Department of informatics

Low-Power Stochastic Arithmetic Feed-Forward Neural Network

Jon-Erik Ruth

Main Subject Thesis

August 11. 1994

Preface

This report is the written part of my work for the Cand.Scient degree in computer science at the Department of Informatics, University of Oslo.

I thank my advisor Yngvar Berg for accepting me as one of his students, and so making this work possible, and for introducing me to the very interesting field of neural computation. I will also thank him for being encouraging, and allowing me a great freedom in choice of methods and solutions. Many thanks to Tor Sverre Lande as well, for his constructive critisism of circuit implementations.

Jon-Erik Ruth

Blindern, February 24, 1995

Contents

List of Figures

CHAPTER

1

Introduction

In the traditional computational paradigm, which was introduced by von Neumann, problems are solved by an ordered set of instructions known as a program. These instructions are fed to the *CPU* in a sequential order, and processed in this order. An instruction typically just do a simple arithmetic operation or change the location of where to fetch the next instruction – a jump-instruction. To solve complex problems one usually have to process the same set of instructions thousands of times inside tight loops, that can be nested inside other tight loops. This means that it can take billions of instructions to solve such problems. Even the fastest supercomputers existing today, running advanced *AI* systems, can not perform real time visual or auditory recognition like you and me, but even simple mammals perform such visual and auditory recognition constantly.

The biological brains ability to perform very complex tasks, with a minimum set of activities and resources, has inspired scientists to study the field of neural computation, and to try to implement artificial neural systems. There are also other properties associated with the biological brain that are desirable. It is very fault tolerant, and even though there are dying nerve cells every minute, it is still functioning without significant loss of performance. It is not like a digital computer where one damaged transistor can make the whole system go down. It is also very flexible, and is continually adapting to the psychological environment, that is, it can be trained to cope with new situations. It is also massively parallel, and has the ability to deal with inconsistent and noisy information. Last, but not least important, is the size of the brain and power consumption compared to its computational abilities.

There is a great difference in the complexity of the computational elements in traditional computers and neural systems. In a digital *CPU* there are a few but very complex elements performing the instructions, and the processing of the instructions is very fast. In neural systems on the other hand there are lots of small simple elements performing a very local computation. These elements are organized in highly parallel networks, and even though the biological processing elements are slow, compared to digital silicon circuits, the high parallelism ensures low total processing times. Most of the networks are organized without feedback loops, so there is a linear dependency between the depth of the network and the total processing time. Only in tasks based on solving numerical

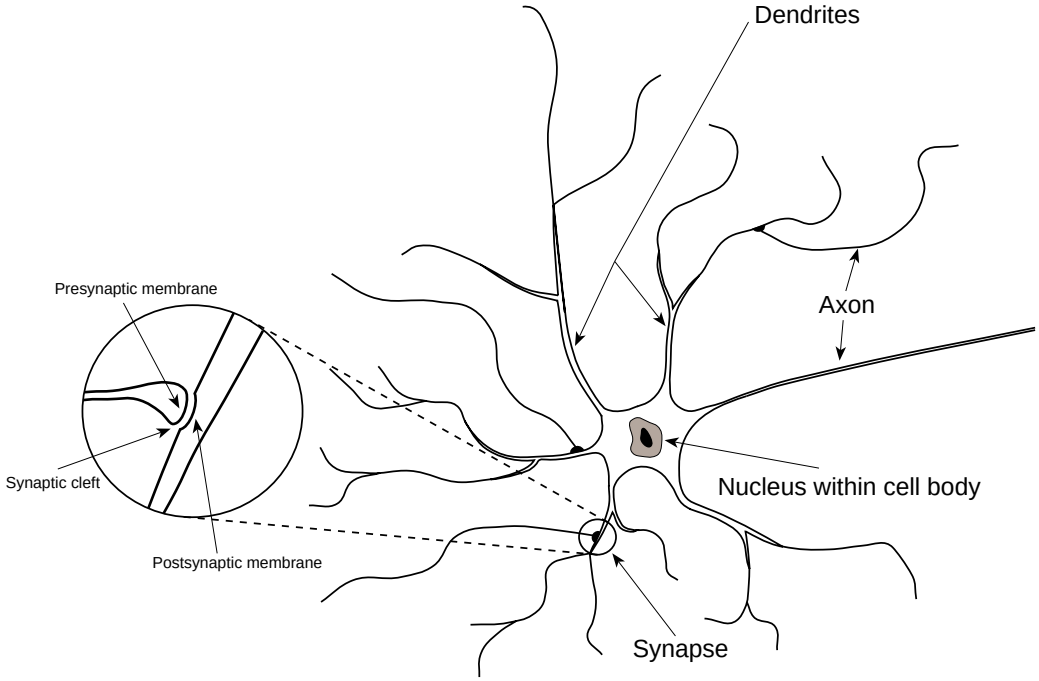


Figure 1.1: Biological neuron. (Source: Adapted from [Mead, 1989b])

equations or answering well defined questions, with high accuracy, the digital computer is superior.

1.1 Biological neural networks - a brief description

The human brain consists of approximately 10^{11} computational elements [Hertz *et al.*, 1991]. These elements are of different types, but they are all called neurons or nerve cells. They are organized in a huge parallel network where the output of one typical neuron can be connected to the inputs of a few thousand other neurons. The number of inputs to one neuron can range from just a few to about a hundred thousand.

The type of neuron covered here have extensions from the cell body, organized in tree-like structures, called dendrites (see figure 1.1). These nerve fibers serves the purpose of transmitting electrical charge to and from the cell body. The whole structure is called a dendritic tree. The dendrites are covered with elements called synapses, and it is through these synapses most of the neural interaction, and primary information-processing, takes place. Most neurons have one long fiber extending from the cell body, called the axon. This fiber split up into branches, and each branch finally connect to a synapse on another neuron. The axon therefore take the function as the neurons output, and the synapses serves as inputs. There are other types of communication in a biological brain, like in neurons without an axon, but these aspects are not covered in this text

The signal type used for communication between neurons are electric pulses with an amplitude of a few hundred millivolt, and a duration in the range of milliseconds. When such a pulse reaches the synapse on another neuron, the change of potential on the presynaptic membrane initiate a chain of chemical processes in the synaptic cleft,

which results in the opening of ion-specific channels in the postsynaptic membrane. The channels are either for positively or negatively charged ions. The conductance of the postsynaptic membrane either charges or discharges the capacitance in the cell body. The amount of charge conducted depends on the potential on the presynaptic membrane, and the existence of chloride channels which increases the conductance of the membrane.

When the potential of the cell body reaches a given threshold voltage, usually about $-40mV$, a chemical process inside the cell body generates a pulse that is propagated down the axon. While the neuron is in a *firing* state, it lacks the ability to fire again. It has to wait a specific time called a refractory period, which brings the potential of the cell body back to its initial state, before it can fire again.

The complexity of a neural system does not derive from the complexity of its components, but from the complexity of the interaction between these components. Even though we do, to some extent, understand the behavior of these components, the organization of them are by no means fully understood. A description of some network-types can be found in [Kandel and Schwartz, 1985]. It is clear that the interconnection of neural components is the secret of thought, and it is assumed that the large connectivity between the components, is the most significant reason for the high redundancy of neural systems.

