner worked on a single feature-multichannel basis. The spatial combiner was applied to each feature separately to identify multichannel seizure precursors.

6.2.4.3.2 Feature Integrator I

This feature integrator integrated decisions made for any segment in the previous step to locate seizure precursors as shown in Figure (6.3).

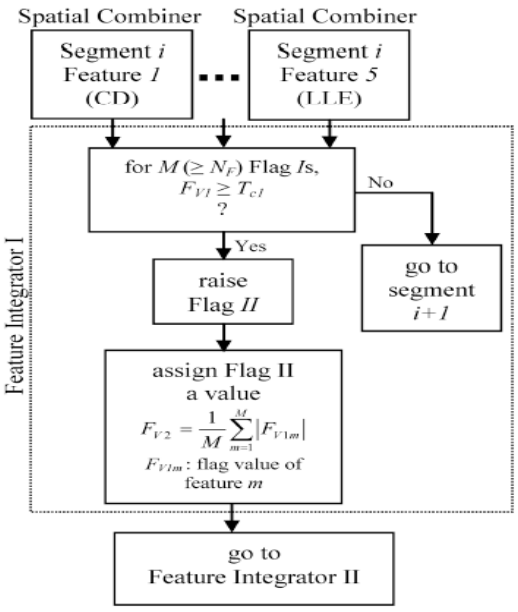


Figure 6.4: Feature integrator I block in the rule-based decision making stage

6.2.4.3.3 Feature Integrator II

This feature integrator integrated flag Is and flag IIs. For any segment, if a flag II was raised using the univariate measures while a flag I was also raised using the bivariate measures (provided that their flag values exceeded a significance threshold Tc2), then a flag III, which represented a definitive seizure precursor, is raised for that segment. This is shown in Figure (6.4).

6.2.4.3.4 Post-processing

In the post-processing step, any flag IIIs not followed by at least three other flag IIIs were rejected as short false predictions representing precursors whose lengths did not exceed 40 sec. All of the remaining flag IIIs were considered as definite predictions.

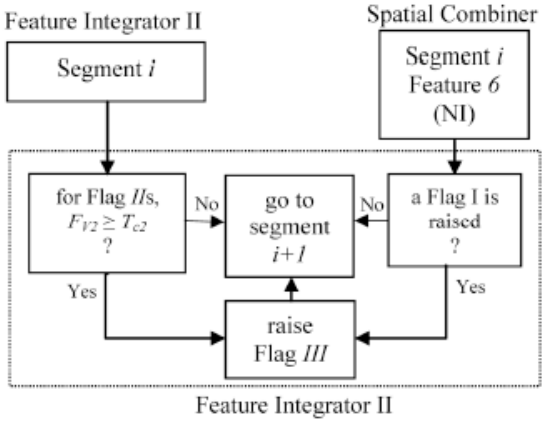


Figure 6.5: Feature integrator II block in the rule-based decision making stage

6.2.5 Results for the previous algorithms

The results of the previous algorithms were not satisfactory at all, either they consume a lot of running time on Matlab, or the output is not as expected. Thus, we started innovating new algorithms.

6.3 Prediction Techniques II

As mentioned above, we started innovating new prediction algorithms. We thought that we need a simpler technique based on statistical values to be operated with minimum power.

6.3.1 Forecasting Technique

The simplest forecasting model is the mean model, which works on the time series signal that consists of independently and identically distributed values. Then, the next value should be predicted to be equal to the historical sample mean. To forecast one sample, we use the following equation

$$\text{Forecasted Sample} = \bar{X} \pm \left(t \times SE_{fcst}\right) \qquad (6.5)$$

where $\bar{X}$ is the sample mean of n size population, is t critical *t*-value and SE_{fcst} is standard error of the forecast which measures the forecasting risk.

$$\bar{X} = \sum_{i=1}^{n} x_i / n \qquad (6.6)$$

$$SE_{fcst} = \sqrt{s^2 + SE^2{}_{mean}} \qquad (6.7)$$

where s is the standard deviation of n size population and SE_{mean} is standard error of the mean which measures the parameter risk.

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{X})^2}{n-1} \qquad (6.8)$$

$$SE_{mean} = {}^{s}/_{\sqrt{n}} \qquad (6.9)$$

As we are interested in seizure-like activities which is rare and have low possibilities, thus 95% is a suitable confidence interval and corresponding t-value equals 2 as mentioned in Robert Nau's, “Review of basic statistics and the simplest forecasting model: the sample mean”. Now, as we operate on a window that consists of 1024

samples; therefore we are going to use 1st second in the window to forecast the next second and so on as shown in Figure 6.6.

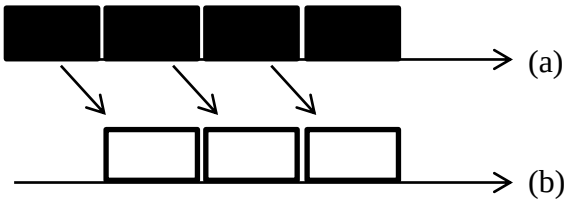


Figure 6.6: Showing how the window is divided (a) to use every second to forecast the following second (b)

Moreover, as shown in equation (6.5), this algorithm does not forecast the whole window; it just uses the previous window's statistical features –mean and standard deviation- to forecast a sample. Thus, we consider predicted second consists of identical samples that are equal to the forecasted sample. Thus, $n = 256$ and $t = 2$ to achieve 95% confidence interval. After calculating each predicted group of samples, we calculate the cross-correlation between each second and its predicted version as well as the auto-correlation of the second with itself. The nearer the cross-correlation value to auto-correlation value, the more accurate prediction is, which means that there is no seizure-like activity.

6.3.2 Mean with Basic Threshold Technique

As the previous algorithm is complicated and difficult to be implemented as the cross-correlation function and auto-correlation functions are very hardware-demanding algorithms, we thought of simplifying this technique to only use the mean value. We permit decreasing in the accuracy of the prediction block as it just triggers the detection block to simplify the hardware and reduce the overall system power consumption. This algorithm is comparing the mean of each two consecutive windows by the following condition

$$mean_{new}/_{mean_{old}} > threshold \qquad (6.10)$$

If this condition is valid, then there is an expected seizure-like activity that could happen. Thus, the prediction block triggers the detection block.

6.3.3 Mean with Basic Threshold Technique

To avoid the fine tuning required for the previous algorithm's threshold and to be able to track the patient's status, we introduce an adaptive algorithm in which the value of the threshold swings between more options to overcome triggering the detection block too much. It depends on comparing the windows' means as the previous algorithm and to be more accurate the value of the threshold doesn't change after one comparison, it changes after having the same result for specific number of windows; also it depends on the detection block decision as shown in the following flowchart in Figure 6.7.

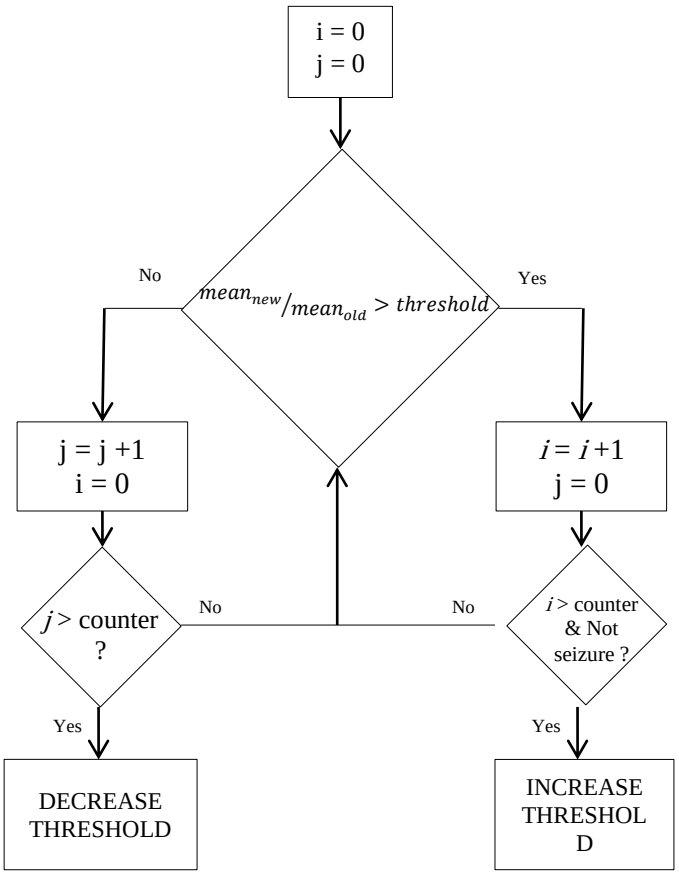


Figure 6.7: Flowchart of the adaptive threshold algorithm

6.4 Detailed Data Flow

After the preprocessing of the input iEEG data including sampling and window partitioning, the basic data unit that we deal with is the window –which is 4 seconds of the recorded input iEEG data sampled with 256 Hz-.

The first block is the prediction block. As long as the prediction block's condition is not satisfied, the system stays in this block. Once the condition is true, the detection block is triggered and is ready to work on the next input window.

The windows' counter is the block that responsible for handling the operation of prediction and detection blocks. Once the detection block is triggered and ready to work, the windows' counter counts certain number of windows in which if the detection block failed to detect a seizure, the detection block is turned off and the prediction block is triggered again. The final data flow diagram is shown in Figure 6.8

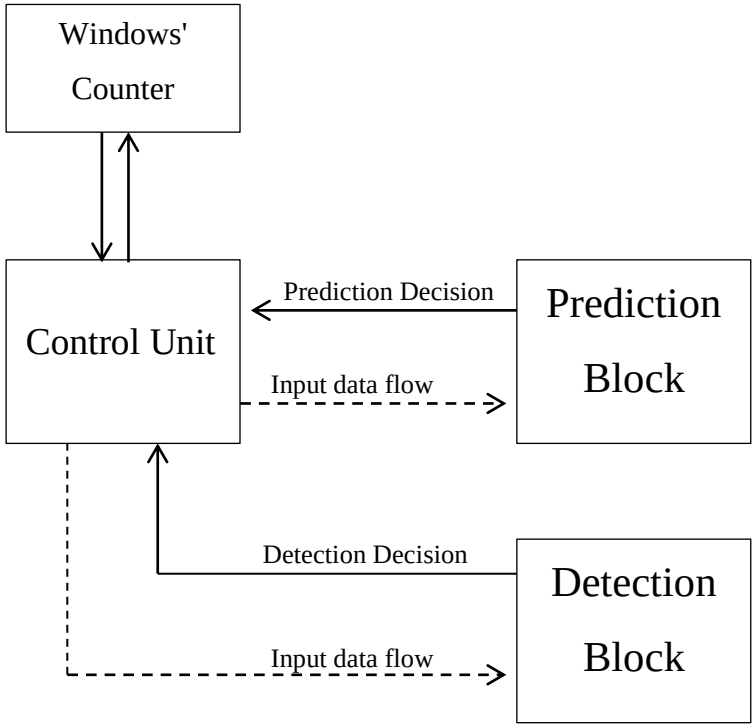


Figure 6.8: Data flow diagram for the final prediction and detection algorithms

6.5 Simulations and Results

To be able to get fair results, we got the same input data used with the detection algorithm only, to compare the sensitivity, specificity and accuracy of the standalone detection algorithm with these of the full algorithm –prediction and detection algorithms together- as shown in Tables 5.1 and 5.2.

It is to be noted that the data flow diagram shown in Figure (6.8) considers the detection block working with the proposed algorithm previously illustrated in Figure (5.5).

We have four testing metrics to test this algorithm. The first three are the sensitivity, the specificity and the accuracy of the overall algorithm which should match the standalone detection results which are average sensitivity of 79.62%, average specificity of 99.46% and average accuracy of 98.94%.

The fourth metric is the detection block duty time; as the detection block –without the prediction block- works 100% of the time. After deploying one of these prediction algorithms, the detection block only works when it is triggered by the prediction block which for sure decreases its working time. However, this working duration varies depending on the used prediction algorithm –if the prediction algorithms' accuracy is low, it is a must to trigger the detection block too much which means the detection block works most of the time. Therefore, the detection block duty time may be close to 100%. Thus, we need the overall accuracy to be maximum as much as possible and the detection block duty time to be minimum as much as possible.

6.5.1 Forecasting Technique Results

As shown below in Table 6.1, the full algorithm- prediction with detection algorithms- accuracy is 96.57% which is somehow close to 98.94%. However, the detection duty time is 93% which means that the prediction block triggers the detection block frequently and the detection block almost works as a standalone without any power consumption.

Patient#	Detection Duty Time	Sensitivity	Specificity	Accuracy
1	**93.82%**	**68.57%**	**95.47%**	**95.14%**
3	**93.23%**	**50%**	**97.28%**	**96.46%**
5	**93.92%**	**67.39%**	**99.1%**	**98.1%**
Average	**93.66%**	**61.99%**	**97.28%**	**96.57%**

Table 6.1: Results for the forecasting algorithm

6.5.2 Mean with Basic Threshold Technique Results

In contrary to the forecasting algorithm, the basic mean algorithm's accuracy is 97%, which is better than the accuracy of the forecasting algorithm as a predictor. Also, the detection duty time is much better as well as shown in Table 6.2. However, the basic mean algorithm is incapable of tracking the patient's status causing a problem of the threshold's fine tuning.

Patient#	Detection Duty Time	Sensitivity	Specificity	Accuracy
1	**50.59%**	**68.57%**	**96.1%**	**95.76%**
3	**88.92%**	**39.22%**	**98.2%**	**97.15%**
5	**79.64%**	**65.22%**	**99.18%**	**98.1%**
Average	**73.05%**	**57.67%**	**97.82%**	**97%**

Table 6.2: Results for the mean with basic threshold algorithm

6.5.3 Adaptive Threshold Technique Results

The adaptive algorithm shows the best results among the three prediction algorithms as shown in Table 6.3.

Patient#	Detection Duty Time	Sensitivity	Specificity	Accuracy
1	**50.59%**	**68.57%**	**96.1%**	**95.76%**
3	**8.85%**	**54.9%**	**97.88%**	**97.12%**
5	**51.72%**	**63.04%**	**99.5%**	**98.33%**
Average	**37.05%**	**62.17%**	**97.83%**	**97.07%**

Table 6.3: Results for the adaptive threshold algorithm

6.5.4 Final Results

We deployed the adaptive threshold algorithm-which is totally developed by our team and is introduced for the first time- as it showed much flexibility in determining the prediction threshold as well as there are no mathematical complications with 97.07% accuracy which is very close to the standalone detection algorithm. Also, as mentioned in Table 6.3, it reaches high detection accuracy with the least detection block duty time (37.05%) among all the other prediction algorithms.

Chapter 7: Hardware Design Implementation

We have implemented the full prediction & detection algorithm – adaptive threshold predictor with energy/coastline features and multi-window event based classifier- in VHDL to be able to take our design to the next phase; which is the hardware implementation.

7.1 Finite State Machine

Our design is founded based on Finite state machine technique, as we designed four state machines; one for prediction block, one for energy branch, one for coastline branch & one for decision making as shown in Figures 7.1, 7.2 & 7.3.

