CUFE

Communications and Computer Engineering (CCE-E)

Credit Hours System

Spring 2018
Senior-2 Level
Graduation Project-2

CCEN-481

Graduation Project-2

"Seizure Detection"

Final Report

Submitted by:

Abdel-Malik Mohamed Sabreen

Adel Ahmed Samir

Lojaine AlaaElDin ElMahdy

Mirna HossamElDin

Mohamed Hany Tawfik

Omneia Osama Elshaer

Supervised by:

Dr. Hassan Mostafa


ONE Lab
Biomedical Signal Processing
Seizure Detection
Project Overview
Work Highlights
EEG Signal Reading
MACHINE LEARNING
مدرج ٨٢٠٨

Acknowledgement

We would like to show our extreme gratitude and thankfulness for all who helped us in our graduation project throughout a whole year. The outstandingly continuous support has extremely helped us to demonstrate our research project at the best possible way till the end. We can never miss the chance to describe our sincere thanks to them as a mark of recognition and appreciation. We would like to thank:

- Dr. Hassan Mostafa
- Eng. Mostafa Mahmoud
- Eng. Mohamed Adel

Executive Summary

Epilepsy is a condition in which the patient has recurrent seizures. Seizure is defined as an abnormal disorder discharging of the nerve cells, resulting in a temporary disturbance of sensory and mental function. Seizures could cause damage to the body and might lead to death in critical cases.

This thesis discusses a seizure detection algorithm to treat seizures by implanting a chip in the patient's brain and detecting seizures and non-seizures periods.

The electrical activities of patients are recorded using Electroencephalogram (EEG) which measures the voltage fluctuations of the brain. A classification method is needed to categorize between seizures and non-seizures. The thesis provides a discussion of two algorithms for the classification process by applying machine learning concepts using support vector machine (SVM): Sequential Minimum Optimization and Gilbert's Algorithm.

The main scope of the solution to the problem of seizure detection is researching, implementing and analyzing Gilbert's algorithm in SVM.

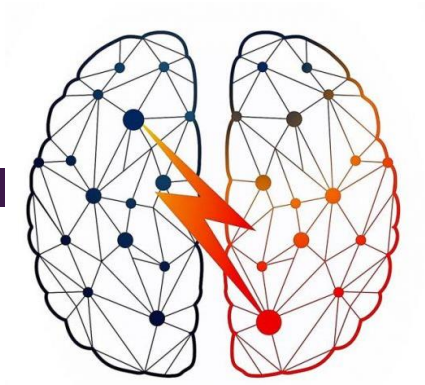
MATLAB code of the modified algorithm was implemented on the dataset and the performance measurements were obtained and compared to the previous used algorithm SMO. After getting satisfying results, block diagram of the overall system was designed for the next stage of implementing RTL (Register Transfer Level) phase. In the RTL phase, maping the code and the design architecture to the VHDL and Verilog hardware descriptive language is done. Some improvements in the MATLAB code have been added.

Finally, burning the training algorithm and the classification code on the FPGA has been used in proving the concept of detecting seizures successfully with an acceptable accuracy and sensitivity using Gilbert's Algorithm.

Contents

List of Figures

List of Tables



1

Introduction: From Brain to Train

Epilepsy has been around for centuries. In the past, in many societies, people with Epilepsy were believed to be possessed by demons and were often burned. Other societies thought that they had weak or inferior minds. In the late 1800s, Dr. John Hughlings Jackson, a British neurologist, discovered Epilepsy and how it affected the human brain. He found the main characteristics of Epilepsy that led him to discover this disorder. Dr. Jackson also found out that Epilepsy causes seizures and what the seizures do to the individual [1].

Developing different methods of treatment has never ended. Medication remains the most common treatment for people with Epilepsy. However, people diagnosed with Epilepsy aim for Epilepsy with no seizures with no side effects. If you are on a good drug, you're likely to have good control of your seizures and you might not even know that there are better approaches out there. When medicines don't work, you can choose the Epilepsy surgery track which is good also for people who are in the early stages of the disease. In 2001, a study proved that 60% of people who did the surgery had no seizures, while, 8% of people who had taken the best medication but didn't get surgery got the same results. When medication and surgeries don't work, implantable devices play a good role in Epilepsy treatment.

One of these devices is: The Vagus Nerve Stimulation (VNS). It prevents seizures by sending regular pulses to the brain via vagus nerve. This device is implanted in the chest, powered by a small battery under the skin of the chest and a wire from the device is wound around the vagus

nerve in the neck. The other stimulator is called NeuroPace which is the scope of our project in which the device sends out pulses at a certain interval with the aid of electrodes placed on the surface of the brain. By simulating an EEG signals and sensing their pattern, electrical pulses that disrupt their patterns are fired up. [4]

This introductory chapter will describe an overview of the project, its relevance to our studies and the engineering approach in the project. Furthermore, reasons beyond using FPGA in the project and the users of the project are discussed. Finally, a roadmap of the chapters that will follow throughout this thesis is given.

1.1 Project Overview

Epilepsy is a neurological disorder caused by abnormal electrical discharges in the brain. Approximately 0.7% of the world population suffers from epileptic seizures. More than 50 million people are affected by Epilepsy worldwide with more than 2.2 million in the US [2]. As a consequence, treatment costs about $9.6 billion annually for direct medical care of epileptic patients in US such as anti-convulsive medication and resistive surgery [3]. However, these traditional methods for dealing with Epilepsy do not give efficient solution for this disease.

Epilepsy changes the brain state to an abnormal state that causes seizure periods to the patients. Seizures could cause drastic damage to the body and even more critical situations where it could lead to death. A solution for this problem is required and it is the scope of this project.

1.2 Relevance

Our project is somehow relevant to some courses that we studied throughout our study in the Communications and Computer Engineering Department in Cairo University. We studied MATLAB and it was very essential in dealing with features obtained from the patients' dataset. Moreover, Electronics-2 course provided us with the basics of digital design using CMOS transistors. Linear Algebra courses were helpful in understanding Machine Learning concepts. Last but not least, Computer Architecture course was exceptionally beneficial in developing our skills in designing the hardware block diagram that was converted to RTL then implemented on FPGA to produce the chip we target.

1.3 Engineering Approach

A solution to the problem is to build a chip that will be implanted in the patient's brain. The target of this chip is to employ Machine Learning (ML) algorithms to be able to detect a seizure for any patient and hence trigger electric stimuli that will be able to retrieve the brain back to its normal conditions as illustrated in Figure 1.

Three main aspects need to be considered to maintain acceptable results:

- ***Chip accuracy***: Building and employing new Machine Learning algorithm that best fits the seizure detection problem.
- ***Chip power consumption:*** Lower power consumption rates are targeted in this project than ever.
- ***Chip Area:*** The chip will be implanted at the subject brain; therefore, area is one of the most important aspects.

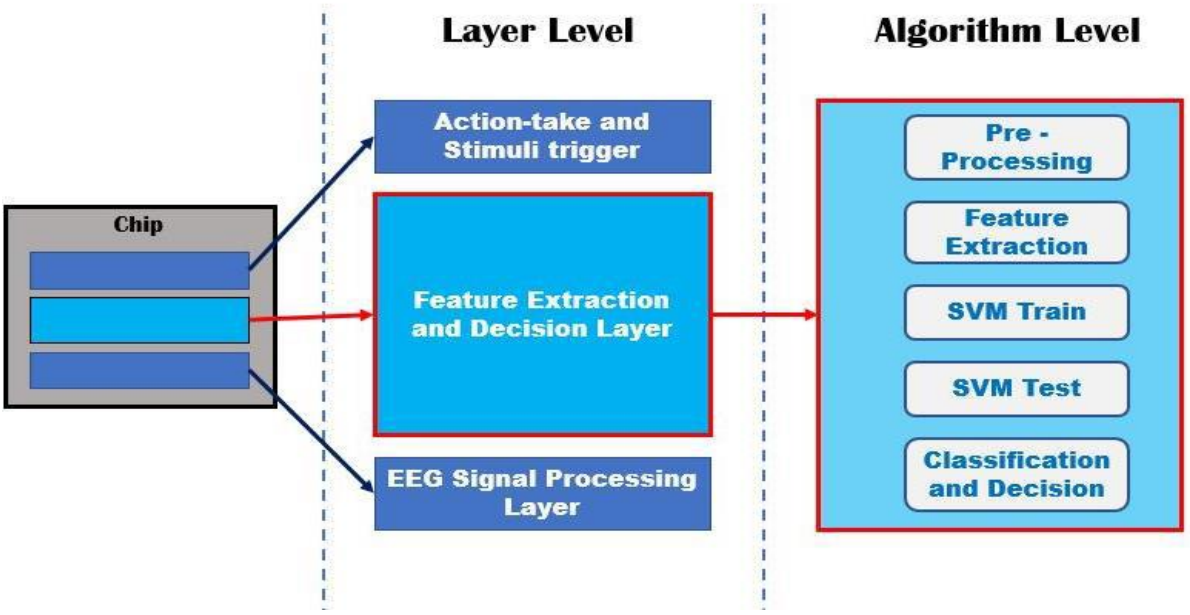


Figure 1.1 Project Levels

The middle layer (feature extraction and decision layer) is the main interest of the required design problem. The techniques of this layer are to first build an effective Machine Learning algorithm that is able to efficiently detect a seizure from the readings of the EEG signal (electric signals of the brain) using [MATLAB] tool. Then, this algorithm will be mapped to low level

hardware descriptive language to be able to define its hardware layout and simulation using [ModelSim] simulation tool.

Eventually FPGA mapping for the HDL code is conducted to produce the end product of the project.

1.4 Use of FPGA

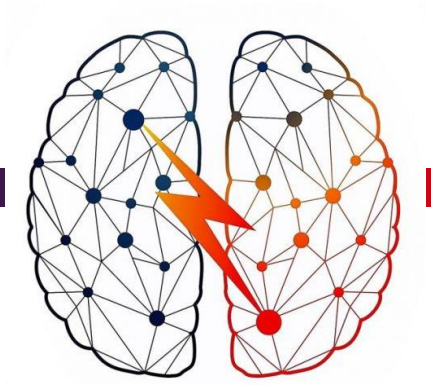
An FPGA device is targeted because of its runtime reconfigurability, which allows the design to adapt to different types of input data (practical to the diversity problem of epilepsy). The use of FPGA device would effectively aid the online training of the SVM especially as it is considered a large-scale classification problem; hence, taking advantage of the FPGA resources maximizing its utilization would prove very helpful for the design problem.

1.5 Users Of The Project

The project is aimed to serve people who are suffering from epilepsy. The targeted users will be medical institutes (e.g. hospitals), medical industry and the end user product would be epileptic patients.

1.6 Road map to the sections of the report

The chapters are organized in a sequence that makes it easy to the reader to understand our project. Chapter 2 describes the medical background needed to know the behavior of epileptic seizures. Chapters 3 and 4 describe how the signals obtained from patients' brains are processed and converted to features that are classified eventually to seizure or non-seizure signals. Chapters 5, 6 and 7 describe basics of machine learning, the Support Vector Machine (SVM) that is the classifier of seizure and non-seizure points and special algorithms implementing SVM. Chapters 8 and 9 describe our mapping to hardware along with the use of FPGA. Chapters 10, 11 and 12 discuss our results obtained in our work throughout the project along with our recommendations and conclusion. Ultimately, there design codes are attached in the appendix.



2

Biomedical Background: Human Brain and Epilepsy

This chapter will discuss the human brain and the nervous system very briefly so as to grasp the biomedical idea beyond the project. Furthermore, seizure definition is introduced from a biomedical perspective.

2.1 Human Brain and Nervous System

The human nervous system is the most magnificent part of the human body, resembling a software. It is responsible for interpreting sensory information received from the environment so that humans can behave as humans do. It has the brain and the spinal cord as the two main parts of it. The human nervous system is composed of millions of neurons, they are small nerve cells that helps the flow of electric pulse inside the human body. These pulses traveling in our bodies are not only electrical; they are rather electrical and chemical. This makes the chemistry in the brain much more complex and interesting.

We are going to investigate the operation of the flow of pulse through the brain, but on a cell level. This chapter is dedicated to presenting concepts necessary to understand the brain as a system.

It is the neurons that are responsible for the processing and transmission of information. Neurons come in many different shapes and sizes, but they are composed of four basic structures: dendrites, soma or cell body, axon and synaptic terminals.

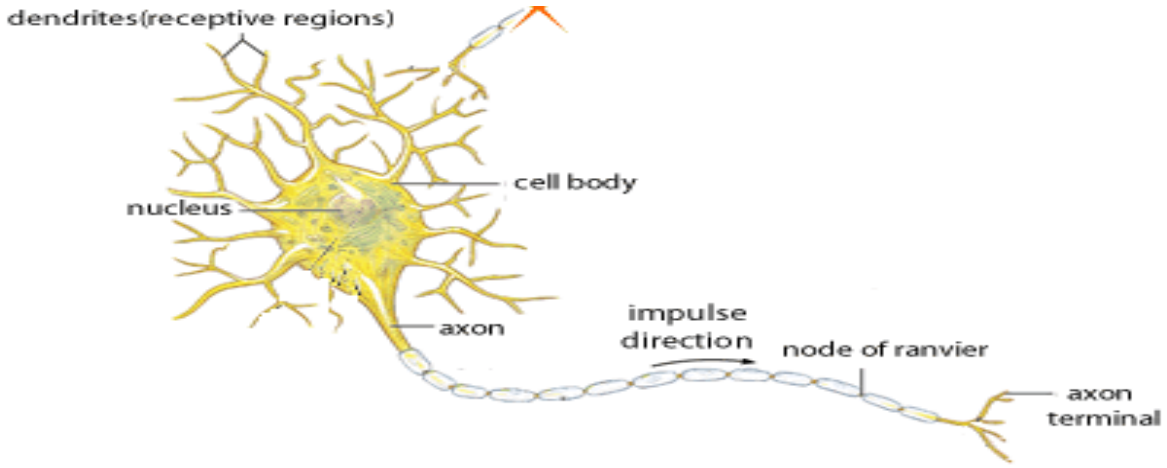


Figure 2.1 Neuron

The neuron consists of dendrites where it receives input. The input then travels through the cell body in which decisions about firing an action is done, that results in mostly movement. The output is then fired through the axons. Each neuron is not connected directly to the other, however, there are gaps between them, and connected by the synapse.

This fired action is called action potential. The process includes both chemical and electrical reactions. It first occurs at the synapse. These synapses contain a lot of neurotransmitters that are triggered by the calcium ions when electric pulse is passing. The neurotransmitters move from the pre-synapse of a cell to the gates of the post synaptic cell, in which it stimulates the sodium gates, opening them. By opening the sodium gate, more sodium ions flow inside the post synaptic cell, making it more positively charged than its neutral state. The positively charged sodium ions flow inside the neuron till it reaches the cell body, that's responsible for the decision of firing the action potential. The cell body accumulates all the inputs, and compares the electric signal to a certain threshold, giving us two possible types of action potentials: Excitatory and inhibitory action potentials.

Excitatory, is when the signal is more than the threshold value, allowing action potential to be fired. Inhibitory, is when the signal is much less than the threshold value, so no action potential is fired. A single Inhibitory post-synaptic potential typically has a larger effect than a single

excitatory post-synaptic potential because inhibitory synapses tend to form closer to the soma. However, the total number of EPSPs is greater than IPSPs and thus the effects level out. The balance of incoming IPSPs and EPSPs on a single neuron determines whether the post-synaptic cell fires an action potential. The following figure shows the effect of the EPSP and IPSP on the depolarization and repolarization process.

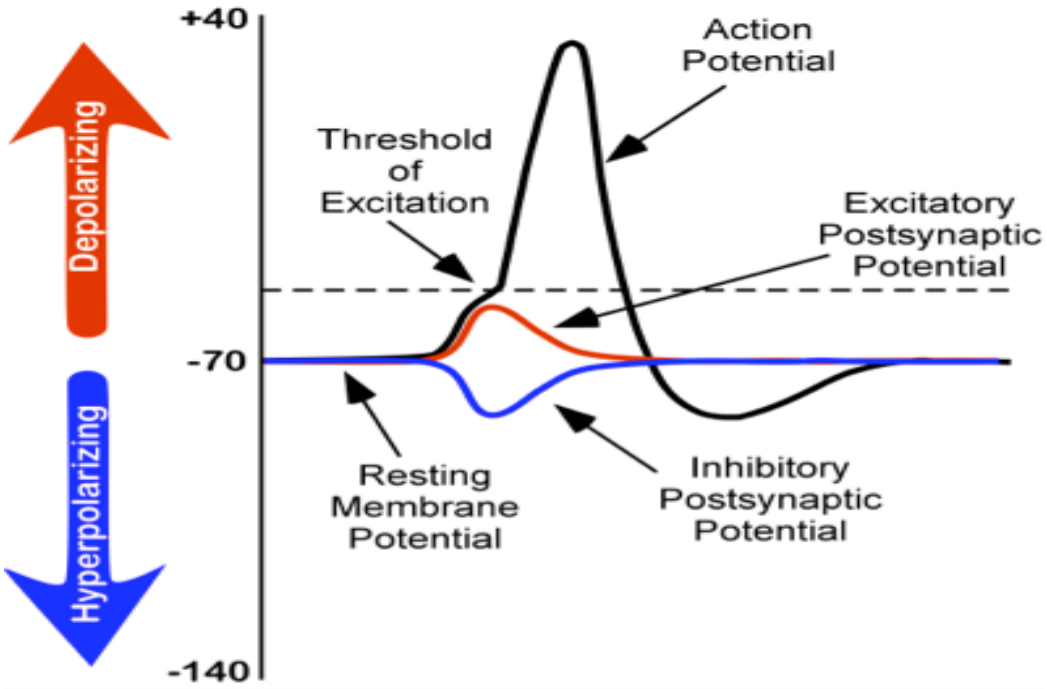


Figure 2.2 EPSP and IPSP

This phenomenon can only happen because of the cell membrane, which is under normal conditions, impermeable to fluids. The presence of gates allows the transmission of ions through this membrane.

According to these decisions done by the cell body the output is determined that travels through the axon in a form of electric pulse. These electric pulses are maintained by the different concentrations of sodium ions inside and outside the cell, thus allowing this electric flow till it reaches the synapse at the output, repeating again the function of synapse in transmitting the signal. There are many different types of gates, for example: Voltage-gated ions gates that open or close according to the differences in voltage between the inside and the outside of the neuron. When a certain threshold is reached, the gates open allowing the ions to flow and generate the action potential.

2.2 Seizure definition

Till this moment, no one has exactly proven why a person can have a seizure and another one doesn't. Seizure is a neurological disorder caused by structural abnormalities of the brain.

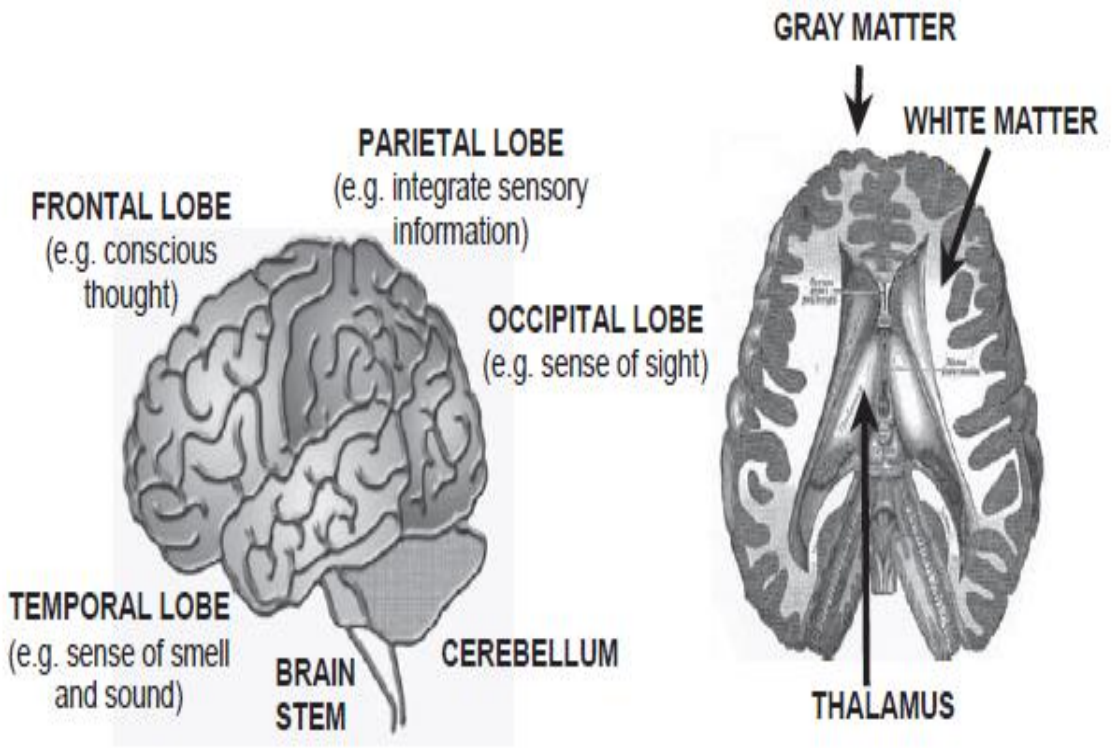


Figure 2.3 Brain major lobes

Each region of the brain has its own task. The different functional regions of the cortex (temporal lobe, parietal lobe, occipital lobe, frontal lobe) are responsible for motor control as well as cognitive and memory functions. Understanding the initiation of a seizure may lead to the ability to predict its onset. That's why we had to understand how the brain works and the different parts of it.

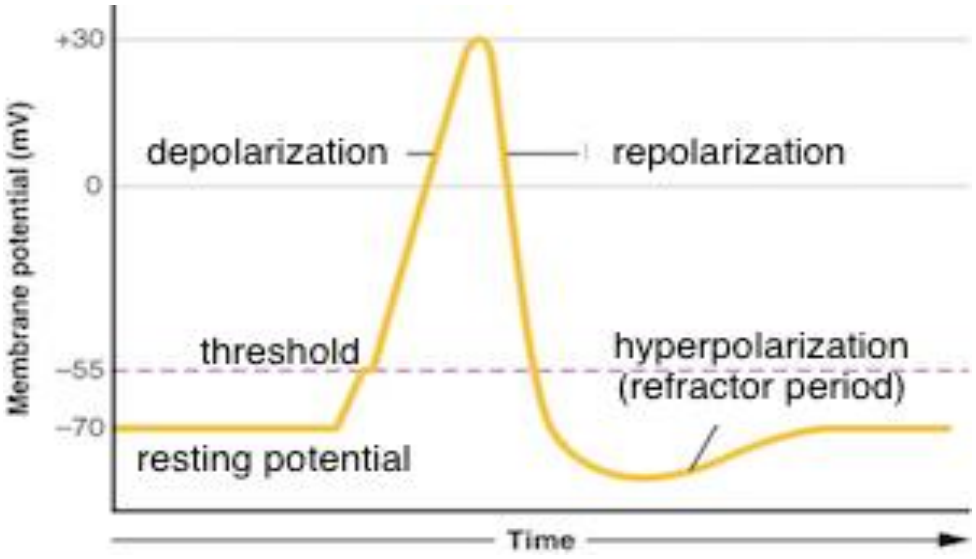
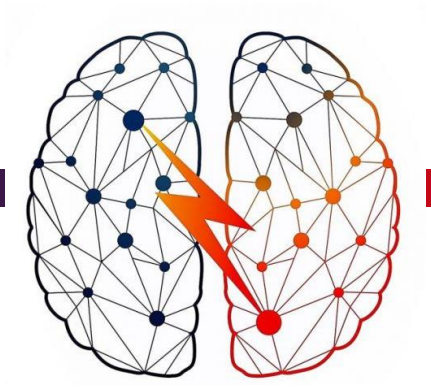


Figure 2.4 Depolarization and Repolarization

As shown in the previous figure, the normal operation is the depolarization of the cell, where the voltage increases because of the entry of the sodium. Then, repolarization occurs by the opening of the potassium channel until it reaches the complete repolarization and the cell becomes in its normal state again.

This happens in the axon until it gets to the synapse. This synapse sends a neurotransmitter to the neighbor synapse and it keeps moving from one to the other till it gets to the brain. In seizure, the depolarization occurs, but the repolarization doesn't, which keeps the cell in its abnormal state due to the lack of inhibitory neurotransmitters. If this gets to the parietal lobe, which is responsible for the movement, the patient may have epilepsy.

The next chapter will focus on describing EEG signals which are signals that are extracted from the epileptic patients' brains.



3

EEG: Bio-signal Processing

This chapter focuses on describing the EEG and its role in diagnosing epilepsy. In the brain, neurons exploit chemical reaction to generate electricity to control different bodily actions and this ongoing electrical activity can be recorded graphically which is popularly known as Electroencephalogram (EEG).

EEG is a well-accepted tool for epileptic seizure prediction/detection that can measure the voltage fluctuations of the brain. EEG has high temporal resolution. This means that it can capture fast changes in current flows. Meanwhile, EEG has poor spatial resolution which means that measurements are limited by the number of electrodes, their placement and properties of the head.

3.1 Types of EEG

There are two types of EEG to consider:

- **Scalp EEG:**

In Scalp EEG, recordings are heavily attenuated. It is used as a preliminary step to more detailed intra-cranial records.

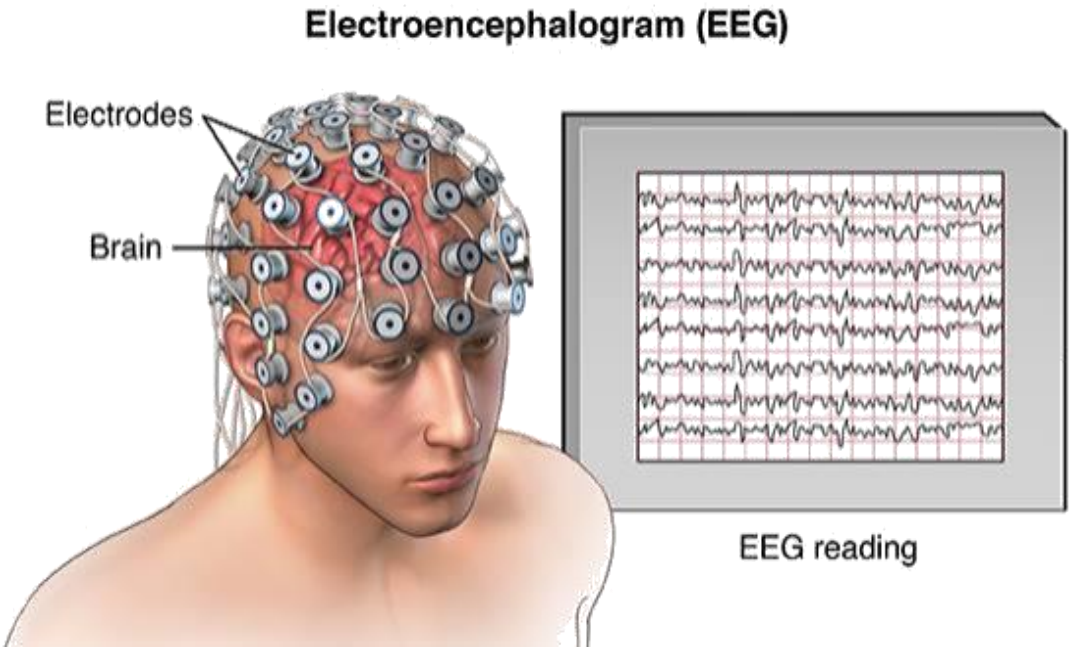


Figure 3.1 EEG Electrodes

❖ **Intra-Cranial EEG:**

In Intra-Cranial EEG, recordings can be taken from:

- Cortical electrodes (placed on Cortex)
- Depth electrodes (penetrate to sub-cortical systems such as the thalamus)

Intra-cranial records are often obtained for pre-surgical analysis to determine regions of the brain to be resected. Such procedures are relatively rare and data of this nature are more difficult to obtain. The difference between normal and epileptic EEG signals will be explained in the next two sections.