The choice of signal representation for information transmission in biological systems is not a result of accidental circumstances. During millions of years of natures own deadly effective evaluation system, named *natural selection*, biological nervous systems have evolved to the size, complexity and versatility of human brains. This makes it very difficult to argue against the choice of information representation in natural systems. One can try to use the poor signal transmission abilities of chemically based transmission media as the axon, to explain the development of pulse representation. But any computational or transmitting media of the physical size of neurons, synapses and axons, implemented in a practically manageable material, will introduce a large potential for noise and variations of the media responses to electrical potentials and currents. It should be no doubt that to achieve such massively parallelism and large number of computational elements as in biological brains, the physical size of the computational building blocks and signal lines must be minimized. The pulse representation is therefore a consequence of other factors of much more importance than poor transmission media.

Details about biological topics can be found in [Shepherd, 1979] and [Mead, 1989b].

1.2 Artificial neural networks

The first approach to model the computational operation performed by a neuron was done by [McCulloch and Pitts, 1943], and the fundamental issues of their work are presented in [Hertz *et al.*, 1991]. They proposed a simple model of a neuron as a binary threshold unit. This unit perform a weighting of its inputs from other units, sums up these inputs, and then outputs a one or a zero depending on whether this sum is larger or smaller than a certain threshold μ:

$$o_i = \Theta(\sum_j w_{ij} o_j - \mu_i),$$

where $\Theta(x)$ is the unit *step function*

$$\Theta(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} .$$

A binary '1' and '0' on the output represent a *firing* and *non-firing* state respectively. This is a straightforward binary modeling of the biological neuron discussed in the previous section. Most of the existing network models use, with some variations, this implementation of a neuron.

Due to the lack of knowledge about what kind of algorithm the brain use in its training process, one usually do not try to reconstruct the biological counterpart to a great detail when designing an artificial implementation of the learning part of neural networks. It is more convenient to develop abstract mathematical models for training that resembles the biological neural networks to some degree. The key aspects of biological neural systems, ie. simple and consistent building-blocks, high redundancy and adaptability, are included in these models. The most common models are the Hopfield model [Hopfield, 1982], recurrent networks and layered feed-forward networks.

Most implementations of artificial neural networks do not use pulses to represent information. In software implementations all quantities are represented by integer or real values. This is also the case for most digital implementations. In most analog implementations voltages and currents represent the synaptic weights and firing rates of neurons. Only a few serious attempts have been made to actually implement artificial neural networks incorporating the beneficial aspects of representing information with pulses.

1.3 The Issues

The main issue of this thesis is to implement an artificial neural network of the layered feed-forward type, usually called perceptron networks or just perceptrons, in hardware. The network incorporates an on-chip backpropagation of error learning algorithm for training, and on-chip storage of synaptic weights in UV-light programmable analog memories. Both on-chip learning and storing of weights are essential if one wants to make cheap autonomous neural networks capable of solving real-time problems in embedded systems.

At an early stage I decided that the network to be implemented should be able to solve a problem that is not linearly separable, like the parity problem. Such problems are hard to solve, and a network solving these kind of problems must have hidden layers. Linearly inseparable problems are therefore often used for testing and evaluation of neural network designs.

Backed with nature's evolutionary conclusion that pulse representation is the best way of representing an analog value in a network of computational elements where element size should be downward scalable and network size upward scalable, it should be reasonable to use this representation in a hardware implementation of an artificial neural network too. Even though we do not know how the learning part of biological neural systems is implemented, which means we can not be certain about the signal representation used in the learning part, pulse streams are both used in the feed-forward and backpropagation part of the network. The reason for this was that more compatibility between processing elements was ensured if all of them used the same coding.

The same type of basic computational circuitry can be used in both the feed-forward and the backpropagation calculation.

The exact representation of a value as a pulse stream used in the implementation is by no means a copy of the biological counterpart. It is used a stochastic representation, which ensures that complex computations can be performed with only a minimum requirement of hardware. One of the nicest non-linearities of biological neural systems, the non-linear saturation of signal quantities, is also easily implemented with this representation. From a more philosophical point of view,

> one may hypothesize that a little ingredient of stochastics in nervous systems as it is provided by some noise from unreliable elements and the indeterminism originating from probabilistic processing is by no means an evolutionary accident but a precise reflection of certain environmental conditions which otherwise would have been very difficult to catch.
>
> [Banzhaf, 1988].

All the computations including the stochastically represented values are based on boolean algebra. This means that the circuitry implementing the stochastic computing elements, which is the elements performing both the feed-forward and backpropagation of error computation, is of a digital type. To ensure high network scalability, that is eliminate problems with high power consumption and heat dissipation, the digital circuits are implemented in a low power design.

All storing of synaptic weights are done on-chip in analog UV-programmable floating gate memories. The circuitry for programming these memories, and converting the programmed analog voltage to a pulse-encoded weight, are of course analog.

Two chips were made to test the network. The first chip contained all necessary test structures to investigate the behavior of the network elements thoroughly. This chip was implemented with the $2\mu m$ P-well process from Orbit Semiconductor Inc. The second chip contained the complete network, with two additional test structures. For this chip the $2\mu m$ N-well analog process, from the same silicon foundry, was used for the physical implementation.

In chapter 2, I present background material of feed-forward neural networks and the backpropagation of error learning algorithm, and I also give an introduction to low power digital CMOS circuits.

In chapter 3, the specialized theoretical equations, modeling a network with pulse stream signals in both the feed-forward and backpropagation part, are derived and the circuitry implementing these equations is presented.

In chapter 4 the necessary hardware to generate the error of an output signal from the network is presented, together with simulations or measured results of the circuits behavior.

In chapter 5, I investigate two different types of analog memory devices. Both types use a floating-gate of a transistor as capacitor to store the programmed value, but two different approaches are used to program the floating-gate. One use UV-light exposure to lower the resistance of the silicon dioxide so current can pass, the other use Fowler-Nordheim tunneling. These investigations leads to the choice of a UV-light memory for my network. The circuitry needed to program the floating-gate, and to convert the stored voltage to a pulse stream representing the weight is then presented.

In chapter 6, the different circuits developed in chapter 3, 4 and 5 are assembled into a network with 3 inputs, 6 hidden neurons and 2 outputs. Measurements of the performance of the network are also presented here.

Chapter 7 gives a summary and conclusion of the work. What works as expected, and what can be done to improve the network even further.

CHAPTER

2

Background

In this chapter a brief introduction to the fields of feed-forward neural networks and low power digital CMOS design is given. The former will give the reader the background material to understand the development and implementation of the network building-blocks at a system-level, while the latter will help understand transistor-level implementations.