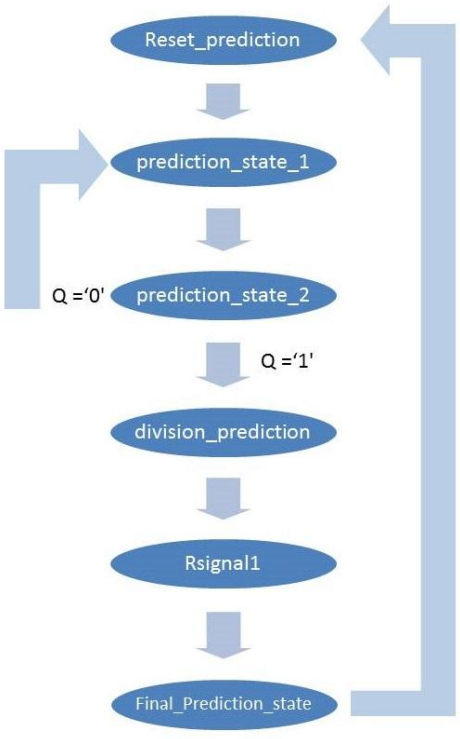


Figure 7.1: Prediction Branch Finite State Machine

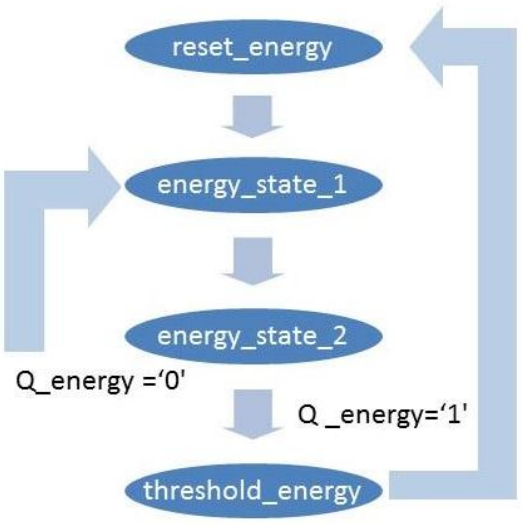


Figure 7.2: Energy Branch Finite State Machine

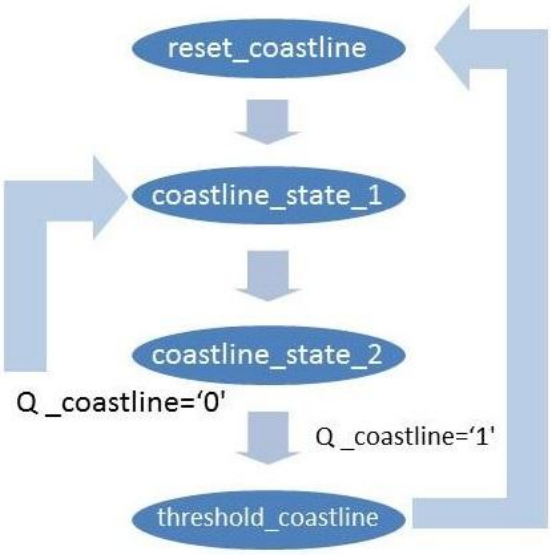


Figure 7.3: Coastline Branch Finite State Machine

7.2 VHDL Code

Full code is mentioned in appendix F. This code is synthesizable, tested on FPGA and working as expected.

7.3 ISIM Results

The *output* signal resembles the final output after all the prediction and detection took place, as shown in below Figure 7.4.

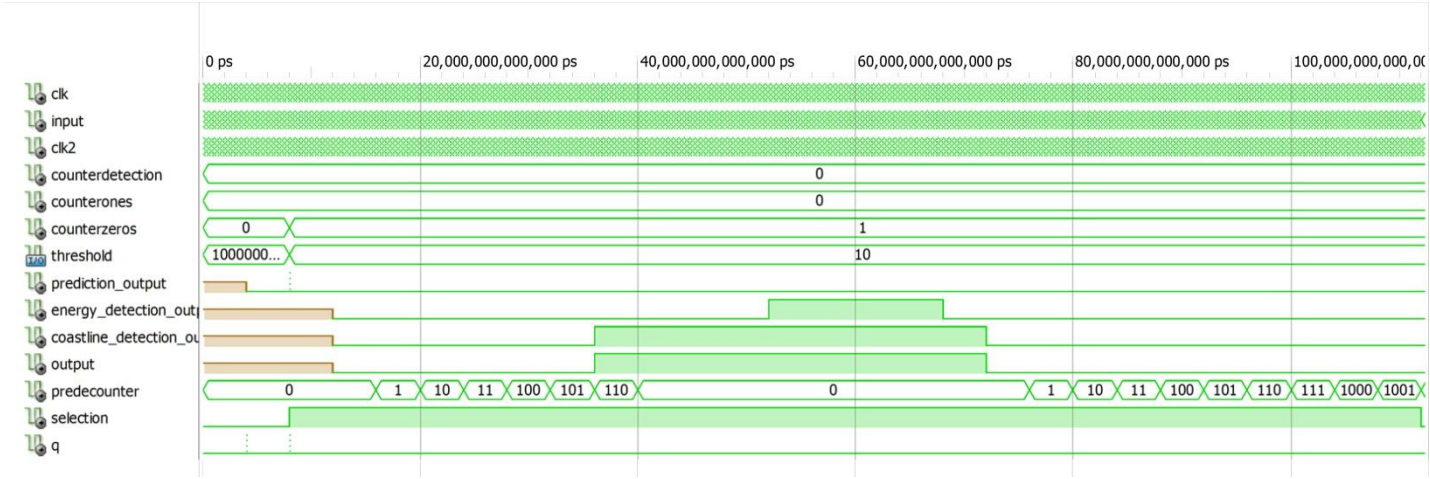


Figure 7.4: Results from ISim Simulation for the full VHDL code

7.4 RTL Schematic

As shown below in Figure 7.5, the RTL schematic of the system generated from the VHDL code resembles the block diagram of the overall system –Prediction with adaptive threshold algorithm followed detection with energy & coastlines feature then decision making block.

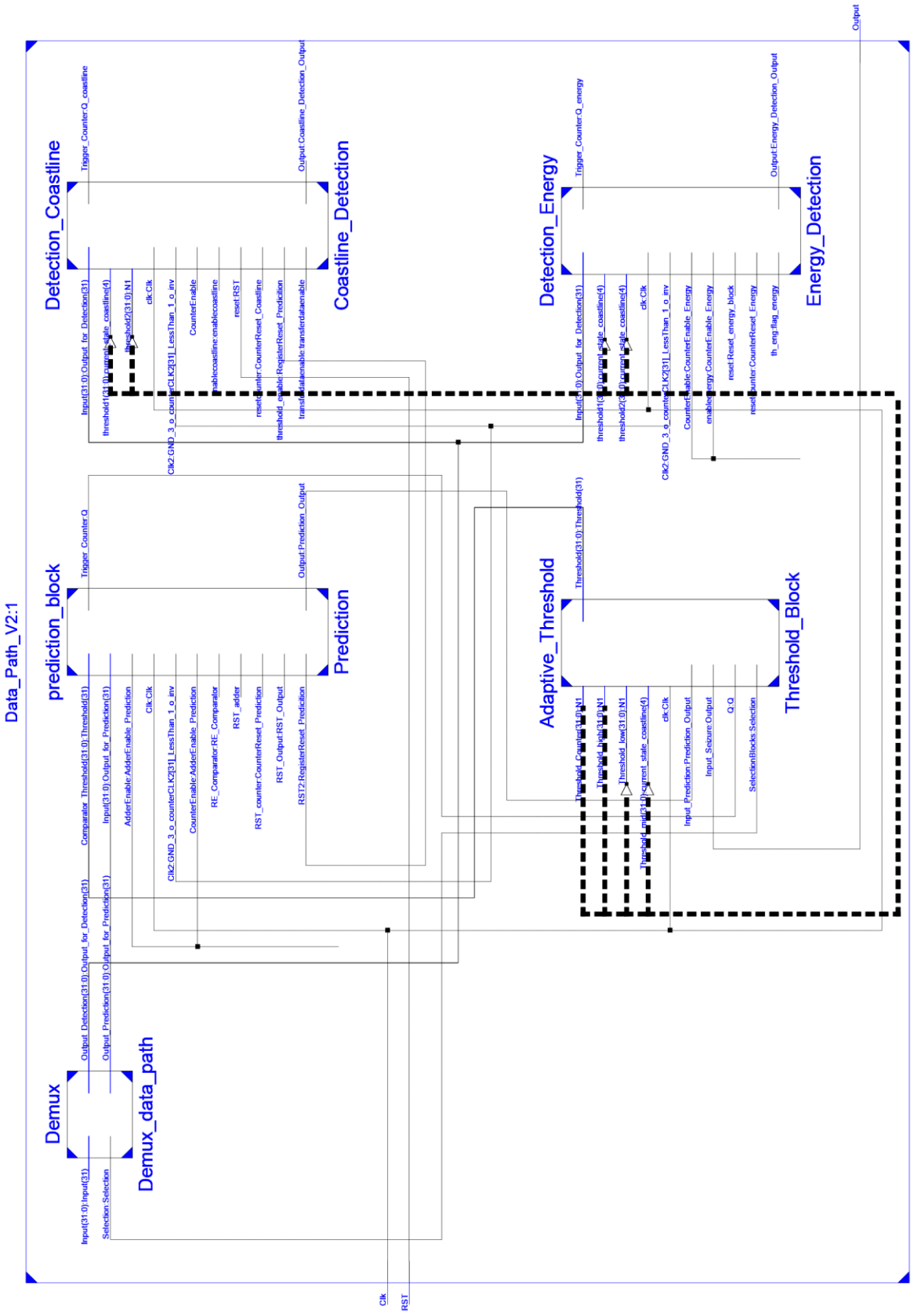


Figure 7.5: RTL schematic of the VHDL code

Chapter 8: FPGA Interface

8.1 Introduction

FPGA stands for “Field Programmable Gate Array”. As you may already know, FPGA essentially is a huge array of gates which can be programmed and reconFigured anytime anywhere. “Huge array of gates” is an oversimplified description of FPGA.

FPGA is indeed much more complex than simple array of gates. Some FPGAs has built in hard blocks such as Memory controllers, high speed communication interfaces, PCIe (Peripheral Component Interconnect Express) Endpoints and many other blocks. But the point is, there are a lot gates inside the FPGA which can be arbitrarily connected together to make a circuit of your choice. More or less like connecting individual logic gate ICs. FPGAs are manufactured by companies like Xilinx, Altera, Microsemi and others.

FPGA is one of the main digital design process stages, used for many purposes such as: customizing SoC (System on Chip) with field upgradable blocks, and ASIC emulation and verification, the latter is the case in our project.

8.2 Procedure

We used Xilinx Spartan 6 in our testing, and here is a small tutorial for how to run the code on FPGA

Step 1: Assign your inputs and outputs to certain switches and leds.

This is done by generating an UCF file by the following steps :

- Project > New Source > Implementation Constraint File
- Open the file.ucf which is added to your project
- Open the hardware user guide to know which pins are corresponding to the required switches and leds
- Assign the inputs and the outputs to the required pins for example if you have an input whose name is (IN1) and you want to enter this input from switch (S2.2) which corresponds to pin (Y6) and you want to show the output whose

name is (OUT1) on (GPIO_LED_1) which corresponds to pin (AB4). You should write the following in the UCF file

```
NET  IN1 LOC="Y6"; NET OUT1 LOC="AB4";
```

Step 2: Configure the clock of your design (If it is needed)

- Add the IPcore of the clock wizard as shown in Figure 8.1 & 8.2

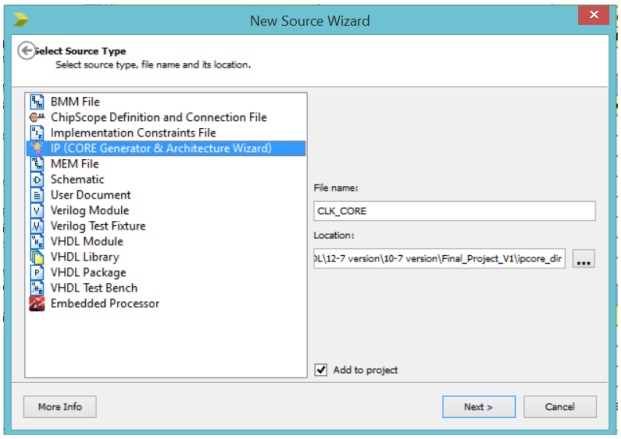


Figure 8.1: Screenshot for the clock wizard in step 2-1

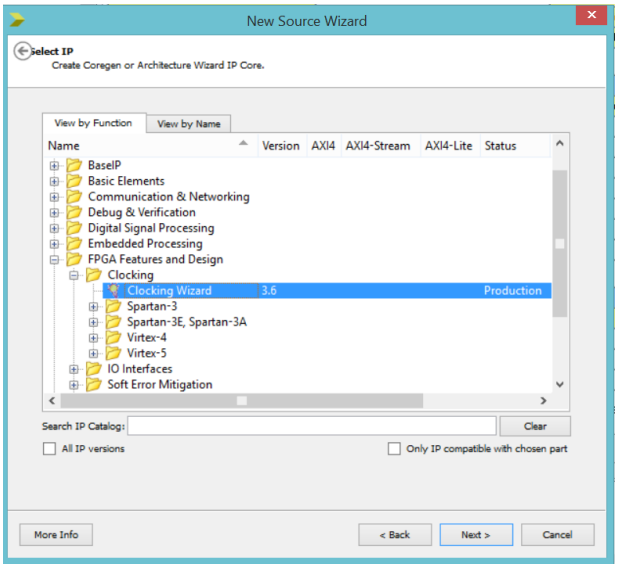


Figure 8.2: Screenshot for the clock wizard in step 2-2

- Input the clock crystal frequency embedded on the FPGA and choose single ended capable pin as shown in Figure 8.3

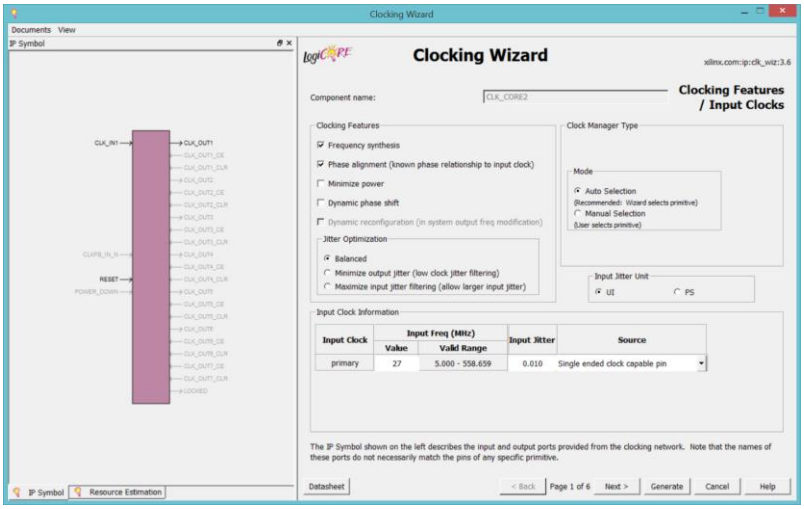


Figure 8.3: Screenshot for the clock wizard core generator

- Then click next and add the output clock you need to be generated by the clock wizard and turn off the rest of the output clock as shown in Figure 8.4

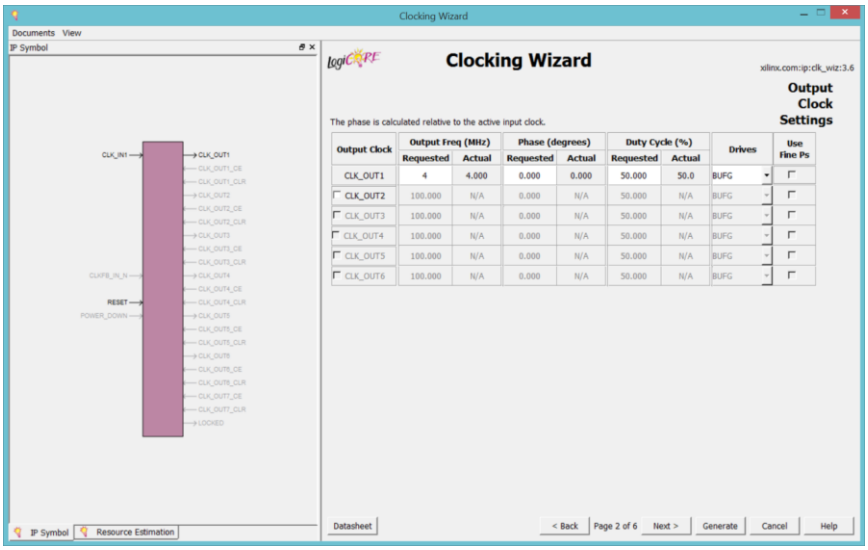


Figure 8.4: Screenshot for the clock wizard core generator while choosing the output frequency

- Uncheck the locked and finally click Generate.
- Then go to the generated clock wizard file (.xco) instance and choose View HDL instantiation Template, select the part shown below and copy it into your code as a component and then copy the other part as an instance of this component as shown in Figure 8.5

