Stochastic resonance and finite resolution in a network of leaky integrate-and-fire neurons

Nhamoinesu Mtetwa

Department of Computing Science and Mathematics

University of Stirling, Scotland

Thesis submitted for the degree of Doctor of Philosophy

2003

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

Declaration

I declare that this thesis was composed by me and that the work contained therein is my own. Any other people's work has been explicitly acknowledged.

Acknowledgements

This thesis wouldn't have been possible without the assistance of the people acknowledged below. To my supervisor, Prof. Leslie S. Smith, I wish to say thank you for being the best supervisor one could ever wish for. I am specifically thankful for giving me the freedom to walk into your office anytime. This made it easy for me to approach you and discuss any research issues as they arose.

Dr. Amir Hussain, a note of thanks for advice on the signal processing aspects of my research.

Dr. Mike Roberts and Dr. Douglas McLean, I appreciate your significant help on the mathematical aspects of this thesis.

Katie Howie, I am grateful for your help with the statistical analysis of the results.

CSG (Sam, Graham and Claire) and Frank, you were excellent on assisting me with the technical aspects of my work.

Dr. David Sterratt and Dr. Catherine Breslin, thank you for taking time to read the thesis and providing useful comments.

Laura Kelling, you are a genius at proofreading, much appreciated.

Dr. Hans Plesser, your work was the original inspiration to this thesis. I thank you for your readiness to answer any questions during the conducting of this research.

CCCN, thank you for useful feedback during research seminars.

Stirling University, thank you for funding my research. This work would not have been possible without the funding.

Last but not least, I wish to thank Thandi for being there for me during the highs and the lows of this work.

Abstract

This thesis is a study of stochastic resonance (SR) in a discrete implementation of a leaky integrate-and-fire (LIF) neuron network. The aim was to determine if SR can be realised in limited precision discrete systems implemented on digital hardware.

How neuronal modelling connects with SR is discussed. Analysis techniques for noisy spike trains are described, ranging from rate coding, statistical measures, and signal processing measures like power spectrum and signal-to-noise ratio (SNR). The main problem in computing spike train power spectra is how to get equi-spaced sample amplitudes given the short duration of spikes relative to their frequency. Three different methods of computing the SNR of a spike train given its power spectrum are described. The main problem is how to separate the power at the frequencies of interest from the noise power as the spike train encodes both noise and the signal of interest.

Two models of the LIF neuron were developed, one continuous and one discrete, and the results compared. The discrete model allowed variation of the precision of the simulation values allowing investigation of the effect of precision limitation on SR. The main difference between the two models lies in the evolution of the membrane potential. When both models are allowed to decay from a high start value in the absence of input, the discrete model does not completely discharge while the continuous model discharges to almost zero.

The results of simulating the discrete model on an FPGA and the continuous model on a PC showed that SR can be realised in discrete low resolution digital systems. SR was found to be sensitive to the precision of the values in the simulations. For a single neuron, we find that SR increases between 10 bits and 12 bits resolution after which it saturates. For a feed-forward network with multiple input neurons and one output neuron, SR is stronger with more than 6 input neurons and it saturates at a higher resolution. We conclude that stochastic resonance can manifest in discrete systems though to a lesser extent compared to continuous systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is concerned with the realisation of stochastic resonance in a leaky integrate-and-fire (LIF) neural network, modelled discretely and a comparison with what has been observed in floating point based LIF neurons. Stochastic resonance can be loosely defined as the amplification of signals using noise. The original inspiration of this work comes from the observed apparent randomness of spike trains from *in vivo* recordings when a neuron is presented with the same stimulus several times. This suggests that neurons communicate using a noisy code. Neurons achieve feats which programmed systems cannot yet achieve using this noisy looking code. The spike trains appear noisy to us because we do not understand the code that neurons use to communicate. It also turns out that the models that researchers develop are often able to reproduce results from real neurons when noise is deliberately added one way or another to the models. This suggests that there is a certain type and amount of noise which neurons require to function properly. In electronic systems, engineers invest a lot of time and money in trying to minimise noise (Ott, 1976) yet neurons seem to need noise to function properly (Mainen and Sejnowski, 1995). The tools scientists and engineers have developed are generally not intended for handling noisy signals. Stochastic resonance provides a constrained way of realising certain benefits of noise. Stochastic resonance is well established in continuous systems but not well established in discrete systems. This thesis discusses discrete-based stochastic resonance

in an integrate-and-fire type neural network and compares it with the well established continuous based stochastic resonance.

1.1 Aim of the thesis

The aim of this thesis is to determine whether stochastic resonance relies on the underlying continuous nature of a system or whether a linearly discretised system is able to display stochastic resonance. A follow on question to this is that of how discrete-based stochastic resonance would be affected by varying the precision of the values in the simulations.

The thesis is primarily about a comparison between the realisation of stochastic resonance in single LIF neurons, and in a small network of LIF neurons both implemented on two different platforms. The first platform is floating point (Java on a PC) hence high resolution, while the second platform is discrete and low resolution (Handel-C on Field Programmable Gate Array (FPGA)). On the integer platform we were able to vary the resolution (or word length) at which computations are done.

1.2 Methodology

In order to realise the aims outlined above, we developed a floating point based LIF neuron network model and implemented it in Java. We then derived an integer (discrete) model from the floating point model using an approximation. The integer model was implemented on an FPGA using Handel-C. The two network models were run using the same input except that the integer model received quantised versions of what the floating point model received. Spectral analysis was performed on the resulting spike trains to ascertain stochastic resonance. In the process of developing the models and analysing the subsequent results, we made some interesting observations which we present under contribution to knowledge.

1.3 Contribution to knowledge

The underlying goal of the project from which the thesis topic emerged was to try to bring the work of modelling noisy neurons closer to what could possibly be used in a real life gadget for a reasonable amount of money. Literature abounds with neuronal models whose practical viability has not been tested. By implementing neuronal stochastic resonance on a precision-constrained digital platform of the FPGA type we aim to bring neuronal modelling closer to its application in an as yet undefined application. Prosthetic devices like hearing aids would be a good target. Floating point realisation would require either huge, expensive Application Specific Integrated Circuits (ASICs) or huge FPGAs. We aim to come up with limits in terms of the numerical precision required for an integer model to approximate what has already been achieved by floating point models.

There are two main contributions to knowledge from this thesis. The first concerns the realisation of stochastic resonance in discrete systems. We show that discretised systems can display stochastic resonance though to a reduced extent compared with continuous systems. This is important to know because it means that one can develop real-time stochastic resonance based digital electronic systems. This is hard to achieve with floating point representation as it requires a large amount of chip area. In addition, it makes FPGA implementation possible.

The second contribution is the discovery that discrete based stochastic resonance is sensitive to the precision of the numbers within the simulation. Our results show that with a resolution of 12 bits one can realise discrete stochastic resonance which is almost comparable to that of continuous systems. This is good news because it means that one can get real-time performance with reduced chip area using commercially available FPGA chips. Because the resolution of the values in the simulations is inversely proportional to chip area, smaller resolutions allow us to build bigger networks than one could with the bigger resolutions associated with floating point representation.

1.4 Outline of thesis chapter by chapter

Chapters two and three are mainly review chapters but we also raise some questions which our models answer in subsequent chapters. The rest of the thesis describes the modelling work that was done and the analysis and discussion of results. The techniques used for analysing the results are also discussed.

Chapter 2 deals with general neurophysiology and how some of this physiology is represented in computer models of neurons. We begin with a description of a canonical neuron according to Amit (1989). Modelling of spiking neurons plays an important role in this thesis and therefore a large part of this chapter is devoted to their discussion. We note that neurons can be modelled at different levels of detail by describing detailed models like the Hodgkin-Huxley type to simplified models such as the integrate-and-fire type. The issue of noise is also central to the thesis. Accordingly, a section is devoted to discussions on the origins of noise in the central nervous system and how models cope with noise. After discussing the modelling of neurons in detail we then introduce the issue of how these models are implemented in digital hardware especially of the FPGA type. This brings in the issue of precision limitation because of the discrete nature of digital hardware.

Chapter 3 is devoted to stochastic resonance. We start by defining stochastic resonance, followed by a description of its origins. This chapter also discusses how stochastic resonance is measured and how it manifests in neuronal models of the integrate-and-fire type.

Chapter 4 deals with the techniques that we use in the thesis to analyse noisy spike trains and to quantify stochastic resonance. We describe the various measures that can be derived from spike trains which include neural codes, statistics like the coefficient of variation and correlation functions, and signal processing-based measures like the power spectrum and signal-to-noise ratio.

Chapter 5 is concerned with models that were used to simulate continuous based and discrete based stochastic resonance in single LIF neurons and in a small network of LIF neurons. We start by describing the network topology used in the models followed by the floating point based LIF neuron model. We then derive the integer model as an

approximation of the floating point LIF neuron model. We describe a discretised floating point model which is midway between the floating point model and the integer model. This model was necessitated by the fact that the integer model and the floating point models are too far apart. We also compare these models. After introducing the models and the network topology, next we describe the Java classes for implementing the floating point model and the hardware (Field Programmable Gate Array) and Handel-C (a hardware compilation language) for implementing the integer model.

In chapter 6 we present and discuss the results of simulating stochastic resonance on the two platforms. We identify the quantisation of the activation as the main source of differences between the integer and floating point models. We also report results on the effect of precision limitation on stochastic resonance.

Finally, in chapter 7 we list what has been achieved in this thesis and what it means to the neuroscience and digital design communities. We finish with suggestions for further work on the thesis topic.

Chapter 2

Background

This chapter provides background material to neuronal modelling. Since our particular interest is stochastic resonance, we will limit what would otherwise be an impossibly large review to only those aspects of neuronal modelling which support the main subject of interest.

The task of analysing neurons begins by looking at models which completely neglect the dendritic tree and replace the conductance-based description of the spiking process by one of two canonical descriptions. These two steps dramatically reduce the complexity of the problem of characterising the electrical behaviour of neurons. Instead of solving coupled non-linear differential equations, we are left with a single ordinary differential equation. Such simplifications allow us to treat formally networks of large numbers of interconnected neurons, as exemplified in the neural network literature, and to simulate dynamics. Understanding any complex system always entails choosing a level of description that retains key properties of the system while removing those properties which are not essential for the purpose at hand.

In this chapter we will discuss some neural models and how they are affected by noise and it is this noise aspect that brings in the issue of stochastic resonance. We will also discuss the problems associated with implementing some of these models on digital hardware.

Section 2.1 deals with the main components of a neuron from a functional and modelling point of view. This is followed by a discussion of the various levels at which neurons can be modelled in section 2.2. Section 2.3 discusses the noisiness of neurons and we follow this up with a discussion of how noise is incorporated into neuronal models in section 2.4. In section 2.5 we discuss the problems we face when we try to map artificial neural networks onto hardware. FPGAs, which is the hardware that is used in this thesis are described in section 2.6. The chapter ends with a summary in section 2.7.

2.1 Neurophysiology of neurons

There are a large variety of neurons in the nervous system of animals and humans, with variations in structure, function and size. We will restrict ourselves in this thesis to a canonical neuron (Amit, 1989), which presents the underlying functional skeleton for all neurons. The canonical neuron is divided into three parts (see figure 2.1), an input part (the *dendritic tree*), a processing part (the *soma*), and a signal transmission part (the *axon*). There are arguments about how much processing takes place on the dendrites. We will restrict ourselves to the processing that takes place at the soma. The detailed physiology of a real neuron is too complicated to discuss here, let alone attempt to model in a network. In this section we discuss the basic functions of a neuron. Even this basic description is still not simple enough for most modelling purposes.

Neurons communicate via synapses, which are the interfaces between the dendrites of the post-synaptic neurons and axons of presynaptic neurons. There is a small number of branches of dendritic trees entering the soma of the neuron. Usually, one axon leaves the soma and then, downstream, it branches repeatedly to communicate with many postsynaptic neurons, i.e., the neurons the specified neuron is talking to.

Having described the three main components of a neuron, next we discuss some of the models used to simulate real neurons. In the next section we will discuss some of the levels at which neurons are modelled. It was discovered that even the canonical neuron that we have just described results in complicated models consisting of coupled non-linear differ-

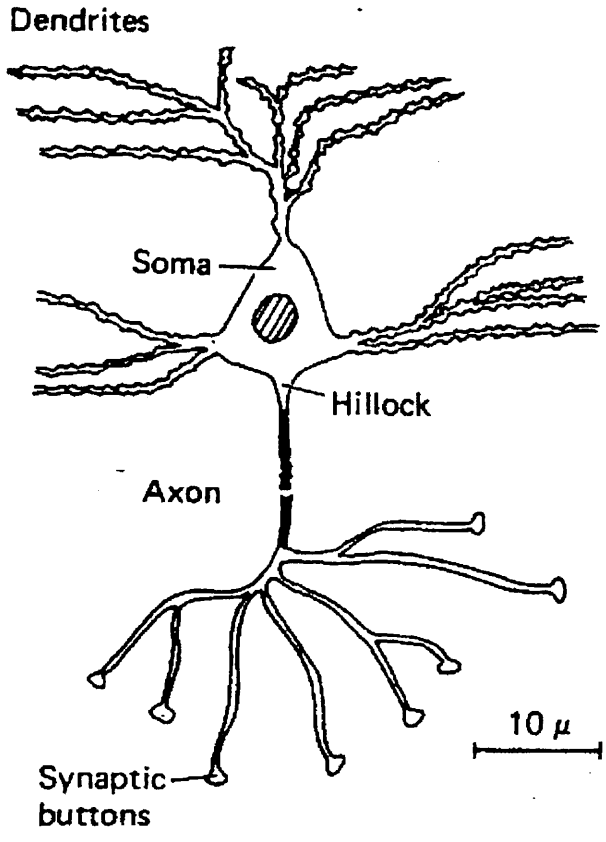

Figure 2.1: Schematic representation of a canonical neuron, from Amit (1989).

ential equations which are not mathematically tractable and are expensive to implement in hardware.

2.2 Neuronal modelling

Neuronal activity can be described at several levels of abstraction. On a microscopic level, there are a large number of ion channels (Kandel et al., 2000; Breslin and Smith, 1999), pores in the membrane which open and close depending on the voltage and the presence (or absence) of various chemical messengers. Although one can model these individually, it

is more common to lump them together and model patches of membrane. Such models are called compartmental models. Compartmental models are used to model neurons at this level of detail (Maass and Bishop, 1999), they can also be used to model the morphology of neurons.

At a higher level of abstraction, we do not worry about the spatial structure of a neuron nor about the exact ionic mechanisms. We consider the neuron as a homogeneous unit which generates spikes if the total excitation is sufficiently large (Koch, 1999). This is the level of the integrate-and-fire neuron model (an example of the so called threshold-fire neural models).

Threshold-fire neuron models should be contrasted with rate models which are reviewed further on in this chapter. Rate models neglect the pulse structure of the neuronal output, and therefore have a higher level of abstraction. On a yet coarser level would be models which describe the activity in, and interaction between, cortical columns or even whole brain areas.

2.2.1 Detailed models

In this subsection we briefly discuss some of the neuron models which are used to model certain functionalities of real neurons in detail. The models reflect the level of abstraction at which the neurons are being modelled.

Hodgkin-Huxley model

The classic description of neuronal spiking dates back to the work of Hodgkin and Huxley (reviewed in Kandel et al. (2000) but based on the original 1952 paper by Hodgkin and Huxley) who summarised their extensive experimental studies on the squid giant axon with four coupled nonlinear differential equations. Using these equations and an appropriate set of parameters, Hodgkin and Huxley were able to explain data from experiments on the giant axon of the squid. This model takes into account the exact properties of neurons, expressed in terms of membrane potential and the ionic conductances and potentials. Ac-

tion potentials, corresponding to pulse-like depolarisations of the neuron's transmembrane potential, are the result of ionic current flows. This model has an inherent threshold. It is however obvious that such an approach is too detailed for modelling large networks of neurons, at least, with current technologies.

The system of equations proposed by Hodgkin and Huxley is rather complicated (Maass and Bishop, 1999). For this reason, several simplifications of the Hodgkin-Huxley equations have been proposed. The most common reduces the set of four equations to a system of two equations (Koch and Segev, 1999). One such resulting two dimensional model is called the FitzHugh-Nagumo model. For a review of the methods and results see the article by Rinzel and Ermentrout in the review collection (Koch and Segev, 1999). Even this is still not simple enough for most modelling purposes. This model also has an inherent threshold. Even though it is considerably simpler than the Hodgkin-Huxley model, it is still too complicated to be mathematically tractable. For a further reduction of the Fitzhugh-Nagumo model to an integrate-and-fire model see (Abbott and Kepler, 1990).

Compartmental models

The neuron model described so far does not take into consideration the spatial structure of the neuron. A neuron has been considered to be a point-like element and the main focus has been on the process of spike generation. Compartmental models provide a more detailed description of neuronal dynamics and morphology by explicitly taking into account the spatial structure of the dendritic tree and by modelling the synaptic transmission at a greater level of detail. Additionally, other ionic currents beyond the sodium and potassium currents incorporated in the Hodgkin-Huxley model are included in these models. Compartmental models also allow non-homogeneous placing of ion channels.

Conductance-based and compartmental models are more suitable for modelling single neurons in detail (Maass and Bishop, 1999). The amount of detail they involve render them unsuitable for modelling large networks. There is not a large literature which discusses the implementation of neuronal stochastic resonance using these detailed models. Some of

the attempts to model stochastic resonance in detail include the works of Hutcheon et al. (1996), Lee et al. (1998), and Lee and Kim (1999). It is worth noting that there are hardware versions of these detailed models as evidenced by the work of Mahowald and Douglas (1991). This level of detail will not be cheap to model in hardware so next we discuss simplified models which are more likely to be realisable cheaply on digital hardware platforms as networks.

2.2.2 Simple models

In this subsection we discuss reduced versions of the models discussed above which are more amenable to the simulation of large networks. We will discuss rate-based and integrate-and-fire models.

Rate-based models

In standard neural network theory, neural activity is described in terms of spike rates. The rate v_i of neuron i is an analogue variable which depends non-linearly upon the excitation u_i of the neuron:

$$v_i = g(u_i) \tag{2.1}$$

where $g(.)$ is usually taken as the sigmoidal function with $g(u) \to 0$ for $u \to -\infty$ and $g(u) \to 1$ for $u \to \infty$ (Maass and Bishop, 1999). The excitation is given by a linear sum over all input connections:

$$u_i = \sum_j w_{ij} v_j \tag{2.2}$$

where v_j is the output rate of a presynaptic neuron j. The sum runs over all neurons which send signals to neuron i. The weight factors w_{ij} give the weight attributed to connections from j to i. w_{ij} is the synaptic weight, equations 2.1 and 2.2 can be combined into a single equation 2.3:

$$v_i = g(\sum_j w_{ij} v_j) \tag{2.3}$$

which is the starting point of standard neural network theory (Hertz et al., 1991). Equation 2.3 is a static equation. It applies to situations where a stationary input (a set of firing rates v_j) is mapped to stationary output (the rate v_i). Equation 2.3 is not useful for investigating stochastic resonance in its current state because stochastic resonance deals with time-varying signals. It is possible to make the equation time-dependent. A straightforward way to introduce dynamics into the rate equation is to replace it by a differential equation (Koch, 1999; Abbott, 1991)

$$\tau \frac{dv_i}{dt} = -v_i + g(\sum_j w_{ij} v_j) \tag{2.4}$$

with a time constant τ. For stationary input and output, the left-hand side of 2.4 disappears and it reduces to 2.3.

Equation 2.4 provides a convenient way to introduce some time dependence in the rate model but can it be considered a realistic description of neuronal activity? As pointed out by Maass and Bishop (1999), an analogue variable described by a spike count measure requires a long temporal averaging window. It can therefore be only used if the input and the output change on a slow time scale. Considering the fact that, for example, the visual input changes due to saccades on average every 200 ms (Kandel et al., 2000), a slowly changing input cannot always be assumed. It has therefore been argued that the rate equation (2.4) refers to a population average rather than to a temporal average (Koch, 1999). The details of temporally averaged rates and other types of spike-based rates are discussed in chapter 4.

Integrate-and-fire models

The integrate-and-fire model is an important example of the so called "threshold-fire" models. It comes in two types, namely, the perfect integrate-and-fire (IF) and the leaky integrate-and-fire (LIF) models (figure 2.2). The perfect integrate-and-fire model is simple but quite powerful. This model assumes that the neuron integrates its inputs and generates a spike when a voltage threshold is reached. In mathematical terms it is described in its

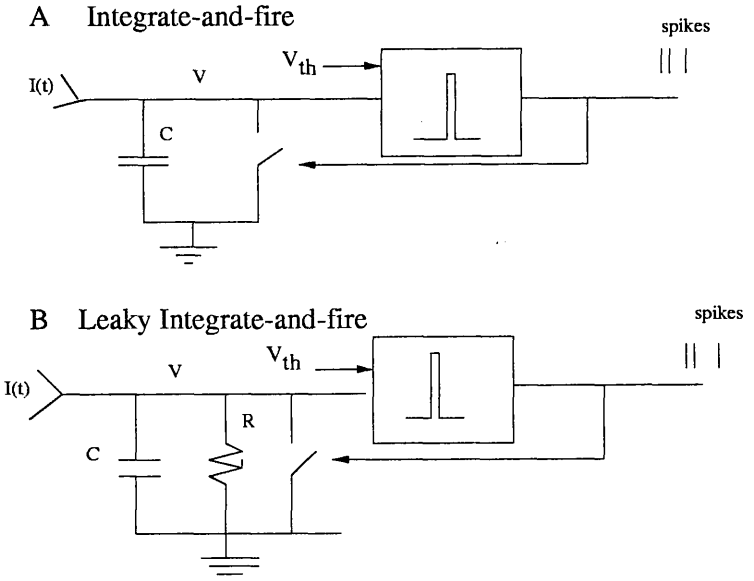

Figure 2.2: Variants of the integrate-and-fire model. Common to both are passive integration with a single compartment for subthreshold domain and a voltage threshold V_{th}. When the membrane potential V reaches threshold V_{th}, a pulse is generated and the circuit is short circuited. (A) Perfect or non-leaky integrate-and-fire model. (B) Leaky (or forgetful) integrate-and-fire model unit accounts for the decay of the membrane potential by an additional component, a leak resistance R.

subthreshold form by equation 2.5 (Koch and Segev, 1999):

$$C\frac{dV}{dt} = I(t), \tag{2.5}$$

where $I(t)$ is the input current, integrated to yield the membrane voltage $V(t)$. C represents the capacitance of the model cell. The equation specifies the evolution of the membrane potential in the subthreshold domain. A spike is generated when $V(t)$ reaches the threshold V_{th} and the membrane voltage is reset to V_0 (where V_0 is the reset value often set to zero for simplicity) immediately after a spike. This model will sum linearly two subthreshold inputs irrespective of their temporal separation because it has no decay term.

The model which will be used in this thesis is the *leaky* integrate-and-fire neuron. This model idealises the neuron as a capacitor C in parallel with a resistor R to resemble

the ion channels and pumps in real neurons (figure 2.2). In the absence of input, the potential difference across the capacitor will be $V = V_0$ where $V_0 = 0$ without a battery. The effective input current $I(t)$ may hyperpolarise or depolarise the membrane. Once the neuron is sufficiently depolarised for the potential $V(t)$ to reach the threshold V_{th}, a spike is discharged and $V(t)$ is set to V_0. This results in the following equation for the dynamics of the neuron (Tuckwell, 1988a):

$$C\frac{dV}{dt} = -\frac{V}{R} + I(t) \tag{2.6}$$

Equation 2.6 describes the subthreshold dynamics of the leaky integrate-and-fire model. The leak term R represents the resistance to current flowing out of the cell and C is defined as it is in equation 2.5.

With a normalisation of the capacitance C, equation 2.6 can be rewritten as:

$$\frac{dV}{dt} = -\frac{1}{\tau}V(t) + I(t). \tag{2.7}$$

where $\tau = RC$ is the membrane time constant. The action potential generation is not an inherent part of the model as in more complex models. Equations 2.5 and 2.7 do not have a threshold as part of it hence they are only valid for subthreshold voltages. Spike generation and the reset of the membrane potential have to be added explicitly to fully describe the neuron. Equation 2.7 is a first order differential equation that describes the membrane potential dynamics in between firing events. When $V(t)$ reaches a threshold V_{th}, the membrane potential is reset to V_0 and an output spike is generated. For recent results on the leaky integrate-and-fire model from a computational point of view see Bugmann (1991); Tal and Schwartz (1997); Maass and Bishop (1999).

With the initial condition $V(t_0) = V_0$, the solution to equation 2.7 reads (Scharstein, 1979)

$$V(t) = V_0 e^{-(t-t_0)/\tau} + \int_{t_0}^{t} I(u)e^{(u-t)/\tau}du. \tag{2.8}$$

Refractoriness

In a real neuron, for a few milliseconds after a spike has been fired, it may be virtually impossible to initiate another spike. This is called the *refractory period.* For a longer interval known as the *relative refractory period*, lasting up to tens of milliseconds after a spike (Dayan and Abbott, 2001) depending on the type of neuron, it is more difficult to evoke a spike. Recordings from real neurons have revealed that the *absolute refractory period* (period of total silence) lasts about 12 ms on average (Koch, 1999). The reason for the inability of the membrane to discharge again immediately after a spike is the inactivation of certain currents.

It is straightforward to include an absolute refractory period into the LIF neuron model. After a spike at time $t^{(f)}$, we force the threshold V_{th} to a very high value K and keep it there during a time t_{ref}. At time $t^{(f)} + t_{ref}$ we bring V_{th} back to its original value. This is how the absolute refractory period is implemented in the models discussed in chapter 5. It is also common to implement the relative refractory period by bringing V_{th} gradually back to its original value.

Why LIF?

The LIF will be the model of choice to investigate stochastic resonance in this thesis because it gives priority to simplicity and speed. By omitting fine-grained detail of neuronal behaviour, it promotes the simulation of large neural networks at a level that can be considered only slightly lower than connectionist-style models. The leaky integrate-and-fire model is widely used in neuronal modelling and indeed stochastic resonance because it captures much of the important dynamics of real neurons (Koch (1999)), and it can cope very well with continuously time varying signals (Smith, 1996, 1998; Glover et al., 1999). It captures the two key aspects of neuronal excitability: a passive, integrating subthreshold domain and the generation of stereotypical impulses once a threshold has been exceeded.

These models (IF and LIF) despite being considered as over simplified, play an indisputable role in computational neuroscience. The seemingly simple LIF model has success-

fully been used to explain the high temporal precision achieved in the auditory system (Gerstner et al., 1996) and visual system (Marsalek et al., 1997) in spite of the comparatively long membrane time constants. It has also been employed widely in the ongoing debate about the origin of spike-rate variability in cortical neurons (Troyer and Miller, 1997; Bugmann et al., 1997; Feng, 1997), and as spike generator, in studies on synaptic gain control (Abbott et al., 1997). This arguably makes it the most widespread model in studies on neuronal information processing. A comparison of the integrate-and-fire model with the Hodgkin-Huxley model, when both receive a stochastic input current finds that the threshold model provides a good approximation (Plesser, 1999). It has been shown mathematically that the LIF corresponds to a first-order approximation of the full Hodgkin-Huxley model (Kistler et al., 1997; Stevens and Zador, 1998). The LIF is very popular in the study of stochastic resonance in neurobiology. The work of Stemmler (1996) showed that realistic models like the Hodgkin-Huxley model exhibit a stochastic resonance phenomenon that is comparable to that of an LIF neuron model when both models receive the same type of input. LIF neurons have also been shown to be used as powerful computing systems (Maass (1996, 1997)) and they are quite straightforward to implement in either software or hardware (Wolpert and Micheli-Tzanakou, 1996; Jahnke et al., 1997; Grassmann et al., 2002). A recent review of the integrate-and-fire model, giving details of the various variations of the model and comparisons with both experimental data and other models, is provided by Koch (1999).

Based on the belief that one should use the simplest model that can account for a given phenomenon, the integrate-and-fire model will be the only model used in this thesis. This agrees with what Henry Tuckwell reminds the reader in his book *Introduction to theoretical neurobiology* (Tuckwell, 1988b)

> ... one fundamental principle in neural modelling is that one should use the simplest model that is capable of predicting the experimental phenomena of interest.

Disadvantages of the LIF model

The LIF neuron model lacks the spatial structure seen in real neurons (Tuckwell, 1988a, 1989). All the external currents arriving at the neuron are included in the variable $I(t)$, meaning that all the synaptic inputs are treated as if they were arriving at a single point. This is why LIF neurons are sometimes called point neurons. Thus the model does not account for the position of synaptic inputs on the soma and the corresponding travel times of the postsynaptic potentials. The model typically lacks any of the specific currents that underly spiking, and thus the complex mechanisms by which the sodium and potassium conductances cause action potentials to be generated are not part of the model. An important assumption is also that synaptic inputs interact in a linear manner, so that phenomena involving nonlinear interaction of synaptic inputs, such as pulse-paired facilitation or depression (Shepherd, 1990) and synaptic inputs that depend upon postsynaptic currents, are neglected.

In this section we have described two types of models. We noted that compartmental and conductance-based models are more suitable for modelling single cells in detail, especially complex cells. The level of detail involved in the models makes them unsuitable for simulating on digital hardware, even at single neuron level, let alone as networks of such neurons. On the other hand, integrate-and-fire type models are simple enough to simulate on digital hardware.

There are two ways of looking at the models just described here. We can either treat them as deterministic models or non-deterministic. The choice between the two is guided by the type of input $I(t)$ that we supply. Non-deterministic models are considered when the input has a random element in it and deterministic models are considered when the input has no random element in it. The random element is usually referred to as noise. In the next section we will discuss the merits of this randomness and where it originates from in real neurons. In section 2.4 we discuss ways of introducing noise into an LIF neuron model.

2.3 Noise in neurons

In this section we will begin by looking at some of the possible sources of noise in the central nervous system and also discuss some of the benefits of the noise to the functioning of neurons. The term noise usually denotes something negative that blurs the signal processing. However, in some cases it could be a message by itself or a highly desirable part of the message important for its processing. The problem is that without understanding all the details of the neural code, we cannot distinguish signal from noise. From a mathematical point of view the introduction of stochasticity into the description of a neuron represents an increase in complexity. On the other hand, from the point of view of biological realism it simplifies the task substantially, as all the features considered at the current stage to be marginal can be attributed as a system noise. In chapter 4 the techniques used in this thesis to process noisy spike trains will be discussed.

2.3.1 Origins of noise in neurons

Some of the possible sources of noise in the nervous system are the membrane, the state of arousal, the spike generation mechanism, synapses and uncorrelated input signals.

For the membrane, noise comes from the gating of channels and the pumping of ions, which are discrete processes, giving rise to fluctuations (Kandel et al., 2000). Noise analysis techniques are used to analyse the noisiness of the membrane potential (Bryant and Segundo, 1976; Traynelis and Jaramillo, 1998). For example, when a voltage-dependent ion channel undergoes random transitions between open and closed states, the result is a fluctuating current. This current introduces a voltage noise (noise in the membrane potential) that is sensed not only by this channel, but also by other voltage-dependent channels. However, it is believed that the sheer number and supposed independence of channels and pumps averages the fluctuations to a level where they are largely irrelevant to the operation of the neuron (Plesser, 1999). The membrane is thus not a major source of irregularities.

Larger irregularities are generated at the chemical synapses: each signal transmission between pre- and postsynaptic neuron is achieved by the release of a random number of

vesicles of neurotransmitters (Senn et al., 1998). This results in measurable fluctuations in post-synaptic currents and potentials. The stochastic nature of synaptic transmission is exemplified by the fluctuations in both the number of quanta released by a nerve terminal in response to some stimulus and the number of postsynaptic receptors activated by the neurotransmitter release (Traynelis and Jaramillo, 1998).

The other source of irregularity in neuronal activity is not at the molecular or synaptic level. Experiments by Mainen and Sejnowski (1995) demonstrated that the irregularity in firing patterns of cortical neurons can largely be traced back to the irregularity of the input impinging on these neurons. This is consistent with the observation that neurons fire in a more apparently random way when recorded from living animals, while generating regular spike sequences in slice preparations.

2.3.2 Some benefits of noise

At this point it is appropriate to note that noise does not always hinder signal transmission. The work of Marsalek et al. (1997) proved that the transmission of a signal between input to neurons and output from neurons is relatively noise free. In this paper they measured the amount of jitter in an input spike train and then measured the amount of jitter in the output spike train and they discovered that the ratio of the input jitter to the output jitter is always less than 1. This result shows that even though synapses are a source of noise, the noise does not hinder the transmission of signals but somehow it can aid the propagation of the signal. This will be pursued further when we look at a network and show that the signal-to-noise ratio improves as a signal propagates across the network. Maass and Natschlager (1999) also confirmed this result by showing both theoretically and in simulation that the known unreliability of synapses does not impede useful computation. Rather, it allows fast analogue computation in the time-domain. Independently, Gerstner (2000) has shown that noisy synaptic functionality can stabilise populations of spiking neurons and random noise clearly plays an important constructive role in real neural systems (Godiver and Chapeau-Blondeau, 1996). Indeed, noise can even improve performance in networks as far removed

from biological plausibility as the Multilayer Perceptron, as shown by the work of Murray and Edwards (1993).

The central nervous system (CNS) is, therefore, a noisy environment. The noise seems to mainly originate from synaptic activity and the stimuli which impinge on sensory neurons. The presence of noise affects the behaviour of relatively simple systems, such as motor neurons, whose discharge is a function of the weighted contribution of multiple inputs (Traynelis and Jaramillo, 1998). The question to be asked then is: is this noise merely a consequence of the underlying stochastic process or does noise play an active (or otherwise) role in information processing? The notion that noise could play a constructive role abounds in literature (Volgushev and Eysel, 2000; Benzi et al., 1982; Murray and Edwards, 1994). Several strategies have been conceived to use noise in a useful fashion.

We view neurons as signal processing devices which means one can look to conventional signal processing techniques for methods to deal with, and indeed take advantage of, noisy neural output.

2.3.3 Noise and signal processing

A common initial reaction is that a random signal with ill-defined properties can have little place in any scientific theory of signal analysis, but the opposite is nearer to the truth (Lynn, 1989). Suppose, for example, it is desired to send a message along a telegraph link. It is almost valueless to send a known, deterministic, message since the person at the receiving end learns so little by its reception. As a simple example, it would be pointless to transmit a continuous sinusoidal tone, since once its amplitude, frequency and phase characteristics have been determined by the receiver, no further information is conveyed. But if the tone is switched on and off as in Morse-code, the receiver does not know whether a "dot" or "dash" is to be sent next, and it is this nonstationarity or uncertainty about the future of the signal that conveys useful information (Shannon, 1948). Viewed in this light, it is no surprise that spike trains from *in vivo* recordings appear quite noisy. It looks like neurons naturally generate random signals which are rich in information content. The

irony though, is that, even with this randomness in the neural activity, there is a high degree of consistency in most perceptual and cognitive tasks. In our view, this apparent paradox is the key to cracking the elusive neural code.

In signal processing, noise has a not entirely undeserved reputation for difficulty and obscurity. This is due partly to the complexity of systems in which noise is significant and partly to the unfamiliar nature of the mathematical tools involved (Robinson, 1974). For most of the analysis of spike trains we shall be using the language of signal processing theory, where the emphasis is on the frequency or spectral components of the signal and the noise in a system, and for this purpose the most useful concept in this thesis for the analysis of neuronal data is that of the power spectrum of the spike train.

In the next subsection we discuss the link between noise and neuronal activity. This is important because there are a number of useful results which have been reported in the literature as a result of adding noise to neuronal models.

2.3.4 Noise and neural coding

Experimental data recorded from very different neuronal structures and under different experimental conditions suggest the presence of stochastic variations in neuronal activity (Tuckwell, 1988b; Lansky and Lanska, 1997). We may assume that there is a random component, generally considered as noise contained in the input and/or output signal.

For many years scientists have known that the brain's response to outside information, such as a visual image, depends on the background of the internal signals that pervades the brain. As a result, if one provides a stimulus to a single neuron, it does not respond the same way each time. Most early researchers assumed that the brain resolves this issue by not relying on any one neuron. They thought that the redundancy of the neural system should ensure that if some cells are distracted or confused, many more will be alert and responding appropriately. This means that the brain was envisioned as averaging the output from perhaps hundreds of thousands of individual neurons to eliminate the variability due to anything but some common signal such as the visual image. This is a

good idea based on a faulty assumption, as observed by Grinvald in Raloff (1996) because such an averaging would only help if each neuron is influenced by different sources of noise than that which affects its neighbours. It turns out, however that these neurons are listening simultaneously to the same chorus of voices.

In the context of neural coding, "noise" is used to mean anything other than a strictly deterministic response of a neuron to the identified signal (Mar et al., 1996). "Noise" is also used to refer to stochastic aspects of the "signal" (such as synaptic vesicle release or photon arrival times), or to anything "non-signal". When neural responses are said to be "noisy" one of two things is meant: either the neural response is *irregular* or is *unreproducible.* The regularity of spike trains has to do with the fact that a cell firing 100 spikes a second on average is not necessarily firing 1 spike every 1/100 of a second like a clock; the distribution of intervals between spikes may be very wide. The reproducibility of spike trains has to do with the difference in neural response from trial to trial when the same stimulus is presented, especially with respect to the precision of the timing of individual action potentials. In an experiment involving rat neocortical slices Mainen and Sejnowski (1995) observed that constant stimuli led to imprecise spike trains, whereas stimuli with fluctuations resembling synaptic activity produced spike trains with timing reproducible to less than 1 millisecond. These data suggest a low intrinsic noise level in spike generation, which could allow cortical neurons to accurately transform synaptic input into spike sequences, supporting a possible role for spike timing in the processing of cortical information by the neocortex.

In this section we have discussed the issue of noise in neuronal models and the possible sources of the noise in the central nervous system. Also described in this subsection is how noise affects the techniques that are used to analyse noisy spike trains. In the next section we look into the issue of how to introduce noise into the LIF neuron model. It is through this introduction of noise that makes an LIF neuron exhibit stochastic resonance.

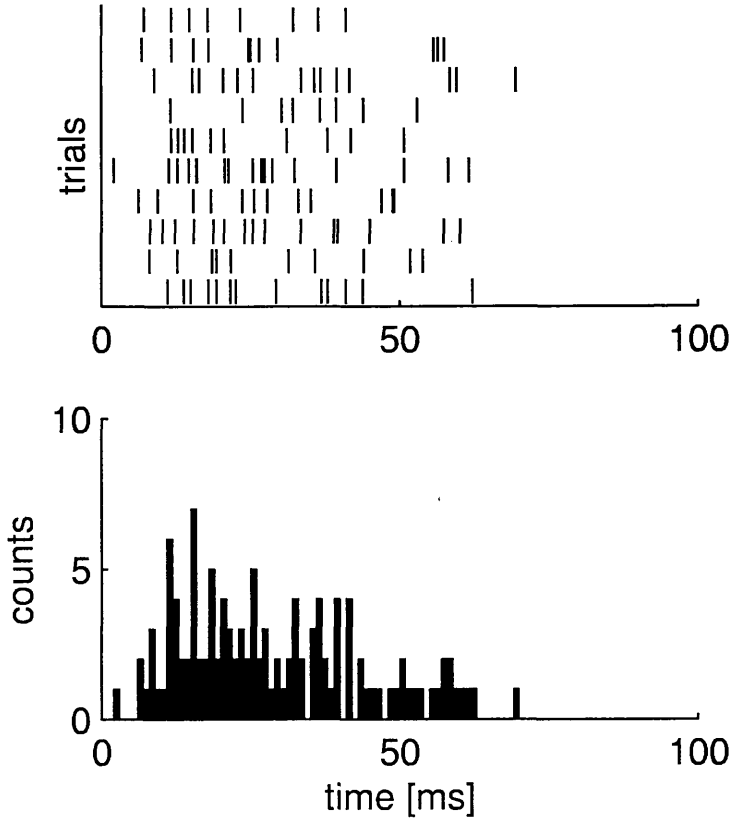

Figure 2.3: Variability of neuronal responses. The figure shows a raster plot of 10 individual spike trains to the same stimulus. Each small line in the raster plot marks the time of occurrence of a single spike. The bottom panel shows the Peri-Stimulus-Time Histogram (PSTH) resulting from counting the number of spikes in 1 ms bins. Adopted from Sterratt (2002).

2.4 Models of noise

Noise is omnipresent in biological systems due to non-zero temperatures and finite numbers of molecules, vesicles and ion channels. The effects of noise include failures in synaptic transmission and different responses of a neuron to the same input (see figure 2.3). We discuss here ways of introducing stochasticity into neuronal models by looking at a LIF neuron model.

There are various ways to introduce noise into a LIF neuron model. Here we discuss three, but we concentrate on one for implementation purposes in chapter 5. The three are (a) *noisy threshold*, (b) *noisy reset* and (c) *noisy integration*. In all cases we are interested in the effect of the noise on the firing period. In each case we give the expressions derived in Maass and Bishop (1999) for predicting when a spike will occur next given that it occurred just now.

In the presence of noise we can no longer predict the exact time of firing. Instead we ask the following question: what is the probability that the next spike occurs between t and $t+\Delta t$ given that the last spike occurred at $t^{(0)}$ and that the total input in the noiseless case is $I(t)$? For $\Delta t \to 0$ this defines the probability density for firing:

$$P_I(t \mid t^{(0)}) \tag{2.9}$$

which we would like to calculate for each of the three noise models. We can interpret $P_I(t \mid t^{(0)})$ as the distribution of intervals in the presence of input potential I. The lower index is a reminder that the distribution depends on the time course of $I(t')$ for $t^{(0)} < t' < t$. We now discuss each of the three models of noise in turn.

2.4.1 Noisy threshold

In this first noise model, we assume that the neuron can fire even though the formal threshold V_{th} has not been reached and might not fire, even though V_{th} has been exceeded. To do this consistently, we introduce an escape rate ρ which depends on the distance between the momentary value of the membrane potential V and the threshold V_{th}:

$$\rho = f(V - V_{th}) \tag{2.10}$$

In the mathematical literature, the quantity ρ would be called a *stochastic intensity* (Cox and Lewis, 1966). The choice of the function f is arbitrary. A reasonable assumption is an exponential dependence:

$$\rho = \frac{1}{\tau_0} \exp[\beta(V - V_{th})] \tag{2.11}$$

which can be motivated by the Arrehenius formula for chemical reaction rates (Plesser and Gerstner., 2000). β and τ_0 are parameters. Note that the escape rate ρ is implicitly time-dependent, since the membrane potential $V(t) = \eta(t - t^{(0)}) + I(t)$ varies over time. The kernel $\eta(t)$ describes the response of a neuron i to its own spikes.

Let us now calculate $P_I(t^{(1)}/t^{(0)})$, the probability density of having a spike at $t^{(1)}$ given that the last spike occurred at $t^{(0)}$ in the presence of input potential $I(t)$ for $t > t^{(0)}$. At each moment of time, the value of $V(t)$ determines the escape rate $\rho(t)$. In order to emit the next spike at time $t^{(1)}$, the neuron has to "survive" the interval $(t^{(0)}, t^{(1)})$ without firing and then fires at $t^{(1)}$. Given the escape rate $\rho(t)$, the probability of survival from $t^{(0)}$ to $t^{(1)}$ without firing is given by equation 2.12 (Maass and Bishop, 1999):

$$S_\rho(t^{(1)} \mid t^{(0)}) = \exp(-\int_{t^{(0)}}^{t^{(1)}} \rho(t)dt) \tag{2.12}$$

The probability density at time $t^{(1)}$ is $\rho(t^{(1)})$, thus from equation 2.12 we have:

$$P_I(t^{(1)} \mid t^{(0)}) = \rho(t^{(1)}) \exp(-\int_{t^{(0)}}^{t^{(1)}} \rho(t)dt) \tag{2.13}$$

which is the desired result. Also see Cox and Miller (1965) and Gerstner and van Hemmen (1992) for a more detailed derivation of equation 2.13.

2.4.2 Noisy reset

In this noise model, firing occurs on $V(t^{(f)})$ reaching V_{th} from below, i.e. $V(t^{(f)}) = V_{th}$. Noise is induced into the formulation of reset and refractoriness.

Let us consider the Integrate-and-Fire neuron with absolute refractoriness. The duration of refractoriness t_{ref} is not fixed, but chosen stochastically from a distribution $P(t_{ref})$ with mean $\bar{t}$. Naturally we have to require $P(x)$ to vanish for $x < 0$. This is equivalent to replacing the term $\eta(t - \hat{t})$ by $\eta(t - \hat{t} - t_{ref})$.

As with the preceding noise model, we are interested in calculating the firing density $P_I(t^{(1)}|t^{(0)})$ given some input potential $I(t)$. To simplify matters we will assume that the input is constant $I(t) = I_0$. The first spike has occurred at $t^{(0)}$. The firing time $t^{(1)}$ of the

next spike can now be found from the threshold condition $V_{th} = \eta(t^{(1)}-t^{(0)}-t_{ref})+I_0$. In the absence of noise and for a refractory period $\bar{t}$, the next firing would occur at $t^{(1)} = t^{(0)} + T$ where T is the interspike interval. If due to the noise, the value of the refractory period is $t_{ref} \neq \bar{t}$, then the interval is $t^{(1)} - t^{(0)} = T + t_{ref} - \bar{t}$. The firing probability density in the presence of noise is therefore:

$$P_I(t^{(1)} \mid t^{(0)}) = p(t^{(1)} - t^{(0)} - T + \bar{t}) \tag{2.14}$$

where $p(.)$ is the distribution of the refractory period. This result can be explained as follows: a change in the absolute refractory period shifts the trajectory horizontally. A stochastic component in the refractory period generates a stochastic shift in the firing time. Although a stochastic shift in the refractoriness is probably not realistic, it is a convenient way to introduce noise in a system (Koch and Segev, 1999).