2.1 Feed-forward neural networks

2.1.1 The problem

In all types of neural networks the problem is to map a set of input-patterns to a set of output-patterns. To solve this problem we need at least a layer of input-neurons and a layer of output-neurons. If the output-patterns are very different from the input-patterns, like in a linearly inseparable problem as the parity problem, we also need hidden layers of neurons, as stated by [Rumelhart *et al.*, 1986, Hertz *et al.*, 1991]. With the expression *hidden layer* one refers to a layer of neurons that is not directly connected to the inputs or outputs. A typical structure of a feed-forward neural network with hidden layers is shown in figure 2.1.

2.1.2 Network topology and fault tolerance

It has been shown that it is sufficient with one layer of hidden neurons to approximate any continuous function to any desired accuracy [Hornik *et al.*, 1989]. But this theorem does not state anything about *how many* neurons that are required in the hidden layer to approximate a given function, and there is no such rule for networks with several hidden layers either. The only thing we now is that for networks with only one hidden layer, the number of neurons in this layer may increase exponentially with the number of inputs [Hertz *et al.*, 1991]. The design of network topology is therefore still a black art, and I will not cover it any further in this text.

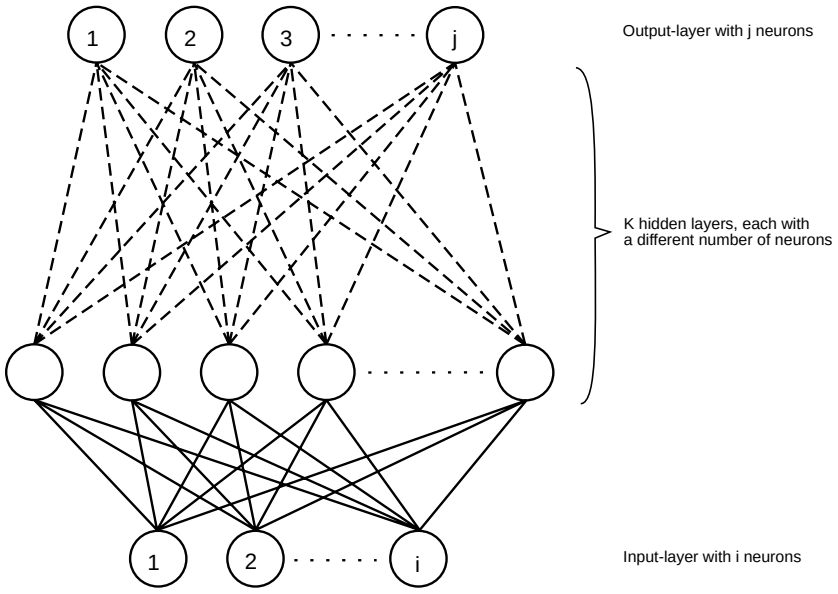


Figure 2.1: A typical layered structure of a feed-forward network

All artificial neural networks are highly scalable. There is no limit of the number of neurons in a layer, or the number of inputs and outputs of a neuron. This makes it possible to implement networks that are fully functional even if parts of them are not working. A network with more inputs, outputs and hidden neurons than strictly necessary to solve a specific problem, can solve the problem even if some neurons are *dead* at the manufacturing time, and/or are damaged due to wear out.

2.1.3 The feed-forward computation

A very important aspect of layered feed-forward networks is that all signals involved in the mapping of a input-pattern to a output-pattern, is fed from the outputs of neurons in one layer, to the inputs of neurons in the *next* layer. There is *no* feedback of signals during this mapping, neither between a neuron and neurons in previous layers or neurons in the same layer. Usually there are no connections to neurons more than one layer ahead, as shown in figure 2.1, but sometimes though, some neurons can be shortened to simplify a network. But then of course you must be sure that the network you get can be trained to solve the problem it is supposed to do. [Rumelhart *et al.*, 1986] show some examples of such networks.

The computation performed by a neuron in a feed-forward neural network is almost the same that [McCulloch and Pitts, 1943] proposed. The output is an explicit function of the input, and is described by

$$o_{pj} = f_j\left(\sum_i w_{ji} o_{pi} + \theta_j\right) = f_j\left(net_{pj}\right), \tag{2.1}$$

where w_{ji} is the weight of input i to neuron j, o_{pi} is input i, that is output i from the previous layer, for input-pattern p, θ_j is the threshold value and f_j is the activation function for neuron j. Before we specify the activation function any further, we better take a look at the learning algorithm, since this algorithm put some limits on it.

2.1.4 Learning by backpropagation of error

There are several different methods for setting synaptic weights and threshold values. The most common, and the one I will use in my implementation, is the backpropagation of error algorithm. This is a *supervised learning* algorithm, which means that you have to teach the network how to respond on a particular set of input-patterns. This teaching incorporates the following steps:

- Present an input-pattern.
- Read out the produced output-pattern.
- Compare the produced output-pattern with the desired output-pattern, and generate an error signal if there is a difference.
- This error-signal is fed to the output-neurons, and propagated through the network in the opposite direction of the feed-forward signals.
- The weights and thresholds are then changed on basis of these error signals to reduce the difference between the output and the target.

These steps are either repeated in discrete steps or performed simultaneously in a *true* parallel and continuous manner for all input-patterns, until the network responds correctly for all of them.

The learning algorithm can be expressed with two equations. One that measures the error on the output, and one that expresses the change of a given weight. The error is usually measured as the difference between the desired output, or target, and the actual output. The weight-change function is then defined to be proportional to the derivative of the square of the measured error for each output-pattern with respect to each weight, and with negative constant of proportionality [Rumelhart *et al.*, 1986]. This will implement a gradient decent search in the error space for the minimum error. To be more specific, let

$$E_p = \frac{1}{2}\sum_j \left(t_{pj} - o_{pj}\right)^2 \tag{2.2}$$

be the square of the error measured for input-pattern p, where t_{pj} represents the target for output neuron j, and o_{pj} the actual output. The weight-change equation is then defined to be

$$\Delta_p w_{ji} = -\eta \frac{\partial E_p}{\partial w_{ji}}\,, \tag{2.3}$$

where $\Delta_p w_{ji}$ is the change of the weight, and η is a scaling-factor that defines the learning rate of the algorithm. The solution to this differentiation can be stated in two equations, depending on the weight under consideration. If the weight belongs to an output-neuron the differentiation is straightforward, and we get

$$\Delta_p w_{ji} = -\eta \frac{\partial E_p}{\partial w_{ji}} = \eta \left(t_{pj} - o_{pj}\right) f_j'\left(net_{pj}\right) o_{pi} = \eta \delta_{pj} o_{pi}\,, \tag{2.4}$$

where f_j' is the derivative of the activation function for output-neuron j and o_{pi} is input i to this neuron for pattern p. The δ-term is only the standard way of expressing the

error scaled by the derivative of the output. If the weight belongs to a hidden neuron, one apply the chain rule to equation 2.3 and get

$$\Delta_p w_{ji} = -\eta \frac{\partial E_p}{\partial w_{ji}} = \eta \left(\sum_k \delta_{pk} w_{kj} \right) f_j' \left(net_{pj} \right) o_{pi} = \eta \delta_{pj} o_{pi} \,, \tag{2.5}$$

where δ_{pk} is the δ for neuron k in the subsequent layer. Since threshold values can be looked upon as weighted inputs where the input values are always clamped at -1 or 1, these can be trained with the same set of equations. This implementation of the backpropagation of error learning algorithm is called the *generalized delta rule*, and the derivations of equation 2.4 and 2.5 can be found in [Rumelhart *et al.*, 1986, Hertz *et al.*, 1991].

