Masters's Thesis

Hardware Design of Real-Time Neural Signal Generator

Anil Kumar Metla

Department of Electrical and Information Technology,
Faculty of Engineering, LTH, Lund University, 2016.

Hardware Design of Real-Time Neural Signal Generator

Anil Kumar Metla
mas09ame@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Palmi.T.Thorbergsson

Co-Supervisor: Johan Lofgren

Examiner: Anders.J.Johansson

December 16, 2015

Printed in Sweden
E-huset, Lund, 2015

Abstract

Brain Machine Interface (BMI) denominates a collection of systems which interface with the Central Nervous System (CNS). Implementation of a BMI involves the detection, extraction, processing, and translation of the signals from the Central Nervous System.

A simulator to generate extracellular recordings is proposed by P.T.Thorbergsson, H. Jorntell, F.Bengtsson, M.Garwicz, J. Schouenborg, A.J Johansson in [1]. The described simulator is available as a script implemented in the numerical computing software "Matlab". To measure the performance of the BMI systems, the Matlab script is needed to be implemented on a Field Programmable Gate Array (FPGA) providing real-time signals. The first stage of the implementation on the FGPA requires a hardware design of the simulator. This report presents the hardware design of the Real-Time Neural Simulator.

The Real-Time Neural Simulator the original Matlab script was to be adapted for the hardware design. The adaptation process involved replacing the Matlab closed source signal processing and mathematical functions with hardware implementable algorithms. The performances of these algorithms are successfully verified against the Matlab functions.

The Neural Simulator Matlab script is converted from floating point to fixed point implementation and the performance is verified for different word lengths. One of the quantitative measurements for comparison of the output between original script and final fixed point script was the percent of energy difference of the output signal. For a word length of 13-bits with 1 sign, 6 decimal, and 6 fractional bits, the calculated difference is less than 5 percent depicting the strong resemblance between the outputs.

Finally the hardware design of the simulator constitutes of interconnected hardware units. At the highest level a "Data Processing Unit" generates the target and the noise recordings, "Controller" enables, disables the processing modules and routes the data between them, and finally the RAM and ROM form the memory units to store the input and processed data.

Acknowledgments

I thank my supervisor Palmi.T.Thorbergsson for being very patient and supportive all the way through my thesis, It has been a great learning experience. Thanks to Johan Lofgren, co-supervisor for his wonderful guidance. I am thankful to my examiner Dr.Anders Johansson for giving me the opportunity to work with the thesis, Dr.Anders has been a great inspiration. I would also like to thank Dr.Joachim Rodrigues for helping to find the thesis project and teaching the most important courses that formed basis of my knowledge of System on Chip, also he was a great motivation and inspiration for me throughout the Master's Program. I thank all the professors and lecturers who have helped to learn and encouraged to explore during the Master's program. Love my family and friends for being supportive till the end.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction and Overview

Aim of the thesis is to present an FPGA implementable hardware design of the Real-Time Neural Simulator based on the work "Spike Library Based Simulator for Extracellular Single Unit Neuronal Signals" by P. T. Thorbergssson, H. Jorntell, F. Bengtsson, M. Garwicz, J. Schouenborg, A. J. Johansson. The simulator proposed in [1] is implemented as a Matlab script by Palmi. T. Thorbergsson, formed the baseline code for the thesis.

Understanding the simulator proposed in [1] and the related concepts formed the Literature study of the thesis. The literature study is followed by the study of the simulator implemented in Matlab code. During the study phase, the simulator is run multiple times in the Matlab and output is studied.

Replacing the various signal processing and mathematical functions, adapting parts of code to suit parallel processing, and modifying entire code to use fixed point data formed the final stage of working with Matlab script. Once the fixed point implementation was successful, a hardware design of the adapted Matlab code is proposed to achieve the goal of the thesis.

Figure 1.1 and the text below will explain the various steps involved in the implementation of the thesis and their presentation in this report. Chapters 2-7 will explain the adaptation and design process while chapter 8 will present the results.

- Literature Study: The Background of Brain Machine Interface is analyzed and is explained in chapter 2.

- Analysis of the Simulator: A functional analysis is performed on the original simulator script proposed in [1]. A data analysis is also performed on the simulator. Chapter 3 in detail explains the proposed simulator in detail along with the Matlab script.

- Replacement of Library functions: The Matlab script of the simulator proposed in [1] uses the signal processing, mathematical and data processing functions from the Matlab library. Since source code for the Matlab library functions is not available, equivalent algorithms providing the required functionality are to be implemented. Chapter 4 lists and explains in detail the various algorithms used to facilitate the replacement of the library function. It also explains the rationale behind choosing the respective algorithms.

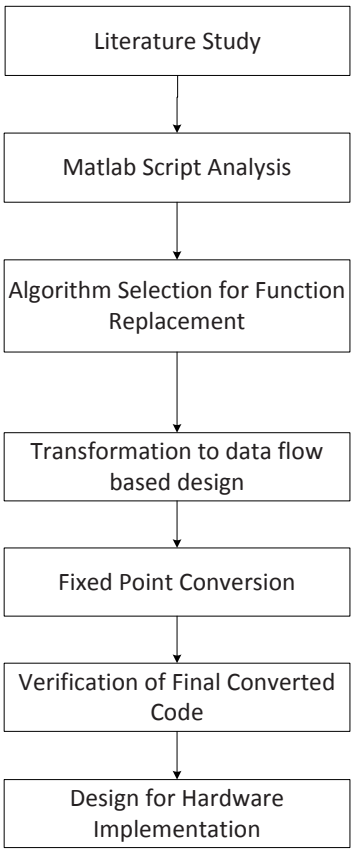

Figure 1.1: Thesis Implementation Flow

- Adaptation of Data Processing: The Matlab runs on a Personal computer with large memory and can be used for storing huge amounts of data and perform required operations on it. The same is not feasible in hardware since it requires huge amount of memory and implementation overhead. To

achieve the same output without storing the data, the implementation of the simulator is changed to a model where the data is processed sequentially and continuously producing the required output while eliminating the need for storing huge amounts of data. Chapter 4 shows the analysis of the simulator data flow and explains in detail the data processing adaptation process.

- Fixed Point Implementation: Matlab implementation of the simulator is in floating point. Implementing floating point in hardware is costly both in terms of speed and area and requires a floating point unit. Implementation using fixed point on hardware reduces the complexity of the implementation and eliminates the requirement for the floating point unit. The disadvantage with the fixed point implementation is the deviation in the output since the word length which includes fractional point length determines the amount of error in the data representation and processing. With a selection of proper word length and fractional length the error can be minimized and an output with good accuracy can be obtained.
- Design of the hardware: The final step in the process is to design the hardware that implements the adapted fixed point simulator. The hardware design is performed with abstraction at each level of design which is a recurring process until it arrives at the basic building blocks. The detailed hardware design is explained in Chapters 5, 6, and 7

Chapter 2
Background

Central Nervous system in the humans is responsible for the transmission, reception and processing of information from different parts of the body. Central Nervous System (CNS) constitutes of the Brain and the Spinal cord. Spinal cord is the interface between the brain and the peripheral nervous system and the center for various reflexes. Brain is not only responsible for the simple sensory information reception and motor information transmission, but also for the more advanced and complex functionalities of Cognition and Behavior.

Nervous system achieves its functionality by the interconnected cells. There are two major classes of cells in the nervous system, Neurons and the Glial cells or Glia. Neurons are the basic units of the brain, the interconnections and the transmission of the signals between the neurons is responsible for the functionality of the brain. Glial cells which derive their name from the Greek for glue surround cell bodies, axons and dendrites of neurons [3], Glia are not directly involved in the electrical signaling. The further discussion about Glia is out of scope of this thesis, interested readers can refer to [3] for more information.

2.1 Neurons

Neurons are the signaling units of the nervous system. A typical neuron as shown in Figure 2.1 has four regions, Cell body, Dendrites, Axon and Presynaptic terminals [3]. Cell body or Soma is the place where metabolic activity takes place. Soma contains the Nucleus where the proteins are synthesized. Dendrites are extensions of the cell, and they branch out from the cell body. Dendrites are responsible for receiving the incoming signals from the other cells. An Axon is responsible for carrying the electrical signals from the cell, called action potentials. Axon hillock is the last part of the cell before the beginning of the axon. Axon hillock is responsible for adding the input excitation potentials to the cell before being passed on to the initial segment of the axon. The axon can be quite long and also can be branched out before ending on either other cells or the dendrites of the other cells. An axon carries electrical potential of amplitude of 100mV without any drop. The axon is insulated in the Myelin sheath which helps the insulation of axon. Finally the places where axon makes contact with other cells are called synapses. Presynaptic cell is the one transmitting the signal and the post-synaptic cell is the receiving cell.

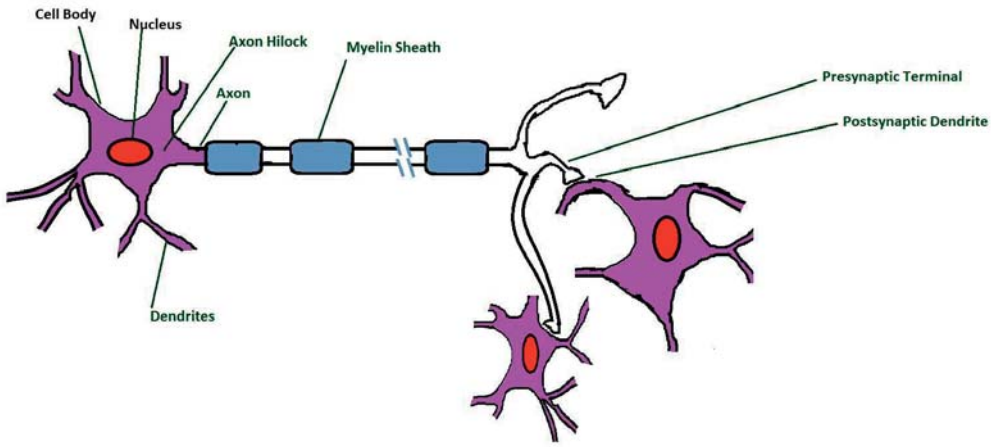

Figure 2.1: Structure of a Neuron

2.1.1 Neuron Signaling and Action Potential

Action potentials are the signals responsible for the transmission, reception and the analysis functionality by the brain. The information conveyed through this action potential takes different pathways. The brain analyzes the pattern of the incoming signal and pathway it took to analyze the received information. Each neuron maintains an electrical potential difference between the inside the cell and outside of it called resting membrane potential. The input electrical signal from the other neurons is called the synaptic potential. The input electrical signals to the neurons causes decrease in the potential difference. When the total sum of input signals exceed a threshold at the start of axon called trigger zone an action potential is generated. The action potential is approximately of 100mV amplitude and can travel long distances across the neuron. The generation of action potential is an all or nothing affair, which means an action potential is only generated when the input signal sum is above a threshold, otherwise it will be nothing. Since the action potential is the signaling mechanism among the neurons, any study regarding the brain starts with measuring and recording of the action potentials. The various techniques to measure the electrical activity of the brain rely on the measurement of action potentials and an generated action potential is called "spike" at the recording end.

2.2 Introduction to Brain Machine Interface

The field of Brain Machine Interface (BMI) technology aims at developing efficient and reliable electromechanical system that is directly controlled by the signals from the human brain. It is a multi-disciplinary field involving synchronized efforts between Medicinal Science, Electrical Engineering, Signal Processing and Mechanical

Engineering fields.

To understand the Brain Machine Interface it is very important to understand the mechanism of communication between human brain and various organs of humans or animals. Two important functions of peripheral nervous system of human brain is to carry information to central nervous system via various sensory organs called sensory or afferent division and send control signals from central nervous system to various organs via the motor nervous system or efferent division.Figure 2.2 shows the afferent and efferent divisions of the PNS.

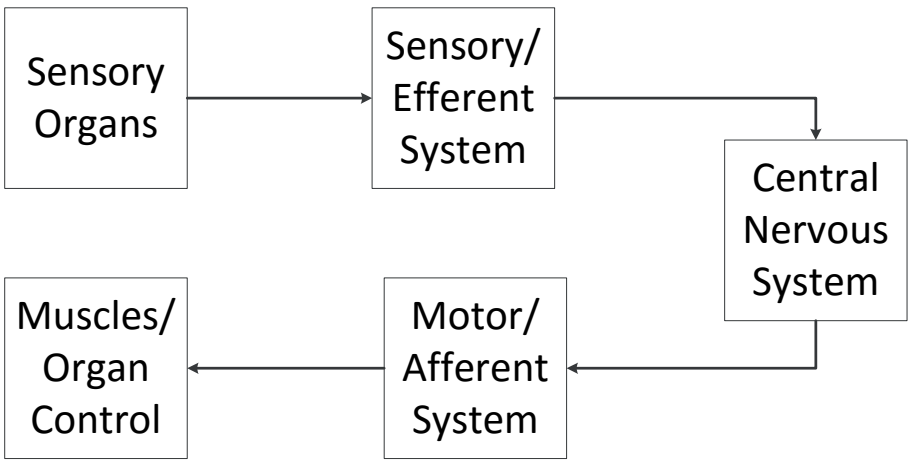

Figure 2.2: Peripheral Nervous System

A true BMI system mimics the natural communication and control between the Central Nervous system and the organs, It is achieved by recording the brain signals, translating them into control signals for the corresponding BMI applications. In essence the BMI provides the non muscular communication between the human brain and the designated application. Figure 2.3 shows the block diagram of the Brain Machine interface [4]

A Brain Machine Interface system can be divided into following constituent blocks

- Sensors: The activity of the brain has to be recorded using the sensors. The sensor translates the brain activity into useful signals to the later stages of system. There are different kinds of mechanisms for signal recording using sensors and is discussed in detail in next section.

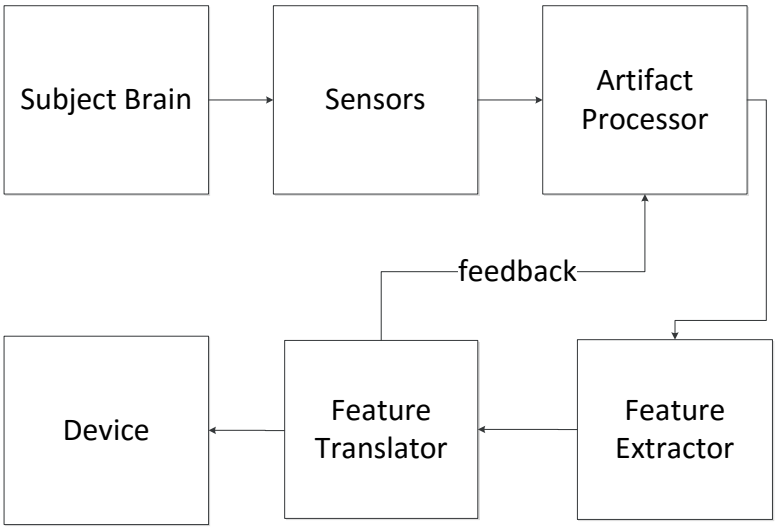

Figure 2.3: Brain Machine Interface

- Artifact Processor: This block is responsible for removing noise and artifacts from the recorded signal. The requirement of artifact processor is subject to the mechanism of signal recording.
- Feature Extractor: Feature extractor analyzes the input brain signal and converts it into a recognizable neurological phenomenon. It basically detects the spikes of the brains, then sorts and correlates it to a particular function of brain from a set of known functionalities. The output of this block is referred to by some as "feature vector" [2]
- Feature Translator: Feature translator maps the feature vector to a control signal that can be applied to the device. The mapping of feature vector to control signal is a very important aspect of the BMI and a feedback from this block is applied to the artifact processor to adapt the process to produce more accurate output.

2.3 Signal Recording Mechanisms

The mechanism of recording of the brain activity is an important part of the BMI system. There have been many types of mechanisms proposed and these mechanisms measure different varieties of brain activities. The two dominant activities measured are

- Electrical Activity: Electrical activity inside the brain is generated by the electro-chemical transmitters and receptors exchanging the information among

the neurons. The electrical activity in the brain can be measured by using electrodes measuring the currents and voltages.

- Chemical Activity: Blood carries oxygen and glucose into the brain and in case of any activity in that part of the brain the oxygen and glucose content vary causing a change in hemoglobin composition in that area. The chemical activity can be measured by using the imaging methods where the different part of brain causes different colors in the image spectrum.

Both the electrical and chemical activity in the brain can be measured using different mechanisms. The brain activity recording techniques can be grouped into two based on the placement of sensors, Invasive and non-Invasive methods. The invasive methods involve implanting the sensor electrodes directly inside the brain, the non-invasive methods externally measure activity of the brain. Non-invasive methods use externally placed electrodes or imaging techniques for measuring the activity in the brain.