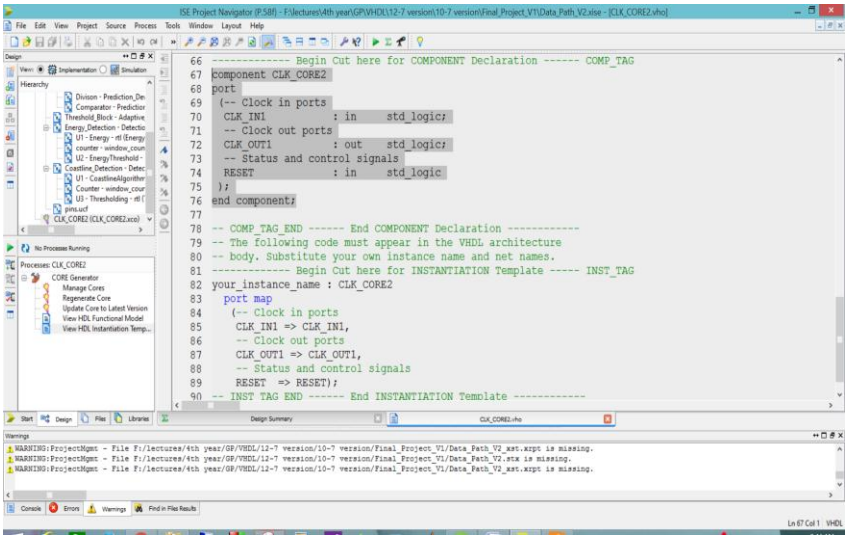


Figure 8.4: Screenshot for HDL instantiation Template

- Then you must define the oscillator pin that you have used in (.ucf). As shown in Figure 8.5 that the single ended clock pin is AB13 so you should write this line in the (.ucf)

```
NET CLK LOC="AB13";
```

Table 1-9: **SP605 Clock Source Connections**

Source	U1 FPGA Pin	Schematic Net Name	Pin Number	Pin Name
U6 200MHZ OSC	K22	SYSCLK_N	5	OUT_B
	K21	SYSCLK_P	4	OUT
X2 27MHZ OSC	AB13	USER_CLOCK	5	OUT
USER_SMA_CLOCK	M19	USER_SMA_CLOCK_N	J38.1	–
SMA Connectors	M20	USER_SMA_CLOCK_P	J41.1	–

Figure 8.5: Screenshot from Spartan-6 I/O Manual

Step 3: Check that (Synthesize, Implement Design and Generate Programming File) are done without any errors as shown in Figure 8.1

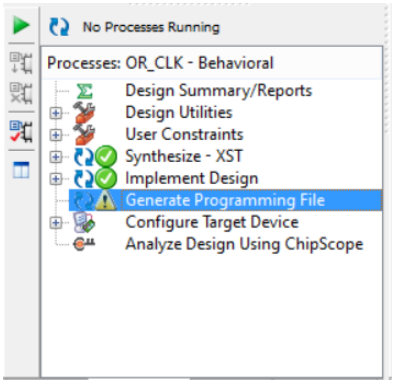


Figure 8.6: Screenshot for the final output of step 3

Step 4: Burn the code on the FPGA and this can be done by the following steps:

- Double Click on ConFigure Target Device, The ISE Impact will be opened.
- Make sure that the **USB_JTAG** cable is connected to the computer.
- In the ISE Impact, Click on Boundary Scan on the top left and then press Ctrl+I to initialize chain as shown in Figure 8.7 and Figure 8.8.

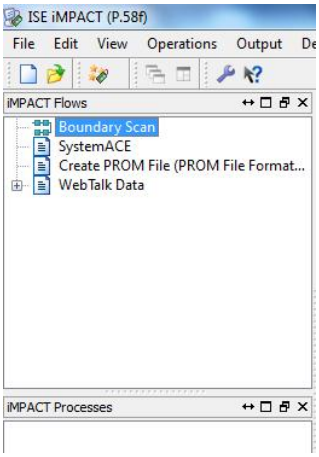


Figure 8.7: Screenshot for the boundary scan

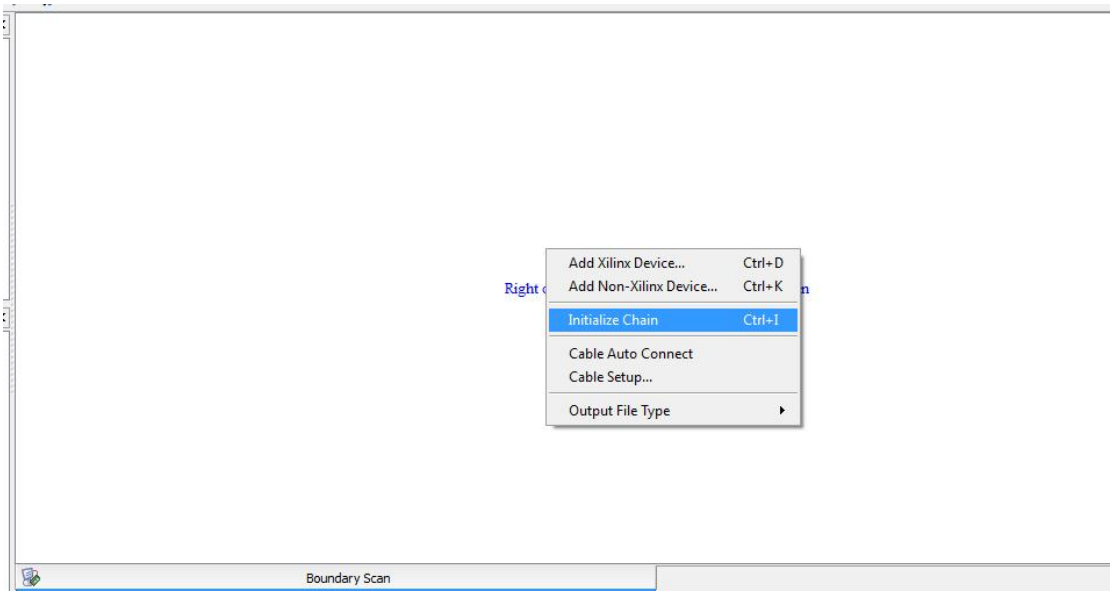


Figure 8.8: Screenshot to initialize the chain

- Now the ISE Impact will find the FPGA
- Double Click on the FPGA icon then add the .bit file which will be found in the directory of your project as shown in Figure 8.9

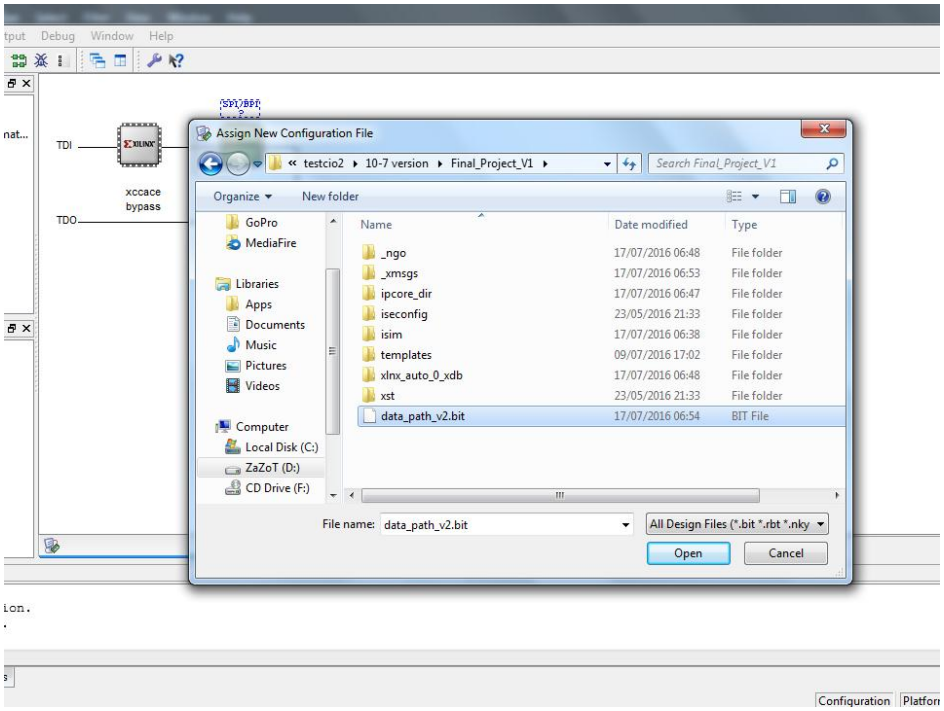


Figure 8.9: Screenshot to load the .bit file

- Right click on the FPGA IC and click program or choose program from the side toolbar as shown in Figure 8.10.

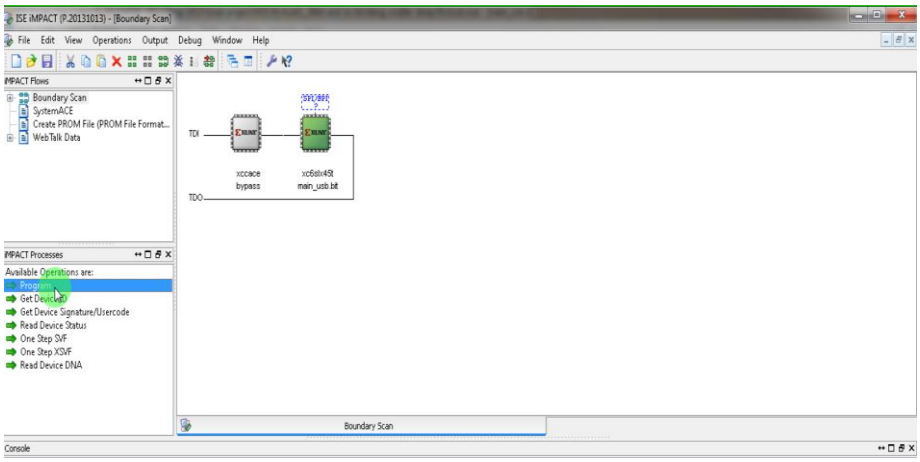


Figure 8.10: Screenshot for programming the .bit file

- Then a message "***Program Succeeded***" is displayed on the screen.

Chapter 9: Conclusion and Next Phase

9.1 Conclusion

Hereby we tried to deliver our humble work all over the past 10 months starting from surveying more than 100 papers to countless run Matlab simulations and written VHDL lines trying to enrich the human race with this simple output just in case someone would discover this is somehow beneficial to help these people who suffer from the symptoms of epileptic seizures.

9.2 Next Phase

The next phase –as we see it- should include two tracks; technical track as converting this proposed design to ASIC design followed by fabrication to test these algorithms on layout level, and a non-technical track by marketing for the system among the biomedical systems companies to get the suitable capital to help completing the research and sustaining the development of this project.

References

Epilepsy & Behavior 37 (2014) 291–307 (Seizure detection, seizure prediction, and closed-loop warning systems in epilepsy)

Methods for Seizure Detection and Prediction: An Overview Giorgos Giannakakis, Vangelis Sakkalis, Matthew Pediaditis, and Manolis Tsiknakis

Comment on 'Interpretation of the Lempel-Ziv Complexity Measure in the context of Biomedical Signal Analysis' Karthi Balasubramanian, Gayathri R Prabhu, Nithin Nagaraj

Florian Mormann, Ralph G. Andrzejak, Christian E. Elger and Klaus Lehnertz, "Seizure prediction: the long and winding road" Brain (2007), 130, 314–333 doi:10.1093/brain/awl241

A rule-based seizure prediction method for focal neocortical epilepsy Ardalan Aarabi and Bin He

Yinxia Liu, Weidong Zhou, Qi Yuan, and Shuangshuang Chen, "Automatic Seizure Detection Using Wavelet Transform and SVM in Long-Term Intracranial EEG" in IEEE Transactions on Neural Systems and Rehabilitation Engineering, Vol. 20, No. 6, November 2012.

Andrew M. White, Philip A. Williams, Damien J. Ferraro, Suzanne Clark, Shilpa D. Kadam, F. Edward Dudek, Kevin J. Staley, "Efficient unsupervised algorithms for the detection of seizures in continuous EEG recordings from rats after brain injury" in Journal of Neuroscience Methods 152 (2006) 255–266.

Shriram Raghunathan, Sumeet K. Gupta, Himanshu S. Markandeya, Kaushik Roy, Pedro P. Irazoqui, "A hardware-algorithm co-design approach to optimize seizure detection algorithms for implantable applications," in Journal of Neuroscience Methods 193 (2010) 106–117.

Goldberger AL, Amaral LAN, Glass L, Hausdorff JM, Ivanov PCh, Mark RG, Mietus JE, Moody GB, Peng C-K, Stanley HE. PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals. Circulation 101(23):e215-e220 [Circulation Electronic Pages; http://circ.ahajournals.org/cgi/content/full/101/23/e215]; 2000 (June 13).

Robert Nau, "Review of basic statistics and the simplest forecasting model: the sample mean" Fuqua School of Business, Duke University, August 2014.

Appendices

Appendix A: Prediction MATLAB Full Code

A.1 Main Code: mainCode.m

```
clc
clear
close all

file_name='chb03_36.edf';
%---------------------------- Preset Values-------------------------
channel=4;                    %Channel number in the .edf file
patient_number =3;
%The patient number according to http://physionet.org/pn6/chbmit/
data=ReadEDF(file_name);     %The data vector

predictionTecnhnique=input('Please enter 1 for Forcasting prediction
technique,2 for Basic Mean prediction technique, or 3 for Adaptive
prediction technique');

while(predictionTecnhnique<1 || predictionTecnhnique>3)
   predictionTecnhnique=input('Wrong Choice. Please enter 1 for
Forcasting prediction technique,2 for Basic Mean prediction
technique, or 3 for Adaptive prediction technique');
end

seconds = 5;%The number of seconds within a window
fs = 256;%The sampling frequency
start_1 =1725;%The start of a seizure within an .edf file
ending_1 =1778;%The end of a seizure within an .edf file
no_of_windows_detection = 15;%If we enter the detection state, we
will stay in that state for 15 windows

%-----------------------------Thresholds----------------------------
threshold_coastline_1=10;%The beta threshold
threshold_coastline_2=2e4;%The alpha threshold
threshold_energy_1=6;%The beta threshold
threshold_energy_2=0.3e4;%The alpha threshold

threshold_prediction_forecasting = 0.9;%Forecasting threshold

threshold_prediction_basic_mean = 1;%Basic mean threshold

thre_pred_low =1;%Adaptive threshold first threshold
thre_pred_mid =2;%Adaptive threshold second threshold
thre_pred_high=3;%Adaptive threshold third threshold


%---------------- Preprocessing ----- DO NOT CHANGE-----------------

data=cell2mat(data(:,channel));
[data,window_length,DataLength] = PreProcessing(data,seconds,fs);

predictionArray = zeros(1,DataLength);
detectionArray = zeros(1,DataLength);
```

```
energyarray = zeros(1,DataLength);
coastlinearray = zeros(1,DataLength);
DecisionMatrix = zeros(1,DataLength/window_length);
not_seizure = data;


%%%if there are more than one seizure, just add more than of the next
%%%variables, seizure_NUMBER is seizure place as samples &
%%%seizure_window_NUMBER is seizure place as windows
seizure_1 = zeros(1,DataLength);
seizure_1((start_1*fs):(ending_1*fs))                                    =
data((start_1*fs):(ending_1*fs));
seizure_window_1 = zeros(1,DataLength/window_length);
seizure_window_1(floor(start_1/seconds):floor(ending_1/seconds))=ones
(1,length(floor(start_1/seconds):floor(ending_1/seconds)));

%--------------Initial values to run the code-----DO NOT CHANGE------
thre_predArray =[thre_pred_low thre_pred_mid thre_pred_high];
meanValue = 0;
counterOnes = 0;
counterZeros= 0;
counterDetection =0;
counterPredictionThreshold=3;
threshold_decision = 0;
prediction_detectionFlag =0;
%This flag is responsible for switching between the detection state
and the prediction state
Decision=0;
final_threshold=thre_pred_mid;
counterCL =0;
counterE=0;
detectionDutyCounter=0;

%------------------Code Core------ DO NOT CHANGE----------------------
j=1;    %A counter that loops over the data vector window by window
i=1;     %A pointer that points at the start of every window within
the data vector
while (j<=DataLength/window_length)

    datawithinwindow = data(i:i+window_length-1);

    %-----------------------Prediction Block-------------------------
------

    if (prediction_detectionFlag == 0)

       if(predictionTecnhnique==1)
            prediction_detectionFlag=
PredictionForecast(datawithinwindow,window_length,threshold_predictio
n_forecasting);
       end

     if(predictionTecnhnique==2)
            [prediction_detectionFlag,meanValue]=
PredictionBasicMean(datawithinwindow,meanValue,threshold_prediction_b
asic_mean);
     end

        if(predictionTecnhnique==3)
```

```
            [prediction_detectionFlag,meanValue,counterOnes,counterZe
            ros,counterDetection,final_threshold] =
            PredictionAdaptive(datawithinwindow,thre_predArray,final_
            threshold,counterPredictionThreshold,counterOnes,counterZ
            eros,counterDetection,meanValue,Decision,prediction_detec
            tionFlag);
        end
         j=j+1;
         i=i+window_length;


%----------------Detection Block-------- DO NOT CHANGE---------------

    elseif (prediction_detectionFlag == 1)
        predictionArray(i)=1000; %1000 just to be visible in stem
        k=1;    %A counter responsible for counting the number of
windows withing the detection state
        z=i;   %A pointer that points at the start of every window
within the data vector
        while((k<=no_of_windows_detection    ||    Decision==1)    &&
j<=DataLength/window_length)
            datawithinwindow = data(z:z+window_length-1);
            EnergyValue =  EnergyAvg(datawithinwindow);
            CoastLineValue = CoastLine(datawithinwindow);
            [Classifier1,counterCL]      =      CoastLineThreshold(
CoastLineValue,threshold_coastline_1,threshold_coastline_2,counterCL)
;
            [Classifier2,counterE]        =        EnergyThreshold(
EnergyValue,threshold_energy_1,threshold_energy_2,counterE);
            detectionDutyCounter=detectionDutyCounter+1;

%--------------Decision Making Block-------- DO NOT CHANGE-----------
            Decision = (Classifier1 || Classifier2);
            if(Decision ==1)
                 detectionArray(z:z+window_length-1) = 1000;
                 %1000 just to be visible in stem
                 DecisionMatrix(j) =1;
                 %The vector that carries the ORed value of the
           Average Energy and the Coastline Classifiers
             end
            k=k+1;
            j=j+1;
            z=z+(seconds*fs);
        end
       prediction_detectionFlag =0;
      i=z;
    end
end


%----------------------Output Handling-----------------------------

stem(predictionArray,'g')
hold on
stem (detectionArray,'k')
hold on
plot(not_seizure,'b')
 hold on
plot(seizure_1,'r')
```

```
legend('Detection Block Trigger','Detected Seizure','Normal iEEG
Data','Seizure Part')
     if(predictionTecnhnique==1)
title(['Prediction using Forecasting Technique for patient number '
num2str(patient_number)])
     elseif(predictionTecnhnique==2)
title(['Prediction using Mean with Basic Threshold Technique for
patient number ' num2str(patient_number)])
     else
title(['Prediction using Adaptive Threshold Technique for patient
number ' num2str(patient_number)])
     end

[TN TP FN FP] = parameters(seizure_window_1,DecisionMatrix);
TP
FP
TN
FN

[DetectionDuty]= detectionDutyCounter*100/(DataLength/window_length);
DetectionDuty
```