2.1.5 The activation function

If we take a closer look at equation 2.4, we see that the activation function has to be differentiable. This excludes the unit step function that [McCulloch and Pitts, 1943] used, since this function is discontinuous and therefore not differentiable. Instead we pick a non-linear and differentiable function. The function has to be non-linear, because hidden neurons with linear activation function provides no advantage over plain two-layered networks [Rumelhart *et al.*, 1986]. The function must also have the property of being non-decreasing. That means a neuron shall not be able to go from a firing state to a non-firing state with an increase in the input to the neuron [Rumelhart *et al.*, 1986]. One usually also wants the activation function to saturate at both extremes to keep the feed-forward signals from getting out of range, even though this is not necessary in theory.

2.2 Low power digital CMOS design

I have already pointed out that scalability and fault tolerance is two key aspects of neural networks. These networks are therefore well suited for *Ultra Large Scale Integration* (ULSI). Chips covering a whole wafer may be approached. But even though neural networks have high redundancy to the manufacturing errors which will be present in chips of these sizes, the power consumption of such large chips may be a problem. Using a standard complementary CMOS digital implementation, the heat dissipation will be high enough to destroy the chip. So a fully scalable implementation must have very low power consumption, to reduce heat dissipation.

As mentioned in the introduction, the network is designed using a mix of digital and analog circuits. Both types must be of a low power design. Low power analog circuits can be designed using a subthreshold analog technique. Details about this will not be covered, as it is expected that the reader have basic knowledge about this topic. For an excellent discussion see [Mead, 1989b].

2.2.1 Digital circuits with static current limitations

There are several approaches that can be made to reduce the current flowing through a transistor operated as a switch. In a complementary CMOS design one can reduce

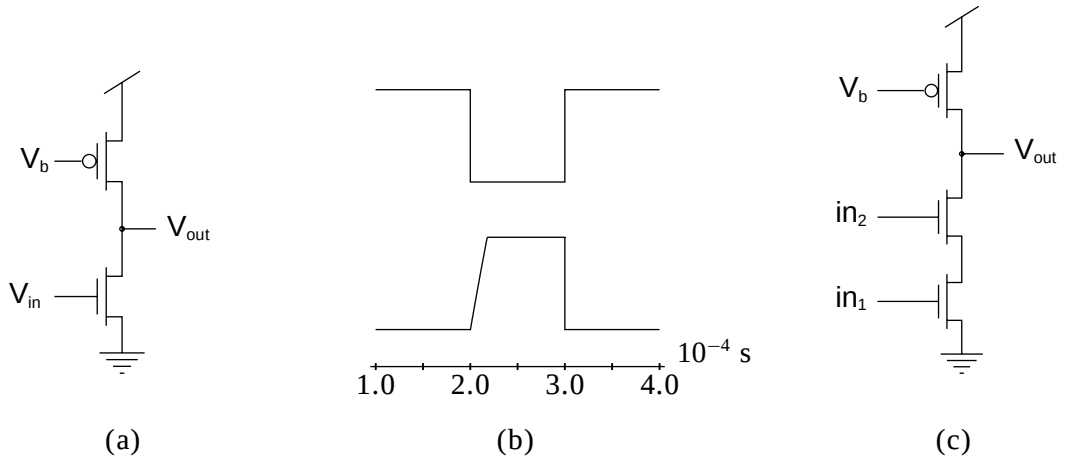


Figure 2.2: Low power digital circuits: (a) inverter gate, (b) its simulated behavior and (c) a nand gate.

the power supply voltage, usually from 5 volt to 3.3 volt, giving a current reduction of the same amount. But this technique does not reduce the heat dissipation enough, and because of smaller tri-state margins, special care is required during the design. Submicron processing and special operating temperatures may also be required [Foty and Nowak, 1994] , which result in higher development and manufacturing costs. For these reasons I chose another design approach.

Another approach is suggested by Lazzaro [Lazzaro, 1992a], which is a design technique very similar to NMOS design. But instead of using a bleeder transistor as a pull-up device, one uses a subthreshold biased p-type transistor. The transistor schematic of a digital inverter gate and NAND gate of this design, are shown in figure 2.2 (a) and (c) respectively. The static current through the transistors is limited by the bias current of the p-type pull up transistor.

Figure 2.2 (b) shows a simulation of the inverter response for a pull-up bias voltage (V_b) of 4.3 volt. As we can see the switching time, that is the raise time, is not fast at all. Faster raise times can be achieved by lowering the bias voltage, but that means higher static currents and power consumption as well. But in highly parallel neural networks the speed of each network component is not critical. The highly parallel processing more than compensates the slow circuits.

CHAPTER

3

Stochastic computing network elements

The contents of this chapter are related to the representation of values as stochastic pulse streams, and the computing elements needed to build a feed-forward neural network with a backpropagation of error learning scheme. Only the computational elements are covered here. The storage of synaptic weights is covered in chapter 5.

There are several reasons to use pulse stream encoding of neural signals instead of pure analog signals. Logic pulse signals are more redundant to noise than analog currents or especially voltages. Our own brains are good examples of this. The signal transmitters are poor, and a lot of noise are added to the signals, but they are easy to reconstruct at required intervals.

Another reason is that digital logic is well developed, and as shown later in this chapter, some arithmetic operations on pulse streams can be done with a minimum of transistors. This is a very important feature, since physically small computational elements makes larger networks and more computational power on a dedicated silicon area possible.

Why not use a common digital approach with binary coding of signals? The main reason is simply that synaptic weighting includes multiplication and storing of weights, and the binary hardware needed for these operations occupy to much silicon area. Representing values in binary form also imply a lot of wiring. Digital serial communication is seldom beneficial for local communication and is therefore not attractive at all in neural networks, where all communication is very local. The result is that it is difficult, if not impossible, to design large systems on one chip.

3.1 Representing values with pulse streams

There are several slightly different methods for representing values as sequences of pulses. [Murray and Tarassenko, 1994] mention six different, were variations of pulse amplitude, pulse width, pulse frequency, pulse density, pulse phase and pulse code modulation (weighted bits) can be used to represent the actual value. All of these encoding techniques can be used in VLSI neural systems, but not all are equally well suited. With pulse amplitude modulation one encounter the same noise problem as with pure analog voltage encoding. The pulse phase modulation technique is more robust to noise. The pulse code modulation (PCM) technique is used both in data transmission systems and 1-bit D/A converters for compact disc players, but it is made for pure digital constructions and has never been used in neural systems.

The last three variants are all very attractive. They all share the same advantage of using time as the information coding axis, and they are therefore only susceptible to edge-jitter noise. Such noise will be less significant in a conventionally noisy and adaptive environment as a neural system. Another aspect that must be taken into consideration is that since the actual value such pulse stream signals represents is measured over time, with normal gaussian distributed noise, the noise will average out and be even less significant.