3.2 Normal EEG

The Normal EEG visually recognizes general patterns that exist consistently in the majority of the population. Absence of such patterns does not imply abnormality. EEG alone is not sufficient alone for diagnosis of epilepsy. The following figures show different Normal EEG signals for different states of alertness. The voltage magnitude in each channel is shown relative to each other. In (a) an example of an awake alpha rhythm shows the 10Hz activity present in only

some (posterior) channels. In contrast to the slow waveforms of sleep stages in (b) and (c), the awake EEG shows a lot more variability between channels, demonstrating the more global nature of sleep versus awake states.

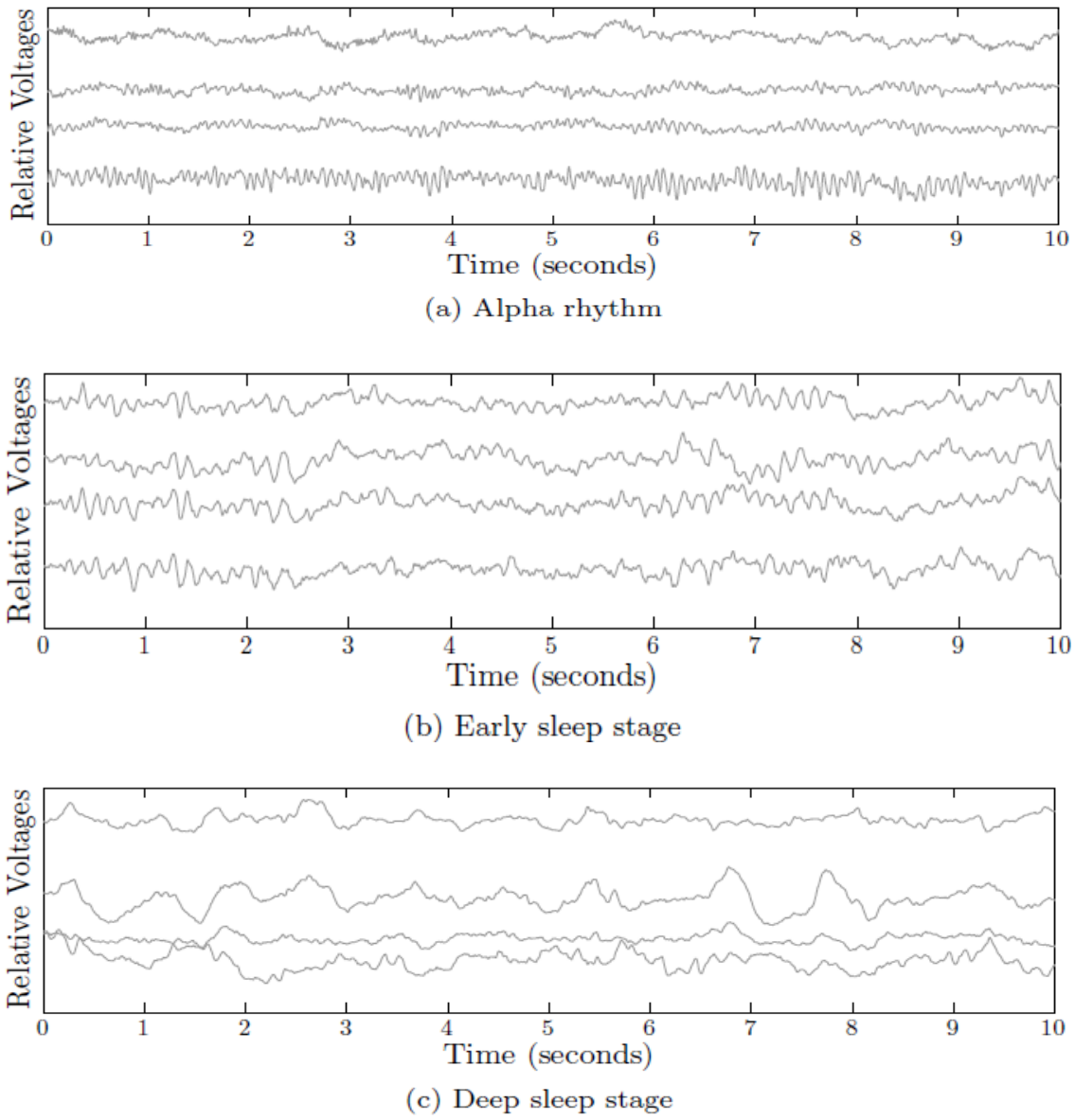


Figure 3.2 Normal EEG Signals

3.3 Epileptic EEG

The variability between epilepsies means that there is no single epileptic EEG. For example, inter-seizure periods can be as short as a few seconds or as long as years. The following figures show a sample Epileptic EEG signal. They are an example of a complete seizure of approximately 110 seconds duration, with its start and end as marked. The magnitude of the EEG during a seizure is much larger than that preceding it. We notice that the seizure evolves over time, with changes in morphology as well as fundamental frequency. We also notice the artifact that occurs at about the 73rd second. This is an example of electrodes becoming temporarily disconnected. The different time periods of EEG signals are discussed in the next section.

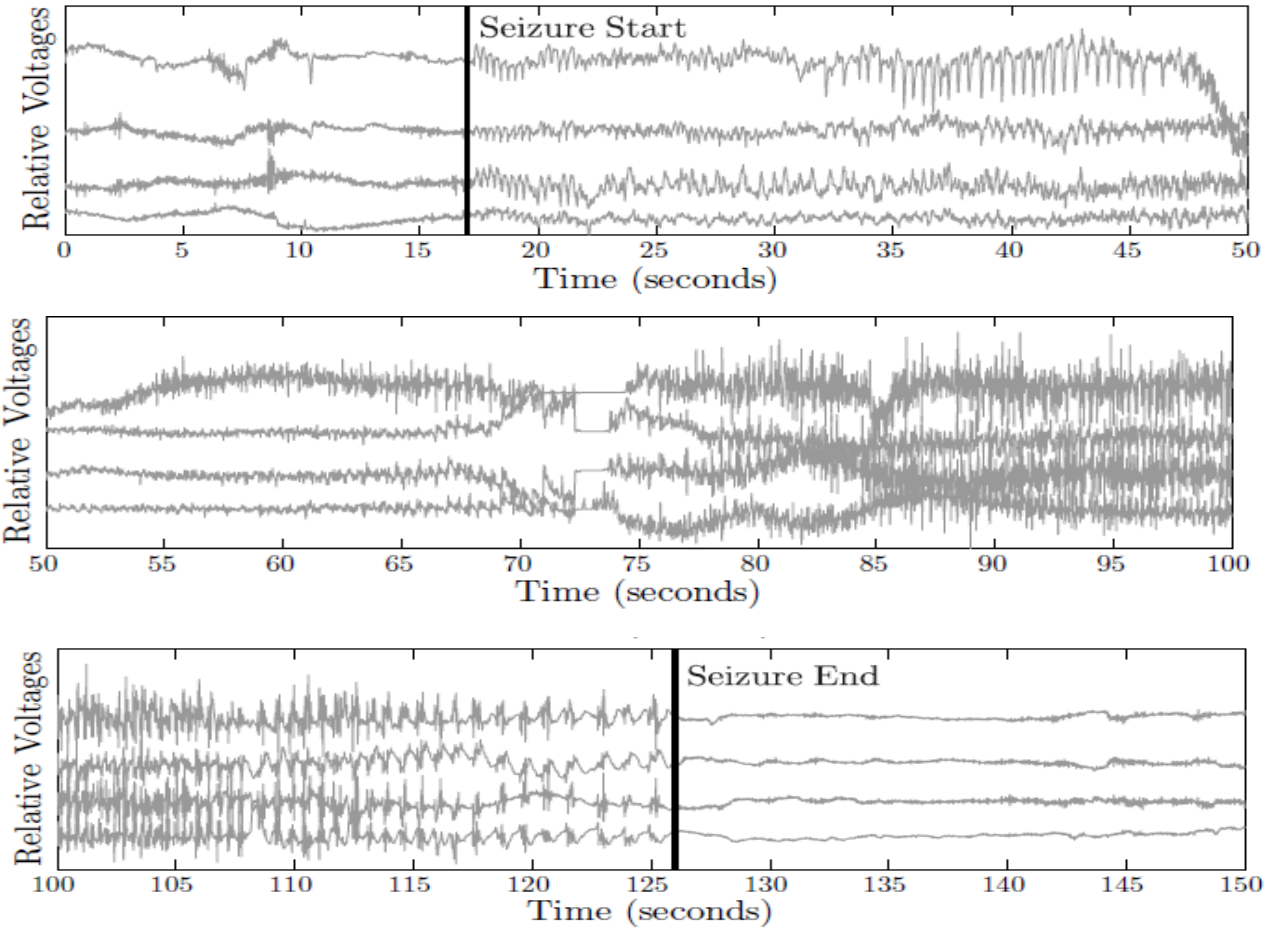


Figure 3.4 Epileptic EEG Signals

3.4 Periods of EEG Signals

EEG signals from an epileptic patient can be divided into five periods or stages:

- Non-seizure Period
- Ictal Period
- Preictal Period
- Post-ictal Period
- Interictal Period

Non-seizure period is the period when no epileptic syndrome is visible.

Ictal period is the actual seizure period, normally duration is 1 to 3 minutes.

Preictal period is 30 to 60 minutes before ictal period.

Post-ictal period is 30 to 60 minutes after ictal period.

Interictal period is the period between post-ictal period to pre-ictal period of the immediate next ictal. Some portion of the interictal period, which does not have any epileptic syndrome, can be defined as a non-seizure period.

Prediction and detection of seizures by analyzing ictal, pre-ictal, and interictal could alert a patient of the next seizure and also could lead to better treatment and safety. In order to be able to perform signal processing on the EEG signals to do seizure detection, some features need to be extracted. The different relevant features are discussed in the next section.

3.5 Extracted Features from EEG Signals

After removal of noise in the pre-processing stage (will be discussed in details later), certain features from the EEG signals are extracted. The features that are extracted belong to time domain. These time domain features give numerical indication to the visual patterns observed from the EEG signals such as the increase in amplitude, increase in frequency during epileptic events. These features are statistical elaborations that are very important in quantifying the EEG signals for classification.

The features that are going to be discussed now are as follows:

1) Energy
2) Coastline
3) Hjorth Variance Parameter

3.5.1 Energy

This feature calculates average energy of the EEG signal. This is done by averaging the instantaneous energy. This is done because instantaneous energy is not of much significance as the average energy which is more useful because it is done over a mean behavior. A sliding window is used to calculate the instantaneous energy. The window is of size N while K is the window number of size N.

The following two equations represent the instantaneous energy and average energy respectively:

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

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

3.5.2 Coastline

This feature calculates the sum of the absolute value of the distance between two successive data points. The coastline is calculated using the following equation:

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

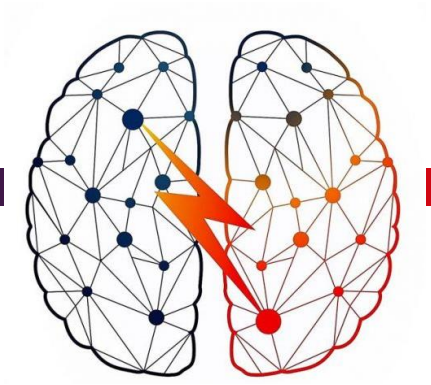
3.5.3 Hjorth Variance Parameter

This feature calculates the variance of the EEG signal amplitude. The variance is a window of N samples is calculated and averaged to obtain the mean variance. Hjorth variance parameter is calculated as follows:

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

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

In the next chapter, it will be discussed how classification is done using the features calculated. Detection and prediction techniques will be elaborated in details.



4

Classification:

Seizrue or non-siezure?

This chapter's focus is the classification between seizure and non-seizure periods in EEG signals. The classification can be done using either seizure detection or seizure prediction. Detection differs from prediction due to the following reasons. One thing is that a prediction model is required to classify correctly between preictal and interictal periods while a detection model should classify accurately between ictal and non-seizure or interictal periods. Seizure prediction is harder than seizure detection because the similarity between ictal and interictal signals is much higher than that of preictal and interictal signals.

Conceptually, seizure detection and seizure prediction are different. Seizure detection technique only knows the presence of a seizure when the characteristics of a seizure have appeared in the biologic signals being monitored. On the other hand, seizure prediction technique estimates the beginning of a seizure before it actually starts.

The following section will focus on seizure detection technique including the steps done to accurately do classification with Seizure detection.

4.1 Seizure Detection

Seizure detection includes four stages:

- Pre-processing
- Feature Extraction
- Feature Selection
- Feature Classification

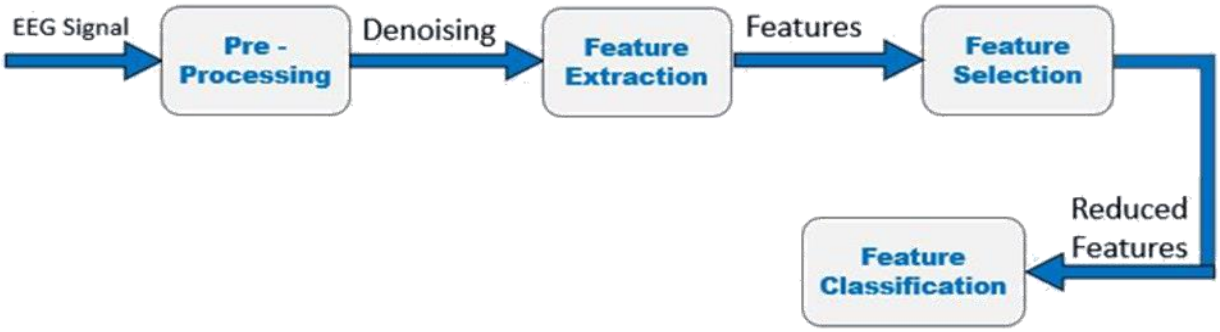


Figure 4.1 Seizure Detection Overview

4.1.1 Pre-processing

In the previous chapter we mentioned the mechanism of EEG and how it records the signals that are to be studied to detect seizures. However, the produced signals from the EEG are noised and disturbed. Hence, a pre-processing step is exploited in order to eliminate the influence of disturbance (i.e. artifacts). The artifact can be divided into two parts; one is physiological artifact that came from the body and another is non-physiological artifact that comes from environment and instruments. There are several types of physiological artifacts such as muscle artifact, pulse artifact and eye blinking artifact. The non-physiological artifacts are power line artifact and sweat artifact (i.e. water, minerals, and lactate so on). Notch filter can be used to remove line noise interference. Wavelet transform is an efficient denoising technique that is introduced to the non-linear and non-stationary EEG signals.

4.1.2 Feature Extraction

Now we have the clear EEG signals that are ready to be examined to detect seizures. For the sake of classification, relevant features are needed to be extracted from EEG signals. As the studies of researchers pointed to the problem of applying highly optimized algorithms to small, selected datasets, because the results cannot be reproduced on unselected, larger datasets; it is important to use real time classification techniques for detection purposes for actual applications. Therefore, feature extraction is considered a key to the performance of a classifier. The calculated features are fed into the standard support vector machine (SVM) for classification purposes and that will be discussed in a later chapter.

Two algorithms that are used in feature extraction will be considered. They are DWT which stands for Discrete Wavelet Transform and EMD which stands for Empirical Mode Decomposition.

DWT is similar to the Fourier transform. It is applied on EEG signals to decompose the signal into several scales to get information about frequency components which present and enhance the information about the signal for further processing. The relative wavelet statistical feature coefficient is computed in time domain. The extracted relative wavelet energy features are passed to classifiers for the classification purpose. It captures both frequency and location information (location in time). Mathematically, the wavelet will correlate with the signal if the unknown signal contains information of similar frequency.

EMD is used with nonlinear and nonstationary signal analysis. This technique breaks the signal into various finite small number of components called Intrinsic Mode Functions (IMFs). It is more preferable as this decomposition technique depends on local characteristics of dataset instead of pre-defined basis functions. Therefore, it is considered highly efficient and adaptive unlike the Fourier transform that converts the input signal from domain to another.

4.1.3 Feature Selection

Feature selection is a technique that removes the irrelevant features from the feature set and selects the most relevant ones. As we mentioned before, in some applications it might be desired to pick a subset of the original features rather then find a mapping that uses all of the original features. The benefits of finding this subset of features could be in saving cost of computing unnecessary features as well as saving cost of sensors. Principle Component Analysis (PCA) and Independent Component Analysis (ICA) can be used for dimensionality reduction of the features.

4.1.4 Feature Classification

After feature selection, we take the output from the feature selection to the classifier. The role of the classifier is to determine the patient's state, having a seizure or not. The main idea is to indicate a hyperplane that helps in classifying the signal coming from the feature selection. Using machine learning algorithms, a hyperplane can be detected. These machine learning details will be covered in the next chapter.

Feature extraction, analysis, and classification of EEG signals are still challenging issues for researchers due to the variations of the brain signals. Variations of EEG signals depend on different brain locations, number of channels, and different patterns of signals from different people. Another challenge for epileptic seizure detection/prediction from EEG signals is to get reasonable accuracy for real time applications.

The following section will focus on seizure prediction technique including the steps done to accurately do classification with Seizure prediction.

4.2 Seizure Prediction

Seizure detection includes six stages:

- Pre-processing
- Feature Extraction
- Feature Selection
- Feature Classification
- Regularization
- Decision Function

It is clear that the first four stages are identical to that of seizure detection while the last two stages are specific to seizure prediction only.

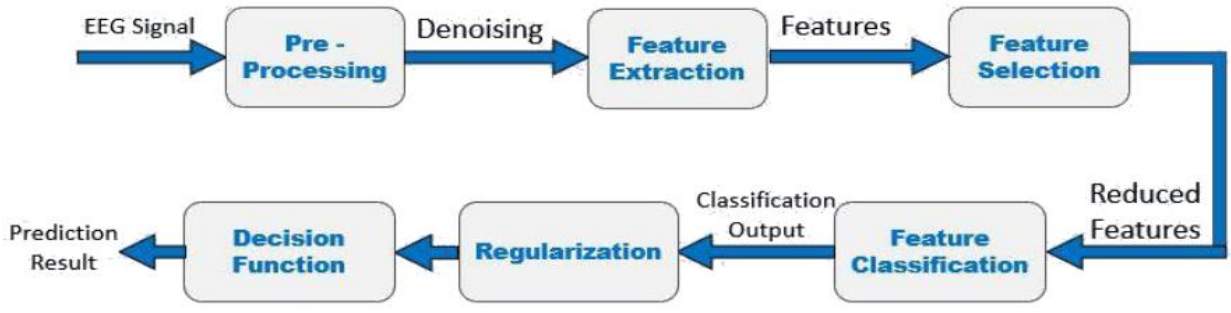


Figure 4.2 Seizure Prediction Overview

4.2.1 Definition

The predictor is a detector but the difference is the time in which the measurements are taken. The prediction is discovering the probability of having the disease before detecting it. Apart from "predicting" itself, predictor could be employed to save the power only if we guaranteed having a lower power consumption than the detector. The detector is working continuously so it consumes high power as it works on very complex equations. The proposed solution is that we can turn off the detector while turning on the predictor. When the predictor predicts having a seizure, it could send triggers to the detector and then the detector could be on again to proceed. The prediction block has two more blocks than the detector which are regularization and decision function.

4.2.2 Regularization

To separate between 2 types of data after classification, the over-fitting concept can be used. Over-fitting is separating two sets of data based on existing classified data. It guarantees 100% separation between the two data sets, but it doesn't introduce predictability. In order to introduce predictability, a regularized curve is convenient to predict that a certain data sets most probably they belong to seizure. However, it does not guarantee 100% separation. The following figure shows the separation between two data sets by an over-fitting curve and a regularized curve.

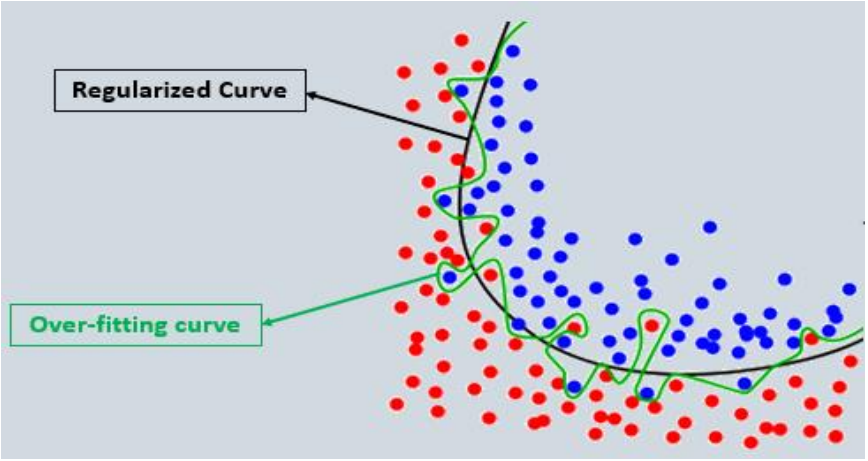


Figure 4.3 Over-fitting curve vs. Regularized curve in predictability

4.2.3 Decision Function

Our goal is to employ a new machine learning that best fits the seizure detection problem; thus, three main aspects are considered in our project:

1. **Chip accuracy:**

A classification output of an EEG signal might not provide accurate predicted results; thus, sometimes a decision function can be formulated based on the combined classification output of different time-window of an EEG signal or a number of EEG signals in different channels.

For the seizure detection, a number of criteria (such as accuracy, sensitivity, and specificity) are used to verify the classification outcome. Accuracy is determined as an overall performance measurement; however, we mostly care about the sensitivity.

The accuracy, sensitivity and specificity are defined as follows where TP is true positive, TN is true negative, FP is false positive, and FN is false negative:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}\ x\ 100 \tag{4.1}$$

$$Sensitivity = \frac{TP}{TP + FN}\ x\ 100 \tag{4.2}$$

$$Specificity = \frac{TN}{TN + FP}\ x\ 100 \tag{4.3}$$

Where accuracy is defined as the number of correctly detected seizures and non-seizures from the total number of seizures and non-seizures cases, sensitivity is defined as the number of correctly detected seizures from the total number of seizures and specificity is defined as the number of correctly detected non-seizures from the total number of non-seizures.

After optimizing those criteria by multiple iterations over the classification and taking feedback to the decision function, the performance of epileptic seizure prediction can be measured by measuring the prediction accuracy and false prediction rate.

$$\text{Prediction accuracy} = \frac{N_S}{N_A}\ x\ 100 \qquad (4.4)$$

$$\text{False prediction rate} = \frac{N_F}{N_T} \text{ x } 100 \qquad (4.5)$$

Where Ns is the number of correctly predicted seizures, Na is the total number of seizures, NF is the number of inaccurately predicted seizures and NT is the total time of EEG signals.

2. Chip Power consumption:

Having high sensitivity detection leads to consuming higher power due to the number of triggering indicating the existence of seizures.

Low consumption rate is a need to save the battery life in the chip that will be implanted in the patient's brain.

Changing the battery frequently has two main disadvantages:

- High cost of the battery
- Increase in the number of times of the surgical interference operation for replacing the battery

An optimization of the RTL design architecture of Gilbert's algorithm helps in lowering the chip power consumption rate.

3. Chip Area:

A control in the size of the chip area is needed as the chip will be implanted in the subject's brain. Having a small area size is controlled by a good and an optimized RTL design architecture.

The next section will show the results obtained on different seizure prediction techniques in a survey dedicated to seizure prediction only.

4.3 Prediction Survey

4.3.1 Introduction

All predicting models try to find out reliable measures as precursors of impending seizures. The measures should have strong correlation with the preictal stage of epilepsy cycle, to be able to predict the seizure before it happens. Most of the prediction techniques published up to now use a so-called moving window analysis in which some (linear or non-linear) characterizing measure is calculated from a window of EEG data with a pre-defined length, then the subsequent window of EEG is analyzed, and so forth.

Depending on whether the employed measure is used to characterize a single EEG channel or relations between two or more channels. The duration of these analysis windows usually ranges between 10 and 40 s.

We can categorize the EEG features into three main groups of univariates, bivariate, and multivariate. It was conducted a comprehensive review comparing most univariate and bivariate techniques and showed some intended results but none of them has succeeded in a reliable seizure prediction. Each group of univariates, bivariate and multivariate can be divided into two groups of linear and nonlinear measures.

Table 4-1 Prediction Techniques

Univariate linear	**Univariate non-linear**	**Bivariate linear**	**Non-linear Bivariate**
Statistical moments	Largest Lyapunov exponent	Maximum linear cross-correlation.	Non-linear interdependence.
Characteristics of the autocorrelation function	Estimate of an effective correlation dimension		Conditional probability-based index
Hjorth parameters	Dynamic similarity index		Index based on Shannon entropy.
Spectral band power	Algorithmic complexity		Phase synchronization.

4.3.2 Statistical versus algorithmic approaches

A statistical design is retrospective by nature and compares the amplitude distributions of the characterizing measures from the inter-ictal with those from the assumed pre-ictal period in one way or another. We use this design for investigating and comparing the potential predictive performance of different characterizing measures under different conditions.

On the other hand, an algorithmic analysis uses a design that produces a time-resolved output (i.e. an output for every point of a time profile). With respect to practical application, the algorithm should ideally be prospective, we can understand from this that its output for a given time should be a function of the information available at this time. Prediction algorithms usually employ certain thresholds. If the time profile of a characterizing measure crosses the threshold, the algorithm produces an alarm. This alarm can be either true or false, depending on whether it is actually followed by a seizure or not. For this distinction, it is necessary to define a prediction horizon which is the period after an alarm within which a seizure is expected. If an alarm is followed by a seizure within the prediction horizon, it is classified as a true alarm (true positive), otherwise it is regarded as a false alarm (false positive).