2.4.3 Noisy integration

The final and most popular way for introducing noise into an integrate-and-fire model is by adding on the right hand side of equation 2.7 a stochastic noise current $I_{noise}(t)$:

$$\frac{dV}{dt} = -\frac{V(t)}{\tau} + I(t) + I_{noise}(t) \tag{2.15}$$

with vanishing mean and finite variance. This choice of noise term could be motivated either by spontaneous openings of ion channels, or else by stochastic arrival of excitatory and inhibitory inputs on dendrites of cortical neurons. The noise can cause the actual membrane trajectory to drift away from the noiseless reference trajectory. To get the distribution $P_I(t^{(1)} \mid t^{(0)})$ we have to solve the first-passage time problem of equation 2.15 with initial value V_0 and absorbing boundary condition at V_{th}. Although equation 2.15 looks simple, it turns out that the first-passage time problem for arbitrary input current $I(t)$ is rather difficult to solve analytically (Tuckwell, 1988b; Plesser, 1999). Solutions for the first-passage time problem for constant input $I(t) = I_0$ can be found in textbooks (see for example (Tuckwell, 1988b, 1989)). Most work investigating stochastic resonance in neuroscience is based on the LIF model with noisy integration (equation 2.15) which

is similar to the Langevin equation (Plesser, 1999). In subsequent chapters equation 2.15 will be revisited as it forms the basis of the simulation models for this thesis.

Having discussed neuronal modelling and the issue of noise in these models, next we discuss the issues which arise when these models are implemented on digital hardware as pointed out earlier in the chapter. This is of interest because the LIF model has been implemented in digital hardware before (see the work of Smith et al. (1998) for comparing hardware and software implementations of LIF neurons for sound processing and Glover et al. (1999) for hardware implementation of LIF neurons) but not for the purposes of investigating the effect of limited precision which is what we intend to do. Implementing a noisy LIF neuron model on digital hardware raises some interesting questions. We have different sources of noise interacting. The digitisation process introduces quantisation noise, the input has a noise component which is also digitised introducing noise upon noise. It is not clear how all these forms of noise affect stochastic resonance.

2.5 Artificial neural networks in hardware

The motivation for digital implementation of artificial neural networks (ANN) is as diverse as the background of the researchers (biologists, mathematicians, physicists and engineers) who contribute to the field (Maass and Bishop, 1999). On one side there are those who need detailed simulations of a single neuron while on the other side there are those who need to simulate networks of simpler neurons. In this section we will discuss some of the issues to be considered when mapping an artificial neural network model onto hardware. Some of the issues to be discussed include quantisation, limited precision, and restraining complexity. We will concentrate mainly on digital implementations of neural networks with particular attention to Field Programmable Gate Array implementation examples of which can be found in the works of Perez-Uribe and Sanchez (1997) and Waldemark et al. (1998). Due to programmability, digital hardware offers a high degree of flexibility and provides a platform for simulation on neuronal level as well as network level.

2.5.1 Mapping artificial neural networks onto hardware

In order to take advantage of the parallelism offered by artificial neural networks, hardware implementations are a good option. They offer the possibility of large performance increases, allowing networks to run at or near real time, and also large improvements in size, allowing those near real-time neural networks to be incorporated in other equipment. However, most neural network models were not designed for hardware implementations, hence adaptations are needed. There are many technologies for realising artificial neural networks on hardware (digital, analog electronics, optics, and hybrid techniques). Even though these implementations are different, the common denominator is mapping neural network algorithms onto reliable, compact and fast hardware. According to Moerland and Fiesler (1997), any hardware implementation has to optimise three main constraints: *accuracy, space* and *processing speed.* The design of hardware implementations is governed by a balancing of these three constraints. Analog implementations, for instance, are very efficient in terms of chip area and processing speed, but suffer from limited accuracy of the network components. In general, this amounts to a trade-off between the accuracy of the implementation and the reliability of its performance. In mapping neural networks on to silicon hardware, the first choice is analogue or digital (Hecht-Nielsen, 1989). Analogue networks suffer from noise problems, and also tend to be temperature sensitive. It can also be difficult to maintain weight values, without drift, over long periods (Forrester, 1993). Indeed a common solution is to use a digital value for the weight and to convert it to analogue when required (Glover, 1999). Digital networks require that all the values be digitised, and this (as we see below) brings its own problems.

2.5.2 Quantisation and limited precision in DSPs

In general, digital hardware is not fully implemented (Hecht-Nielsen, 1989), in the sense that some elements of the digital hardware are used to represent more than one part of the neuron. The best example is the synaptic multiplier (Bade and Hutchings, 1994), which, if fully implemented would be replicated at each synapse, but this is costly in terms of chip

area. For floating point, this is not possible. For fixed-point, it suggests a lower resolution multiplier at each synapse. Partially implemented systems, in which the multiplier is shared between many synapses, and possibly even between many synapses on different neurons, could use floating point, but this then requires careful multiplexing of inputs and outputs.

For simulations on workstations, floating point representation is a natural choice. The situation is quite different on digital hardware. These systems almost never provide a floating point unit, because it would require large chip area. Quantisation is inevitable, but with quantisation comes the question of what the representation precision should be. We have two choices: fixed-point and integer representations. Regarding the representation with fixed numbers, we have to know the required precision $k.f$, where k is the bit length of the integer part and f is the bit length of the fraction part. The representation of fixed-point compared to integer numbers has the advantage that we can change the scaling by shifting the decimal point without changing the total word length $k + f$ although we then need to store the scale. The problem of limited precision is not peculiar to artificial neural networks only, rather it is a general problem for digital signal processors (Ifeachor and Jervis, 2002). Any digital signal processor suffers from three sources of error due to limited word length in measurement and processing of the signal:

- limited precision due to word length when analog signal is converted to digital form;
- errors in arithmetic due to limited precision within the processor;
- limited precision due to word length when samples are converted back to analog.

These errors are often called "quantisation errors". The effects of quantisation are both nonlinear and signal-dependent. Nonlinear means that we cannot calculate their effect using standard mathematics. Signal-dependent means that even if we could calculate their effect, we would have to do so separately for every type of signal we expect, which is unrealistic. The effect of quantisation on digital signal processors can be analysed by mathematical models which regard the quantisation noise as if it were a source of random noise (Xie and Jabri, 1992).

Limited precision in artificial neural networks

Hardware implementations of neural networks use a representation of the network parameters with a limited accuracy because the use of high precision does not match the goal of developing fast and compact implementations. The main effect of limited precision on artificial neural networks is that certain word lengths result in weight update values (for backprop-type of networks) and activation values (for integrate-and-fire type of networks) which are below the quantisation step resulting in these values not changing (Forrester, 1993). For back-propagation type of networks learning is affected by reduction in weight precision. Moerland and Fiesler (1997) report that there is a critical weight precision value n below which training fails for backpropagation, since we are reliant on taking many small weight steps in training. However, once the network has been trained, we can reduce the weight accuracy during the recall phase without having too large an effect on performance (Hohfeld and Fahlman, 1992). This value n is problem dependent. Sackinger (1997) studied the degradation caused by finite precision within the 4 to 16 bits range on three pattern recognition algorithms - an optical character recogniser, a pen-based handwriting recogniser and a speech recogniser. His results showed that these algorithms can be implemented with 8-bit arithmetic while incurring a negligible loss in recognition accuracy.

Clearly, the earlier results on limited precision do not really supply definite guidance on what the effect of limited precision will be in stochastic resonance systems. It is not clear whether the reduction in precision will affect stochastic resonance, and if so, in what ways.

In this section we discussed the issues to do with mapping artificial neural networks onto hardware. The common thing between the different technologies for implementing artificial neural networks in hardware is the optimisation of three constraints: accuracy, speed and space. The next section will discuss FPGAs which is the development platform for this thesis.

Using FPGAs as the digital platform, this thesis investigates the effect on neuronal stochastic resonance of limited precision using a small LIF network model which is also

implemented using floating-point representation in Java for comparison purposes. Next we discuss the technology behind FPGAs.

2.6 Field Programmable Gate Arrays

A Field Programmable Gate Array is an array of configurable logic modules that communicate with one another and with user-programmable I/O-blocks via wires within routing channels. In an FPGA, existing wire resources that run in horizontal and vertical columns (routing channels) can be connected via programmable connection points.

Thus there are three types of programmable elements in an FPGA (see figure 2.4),

- *Logic cell modules* or *configurable logic blocks (CLBs)* implement most of the logic in FPGAs. The logic inside these modules connects to the programmable interconnect resources outside the block.
- *Connection points* between wires in routing channels connect logic cell modules with each other and the I/O-blocks.
- *I/O-blocks* provide the interface between the outside world and the internal logic.

In order to implement a system on an FPGA, the system to be implemented must first be split into pieces that are small enough to be placed in logic cell modules. Then the pieces are connected, both with each other as well as with input and output blocks. This process is called *place-and-route* and is done by software which the manufacturer of the FPGA supplies. The end result of this process is a configuration that can be loaded to the FPGA quickly.

The individual cells are interconnected by a matrix of wires and programmable switches. Each logic cell implements the logical function of gates. In most hardware that is used in computing today, the logical functions of gates are fixed and cannot be modified. In FPGAs, however, both the logic functions performed within the logic blocks and the connections between them can be altered by sending signals to the chip. These blocks are

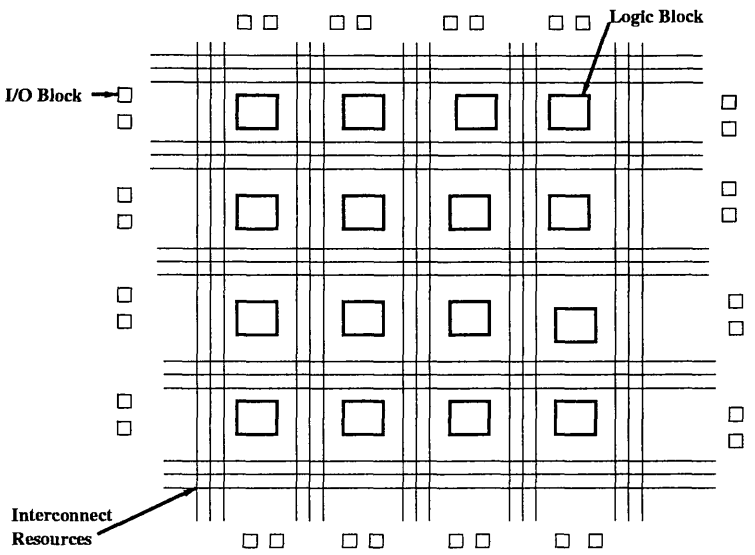

Figure 2.4: Structure of an FPGA

structurally similar to the gate arrays used in Application Specific Integrated Circuits (ASICs), but whereas standard gate arrays are configured during manufacture, the configurable logic blocks in FPGAs can be rewired and reprogrammed repeatedly, long after the integrated circuit has left the factory. A user's design is implemented by specifying the simple logic function for each cell and selectively closing the switches in the interconnect matrix. The array of logic cells and interconnect form a fabric of basic building blocks for logic circuits. Complex designs are created by combining these basic blocks to create the desired circuit.

Field programmable means that the FPGA's function is defined by the user's program (in the "field") rather than by the manufacturer of the device (in the "factory"). A typical integrated circuit performs a particular function defined at the time of manufacture. In contrast, the FPGAs function is defined by a program written by someone other than the device manufacturer. Depending on the particular device, the program is either 'burned' in permanently or semi-permanently as part of the board assembly process, or is loaded from an external memory each time the device is powered up. This user programmability gives the user access to complex integrated designs without the high engineering cost associated

with AISCs.

The key that has opened the door to reconfigurable computing is the design of new FPGAs that can be configured extremely quickly. The earliest FPGAs required several seconds or more to change their connections.

2.6.1 FPGA Architectures

A generic description of an FPGA is a programmable device with an internal array of logic blocks, surrounded by a ring of programmable input/output blocks, connected together via programmable interconnect. There are a wide variety of sub-architectures within this group. The secret to density and performance in these devices lies in the logic contained in their logic blocks and on the performance and efficiency of their routing architecture.

There are two primary classes of FPGA architectures: *coarse-grained*, and *fine-grained*. Coarse-grained architectures consist of fairly large but powerful configurable logic blocks. A single logic block often contains two or more look-up tables and two or more flip-flops and it is capable of adding or comparing two binary digits (Bostock, 1996). In a majority of these architectures, a four-input look-up table implements the actual logic.

The other architecture type is called fine-grained. In these devices, there is a large number of relatively simple logic blocks. The logic block usually contains either a two-input logic function or a 4-to-1 multiplexer and a flip-flop.

Another difference in architectures is the underlying process technology used to manufacture the device. Currently, the highest-density FPGAs are built using static memory (SRAM) technology, similar to microprocessors. The other common process technology is called anti-fuse, which is described subsequently, which has benefits for more plentiful programmable interconnect.

The architecture of a FPGA can be symmetric, asymmetric, or arranged as a sea-of-gates. These are illustrated in figure 2.5. For simplicity, inputs and outputs are not illustrated in the figure. More information about these and other types of architectures can be found elsewhere (Bostock, 1996; Brown and Rose, 1996). In a symmetric architecture,

the logic cell modules and channels form a symmetric pattern. An asymmetric structure, which is also called row-based structure, contains logic cell modules and channels placed in rows. The sea-of-gates structure contains many small logic cell modules.

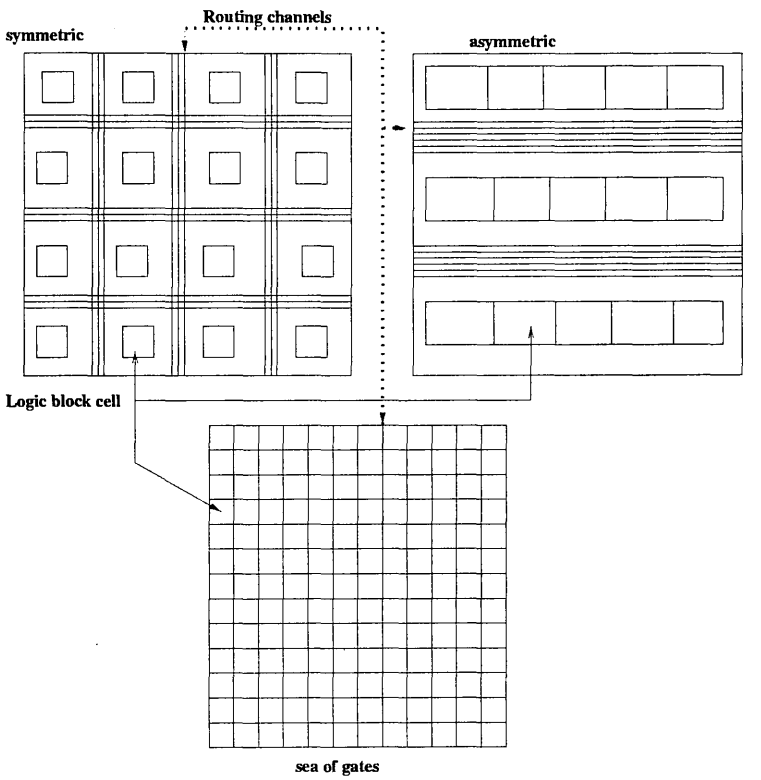

Figure 2.5: Different FPGA architectures

2.6.2 Static RAM Technology

In the Static RAM FPGA, programmable connections are made using transistors, transmission gates, or multiplexers that are controlled by SRAM cells. This is similar to the technology used in static RAM devices in microprocessors but with a few modifications. The RAM cells in a memory device are designed for the fastest possible read/write performance. The RAM cells in a programmable device are usually designed for stability instead of read/write performance. The advantage of this technology is that it allows fast

in-circuit reconfiguration. The major disadvantage is the size of the chip required by the RAM technology.

Because static memory is volatile, SRAM-based devices are "booted" after power-on. This makes them in-system programmable and re-programmable, even in real-time. As a result, SRAM-based FPGAs are common in reconfigurable computing applications where the device's function is dynamically changed. The configuration process typically requires only a few hundred milliseconds at most (Brown and Rose, 1996). Most SRAM-based devices can boot themselves automatically at power-on much like a microprocessor. Furthermore, most SRAM-based devices are designed to work with either standard byte-wide Programmable Read Only Memories (PROMs) or with sequential-access serial PROMs. They require some form of external configuration memory source. The configuration memory holds the program that defines how each of the logic blocks functions, which I/O blocks are inputs and outputs, and how the blocks are interconnected together. The FPGA either self-loads its configuration memory or an external processor downloads the memory into the FPGA. When self-loading, the FPGA addresses a standard byte-wide PROM much like a processor addresses its boot PROM or uses a special sequential-access serial PROM. When downloaded by a processor, the FPGA appears much like a standard microprocessor peripheral.

In this thesis we use one of Xilinx's SRAM based FPGAs which will be described in more detail in chapter 5.

2.6.3 Fuse and Anti-fuse

Fuse technology was the original programming technology for programmable logic. A fuse is a metal link (connection) that can be programmed by passing a current through the fusible link and vaporising the link. Fuse technology is one-time programmable (OTP).

Like the fuse, anti-fuse devices are also one-time programmable. Once programmed, they cannot be modified, but they also retain their program when the power is off. Anti-fuse devices are programmed in a device programmer either by the end user or by the

factory or distributor. The anti- part of anti-fuse comes from its programming method. Instead of breaking a metal connection by passing current through it, a link is grown to make a connection. They are usually physically quite small. Consequently, anti-fuse technology has benefits for creating programmable interconnect. However, they require large programming transistors on the device.

Having discussed the different technologies there is for FPGAs, next we look at how FPGAs are programmed. There are two different ways of looking at FPGAs. We can look at FPGAs from an engineering point of view in which case we design circuit diagrams using schematic tools and use the output of these to configure the FPGA. The other way of programming FPGAs is to design algorithms and then code the algorithms using hardware description/compilation languages to program the FPGA. In this thesis we will concentrate on the latter technique.

2.6.4 Hardware description languages and programming FPGAs

A hardware description language is similar to a software programming language, but instead of software a hardware description language describes hardware.

Hardware description languages have made hardware design significantly faster compared to traditional schematic approaches. In many cases, a high level description is enough because the low level details of a circuit can be synthesised from a behavioural (the behavioural description defines the structure and behaviour of a design) or Register-Transfer Level (RTL) level description. This is not always true however. Sometimes the only possibility is to use a detailed structural description, which in practice is a massive textual description of a schematic. These descriptions tend to be hard to maintain because it is not easy to see the structure from, say, several hundreds of lines of code written in some hardware description language.

In virtually every hardware description language (HDL) it is possible to use libraries of components containing all the details. However, it can be a very hard job to create these libraries.

VHDL

The VHSIC (Very High Speed Integrated Circuit) Hardware Description Language (VHDL) was developed by the US Department of Defence and standardised by the IEEE in 1987 (David, 1989; Cohen, 1999; Holmstrum, 2000). This language was intended to assist in the development, documentation and exchange of designs. Since its development, VHDL has been widely accepted by the design community as a description language and they have developed a wide range of tools for capture (using graphical front-ends), simulation, debugging, verification and synthesis. VHDL supports gate-level and behavioural level descriptions as well as RTL descriptions. Descriptions written in VHDL can be synthesised to almost any FPGA on the market today. The drawback for VHDL, however, is that it has a complicated structure.

Hardware compilation and the Handel-C language

Hardware compilation is a powerful tool for hardware/software codesign. Conventionally, it is common to draft a system using a high level programming language, like C, and then realise the hardware part of that system using a hardware description language like VHDL. Hardware compilation using Handel-C simplifies the design process, since it provides the convenience of a C-like high-level programming language to generate the hardware as well.

Handel-C is a programming language designed for compiling programs into hardware images of FPGAs or ASICs. It is basically a small subset of C, extended with a few constructs for configuring the hardware device and to support the generation of efficient hardware. It comprises all common expressions necessary to describe complex algorithms, but lacks processor-oriented features like pointers and floating point arithmetic. The programs are mapped into hardware at the netlist level (A netlist file describes how the individual components of a circuit are connected together). It must be noted though that Handel-C itself does not offer support for pre-placing components, but this can be achieved by using macros (Lawrence, 1997). The current Handel-C compiler v3.1 supports most Xilinx products.

Opposed to other approaches of high-level language hardware design, which actually use C to describe the behaviour and simply translate it into netlist, Handel-C targets hardware directly, and provides a few hardware optimising features. A big advantage, compared to the C-translators, is that variables and constants can be given certain widths, as small as one bit. When using C as the describing language, the smallest integer is possibly 8 bits, if not 16 bits, which means that one wastes at least 7 bits when declaring a simple flag. Also, Handel-C provides bit manipulation operators and the possibility of parallel processing of single statements or whole modules. This can not be realised with other approaches based on a sequential language.

Language comparisons

There exist several hardware description languages, for instance Handel-C (Aubury et al. (1996)) and VHDL. The language Handel-C has a simple syntax and it offers a convenient way to describe hardware on a high level (Holmstrom and Sere, 1998). VHDL is supported by a huge number of tools which support almost any type of FPGA. Signals in Handel-C are different from signals in VHDL; they are assigned to immediately, and only hold their value for one clock cycle. The complexity of VHDL, on the other hand, led us to believe that a simpler approach will provide a better foundation on which to build the neural models. Between Handel-C and VHDL, VHDL is the one that can be used for practically any FPGA on the market today. It is an industry standard and is supported by a wealth of tools. Compared to Handel-C, however, VHDL has a rather complicated syntax and it is also too verbose.

Comparing Handel-C and VHDL shows that the aims of these languages are quite different. VHDL is designed for hardware engineers who want to create sophisticated circuits. It provides all the constructs necessary to craft complex, tailor made hardware designs. By choosing the right elements and language constructs in the right order, the specialist can specify every single gate or flip-flop built and manipulate the propagation delays of signals throughout the system. This means that VHDL expects that the developer

knows about gate-level hardware and requires him or her to continuously think about the gate-level effects of every single code sequence. This quite easily distracts from the actual algorithm or functional subject and ties up a lot of the designer's attention.

In contrast, Handel-C is not designed to be a hardware description language, but a high-level programming language with hardware output. It does not provide highly specialised hardware features and allows only the design of digital, synchronous circuits. Instead of trying to cover all design particularities, its focus is on prototyping and optimising at the algorithm level. The low-level problems are hidden completely, all the gate-level decisions and optimisation are done by the compiler so that the programmer can focus his or her mind on the task he or she wants to implement. As a result, hardware design using Handel-C is more like programming than hardware engineering, and in fact, this language is developed for programmers who have no hardware knowledge at all.

To sum up, hardware design with Handel-C is very much like programming. Unlike with hardware description languages, the designer is not confronted with gate-level problems like fan-in and fan-out or choosing the appropriate type of gates or registers to be used. Apart from freeing the programmer's mind from low-level decisions, it is much faster and more convenient to describe the system's desired behaviour at the algorithmic level. The fast compilation, combined with a high-level simulator, allows one to try out several implementation strategies within a very short time. However, Handel-C is not without its own weaknesses. There are not many synthesis tools for Handel-C which means it is only usable on a select set of FPGAs. The main inefficiency for Handel-C is that the existing synthesis tool requires a great deal of place and route optimisation.

2.6.5 Applications

FPGAs have gained acceptance over the past decade because users can apply them to a wide range of applications. Algorithms that contain a high degree of binary-level operations, for example cipher and integer calculations are well suited for FPGAs, while multiplication with high precision and floating-point operations are not well suited.

A typical traditional FPGA application is prototyping designs to be implemented in gate arrays using one or more FPGAs. Another application is the emulation of entire large hardware systems via the use of many interconnected FPGAs. After creating and testing the prototype, the FPGA can either be used as such in the target system or be replaced by an ASIC. The FPGA is the best solution for small systems where delivery time and price are more important than having the fastest possible circuit. Producing an ASIC is a complicated process that usually only a third-party vendor can handle. Delivery time for ASICs is long compared to the time for configuring an FPGA. Because of the high costs, ASICs are a more expensive alternative than FPGAs except for large quantities. Another benefit of using FPGAs is the possibility to correct possible design flaws later on by simply reconfiguring the circuit as opposed to ASICs that cannot be corrected once it is produced. Despite verification efforts, there is no guarantee that a design is error-free.

It used to be the case that the FPGAs in a system were for the most part configured only when the system was started up. However, FPGAs can in principle be reconfigured an infinite number of times. Nowadays the full flexibility of the FPGA is more commonly used than before. FPGAs are now quite often used together with microprocessors in so-called reconfigurable (custom) computing systems. FPGAs form the core of reconfigurable computing systems.

Reconfigurable computing

Reconfigurable computing systems can be defined as FPGAs combined with external memories and processors (van den Bout et al., 1992). The FPGA is the core component of such a system (Celoxica, 2000). Reconfigurable systems can also be defined as computing platforms whose architectures can be modified by software to suit the application at hand. Computing devices can make use of configurable elements in many different ways. The least demanding technique is to switch between functions on command — the hardware equivalent of quitting one program and then running another. Slow reconfiguration, on the order of several seconds, may well be acceptable in such an application. Fast program-

ming times permit dynamic design swapping: a single FPGA performs a series of tasks in rapid succession, reconfiguring itself between each one. Such designs operate the chip in a time-sharing mode and swap between successive configurations so rapidly that it appears the FPGA is performing all its functions at once. A good example of this approach was reported by Villasenor and Mangione-Smith (1997). They built a single-chip video transmission system that reconfigures itself four times per frame. It thus requires only a quarter of the hardware that would be needed for a fixed ASIC. The FPGA first stores an incoming video signal in memory, then applies two different image-processing transformations and finally transforms itself into a modem to send the signal onward.

The most challenging and potentially most powerful form of reconfigurable computing involves the hardware reconfiguring itself on the fly as it executes a task, refining its own programming for improved performance. An image-recognition system might tune itself in response to a tentative identification of the object it is looking at: if an image contained a car or a truck, parts of the circuitry originally intended for tracking high-speed aircraft or slow-moving people could be reconfigured to focus instead on land vehicles. FPGAs offer many possibilities for reconfigurable computing.

Disadvantages of FPGAs

Not all computations can be implemented efficiently with today's FPGAs: they are well suited to algorithms composed of bit-level operations, such as pattern matching and integer arithmetic, but they are ill-suited to certain types of numeric operations, such as high-precision multiplication or floating-point calculations. Dedicated multiplier circuits such as those used in microprocessor and digital signal chips can be optimised to perform more efficiently than multiplier circuits constructed from configurable logic blocks in an FPGA. Furthermore, FPGAs currently provide very little on chip memory for storage of immediate results in computations; thus, many configurable computing applications require large external memories. The transfer of data to and from the FPGA increases power consumption and may slow down the computations. This high power consumption

has kept FPGAs out of the ever-growing hand-held devices market.

2.7 Summary

The main purpose of this chapter was to give some background knowledge about the neural modelling, mapping artificial neural networks on to hardware, and a description of FPGAs which is the digital hardware platform on which some the simulations will be done.

In section 2.1 of this chapter we discussed the basic physiology of neurons as consisting of a dendritic arbor, a soma and the axon, and concluded that even that basic physiology is still too complicated to model in its entirety in software, let alone in hardware. We looked at the different levels at which neurons are modelled. A model can be single-channel plus morphology or compartmental (in which ion channels are lumped together) or single compartment (point neuron). We concluded that the more detailed models like conductance-based models and compartmental models are more suitable for modelling single neurons in detail while simplified models of the threshold-fire type are more tailored to modelling large networks of neurons. We discussed in detail the LIF neuron model which provides a neuronal model of complexity and sophistication at a level between that of McCulloch-Pitts and Perceptrons, and biologically realistic multiple models. We conclude that the LIF neuron model is good for modelling stochastic resonance.

We also noted that the natural parallel nature of artificial neural networks renders them suitable to hardware implementation. Unfortunately the way neural network models were original designed does not make their realisation on hardware an easy thing to do. Despite there being many technologies for realising artificial neural networks in hardware, they all have the common denominator of mapping neural networks algorithms onto reliable, compact and fast hardware. We also noted that mapping artificial neural networks onto hardware boils down to optimising four main constraints: accuracy, space, cost and processing speed. The different technologies are just a trade-off of these three constraints. For instance analogue implementations are very efficient in terms of chip area and processing speed, but suffer from limited accuracy of network components.

This chapter also discussed FPGAs as one of the development platforms for one of the models we will use in this thesis. We noted that FPGAs are the best solution for small systems where delivery time and price are more important than having the fastest circuit possible. FPGAs are generally preferred to ASICs which is the more expensive alternative type because they offer the possibility of correcting possible design flaws later by simply reconfiguring the circuit, as compared with ASICs that cannot be corrected once they are produced.

In the next chapter we discuss the concept of stochastic resonance and how it connects with neuronal modelling.

Chapter 3

Stochastic Resonance

Periodically modulated stochastic processes have been studied intensely over the last two decades under the paradigm of stochastic resonance (SR). The concept of stochastic resonance has met with particular attention in the neurosciences. The brain achieves an extraordinarily good signal processing performance in the presence of noise from a wide range of sources, ranging from stochastic membrane channel openings on a molecular level, to highly irregular firing patterns of individual neurons and distracting stimuli in perception. The improvement of signal transduction by noise on all levels has now been demonstrated experimentally Bezrukov and Vodyanoy (1997a); Douglas et al. (1993); Stemmler et al. (1995) and may help to improve cochlear implants for the deaf as shown by the work of Morse and Evans (1996). Recently, the behavioural relevance of stochastic resonance has been reported by Usher and Feingold (2000), underlining the importance of stochastic resonance.

In this chapter we start by describing stochastic resonance and then focus on stochastic resonance in neuronal systems using the leaky integrate-and-fire neuron model. Section 3.1 defines stochastic resonance and the history of stochastic resonance is discussed in section 3.2. The measures used to quantify stochastic resonance are given in section 3.3 and the connection between stochastic resonance and neuronal modelling is introduced in section 3.4. Sections 3.5 and 3.6 discuss some of the tools for modelling stochastic resonance and

the possible benefits of stochastic resonance respectively. The chapter ends with some concluding remarks.

3.1 What is Stochastic Resonance?

Stochastic resonance describes a phenomenon that is manifest in non-linear systems whereby generally feeble input information (such as a weak signal) can be amplified and optimised by the assistance of noise (Gammaitoni et al., 1998).

The concept of *stochastic resonance* is best described by way of an analogy. One imagines a ball sitting in one of two wells (figure 3.1) with a gentle force rocking the whole system to and fro. This force corresponds to some weak periodic signal. Under its influence, the ball rolls around in the bottom of one well. If the ball's movements are detectable only when it jumps from one well to another, this weak periodic force will remain hidden. One might think that adding noise to the system by shaking the container, for example, would be likely to mask the rocking motion further. In fact it does the opposite. The weak force coupled with noise can, on occasion, give the ball enough energy to surmount the barrier between the two wells. Over time, the ball appears to jump back and forth apparently at random. The theory of stochastic resonance relies on the fact that these jumps are not entirely unpredictable: the chance that the ball switches wells from one moment to the next is far greater if the weak periodic force is at its peak (Moss and Wiesenfeld, 1995). The timing of the jumps reveals something about the periodicity of the driving force. The appearance of stochastic resonance in general can roughly be explained as follows: the system in question will generate discernible output every time its internal state surmounts a barrier. If the deterministic input signal is too weak to induce crossings, the system will be silent in the absence of noise, and weak noise will induce only rare, incoherent crossings. Strong noise on the other hand, will induce frequent but random transitions. At an intermediate noise intensity though, the rate of noise-induced crossings will coincide with the timescale set by the input signal, yielding a coherent output signal. Stochastic resonance is thus a cooperative effect between signal and noise (Plesser, 1999). The effect

requires three basic ingredients (Gammaitoni et al., 1998): (i) an energetic barrier or, more generally, a form of threshold; (ii) a weak coherent input (such as a weak periodic signal); and (iii) a source of noise that is inherent in the system, or that adds to the coherent input. Given these features, the response of the system undergoes resonance-like behaviour as a function of the noise level; hence the name stochastic resonance.

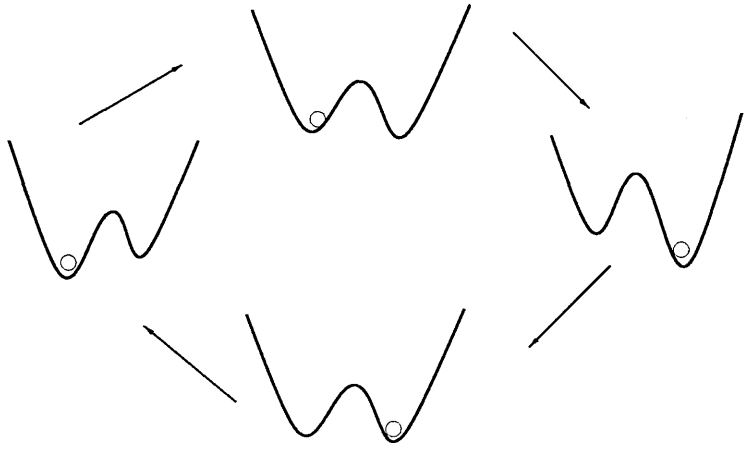

Figure 3.1: Stochastic resonance in double wells. Adopted from (Gammaitoni et al., 1998). The ball will start in one well. If we apply a weak periodic force which lowers and raises the wells sinusoidally, the ball will be rocking within one well. The lowering of the wells is similar to rocking the system. If we couple this force with shaking the system then from time to time the ball will hop from one well to the other depending on the strength of the shaking. The shaking of the system is a way of introducing noise to the system. The synchronicity of these hops depends on the size of the shaking. Too much shaking will result in the ball hopping between wells in a haphazard manner, too gentle shaking results in rare and incoherent hopping. There is an optimal amount of shaking which synchronises the hopping of the ball with the periodicity of the lowering of the wells. Basically this system depicts stochastic resonance in that: no shaking means no hopping, too much shaking results in frequent but random hopping, and optimal shaking synchronises the hopping with the periodicity of the rocking force.

Over time, the original notion of stochastic resonance has widened to include a number

of different mechanisms (Gammaitoni et al., 1998). The unifying feature is that noise can improve a system's sensitivity to weak signals, such as periodic inputs. One measure of sensitivity is the output signal-to-noise ratio (SNR) and a signature for stochastic resonance is when this ratio passes through a maximum as a function of the applied noise intensity (Barbi et al., 2000). The earliest research focused on bistable systems, but later research includes excitable systems that have a single rest-state (Longtin, 1993; Collins et al., 1995a; Marino et al., 2002). The study of these systems was motivated by the potential relevance of stochastic resonance to neuronal dynamics.

Stochastic resonance should not be confused with 'dithering' also known as stochastic linearisation, a technique where in periodic or random forcing, noise is intentionally introduced to overcome regions of 'dead' dynamical behaviour in self regulating systems (French et al., 1972; Gammaitoni, 1995; Chialvo et al., 1997). An example of dithering is the linearisation of the firing rate $r(I)$ — current I curve in an LIF neuron model. If we take an LIF neuron with constant input I without noise and plot the firing rate r as a function of I, then $r(I) = 0$ for $I < I^*$, where I^* is that input strength where the membrane potential first reaches threshold, i.e. V_{th}/R, in figure 2.2 B. At I^*, $r(I)$ has infinite slope, and as I increases, $r(I)$ rises, while the slope of $r(I)$ goes down; $r(I)$ looks like a log curve (see figure 3.2).

If we plot the same figure, but for various noise intensities: the part of $r(I)$ around I^* will become more and more linear as the noise increases (see figure 3.2). This is essentially linearisation by noise: the input-output relation of the neuron becomes linear over a range of inputs. This is good for signal transmission, because if signal parameters are such that the input is only within the range where $r(I)$ is (roughly) linear, one has linear signal transmission, i.e. the signal is transmitted undistorted. Dithering is also used in electronic circuits, e.g. CD-players: the digital-analog converter should produce a smooth analog output from a digital "staircase" input. To this end, a little bit of noise is added to the output to smear out the steps.

In this section stochastic resonance was described and we drew a distinction between stochastic resonance and linearisation by noise. In the next section we will give a brief

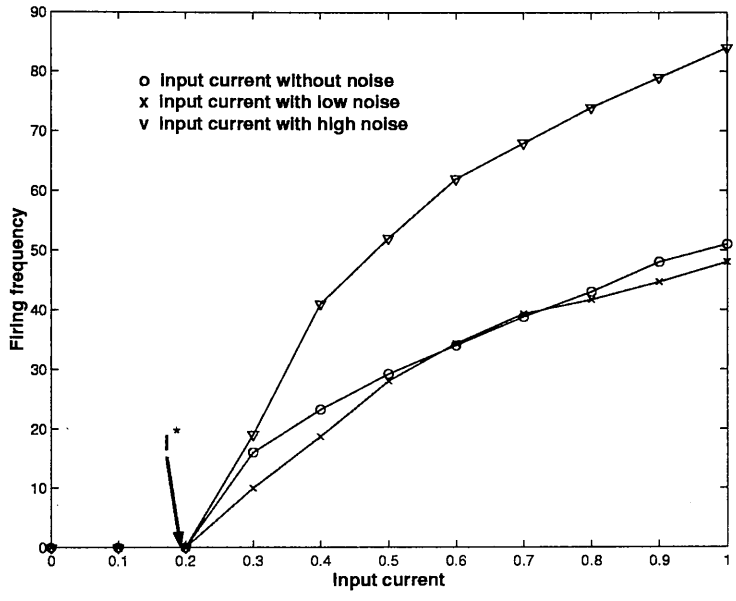

Figure 3.2: Noise linearisation. Without noise the firing rate above I^* is not linear (o). With the introduction of noise, the firing frequency immediately above I^* is linearised (+ and v).

discussion of the history of stochastic resonance.

3.2 History of stochastic resonance

The term stochastic resonance first appeared in 1981 to describe "the cooperative effect between internal mechanism and the external periodic forcing" in some nonlinear dynamical system (Benzi et al., 1981). This concept was originally put forward to explain the periodically recurrent ice ages on Earth. Statistical analysis of ice volume variations shows that ice ages have a periodicity of 10^5 years. The only other variable with that time scale in Earth dynamics is the modulation period of its orbit eccentricity, with ensuing variations in solar radiations on the earth surface of 0.1 percent.

It was suggested that short term climate fluctuations, such as annual fluctuations in solar radiations, could amplify the effects of periodical variations of the Earth's orbit.

In the model of Benzi et al. (1981) the Earth's climate was represented by a double-well potential, with one well corresponding to an ice age period, and the other a warmer climate. The small modulation of the earth's orbital eccentricity is represented by a weak periodic forcing. The periodic variations of the Earth's orbit were too small to cause well-hopping. The short term climate fluctuations were modelled by Gaussian white noise. If the noise is properly tuned, synchronised hopping between the cold and warm climate could significantly enhance the response of the earth's climate to the weak perturbations caused by the Earth's orbital eccentricity, according to arguments by Benzi et al. (1981, 1982). Benzi et al. coined the term stochastic resonance because the driven system only resonates with the forcing system in the presence of noise. Short-term climatic fluctuations, such as the annual fluctuations in solar radiation, are modelled by Gaussian white noise. It is interesting to note that this paradoxical explanation of the periodicity of the Earth ice ages remains today unconfirmed and is still a subject of debate, whereas the general phenomenon of stochastic resonance has been firmly established in a large number of physical systems.

Experimentally, stochastic resonance was first demonstrated with a noise driven electronic circuit known as a Schmitt trigger (Fauve and Heslot, 1983); this work was also the first to characterise the phenomenon in terms of signal-to-noise ratio (SNR). It took several years before the interest of physicists (who have done most of the theoretical analysis) ignited, sparked by the demonstration of stochastic resonance in a bistable ring-laser experiment (Weisenfeld and Moss, 1995). Stochastic resonance has been reported in a wide variety of physical systems and the general theory is well in hand. Now stochastic resonance has crossed disciplinary boundaries: its role in sensory biology has been explored in experiments on single crayfish neurons (Douglas et al., 1993), cat visual cortex (Anderson et al., 2000), cricket cercal sensory system (Levin and Miller, 1996), human memory retrieval (Usher and Feingold, 2000) and perspective brain function by experiments on people's ability to resolve ambiguous figures (Riani and Simonotto, 1994). Today, we know that stochastic resonance is even more general than the bistable picture implies. Even simpler systems, including those with a single potential well and integrate-and-fire dynamics can exhibit stochastic resonance or stochastic resonance-like properties. The nonlinearity

in integrate-and-fire models is simply the on/off nature of the output. Stochastic resonance is also reported to occur in non-dynamical and threshold-free systems (Bezrukov and Vodyanoy, 1997b).

Under the widened notion of stochastic resonance, the first non-bistable systems discussed were excitable systems (Longtin, 1993). In contrast to bistable systems, excitable systems have only one stable state (the rest state), but possess a threshold to an excited state which is not stable and decays after relatively long time (in comparison to the relaxation rate of small perturbations around the stable state) to the rest state. Soon afterwards threshold detectors were discovered as a class of simple systems exhibiting stochastic resonance (Jung, 1994; Wiesenfeld et al., 1994).

The development of stochastic resonance took a large leap forward when its potential relevance for neurophysiological processes was recognised. Longtin et al. (1991) observed that interspike interval histograms of periodically stimulated neurons exhibit a remarkable resemblance to residence-time distributions of periodically driven bistable systems. The framework developed for excitable and threshold dynamical systems has paved the way for stochastic resonance applications in neurophysiology.

In this section we gave a brief historical perspective of how the term stochastic resonance came into being. We also pointed out that the problem which gave rise to the phenomenon of stochastic resonance is still unresolved but the concept of stochastic resonance is flourishing in neurophysiological circles. In the next section we will present some of the measures which are used to quantify stochastic resonance.

3.3 Quantifying stochastic resonance in neurons

Having discussed the main physical ideas of stochastic resonance in the preceding section, we next define the observables that actually quantify the effect. There is no consensus in the literature on how to measure stochastic resonance but signal-to-noise ratio seems to be the preferred choice for periodic stochastic resonance. The observables should be physically motivated, easily measurable, and/or be of technical relevance (Gammaitoni

et al., 1998). From these functions, a measure of the coherence of the system's output signal with the input signal can be obtained. Monitoring the changes in the coherence measure as the relevant parameters (signal frequency and noise strength) are changed will allow us to establish the existence of stochastic resonance. Stochastic resonance is exhibited by showing that the coherence measure goes through a maximum as the input noise is increased. For the double well systems, the only events that transmit some information about the weak periodic forcing are the well to well transitions, not the intra-well motions. Therefore, more generally, stochastic resonance will be quantified by measuring the changes of state of a system, not the fluctuations within one state. The measures to be discussed here are power spectrum, interspike interval histograms and signal-to-noise ratio.

3.3.1 Power spectrum

In the seminal paper by Benzi et al. (1981), stochastic resonance was quantified by the intensity of a peak in the power spectrum. Figure 3.3 shows how the height of the peak at 20 Hz starts off low for low noise, rises to a high value for optimal noise and falls back to a low value again as the high noise dominates. Observables based on power spectrum are indeed very convenient in theory and experiment, since they have immediate intuitive meaning and are readily measurable. The details of how to obtain the power spectrum for spike trains are presented in chapter 4.

3.3.2 Interspike interval histograms

Although the power spectrum is the most widely used measure for comparing the output and the input, it is not the only possibility. An alternative is the residence-time probability distribution more familiarly known to experimental neuro-biologists as the interspike interval histogram (ISIH). This measure is composed of a set of peaks which are widely spread for very low noise values but become more coherent for larger noise intensity (see figure 3.4). This qualitative structure of the histogram is very common, and is not in itself a signature for stochastic resonance. Rather, stochastic resonance is identified with

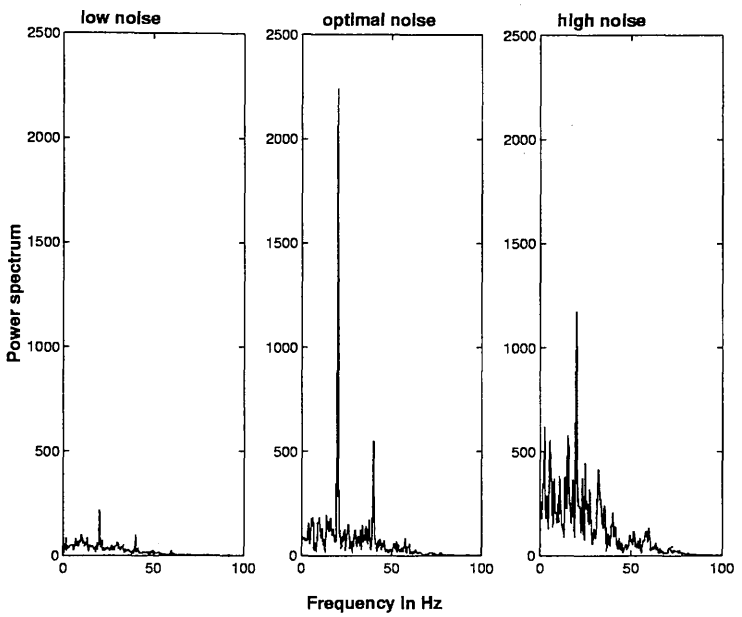

Figure 3.3: Power spectrum from a spike train of a leaky integrate-and-fire neuron receiving a 20 Hz subthreshold periodic signal plus Gaussian white noise at different amplitudes. The peak at 20 Hz is small for small noise, highest for optimal noise and small again for high noise. This is a sample result from simulations to be discussed in chapter 5.

the more particular behaviour shown in figure 3.4 where the amplitudes of the lower-order peaks pass through a maximum with increasing noise intensity. It must be pointed out that the typical shape of ISIHs under stochastic resonance is not unique to stochastic resonance. Similar histograms from both neuron models and actual biological preparations show stochastic resonance or at least a strong connection between the ability of a sensory neuron to transmit coherent information and its internal and external noise (Gammaitoni et al., 1998). Quantifiers that are based on the interval distributions emphasise the synchronisation aspect of stochastic resonance. The resemblance of interspike interval histograms and residence-time distributions of noise driven bistable systems, connected stochastic resonance research with neuronal processes.