[Murray and Tarassenko, 1994] use both the pulse frequency and pulse width modulation techniques in neural networks, and [Mead, 1989b], [Lazzaro and Mead, 1989abc] , [Lazzaro, 1991ab,1992ab] , and [Meador *et al.*, 1991] use pulse density modulation, also described as *mean rate encoding*, in very biologically inspired implementations. The network presented in this text employ a mix of pulse frequency, pulse width and pulse density modulation.

3.2 Stochastic pulse streams

In stochastic computation, values are represented as the probability that a logic signal will be *on* or *off* at a given time. Arithmetic operations are performed by virtue of the completely random and uncorrelated nature of the logic signals representing data. The actual probability of such a pulse stream being on, can be found through moving time integration of the pulse stream, and there is no way one can predict the sequence of logic levels on basis of a probability. A given value can be represented by several quite different pulse streams. A quantity represented by a pulse stream signal being on half of the time and else off, can give rise to a lot of quite different pulse streams. One extreme possibility is that the signal line is on for first half of the time and off for the last half. Another possibility is a signal fluctuating nicely at a steady frequency, with equally spaced pulses. The most common types of stochastic pulse stream on the other hand, consists of pulses of totally random width and inter-pulse distances. There is no limitations on the pulse-width or frequency, but the pulse amplitude is constant. Because of this signals of this type often appears like random noise. This is generally regarded as a waste product, but not in the context used here. Such sequences of logic levels where successive levels are statistically independent, and the probability of the logic level being on is a measure of the quantity, are called *Bernoulli sequences*. Figure 3.2 shows a pulse stream signal and its moving probability of being on.

This method of representing information was first presented by von Neumann in

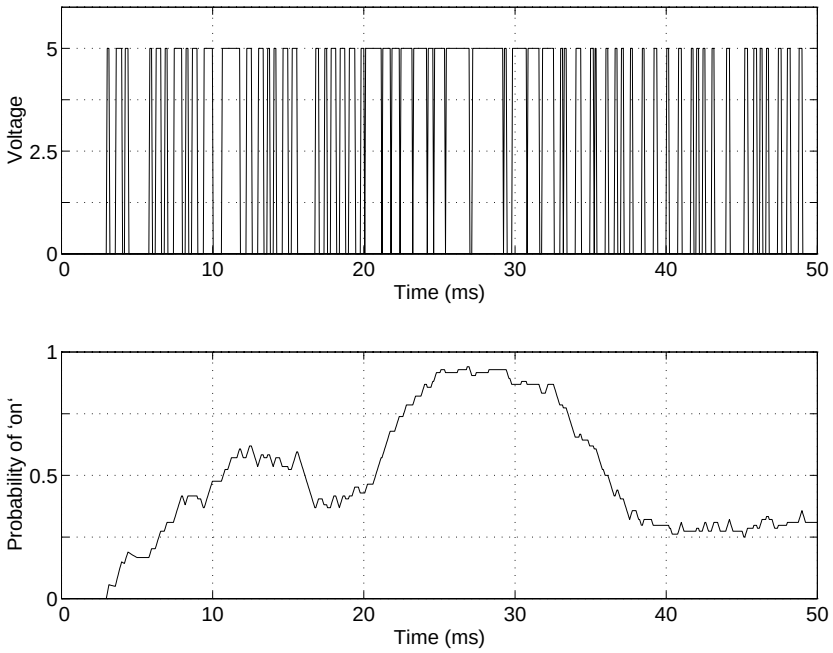


Figure 3.1: Pulse stream signal: The actual pulse stream at the top, and its moving average probability value at the bottom. The moving average is calculated over the last 5*ms*.

his classical paper [von Neumann, 1956]. In this paper he speculated on a statistical principle through which some properties of the nervous system could be explained. His work was not aimed at making any hypothesis about neurophysiological phenomena. His intentions was to show that basically inaccurate representation of information and unreliable processing components, through redundancy, could be made reliable to yield accurate results.

During the last half of the 1960's much work was put into this field, to make conventional computers, specialized hardware to make approximate solutions to partial differential equations and circuitry to mimic parts of the nervous system. [Gaines, 1969] presents some of the results of all this work, and gives an overview of parts of stochastic computation, and its implementation in hardware. Three different ways of representing a value is sketched out, depending on the constraints of the values needed in a given system. Two of these representations are of interest for the neural network implementation presented in this thesis. If a system only uses unipolar values, either positive or negative, the representation mentioned at the start of section 3.2 can be used directly. A quantity E in the range $0 \leq E \leq V$, is represented by the pulse stream signal A with generating probability P, were

$$P = P\,(A = 1) = \frac{E}{V}\,.$$

Maximum quantity is represented by a signal which is always on, and zero quantity by a signal which is always off.

If bipolar values are needed, as is true for most problems, we can differentiate the signal on two lines, one representing the the negative part and the other the positive

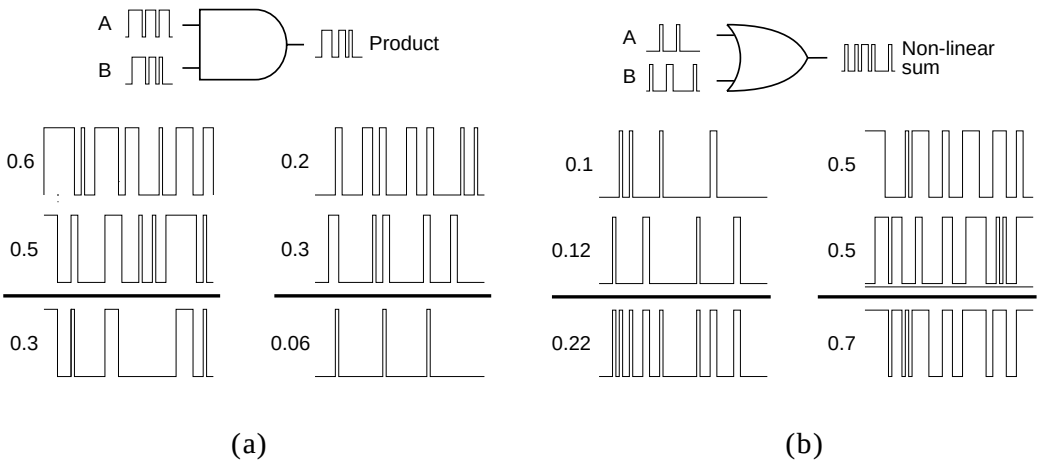


Figure 3.2: Simple logic gate arithmetic: (a) Multiplication performed by an AND-gate. (b) The non-linear summation performed by a wired or-gate. For low levels of activity the summation is linear, but for high levels the summation saturates.

part of the value. The transformation from actual quantity to pulse stream probability for a bipolar quantity E such that $-V \leq E \leq V$, will be described by [Gaines, 1969]:

$$\frac{E}{V} = P\,(positive\ line\ on) - P\,(negative\ line\ on)\ . \tag{3.1}$$

Maximum positive quantity is represented by the positive line always on, and the negative line always off. For maximum negative quantity the negative line is always on, and the positive always off. Zero value is represented by both lines off, or both lines fluctuating with the same probability of being on.