To compare quality of various techniques that measure brain activity there requires some parameters that are important that need to be calculated, they are

- Temporal Resolution: The smallest period of neuronal activity that can be measured and distinguished. Smaller the values better the performance of the system.
- Spatial Resolution: Measure of capability to distinguish between two locations inside the brain. Higher the spatial resolution results in better ability to localize the changes.

2.3.1 Non-Invasive Technologies

Non-Invasive techniques use imaging techniques and external electrodes to measure the brain activity. The use external sensors and are harmless but the signal is prone to noise and interference and require a powerful artifact processor. Also the non-invasive techniques provide low spatial and temporal resolutions. Some of the non-invasive technologies are listed below.

- Electroencephalography (EEG): EEG uses electrodes placed on the scalp for measuring the electrical potentials generated. EEG uses gel between electrodes and the scalp making it a wet electrode system. Since the signals are passing through different layers of brain and scalp before reaching the electrodes the signals are prone to high interference from the bio-electrical activity in the environment. EEG recordings have better temporal resolution while limiting the spatial resolution.
- Magnetoencephalography (MEG): In this mechanism the magnetic field produced outside the brain by the intracellular currents is measured. The MEG techniques provide high spatial and temporal resolutions but the systems are still in their initial stages of development.
- functional Near Infrared Spectroscopy(fNRIS): In fNRIS infrared light is projected into brain via the scalp and measuring the optical changes across the spectrum from the reflected light. fNRIS can be used to construct

functional maps of brain activity. fNRIS has high spatial resolution and less temporal resolution.

- functional Magnetic Resonance Imaging(fMRI): fMRI measures the small changes in the Blood Oxidation level-dependent signals associated. It measures the oxidation and glucose levels during the activity in the brain. The advantage is high spatial resolution but low temporal resolution.

2.3.2 Invasive Technologies

Invasive technologies need surgery called Craniotomy to implant the electrodes. The advantages of the invasive methods include high temporal and spatial resolution of the signal and less interference, but they require surgery to implant the electrodes which could be dangerous and once implanted they cannot be moved around.

- Electro Corticogram(ECoG): When the electrodes are placed on the surface of the cortex the resulting signal recorded is called the Electrocorticogram. It doesn't damage neurons since it doesn't penetrate brain. Good spatial and temporal resolution and high frequency range. It is also less prone to artifacts and noise since the distance between measured activity and device is reduced.
- Intra Cortical Neuron recording: Intra cortical neuron recording measures the electrical activity by placing the electrodes inside the brain. The electrodes are implanted inside the cortex. The electrodes inside the cortex measure the spike signals and field potentials from neurons. This technique can again be divided into two kinds of techniques, Intra-cellular recordings and extra-cellular recordings.
  - In the intracellular recordings the electrodes are placed inside single neuron to measure the activity in that neuron. This mechanism has advantages that each neuron can be individually identified and measured but disadvantage that it can damage the neuron and also a small movement of electrode might result in different measurements.
  - In the extracellular recordings the electrodes are placed near the neurons of interest and the activity of a group of neurons is measured using a group of electrodes. This technique has an advantage that the neurons are not damaged. But the measured activity is prone to noise from other neurons. Extracellular recording system is explained in detail in the next section. Both the techniques provide a high spatial and temporal resolution and low noise artifacts.

2.4 Extracellular Recording of Brain Signals

Extracellular signal recordings measure the electric potentials simultaneously from large number of neurons from the the area around the sensor. The measurement of the potential is achieved by the implanted electrodes or electrode arrays near

the neurons. Since the extracellular signal recording is done near the neurons it measures not only the electrical potentials from the neurons that are near but also the various other signals from the environment.

The constituents of an extracellular signal are, "the spike" signal which is the measurement from the action potential from the excited neurons that are close to the electrodes within approximate distance of 50 um [1]. The shape of the spike varies among the neurons, is dependent on the structure of the neuron and the spatial relation between the neuron and the electrode. This variation in spike shapes helps to sort and assign the origin of the spike to individual neurons via "spike sorting". At any given instant the electrodes also record the spiking component from the distant neurons and this component of the signal is called the physiological noise. The electrodes also measure the local field potentials such as the synaptic potential, the signal component from these potentials is of low-frequency in nature and can be filtered out.

The spikes, local field potentials and the physiological noise signal components are generated by the physiological process in the brain. The other signal components are caused by the electrical measurement equipment being used for the recording. The first electrical equipment component is the thermal noise generated in the analog front end of the system, the other one being the power line interference signal with frequency of 50Hz/60Hz based on the power-line signal frequency.

2.5 Extracellular Signal Recording System

BMI based on the Extracellular signal recording system looks much alike the generic BMI showed in 2.3, the differences in the systems reflect the extracellular nature of the recording. Figure 2.4 shows the block diagram for the Extracellular recording based BMI.

In extracellular recording, the implanted electrodes are responsible for the capture of the signals from the brain. Semiconductor, polymer based electrodes arranged Linear or planar arrays form few of the popular choices for the electrodes. Signals recorded by the electrodes are in order of microvolts requiring amplification of these signals. The amplification of the signals is done either by the embedded electronic amplification devices on the electrodes or by standalone amplification device. The important factor during the amplification process is to minimize the noise by maximum input impedance of the amplifier.

The amplified signal is converted into digital domain using the A/D converter. The design criteria of the A/D converter depend on the highest frequency component representing the relevant signal. The signal bandwidth determines the sampling rate and the resolution of the signal. Ideally it is desirable to have higher sampling rate and higher resolution, but higher the data higher the computational requirements thereby putting an upper limit on both sampling rate and resolution. Many papers have presented mechanisms to determine the optimal values for bandwidth and resolution, and the final values are determined on individual setup requirements

In the filtering stage the Low Frequency Potentials are filtered out from the

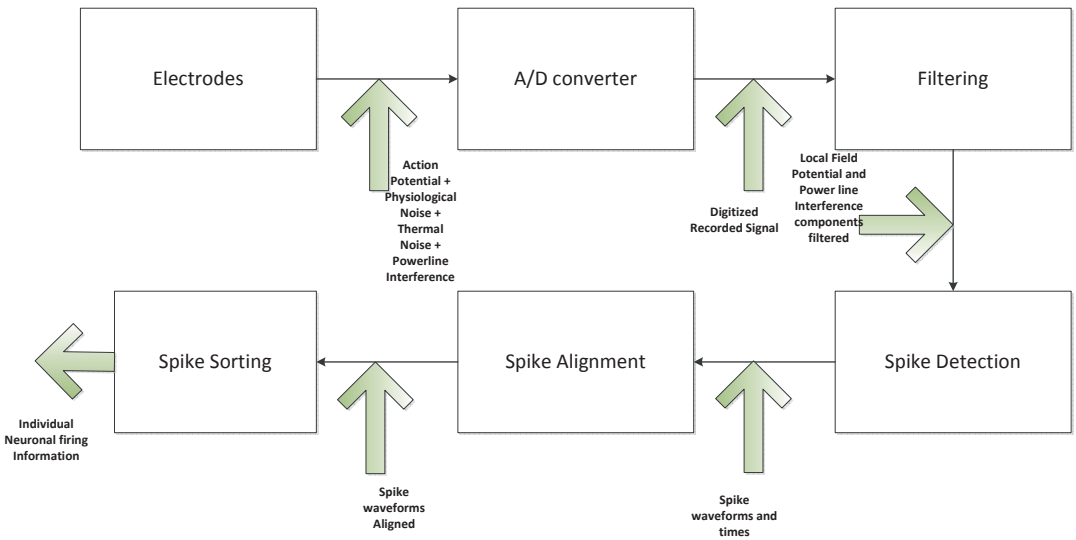

Figure 2.4: Brain Machine Interface

high-frequency spiking component. Usually a bandpass filter is used filter both the LFP's and noisy appearance of the spike shapes.

Spike detection stage has the task of detecting and extracting the spike waveforms form the recording. The input of spike detector is the filtered signal while the output is the timestamps of the action potentials and the spike waveforms. Spike detection can be performed both manually and automatically using algorithms. Manual detection can be performed by reducing the waveforms into shape parameters, plotting them and clustering them by manual inspection. Manual detection has the advantages of a chance to refine the inspection criteria to suit the needs. The disadvantages of manual inspection is that its time intensive with large clusters, not optimal for complex waveforms. The Automatic detection on the other hand is usually is based on processing spikes around a given amplitude threshold. Most of the algorithms proposed for the spike detection involves calculating of the threshold posing a risk of the threshold being too high or too low. If the threshold is too high then there is chance of missing on actual spikes, if the threshold is too low then the noise can be detected as a spike. A good approach for the selecting threshold is to allow some noise while missing none of the actual spikes since the noise can be processed and removed in the later stages.

Before the waveforms are sorted, it is required that they are aligned with each

other properly. Since the actual time at which the spike has crossed the threshold lies between two samples, the extracted samples for the waveform will be shifted in time. This shift will cause misalignment among spikes and is called spike detection jitter. One of the mechanisms to remove the jitter can be by up-sampling the data, aligning the waveforms around a reference point such as its center of mass and then down-sampling in the end.

Final stage in the BMI is the spike sorting, in this stage spikes from different neurons are identified along with their spike times and finally generating individual spike trains. The spike sorting involves feature extraction and clustering the spikes. Feature extraction involves extracting the differentiating characteristic features for spikes from each of the neurons. Feature extraction can be achieved by Discrete wavelet transforms among other mechanisms. Clustering of spikes involves feature based grouping and assigning the groups to the respective neurons. There are various clustering algorithms proposed that involves assumption of Gaussian distribution of the clusters.

Once the results of a BMI is presented, there is a need for a process to validate the output of the BMI. Validation requires the prior knowledge and ground truth of the incoming recorded neuronal spikes. As it is not possible to have ground truth from actual functioning neurons, there is a requirement for a simulator that can provide ground truth while simulating the neuronal activity. One of such simulators is proposed by Palmi.T.Thorbergsson etal in [1], and is explored in the next chapter.

Chapter 3

Analysis of the Spike Library Based Neural Simulator and Matlab Script

3.1 Introduction

The success of implementing an Extracellular recording based BMI system depends on the accurate detection and clustering of the neuronal spikes. The precision of the detection and clustering depends on the algorithms selected. The selected algorithms also influence the design of the signal acquisition and processing hardware. The task of selecting algorithms requires benchmarking by qualitative and quantitative evaluation, and comparison. Any benchmarking strategy of the spike detection and sorting algorithms requires the knowledge of the ground truth from the Central Nervous System(CNS). For the real-world extracellular neuronal recordings it is impossible to have the ground truth information hence requiring the neuronal simulators. The requirement of such a simulator would be to generate recordings that mimic the actual neuronal activity and provide the ground truth of the generated recording signal.

A simulator based on spike library is proposed by P.T.Thorbergsson etal in [1]. The proposed simulator generates extracellular recordings to be processed by the spike detection and sorting algorithms along with the ground truth. The output of the simulator has shown the properties of the extracellular recordings with chronically implanted micro-electrode arrays in the central nervous system. This chapter describes in detail the modeling parameters, neuronal signal generation algorithms, input and output data of the simulator.

3.2 Neural Simulator Overview

A true extracellular recording of a brain with implanted electrodes will contain the spike signal from the neuron of interest, the noise from surrounding neurons, and thermal noise at input of recording amplifier. For a given recording duration the interval between each spike generated by a neuron varies among different neurons. This interval between generation of two spikes is called as the inter spike interval (ISI). The neurons need a cool down period between generating two spikes called refractory period thus setting a higher limit on the frequency of spike generation.

To generate the extracellular recordings via simulation, a simulator need to model various parameters to provide a similar output to that of actual recordings.

Simulator proposed in [1] models various attributes of individual neuronal spike generation, extends them to multiple neurons and finally performs the necessary addition and transformations to generate the final simulated extracellular recording. The simulator models the neuron distribution, inter spike interval distribution, spike amplitude and applies them to the spike library waveforms to generate the basic neuronal recording which then is added to the modeled thermal noise to generate the final simulated extracellular signal output. The following sections describe in detail the modeled attributes, thermal noise, input waveforms and the algorithm used in the simulator.

3.3 Neuronal Spikes

Simulator uses the spike waveforms detected and extracted from recordings performed in various regions in the cat cerebellum. The recordings thus extracted are processed through the open source software package "Chronux". The extraction and processing of recordings resulted in 85 spike waveforms that were up sampled to 100kHz while storing. The duration of each spike is defined as the "time period where the absolute amplitude of the largest phase of spike is above half its peak value" [1]. Spikes with smaller duration are referred to as "fast spikes" which form the majority of the recorded spikes.

As part of [2] the spike library in [1] is statistically modeled and six principal waveforms are extracted with their weight distribution which is a 6-dimensional 2-component Gaussian mixture model. This model in [2] can be used to synthesize a wide range of spikes that can be detected in recordings of the brain. Since the usage of model would make the implementation more complicated, a set of 2000 spikes that are most possible during the recordings from brain are generated as a library with 100kHz sampling rate. The set of 2000 spikes based on [2] forms the spike library used in the simulation.

3.4 Attribute Models

The models of neuron distribution, spike amplitude, and the Inter spike interval used in the simulator are explained below.

3.4.1 Neuron Distribution

The simulator mimics the extracellular recordings of brain with micro-electrode arrays. The simulator assumes an isotropic distribution of neurons around the electrode with spherical volume of influence on electrode. The volume surrounding electrode is divided into two parts near field and far field, the neurons inside the near field are called "Target Units" and neurons inside the far field are "Noise Units". The target units are the neurons of interest for recording, while the neurons inside far field contribute to the physiological noise.

3.4.2 Spike Amplitude

Extracellular recordings usually involve the recording of the amplitude of the voltage of the neuronal spikes from different Neurons. The recorded amplitudes target units normally exceed certain threshold while the amplitudes from the other units decay with respect to the distance. The simulator models the amplitude variation by assuming the target unit voltage as 1, while the noise units amplitude decay is calculated using the the equation below.

$$A = \frac{1}{(Kr+1)^n}$$

n is between 2 and 3
K is the scaling factor that specifies the rate of decay
r is the radial distance of the noise unit from the electrode

3.4.3 Inter Spike Interval

Spike generation of a neuron contains a firing period where it generates the spike wave and an interval which it takes before it fires the next spike. The simulator assumes spike generation times as a renewal process with gamma distributed inter spike intervals for both the target and noise units. The following equation is used to generate the spike times for any given unit 'p'.

$$SpikeTimes[\tau(n)] = \sum_{j=1}^{n} ISI_j, \qquad ISI \sim \Gamma(k,\theta) \tag{3.1}$$

where k and theta are the shape and scale factors of the gamma distribution respectively. For different mean firing rates $\bar{f}$ the value of shape factor varies for different units. The scale factor for gamma distribution is given by the mean ISI, and shape factor k. Simulator assumes different firing rates for different noise unit which are drawn from a uniform distribution.

$$\theta = \frac{\overline{ISI}}{k} = \frac{1}{\bar{f}_a} \qquad where, \overline{ISI} - MeanInterspikeinterval$$

3.5 Thermal Noise Modeling

Thermal noise is modeled using the formula below. It represents the thermal noise present at the input of recording amplifier and calculates the RMS of the noise.

$$v = \sqrt{4kTBR}$$

k- is Botlzman constant, T- Temperature
B- Bandwidth of the system
R - Input resistance of the electrode and amplifier

3.6 Algorithm and Output

Simulator generates a final recording for a given duration with spikes from target, physiological noise from noise units and thermal noise. The simulator also gives out

the values of various parameters used in the simulation called ground truth. The important parts of the ground truth are the target unit spike times and the target unit waveforms. With the recorded waveform and ground truth signal processing algorithms in subsequent stages of BMI can be successfully benchmarked and verified.

3.7 Matlab Implementation

This section in detail explains the Matlab implementation of the simulator. Figure 3.1 shows the block diagram of the implementation in the Matlab.