A.2 Coastline Function Code: Coastline.m

```
%This function calculates the coastline of a window
function [ CL ] = CoastLine( datawithinwindow )

window_length = length(datawithinwindow);
Dist_bet_2_succsessive=zeros(1,window_length);

for i=2:1:window_length
    Dist_bet_2_succsessive(i-1)=abs(datawithinwindow(i)-
datawithinwindow(i-1));
end

CL=sum(Dist_bet_2_succsessive);

end
```

A.3 Coastline Thresholding Code: CoastlineThreshold.m

```
%This function calculates the multiwindow even based classifier of
coastline
function [ classifier_CL,counter ] = CoastLineThreshold(
CoastLineValue,threshold_coastline_1,threshold_coastline_2,counter )
if CoastLineValue >=threshold_coastline_2
    sez_coastline=1;
else
    sez_coastline=0;
end

if sez_coastline==0
    counter=0;
elseif sez_coastline==1
    counter=counter+1;
end
```

```
if counter>=threshold_coastline_1
    classifier_CL =1;
else
    classifier_CL=0;
end
```

A.4 Average Energy Function Code: EnergyAvg.m

```
%This function calculates the Average Energy of a window
function [ Eavg ] = EnergyAvg( datawithinwindow )

window_length = length(datawithinwindow);
E=datawithinwindow.^2;
Eavg =1/window_length*sum(E);
End
```

A.5 Average Energy Thresholding Code: EnergyThreshold.m

```
%This function calculates the multi window event based classifier of
Average Energy
Function [classifier_E,counter]=
EnergyThreshold(EnergyValue,threshold_energy_1,threshold_energy_2,cou
nter )

if EnergyValue >=threshold_energy_2
    sez_energy=1;
else
    sez_energy=0;
end

if sez_energy==0
    counter=0;
elseif sez_energy==1
    counter=counter+1;
end

if counter>=threshold_energy_1
    classifier_E =1;
else
    classifier_E=0;

end
```

A.6 Efficiency Parameters Function Code: parameters.m

```
%This function calculates the True Positives, The True Negatives, The
False
%Positives, and The False Negatives.
function [TN TP FN FP] =
parameters(original_signal,calculated_signal)
    TN = 0;
    TP = 0;
    FN = 0;
    FP = 0;
```

```
    length_original = length(original_signal);
    length_calculated = length(calculated_signal);
    min_length = min(length_calculated,length_original);
    for i=1:1:min_length
        if(original_signal(i) == calculated_signal(i))
            if(original_signal(i) == 1)
                TP = TP + 1;
            else
                TN = TN + 1;
            end
        else
            if((original_signal(i) == 0)&&(calculated_signal(i)==1))
                FP = FP + 1 ;
            else
                FN = FN + 1;
            end
        end
    end
end
```

A.7 Adaptive Prediction Function Code: PredictionAdaptive.m

```
%This function calculates the adaptive prediction
function
[prediction,meanvalue,counterOnes,counterZeros,counterDetection,final
_threshold] =
PredictionAdaptive(datawithinwindow,thre_predArray,final_threshold,th
reshold2,counterOnes,counterZeros,counterDetection,oldmean,DetectionD
ecision,predictionflag)

meanvalue = mean(abs(datawithinwindow));

if (predictionflag ==1 && DetectionDecision ==0)
    counterDetection = counterDetection+1;

elseif(predictionflag ==1 && DetectionDecision ==1)
    counterDetection =0;
    counterZeros=0;
    counterOnes=0;
end

if ((meanvalue/oldmean>final_threshold))

    prediction=1;
    counterOnes=counterOnes+1;
    counterZeros=0;
else
    prediction =0;
    counterZeros=counterZeros+1;
    counterOnes=0;
end

if counterZeros>=threshold2
    counterOnes=0;
    counterZeros=0;
    counterDetection =0;
    threshold_decision=1;
```

```
elseif counterOnes>=threshold2 && counterDetection>=threshold2
    counterOnes=0;
    counterZeros=0;
    counterDetection =0;
    threshold_decision =-1;

else threshold_decision=0;
end

switch threshold_decision
    case -1
        if(final_threshold == thre_predArray(3))
            final_threshold = thre_predArray(2);
        else final_threshold = thre_predArray(1);
        end
    case 1
        if(final_threshold == thre_predArray(1))
            final_threshold = thre_predArray(2);
        else final_threshold = thre_predArray(3);
        end
    case 0
        final_threshold=final_threshold;
    otherwise
        final_threshold = thre_predArray(2);
end
end
```

A.8 Basic Mean Function Code: PredictionBasicMean.m

```
%This function calculates the basic mean prediction
  function [prediction,meanvalue] =
PredictionBasicMean(datawithinwindow,oldmean,threshold)

meanvalue = mean(abs(datawithinwindow));

if ((meanvalue/oldmean>threshold))

    prediction=1;

else
    prediction =0;

end
```

A.9 Forecast Technique Function Code: PredictionForecast.m

```
%This function calculates the forecasting prediction
function [ prediction ] = PredictionForecast(
data,window_length,threshold)

prediction = 0;
k=1;
for i=1:(window_length)/4:window_length,
    quawindow=data(i:i+window_length/4-1);
    meanvalue(k) = mean(quawindow);
    S(k) = std(quawindow);
    newsample1(i:i+window_length/4-1) =meanvalue(k)+(2*S(k));
    newsample2(i:i+window_length/4-1) =meanvalue(k)-(2*S(k));
```

```
    k=k+1;
end
    oldquart1 = newsample1(i:i+window_length/4-1);
    oldquart2 = newsample2(i:i+window_length/4-1);

newsample1 = newsample1(1:3*window_length/4);
newsample2 = newsample2(1:3*window_length/4);
datanew = data(window_length/4+1:window_length);

sim0 = max(abs(xcorr(datanew,datanew)));
sim1 = max(abs(xcorr(newsample1,datanew)));
sim2 = max(abs(xcorr(newsample2,datanew)));

    if(sim1 > sim2)
        simf =sim1;
    else simf =sim2;
    end

    if (simf/sim0 >= threshold)
        prediction =0;

    else prediction =1;
    end
```

A.10 Pre-processing Function Code: PreProcessing.m

```
%This function calculates window length, and the data vector length.
function [data_filtered, window_length,DataLength ] = PreProcessing(
data,number_of_seconds,fs )

window_length = number_of_seconds*fs;
DataLength = window_length*floor(length(data)/(window_length));
data_filtered=data;
end
```

Appendix B: Detection MATLAB Codes

B.1 Coastline Feature

```
abs_bet_2_succsessive=zeros(length(data),1);%This  vector  will  have
the absolute difference between two successive EEG data points
k=1;%The window number
for i=2:1:length(data)
    if i+N-1>length(data)
        break
    end
    abs_bet_2_succsessive(k)=abs(data(i)-data(i-1));
    k=k+1;
end

CL=zeros(1,length(data)/N);%coastline vector
k=1;
i=1;
for i=1:N:length(abs_bet_2_succsessive)
  if i+N-1>length(abs_bet_2_succsessive)
    break
end
CL(k)=sum(abs_bet_2_succsessive(i:i+N-1));
k=k+1;
end
```

B.2 Coastline Thresholding

```
%The alpha Thresholding

threshold_coastline_1=40000;%The Alpha threshold

for i=1:length(CL)
    if CL(i)>=threshold_coastline_1
        sez_coastline(i)=1;
    else
        sez_coastline(i)=0;
    end
end

%The Beta Thresholding

threshold_coastline_2=5;%The Beta threshold

counter=0;%If this variable is more than or equal Beta then we have a
seizure event else otherwise
for i=1:length(sez_coastline)
     if sez_coastline(i)==0
        counter=0;
        classifier_CL(i)=0; %classifier_CL variable is the output of
the %multi-window event classifier
        continue
     end

     if sez_coastline(i)==1
     counter=counter+1;
     end
     if counter>=threshold_coastline_2
```

```
        classifier_CL(i-counter+1:i)=ones(1,length(i-counter+1:i));
    end
end
```

B.3 Average Energy Feature

```
E=data.^2;
k=1;
for i1=1:N:length(data)
    if i1+N-1 >length(E)
        break;
    end
    Eavg(k)=1/N*sum(E(i1:i1+N-1));%This is the Average Energy Vector
    k=k+1;
end
```

B.4 Average Energy Thresholding

```
threshold_energy_1=2000;  %The Alpha threshold

%The Alpha Thresholding
for i=1:length(Eavg)
    if Eavg(i)>=threshold_energy_1
        sez_E(i)=1;
    else
        sez_E(i)=0;
    end
end

%The Beta Thresholding

threshold_energy_2=12;%The Beta threshold

counter=0;%If this variable is more than or equal Beta then we have a
seizure event else otherwise
for i=1:length(sez_E)
    if sez_E(i)==0
       counter=0;
       classifier_E(i)=0; ;%classifier_E vector is the output of the
multi-window event %based classifier
       continue
    end
    if sez_E(i)==1
    counter=counter+1;
    end
    if counter>=threshold_energy_2
        classifier_E(i-counter+1:i)=ones(1,length(i-counter+1:i));
    end
end
```

B.5 Final Decision

```
EavgORCL=classifier_CL | classifier_E;
%This vector is the ORing between the %coastline classifier and the
Average eneregy classifier
```

B.6 Actual Seizure Locations Vector

```
seconds=4;%The number of seconds in each window
start=362;%Start of the seizure in seconds
ending=414;%End of seizure in seconds

sez_true=zeros(1,length(classifier_E));
%This vector is the actual locations %of the seizure based on the
summary of the pateint. Each point in this %vector contains
information about the whole window

sez_true(floor(start/seconds):floor(ending/seconds))=ones(1,length(fl
oor(start/seconds):floor(ending/seconds)));
```

B.7 Plotting The Results

```
Figure
subplot(6,1,1)
stem(CL(40:140))
title('CL')

subplot(6,1,3)
stem(classifier_CL(40:140))
title('CL Classifier')

subplot(6,1,2)
stem(Eavg(40:140))
title('Eavg')

subplot(6,1,4)
stem(classifier_E(40:140))
title('Eavg Classifier')

subplot(6,1,5)
stem(EavgORCL(40:140))
title('EavgORCL')

subplot(6,1,6)
stem(sez_true(40:140))
title('True Siezure Locations')
```

Appendix C: MATLAB Codes for Wavelet Domain Features & SVM

C.1 Partitioning the data into epochs based on N

```
k=1;
for i=1:N:length(data)
    if i+N-1 >length(data)
        break;
    end
    partitonedData(k,:)=data(i:i+N-1);%This matrix contains the
partitioned data
    k=k+1;
end
```

C.2 Pre-requirements to calculate the wavelet features

Please read the documentation of the following MATLAB functions:
wavedec
detcoef

The code:

```
fs=256;%The sampling frequency
for i=1:length(partitonedData(:,1))
    [C,L]=wavedec(transpose(partitonedData(i,:)),5,'db4');
    [D1,D2,D3,D4,D5]=detcoef(C,L,[1,2,3,4,5]);
    Length=length(D1);
    D=zeros(Length,5);
    D(:,1)=D1+D(:,1);
    D(1:length(D2),2)=D2+D(1:length(D2),2);
    D(1:length(D3),3)=D3+D(1:length(D3),3);
    D(1:length(D4),4)=D4+D(1:length(D4),4);
    D(1:length(D5),5)=D5+D(1:length(D5),5);
    T=1/fs;
```

C.3 Calculating the Fluctuation Index Feature

```
    temp=zeros(length(D1),1);
    for i2=1:4
        temp=temp+abs(D(:,i2+1)-D(:,i2));
    end
    FI(i,:)=(1/5).*temp; %The Fluctutation index matrix
```

C.4 Calculating the Coefficient of variation Feature

```
mean=zeros(length(D1),1);
for i4=1:5
mean=mean+D(:,i4);
end
temp=zeros(length(D1),1);
for i4=1:5
    temp=temp+(D(:,i4)-mean).^2;
end
temp2=(1/5).*temp;
sigma=sqrt(temp2);
CoeffVar(i,:)=(sigma.^2)./(mean.^2);%the coefficient of variation
matrix
```

C.4 Calculating the Relative Energy Feature

```
E=D.^2 *T/5;
Etot=E(:,1)+E(:,2)+E(:,3)+E(:,4)+E(:,5);
sum=0;
for i3=1:length(Etot)
sum=Etot(i3)+sum;
end
ER(i,:)=Etot./sum;%the relative energy matrix

end% end of for loop whose counter is I (the most outer loop)
```

C.5 Support Vector Machine (SVM)

It is worth mentioning that you can train/test any set of features, but here we will use the Average Energy and the Coastline features as an example.