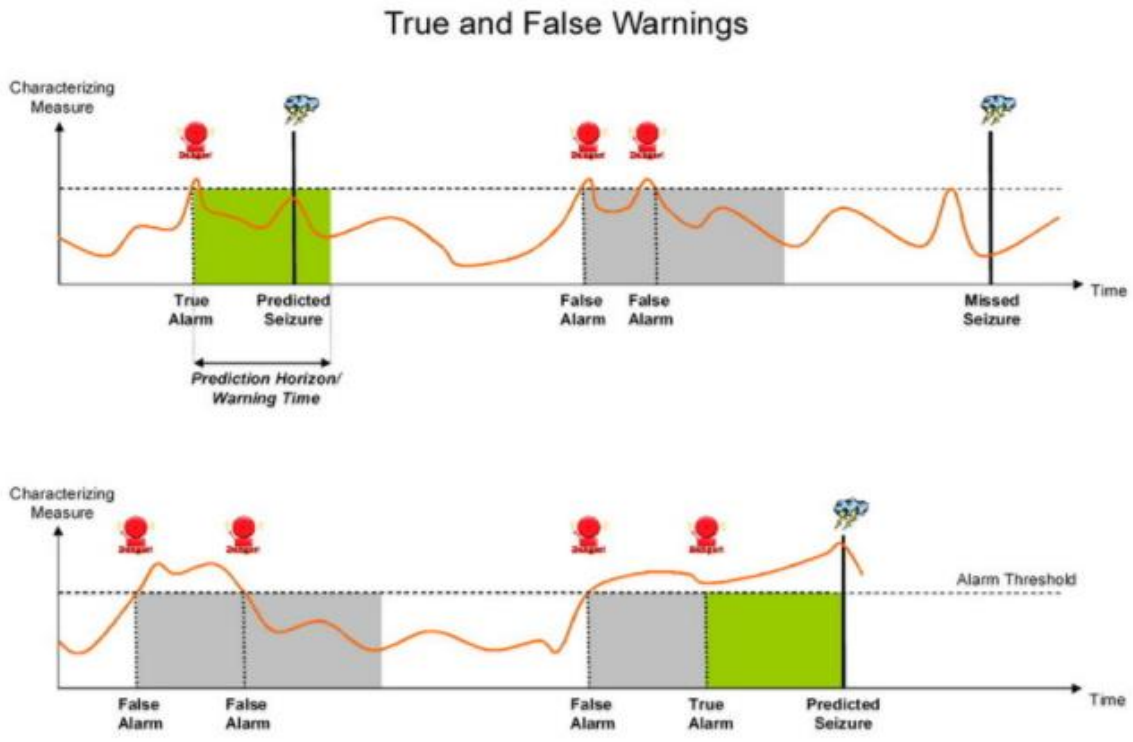
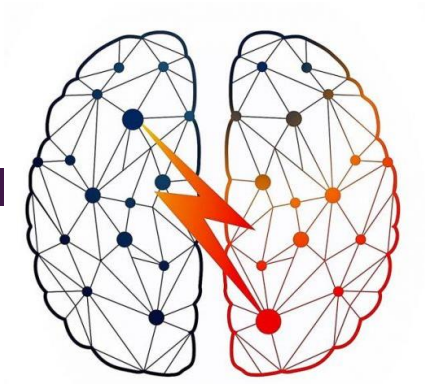


Figure 4.4 prediction with pre-defined threshold

4.3.3 Summary

For a general comparison between the univariate and bivariate, univariate measures are sensitive to those changes only before a seizure in relation to the period immediately preceding these changes. However, bivariate measures were found to reflect changes in dynamics on a longer time scale starting hours before a seizure. Despite various models have been proposed for seizure prediction, most of them focused on the univariate measures from individual EEG channels.

The next chapter will introduce some essential machine learning concepts that were needed in the project. Afterwards, it will use these concepts to show their role in seizure detection.



5

Machine Learning: Finding the Optimal Model

Machine learning is a data analytics technique that teaches computers to do what comes naturally to humans: learning from experience. Machine learning algorithms use computational methods to “learn” information directly from data without relying on a predetermined equation as a model. The algorithms adaptively improve their performance as the number of samples available for learning increases. From another perspective, Machine learning could be considered as a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.

Throughout this chapter, machine learning is discussed in conceptual depth to a certain extent while abstracting the mathematical models and theories from the reader so that the following chapters can be easily followed and understood.

In section 1, the importance of machine learning is illustrated. In section 2, several uses of machine learning are discussed. Followed in section 3 by the major concepts and theories of how machine learning works. In section 4 machine learning is projected upon the seizure detection problem. Eventually, in section 5 the training data-set is explained.

5.1 Importance of Machine Learning

Machine learning has several very practical applications that drive the kind of real problem solving – such as time and money savings – that have the potential to dramatically impact the solution to a certain problem. At Interactions in particular, Machine Learning introduced tremendous impact within the customer care industry, whereby machine learning is allowing people to get things done more quickly and efficiently. Through Virtual Assistant solutions, machine learning automates tasks that would otherwise need to be performed by a live agent – such as changing a password or checking an account balance. This frees up valuable agent time that can be used to focus on the kind of customer care that humans perform best: high touch, complicated decision-making that is not as easily handled by a machine.

Machine learning has made dramatic improvements in the past few years, but it is still very far from reaching human performance. Many times, the machine needs the assistance of human to complete its task.

From another prospective, and from where the previous chapter left, Machine Learning proves its importance in efficient solving for a classification problem. In contrast to the “fixed” threshold detection mentioned earlier, machine learning is able to provide more “flexible” and “adaptive” classification parameters. These specs allow machine learning algorithms for classification to provide better performance and results.

5.2 Uses of Machine Learning

Financial services

Banks and other businesses in the financial industry use machine learning technology for two key purposes: to identify important insights in data and prevent fraud. The insights can identify investment opportunities, or help investors know when to trade.

Government

Government agencies such as public safety and utilities have a particular need for machine learning since they have multiple sources of data that can be mined for insights. Analyzing sensor data, for example, identifies ways to increase efficiency and save money.

Health care

Machine learning is a fast-growing trend in the health care industry, thanks to the advent of wearable devices and sensors that can use data to assess a patient's health in real time. The technology can also help medical experts analyze data to identify trends or red flags that may lead to improved diagnoses and treatment.

Marketing and sales

Websites recommending items you might like based on previous purchases are using machine learning to analyze your buying history – and promote other items you'd be interested in.

Oil and gas

Finding new energy sources. Analyzing minerals in the ground. Predicting refinery sensor failure. Streamlining oil distribution to make it more efficient and cost-effective. The number of machine learning use cases for this industry is vast – and still expanding.

Transportation

Analyzing data to identify patterns and trends is key to the transportation industry, which relies on making routes more efficient and predicting potential problems to increase profitability.

5.3 How it works?

Machine learning uses two types of techniques: supervised learning, which trains a model on known input and output data so that it can predict future outputs, and unsupervised learning, which finds hidden patterns or intrinsic structures in input data.

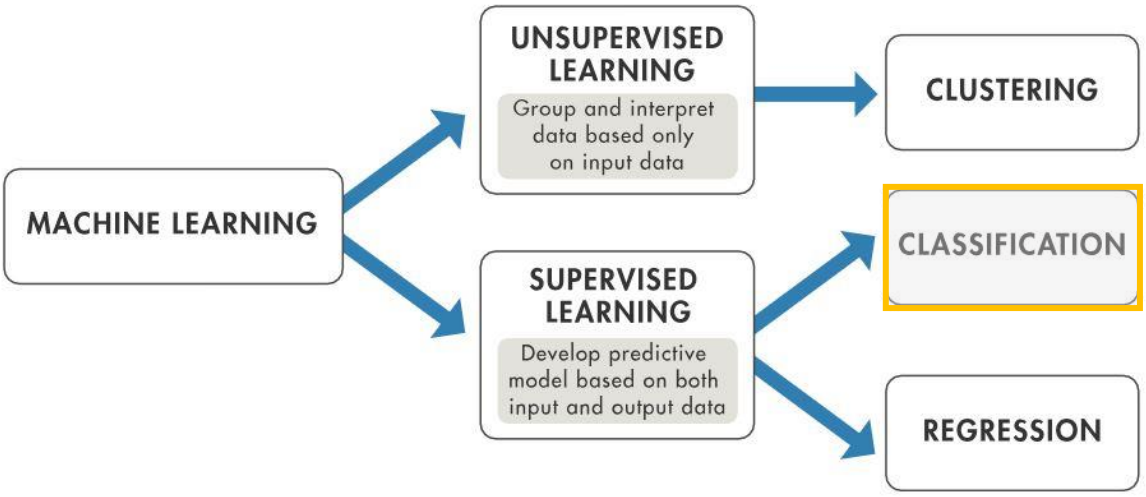


Figure 5.1 Machine learning techniques include both unsupervised and supervised learning

5.3.1 Supervised Learning

Supervised machine learning builds a model that makes predictions based on evidence in the presence of uncertainty. A supervised learning algorithm takes a known set of input data and known responses to the data (output/labels) and trains a model to generate reasonable predictions for the response to new data.

Supervised learning uses classification and regression techniques to develop predictive models. Where in the context of this project, classification is applied to seizure detection problem to classify between seizure and none seizure points from the processing of the EEG signal. Classification techniques predict discrete responses—for example, whether an email is genuine or spam, or whether a tumor is cancerous or benign. Classification models classify input data into categories. Typical applications include medical imaging, speech recognition, and credit scoring. Classification is best employed if the data can be tagged, categorized, or separated into specific groups or classes.

Common algorithms for performing classification include support vector machine (SVM)- discussed in chapter 6- , boosted and bagged decision trees, k-nearest neighbor, Naïve Bayes, discriminant analysis, logistic regression, and neural networks.

Regression techniques predict continuous responses—for example, changes in temperature or fluctuations in power demand. Typical applications include electricity load forecasting and algorithmic trading. Regression -on the other hand- is best employed when working with a data range or if the nature of the responses is a real number, such as temperature or the time until failure for a piece of equipment.

5.3.2 Unsupervised Learning

Unsupervised learning finds hidden patterns or intrinsic structures in data. It is used to draw inferences from datasets consisting of input data without labeled responses.

Clustering is the most common unsupervised learning technique. It is used for exploratory data analysis to find hidden patterns or groupings in data. Applications for cluster analysis include gene sequence analysis, market research, and object recognition.

For example, if a cell phone company wants optimize the locations where they build cell phone towers, they can use machine learning to estimate the number of clusters of people relying on their towers. A phone can only talk to one tower at a time, so the team uses clustering algorithms to design the best placement of cell towers to optimize signal reception for groups, or clusters, of their customers.

Common algorithms for performing clustering include k-means and k-medoids, hierarchical clustering, Gaussian mixture models, hidden Markov models, self-organizing maps, fuzzy c-means clustering, and subtractive clustering.

5.3.3 How Do You Decide Which Machine Learning Algorithm to Use?

Choosing the right algorithm can seem overwhelming—there are dozens of supervised and unsupervised machine learning algorithms, and each takes a different approach to learning.

Here are some guidelines on choosing between supervised and unsupervised machine learning: Choose supervised learning if you need to train a model to make a prediction--for

example, the future value of a continuous variable, such as temperature or a stock price, or a classification—for example, identify makes of cars from webcam video footage.

Choose unsupervised learning if you need to explore your data and want to train a model to find a good internal representation.

5.4 Machine Learning and Seizure Detection

Seizure Detection problem is clearly a classification problem, where features extracted from the EEG signal need to be classified into either seizure or non-seizure point and hence take a suitable action. This is where machine learning is obviously present to introduce better performance.

5.5 Dataset

As mentioned in previous sections, Supervised Machine Learning problems include the data entries labels (or corresponding classes) for the training process. There exist several epilepsy datasets in which they offer the EEG signals collected from different patients with corresponding information about seizure periods. This dataset is fed to the SVM training algorithm to be able to test and measure performance of the training algorithm.

This database, collected at the Children's Hospital Boston, consists of EEG 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.

23 cases were taken from both genders with each case containing between 9 and 42 continuous EDF files from a single subject.

EDF is a simple and flexible format for exchange and storage of multichannel biological and physical signals. In most cases, the EDF files contain 1 hour of digitized EEG signals retrieved from 23 channels. See figure 5.2.

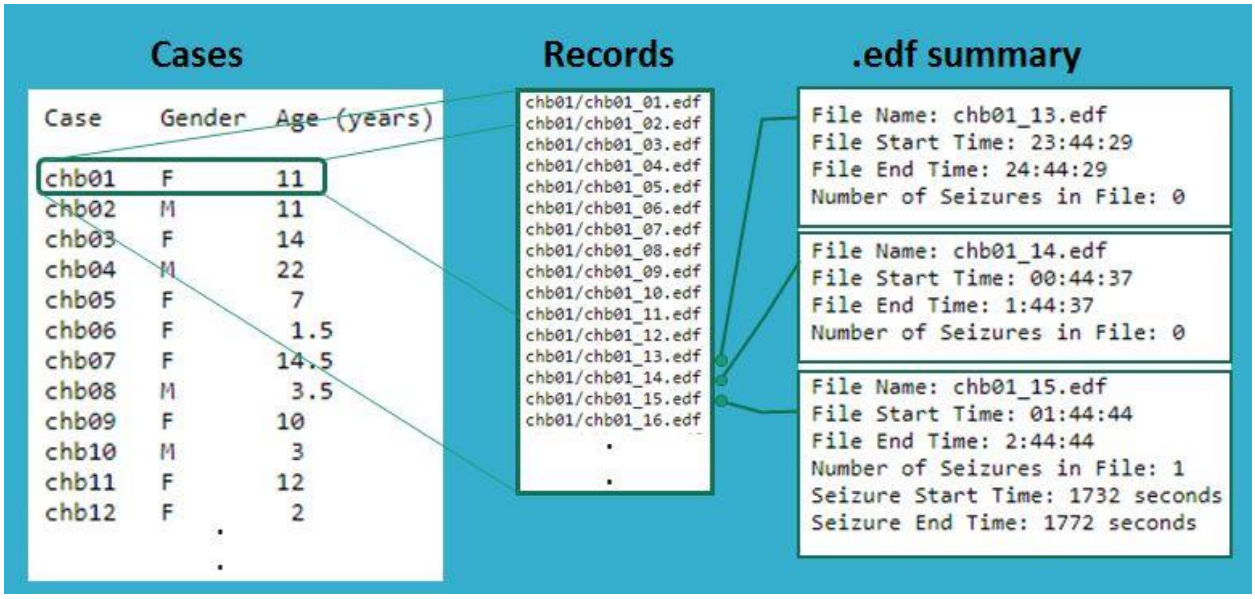
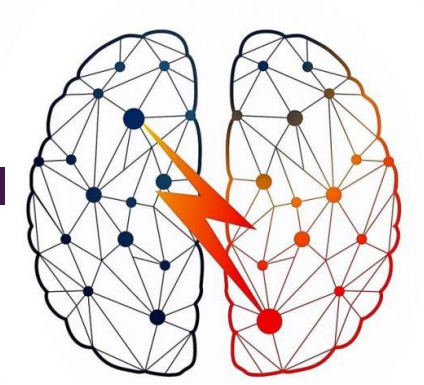


Figure 5.2 Dataset Details

The dataset we used was an essential part in the project flow. The dataset is the core thing in which we classify seizure vs. non-seizure points using the Support Vector Machine (SVM). Hence, the dataset helped us in successfully obtaining accurate classification results. It is noteworthy to mention that ONE LAB is considering obtaining their own dataset to be used in similar projects in the foreseeable future in order to have more accurate results.

The following chapter will focus on defining the Support Vector Machine (SVM) classifier which is a very important topic in the classification between seizure and non-seizure EEG signals.



6

Support Vector Machines: Large margin classifiers

This chapter's focus is the SVM. There are two sections in this chapter. Section 1 defines the SVM's hyperplane and its role in classification. Section 2 illustrates how the maximum margin that is needed to perform classification is computed.

6.1 SVM's Hyperplane

The support vector machine SVM is a machine learning algorithm used to obtain the optimized maximum margin around a hyperplane needed for classification process. That's why SVM is called "large margin classifier". Maximum margin defines the maximum distance between the hyperplane center and the closest point to the plane which define the support vector (Xn). The support vector plays a significant role in supporting the plane as they participate in the definition of the separation hyperplane and in achieving the margin by computing the maximum distance between the Xn points and the hyperplane.

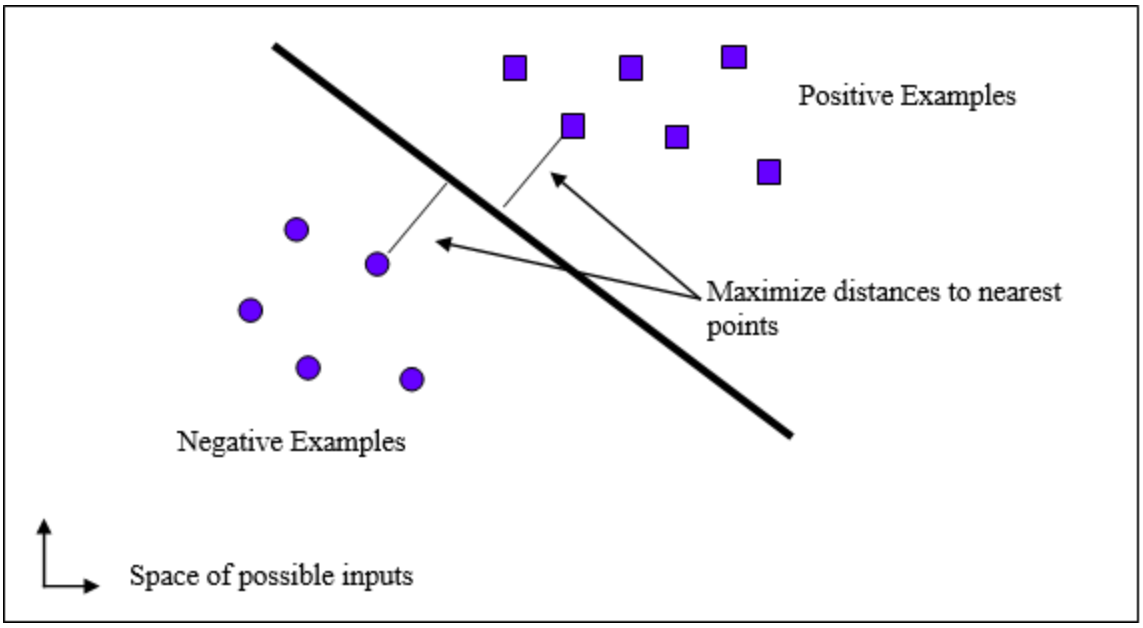


Figure 6.1 Linear Support Vector Machine

SVM is a hyperplane that separates a set of positive examples from a set of negative examples which represents with maximum margin (see figure 6.1). In the linear case, the margin is defined by the distance of the hyperplane to the nearest of the positive and negative examples.

6.2 Computing the maximum margin

The formula for the output of a linear SVM is

$$u = \vec{w} \cdot \vec{x} - b, \tag{6.1}$$

Where w is the normal vector to the hyperplane and x is the input vector. The separating hyperplane is the plane u=0. The nearest points lie on the planes $u = \pm 1$.

The margin m is thus

$$m = \frac{1}{||w||_2}. \tag{6.2}$$

Maximizing margin can be expressed via the following optimization problem:

$$\min_{\vec{w},b} \tfrac{1}{2}||\vec{w}||^2 \text{ subject to } y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1, \forall i, \tag{6.3}$$

Where xi is the ith training example and yi is the correct output of the SVM for the ith training example. The value yi is +1 for the positive examples in a class and –1 for the negative examples. [9]

6.3 The kernel

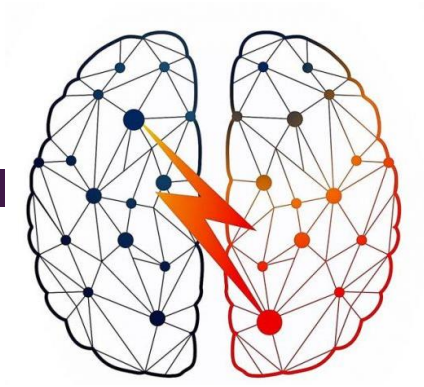
In machine learning, kernel methods are a class of algorithms for pattern analysis, whose best-known member is the support vector machine (SVM). The general task of pattern analysis is to find and study general types of relations (for example clusters, correlations, classifications) in datasets. For many algorithms that solve these tasks, the data in raw representation have to be explicitly transformed into feature vector representations via a user-specified feature map: in contrast, kernel methods require only a user-specified kernel, i.e., a similarity function over pairs of data points in raw representation. [16]

Kernel methods owe their name to the use of kernel functions, which enable them to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between the images of all pairs of data in the feature space. This operation is often computationally cheaper than the explicit computation of the coordinates. This approach is called the "kernel trick". Kernel functions have been introduced for sequence data, graphs, text, images, as well as vectors.

Some of the popular kernels:

- Fisher kernel
- Graph kernels
- Kernel smoother
- Polynomial kernel
- Radial basis function kernel (RBF)
- String kernels

The next chapter will elaborate two SVM algorithms that are useful for the classification problem which are Sequential Minimal Optimization (SMO) and Gilbert’s algorithm. They are related to seizure detection not seizure prediction.



7

Training of Support Vector Machine

This chapter has two main sections. Section 1 describes in details the SMO algorithm. Likewise, chapter 2 illustrates deeply Gilbert's algorithm. It is noteworthy to say that Gilbert's algorithm was chosen to be implemented in our project as the SVM classifier.

7.1 SMO Algorithm

Sequential Minimal Optimization or SMO is an algorithm for training support vector machine. Training a Support Vector Machine (SVM) requires the solution of a very large quadratic programming (QP) optimization problem. QP is an example of optimization problem that takes a certain form. Solving QP must identify the inequality and equality constrains "upper and lower bound". However, problems can get every large with thousands of variables and constrains, change over time this is especially important in real time optimization and take much time for calculations.

Not to mention that the quadratic form involves a matrix that has a number of elements equal to the square of the number of training examples. This matrix cannot be fit into 128 Megabytes if there are more than 4000 training examples.

Since SMO breaks the large QP into smaller QP problem that can be solved analytically in QP steps. Therefore, it avoids the calculation of matrices as known in the equation of QP in MATLAB. SMO scales somewhere between linear and quadratic unlike chunking SVM that scales

between linear and cubic. Therefore, the SMO was considered a suitable algorithm for classification as it's faster.

7.1.1 Find and choose alpha

First, we need to find alpha using Lagrange equation which is the QP problem that the SMO algorithm will solve:

$$\min_{\vec{\alpha}} \Psi(\vec{\alpha}) = \min_{\vec{\alpha}} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j K(\vec{x}_i, \vec{x}_j) \alpha_i \alpha_j - \sum_{i=1}^{N} \alpha_i$$

Then, we choose alpha that violates KKT conditions:

$$\begin{aligned} \alpha_i = 0 &\Leftrightarrow y_i u_i \geq 1, \\ 0 < \alpha_i < C &\Leftrightarrow y_i u_i = 1, \\ \alpha_i = C &\Leftrightarrow y_i u_i \leq 1. \end{aligned}$$

C is the penalty of error, which we as designers choose its value to see how much the tolerance is we can work with. If this C is too large, we get back to the ideal case where there is no margin.

7.1.2 Choosing Lagrange multiplier α_2

$$\min_{\vec{w}, b, \vec{\xi}} \frac{1}{2} ||\vec{w}||^2 + C \sum_{i=1}^{N} \xi_i$$

By choosing α_1 and calculating its error E1, Lagrange multiplier α_2 is chosen from the inner loop of the algorithm. We choose Lagrange multiplier $\alpha_2$that maximizes the error difference from the set of alphas. Choosing Lagrange multiplier α_2 that satisfies UY=1 & $0 < \alpha_2 < C$.

7.1.3 Optimizing Lagrange multiplier α1 & α2

Because there are only two multipliers, the constraints can be easily displayed in two dimensions like the figure shown below. After choosing Lagrange multiplier that defines the inequality constraints, we need to set lower and higher limit of a box that defines the equality constraint. The bound constraints cause the Lagrange multipliers to lie within a box, while the linear equality constraint causes the Lagrange multipliers to lie on a diagonal line.

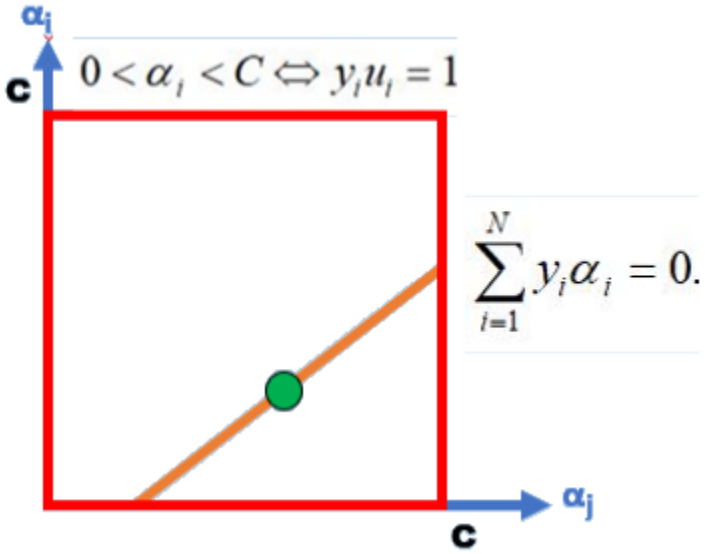


Figure 7.1 the constraints displayed in two dimensions

We will choose α_2 that satisfies the minimum objective function.

For y1 not equal y2, $L = \max(0, \alpha_2 - \alpha_1), \qquad H = \min(C, C + \alpha_2 - \alpha_1).$

For y1 equal y2, $L = \max(0, \alpha_2 + \alpha_1 - C), \qquad H = \min(C, \alpha_2 + \alpha_1).$

The second derivative of the objective function along the diagonal line can be expressed as:

$$\eta = K(\vec{x}_1, \vec{x}_1) + K(\vec{x}_2, \vec{x}_2) - 2K(\vec{x}_1, \vec{x}_2).$$

Under normal circumstances, the objective function will be positive definite, there will be a minimum along the direction of the linear equality constraint, and η will be greater than zero. In this case, SMO computes the minimum along the direction of the constraint:

$$\alpha_2^{\text{new}} = \alpha_2 + \frac{y_2(E_1 - E_2)}{\eta},$$

As a next step, the constrained minimum is found by clipping the unconstrained minimum to he ends of the line segment:

$$\propto_2^{\text{new, clipped}} = \begin{cases} H \ , \ if \quad \propto_2^{\text{new}} \geq \text{H}; \\ \propto_2^{\text{new}} , \ if \quad L \leq \propto_2^{\text{new}} \leq \text{H}; \\ L \ , \ if \quad \propto_2^{\text{new}} \leq L. \end{cases}$$

The value of α_1 is computed from the new, clipped, α_2: $\alpha_1^{\text{new}} = \alpha_1 + s(\alpha_2 - \alpha_2^{\text{new,clipped}})$.

Thus, we have chosen the optimized α_2 that satisfies the minimum objective function which will be the nearest point to the center of the contour lines of the objective function.

The classification can now be accomplished by computed the weighting vector W, the threshold b and updating the hyperplane equation.

$$\vec{w} = \sum_{i=1}^{N} y_i \alpha_i \vec{x}_i, \quad b = \vec{w} \cdot \vec{x}_k - y_k \text{ for some } \alpha_k > 0.$$

Then, we return to the data set and choose the points that satisfy the hyperplane equation and its UY >1 to set its alpha equal to zero. Now, we succeeded to exclude the points that have their Lagrange multiplier equal zero and thus can be excluded from the next iterations.

We apply the KKT conditions again for the next iterations till all alphas satisfy the KKT conditions and we reach the optimized hyperplane equation.