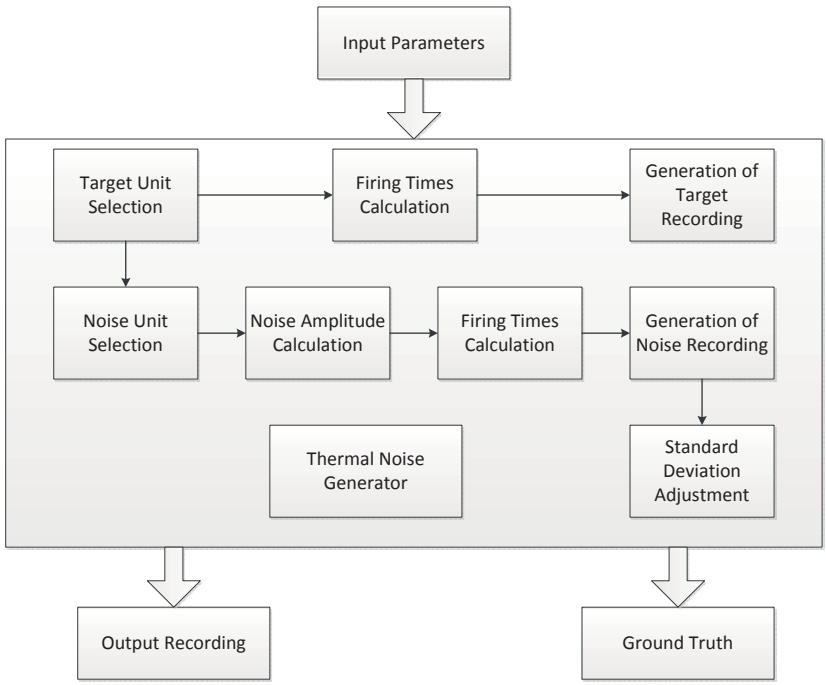

Figure 3.1: Neural Simulator Block Diagram

From the block diagram it can be seen that the simulator takes a set of input parameters, generates target and noise wave forms and sends out the recording and ground truth. The Matlab implementation selects target units followed by the selection of noise units and then generates the target/noise unit amplitudes, firing rates and finally the recording in case of target units and a standard deviation adjustment is performed before generating the noise recording. Input parameters, Target recording generation, Noise recording generation and Final output of the

Matlab implementation is explained in detailed in the following subsections.

3.7.1 Input Parameters

The simulator can be started in Matlab with mandatory input parameter values of duration, output sample rate and number of target units, also the optional inputs can be specified. Table 3.1 shows each of the input parameter with unit used and its significance. In case the optional input parameters are not specified the simulator assumes default values for them. These input parameters are used to derive the modeling values for output generation. The spike waveforms are read from the spike library.

Input Parameter	Unit	Significance
Duration (D)	second	The Duration of Simulation
Sampling rate (f $_s$)	Hz	Sampling rate of output recording
Target Units (N $_u$)	-	Number of target units
Standard Deviation of physiological Noise (σ $_n$)	-	Standard Deviation of Physiological background noise
ffiremeanu(f_u)	spikes/sec	Mean firing rate of target units
ffiremeann(f_n)	spikes/sec	Mean firing rate of noise units
Target Distance	m	Target Unit Amplitude
ISIKGam(k)	-	InterspikeInterval gamma distribution shape parameter
KAmp(K)	-	Rate of amplitude decay in far field
SpikeID	-	Index Number of spike(s) to use for target unit(s)
Thermal Noise Select	-	Selection of Thermal Noise
System Bandwidth (B)	Hz	System Bandwidth
Temperature(K)	Kelvin	Noise Temperature
Input Resistance at the Amplifier(R)	Ohm	Thermal Resistance
Boltz Constant(k)	J/K	Boltzmann constant

Table 3.1: Input parameters for the simulator

3.7.2 Target Recording generation

Target unit generation is explained with a sequence of steps given below.

- Target Unit Selection: Target unit selection involves assignment of a spike waveform for the target units. Index of the spike waveforms can either be specified via the input to the simulator or randomly selected. When spike waveform indexes are specified as input, then corresponding units are taken for generation. Else they are randomly selected using the rand-sample function where each index is equally likely to be selected. The selected target unit indexes are removed from the selection for the noise units since the noise units cannot have same spike waveform as target units.

- Target Firing Time: Inter spike interval between target units is calculated using the equation 3.1. The inter spike interval is a gamma distributed process which is given by its shape and scaling factor. Scaling factor is given by 3.4.3, the target mean firing rate and shape parameter can be either specified as inputs or default values are considered. The shape and scale factors thus calculated are provided for the gamma distributed random number generator function. The final output of gamma distributed random number generation function is utilized to generate the inter-spike intervals using the formula 3.1.

  One important calculation required is the calculation of total number of inter spike intervals required that determines the number of outputs required from the gamma distributed random number generator. Approximate number of spikes for a given duration is calculated using the equation below, and then the final count is made by adding the intervals and keeping the values that fall inside the duration of recording.

$$Number of Spikes = \frac{Duration of Recording}{ISI_{Mean}}$$

- Target Recording: The amplitude scaled target spike waveforms are spread across the duration of recording separated by the inter spike intervals calculated.

3.7.3 Noise Recording generation

Noise recording is a sum of physiological noise and thermal noise.

Physiological noise generation

As explained the physiological noise is the result of the interference of the spike waveforms generated by the neurons surrounding the Target Neurons. Simulating the physiological noise is explained in series of steps below

- Noise Unit Selection: Noise unit selection involves assignment of spike wave form for the noise units. Indexes of the spike waveforms are selected randomly from the library using the random sample function with replacement

which means the same spike waveform can be assigned to more than one neuron. The number of noise units in any recording is 960 based on an approximation [1].

- Noise Unit Amplitude Calculation: Second step is to generate the scaling factors. Scaling factors are generated according to the equation 3.4.2, where n is 2.9. In that equation the r is the distance of the noise unit from the electrode. Since the volume around the electrode is assumed as a sphere, each point in the sphere is represented with the (x,y,z) co-ordinates. To calculate the distance of the noise unit from the electrode 3 coordinates (x,y,z) are necessary, but since it is assumed a isotropic distribution of points a uniform random number generator output is used. Also for the neuron to contribute to the noise it should be inside the far field which is between the Radius of the Inner sphere Ri and Radius of the outer sphere Ro , the values of Ri and Ro are 50 and 140um respectively. Finally a random position is generated by following equation where rand function generates the uniform random numbers required, Ro is the radius of outer sphere of influence.

$$xyzt = -R_o * 2 + 2 * 2 * R_o.rand(1, 3)$$

With the random position defined by the above equation, the radius is calculated by calculating the Euclidean distance from the origin. The calculated radius is checked if it is between near-field and far-field radii values of 50um and 140um respectively. If it is inside the radius is used to calculate amplitude using the equation else it is generated again. Thus generated radius is used in 3.4.2 to generate the amplitude scaling factor applied to the noise unit spike waveforms.

- Noise Unit Inter spike Intervals: The noise unit inter spike intervals are calculated in the same way as the target units except that the simulator only takes the input of maximum and minimum firing rates from which the values from which each of the noise units generates the firing rate. The firing rate for the noise units is calculated using the equation given below, the random number generator used is uniformly distributed random number which generates the values between 0-1 thus distributing the firing rates between min and max values.

$$firingrate = firingrate_{min} + (firingrate_{max} - firingrate_{min}) * rand(1)$$

The values generated above along with the scaling factor are used to generate a gamma distributed inter spike intervals.

- Noise recording: The amplitude scaled noise spikes are spread across the duration of recording according to the individual inter spike intervals. The total physiological noise output is sum of individual noise recordings generated.

- Standard Deviation Adjustment: The physiological noise generated in above step is scaled to achieve required noise level. This is achieved by calculating the standard deviation of noise generated in the above step and then taking ratio of required standard deviation to the calculated standard deviation. The ratio gives the scaling factor that is applied to the total physiological noise generated in above step to generate the final physiological noise.

Thermal Noise Generation

Thermal noise is calculated from the RMS of the noise by using the formula below, where T is temperature, k- Boltzmann Constant , B is Bandwidth and R is the Input resistance all values are optional input values to the simulator.

$$v = \sqrt{4kTBR}$$

The rms value calculated above is substituted in equation below to generate a thermal noise component to the recording. Normally distributed random number is used as an input to the thermal noise generation.

$$Recording = randn * v * A$$

A - Amplitude of the noise
randn - random number drawn from a normal distribution

Ground Truth field	parameter
recording	spike wave forms spike wave form ids spike times spike duration
signals	Target unit recording Noise Recording Thermal noise recording
simulation parameters	Target unit firing rates Noise unit firing rates inter-spike interval kAmp Target Amplitude scaling factor standard deviation of physiological noise Gain, Resistance, Bandwidth and Temperature for calculating thermal noise

Table 3.2: Ground truth the simulation

3.7.4 Output Recording

The final output recording is a sum of the target spike recording and the noise spike recording which includes physiological noise and thermal noise. Since without the ground truth the recording is of no value, a ground truth is generated. The ground truth output consists of the recoding details and the signals and simulation parameters as shown in table 3.2

Chapter 4

Analysis and adaption of Matlab library functions

The Neural Simulator proposed in [1] is implemented in Matlab using the library functions. Since the aim of the thesis is to design the simulator for hardware implementation, the source code should be available for all the library functions. Matlab is a closed source package, each of the library functions has to be replaced with equivalent algorithm implementations that provide similar outputs.

Matlab implementation of the simulator is executed on the PC which has abundant hardware resources available. The hardware design involves a resource constrained system. Any system implementation that requires storing of huge amounts of data and bulk processing has to be modified to suit the hardware design for FPGA implementation.

Important phase in any hardware design is the selection of the data representation, which can either be fixed point or floating point. The selection of floating point would consume huge hardware resources hence the hardware design of the simulator uses fixed point. The fixed point implementation requires the selection of the word length of the implemented system. The word length selection requires the conversion of the Matlab code into fixed point and running the system at different word lengths.

First section of the chapter explains the algorithm selection criteria, fixed point implementation details, and finally the removal of bulk processing of data by data flow conversion. Section 4.2 explains the data flow conversion of the variance calculation. Each section following the first section will explain an individual algorithm used, and results from the implementation.

4.1 Algorithm Implementation

4.1.1 Algorithm Selection Process

The replacement of library functions involves selection of a algorithm that is suitable for hardware implementation while producing the output that is reasonably accurate. The process followed for the selection of algorithm used as a Matlab library function replacement is shown in figure 4.1.

- Identification of the Matlab algorithm: This step involves the identification

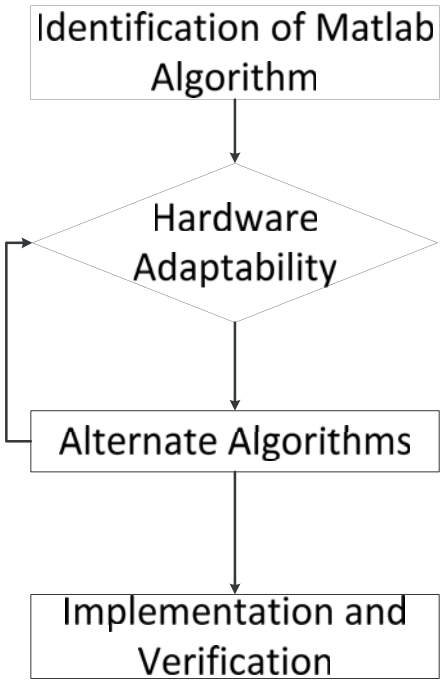

Figure 4.1: Algorithm Selection Process Flow Chart

of the algorithm used in the implementation of the particular library function in Matlab. This step helps in producing the output that accurately resembles the Matlab output.

- Check Hardware Adaptability: In this step the identified algorithms are checked if they are easy to implement in hardware by looking at the complexity of the algorithm. Any algorithm that uses shifts and additions is considered as easy to implement on hardware.
- Find Alternate Algorithm: This step is reached if the algorithm used in Matlab is unknown, or implementation details are unknown or too complex to implement on hardware. Selection criteria for the algorithm is again reasonably accurate output and hardware adaptability.
- Implementation and Verification: The implementation of algorithm involves two parts, the first part involves floating point implementation and comparison of output with Matlab library function. The second part involves fixed point implementation and comparison of output for various wordlenghts of the implementation.

4.1.2 List of Library Functions

All the library functions used in the Matlab implementation of simulator are identified and then analyzed if they require to be adapted. The functions that are not adapted are the ones involving the file operations. 4.1 shows the library functions identified to be replaced with user defined algorithms.

Function	Significance	Algorithm	Matlab Algorithm
rand	uniformly distributed random number generator	mersenne-twister algorithm	Mersenne Twister mt-19997
randn	normally distributed random number generator	Ziggurat	Ziggurat
gamrnd	gamma distributed random number generator	Ziggurat	Ziggurat
log	logarithm calculation	cordic	unknown
exp	exponent calculation	cordic	unknown
sqrt	square root calculation	libfixmath	unknown
div	Division calculation	libfixmath	unknown
randsample	for random sampling	random sampling algorithm from Matlab open source	unknown
randomperm	internally used by randample	Matlab open source	unknown
sort	sorting	insertion sort	unknown
max	max value	simple compare	unknown

Table 4.1: Implemented Library functions and algorithms

4.1.3 FixedPoint Conversion

The next step in the Real Time Neural Simulator hardware design is the fixed point conversion. The code in Matlab is using the floating point numbers for calculations. Implementing the design in the hardware using the floating point is costly as it requires a floating point unit. Fixed point implementation is straight forward in the hardware designing. Fixed point implementation causes error in the results due to the rounding and truncation. Also the error is propagated along the calculations and can lead to huge error in the final output. To verify the effects of finite length data on the implementation of simulator, all the functions and data is converted into fixed point functions. This is achieved by converting the floating point data to integer data and performing the operations on the integer data.

4.2 Data Flow Conversion

Physiological noise unit recording is normalized by a scaling factor of ratio of calculated and expected standard deviation of the same [1]. This calculation is easy to do in Matlab since the entire recording is calculated and held in an array which is fed as input to the standard deviation function. Implementing similar algorithm on hardware is impractical because the amount of data to be stored is variable and the generated data is continuously utilized without storing. If exactly same algorithm as Matlab is to be implemented the simulator need to have really huge memory available and the noise generation has to be run twice, this approach also limits the simulator running continuously for longer periods. In order to overcome these limitations an estimate of standard deviation has to be performed where the estimated value is within the acceptable error limit.

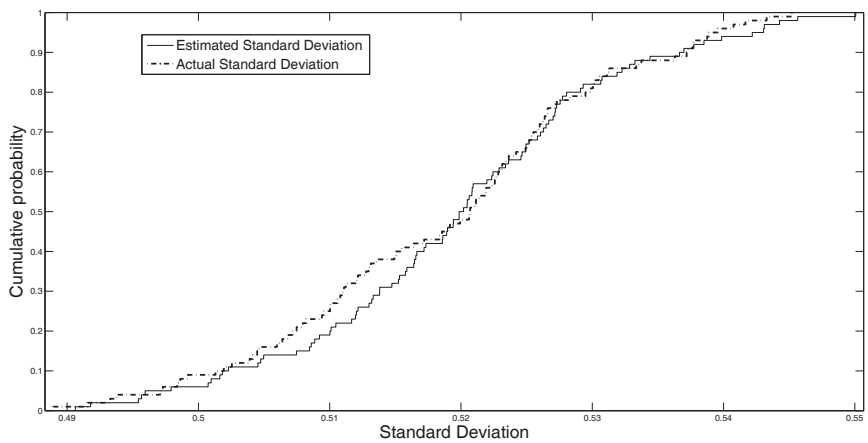

Figure 4.2: CDF of the standard deviation calculation

Calculation of the standard deviation is based on the identity that the variance of the sum of independent random variables is the sum of the individual variances.

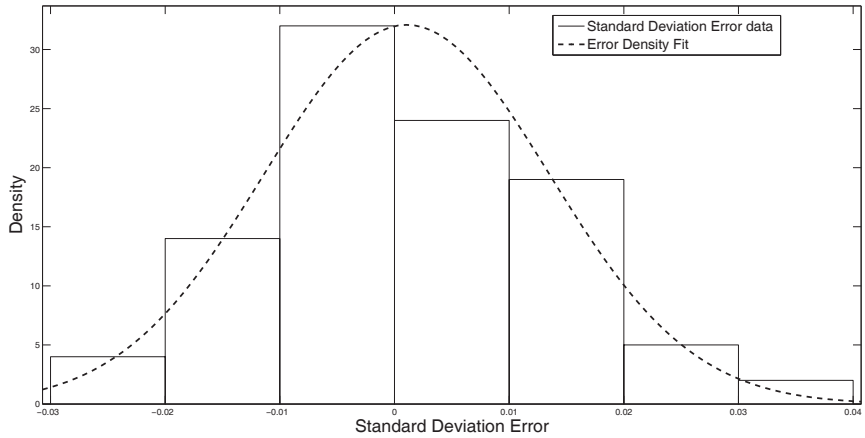

Figure 4.3: PDF of the error

The Energy of a signal v(k) is given by

$$E_v = \sum |v(k)|^2 \tag{4.1}$$

Variance of a signal is given by

$$E_v^2 = \frac{1}{K} \sum_k |v(k) - \mu|^2 \text{where K is the length of v(k) and } \mu \text{ is the mean of v(k)} \tag{4.2}$$

The variance of the n-th spike train is estimated as

$$var(V_n) = E(Vn^2) - (E(V_n))^2 \tag{4.3}$$

Assuming K samples in the recording and L samples per spike and f_n is the mean firing rate of neuron n we have

$$E(V_n^2) = f_n * D * L * E(S_n^2)/K \tag{4.4}$$

substituting D = K/f_s and S_n = A_n*S_{0n}

$$E(V_n^2) = (f_n/f_s) * L * A_n^2 * E(S_{0n}^2) \tag{4.5}$$

$$E(V_n) = (f_n) * D * L * E(S_n)/K = (f_n/f_s) * L * (E(S_{0n}))^2 \tag{4.6}$$

simplifying

$$variance = E(V_n^2) - E(V_n)^2 = (f_n/f_s) * L * A_n^2 * E(S_{0n}^2) - (fn/fs) * L * (E(S_{0n}))^2 \tag{4.7}$$

$$stddev = \sqrt{(sum_n(variance))} \tag{4.8}$$

After implementing the standard deviation estimation algorithm, the calculation of standard deviation is done with estimation and the actual formula and the results are used to plot the CDF of the standard deviation and the PDF of the error and the results are shown in 4.2, and 4.3. It can be observed that the standard deviation calculated from the estimation is within the acceptable error range and can be used.