C.6 SVM Training

First, look up the `svmtrain` function in MATLAB documentation. You could train the SVM with any length of features. However, the longer the length of the training data, the smarter the SVM is.

```
trainingData=[Eavg' CL'];%If you you want to train the time domain
features uncomment this and comment the previous line

svmTrain=svmtrain(trainingData, sez_true);%The output structure of
svmtrain function
```

C.7 SVM Testing

Here you should first load a new file (`chb03_05.edf` for example), then you have to calculate the same features you trained in order to classify and test them.

```
 svmTest=[Eavg' CL'];%this vector is different from the trainingData
vector. %it comes from the new loaded file.

svmClassification=svmclassify(svmTrain,svmTest);%The output vector
that %represents the SVM classifier
```

Appendix D: EEG Data Signal Reading MATLAB Code

D.1 Loading Data into MATLAB environment

```
%Here we are using the file "chb03_01.edf"
channel=4;%The Channel number in the edf file
data=ReadEDF('chb03_01.edf');
data=cell2mat(data(:,channel));%Now data variable is a vector which has the raw data of the EEG signal of channel number 4 for the file chb03_01.edf
```

D.2 ReadEDF.m Function

```
function [data, header] = readEDF(filename)

fid = fopen(filename,'r','ieee-le');

%%% HEADER LOAD
% PART1: (GENERAL)
hdr = char(fread(fid,256,'uchar')');
header.ver=str2num(hdr(1:8));                % 8 ascii : version of this data format (0)
header.patientID  = char(hdr(9:88));         % 80 ascii : local patient identification
header.recordID  = char(hdr(89:168));        % 80 ascii : local recording identification
header.startdate=char(hdr(169:176));         % 8 ascii : startdate of recording (dd.mm.yy)
header.starttime   = char(hdr(177:184));     % 8 ascii : starttime of recording (hh.mm.ss)
header.length = str2num (hdr(185:192));      % 8 ascii : number of bytes in header record
reserved = hdr(193:236); % [EDF+C       ] % 44 ascii : reserved
header.records = str2num (hdr(237:244)); % 8 ascii : number of data records (-1 if unknown)
header.duration = str2num (hdr(245:252)); % 8 ascii : duration of a data record, in seconds
header.channels  =  str2num  (hdr(253:256));%  4  ascii  :  number  of signals (ns) in data record

%%%% PART2 (DEPENDS ON QUANTITY OF CHANNELS)

header.labels=cellstr(char(fread(fid,[16,header.channels],'char')'));
% ns * 16 ascii : ns * label (e.g. EEG FpzCz or Body temp)
header.transducer
=cellstr(char(fread(fid,[80,header.channels],'char')'));  %  ns  *  80 ascii : ns * transducer type (e.g. AgAgCl electrode)
header.units = cellstr(char(fread(fid,[8,header.channels],'char')'));
% ns * 8 ascii : ns * physical dimension (e.g. uV or degreeC)
header.physmin                                                    =
str2num(char(fread(fid,[8,header.channels],'char')')); % ns * 8 ascii : ns * physical minimum (e.g. -500 or 34)
header.physmax                                                    =
str2num(char(fread(fid,[8,header.channels],'char')')); % ns * 8 ascii : ns * physical maximum (e.g. 500 or 40)
```

```
header.digmin                                                          =
str2num(char(fread(fid,[8,header.channels],'char')')); % ns * 8 ascii
: ns * digital minimum (e.g. -2048)
header.digmax                                                          =
str2num(char(fread(fid,[8,header.channels],'char')')); % ns * 8 ascii
: ns * digital maximum (e.g. 2047)
header.prefilt
=cellstr(char(fread(fid,[80,header.channels],'char')')); % ns * 80
ascii : ns * prefiltering (e.g. HP:0.1Hz LP:75Hz)
header.samplerate                                                      =
str2num(char(fread(fid,[8,header.channels],'char')')); % ns * 8 ascii
: ns * nr of samples in each data record
reserved = char(fread(fid,[32,header.channels],'char')'); % ns * 32
ascii : ns * reserved


f1=find(cellfun('isempty', regexp(header.labels, 'EDF Annotations',
'once'))==0); % Channels number with the EDF Annotations
f2=find(cellfun('isempty',      regexp(header.labels,      'Status',
'once'))==0); % Channels number with the EDF Annotations
f=[f1(:); f2(:)];
%%%%%% PART 3: Loading of signals

%Structure of the data in format EDF:

%[block1 block2 .. , block N], where N=header.records
% Block structure:
% [(d seconds of 1 channel) (d seconds of 2 channel) ... (d seconds
of Ñh channel)], Where Ñh - quantity of channels, d - duration of the
block
% Ch = header.channels
% d = header.duration

Ch_data = fread(fid,'int16'); % Loading of signals


fclose(fid); % close a file

%%%%% PART 4: Transformation of the data
if header.records<0, % If the quantity of blocks is not known
R=sum(header.duration*header.samplerate); % Length of one block
header.records=fix(length(Ch_data)./R); % Quantity of written down
blocks
end

% Separating a read signal into blocks
Ch_data=reshape(Ch_data, [], header.records);

% establishing calibration parametres

sf    =    (header.physmax    -    header.physmin)./(header.digmax    -
header.digmin);
dc = header.physmax - sf.* header.digmax;

data=cell(1, header.channels);
Rs=cumsum([1; header.duration*header.samplerate]); % ñòðîêà èíäåêñîâ
ïîäáëîêîâ êàíàëîâ Rs(k):Rs(k+1)-1

% separating of signals of everyone the channel from blocks
```

```
% and recording of signals in structure of cells

for k=1:header.channels

data{k}=reshape(Ch_data(Rs(k):Rs(k+1)-1, :), [], 1);
if sum(k==f)==0 % non Ànnotation
% Calibration of the data
data{k}=data{k}.*sf(k)+dc(k);
end
end

% PART 5: ANNOTATION READ
    header.annotation.event={};
    header.annotation.starttime=[];
    header.annotation.duration=[];
    header.annotation.data={};

if sum(f)>0

try

for p1=1:length(f)
Annt=char(typecast(int16(data{f(p1)}), 'uint8'))';

% separate of annotation on blocks
Annt=buffer(Annt, header.samplerate(f(p1)).*2, 0)';
ANsize=size(Annt);
    for p2=1:ANsize(1)
   % search TALs starttime
    Annt1=Annt(p2, :);
    Tstart=regexp(Annt1, '+');
    Tstart=[Tstart(2:end) ANsize(2)];

    for p3=1:length(Tstart)-1
   A=Annt1(Tstart(p3):Tstart(p3+1)-1); % TALs block
   header.annotation.data={header.annotation.data{:} A};

      % duration and starttime TALs
       Tds=find(A==20 | A==21);
        if length(Tds)>2
            td=str2num(A(Tds(1)+1:Tds(2)-1));
            if isempty(td), td=0; end
           header.annotation.duration=[header.annotation.duration(:);
td];

header.annotation.starttime=[header.annotation.starttime(:);
str2num(A(2:Tds(1)-1))];
           header.annotation.event={header.annotation.event{:}
A(Tds(2)+1:Tds(end)-1)};
         else
           header.annotation.duration=[header.annotation.duration(:);
0];

header.annotation.starttime=[header.annotation.starttime(:);
str2num(A(2:Tds(1)-1))];
           header.annotation.event={header.annotation.event{:}
A(Tds(1)+1:Tds(end)-1)};
        end
    end
```

```
    end
end

% delete annotation
a=find(cell2mat(cellfun(@length,              header.annotation.event,
'UniformOutput', false))==0);
header.annotation.event(a)=[];
header.annotation.starttime(a)=[];
header.annotation.duration(a)=[];

end
end

header.samplerate(f)=[];
header.channels=header.channels-length(f);
header.labels(f)=[];
header.transducer(f)=[];
header.units(f)=[];
header.physmin(f)=[];
header.physmax(f)=[];
header.digmin(f)=[];
header.digmax(f)=[];
header.prefilt(f)=[];
data(f)=[];
```

D.3 Downloading Patients' Data

You can find real patients' data on this web site: http://physionet.org/pn6/chbmit/

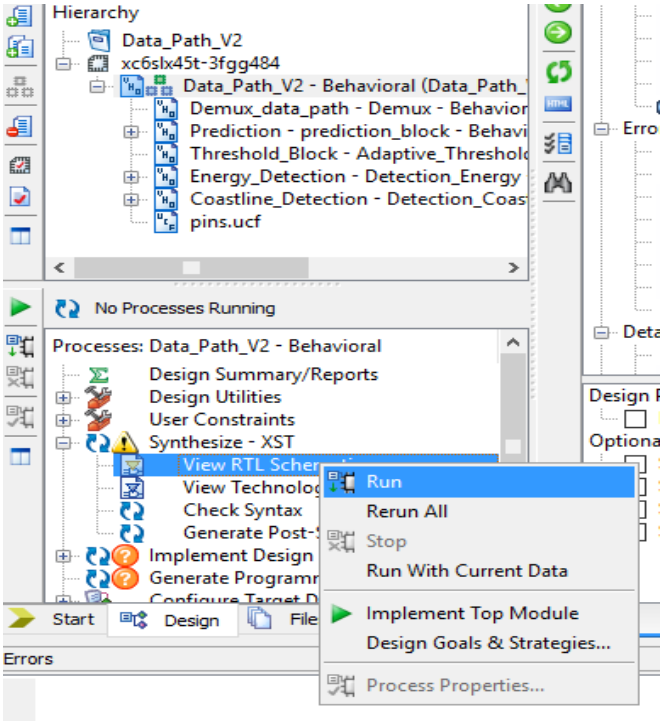
You will find the data of a number of patients where each patient is given a name chbXX, i.e: the folder chb01 belongs to the first patient. Once you open the patient's folder you will find the ".edf" data (The EEG signal).

You will also find a summary for each patient -chbXX-summary.txt for in the folder of the first patient-and you will find a lot of information here such as the .edf files that contain the seizure and the moment of occurrence of that seizure.

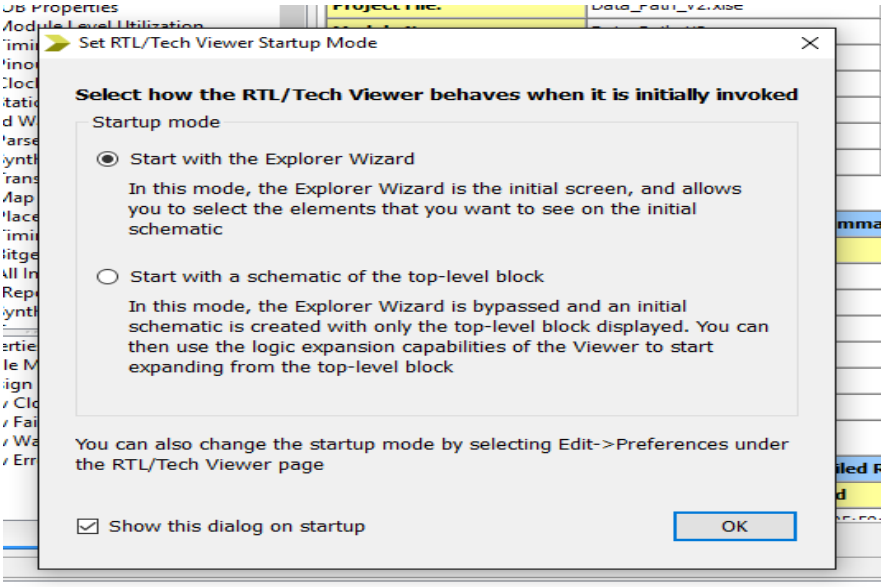
Appendix E: Generating the full RTL Schematic from ISE project

After the HDL synthesis phase of the synthesis process, you can display a schematic representation of your synthesized source file. This schematic shows a representation of the pre-optimized design in terms of generic symbols, such as adders, multipliers, counters, AND gates, and OR gates, that are independent of the targeted Xilinx® device. Viewing this schematic may help you discover design issues early in the design process. This can be done simply using ISE with the following steps

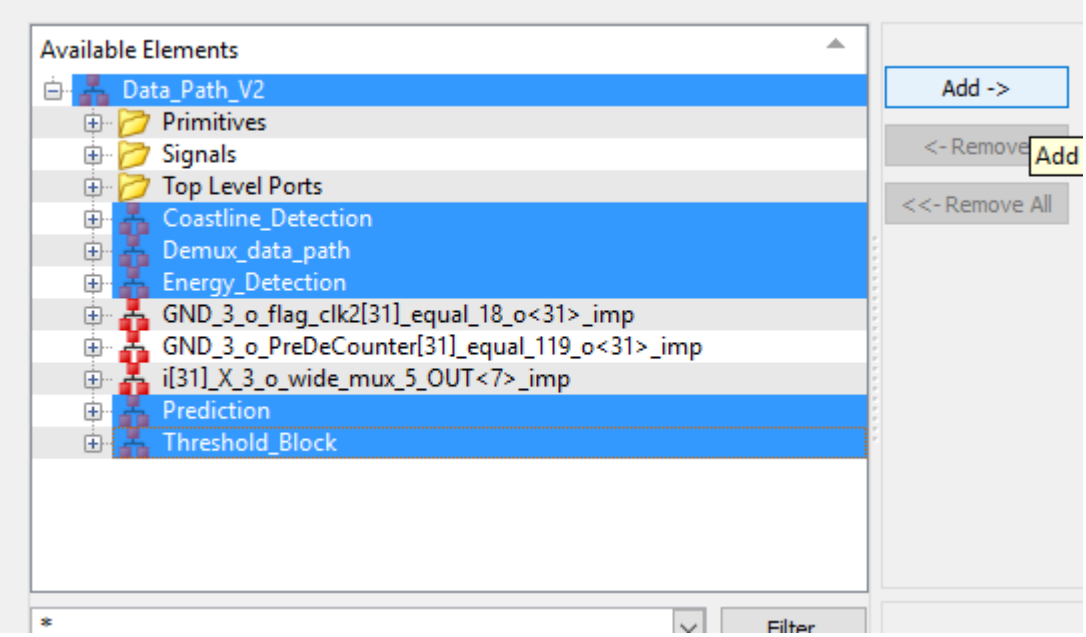
Step1: Right Click on (View RTL Schematic) from the synthesize tab then press Run.

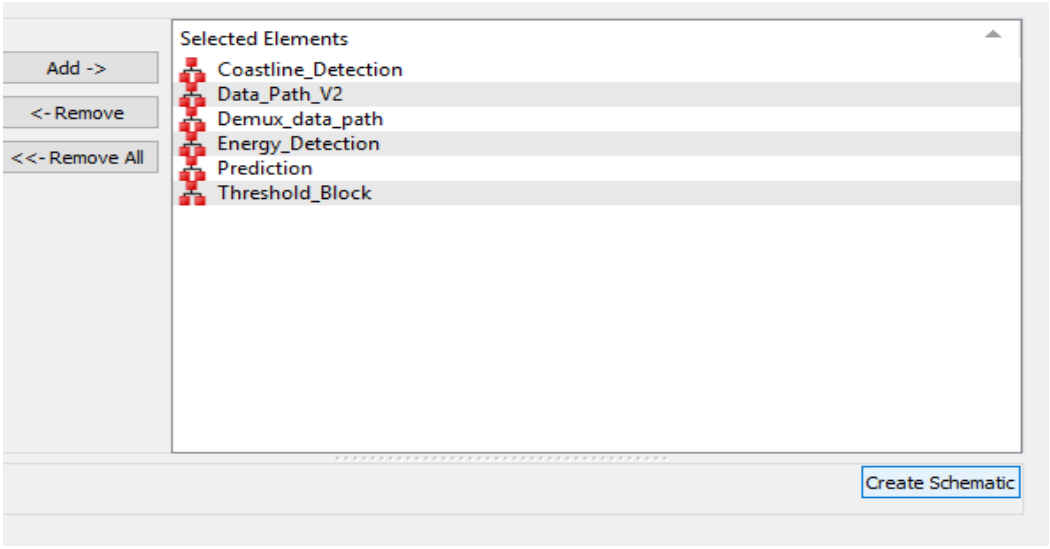


Step2: Choose Start with the Explorer Wizard

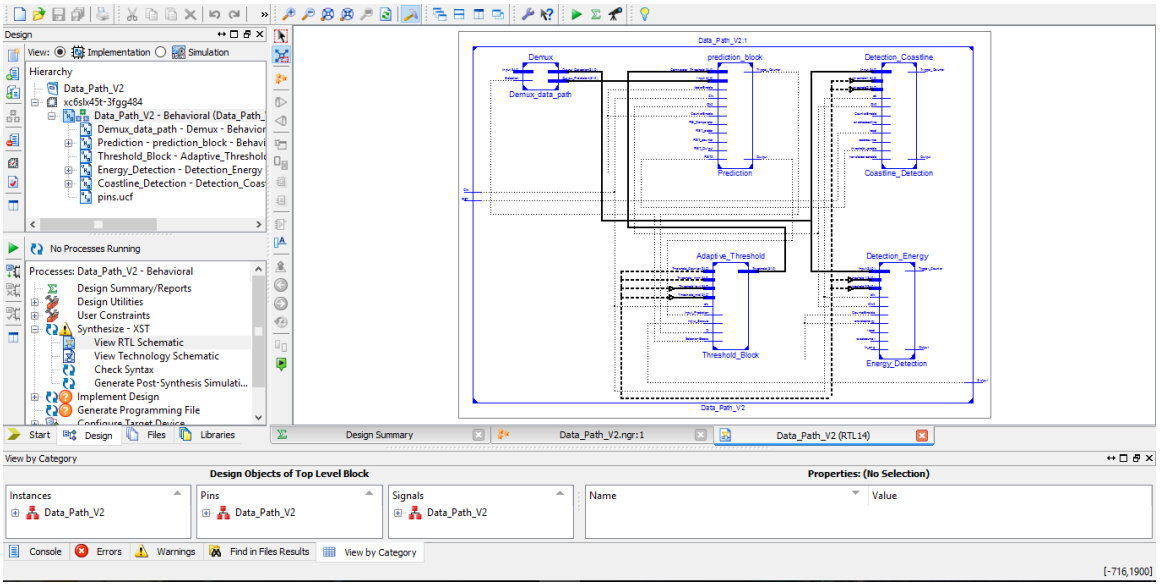


Step 3: Add all the blocks that you want to be shown





Now you will have the RTL Schematic, so that you can understand the codes easily by tracking the signals to the blocks. You can also show the components of any block by right click on the block then click on show block contents.