3.3 Arithmetic operations on stochastic signals

The two most common arithmetic computations in neural synaptic elements are multiplication and summation. Necessary hardware to carry out these two operations must therefore be developed.

Under the restriction that two unipolar signals are stochastically uncorrelated, that is their probabilities of being on are statistically independent, the product of the two signals can be computed by a single AND gate [Gaines, 1969]. The output signal's probability of being on is given by the product of the probabilities on the input. This is reasonable since the output only is on if both inputs are on. Figure 3.2 (a) shows such multiplications for two different sets of pulse streams.

For bipolar quantities the positive output should be on if both positive or both negative inputs are on, and otherwise turned off. The negative output should be on if one positive and one negative input are on at the same time. Written in terms of boolean algebra we get [Gaines, 1969]:

$$\begin{aligned} o^+ &= a^+b^+ + a^-b^-\,, \\ o^- &= a^+b^- + a^-b^+\,. \end{aligned} \tag{3.2}$$

Summation of stochastic pulse streams can be done in several ways, depending on the accuracy needed. The simplest way is to use wired-OR summation. Just hook the

signal lines to be summed up to the same node. The summation performed by this scheme is not linear for all types of signals. As the probabilities of being on increases, pulse overlap also increases, and the summation saturates gradually as shown in figure 3.2 (b). For some applications this is not satisfactory, but in neural systems it can be used as a very desirable feature. The generated sums will not add up to values exceeding beyond the defined limits. Overflow will never occur, regardless of the number of signals summed. The upper limit is always represented with a signal always on and the lower limit by a signal always off. There is in theory an infinite number of achievable states between the two limits. In practice the density is as good as for a pure analog signal. There is also another great feature with non-linear summation. If we recall from section 2.1, the output of a neuron is the result of a non-linear transformation of the sum of the weighted inputs through an activation function. With a non-linear summation scheme most of the work with this transformation is already done. For a review of other methods see [Gaines, 1969].

[Tomlinson *et al.*, 1990] express the non-linear sum saturation as

$$1 - e^{-\sum_i p_i} ,$$

where p_i is the probability of signal line i being on. This is a very interesting result, but unfortunately the initial condition used in the derivation makes you doubt if it is true or not. I will therefore give a more precise derivation.

To find the non-linear relationship between the probability of the sum being on, and the probabilities of each input $i = 1, \ldots, n$ being on, we look at the inputs as a series of n independent Bernoulli trials, each having probability a_i of a logic on. Then we use binomial distribution to express the probability of one or more of the inputs being on, and get

$$P\,(1 \textit{ or more inputs on}) = \sum_{k=1}^{n} \binom{n}{k} \left(\frac{\sum_i a_i}{n}\right)^k \left(1 - \frac{\sum_i a_i}{n}\right)^{n-k} ,$$

were k represents the number of inputs that is on. $\frac{\sum_i a_i}{n}$ is the average probability of any input being on.

This is not an entirely pleasant equation to compute, but using the well known statistical limit theorem of Poisson, the problem turns out to be quite simple. By using the fact that $P(Y = 1) = 1 - P(Y = 0)$, we get

$$\begin{aligned} P\,(1 \textit{ or more inputs on}) = P\,(SUM = 1) &= \sum_{k=1}^{\infty} \frac{e^{-\sum_i a_i} \left(\sum_i a_i\right)^k}{k!} \\ &= 1 - \frac{e^{-\sum_i a_i} \left(\sum_i a_i\right)^0}{0!} \\ &= 1 - e^{-\sum_i a_i} . \end{aligned} \tag{3.3}$$

The limit theorem is only an approximation, but it is well accepted in statistical analysis. The agreement between the actual value and the approximation converges as the number of inputs increases.

A plot of equation 3.3 is shown in figure 3.3, and as one can see this function takes the form of the upper half of a sigmoid type function often used as the non-linear activation function for neurons.

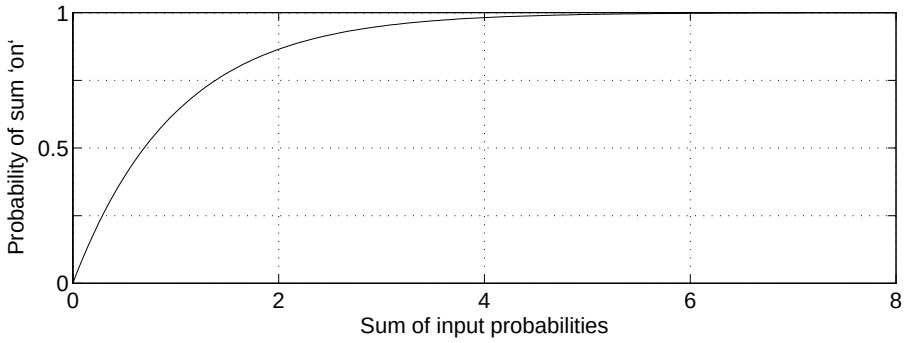


Figure 3.3: Saturation of wired-OR summation of pulse streams.

3.4 Feed-forward network elements

3.4.1 Previous work

Even though both [Gaines, 1969], [Ribeiro, 1967] and other researchers in this field during the late 1960's mention the potential of using stochastic signals and computations in artificial neural networks, the first serious attempt to make a VLSI implementation, that I know about, was done by [Tomlinson and Walker, 1990, Tomlinson *et al.*, 1990] at Neural Semiconductor, Inc. In this implementation the feed-forward part is combinatorial and asynchronous, but the weights are synchronously clocked. The implementation lacked on-chip learning. Networks were trained off-chip, and then the weight values were dumped into a memory on the chip. Later [Eguchi *et al.*, 1991, Dickson *et al.*, 1993] have extended this implementation scheme with on-chip learning.

None of these implementations are implemented in a low-power technique, and they either use fully digital synaptic memories, or analog memories where the stored analog weights are converted to a digital value through an A/D-converter. Both of these implementations tends to consume a lot of silicon area, mostly because of the number of discrete weight levels needed to make the probability of learning convergence sufficiently large. Usually up to 16-bit density is needed.

Both [Eguchi *et al.*, 1991] and [Dickson *et al.*, 1993] report good results on learning capability of their implementation. On the other hand the learning algorithm implemented by [Eguchi *et al.*, 1991] is a very simplified version of backpropagation, and [Dickson *et al.*, 1993] points out that this implementation is insufficient for several real world applications.

For a description of the differences of the implementation presented in this thesis and the referenced implementations, see table 3.1.

3.4.2 Implementing the forward computation

Before we can start the actual design of hardware elements to implement the feed-forward computation in a network, we have to decide what type of signal representation to use for the different parts. The decision is actually rather straight forward.