7.1.4 SMO Block diagram

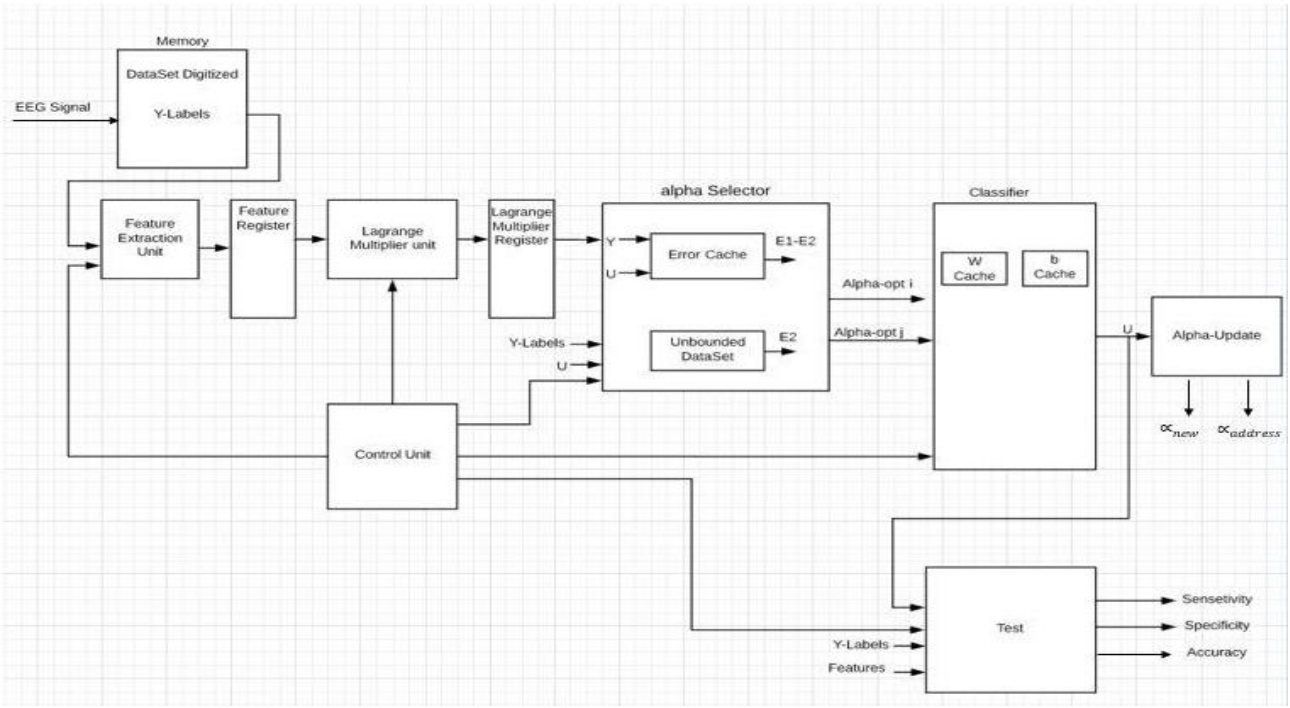


Figure 7.2 SMO Block Diagram

7.1.5 SMO Performance Results

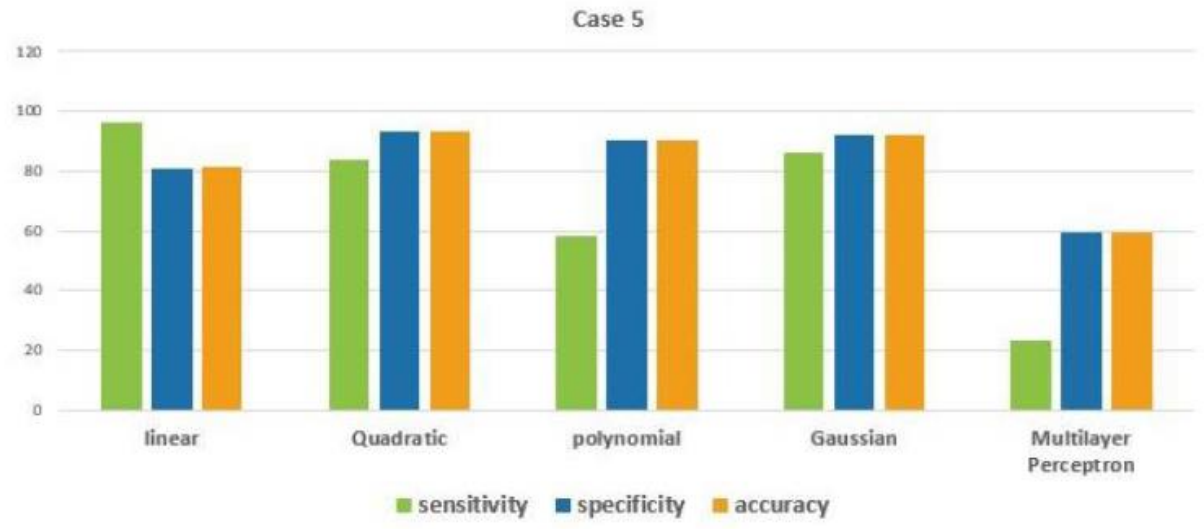


Figure 7.3 Different Kernel Performance of SMO for patient 5

7.2 Gilbert's Algorithm

In 1966, Gilbert algorithm was introduced to the machine learning field. In 2000, it has been used as a support vector machine but it had slow convergence time. In 2005, Gilbert had been modified to solve this slow rate of convergence and this is the algorithm that will be used the seizure detection. As we mentioned before that the Support Vector Machines are obtained by solving a constrained quadratic programming problem (SVM QP). Since the SVM QP problem is often too large for standard solvers, SVM specific training algorithms are used which provides decomposition of the full SVM QP problem into subproblems. Such algorithms are SMO (mentioned in the previous sections) and Gilbert's algorithm. Recently, various research works approach the SVM training from a geometric view of the problem. These proposed methods are based on the application of a nearest point algorithm "Gilbert's Algorithm" to the geometric expression of the SVM training problem.

Since the objective of SVM is to construct a separating hyperplane $w \cdot x - b = 0$ to attain maximum separation between the classes as shown in figure (7.4).

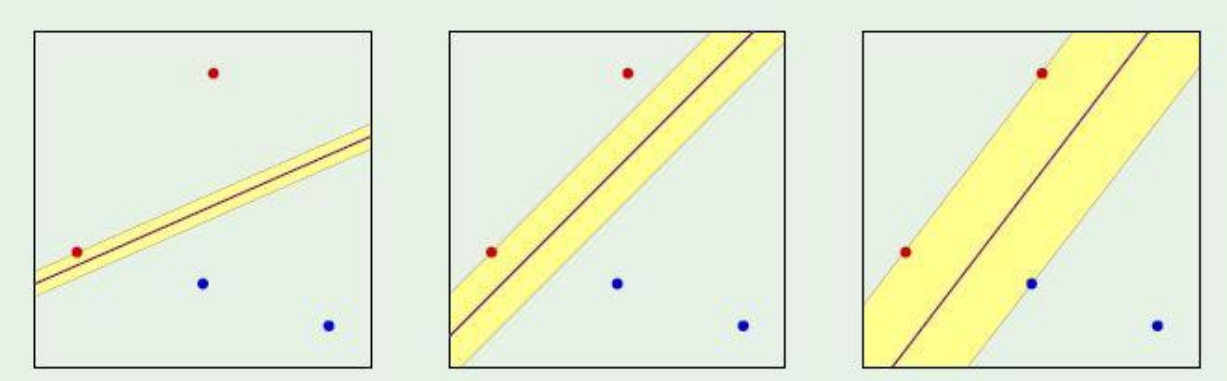


Figure 7.4 From left to right, worst to best separating hyperplane (according to the maximum margin)

Gilbert's algorithm uses the concept of Minkowski set difference. Given two convex representing the two classes of positive and negative seizures, the normal to the separating hyperplane $\frac{2}{||w*||}$ can be obtained ||u*-v*|| as both points u* and v* are the closest points on the two convex of the two classes as shown in figure (7.5).

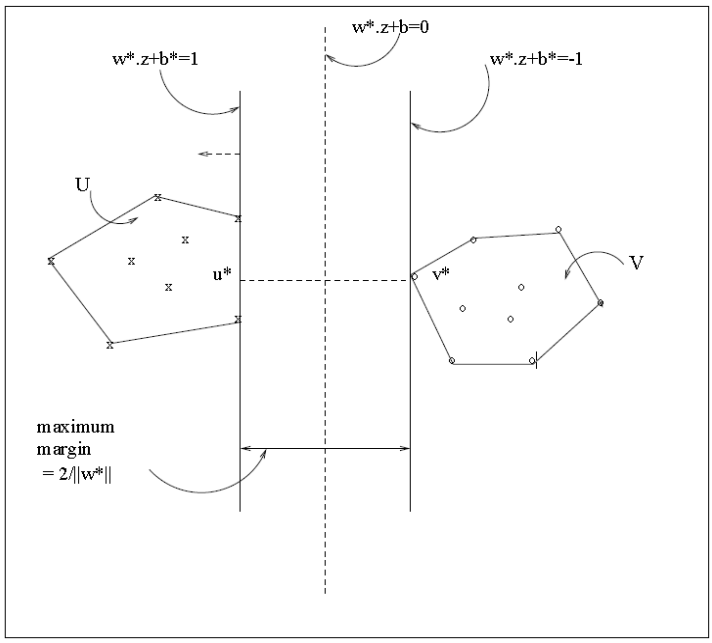


Figure 7.5 The convex hull of the two classes

The problem of finding the minimum distance between two convex hulls is known as the nearest point problem (NPP). Gilbert's algorithm is one of the first algorithms suggested for solving NPP. It is applied on the secant convex hull S which denotes the Minkowski set difference of U and V, where U and V are the convex hulls of each class of training data.

$S= \{s: s = u - v, u \in U, v \in V\}: Y_u = 1, Y_v = -1.$

The solution to the SVM problem is the point s*, which belongs to the secant convex hull's perimeter and is closest to the origin. Gilbert's Algorithm locates the point of a convex hull closest to the origin with recurring linear steps. knowing that s*=u*- v* as shown in figure (7.6).

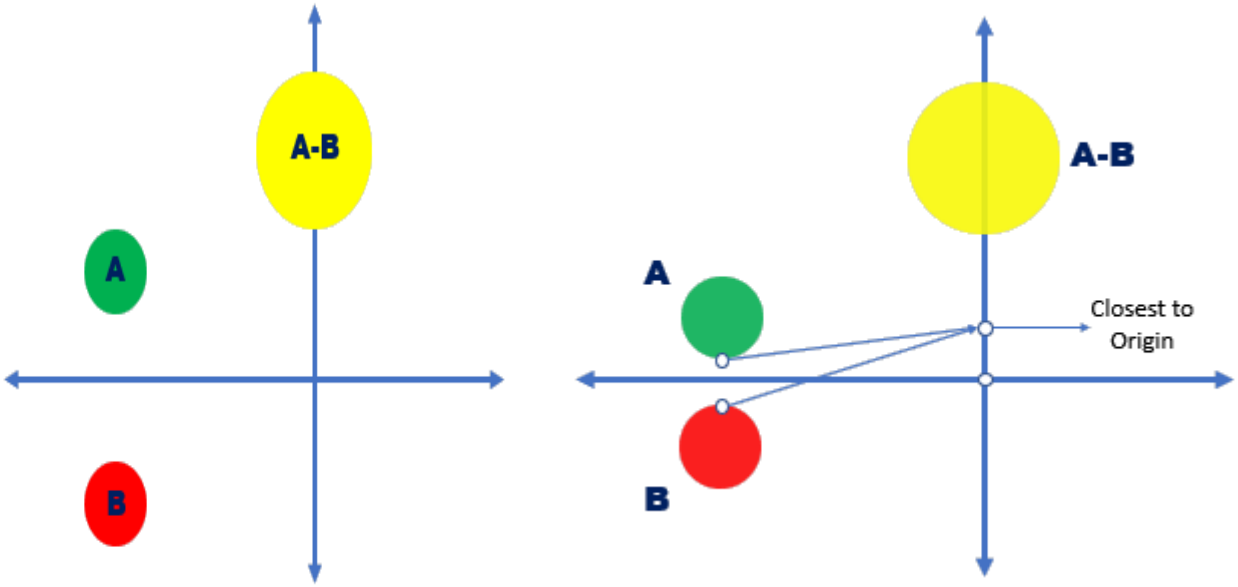


Figure 7.6 Minkawski of un-intersected shapes

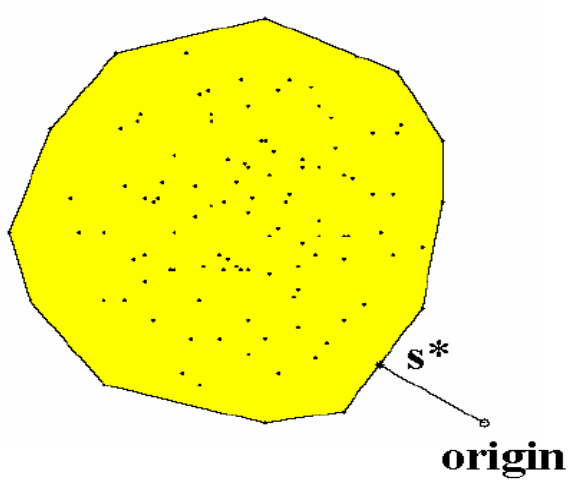


Figure 7.7 secant convex hull S which denotes the Minkowski set difference of U and V

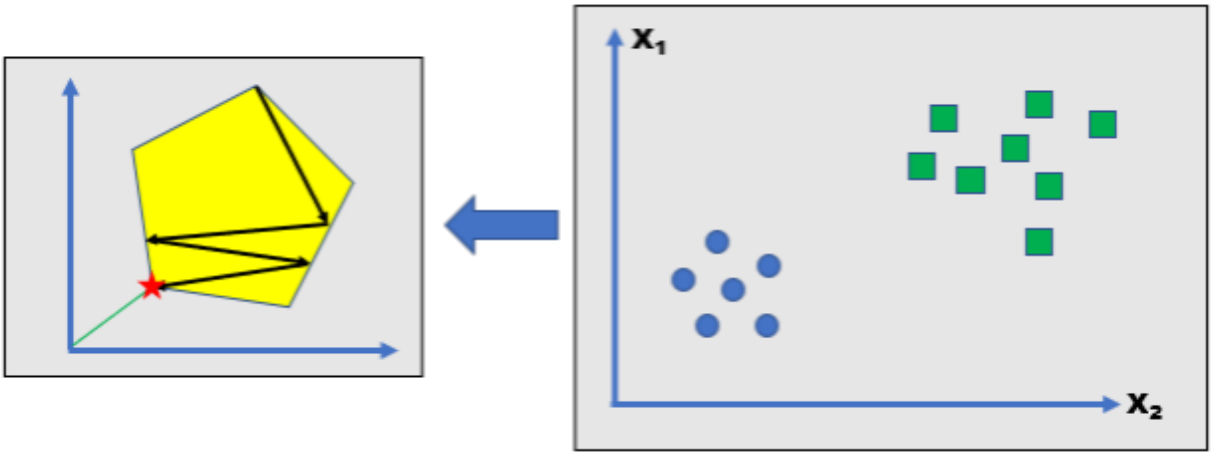


Figure 7.8 Linear steps iterations for reaching the point closest to the origin

7.2.1 How does the algorithm work?

The algorithm stars form a random point w_{k-1}, where k is defined as the total number of iterations, then it allocates the point g*(- w_{k-1}), whose projection in the direction of -w_{k-1} is the closest to the origin. This point lies on the secant's perimeter.

The goal is to find a point that lie on this segment and can be the closest to the origin. Therefore, there are three cases for this point, it can be the old point w_{k-1}, the new point g*(- w_{k-1}) or a point that lie on the segment between the two mentioned points. In order to identify which case, two parameters called top and bot need to be calculated.

Thus, g*(- w_{k-1}) is the point of S that maximizes the inner product with w_{k-1}. This value can be computed by finding g^*_U and g^*_V which are the points u and v of classes U and V respectively that maximize the inner products -w_{k-1} . u and w_{k-1} . v:

$$g^*(-w_{k-1}) = g^*_u(-w_{k-1}) - g^*_v(w_{k-1}).$$

Then it allocates the point w_k which lies on the segment [w_{k-1}, g*(-w_{k-1})] closet to the origin which may not be part of the secant.

$$w_k = \begin{cases} w_{k-1} & , \ if\ top \le 0 \\ g*(-w_{k-1}) & , \ if\ bot \le top \\ w_{k-1} + \lambda(g*(-w_{k-1}) - w_{k-1}) & , else \end{cases}$$

, Where: $top = -w_{k-1} . (g*(-w_{k-1}) - w_{k-1})$

$bot = ||(g*(-w_{k-1}) - w_{k-1}||^2$

$\lambda = \frac{top}{bot} < 1$

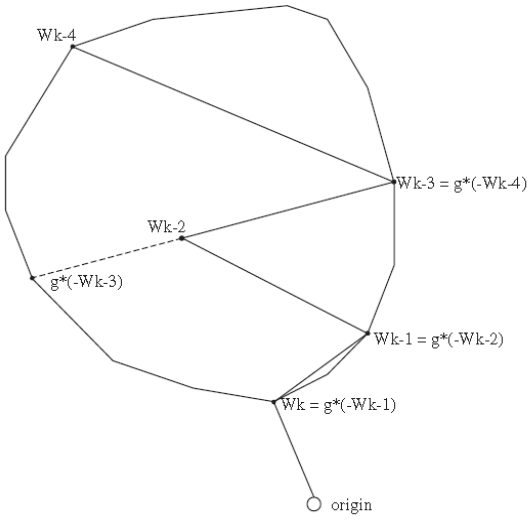


Figure 7.9 Iteration steps of Gilbert Algorithm

In the case of top less than bot which defines a point between the old and new point, a circle of radius bot and having the origin as the center could be assumed. The difference in the distance between the bot and the bot is the Lamda. Thus, from $w_{k-1,}$ a distance Lamda on the segment towards g*(- w_{k-1}) defines the point w_k. These steps are repeated till convergence.

The terminating condition of Gilbert's is selecting the same point w_k again in a following iteration.

According to our research, a modified version of Gilbert's Algorithm for the fast computation of the Support Vector Machine hyperplane was introduced by computing the angles instead of the norms. This modification seemed to converge faster to s*. This operation is done by computing $(w_k \cdot s*)/(||\, w_k||\, ||s*||)$ instead of $||\, w_k - s*||$.

7.2.2 Overview of Gilbert's Algorithm

1. Choose a point w_1 in S.
2. Identify the point $g^*(- w_1)$ in S closest to the origin in the direction of $- w_1$.
3. Identify the point w_2 on the line from w_1 to $g^*(- w_1)$ closest to the origin.
4. Repeat 2-3.
5. $s^* = \lim_{k \to \infty} wk$.

The following figure (7.10) shows the modified version of Gilbert's Algorithm.

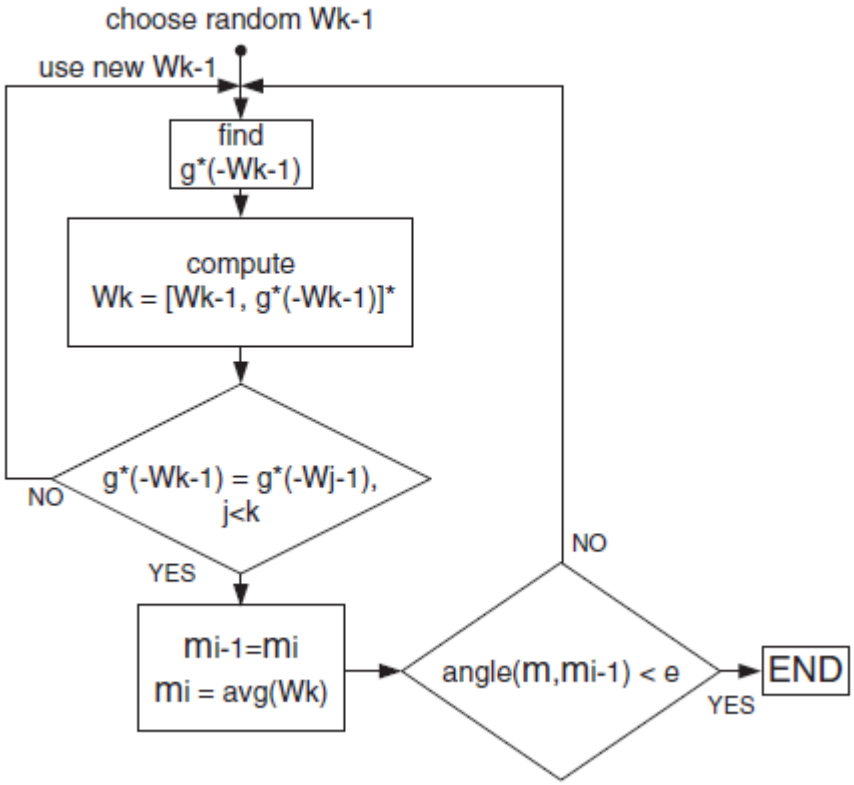


Figure 7.10 Gilbert's Algorithm Flow

7.2.3 Gilbert's Algorithm results

In order to obtain a satisfied result, a parameter "C ~" is controlled. By default, in Gilbert's Algorithm, C~ is set to '1', which gives non-promising results as less than 60% of accuracy.

With a c~ = 0.0016, a better performance is obtained. The following figure (7.11) shows the obtained statistics of different patients from the available data set and shows their sensitivity, specificity and accuracy.

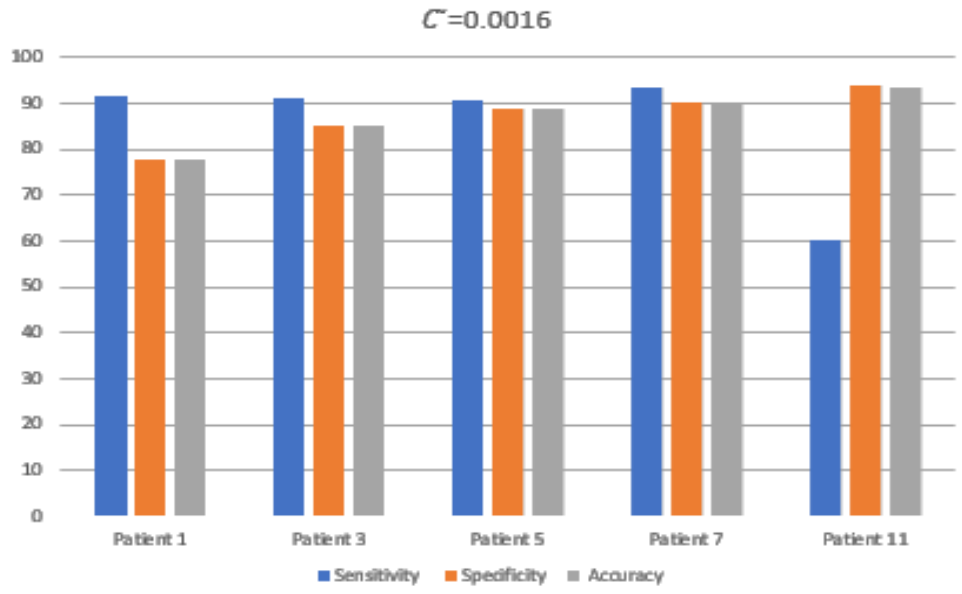
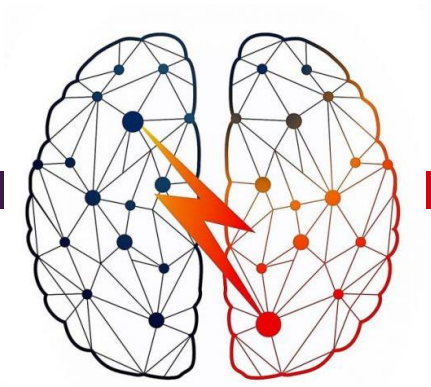


Figure 7.11 Gilbert's Results

The following chapter will explain how Gilbert's algorithm was implemented as hardware in RTL stage. In other words, it describes how the migration to hardware was done.



8

Migration to hardware

Proving the high efficiency of the MATLAB code is just the beginning. The main objective of the project is to implement a single chip, that will be implanted in the brain of the patient. To achieve this goal, all you need to do is to map your MATLAB code, to a block diagram, showing the digital logic circuit of your algorithm. register-transfer level (RTL) is a design abstraction which models a synchronous digital circuit in terms of the flow of digital signals (data) between hardware registers, and the logical operations performed on those signals. This is used in Hardware Description Language(HDL) like VHDL and Verilog, which are languages used to create high level representation of a circuit from which lower-level representations and ultimately actual wiring can be derived as we will see in the next chapter where we will show the FPGA implementation.

8.1 Block Diagram

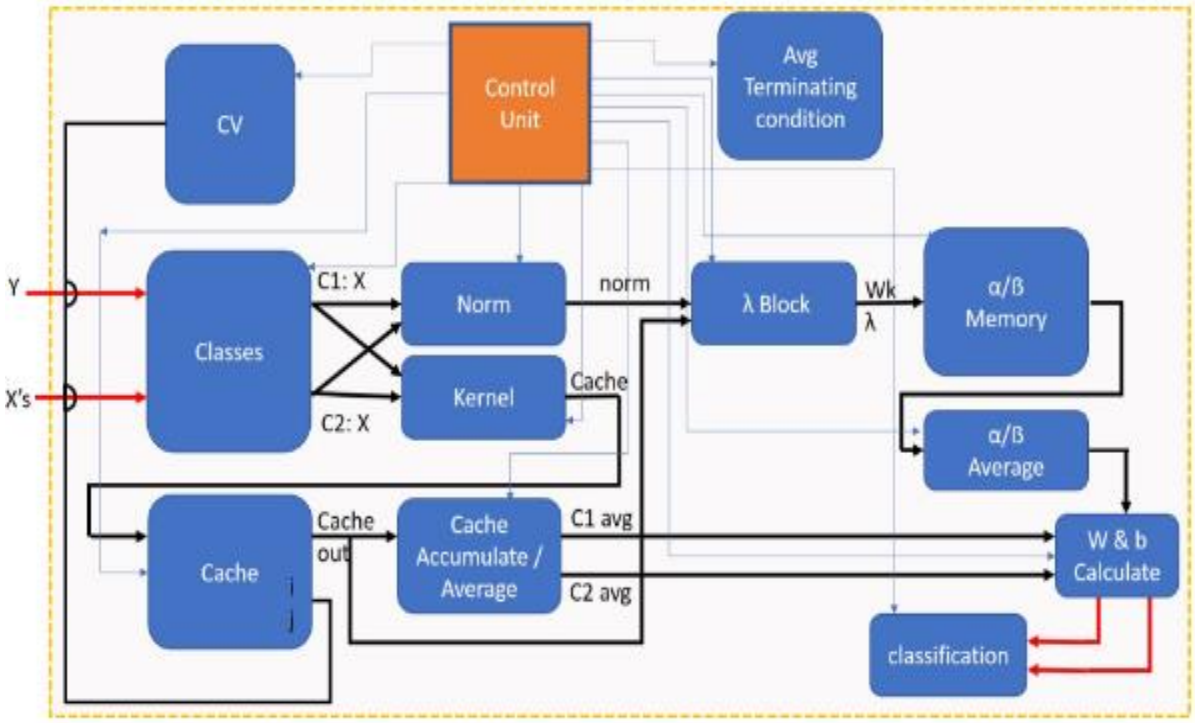


Figure 8.1 Block diagram

As shown, the block diagram is showing the memories, caches, the inputs and the outputs of each block. This is a very important step because, the design is the basic document we will get back to, when writing the VHDL code. This is why we needed to give this step the sufficient time, and make sure the design is correct.