Appendix F: Full VHDL Code

F.1 Data_Path_TB (Test Bench Code)

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   08:53:43 05/10/2016
-- Design Name:
-- Module Name:
E:/Xilinx_ISE/14.5/projects/Data_Path_V2/Data_Path_V2_TB.vhd
-- Project Name:  Data_Path_V2
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: Data_Path_V2
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types
std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx
recommends
-- that these types always be used for the top-level I/O of a design
in order
-- to guarantee that the testbench will bind correctly to the post-
implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use ieee.std_logic_textio.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY Data_Path_V2_TB IS
END Data_Path_V2_TB;

ARCHITECTURE behavior OF Data_Path_V2_TB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Data_Path_V2
    PORT(
         Clk : IN  std_logic;
         RST : IN  std_logic;
            Output :inout STD_LOGIC
        );
    END COMPONENT;
```

```
   --Inputs
   signal Clk : std_logic := '0';
   signal RST : std_logic := '0';
    signal output: std_logic := '0';

   -- Clock period definitions close to 27MHz
   constant Clk_period : time := 37 ns;


BEGIN
    -- Instantiate the Unit Under Test (UUT)
   uut: Data_Path_V2 PORT MAP (
          Clk => Clk,
          RST => RST,
             Output => Output
        );

   -- Clock process definitions
   Clk_process :process
   begin
        Clk <= '0';
        wait for Clk_period/2;
        Clk <= '1';
        wait for Clk_period/2;
   end process;

END;
```

F.2 Data_Path

```
----------------------------------------------------------------------------
-------------
-- Company:
-- Engineer: Ahmed Yasser - Al Moataz Bellah Refeat - Taha Shawky -
Kareem Ayman - Mohamed Mahmoud Kamal - Mohamed Moustafa Abd El Rahman
--
-- Create Date:    08:46:12 05/10/2016
-- Design Name:
-- Module Name:    Data_Path_V2 - Behavioral
-- Project Name:     Seizure detector and predictor
-- Target Devices:
-- Tool versions:
-- Description:
--              This file(module) contains our whole algorithm (Demux
, Prediction, Detection& finite state machine (FSM)).
--              All branches are connected together using port
mapping.
--              First, the data(sample) enteres the Demux.Initially,
the Demux passes the data to the prediction block.
--              If the prediction block sets the output to 0, the
selection remains 0 and predictor still works,
--              while if the prediction block sets the output to 1,
the selection line to the Demux becomes 1 and
--              Detection block is turned on.The data then passed
from the Demux to the Detection block.
--              The detection alogrithm begins determining if there
is a seizure or not.
--              If the detector states that there is no seizure for
10 windows the selection becomes 0 and the predictor
--              turns on again and the detector goes off.
```

```
--              If the detector states that this window is a seizure,
the stimulator starts, then the detection continues
--              work again till the seizure ends and the next 10
windows pass without seizure.after that the predictor
--              starts to work again and so on.
--              The FSM is the responsible for the signals that are
sent to all the blocks.
--              The FSM consists of 4 partitions:
--                  1- Prediction
--                  2- Detection - Energy
--                  3- Detection - Coastline
--                  4- Decision making
--
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Data_Path_V2 is
    Port (
              Clk : in STD_LOGIC;
              RST : in STD_LOGIC;
              Output: inout STD_LOGIC :='0'
             );
end Data_Path_V2;

architecture Behavioral of Data_Path_V2 is


constant energy_threshold1 : integer := 10;  --
Counter_Threshold_Energy
constant energy_threshold2 : integer := 2000; --
Amplitude_Threshold_Energy
constant coastline_threshold1 : integer := 15000; --
Amplitude_Threshold_Coastline
constant coastline_threshold2 : integer := 5; --
Counter_Threshold_Coastline
constant Threshold_Counter : integer := 3; --Pediction Counter
Threshold
constant Threshold_low : integer := 1; -- Low prediction Threshold
constant Threshold_mid : integer := 2; -- Medium prediction Threshold
constant Threshold_high : integer := 3; -- High prediction Threshold


component prediction_block is
    Port ( Input : in  integer;
              Clk : in STD_LOGIC;
              Clk2 : in STD_LOGIC;
              RST_counter : in STD_LOGIC;
              RST_adder : in STD_LOGIC;
              RST2 : in STD_LOGIC;
              AdderEnable : in STD_LOGIC; -- to enable summation
process
```

```
              CounterEnable : in STD_LOGIC; -- to enable counter to
count
              RE_Comparator : in  STD_LOGIC; -- to enable data to
comes out from the reg to the comparator
              Comparator_Threshold : in integer;
              Trigger_Counter : out STD_LOGIC;
              RST_Output : in STD_LOGIC;
           Output : inout  STD_LOGIC);
end component;

component Demux is
Port (
    Input : in integer;
    Selection : in STD_LOGIC;
    Output_Prediction : out integer;
    Output_Detection : out integer
    );
end component;

component Detection_Energy is
port (

Input:in integer;
Output:inout std_logic;
clk: in std_logic;
Clk2 : in STD_LOGIC;
enableenergy: in std_logic;
th_eng:in std_logic;
CounterEnable : in std_logic;
threshold1: in integer;
threshold2: in integer;
reset: in std_logic;
resetcounter: in std_logic;
Trigger_Counter : out STD_LOGIC
);
end component ;

component Detection_Coastline is
port (

Input:in integer;
Output:inout std_logic;
clk: in std_logic;
Clk2 : in STD_LOGIC;
transferdataenable: in std_logic;
threshold_enable : in STD_LOGIC;
enablecoastline: in std_logic;
CounterEnable: in std_logic;
threshold1: in integer;
threshold2: in integer;
reset: in std_logic;
resetcounter: in std_logic;
Trigger_Counter : out STD_LOGIC
);

end component;

component Adaptive_Threshold is
    Port ( clk:in STD_LOGIC;
              Q:in STD_LOGIC;
```

```
              Input_Prediction : in  STD_LOGIC;
             Input_Seizure : in  STD_LOGIC;
                SelectionBlocks : in STD_LOGIC;
             Threshold : inout  integer := Threshold_mid;
                Threshold_Counter : in  integer;
             Threshold_low : in  integer;
             Threshold_mid : in  integer;
             Threshold_high : in  integer);
end component;

-- If you want to add more inputs from the text file conveted from
Matlab
-- chang the size of input array to the size that you want
-- and paste the input array below

TYPE Input_Array IS ARRAY (1 to 10) OF INTEGER;
CONSTANT InArray : Input_Array := (
-13 ,
-13 ,
-13 ,
-8 ,
-7 ,
-6 ,
58 ,
59 ,
60 ,
63
);

----Signals for Control Unit
signal Input,Output_for_Prediction , Output_for_Detection : integer ;
signal Prediction_Output , Energy_Detection_Output ,
Coastline_Detection_Output : STD_LOGIC;
signal PreDeCounter: integer := 0;
signal clk2 : std_logic :='0';
signal Selection : STD_LOGIC := '0'; --DeMux
signal Q: STD_LOGIC; --Trigger of window counter prediction
signal Q_energy: STD_LOGIC; --Trigger of window counter energy
signal Q_coastline: STD_LOGIC; --Trigger of window counter coastline
signal AdderEnable_Prediction : STD_LOGIC;
signal AdderReset_Prediction : STD_LOGIC;
signal RegisterReset_Predicition : STD_LOGIC := '0';
signal CounterEnable_Prediction : STD_LOGIC;
signal CounterEnable_Coastline : STD_LOGIC;
signal CounterEnable_Energy : STD_LOGIC;
signal CounterReset_Prediction : STD_LOGIC;
signal CounterReset_Coastline : STD_LOGIC;
signal CounterReset_Energy : STD_LOGIC;
signal RE_Comparator : STD_LOGIC; -- prediction
signal enableenergy: std_logic; -- energy
signal th_eng: std_logic; -- threshold enable for energy
signal Reset_energy_block: std_logic; -- Reset output of energy
algorithm
signal transferdataenable: std_logic; -- coastline
signal enablecoastline: std_logic; -- coastline
signal threshold_enable : std_logic; -- threshold enable for
coastline
signal flag_energy: std_logic; -- to know if energy has finished
signal flag_coastline: std_logic; -- to know if coastline has
finished
```

```
signal flag_prediction: std_logic; -- to know if prediction has
finished
signal RST_Output : STD_LOGIC;
signal Threshold : integer;
signal flag_clk2 :integer:=0;


type state is
(unintialized,reset_prediction,prediction_state_2,prediction_state_1,
division_prediction,Rsignal1,Final_Prediction_state, --prediction

reset_coastline,coastline_state_2,coastline_state_1,threshold_coastli
ne, --coastline

reset_energy,energy_state_2,energy_state_1,threshold_energy --energy
                    );
signal current_state_prediction, next_state_prediction: state;
signal current_state_energy, next_state_energy: state;
signal current_state_coastline, next_state_coastline: state;
signal current_state_decision, next_state_decision: state;

-------------------------------------------
signal counterCLK2 : INTEGER := 0;

begin

-- This process is used to generate the clock that samples the input
(slow clock = 250 Hz)
process (clk)
    begin
        if counterCLK2 < 108000 then
        if rising_edge(clk) then
        counterCLK2<=counterCLK2 +1;
        end if;
        clk2 <= '0';

        else
        clk2 <='1';
        counterCLK2 <= 0;
        end if;
end process;


PROCESS(clk2)
    VARIABLE i : INTEGER := 0;
    BEGIN
        IF RISING_EDGE(clk2) THEN
            i:=i+1;
            IF(i<10) THEN -- input array size
                Input <= InArray(i);
            ELSE
                Input <= 0;
            END IF;
        END IF;

END PROCESS;
```

```
process(clk,RST,clk2)
     begin
      if (RST = '1') then
                current_state_prediction <= reset_prediction;
                current_state_energy <= reset_energy;
                current_state_coastline <= reset_coastline;
        else
                if(clk'event and clk='1')then
                    current_state_prediction<=next_state_prediction;
                    current_state_energy<=next_state_energy;
                    current_state_coastline<=next_state_coastline;
                end if;

                if(clk2'event and clk2='1')then
                    flag_clk2 <= flag_clk2 + 1;
                else
                    flag_clk2 <= 0;
                end if;
        end if;
end process;


-- PREDICTION BRANCH FSM
process (current_state_prediction,Selection,RST,flag_clk2,Q)
Begin
if (Selection ='0') then
Case current_state_prediction is

When reset_prediction =>
                RST_Output <= '0';
                AdderEnable_Prediction <='0';
                CounterEnable_Prediction <='0';
                AdderReset_Prediction <='1';
                RE_Comparator <='0';
                CounterReset_Prediction <='1';
                flag_prediction <= '0';
                next_state_prediction <= prediction_state_1;

When prediction_state_2 =>
                RST_Output <= '0';
                AdderEnable_Prediction <='1';
                CounterEnable_Prediction <='1';
                AdderReset_Prediction <='0';
                RE_Comparator <='0';
                CounterReset_Prediction <='0';
                flag_prediction <= '0';
                if (RST ='0' and Q ='0') then
                next_state_prediction <= prediction_state_1;
                elsif (RST ='0' and Q ='1') then
                next_state_prediction <= division_prediction;
                end if;

When prediction_state_1 =>
                RST_Output <= '0';
                AdderEnable_Prediction <='0';
                AdderReset_Prediction <='0';
                CounterEnable_Prediction <='0';
                RE_Comparator <='0';
                CounterReset_Prediction <='0';
                flag_prediction <= '0';
```

```
                if (RST ='0' and Q ='0' and flag_clk2 =1) then
                next_state_prediction <= prediction_state_2;
                elsif (RST ='0' and Q ='1') then
                next_state_prediction <= division_prediction;
                else    next_state_prediction <= prediction_state_1;
                end if;


When division_prediction =>
                RST_Output <= '0';
                AdderEnable_Prediction <='0';
                AdderReset_Prediction <='0';
                CounterEnable_Prediction <='0';
                RE_Comparator <='0';
                CounterReset_Prediction <='0';
                flag_prediction <= '0';
                next_state_prediction <= Rsignal1;

When Rsignal1 =>
                RST_Output <= '0';
                AdderEnable_Prediction <='0';
                AdderReset_Prediction <='0';
                CounterEnable_Prediction <='0';
                RE_Comparator <='1';
                CounterReset_Prediction <='1';
                flag_prediction <= '0';
                next_state_prediction <= Final_Prediction_state;

When Final_Prediction_state =>
                RST_Output <= '1';
                AdderEnable_Prediction <='0';
                AdderReset_Prediction <='0';
                CounterEnable_Prediction <='0';
                RE_Comparator <='0';
                CounterReset_Prediction <='0';
                flag_prediction <= '1';
                next_state_prediction <= reset_prediction;

when others =>  next_state_prediction <= reset_prediction;
end case;
end if;
end process;



-- COASTLINE BRANCH FSM
process (current_state_coastline,Selection,RST,flag_clk2,Q_coastline)
Begin

if (Selection ='1') then
Case current_state_coastline is

When reset_coastline =>
                RegisterReset_Predicition <= '0';
                CounterEnable_coastline <='0';
                CounterReset_coastline <='1';
                transferdataenable <='0';
                threshold_enable <='0';
                enablecoastline <='0';
```

```
                flag_coastline<='0';
                next_state_coastline <= coastline_state_1;


When coastline_state_2 =>
                RegisterReset_Predicition <= '0';
                CounterEnable_coastline <='1';
                CounterReset_coastline <='0';
                transferdataenable <='0';
                threshold_enable <='0';
                enablecoastline <='1';
                flag_coastline<='0';
                if (RST ='0' and Q_coastline ='0') then
                next_state_coastline <= coastline_state_1;
                elsif (RST ='0' and Q_coastline ='1') then
                next_state_coastline <= threshold_coastline;
                end if;

When coastline_state_1 =>
                RegisterReset_Predicition <= '0';
                CounterEnable_coastline <='0';
                CounterReset_coastline <='0';
                transferdataenable <='0';
                threshold_enable <='0';
                enablecoastline <='0';
                flag_coastline<='0';
                if (RST ='0' and Q_coastline ='0' and flag_clk2 =1)
then
                transferdataenable <='1';
                next_state_coastline <= coastline_state_2;
                elsif (RST ='0' and Q_coastline ='1') then
                enablecoastline <='1';
                next_state_coastline <= threshold_coastline;
                else next_state_coastline<=coastline_state_1;
                end if;

When threshold_coastline =>
                RegisterReset_Predicition <= '1';
                CounterEnable_coastline <='0';
                CounterReset_coastline <='1';
                transferdataenable <='0';
                threshold_enable <='1';
                enablecoastline <='0';
                flag_coastline<='1';
                next_state_coastline <= reset_coastline;

when others =>

                next_state_coastline <= reset_coastline;
end case;
end if;
end process;