4.3 Rand - Uniformly distributed random numbers

Mersenne-Twister algorithm is proposed by Matsumoto and Nishimura [7]. The MT19937 has the period of 2^{19937}-1, the period of any pseudo random number generator is the sequence of random numbers it will produce before it repeats again. When compared to the number of neurons that are in the far field, the period is very high thus providing very high degree of randomness in the firing rates and position of noise units. The output of the algorithm is a sequence of word vectors that is uniformly distributed between 0 and 2^w-1 where w is the word length and then normalizing the vectors gives the required random number between 0 and 1.

Mersenne Twister algorithm in [7] comes with a c-program which is adapted and implemented for this thesis. Figure 4.4 shows the output of 10000 random numbers generated from Matlab library function and the Mersenne Twister algorithm implemented fit into a Cumulative Distribution Frequency. It can be seen from the plot that the outputs are matching very closely and the distribution is uniform across the interval 0 to 1.

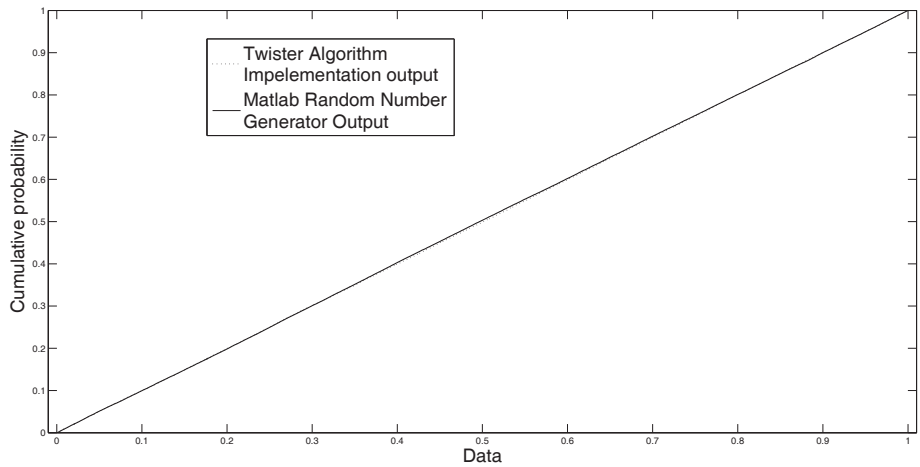

Figure 4.4: Random Number generators compared via CDF

Figures 4.5 present the result of the fixed point adaption of the twister algorithm compared to Matlab output at different word lengths. The curve fit show

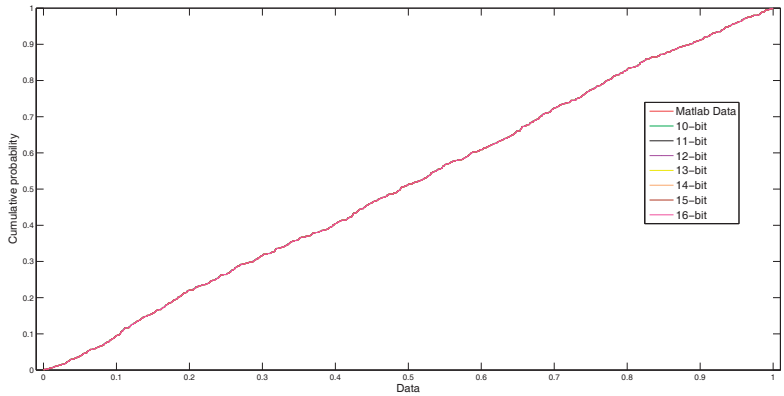

Figure 4.5: Fixed point implementation CDF for uniform random number generator and Matlab function

very minor difference for the different word lengths and the curves look uniform in the interval 0-1. Since the error is very minimal and the curves look identical, the random number generator is verified to be successfully adapted to fixed point.

4.4 Randn- Normal distribution random numbers

An ancient Mesopotamian rectangular tower is called Ziggurat. For generation of the normally distributed random numbers Matlab uses a version of the Ziggurat algorithm. Ziggurat algorithm was proposed by Marsaglia and Tsang [8] and derives its name from the appearance of the layered rectangles used in the algorithm.

The Ziggurat algorithm expresses the required density as a combination of simple densities and complicated residual density. The algorithm is a rejection sampling algorithm. The algorithm constructs a ziggurat around the required density function with a set of rules. The rules are that the number of rectangles must be a power of 2 (Can be 64, 128 and 256), areas of the rectangles must be constant and equal. These rules result in a ziggurat design with rectangles increasing width and decreasing height when parsed from top to bottom, this also results in the last rectangle tailing off to infinity. The algorithm works by selecting a uniform point (x,y) from one of the randomly chosen rectangles. If the random point is under the required distribution curve, then it is part of the distribution, otherwise it is rejected and next point is generated. A detailed explanation of ziggurat algorithm is out of scope of this thesis report, refer [8] for further information.

The Ziggurat algorithm is implemented with 128 sets is implemented in Matlab and the results are stored. Figure 4.6 shows the output of both Matlab library function and the Ziggurat implementation output for 10000 random numbers generated fit into a curve and the PDF is calculated. It can be seen from the plots that the distribution curves are following each other closely and also perfectly centered

around 0. The PDF shows the expected distribution pattern from the normally distributed random number generator. Since the results are satisfactory and the algorithm is adaptable to hardware implementation it is selected for the hardware implementation.

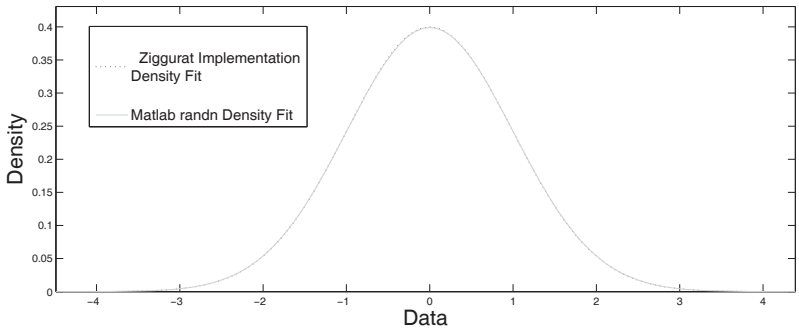

Figure 4.6: Normal distribution random number output comparision of ziggurat implementation and Matlab library function

4.5 gamrnd- Gamma distributed random numbers

The algorithm used for generating the gamma distributed random numbers by the Matlab is proposed by Marsaglia and Tsang in [9]. The algorithm draws a random numbers from the normal distribution and applies following steps to generate the gamma distribution.

- 1. Setup : item d=a-1/3 where a is the scale factor
- 2. Calculate c = $1/\sqrt{9d}$
- 3. Calculate v = (1+c*x)∧3, with x standard normal distributed random number. Repeat if v <= 0
- 4. Generate uniform random number U
- 5. U<1-0.0331*x∧4 return d*v
- 6. ln(U) < 0.5*x∧2+d-d*v+d*ln(v) return d*v (ln -natural logarithm)
- 7. Repeat step 3

[9] also provides an implementation of the algorithm in C code. The C code was adapted into a Matlab implementation and the results are calculated. Figures **??** and 4.7 shows the output of both Matlab library function and the gamma implementation output for 10000 random numbers for the same shape and scale factors generated fit into a curve and also the PDF is shown respectively. It can be seen from the plots that the distribution curves are following each other closely and

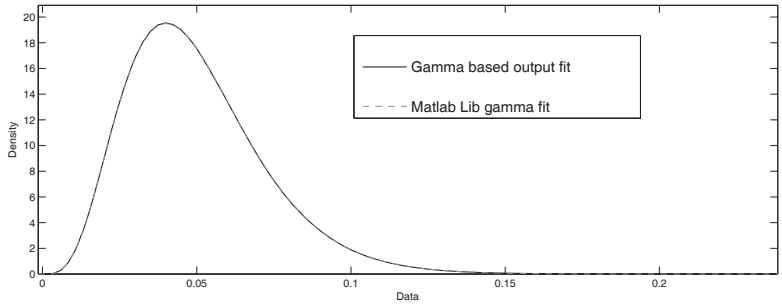

Figure 4.7: PDF of Gamma distributed Random number outputs from Matlab function and Ziggurat based implementation

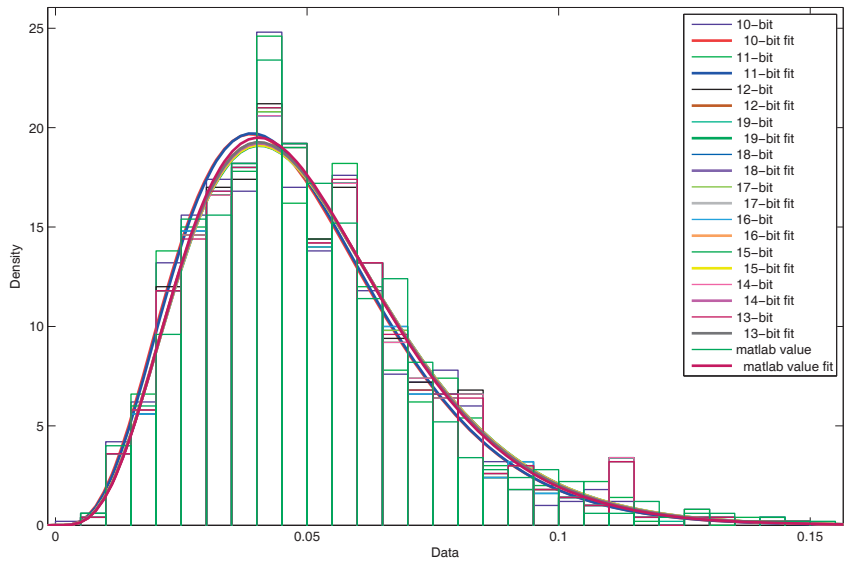

Figure 4.8: pdf overlapped with histogram of data of Matlab output and fixed point output for different word lengths

also scaled accordingly. Since the algorithm satisfies both ease of implementation on hardware and very good result it is selected for hardware implementation.

The gamma random number generator internally uses the normal random number generator for the calculation, hence the final output comparison of the gamma distribution output is a test for verification for both the gamma and normal distributed algorithms. Figure 4.8 show the curve fit output and the pdf output for

comparison between the Matlab output and the fixed point output for various word lengths of the gamma distributed random number. From the plots it is observed that the different word lengths have very little effect on the output generated but higher the word length the closer the curve is following the Matlab output. As the outputs are matching each other closely, the fixed point implementation of gamma distributed and normally distributed random number generators is successfully verified.

4.6 log, exp - cordic algorithm

The Cordic algorithm was initially developed for basic trigonometric operations. Cordic Algorithm was proposed by Jack.E.Volder in 1959 for use in the real-time digital computer for computing the trigonometric functions [10]. CORDIC is an acronym for Coordinate Rotation Digital Computer. The cordic algorithm works in two computing modes, rotation and vectoring.

Cordic algorithm does step by step rotations to calculate the angle or coordinates depending on the mode of operation. For the calculation of the logarithm, and exponent functions, the vectoring mode is used. In the vectoring mode the coordinate components are given and the magnitude and angle of original vector are computed. In 1971, J.S Walther proposed a unified algorithm for calculating linear, trigonometric and the hyperbolic functions [12]. The details of the cordic algorithm can be read more at [10], [15], [**?**].

$$x_{k+1} = x_k - md_k y_k 2^{-k} \tag{4.9}$$
$$y_{k+1} = y_k + d_k x_k 2^{-k} \tag{4.10}$$
$$z_k + 1 = z_k - d_k \sigma_k \tag{4.11}$$
$$m = 0, 1, -1 \tag{4.12}$$
$$d_k = sgn(z_k) for rotation mode \tag{4.13}$$
$$d_k = -sgn(y_k) for vectoring mode \tag{4.14}$$
$$\tag{4.15}$$

A modified version of the cordic algorithm was implemented for calculating the log, and exp functions and the plots 4.9, 4.11 show the comparison between the Matlab and the implemented cordic algorithms. As it can be observed from the plots the difference is negligible. Hence the algorithms implementation is successful.

Figure 4.10 shows the error between the output of logarithm of various word lengths and the Matlab library function. It can be seen that the error reduces as the word length increases. Also the error is in tolerable level for all the word lengths. The results show that the fixed point implementation of logarithm is generating output comparable to Matlab library function and the implementation is successfully verified.

Figure 4.12 shows the error between the output of exponent for various word lengths and the Matlab library function. It can be seen that the error reduces as the word length increases. Also the error is in tolerable level for all the word

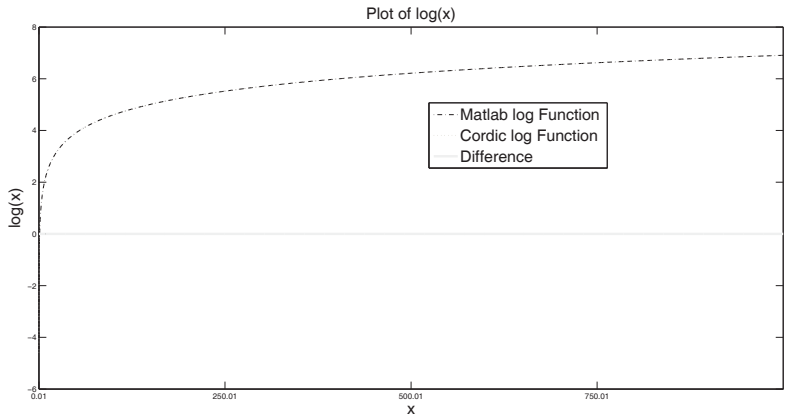

Figure 4.9: Comparison between cordic algorithm and Matlab implementation for Calculation of logarithm

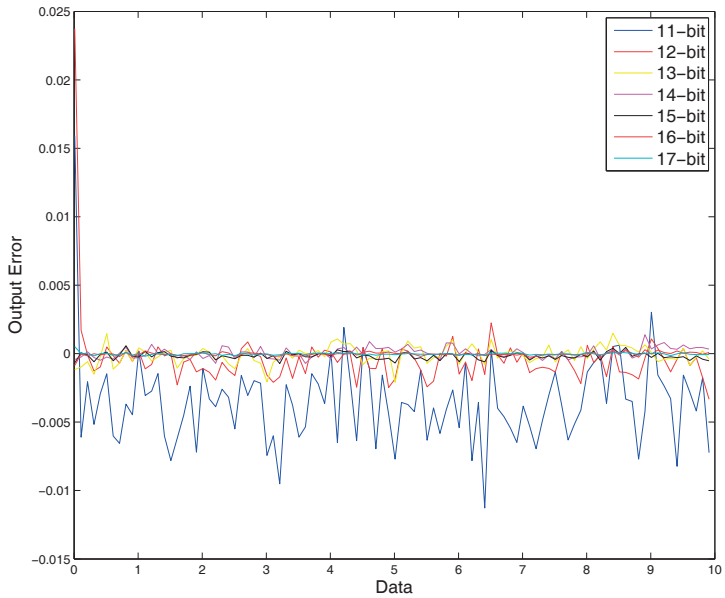

Figure 4.10: Error plot of difference between the logarithm Matlab output and the fixed point output for different word lengths

lengths. The output generated by the exponent function is closely matching with the actual output and low error. The results thus verify that the implementation

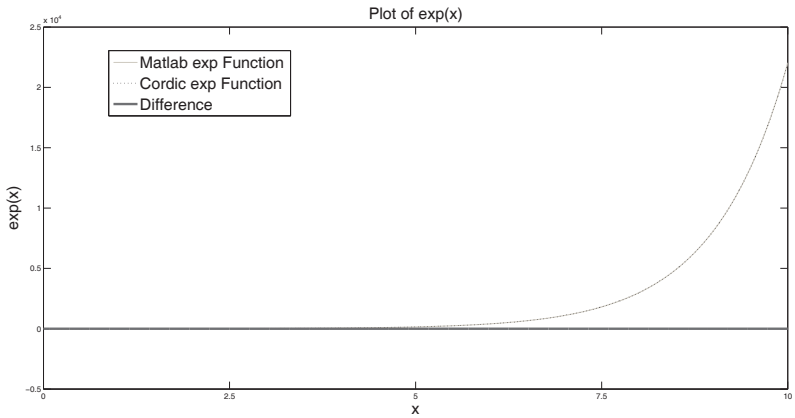

Figure 4.11: Comparison between cordic algorithm and Matlab implementation for Calculation of exponent

of the exponent in fixed point is successful.