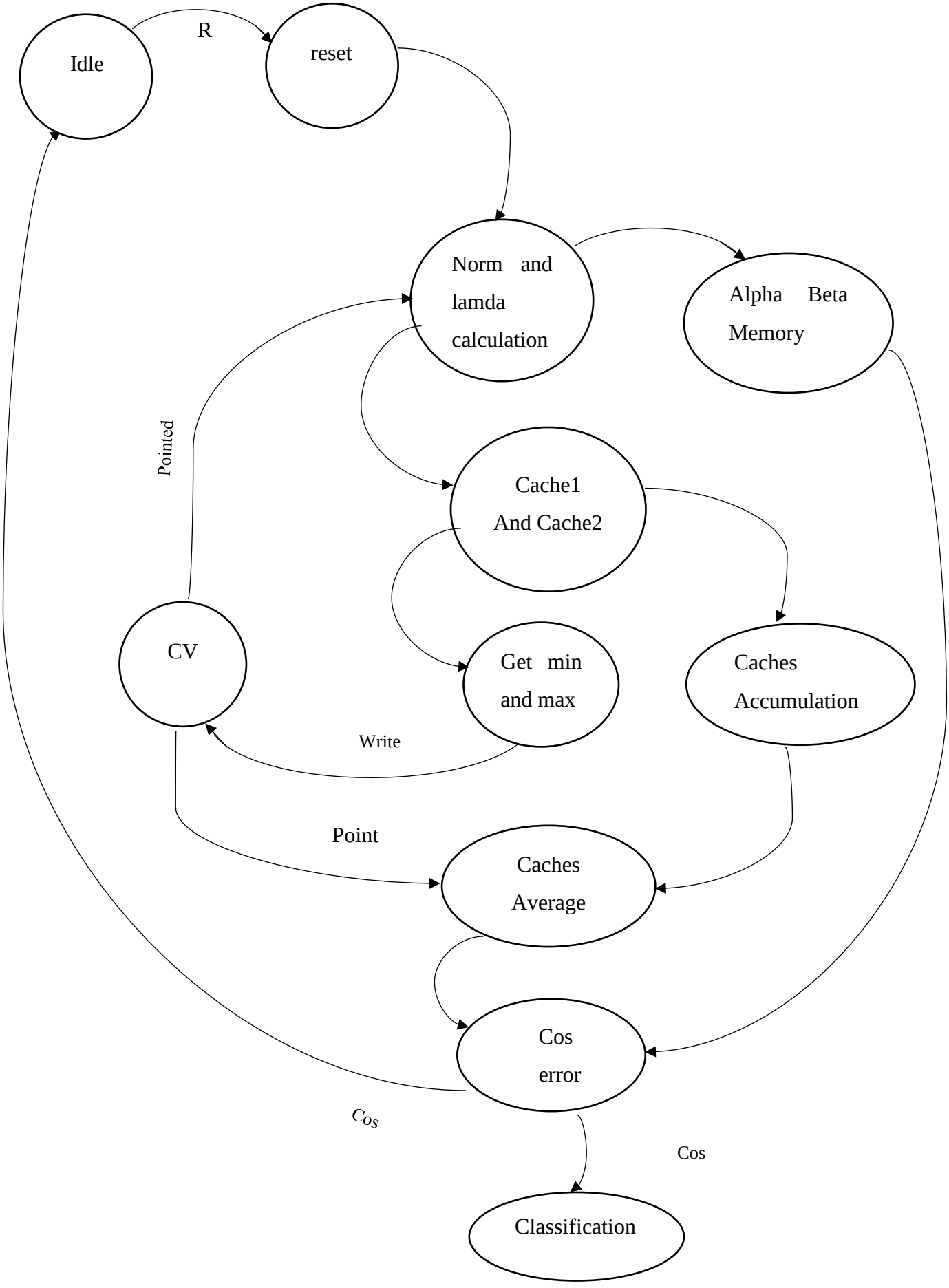
8.2 VHDL-Verilog code

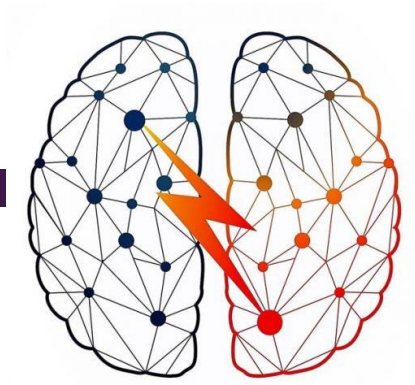
Implementing the block diagram using VHDL and Verilog was our main work for a long time. When writing the code, we had to think about the next phase which is the FPGA, so we needed to make sure that our code is synthesizable. What synthesizability means is that the code could be mapped to a hardware circuit. For example, most FPGAs do not support floating point numbers, so we had to use a trick where we scale up the floating numbers to make it an integer number that can be represented easily and then we scale down the result to understand what the real number was. Also, some operations are not synthesizable, like dividing 2 different numbers, so we couldn't just use the division sign in VHDL, but we needed to implement a divider component by ourselves to perform this task. Also, one of the toughest component is the kernel, which requires exponential

calculations. Exponential with random base and exponent is also not synthesizable, so we had to use the LUTs (Look up Tables). We also needed Finite State Machine (FSM) in our code. This is a very important topic so we will consider it in the following section.

8.3 Finite State Machine

The full code is in the appendix. The following code was tested on altera Cyclone V FPGA and checked for synthesizability.


Idle
R
reset
Norm and lamda calculation
Alpha Beta Memory
Pointed
Cache1 And Cache2
CV
Get min and max
Caches Accumulation
Write
Point
Caches Average
Cos error
Cos
Classification



9

FPGA

Field Programmable Gate Array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing; hence the term "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL), The most common HDL languages are VHDL, Verilog and System Verilog. In our design we used VHDL and Verilog. (*See appendices for design codes*).

FPGAs contain an array of programmable logic elements, and a hierarchy of reconfigurable interconnections that allow the blocks to be "wired together", like many logic gates that can be inter-wired in different configurations. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates. In most FPGAs, logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

While this chapter discusses the FPGA -which is used in this project-, it is important to mention another class of a -somehow- similar device which is ASIC. ASIC -standing for Application Specific Integrated Circuits- is similar to FPGA but it lacks the -general purpose- features. It is Integrated Circuit (IC) that is customized to perform certain task once it is fabricated.

Throughout this chapter, a more detailed comparison between FPGA and ASIC is held in section 9.1. In section 9.2 the functions of FPGA in this project design is listed and discussed. Followed in section 9.3 by an overview of the basic features and virtues that allowed the FPGA to be a potential candidate for this project. In section 9.4, the details of the used Altera FPGA device

alongside with its design specs are listed. Where eventually, in section 9.5, the process of programming the FPGA and general considerations are discussed.

9.1 FPGA Vs ASIC

The translation of the architecture into a register transfer level description in HDL allows hardware designers to consider the design at the appropriate level of abstraction. It's here that a major advantage of FPGA over ASIC can be found. It's good for prototyping and has a low volume designs as cost would be less. It's also a faster time to market. No layout and manufacturing steps needed. That's why it's preferable to map the design into FPGA before ASIC where the last needs longer design time to take care of all manufacturing steps as it's once manufactured, it would need to spin again a new chip in case of bugs. Then, the ASIC would come its role next to be better for its lower power, lower unit costs and faster than FPGA with higher performance.

9.2 Function of FPGA in the design

In this project, the need of FPGA is very critical to boost the performance of our design. As the FPGA can perform the SVM training on the data much faster than the CPU. This is due to the dividable nature of the algorithm used. The capability of the FPGA to provide a high level of parallelism is employed to solve the training problem in parallel chunks yielding much faster training.

In the development phase, Assigning the training function to the FPGA off-loads it from the CPU. Leaving the CPU to process performance measurements simultaneously using the updated parameters that the FPGA provides. This could be done by passing the continuously updated training parameters from the FPGA to the CPU, allowing for hardware acceleration.

FPGA is very suitable in this design for its ability of reconfiguration. This is employed in the algorithms methods that we have for further future expansion. (That is; to store all the training algorithms on the chip and reconfigure to the currently required as explained previously). So, it's more efficient to consider this vision in the current development phase.

The FPGA enables high processing of data, as it's needed in Gilbert algorithm to help the fast convergence of data point, hence, allowing faster response.

The concept of implanting Gilbert algorithm involves dealing with multiple memories and data accumulation, that the FPGA gives a great advantage as it support fast memory access.

9.3 FPGA virtues

The need to use Field Programmable Gate Array (FPGA) in this design was essential. An FPGA has the feature that it can be reprogrammed to perform a different function other than that it was initially intended to. FPGAs are very quick in processing our data as they have an outstanding digital processing performance. Furthermore, utilizing multichannel synchronization and processing is not a difficult task when using an FPGA. In other words, FPGA helps us adapt the design parameters to the required performance because of its beneficial re-programmability and re-configurability.

Intel FPGA is well suited in the processing of the data used with high speed and high channel density signals for two main reasons:

- ease of multichannel synchronization and processing
- impressive digital processing performance

9.4 FPGA in use (Altera)

The FPGA device used in this project was provided by Altera - an Intel business unit. DE10-Nano development board (figure 9.1) from terasiC was used to develop and test the proposed design.

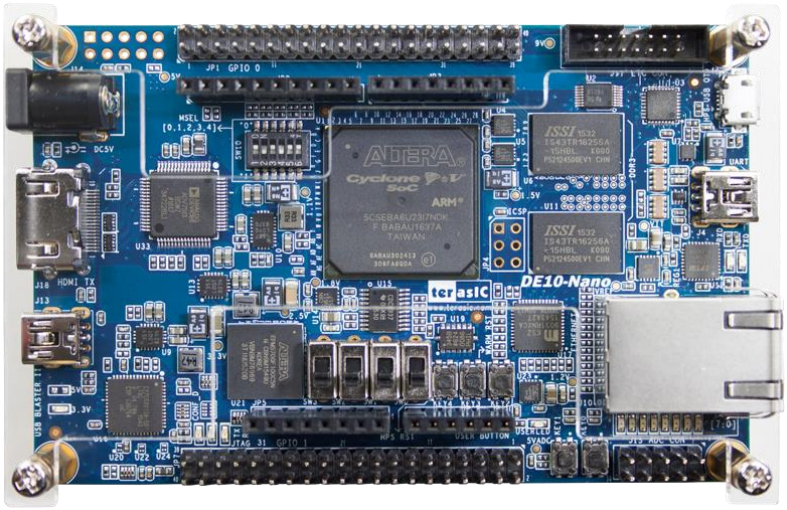


Figure 9.1 DE10-Nano Development Kit

The FPGA fabric on-board of the DE10-Nano is a Cyclone V series which offers a low cost and power consumption.

The Development kit supports not only the FPGA but offers the ability to employ HPS (Hard Processor System) in the design as well.

In the following tables some of the major specs of the FPGA and the HPS are listed to account for the device utilization in later chapters. However, it is important to note that the design of this project targeted the FPGA fabric only with no major role for the HPS.

Acquisition of Altera by Intel

On Dec. 28, 2015 – Intel Corporation a ("Intel") announced that it has completed the acquisition of Altera Corporation ("Altera") a leading provider of field-programmable gate array (FPGA) technology. The acquisition complements Intel's leading-edge product portfolio and enables new classes of products in the high-growth data center and Internet of Things (IoT) market segments.

"Altera is now part of Intel, and together we will make the next generation of semiconductors not only better but able to do more," said Brian Krzanich, Intel CEO. "We will apply Moore's Law to grow today's FPGA business, and we'll invent new products that make amazing experiences of the future possible – experiences like autonomous driving and ***machine learning."***

ALTERA®
now part of Intel

Table 9-1 Specs of DE10-Nano systems

Specs.	Hard Processor System	FPGA
	Processor Dual-core ARM* Cortex*-A9 MPCore processor at 800 MHz Neon™ media-processing engine with double-precision floating point unit 32 KB L1 instruction cache 32 KB L1 data cache 512 KB shared L2 cache Memory 64 KB on-chip SRAM 1 GB DDR3 SDRAM (32-bit data) 8 GB microSD* flash memory card **Processor I/O** 1 gigabit ethernet PHY with RJ45 connector 1 USB 2.0 On-The-Go (OTG) port, USB Micro-AB connector microSD* card interface and socket Accelerometer (I2C interface plus interrupt) UART to USB, USB Mini-B connector Warm reset button, cold reset button One user button and one user LED	**Programmable logic** Logic elements (LE): 110KLE 5,570 kilobits memory 224 18 x 19 multipliers 112 variable precision DSP blocks 6 phased-locked loops (PLL) 145 User defined I/O **FPGA I/O interfaces** 2 push buttons 4 slide switches 8 LEDs Three 50 MHz clock sources from the clock generator Two 40-pin expansion headers with diode protection One Arduino expansion header (Arduino UNO* R3 compatibility), 8-channel, 12-bit A/D converter, 500 ksps, 4-pin serial peripheral interface (SPI)

9.5 Simulation and Synthesis

The process of programming the FPGA includes several stages that allow the final design to come into light. Of course, starting with block diagram designing that marks the high-level implementation of the main entities of the design. Code generation is conducted next using VHDL/Verilog. A critical step is followed which is the gate-level simulation of the design. Using Altera-ModelSim. ModelSim simulates and shows the different waveforms of different signals and the user should verify that the output signals are correct against the input signals (figure 9.2). Testbenches could be used in this context, where they can automate the testing input scenarios and compare them to the expected outputs and assert warnings and error messages in case of error in the simulation.

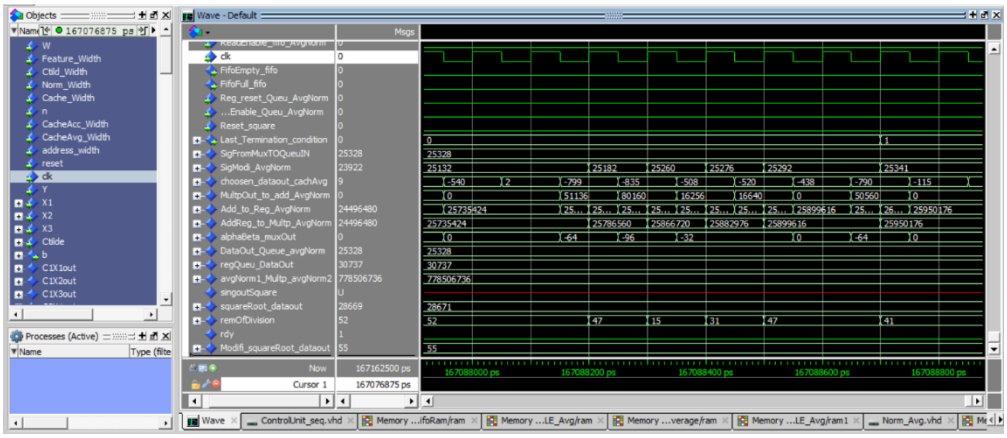


Figure 9.2 Example of waveform simulation using ModelSim

If the simulations are successful, the next stage takes place which is “Synthesis”. Synthesis is the ability for the HDL code to map into the targeted FPGA. There exist several synthesis tools. In this project Quartus Prime is used to conduct the synthesis process. Finally, after successful synthesis binary programming stream is transmitted to the FPGA programmer to program it. Testing the final design on the FPGA and repeating the loop again for optimization and meeting the performance parameters.

9.6 Power Analysis

One of the most important aspects of the design is to calculate the power of the RTL and try to optimize it as much as you need.

9.6.1 Thermal power

Thermal power is the component of total power that is dissipated within the device package. Designers need to consider the thermal power in determining whether they need to deploy thermal solutions on the FPGA, such as heat sinks, to keep the internal die-junction temperature within the recommended operating conditions.

9.6.2 Static Power

Static power is the power consumed by a device due to leakage currents when there is no activity or switching in the design. Therefore, this type of power is independent of the actual design. This data can be extracted from the FPGA device data sheet.

9.6.3 Dynamic Power

This is the power consumed through device operation caused by internal nodes in the FPGA toggling. That is, the charging and discharging of capacitive loads in the logic array and routing. The main variables affecting dynamic power are capacitance charging, supply voltage, and clock frequency. Dynamic power is design dependent and is heavily influenced by the users RTL style.

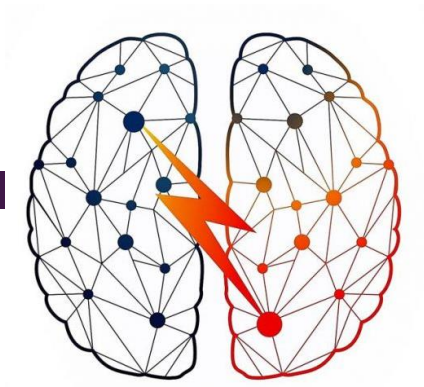
9.6.4 Power Calculations

To calculate the thermal power dissipation, all you need to do is to compile your files and go to Processing => Start => Start Power Analyzer. If you check the power analyzer report, you will find the dynamic power equals 0, which makes sense because till this point, there is no clock or any signals going through the components of the design, so there is no dynamic power.

To calculate the dynamic power, you will need to generate Value Change Dump (VCD) which is a standard file that contains all the simulation waveform information that is useful

for debugging simulation. It contains all the signals in the design. Also, you will need a testbench which simulates the clock and the initial values of all the inputs. To compile your testbench, go to Assignments => Settings => Simulation, add your testbench and mark Generate VCD. Now, the dynamic power will be generated.

The next chapter will be specified to the main results we obtained from our project. It will also include a conclusion of all what has been done throughout the entire course of the project highlighting the important results that have been reached.



10

Design Highlights

Throughout the developments of this project several design highlights stood out and deserved to be pointed out on a dedicated chapter to mark their contribution to the project development.

10.1 Fixed Point Conversion

One of the many benefits of an FPGA-based solution is the ability to implement a mathematical algorithm in the best possible manner for the problem at hand. For example, if response time is critical, then we can pipeline the stages of mathematics. But if accuracy of the result is more important, more bits can be used to ensure the achieving of the desired precision. Of course, many modern FPGAs also provide the benefit of embedded multipliers and DSP slices, which can be used to obtain the optimal implementation in the target device.

There are two methods of representing numbers within a design, fixed- or floating-point number systems. Fixed-point representation maintains the decimal point within a fixed position, allowing for straightforward arithmetic operations. The major drawback of the fixed-point system is that to represent larger numbers or to achieve a more accurate result with fractional numbers

The normal way of representing the split between integer and fractional bits within a fixed-point number is x,y where x represents the number of integer bits and y the number of fractional bits. For example, 8,8 represents 8 integer bits and 8 fractional bits, while 16,0 represents 16 integer and 0 fractional.

The number of integer bits required depends upon the maximum integer value the number is required to store, while the number of fractional bits will depend upon the accuracy of the final result. In order to add, subtract or divide, the decimal points of both numbers must be aligned. This is done by either multiply the number with more integer bits by 2X or divide the number with the fewest integer bits by 2X. Division, however, will reduce the accuracy and may lead to a result that is outside the allowable tolerance. Since all numbers are stored in base-two scaling, scaling the number up or down can be easily done shifting one place to the left or right for each power of 2 required to balance the two decimal points. Therefore, adding together two numbers that are scaled 8,8 and 9,7, the 9,7 number can be either scaled up by a factor of 21 or the 8,8 format can be scaled down to a 9,7 format, if the loss of a least-significant bit is acceptable.

10.2 Contact Vector Entity

In the development of the design main entities, Contact Vector entity was responsible to accumulate history of the processed points and check for each new point entry if it was repeated. If so, the algorithm terminates from the first termination condition.

Memory RAM Blocks were used to save the current history of the processed points. However, the highlight of the entity that when a new point is written into the CV Memory, a CV monitor entity starts searching for this point in a descending manner (i.e. from down the memory to the top). This approach showed improved convergence time, since most probably and near convergence iterations, the repeated points tend to be successive. That is both repeated points would occupy recent memory addresses. Searching the memory from top to down in such case would consume much more clock cycles to find that the repeated point was in the preceding address of the just added point.

10.3 Exponential using LUT

As mentioned in chapter 6, the Kernel used in the design is RBF kernel. RBF kernel includes the calculation of exp function. Exponential functions are often hard to implement and uses high device resources. A proposed approach to building the exp(x) entity was to find the Taylor series for the exp(x) function. However, it was found that for an accepted accuracy in the desired range of the input argument “x” up to 27 terms of the series were required! Of course, such high number

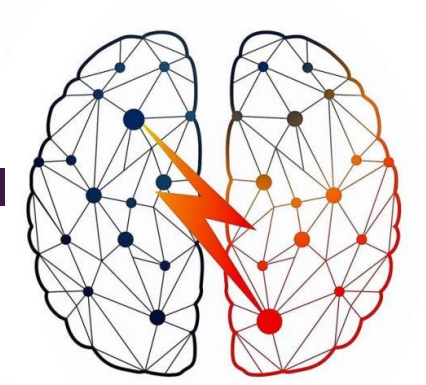
of terms would eat up most of the device resources specially the DSP blocks that would accommodate the multiplication and factorial of the different terms.

An alternative approach to this issue was to use the LUT. Look Up Tables are elements inside the FPGA that would map certain input to certain outputs. MATLAB used to generate a look up table for exp(x). Hence, the exp entity was designed such that it represents a ROM loaded with the values of exp(x) where x is interpreted as the address of the memory. In more simple way, this means that going to address “x” in the LUT memory, the value of exp(x) is stored.

This approach is much more effective resources wise were it proves better performance than the Taylor series approach, however that was traded off with more memory block consumption. A case in which the targeted FPGA total memory of around 5 Mb could accommodate the design entire memories easily.

10.4 Booth-Multiplier

The design included plenty of multiplication operations. Multiplication operations often maps to DSP slices when synthesizing. Therefore, and to not fully depend on the target DSP slices number (often not so many are there) and to give more chances for smaller area, Booth-Multipliers were used. Booth-Multipliers are multiplication entities that employ booth algorithm to find the multiplication of two inputs. Booth multipliers are known for their fast calculations, the output is calculated in combinational logic (i.e. in the same clock cycle that the inputs change, the output changes accordingly).



11

Results

In this chapter the detailed results of the project proposed algorithm and design are discussed. The results were obtained from simulating the data of 4 patients from the given dataset. Which were reduced from 10 patients. This reduction arose because of the complex feature space of the remaining 6 cases as shown in the following figure.

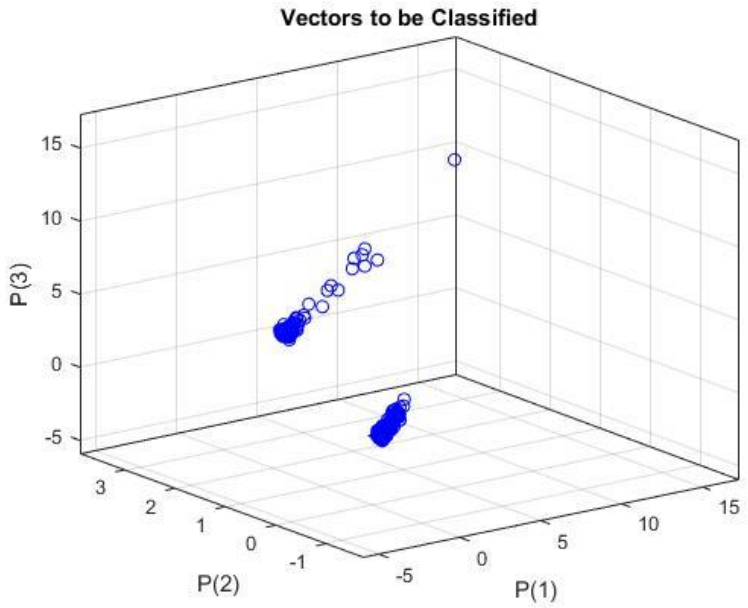


Figure 11.1 Feature space of a complex patient

The complexity of such a case, is that the seizure and none-seizure points are not easily separable -On another context both classes are nearly fully overlapping-. In such situations the algorithms performance deteriorates unexpectedly and the results they produce are not accurate enough, hence eliminated from design development cycle.

However, several solutions to overcome this problem were employed. For instance, but not limiting RBF Kernel was replaced instead of the Linear kernel which allowed for higher dimension separation. Another approach was in sweeping against a design parameter $\tilde{C}$ till producing satisfying results.

11.1 Minimum Requirements for a candidate algorithm

For the SVM Training algorithms to be a potential candidate, they must satisfy the performance parameters (previously mentioned in section 4.2.3) minimum requirement of 80% each. If the simulations (using MATLAB) fulfilled this condition, then the algorithm is qualified to start the RTL development loop.

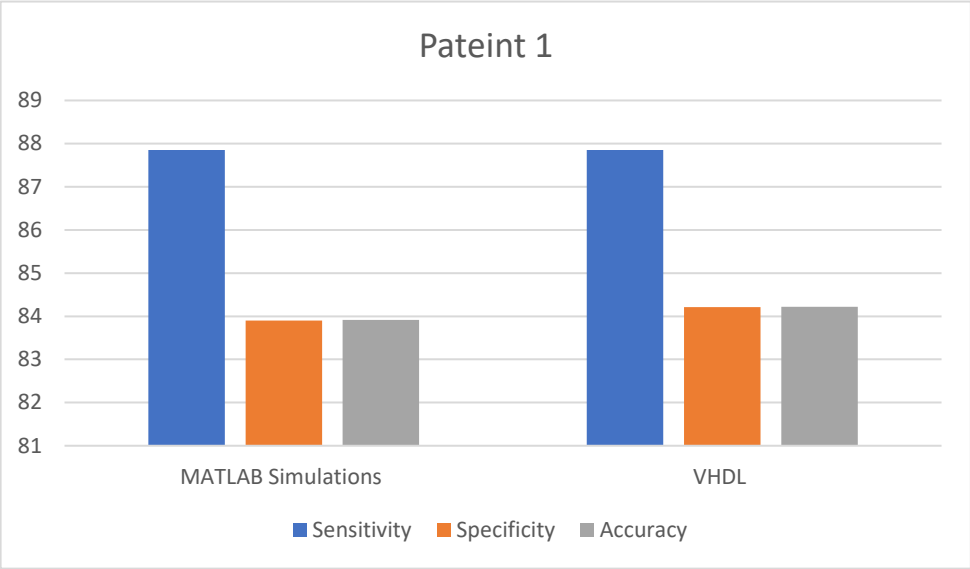
At the final stage of the RTL processes, the results of the gate-level simulations are tested and the performance parameters are re-calculated and verified to meet their MATLAB simulation counterparts within a certain percentage of accepted error. The error percentage between the MATLAB Simulations and Gate-Level simulations is present due several factors such as the fixed-point conversion step (*see Design Highlights*).

11.2 MATLAB and RTL Simulation results

Primary MATLAB simulation results are discussed in this context for different patients. Compared to the RTL simulation of the same patient cases.

It’s important to note that the RTL simulation results are obtained by feeding the training parameters that are calculated from the ModelSim RTL simulation into the MATLAB algorithm simulation.