-- ENERGY BRANCH FSM
process (current_state_energy,Selection,RST,flag_clk2,Q_energy)
Begin
if (Selection ='1') then
```

```
Case current_state_energy is

When reset_energy =>
                CounterEnable_energy <='0';
                Reset_energy_block <= '1';
                CounterReset_energy <='1';
                enableenergy<='0';
                flag_energy<='0';
                th_eng<='0';
                next_state_energy <= energy_state_1;

When energy_state_2 =>
                CounterEnable_energy <='1';
                Reset_energy_block <= '0';
                CounterReset_energy <='0';
                enableenergy<='1';
                flag_energy<='0';
                th_eng<='0';
                if (RST ='0' and Q_energy ='1') then
                next_state_energy <= threshold_energy;
                else next_state_energy <= energy_state_1;
                end if;

When energy_state_1 =>
                CounterEnable_energy <='0';
                Reset_energy_block <= '0';
                CounterReset_energy <='0';
                enableenergy<='0';
                flag_energy<='0';
                th_eng<='0';
                if (RST ='0' and Q_energy ='0' and flag_clk2 =1) then
                next_state_energy <= energy_state_2;
                elsif (RST ='0' and Q_energy ='1') then
                next_state_energy <= threshold_energy;
                else    next_state_energy <= energy_state_1;
                end if;


When threshold_energy =>
                CounterEnable_energy <='0';
                Reset_energy_block <= '0';
                CounterReset_energy <='1';
                enableenergy<='0';
                th_eng<='1';
                flag_energy<='1';
                next_state_energy <= reset_energy;


when others =>next_state_energy <= reset_energy;
end case;
end if;
end process;


-- DECISION MAKING BLOCK
process (RST,flag_coastline,flag_energy,flag_prediction)
Begin

if(Selection = '1') then
```

```
Output <= Energy_Detection_Output or Coastline_Detection_Output ;


if (Energy_Detection_Output = '0' and Coastline_Detection_Output =
'0' and flag_energy ='1' and flag_coastline ='1') then
    PreDeCounter <= PreDeCounter + 1 ;
elsif ((Energy_Detection_Output = '1' or Coastline_Detection_Output =
'1') and (flag_energy ='1' or flag_coastline ='1')) then
    PreDeCounter <= 0;
end if;


if (PreDeCounter = 10) then -- Window's counter & its value here is
10, you can change it
    PreDeCounter <= 0;
    Selection <= '0';
else
        Selection <= '1';
end if;

else

    if(flag_prediction = '1') then
        if (Prediction_Output = '1') then
            Selection <= '1';
        else
            Selection <= '0';
        end if;
    end if;

end if;
end process;


-- PORTMAPPING

Demux_data_path: Demux port map (Input , Selection ,
Output_for_Prediction , Output_for_Detection);
Prediction: prediction_block port map (Output_for_Prediction , Clk ,
Clk2 , CounterReset_Prediction , AdderReset_Prediction ,
RegisterReset_Predicition , AdderEnable_Prediction ,
CounterEnable_Prediction , RE_Comparator , Threshold , Q , RST_Output
,Prediction_Output);
Threshold_Block: Adaptive_Threshold port map (clk,Q,Prediction_Output
, Output , Selection , Threshold , Threshold_Counter , Threshold_low
, Threshold_mid , Threshold_high);
Energy_Detection: Detection_Energy port map (Output_for_Detection ,
Energy_Detection_Output , Clk , Clk2 , enableenergy , th_eng ,
CounterEnable_Energy , energy_threshold1 ,
energy_threshold2,Reset_energy_block  ,CounterReset_Energy ,
Q_energy);
Coastline_Detection : Detection_Coastline port
map(Output_for_Detection , Coastline_Detection_Output , Clk , Clk2  ,
transferdataenable , threshold_enable , enablecoastline ,
CounterEnable_Coastline , coastline_threshold1 , coastline_threshold2
,RST ,CounterReset_Coastline, Q_coastline);


end Behavioral;
```

F.3 DeMUX

```
----------------------------------------------------------------------
-------------
-- Company:
-- Engineer: Ahmed Yasser - Al Moataz Bellah Refeat - Taha Shawky -
Kareem Ayman - Mohamed Mahmoud Kamal - Mohamed Moustafa Abd El Rahman
--
-- Create Date:    19:13:12 04/30/2016
-- Design Name:
-- Module Name:    Demux - Behavioral
-- Project Name:     Seizure detector and predictor
-- Target Devices:
-- Tool versions:
-- Description:
--              The Demux has a selection line that will state which
block will work (Prediction or Detection)
--              It passes the input to one of the two blocks only(the
other has zeros)
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Demux is
Port (
    Input : in integer;
    Selection : in STD_LOGIC;
    Output_Prediction : out integer;
    Output_Detection : out integer
    );
end Demux;

architecture Behavioral of Demux is

begin

Demux_process: process(Input,Selection)
begin

if (Selection = '0') then -- predictor will work and detector is off
Output_Prediction <= Input;
Output_Detection <= 0;

elsif (Selection ='1') then ---- detector will work and predictor is
off
Output_Prediction <= 0;
Output_Detection <= Input;

--else
end if;

end process;
end Behavioral;
```

F.4 Prediction_Block

```
----------------------------------------------------------------------
-------------
-- Company:
-- Engineer: Ahmed Yasser - Al Moataz Bellah Refeat - Taha Shawky -
Kareem Ayman - Mohamed Mahmoud Kamal - Mohamed Moustafa Abd El Rahman
--
-- Create Date:    11:57:45 04/21/2016
-- Design Name:
-- Module Name:    prediction_block - Behavioral
-- Project Name:     Seizure detector and predictor
-- Target Devices:
-- Tool versions:
-- Description:
--              This module describes the prediction branch which
consists of the adder , division and comparator
--              blocks.in addition to a counter that indicates the
completing of the window.
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity prediction_block is
    Port ( Input : in  integer;
               Clk : in STD_LOGIC;
               Clk2 : in STD_LOGIC;
               RST_counter : in STD_LOGIC;
               RST_adder : in STD_LOGIC;
               RST2 : in STD_LOGIC;
               AdderEnable : in STD_LOGIC; -- to enable summation
process
               CounterEnable : in STD_LOGIC; -- to enable counter to
count
               RE_Comparator : in  STD_LOGIC; -- to enable data to
comes out from the reg to the comparator
               Comparator_Threshold : in integer;
               Trigger_Counter : out STD_LOGIC;
               RST_Output : in STD_LOGIC;
           Output : inout  STD_LOGIC);
end prediction_block;

architecture Behavioral of prediction_block is

component Prediction_Adder is
    Port ( a : in  integer;
           b : in  integer;
              AdderEnable : in STD_LOGIC;
           clk: in STD_LOGIC;
              reset: in STD_LOGIC;
              c : out  integer);
```

```
end component;

component window_counter is
    Port ( clk : in  STD_LOGIC;
           Enable : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           Q : out  STD_LOGIC := '0');
end component;

component Prediction_Devision is
    Port ( Input : in  integer;
              Trigger : in STD_LOGIC;
              clk: in STD_LOGIC;
           Output : out  integer);
end component;

component Prediction_Comparator is
    Port ( Input : in  integer;
              Threshold : in integer;
           RE_Comparator : in  STD_LOGIC;
              Trigger : in STD_LOGIC;
              clk : in  STD_LOGIC;
           Output : inout  STD_LOGIC;
              RST2 : in  STD_LOGIC;
              RST_Output : in STD_LOGIC
              );
end component;

Signal Output_Divison :  integer := 0;
signal Input_FlipFlop  : integer := 0;
Signal Trigger : STD_LOGIC := '0';

begin
Adder : Prediction_Adder port map ( Input , Input_FlipFlop ,
AdderEnable , Clk , RST_adder , Input_FlipFlop );
WindowCounter : window_counter port map (Clk2 , CounterEnable ,
RST_counter , Trigger);
Divison : Prediction_Devision port map ( Input_FlipFlop , Trigger ,
CLK, Output_Divison );
Comparator : Prediction_Comparator port map ( Output_Divison ,
Comparator_Threshold , RE_Comparator , Trigger, clk , Output , RST2 ,
RST_Output);
Trigger_Counter <= Trigger;
end Behavioral;
```

F.4.1 Prediction_Adder

```
----------------------------------------------------------------------------
-------------
-- Company:
-- Engineer: Ahmed Yasser - Al Moataz Bellah Refeat - Taha Shawky -
Kareem Ayman - Mohamed Mahmoud Kamal - Mohamed Moustafa Abd El Rahman
--
-- Create Date:    19:26:45 04/19/2016
-- Design Name:
-- Module Name:    Prediction_Adder - Behavioral
-- Project Name:     Seizure detector and predictor
```

```
-- Target Devices:
-- Tool versions:
-- Description:
--              Adding the new input(sample) to the previous one and
so on till the window finishes.
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity Prediction_Adder is
    Port ( a : in  integer;
           b : in  integer;
              AdderEnable : in STD_LOGIC;
           clk: in STD_LOGIC;
              reset: in STD_LOGIC;
              c : out  integer:= 0);
end Prediction_Adder;
architecture Behavioral of Prediction_Adder is

begin
Adding_process:process(clk,reset)
begin

if(reset = '1') then
c<= 0;
end if;

if(rising_edge(clk) and (AdderEnable = '1')) then -- AdderEnable is
used to avoid adding undesired values as the input remains on the
wire for a long time
c <= abs(a + b); -- Adding In rising Edge of Clock Only!!!
end if;
end process;
end Behavioral;
```

F.4.2 Windows_Counter

```
----------------------------------------------------------------------
-------------
-- Company:
-- Engineer: Ahmed Yasser - Al Moataz Bellah Refeat - Taha Shawky -
Kareem Ayman - Mohamed Mahmoud Kamal - Mohamed Moustafa Abd El Rahman
--
-- Create Date:    03:09:07 04/23/2016
-- Design Name:
-- Module Name:    up_counter - Behavioral
-- Project Name:     Seizure detector and predictor
-- Target Devices:
```

```
-- Tool versions:
-- Description:
--              This is the most important block as it is responsible
for the window.In other words, this block
--              indicates whether the window is completed or not.
--              We are working on 256 sample/second and our window is
4 seconds so we have 1024 sample/window.
--              Each new input is considered as a sample so after
1024 input the window is completed.
--              We have 2 clocks in this project.the normal clock
which is very fast(responsible for all operations)
--              and the other clock is responsible for entering a new
input,this one is much slower than the normal one
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;

entity window_counter is
    Port ( clk : in  STD_LOGIC;
           Enable : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           Q : out  STD_LOGIC := '0');
end window_counter;

architecture Behavioral of window_counter is

signal temp : integer := 0; -- the counter that counter the number of
inputs

begin

process (RST,clk)
begin
if (RST='1') then
    temp<=0;
    Q<='0';
end if;
if (clk'event and clk='1') then -- we need an enable to control the
counter bec. without this enable, the counter(temp) will count with
the fast clock not with each new input and this is wrong
        temp <= temp + 1;
end if;

if (temp = 1000) then -- 1000 is the number of samples per window as
discussed before
Q<='1'; -- window is completed
end if;

end process ;

end Behavioral
```

F.4.3 Division_Prediction

```
----------------------------------------------------------------------
-------------
-- Company:
-- Engineer: Ahmed Yasser - Al Moataz Bellah Refeat - Taha Shawky -
Kareem Ayman - Mohamed Mahmoud Kamal - Mohamed Moustafa Abd El Rahman
--
-- Create Date:    20:26:00 04/19/2016
-- Design Name:
-- Module Name:    Prediction_Devision - Behavioral
-- Project Name:     Seizure detector and predictor
-- Target Devices:
-- Tool versions:
-- Description:
--          Get the average of the output of the adder (after the
window finishes).
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_textio.all;

entity Prediction_Devision is
    Port ( Input : in  integer;
              Trigger : in STD_LOGIC;
              clk: in STD_LOGIC;
           Output : out  integer:= 1 );
end Prediction_Devision;

-- Note This Division Block will Act as Shift Register

architecture Behavioral of Prediction_Devision is

begin

Shift_process:process(Input,clk)
begin

if (Trigger = '1') then -- to ensure that the window has been
finished
Output <= Input / 1000; -- we divide by the window size (256 sample *
4 secs)
end if;


end Process;

end Behavioral;
```

F.4.4 Prediction_Comparator

```
----------------------------------------------------------------------
-------------
-- Company:
-- Engineer: Ahmed Yasser - Al Moataz Bellah Refeat - Taha Shawky -
Kareem Ayman - Mohamed Mahmoud Kamal - Mohamed Moustafa Abd El Rahman
--
-- Create Date:    20:44:53 04/19/2016
-- Design Name:
-- Module Name:    Prediction_Comparator - Behavioral
-- Project Name:     Seizure detector and predictor
-- Target Devices:
-- Tool versions:
-- Description:
--              The input to the comparator(the current window)
enters in Reg1 while Reg2 contains the old window value.
--              According to a certain equation (includes
Reg1,Reg2&threshold) the predictor determines whether a seizure
--              is coming or not.
--              After the decision is taken,the value of Reg1 is
transfered to Reg2 and the new window value will enter
--              Reg1 and the loop begins again.
--              When the predictor expect a coming seizure, the
selection line to Demux becomes 1 and detector turns on.
--              IF no seizure is expected then the predictor
continues to work (detector is off) and selection line to
--              Demux remains 0.
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_textio.all;
use std.textio.all;


entity Prediction_Comparator is
    Port ( Input : in  integer;
              Threshold : in integer;
           RE_Comparator : in  STD_LOGIC;
              Trigger : in STD_LOGIC;
              clk : in  STD_LOGIC;
              Output : inout  STD_LOGIC;
              RST2 : in  STD_LOGIC;
              RST_Output : in  STD_LOGIC
              );
end Prediction_Comparator;

architecture Behavioral of Prediction_Comparator is

shared variable Reg1:integer:=1;
```

```
shared variable Reg2:integer:=1;


begin


Reg1_process:process(RE_Comparator , RST_Output , RST2)
begin

if RST2='1' Then
    Reg1:=1;
    Reg2:=1;

else

    if (RST_Output = '1') Then
        output<='0';

    else

        if( RE_Comparator = '1' ) then -- to control the comparator
block (when to be on and when to be off)
        Reg1:=Input;

            if(((Reg1)>=(Threshold*Reg2)) and (Threshold>0)) Then --
The equation
            output<='1'; --Expecting a seizure (detecot will turn on
now)
            else

            output<='0'; --Expecting no seizure (predictor continues
work and detecot still off)
                if((Trigger = '1')) then -- to ensure that the
transfer happens only when the window finishes.
                Reg2:=Reg1;
                end if;

            end if;
        end if;

    end if;
end if;
end process;

end Behavioral;
```

F.5 Adaptive_Thrreshold

```
----------------------------------------------------------------------
-------------
-- Company:
-- Engineer: Ahmed Yasser - Al Moataz Bellah Refeat - Taha Shawky -
Kareem Ayman - Mohamed Mahmoud Kamal - Mohamed Moustafa Abd El Rahman
--
-- Create Date:    16:11:53 05/13/2016
-- Design Name:
-- Module Name:    Adaptive_Threshold - Behavioral
-- Project Name:     Seizure detector and predictor
-- Target Devices:
-- Tool versions:
-- Description:
--          Our prediction technique is adaptive which means when the
predictor states there is no seizure for a long
--          time,the threshold becomes lower and vice versa , if the
predictor states there are many seizures (false after detection)
--          the threshold becomes higher and so on.
--          this module is responsible for changing the threshold as
described above.
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Adaptive_Threshold is
    Port ( clk:in STD_LOGIC;
              Q:in STD_LOGIC;
              Input_Prediction : in  STD_LOGIC;
           Input_Seizure : in  STD_LOGIC;
              SelectionBlocks : in STD_LOGIC;
           Threshold : inout  integer ;
              Threshold_Counter : in  integer;
           Threshold_low : in  integer;
           Threshold_mid : in  integer;
           Threshold_high : in  integer);
end Adaptive_Threshold;

architecture Behavioral of Adaptive_Threshold is

signal counterZeros : integer := 0; -- count how many window
expecting non seizure(predictor)
signal counterOnes : integer := 0; -- count how many window expecting
seizure(predictor)
signal counterDetection : integer := 0; -- count how many false
seizure(detector , after detection finishes)
signal Smart_Threshold : integer := Threshold_mid; --the threshold
which will be sent to the prediction block
begin