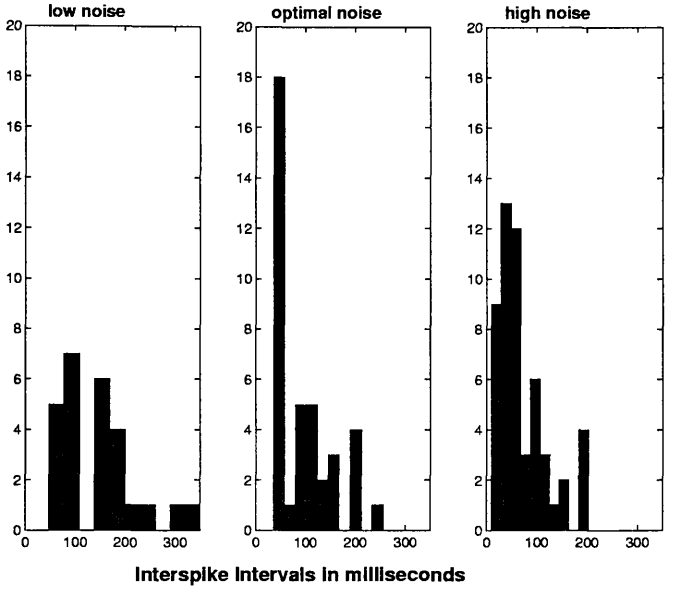

Figure 3.4: Interspike interval histograms for spike trains from a leaky integrate-and-fire neuron model receiving a 20 Hz subthreshold periodic signal plus Gaussian white noise of varying amplitude. This is a sample result from simulations to be discussed in chapter 5.

3.3.3 Signal-to-noise ratio

The most common way of quantifying stochastic resonance is through the SNR. This is readily obtained from the output spike train by forming the power spectrum, which measures the frequency content of a time series. The signature of stochastic resonance is that this SNR is zero for zero added noise (that is, no noise implies no switching or threshold crossings, hence no output), and rises sharply to a maximum at some optimal noise intensity, and decreases gradually for larger noise intensity as randomisation overrides the cooperative effect (Gammaitoni et al., 1998). The detailed shape of this curve depends on the input signal frequency and the other system parameters (see section 6.6 in chapter 6 for a discussion of the effect of neuronal parameters on stochastic resonance). A typical SNR plot illustrating the stochastic resonance effect is shown in figure 3.5.

In this work, the phenomenon of stochastic resonance for periodic input will be exhibited by showing a non-monotonic variation of the SNR as a function of the input noise. The use

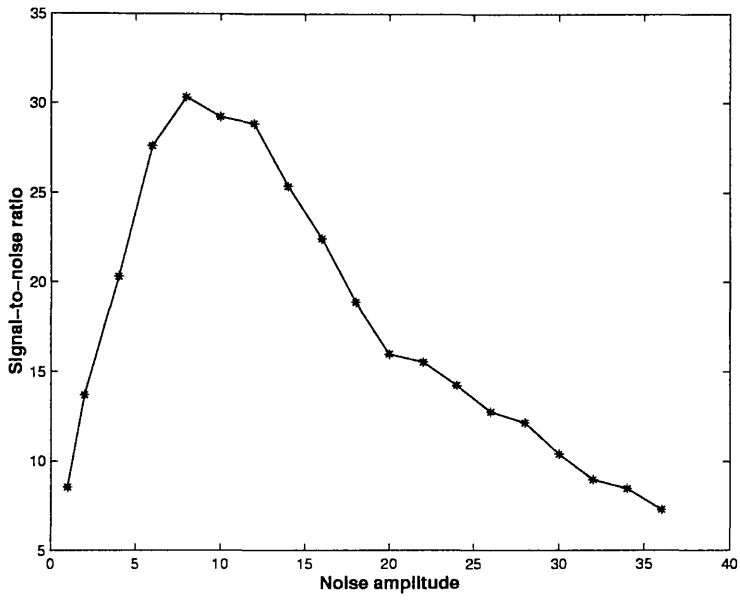

Figure 3.5: SNR plot from the spike trains of a leaky integrate-and-fire neuron model receiving a 20 Hz subthreshold periodic signal plus Gaussian white noise of varying amplitudes. This is a sample result from simulations to be discussed in chapter 5.

of SNR to measure periodic stochastic resonance in this thesis was motivated by the work of Stemmler (1996). Using the spike count over a fixed time window as an estimate of the firing rate, Stemmler (1996) derived an expression for the Fisher information for an LIF neuron and discovered that a strikingly similar formula holds for the spike output signal-to-noise ratio of an LIF neuron in response to a sinusoidal input. And he went on to show that the two expressions become identical in the limit of low firing rates. This means that SNR is a credible substitute for entropy based methods (Godivier and Chapeau-Blondeau, 1998; Heneghan et al., 1996) for quantifying stochastic resonance especially for periodic input. In conclusion Stemmler argued that despite the close similarity between the formulae for SNR and the entropy based Fisher information, the more fundamental definition of stochastic resonance in neuronal models is in terms of the Fisher information.

3.3.4 Other measures

In the case of non-periodic inputs (which give rise to aperiodic stochastic resonance (Collins et al., 1996a), not considered in this thesis), an input/output cross-correlation measure is used (Chialvo et al., 1997). Recently, studies have been carried out to quantify stochastic resonance from an information theory point of view (Heneghan et al., 1996; Godiver and Chapeau-Blondeau, 1996; Robinson et al., 1998). These two measures will not be discussed further because they will not be used in this thesis.

3.4 Aperiodic vs periodic stochastic resonance

The input signal to a simple model such as the leaky integrate-and-fire unit can be constant, periodic or arbitrarily varying in time. Regardless of the nature of the signal, the relevant information quantities have the same asymptotic scaling properties as a function of the noise amplitude (Stemmler, 1996). Whether we wish to determine the mutual information between input and spike output, the SNR for the sinusoidal input, or the probability of correctly detecting a constant signal within a limited time—for subthreshold inputs, these quantities are given by the class of equations for stochastic resonance (Stemmler, 1996):

$$\mathrm{SNR}, d', I, P(.) \propto \sigma^{-\alpha} \exp(-\beta \frac{\Delta^2}{\sigma^2}) \tag{3.1}$$

where

- SNR is the signal-to-noise ratio;
- d' is the Mahalonobis distance (a normalised measure of how far apart two probability distributions are);
- I is the mutual information;
- P(.) is the probability of detecting a signal within a limited time;
- σ is the standard deviation of the input noise;

- Δ is the effective input current threshold;
- The exponents α and β depend on the quantity on the left hand side and may differ for different models of spiking neurons.

This result shows that results for periodic stochastic resonance can be extended to aperiodic stochastic resonance without any loss in generality. In keeping with our desire to keep the model used in this thesis simple we will only model periodic stochastic resonance.

Power spectrum, interspike interval histograms and SNR were identified as the main quantifiers for stochastic resonance. In the next section we discuss the connection between stochastic resonance and neuronal modelling.

3.5 Stochastic resonance in neuronal models

In this section we discuss the realisation of stochastic resonance in neuronal models. We note that stochastic resonance in neuronal models was motivated by stochastic resonance in threshold detectors, hence we will discuss stochastic resonance in threshold detectors first.

3.5.1 Threshold detector

Even though stochastic resonance was first put forward in bistable systems, it was later demonstrated in a non-dynamical threshold system, with the benefit of greatly simplifying the problem by removing all dynamical complications. The simplest explanation of stochastic resonance is found in threshold detectors.

Consider a threshold detector that outputs 1 if the input is larger than a threshold value V_{th} and 0 otherwise, as illustrated in figure 3.6. In this case, the minimum requirements for a system to exhibit stochastic resonance are (Moss et al., 1994; Hess and Albano, 1998):

- a form of threshold
- a subthreshold coherent input

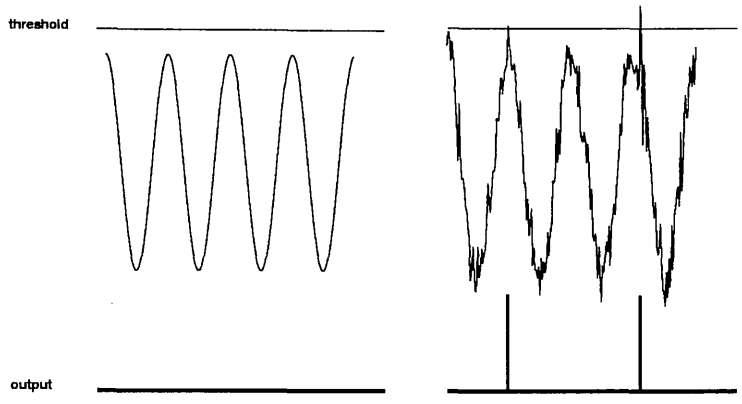

Figure 3.6: Threshold detector: Left: Subthreshold input. Right: noisy threshold-reaching input. Adapted from Hohn (2000)

- a source of noise intrinsic and/or extrinsic to the system that adds to the input signal.

For a threshold detector, in the absence of noise, the subthreshold input signal is by definition too weak to alone exceed the threshold, so that there will be no threshold crossings and the output SNR will be zero. Adding noise, the threshold is crossed at times more or less correlated with the weak input signal (as shown in figure 3.6), and the SNR increases. When the noise intensity becomes too large, the input crosses the threshold randomly and the SNR is small again. Consequently there exists an optimum non-zero noise value for which the SNR is maximum. Stochastic resonance in threshold detectors, and more generally in a threshold non-linearity, is well understood. Simple cartoon versions of excitable systems known as threshold detectors have been studied intensively because they are accessible to full analytical treatment. In these models, the detector responds (by sending out a pulse, for example) whenever the sum of the signal and the noise crosses a threshold.

3.5.2 Stochastic resonance in neurons

The basic concepts of stochastic resonance and a mathematical description of neurons having been introduced in the previous sections and chapter, a more detailed presentation

of stochastic resonance in neurons can now be given.

The application of stochastic resonance to neurons first arose as a consequence of the similarity between interspike interval of periodically stimulated neurons and the residence-time distributions of periodically driven bistable systems (Longtin, 1993).

Even though the relationship between neurons and stochastic resonance was first based on bistable dynamics, the firing state of a neuron is not a stable state and a description as a nonlinear non-bistable system is more appropriate. From equation 2.7, neurons can be modelled as threshold detectors with a deterministic reset after firing. This means that neurons can be described as having one stable fixed point, their resting value, and one unstable fixed point, their threshold value. Such a description was first put forward by Wiesenfeld et al. (1994) and Moss et al. (1994) and was coined "stochastic resonance on a circle" due to the way this concept was illustrated (see figure 3.7).

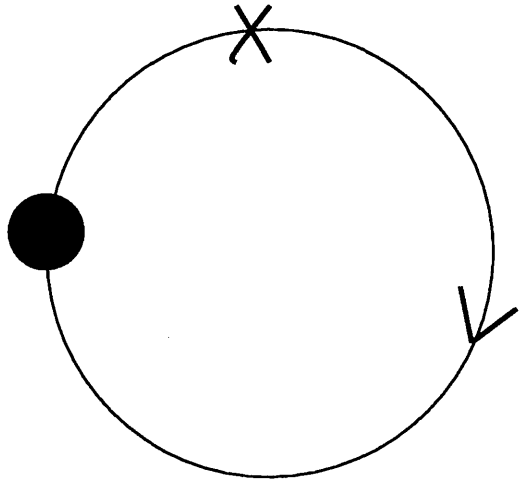

Figure 3.7: Representation of stochastic resonance for a monostable system (after Wiesenfeld et al. (1994)). The dot represents the stable state, and the cross the firing state. The state point is most of the time in the neighbourhood of the dot. When the state point passes the cross, e.g. when the threshold is crossed, the state point goes deterministically back to the dot following the arrow.

The reset mechanism makes the study of such nonlinear systems much more complicated

than the simple threshold detector presented in the previous section.

Stochastic resonance in neurons has been the subject of a considerable number of studies using neuron models with different levels of complexity. They range from all-numerical studies of networks of real neurons requiring computer intensive simulations to analytical studies of single integrate-and-fire neurons. The FitzHugh-Nagumo (FHN) neuron model has been especially widely studied in the context of stochastic resonance, by using analytical approximations (Wiesenfeld et al., 1994) , electronic models (Moss et al., 1993) and numerical simulations (Pei et al., 1995; Kanamaru et al., 1999).

Stochastic resonance in an LIF

We can add noise to the LIF neuron by adding an extra term to the LIF equation as we suggested in the noise models in section 2.4. In between two spikes, the membrane potential of the LIF is governed by:

$$\tau \frac{dV}{dt} = -V(t) + I(t) + \sigma\xi(t) \tag{3.2}$$

where τ is the membrane time constant and $\xi(t)$ is Gaussian white noise which is from sources that are uncorrelated to $I(t)$ and is additive to the input $I(t)$ and σ is the noise amplitude. As the potential reaches threshold V_{th}, a spike is recorded and the potential is reset to $V(t) = V_0$. The evolution of the membrane potential $V(t)$ is equivalent to an Ornstein-Uhlenbeck process with drift $I(t)$ and absorbing boundary $V(t) = V_{th}$ (Tuckwell, 1988b). The output of the neuron is modelled as a sequence of delta pulses $f(t) = \sum_k \delta(t - t_k)$ at the times of threshold crossings $t_k = \{t|V(t) = V_{th}\}$. This spike train is a stochastic point process, specified entirely by the spike times t_k. For quite some time this biologically plausible model escaped a rigorous analysis. The main reason for that was problems caused by the reset after each spike. Because of this, the analysis has to be done for each interspike interval separately and then the pieces put together to obtain the spike train as a whole. The signal processing performance of the neuron is judged by the SNR of the output spike train (see section 4.6 for the computation of the SNR for a spike train). The SNR is maximal at an optimal noise amplitude for fixed stimulus frequency and at a

resonance frequency for fixed noise amplitude. The latter resonance is a result of a time scale matching between stimulus and membrane time constant.

An important step in the analysis of integrate-and-fire neurons was a study by Bulsara et al. (1996) in which the first-passage time density, i.e. the probability density for the membrane potential to cross the threshold, was first approximated using the so called method of images (see Tuckwell (1988b) for a discussion). Plesser and Geisel (1999) and Burkitt and Clark (2000) also analysed the first-passage time density for a leaky integrate-and-fire neuron model using a Markov Chain method. People have to resort to numerical solutions for the first-passage time problem for an LIF neuron because there is no known analytical solution.

Stochastic resonance has also been widely studied in networks of neuronal models as evidenced by the works of Gluckman et al. (1996) and Shimokawa et al. (1999). We will be looking at stochastic resonance in a network of LIF neurons in chapters 5 and 6.

The connection between neuronal modelling and stochastic resonance was through the similarity between the residence time distributions of periodically driven bistable systems and the ISIHs for periodically stimulated neurons. Neuronal stochastic resonance is more related to threshold detectors than to bistable systems because neurons have one stable state, the rest state. Having discussed fundamentals of stochastic resonance in the previous sections we now turn our attention to some of the tools and techniques that are used to investigate this phenomenon.

3.6 Tools and techniques of modelling stochastic resonance

The seminal paper by Benzi et al. (1981) provoked no immediate reaction in the literature. Apart from a few early theoretical studies by Nicolis and Nicolis (1982) and Benzi et al. (1982), only one experimental paper (Fauve and Heslot, 1983) addressed the phenomenon of stochastic resonance. One reason may be that the simulations required needed faster

machines than were easily available to researchers at the time. The experimental article by McNamara et al. (1988) marked a renaissance of stochastic resonance, which has developed and flourished ever since in different directions. The present knowledge of stochastic resonance has been reached through a variety of investigation tools. In this section we will outline the most popular ones.

3.6.1 Digital simulation

The first demonstration of stochastic resonance was produced by simulating the model of climate change by Benzi et al. (1981) on a Digital Instruments minicomputer (model PDP 11), an advanced computer at the time. Nowadays accurate digital simulations of either continuous or discrete stochastic processes can be carried out easily on desktop computers. Regardless of the particular algorithm adopted in the diverse cases, digital simulations proved particularly useful in the study of stochastic resonance in numerous cases.

3.6.2 Analogue simulation

This type of simulation allows more flexibility than digital simulation and for this reason has been preferred by many researchers as evidenced by the works of Dykman et al. (1990a,b); Gingl et al. (1995a,b); Kaufman et al. (1996); Luchinsky et al. (1999b) and Luchinsky et al. (1999a). Analogue simulators of stochastic processes are easy to design and assemble (Gammaitoni et al., 1998). Their results are not as accurate as digital simulations but offer some advantages: (a) a large range of parameter space can be explored rather quickly; and (b) high dimensional systems may be simulated more readily than by computers, though systematic inaccuracies must be estimated and treated carefully. Noise is also not easy to control in analogue systems and their inherent nonlinearities make them difficult to deal with.

3.6.3 Physiological experiments

To date there are a few experiments where stochastic resonance has been firmly established in a physiological experiment. Gluckman et al. (1996) report on observing stochastic resonance in mammalian Hippocampal slices and Douglas et al. (1993) report on stochastic resonance being exhibited in the tail fan of a crayfish. The most recent human related evidence of stochastic resonance comes from the work of Mori and Kai (2002). In their study, Mori and Kai shone light signals into the eyes of five students while measuring their electroencephalogram. The researchers shone periodic signals onto the right eyelids and noisy signals onto the left eyelids of the students as they rested, and measured the intensity of their alpha brain waves. They found a sharp peak at 5 Hz, the frequency of the periodic signal. But when they increased the strength of the noise signal relative to the periodic signal, a 'harmonic' peak emerged in the alpha waves at 10 Hz. As the noise signal became stronger, this peak first intensified and then diminished, a feature characteristic of stochastic resonance.

In this section we described some of the tools which were instrumental to the development of stochastic resonance. The results presented in this thesis portray stochastic resonance modelled on a PC and on an FPGA. The details of the techniques and models used are discussed in chapter 5. In the next section we discuss some of the benefits of stochastic resonance to neuronal modelling.

3.7 Some of the benefits of stochastic resonance

It is in the simplest systems that a positive role of noise is becoming particularly evident. This thinking has been driven by the discovery of stochastic resonance in excitable systems. For biological systems that exhibit a threshold, such as spike-generating neurons, subthreshold signals have no effect on the output of the system. In other words, all information present in the signal is lost. It is now evident that given the existence of a threshold, noise arising from different sources can enhance signal detection by allowing the

system to reach threshold. It may be argued that such a strategy would bypass the threshold as a safety device against "false positives", for example a cell with a 5 mV threshold might specifically have been designed to avoid detecting a 3 mV stimulus. However, for some sensory systems, such false positives do not exist. For example, hair cells in the mechanosensory receptors of the vestibular and auditory systems, respond in a graded fashion to the smallest motion (Traynelis and Jaramillo, 1998). A signal too weak to trigger afferent firing is essentially lost. As shown in experimental work, stochastic resonance could improve sensory systems in several ways.

Stochastic resonance has been described in the past as a nonlinear co-operative effect by virtue of which the response of a system to a weak periodic signal is enhanced by the addition of an optimal amount of noise. We discuss a few observations about this definition to help illustrate how widespread this effect is.

- In addition to simple threshold and bistable potential systems, stochastic resonance is also seen in the reset-and-fire excitable system. This system, much like a neuron, integrates a stimulus until a threshold is reached, at which time the system "fires" and resets. It has been shown that noise can improve the quality of transmission in the nervous system. More recent experiments in the cricket cercal sensory system have shown that the transmission of signals by the sensory cercal, measured in bits per action potential, is also enhanced by noise (Levin and Miller, 1996). Furthermore, these signals were fairly broad banded (5-400Hz), indicating that stochastic resonance can exist across the spectrum of frequencies present in axons.

- The signal need not be periodic: although periodic signals are commonly used experimentally, sensory systems are rarely subject to a pure periodic stimulus. However, recent experiments on rat cutaneous mechanoreceptors indicate that noise can enhance the transmission of aperiodic stimuli, again suggesting a role for noise in enhancing sensory transmission of biologically relevant signals (Collins et al., 1996b; Gluckman et al., 1996).

- Noise need not be optimised: for the relatively simple systems discussed in the lit-

erature there is an optimum noise level. Below this ideal level the system is not optimised, whereas beyond it the SNR is degraded. However, such a dependence is not an obligatory feature for all systems that exhibit stochastic resonance.

- It should be noted that although white noise is often used to model stochastic resonance, coloured noise can effectively produce similar behaviours (Capurro et al., 1998; Nozaki et al., 1999). An example is the noise generated by the random gating of ion channels. This noise, rather than white, has spectral characteristics that are determined by the transition rates between different states.

- Noiseless neurons typically have a very limited dynamic encoding range. If a constant stimulus is subthreshold, no spikes occur, if it is suprathreshold, firing at a constant rate occurs. This rate is not dependent on the value of the stimulus given that it is greater than threshold. Various authors have shown that noise can linearise the firing frequency versus stimulus-amplitude relation of the neuron (Gammaitoni, 1995; Shimokawa et al., 1999). Although it is still far from clear whether the encoding uses mean spike rate, the interspike interval or the precise timing of the interspike intervals in a spike train, it is interesting that the linearisation property arises naturally in the LIF neuron model.

3.8 Some misconceptions about stochastic resonance

The existence of a maximum in output SNR at an optimised non-zero noise level induced high expectations towards stochastic resonance as an extraordinary tool in signal processing and transmission. Very often stochastic resonance is described using ambiguous statements like "noise helps in detecting small signals..." or "stochastic resonance allows us to detect small signals". In these expressions, the ambiguity lies in the word "small". Small can be used to denote a subthreshold signal or a signal smaller than the noise in which it is hidden — and the two meanings are quite different. In the case of small meaning "subthreshold", a signal is small given a detector (for a lower threshold it is no longer small), while small

meaning "small compared to noise" is more of an absolute measure. In this thesis, small will always be used to speak of a signal that is small compared with the threshold of the LIF neuron, that is, a signal which does not make the LIF neuron fire.

It is worth emphasising that the problems of amplifying a signal or detecting it when masked by noise are quite different, in spite of the fact that the two words (amplification and detection) are often used in an equivalent way. Traditionally the most used methods for detecting small signals (small compared to the root mean square (RMS) of its fluctuations) are spectral analysis and averaging. Spectral analysis can be realised by using FFT, wavelet transforms, or by using narrow-band filters.

In this section we described some of the benefits of stochastic resonance to neuronal modelling. We also noted that some of the things stochastic resonance is associated with are much better done using other techniques. For instance, extracting a signal buried in noise can be done using spectral analysis techniques like wavelet transforms.

3.9 Summary

The purpose of this chapter has been to introduce the concept of stochastic resonance and how it applies to neural systems.

Stochastic resonance was roughly defined as a co-operative effect between noise and signal to aid the detection of the signal. It was originally developed to explain the changes in the Earth's climate but has since been extended to explain phenomena in fields such as neuroscience and optics. There are basically two kinds of systems in which stochastic resonance occurs: bistable dynamic systems (systems with two stable states) and excitable systems (systems with a threshold and one stable state). In both cases a clear maximum in the output signal and in the output coherence measure occurs at "tuned" values of the input noise. Neurons are examples of excitable systems. In neuroscience, stochastic resonance is quantified by interspike interval histograms, power spectrum and signal-to-noise ratio.

The tools and techniques used in investigating stochastic resonance in neuroscience range from analogue and digital simulation to actual experimental work.

In summary, stochastic resonance might be a general strategy employed by the central nervous system for the improved detection of weak signals. However, the effects of stochastic resonance in sensory processing might extend past an improvement in signal detection. As information flows towards progressively more central relay stations, it is handled by systems that might exhibit stochastic resonance, resulting in improved information processing. This enhancement might start to take place at the earliest stages of processing, as recently suggested by the enhanced vowel coding obtained with the addition of noise to cochlear implants and the amplification of amplitude modulated signals on the auditory nerve fibre by the neurons of the cochlear nucleus.

In our view there is nothing very special about stochastic resonance, it is just one of the many ways that the CNS may be using to encode signals in a noisy environment. It also provides a constrained way of investigating the role of noise in the central nervous system.

Chapter 4

Measurements from spike trains

The purpose of this chapter is to discuss some of the techniques used in analysing noisy spike trains. In section 4.1 we mainly look at the stochasticity of spike trains and some of the statistical distributions that are used to characterise them. Spike trains are used by some neurons to communicate information. Section 4.2 describes some of the potential coding schemes we believe neurons might use to convey information to each other. In section 4.3 we describe some statistical measures that are used to characterise the degree of randomness in a spike train relative to a Poisson distributed spike train. Section 4.4 describes the autocorrelation function. In section 4.5 we describe the power spectrum. We start by reviewing the different methods of sampling spike trains to obtain their Fourier components. We then describe how the power spectrum of a spike train can be computed. Finally, in section 4.6 we describe how to compute the signal-to-noise ratio of a spike train from its power spectrum.

4.1 Stochasticity in spike trains

A key property of many neurons' spike trains is their seemingly *stochastic* or random nature. This randomness is apparent in the highly irregular discharge pattern of a central neuron to a sensory stimulus whose details are rarely reproducible from one trial to the

next (Mainen and Sejnowski, 1995). The apparent lack of reproducible spike patterns has been one of the principal arguments in favour of the hypothesis that neurons only care about the firing frequency averaged over very long time windows (Koch, 1999). Such a *mean rate code* is very robust but is relatively inefficient in terms of transmitting maximal information per spike and is very slow at transmitting data. Encoding information in the intervals between spikes is much more efficient, in particular if correlated across multiple neurons and can also be much faster. Such a scheme does place a premium on postsynaptic neurons that can somehow decode this information.

Because little or no information can be encoded into a stream of regularly spaced action potentials (it codes only one value, namely the rate), this raises the question of how variable neuronal firing really is. How can the observed randomness be explained on the basis of the cell's biophysics and synaptic input? The mathematical theory of stochastic point processes and the field of statistical signal processing offer a number of tools suitable for analysing the properties of spike trains. We will study these here and will relate them to simple models of biophysics. This will enable us to infer something about the integrative mechanisms underlying neuronal firing activities.

4.1.1 Spike trains as a point process

The simplest model of neuronal firing statistics is the Poisson model. The defining feature of a Poisson process is that the firing of one spike occurs with some probability per unit time, the rate, and this rate can depend on time but not on the occurrence times of the other spikes (Tuckwell, 1988b).

A sequence of action potentials, or spike trains can be modelled by a one dimensional *point process*. By definition, a regular point process is such that no more than one event can occur in a sufficiently small interval, and that the probability of one event occurring in a small interval is proportional to the duration of the interval (Cox and Miller, 1965). This description of a neuron in terms of the probability of firing is consistent with the stochastic nature of spike trains as introduced in the literature. The sequence of action potentials

produced by a neuron can be characterised in terms of a stochastic point process. The simplification implies that spike duration and shape are neglected and all neuronal activity is represented by uniform events appearing in time. This is a reasonable simplification because spikes have been found to be similar in size and shape, and propagate with no attenuation. This means that the only information the rest of the nervous system has from spiking sensory neurons is the times of their spikes. This simplification also means that we do not have to worry about the spatial size of the neurons that we model which would be an issue because spiking neurons come in all shapes and sizes.

When all action potentials are taken to be identical and only their localised times of occurrences are considered, one obtains a discrete series of time events, $t_1, t_2, ..., t_n$, where t_i = time of arrival of the ith spike, characterising the spike train. It is this series of events that is transmitted down the axon to all the cell's target neurons and that contains most, if not all, of the information the cell is conveying. It is worth noting here that not all neurons produce spike trains. There are graded response neurons that do not produce stereotyped action potentials and there are also neurons that produce chemical outputs — hormones, such as oxytocin (a chemical messenger which is also a form of output) (Kandel et al., 2000).

As mentioned previously, spike trains can be modelled by stochastic point processes. Let t_i be a set of spiking times with a certain distribution. In computational neuroscience, spikes are commonly represented by a delta function, and a spike train is therefore a simple sum of delta functions, or δ-spikes:

$$x(t) = \sum_i \delta(t - t_i). \tag{4.1}$$

This mathematical representation can be confusing at first since real spikes have a finite amplitude and width. A justification along the lines of Rieke et al. (1997) follows. Suppose that a spike train recorded from an experiment is analysed with a discretised time bin Δt such that there is at most one spike per bin. Define $f(x)$ such that $f(x) = 1$ if $|x| < 0.5$ and $f(x) = 0$ otherwise, and a function $n(t)$ with value 1 if there is a spike in the bin

centred on t and value 0 otherwise. $n(t)$ can be written as:

$$n(t) = \sum_i f(\frac{t - t_i}{\Delta t}). \tag{4.2}$$

The average value of $n(t)$ is the probability that a spike will occur in a bin of width Δt centred on t. The firing rate $r(t)$ is obtained by normalising $n(t)$ by the bin size Δt and taking the limit $\Delta t \to 0$. $r(t)$ reads:

$$\begin{aligned} r(t) &= \lim_{\Delta t \to 0} \frac{\langle n(t) \rangle}{\Delta t} \\ &= \langle \sum_i \lim_{\Delta t \to 0} \frac{1}{\Delta t} f(\frac{t - t_i}{\Delta t}) \rangle \\ &= \langle \sum_i \delta(t - t_i) \rangle \\ &= \langle x(t) \rangle \end{aligned} \tag{4.3}$$

The firing rate $r(t)$ is therefore the average spike rate, which justifies the choice of the δ-shaped spikes. Integrating the firing rate over a finite duration T_0 gives the average number of spikes present in this time window.

The assumption that there is at most one spike in a bin of width Δt is in agreement with the description of the spike train in terms of point processes. This is also possible because of the absolute refractory period which means Δt will not need to be infinitely small and because real spikes have a finite duration. In this thesis, the model neurons considered are stimulated by a periodic external signal, for instance a pure tone for the case of neurons along the auditory pathway.

In this section we introduced point processes as one way of representing neuronal spikes as a time series which greatly simplifies the analysis of the spike trains. In the next section we discuss some of the codes that may be used by neurons to communicate with each other.

4.2 Neural codes

Over the past hundred years, biological researchers have accumulated an enormous amount of detailed knowledge about the structure and the function of the brain see, for example,

Kandel et al. (2000). In spite of all this detailed information about neurons and their connections, one fundamental question is still unresolved: What are the codes used by neurons or how do neurons use spikes to communicate with each other? Do neurons communicate by a "rate code" or a "pulse code"? How the neural output is represented, as a series of discrete pulses or as a continuous firing rate, relates to the code used by the nervous system to transmit information between cells.

A quick survey of the experimental literature reveals that there is no unique and well-defined concept of "mean firing rate". There are about three different notions of rate which are often confused and used simultaneously. The three definitions refer to three different averaging procedures: either an average over time, or an average over several repetitions of the same experiment, or an average over a population of neurons. In the following sections we consider these three concepts.

4.2.1 Rate as a spike count

The first and most commonly used definition of a firing rate refers to a temporal average which is essentially the spike count in an interval of a certain length divided by the interval length. Given the pulse-like nature of spike trains, the standard procedure to quantify the neuronal response is to count how many spikes arrived within some sampling window T and to divide this number by T. This definition of rate has been successfully used in many preparations. Classical results (see the pioneering work of Adrian reviewed in Rieke et al. (1997)) show that the experimenter as an external observer can evaluate and classify neuronal firing by a spike measure. This begs the question: is this really the code used by neurons in the brain? In other words, is a neuron receiving a signal from a sensory neuron only looking at and reacting to the number of spikes it receives in a time window of, say 200 ms?

It is quite clear that an approach based on a temporal average neglects all the information possibly contained in the exact timing of spikes. It is therefore no surprise that the firing rate concept has been repeatedly criticised (Koch, 1999) and is subject to an ongoing

debate.

It is important to understand the artificial nature of a rate based on spike count. From behavioural experiments it is known that reaction times are often rather short. A fly can react to new stimuli and change direction of flight within 30-40 ms as reported in the work of Rieke et al. (1997) which is just enough time for a neuron to fire a few spikes and certainly not enough time for counting spikes and averaging over some long time window. This means the nervous system has to make a decision based on a few spikes at most.

Temporal averaging can work well in cases where the stimulus is constant or slowly varying and does not require a fast reaction of the organism. This situation is only feasible in an experimental setting, in the real-world, stimuli are hardly ever stationary, but change on a fast time scale.

Despite its shortcomings, the concept of a temporal average rate code is widely used not only in experiments, but also in models of neural networks. Collins et al. (1995b) used a firing rate based model to establish stochastic resonance in a network of parallel model neurons. The firing rate code has also led to the idea that a neuron transforms information about a single input variable (the stimulus strength) into a single continuous output variable (the firing rate). This is good for investigating stochastic resonance because we can easily get spectral estimates of the spike train by suitably sampling the firing rate which is a continuous variable (see section 4.5). From the point of view of rate coding, spikes are just a convenient way to transmit the analogue output variable (firing rate) over long distances. Furthermore, the randomness encountered in real spike trains of neurons in the cortex must be considered as noise which should be averaged out over a large number of spikes.

4.2.2 Rate as a spike density

There is a second definition of rate which works for stationary as well as for time-dependent stimuli. In this case the experimenter records from a neuron while stimulating with some input sequence. The same stimulation sequence is repeated several times and the neuronal

response is reported in a Peri-Stimulus-Time Histogram (PSTH). This results in a time-dependent firing rate of the neuron. The time t is measured with respect to the start of the stimulation sequence and Δt is typically in the range of a few milliseconds. We can easily get the spike density $r(t)$ by doing the following: we count the number of occurrences of spikes $n_K(t, \Delta t)$, which is the total number of spikes found in the interval $(t, t+\Delta t)$ during K repetitions of the stimulus, divided by the number of repetitions K. This is a measure of the typical activity of the neuron between time t and $t + \Delta t$. A further division by the interval length Δt yields the spike density of the PSTH (Gerstner and Kistler, 2002):

$$r(t) = \frac{1}{\Delta t} \frac{n_K(t; \Delta t)}{K} \tag{4.4}$$

The obvious problem with this approach is that it cannot be the decoding scheme used by neurons in the brain. If we consider a frog which wants to catch a fly, this decoding scheme entails that the frog waits for the fly to make several runs along the same trajectory before the frog catches the fly. The frog has to make a decision based on a single "run" — each fly and each trajectory are different.

This coding scheme makes sense if there is a large population of independent neurons that receive the same stimulus. Instead of recording from a population of N neurons in a single run, it is easier to record from one neuron and average over N repeated runs.

The PSTH can be convolved with a smooth function like a Gaussian to produce a continuous instantaneous rate variable which could be sampled to compute the power spectrum of the spike train.

4.2.3 Rate as a population activity

Often many neurons have similar properties and respond to the same stimuli. If we assume a population of neurons with identical properties, particularly the same input and output connections, we can then define a spike-based population activity variable which varies rapidly and can reflect changes in stimulus conditions. The problem with this scheme is that we have assumed a homogeneous population of neurons which is hardly realistic. Nevertheless, rate as a population activity may be a useful coding principle in some areas

of the brain such as neurons in the primary visual cortex which exhibit synchrony when separate stimuli can be considered as part of a whole, for example, the same image (Shalden and Newsome, 1998).

4.2.4 Phase based codes

We can apply a coding of "time-to-first-spike" also in the situation where the reference signal is not a single event, but a periodic signal. In the hippocampus, in the olfactory system, and also in other areas of the brain, oscillations of the same global variable (for example the population activity) are quite common (Koch, 1999). These oscillations serve as an internal reference signal. Neural spike trains then encode the information in the phase of a pulse with respect to the background oscillation. If the input does not change from one cycle to the next, then the same pattern of phases repeats periodically.

4.2.5 Correlation codes

We can also use spikes from other neurons as the reference signal for a pulse. For example, synchrony between a pair or many neurons could signify special events and convey information which is not contained in the firing rate of the neurons. One famous notion is that synchrony could mean "belonging" together (von der Malsburg, 1994; Gerstner and Kistler, 2002).

In an instantaneous firing rate, the generation of each spike is independent of other spikes in the trains (neglecting the refractory period and bursting), only a single number, the rate, matters. In a correlation code this assumption is abandoned in favour of coupling among pairs, triplets, or higher order groupings of spikes.

A generic problem with the assumption of correlation codes is the question of decoding. It is unclear what sort of biophysical mechanisms are required to exploit information hidden in such correlations among spikes (Koch, 1999).

4.2.6 Population and correlated codes

We can distinguish two types of population codes, correlated ones and uncorrelated ones. The latter are straightforward: here the information from numerous neurons is combined into a population code but without taking into account any correlations among neurons. There is plenty of good evidence for such codes in a great variety of different sensory and motor systems, ranging from the cricket cercal interneurons encoding the direction the wind is blowing from (Theunissen and Miller, 1991), to the larger ensembles encoding the direction of sound in the Barn owl (Gerstner et al., 1997), and eye movements in the mammalian superior colliculus, to the posterior parietal cortex in the monkey encoding its representation of space (Lee et al., 1988).

Correlation population codes exploit the exact temporal relationships among streams of action potentials. One way to discover such codes is to record from two or more neurons simultaneously and to measure the cross correlation function.

Physiological evidence indicates that cross correlations among groups of neurons appear to encode various stimulus features.

Under certain conditions this might be different, if a cell, say in the cortex, has access to the spiking output of many cells with the same receptive field properties, the temporal average of the single presynaptic neuron can be replaced by an ensemble average over a population of neurons, thereby approximating the firing rate. In many cases population rate coding cannot occur for lack of a sufficiently large cell population to average over. In the insect, for instance, a very small number of clearly identifiable neurons code for particular features of the sensory input and no ensemble averaging occurs.

4.2.7 Discussion: spikes or rates?

The dividing line between temporal codes and firing rate codes is not always clearly drawn. Some codes which were first proposed as pure examples of pulse codes have later been interpreted as variations of rate codes (Koch, 1999). For example, a code based on population activities introduced earlier as an example of a rate code may be used for very fast tem-

poral coding schemes, population activity reacts quickly to any change in the stimulus (Gerstner, 2000). Thus rate coding in the sense of a population average is consistent with fast temporal information processing, whereas rate coding in the sense of a naive spike count measure is not.

We do not think it helps anyone to get mired in the debate of whether or not to call a given code a rate code. What is important, in our opinion, is to have a coding scheme which allows neurons to quickly respond to stimulus changes. A mere spike counting code with a long time window is unable to do this, but many of the other codes are. The name of such a code, whether it is deemed a rate code or not, is of minor importance.

In the next section we discuss interspike interval distributions and some statistics which can be derived from them.

4.3 Interspike interval distributions

The variability of a neuronal spike train is a vital indicator of the processing a neuron does on its synaptic inputs. One way of characterising the spike train is to look at the distribution of interspike intervals (ISIs). We can calculate some statistics based on the ISIs which give us clues about the type of processing that the neuron performs on its synaptic input. As pointed out in Chapter 3, the distribution of ISIs is a good measure of stochastic resonance, but it is not used to infer stochastic resonance only. It can also be used as a measure of synchronisation. The mean and variance of ISIs are used to calculate two measures which tell us the degree of variability in synaptic input to a neuron.

4.3.1 Coefficient of variation

The simplest of such statistics is the coefficient of variation (C_V) of the ISIs. C_V is defined as the standard deviation σ_t of the ISI distribution normalised by the mean ISI μ_t:

$$C_V = \frac{\sigma_t}{\mu_t} \tag{4.5}$$

with

$$\mu_t = \int_0^\infty t \quad p(t)dt, \tag{4.6}$$

and

$$\sigma_t{}^2 = \int_0^\infty (t - \mu_t)^2 p(t)dt \tag{4.7}$$

where $p(t)$ is the probability density distribution of the ISIs. C_V is a measure of spike train irregularity (Christodoulou and Bugmann, 2001). The coefficient of variation is equal to 1 for Poisson spike trains, since $\sigma_t = \mu_t$. The $C_V = 1$ for a Poisson spike train, indicates pure randomness in the spike train. This means that C_V values for other models are compared with the Poisson model, the closer the C_V is to 1 the more variability there is in that spike train. For a Gamma distribution of order n, $\sigma_t{}^2 = \mu_t{}^2/n$ and $C_V = 1/\sqrt{n}$. Integrating with an LIF neuron over a large number of small inputs gives rise to very regular spike trains as the ISI tends towards a normal distribution (Koch, 1999; Brown et al., 1999). This result can be explained by invoking the *central limit theorem* which states that as the number n of independent variables x_i goes to infinity, the random variable defined by the mean of all x_i's, $\langle x \rangle = (1/n)\sum_{i=1}^{n} x_i$, has a Gaussian distribution. In the limit of an integrate-and-fire neuron under constant current input, $C_V \to 0$. A refractory period lowers the C_V at high firing rates because it tends to force regularity in the interspike interval duration. In the ideal case of an absolute refractory period, the interspike interval probability density will be shifted to the right of the time axis, $p(t) \to p_{ref}(t) = p(t - t_{ref})$ and the new coefficient of variation is:

$$C_{Vref} = (1 - t_{ref}/\mu_t)C_V, \tag{4.8}$$

so that $C_{Vref} \to 0$ as $\mu_t \to t_{ref}$ (Koch, 1999).

The membrane time constant τ of an LIF neuron will also affect the coefficient of variation in different ways. If $n_{th} > 1$ (n_{th} is the number of inputs summed) coincident inputs are needed to fire the cell, a large τ will regularise the spike train by averaging the arrival of synaptic inputs over time, whereas a short τ will increase sensitivity to coincident inputs and thus boost variability.

4.3.2 Spike count distribution and Fano factor

The C_V is a useful measure of short-term variability because it is obtained from the interspike interval distribution. It yields a complete characterisation of variability only if the occurrence of a spike depends exclusively on the time of the previous spike and not on the past history of the spike train. This is the case for a spike train generated by a renewal process. By definition, successive intervals of a renewal process are independent and identically distributed (Cox and Lewis, 1966). Unfortunately the spike train of a leaky integrate-and-fire neuron model with a refractory period and receiving noisy sinusoidal input is not a renewal process. To make it a renewal process one would have to reset the stimulus each time the neuron fires so that the stimulus always starts at the same phase (Plesser and Tanaka, 1997).

Information on variability beyond the first interspike interval can be gleaned by the distribution of spike counts measured over a time period of length T. If we consider a Poisson spike train with mean firing rate $f = 1/\mu_t$, then the probability $p(n)$ of obtaining n spikes in the observation window T is (Koch, 1999):

$$p(n) = \frac{(fT)^n e^{-fT}}{n!} \tag{4.9}$$

The variability in the spike count distribution is characterised by the ratio of variance, $V(T)$, to the mean, $N(T)$, of $p(n)$ (Koch, 1999):

$$F(T) = \frac{V(T)}{N(T)} \tag{4.10}$$

This quantity is called the "index of dispersion" or "Fano factor". The Fano factor can be viewed as a kind of 'noise-to-signal' ratio; it is a measure of the reliability with which the spike count could be estimated from a time window that on average contains several spikes. In fact, for a renewal process spike generator, the distribution $p(n)$ of spike counts can be shown, by the central limit theorem (Feller, 1971), to be normally distributed (asymptotically, as the number of trials becomes large) with mean $\mu_T = T/\mu_t$ and variance ${\sigma_T}^2 = T \times {\sigma^2}_t/\mu_t^3$, where σ_t and μ_t are, respectively the standard deviation and mean of the ISI distribution $p(t)$. Thus the Fano factor is related to the coefficient of variation C_V by

$C_V = \sqrt{F}$. For a Poisson spike train, $F(T) = 1$ because mean and variance have the same value. Spike trains that are more regular than Poisson will have an index of dispersion smaller than 1 (Koch, 1999).

4.3.3 Estimating ISI distribution parameters

Initial estimates of the mean and variance of the ISIs from empirical data can be given by the following sample estimates. The mean of ISI $\hat{\mu}_t$ is given by:

$$\hat{\mu}_t = \frac{1}{k}\sum_{i=1}^{k} t_i \tag{4.11}$$

where $t_1, ..., t_k$ are the observed ISIs. The variance σ_t^2 of ISIs is given by

$$\hat{\sigma}_t^2 = \frac{1}{k}\sum_{i=1}^{k}(t_i - \hat{\mu}_t)^2 \tag{4.12}$$

In general, the accuracy of the estimates will depend on the extent of correlations between successive interspike intervals of the spike train. The most favourable case is the renewal process because successive intervals are independent. In this case, the variance of $\hat{\mu}_t$ is given by σ_t^2/k and decreases linearly with the number of observations from an initial value equal to the variance, σ_t^2, of the ISI distribution. Unfortunately an LIF receiving a periodic signal and Gaussian white noise is not a renewal process which means the estimates given above may not apply to its output. A numerical way of obtaining the distribution of the intervals for a noisy LIF neuron model is described by Plesser and Geisel (1999) and Hohn (2000).