The quantity of input values fed to the neurons in the input layer, and the output from all neurons, can be represented by a unipolar stochastic pulse stream. These values are not negative. They only moves from a non-firing state, to a firing state. In the notation

	Tomlinson	**Eguchi**	**Dickson**	**This thesis**
Stochastic	Yes	Yes	Yes	Yes
On-chip learning	No	Yes	Yes	Yes
Learning algorithm	—	Very simplified backprop	Backprop	Backprop
On-chip weight storage	Yes	Yes	Yes	Yes
Memory type	Analog with A/D-converters	Static RAM	Up/down counters	Analog
Implementation type	Digital CMOS (Analog weights)	Digital CMOS	Digital CMOS	Low-Power digital/analog CMOS
Other aspects	Clocked weights	Clocked weights	Clocked weights	Fully asynchronous

Table 3.1: Table showing differences between the implementation presented in this thesis and earlier implementations.

used above the input/output values are represented by $o_{pi} = P(O_{pi} = 1)$, where $o_{pi} = 1$ represents a maximum firing output and $o_{pi} = 0$ a non-firing output.

Since synapses can be either excitatory or inhibitory, the weight of an input must be represented by a bipolar quantity. The positive and negative parts of a weight signal is represented by $w_{ji}^{+} = P(W_{ji}^{+} = 1)$ and $w_{ji}^{-} = P(W_{ji}^{-} = 1)$ respectively.

The weighted input must of course also be represented by a bipolar quantity. From equation 2.1 we know that the weighted input is the product of the input and the weight. The multiplication of bipolar signals was presented in equation 3.2, and expressing the weight multiplication in the same way, regarding one of the signals as unipolar, the weighted input becomes

$$\begin{aligned} P\,(positive\ weighted\ input\ on) &= w_{ji}^{+} o_{pi} \\ P\,(negative\ weighted\ input\ on) &= w_{ji}^{-} o_{pi}\,. \end{aligned} \tag{3.4}$$

The inverted weighted input signals can be computed with 3 transistors in a low-power design, excluding the necessary pull-up transistors, as shown in figure 3.4. This is a considerably less than the 3 NAND gates and one inverter used in complementary CMOS implementations [Tomlinson and Walker, 1990]. The circuit in figure 3.4 is the differential pair from analog MOS design. It can be used under the restriction that only one of the differential lines are on at a given time. Transistor Q_2 and Q_3 can *not* both be open at the same time. Just imagine a scenario where Q_1 is closed, and Q_2 and Q_3 are open. Then the two net-input lines would be shortened, and contain the same signal value. But as we shall see later, it is not difficult to implement circuitry that generate weight signals fulfilling this restriction.

The summation of the weighted net-inputs are performed by wired-OR summation. In figure 3.4 the two lines representing the inverse positive and inverse negative net-inputs are marked $1 - net_{pj}^{+}$ and $1 - net_{pj}^{-}$ respectively. The total quantity of net-input can

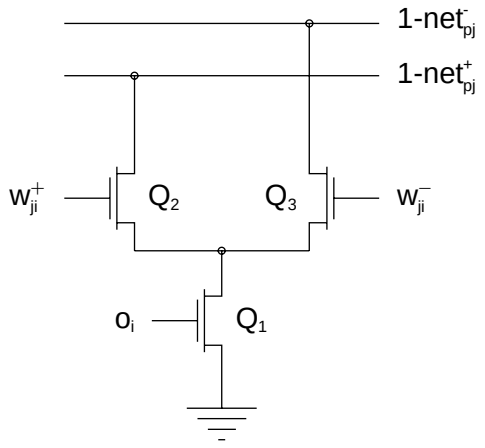


Figure 3.4: Three transistor weight computation circuit.

be described with the two equations, [Tomlinson *et al.*, 1990]:

$$\begin{aligned} net^+_{pj} &= \sum_i w^+_{ji} o_{pi} = 1 - \prod_i \left(1 - w^+_{ji} o_{pi}\right) \\ net^-_{pj} &= \sum_i w^-_{ji} o_{pi} = 1 - \prod_i \left(1 - w^-_{ji} o_{pi}\right) . \end{aligned} \tag{3.5}$$

The positive and the negative net-input incorporates the non-linear saturation described in equation 3.3. This fact can be taken advantage of in the generation of the non-linear activation function performed by the neuron. Equation 3.1 represents the transformation from differential bipolar representation to the actual quantity. The negative quantity is subtracted from the positive quantity. Both quantities are the result of a non-linear sum, and have the characteristic shown in figure 3.3, but of opposite signs. The subtraction of two such characteristics, gives a result that takes the form of a complete sigmoid function, which is the most common function to use as an activation function in neurons. The subtraction can be looked up on as a merging of two characteristics as shown in figure 3.3, where one characteristic is a mirror of the other characteristic. Due to this the neuron do not need to take any further action to implement the activation function. Only a transformation of the bipolar quantity to a unipolar one is necessary.

The transformation to unipolar representation can be obtained as follows. The output should obviously be off if both net-inputs are off, or if the positive net-input is off and the negative net-input is on. The output should be on if the positive net-input is on and the negative off. But what about the last case, where both the positive and negative net-input are on at the same time? [Tomlinson *et al.*, 1990] suggests a simplification, and letting the output be off for this case. This leads to the quite simple expression

$$o_{pj} = net^+_{pj}\left(1 - net^-_{pj}\right) = 1 - \left(\left(1 - net^+_{pj}\right) + net^-_{pj}\right) \tag{3.6}$$

for the output. As we can see, the negative net-input is contributing more to the output than the positive net-input do. One inhibitory input on will set the output off, even if all excitatory inputs are on. This also indicates that the activation function will vary, depending on the number of inputs to a neuron. As the number of inputs increases the steepness of the transfer function will decrease. One inhibitory weight will keep the probability of the output *on* low.

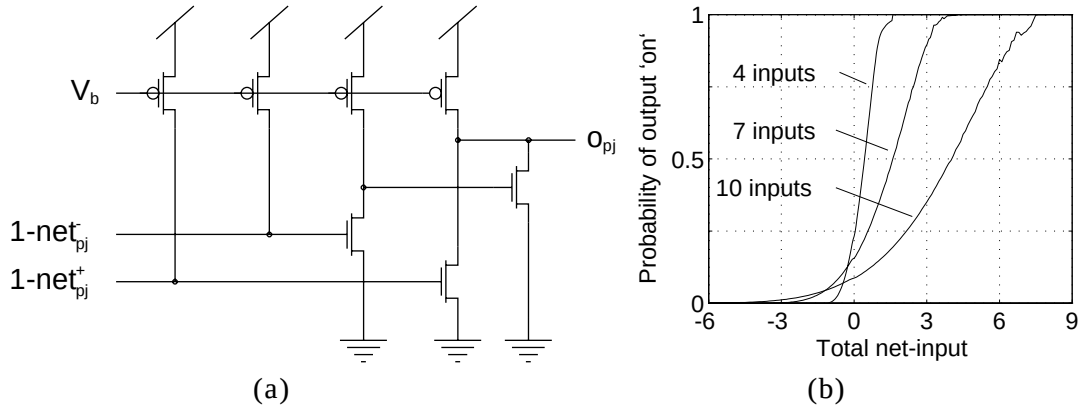


Figure 3.5: Circuitry to perform the neuronal computation and the total transfer function: (a) The neuron circuit, which incorporates the pull-up transistors for the net-inputs. (b) The transfer-function for neurons with three different number of inputs.