11.2.1 Case: Patient (1)



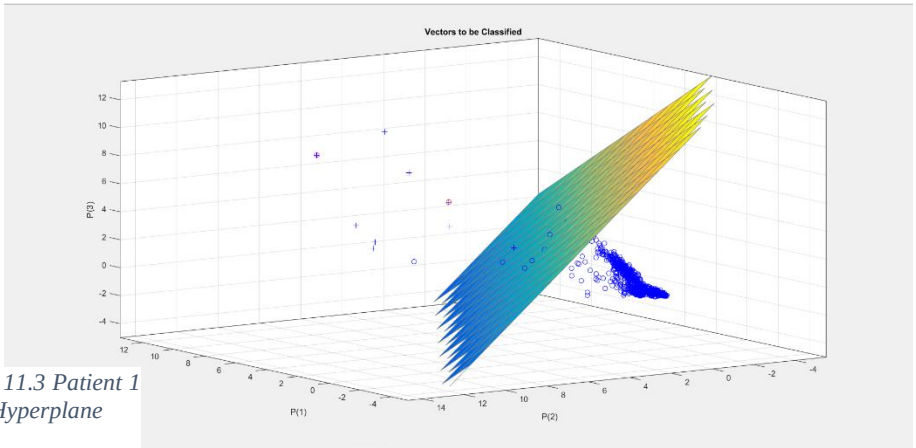


Figure 11.3 Patient 1 MATLAB Hyperplane

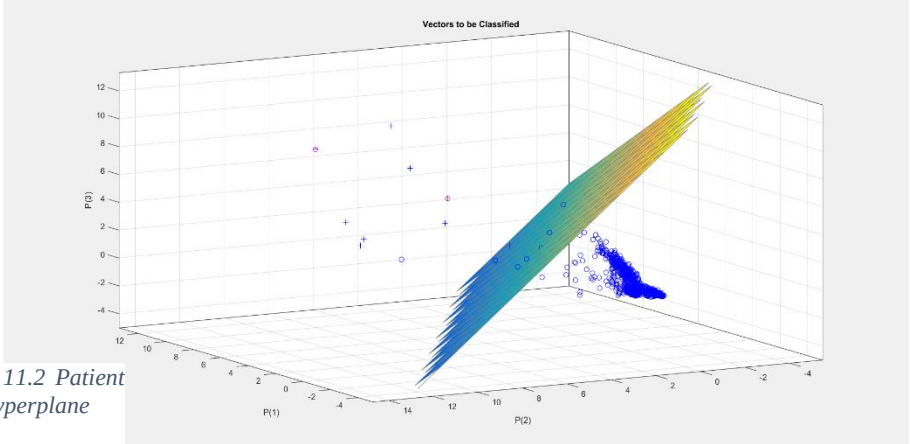
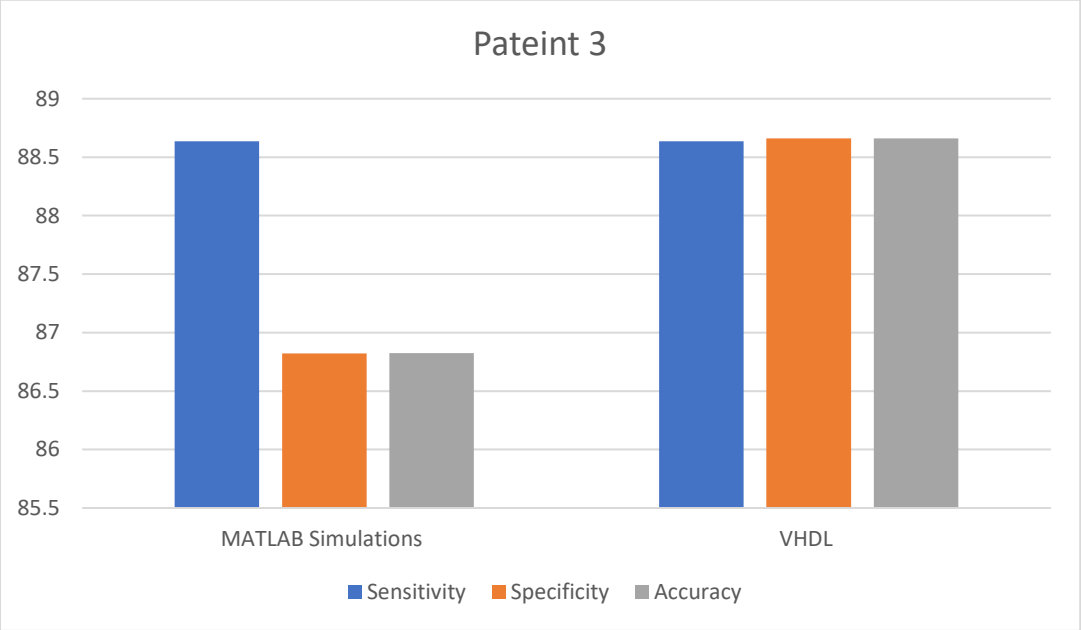


Figure 11.2 Patient 1 VHDL Hyperplane

11.2.2 Case: Patient (3)



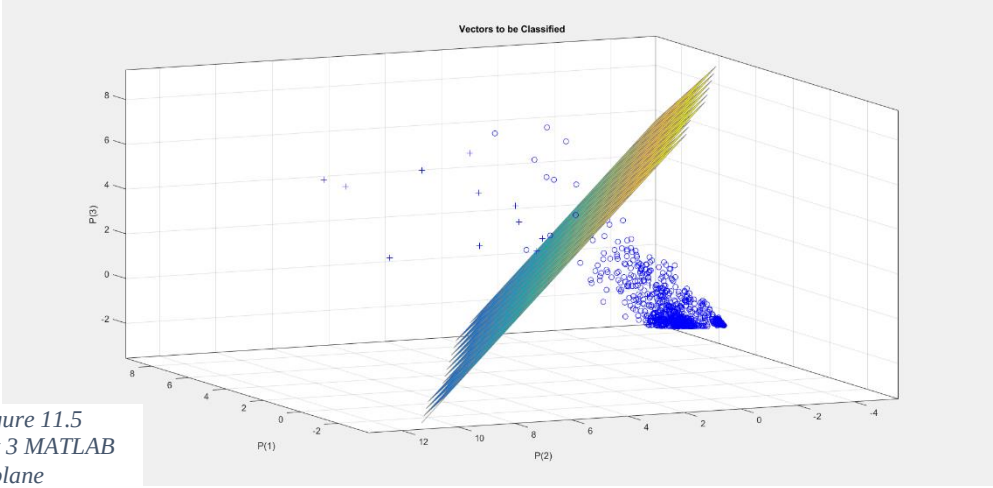


Figure 11.5 Patient 3 MATLAB Hyperplane

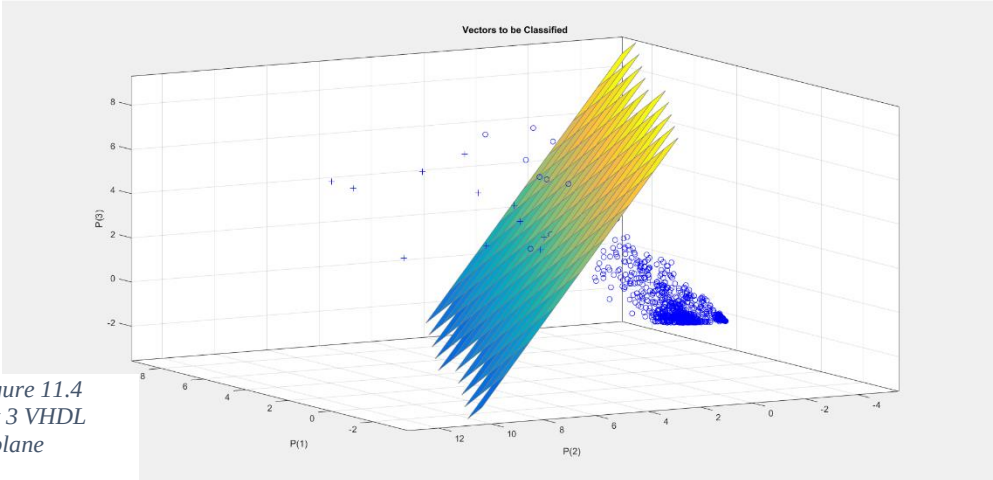
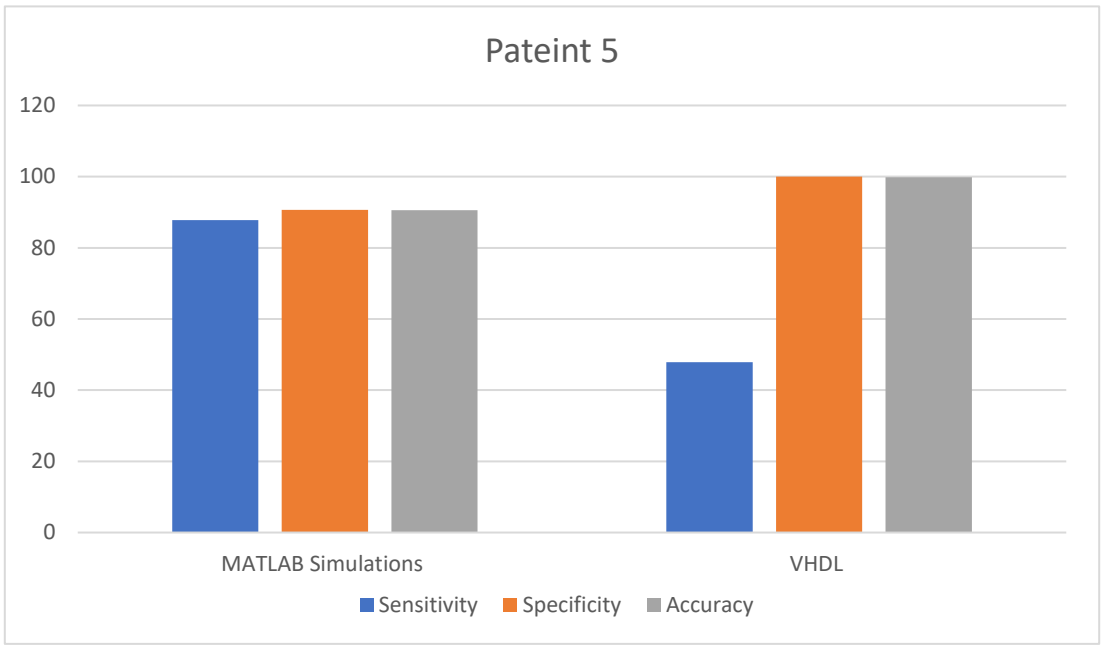


Figure 11.4 Patient 3 VHDL Hyperplane

11.2.3 Case: Patient (5)



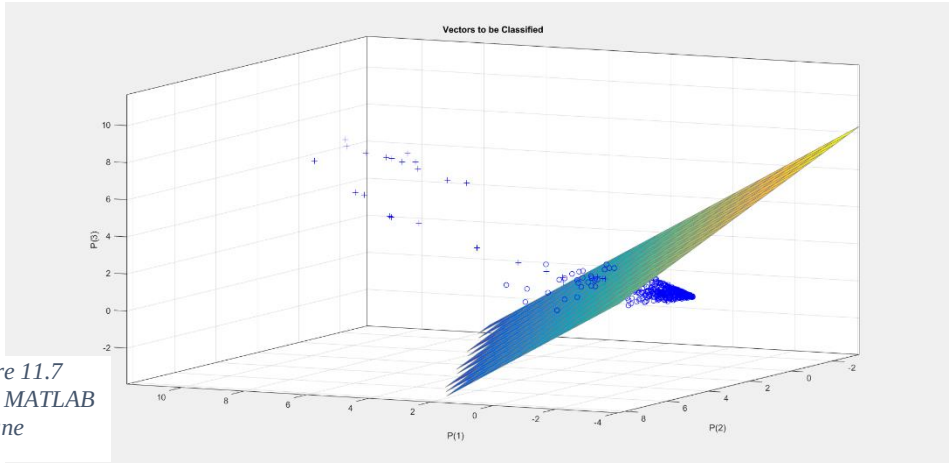


Figure 11.7 Patient 5 MATLAB Hyperplane

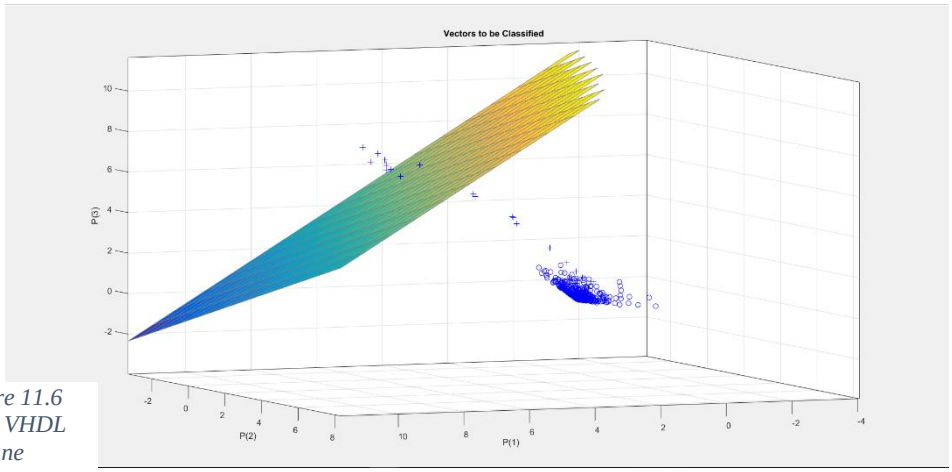


Figure 11.6 Patient 5 VHDL Hyperplane

11.3 Discussion and Analysis of the results

From observing the previous figures, patient 1 and patient 3 RTL simulations gave promising results that were close to the results obtained from MATLAB simulations.

Since feature space of both patient 1 and 3 are less overlapping. And although, some approximations were carried out in simulating RTL the algorithm could find the proper support vectors to draw the hyperplane in between. On the other hand, seizure and non-seizure points of patient 5 are more overlapping in the feature space which caused more complexions in finding the proper margin in the presence of the approximations. Hence, gave lower results in comparison with MATLAB simulation. This is obvious from the previous hyperplane figure (11.7) which tends to be favoring non-seizure points. This shift to the upside towards distant seizure points introduced severe error which reflected on the performance parameters.

11.4 Resources usage

The project design is divided into two main modules, the Training Module and Classification Module. The training module function is to provide the “W” and “b” (i.e. the training parameters) of the SVM. Whereas in Classification module it uses these parameters to conduct the online classification of the extracted points.

Table 11-1 Classification Module Resources Utilization

Revision Name	Main
Top-level Entity	Main
Family	Cyclone V
Device	5CSEBA6U23I7
Logic Utilization (in ALMs)	256/41,910 (<1%)
Total Registers	169
Total Pins	12/314 (4%)
Total Block Memory Bits	100,800/5,662,720 (2%)
Total DSP Blocks	15/112 (13%)

Table 11-2 Gilbert Training Module Resources Utilization

Revision Name	Main
Top-level Entity	Integration_final
Family	Cyclone V
Device	5CSEBA6U23I7
Logic Utilization (in ALMs)	9,186/41,910 (22%)
Total Registers	1014
Total Pins	18/314 (6%)
Total Block Memory Bits	250,320/5,662,720 (4%)
Total DSP Blocks	12/112 (12%)

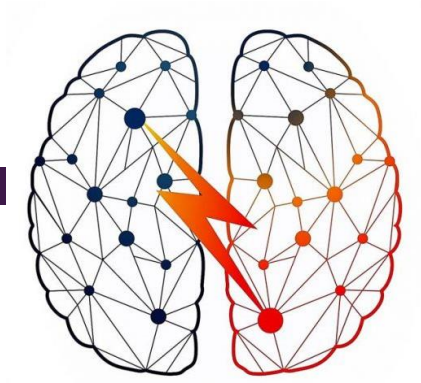
From the previous tables, table 11-1 illustrates the resources usage of the classification entity. As seen the logic utilization is less than 1% of the FPGA ALMs. This low utilization of the ALMs is due to the fact that the Classification module employed DSP blocks instead of logic to calculate the classification equation. On the other hand, the ALMs utilization in the Gilbert Integration module is much more (22%). This is due to using a -designed from scratch- entities to perform most of algorithm calculations.

11.5 Power Analysis Results

The power analysis both design entities are conducted after successful and high confidence estimation using sufficient toggle rates provided by testbenches.

Table 11-3 Power Dissipation Results

Power Type	Training	Classification
Total Thermal Power Dissipated	526.52 mW	477.75 mW
Core Dynamic Thermal Power Dissipated	103.03 mW	53.10 mW
Core Static Thermal Power Dissipation	413.45 mW	412.78 mw



12

Conclusion

In the thesis, the problem of seizure detection using support vector machine (SVM) was addressed to classify between seizures and non- seizures classes. The SVM goal is to be able to draw an accurate hyperplane between the two classes to categorize the signals of a patient's brain to be correctly detected.

One of the main contributions of our work is to express this task with a new algorithm that could give a better performance than previous implemented algorithms. This algorithm is called "Gilbert's Algorithm".

A discussion of Sequential Minimum Optimization (SMO) method as SVM classification method in training the data has been provided with its results. This was an important stage in the understanding of the detection problem.

The main focus of the thesis was proving that Gilbert's Algorithm - the geometrical approach used long time ago in few applications - is able to be a good classifier for seizure detection problems. The analysis leads to the following conclusions:

- At first, the obtained results were not satisfying; the sensitivity did not exceed 69.5652 %.
- Various trials were carried on in order to improve this percentage along the other measurements such as changing the kernel and increasing the training hours.
- Finally, improving the sensitivity to 90% was managed with high specificity and accuracy by changing a parameter "$\tilde{C}$" to 0.0016 and the kernel to Radial Basis Function (RBF).

These findings provide a potential mechanism for employing Gilbert's Algorithm on an embedded chip that will be implanted in the subject brain. Hence, triggering electric stimuli that will be able to retrieve the brain back to its normal state.

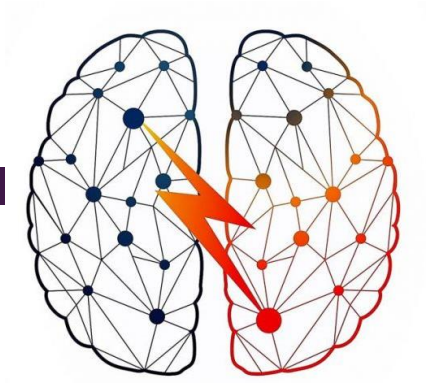
Before manufacturing the chip, a proof of concept is preferred to be done using FPGA due to its beneficial re-programmability and re-configurability. Burning Gilbert's training algorithm and classifier on FPGA is done successfully. This is very much the key component in future attempts to build the chip using ASIC technology.

12.1.1 Future Work

Many optimizations have been left for the future work to obtain the optimal minimum power by analyzing it on Quartus Prime tool for the FPGA used. The target of the chip is to consume a low power that the battery can survive long periods. This is could be done by modifying and optimizing each module and different parts of the VHDL code till reaching an acceptable power without loss of accuracy and performance. This is desirable in the seizure detection problem for the following stage: the ASIC stage.

Future work concerns deeper analysis in the ASIC field in order to build the chip that would contribute in seizure elimination.

A future vision in the design problem is to store all the candidate training algorithms in the chip and use partial dynamic reconfiguration to let the chip use what is currently optimum. (i.e. Low power required? -> reconfigure to Algorithm-1, Fast Training required? -> reconfigure to Algorithm-2).



13

Achievements

This chapter is intended to highlight our achievements throughout a period of nearly nine months of work to do the project at the best possible way. Furthermore, the awards we received are documented and shown below.

13.1 Achievements

These are achievements related to certain benchmarks or highlights we successfully accomplished throughout the project course. They are related to direct tasks fulfillment and they represent a chronological progress since the beginning of our work. Our achievements are shown as follows:

- Understanding problem definition and Medical background beyond epilepsy
- Studying Machine learning concepts we needed
- SVM using SMO algorithm Research and Simulation
- SVM using Gilbert's algorithm Research
- Matlab code simulation for Gilbert's algorithm
- Modification of the Matlab code to reach an acceptable performance
- Design architecture of hardware block diagram
- Mapping the Matlab code to VHDL and Verilog hardware descriptive language
- Implementing the VHDL and Verilog files in the RTL phase
- FPGA mapping and burning

- Detecting seizure and non-seizure on FPGA having our Matlab code results
- Getting power analysis to our design on FPGA

13.2 Awards

This section is concerned with different awards that we were honored to receive throughout our graduation project period. Our awards are listed as follows:

- Qualification as a regional semi-finalist in the Innovate FPGA contest sponsored by Intel in 31st January 2018.
- Qualification as a regional finalist in the Innovate FPGA contest sponsored by Intel in 31st May 2018.
- Receiving a fund by ITAC in 25th March 2018.
- Winning the first place in EECE Department Day for preparing the best poster for competing Graduation Project teams in 14th April 2018.

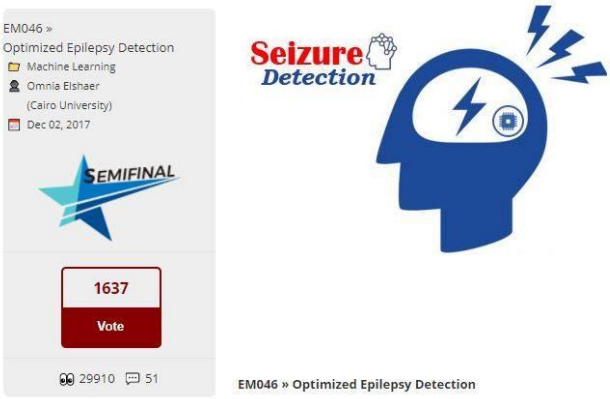


Figure 13.1 Regional semi-finalist in the Innovate FPGA contest sponsored by Intel

Figure 13.2 Regional finalist in the Innovate FPGA contest sponsored by Intel

Recommendations:

We have dedicated this to anyone interested in doing similar projects to our one to consider the following points. Although it may seem obvious, it is very important to indicate the importance of having a substantial background of programming skills using Matlab. Moreover, it is very advisable to learn basic machine learning concepts as they are fundamental in the project progress. Furthermore, it will be a great advantage to practice reading some scientific papers in advance of project beginning to enhance the analytical skills of understanding new information. Another very beneficial advice is to extremely improve RTL skills whether using VHDL or Verilog. Last but not least, it is recommended to get some knowledge on FPGA, its functionality and operation because it is the step in which the design is mapped to a real physical chip.

Appendix

VHDL/Verilog Codes:

13.3 alphabetaMem

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


Entity alphabetaMem is
Generic (Addr_Width    : integer := 4;
     Data_Width    : integer := 16;
        n      : integer := 16);
port(
    clk : in std_logic;
    reset: in std_logic;
    WkSelect: in std_logic_vector(1 downto 0);
    lamda: in std_logic_vector(n-1 downto 0);
    CV_Point_address: in std_logic_vector(Addr_Width-1 downto 0);
CVcounter: in unsigned(16-1 downto 0);
    Datacounter: in unsigned(16-1 downto 0);
    weight_one: in std_logic_vector(Data_Width-1 downto 0);
average_enable: in std_logic;
    copyTobackUp_enable: in std_logic;
    read_enable_norm_avg: in std_logic;
    read_enable_BackUp: in std_logic;
    alpha_beta_wake_up: in std_logic;
    First_Run: in std_logic;    Out_alphabeta_average: out
std_logic_vector(Data_Width-1 downto 0);
    OutBackUp: out std_logic_vector(Data_Width-1 downto 0)

);
end entity alphabetaMem;

architecture arch_alphabeta_Mem of alphabetaMem is

--------------------COMPONENTS INST:-------------------
component data_Memory is
    Generic (
    DATA_Mem_WIDTH: integer := 16;
    MeM_DEPTH: integer := 16
);

port (
    clk         : in std_logic;
    write_enable    : in std_logic;
    read_enable     : in std_logic;
```

```
        address     : in std_logic_vector(MeM_DEPTH-1 downto 0);
        datain      : in std_logic_vector(DATA_Mem_WIDTH-1  downto 0);

        dataout     : out std_logic_vector(DATA_Mem_WIDTH-1 downto 0)
    );
    end component data_Memory;

    Component BoothTop is
    port(
        M: in std_logic_vector(15 downto 0);
        Q: in std_logic_vector(15 downto 0);
        Z: out std_logic_vector(31 downto 0)
    );
    end Component BoothTop;

    component divider is
    generic(input_width: integer:=30);
    port(

        Q: in std_logic_vector(input_width-1 downto 0);
        M: in std_logic_vector(input_width-1 downto 0);
        Quo: out std_logic_vector(input_width-1 downto 0);
        Remi: out std_logic_vector(input_width-1 downto 0)
    );
    end component;
    ------------------------------------------------------------------------
    TYPE State_type IS (OldPoint, NewPoint, SegmentPoint,Accumulation,
CalcAvg,idle,ReadAlphaNew,CopyToBckUp,ReadBckUp );  -- Define the states
    SIGNAL State : State_Type;

    TYPE State_type_acc IS (read_acc,write_acc);  -- Define the states
    SIGNAL State_acc : State_Type_acc;

    TYPE State_type_avg IS (read_avg,write_avg);  -- Define the states
    SIGNAL State_avg : State_Type_avg;

    TYPE Alpha_Beta_Switch is (state_on,state_off);
    SIGNAL Switch_ab : Alpha_Beta_Switch;

    ------------
    SIGNAL reg_trigger: std_logic;
    SIGNAL write_enable_alphabeta: std_logic;
    SIGNAL read_enable_alphabeta: std_logic;
    SIGNAL sigAddress: std_logic_vector(Addr_Width-1 downto 0);
    SIGNAL accCounter: unsigned(16-1 downto 0);
    SIGNAL address_alphabeta: std_logic_vector(Addr_Width-1 downto 0);
    SIGNAL sigCVcounter: unsigned(16-1 downto 0);
    SIGNAL R_W: std_logic; --Flag bit, when 0 => Read , when 1 => Write
    SIGNAL siglamda: std_logic_vector(n-1 downto 0);
    SIGNAL sigDataout32: std_logic_vector((2*Data_Width)-1 downto 0);
    ----------------------------FSM-----------------------------------
    signal Data_in_FSMtoM: std_logic_vector(Data_Width-1 downto 0);
    signal Data_out_MtoFSM: std_logic_vector(Data_Width-1 downto 0);
    ------------------------------acc--avg-------------------------------
    SIGNAL Data_in_Acc_Avg: std_logic_vector(Data_Width-1 downto 0);
    SIGNAL Data_in_Acc_Avg_sig: std_logic_vector(Data_Width-1 downto 0);
    SIGNAL Data_out_Acc_Avg: std_logic_vector(Data_Width-1 downto 0);
```

```
    signal write_enable_Acc_Avg: std_logic;--write enable for acc_avg mem
    signal read_enable_Acc_Avg: std_logic;--write enable for acc_avg mem
    SIGNAL first_accumulation: std_logic; -- =1-> first time to accum
    SIGNAL first_copyToBackup: std_logic;
    signal address_Acc_Avg:std_logic_vector(Addr_Width-1 downto 0);
    SIGNAL sigRead_enable_avg: std_logic;
    SIGNAL address_Acc_Avg_sequen: unsigned(Addr_Width-1 downto 0);
    SIGNAL sigDataout_div: std_logic_vector(Data_Width-1 downto 0);
    SIGNAL Acc_flag: std_logic;
    SIGNAL sum_acc_MtoFSM: std_logic_vector(Data_Width-1 downto 0);
    ----------------------Backup-----------------------------------------
    SIGNAL write_enable_BackUp: std_logic;
    SIGNAL address_backup:std_logic_vector(Addr_Width-1 downto 0);
    SIGNAL address_backup_sequen: unsigned(Addr_Width-1 downto 0);
    SIGNAL Data_in_BckUp:std_logic_vector(Data_Width-1 downto 0);
    signal first_run_sel: std_logic;   --- to mux ;
    signal first_run_in: std_logic; -----  off the first sig
    ----------------------------------------------------------------------
    Begin
    first_run_in<= first_run when first_run_sel='1' else
            '0';
    State<= ReadAlphaNew when read_enable_norm_avg='1' else
        CopyToBckUp when copyTobackUp_enable='1' else
        ReadBckUp when read_enable_BackUp='1' else
        CalcAvg when average_enable='1' and Acc_flag='0' else
        Accumulation when Acc_flag='1' else
        SegmentPoint when WkSelect= "00" and average_enable='0' and
Acc_flag='0' else
       OldPoint when WKSelect="10" and average_enable='0' and Acc_flag='0'
and First_Run_in='0' else
       NewPoint when ((WKSelect="11" and average_enable='0' and
Acc_flag='0') or First_Run_in='1')  else
       idle;
    Data_in_Acc_Avg_sig<=
std_logic_vector(resize(signed(Data_out_MtoFSM(Data_Width-1 downto 5)),16)
) when first_accumulation='1' else
             Data_in_Acc_Avg;

    siglamda<= std_logic_vector("0100000000" - (signed (lamda)));
    multipI: BoothTop port map(siglamda,Data_out_MtoFSM,sigDataout32);
    ----------------------DIVIDER ---------------------
    div: divider generic map(input_width=>16) port
map(Data_out_Acc_Avg,std_logic_vector(CVcounter),sigDataout_div);