In this section we discussed the interspike intervals and the measures of variability which can be derived from them. These measures of variability measure the degree of randomness in the input to the neuron. We noted that the Poisson distribution is the standard because it represents pure randomness. We also discovered that the distribution of the ISIs for the noisy LIF neuron with time dependent input is not straightforward because the intervals are not independent and identically distributed. In the next section we discuss the autocorrelation function which measures the correlation between spikes separated by

1 or more spikes in between. The ISIs give us information about consecutive spikes but sometimes we need to know the relationship between spikes separated by different time lags. We also note later that the autocorrelation function has a very special relationship with the power spectrum which we discuss in section 4.5.

4.4 Autocorrelation function

The mean spike count $N(T)$ (discussed in section 4.3.2) is well suited to convey information about static components of a stimulus in a time interval of length T, but stimulus parameters that vary over time during such an interval cannot be encoded by $N(T)$ alone. Second-order changes in the dynamics of neuronal firing are captured in the autocorrelation function of the spike train. Let $x(t)$ be the spike train of a neuron, represented by a sequence of δ pulses at the time of spike occurrences, as defined in section 4.1.

The mean firing frequency for an ensemble of observations of $x(t)$ is $\bar{r} = \langle x(t) \rangle$ and is assumed to be independent of t (i.e. we assume that the spike train is stationary). The autocorrelation function with lag τ is defined as the average joint probability density of a spike at time t and $t + \tau$, minus their mean values:

$$R_{xx}(\tau) = \langle (x(t) - \bar{r})(x(t+\tau) - \bar{r}) \rangle \tag{4.13}$$

By time invariance (stationarity), $R_{xx}(\tau)$ is assumed to be independent of the absolute time point t. It follows from this assumption that $R_{xx}(\tau) = R_{xx}(-\tau)$. For a Poisson process, $R_{xx}(\tau) = \bar{r}\delta(\tau)$ (Koch and Segev, 1999). The δ-function at the origin corresponds to the sure event of a spike at point t given a spike at point t, while for $\tau \neq 0$, the autocorrelation function $R_{xx}(\tau)$ vanishes, meaning that two spikes separated by an arbitrary time interval τ are completely uncorrelated (complete independence between two events which defines a Poisson process).

The autocorrelation function is a measure of a neuron's memory of previous spikes. It is not used directly as far as measuring stochastic resonance is concerned. We introduced the autocorrelation function in this section because it can be used to compute the power

spectrum of a spike train which is a quantifier of stochastic resonance.

4.5 Power spectrum

Analysis in the frequency domain offers a compact description of the moments of a stochastic process and when applied to the input and output of a system, provides a means of estimating the frequency response function and the coherence function of the system. Spectral analysis provides a powerful technique for describing the lower order moments of a stochastic process and interactions between two or more stochastic processes. A major problem in the application of spectral analysis to neuronal spike trains is how to obtain equi-spaced samples of spike trains which will give unbiased and alias-free spectral estimates. A number of different ways of calculating the power spectrum of a spike train exist. We will review some of them before we introduce the method that will be used in this thesis.

4.5.1 Sampling spike trains

Because of the short duration of action potentials relative to their frequency of occurrence in many situations, it has become commonplace to treat action potentials as "events" in time and to assume that the only parameters of importance in neural coding depend in some way on the occurrence times of the action potential. This representation presents problems to the implementation of spectral analysis for neuronal spike trains. On the one hand a continuous signal is desirable for implementation of the Fast Fourier Transform (FFT) algorithm by regular sampling, and on the other hand treatment as a point process is more acceptable and conventional but poses computational problems. For continuous signals, we can sample them to produce a regularly spaced series of amplitudes for use in the FFT. A series of action potentials, represented by equal amplitude spikes at irregular times of occurrence, cannot be sampled in a similar manner. Figure 4.1 summarises the problem we have when it comes to sampling spike trains so that we can perform algorithms like FFT to get the spectral estimates of the spike train. A number of different approaches

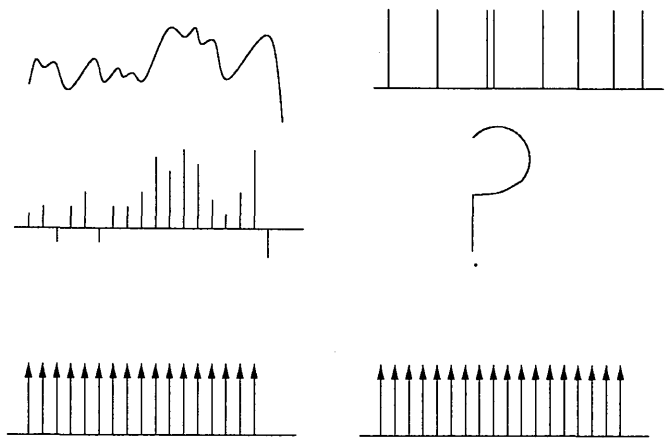

Figure 4.1: Continuous signals, such as that illustrated in upper left, are sampled by use of Dirac comb (lower left) to produce a regularly spaced series of amplitudes for use in Fast Fourier Transform (middle left). A series of action potentials represented by equal amplitude spikes at irregular occurrence, cannot be sampled in a similar manner by the same Dirac comb (right).

to the problem using both possible presentations have been suggested or implemented. We shall briefly review these approaches and their various characteristics.

4.5.2 Treatment as a continuous process

If $x(t)$ is a recording from a neuron, which is a continuous, bandlimited signal of finite energy, then one could simply use the Nyquist criterion to choose an appropriate sampling interval, Δt. However, because of the fast rise time of an action potential, the sampling will be of the order of several thousand samples/second and processing on the computer restricts the record length N to $T = N\Delta t$. The spectral estimate so obtained will be from an unreasonably short length of $x(t)$ and will have a low resolution since the fundamental frequency f_0 of the spectral estimate is inversely proportional to the record length (Priestley, 1982).

A criticism of this technique by French and Holden (1971) is that the spectral estimates

obtained from this technique will reflect the shape of the spikes, which depends on the recording geometry. Since the spectrum obtained contains components determined by the recording conditions, any change in these conditions will be reflected in spectral estimates.

4.5.3 Treatment as a point process

If the spike train is to be considered as a series of events, there are several ways in which it may be analysed: in terms of the series of interspike intervals, in terms of a counting process, in terms of the instantaneous frequency of the spikes or in terms of a series of delta-functions at the spike occurrence times.

Sampling a series of interspike intervals

If the spike train is considered in terms of its interspike intervals I_i then the power spectrum of the intervals may be defined by the following equation (French and Holden, 1971):

$$S(f) = \frac{1}{2} \sum_{j=-\infty}^{\infty} C_j \cos(f) \tag{4.14}$$

where f is frequency and C_j, the autocorrelation of the intervals is defined by:

$$C_j = E([I_i - \bar{I}][I_{i+j} - \bar{I}]) \tag{4.15}$$

where $\bar{I}$ is the mean of the series I_i. Equations for unbiased estimators of the power spectrum of an interval process are given in Cox and Lewis (1966). The spectral estimate given above is an estimate of the spectrum of intervals of the spike train, rather than the spike train itself, and this spectrum, although perhaps of interest in its own right, is not readily interpretable. Furthermore, since the series I_i is not a function of time but of interval number, the spectrum obtained by the above equation is not a function of frequency. This makes the spectral estimates resulting from this technique unsuitable for investigating stochastic resonance.

Sampling a stochastic point process

If the spike train is treated as a stochastic point process, specified by events labelled by the random value of a continuous parameter (time), then it may be characterised by the cumulative number $n(t)$ of events which have occurred up to time t. This is called a counting process. This characterisation of a spike train by the cumulative distribution function $n(t)$ does not make any assumptions about the way the signal is coded in the spike train. A spectral estimate of this counting process can be obtained using methods described by French and Holden (1971). Thus to sample a spike train, one need only count the number of spikes in an interval. The problem with this technique is that the counting introduces both bias and aliasing. This occurs because the sample obtained by the counting procedure uses the result of an integration (to obtain $n(t)$) and a finite difference operation.

Sampling instantaneous frequencies

One method of regularly sampling a spike train is to generate a continuous signal representing the (average) instantaneous "frequency" (more properly the reciprocal of the interspike interval) of the spike train and to sample this waveform at a rate faster than half the average spike rate (French and Holden, 1971). When the sample instant falls on a discontinuity in the wave form, the sample value at this point may be taken as equal to the average of the adjacent instantaneous frequencies. This process is illustrated in figure 4.2. This technique is open to criticism. The instantaneous frequency q_i, is related to the preceding interval I_{i-1} by $q_i = 1/I_{i-1}$. For a fairly regular spike train, where deviations from the mean carrier frequency are small, deviations in intervals are simply related to deviations in instantaneous frequency by:

$$\frac{\Delta I_{i-1}}{\bar{I}} = -\frac{\Delta q_i}{\bar{q}} \tag{4.16}$$

where $\Delta I_{i-1} = I_{i-1} - \bar{I}$, $\Delta q_i = q_i - \bar{q}$ and $\bar{I}$ and $\bar{q}$ are the means of the intervals and the instantaneous frequencies respectively. However, for an irregular spike train with large deviations in frequency about the mean, there is no simple relationship between deviations

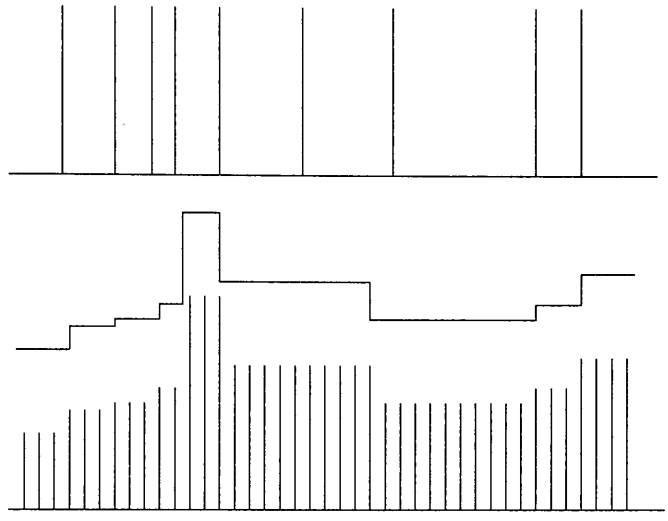

Figure 4.2: An irregular spike train as shown at the top of the diagram may be converted into an instantaneous frequency function (centre). The instantaneous frequency function has discontinuities at each spike occurrence time but may be sampled regularly to give equispaced amplitudes (lower) if mean values are assumed at the discontinuities.

about the mean in interval and instantaneous frequency, and so the spectral estimates from the samples of instantaneous frequency are not simply related to a spectral estimate obtained from other techniques.

Sampling Dirac delta functions

If the spike train is considered as a series of Dirac delta functions occurring at times t_i (equation 4.1), the spectrum can simply be estimated by evaluating a series of sine and cosine terms at the occurrence times t_i (French and Holden, 1971):

$$S(f) = \frac{[(\sum_i \cos ft_i)^2 + (\sum_i \sin ft_i)^2]}{T} \tag{4.17}$$

This is a simple calculation but it involves as many calculations as the original discrete Fourier Transform, which makes it computationally expensive.

In this section we have reviewed some of the techniques that could be used to sample spike trains and then derive spectral estimates. We saw that most of the methods fall short

of the requirements. We would like a method which preserves the input signal frequency as much as possible because stochastic resonance requires us to look at the coherence of the output to the input signal in the frequency space. In the next subsection we will introduce another method of estimating the spectra of spike trains. This method is described in chapter 9 of Koch and Segev (1999) and explained in the next subsection. It is this method that will be used to obtain the results presented in chapter 6 of this thesis.

4.5.4 Power spectrum of a spike train

The power spectrum is defined as the square of the modulus of the Fourier transform (Priestley, 1982). Because the autocorrelation is real and symmetric (i.e. $R_{xx}(\tau) = R_{xx}(-\tau)$), its Fourier transform:

$$S_{xx}(f) = \int_{-\infty}^{+\infty} R_{xx}(\tau) e^{2\pi f i \tau} d\tau \tag{4.18}$$

is also real and symmetric. This means an alternative way of computing the power spectrum is to find the Fourier transform of the autocorrelation function. $S_{xx}(f)$ is a positive function of frequency which represents a measure of the frequency content of the spike train.

The autocorrelation function can be expressed as the inverse Fourier transform of the power spectrum:

$$R_{xx}(\tau) = \frac{1}{2} \int_{-\infty}^{+\infty} S_{xx}(f) e^{-2\pi f i \tau} df \tag{4.19}$$

For a Poisson process spike train, the Fourier transform of $R_{xx}(\tau)$ yields $S_{xx}(f) = \bar{r}$, and thus all frequencies are equally represented.

Estimation of power spectrum

The starting point for power spectral density estimation is the so called Wiener-Kinchin formula (Koch and Segev, 1999):

$$S_{xx}(f) = \lim_{T \to \infty} \frac{1}{T} |X_0(f)|^2 \tag{4.20}$$

where :

$$X_0(f) = \int_0^T x_0(t)e^{2\pi fit}dt, \quad x_0(t) = x(t) - \bar{r} \tag{4.21}$$

which states that the power spectrum can be obtained directly as the modulus of the Fourier-transformed series $X_0(f)$.

The action potential events are of the form $x = \{x_1, ..., x_N\}$, where $x_i = x(t_i)$, $t_i = i.\Delta t (i = 1, ..., N)$ and $T = N\Delta t$ is the recording time. The value of x_i is either 0 (if no action potential occurred in the interval $t_i \pm (1/2)\Delta t$) or $1/\Delta t$ (if an action potential occurred in the interval $t_i \pm (1/2)\Delta t$) which is the discrete approximation of the continuous $\delta-$function. The continuous Fourier transform is approximated by the discrete Fourier transform:

$$\hat{X}_0(f_j) = \Delta t \tilde{X}_{0j}, \tag{4.22}$$

where

$$\tilde{X}_{0j} = \sum_{m=1}^{N} x_m e^{2\pi i f_j t_m}, \tag{4.23}$$

where $f_j = \omega_j/2\pi$ takes values at the discrete frequencies $f_j = j/N\Delta t, j = -N/2, ..., N/2$ (for N even).

An estimator for power spectral density is given by the periodogram:

$$\hat{S}(f_j) = \frac{(\Delta t)^2}{T}|\tilde{X}_{0j}|^2, \quad j = -\frac{N}{2}, ..., \frac{N}{2} \tag{4.24}$$

Without any form of averaging, this estimate will be very unreliable. A computationally convenient averaging procedure is to subdivide the observation series into k contiguous segments $l = 1, ..., k$, compute the periodogram $\hat{S}_l(f_j)$ separately over each segment, and then average :

$$\hat{S}(f_j) = \frac{1}{k}\sum_{l=1}^{k} \hat{S}_l(f_j). \tag{4.25}$$

The estimate of the power spectrum is further improved by multiplying each segment of data with the Bartlett window function prior to Fourier-transforming (Priestley, 1982). This minimises the boundary effects due to the finite size of the samples.

In this section we reviewed some of the techniques that are used to sample spike trains. We noted the main problem in trying to sample spikes is their short duration relative to

their frequency of occurrence. In order for one to do spectral analysis on spike trains one needs to form a continuous signal from the spike trains which one can then sample. We also noted that forming this continuous signal from spike trains presents problems.

Having come up with a way of calculating the power spectrum of a spike train, in the next section we use the calculated power spectra to compute the signal-to-noise ratio which provides the most objective measure of stochastic resonance.

4.6 Signal-to-noise ratio

This measure has been designated as the measure for stochastic resonance (Barbi et al., 2000). SNR is a good performance measure of stimulus encoding as a function of frequency and it is also a measure of signal quality. If we assume a periodic signal $s(t)$ with power spectrum $S(f)$ and Gaussian white noise $\xi(t)$ with a power spectrum $N(f)$ then the input signal-to-noise ratio (SNR) is defined as:

$$SNR = \frac{\sum_f S(f)}{\sum_f N(f)} \tag{4.26}$$

This definition works quite well when one is looking at the input SNR (where input SNR is the SNR of the subthreshold periodic signal and the noise feeding into a neuron) in which the signal and noise can clearly be distinguished. It is not so clear when we look at a spike train because the spike train encodes both the noise and the signal. In this work, we propose three ways of calculating SNR from the power spectrum of a spike train and we report on the results from all three methods. The three methods assume that the power spectrum has been computed using the method just described in the previous section. For the rest of the thesis stochastic resonance results will be based on one of these methods after evaluating their performance. For lack of better names these methods are just going to be called method 1, method 2 and method 3.

4.6.1 Method 1

Method 1 is described as follows - given the power spectrum of the spike train $S(f)$, we partition it into two parts (see figure 4.3). The first part is the signal band here called $\overline{(S+N)}$, centred at signal frequency f, and the noise band $\overline{N}$, centred at some frequency n away from the signal frequency. These two bands have equal width $2r+1$ where r is an integer representing the offset on either side of the frequency of interest. The signal band is chosen in such a way that the band is centred on the frequency of the input signal. The noise band is situated in a region which is far removed from the signal band. The rationale for this is that we expect the signal energy to be concentrated within a region around the signal frequency and due to leakage problems we have to consider a neighbourhood of length r on either side of the signal frequency. As for the noise band, we put it away from the signal frequency because we feel that the influence of the signal away from the signal frequency is very minimal which means that most of the energy in this band will be mostly due to noise. This method relies on the noise power being the same in both bands which is a reasonable assumption given that we are using Gaussian white noise. Having defined signal and noise power spectra, the SNR of a spike train can be computed as follows:

$$SNR = \frac{\overline{(S+N)} - \overline{N}}{\overline{N}} \tag{4.27}$$

where :

$$\overline{S+N} = \sum_{x=(f-r)}^{f+r} S(x) \tag{4.28}$$

and:

$$\overline{N} = \sum_{x=n-r}^{n+r} S(x) \tag{4.29}$$

4.6.2 Method 2

Method 2 defines the signal region in the same way as method 1. The difference is in the definition of noise (figure 4.4). In this case, noise is defined as everything outside the signal

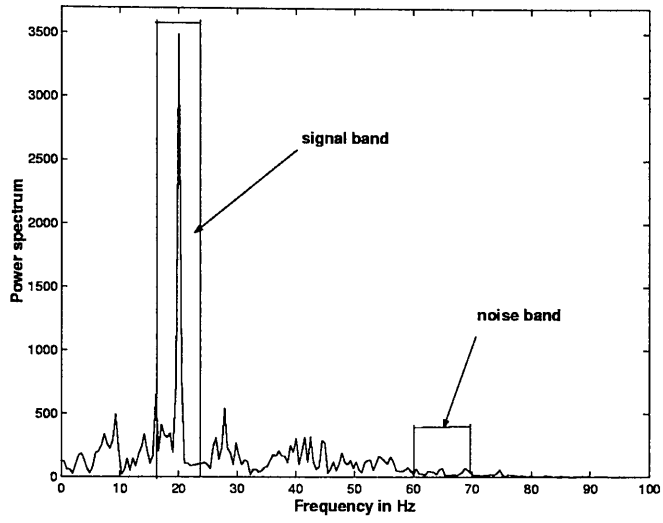

Figure 4.3: Power spectrum of spike train: Method 1, the signal and noise bands have equal lengths but they are widely separated from each other.

region. In this case we sum all the components within the signal band and divide by the length of the band. We do the same for the noise, we sum all the components and divide by the size of the noise band as shown in the equation below:

$$SNR = \frac{\frac{\sum_{x=f-r}^{f+r} S(x)}{2r+1} - \frac{\sum_{x=0}^{fc} S(x) - \sum_{x=f-r}^{f+r} S(x)}{fc-2r+1}}{\frac{\sum_{x=0}^{fc} S(x) - \sum_{x=f-r}^{f+r} S(x)}{fc-2r+1}} \tag{4.30}$$

where f_c is the cut off frequency. This method assumes that noise power is distributed similarly over the whole spectrum. This assumption is not a good one if the spectrum has some significant harmonics, in that case we cannot treat everything away from the frequency of interest as noise.

4.6.3 Method 3

The third method is illustrated in figure 4.5. Here we consider the energy at the signal frequency and ignore the issue of leakage and divide that by the average energy surrounding

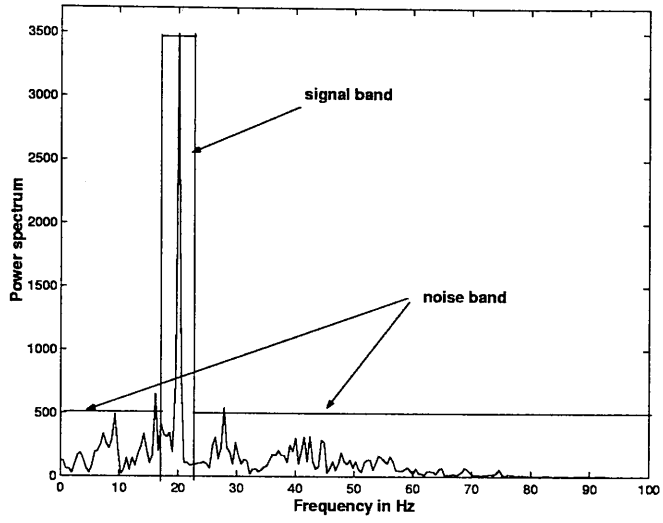

Figure 4.4: Power spectrum of spike train, Method 2: the signal and noise bands have different lengths.

the signal frequency (but excluding the power spectrum at the signal frequency) as follows:

$$SNR = \frac{S(f)}{\frac{\sum_{x=f-r}^{f-1} S(x)+\sum_{x=f+1}^{f+r} S(x)}{2r}} \tag{4.31}$$

The assumption in this method is that noise in the immediate area around the signal frequency is the same. This is a much more reasonable assumption because we only make assumptions about the distribution of noise near the frequency of interest.

4.6.4 Comparison of the methods

Looking at figure 4.6, we can see that method 3 gives higher values than both method 1 and method 2 because the graph for method 3 is consistently above that of method 1 and method 2. Methods 1 and 2 are almost indistinguishable. This may be due to the fact that the signal component is similar and the noise averages out to the same in both cases because we are using Gaussian white noise. Methods 1 and 2 are good in situations where we are dealing with an input signal where there is one dominant frequency like a pure tone and the noise has a flat spectrum. In this case we expect most of the signal

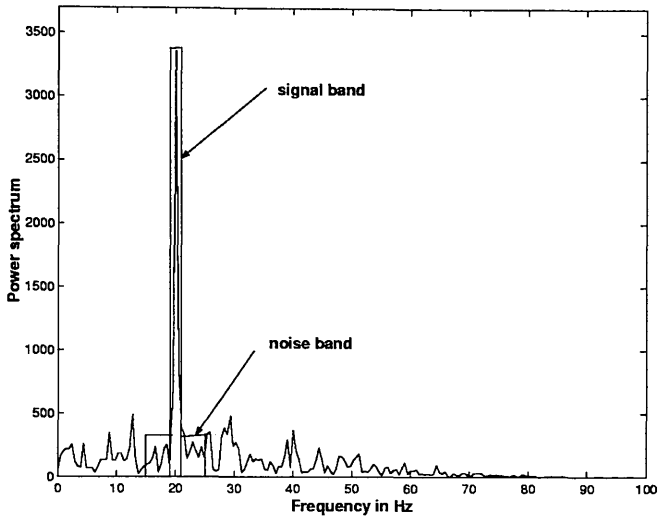

Figure 4.5: Power spectrum of spike train, Method 3: the signal is just the power spectrum at the signal frequency while the noise is just the background energy around the signal frequency.

energy to be concentrated around the signal frequency. On the other hand method 3 is better in situations where we are dealing with a broadband signal or a signal with several harmonics. In this case we cannot afford to treat all the energy away from the signal fundamental frequency as due to noise. For the rest of this thesis, SNR results will be based on method 3 not only because it gives the highest SNR values, but mainly because the power spectra of the spike trains have been seen to have a significant first harmonic for some noise values (see figure 3.3) which is definitely not due to noise. We also choose method 3 because we are stimulating the LIF neurons with single frequency signals as input. The other reason for choosing method 3 is that it makes a more reasonable assumption about the noise levels in the signal as compared to the other two methods. Specifically, the assumption is that the noise level near the frequency of interest is the one we are computing the SNR with. The other methods need to make assumptions about the noise level far from the frequency of interest. The effect of the value of r in the computing of the SNR for the three methods is shown in figure 4.7. Methods 1 and 3 seem to be less

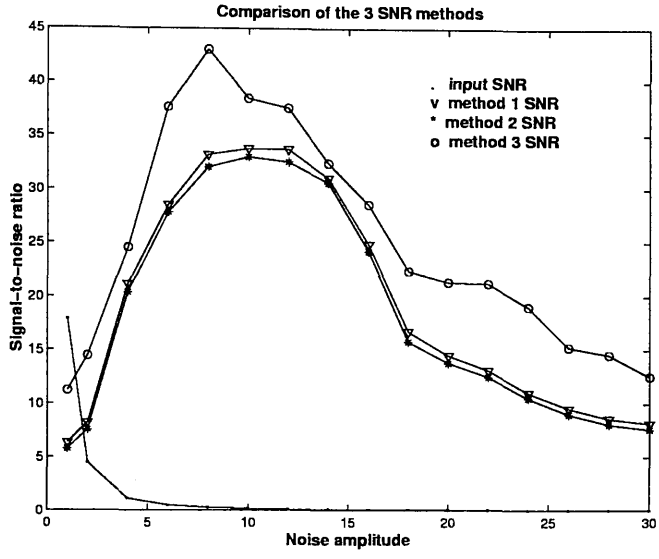

Figure 4.6: The result of using three different ways of computing SNR from spectral estimates of spike trains for a model LIF neuron receiving subthreshold 20 Hz sinusoidal signal plus Gaussian white noise.

sensitive to the value of r as compared to method 2. Method 2 is sensitive to the changes in r. As r increases the SNR value for method 2 drops because the signal is being diluted by the noise which is being averaged as part of the signal. This provides another reason for not choosing method 2.

4.7 Discussion

We have described the various neural codes and came to the conclusion that they are distinguished by the averaging technique used (time averaging, trial averaging and population averaging). Unfortunately these averages are not mutually exclusive. For example we can repeatedly count the number of spikes in a given time window after presenting a stimulus (time and trial averaging). Or we can count the number of spikes produced by a population of neurons in a given time window (time and population). We also noted that the distinction between time and trial averaging is not clear. It seems from literature

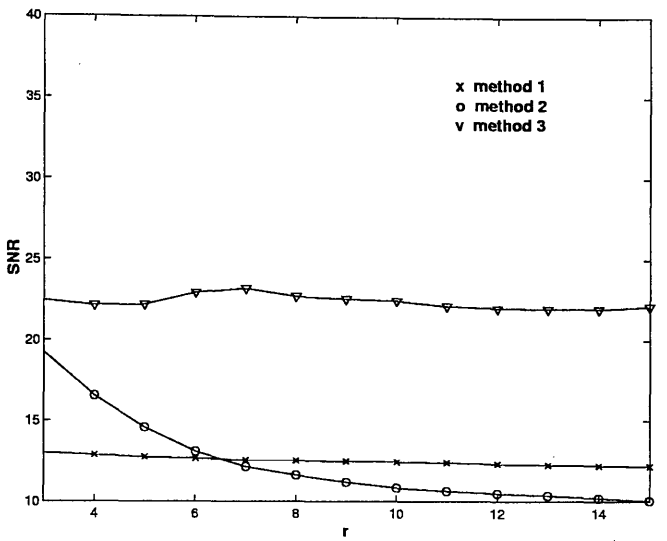

Figure 4.7: The effect of the size of r in the computation of SNR using methods 1, 2 and 3. These values were computed at a noise amplitude of 10 relative to figure 4.6

that if we measure spikes with millisecond precision, the code could count as temporal. On the other hand, if we use a window of length of tens of milliseconds, the code would count as a rate (Sterratt, 2002). The problem with this is that the dividing line between the two is not clear. For the purposes of quantifying stochastic resonance, we need a code that responds to stimulus changes. A mere spike counting code with a long time window is unable to do that but most of the other codes can.

Concerning the power spectrum, we noted that even though spectral analysis provides a powerful technique for describing the lower order moments of a stochastic process, spike trains present a major problem in the application of spectral analysis to them. The problem is that there is no easy way to obtain equi-spaced samples of spike trains which will give unbiased and alias-free spectral estimates. The main problem is their short duration relative to their frequency of occurrence. We discussed various techniques of sampling spikes. For continuous signals, we can sample them to produce a regularly spaced series of amplitudes, spikes at irregular times of occurrence, cannot be sampled in a similar manner.

We also discussed and evaluated three different methods of computing the SNR of

a spike train given its power spectra. The general formula (see equation 4.26) that we used for computing SNR for each of the three methods was inspired by that used by Chapeau-Blondeau et al. (1996). The main difference between our method and that of Chapeau-Blondeau et al. is that their neuronal models received noisy spike trains denoted by $\eta(t)$ and periodic spike trains denoted by $s(t)$ as input to the neuron. The output spike train was denoted by $y(t)$. When it came to the calculation of the SNR they divided the output spike train power at the signal frequency $S_{yy}(1/T_s)$ by the input noise power $N(1/T_s)$ at signal frequency. The weakness of this method is that it relies on one knowing the input noise power. Unfortunately we can never be sure of the exact input that a neuron is receiving during recording which means we can only observe the output without really knowing what the input looks like. This calls for a method of computing SNR which does not assume anything about the input noise. The method should just be based on the output spike train. The three methods that we discussed in this chapter meet this requirement.

The main differences between the three methods that we used is that they make different assumptions about the distribution of the noise power and the definition of the signal power. We preferred method 3 to the other two for the rest of the thesis because it compares the signal power at the frequency of interest with the surrounding noise power. The other two methods need to make assumptions about noise levels far from the frequency of interest.

4.8 Summary

In this chapter we have described some of the measures which can be derived from a spike train. We started with rates that are generally described as either temporal average, average from several repetitions or population-based average depending on the averaging technique used. We also described some statistics from the interspike intervals and the distribution of the intervals. Using the interspike intervals we described statistics which are due to static parts of the stimulus and those which are due to time-varying aspects of the stimulus. We also described the autocorrelation function and how the power spectrum

can be derived from it. Using the power spectrum we described three different methods of calculating the signal-to-noise ratio of a spike train.

Chapter 5

Simulation models

This chapter discusses the modelling work that was done. We start by introducing and justifying the network model that we used to investigate stochastic resonance. In section 5.2 we describe the floating point model for implementing the network model described in section 5.1. The derivation of the integer model from the floating point model is described in section 5.3. We also introduce a model that is between floating point and integer representation to narrow the gap between these two models in section 5.2. This new model is a discretised floating point model. Because we are interested in investigating the effect on stochastic resonance of precision limitation, we will also discuss implementation of both the integer and floating point models of this network model in sections 5.4 and 5.5 respectively. The FPGA hardware will be discussed as well. This chapter concludes with a comparison (section 5.6) and discussion (section 5.7) of the models that we used to investigate the effect of limited precision on stochastic resonance.

5.1 Network model

In this section we discuss the general network model that we used to study stochastic resonance in a network setup. We start by describing and justifying the network model followed by a description of how our work relates to work by other researchers.

5.1.1 Network architecture

To investigate stochastic resonance in a network environment the network structure in figure 5.1 was adopted. All the neurons are identical except for the difference in input type between the input neurons and the output neuron. The input neurons receive a common sinusoidal subthreshold signal plus independent uncorrelated bandlimited Gaussian white noise. The output neuron receives spike trains from the input neurons via synapses which generate current from the spikes using an α synapse which is described further on in this section. This network structure (figure 5.1) was inspired by the layout of multipolar stellate

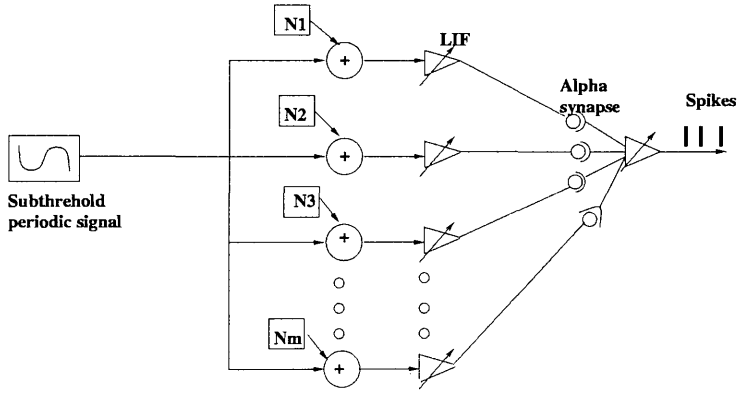

Figure 5.1: Network of Leaky integrate-and-fire model neurons: the input neurons receive a common subthreshold periodic sinusoidal signal and independent Gaussian white noise streams N_m (N_m are the independent noise streams and m is the neuron number).

cells in the cochlear nucleus (Frisina et al., 1994). Each of these cells receives input from several auditory nerve fibres that are tuned to a certain frequency (Pickles, 1988). These neurons are said to amplify the amplitude modulation in the auditory nerve fibres (Rhode and Greenberg, 1994).

The input layer neurons receive continuous inputs which are composed of a subthreshold sinusoid signal and bandlimited Gaussian white noise. The input neurons are like sensory neurons which receive transduced sensation (e.g. odour, light, sound, etc). For the input neurons we can control the strength of the input noise and we know what is noise and what

is signal. By using continuous input we can easily control the frequency and amplitude of the input signal. The input for the output neuron consists of spike trains from the input layer. By just looking at the spikes which constitute the input for the output neuron we are not able to tell which is signal and which is noise because the spikes encode both the noise and the signal. Because we can control the input (noise amplitude, and signal frequency and amplitude) for the input neurons, we can make sure that the signal is subthreshold in the absence of noise, ensuring that the input neurons only spike with the aid of noise. This means we can talk about stochastic resonance in the conventional sense as far as input neurons are concerned. As for the output neuron, things are a bit different because we cannot precisely tell how much signal or noise there is in the incoming spike trains which means we can not make the signal component subthreshold for the output neurons. This means the spike trains of the output neuron portray some kind of stochastic resonance plus other things, one of which is coincidence detection.

5.1.2 Model in detail

In the analysis presented in this thesis we consider an output neuron that receives spike-based input from a set of N independent input neurons each of which is stimulated by $I(t)$ (see equation 5.1). Table 5.1 lists the symbols used in this chapter.

This model is an extension of that by Burkitt and Clark (1999) and Burkitt and Clark (2000) and is similar to that of Shimokawa et al. (1999), Collins et al. (1995b), and Zalanyi et al. (2001).

The model by Burkitt and Clark analyses the timing information contained in the response of a neuron to noisy periodic synaptic input for the leaky integrate-and-fire neural model. They address the question of the relationship between the timing of the synaptic inputs and the output spikes. This requires an analysis of the interspike interval distribution of the output spikes, which is obtained in the Gaussian approximation. They found that the synchronisation of the output spikes increases sharply as the inputs become synchronised. This enhancement of synchronisation is most pronounced for large numbers

symbol	meaning
ω	angular frequency in radians
σ	noise amplitude
ξ	Gaussian white noise
a_k	synaptic strength for synapse k
t_{ki}	spike time for the i^{th} from neuron k
$\alpha_k(.)$	synaptic response function
τ_s	synaptic time constant
$\Delta^{as}{}_k$	axonal delay for fibre k
Δt	simulation time step
δt	time step for piecewise continuity
V_{fp}	floating point model membrane potential
V_{fpd}	discretised floating point model membrane potential
V_{int}	integer model membrane potential
γ	discretisation step
r	discretised floating point activation
$\lceil .. \rceil^+$	nearest highest integer

Table 5.1: Table shows the list of the symbols used in this chapter.

of inputs and lower frequencies of modulation. This is significant because it is in these lower frequencies that stochastic resonance has been established. A large number of input neurons is also consistent with a weak coupling between input neurons and the output neuron.

Shimokawa et al. analysed the transmission of sine-like periodic signals by an ensemble of leaky integrate-and-fire model neurons in the presence of additive noise. They observed that when the number of units in the ensemble is large enough, the process formed by pooling the spike trains of all units is an inhomogeneous Poisson process. They showed that

firing precision in response to subthreshold stimulation is maximised at some intermediate noise value, and argued that in this regime the ensemble can reliably transmit fast periodic signals below the resolution of the individual units.

The model by Collins et al. considered a summing network of identical excitable units, which were taken to be FitzHugh-Nagumo (FHN) model neurons. Each unit was subjected to a common input signal and independent noise. They assumed that information was transmitted by each unit via temporal changes in its firing rate. This assumption is valid for many sensory neurons (Shepherd, 1988). The network was operated by summing the mean firing rate signal from each unit to obtain a resultant mean firing rate signal for the entire system. The measure for stochastic resonance in this work was the so called normalised power norm which is a normalised cross-correlation between the input signal and the output mean firing rate. The power norm measures the coherence between the input signal and the network response. The results for a single-unit showed the characteristic signature of SR-type behaviour: a rapid rise to a clear peak and then a slow decrease for higher noise values. As the number of units was increased, the peak for the power norm value increased. Thus, the stimulus-response coherence was enhanced. As the size of the network was increased, the power norm asymptotically approached a plateau for a range of noise amplitudes larger than the optimal noise value for a single unit. This plateau indicates that for a large network, maximum fidelity is attained for a wide range of noise amplitudes.

In their paper, Zalanyi et al. examined in an integrate-and-fire neuron network the interaction between stochasticity and synaptic depression. They discovered that this kind of synaptic plasticity mechanism increased the SNR in some parameter regimes. The behaviour of the network without synaptic modification was characterised as an enhanced performance, when compared to a single neuron model. Increasing the noise with weaker synaptic coupling produced higher SNR on the output layer compared to the stronger synaptic coupling case. They showed that synaptic depression has a beneficial role in amplifying noisy signals, although the parameter regime they found was small.

The difference between our model and that of Burkitt and Clark is that in the case of

Burkitt and Clark, the input neurons' spike trains are represented by simulated Poisson processes with time-dependent rates whereas in our case each input neuron is actually an LIF neuron model stimulated by continuous input. The model of Shimokawa et al. differs from the one described here in that they did not use synapses onto the output neuron, they pooled the spike trains of all the input neurons. The difference between our model and that of Collins et al. is that we used an LIF neuron model with periodic input instead of the FHN model with an aperiodic input. The other difference is that like Shimokawa et al. (1999), Collins et al. (1995b) sum the output of the input neurons whereas we use a synapse. They also use the power norm to measure stochastic resonance instead of SNR. The similarities are in the architecture of the network and the fact that each input neuron receives the same input signal and independent noise. The model of Zalanyi et al. is the closest to our model except that they used a depressing synapse instead of a fixed conductance α-function synapse used here.

Each of the input neurons in our model is similar to that described by Plesser and Geisel (1999) and Barbi et al. (2000). The output of each of these neurons can be described as a Poisson process with a time-dependent rate $\lambda_k(t)$ (Hohn, 2000; Shimokawa et al., 1999). Each input neuron k represents an afferent fibre. The membrane potential for each neuron model is reset to its initial value at time $t = 0$, $V(0) = V_0$ after a spike has been generated. A spike is produced only when the membrane voltage equals or exceeds the threshold, V_{th}, which has a potential difference with the reset potential of $\theta = V_{th} - V_0$.

The potential at each of the neurons at any time t is given by the LIF model equation from chapter 2, which we restate here:

$$\frac{dV}{dt} = I(t) - \frac{V(t)}{\tau} \tag{5.1}$$

where for the input neurons $I(t) = I_0(1 + b\cos(\omega t)) + \sigma\xi(t)$ and $\xi(t)$ is a Gaussian white noise as given by Barbi et al. (2000) from independent and uncorrelated sources and σ is the noise amplitude. I_0 and b are constants. At the output neuron

$$I(t) = \sum_{k=1}^{N} a_k I_k(t) \tag{5.2}$$

where

$$I_k(t) = \sum_{i=1}^{n} \alpha_k(t - t_{k_i}) \tag{5.3}$$

where the index $k = 1, ..., N$ denotes the input neurons and the index i denotes the i^{th} input from a particular input neuron k, whose time of arrival is $t_{k_i} (0 < t_{k_1} < t_{k_2} < ... < t_{k_n})$ and n is the number of active spikes from input neuron k. The amplitude of the inputs from fibre k is a_k, which is positive for excitatory postsynaptic potentials (EPSPs) and negative for inhibitory postsynaptic potentials (IPSPs), and the time course of an input at the site of spike generation is described by the synaptic response function $\alpha_k(t)$ for the leaky integrate-and-fire model.

The results presented here are for a network structure with a uniform synaptic response function which is given by (5.4):

$$\alpha_k(t) = \frac{1}{\tau_s} e^{-\frac{t-\Delta^{as}}{\tau_s}} \quad \text{for} \quad t \geq 0, \quad \text{and} \quad 0 \quad \text{for} \quad t < 0. \tag{5.4}$$

where τ_s is the synaptic time constant for synapse k in the millisecond range and Δ^{ax} is the axonal transmission delay for synapse k. Since Δ^{ax} is the same for all the neurons we can ignore it because it will just shift the output signal back in time by the same amount for all the neurons. The membrane potential rises upon the arrival of an EPSP and then decays exponentially between inputs. The decay of the EPSP across the membrane means that the contributions from EPSPs that arrive earlier have partially decayed by the time that later EPSPs arrive. Synaptic response functions with an arbitrary time course may also be analysed, which may allow a more accurate model of the time course of the incoming EPSP. The above synaptic response function is the same for all inputs, but the model allows response functions with different time courses for different input fibres by adding the index k to the synaptic time constant τ_{sk} and the synaptic delay ${\Delta_k}^{as}$. This enables, for example, the effect of propagation of the PSPs along the dendritic tree to be modelled for inputs with synapses at different distances from the site at which the neuron generates an action potential (typically the axon hillock).

However, for simplicity we consider here the situation where the postsynaptic potentials from the inputs are all excitatory and equal in amplitude, i.e. $a_k = a > 0$, and time course,

$\alpha_k(t)$ is the same for all the input neurons. The amplitudes or synaptic weights a were determined by looking at synaptic weights which maximise SNR. Consequently we drop the index k on the amplitude a_k and the synaptic response function $\alpha_k(t)$ in the rest of the discussion, since all input fibre characteristics are identical. The simplified equations for the synaptic current $I(t)$ for the output neuron are as follows:

$$I(t) = a\sum_{k=1}^{N} I_k(t), \quad I_k(t) = \sum_{i=1}^{m} \alpha(t - t_{ki}) \tag{5.5}$$

The synaptic weight a was determined as follows. The network was run with one neuron and the maximum SNR value, SNR_{max} was determined from the SNR versus noise plot. The noise strength σ_{max} which resulted in SNR_{max} was also noted. The network was then run with one input and one output neuron keeping the noise value at σ_{max}, the value that gave us the maximum SNR value for the single neuron case. The synaptic weight was changed until the SNR value for the output neuron equalled the maximum value for the single neuron case. The resulting synaptic weight W (which equalises the SNR at the output with SNR_{max}) was then divided by the number of coincident spikes needed to make the output neuron fire a single spike with a weight of 1. That is, the synaptic weight a for all the synapses is given by: $a = \frac{W}{n}$. The rationale behind this synaptic weight was to ensure that the coupling between the input neurons and the output neuron is weak, that is, a single spike from an input neuron in a multiple neuron network does not cause a spike on the output neuron. Coupling is said to be strong if each spike from the input neurons causes a spike on the output neuron. According to the results reported by Zalanyi et al. (2001) stochastic resonance was stronger for weaker coupling between input neurons and the output neuron than for stronger coupling.

The network model depicted in figure 5.1 can be used to investigate stochastic resonance at the input neurons and at the output neurons. One of the aims is to see how many input layer neurons are needed before we see an improvement of the SNR at the output neuron over the input neurons. This means we can use this network to investigate both stochastic resonance in a single neuron and in a network. Networks of a similar nature have been investigated by other researchers for a variety of reasons which include synchronisation,

coincidence detection and stochastic resonance. We discuss here some of that work.

5.1.3 Related work

Burkitt and Clark (2000) analysed an integrate-and-fire model with periodic input, using a deterministic differential equation without any noise. They found that there were three regions in parameter space (frequency of synaptic inputs, average rate of inputs, synchronisation of inputs, and the time constant of the membrane decay) with distinctly different behaviours: a phase locking region, a region with an apparently non-periodic behaviour, and a region where firing eventually dies out. Work on the effect of a periodically varying rate of input to noisy integrate-and-fire neurons has focused largely upon stochastic counterparts to this approach, using numerical solutions, such as in the study of the temporal acuity and localisation in the Barn owl auditory system (Gerstner et al., 1997) and (Maass and Bishop, 1999).

Much of the recent work on the response of neural systems to noisy periodic synaptic inputs has been motivated by an interest in the applicability of stochastic resonance to these systems.

A further impetus to the study of periodic synaptic input has been the observation of synchronised periodical activity of neurons in various brain areas, such as oscillations observed in the visual cortex (Gray and Singer, 1989). This has led to the hypothesis that synchronised activity of neurons may play a role in brain functioning, such as pattern segmentation and feature binding (von der Malsburg and Schneider, 1986). It has also been proposed that the information contained in a spike sequence could be coded using the relationship between spike times and the oscillation of the periodic background signal. Synchronised or coherent oscillatory activity of a population of neurons is thought to be a vital feature of temporal coding in the brain (Gerstner and Kistler, 2002).