process(Input_Prediction , Input_Seizure , SelectionBlocks , Q)
```

```
begin

if(Q'event and Q='1')then -- when window finishes
if (Input_Seizure = '0' and SelectionBlocks = '1') then -- false
positive
    counterDetection <= counterDetection + 1;
elsif (Input_Seizure = '1' and SelectionBlocks = '1') then --true
seizure
    counterDetection <= 0;
    counterZeros <= 0;
    counterOnes <=0;
end if;

if (SelectionBlocks = '0') then -- prediction state
if (Input_Prediction = '0' ) then -- output of predictor is zero
(expecting no seizure)
    counterZeros <= counterZeros + 1;
     counterOnes <= 0;
elsif (Input_Prediction = '1' ) then -- output of predictor is one
(expecting seizure)
    counterOnes <= counterOnes + 1;
     counterZeros <= 0;
end if;

if (counterZeros >= Threshold_Counter) then
    counterZeros <= 0;
      counterOnes <= 0;
      if(Smart_Threshold = Threshold_high) then
        Smart_Threshold <= Threshold_mid;
        Threshold <= Smart_Threshold;
     else
        Smart_Threshold <= Threshold_low;
        Threshold <= Smart_Threshold;
     end if;


elsif ((counterOnes >= Threshold_Counter) and (counterDetection >=
Threshold_Counter)) then
       counterZeros <= 0;
    counterOnes <= 0;
    if(Smart_Threshold = Threshold_low) then
        Smart_Threshold <= Threshold_mid;
        Threshold <= Smart_Threshold;
     else
        Smart_Threshold <= Threshold_high;
        Threshold <= Smart_Threshold;
     end if;
else
Smart_Threshold <= threshold_mid;
Threshold <= Smart_Threshold;

end if;
end if;

end if;
end process;

end Behavioral;
```

F.6 Energy_Detection

```
----------------------------------------------------------------------
-------------
-- Company:
-- Engineer: Ahmed Yasser - Al Moataz Bellah Refeat - Taha Shawky -
Kareem Ayman - Mohamed Mahmoud Kamal - Mohamed Moustafa Abd El Rahman
--
-- Create Date:    17:13:32 04/22/2016
-- Design Name:
-- Module Name:    mainEnergy - Behavioral
-- Project Name:     Seizure detector and predictor
-- Target Devices:
-- Tool versions:
-- Description:
--              This module describes the energy branch which
consists of the energy algorithm and thresholding
--              block.in addition to a counter that indicates the
completing of the window.
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity Detection_Energy is
port (

Input:in integer;
Output:inout std_logic;
clk: in std_logic;
Clk2 : in STD_LOGIC;
enableenergy: in std_logic;
th_eng:in std_logic;
CounterEnable : in std_logic;
threshold1: in integer;
threshold2: in integer;
reset: in std_logic;
resetcounter: in std_logic;
Trigger_Counter : out STD_LOGIC


);
end Detection_Energy ;

-------------------------------
Architecture behavior of Detection_Energy is

component Energy is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           enable : in  STD_LOGIC;
           sample : in  integer;
```

```
            E : inout  integer);
end component;

component EnergyThreshold is
    PORT( clk                                   :   IN    std_logic;
        reset                                   :   IN    std_logic;
        enable                                  :   IN    std_logic;
        th_eng                                  :   IN    std_logic;
        EnergyValue                             :   IN    integer;
        threshold_energy_1                      :   IN integer;
        threshold_energy_2                      :   IN    integer;
        classifier_E                            :   inOUT   std_logic
        );
end component;

component window_counter is
    Port ( clk : in  STD_LOGIC;
           Enable : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           Q : out  STD_LOGIC := '0');
end component;

signal energy_value_signal : integer := 0;
signal Trigger: STD_LOGIC :='0';

begin

U1: energy PORT MAP (clk=>clk, reset=>reset, enable=>enableenergy,
sample=>Input, E=>energy_value_signal);
counter: window_counter port map
(clk2,CounterEnable,resetcounter,Trigger);
U2: EnergyThreshold PORT MAP (clk=>clk, th_eng=>th_eng, reset=>reset,
enable=>Trigger, EnergyValue=>energy_value_signal,
threshold_energy_1=>threshold1, threshold_energy_2=>threshold2,
classifier_E=>Output);
Trigger_Counter<=Trigger;


END behavior;
```

F.6.1 Energy_Block

```
----------------------------------------------------------------------
-------------
-- Company:
-- Engineer: Ahmed Yasser - Al Moataz Bellah Refeat - Taha Shawky -
Kareem Ayman - Mohamed Mahmoud Kamal - Mohamed Moustafa Abd El Rahman
--
-- Create Date:    17:09:39 04/22/2016
-- Design Name:
-- Module Name:    energy - Behavioral
-- Project Name:     Seizure detector and predictor
-- Target Devices:
-- Tool versions:
-- Description:
--          The energy algorithm is that the coming input(sample) is
squared then add the resulted value to the previous
```

```
--          one and so on.Then the loop begins again till the window
ends.
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.numeric_std.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Energy is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           enable : in  STD_LOGIC;
           sample : in  integer;
           E : inout  integer :=0);
end Energy;

ARCHITECTURE rtl OF Energy IS

  -- Signals
  SIGNAL enb: std_logic; -- to enable energy block
    signal mul_temp : integer;

BEGIN
  enb <= enable;
  EnergyAvg_1_output : PROCESS (sample,clk,reset)
  BEGIN
  mul_temp<=0;
  if (reset ='1') then
  E<=0;
  end if;

if ((enb = '1')) then
      mul_temp <= sample * sample; -- squaring the coming input
      E <= E + mul_temp; -- adding the result of squaring to the
previous one
     end if;

  END PROCESS EnergyAvg_1_output;
END rtl;
```

F.6.2 Windows_counter

As mentioned above in appendix F.4.2.

F.6.3 Energy_Threshold

```
----------------------------------------------------------------------
-------------
-- Company:
-- Engineer: Ahmed Yasser - Al Moataz Bellah Refeat - Taha Shawky -
Kareem Ayman - Mohamed Mahmoud Kamal - Mohamed Moustafa Abd El Rahman
--
-- Create Date:    17:12:39 04/22/2016
-- Design Name:
-- Module Name:    EnergyThreshold - Behavioral
-- Project Name:     Seizure detector and predictor
-- Target Devices:
-- Tool versions:
-- Description:
--              This module is for thresholding of the Energy
algorithm.
--              The energy output of a windows is compared with an
amplitude threshold.If it is greater than
--              this threshold, increase a counter by 1 , if not
reset the counter.
--              The counter is compared with another threshold.The
threshold is set by how many successive windows
--              have a coastline output greater than the amplitude
threshold.
--              If the counter becomes greater than the second
threshold this indicates the presence of a seizure.
--              The TWO thresholds are determined by multiple
simulations on matlab and they are patient specific.
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.ALL;
use ieee.std_logic_textio.all;
use std.textio.all;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

ENTITY EnergyThreshold IS
```

```
  PORT( clk                                 :   IN    std_logic;
        reset                               :   IN    std_logic;
        enable                          :   IN    std_logic;
        th_eng                          :   IN    std_logic;
        EnergyValue                         :   IN    integer;
        threshold_energy_1                  :   IN integer;
        threshold_energy_2                  :   IN    integer;
        classifier_E                        :   inOUT   std_logic :='0'
        );
END EnergyThreshold;

ARCHITECTURE rtl OF EnergyThreshold IS

  -- Signals
  SIGNAL enb                                : std_logic;
signal counter : integer :=0 ;
signal AvgEnergyValue : integer :=0 ;
signal Write_counter : integer := 0;
signal Write_counter2 : integer := 0;

BEGIN
  enb <= enable; -- if enb = 1 this means the window is completed
  EnergyThreshold_1_output : PROCESS (reset,enb,th_eng)

  BEGIN


AvgEnergyValue <= EnergyValue / 1000; -- to get average value of the
window

if (enable ='1' and th_eng='1') then  -- to enable threshold
block(th_eng signal comes from FSM)

     IF AvgEnergyValue >= threshold_energy_2 THEN -- the comparison
with the amplitude threshold
        counter <= counter + 1;
     ELSE
        counter <= 0;
    END IF;

    IF counter >= threshold_energy_1 THEN -- the comparison with the
second threshold
      classifier_E <= '1';

    ELSE
      classifier_E <= '0';
    END IF;

end if;
  END PROCESS EnergyThreshold_1_output;

END rtl;
```

F.7 Coastline_Detection

```
----------------------------------------------------------------------
-------------
-- Company:
-- Engineer: Ahmed Yasser - Al Moataz Bellah Refeat - Taha Shawky -
Kareem Ayman - Mohamed Mahmoud Kamal - Mohamed Moustafa Abd El Rahman
--
-- Create Date:    17:36:01 04/30/2016
-- Design Name:
-- Module Name:    mainCoastLine - Behavioral
-- Project Name:     Seizure detector and predictor
-- Target Devices:
-- Tool versions:
-- Description:
--              This module describes the coastline branch which
consists of the coastline algorithm and thresholding
--              block.in addition to a counter that indicates the
completing of the window.
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Detection_Coastline is
port (

Input:in integer;
Output:inout std_logic;
clk: in std_logic;
Clk2 : in STD_LOGIC;
transferdataenable: in std_logic;
threshold_enable : in STD_LOGIC;
enablecoastline: in std_logic;
CounterEnable: in std_logic;
threshold1: in integer;
threshold2: in integer;
reset: in std_logic;
resetcounter: in std_logic;
Trigger_Counter : out STD_LOGIC
);

end entity;
```

```
------------------------
architecture Behavioral of Detection_Coastline is

component CoastlineAlgorithm is

  Port ( Input : in  integer;
           CL_Value : out integer;
          enablecoastline : in  STD_LOGIC;
             transferData : in STD_LOGIC;
             clk : in  STD_LOGIC;
             RST : in  STD_LOGIC
             );

end component;


component Thresholding IS
  PORT( clk                                 :   IN    std_logic;
        reset                               :   IN    std_logic;
        enable                          :   IN    std_logic;
          threshold_enable                    :  IN std_logic;
        CoastLineValue                        :   IN    integer;  --
double
        threshold_coastline_1                 :   IN integer;  --
double1
        threshold_coastline_2                 :   IN    integer;  --
double
        classifier_CL                         :   inOUT   std_logic --
double
        );
END component;

component window_counter is
    Port ( clk : in  STD_LOGIC;
           Enable : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           Q : out  STD_LOGIC := '0');
end component;

signal coastline_value_signal : integer := 0;
signal adder_value : integer := 0;
signal Trigger : std_logic := '0';

begin

U1: CoastlineAlgorithm PORT MAP (clk=>clk, RST=>resetcounter,
Input=>Input, CL_Value=>coastline_value_signal,
transferData=>transferdataenable, enablecoastline=>enablecoastline);
Counter: window_counter port map
(clk2,CounterEnable,resetcounter,Trigger);
U3: Thresholding PORT MAP(clk=>clk, reset=>reset,
threshold_enable=>threshold_enable, enable=>Trigger,
CoastLineValue=>coastline_value_signal,
threshold_coastline_1=>threshold1, threshold_coastline_2=>threshold2,
classifier_CL=>Output);
Trigger_Counter <= Trigger;

end Behavioral;
```

F.7.1 Coastline_Block

```
----------------------------------------------------------------------
-------------
-- Company:
-- Engineer: Ahmed Yasser - Al Moataz Bellah Refeat - Taha Shawky -
Kareem Ayman - Mohamed Mahmoud Kamal - Mohamed Moustafa Abd El Rahman
--
-- Create Date:    17:59:53 04/30/2016
-- Design Name:
-- Module Name:    CoastlineAlgorithm - Behavioral
-- Project Name:   Seizure detector and predictor
-- Target Devices:
-- Tool versions:
-- Description:
--          We have in this module 2 registers (Reg1,Reg2). The
coastline algotithm is to subtract the values in
--          Reg1 and Reg2 then add the resulted value to the previous
one and so on.
--          After subtracting and adding the value in Reg2 is
transfered to Reg1 and the new input will transfer
--          to Reg2.Then the loop begins again till the window ends.
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity CoastlineAlgorithm is

  Port ( Input : in  integer;
           CL_Value : out integer;
          enablecoastline : in  STD_LOGIC;
             transferData : in STD_LOGIC;
             clk : in  STD_LOGIC;
             RST : in  STD_LOGIC
             );

end CoastlineAlgorithm;

architecture Behavioral of CoastlineAlgorithm is

signal Reg1:integer := 0;
signal Reg2:integer := 0;
signal Coastline_value : integer := 0;
begin
```

```
RST_Process:process(RST,clk)
begin
-------------------------------------------------------------------
-----------------------
-- IF RST = 1 , the registers & coastline value (the value resulted
from registers' subtraction are cleared) --
    if RST='1' Then
    Reg1<=0;
    Reg2<=0;
    Coastline_value<=0;
    end if;

if(enablecoastline = '1' ) then  -- allows coastline algorithm to
begin.this signal comes from FSM

Reg2<=Input; -- input in Reg2
Coastline_value <=  Coastline_value + abs(Reg2-Reg1); -- subtracting
and adding
CL_Value <= Coastline_value; -- put the result in the output
end if;

if((transferData = '1')) then -- responsible for transfering the data
from Reg2 to Reg1 and comes from FSM
Reg1<=Reg2;
end if;

end process;

end Behavioral;
```

F.7.2 Windows_counter

As mentioned above in appendix F.4.2.

F.7.3 Coastline_Threshold

```
-------------------------------------------------------------------
-------------
-- Company:
-- Engineer: Ahmed Yasser - Al Moataz Bellah Refeat - Taha Shawky -
Kareem Ayman - Mohamed Mahmoud Kamal - Mohamed Moustafa Abd El Rahman
--
-- Create Date:    19:59:19 04/30/2016
-- Design Name:
-- Module Name:    Thresholding - Behavioral
-- Project Name:     Seizure detector and predictor
-- Target Devices:
-- Tool versions:
-- Description:
```

```
--              This module is for thresholding of the Coastline
algorithm.
--              The coastline output of a windows is compared with an
amplitude threshold.If it is greater than
--              this threshold, increase a counter by 1 , if not
reset the counter.
--              The counter is compared with another threshold.The
threshold is set by how many successive windows
--              have a coastline output greater than the amplitude
threshold.
--              If the counter becomes greater than the second
threshold this indicates the presence of a seizure.
--              The TWO thresholds are determined by multiple
simulations on matlab and they are patient specific.
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_textio.all;
use std.textio.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

ENTITY Thresholding IS
  PORT( clk:   IN    std_logic;
        reset :   IN    std_logic;
        enable:   IN    std_logic;
        threshold_enable:   IN    std_logic;
        CoastLineValue:   IN    integer;
        threshold_coastline_1  :   IN integer;
        threshold_coastline_2  :   IN    integer;
        classifier_CL  :   inOUT   std_logic := '0'       );
END Thresholding;


ARCHITECTURE rtl OF Thresholding IS

  -- Signals
  SIGNAL enb                              : std_logic;

SIGNAL counter : integer :=0 ; -- Counter to count the number of
successive windows have coastline values greater than the amplitude
threshold
BEGIN
  enb <= enable; -- if enb = 1 this means the window is completed
```

```
  CoastlineThreshold_output : PROCESS (CoastLineValue,clk,reset)

  BEGIN

  if (reset ='1') then

  counter <= 0;
  classifier_CL <='0';
  end if;


if (enb = '1' and threshold_enable= '1') then    -- to enable
threshold block(threshold_enable signal comes from FSM)

    IF CoastLineValue >= threshold_coastline_1 THEN -- the comparison
with the amplitude threshold
        counter <= counter + 1;
    ELSE
        counter <= 0;

    END IF;

    IF counter >= threshold_coastline_2 THEN -- the comparison with
the second threshold
      classifier_CL <= '1';
    ELSE
      classifier_CL <= '0';
    END IF;

end if;
  END PROCESS CoastlineThreshold_output;

END rtl;
```

F.8 pin.ucf file

```
NET CLK  LOC="AB13"; //To generate the 27 MHz master clock that is
downsampled to get the low sampling frequnecy (250 Hz)
NET RST  LOC="F3"; //Push Button to reset the code
```