    Memory_Alphabeta: data_Memory generic map (DATA_Mem_WIDTH =>
Data_Width,MeM_DEPTH=>Addr_Width) port map(clk,write_enable_alphabeta,
read_enable_alphabeta, address_alphabeta,Data_in_FSMtoM,Data_out_MtoFSM);
    Memory_Acc_Average: data_Memory generic map (DATA_Mem_WIDTH =>
Data_Width,MeM_DEPTH=>Addr_Width) port map(clk,write_enable_Acc_Avg,
sigRead_enable_avg, address_Acc_Avg,Data_in_Acc_Avg_sig,Data_out_Acc_Avg);
    Memory_BackUp: data_Memory generic map (DATA_Mem_WIDTH =>
Data_Width,MeM_DEPTH=>Addr_Width) port map(clk,write_enable_BackUp,
read_enable_BackUp, address_backup,Data_out_Acc_Avg,OutBackUp);
    Out_alphabeta_average<=Data_out_Acc_Avg;
    reg_trigger<= average_enable or copyTobackUp_enable or
read_enable_norm_avg or
            read_enable_BackUp or alpha_beta_wake_up;
```

```
main_Proc:  process(clk,sigAddress,sigDataout32,reset,reg_trigger) is
  begin
    if(reset='1') then
        sigAddress<=(others=>'0');
        sigCVcounter<=(others=>'0');
        address_alphabeta<=(others=>'0');
        state_acc<=read_acc;
        state_avg<=read_avg;
        accCounter<=(others=>'0');
        first_accumulation<='1';
        R_W<='0';
        address_Acc_Avg_sequen<=(others=>'0');
        address_backup_sequen<=(others=>'0');
        Acc_flag<='0';
        address_backUp<=(others=>'0');
        Switch_ab<=state_off;
        first_run_sel<='1';

    end if;

    if rising_edge(clk) THEN
CASE Switch_ab IS
    when state_off=>
      if(reg_trigger='1') then
            Switch_ab<=state_on;
        else
            Switch_ab<=state_off;
            accCounter<=(others=>'0');
            address_backup_sequen<=(others=>'0');
            address_alphabeta<=(others=>'0');
        end if;

    when state_on=>
        CASE State IS
            when OldPoint=>
                write_enable_alphabeta<='0';
                read_enable_alphabeta<='1';
                sigRead_enable_avg<=read_enable_Acc_Avg
                Acc_flag<='1';
                state_avg<=read_avg;
            when NewPoint=>
                if accCounter < Datacounter then
                 sigAddress<= std_logic_vector(unsigned(sigAddress)+1);
                    Data_in_FSMtoM<=(others=>'0');
                    write_enable_alphabeta<='1';
                    accCounter<= accCounter+1;
                    address_alphabeta<=sigaddress;
                elsif accCounter= Datacounter then
                    Data_in_FSMtoM<= weight_one;
                    accCounter<= accCounter+1;
                    address_alphabeta<= CV_Point_address;
                elsif accCounter=(Datacounter)+1 then
                    write_enable_alphabeta<='0';
                    accCounter<=(others=>'0');
                    address_alphabeta<=(others=>'0');
                    sigAddress<=(others=>'0');
```

```
                        sigRead_enable_avg<=read_enable_Acc_Avg;
                        if(First_Run_in='1') then
                            Acc_flag<='0';
                            first_run_sel<='0';
                            switch_ab<=state_off;
                        else
                            Acc_flag<='1';
                            state_avg<=read_avg;
                        end if;
                    end if;
                when SegmentPoint=>
                    if accCounter < Datacounter then
                        if R_W = '0' then --if read operation:
                            write_enable_alphabeta<='0';
                            read_enable_alphabeta<='1';=
                            address_alphabeta<=sigAddress;
                            R_W <= '1';
                        elsif R_W ='1' then -- if write operation
                            write_enable_alphabeta<='1';
                            read_enable_alphabeta<='0';
                            Data_in_FSMtoM<=sigDataout32(23 downto 8);
                     sigAddress<= std_logic_vector(unsigned(sigAddress)+1);
                            accCounter<=accCounter+1;
                            address_alphabeta<=sigAddress;
                            R_W <= '0'; -- Next operation is Read
                        end if;
                    elsif accCounter=Datacounter then
                        write_enable_alphabeta<='1';
                        read_enable_alphabeta<='1';
                        accCounter<=accCounter+1;
                        address_alphabeta<=CV_Point_address;
                    elsif accCounter = (Datacounter+1) then
                        if (weight_one(weight_one'HIGH)='0') then
                            Data_in_FSMtoM<= std_logic_vector(unsigned
(Data_out_MtoFSM) + unsigned (lamda(8 downto 0)&"0000")); -- (4,12)+(8,8)
                        else
                            Data_in_FSMtoM<= std_logic_vector(unsigned
(Data_out_MtoFSM) - unsigned (lamda(8 downto 0)&"0000"));
                        end if;
                       write_enable_alphabeta<='1';
                        accCounter<=(others=>'0');
                        read_enable_alphabeta<='1';
                        sigRead_enable_avg<='1';
                        sigAddress<=(others=>'0');
                        Acc_flag<='1';
                        state_acc<=read_acc;
                    end if;
                WHEN CalcAvg=>
                    case state_avg IS
                        WHEN read_avg=>
                            if accCounter < Datacounter then
                            write_enable_Acc_Avg<='0';
                            sigRead_enable_avg<='1';
                            address_Acc_Avg<=sigaddress;
                            state_avg<= write_avg;
                            else
                            state_avg<= read_avg;
```

```
                        sigRead_enable_avg<='0';
                        write_enable_Acc_Avg<='0';
                        Switch_ab<=state_off;
                        end if;
                    WHEN write_avg=>
                        if accCounter <Datacounter then
                        write_enable_Acc_Avg<='1';
                        sigRead_enable_avg<='0';
                        address_Acc_Avg<=sigaddress;
                        Data_in_Acc_Avg<=(
sigDataout_div(sigDataout_div'HIGH)&sigDataout_div(9 downto 0)&"00000");
              sigAddress<= std_logic_vector(unsigned (sigAddress)+1);
                        accCounter<=accCounter+1;
                        state_avg<= read_avg;
                        else
                        state_avg<= read_avg;
                        sigRead_enable_avg<='0';
                        write_enable_Acc_Avg<='0';
                        Switch_ab<=state_off;
                        end if;
                end case;
             WHEN Accumulation  =>
                case state_acc IS
                    WHEN read_acc=>
                        if accCounter < Datacounter then
                        write_enable_alphabeta<='0';
                        write_enable_Acc_Avg<='0';
                        address_alphabeta<=sigaddress;
                        address_Acc_Avg<=sigaddress;
                        read_enable_alphabeta<='1';
                        sigRead_enable_avg<='1';
                        state_acc<= write_acc;
                        else
                        write_enable_Acc_Avg<='0';
                        read_enable_alphabeta<='0';
                        sigRead_enable_avg<='0';
                        accCounter<=(others=>'0');
                        sigAddress<=(others=>'0');
                        first_accumulation<='0';
                        Acc_flag<='0';
                        Switch_ab<=state_off;
                        end if;

                    WHEN write_acc=>
                        if accCounter < Datacounter then
                        Data_in_Acc_Avg<=std_logic_vector(
signed(Data_out_Acc_Avg) + resize(signed(Data_out_MtoFSM(Data_Width-1
downto 5)),16)  );
                        write_enable_Acc_Avg<='1';
 sigaddress<= std_logic_vector(unsigned (sigAddress)+1);
                        accCounter<=accCounter+1;
                        state_acc<= read_acc;
                        else -- accu finished
                        state_acc<= read_acc;
                        write_enable_Acc_Avg<='0';
                        read_enable_alphabeta<='0';
                        sigRead_enable_avg<='0';
```

```
                                accCounter<=(others=>'0');
                                sigAddress<=(others=>'0');
                                first_accumulation<='0';
                                Acc_flag<='0';
                                Switch_ab<=state_off;
                                end if;
                        end case;
                    WHEN ReadAlphaNew =>
                        if(read_enable_norm_avg='1' and accCounter<Datacounter)
then
                            sigRead_enable_avg<='1';
                            address_backup_sequen<= address_backup_sequen+1;
address_Acc_Avg<=std_logic_vector(address_backup_sequen);
                            accCounter<= accCounter+1;
                        else
                            address_Acc_Avg_sequen<=(others=>'0');
                            Switch_ab<=state_off;

                        end if;
                    WHEN CopyToBckUp=>
                        if(copyTobackUp_enable='1' and accCounter<Datacounter)
then
                            sigRead_enable_avg<='1';
                            write_enable_BackUp<='1';
                            address_backup_sequen<= address_backup_sequen+1;
   address_Acc_Avg<=std_logic_vector(address_backup_sequen);
address_backUp<=std_logic_vector(address_backup_sequen);
                           accCounter<=accCounter+1;
                        else
                            address_backup_sequen<=(others=>'0');
                            Switch_ab<=state_off;
                        end if;
                            WHEN ReadBckUp=>
                        if(read_enable_BackUp='1' and accCounter<Datacounter)
then

                            address_backup_sequen<= address_backup_sequen+1;

address_backUp<=std_logic_vector(address_backup_sequen);
                            accCounter<=accCounter+1;
                        else
                            address_backup_sequen<=(others=>'0');
                            Switch_ab<=state_off;

                        end if;
                    WHEN idle=>
                sigAddress<=(others=>'0');
                sigCVcounter<=(others=>'0');
                address_alphabeta<=(others=>'0');
                state_acc<=read_acc;
                state_avg<=read_avg;
                accCounter<=(others=>'0');
                first_accumulation<='1';
                R_W<='0';
                address_Acc_Avg_sequen<=(others=>'0');
                address_backup_sequen<=(others=>'0');
                Acc_flag<='0';
```

```
        Switch_ab<=state_off;
            WHEN others =>
        end CASE;
end Case;
    end if;

end process;
end architecture arch_alphabeta_Mem;
```

13.4 CacheAcc

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity cacheAcc_1 is
Generic (CacheAcc_Width : integer := 27; --27 bit(15,12)
     Cache_Width : integer := 23;--23 bit (11,12)
    address_width:integer :=16);
port (
     clk : in std_logic;
     write_enable_cachAcc1 : in std_logic;
         read_enable_cachAcc1 : in std_logic;
         datain_cachAcc1 : in std_logic_vector(Cache_Width-1 downto 0);

     DataOut_cachAcc1:out std_logic_vector(CacheAcc_Width-1 downto 0);

         reg_reset_chachAcc1: in std_logic;
         reg_enable_cahcAcc1: in std_logic;
         MuxDataIN_SeclAcc1  : in std_logic
);
end entity cacheAcc_1;

architecture cacheAcc_arch_1 of cacheAcc_1 is
component data_Memory is
    Generic (
    DATA_Mem_WIDTH: integer := 16;
    MeM_DEPTH: integer := 16
    );
port ( clk : in std_logic;
write_enable : in std_logic;
read_enable : in std_logic;
address: in std_logic_vector(MeM_DEPTH-1 downto 0);
datain : in std_logic_vector(DATA_Mem_WIDTH - 1  downto 0);
dataout : out std_logic_vector(DATA_Mem_WIDTH - 1 downto 0)
);
end component;
-------------------------------------------------------------------
component N_bitfulladder is
Generic (n : integer := 16);
port
(a,b:in std_logic_vector(n-1 downto 0);
f:out std_logic_vector(n-1 downto 0);
cout:out std_logic
```

```
   );
   end component;
   ------------------------------------------------------------------------
   component reg is
   generic(n:integer);
   port(
       clk,rst,wenable:in std_logic;
       d:in std_logic_vector(n-1 downto 0);
       q:out std_logic_vector(n-1 downto 0)
   );
   end component;
   ------------------------------------------------------------------------
-
   component mux2x1 is
   Generic (n:integer);
   port(
   d1:in std_logic_vector(n-1 downto 0);
   d2:in std_logic_vector(n-1 downto 0);
   s:in std_logic;
   q:out std_logic_vector(n-1 downto 0)
   );
   end component;
   ------------------------------------------------------------------------
-
   signal Sig_datain_cachAcc1:std_logic_vector(CacheAcc_Width-1 downto
0);--data coming from cache
   signal addressIn_cacheAcc1 :std_logic_vector(address_width-1 downto
0);--address in direct to memory
   signal addreIncreas_cahceAcc1 :std_logic_vector(address_width-1 downto
0); --address after increment
   signal dataAcc_cachAcc1 :std_logic_vector( CacheAcc_Width-1 downto 0);
   signal DataOutSignal_cachAcc1 :std_logic_vector( CacheAcc_Width-1
downto 0);
   signal cout_10: std_logic;

   signal dataAcc_MuxOut_cachAcc1 :std_logic_vector( CacheAcc_Width-1
downto 0); --signal after Mux before data in coming data or accu. data
   begin

   Sig_datain_cachAcc1<=datain_cachAcc1(datain_cachAcc1'High)&datain_cachA
cc1(datain_cachAcc1'High)&datain_cachAcc1(datain_cachAcc1'High)&datain_cac
hAcc1(datain_cachAcc1'High)&datain_cachAcc1;
   dataAcc_cachAcc1<=std_logic_vector(signed
(Sig_datain_cachAcc1)+signed(DataOutSignal_cachAcc1));--27 bit(15,12)

     adderIcreasing:N_bitfulladder generic map(n => address_width) port
map(addressIn_cacheAcc1,"0000000000000001", addreIncreas_cahceAcc1,
cout_10);
     regggg     : reg generic map(n => address_width) port
map(clk,reg_reset_chachAcc1,reg_enable_cahcAcc1,addreIncreas_cahceAcc1,add
ressIn_cacheAcc1);
     cacheAcc_MeMo : data_Memory generic map(DATA_Mem_WIDTH=>
CacheAcc_Width,MeM_DEPTH=>address_width)port
map(clk,write_enable_cachAcc1,
read_enable_cachAcc1,addressIn_cacheAcc1,dataAcc_MuxOut_cachAcc1,DataOutSi
gnal_cachAcc1);
```

```
    MuxDataIn : mux2x1 generic map(n => CacheAcc_Width) port
map(Sig_datain_cachAcc1,dataAcc_cachAcc1,MuxDataIN_SeclAcc1,dataAcc_MuxOut
_cachAcc1);
  DataOut_cachAcc1<=DataOutSignal_cachAcc1;
  end architecture cacheAcc_arch_1;
```

13.5 cacheAvg_1

```
  Library ieee;
  use ieee.std_logic_1164.all;
  USE IEEE.numeric_std.all;

  Entity cacheAvg_1 is
  Generic (n : integer := 16;
      CacheAcc_Width : integer := 27;
      CacheAvg_Width : integer := 16;
      address_width:integer :=16);
  port (
       clk : in std_logic;
       write_enable_cachAvg1 : in std_logic;
           read_enable_cachAvg1  : in std_logic;
           datain_cachAvg1       : in std_logic_vector(CacheAcc_Width-1
downto 0);
       DataOut_cachAvg1      :out std_logic_vector(CacheAvg_Width-1
downto 0);
           reg_reset_chachAvg1   : in std_logic;
           reg_enable_cahcAvg1   : in std_logic;
       Avg_Counter           : in unsigned(n-1 downto 0);

    addreMuxOut_cachAvg_b:out std_logic_vector(address_width-1 downto 0);
  end entity cacheAvg_1;

  architecture cacheAvg_1_arch of cacheAvg_1 is
  component data_Memory is
      Generic (
      DATA_Mem_WIDTH: integer := 16;
      MeM_DEPTH: integer := 16
      );
  port ( clk : in std_logic;
  write_enable : in std_logic;
  read_enable : in std_logic;
  address: in std_logic_vector(MeM_DEPTH-1 downto 0);
  datain : in std_logic_vector(DATA_Mem_WIDTH - 1  downto 0);
  dataout : out std_logic_vector(DATA_Mem_WIDTH - 1 downto 0)
  );
  end component;
  ------------------------------------------------------------------
  component N_bitfulladder is
  Generic (n : integer := 16);
  port
  (a,b:in std_logic_vector(n-1 downto 0);
  f:out std_logic_vector(n-1 downto 0);
```

```
    cout:out std_logic
    );
    end component;
    ------------------------------------------------------------------
    component reg is
    generic(n:integer);
    port(
        clk,rst,wenable:in std_logic;
        d:in std_logic_vector(n-1 downto 0);
        q:out std_logic_vector(n-1 downto 0)
    );
    end component;
    ------------------------------------------------------------------
    component mux2x1 is
    Generic (n:integer);
    port(
    d1:in std_logic_vector(n-1 downto 0);
    d2:in std_logic_vector(n-1 downto 0);
    s:in std_logic;
    q:out std_logic_vector(n-1 downto 0)
    );
    end component;
    ------------------------------------------------------------------
    Component divider is
    Generic (input_width : integer := 16);
    port(
        Q: in std_logic_vector(input_width-1 downto 0);
        M: in std_logic_vector(input_width-1 downto 0);
        Quo: out std_logic_vector(input_width-1 downto 0);
        Remi: out std_logic_vector(input_width-1 downto 0)
    );
    end Component divider;
    ------------------------------------------------------------------

    signal addressIn_cacheAvg1 :std_logic_vector(address_width-1 downto 0);
    signal addreIncreas_cahceAvg1 :std_logic_vector(address_width-1 downto
0);
    signal addreMuxOut_cachAvg1 :std_logic_vector(address_width-1 downto
0);
    signal Divided_Avg_value:std_logic_vector(CacheAcc_Width-1 downto 0);
    signal overF :std_logic;
    signal cout: std_logic;
    signal Remn: std_logic_vector(CacheAcc_Width-1 downto 0);

    signal SigAvg_Counter :std_logic_vector(CacheAcc_Width-1 downto 0);

    begin
    SigAvg_Counter<=std_logic_vector("00000000000"&Avg_Counter);

    LABLE_Division: divider generic map(input_width=>CacheAcc_Width) port
map(datain_cachAvg1,SigAvg_Counter,Divided_Avg_value,Remn);

     LABLE_AddreInc    :N_bitfulladder generic map(n => address_width) port
map(addressIn_cacheAvg1,"0000000000000001", addreIncreas_cahceAvg1, cout);

     LABLE_Avg         : data_Memory generic
map(DATA_Mem_WIDTH=>CacheAvg_Width,MeM_DEPTH=>address_width) port
```

```
map(clk,write_enable_cachAvg1,
read_enable_cachAvg1,addressIn_cacheAvg1,Divided_Avg_value(20 downto
5),DataOut_cachAvg1);
    LABLE_Reg_Avg     : reg generic map(n => address_width) port
map(clk,reg_reset_chachAvg1,reg_enable_cahcAvg1,addreIncreas_cahceAvg1,add
ressIn_cacheAvg1);

   addreMuxOut_cachAvg_b<=addreMuxOut_cachAvg1;
   end architecture cacheAvg_1_arch;
```

13.6Kernel_block

```
   LIBRARY IEEE;
   USE IEEE.std_logic_1164.all;
   USE IEEE.numeric_std.all;


   ENTITY Kernel_block IS
   Generic (n : integer := 17);
   PORT (
   clk,reset : in std_logic;
   ------------inputs-------------------------------------------------
   Feature1_Class1_Data  : in std_logic_vector(n-1 downto 0);   --fx1
   Feature2_Class1_Data  : in std_logic_vector(n-1 downto 0);   --fx2
   Feature3_Class1_Data  : in std_logic_vector(n-1 downto 0);   --fx3

   Feature1_Class2_Data  : in std_logic_vector(n-1 downto 0);   --fy1
   Feature2_Class2_Data  : in std_logic_vector(n-1 downto 0);   --fy2
   Feature3_Class2_Data  : in std_logic_vector(n-1 downto 0);   --fy3

   ----------------------Kernel Relative Point Features Reg Signals-------
---------------
   reg_reset_Kernel_RelativePoint_F    : in std_logic;
   reg_enable_Kernel_RelativePoint_F   : in std_logic;
   --------------------Kernel class Features Reg Signals------------------
-------
   Kernal_classF_Mux_Selc: in std_logic;
   reg_reset_Kernel_classF    : in std_logic;
   reg_enable_Kernel_classF   : in std_logic;


   ------------------------------------------------------------------------
   Output_final: out std_logic_vector(15 downto 0); -- (4,12)
   OutCountSig: out std_logic;
   Count_TostartRead: out std_logic_vector (2 downto 0)
    );

   END ENTITY Kernel_block;

   ARCHITECTURE Arch_Kernel_block OF Kernel_block IS

   ------------------------------------------------------------------------
   component B4_Kernel is
   Generic (n : integer := 17);
   port (
```

```
    clk : in std_logic;

    Feature1_Class1_Data  : in std_logic_vector(n-1 downto 0);   --fx1
    Feature2_Class1_Data  : in std_logic_vector(n-1 downto 0);   --fx2
    Feature3_Class1_Data  : in std_logic_vector(n-1 downto 0);   --fx3

    Feature1_Class2_Data  : in std_logic_vector(n-1 downto 0);   --fy1
    Feature2_Class2_Data  : in std_logic_vector(n-1 downto 0);   --fy2
    Feature3_Class2_Data  : in std_logic_vector(n-1 downto 0);   --fy3

    ----------------------Kernel Relative Point Features Reg Signals-----
    reg_reset_Kernel_RelativePoint_F   : in std_logic;
    reg_enable_Kernel_RelativePoint_F  : in std_logic;

    Kernal_RelativePoint_Mux_3Feature_Selc : in std_logic_vector(1 downto
0);
    --------------------Kernel class Features Reg Signals------------------
-------
    Kernal_classF_Mux_Selc: in std_logic;

    reg_reset_Kernel_classF    : in std_logic;
    reg_enable_Kernel_classF   : in std_logic;

    Kernal_class_Mux_3Feature_Selc : in std_logic_vector(1 downto 0);

    Kernal_classFeature_Mux_dataOut: out std_logic_vector(n-1 downto 0);
    Kernal_RelativePoint_Mux_dataOut:out std_logic_vector(n-1 downto 0)

    );
    end component B4_Kernel;
    ------------------------------------------------------------------------
    component reg is
    generic(n:integer:=16);
    port(clk,rst,wenable:in std_logic;
    d:in std_logic_vector(n-1 downto 0);
    q:out std_logic_vector(n-1 downto 0)
    );
    end component;
    ----------------------------------------------------------------
    component exp_lut is
    port(
      i_x            : in  std_logic_vector( 15 downto 0);  --(4,12)
      o_exp          : out std_logic_vector( 15 downto 0)); --(4,12)
    end component exp_lut;
    ----------------------------------------------------------------
    signal count_3 : std_logic_vector(2 downto 0);
    signal Kernal_classFeature_Mux_dataOutSig : std_logic_vector(n-1 downto
0);
    signal Kernal_RelativePoint_Mux_dataOutSig : std_logic_vector(n-1
downto 0);
    signal Kernal_output :  std_logic_vector(15 downto 0);
    signal Kernal_ADDOutput_accuml : std_logic_vector(36 downto 0);
    signal Reg_buffer1,Reg_buffer2,Reg_buffer3 :  std_logic ;
    signal difference : std_logic_vector(17 downto 0);
    ------------------------------------------------------------------------
signal Kernal_class_Mux_3Feature_Selc_sig: std_logic_vector(1 downto 0);
```

```
   signal Kernal_RelativePoint_Mux_3Feature_Selc_sig:  std_logic_vector(1
downto 0);
   signal sqred : std_logic_vector(35 downto 0);
   signal Negative_Squared: signed(36 downto 0);
   signal conc_for_Neg:std_logic_vector( 15 downto 0);
   begin
   input :B4_Kernel generic map
(n=>17)portmap(clk,Feature1_Class1_Data,Feature2_Class1_Data,Feature3_Clas
s1_Data ,Feature1_Class2_Data,Feature2_Class2_Data ,Feature3_Class2_Data
   ,reg_reset_Kernel_RelativePoint_F,reg_enable_Kernel_RelativePoint_F,Ker
nal_RelativePoint_Mux_3Feature_Selc_sig,Kernal_classF_Mux_Selc,reg_reset_K
ernel_classF,reg_enable_Kernel_classF,Kernal_class_Mux_3Feature_Selc_sig,K
ernal_classFeature_Mux_dataOutSig,Kernal_RelativePoint_Mux_dataOutSig);

   difference<=std_logic_vector(signed(Kernal_classFeature_Mux_dataOutSig(
   Kernal_classFeature_Mux_dataOutSig'HIGH)&Kernal_classFeature_Mux_dataOu
   tSig)-
   signed(Kernal_RelativePoint_Mux_dataOutSig(Kernal_RelativePoint_Mux_dat
   aOutSig'HIGH)&Kernal_RelativePoint_Mux_dataOutSig));
   sqred<= std_logic_vector(signed(difference)*signed(difference));
   Negative_Squared <= to_signed(0,37) - signed( Kernal_ADDOutput_accuml
);
   conc_for_Neg<=std_logic_vector(Negative_Squared(Negative_Squared'HIGH)&
Negative_Squared(26 downto 12)) when signed(Negative_Squared)>(-117440512)
          else "1000111111100100";
   Exp: exp_lut port map ( conc_for_Neg ,  Kernal_output );

   process(clk,reset,reg_enable_Kernel_classF)
   begin

       if reset='1' then
       count_3<="000";
       Kernal_ADDOutput_accuml <= (others=>'0');

       elsif (rising_edge(clk))then
           Reg_buffer1<=reg_enable_Kernel_classF;  -- staling part
           Reg_buffer2<= Reg_buffer1;
           if(Reg_buffer2='1')then
                Kernal_ADDOutput_accuml
<=std_logic_vector('0'&signed(sqred)+signed(Kernal_ADDOutput_accuml));
               count_3 <= std_logic_vector(unsigned(count_3)+1);
           end if;
           if count_3="011"then
           count_3<="000";
            Kernal_ADDOutput_accuml <= (others=>'0');
           end if;
       end if;
     end process;
   Kernal_class_Mux_3Feature_Selc_sig<= count_3(1 downto 0);
   Kernal_RelativePoint_Mux_3Feature_Selc_sig<=count_3(1 downto 0) ;
   OutCountSig <='1' when  count_3="011" else
            '0' ;
   Output_final <= Kernal_output when count_3="011";
   Count_TostartRead<=count_3;
   --in CU every time count =3 , el CU teb3at signal reset
   END Arch_Kernel_block;
```

13.7 Norm_avg

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

Entity Norm_avg is
Generic (Data_Width : integer := 16);
port (
clk: in std_logic;
DataOut_cachAvg1 :in std_logic_vector(Data_Width-1 downto 0);
DataOut_cachAvg2 :in std_logic_vector(Data_Width-1 downto 0);
Mux_Select_Avg1_or_Avg2: in std_logic;
i_Old_alphabeta:in std_logic_vector(Data_Width-1 downto 0);
j_Old_alphabeta:in std_logic_vector(Data_Width-1 downto 0);
i_new_alphabeta:in std_logic_vector(Data_Width-1 downto 0);
j_new_alphabeta:in std_logic_vector(Data_Width-1 downto 0);
Mux_Select_alpha: in std_logic_vector(1 downto 0);
Reg_reset_AvgNorm : in std_logic;
Reg_Enable_AvgNorm : in std_logic;
Mux_Select_AvgNorm_Que_in: in std_logic;
Reset_fifo_AvgNorm: in std_logic;
WriteEnable_fifo_AvgNorm: in std_logic;
ReadEnable_fifo_AvgNorm: in std_logic;
FifoEmpty_fifo: out std_logic;
FifoFull_fifo: out std_logic;
Reg_reset_Queu_AvgNorm : in std_logic;
Reg_Enable_Queu_AvgNorm : in std_logic;
Reset_square : in std_logic;
Last_Termination_condition:out std_logic_vector(1 downto 0));
end entity Norm_avg;

architecture Norm_avg_arch of Norm_avg is
component reg is
generic(n:integer);
port(
    clk,rst,wenable:in std_logic;
    d:in std_logic_vector(n-1 downto 0);
    q:out std_logic_vector(n-1 downto 0)
);
end component;
-----------------------------------------------------------------
component mux2x1 is
Generic (n:integer);
port(
d1:in std_logic_vector(n-1 downto 0);
d2:in std_logic_vector(n-1 downto 0);
s:in std_logic;
q:out std_logic_vector(n-1 downto 0)
);
end component;
-----------------------------------------------------------------
```