In a recent paper by Kempter et al. (1998), the response of an integrate-and-fire neuron to noisy periodic spike trains is studied and a thorough analysis of the relationship between the input and output rates is carried out. Their results show how the output rates increase

with both the input rate and synchrony, and how it depends upon the neural parameters, such as threshold, the number of synapses and the time course of the postsynaptic response to the inputs. They are also able to identify the conditions under which a neuron can act as a coincidence detector and thus convert a temporal code into a rate code. They identify two parameters, namely the coherence gain (which provides a measure of the mean output firing rate for synchronised versus random input) and the quality factor for coincidence detection (which provides a measure of the difference in the neural response to random and synchronised inputs), which largely characterise the performance of the coincidence detector, and they plot these qualities for representative values of the neural parameters over the range of input vector strengths. However, since their analysis concerns only the output rate, they are not able to predict quantities that depend upon the details of the timing of individual output spikes.

Kempter et al. (1998) also noted that another prominent example where coherent or phased-locked activity of neurons is known to be important is the early auditory processing in mammals, reptiles and birds. Spikes are found to be phase-locked to external acoustic stimuli with frequencies up to 8 kHz in the Barn owl. In the Barn owl and various other animals, the relative timing of spikes is used to transmit information about the azimuthal position of sound source. In doing this task, the degree of synchrony of two groups of neurons is read out and transformed into a firing rate pattern, which can then be used for further processes and to control motor actions. The essential step of translating a temporal code into a rate code is performed by neurons that work as coincidence detectors.

In the same paper Kempter et al. (1998), focus on the question of whether the task of transforming a spike code into a rate code can be done by a single neuron. The issue of how neurons read out a temporal structure of the input and how they transform this structure into a firing rate pattern has been addressed by several authors and is attracting an increasing amount of interest. Konig et al. (1996) have argued that the main prerequisite for coincidence detection is that the mean ISI is long compared with the integration time that neurons need to sum synaptic potentials effectively. The importance of the effective membrane time constant of neurons has also been emphasised by Softky (1994).

5.1.4 Comparison of our model with previous work

Our interest in this network model is slightly different from the focus of previous work. In general we are interested in investigating stochastic resonance in a network of parallel identical LIF model neurons synapsing onto an identical LIF neuron model. This network architecture is general enough to apply to many areas of the brain. The homogeneity is a simplification for modelling purposes and we in no way suppose that it has any correlates in real brain cells. The network architecture allows one to look at stochastic resonance based on both spike-based stimulation (representing stochastic resonance in neurons in the higher areas of the brain) and continuous stimulation (representing stochastic resonance in sensory neurons). We are also able to look at how stochastic resonance at the output neuron is affected by the number of input neurons. All these aspects are investigated on two platforms: integer (digital) and floating point and the results of these two platforms are compared. The issue of synchronisation is investigated by looking at the changes in the height of the leading peaks of ISIHs as the noise amplitude is changed. To investigate all these issues we set up the network model in figure 5.1 in Java (floating point) and Handel-C (integer). As for Java we do computations at two resolutions: 32 bits (float type in Java) and 64 bits (double type in Java). On the integer platform we do computations from resolutions of 32 bits downwards. We will discuss each of the models in turn in the next section.

In this section we introduced the network model that is at the core of this thesis. We noted that a network of a similar structure has been looked at by a number of researchers previously, with a variety of interests which includes the degree of synchrony in the synaptic input to a neuron, coincidence detection and stochastic resonance. In the next sections we are going to discuss how this network model was implemented using floating point representation. We will also introduce an integer model which will be implemented on an FPGA. The floating point model would take up too much chip area if implemented on FPGA and would not result in real-time performance either. The integer model will allow us to build larger networks with possibilities of real-time performance on the FPGA. FPGA

chips are used because they are cheaply available off-the-shelf compared with ASICs.

5.2 Floating point model

To simulate a network of LIF neurons directly, differential equations coupled by current pulses must be integrated numerically. Important things to consider when integrating differential equations numerically are which numerical method to use and how large the time step can be before the accuracy of the integration becomes unacceptable. In this section we start by discussing the floating point model in general followed by a sub-model that is derived to narrow the gap between floating point representation and integer representation. The section concludes by discussing the implementation of the floating point model in Java.

The floating point model implements an Euler approximation of the LIF neural model given by the following equation 5.1 repeated here:

$$\frac{dV}{dt} = I(t) - \frac{V}{\tau} \tag{5.6}$$

where the parameters are as defined before. The Euler approximation of the equation is given by:

$$V(t + \Delta t) = V(t)(1 - \frac{\Delta t}{\tau}) + \Delta t \times I(t) \tag{5.7}$$

The simple Euler method was used because it results in a simple model which does not take up too much chip area when implemented on the FPGA. Remembering that $I(t)$ is a sum of a stochastic component $\xi(t)$ and a sinusoidal component $I_0(1 + m\cos(\omega t))$, for the input neuron we need to check if an Euler approximation was the right choice of approximation method. Given that $\xi(t) \sim N(0, 1)$ i.e. Gaussian white noise ordinarily we would not be able to numerically solve equation 5.6 using an Euler approximation.

We are justified in using an Euler approximation because $\xi(t)$ is bandlimited. This means that $\xi(t)$ reduces to a step function, i.e. the random function

$$\xi_{step}(t) \equiv \xi(t_i) \quad \text{for} \quad t \in [t_i, t_{i+1}) \quad (i \in I\!N) \tag{5.8}$$

is piecewise continuous if $t_{i+1} - t_i \equiv \delta t > 0$. Thus as long as the numerical integration step-size $\Delta t > \delta t$, numerical integration will work.

The noise is bandlimited because we are only interested in noise frequencies which are not more than the signal frequencies we are interested in. The highest frequency that we deal with is limited by the value of the absolute refractory period t_{ref} because the neurons cannot fire faster than the reciprocal of t_{ref}. For non-bandpassed noise one will have to use computationally intensive techniques for numerically solving stochastic differential equations like the one described by Mannela and Palleschi (1989).

Next we introduce a discretised version of the floating point model that we have just described.

5.2.1 Discretised floating point model

The distance between a floating point model and an integer model is too big for the results to be compared directly. In order to narrow the gap between the two models and also to facilitate our own understanding of what the integer model was doing we decided on a model which is somewhere between floating point and integer models. We decided that the most important part of an LIF model's subthreshold dynamics is the evolution of the membrane potential, rather than the reset. We decided to discretise the evolution of the membrane potential of the floating point model by forcing it to evolve in discrete steps.

If we define V_{fp} to be the membrane potential for the floating point model and V_{fpd} to be the membrane potential for the discretised floating point model and V_{int} to be membrane potential value for the integer model, then

$$V_{fp} \epsilon [V_0, V_{th}] \tag{5.9}$$

and:

$$V_{int} \epsilon [0, 2^N - 1] \tag{5.10}$$

where V_0 is the reset voltage, V_{th} is the threshold voltage and N is the resolution of the integer model. We then define γ to be the discretisation step of the membrane potential

for the floating point model V_{fp} can take scaled by the number of discrete points 2^N.

$$\gamma = \frac{V_{th} - V_0}{2^N} \tag{5.11}$$

where γ is the discretisation step.

Let r be the value by which the discretised membrane potential changes, then the activation is discretised as follows:

$$r = \left\lceil \frac{I_{leak} + I_{in}}{\gamma} \right\rceil^+ \tag{5.12}$$

Then the discretised membrane potential V_{fpd} takes the following values:

$$V_{fpd} \epsilon [V_0, V_0 + 1\gamma, V_0 + 2\gamma, V_0 + 3\gamma, ..., V_0 + 2^N\gamma] \tag{5.13}$$

$\left\lceil .. \right\rceil^+$ denotes the nearest highest integer. The evolution equation for the membrane potential then becomes

$$V_{t+1} = V_t + r\gamma \tag{5.14}$$

where $r\gamma \leq f(I_{leak} + I_{in}) \leq (r+1)\gamma$.

In this section, we introduced the floating point model which will be implemented to realise the network given in figure 5.1. We also justified why we decided to use an Euler approximation for numerically solving the noisy LIF neuron model equation. The section concludes with the description of a discretised model version of the floating point model which is intended to narrow the gap between the full floating point model and the integer model. In the next section we introduce the integer model which will be implemented on a digital hardware platform.

5.3 Integer model

In this section we introduce the integer model which is a transformation of the Euler approximation of the floating point model. This was necessary because we want to use FPGAs, and with floating point representation and computation, the circuitry would be much too large when implemented on the FPGA.

5.3.1 Model transformation

The integer model is derived from the Euler approximation of the floating point model. The floating point model is made suitable for implementation in Handel-C on the FPGA by transforming it into an integer model as follows: (i) making all values integers and (ii) converting floating point operations like division and multiplication into division and multiplication by powers of 2 using bit-shifting. $\frac{\Delta t}{\tau}$ is expressed as 2^l (we call l, an integer, the leakage factor of the FPGA implementation) and Δt (in the last term $\Delta t \times I(t)$ of equation 5.7) is expressed as 2^i, where i is an integer as well. This is because integers take less space to represent and full integer division is expensive to implement (both in number of gates and speed) whereas bit-shifting is quite fast. In this work a full integer division resulted in the LIF neuron requiring 87,236 gates compared to 10,718 gates when bit shifting is implemented. For a neuron with membrane time constant $\tau \equiv 10^{-n}s$ and time step $\Delta t \equiv 10^{-k}s$ we have the following linear approximation:

$$\begin{aligned} V(t+\Delta t) &= V(t)(1-\frac{\Delta t}{\tau}) + \Delta t \times I(t) \\ &= V(t)(1-\frac{1\times 10^{-k}}{1\times 10^{-n}}) + 1\times 10^{-k}\times I(t) \\ &\approx V(t)(1-\frac{1\times(1\times 2^{-3k})}{(1\times 2^{-3n})}) + 1\times 2^{-3k}\times I(t) \\ &= V(t)(1-\frac{1}{1\times 2^{3(k-n)}}) + \frac{I(t)}{2^{3k}} \end{aligned} \tag{5.15}$$

In integer form the above equation becomes becomes:

$$\bar{V}(t+1) \approx \bar{V}(t) - (\bar{V}(t) >> (3(k-n))) + (\bar{I}(t) >> (3k)) \tag{5.16}$$

where $>>$ denotes bit-shifting to the right which is division in powers of 2. Equation 5.16 is the integer model which was implemented on the FPGA using the language Handel-C.

In this section we have described some of the models which were used in investigating stochastic resonance on an integer platform. We showed how the integer model was derived from the floating point model which would later on allow us to compare the results from both models. In the next sections we will describe how these models were implemented.

5.4 Implementation of the floating point model

In this section we discuss how the floating point model was implemented. We begin by describing the implementation of the floating point model in Java. The floating point model is implemented in two forms: a 32 bit implementation using the float type in Java and a 64 bit implementation using the 64 bit double type in Java. This was done so as to have a balanced comparison between the 32 bit floating point implementation and 32 bit integer implementation.

5.4.1 Floating point implementation: Java classes

The network model depicted in figure 5.1 was set up in Java using three classes which are shown in figure 5.2 in UML-like format. Each neuron is an object and each synapse is an object. The network itself is an object which consists of many neuron objects connected by synapse objects. There are two types of neuron objects which are an input neuron object, and an output neuron object. These two objects are similar and they are distinguished by the type of input that they receive. As we saw earlier in the chapter, input neurons receive continuous stimulation while the output neuron is stimulated by spike trains from the input neurons. We used Java because we needed a network of homogeneous neurons and synapses. It was much easier to use Java classes and then put the network together as a composite object. The use of Java also ensured that the network size can be changed easily by adding or deleting objects as it becomes appropriate. This allows us to investigate the network dynamics as the number of input neurons is changed. Next we give a brief description of each class and its functions.

Network class

There is only one object of type network. Its job is to set up the network once it is given the number of neurons that make up the network. The network object is the one that creates the neuron and synapse objects. It receives the number of neurons and the initial randomisation seed as parameters and it assigns one neuron object as the output

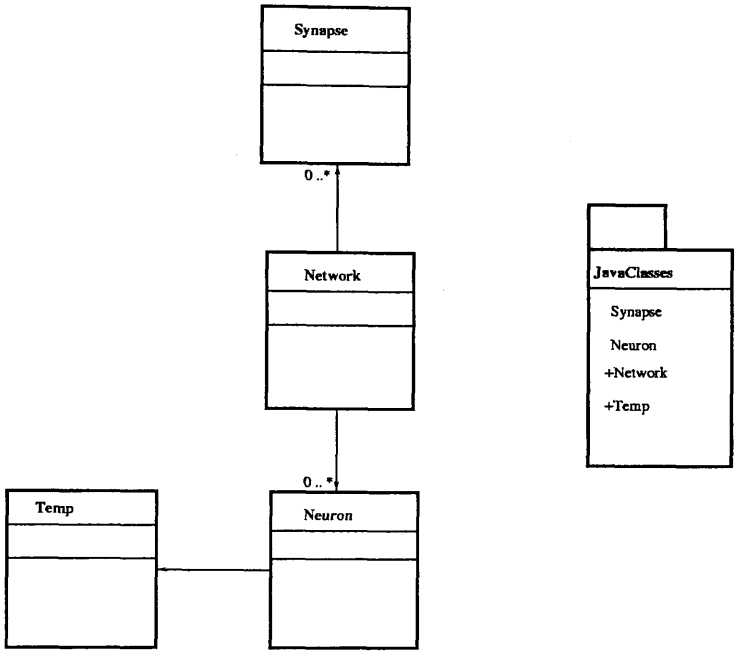

Figure 5.2: Class diagram

object and the rest as input neuron objects and assigns each one of them a randomisation seed and also links each neuron object to a synapse object which is in turn linked to the output neuron object. It then sets up the network and generates the sinusoidal signal. The network object collects results from all the objects and writes them into appropriate files for further processing in Matlab.

Neuron class

The neuron class is the one which implements the LIF neuron model given by equation 5.7. There are two types of neuron objects: an input neuron and an output neuron. The two neuron objects are instances of the same Neuron class, they are differentiated by a flag. Each input neuron object computes its own noise from the seed that it gets from the network object. The noise and the sinusoidal current from the network object serve as the input to each input neuron object. The output spikes of each input neuron object pass through a synapse which converts them to a current which is then integrated by the

output neuron object.

Synapse class

Each input neuron object is linked to the output neuron object via an α synapse object which converts the output spikes of the input neuron object that it is connected to into a current which is the input to the output object. The synapse object turns these spikes into a current using an α function. Each synapse object keeps a vector of active spikes which contribute current according to the α function. Each spike has a life span of 100 ms in an active spike vector after which it is removed from the vector of active spikes.

Temp class

The class Temp is just an auxiliary class for implementing static variables during the generation of the noise. Each neuron object is linked to a Temp object.

5.4.2 Floating-point number representation in Java

Real numbers in Java are represented with *float* and *double* data types. Float is 32-bit, single precision floating point value, and double is a 64-bit, double precision floating point value (Flanagan, 2002). Both types adhere to the IEEE 754-1985 standard, which specifies both the format of the numbers and the behaviour of arithmetic for the numbers. The default floating point type in Java is double. To include a float value literally in a program, one has to follow the number by the character f or F:

```
double d = 9.07E12;
float f = 9.07E12f;
```

Most real numbers, by their very nature, cannot be represented exactly in any finite number of bits. Thus, it is important to stress that float and double values are only approximations of the numbers they are meant to represent. A float is a 32-bit approximation, which results in at least 6 significant decimal digits, and a double is a 64-bit approximation, which results

in at least 15 significant digits. In practice, these data types are suitable for most real-number computations and we assume that our neural modelling work is no exception.

5.4.3 Floating point Java models

We implemented two floating point Java models, one with float and one with double. The double data type is the default type so we first implemented the network using the double type which means the resolution was 64 bits. We then converted this network to a 32 bit float type by type casting. The 32-bit floating point Java model is just a conversion of the 64-bit floating point Java implementation by making sure that all values are cast to float values because without casting the values would revert to being double. The 32-bit floating type network is the best candidate to compare with 32-bit integer resolution on the hardware platform.

The Java implementation of the floating point model was discussed in this section. Because of there being two main floating point type in Java with different resolutions we implemented two floating point models one for the double data type and the other for the float data type. In the next section we discuss the implementation of the integer model.

5.5 Integer model implementation

In this section we discuss the implementation of the integer model. The hardware on which the integer model was implemented will also be described including the implementation itself. The integer model was implemented in Handel-C which runs on an FPGA that sits on an RC1000-PP board. In this section we will discuss the board layout, the FPGA used and the implementation of the network model in the Handel-C language. Handel-C allows us to define integers of different lengths which allowed us to do computations at different resolutions. An FPGA was chosen because it is a good prototyping platform for ASIC designs because of its reprogrammability and reconfigurability. An FPGA also offers shorter system development time, and also design flexibility and it is a standard

off-the-shelf product nowadays.

5.5.1 RC1000-PP board

This board provides very high performance real-time integer processing capability and is designed to be fully supported by the Handel-C language tools. These tools enable a software engineer to directly target the FPGA in a similar fashion to classical microprocessor cross-compiler development tools, without recourse to a Hardware Description Language (HDL). This allows the software or hardware engineer to realise simply the raw processing capability of the FPGA. The board has the following main features:

- PCI bus to host for the embedded system;
- Advanced high-capability FPGA;
- 8 Mbytes of SRAM for dynamic data storage accessible to host and FPGA; and
- 2 × P MC daughter-board sites and a 50 pin unassigned header for I/O.

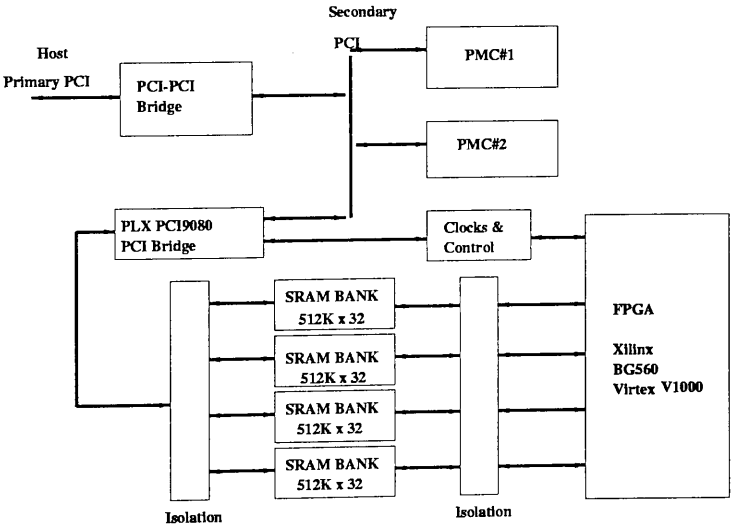

Figure 5.3: The RC1000-PP board layout

Description of the board

The RC1000-PP hardware platform is a standard PCI bus card equipped with a Xilinx Virtex FPGA XCV1000 BG650 part with up to 1,000,000 system gates (see figure 5.3). It has 8 Mbytes of SRAM directly connected to the FPGA in four 32 bit wide memory banks. The memory is visible to the host CPU across the PCI bus as if it were normal memory. Each of the 4 banks may be granted to either the host CPU or the FPGA at any one time. Data can therefore be shared between the host CPU and the FPGA by placing it in SRAM on the board. It is then accessible to the FPGA directly and to the host CPU either by DMA transfers across the PCI bus or simply as a virtual address.

The board is equipped with two industry standard PMC connectors for directly connecting other processors and I/O devices to the FPGA. A PCI-PCI bridge chip also connects these interfaces to the host PCI bus, thereby protecting the available bandwidth from the PMC to the FPGA from host PCI bus traffic. A 50 pin unassigned header is provided for either inter-board communication, allowing multiple RC1000-PPs to be connected in parallel or for connecting custom interfaces.

The software support provides Linux (Intel), Windows98 and NT4.0+ drivers for the board, together with application examples written in Handel-C, or the board may be programmed using Xilinx Alliance Series and Foundation Series software tools and the EDA tool.

The RC1000-PP board comes with software to support communication between the host (host support software) and the FPGA (Handel-C support software). The host support software enables one to identify the card, configure the FPGA, communicate with the FPGA, initialise the hardware and software, set the clock, do data transfer and clean up the board. The Handel-C support software enables one to write FPGA programs in Handel-C and communicate with the host. Next we discuss some of these processes and how they are realised.

Identifying an RC1000-PP card

Each RC1000-PP card has two means of identification. Firstly, the serial number which is factory programmed into the card and is guaranteed to be unique across all the cards produced. Secondly, a card ID is programmed into the card. This card ID can be set by the user with the *setid* utility and is the preferred means of identifying a card. The host support software uses the card ID to identify a card when creating a handle for it. It also provides a function for querying which cards are in the system.

Initialising the hardware and software

The first step for any program is to initialise the RC1000-PP hardware and support software. This is done by calling the PP1000OpenCard() function specifying the the card's ID. This function will return a handle which must be used to identify the RC1000-PP board in future calls to the support software.

Setting the programmable clock

The RC1000-PP has two programmable clocks as two of the clock sources for the FPGA (the others being a fixed bus clock and a clock input connector). The RC1000-PP support software provides a function to set the programmable clock period for the FPGA designs that use the clock. This rate must be set before configuring the FPGA to ensure that the design is not over-clocked when it first starts.

Configuring the FPGA

The host support software provides a set of functions to configure the FPGA on an RC1000-PP board. There are three ways of configuring from FPGA image files:

- Configuring directly from a file
  - This method requires only a single function call but must re-load the configuration file from disk each time the FPGA is configured. Configuring directly from

a file is done using the PP1000ConfigureFromFile() function.

- Loading an FPGA image file into memory ready for configuring later
  - This method allows the configuration file to be loaded once and used many times. PP1000LoadFile() is the function for loading the image file into memory and PP1000ConfigureFPGA() is the function for actually doing the configuration later.
- Including a static array containing configuration information in the host code ready for configuration later
  - This method removes the need for separate configuration files and allows the configuration to be linked in with the host executable. The image file can be registered with support software using the PP1000RegisterImage() function and the PP1000ConfigureFPGA() function can be used to do the actual configuration whenever it needs to be done.

The first option is the simplest, but requires re-reading the configuration file every time the FPGA is configured which can be a significant overhead when frequently re-configuring the FPGA. The second option overcomes this limitation while the third option allows fast configuration without the need for multiple files. The third option means that a single executable can be used which contains the host program and the FPGA configuration information. The RC1000-PP support software package contains a utility called *gencfg* to generate static arrays from FPGA configuration files to help with the third option above.

Communicating with the FPGA

There are three methods of communicating with the FPGA.

- Single bit signalling using 2 pins on the FPGA
  - This method can be used to signal a state to the FPGA or to the host.

- Single byte data transfer using control/status ports on the RC1000-PP board.
  - This method can be used to send short control messages to the FPGA or short status messages from the FPGA. The RC1000-PP has a single byte wide port in either direction between the host and the FPGA. This port can be used to send short messages as control messages or status bytes between the two parties
- Bulk data transfer using the DMA controller and the banks of SRAM
  - This is the recommended method for large data transfer. The RC1000-PP has up to 4 banks of SRAM fitted. Each bank can be granted to either the host or the FPGA (but not to both) at any one time. When a memory bank is granted to the host, the DMA controller can transfer data between host memory and the SRAM memory bank. When a memory bank is granted to the FPGA, it can access the data in memory to read source data from the host or fill in return data to the host. One of the two methods of communication discussed above can be used to synchronise the swapping of the ownership of a memory bank. The transfer can be either a contiguous block of memory (1D transfer) or a series of short transfers with gaps in between (2D transfer). The 2D transfer is intended for transferring image data.

5.5.2 FPGA mimicked in Java

Despite the availability of a simulation tool in the DK1 development environment we still went ahead and simulated part of the FPGA model in Java as a way of testing the FPGA code. The simulator was found to be in adequate because it does not allow one to read-in external values. This means that signals had to be generated on the FPGA which we could do for the sinusoid but could not be easily done for the noise. Noise would require a separate generator and the noise would have to be read-in from outside which the Handel-C simulator cannot do. The mimicked model is just an implementation of the integer model in Java. Instead of bit-shifting we used integer division by powers of 2 in Java.

This implementation was done to make sure we understood and knew what the FPGA was doing. We could not simulate the whole model including different integer lengths. We just implemented the 16 bit integer type (one of the basic data types in Java) and compared the results with 16 bit integer resolution on the FPGA. The result of the comparison is discussed in the model comparison section, section 5.6.

5.5.3 Handel-C implementation

The integer model was first simulated in Java as described above so as to have something to compare the FPGA results with. The model was then implemented in Handel-C. The Handel-C design flow can be illustrated as in figure 5.4. In the design flow, the compiler is involved in the simulate and synthesis steps. Place and route is done by the FPGA manufacturer's tools, in our case Xilinx's Foundation Series 3.2i. The code was thoroughly tested until it produced the desired results. The compiled code was then passed through the synthesis tools to produce a netlist file. The netlist file (defined in section 2.6) was then passed through the Xilinx Foundation Series 3.2i Place and Route tool to produce a bitmap file which is then used to configure the FPGA. Because the whole aim of this thesis was to

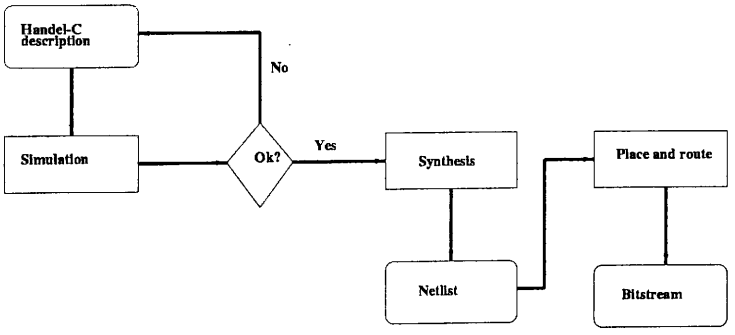

Figure 5.4: Handel-C design flow

compare the results between an integer platform and a floating point platform, the integer implementation had to meet two constraints. Firstly it had to be as close as possible to the floating point implementation so as to have reasonable grounds for comparison. Secondly,

this proximity had to be met using a language which is quite far from Java. These two requirements were met by deriving the integer model from the floating point model using the approximation described in section 5.3.

The integer model was implemented in Handel-C running under Celoxica Ltd's DK1 design suite. DK1 is a development environment which comes with a Handel-C version 3.0 compiler, synthesiser, debugger and a hardware simulator.

Each neuron was implemented as a structure and each synapse was also implemented as a structure. Just like in the Java model all the neurons are identical apart from the type of input they receive. The network is also set up by specifying the number of neurons in the network and then one of them is designated as the output neuron while the rest are made input neurons. Each input neuron is connected to the output neuron via a synapse which implements an α function using a lookup table. The values for the lookup table are taken from the Java model and they are quantised and stored in the lookup table.

The program is parameterised and the parameters are passed via SRAM. The input neurons read the input signals from SRAM where they are put by the host program which loads the FPGA image file. The subthreshold sinusoidal signal and the Gaussian white noise are generated and quantised in Java and stored in files.

The initial idea was to implement the neurons in parallel. However we had to abandon parallel implementation of the synapse model due to technical problems with the FPGA synthesis software. The problem was that the synthesiser in the DK1 development environment allocated a huge number of gates initially and then spent a lot of time trying to optimise that. The huge number of gates just used up all available virtual memory, causing the system to thrash, thereby taking more than 72 hours to compile a network of 7 neurons. This problem severely limited the size of the network that we could implement. As far as we know there is no other synthesis tool for Handel-C apart from the one which Celoxica provides as part of their DK1 development environment. Because we could not implement parallelism we had to abandon the issues to do with timing and real-time processing for the network.

A C++ program is used to load the input signals (noise and subthreshold sinusoidal)

into SRAM using the DMA facility. The data transfer is done using DMA between the host PC and the SRAM on the RC1000-PP board. The same C++ program configures the FPGA and loads parameters like network size and and membrane time constant. The same C++ program reads results from SRAM into files for further processing. The sequence of events for configuring the FPGA using the C++ are as follows:

1. Read the noise and signal into host machine RAM.
2. Load look-up table into host machine RAM
3. Load network parameters into host RAM
4. Install error handler
5. Open the card in the system
6. Request the memory banks
7. Set up DMA channel
8. Do DMA to card
9. Free the DMA channel
10. Configure FPGA from a file
11. Do computation
12. Signal host that FPGA is done
13. Set up read from card
14. Do DMA from card
15. Free the DMA channel
16. Write results into files

This process was repeated for different noise streams and the results stored in files were retrieved and processed in Matlab to get ISIs, power spectra and SNR values.

Signal and threshold quantisation

The floating point model can take inputs directly in floating point. However the integer model requires integer inputs which means the signals for the floating point model had to be quantised. The floating point subthreshold sinusoidal signal and the Gaussian white

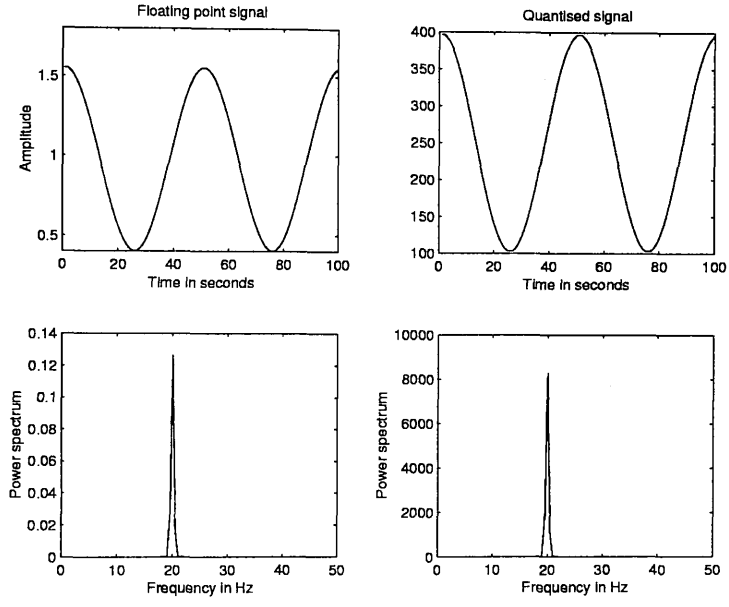

Figure 5.5: 20 Hz input subthreshold signal and power spectra plots. Top row left to right: Floating point input signal and integer input signal. Bottom row: Power spectra of floating point signal(left) and integer signal (right). The quantised signal preserves the frequency spectrum of the signal.

noise used in the Java model are quantised by multiplying them by 2^n (where n is the resolution or the length of the integer). The same is done to the threshold. Quantisation is a non-linear process which means we cannot be sure that we are giving equivalent signals to both models. Since we are investigating stochastic resonance, the most important thing to check is that the frequency spectrum of the signal is preserved to a satisfactory degree. To check that the floating point signal and the quantised integer model signal still compare we go into the frequency domain by way of a Fourier Transform. Both the integer and the

floating point signals are Fourier transformed. Figure 5.5 shows that the integer model preserves the frequency spectrum of the floating point signal.

The implementation of the integer model in Java and on the FPGA was discussed. It was mentioned that the Java implementation of the integer model was just a testing tool for the hardware model due to lack of verification tools in the development environment used. The limited precision aspect of the hardware model was made possible because in Handel-C we can declare integer variables of different widths. In the next section we compare the floating point and integer models.

5.6 Comparison of the models

In this section we will compare all the models that we used. We will start by comparing the two floating point models, i.e. the 32-bit floating point and the 64-bit floating point. The floating point model is compared with the discretised floating point model. The performance of the Euler approximation is compared with the analytical decay to test the suitability of the Euler method. We will also compare the simulated integer model in Java and the one implemented in Handel-C for the same resolution. Lastly we will compare floating point models versus the integer model. To compare the models we looked at the leakage. The membrane potential for the models being compared are initialised to high start values which are close to the threshold values and are allowed to decay without any input. We then plot the decaying potentials for both models and compare them. The time step for the Euler approximation is 0.001s and the membrane time is 0.02s.

5.6.1 32-bit floating point versus 64-bit floating point

Using the following code we generated table 5.2 which illustrates the differences in the degree of accuracy between the *float* and the *double* data types in Java:

```
float eps = 1.0f;
double eps1 = 1.0;
```

```
while ( (float) 1.0 + eps/(float)2.0 > (float)1.0 ){
      eps /= (float)2.0;
      eps1 /=2.0;
       System.out.println("eps = "+ eps+"  eps1 =  "+eps1);
}
```

eps is the smallest number that, when added to 1, yields something larger than 1. The type cast to float was used everywhere to make sure no conversions to double happen. Table 5.2 shows that up to about 7 significant figures there is no difference between float and double data types. Most of the values that we will be dealing with in our simulations will not be affected because they are within this degree of accuracy. This result is confirmed by figure 5.6 which shows no difference in membrane potential values for float and double data types in Java when they are both allowed to decay from high start values without input. When these models are plotted on the same axis they match point for point. As a result, in Chapter 6 we will only consider the 32-bit floating point model for comparison with the integer model.

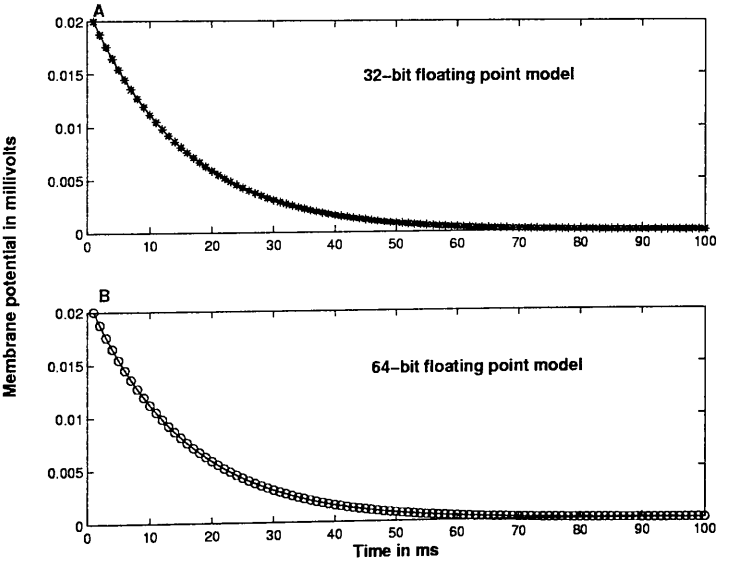

Figure 5.6: Comparison of 32-bit and 64-bit floating point models

5.6.2 Floating point versus discretised floating point

Figure 5.7 shows that the effect of discretising the evolution of the membrane potential is to slow down the rate at which the membrane leaks. The membrane potential for the discretised floating point model leaks to a value which is above that of the floating point model. Initially we thought this might be because we always take the upper bound of $[V - V\frac{\Delta t}{\tau}]$. However, the results of taking the lower bound of $[V - V\frac{\Delta t}{\tau}]$ are similar to those of taking the upper bound. This may be due to the fact that we are using high resolution values (32 bits) for the discretised floating point model.

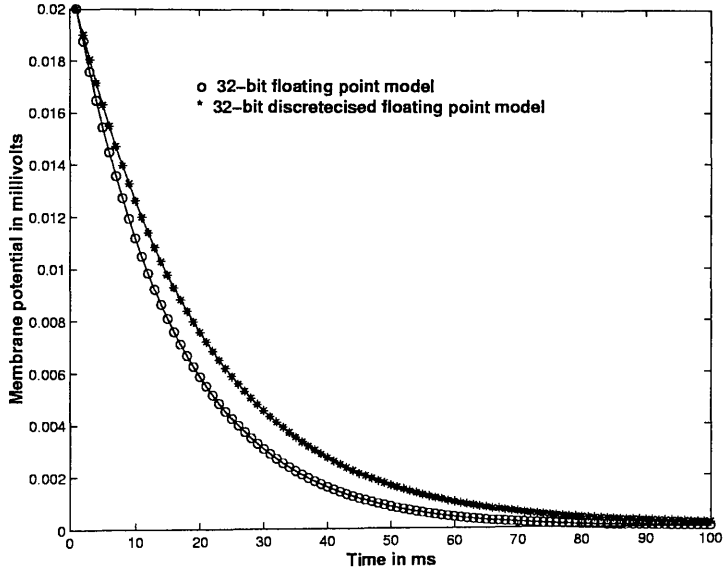

Figure 5.7: Comparison of 32-bit floating model and discretised 32-bit floating point model

5.6.3 Simulated decay versus analytical decay

Allowing the membrane potential to decay without input means that we set $I(t) = 0$ in the LIF equation which means we are left with the following equation:

$$\frac{dV}{dt} = -\frac{V}{\tau} \tag{5.17}$$

which has the following analytical solution:

$$V(t) = Ce^{-\frac{t}{\tau}} \tag{5.18}$$

Given that $V(0) = 0.02$ because we start the membrane from a high start value and let it decay, we have $C = 0.02$ and equation 5.18 becomes:

$$V(t) = 0.02e^{-\frac{t}{\tau}} \tag{5.19}$$

Figure 5.8 compares the decay curves for the analytical solution with the Euler approximated full floating point model. Figure 5.8 shows that there is a difference in the decay

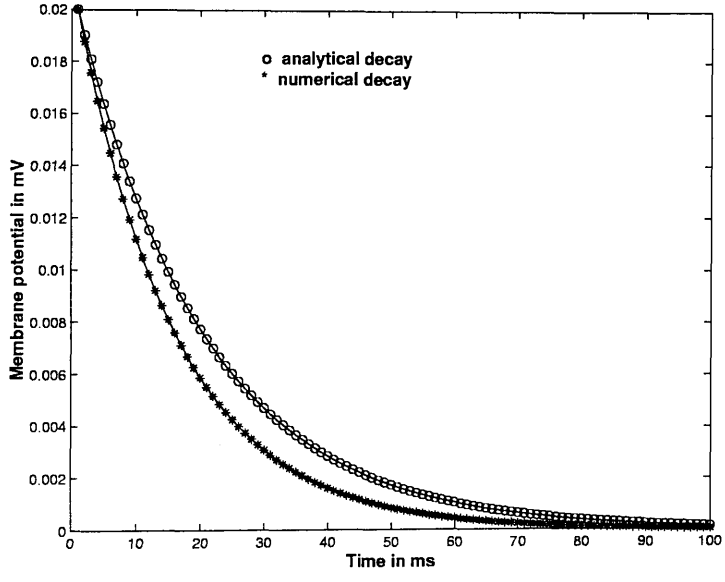

Figure 5.8: Comparison of the numerical decay with the analytical solution decay.

between the analytical model decay and the Euler approximated model decay. The Euler approximated model decays faster than the analytical model. The decay of the discretised model is close to that of the Euler approximated model which means both approximated models decay faster than the analytical model. The difference between the analytical and the numerical decays is not very big and the shape of the decay curves is similar. This means an Euler approximation is a good choice given that it results in a smaller design

on the FPGA. More accurate numerical methods like Runge-Kutta (Nagle and Saff, 1986) could have been used but they were avoided because they would result in larger hardware designs which would result in a smaller network on the FPGA.

5.6.4 Integer model vs floating point model

The integer Java model was developed to test the FPGA model. The results in figure 5.9 show that the FPGA model for one neuron is identical to that of the Java simulation of the FPGA model for a single neuron. This means that our implementation of the integer model on the FPGA was a success.

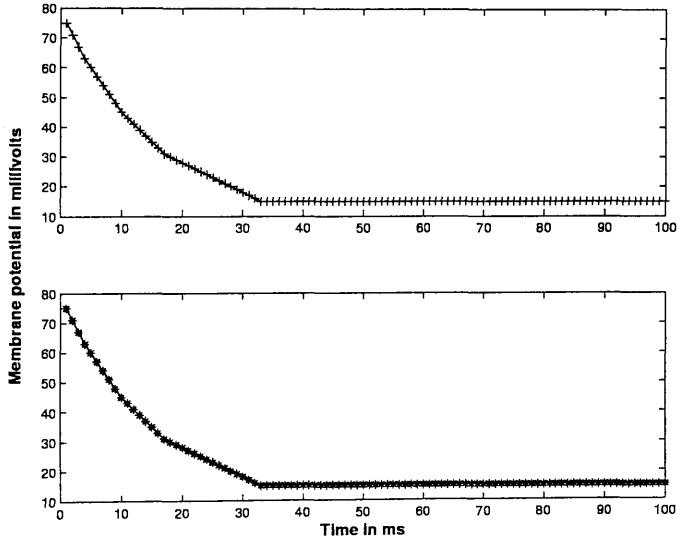

Figure 5.9: Comparison of the integer model (12 bits) in Handel-C (top) with the integer model simulated in Java (bottom)

Next we compare the FPGA implemented integer model with the floating point model. Due to the differences in scale between floating point values and integer values, the decaying potentials for these two models are normalised so that we can plot them on the same axis. As can be seen from figure 5.10 the membrane potential for the floating point model decays to almost zero in the absence of input but that of the integer model decays to a constant

value. This shows that the integer modelled LIF neuron does not completely discharge. This reduces the charging time for the integer model. The effects of this will be discussed in the next section.

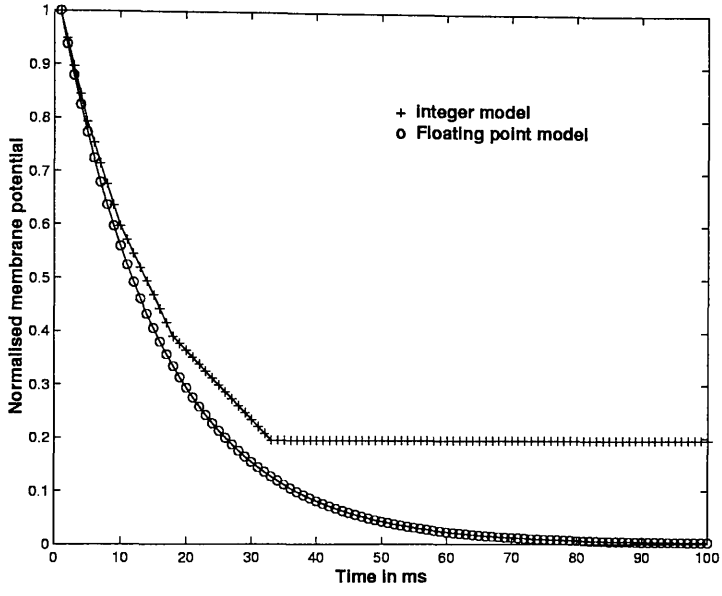

Figure 5.10: Normalised membrane potential decay (from high start values) for the integer (12 bits resolution) and floating point models in the absence of any input.

5.7 Discussion

In this section we will discuss issues to do with numerical accuracy. We will look at the following: (i) the effects of using an Euler approximation; (ii) the likely effect of discretisation on the leakage rate; and (iii) the effects of quantisation on the evolution of the membrane potential.

5.7.1 The time step and the accuracy of the Euler method

To simulate networks of LIF neurons directly, differential equations coupled by different current pulses must be integrated numerically. Two important questions when integrating

differential equations are what numerical method to use and how large the time step can be before the integration becomes unacceptable.

Hansel et al. (1998) investigated the question of how the size of the time step in a network of IF neurons affects accuracy. They found out that the network's synchronisation properties were particularly sensitive to the time step. Hansel et al. found out that with both simple Euler and Runge-Kutta integration methods, a small time step (0.0005s to 0.001s) was required to reproduce the synchronisation properties discovered using the exact method. This was because the discrete simulation time steps meant that the firing times were also discrete which lead to considerable reduction in the accuracy. The accuracy for a given time step could be improved by linearly interpolating the membrane potential between time steps to find the firing time. In our case here we see that the graphs for the analytical decay and both the simulated decays (Euler and discrete) are not too far apart which means that our choice of time step for the simple Euler method was good. A big time step can result in instability for the simple Euler method but this will not be an issue here because we require the time step Δt to be very small compared to the sampling period T so that the neuron can sample the signal adequately.

The leakage rate is coupled to the time step Δt. The relationship between the two is such that the larger the time step, the larger the leakage will be. In our simulations these two values are kept constant. The likely effect of discretisation on the leakage rate is inaccuracies in the precise value of the leakage at each time step which introduces inaccuracies in the evolution of the membrane potential.

5.7.2 Effect of quantisation on activation

To investigate the effect of quantisation on the activation we look at membrane potential leakage. The membrane potentials for both the integer and floating point models are initialised to high start values which are close to the threshold values and are allowed to decay without any input.

We then plot the decaying membrane potentials for both floating point model and

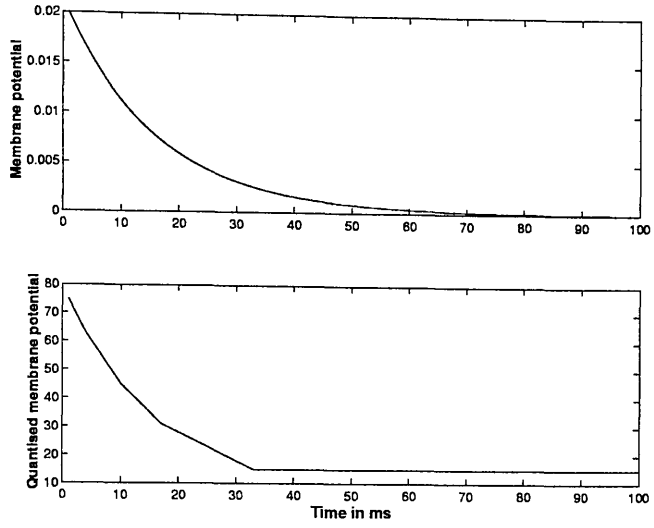

Figure 5.11: Decay of the membrane potential initialised from a high start value with no input for both the floating point model (Top) and the integer model using 12 bits resolution (Bottom).

integer model with limited resolution, see figure 5.11. We also normalise the decaying membrane potentials for the two models and plot them on the same axis (see figure 5.10).