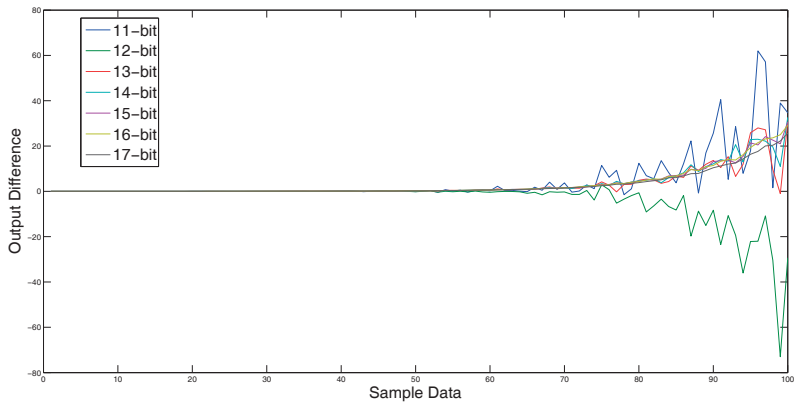

Figure 4.12: Error plot of difference between the exponent Matlab output and the fixed point implementation output for different word lengths

4.6.1 sqrt - square root

The square root method implemented is from the libfix_math. The results of the implementation are shown in plots 4.13 and 4.14.

Observing the plots it is understood that the higher word length generates

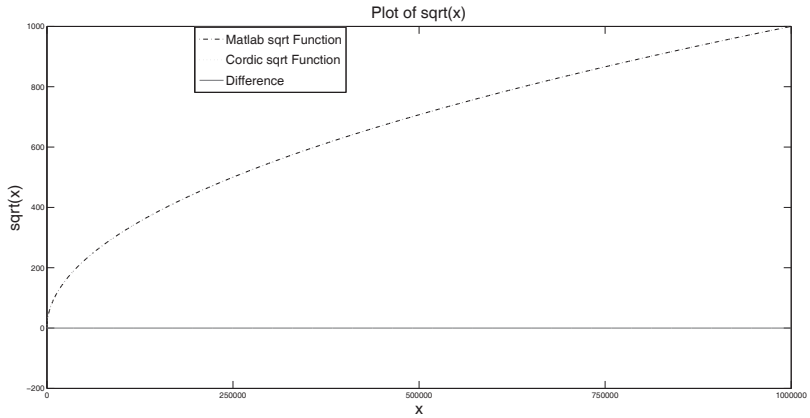

Figure 4.13: Comparison between cordic algorithm and Matlab implementation for Calculation of square root

the lesser output error. The error values are within the tolerable level making the fixed point implementation verification successful

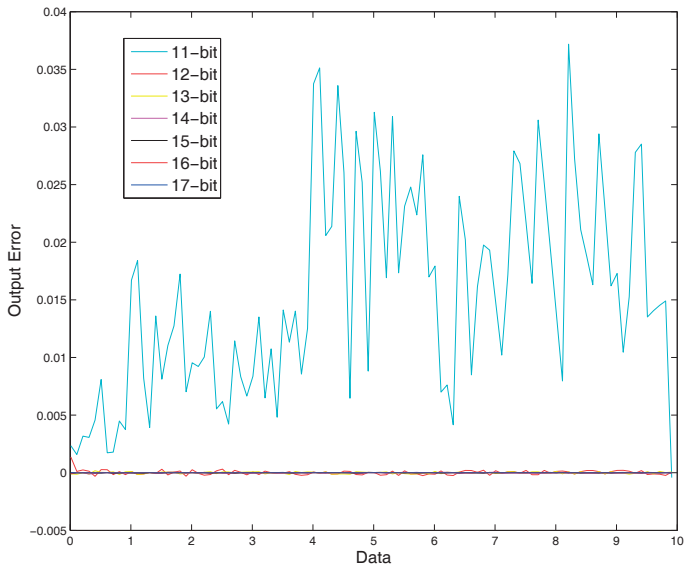

Figure 4.14: Error plot of difference between the square root Matlab output and the fixed point implementation output for different word lengths

4.7 division, randsample, randperm

The analysis of various division algorithms showed that it is effective to directly implement the algorithm in fixed point than implementing a floating point and converting it to fixed point. Hence a fixed point algorithm based on non restoring division mechanism is implemented.

The randsample implementation is derived from the code from Matlab library. Random sampling is achieved by generating the requested number of uniformly distributed random numbers. The random numbers are sorted along with the position indexes. The first N- Position indexes form the N-random samples.

The Division, randsample, and randperm algorithms are successfully converted to the fixed point and the outputs have been verified. The division algorithm used is borrowed from the libfixmath. Finally the sorting algorithm used is the insertion sort. The insertion sort algorithm is very flexible and easily implemented on a hardware hence it is used.

Chapter 5

Hardware Design of the Simulator

5.1 Hardware Implementation at Glance

Designing a system for implementation on an Application Specific Integrated Circuit (ASIC) or a Field-Programmable Gate Array (FPGA) requires a complete understanding of the system being implemented at both macro and micro levels. Figure 5.1 presents the hardware design overview of the simulator. The overview shows the design at a simple level with emphasis on the representation of the data flow to generate the output.

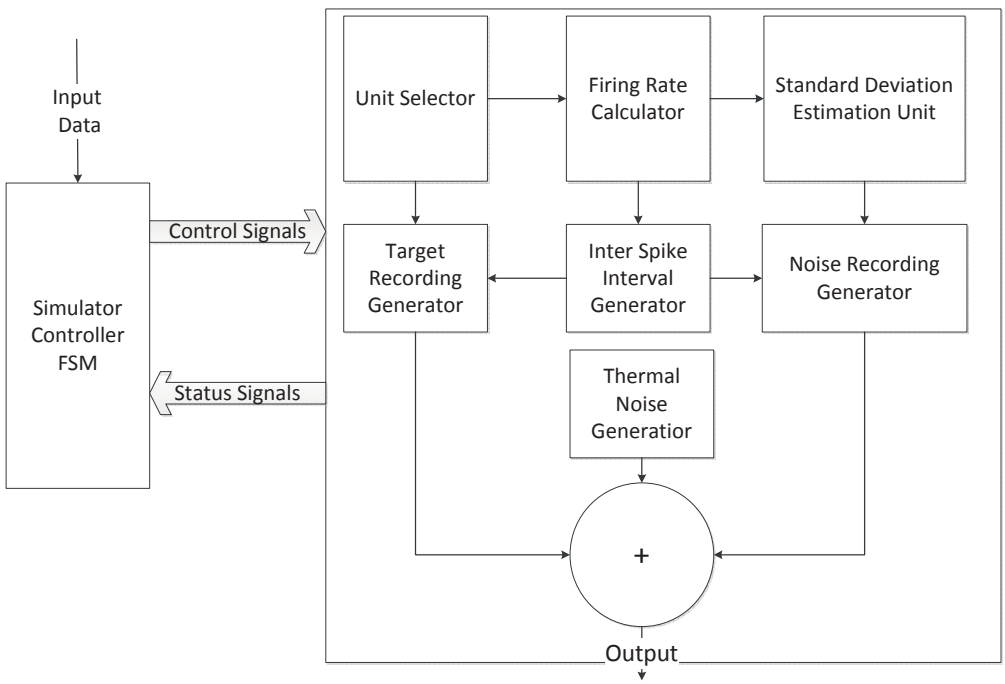

Figure 5.1: Simulator Overview

Implemented Hardware design of the simulator should generate of a recorded signal, and a ground truth signal that can be used as input for the subsequent stages in a BMI system. Feasibility of implementation, Ease of development and reasonably accurate output are the three basic design rules followed during the hardware design of the Neural Simulator.

The system consists of two parts, "Simulator Controller" and "Data Processing Unit". Simulator Controller issues the control signals that enables and disables various functional blocks in the data processing unit. Data Processing Unit Interprets the control signals to perform desired operations then informs the controller via status signals.

Simulator controller reads the input data, stores it in the memory and enables the data processing unit to perform data operations. Simulator controller also acts on the status signals from the data processing unit. Status of each operation is analyzed and next operation is selected via the control signals. It is implemented as a Finite State Machine (FSM).

In Data Processing unit, unit selector randomly selects target and noise spike waveforms used in the simulation. The times at which the target and noise units are placed in the final recording is generated by the Inter-Spike Interval generator, the inter-spike intervals generated are based on the firing rates of the target and noise units from the firing rate calculator. Standard deviation estimation unit calculates the estimated standard deviation for the noise recording. The ratio of the estimated standard deviation to the expected standard deviation is used as final normalization value for the noise recordings. Finally the Target spike samples, noise unit samples, and the thermal noise samples are added together to produce the simulator output.

Following sections will explain the detailed design of the hardware implementation of the simulator.

5.2 Top Level View of the Hardware Design

Figure 5.2 shows the detailed top level view of the simulator. To start the simulator, the input ports are provided with the simulation parameters and start signal resulting in the start of the Simulator controller. The simulator controller can be run in two modes, one where the recordings are continuously generated and the other is a timing mode with a maximum of 3600 seconds of recording generation. These modes can be selected via the input to the simulator. Simulator controller reads in the parameters for the simulation and stores them in the registers. Read-Only Memory contains 2000 spikes of 13-bit resolution. Once the input data is read, simulator controller enables the preprocessing unit. In the preprocessing unit, first the spikes id's for the target and noise units are selected randomly. After the spike id generation, firing rates are calculated for the target and the noise units. Calculating the amplitude for the noise units will complete the preprocessing stage. Selected spike id's, Amplitude for the noise units and the firing rates for target and noise units are stored in the Random Access Memory (RAM). The requirement on the RAM for the simulator implementation is a size of 16kB with 13-bit resolution. After the preprocessing stage the standard deviation estimation

is performed for the noise units and stored in register.

In the output generation stage interspike interval for the target units and noise units is continuously generated to facilitate both continuous and timing mode of the simulator. Physiological noise output generation block scales the noise spikes with the calculated amplitude and adds them to generate the physiological noise signal. The generated physiological noise is scaled to the required standard deviation. The generated physiological noise is added to the target and the thermal noise to generate the output. The output generation stage also generates the ground truth of the output recording.

Design of these blocks is explained in detail in current chapter and chapters 6 and 7.

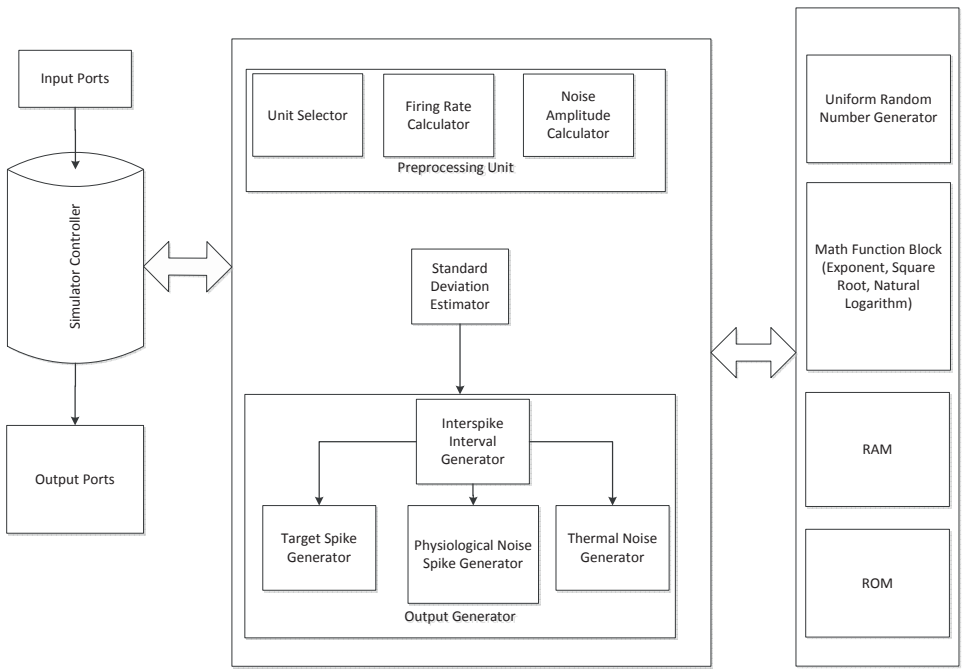

Figure 5.2: Simulator Overview

The major functional blocks are the simulator controller, preprocessing unit, standard deviation estimator, Output Generator, Uniform Random Number Generator, Math functional blocks, memory blocks RAM and ROM, finally input and output ports.

To reduce the implementation area, the design has only one RAM, one ROM and a single instance of the uniform random number generator and the math functional blocks. This implies that there are multiplexers between the Data processing unit and these units. Every time the thesis mentions any of these

units, it is assumed all of them refer to the same instance.

Simulator controller is responsible for the generation of the control signals and reading of the input data. Data is transmitted in and out of the simulator via the Input and Output Ports (I/O Ports). preprocessing unit is responsible for selection of the spike id's for the target and noise units, firing rate calculation and Amplitude calculation for the noise units. Standard deviation estimation for the noise units is done in the standard deviation estimator. Finally the interspike intervals, output recording and ground truth are generated in the output generator. Together the preprocessing unit, standard deviation estimator, and output generator forms the data processing unit, explained in detail in chapter 5. Random Number Generators: Simulator requires multiple implementations of random numbers with uniform, normal and gamma distributions. Since the hardware implementation of each of these more than once is expensive, they are implemented as a block and multiplexed among various data processing units. Chapter 6 explains the hardware design of these random number generators Math functional blocks contains implementation of mathematical functions of Natural logarithm, square root, exponent, and divider used in data processing. Design of the mathematical functional blocks is implemented but out of scope of this report.

5.3 Simulator Controller

Primary task of simulator controller is to enable and disable different functional blocks in the data processing unit to facilitate the flow of data among them. It is implemented as a finite state machine, Figure 5.3 shows a simplified version of the state machine. Transition from each state to next state is facilitated by the status signals, while the transition triggers the control signals to the data processing unit. A stop signal to the simulator at any time transits initiates the transition to the idle states from all the other states, though not shown in the state machine for a simpler view.

5.4 I/O Ports

I/O ports are the mechanism to feed the input data, and collect the output from the simulator. Since I/O ports translate to I/O pads on the hardware and the I/O pads take huge area on the FPGA, it is good to keep the I/O ports as minimum as possible. It should be noted that the simulation parameters to the hardware design of the simulator is different from the Matlab script. Inputs spike library name and save_file name are dropped since they are not relevant, inputs like spike id selection of the target is also dropped since it increases the complexity of the implementation for little flexibility, but in cases where the target id need to be specified they can be specified via the ROM.