```
component fifo2 is
    generic (m: integer:=16);
    Port (
            Clk          : in std_logic;
            Reset    : in std_logic;
            WriteEnable  : in std_logic;
            ReadEnable   : in std_logic;
            DataIn       : in std_logic_vector(m-1 downto 0);
            DataOut      : out std_logic_vector(m-1 downto 0);
                FifoEmpty    : out std_logic;
            FifoFull     : out std_logic
        );
END component;
-----------------------------------------------------------------
Component sqrt32 is
port(
clk: in  std_logic;
rdy:out std_logic;
reset:in  std_logic;
x:in std_logic_vector(31 downto 0);
acc:out std_logic_vector(15 downto 0)
);
end Component sqrt32;
-----------------------------------------------------------------
component mux4x1 is
Generic (n:integer);
port(
d1:in std_logic_vector(n-1 downto 0);
d2:in std_logic_vector(n-1 downto 0);
d3:in std_logic_vector(n-1 downto 0);
d4:in std_logic_vector(n-1 downto 0);
s:in std_logic_vector(1 downto 0);
q:out std_logic_vector(n-1 downto 0));
end component;
-----------------------------------------------------------------
Component divider is
Generic (input_width : integer := 16);
port(
    Q: in std_logic_vector(input_width-1 downto 0);
    M: in std_logic_vector(input_width-1 downto 0);
    Quo: out std_logic_vector(input_width-1 downto 0);
    Remi: out std_logic_vector(input_width-1 downto 0)
);
end Component divider;
-----------------------------------------------------------------
Component comparator is
Generic (n : integer := 16);
port(   In1 : IN std_logic_vector(n-1 downto 0);--top
    In2 : IN std_logic_vector(n-1 downto 0); --bot
        Y : out std_logic_vector(1 downto 0)


);
end Component;
-----------------------------------------------------------------
signal SigFromMuxTOQueuIN:std_logic_vector(Data_Width-1 downto 0);
signal SigModi_AvgNorm:std_logic_vector(Data_Width-1 downto 0);
```

```
   signal choosen_dataout_cachAvg:std_logic_vector(Data_Width-1 downto 0);
   signal MultpOut_to_add_AvgNorm:std_logic_vector((2*Data_Width)-1 downto
0);
   signal Add_to_Reg_AvgNorm:std_logic_vector((2*Data_Width)-1 downto 0);
   signal AddReg_to_Multp_AvgNorm:std_logic_vector((2*Data_Width)-1 downto
0);
   signal alphaBeta_muxOut:std_logic_vector(Data_Width-1 downto 0);
   signal DataOut_Queue_avgNorm:std_logic_vector(Data_Width-1 downto 0);
   signal regQueu_DataOut:std_logic_vector(Data_Width-1 downto 0);
   signal avgNorm1_Multp_avgNorm2:std_logic_vector((2*Data_Width)-1 downto
0);
   signal singoutSquare:std_logic;
   signal squareRoot_dataout:std_logic_vector(Data_Width-1 downto 0);
   signal squareReg_dataOut:std_logic_vector(Data_Width-1 downto 0);
   signal remOfDivision:std_logic_vector(Data_Width-1 downto 0);
   signal rdy:std_logic;
   signal Modifi_squareRoot_dataout:std_logic_vector(Data_Width-1 downto
0);
   signal coserror:std_logic_vector(Data_Width-1 downto 0);

   begin
   MultpOut_to_add_AvgNorm <= std_logic_vector(
signed(choosen_dataout_cachAvg) * signed(alphaBeta_muxOut));--32 bit
(13,19)
   Add_to_Reg_AvgNorm      <= std_logic_vector(
signed(MultpOut_to_add_AvgNorm) + signed(AddReg_to_Multp_AvgNorm));--32
bit(13,19)
   avgNorm1_Multp_avgNorm2 <= std_logic_vector( signed(regQueu_DataOut) *
signed(DataOut_Queue_avgNorm)); --16 bit (7,9)+ 16 bit (7,9 --32bit(14,18)

   SigModi_AvgNorm<=AddReg_to_Multp_AvgNorm(25 downto 10) ; --16 bit (7,9)
downscaled from 32 bit (13,19)

   Modifi_squareRoot_dataout<="000000000"&squareRoot_dataout(15 downto
9);-- 16 bit (16,0)

   LABLE_MUX4_alphaBeta: mux4x1 generic map(n => Data_Width) port
map(i_Old_alphabeta,j_Old_alphabeta,i_new_alphabeta,j_new_alphabeta,
                                    Mux_Select_alpha,alphaBeta_muxOut);

   LABLE_MUX2_cacheAvg: mux2x1 generic map(n => Data_Width) port
map(DataOut_cachAvg1,DataOut_cachAvg2,

Mux_Select_Avg1_or_Avg2,choosen_dataout_cachAvg);

   LABLE_AddToReg_AvgNorm:reg generic map(n => 2*Data_Width) port
map(clk,Reg_reset_AvgNorm,Reg_Enable_AvgNorm,Add_to_Reg_AvgNorm,AddReg_to_
Multp_AvgNorm);

   LABLE_Queue:fifo2 generic map(m => Data_Width) port
map(clk,Reset_fifo_AvgNorm,WriteEnable_fifo_AvgNorm,ReadEnable_fifo_AvgNor
m,SigFromMuxTOQueuIN,DataOut_Queue_avgNorm,
                        FifoEmpty_fifo,FifoFull_fifo);

   LABLE_QueReg_ToQueIN: mux2x1 generic map(n =>  Data_Width) port
map(SigModi_AvgNorm,DataOut_Queue_avgNorm,
```

```
    Mux_Select_AvgNorm_Que_in,SigFromMuxTOQueuIN);

      LABLE_Queue_Reg:reg generic map(n => Data_Width) port
    map(clk,Reg_reset_Queu_AvgNorm,Reg_Enable_Queu_AvgNorm,DataOut_Queue_avgNo
    rm,regQueu_DataOut);

      LABLE_queue_to_square:sqrt32 port
    map(clk,rdy,Reset_square,avgNorm1_Multp_avgNorm2,squareRoot_dataout); --
    16 bit(7,9)
LABLE_SquareRoot_Division:divider generic map(input_width => Data_Width) port
map(SigModi_AvgNorm,Modifi_squareRoot_dataout,coserror,remOfDivision);--16
bit(7,9)/ 16 bit(16,0)--16 bit(7,9)
      LABLE_Last_Termination_condition:comparator generic map(n =>
    Data_Width) port
    map(coserror,"0000001000000000",Last_Termination_condition); --01
    cosserror >1  00&10 cosserror<1

      end architecture Norm_avg_arch;
```

13.8 NormCalculate

```
      LIBRARY IEEE;
      USE IEEE.std_logic_1164.all;
      USE IEEE.numeric_std.all;

      ENTITY NormCalculate IS
      Generic (Feature_Width : integer := 17;
           Ctild_Width : integer := 10;
           Norm_Width : integer := 23;
               Cache_Width : integer := 23;
              n     : integer := 16
      );
      PORT (
                           clk   : in std_logic;
                    Reset_fifo_Norm_wkblock   : in std_logic;
                  WriteEnable_fifo_Norm_wkblock   : in std_logic;
                 ReadEnable_fifo_Norm_wkblock   : in std_logic;

                      FifoEmpty_fifo_Norm_wkblock   : out std_logic;
                       FifoFull_fifo_Norm_wkblock   : out std_logic;

                           reg_reset_WkBlock   : in std_logic;
                       reg_enable_WkBlock   : in std_logic;


                         Ctilde        : in std_logic_vector(Ctild_Width-1
    downto 0);
                     Feature1_Class1_Data   : in std_logic_vector(Feature_Width-1
    downto 0);
```

```
                            Feature2_Class1_Data  : in
std_logic_vector(Feature_Width-1 downto 0);
                Feature3_Class1_Data  : in std_logic_vector(Feature_Width-1
downto 0);

                Feature1_Class2_Data  : in std_logic_vector(Feature_Width-1
downto 0);
                            Feature2_Class2_Data  : in
std_logic_vector(Feature_Width-1 downto 0);
                Feature3_Class2_Data  : in std_logic_vector(Feature_Width-1
downto 0);

                Select_Norm_queueIn   : in std_logic;

                    Cache_i_max   : IN std_logic_vector(Cache_Width-1
downto 0);
                                        Cache_j_min   : IN
std_logic_vector(Cache_Width-1 downto 0);
                        WkSelect      : OUT std_logic_vector(1 downto 0);
                                            lamda   : OUT std_logic_vector(n-
1 downto 0);
                    max_min_enable: in std_logic

                         );

  END ENTITY NormCalculate;


  ARCHITECTURE Arch_NormCalculate OF NormCalculate IS

  Component DotProduct IS
  Generic (Feature_Width: integer := 16);
  PORT (
      Input1  : in std_logic_vector(Feature_Width-1 downto 0);
      Input2  : in std_logic_vector(Feature_Width-1 downto 0);
      Input3  : in std_logic_vector(Feature_Width-1 downto 0);
      Input4  : in std_logic_vector(Feature_Width-1 downto 0);
      Input5  : in std_logic_vector(Feature_Width-1 downto 0);
      Input6  : in std_logic_vector(Feature_Width-1 downto 0);
      Result  : out std_logic_vector((2*Feature_Width)+1 downto 0)
  );
  end Component ;
  ------------------------------------------------------------------
  Component mux2x1 is
  Generic (n:integer);
  port(
  d1:in std_logic_vector(n-1 downto 0);
  d2:in std_logic_vector(n-1 downto 0);
  s:in std_logic;
  q:out std_logic_vector(n-1 downto 0)
  );
  end Component;
  ------------------------------------------------------------------
  Component fifo2 is
      generic (m: integer:=16);
      Port (
              Clk          : in std_logic;
```

```
                Reset     : in std_logic;
                WriteEnable   : in std_logic;
                ReadEnable    : in std_logic;
                DataIn        : in std_logic_vector(m-1 downto 0);
                DataOut       : out std_logic_vector(m-1 downto 0);
                    FifoEmpty     : out std_logic;
                FifoFull      : out std_logic
            );
    END Component;
    ------------------------------------------------------------------------
    Component Wk_calculate is
    Generic (
        Feature_Width   : integer := 16;
        Cache_Width : integer := 16;
        Norm_Widht  : integer := 23;
        n       : integer := 16
    );
    port(
        Cache_i_sig : IN signed(Cache_Width-1 downto 0);-- (4,12)
            Cache_j_sig : IN signed(Cache_Width-1 downto 0); --(4,12)
        Wknorm2 : IN signed(Norm_Widht-1 downto 0); --(12,8)
        Norm2sij: IN signed(Norm_Widht-1 downto 0); --(12,8)
        WkSelect: OUT std_logic_vector(1 downto 0);
        lamda   : OUT std_logic_vector(n-1 downto 0);
        Wk : OUT std_logic_vector(Norm_Widht-1 downto 0);
        max_min_enable: in std_logic
    );

    end Component ;
    ------------------------------------------------------------------------
    Component reg is
    generic(n:integer:=16);
    port(clk,rst,wenable:in std_logic;
    d:in std_logic_vector(n-1 downto 0);
    q:out std_logic_vector(n-1 downto 0)
    );
    end Component;
    ------------------------------------------------------------------------

    signal X_Squared              : std_logic_vector((2*Feature_Width)+1
downto 0); --36 bits (12,24)
    signal Y_Squared              : std_logic_vector((2*Feature_Width)+1
downto 0);
    signal X_times_Y              : std_logic_vector((2*Feature_Width)+1
downto 0);
    signal Two_times_X_times_Y    : std_logic_vector(Feature_Width+2 downto
0); --20 bit (13,7)
    signal NormCalculated         : std_logic_vector(Feature_Width+4 downto
0);--22 bit(14,8)
    signal Two_Divide_Ctilde      : std_logic_vector(Ctild_Width downto 0);

    signal NormOfDot_toMux        : std_logic_vector(Norm_Width-1 downto
0);
    signal Sig_Wk_new_point_norm  : std_logic_vector(Norm_Width-1 downto
0);
    signal MuxOfNorm_ToQueue      : std_logic_vector(Norm_Width-1 downto
0);
```

```
  signal NormQueueRegIn_Wkblock  : std_logic_vector(Norm_Width-1 downto
0);

  signal NormQueueRegOut_NormOld  : std_logic_vector(Norm_Width-1 downto
0);

  BEGIN
  Two_times_X_times_Y <=X_times_Y(X_times_Y'HIGH downto
Feature_Width)&'0';
  NormCalculated <= std_logic_vector(
signed("00"&X_Squared(X_Squared'HIGH downto Feature_Width-1))
                    + signed("00"&Y_Squared(Y_Squared'HIGH downto
Feature_Width-1))
                    -
signed(Two_times_X_times_Y(Two_times_X_times_Y'High)&Two_times_X_times_Y(T
wo_times_X_times_Y'HIGH downto 0)&'0') ); -- 21 bit(13,8)--->22(14,8)

  Two_Divide_Ctilde <= Ctilde&'0';--11 bit

  NormOfDot_toMux <= std_logic_vector(
signed(NormCalculated(NormCalculated'High)&NormCalculated) +
signed("0000"&Two_Divide_Ctilde&"00000000") );
  LABEL1: DotProduct generic map (Feature_Width =>17) port map
(Feature1_Class1_Data, Feature1_Class1_Data, Feature2_Class1_Data,
Feature2_Class1_Data, Feature3_Class1_Data, Feature3_Class1_Data,
X_Squared); --(10,24)
  LABEL2: DotProduct generic map (Feature_Width =>17) port map
(Feature1_Class2_Data, Feature1_Class2_Data, Feature2_Class2_Data,
Feature2_Class2_Data, Feature3_Class2_Data, Feature3_Class2_Data,
Y_Squared); --(10,24)
  LABEL3: DotProduct generic map (Feature_Width =>17) port map
(Feature1_Class1_Data, Feature1_Class2_Data, Feature2_Class1_Data,
Feature2_Class2_Data, Feature3_Class1_Data, Feature3_Class2_Data,
X_times_Y); --(10,24) => (10,6)

  LABEL4: mux2x1 generic map(n=>Norm_Width) port map
(Sig_Wk_new_point_norm
,NormOfDot_toMux,Select_Norm_queueIn,MuxOfNorm_ToQueue); --(15,8)
  LABEL5: fifo2 generic map(m=>Norm_Width) port map
(clk,Reset_fifo_Norm_wkblock,WriteEnable_fifo_Norm_wkblock,ReadEnable_fifo
_Norm_wkblock,MuxOfNorm_ToQueue,NormQueueRegIn_Wkblock,FifoEmpty_fifo_Norm
_wkblock,FifoFull_fifo_Norm_wkblock);
  LABEL6: Wk_calculate generic map (Feature_Width=>
Feature_Width,Cache_Width=> Cache_Width,Norm_Widht=>Norm_Width,n=>n)port
map
(signed(Cache_i_max),signed(Cache_j_min),signed(NormQueueRegOut_NormOld),s
igned(NormQueueRegIn_Wkblock),WkSelect
,lamda,Sig_Wk_new_point_norm,max_min_enable);
  LABEL7: reg generic map(n=>Norm_Width)port
map(clk,reg_reset_WkBlock,reg_enable_WkBlock,NormQueueRegIn_Wkblock,NormQu
eueRegOut_NormOld);


  END Arch_NormCalculate;
```

13.9Wk_calculate

```
Library ieee;
Use ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

Entity Wk_calculate is
Generic (
    Feature_Width   : integer := 17;
    Cache_Width : integer := 23;
    Norm_Widht  : integer := 23;
    n       : integer := 16
);
port(
    Cache_i_sig: IN signed(Cache_Width-1 downto 0);-- (11,12)
        Cache_j_sig : IN signed(Cache_Width-1 downto 0); --(11,12)
    Wknorm2 : IN signed(Norm_Widht-1 downto 0); --(15,8)
    Norm2sij: IN signed(Norm_Widht-1 downto 0); --(15,8)
    WkSelect: OUT std_logic_vector(1 downto 0);
    lamda   : OUT std_logic_vector(n-1 downto 0);
    Wk : OUT std_logic_vector(Norm_Widht-1 downto 0);
    max_min_enable: in std_logic

);

end entity Wk_calculate ;

Architecture Wk_calculate_arch of Wk_calculate is

-------COMPONENTS: ------
Component comparator is
Generic (n : integer := 16);
port(   In1 : IN std_logic_vector(n-1 downto 0);--top
    In2 : IN std_logic_vector(n-1 downto 0); --bot
        Y : out std_logic_vector(1 downto 0)


);
end Component comparator;

Component divider is
Generic (input_width : integer := 16);
port(
    Q: in std_logic_vector(input_width-1 downto 0);
    M: in std_logic_vector(input_width-1 downto 0);
    Quo: out std_logic_vector(input_width-1 downto 0);
    Remi: out std_logic_vector(input_width-1 downto 0)
);
end Component divider;
Component BoothTop is
port(
    M: in std_logic_vector(15 downto 0);
    Q: in std_logic_vector(15 downto 0);
    Z: out std_logic_vector(31 downto 0)
```

```
    );
    end Component BoothTop;

    ---------------------------

    SIGNAL wksijmul2sig: signed(Cache_Width downto 0); --24 bit (12,12)
    SIGNAL sigTop: signed(Cache_Width+3 downto 0);--27 bit(15,12)

    SIGNAL sigWksij: signed(Cache_Width-1 downto 0);--23 bit (11,12)
    SIGNAL sigBot:  signed(Cache_Width+3 downto 0);--27 bit(15,12)
    SIGNAL siglamda: std_logic_vector(34 downto 0);
    SIGNAL mulLamdaTop: std_logic_vector(42 downto 0);
    SIGNAL onSegPt: std_logic_vector(Norm_Widht-1 downto 0);
    SIGNAL remainder: std_logic_vector(34 downto 0);
    SIGNAL sigWkSelect: std_logic_vector(1 downto 0);

    signal Cast_sigTop:std_logic_vector(Cache_Width+3 downto 0);
    signal Cast_sigBot:std_logic_vector(Cache_Width+3 downto 0);--27
    signal test:std_logic_vector(Cache_Width+3 downto 0);

    signal Div_Cast_sigTop:std_logic_vector(34 downto 0);
    signal Div_Cast_sigBot:std_logic_vector(34 downto 0);

    signal sigtestWknorm2 :signed(Cache_Width+3 downto 0);
    signal Cache_i: signed(Cache_Width-1 downto 0);-- (11,12)
    signal Cache_j: signed(Cache_Width-1 downto 0);-- (11,12)

    Signal Square_sigTop: std_logic_vector(53 downto 0); --sigTop 27 bit
then squared 54 bit (53 downto 0)(30,24)
    Signal Reminder_squareTop: std_logic_vector(53 downto 0);
    Signal SquareDiv_Bot: std_logic_vector(53 downto 0); --(30,24)-->(15,8)
    signal Top2_Bot:std_logic_vector(53 downto 0); --54 bit(46,8) and
downnto 16 bit (15,8)



    Begin

    Cache_i<= Cache_i_sig when max_min_enable='1';
    Cache_j<= Cache_j_sig when max_min_enable='1';


    sigWksij <= Cache_i - Cache_j; --23 bit (11,12)

    test<=std_logic_vector( resize(sigWksij,27));

    sigtestWknorm2<= Wknorm2&"0000";

    sigTop <= sigtestWknorm2 - signed(test); --27 bit(15,12)

    Cast_sigTop<=std_logic_vector(sigTop);

    wksijmul2sig <=  sigWksij&'0'; --23(11,12) *2 24(12,12)

    sigBot <= (Wknorm2&"0000")- resize(wksijmul2sig,27) +
(Norm2sij&"0000");--27 bit (15,12)
    Cast_sigBot<=std_logic_vector(sigBot);
```

```
   --============================================
   Square_sigTop<=std_logic_vector(sigTop*sigTop);--squared signal (30,24)
   SquareDiv_Bot<=std_logic_vector("000000000000000000000000"&Cast_sigBot&"
0000");--(38,16)
   ------------------------------ COMPARISION:
   CompTopBot:
comparatorgenericmap(n=>27)portmap(Cast_sigTop,Cast_sigBot,sigWkSelect);

   ------------------------------ DIVIDER (find lamda):
   Div_Cast_sigTop<=std_logic_vector(Cast_sigTop&"00000000");
   Div_Cast_sigBot<=std_logic_vector("00000000"&Cast_sigBot);

   siglamdaport: divider generic map(input_width => 35) port
map(Div_Cast_sigTop,Div_Cast_sigBot,siglamda,remainder);

   sigSquare: divider generic map(input_width => 54) port
map(Square_sigTop,SquareDiv_Bot,Top2_Bot,Reminder_squareTop);

   lamda<=siglamda(15 downto 0);
   onSegPt<= std_logic_vector(Wknorm2 - signed(Top2_Bot(22 downto 0))); --

   ------------------------------ MULTIPLIXER
   WKSelect <= sigWkSelect;

   Wk  <= onSegPt when sigWKSelect = "00" else
        std_logic_vector(Wknorm2) when sigWKSelect = "10" else
        std_logic_vector(Norm2sij) when sigWKSelect = "11" else
        (others=>'0');


   end Wk_calculate_arch;
```

13.10exp_lut

```
   library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.numeric_std.all;
   entity exp_lut is
   port(
     i_x            : in  std_logic_vector( 15 downto 0);  --(4,12)
     o_exp          : out std_logic_vector( 15 downto 0)); --(4,12)
   end exp_lut;
   architecture rtl of exp_lut is

   constant C_LUT_DEPTH    : integer := 14338;
   constant C_LUT_BIT      : integer := 16;

   type t_lut_exp is array(0 to C_LUT_DEPTH-1) of integer range 0 to
(2**C_LUT_BIT);
   constant C_LUT_ADDR_OFFSET         : unsigned(15 downto 0):=
to_unsigned(28672,16);
```

```
  signal lut_addr                    : signed(14 downto 0);
  signal lut_preaddr                 : signed(15 downto 0);
  signal lut_value                   : std_logic_vector(C_LUT_BIT-1
downto 0);

  constant C_LUT_EXP : t_lut_exp :=(
0,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,4,4,
4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
4,4,4,4,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,5,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,
6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
7,7,7,7,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
8,8,8,8,8,8,8,8,8,8,8,8,8,8,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,
9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,
9,9,9,9,9,9,9,9,9,9,9,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,1
0,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,
10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10
,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,1
0,10,10,10,10,10,10,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,
11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11
,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,1
1,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,12,12,12,
12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12
,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,1
2,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
12,12,12,12,12,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13
,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,1
3,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,
13,13,13,13,13,13,13,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14
,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,1
4,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,14,
14,14,14,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15
,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,1
5,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,16,16,16,16,16,
16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16,16
,……, --generated using MATLAB script);
```

```
  begin

  lut_preaddr <= signed(C_LUT_ADDR_OFFSET) + signed(i_x); -- Input +
28672

  lut_addr    <=   to_signed(C_LUT_DEPTH-1,15) when signed(i_x) > 0 else -
-outputs exp(0) if the i/p > 0
          lut_preaddr(15 downto 1)+1 when
lut_preaddr(lut_preaddr'HIGH)='0' else --outputs exp(i/p) if i/p [-7,0]
          (others=>'0'); --outputs zero if i/p < -7

   o_exp      <= lut_value;
   lut_value <=
std_logic_vector(to_signed(C_LUT_EXP(to_integer(lut_addr)),C_LUT_BIT));


  end rtl;
```

References

[1] *Epilepsy and Seizures - A Historical Perspective - Myths and Misconceptions*. (n.d.). Retrieved from http://science.jrank.org/pages/cma5hkecsy/Epilepsy-Seizures-Historical-Perspective.html

[2] M. England, C. Liverman, A. Schultz, and L. Strawbridge, "Epilepsy across the spectrum: Promoting health and understanding: A summary of the Institute of Medicine report," Epilepsy Behav., vol. 25, no. 2, pp. 266–276, Oct. 2012.

[3] D. Yoon, K. Frick, D. Carr, and J. Austin, "Economic impact of epilepsy in the United States," Epilepsia, vol. 50, no. 10, pp. 2186–2191, Oct. 2009.

[4] On the Frontline of Epilepsy Treatment. (n.d.). Retrieved from https://www.webmd.com/epilepsy/frontline-epilepsy-treatment#2

[5] A. Varsavsky, I. Mareels & M. Cook. (2011). *Epileptic Seizures and the EEG.* New York: CRC.

[6] A. Abo El-Makarem, A. Fouad, T. Kamel, K. Mohamed, M. Kamal El-Din & M. Abd El-Rahman. (2016). Implantable Seizure Detector & Predictor. Cairo: Faculty of Engineering, Cairo University.

[7] M. Parvez & M. Paul. (2016). *Prediction and Detection of Epileptic Seizure.* Australia: Charles Sturt University.

[8] A. Shoeb. (2009, September). *CHB-MIT Scalp EEG Database*. Retrieved from https://www.physionet.org/pn6/chbmit/

[9] J. Platt. (1998). *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines.* Microsoft Research.

[10] P. Carney, S. Myers & J. Geyer. (2011, December 22). *Seizure Prediction: Methods*. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3233702/

[11] N. Moghim & D. Corne. (2014, June 9). *Predicting Epileptic Seizures in Advance*. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4049812/

[12] F. Mormann, R. Andrzejak, C. Elger & K. Lehnertz. (2006, August 10). *Seizure prediction: the long and winding road*. Retrieved from https://fenix.tecnico.ulisboa.pt/downloadFile/3779571469443/SeizurePrediction.pdf

[13] A. Aarabi, R. Fazel-Rezai & Y. Aghakhani. (2009). *EEG Seizure Prediction: Measures and Challenges.* Minnesota: IEEE Engineering in Medicine and Biology.

[14] T. Maiwald, M. Winterhalder, R. Aschenbrenner-Scheibe, H. Voss, A. Schulze-Bonhage & J. Timmer. (2004). *Comparison of three nonlinear seizure prediction methods by means of the seizure prediction characteristic.* Freiburg: Freiburg Center for Data Analysis and Modeling.

[15] M. Papadonikolakis & C. Bouganis. (n.d.). *Efficient FPGA Mapping of Gilbert's Algorithm for SVM Training on Large-scale Classification problems.* London: Imperial College London.

[16] Theodoridis, Sergios (2008). Pattern Recognition. Elsevier B.V. p. 203. ISBN 9780080949123.

[17] *Intel Completes Acquisition of Altera*. (2015, December 28). Retrieved from https://newsroom.intel.com/news-releases/intel-completes-acquisition-of-altera/

[18] *Machine Learning: What it is and why it matters*. (n.d.). Retrieved from https://www.sas.com/en_us/insights/analytics/machine-learning.html#machine-learning-importance

[19] Simpson, P. (2010). *FPGA Design: Best Practices for Team-based Design* . San Jose: Springer.

[20] Taylor, A. (2012, July 8). *The basics of FPGA mathematics*. Retrieved from https://www.eetimes.com/document.asp?doc_id=1279807

[21] *What Is Machine Learning? 3 things you need to know*. (n.d.). Retrieved from https://www.mathworks.com/discovery/machine-learning.html