As can be seen from figures 5.10 and 5.11, the membrane potential for the floating point model decays to almost zero in the absence of input but that of the integer model decays to $2^k - 1$ where k is the leakage factor defined in equation 5.16. This means if the leakage factor is 3 the integer neuron will not leak below $2^3 - 1 = 7$ in the absence of input, if the leakage factor is 4 it will not leak below $2^4 - 1 = 15$. The leakage factor is coupled to the membrane time constant τ and the size of the time step Δt. We are therefore limited to resolutions which result in threshold values which are above $2^k - 1$ when quantised. As a result only resolutions which start from 10 bits were considered for the chosen values of the membrane time constant $\tau = 0.02s$ and $\Delta t = 0.001s$. The effect of activation quantisation is that the changes in $V(t)$ over the time step Δt implemented by equation 5.7 are forced into integer values and that this integer value ceases to decrease when the following holds:

$$((V(t) >> 3(k-n)) + (I(t) >> 3k)) < 1 \tag{5.20}$$

(see equation 5.16). This explains why the membrane potential for the integer does not leak to zero in the absence of any input. This leakage problem in the integer model is also coupled to the size of the time step Δt. Decreasing the time step makes the problem worse as $\frac{V(t)\times\Delta t}{\tau}$, the decrement is proportional to Δt. Yet increasing Δt is not normally an option, as we need $\Delta t \ll T$, where T is the period of the input signal. We can get round this problem by introducing some logic to force the membrane potential to decay if there is a prolonged net input of zero. The only problem is that such a mechanism will cost us in terms of logic and yet we do not know what it will buy us in terms of improvement in the results. Signals in real neurons are usually very small so that representing them with anything less than 10 bits would not be a reasonable thing to do.

5.8 Summary

In this chapter we have described the neural network topology at the core of this thesis and all the models that we developed based on it. The models were also compared and it has been noted that for the range of values that we are dealing with the float and the double data types do not show any difference in the way the membrane potential evolves. We also noted that discretising the floating point model as an immediate step before moving to the integer model was important because it showed that as we make the leakage step of the membrane potential coarser, it has the effect of slowing down the decay and also the membrane potential decays to a value which is higher than that of a full floating point model. On comparing floating point and integer models we realised that integer model decays to a constant value depending on the resolution used.

In the next chapter we will present and discuss the results which were realised from running the simulation models that we just discussed in this chapter.

float(eps)	double(eps1)
0.5	0.5
0.25	0.25
0.125	0.125
0.0625	0.0625
0.03125	0.03125
0.015625	0.015625
0.0078125	0.0078125
0.00390625	0.00390625
0.001953125	0.001953125
9.765625E-4	9.765625E-4
4.8828125E-4	4.8828125E-4
2.4414062E-4	2.44140625E-4
1.2207031E-4	1.220703125E-4
6.1035156E-5	6.103515625E-5
3.0517578E-5	3.0517578125E-5
1.5258789E-5	1.52587890625E-5
7.6293945E-6	7.62939453125E-6
3.8146973E-6	3.814697265625E-6
1.9073486E-6	1.9073486328125E-6
9.536743E-7	9.5367431640625E-7
4.7683716E-7	4.76837158203125E-7
2.3841858E-7	2.384185791015625E-7
1.1920929E-7	1.1920928955078125E-7

Table 5.2: Table shows the difference in accuracy between the *float* and *double* types in Java.

Chapter 6

Stochastic resonance in simulated systems

In this chapter we present and discuss the results of the simulations of the models we discussed in Chapter 5. The task we set ourselves at the beginning of this thesis was to investigate whether stochastic resonance relies on the continuous nature of the underlying system or whether a linearly discretised system is able to display stochastic resonance. We also went on to ask if stochastic resonance is realisable in a discrete system, what would be the effect of varying the lengths of the numbers within the simulations on stochastic resonance? In this chapter we will show that we succeeded in realising stochastic resonance in a linearly discretised system using an integer-based model of an LIF neuron on digital hardware for both single neurons and the small network topology given in figure 5.1. The integer model was implemented on an FPGA (using Handel-C) which was described in chapter 5. The floating point model was implemented in Java. Results for stochastic resonance in an input neuron and the output neuron for both integer model (hardware) and the floating point model (software) will be presented and compared. By input neuron we mean any one of the neurons in the input layer of the network topology of figure 5.1 and by output neuron we mean the neuron in the output layer of the same network topology. A statistical technique for carrying out multiple comparisons called repeated measures

analysis of variance described in Winer et al. (1991) and Joiner (1994) will be used to see if the observed differences (on graphs) in stochastic resonance between the integer model and the floating point model and within the different integer lengths for the integer model are statistically significant. The results for a single input neuron will be presented first and the variation at single neuron level between software and hardware will be discussed as well. Secondly, the output neuron results will be presented and discussed and the difference between the hardware and software implementation at network level will be discussed. The input neuron and the output neuron results will also be compared across the two platforms of implementation. The results presented in this chapter can be found in summary form in Mtetwa et al. (2002a) and Mtetwa et al. (2002b). The results show that there is stochastic resonance in the spike trains of both input neurons and output neuron for both the integer and floating point models.

6.1 Methodology

The floating point model was implemented in Java to numerically solve the equation 5.6 (described in Chapter 5) for each neuron. The integer model was implemented by Handel-C code to implement equation 5.16 (also described in Chapter 5).

The Gaussian white noise which formed part of the input was generated using subroutine *ran1* in Press et al. (1995) to generate independent uniform variates, whence the Box-Muller transformation provides the white noise sequence. The stochastic equation 5.6 is then integrated by an Euler approximation with a fixed step of $\Delta t = 1 \times 10^{-3}s$, for a duration of 4,096s. This is obviously a small sample but due to memory constraints on the FPGA that is all that was possible. In order to overcome the problems which may be caused by the small sample size, we repeated each run 12 times with a different noise seed each time assuming ergodicity (see Koch and Segev (1999)).

On the FPGA we could not achieve a network of more than 7 neurons. This limit was caused by the inefficiencies of the DK1 synthesis tools. The problem was that in trying to generate a netlist file, DK1 initially throws up a large number of gates which take up all

the resources of the host machine. For instance, for a network of 7 neurons, DK1 initially allocates about 3 million gates and then spends approximately 72 hours optimising it down to about 147,000 gates. Unfortunately the host machine (Pentium 500 MHz speed, 785,832 Kbytes RAM) could not cope with this initial large number of gates for networks with more than 7 neurons. The relationship between the number of neurons in the network and the number of gates used on the FPGA is nearly linear as shown in figure 6.1. The bit of the graph which is not linear is due to the fact that the gate count includes the synapse module which is shared by all the input neurons. Since the relationship between network size and number of gates is nearly linear we can use simple proportion to estimate the size of network that we could have been able to fit on the FPGA. For our FPGA with 1 million gates we could have been able to fit a network of about 40 neurons given that placing and routing is never 100 percent of all the gates on the chip (Bostock, 1996).

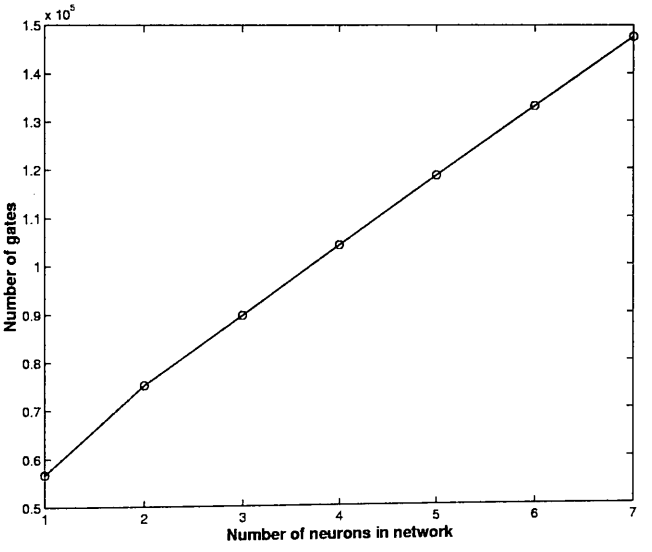

Figure 6.1: This graph shows how the number of gates increases with network size. These results also include the gate count for the synapse module.

This network was run several times with different noise intensity values and different signal frequencies. The spike trains of the input neuron and those of the output neuron are collected and used to calculate the power spectrum and SNR at both the input and output

neurons. The power spectra and SNR are calculated on the resulting spike train of both input and output neurons using the methods discussed in Chapter 4. SNR is calculated using method 3 described in section 4.6.

As mentioned in Chapter 3, there are several measures which are widely used in the literature to quantify stochastic resonance. In this chapter we report results based on just two measures, namely power spectrum and SNR for different noise amplitudes. The interspike interval histograms (ISIHs) have been left out because they cannot be used for quantitative analysis of stochastic resonance. These measures were computed from the spike trains using Matlab.

6.1.1 Network parameters

The results presented here were obtained from simulations of the network in figure 5.1 with the following network parameters. The choice of parameters was motivated by the applicability of the network topology to auditory signal processing and the frequency regimes in which stochastic resonance has been well studied according to literature. Stochastic resonance is well established in the low frequency regime according to Gammaitoni et al. (1998) and Wiesenfeld and Jaramillo (1998). Hence the choice of the 20-30 Hz frequency range. LIF neurons model real neurons which operate on continuous time-varying signals and the choice of time constants must match those of the signals. The membrane time constant τ was motivated by the choice of frequency and also its possible role for sound segregation in auditory scene analysis (Glover et al., 1999). The absolute refractory period t_{ref} was chosen based on values reported in the literature on neuronal modelling (Dayan and Abbott, 2001). t_{ref} sets the limit for the firing frequency of the neuron, the neuron can not fire beyond $\frac{1}{t_{ref}}$ Hz.

- membrane time constant $\tau = R \times C = 20$ ms
- threshold $\theta = 20$ mV
- absolute refractory period $t_{ref} = 10$ ms

- synaptic time constant $\tau_s = 10$ ms
- Gaussian white noise $\xi(t) \sim N(0,1)$

The figures given in this chapter show how the power spectrum and SNR of the spike train changes with an increase in noise intensity.

In this section we described how the simulations were run and how the parameters of the simulations were chosen. In the following sections we will present the results.

6.2 Stochastic resonance in the input neurons

In this section we will present results supporting the existence of stochastic resonance in a single input neuron for both the floating point and integer models. Having discussed the effect of quantisation on the evolution of the membrane potential or activation in the previous chapter, we will now look at power spectra and SNR results of spike trains for both the floating point and the integer model. The results presented in this section are for an input neuron in the network topology of figure 5.1.

6.2.1 Stochastic resonance in an input neuron using the floating point model

Floating point simulation of stochastic resonance in a single LIF neuron with continuous subthreshold sinusoidal stimulation is well established, (see Plesser and Tanaka (1997) and Plesser and Geisel (1999) and Stemmler (1996) for details). We will characterise stochastic resonance in a single neuron using power spectra and SNR plots. Figures 6.2 and 6.3 summarise the results for stochastic resonance in a floating point based input neuron. Figure 6.2 shows that the main peak of the power spectrum (at signal frequency in this case of 20 Hz) of the spike train of a floating point based input neuron goes through a maximum as the amount of noise is increased, a phenomenon characteristic of stochastic resonance. The same phenomenon is emphasised by the SNR plot in figure 6.3 which shows

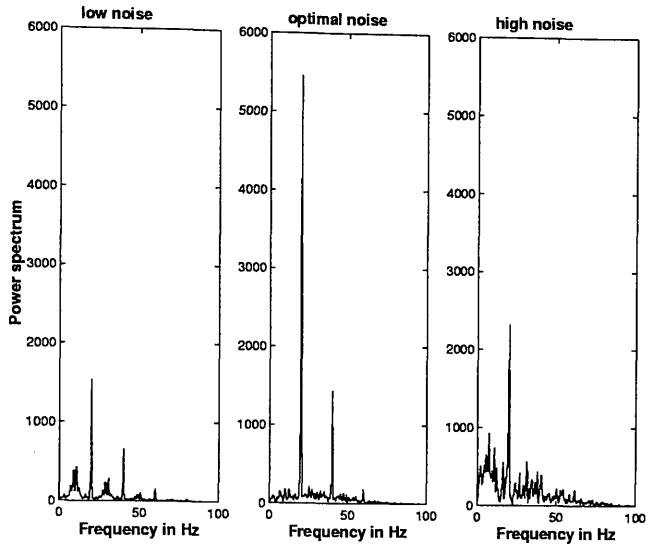

Figure 6.2: Stochastic resonance in the spike train power spectrum for a floating point input neuron stimulated by a 20 Hz subthreshold sinusoidal stimulus plus Gaussian white noise.

the SNR going through a maximum as the noise strength is increased. This shape of the SNR curve has been designated as the signature for stochastic resonance by Barbi et al. (2000). These results confirm what is already well established in the literature.

6.2.2 Stochastic resonance in an input neuron in the integer model

As far as we can determine, neuronal stochastic resonance has not been studied or simulated in limited precision discrete systems. Here we present results of the realisation of stochastic resonance in an integer based input neuron model implemented on an FPGA. We also present results which show the effect on stochastic resonance of varying the length of the integers used in the simulations of stochastic resonance in discrete systems. Figures 6.4 and 6.5 have the same shapes as corresponding figures in the previous subsection. The power spectrum plots (figure 6.4) show the familiar effect whereby the height of the main

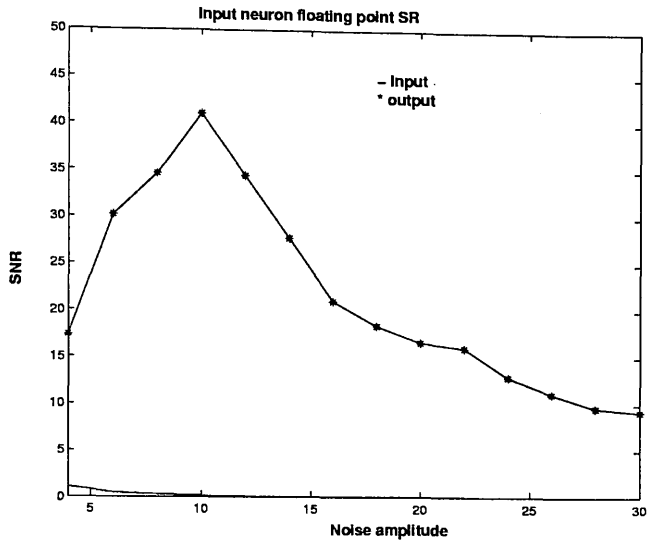

Figure 6.3: Spike train SNR for the floating point input neuron stimulated by a 20 Hz subthreshold sinusoid plus Gaussian white noise.

peak of the power spectrum goes through a maximum as noise strength is increased. This same effect is depicted in the SNR plot in figure 6.5 in which the SNR maximises as noise amplitude is increased. These two figures (6.4 and 6.5) show that stochastic resonance is realisable in a discrete system for a reasonable integer length here 12 bits. As we pointed out in Chapter 2, the issue of limited precision on hardware implemented artificial neural networks is of general interest. In back-propagation type of networks, it is of interest especially during the training phase because the level of precision determines whether the training will converge to a global minimum or not. Higher resolutions would be good for training but they result in smaller networks because chip area is inversely proportional to the resolution. For back-propagation type of networks 12 bits is a good achievement especially during the training phase as reported by the work of Hohfeld and Fahlman (1992).

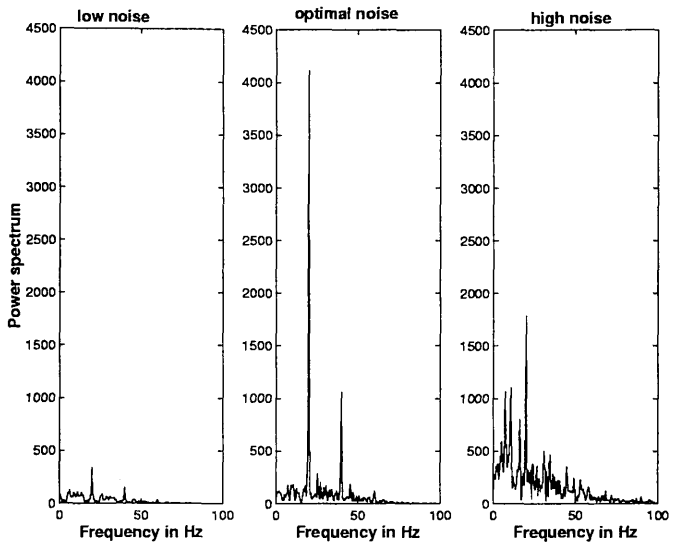

Figure 6.4: Demonstration of stochastic resonance in the power spectrum for an integer-based (12 bits) single input neuron stimulated by a 20 Hz subthreshold sinusoid plus Gaussian white noise.

6.2.3 Effect of limited precision on discrete based stochastic resonance for an input neuron

The effect of limited precision on stochastic resonance is shown in figure 6.7. All the integer lengths considered exhibit the stochastic resonance phenomenon. As explained in Chapter 5, we were only able to realise results starting from 10 bits using the integer model. We observed that stochastic resonance is poor below 10 bits and saturates at 12 bits such that any further increase in integer length does not yield any improvement in stochastic resonance as shown by figure 6.6 (Mtetwa et al., 2002b). The SNR graph for 32 bits is not significantly different from that of 12 bits. The effect of resolution is quite strong at low noise values but diminishes at high noise values and this is due to the quantisation of small numbers. This means that the effect of quantisation noise is stronger at low input noise values. At the high noise end, the effect of quantisation is overridden by the strong input noise. A multiple comparison t-test of the SNR values for the resolutions shows

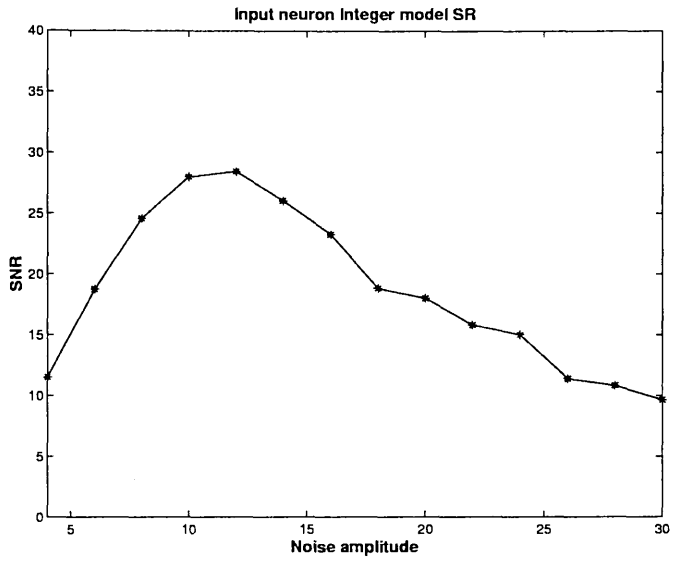

Figure 6.5: Stochastic resonance in the spike train for an integer-based (12 bits resolution) input neuron stimulated by a 20 Hz subthreshold sinusoid plus Gaussian white noise.

that some of the differences in stochastic resonance notable in figure 6.7 are statistically significant at 5% level as shown in table 6.1. The same analysis showed that the floating point model results are statistically significantly different from all the integer resolutions considered. Table 6.1 shows the results of a statistical multiple comparison t-test of the different resolutions' SNR values. For signal processing purposes, we can tolerate noise which is within the low and optimal range up to about noise amplitude 20 because the signal is still clearly detectable in the SNR. It turns out that this is also the range within which the effect of resolution on stochastic resonance in the discrete system is evident. This means that the benefit of increasing the lengths of the integers in the simulations is seen at low and optimal noise levels for the input neurons.

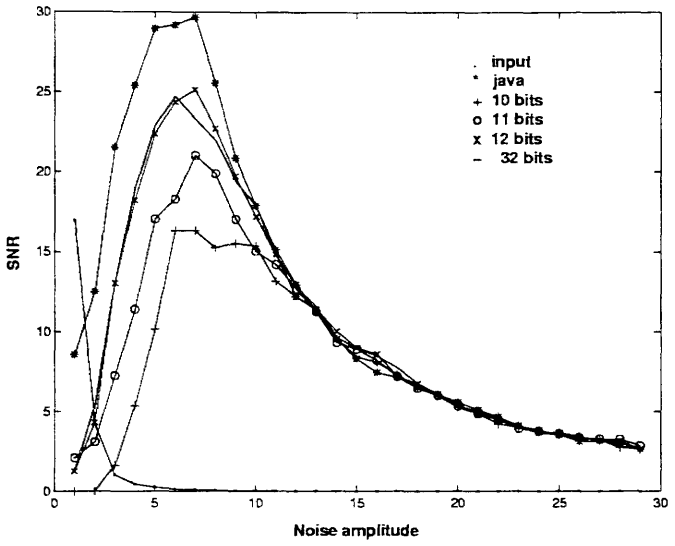

Figure 6.6: 20 Hz subthreshold signal SNR plots for both Java (Floating point) and FPGA (different integer resolutions: 10, 11, 12 and 32 bits as shown on graph) compared with a common input SNR.

6.2.4 Comparison between integer-based and floating point based stochastic resonance for the input neuron

We will now compare the stochastic resonance for the integer model with that for the floating point model for the input neuron. Figures 6.8 and 6.9 show that we have stochastic resonance in both the floating and the integer models though stochastic resonance is stronger in the floating point model than in the integer model for most of the noise values considered. The bottom row of figure 6.8 shows that the peaks in the power spectra for the integer model are always smaller than the corresponding peaks for the floating point model (top row). Comparing the SNR peaks for the integer (see figure 6.5) and floating point (see figure 6.3) models it can be observed that the peak for the floating point model is sharp whereas that of the integer model is broad.

The SNR graphs for the floating point and integer model merge at high noise values as shown in figure 6.9. This shows that at this stage it is noise that is dominating, not the

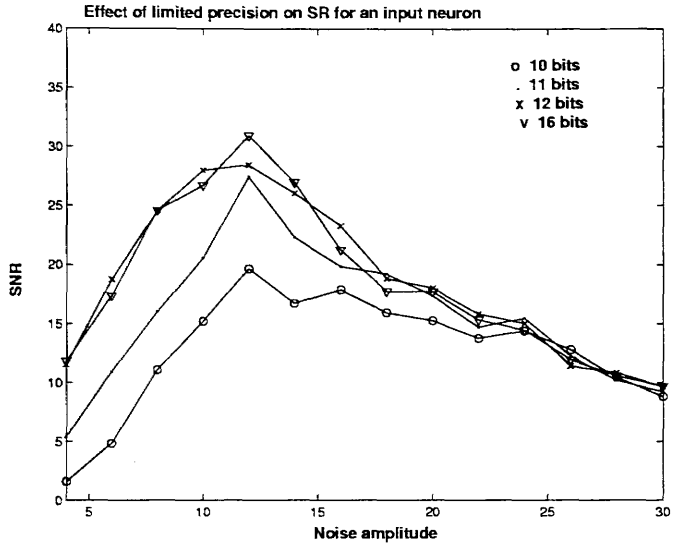

Figure 6.7: Stochastic resonance in the spike train for an integer-based input neuron at different resolutions, stimulated by a 20 Hz subthreshold sinusoid plus Gaussian white noise.

limited precision effect. The fact that both integer model and floating point model SNR values are affected means that the merging of graphs at the high noise end is not due to quantisation either. In comparing these two models, two things must be put in perspective. Firstly, the integer model is an approximation of the floating point model. The second thing is that the integer model has an extra source of noise due to the quantisation process which introduces quantisation noise. Given these two handicaps and also considering the difference in resolution between the two models, it is only fair to say that the integer model is quite comparable to the floating point in exhibiting the stochastic resonance phenomenon.

In this section we have shown that stochastic resonance exists in discrete systems and that it is sensitive to the resolution of the values within the simulations especially at low noise values. It has also been shown that stochastic resonance is stronger in floating point simulations than in integer based simulations. The results presented in this section

resolution in bits	10	11	12	16
10	X	significant	significant	significant
11	significant	X	significant	significant
12	significant	significant	X	not significant
16	significant	significant	not significant	X

Table 6.1: Table showing which resolutions are significant or not. X represents meaningless comparisons. *Significant* means that the two resolutions being compared were found to be significantly different at 5% level. *not significant* means that the two resolutions were found to be not statistical significant when compared at 5% level.

also show that using resolutions as small as 10 bits we can realise stochastic resonance in discrete systems simulated on digital hardware. This could be good news for electronic systems because it means we can build stochastic resonance based systems with real-time performance, something not easily or cheaply achievable with floating point models.

In the next section we will look at stochastic resonance at the output neuron in both the floating point and integer models.

6.3 Stochastic resonance in the output neuron

In the previous section we discussed stochastic resonance in a single input neuron concentrating on the input neurons of the network topology introduced in Chapter 5. In this section we will look at the output neuron of the same topology. As we pointed out in Chapter 5, the difference between these two types of neurons is that the input neurons receive noisy continuous stimulation while the output neuron receives noisy spike-based input from the input neurons. We will first consider the results from the floating point model followed by the integer model. Because the input to the output neuron is spike-based, hence outside our direct control, we cannot ensure that the signal which it receives is subthreshold. This means that the stochastic resonance that we observe at the output neuron does not conform with the requirement that the sinusoidal component of the input

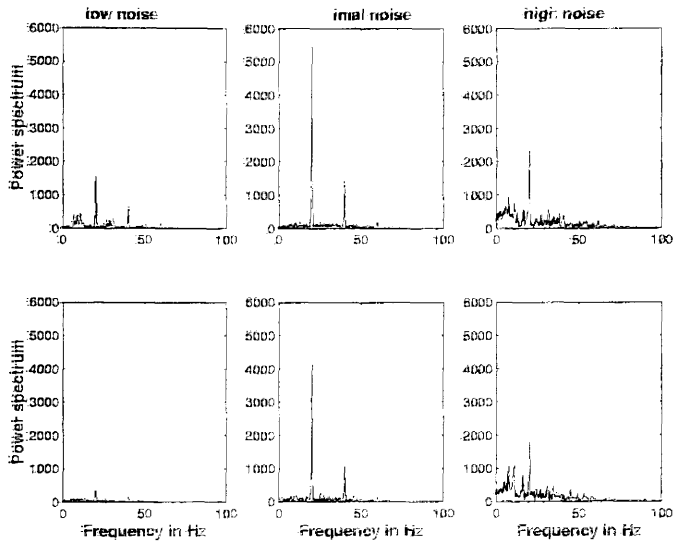

Figure 6.8: Stochastic resonance in the power spectrum. Top: shows typical resonance effect for the floating point model as we move from low noise to high noise values. Bottom: resonance effect in the integer model (12 bits resolution) for the same noise values as in the floating point model. The neuron was stimulated by a 20 Hz subthreshold sinusoid plus Gaussian white noise.

should be subthreshold. Having said this, it is worth noting that if stochastic resonance occurs at all in higher brain areas, as evidenced by the works of Hennig et al. (2001) and Anderson et al. (2000), then it is most likely to be similar to what we get at the output neuron.

6.3.1 Stochastic resonance in the output neuron in the floating point model

Figure 6.10 shows that there is stochastic resonance at the output neuron for the floating point model. The results are for a 5-neuron network. The peak in the SNR here is much broader and less distinct compared to the peak for the input neuron using the same floating point model, see figure 6.3. This could be due to the fact that the output neuron

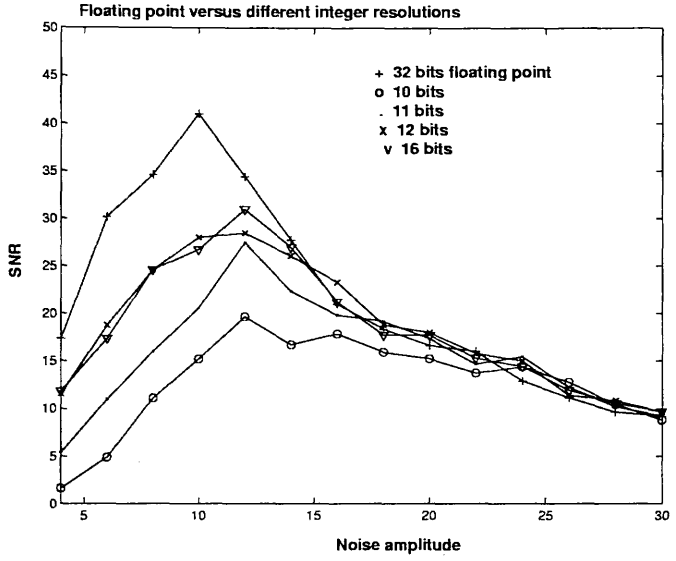

Figure 6.9: Stochastic resonance in the spike train for a floating point and an integer-based input neuron stimulated by a 20 Hz subthreshold sinusoid plus Gaussian white noise.

is averaging the contributions of a small number of input neurons which feed into it. Stochastic resonance at the output neuron increases as we increase the number of input neurons as we will see later on in this chapter.

6.3.2 Stochastic resonance in the output LIF neuron in the integer model

Figure 6.11 shows that there is stochastic resonance at the output neuron of the integer-based (12 bits) neuron just like we observed stochastic resonance at the output neuron for the floating point model. The peak is also broader compared with that of the integer-based input neuron SNR peak.

The effect of limited precision on stochastic resonance is the same at the output neuron as it is at the input neuron. Stochastic resonance increases between 10 bits and 12 bits and ceases to improve after 12 bits as shown in figure 6.12 for a 5-neuron network. The SNR peaks for the resolutions considered are broader and shifted to the right compared to

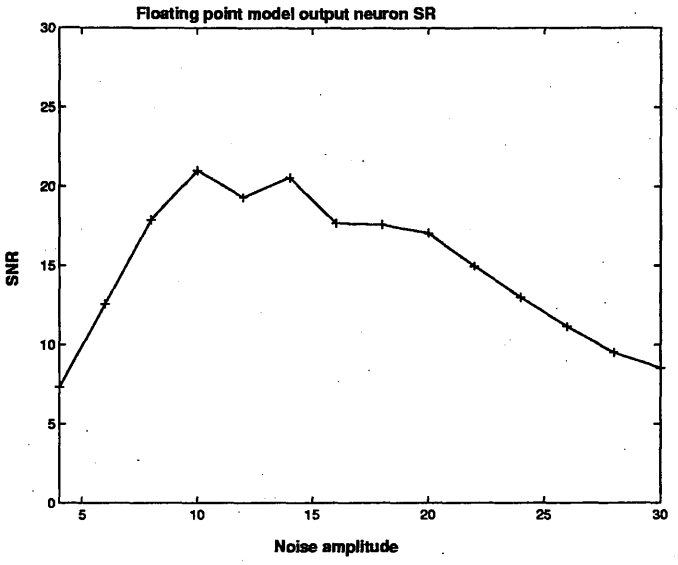

Figure 6.10: Stochastic resonance in the spike train for a floating point output neuron of a 5-neuron network stimulated by the spike trains from the input neurons.

those for the input neuron for the same integer resolutions.

6.3.3 Comparison between integer-based and floating point based stochastic resonance for the output neuron

The floating point model has stronger stochastic resonance than any of the integer model resolutions, this is more distinct at low noise values than at high noise values as shown in figure 6.14. For most of the noise values considered, 12 bits and 16 bits resolutions are marginally better than the floating point model. Looking at figures 6.9 and 6.14 closely, we realise that the difference between the integer and floating point models is more distinct at the input neuron than at the output neuron. The curve for the floating point model SNR in figure 6.9 is well above those of the integer model at low noise values at the input neuron where as the same curve is not so separated from the integer model curves at the output neuron as shown in figure 6.14. Figure 6.13 which shows the plot of [SNR(FP)-SNR(resolution-12)] shows that the difference between floating point and integer (resolution

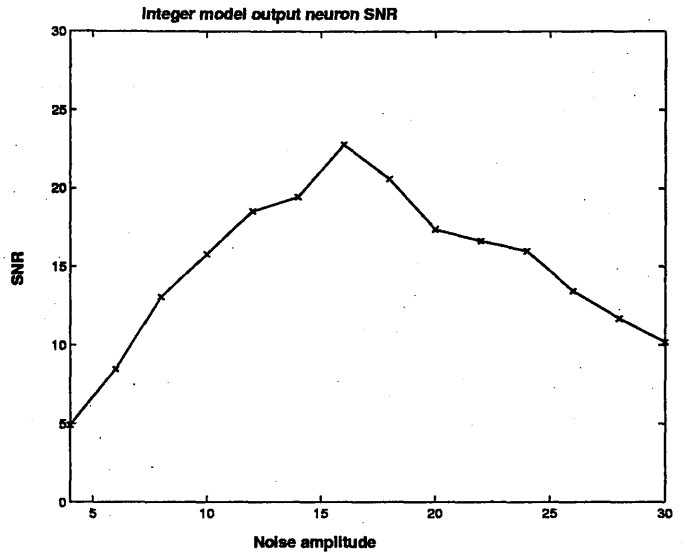

Figure 6.11: Stochastic resonance in the output spike train of a 5-neuron network for an integer-based (12 bits resolution) output neuron stimulated by the spike trains of the input neurons.

12) is greatest for noise values between 5 and 15. The difference is positive within this noise range which confirms the fact that the floating point model is better than the integer model. However, for higher noise values (> 15), the difference is negative which means the integer model is now doing better than the floating point model.

In fact the graphs for 12 and 16 bits integer models have almost the same peak values as that of the floating point model even though these two (12 bits and 16 bits integer) peak at higher noise values than the floating point model. This suggests that there is an interaction between resolution and the number of input neurons at the output neuron because this effect is not observed at the input neuron(Mtetwa et al., 2002a). This result was entirely unexpected and at this point we have no real explanation as to why there seems to be an interaction between the output neuron and the resolution. We suspect that this maybe due to a reduction in quantisation effects because of an increase in resolution and averaging effect of increasing the number of input neurons.

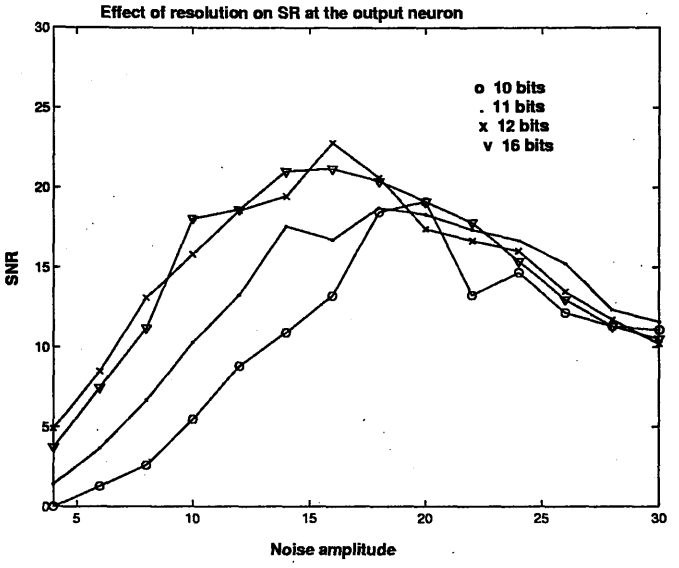

Figure 6.12: Effect of different resolutions on the spike train for an integer-based output neuron (5-neuron network) stimulated by spike trains from the input neurons.

The results presented in this section have shown that there is stochastic resonance at the output neuron in both the floating point and the integer models. The quality of the floating point stochastic resonance is still better than that for the integer model for the noise ranges considered. In the next section we will draw a comparison between the input and output neuron stochastic resonance.

6.4 Comparison between input and output neurons

In this section we will look at the results from a network point of view by comparing the stochastic resonance at the input neurons with that at the output neuron across both platforms. We start by considering the floating point network model stochastic resonance followed by the integer model network stochastic resonance results. We will also consider the effect of varying the frequency of the input subthreshold sinusoidal signal and the subsequent bandpass behaviour of the network on both platforms.

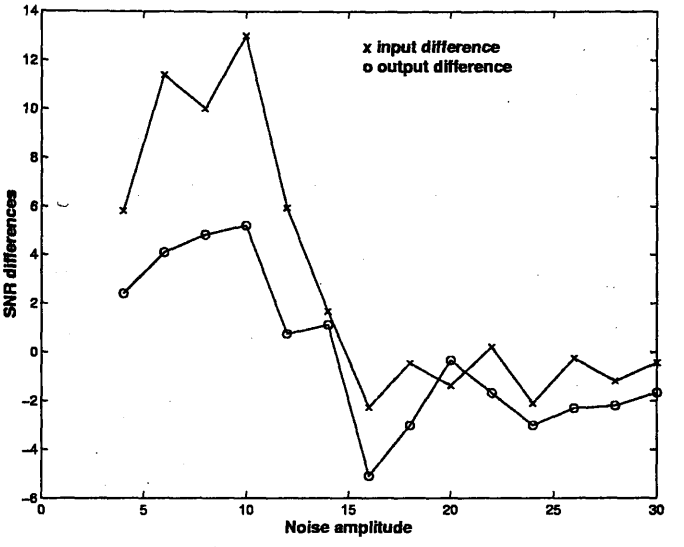

Figure 6.13: A plot of the differences between floating point SNR(FP) and resolution 12 integer SNR(Resolution-12) for both input and output neurons.

6.4.1 Floating point model

Figure 6.15 shows that as the network size increases, the output neuron stochastic resonance gets better than the input neuron stochastic resonance, as shown by the huge difference between the output neuron SNR curves for the 5- and 7-neuron networks. The output neuron SNR curve for the 7-neuron network is well above that for the 5-neuron network for most of the noise values considered. This means that the quality of the signal at the output is better than that at the input neuron for most of the noise values considered as the number of input neurons is increased. Collins et al. (1995b) considered a similar network model to the one looked at here and they reached the same conclusion using a floating point based rate model. Zalanyi et al. (2001) also considered a similar floating point based network with depressing synapses and they also concluded that the SNR at the output neuron increases with an increase in the number of the input neurons. We have shown that the same result can be realised with a much reduced resolution on a platform with real-time capabilities. The other thing to note is that the peak of the SNR curve moves to lower noise values as the number of input neurons is increased as shown by the peaks for

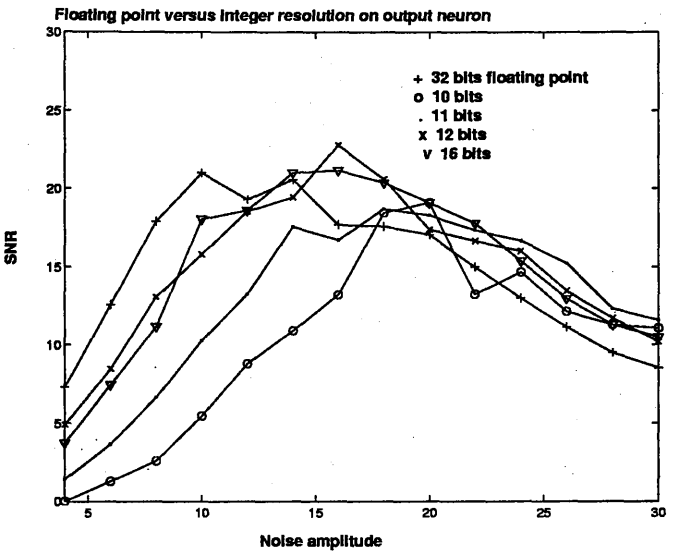

Figure 6.14: Stochastic resonance in the spike train for an integer-based and floating point based output neuron stimulated by the spike trains from the input neurons.

the curves for the 10-neuron and 15-neuron networks in 6.15. Actually, for the parameters used here the peak for a 20-neuron network (not shown in figure) is even further to the left and is sharper than for smaller sized networks. The observation that SNR peaks for higher numbers of input neurons move to lower noise values is probably due to the fact that the weights are fixed which means as the number of input neurons is increased we need less noise to excite the output neuron. The sharpness of the peak for large numbers of input neurons could be addressed by normalising the weights by the number of input neurons. From a network design point of view it is better to have broad peaks because this makes the network tolerant to a broad range of noise amplitude.

6.4.2 Integer model

The same phenomenon observed for the floating point model that stochastic resonance improves as the number of input neurons increases is also observed in the integer model. All the graphs in figure 6.16 show that the output neuron SNR for a 5-neuron network is

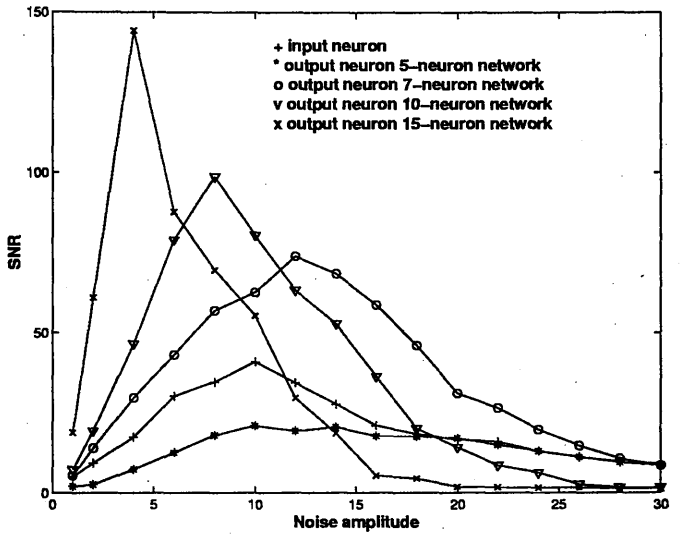

Figure 6.15: Comparison of input neuron stochastic resonance vs output neuron stochastic resonance for different network sizes.

less than that of the input for most of the noise values considered where as that of the 7-neuron network is always above that of both the input neuron and the 5-neuron network output neuron SNR. The effect of resolution is also evident in that the output neuron SNR for the 7-neuron network is much bigger for higher resolutions than for smaller resolutions.

6.4.3 Integer versus floating point model

Looking at figures 6.15 and 6.16 we see that improvement in stochastic resonance between input and output is generally better in the floating point model than the integer model. The output neuron SNR peak for the 5-neuron network is broad compared to that of the 7-neuron network output neuron. The situation at the output neuron for the 7-neuron network is quite interesting. Firstly, the graphs in figure 6.17 are much more separated even at the high noise end than in any of the situations before (at both the input neuron and the 5-neuron network output neuron). This means that the more input neurons we have, the less effect noise has on the signal propagation.

The other thing to note is that even though the graph for the floating point is higher

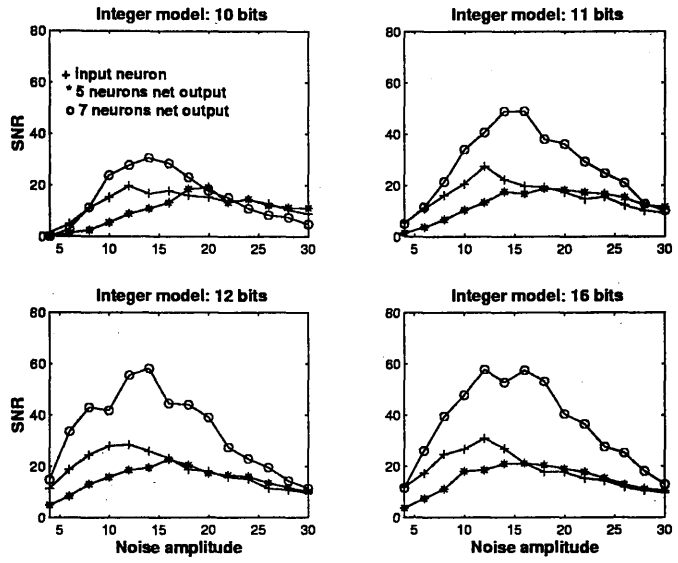

Figure 6.16: Stochastic resonance in the output spike train for an integer-based input and output neurons

than all the different resolutions for the integer model, at high noise values the graphs for 12 and 16 bits overtake it. It is also the case that at the high noise end, 16 bits resolution is better than 12 bits resolution. This is interesting because at the input level, 16 bits is not significantly different from 12 bits resolution (see table 6.1). This reinforces the assertion that there is an interaction between resolution and the number of input neurons at the output neuron. This interaction was entirely unexpected. This means that the effect of resolution on stochastic resonance at the input neuron is different from that at the output neuron. Figure 6.17 suggests that there is an advantage after all in using higher resolutions in the simulation of stochastic resonance in discrete systems and this advantage is better realised at the output neuron of our network topology. This graph also shows that the effect of resolution is better observed at the output neuron when the number of input neurons is increased.