Hardware Design of Neural Simulator has 15 1-bit inputs, eight 1-bit outputs and two 13-bit outputs. Functionality of each pin is explained in detail in the tables 5.1 and 5.2

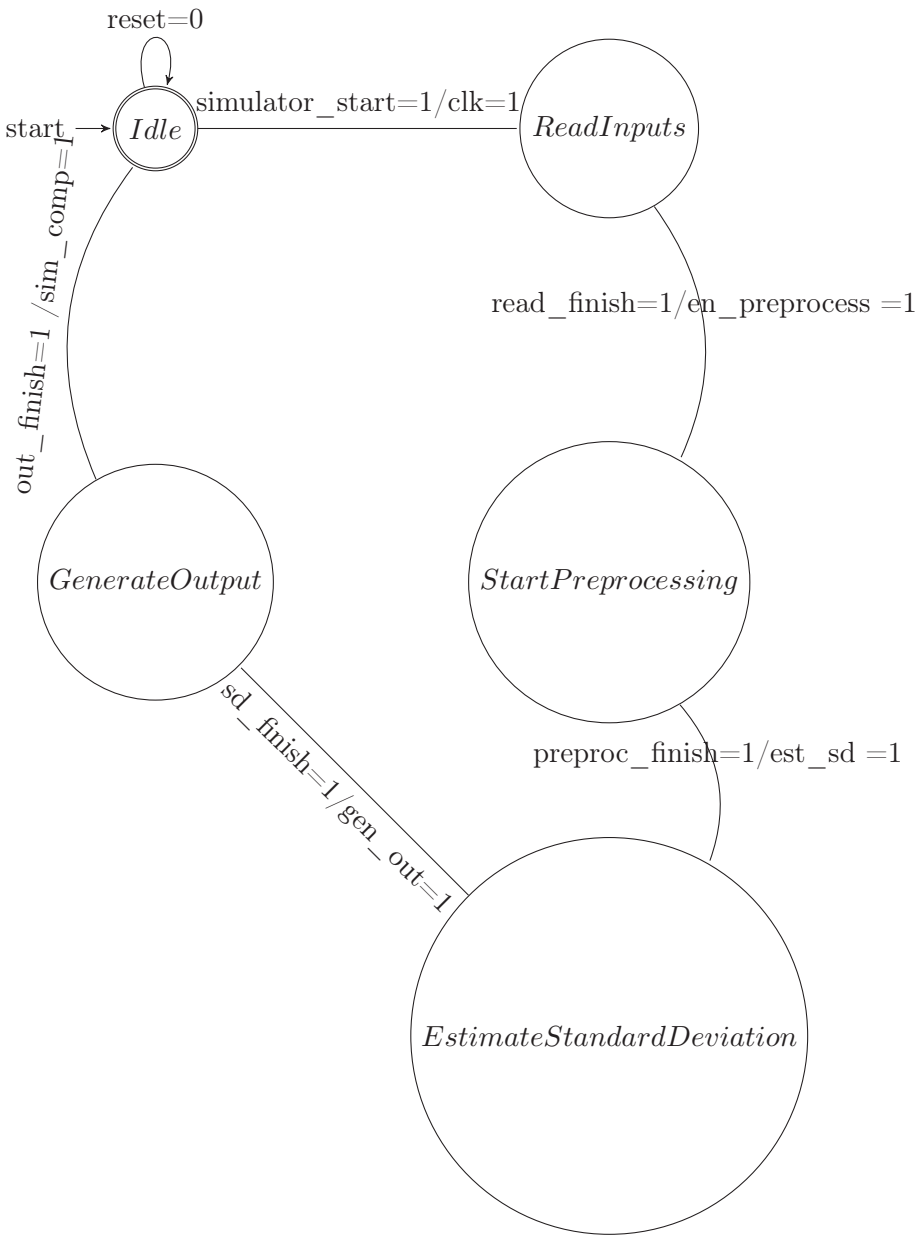

Figure 5.3: Simulator Controller Finite State Machine

clk	1-bit	Simulator clock
rst	1-bit	Simulator reset
start	1-bit	A high on this pin drives Simulator Controller from idle to read state
stop	1-bit	A high on this pin drives Simulator Controller to stop state
num_unit	8-bit serial	Target unit number, maximum 140
duration	12-bit serial	Duration of recording in seconds, maximum 3600. 0 to run until a high stop signal
samplerate	4-bit serial	Sample rate of the output recording in kHz, maximum value is 10kHz
phy_noise_sel	1-bit	1-Enable/0-disable physiological noise generation
th_noise_sel	1-bit	1-Enable/0-disable physiological noise generation
t_noise_param	13-bit serial	Calculated square root of kTBR
isi_gam_def	13-bit serial	Isi distribution shape parameter(k)
p_noise_def	13-bit serial	Standard deviation of the physiological background noise
k_amp_def	13-bit serial	Amplitude decay factor in the far field
meanu	13-bit serial	Mean firing rate for the target units
meann	13-bit serial	Minimum and the maximum value of the noise unit firing rate. The input order is minimum value followed by maximum value
target_dist	13-bit serial	Minimum and maximum value of the distance of target units. The input order is minimum value followed by maximum value

Table 5.1: Input ports of the simulator

scal_amp_start	1-bit	flag which is high when the target units amplitude scaling factor values for the target are available on the sim_out output. Each of the 13-bit values are mapped to the 13-bit target index values one to one where first 13-bit output amplitude scaling factor corresponds to the first 13-bit target index output.
tar_index_start	1-bit	which is high when the index values of the target units are available on the sim_out output. The 13-bit output values of target index is mapped to the amplitude scaling factor values.
scal_target_start	1-bit	which is high when the amplitude scaled target spikes are available on the sim_out.
spike_times	13-bit serial	data output which outputs the inter-spike intervals of the target units corresponding to the target unit indexes.
sim_out	13-bit	which is multiplexed to send out target unit amplitude scaling factor, target index values, amplitude scaled spike values and finally the actual simulator output. The output is at the given sampling rate.
sim_out_start	1-bit	which is high when the simulator output is available on the sim_out.
target_10k	13-bit serial	output of the generated target recording values at 10kHz sample rate.
noise_10k	13-bit serial	output of the generated physiological background noise values at 10kHz sample rate.
thermal_10k	13-bit serial	output of the generated thermal noise values at 10kHz sample rate.

Table 5.2: Output ports of the simulator

Chapter 6
Data Processing Unit

Data processing unit processes the spike data and the simulation parameters to generate the recording output and the ground truth. Preprocessing unit, standard deviation estimator and the output generator form constitute the data processing unit. Design of the individual hardware blocks is explained in the following sections.

6.1 Preprocessing unit

6.1.1 Unit Selector

First part of preprocessing unit is the Target and Noise unit selector. This unit generates the index for the spike waveforms used for target and noise units. This is the equivalent for the Matlab function rand_sample. The output indexes generated by this unit is stored in RAM for use by the other units in the data processing unit.

Figure 6.1 shows the block diagram of the unit selector. The process of unit selection involves reading of the uniformly distributed random number, sorting the random numbers and finally storing the id's in the RAM. The sorting logic is implemented as a finite state machine.

Figure 6.2 shows the flow chart of the unit selection algorithm followed for the implementation. The unit first generates 2000 uniform random numbers and stores them along with the sequence number of generation. These generated random numbers are stored in the RAM, thus each stored value in RAM has both the random number and the sequence number. The random numbers are sorted in ascending order placing the corresponding sequence number in random order. The first 'M' sequence numbers form the target unit id's where M is the number of target units. Rest 960 sequence numbers form the noise unit id's.

6.2 Firing Rate Calculator

Firing rate generator calculates the firing rates for the target and noise units. The target firing rate calculation is simple as it involves the generation of uniform random number and multiplying it with mean target firing rate. This process is repeated for all the target units. The reciprocal of the calculated firing rates which

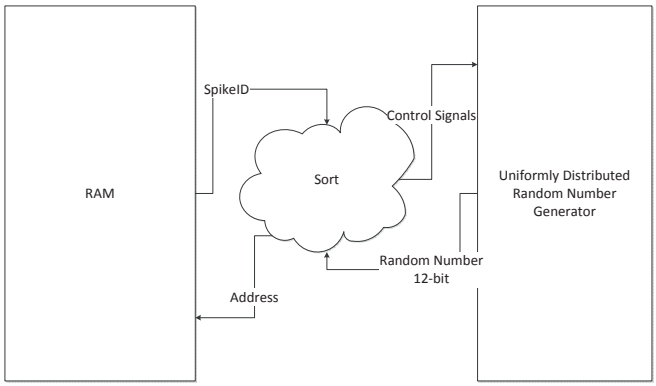

Figure 6.1: Target and Noise spike id Selector Block Diagram

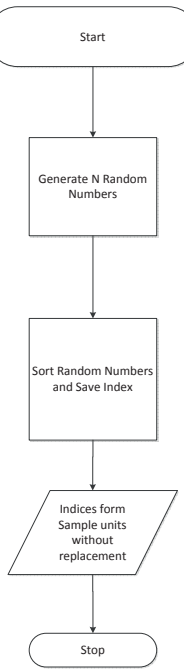

Figure 6.2: Flow chart of Target and Noise Unit Selection Process

is the mean interspike interval is stored in the RAM. Firing rate calculation for noise unit involves the minimum and maximum firing rate values that are input to the simulator. The process of generation of the firing rates for the noise units involve generation of a uniform random number and scaling to fall between the minimum and maximum firing rates. Reciprocal of firing rate for each target unit is calculated, the reciprocal is the mean inter spike interval for that noise unit. Each of the mean ISI's are stored back in the RAM for all the noise units. Figure 6.3 shows the data path for the firing rate generator.

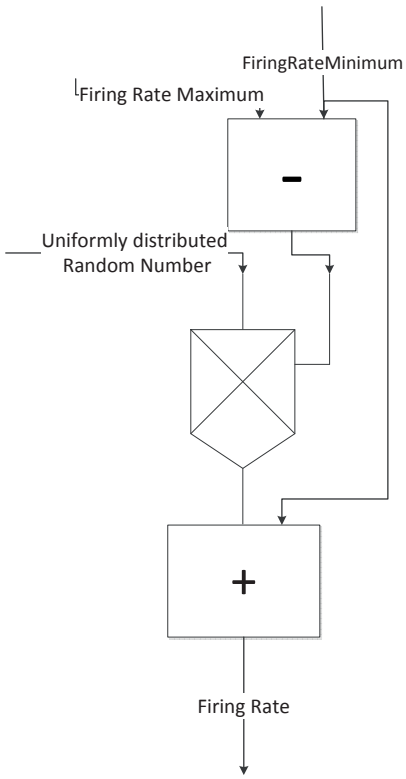

Figure 6.3: Noise Firing rate Generator Data Path

6.3 Noise Amplitude Calculator

Noise Amplitude calculator is responsible for generating the scaling factors for the Amplitude of noise spike wave forms. The amplitude is calculated according the equation 6.3. Radial distance of the noise unit is used for calculating the amplitude scaling factor, radial distance is dependent on the position of the noise unit in the far field. Also a unit is considered to be in far field if its distance is

greater than 50 Micrometers and less than 140 micrometers. Distance calculation process generates random position of the noise unit, calculates the radial distance to the unit and verifies if it is in the far field. If the unit is in the far field, the unit is considered as noise unit else the process is repeated. Flow chart for amplitude calculation is shown in the figure 6.4. Once the radial distance is calculated, the amplitude scaling factor is calculated according to the following equation.

$$A = \frac{1}{(Kr+1)^n}$$

n is between 2 and 3
K is the scaling factor that specifies the rate of decay
r is the radial distance of the noise unit from the electrode

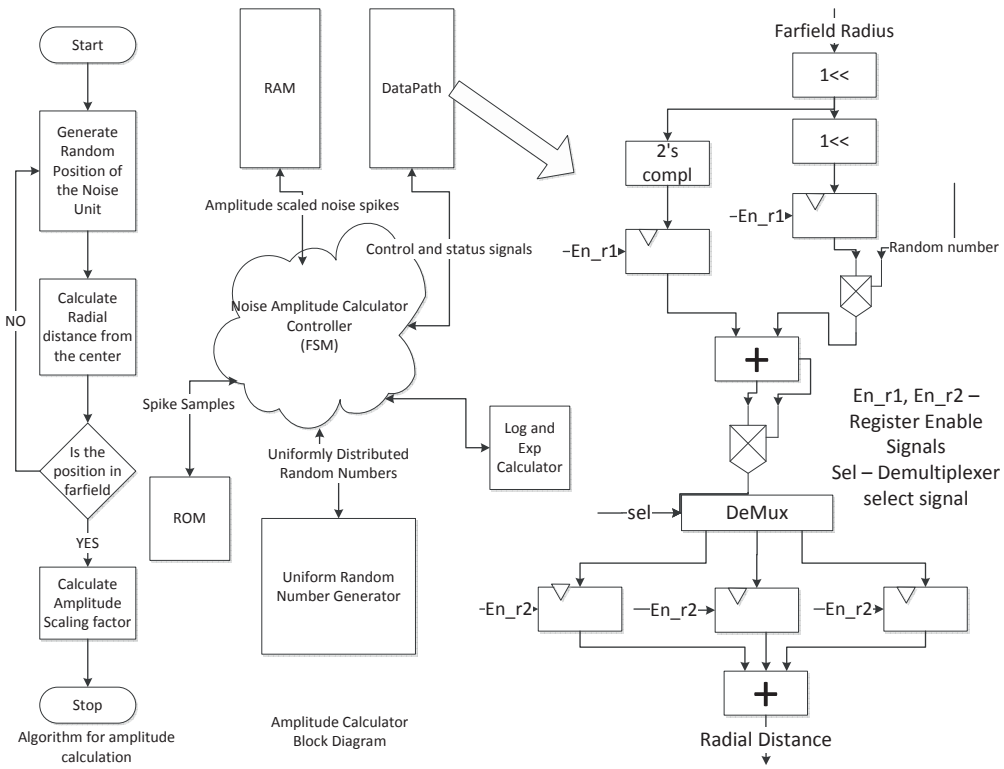

Figure 6.4: Noise Amplitude Calculator

Figure 6.4 shows the block diagram for the noise unit amplitude calculator. The calculator consists of a controller implemented in FSM, and a datapath. The controller is responsible for enabling and disabling of the datapath, while the datapath performs simple arithmetic operations to generate the output. The controller also reads in the random numbers, feeds them to the datapath. After generation of

each scaling factor, the controller reads the corresponding noise unit from ROM, scales the amplitude and stores the scaled samples in the RAM. This process is repeated for all the noise units.

Figure 6.4 shows a section the data path diagram for the noise amplitude scaling factor calculator. The section shown here is used to calculate the radial distance for the random position of the noise unit. The position in 3-dimensional plane requires (x,y,z) coordinates which are generated by the random number. Each coordinate generated is processed and stored in the registers and once all the co-ordinates are generated, processed and stored in Registers. The registers are enabled by controller with control signal En_r2 and the output from registers is fed to the adder to generate the radial distance. The radial distance thus generated is used by the rest of the data path to generate the amplitude.

To generate the amplitude, a 2.9th root of the Kr+1 need to be calculated. The calculation is performed by calculating a natural logarithm of the same and then calculating the exponent of the logarithm result. The natural logarithm calculation and the exponent calculation is performed by the Exp and Log calculator.

6.4 Standard Deviation Estimator

Matlab script scales the generated physiological noise output to match the input standard deviation. This requires the calculation of the standard deviation of the standard deviation of the generated physiological noise. This is not feasible in the hardware since it would require huge memory to store the output and also can't be run in forever loop. To avoid this a standard deviation estimate is calculated as explained in section 4.2. The final equation for estimation is given below. In the equation L represents samples per spike and f_n is the mean firing rate of the nth neuron, A_n Amplitude of the n-th spike.

$$variance = E(V_n^2) - E(V_n) = (f_n/f_s) * L * A_n^2 * E(S_{0n}^2) - (fn/fs) * L * (E(S_{0n}))^2 \tag{6.1}$$

$$stddev = \sqrt{(sum_n(variance))} \tag{6.2}$$

The data required to compute such as the mean squared amplitude, mean amplitude is read from the ROM and the ISI mean is read from the RAM. The communication to the mathematical functions is performed via the math function ports. Also the ports are 15-bit for the math functions since the standard deviation unit needs a different word length which is explained in the chapter 3. Also all the internal calculations involve 15-bit arithmetic and data.

6.5 shows the data path for the generation of the standard deviation. The data path only calculates the variance and the standard deviation is calculated applying the square root to it. The square root calculation is done by the square root function implemented in the Math function block.

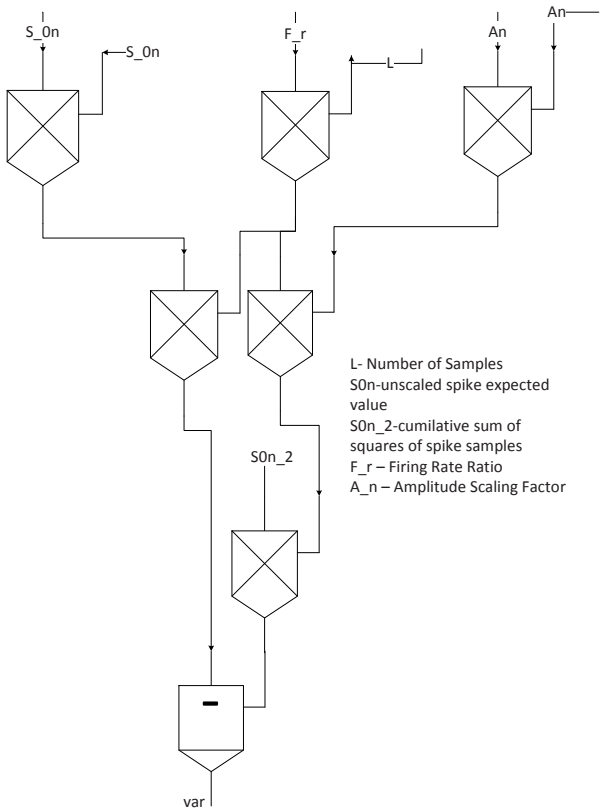

Figure 6.5: Standard Deviation Calculator Data Path Diagram

6.5 Output Generator

Output generator is responsible for generating the target recording, and the ground truth. Figure 6.6 shows the block diagram of the output generator. The output generation is a two part process, the first part is the management of the Interspike intervals and the second part is the management of output recording generation.