The results presented in this section have shown that there is stochastic resonance in both the input neuron and the output neuron implemented on both platforms. It was shown that the stochastic resonance at the output neuron increases with the number of

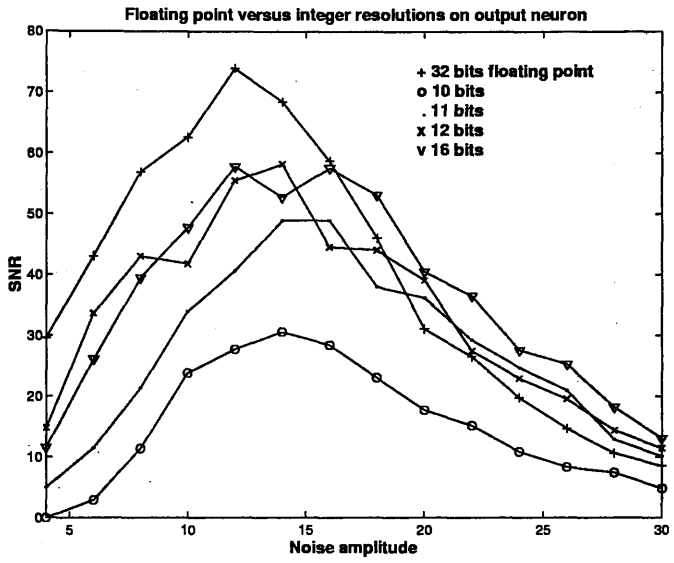

Figure 6.17: Comparison of floating point and integer models at the output neuron of a 7 neurons network.

input layer neurons for both models. However, it was noted that higher resolutions in the integer model outperformed the floating point model at higher noise values. In the next section we will present some results which show the bandpass properties of an LIF neuron with stochastic resonance.

6.5 Bandpass properties of stochastic resonance

As pointed out by Plesser (1998b), an LIF neuron exhibiting the stochastic resonance phenomenon displays bandpass properties in that the quality of stochastic resonance diminishes as the frequency of the subthreshold periodic input is increased while the other parameters are kept constant. This is observed in both the input and output neurons, and in both the floating point model and the integer model. Figure 6.18 shows a decrease in the peaks of the SNR plots (at both the input and the output neurons) for the floating point model as the frequency of the input signal is increased while other network parameters are kept constant. This result reaffirms Plesser's claim in Plesser (1998a) that noise turns an

LIF into a bandpass filter. We have been able to extend this result by showing that this property also holds at network level and that it can be realised in a discrete LIF neuron model. Figure 6.19 shows the same pattern of a decrease in stochastic resonance as the

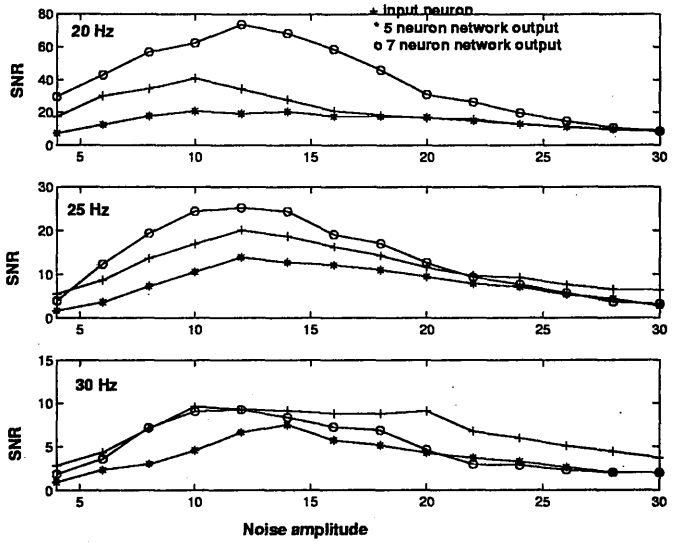

Figure 6.18: Stochastic resonance in the output spike train for an integer-based input and output neurons

frequency of the input signal is increased for the integer model.

It is worth noting that these bandpass properties also depend on the choice of parameters like the membrane time constant τ, and synaptic time constant τ_s and synaptic weights. The value of τ sets a limit to the frequencies of the signals that LIF neuron can follow. The dependence of the power spectrum (which is one of the measures of stochastic resonance) on these parameters is investigated in the next section.

6.6 Effect of network parameters

This section is an attempt to put the results presented in this thesis into a theoretic framework. There are a number of parameters at play in the models and it is useful to examine how they all affect stochastic resonance. The effect of these parameters on

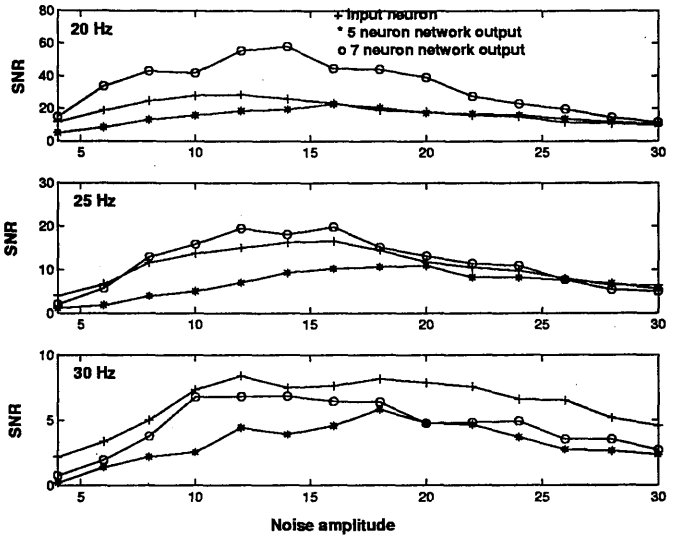

Figure 6.19: Stochastic resonance in the output spike train for an integer-based (12 bits resolution) input and output neurons. Note that the graphs have different scales.

stochastic resonance will be investigated by looking at how the power spectrum of the output neuron spike trains depends on these parameters. This enables one to simulate equations of the power spectrum to narrow the search for optimal parameters to use in the models.

In order to glean information on the relationship between the time constants and the power spectrum of the spike train we will focus our attention on the output neuron. The membrane potential equation for the output neuron is quite accessible because the input to it is not direct noise as it is the case in the input neuron. This means we can numerically solve for $V(t)$ from the LIF equation given in equation 5.1. The solution for $V(t)$ in this case only applies between spikes because of the resetting of $V(t)$ after every spike. We have to resort to looking at the membrane potential because by just looking at a spike train it is difficult to see how its power spectrum is influenced by the change in membrane time constant or synaptic time constant, and synaptic weights. It is our belief that by looking at the power spectrum of $V(t)$ in between spikes, we are able to glean information on the effect of the membrane time constant τ, the synaptic time constant τ_s and synaptic weights

on the power spectrum of the spike train of the output neuron.

We will start with the single neuron situation, i.e. an output neuron receiving spike-based input from one input neuron. The current generated from the spikes is given by:

$$I(t) = a\sum_{i=1}^{n}\alpha(t-t_i) \tag{6.1}$$

where n is the number of spikes contributing to the current and:

$$\alpha(t) = \frac{1}{\tau_s}e^{-\frac{t}{\tau_s}} \tag{6.2}$$

This means the LIF equation becomes:

$$\frac{dV}{dt} = -\frac{V}{\tau} + a\sum_{i=1}^{n}\alpha(t-t_i) \tag{6.3}$$

This results in the following equation:

$$\frac{dV}{dt} + \frac{V}{\tau} = \frac{a}{\tau_s}\sum_{i=1}^{n}e^{-\frac{t-t_i}{\tau_s}} \tag{6.4}$$

which has the following solution:

$$V(t) = \frac{ae^{-\frac{t}{\tau}}}{\tau_s(\frac{1}{\tau}-\frac{1}{\tau_s})}\sum_{i=1}^{n}e^{-\frac{t_i}{\tau_s}}\left[e^{t(\frac{1}{\tau}-\frac{1}{\tau_s})} - e^{t_i(\frac{1}{\tau}-\frac{1}{\tau_s})}\right] \tag{6.5}$$

The Fourier transform $G(\omega)$ of $V(t)$ then becomes:

$$G(\omega) = \frac{a}{2\pi\tau_s(\frac{1}{\tau}-\frac{1}{\tau_s})}\sum_{i=1}^{n}e^{-\frac{t_i}{\tau_s}}\left[\frac{1}{\frac{1}{\tau_s}+j\omega} - \frac{e^{t_i(\frac{1}{\tau}-\frac{1}{\tau_s})}}{\frac{1}{\tau}+j\omega}\right] \tag{6.6}$$

After splitting the complex part into the real and the imaginary part we get:

$$G(\omega) = \frac{a}{2\pi\tau_s(\frac{1}{\tau}-\frac{1}{\tau_s})}\sum_{i=1}^{n}e^{-\frac{t_i}{\tau_s}}\left[\frac{\frac{1}{\tau^2\tau_s}+\frac{\omega^2}{\tau_s}-\frac{e^{t_i(\frac{1}{\tau}-\frac{1}{\tau_s})}}{\tau{\tau_s}^2}-\frac{\omega^2e^{t_i(\frac{1}{\tau}-\frac{1}{\tau_s})}}{\tau}}{(\frac{1}{{\tau_s}^2}+\omega^2)(\frac{1}{\tau^2}+\omega^2)}\right.$$
$$\left.-\frac{j(\frac{\omega}{\tau^2}+\omega^3-\frac{\omega e^{t_i(\frac{1}{\tau}-\frac{1}{\tau_s})}}{{\tau_s}^2}-\omega^3e^{t_i(\frac{1}{\tau}-\frac{1}{\tau_s})})}{(\frac{1}{{\tau_s}^2}+\omega^2)(\frac{1}{\tau^2}+\omega^2)}\right] \tag{6.7}$$

If we concentrate on the complex term in the big square brackets, the power spectrum is given by the following: If we let $e_i = e^{t_i(\frac{1}{\tau}-\frac{1}{\tau_s})}$ then an approximation of the power spectrum is given by the following:

$$|G(\omega)|^2 \approx \frac{a^2}{4\pi^2(\frac{1}{\tau}-\frac{1}{\tau_s})^2}\sum_{i=1}^{n} e^{-\frac{2t_i}{\tau_s}}\left[\frac{\frac{1}{\tau^4{\tau_s}^2}+\frac{\omega^2}{\tau^2{\tau_s}^2}-\frac{e_i}{\tau^3{\tau_s}^3}-\frac{\omega^2 e_i}{\tau^3\tau_s}}{(\frac{1}{{\tau_s}^2}+\omega^2)^2(\frac{1}{\tau^2}+\omega^2)^2}\right.$$
$$+\frac{\frac{\omega^2}{\tau^2{\tau_s}^2}+\frac{\omega^4}{{\tau_s}^2}-\frac{\omega^2 e_i}{\tau{\tau_s}^3}-\frac{\omega^4 e_i}{\tau\tau_s}}{(\frac{1}{{\tau_s}^2}+\omega^2)^2(\frac{1}{\tau^2}+\omega^2)^2}$$
$$-\frac{\frac{e_i}{\tau^3{\tau_s}^3}-\frac{\omega^2 e_i}{\tau{\tau_s}^3}+\frac{{e_i}^2}{\tau^2{\tau_s}^4}+\frac{\omega^2{e_i}^2}{\tau^2{\tau_s}^2}-\frac{\omega^2 e_i}{\tau^3\tau_s}-\frac{\omega^4 e_i}{\tau\tau_s}+\frac{\omega^2{e_i}^2}{\tau^2{\tau_s}^2}+\frac{\omega^4{e_i}^2}{\tau^2}}{(\frac{1}{{\tau_s}^2}+\omega^2)^2(\frac{1}{\tau^2}+\omega^2)^2}$$
$$+\frac{\frac{\omega^2}{\tau^4}+\frac{\omega^3}{\tau^2}-\frac{\omega^2 e_i}{\tau^2{\tau_s}^3}-\frac{\omega^3 e_i}{\tau^2}+\frac{\omega^3}{\tau^2}+\omega^4-\frac{\omega^3 e_i}{{\tau_s}^2}-\omega^4 e_i}{(\frac{1}{{\tau_s}^2}+\omega^2)^2(\frac{1}{\tau^2}+\omega^2)^2}$$
$$\left.-\frac{\frac{\omega^2 e_i}{{\tau_s}^2\tau^2}-\frac{\omega^3 e_i}{{\tau_s}^2}+\frac{\omega^2{e_i}^2}{{\tau_s}^4}-\frac{\omega^3{e_i}^2}{{\tau_s}^2}-\frac{\omega^3 e_i}{\tau^2}-\omega^4 e_i+\frac{\omega^3{e_i}^2}{{\tau_s}^2}+\omega^4{e_i}^2}{(\frac{1}{{\tau_s}^2}+\omega^2)^2(\frac{1}{\tau^2}+\omega^2)^2}\right] \tag{6.8}$$

A complete derivation of equation 6.5 and its Fourier transform 6.6 can be found in appendix A. Equation 6.8 can be simplified as follows:

$$|G(\omega)|^2 = \frac{a^2}{4\pi^2(\frac{1}{\tau}-\frac{1}{\tau_s})^2(\frac{1}{{\tau_s}^2}+\omega^2)^2(\frac{1}{\tau^2}+\omega^2)^2}$$
$$\sum_{i=1}^{n} e^{-\frac{2t_i}{\tau_s}}\left[\omega^4({e_i}^2-2e_i+1)-\frac{2\omega^4 e_i}{\tau_s\tau}-\frac{2\omega^4 e_i}{\tau_s\tau}\right.$$
$$+\frac{\omega^3}{\tau^2}(\omega{e_i}^2-2e_i+2)+\frac{\omega 3}{{\tau_s}^2}(\omega-2e_i)$$
$$+\frac{\omega^2}{{\tau_s}^2\tau^2}({e_i}^2-e_i+2)-\frac{2\omega^2 e_i}{\tau_s\tau^3}$$
$$-\frac{2\omega^2 e_i}{{\tau_s}^3\tau}+\frac{\omega^2}{{\tau_s}^3\tau^3}+\frac{\omega^2 e_i}{{\tau_s}^4}$$
$$\left.+\frac{{e_i}^2}{{\tau_s}^4\tau^2}-\frac{2e_i}{{\tau_s}^3\tau^3}+\frac{1}{{\tau_s}^2\tau^4}\right] \tag{6.9}$$

It is clear that the power spectrum of the membrane potential in between spikes depends on the two time constants (τ and τ_s), the synaptic weight a and the frequency. If we just concentrate on the frequency we see that as the frequency increases, the power spectrum will decrease assuming the time constants are kept constant. This can be seen by inspection from the power spectrum equation because we have a higher power of ω in the denominator than the numerator. This explains the bandpass property of the LIF neuron which we

reported in the previous section. The relationship between the power spectrum and the frequency is of the form $|G(\omega)|^2 = O(\frac{1}{\omega^4})$.

As we increase the weight a, the output power will increase if we keep the other parameters constant. This partly explains the increase in stochastic resonance at the output neuron as we increase the number of neurons because the total weight increases as the number of input neurons increases.

Turning our attention to the time constants, we see that changing the time constants has an effect on the output power. From equation 6.9 both τ and τ_s influence $G(\omega)$ but their influence is complex as both τ and τ_s are in both the numerator and the denominator. Further, equation 6.9 is not valid if $\tau = \tau_s$ although in simulations this can be taken care of.

The above results can be extended to take care of a situation where the output neuron receives input from N input neurons. The membrane potential will be given by equation 6.10 below:

$$V(t) = e^{-\frac{t}{\tau}} \frac{a}{\tau_s(\frac{1}{\tau} - \frac{1}{\tau_s})} \sum_{k=1}^{N} \sum_{i=1}^{n} e^{\frac{t_{ki}}{\tau_s}} \left[e^{t(\frac{1}{\tau} - \frac{1}{\tau_s})} - e^{t_{ki}(\frac{1}{\tau} - \frac{1}{\tau_s})} \right] \tag{6.10}$$

The Fourier transform is given by:

$$G(\omega) = \frac{a}{2\pi\tau_s(\frac{1}{\tau} - \frac{1}{\tau_s})} \sum_{k=1}^{N} \sum_{i=1}^{n} e^{-\frac{t_{ki}}{\tau_s}} \left[\frac{(j\omega + \frac{1}{\tau_s})(\frac{1}{\tau^2} + \omega^2) - e^{-t_{ki}(\frac{1}{\tau} - \frac{1}{\tau_s})}(\frac{1}{\tau} - j\omega)(\omega^2 + \frac{1}{\tau_s{}^2})}{(\omega^2 + \frac{1}{\tau_s{}^2})(\frac{1}{\tau^2} + \omega^2)} \right] \tag{6.11}$$

Let $e_{ki} = e^{-t_{ki}(\frac{1}{\tau} - \frac{1}{\tau_s})}$ and after separating the complex part into real and imaginary parts the Fourier transform is given by the following:

$$G(\omega) = \frac{a}{2\pi\tau_s(\frac{1}{\tau} - \frac{1}{\tau_s})} \sum_{k=1}^{N} \sum_{i=1}^{n} e^{-\frac{t_{ki}}{\tau_s}} \left[\frac{\frac{1}{\tau_s\tau^2} + \frac{e_{ki}}{\tau\tau_s{}^2} + \frac{\omega^2}{\tau_s} + \frac{\omega^2 e_{ki}}{\tau}}{(\omega^2 + \frac{1}{\tau_s{}^2})(\frac{1}{\tau^2} + \omega^2)} - \frac{j(\frac{\omega}{\tau^2} - \frac{\omega e_{ki}}{\tau_s{}^2} + \omega^3 - \omega^3 e_{ki})}{(\omega^2 + \frac{1}{\tau_s{}^2})(\frac{1}{\tau^2} + \omega^2)} \right] \tag{6.12}$$

The power spectrum is then given by:

$$|G(\omega)|^2 \approx \frac{a^2}{4\pi^2(\frac{1}{\tau} - \frac{1}{\tau_s})^2} \sum_{k=1}^{N} \sum_{i=1}^{n} e^{-\frac{2t_{ki}}{\tau_s}} \left[\frac{\frac{1}{\tau_s{}^2\tau^4} + \frac{e_{ki}}{\tau^3\tau_s{}^3} + \frac{\omega^3}{\tau^2\tau_s{}^2} + \frac{\omega^2 e_{ki}}{\tau_s\tau^3}}{(\omega^2 + \frac{1}{\tau_s{}^2})^2(\frac{1}{\tau^2} + \omega^2)^2} + \frac{\frac{e_{ki}}{\tau_3\tau_s{}^2} + \frac{e_{ki}{}^2}{\tau^2\tau_s{}^4} + \frac{\omega e_{ki}}{\tau\tau_s{}^3} + \frac{e_{ki}{}^2\omega^2}{\tau^2\tau_s{}^2}}{(\omega^2 + \frac{1}{\tau_s{}^2})^2(\frac{1}{\tau^2} + \omega^2)^2} \right]$$

$$+\frac{\frac{\omega^2}{{\tau_s}^2\tau^2}+\frac{\omega^2 e_{ki}}{\tau{\tau_s}^3}+\frac{\omega^4}{{\tau_s}^2}+\frac{\omega^4 e_{ki}}{\tau\tau_s}}{(\omega^2+\frac{1}{{\tau_s}^2})^2(\frac{1}{\tau^2}+\omega^2)^2}$$

$$+\frac{\frac{\omega^2 e_{ki}}{\tau_s\tau^3}+\frac{\omega^2 e_{ki}}{{\tau_s}^2\tau^2}+\frac{\omega^4 e_{ki}}{\tau_s\tau}+\frac{\omega^4 {e_{ki}}^2}{\tau^2}}{(\omega^2+\frac{1}{{\tau_s}^2})^2(\frac{1}{\tau^2}+\omega^2)^2}$$

$$+\frac{\frac{\omega^2}{\tau^4}-\frac{\omega^2 e_{ki}}{\tau^2{\tau_s}^2}+\frac{\omega^4}{\tau^2}-\frac{\omega^4 e_{ki}}{\tau}}{(\omega^2+\frac{1}{{\tau_s}^2})^2(\frac{1}{\tau^2}+\omega^2)^2}$$

$$-\frac{\frac{\omega^2 e_{ki}}{\tau^2{\tau_s}^2}+\frac{\omega^2 {e_{ki}}^2}{{\tau_s}^4}-\frac{\omega^4 e_{ki}}{{\tau_s}^2}+\frac{\omega^4 {e_{ki}}^2}{{\tau_s}^2}}{(\omega^2+\frac{1}{{\tau_s}^2})^2(\frac{1}{\tau^2}+\omega^2)^2}$$

$$+\frac{\frac{\omega^4}{\tau^2}-\frac{\omega^4 e_{ki}}{{\tau_s}^2}+\omega^6-\omega^6 e_{ki}-\frac{\omega^4 e_{ki}}{\tau^2}+\frac{\omega^4 e_{ki}}{{\tau_s}^2}-\omega^6 e_{ki}+\omega^6 {e_{ki}}^2}{(\omega^2+\frac{1}{{\tau_s}^2})^2(\frac{1}{\tau^2}+\omega^2)^2}\Bigg] \tag{6.13}$$

A complete derivation of the membrane potential equation 6.10 and its power spectrum 6.14 for the multiple input neurons case is in appendix B. Equation 6.14 can be simplified as follows:

$$|G(\omega)|^2=\frac{a^2}{4\pi^2(\frac{1}{\tau}-\frac{1}{\tau_s})^2(\frac{1}{{\tau_s}^2}+\omega^2)^2(\frac{1}{\tau^2}+\omega^2)^2}$$

$$\sum_{k=1}^{N}\sum_{i=1}^{n}e^{-\frac{2t_{ki}}{\tau_s}}\Bigg[\frac{1}{{\tau_s}^2\tau^4}+\frac{e_{ki}}{{\tau_s}^3\tau^3}+\frac{\omega^2}{{\tau_s}^2\tau^2}(\omega+e_{ki}-{e_{ki}}^2+1)$$

$$+\frac{2\omega^2 e_{ki}}{\tau_s\tau^3}+\frac{e_{ki}}{{\tau_s}^2\tau^3}+\frac{{e_{ki}}^2}{{\tau_s}^4\tau^2}+$$

$$\frac{\omega e_{ki}}{{\tau_s}^3\tau}(\omega+1)$$

$$+\frac{\omega^4}{{\tau_s}^2}({e_{ki}}^2-2e_{ki}+1)+\frac{2\omega^4 e_{ki}}{\tau_s\tau}+\frac{\omega^4}{\tau^2}(2-{e_{ki}}^2)$$

$$+\frac{\omega^2}{\tau^4}+\frac{\omega^2 {e_{ki}}^2}{{\tau_s}^4}$$

$$+\omega^6({e_{ki}}^2-2e_{ki}+1)\Bigg] \tag{6.14}$$

Again we can see that the effect of the parameters τ, τ_s and the synaptic weight a is the same as observed for the single neuron case. The effect of increasing the number of input neurons while keeping the weights constant is that the power increases.

In this section we attempted to investigate the effect of the time constants, frequency and synaptic weights on stochastic resonance. The equations derived in this section do not give the full picture because of the resetting of the membrane potential after every

spike. These equations serve as an initial attempt in trying to understand the effect of network parameters on stochastic resonance. The equations for the power spectrum of the membrane potential derived in this section show that the Fourier components depend on the time constants, the frequency of the input signal and the synaptic weights.

6.7 Discussion

The stochastic resonance effect occurs because noise introduces a degree of independence between the information carried by individual neurons. Although, in the absence of noise, the timing of the spikes is at its most precise (Stocks and Mannella (2001)), thus maximising information transmitted by individual neurons, similar neurons produce identical spike trains. Consequently, no additional information is gained by making simultaneous measurements of similar neurons. In fact, the global information is simply equal to that carried by a single neuron. However, the introduction of neuronal noise randomises the times at which each neuron "samples" the signal and, although this lowers the amount of information carried by individual elements, there is a net gain in information transmitted. This explains why the SNR at the output neuron overtakes that at the input neuron.

These results show that in a population of neurons noise can have a significantly great optimising influence of generic benefit to neuronal signal encoding. Given that perception is based on accumulated information obtained from many nerve fibres, these results suggest that the high levels of noise observed in biological sensory systems may have evolved and are an essential component of an optimal coding strategy. Finally we note that these results have implications on the design of coding strategies for use in implantable cochlear and visual prostheses (Stocks and Mannella, 2001). Such prostheses work by the artificial electrical stimulation of populations of nerve fibres - similar to the model considered here. The results presented here suggest that a combination of noise and subthreshold stimuli could lead to improved information transmission. This strategy is in-keeping with recent results ((Morse and Evans, 1996)) which demonstrate that the coding of *formant* information in cochlear implants can be improved by the use of stimuli corrupted by noise.

6.8 Summary

The results presented in this chapter show that stochastic resonance is realisable in discrete systems as shown by stochastic resonance in the integer model for the network topology in figure 5.1. The quality of floating point stochastic resonance is always better than that of the integer model for the noise ranges that are considered useful. Stochastic resonance in the integer model has the same properties as that for the floating point model. A multiple comparison t-test of the stochastic resonance between the two models revealed that the difference between the floating point model stochastic resonance and that of the best integer model stochastic resonance (12 and 16 bits resolutions) is statistically significant. The same statistical analysis revealed that SNR values for 12 and 16 bits resolutions are not statistically different from each other for input neurons but they are both significantly different from the other two resolutions (10 and 11 bits). It should be noted that these results just give us numbers. What these results do not tell us is whether the differences between the floating point model and the integer model and the differences between the various integer resolutions have any perceptual implications. The results do not tell us whether for instance, sound processed on the floating point platform would be perceived differently from that processed on the integer platform. These questions can only be answered by clinical trials. Unfortunately this thesis does not go that far but it is not difficult to see that this would be one of the many ways of furthering this research as we will point out in section 7.3 for further work in the next chapter.

Chapter 7

Conclusion

To conclude this thesis, we will summarise the main results and then give some ideas for further work. In the introduction, two main themes were developed. The aim was to investigate whether stochastic resonance in a small network of LIF neurons relies on the continuous nature of the underlying system or whether a linearly discretised system is able to display stochastic resonance. It was also of interest to investigate how discrete-based stochastic resonance is affected by changes in the precision of the values in the simulations. In order to investigate these two aims, two LIF models were developed: a floating point and an integer model. The floating point model is based on existing models and the integer model was derived from the floating point one by way of an approximation. Besides the models, we needed good quantitative measures of stochastic resonance to base our comparison on. We settled for the power spectrum and the signal-to-noise ratio.

7.1 Summary of results

The main conclusion of this thesis is that stochastic resonance can be implemented in digital hardware of the FPGA type using an integer-based model, thus permitting cheap real-time systems which rely on stochastic resonance. This shows that stochastic resonance does not rely on the continuous nature of the underlying system. The stochastic resonance

which has been observed for the discrete system is in many ways similar to that of the continuous based system. This is useful because such systems can be used as front-ends for auditory implants like cochlear implants which operate in real-time. In order to arrive at this result we had to come up with relevant models which we implemented in Java on a PC and in Handel-C on FPGA. This modelling work and the analysis of the results resulted in some interesting observations which we outline below.

7.1.1 Power spectrum of spike trains

Our contribution to the computation of the power spectrum of spike trains was a review of the various ways of sampling action potentials. We noted that the main difficulty in sampling action potentials lies in the way they are represented. Because of the short duration of action potentials relative to their frequency of occurrence in many situations, it has become commonplace to treat them as "events" in time and assume that the only parameters of importance in neural coding depend in some way on the occurrence time of the action potential. This representation presents problems to the implementation of spectral analysis for neuronal spike trains. On the one hand a continuous signal is desirable for implementation of the FFT algorithm by regular sampling, on the other hand treatment as a point process is more acceptable and conventional but poses computational problems. We settled for a method in which the spike train (vector of zeros and ones) is multiplied by a constant and the FFT algorithm is applied to the result.

7.1.2 Signal-to-noise ratio

Our contribution to the computation of the SNR of a spike train given its power spectral density was in the analysis of the effects of making different assumptions about the distribution of the noise power relative to the frequencies of interest in the power spectral density. It is not clear in the literature how the SNR of a spike train should be computed. The problem lies in separating noise power from the power at the frequency of interest given that some of the signal power leaks into neighbouring frequencies and that the spike

train encodes both the noise and the signal. In this thesis we evaluated three different ways of computing the SNR of a spike train from its power spectral density. The three methods differ in the assumptions they make about the distribution of noise power relative to the signal of interest. Our conclusion here is that it is better to use a method which makes uniformity assumptions about the distribution of noise power in the vicinity of the frequency of interest. It is this noise power near the frequency of interest that should be used in computing the SNR.

7.1.3 Modelling results

In the neural modelling work, we have shown that using a simple numerical method like Euler, we were able to obtain neuronal stochastic resonance (continuous and discrete based) which compares with that which has been reported in the literature. The main emphasis of the modelling work was suitability for hardware implementation. A comparison of the analytical model decay and the Euler model decay showed that for the time step that we chose, an Euler approximation is adequate to show stochastic resonance in a network of LIF neurons.

We derived an integer model of an LIF neuron which is suitable for hardware implementation from a floating point based model. This is significant because it means we can build large networks of such neurons and simulate neuronal dynamics with stochastic resonance on digital hardware in real time. The main difference between the integer model and the floating point model is in the evolution of the membrane potential. The integer model focused on the suitability for reduced resolution digital hardware implementation on FPGA and precision control. Given the distance between the two models, we decided to narrow the gap by developing a model which is between floating point and integer by forcing the floating point model to evolve in discrete steps.

We successfully implemented the integer model and realised stochastic resonance, which until now has always been associated with floating point based models. The simplification was a result of the realisation that an LIF neuron with Gaussian white noise can be

solved using an Euler approximation provided the noise is bandlimited and the time step is small enough to ensure stability. This means instead of the usual diffusion approximation approach for solving a stochastic LIF neuron model differential equation, which would take up a lot of chip area because of it being floating point in nature, we have a much reduced model which still displays stochastic resonance. The integer based stochastic resonance is less than that realised from the floating point model but only for the low noise values at which stochastic resonance is strongest.

7.1.4 Reduced resolution stochastic resonance

The integer model that we developed also allowed us to study the effect of changing the precision of the values in the simulations on stochastic resonance. Our results show that discrete based stochastic resonance improves with an increase in the resolution of the values in the simulation. Specifically we have shown that in a single neuron, stochastic resonance is poor below 10 bits and saturates at about 12 bits. Any further increase in integer length does not yield an improvement in stochastic resonance for the input neurons. The effect of resolution is quite strong at low noise values but diminishes at high noise values. The effect of quantisation was more manifest at low noise values due to the quantisation of small numbers. This may explain why the maximum SNR is obtained at higher resolution (16 bits) in network models as peak SNR occurs at lower noise levels (see below).

7.1.5 Improvement in stochastic resonance

From a network point of view, we have shown that stochastic resonance improves at the output neuron as the number of input neurons increased. This improvement in stochastic resonance at the output neuron when parallel neurons converge on one neuron justifies the existence of such networks in the central nervous system confirming the findings of Collins et al. (1995a) realised using a floating point based rate model. We have shown that this result also holds for integer-based stochastic resonance. In chapter 6 we presented results (figure 6.15) showing that as the number of input neurons is increased the peak

for the SNR curve for the output neuron increases, becomes narrower and moves to lower noise amplitudes. The observation that the SNR peaks move to lower noise amplitudes is probably due to the fact that the weights are fixed which means as the number of input neurons is increased we need less noise to excite the output neuron. The sharpness and narrowness of the peak for large numbers of input neurons could be addressed by normalising the weights by the number of input neurons. From a network design point of view it is better to have broad peaks because this makes the network tolerant to a broader range of noise amplitude. However, this is at the expense of the maximum SNR that can be achieved. We also observed that for a given noise amplitude increasing the size of the network does not necessarily lead to higher SNR values, though larger networks will have higher SNR if the noise is chosen optimally.

Our results also revealed an unexpected interaction between the resolution and the number of input neurons. For higher resolutions like 16 bits, the integer model output neuron outperformed the floating point-based neuron for high noise values for a network with 7 neurons. This means the integer model is more robust to noise at higher resolutions compared to the floating point model. This shows that there is something to gain after all from using higher resolutions in the integer model. At this point we do not have an explanation for this interaction. However, we know that increasing the resolution reduces quantisation effects and increasing the number of input neurons reduces the effect of noise because we are summing the output of many units. We suspect that a combination of these two factors may be used to explain the interaction between network size and resolution at the output neuron.

7.1.6 Significance of the network topology

The convergence of many LIF neurons onto one neuron (as signified by our network topology) or a few neurons has some neurophysiological correlates in wetware. For instance, in the early visual system there is a lot of convergence from receptors onto the retinal ganglion cells (Sjostrand et al., 1999). There are more sensors than there are retinal ganglion cells.

This may be for the purposes of increasing sensitivity in sensory processing. Our results show that one way of explaining this convergence is for increased sensitivity in sensory systems. If we turn our attention to the early auditory system we see that there are more auditory nerve fibres than there are hair cells which implies a divergence (Pickles, 1988). However, there is a lot of convergence of many auditory nerve fibres onto onset cells in the cochlear nucleus (Romand and Avan, 1997). It is said in the literature that the amplitude modulated signals being carried by the auditory nerve fibres are amplified in the cochlear nucleus through this convergence (Palmer and Winter, 1992, 1993). Our conclusion is that this convergence may be for the purposes of increased sensitivity in the CNS.

7.2 Conclusion

The results summarised above have shown that stochastic resonance is possible in a discrete system and for the small LIF network considered stochastic resonance is very poor below 10 bits but saturates at 12 bits for the input neurons and at higher resolutions for the output neuron. Since resolution is inversely proportional to chip area, a resolution of 12 bits will result in a much smaller hardware design compared to what would be realised using 32 bits floating point representation and computation. Therefore a noisy LIF network of neurons can be implemented on digital hardware at 12 bits resolution and be tested for applications like sound localisation. What is particularly interesting about these results is that they have been achieved using a simple neuronal model and a simple integration technique which all amounts to a minimal use of resources. Despite the simplicity of the model used, the results obtained compare favourably with those obtained using more realistic models like the Hodgkin-Huxley model (Stemmler, 1996). Results from the work of Stemmler show that more realistic models of spiking neurons like the Hodgkin-Huxley model also exhibit the stochastic resonance phenomenon which means this effect is by no means peculiar to the leaky integrate-and-fire model. The result for the original Hodgkin-Huxley model of the squid giant axon shows the familiar stochastic resonance shape using the Fisher information (Cover and Thomas, 1991) metric. Numerous differences exist between the LIF model and

the Hodgkin-Huxley model, from the differences in the firing rate as a function of the input current to the fact that the Hodgkin-Huxley model does not have a fixed voltage threshold for spiking. Nonetheless, both models display this form of stochastic resonance. This means that we are justified in using an LIF neuron model to study stochastic resonance.

The use of SNR as a performance metric does not compromise the results because as pointed in the work of Stemmler (1996), in the limit of low firing rates SNR and entropy based results for an LIF neuron result in similar mathematical expressions. It must be pointed out though that entropy-based measures provide a more fundamental metric and a general definition of stochastic resonance in neuronal models is in terms of the Fisher information. SNR is more suitable for periodic stochastic resonance because for periodic input we are mainly interested in the output power at the signal frequency.

We also pointed out that the use of a periodic input signal does not compromise the results because the relevant information measures for quantifying stochastic resonance have the same asymptotic scaling properties as a function of the noise amplitude irregardless of the nature of the input signal (Stemmler, 1996). This means that results reported here could be extended to aperiodic stochastic resonance.

As we pointed out in the previous chapter, we were unable to have a network with more than 7 neurons on the FPGA due to problems with the synthesis tool. A plot of the network size versus number of gates showed that the number of gates scales linearly with network size. Using linear interpolation it worked out that using our model, an FPGA with 1 million gates would result in a network of about 40 neurons allowing for the fact that placing and routing is never 100 percent of the FPGA size. A network of this size could be used to simulate small groups of neurons especially in insects where small numbers of similar neurons can be identified (see the work of Chapman (2001) on modelling the escape response of crickets).

In terms of application, the results reported here are best used for prototyping systems which could later be implemented on ASIC hardware. This is mainly due to the limitations of the FPGA technology. The limitations are mainly in terms of little on-chip memory which means most of the memory is off-chip which results in an increase of the volume of

I/O traffic. The more I/O traffic we have, the higher the power consumption and for this reason FPGAs have remained outside the ever growing market of hand held devices.

7.3 Further work

One of the ways of extending this work would be to use a standard hardware description language like VHDL to program the FPGA so as to realise bigger networks. VHDL is the current standard and as such there are many synthesis tools which means one can easily avoid the situation we found ourselves in when we realised that the only synthesis tool for Handel-C is not good enough for the network sizes that we initially aimed for. This will allow the simulation of more complex networks.

The next suggestion for further work follows on from the previous one. Once one is able to build a bigger network one should go on to test the networks with realistic signals in real time.

In terms of application, this work could be used in front-end systems for implantable visual and cochlear prostheses or for detecting regularities in noisy inputs in highly sensitive input devices.

The results that we presented in this thesis were based on synthetic signals. Another way of advancing this work is to process realistic signals, such as speech, on this FPGA platform and then do some trials on human subjects to see if the differences in representation and resolution that we observed have any perceptual significance.

Bibliography

Abbott, L. (1991). Firing-rate models for neural populations. In Benhar, O., Bosio, C., Giudice, P. D., and Tabet, E., editors, *Neural networks*, From biology to high-energy physics, pages 179–196.

Abbott, L. and Kepler, T. (1990). *Model neurons: from Hodgkin-Huxley to Hopfield.* Statistical mechanics of neural networks. Springer, Berlin.

Abbott, L., Varela, J., Sen, K., and Nelson, S. (1997). Synaptic depression and cortical gain control. *Science*, 275:220–223.

Amit, D. (1989). *Modelling brain functions — The world of attractor networks.* Cambridge University Press.

Anderson, J., Lampl, I., Gillespie, D., and Ferster, D. (2000). The contribution of noise to contrast invariance of orientation tuning in cat visual cortex. *Science*, 290:1968–1972.

Aubury, M., Page, I., Randall, G., Saul, J., and Watts, R. (1996). *Handel-C language reference guide.* Oxford University, Oxford University computing laboratory.

Bade, S. and Hutchings, B. (1994). FPGA-based stochastic neural networks - implementation. In *IEEE workshop on FPGAs for computing machines workshop*, pages 189–198, Napa, CA.

Barbi, M., Chillemi, S., and Garbo, A. (2000). The leaky integrate-and-fire neuron: a useful tool to investigate SR. *Chaos, solutions and fractals*, pages 1273–1275.

Benzi, R., Parisi, G., Sutera, A., and Vulpiani, A. (1982). Stochastic resonance in climatic changes. *Tellus*, 34:10–16.

Benzi, R., Sutera, A., and Vulpiiani, A. (1981). The mechanism of stochastic resonance. *J. Phys. A*, 14:L453–L457.

Bezrukov, S. and Vodyanoy, I. (1997a). Signal transduction across the alamethicin channels in the presence of noise. *Biophys. J.*, 73:L453–L457.

Bezrukov, S. and Vodyanoy, I. (1997b). Stochastic resonance in non-dynamical systems without response threshold. *Nature*, 385:319–321.

Bostock, G. (1996). *FPGAs and Programmable LSI.* Butterworth Heinemann.

Breslin, C. and Smith, L. (1999). Silicon cellular morphology. *Intern. Journal of Neural Systems*, 9(5):491–495.

Brown, D., Feng, J., and Feerick, S. (1999). Variability of firing of Hodgkin-Huxley and FitzHugh-Nagumo neurons with stochastic synaptic input. *Phys. Rev. Lett.*, 82(23):4731–4734.

Brown, S. and Rose, J. (1996). FPGA and CPLD architectures - A tutorial. *IEEE design and test of computers*, 13(3):42–57.

Bryant, H. and Segundo, J. (1976). Spike initiation by transmembrane current: a white noise analysis. *The Journal of Physiology*, 260(2):279–314.

Bugmann, G. (1991). Summation and multiplication: two distinct operation domains of leaky integrate-and-fire neurons. *Network*, 2:489–509.

Bugmann, G., Christodoulou, C., and Taylor, J. (1997). Role of temporal integration and fluctuation detection in the highly irregular firing of a leaky integrator neuron model with partial reset. *Neural Computation*, 9:985–1000.

Bulsara, A., Elston, T., Doering, C., Lowen, S., and Lindenberg, K. (1996). Cooperative behaviour in periodically driven noisy integrate-and-fire models of neuronal dynamics. *Phys. Rev. E*, 53:3958–3969.

Burkitt, A. and Clark, G. (1999). Analysis of integrate-and-fire neurons: synchronisation of synaptic input and spike output. *Neural Computation*, 11(4):871–901.

Burkitt, A. and Clark, G. (2000). Analysis of synchronisation of the neural response to noisy periodic synaptic input. *Neurocomputing*, 32-33:67–75.

Capurro, A., Pakdaman, K., Nomura, T., and Sato, S. (1998). Aperiodic stochastic resonance with correlated noise. *Phys. Rev. E*, 58:4820–4827.

Celoxica (2000). *Handel-C v2.1 Product information sheet.* Embedded solutions Ltd., *http : //www.embeddedsol.com/tech − info − 2.htm.*

Chapeau-Blondeau, F., Godivier, X., and Chambet, N. (1996). Stochastic resonance in a neuronal model that transmits spike trains. *Physical Review E*, 53:1273–1275.

Chapman, T. (2001). *Morphological and neural modelling of the Orthopteran escape response.* PhD thesis, University of Stirling.

Chialvo, D., Longtin, A., and Muller-Gerking, J. (1997). Stochastic resonance in models of neuronal ensembles. *Phys. Rev. E*, 55:1798–1808.

Christodoulou, C. and Bugmann, G. (2001). Coefficient of variation (CV) vs mean interspike interval (ISI) curves: what do they tell us about the brain? *Neurocomputing*, 38-40:1141–1149.

Cohen, B. (1999). *VHDL coding styles and methodologies.* Kluwer Academic Publishers.

Collins, J., Chow, C., Capela, A., and Imhoff, T. T. (1996a). Aperiodic stochastic resonance. *Phys. Rev. E*, 54:5575–5584.

Collins, J., Chow, C., and Imhoff, T. (1995a). Aperiodic stochastic resonance in excitable systems. *Phys. Rev. E*, 52:3321–3324.

Collins, J., Chow, C., and Imhoff, T. (1995b). Stochastic resonance without tuning. *Nature*, 376:236–237.

Collins, J., Imhoff, T., and Grigg, P. (1996b). Noise-enhanced information transmission in Rat SA1 cutaneous mechanoreceptors via aperiodic stochastic resonance. *Journal of Neurophysiology*, 76(1):642–645.

Cover, T. and Thomas, J. (1991). *Elements of information theory.* New York: Wiley-Interscience.

Cox, D. and Lewis, P. (1966). *The statistical analysis of series of events.* Metheun and Company, London.

Cox, D. and Miller, H. (1965). *The theory of stochastic processes.* Methuen and Company, London.

David, R. (1989). *The VHDL handbook.* Kluwer Academic Publishers.

Dayan, P. and Abbott, L. (2001). *Theoretical neuroscience: computational and mathematical modeling of neural systems.* The MIT Press.

Douglas, J., Wilkens, L., Pantazelou, E., and Moss, F. (1993). Noise enhancement of information transfer in crayfish mechanoreceptor by stochastic resonance. *Nature*, 365:337–340.

Dykman, M., Manella, R., McClintock, P., and Stocks, N. (1990a). Stochastic resonance in bistable systems—comment. *Phys. Rev. Lett.*, 65(20):2606.

Dykman, M., Manella, R., McClintock, P., and Stocks, N. (1990b). Stochastic resonance in the linear and nonlinear response of a bistable system to a periodic field. *JETP Lett.*, 52(3):141–144.

Fauve, S. and Heslot, F. (1983). Stochastic resonance in a bistable system. *Phys. Lett.*, 97A:5–7.

Feller, W. (1971). *An introduction to probability theory and its applications*, volume 2. Wiley, second edition.

Feng, J. (1997). Behaviours of spike output jitter in the integrate-and-fire model. *Phys. Rev. Lett.*, 79:4505–4508.

Flanagan, D. (2002). *Java in a nutshell: a desktop quick reference.* O'Reilly, 4th edition.

Forrester, M. (1993). *The precision required for digital neural hardware.* PhD thesis, University of Wales, Aberystwyth, Department of Computer Science.

French, A. and Holden, A. (1971). Alias-free sampling of neuronal spike trains. *Kybernetik*, 8:165–171.

French, A., Holden, A., and Stein, R. (1972). The estimation of the frequency response function of a mechanoreceptor. *Kybernetik*, 11:15–23.

Frisina, R., Walton, J., and Karcich, K. (1994). Dorsal cochlear nucleus single neurons can enhance processing capabilities in background noise. *Exp. Brain Res.*, 102:160–164.

Gammaitoni, L. (1995). Stochastic resonance and the dithering effect in threshold physical systems. *Physical Review E*, 52(5):4691–4698.

Gammaitoni, L., Hanggi, P., Jung, P., and Marchesoni, P. (1998). Stochastic resonance. *Review Modern Physics*, 70:223–287.

Gerstner, W. (2000). Population dynamics of spiking neurons: Fast transients, asynchronous states, and locking. *Neural Computation*, 12:43–89.

Gerstner, W., Kempter, R., van Hemmen, J., and Wagner, H. (1996). A neuronal learning rule for sub-millisecond temporal coding. *Nature*, 383:76–81.

Gerstner, W., Kempter, R., van Hemmen, J., and Wagner, H. (1997). A developmental rule for coincidence tuning in the barn owl auditory system. In *Computational Neuroscience: Trends in research*, pages 665–669. Plenum Press, New York.

Gerstner, W. and Kistler, W. (2002). *Spiking neuron models.* Cambridge University Press.

Gerstner, W. and van Hemmen, J. (1992). Associative memory in a network of 'spiking' neurons. *Network*, 13:139–164.

Gingl, Z., Kiss, L., and Moss, F. (1995a). Non-dynamical stochastic resonance: theory and expriments with white noise and arbitrarily coloured noise. *Europhys. Lett.*, 29:191–196.