The Management of interspike intervals involves continuous generation of intervals for the target and noise units. The interspike interval generation process is handled by the Interspike Interval controller. At initialization the interspike controller generates intervals for all the noise and target units and stores them in the Interspike Interval Buffer. After the initialization phase, further requests for interspike interval generation are handled by reading interval request buffer. The Interspike interval buffer holds the interspike interval for all the target and noise units. Interval request buffer holds all the requests for the generation of interspike intervals. The interspike interval generation is achieved by applying the scale factor and the mean interspike interval to the interspike interval generator. The

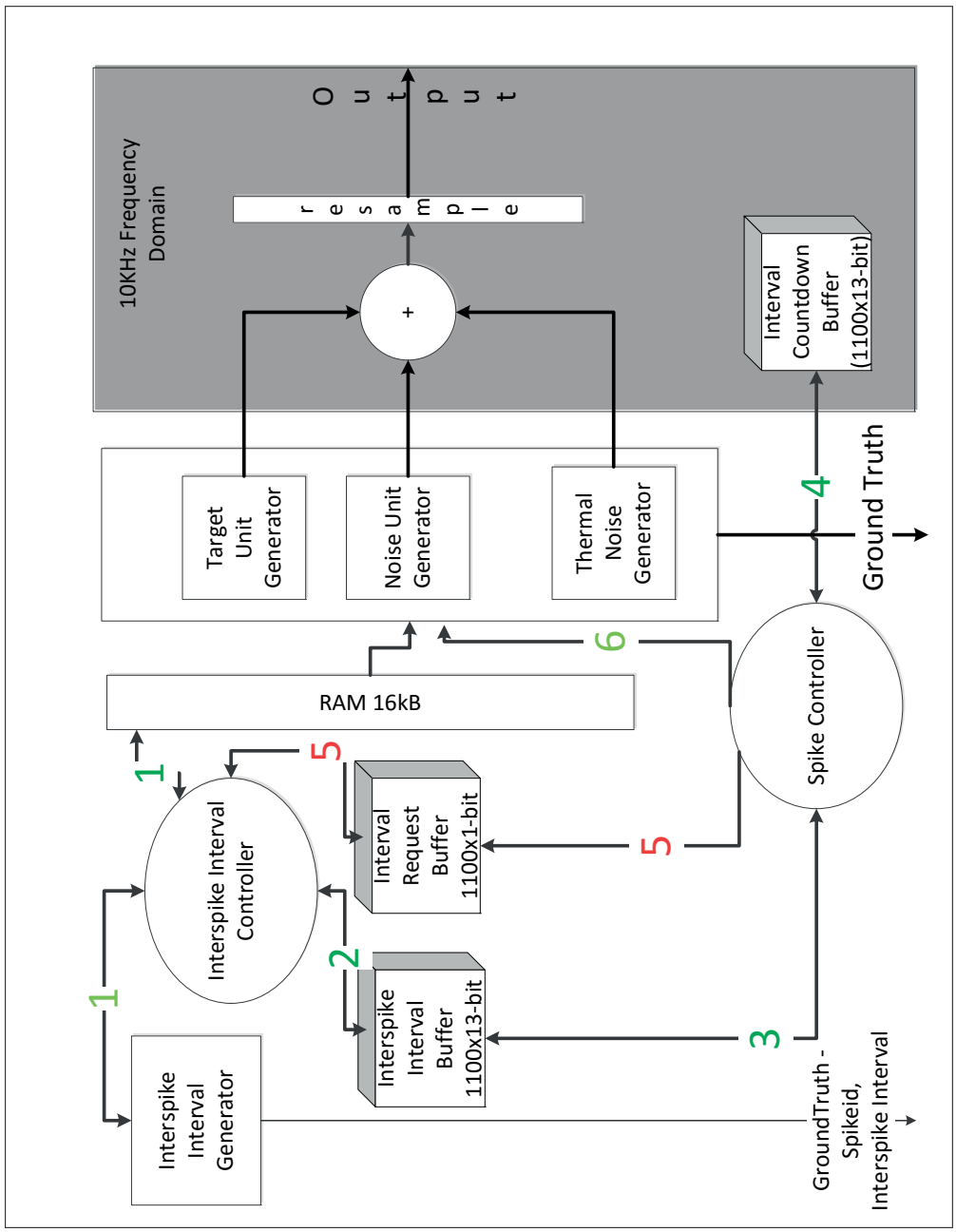

Figure 6.6: Output Generator Data path for target, noise and thermal noise generation along with final output

interspike interval generator consists of a gamma distributed random number generator and a output handler. Output handler sends out the generated interspike

interval and corresponding spike id to the output as a ground truth values.

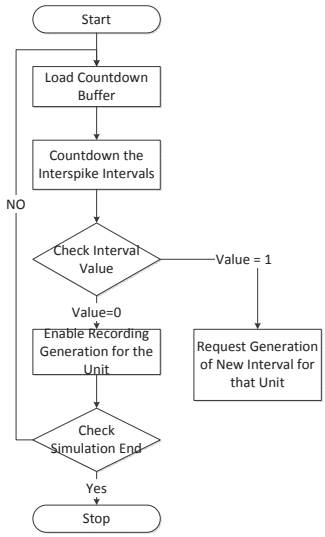

Figure 6.7: Output Generation Process Flow Chart

Spike Controller handles management of the output recording generation and the corresponding ground truth generation. Output generation process is shown in figure 6.7. Interspike intervals are loaded into the Countdown buffer. The Interspike intervals in the countdown buffer are counted down for each clock cycle. When the count for the interval reaches a value of 1 a request for generation of interspike interval for that unit is placed in the interval request buffer. When the count reaches zero recording generation for particular target/noise unit is enabled. The process continues until the end of the simulation. Target Recording Generator reads the samples from the RAM for the target units that are firing at that instance, adds them and loads it in the output register. Noise Recording Generator also reads the samples from the RAM for the Noise units that are firing at that instance, adds them, scales them to the standard deviation and loads them into the output register. Each of the output samples from the target, noise and thermal noise generators is stored in corresponding output registers clocked at a speed of 10KHz. The clocking of the output registers at 10KHz. The output registers output is directly fed to the adder that generates the output recording. The output recording is then downsampled to the required output sampling rate. The output from the registers at 10KHz is also fed to the corresponding ground truth output ports.

6.5.1 Thermal Noise Generator

Thermal Noise Generation unit is used to generate the thermal noise as part of the output generator. The thermal noise is generated by multiplying a normal

distributed random number and with the input rms value of the thermal noise to generate the final thermal noise output. 6.8 shows the thermal noise generator block diagram. The process is to read the normal random number generated, multiply with the rms value and send the data out. The whole process controlled by the controller implemented in state machine.

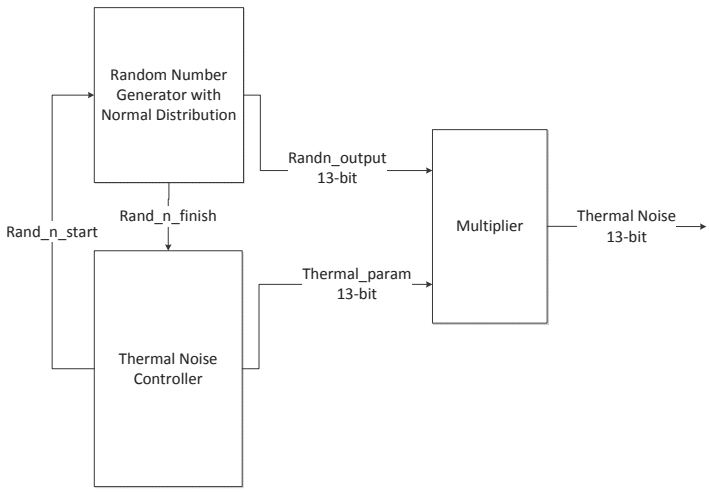

Figure 6.8: Thermal Noise Generator Block Diagram

Chapter 7

Random Number Generators

This chapter explains in detail the design of the Random Number generators. There are three different distributions of random number generators used in the design of Real Time Neural Simulator. The uniformly distributed random number generator is used in the noise amplitude scaling factor generation, random sampling and firing rate calculation. The normally distributed random number generator is used in the generation of thermal noise. Finally the Gamma distributed random number generator is used in the inter-spike interval generation unit.

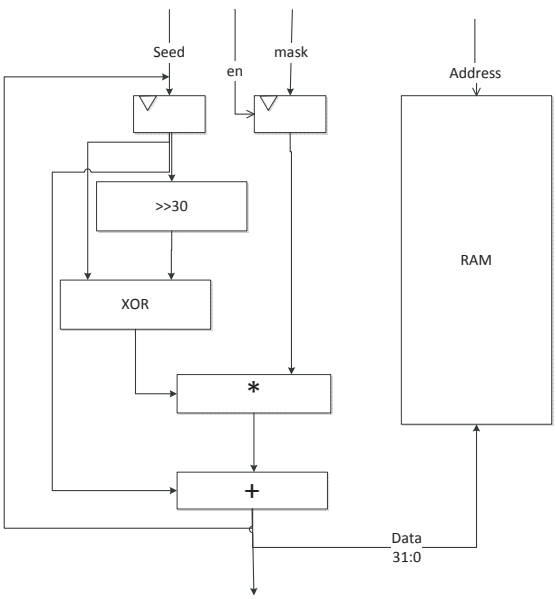

Figure 7.1: Random Number Generator Initialization Data Path Diagram

7.1 Uniformly Distributed Random Number Generator - Mersanne Twister Algorithm

The design of the Simulator uses the Mersenne Twister algorithm for the uniform random number generation. The MT algorithm hardware design contains a controller and two data paths controlled by the controller. The controller is implemented via state machine and its task involves enabling the data paths and controlling the data flow between them. The MT random number generator has an initialization phase during which the input seed is processed to generate the runtime data used in random number generation. After the initialization phase the generator can be used to generate any number of random numbers distributed uniformly. Also moving the generator between initialization phase and generation phase is handled by the controller.

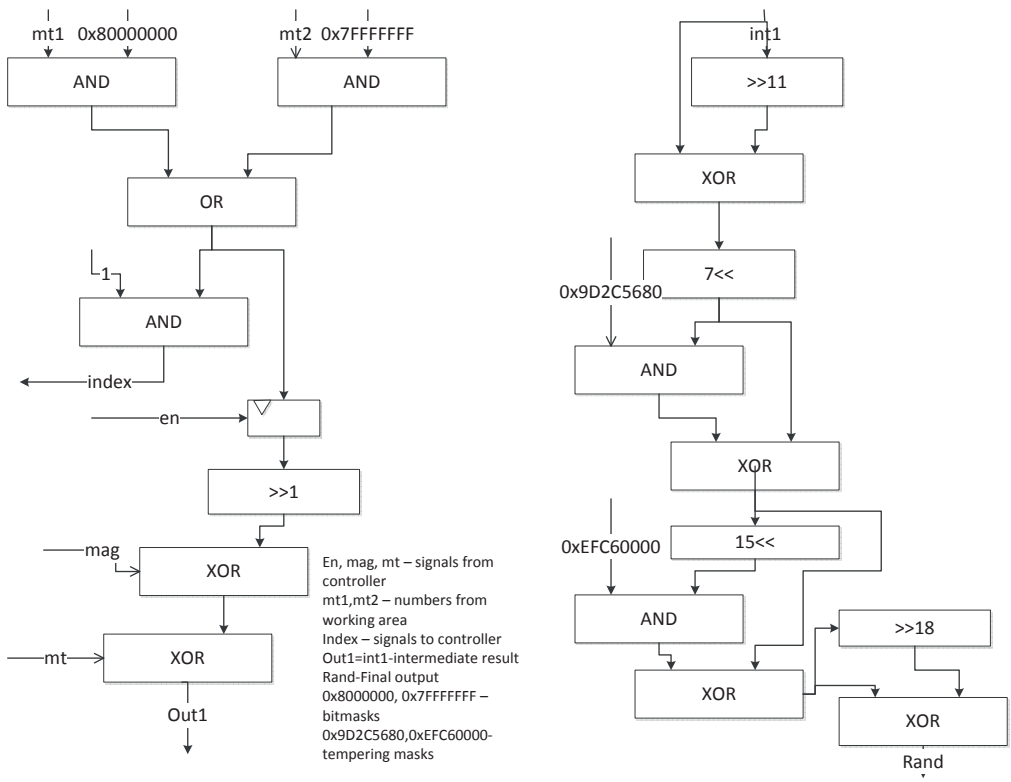

Figure 7.2: Random Number Generator Main Data Path Diagram

Figure 7.1 shows the data path diagram for the initialization of the random number generator. The seed and input values are provided by the controller and once the calculation is finished the initialized data is stored in the RAM and it continues for 624 times to generate the initial working area for the algorithm to

start generating random numbers. The 624 number working area is because of the 623-distributional property of the algorithm.

Figure 7.2 shows the data flow diagram for the main generator, it can be observed that the generator is split into two parts. The first part is started with the controller providing the mt1 and mt2 values read from RAM and then input to the first part of the data flow diagram. The second part of the diagram is the data path for generating the 32-bit random number. To generate the final 13-bit random number, the 32-bit random number generated first is shift by 20 bits and sent as output.

7.2 Normally Distributed Random Number Generator

The generation of random numbers with normal distribution is implemented by the Ziggurat algorithm. The ziggurat hardware design contains a controller and three data paths enabled and controlled by the controller. The controller also handles the data flow between the data paths. Each of the data paths has a specific purpose explained below. The ziggurat algorithm in essence generates a uniformly distributed random number that is processed to determine if it is inside or outside the required distribution curve. The ziggurat algorithm uses two sets of equations to select the rectangles to generate the random number from. One set of equations for selection from the top rectangles and the second set for the trail rectangle of the ziggurat.

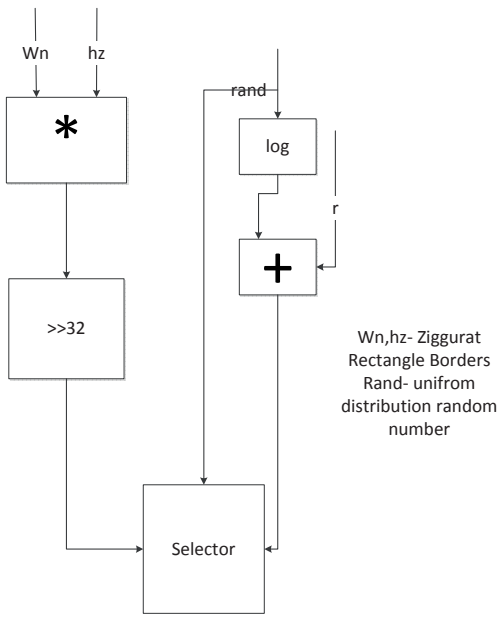

Figure 7.3: Normally Distributed Random Number Data Path

The design of the ziggurat algorithm must handle the random selection of either top rectangles or the trail of the ziggurat. Figure 7.3 shows the data path diagram for both the cases. Left part of the datapath generates the output if one of top rectangles of ziggurat is selected. Right Datapath generates the random number form the trail. The selector determines if the random number has to be taken from the top rectangles or the trail, it also validates if the generated value is inside the distribution. If the selected value is inside the ziggurat, it is sent to the output.

7.3 Gamma Distributed Random Number Generator

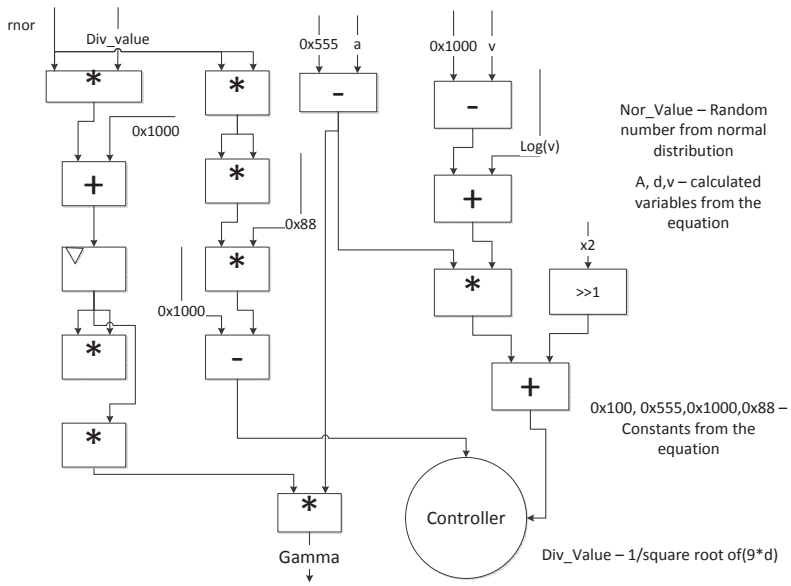

Figure 7.4: Gamma Distributed Random Number Data Path

The simulator uses gamma distributed random numbers in the calculation of inter-spike intervals. Gamma distributed random number generation is based on the algorithm proposed in [9]. The algorithm is simple for implementation and requires the normal distribution random number generator to provide the random number for calculation of gamma distribution. The generator consists of data path that process the data to generate the random number, managed by the controller. The controller is implemented in state machine and is in a specific state at any given point of processing. Figure 7.4 shows the data path and the controller. The datapath design is the fixed point implementation of the gamma algorithm discussed in chapter 4.