Gingl, Z., Kiss, L., and Moss, F. (1995b). Non-dynamical stochastic resonance: theory and expriments with white and various coloured noises. *Nuovo Cimento D*, 17:795–802.

Glover, M. (1999). *An Analogue VLSI Implementation of an Integrate-and-Fire Neural Network for Real-Time Auditory Data Processing.* PhD thesis, University of Ediburgh.

Glover, M., Hamilton, A., and Smith, L. (1999). Analogue VLSI leaky integrate-and-fire neurons and their use in a sound analysis system. *Analog Integrated Circuits and Signal Processing, Special Issue: Microelectronics for Bio-inspired Systems (Selected Papers from MicroNeuro'99 Conference)*, 30(2):91–100.

Gluckman, B., Netoff, T., Neel, E., Spano, W. D., and Schiff, S. (1996). Stochastic resonance in a neuronal network from a mammalian brain. *Physical Review Letters*, 77(19):4098–4101.

Godiver, X. and Chapeau-Blondeau, F. (1996). Noise-enhanced transmission of spike trains in the neuron. *Europhysics Letters*, 35:473–477.

Godivier, X. and Chapeau-Blondeau, F. (1998). Stochastic resonance in the information capacity of a nonlinear dynamic system. *Inter. J. of Bifu. and Chaos*, 8:581–589.

Grassmann, C., Schoenauer, T., and Wolff, C. (2002). PCNN neurocomputers - Event driven and parallel architectures. In *ESANN*, pages 331–336, Bruges, Belgium.

Gray, C. and Singer, W. (1989). Stimulus-specific neuronal oscillations in orientation columns of cat visual cortex. *Proc. of Natl. Acad. Scie. USA*, 86:1698–1702.

Hansel, D., Mato, G., Meunier, C., and Neltner, L. (1998). On numerical simulations of integrate-and-fire neural networks. *Neural Computations*, 10:467–483.

Hecht-Nielsen, R. (1989). *Neurocomputing.* Addison-Wesley Publishing Company.

Heneghan, C., Chow, C., Collins, J., Imhoff, T., Lowen, S.B., and Teich, M. (1996). Information measure quantifying aperiodic stochastic resonance. *Phys. Rev. E.*, 54:R2228–R2231.

Hennig, M., Kerscher, N., Funke, K., and Worgotter, F. (2001). Stochastic resonance in visual cortical neurons: does the eye-tremor actually improve visual acuity? *Neurocomputing*, 44-46C:115–120.

Hertz, J., Krogh, A., and Palmer, R. (1991). *Introduction to the theory of neural computation.* Addison-Wesley.

Hess, S. and Albano, A. (1998). Minimum requirements for stochastic resonance in threshold systems. *Int. J. Bifurcation and chaos*, 8:395–400.

Hohfeld, M. and Fahlman, S. (1992). Probabilistic rounding in neural network learning with limited precision. *Neurocomputing*, 4:291–299.

Hohn, N. (2000). Stochastic resonance in a neuron model with application to the auditory pathway. Master's thesis, University of Melbourne, Department of Otolaryngology.

Holmstrom, S. and Sere, K. (1998). Reconfigurable hardware - a case study in codesign. Technical report 175, TUCS, Turku Centre for Computer Science, Finland.

Holmstrum, M. (2000). *FPGA designs.* PhD thesis, Turkus University.

Hutcheon, B., Miura, R., and Puil, E. (1996). Models of subthreshold membrane resonance in neocortical neurons. *J. Neur. Phys.*, 76:698–714.

Ifeachor, E. and Jervis, B. (2002). *Digital signal processing: A practical approach.* Prentice Hall, 2nd edition.

Jahnke, A., Dschonauer, T., Roth, U., Mohraz, K., and Klar, H. (1997). Simulation of spiking neural networks on different hardware platforms. In Gerstner, W., Germond, A., Hasler, M., and Nicoud, J., editors, *Artificial neural networks - ICANN-97*, Lecture notes in computer science series 1327, pages 1187–1192. Springer-Verlag.

Joiner, R. (1994). *Minitab Handbook.* Duxbury Press.

Jung, P. (1994). Threshold devices: Fractal noise and neural talk. *Phys Rev E*, 50:2513–2522.

Kanamaru, T., Horita, T., and Okabe, Y. (1999). Stochastic resonance for the superimposed periodic pulse train. *Phys. Lett. A*, 255:23–36.

Kandel, E., Schwartz, J., and Jessel, T. (2000). *Principles of neural science.* McGraw-Hill, fourth edition.

Kaufman, I., Luchinsky, D., McClintock, P., Soskin, S., and Stein, N. (1996). High-frequency stochastic resonance in SQUID's. *Phys. Lett. A*, 220:219–223.

Kempter, R., Gerstner, W., van Hemmen, J., and Wagner, H. (1998). Extracting oscillations: neuronal coincidence detection with noisy periodic spike input. *Neural Computation*, 10:1987–2017.

Kistler, W., Gerstner, W., and van Hemmen, J. (1997). Reduction of the Hodgkin-Huxley equations to a single variable threshold model. *Neural Computation*, 9:1025–1045.

Koch, C. (1999). *Biophysics of Computation.* Oxford University Press, NY.

Koch, C. and Segev, I., editors (1999). *Methods in Neural modeling: from ions to networks.* The MIT Press, second edition.

Konig, P., Engel, A., and Singer, W. (1996). Integrator or coincidence detector? The role of the cortical neuron revisited. *Trends in Neuroscience*, pages 130–137.

Lansky, P. and Lanska, V. (1997). Noise in integrate-and-fire models of neuronal dynamics. In Gerstner, W., Germond, A., Hasler, M., and Nicoud, J.-D., editors, *Lecture Notes in Computer Science*, 1327, pages 49–54, Lausanne, Switzerland. Springer.

Lawrence, A. (1997). Macro support for Xilinx achitecture.

Lee, C., Rohrer, W., and Sparks, D. (1988). Population coding of saccadic eye movements by neurons in the superior colliculus. *Nature*, 332:357–360.

Lee, S. and Kim, S. (1999). Parameter dependence of stochastic resonance in the stochastic Hodgkin-Huxley neuron. *Phys. Rev. E*, 60:826–830.

Lee, S., Neiman, A., and Kim, S. (1998). Coherence resonance in a Hodgkin-Huxley neuron. *Phys. Rev. E*, 57:3292–3297.

Levin, J. and Miller, J. (1996). Broadband neural encoding in the cricket cercal sensory system enhanced by stochastic resonance. *Nature*, 380:165–168.

Longtin, A. (1993). Stochastic resonance in neuron models. *J. Stat. Phys.*, 70:309–327.

Longtin, A., Bulsara, A., and Moss, F. (1991). Time-interval sequences in bistable systems and noise-induced transmission of information by sensory neurons. *Phys. Rev Lett.*, 67:656–659.

Luchinsky, D., Mannella, R., McClintock, P. V. E., and Stocks, N. (1999a). Stochastic resonance in electrical circuits I. conventional stochastic resonance. *IEEE Trans. on Circuits and Systems*, 46:1205–1214.

Luchinsky, D., R. Mannella, P. M., and Stocks, N. (1999b). Stochastic resonance in electrical circuits II. nonconventional stochastic resonance. *IEEE Trans. Circuits and Systems*, 46:1215–1224.

Lynn, P. (1989). *An introduction to the analysis and processing of signals.* Macmillan Education, third edition.

Maass, W. (1996). Lower bounds for the computational power of spiking neurons. *Neural Computation*, 8(1):1–40.

Maass, W. (1997). Network of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671.

Maass, W. and Bishop, C. M., editors (1999). *Pulsed neural networks.* MIT.

Maass, W. and Natschlager, T. (1999). Fast analog computation in networks of spiking neurons using unreliable synapses. In *ESANN99 - Proceedings of the European Symposium on Artificial Neural Networks*, pages 417–422.

Mahowald, M. and Douglas, R. (1991). A silicon neuron. *Nature*, 354:515–518.

Mainen, Z. and Sejnowski, T. (1995). Reliability of spike timing in neocortical neurons. *Science*, 268:1503–1506.

Mannela, R. and Palleschi, V. (1989). Fast and precise algorithm for computer simulation of stochastic differential equations. *Physical Review A*, 40:3381–3386.

Mar, D., Chow, C., Gerstner, W., Adams, R., and Collins, J. (1996). Noise-shaping in a population of coupled model neurons. In *Proceedings of the National Academy of Sciences*, pages 10450–10455.

Marino, F., Guidic, M., Barland, S., and Balle, S. (2002). Stochastic resonance in an excitable optical system. *Phys. Rev. Lett.*, 88(4):040601–040604.

Marsalek, P., Koch, C., and Maunsell, J. (1997). On the relationship between synaptic input and spike output jitter in individual neurons. *Proceedings of National Academic Society, Neurobiology*, 94:735–740.

McNamara, B., Wiesenfeld, K., and Roy, R. (1988). Observation of stochastic resonance in a ring laser. *Phys. Rev. Lett.*, 60:2626–2629.

Moerland, P. and Fiesler, E. (1997). Neural network adaptations to hardware implementations. In Fiesler, E. and R.Beale, editors, *Handbook of neural computation.* Institute of Physics Publishing and Oxford University Publishing, New York.

Mori, T. and Kai, S. (2002). Noise-induced entrainment and stochastic resonance in human brain waves. *Phys. Rev. Lett.*, 88(218101):1–4.

Morse, R. and Evans, E. (1996). Enhancement of vowel coding for cochlear implants by addition of noise. *Nature Med*, 2:2626–2629.

Moss, F., Douglas, J., Wilkens, L., Pierson, D., and Pantozelou, E. (1993). Stochastic resonance in an electrical Fitzhugh-Nagumo model. *Ann. N.Y. Acad. Sci.*, 706:26–41.

Moss, F., Pierson, D., and Gorman, D. (1994). Stochastic resonance: a tutorial and update. *Int. J. Bifurcation and chaos*, 4:1383–1397.

Moss, F. and Wiesenfeld, K. (1995). The benefits of background noise. *Scientific American*, 273:50–54.

Mtetwa, N., Smith, L., and Hussain, A. (2002a). Stochastic resonance and finite resolution in a network of leaky integrate-and-fire neurons. In *Artificial neural networks - ICANN 2002*, pages 117–122, Madrid, Spain. Springer.

Mtetwa, N., Smith, L., and Hussain, A. (2002b). Stochastic resonance and finite resolutions in a leaky integrate-and-fire neuron. In *ESANN2002: Proceedings of the European Symposium on Artificial Neural Networks, Bruges, Belgium*, pages 343–348.

Murray, A. and Edwards, P. (1993). Synaptic weight noise during MLP training: Fault tolerence and training improvements. *IEEE Transactions on Neural Networks*, 4(4):722–725.

Murray, A. and Edwards, P. (1994). Enhanced MLP performance and fault tolerence resulting from synaptic weight noise during training. *IEEE Transactions on Neural Networks*, 5:792–802.

Nagle, R. and Saff, E. (1986). *Fundamentals of differential equations.* Addison-Wesley.

Nicolis, C. and Nicolis, G. (1982). Stochastic aspects of climatic transitions — response to periodic forcing. *Tellus*, 34:1–9.

Nozaki, D., Mar, D., Grigg, P., and Collins, J. (1999). Effects of colored noise on stochastic resonance in sensory neurons. *Phys. Rev. Lett.*, 82:2402–2405.

Ott, H. (1976). *Noise reduction techniques in electronic systems.* John Wiley and Sons.

Palmer, A. and Winter, I. (1992). Cochlear nerve and cochlear nucleus response to the fundamental frequency of voiced speech sounds and harmonic complex tones. *Advances in the Biosciences*, 83:231–239.

Palmer, A. and Winter, I. (1993). Coding of the fundamental frequency of voiced speech sounds and harmonic complexes in the cochlear nerve and ventral cochlear nucleus. In Merchan, M., editor, *The mammalian cochlear nuclei: Organisation and function*, pages 373–384. Plenum press, New York.

Pei, X., Bachmann, K., and Moss, F. (1995). The detection of threshold, noise and stochastic resonance in the FitzHugh-Nagumo model. *Phys. Rev. A*, 206:61–65.

Perez-Uribe, A. and Sanchez, E. (1997). FPGA implementation of a network of neuron-like adaptive elements. In Gerstner, W., Germond, A., Hasler, M., and Nicoud, J. D., editors, *Lecture Notes in Computer Science*, 1327, pages 1247–1252, Lausanne, Switzerland. Springer.

Pickles, J. (1988). *An introduction to the physiology of hearing.* Academic press, 2nd edition.

Plesser, H. (1998a). Noise turns integrate-and-fire neuron into bandpass filter. In Elsner, N. and Wehner, R., editors, *Proceedings of the 26th Gottingeh Neurobiology Conference*, volume II, page 760, Thieme,Stuttgart.

Plesser, H. (1998b). Noise turns integrate-fire neuron into bandpass filter. *Gottingen Neurobiology Report*, II:760.

Plesser, H. (1999). *Aspects of Signal Processing in Noisy Neurons.* PhD thesis, Georg-August-Universität, Göttingen.

Plesser, H. and Geisel, T. (1999). Markov analysis of stochastic resonance in a periodically driven integrate-and-fire neuron. *Physical Review E*, 59(6):7008–7017.

Plesser, H. and Gerstner., W. (2000). Noise in integrate-and-fire neurons: from stochastic input to escape rates. *Neural Computation*, 12:367–384.

Plesser, H. and Tanaka, S. (1997). Stochastic resonance in a model neuron with reset. *Physics Letters A*, 225:228–234.

Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. (1995). *Numerical Recipes in C.* Cambridge University Press, second edition.

Priestley, M. (1982). *Spectral analysis of time series.* Oxford.

Raloff, J. (1996). Is noise a neural necessity? *Science News*, 150:330–331.

Rhode, W. and Greenberg, S. (1994). Encoding of amplitude modulation in the cochlear nucleus of the cat. *J. Neurophysiology*, 71:1797–1825.

Riani, M. and Simonotto, E. (1994). Stochastic resonance in the perceptual interpretation of ambiguous figures: A neural network model. *Physics Review Letters*, 72(19):3120–3123.

Rieke, F., Warland, D., Steveninck, R., and Bialek, W. (1997). *SPIKES: Exploring the neural code.* MIT Press, first edition.

Robinson, F. (1974). *Noise and fluctuations in electronic devices and circuits.* Oxford University Press.

Robinson, J., Asraf, D., Bulsara, A., and Inchiosa, M. (1998). Information-theoretic distance measures and the generalisation of stochastic resonance. *Phys. Rev. Lett.*, 81:2850–2853.

Romand, R. and Avan, P. (1997). Anatomical and functional aspects of the cochlear nucleus. In Ehret, G. and Romand, R., editors, *The central auditory system*, pages 97–191. Oxford University Press.

Sackinger, E. (1997). Measurement of finite precision effects in handwriting- and speech-recognition algorithms. In Gerstner, W., Germond, A., Hasler, M., and Nicoud, J.-D., editors, *Artificial Neural Networks - ICANN'97, 7th Intenational conference*, volume 1327 of *Lecture notes in Computer Science*, pages 1223–1228, Lausanne, Switzerland. Springer.

Scharstein, H. (1979). Input-output relationship of the leaky-integrator neuron model. *J. Math. Biology*, 8:403–420.

Senn, W., Segev, I., and Tsodyks, M. (1998). Reading neural synchrony with depressing synapses. *Neural Computation*, 10(4):815–819.

Shalden, M. and Newsome, W. (1998). The variability of the discharge of cortical neurons for connectivity, computation, and information coding. *J. of Neuroscience*, 18:3870–3896.

Shannon, C. (1948). A mathematical theory of computation. *The Bell System Technical Journal*, XXVII(3):379–423.

Shepherd, G. (1988). *Neurobiology.* Oxford University Press, 2nd edition.

Shepherd, G. (1990). *The synaptic organisation of the Brain.* Oxford University Press, third edition.

Shimokawa, T., Rogel, A., Pakdaman, K., and Sato, S. (1999). Stochastic resonance and spike timing in an ensemble of leaky integrate-and-fire neurons model. *Physical Review E*, 59:3461–3470.

Sjostrand, J., Olsson, V., and Conradi, N. (1999). Quantitative estimations of fovea and extra-foveal retinal circuitry in humans. *Vision research*, 39:2987–2998.

Smith, L. (1996). Onset-based sound segmentation. In Touretzky, D., Mozer, M., and Hasselmo, M., editors, *Advances in Neural Information Processing Systems*, volume 8, pages 729–735. MIT Press.

Smith, L. (1998). Extracting features from the short-term structure of cochlear filtered sound. In Bullinaria, J., Glasspool, D., and Houghton, H., editors, *4th Neural computation and Psychology Workshop*, pages 113–125. Springer Verlag.

Smith, L., Glover, M., and Hamilton, A. (1998). A comparison of a hardware and a software integrate-and-fire neural network for clustering onsets in cochlear filtered sound. In Constantinides, T., Kung, S., Niranjan, M., and Wilson, E., editors, *Neural Networks for Signal Processing VIII: Proceedings of the 1998 Workshop*, pages 516–523. IEEE press.

Softky, W. (1994). Sub-millisecond coincidence detection in active dendritic trees. *Neuroscience*, 58:13–41.

Stemmler, M. (1996). A single spike suffices: the simplest form of stochastic resonance in model neurons. *Network: Computation in Neural Systems*, 7(4):687–716.

Stemmler, M., Usher, M., and Niebur, E. (1995). Lateral interactions in primary visual cortex: A model bridging physiology and pyschophysics. *Science*, 269:1877–1880.

Sterratt, D. (2002). *Spikes, synchrony, sequences and Schistocerca's sense of smell.* PhD thesis, University of Edinburgh.

Stevens, C. and Zador, A. (1998). Novel integrate-and-fire-like model of repetitive firing in cortical neurons. In *Proceedings of the 5th joint symposium on neural computation*, Institute of neural computation USCD, La Jolla.

Stocks, N. and Mannella, R. (2001). Generic noise-enhanced coding in neural arrays. *Phys. Rev. E*, 64:030902–1 – 030902–4.

Tal, D. and Schwartz, E. (1997). Computing with the leaky integrate-and-fire neuron: Logarithmic computation and multiplication. *Neural Computation*, 9:305–318.

Theunissen, F. and Miller, J. (1991). Representation of sensory information in the cricket cercal sensory system. II: Information theoretic calculations of system accuracy and optimal tuning curve widths of four primary interneurons. *J. Neurophysiol*, 66:1690–1703.

Traynelis, S. and Jaramillo, F. (1998). Getting the most out of noise in the central nervous system. *Trends in Neurosciences*, 21:137–145.

Troyer, T. and Miller, K. (1997). Physiological gain leads to high ISI variability in a simple model cortical regular spiking cell. *Neural Computation*, 9:971–983.

Tuckwell, H. (1988a). *Introduction to theoretical neurobiology*, volume I. Cambridge University Press.

Tuckwell, H. (1988b). *Introduction to theoretical neurobiology*, volume II. Cambridge University Press.

Tuckwell, H. (1989). *Stochastic processes in the neurosciences*. SIAM, Philadelphia.

Usher, M. and Feingold, M. (2000). Stochastic resonance in the speed of memory retrieval. *Biological Cybernetics*, 83:L11–L16.

van den Bout, D., Morris, J., Thoae, D., Labrozzi, S., Wingo, S., and Hallman, D. (1992). Anyboard - An FPGA-based reconfigurable system. *IEEE Design and Test of Computers*, 13:21–30.

Villasenor, J. and Mangione-Smith, W. (1997). *Configurable computing.* Scientific American, *http : //www.sciam.com/067issue/067villasenor.htm.*

Volgushev, M. and Eysel, U. (2000). Noise makes sense in neural computing. *Science*, 290:1908–1909.

von der Malsburg, C. (1994). The correlation theory of brain function. In Domany, editor, *Models of neural networks II.* Springer.

von der Malsburg, C. and Schneider, W. (1986). A neural cocktail-party processor. *Biol. Cybern.*, 54:29–40.

Waldemark, J., Lindbald, T., Lindsey, C., Waldemark, K., Oberg, J., and Millberg, M. (1998). Pulse coupled neural network implementation in FPGA. In Rogers, S., Fogel, D., Bezdek, J., and Bosachi, B., editors, *Proceedings of SPIE - The International Society for Optical Engineering: Applications and Science of Computational Intelligence*, volume 3390, pages 392–401.

Weisenfeld, K. and Moss, F. (1995). Stochastic resonance and the benefits of noise: from ice ages to the crayfish and squids. *Nature*, 373:33–36.

Wiesenfeld, K. and Jaramillo, F. (1998). Minireview of stochastic resonance. *Chaos*, 8:539–548.

Wiesenfeld, K., Pierson, D., Pantazelou, E., Dames, C., and Moss, F. (1994). Stochastic resonance on a circle. *Phys. Rev. Lett.*, 72:2125–2129.

Winer, B., Brown, D., and Mchels, K. (1991). *Statistical principles in experimental design.* McGraw-Hill, Inc.

Wolpert, S. and Micheli-Tzanakou, E. (1996). A neuromime in VSLI. *IEEE Transactions on Neural Networks*, 7(2):300–306.

Xie, Y. and Jabri, M. (1992). Analysis of quantization in multi-layer neural networks using statistical model.

Zalanyi, L., Bazso, F., and Erdi, P. (2001). The effect of synaptic depression on stochastic resonance. *Neurocomputing*, 38-40:459–465.

Derivation of membrane potent... Fourier transform for the single neuron case

The LIF equation is given by:

$$\frac{dV}{dt} = -\frac{V}{\tau} + I(t)$$

$$I(t) = a\sum_{(a)} \delta(t - t_i)$$

and

$$\left(\frac{1}{\tau}\right)$$

$$\frac{dV}{dt} = -\frac{V}{\tau} + a\sum_{(a)} \delta(t - t_i)$$

$$\frac{dV}{dt} - \frac{V}{\tau} = \frac{a}{\tau}\sum_{(a)} e^{-\frac{(t-t_i)^2}{2\sigma^2}}$$

Solving for V we have:

$$\frac{d}{dt}(Ve^{\frac{t}{\tau}}) = \frac{a}{\tau}\sum_{(a)} e^{-\frac{(t-t_i)^2}{2\sigma^2}} e^{\frac{t}{\tau}}$$

Appendix A

Derivation of membrane potential Fourier transform for the single neuron case

The LIF equation is given by:

$$\frac{dV}{dt} = -\frac{V}{\tau} + I(t) \tag{A.1}$$

where

$$I(t) = a\sum_{i=1}^{n} \alpha(t - t_i) \tag{A.2}$$

and

$$\alpha(t) = \frac{1}{\tau_s} e^{-\frac{t}{\tau_s}} \tag{A.3}$$

$$\frac{dV}{dt} = -\frac{V}{\tau} + a\sum_{i=1}^{n} \alpha(t - t_i) \tag{A.4}$$

$$\frac{dV}{dt} + \frac{V}{\tau} = \frac{a}{\tau_s}\sum_{i=1}^{n} e^{-\frac{t-t_i}{\tau_s}} \tag{A.5}$$

Solving for V we have:

$$\frac{d}{dt}(Ve^{\frac{t}{\tau}}) = \frac{a}{\tau_s}\sum_{i=1}^{n} e^{-\frac{t-t_i}{\tau_s}} e^{\frac{t}{\tau}} \tag{A.6}$$

$$Ve^{\frac{t}{\tau}} = \int_{t_i}^{t} \sum_{i=1}^{n} \frac{a}{\tau_s} e^{(-\frac{u-t_i}{\tau_s}+\frac{u}{\tau})} du + c \tag{A.7}$$

$$Ve^{\frac{t}{\tau}} = \frac{a}{\tau_s} \sum_{i=1}^{n} e^{-\frac{t_i}{\tau_s}} \int_{t_i}^{t} e^{u(\frac{1}{\tau}-\frac{1}{\tau_s})} du + c \tag{A.8}$$

Therefore $V(t)$ is given by the following:

$$V(t) = \frac{ae^{-\frac{t}{\tau}}}{\tau_s(\frac{1}{\tau}-\frac{1}{\tau_s})} \sum_{i=1}^{n} e^{-\frac{t_i}{\tau_s}} \left[e^{t(\frac{1}{\tau}-\frac{1}{\tau_s}} - e^{t_i(\frac{1}{\tau}-\frac{1}{\tau_s})} \right] \tag{A.9}$$

Fourier transform $G(\omega)$ of $V(t)$ is given by:

$$\begin{aligned} G(\omega) &= \frac{ae^{-\frac{t}{\tau}}}{\tau_s(\frac{1}{\tau}-\frac{1}{\tau_s})} \sum_{i=1}^{n} \int_0^{\infty} e^{-j\omega t} e^{-\frac{t_i}{\tau_s}} \left[e^{t(\frac{1}{\tau}-\frac{1}{\tau_s}} - e^{t_i(\frac{1}{\tau}-\frac{1}{\tau_s})} \right] dt \\ &= \frac{ae^{-\frac{t}{\tau}}}{\tau_s(\frac{1}{\tau}-\frac{1}{\tau_s})} \sum_{i=1}^{n} e^{-\frac{t_i}{\tau_s}} \left[\int_0^{\infty} e^{t(\frac{1}{\tau}-\frac{1}{\tau_s})} e^{-\frac{t}{\tau}} e^{-j\omega t} dt - e^{t_i(\frac{1}{\tau}-\frac{1}{\tau_s})} \int_0^{\infty} e^{-\frac{t}{\tau}} e^{-j\omega t} dt \right] \\ &= \frac{ae^{-\frac{t}{\tau}}}{\tau_s(\frac{1}{\tau}-\frac{1}{\tau_s})} \sum_{i=1}^{n} e^{-\frac{t_i}{\tau_s}} \left[\int_0^{\infty} e^{-t(\frac{1}{\tau}+\frac{1}{\tau_s}-\frac{1}{\tau}+j\omega)} dt - e^{t_i(\frac{1}{\tau}-\frac{1}{\tau_s})} \int_0^{\infty} e^{-t(\frac{1}{\tau}+j\omega)} dt \right] \end{aligned} \tag{A.10}$$

$$G(\omega) = \frac{a}{2\pi\tau_s(\frac{1}{\tau}-\frac{1}{\tau_s})} \sum_{i=1}^{n} e^{-\frac{t_i}{\tau_s}} \left[\frac{1}{\frac{1}{\tau_s}+j\omega} - \frac{e^{t_i(\frac{1}{\tau}-\frac{1}{\tau_s})}}{\frac{1}{\tau}+j\omega} \right] \tag{A.11}$$

(A.12)

After separating the complex part into the real and imaginary we get:

$$\begin{aligned} G(\omega) = \frac{a}{2\pi\tau_s(\frac{1}{\tau}-\frac{1}{\tau_s})} \sum_{i=1}^{n} e^{-\frac{t_i}{\tau_s}} \Bigg[& \frac{\frac{1}{\tau^2\tau_s} + \frac{\omega^2}{\tau_s} - \frac{e^{t_i(\frac{1}{\tau}-\frac{1}{\tau_s})}}{\tau{\tau_s}^2} - \frac{a\omega^2 e^{t_i(\frac{1}{\tau}-\frac{1}{\tau_s})}}{\tau}}{(\frac{1}{{\tau_s}^2}+\omega^2)(\frac{1}{\tau^2}+\omega^2)} \\ & - \frac{j(\frac{\omega}{\tau^2}+\omega^3 - \frac{\omega e^{t_i(\frac{1}{\tau}-\frac{1}{\tau_s})}}{{\tau_s}^2} - \omega^3 e^{t_i(\frac{1}{\tau}-\frac{1}{\tau_s})})}{(\frac{1}{{\tau_s}^2}+\omega^2)(\frac{1}{\tau^2}+\omega^2)} \Bigg] \end{aligned} \tag{A.13}$$

If we concentrate on the complex term in the big square brackets we can compute its power spectrum as follows:

Let $e_i = e^{t_i(\frac{1}{\tau}-\frac{1}{\tau_s})}$ then an approximation of the power spectrum is given by the following:

$$|G(\omega)|^2 \approx \frac{a^2}{4\pi^2(\frac{1}{\tau}-\frac{1}{\tau_s})^2}\sum_{i=1}^{n} e^{-\frac{2t_i}{\tau_s}}\left[\left|\frac{\frac{1}{\tau^2\tau_s}+\frac{\omega^2}{\tau_s}-\frac{e_i}{\tau\tau_s{}^2}-\frac{\omega^2 e_i}{\tau}}{(\frac{1}{\tau_s{}^2}+\omega^2)(\frac{1}{\tau^2}+\omega^2)}\right|^2+\left|\frac{\frac{\omega}{\tau^2}+\omega^3-\frac{\omega e_i}{\tau_s{}^2}-\omega^3 e_i}{(\frac{1}{\tau_s{}^2}+\omega^2)(\frac{1}{\tau^2}+\omega^2)}\right|^2\right] \tag{A.14}$$

$$|G(\omega)|^2 \approx \frac{a^2}{4\pi^2(\frac{1}{\tau}-\frac{1}{\tau_s})^2}\sum_{i=1}^{n} e^{-\frac{2t_i}{\tau_s}}\left[\frac{\frac{1}{\tau^4\tau_s{}^2}+\frac{\omega^2}{\tau^2\tau_s{}^2}-\frac{e_i}{\tau^3\tau_s{}^3}-\frac{\omega^2 e_i}{\tau^3\tau_s}+\frac{\omega^2}{\tau^2\tau_s{}^2}+\frac{\omega^4}{\tau_s{}^2}-\frac{\omega^2 e_i}{\tau\tau_s{}^3}-\frac{\omega^4 e_i}{\tau\tau_s}}{(\frac{1}{\tau_s{}^2}+\omega^2)^2(\frac{1}{\tau^2}+\omega^2)^2}\right.$$
$$-\frac{\frac{e_i}{\tau^3\tau_s{}^3}-\frac{\omega^2 e_i}{\tau\tau_s{}^3}+\frac{{e_i}^2}{\tau^2\tau_s{}^4}+\frac{\omega^2 {e_i}^2}{\tau^2\tau_s{}^2}-\frac{\omega^2 e_i}{\tau^3\tau_s}-\frac{\omega^4 e_i}{\tau\tau_s}+\frac{\omega^2 {e_i}^2}{\tau^2\tau_s{}^2}+\frac{\omega^4 {e_i}^2}{\tau^2}}{(\frac{1}{\tau_s{}^2}+\omega^2)^2(\frac{1}{\tau^2}+\omega^2)^2}$$
$$+\frac{\frac{\omega^2}{\tau^4}+\frac{\omega^3}{\tau^2}-\frac{\omega^2 e_i}{\tau^2\tau_s{}^3}-\frac{\omega^3 e_i}{\tau^2}+\frac{\omega^3}{\tau^2}+\omega^4-\frac{\omega^3 e_i}{\tau_s{}^2}-\omega^4 e_i}{(\frac{1}{\tau_s{}^2}+\omega^2)^2(\frac{1}{\tau^2}+\omega^2)^2}$$
$$\left.-\frac{\frac{\omega^2 e_i}{\tau_s{}^2\tau^2}-\frac{\omega^3 e_i}{\tau_s{}^2}+\frac{\omega^2 {e_i}^2}{\tau_s{}^4}-\frac{\omega^3 {e_i}^2}{\tau_s{}^2}-\frac{\omega^3 e_i}{\tau^2}-\omega^4 e_i+\frac{\omega^3 {e_i}^2}{\tau_s{}^2}+\omega^4 {e_i}^2}{(\frac{1}{\tau_s{}^2}+\omega^2)^2(\frac{1}{\tau^2}+\omega^2)^2}\right] \tag{A.15}$$

Appendix B

Derivation of membrane potential Fourier transform for the many neurons case

The LIF equation is given by:

$$\frac{dV}{dt} = -\frac{V}{\tau} + I(t) \tag{B.1}$$

where

$$I(t) = a\sum_{k}^{N}\sum_{i}^{n}\alpha(t - t_{ki}) \tag{B.2}$$

and

$$\alpha(t) = \frac{1}{\tau_s}e^{-\frac{t}{\tau_s}} \tag{B.3}$$

$$\frac{dV}{dt} = -\frac{V}{\tau} + a\sum_{k=1}^{N}\sum_{i=1}^{n}\alpha(t - t_{ki}) \tag{B.4}$$

$$\frac{dV}{dt} + \frac{V}{\tau} = \frac{a}{\tau_s}\sum_{k=1}^{N}\sum_{i=1}^{n}e^{-\frac{t-t_{ki}}{\tau_s}} \tag{B.5}$$

Solving for V we have:

$$\frac{d}{dt}(Ve^{\frac{t}{\tau}}) = \frac{a}{\tau_s}\sum_{k=1}^{N}\sum_{i=1}^{n}e^{-\frac{t-t_{ki}}{\tau_s}}e^{\frac{t}{\tau}} \tag{B.6}$$

$$
\begin{aligned}
Ve^{\frac{t}{\tau}} &= \frac{1}{\tau_s}\int_0^t a\sum_{k=1}^{N}\sum_{i=1}^{n}\alpha(u-t_{ki})e^{\frac{u}{\tau}}du \\
&= \frac{a}{\tau_s}\sum_{k=1}^{N}\sum_{i=1}^{n}\int_0^t \alpha(u-t_{ki})e^{-\frac{u}{\tau}}du \\
&= \frac{a}{\tau_s}\sum_{k=1}^{N}\sum_{i=1}^{n}\int_{t_{ki}}^t e^{-\frac{u-t_{ki}}{\tau_s}}e^{-\frac{u}{\tau}}bu \\
&= \frac{a}{\tau_s}\sum_{k=1}^{N}\sum_{i=1}^{n}e^{\frac{t_{ki}}{\tau_s}}\int_{t_{ki}}^t e^{\frac{u}{\tau}}e^{-\frac{u}{\tau_s}}du \\
&= \frac{a}{\tau_s}\sum_{k=1}^{N}\sum_{i=1}^{n}e^{\frac{t_{ki}}{\tau_s}}\int_{t_{ki}}^t e^{u(\frac{1}{\tau}-\frac{1}{\tau_s})}du \\
&= \frac{a}{\tau_s}\sum_{k=1}^{N}\sum_{i=1}^{n}e^{\frac{t_{ki}}{\tau_s}}\left[\frac{1}{\frac{1}{\tau}-\frac{1}{\tau_s}}e^{u(\frac{1}{\tau}-\frac{1}{\tau_s})}\right]_{t_{ki}}^t \\
&= \frac{a}{\tau_s(\frac{1}{\tau}-\frac{1}{\tau_s})}\sum_{k=1}^{N}\sum_{i=1}^{n}e^{\frac{t_{ki}}{\tau_s}}\left[e^{t(\frac{1}{\tau}-\frac{1}{\tau_s})}-e^{t_{ki}(\frac{1}{\tau}-\frac{1}{\tau_s})}\right]
\end{aligned}
$$

$$
V(t) = e^{-\frac{t}{\tau}}\frac{a}{\tau_s(\frac{1}{\tau}-\frac{1}{\tau_s})}\sum_{k=1}^{N}\sum_{i=1}^{n}e^{\frac{t_{ki}}{\tau_s}}\left[e^{t(\frac{1}{\tau}-\frac{1}{\tau_s})}-e^{t_{ki}(\frac{1}{\tau}-\frac{1}{\tau_s})}\right] \tag{B.7}
$$

The Fourier transform $G(\omega)$ for $V(t)$ is given by:

$$
\begin{aligned}
G(\omega) &= \frac{a}{2\pi\tau_s(\frac{1}{\tau}-\frac{1}{\tau_s})}\sum_{k=1}^{N}\sum_{i=1}^{n}\int_0^\infty e^{-\frac{t}{\tau}}e^{-j\omega t}e^{\frac{t_{ki}}{\tau_s}}\left[e^{t(\frac{1}{\tau}-\frac{1}{\tau_s})}-e^{t_{ki}(\frac{1}{\tau}-\frac{1}{\tau_s})}\right]dt \\
&= \frac{a}{2\pi\tau_s(\frac{1}{\tau}-\frac{1}{\tau_s})}\sum_{k=1}^{N}\sum_{i=1}^{n}e^{-t_{ki}}\left[\int_0^\infty e^{-\frac{t}{\tau}}e^{-j\omega t}e^{t(\frac{1}{\tau}-\frac{1}{\tau_s})}dt - e^{t_{ki}(\frac{1}{\tau}-\frac{1}{\tau_s})}\int_0^\infty e^{-\frac{t}{\tau}}e^{-j\omega t}dt\right] \\
&= \frac{a}{2\pi\tau_s(\frac{1}{\tau}-\frac{1}{\tau_s})}\sum_{k=1}^{N}\sum_{i=1}^{n}e^{-\frac{t_{ki}}{\tau_s}}\left[\int_0^\infty e^{-t(-\frac{1}{\tau}+j\omega+(\frac{1}{\tau}-\frac{1}{\tau_s}))}dt - e^{-t_{ki}(\frac{1}{\tau}-\frac{1}{\tau_s})}\int_0^\infty e^{-t(\frac{1}{\tau}+j\omega)}dt\right] \\
&= \frac{a}{2\pi\tau_s(\frac{1}{\tau}-\frac{1}{\tau_s})}\sum_{k=1}^{N}\sum_{i=1}^{n}e^{-\frac{t_{ki}}{\tau_s}}\left[-\frac{1}{j\omega-\frac{1}{\tau_s}}e^{-t(j\omega+-\frac{1}{\tau_s})}\Big|_0^\infty - e^{-t_{ki}(\frac{1}{\tau}-\frac{1}{\tau_s})}\left[-\frac{1}{-j\omega+\frac{1}{\tau}}e^{-t(j\omega+\frac{1}{\tau})}\right]\Big|_0^\infty\right] \\
&= \frac{a}{2\pi\tau_s(\frac{1}{\tau}-\frac{1}{\tau_s})}\sum_{k=1}^{N}\sum_{i=1}^{n}e^{-\frac{t_{ki}}{\tau_s}}\left[\frac{1}{j\omega-\frac{1}{\tau_s}}-\frac{e^{-t_{ki}(\frac{1}{\tau}-\frac{1}{\tau_s})}}{j\omega+\frac{1}{\tau}}\right]
\end{aligned}
$$

Therefore:

$$
G(\omega) = \frac{a}{2\pi\tau_s(\frac{1}{\tau}-\frac{1}{\tau_s})}\sum_{k=1}^{N}\sum_{i=1}^{n}e^{-\frac{t_{ki}}{\tau_s}}\left[\frac{1}{j\omega-\frac{1}{\tau_s}}-\frac{e^{-t_{ki}(\frac{1}{\tau}-\frac{1}{\tau_s})}}{j\omega+\frac{1}{\tau}}\right] \tag{B.8}
$$

After normalising the denominators we get:

$$G(\omega) = \frac{a}{2\pi\tau_s(\frac{1}{\tau}-\frac{1}{\tau_s})}\sum_{k=1}^{N}\sum_{i=1}^{n} e^{-\frac{t_{ki}}{\tau_s}}\left[\frac{j\omega+\frac{1}{\tau_s}}{\omega^2+\frac{1}{\tau_s{}^2}} - \frac{e^{-t_{ki}(\frac{1}{\tau}-\frac{1}{\tau_s})}(\frac{1}{\tau}-j\omega)}{\frac{1}{\tau^2}+\omega^2}\right] \tag{B.9}$$

Putting everything under the same denominator we get:

$$G(\omega) = \frac{a}{2\pi\tau_s(\frac{1}{\tau}-\frac{1}{\tau_s})}\sum_{k=1}^{N}\sum_{i=1}^{n} e^{-\frac{t_{ki}}{\tau_s}}\left[\frac{(j\omega+\frac{1}{\tau_s})(\frac{1}{\tau^2}+\omega^2) - e^{-t_{ki}(\frac{1}{\tau}-\frac{1}{\tau_s})}(\frac{1}{\tau}-j\omega)(\omega^2+\frac{1}{\tau_s{}^2})}{(\omega^2+\frac{1}{\tau_s{}^2})(\frac{1}{\tau^2}+\omega^2)}\right] \tag{B.10}$$

Let $e_{ki} = e^{-t_{ki}(\frac{1}{\tau}-\frac{1}{\tau_s})}$ and after separating the complex part into real and imaginary parts the Fourier transform is given by the following:

$$G(\omega) = \frac{a}{2\pi\tau_s(\frac{1}{\tau}-\frac{1}{\tau_s})}\sum_{k=1}^{N}\sum_{i=1}^{n} e^{-\frac{t_{ki}}{\tau_s}}\left[\frac{\frac{1}{\tau_s\tau^2}+\frac{e_{ki}}{\tau\tau_s{}^2}+\frac{\omega^2}{\tau_s}+\frac{\omega^2 e_{ki}}{\tau}}{(\omega^2+\frac{1}{\tau_s{}^2})(\frac{1}{\tau^2}+\omega^2)} - \frac{j(\frac{\omega}{\tau^2}-\frac{\omega e_{ki}}{\tau_s{}^2}+\omega^3-\omega^3 e_{ki})}{(\omega^2+\frac{1}{\tau_s{}^2})(\frac{1}{\tau^2}+\omega^2)}\right] \tag{B.11}$$

The power spectrum is calculated as follows:

$$|G(\omega)|^2 \approx \frac{a^2}{4\pi^2(\frac{1}{\tau}-\frac{1}{\tau_s})^2}\sum_{k=1}^{N}\sum_{i=1}^{n} e^{-\frac{2t_{ki}}{\tau_s}}\left[\left|\frac{\frac{1}{\tau_s\tau^2}+\frac{e_{ki}}{\tau\tau_s{}^2}+\frac{\omega^2}{\tau_s}+\frac{\omega^2 e_{ki}}{\tau}}{(\omega^2+\frac{1}{\tau_s{}^2})(\frac{1}{\tau^2}+\omega^2)}\right|^2 + \left|\frac{(\frac{\omega}{\tau^2}-\frac{\omega e_{ki}}{\tau_s{}^2}+\omega^3-\omega^3 e_{ki})}{(\omega^2+\frac{1}{\tau_s{}^2})(\frac{1}{\tau^2}+\omega^2)}\right|^2\right] \tag{B.12}$$

$$\begin{aligned}
|G(\omega)|^2 \approx \frac{a^2}{4\pi^2(\frac{1}{\tau}-\frac{1}{\tau_s})^2}\sum_{k=1}^{N}\sum_{i=1}^{n} e^{-\frac{2t_{ki}}{\tau_s}}\Bigg[&\frac{\frac{1}{\tau_s{}^2\tau^4}+\frac{e_{ki}}{\tau^3\tau_s{}^3}+\frac{\omega^3}{\tau^2\tau_s{}^2}+\frac{\omega^2 e_{ki}}{\tau_s\tau^3}}{(\omega^2+\frac{1}{\tau_s{}^2})^2(\frac{1}{\tau^2}+\omega^2)^2} \\
&+\frac{\frac{e_{ki}}{\tau_3\tau_s{}^2}+\frac{e_{ki}{}^2}{\tau^2\tau_s{}^4}+\frac{\omega e_{ki}}{\tau\tau_s{}^3}+\frac{e_{ki}{}^2\omega^2}{\tau^2\tau_s{}^2}}{(\omega^2+\frac{1}{\tau_s{}^2})^2(\frac{1}{\tau^2}+\omega^2)^2} \\
&+\frac{\frac{\omega^2}{\tau_s{}^2\tau^2}+\frac{\omega^2 e_{ki}}{\tau\tau_s{}^3}+\frac{\omega^4}{\tau_s{}^2}+\frac{\omega^4 e_{ki}}{\tau\tau_s}}{(\omega^2+\frac{1}{\tau_s{}^2})^2(\frac{1}{\tau^2}+\omega^2)^2} \\
&+\frac{\frac{\omega^2 e_{ki}}{\tau_s\tau^3}+\frac{\omega^2 e_{ki}}{\tau_s{}^2\tau^2}+\frac{\omega^4 e_{ki}}{\tau_s\tau}+\frac{\omega^4 e_{ki}{}^2}{\tau^2}}{(\omega^2+\frac{1}{\tau_s{}^2})^2(\frac{1}{\tau^2}+\omega^2)^2} \\
&+\frac{\frac{\omega^2}{\tau^4}-\frac{\omega^2 e_{ki}}{\tau^2\tau_s{}^2}+\frac{\omega^4}{\tau^2}-\frac{\omega^4 e_{ki}}{\tau}}{(\omega^2+\frac{1}{\tau_s{}^2})^2(\frac{1}{\tau^2}+\omega^2)^2} \\
&-\frac{\frac{\omega^2 e_{ki}}{\tau^2\tau_s{}^2}+\frac{\omega^2 e_{ki}{}^2}{\tau_s{}^4}-\frac{\omega^4 e_{ki}}{\tau_s{}^2}+\frac{\omega^4 e_{ki}{}^2}{\tau_s{}^2}}{(\omega^2+\frac{1}{\tau_s{}^2})^2(\frac{1}{\tau^2}+\omega^2)^2} \\
&+\frac{\frac{\omega^4}{\tau^2}-\frac{\omega^4 e_{ki}}{\tau_s{}^2}+\omega^6-\omega^6 e_{ki}-\frac{\omega^4 e_{ki}}{\tau^2}+\frac{\omega^4 e_{ki}}{\tau_s{}^2}-\omega^6 e_{ki}+\omega^6 e_{ki}{}^2}{(\omega^2+\frac{1}{\tau_s{}^2})^2(\frac{1}{\tau^2}+\omega^2)^2}\Bigg]
\end{aligned} \tag{B.13}$$