Chapter 8

Results and Conclusions

The Original Matlab code is successfully adapted to include the user defined algorithms mathematical functions, and random number generators. Another major step in adapting the code is the conversion to fixed point implementation. Initially the code is converted into fixed point implementation with adaptable word length. The adaptable word length helps in understanding the effects of finite word length on the simulator output for different word lengths. In order to verify the finite word length effects there is a need for some measurable quantity of the signal for benchmarking. The bench marking is done by measuring the total energy of the resulting signal as a metric. Assuming an error tolerance of 10 percent is allowed between the energies of the signal of the original Simulation output and the Fixed Point Converted code. The simulator uses signed representation of word length and output word length represented in Q form is Q6.6. This accounts for a maximum of 40 target units with maximum amplitude, translating to 6 decimal, 6 fraction and 1 sign bit.

During the adaptation to fixed point system it is found that the standard deviation calculation requires a special attention for calculation of required word length and is discussed in detail in the next section. Section 8.2 will describe the selection of final word length. Final section concludes with the analysis of results.

8.1 Standard Deviation Calculation Unit Word Length

To select the proper word length the fixed point code was run in parallel with the floating point code for exactly same simulation parameters, this requires pre-generation of random numbers wherever required, also all the library functions were replaced by fixed-point versions. To calculate the word length required the simulator is run by varying the word length between 11-bits to 17-bits and the standard deviation calculated value is stored. Once all the word lengths output is calculated then the percent deviation from the floating point value is calculated and plot in a graph. Figure 8.1 shows that results and it can be seen that the word length of 16-bits is required to keep the error tolerance at 5 percent. The error tolerance for this hardware unit is chosen specially to be low since a small percent change in here causes a huge percentage deviation in the final energy difference since the error is spread across all the noise signal values.

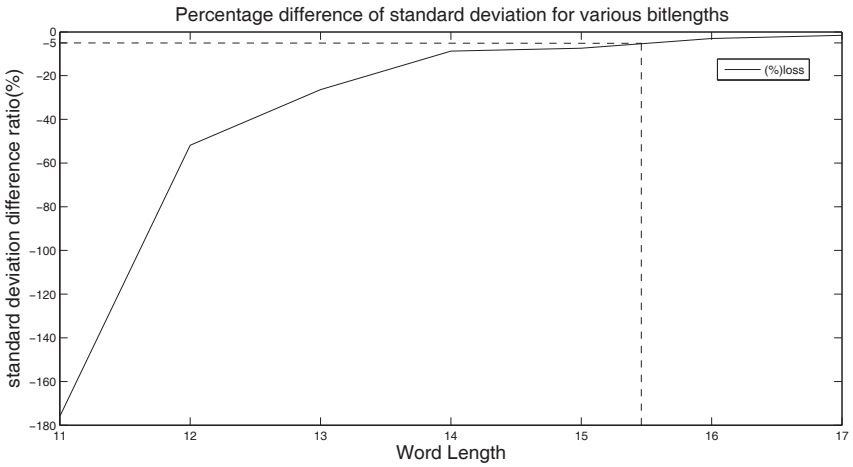

Figure 8.1: Percentage Difference of the Standard Deviation for varying Word Length

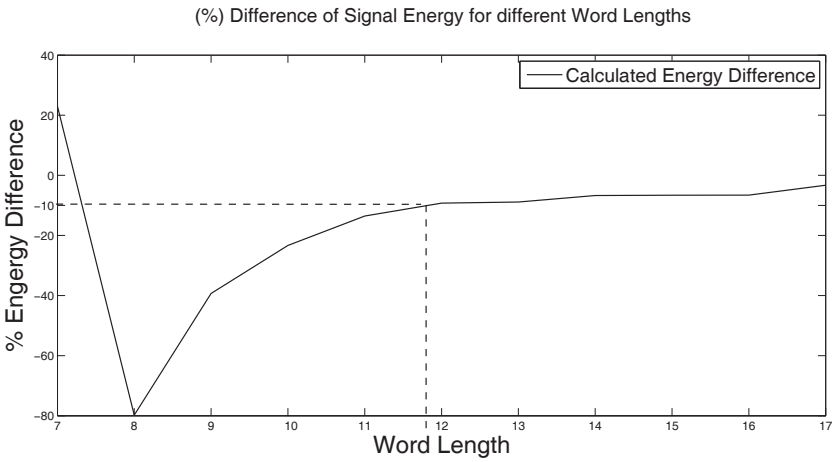

Figure 8.2: Calculated Energy Difference in percent between Original and Fixed Point Converted Simulator Outputs

8.2 Word length for Overall Simulator

Using the 15-bit word length for the standard deviation calculation module the rest of the simulation is run for various word lengths varying from 6-bit to 16-bit and the energy difference of the floating point signal and fixed point signal is calculated and plot in a graph as shown in Figure 8.2. It can be seen from the figure that

a 12-bit value for word length provides the agreed 10 percent tolerance of percent of calculated signal energy of the outputs of original and fixed point adapted simulator. One extra bit is added to compensate for parameter variations and other propagated finite word length effects making the final word length selection as 13-bits.

8.3 Simulation Output

The final simulation was run with 13-bit word length for all the modules and 16-bit for standard deviation calculation and the final result is plotted in figure 8.3, the graph contains portion of the output signal with target spike overlapped. It can be seen that the noise has contributed to some increment in the final combined signal when compared to target only output.

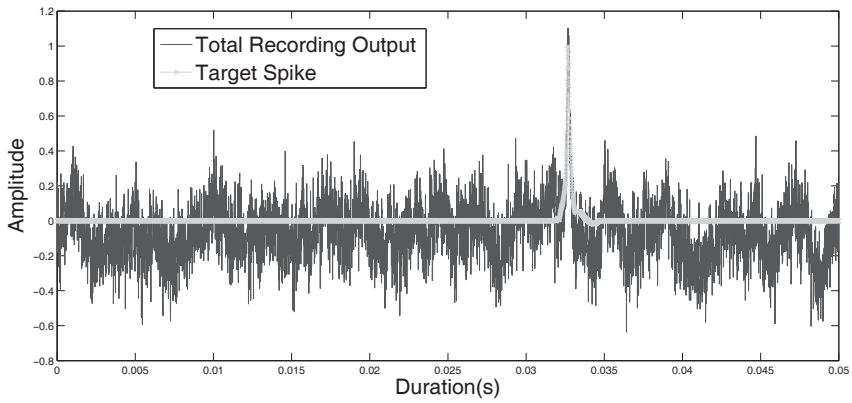

Figure 8.3: Fixed Point Converted Simulator Output with Target spike signal Overlapping

Figure 8.4 shows the comparison between the original simulator output and the fixed point adapted simulator output. It can be seen that the signals are following each other closely but not exactly same because of the randomness in the generation of the scaling factors and other parameters.

Figure 8.5 shows the comparison of the power spectral density between the original simulator and fixed point adapted simulator output. Considering that the calculation of inter spike intervals, and random scaling factors for the noise signal the two outputs show a good match. Figure 8.6 shows the autocorrelation of the original and fixed point simulator outputs and they show very strong similarities.

8.4 Conclusion

The result of the fixed point implementation simulator is very satisfactory and is a major milestone for the hardware design and implementation. Based on the fixed

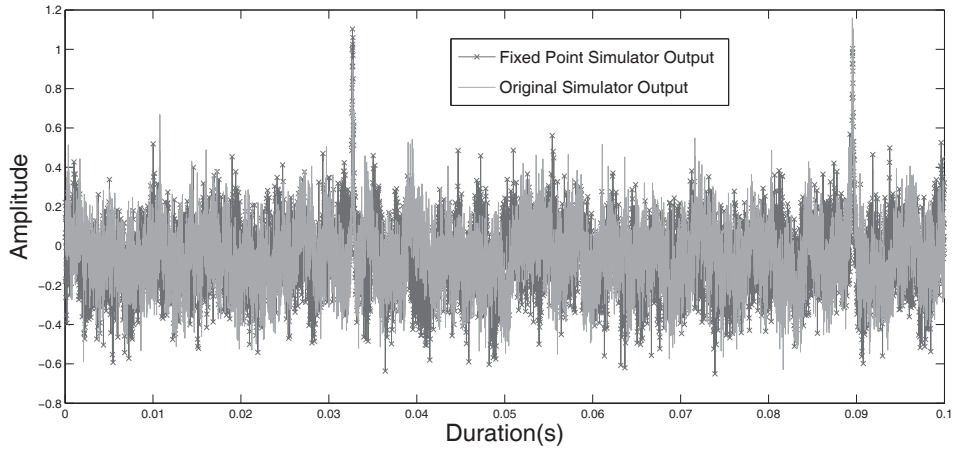

Figure 8.4: Comparison of the Original Simulator output and the fixed point Simulator Output

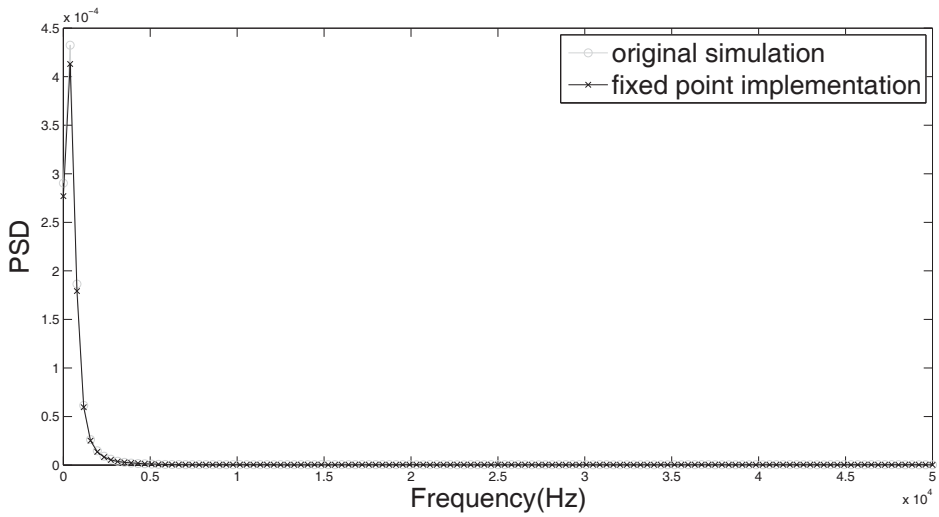

Figure 8.5: Comparison of the Power Spectral Density between Original Simulator output and the Fixed point Simulator Output

point implementation of simulator, the hardware design is made and is explained in detail in previous chapters, the design included the entity diagrams for each of the hardware blocks, data path and state machine to achieve the required functionality supported by the memory.

The design was performed to keep the area of implementation low, this is

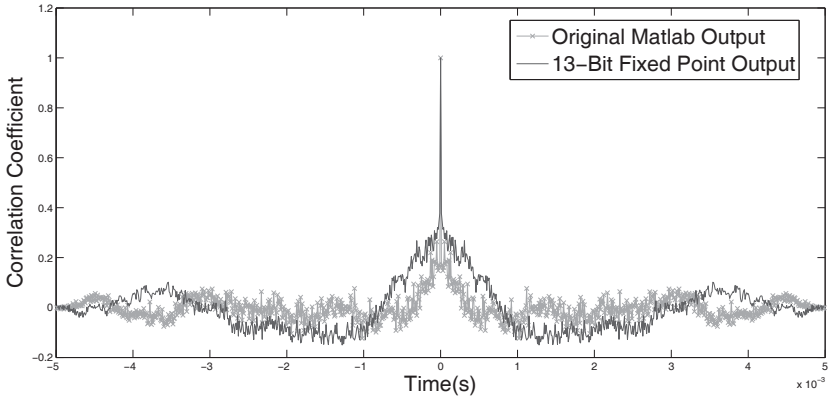

Figure 8.6: Comparison of the Autocorrelation between Original Simulator output and the Fixed point Simulator Output

achieved by multiplexing the mathematical modules and random number generators. Also wherever the usage of additional hardware block provides the speed advantage which outperforms the area advantage, the speed is given the priority and the multiple blocks of same functionality is used. All in all the design was a balance between area and speed performance of the hardware.

The simulator hardware design in its current form has few limitations such as elimination of few of the input parameters, The spike waveforms are stored in ROM requiring the ROM to be re flashed with new data every time a new wave form data is required. Also dynamic generation of the spike wave forms is one more possibility. When it comes to algorithms, in the current implementation mostly the algorithms used in Matlab are used, these algorithms can be replaced with much faster and smarter algorithms. Also the future work should be to implement the design on hardware and verify the results.

The hardware design of the Real Time Neural Simulator is the first step towards implementing it on hardware. Work done in this thesis can be continued in multiple dimensions like optimizing algorithms, hardware implementation, and hardware implementation specifically aiming for area and speed optimization. Also the design can be extended to be interfaced to a Computer with software providing user interface to dynamically program the parameters input data.

The idea of the simulator was to support the evaluation and benchmarking of the signal processing algorithms in the different stages of a BMI system, which means the simulator requires the implementation on a hardware. The hardware design proposed in this thesis is very much feasible for implementation on sufficiently large FPGA's. Though the implementation sacrifices some flexibility of controlling the input parameters. A successful implementation of the presented design on an FPGA can generate the signals that can be fed to a BMI system facilitating the evaluation of performance of the algorithms in that system justifying the proposed intention of the Neural Simulator.

References

[1] P.T. Thorbergsson and H.Jorntell and F.Bengtsson and M.Garwitcz and J.Schouenborg and A.J Johansson, *Spike Library Based Simulator for Extracellular Single Unit Neuronal Signals*,

proc. Annual International Conference of IEEE Engineering in Medicine and Biology Society,Minneapolis, Minesota, USA, sep, 2009, 6998-7001

[2] P.T. Thorbergsson, M.Garwitcz and J.Schouenborg and A.J Johansson, *Statistical Modelling of Spike Libraries for Simulation of Extracellular Recordings in the Cerebellum*,

proc. Annual International Conference of IEEE Engineering in Medicine and Biology Society,Beunos Aires, Argentina, , sep, 2010, 4250-4253

[3] Eric R. Kandel, James H.Schwartz, Thomas M.Jessell, Steven A.Siegelbaum, A.J. Hudspeth, *Principles of Neural Science*, McGrawHill,Newyork, USA

[4] S.G. Mason and A. Bashashati and M.Fatourechi and K.F.Navarro and G.E.Birch, *A Comprehensive Survey of Brain Interface Technlogy Designs*,

proc. Annual International Conference of IEEE Engineering in Medicine and Biology Society,Vol 35, No 2, Feb, 2007,137-169

[5] Bernhard Graimann, Gert Pfurtscheller and Brendan Allison, *Brain-Computer Interfaces Revolutionizing Human-Computer Interaction*,

Springer,Heidelberg, Germany, 2010

[6] Tan D.S, *Brain-Computer Interfaces: Applying our Minds to Human-Computer Interaction*, Springer Verlang,Heidelberg, Germany, 2010

[7] Makoto Matsumoto and Takuji Nishimura, *Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator*,

ACM Transactions on Modeling and Computer Simulation (TOMACS),8, Issue 1, Jan, 1998, 3-30

[8] George Marsaglia and Wai Wan Tsang, *The Ziggurat Method for Generating Random Variables*,

Journal of Statistical Software,5 Issue 8, Feb, 2000, 1548-7660

[9] George Marsaglia and Wai Wan Tsang, *A Simple Method for Generating Gamma Variables*,

ACM Transactions on Mathematical Software , 26, September, 2000, 363-372

[10] Jack E.Volder, *The cordic trignometric computing technique*,

IRE Trans. Electron. Comput. , 8, Jan, 1959, 330-334

[11] Jack E.Volder, *The Birth of Cordic*,

Journal of VLSI Signal Processing 25, 101-105, 2000

[12] J.S.Walther, *A Unified algorithm for elementary functions*,

AFIPS (Spring) of the may 18-20, 1971, spring joint compuer conference, 25,May, 1971,379-385

[13] Bruce H.Edwards, Robert T.Jackson, Jeremy M.Underwood, *How do caclulators calculate*

[14] Israel Koren, *Computer Arithmetic Algorithms*, A.K.Peters, MA, 2000

[15] Matlab, *Matlab*, www.matlab.com

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2016-484

http://www.eit.lth.se