Based on the inverted net-input signals generated by the weighting circuit, the neuron circuit become as shown in figure 3.5 (a), a circuit which incorporates the pull-up transistors for the net-input signal lines. This circuit is of approximately the same size as the inverter and AND gate circuit used in complementary digital implementations.

To make a plot of the activation function on basis of equation 3.3 and 3.6 is beyond my knowledge in statistics. A plot based on empirical data generated from equation 3.4, 3.5 and 3.6 were made instead. Randomly uniform distributed probabilities were assigned to both inputs and weights, and curves were generated by averaging the results for 1 million iterations in the case of 4 and 7 inputs, and 5 million iterations for the 10 input neuron. The resulting curves are plotted in figure 3.5 (b), and we can see that the theoretical observation done previously is correct. There is a variation in the transfer function for different number of synaptic inputs.

3.5 Backpropagation of error with stochastic pulse streams

3.5.1 Weight change implementation

The necessary hardware to do the forward computation in a feed-forward network is now discussed. The next step is to make an implementation of the learning algorithm presented in section 2.1.4. [Tomlinson *et al.*, 1990] presents an implementation of the backpropagation of error algorithm, but this implementation only handles the error of an output of a neuron as a unipolar value. No results from systems using this implementation is presented, and I expect that such a representation will be insufficient for an effective learning to take place.

The implementations used by [Dickson *et al.*, 1993] and [Eguchi *et al.*, 1991] both use bipolar values to represent the error. Both proclaim good results, but the most attractive is the implementation presented by [Dickson *et al.*, 1993], since this is a complete implementation of the backpropagation algorithm. A problem though is the lack

of formal derivation of the special equations for the weight change and error generation. There are also inconsistencies between theoretical equations, and actual implementation. I will therefore give a full derivation of the necessary equations to implement the backpropagation of error learning algorithm with stochastic computing elements. The equations derived here will be aimed at implementations were all calculations involved in the backpropagation part are performed in the synapses, which ensures local computations and a minimum of wiring. For the implementation presented here, a minimization of wiring, at the extra cost of a few transistors for each synapse, results in the most compact networks. The appearance of the equations are therefore different than how [Dickson *et al.*, 1993] presents them, but they are functionally equal.

To implement the backpropagation of error algorithm, we use the same approach as with the feed-forward part. We start with the derivation of the necessary boolean/probability equations that represents the algorithm, and then make straightforward circuit implementations. If we recall from equation 2.3 and 2.2 the weight change is expressed as

$$\Delta_p w_{ji} = -\eta \frac{\partial E_p}{\partial w_{ji}}, \tag{3.7}$$

where

$$E_p = \frac{1}{2} \sum_j \left(t_{pj} - o_{pj}\right) .$$

Since the weight signal is split into two signals, one representing the positive weight and one the negative weight, equation 3.7 must be split into two equations, one representing the change of the weight when the positive signal line is active, and one the change when the negative signal line is active. These two equations, expanded using the chain rule, yields

$$\Delta_p w_{ji}^{+} = -\eta \frac{\partial E_p}{\partial w_{ji}^{+}} = -\eta \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial net_{pj}^{+}} \frac{\partial net_{pj}^{+}}{\partial w_{ji}^{+}} \tag{3.8}$$

$$\Delta_p w_{ji}^{-} = -\eta \frac{\partial E_p}{\partial w_{ji}^{-}} = -\eta \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial net_{pj}^{-}} \frac{\partial net_{pj}^{-}}{\partial w_{ji}^{-}} . \tag{3.9}$$

In equation 3.8 and 3.9 the part $\frac{\partial E_p}{\partial o_{pj}}$ represents the error of the output of a neuron, and its value is assumed to be bipolar. We define the total error value to be represented by

$$\varepsilon_{pj}^{+} \equiv -\frac{\partial E_p^{+}}{\partial o_{pj}} \quad ; \quad \varepsilon_{pj}^{-} \equiv -\frac{\partial E_p^{-}}{\partial o_{pj}} . \tag{3.10}$$

With this representation it should be obvious that a weight should be changed in positive direction when ε_{pj}^{+} is on, and in negative direction when ε_{pj}^{-} is on. Substituting equation 3.10 into equation 3.8 and 3.9, the total positive weight change is modeled by

$$\begin{aligned}
\Delta_p^{+} w_{ji} &= \Delta_p^{+} w_{ji}^{+} + \Delta_p^{+} w_{ji}^{-} \\
&= \eta \left(\varepsilon_{pj}^{+} \frac{\partial o_{pj}}{\partial net_{pj}^{+}} \frac{\partial net_{pj}^{+}}{\partial w_{ji}^{+}} S_{w_{ji}} + \varepsilon_{pj}^{+} \frac{\partial o_{pj}}{\partial net_{pj}^{-}} \frac{\partial net_{pj}^{-}}{\partial w_{ji}^{-}} \left(1 - S_{w_{ji}}\right) \right) \\
&= \eta \varepsilon_{pj}^{+} o_{pi} \left(\left(1 - net_{pj}^{-}\right) \frac{1 - net_{pj}^{+}}{1 - w_{ji}^{+} o_{pi}} S_{w_{ji}} + net_{pj}^{+} \frac{1 - net_{pj}^{-}}{1 - w_{ji}^{-} o_{pi}} \left(1 - S_{w_{ji}}\right) \right) ,
\end{aligned} \tag{3.11}$$

where $S_{w_{ji}}$ represents the sign of the weight, and is on for a positive sign or else off. The sign must be incorporated since the $\Delta_p^+ w_{ji}^+$ and $\Delta_p^+ w_{ji}^-$ should only make contributions when the weight is positive and negative respectively.

As we can see the weight change equation incorporates a division. Implementing division between two stochastic asynchronous pulse streams is not easy at all. [Gaines, 1969] mention a space and time consuming method for synchronous signals, but this method is highly dependent on clocked signals, and can not be converted to asynchronous signals. [Dickson *et al.*, 1993] ignores the divisor with the argument that it do not affect the sign of the quantities nor greatly their magnitude. This is an interesting approach, but since it is not obvious that the effect on the magnitudes are small, further investigation of the introduced error is needed.

Rewriting the division part, using equation 3.5, yields

$$\frac{1 - net_{pj}}{1 - w_{ji} o_{pi}} = \frac{\prod_i \left(1 - w_{ji} o_{pi}\right)}{1 - w_{ji} o_{pi}} .$$

We can see that the result of the division is the sum of the weighted inputs from all the synapses, *except* for the one the weight change is calculated. The magnitude of the weighted input from one synapse will, in most cases, be small compared to the total magnitude of the net-input. The error accumulated by ignoring the divisor is therefore assumed to be small. Only in cases where one particular synapse is contributing most of the total input, the error will be significant and a potential problem with learning may occur.

Imagine a situation where the convergence of learning depends upon one synapse controlling most of the net-input. In such a situation the learning performance will be poor and the learning speed is assumed to decrease significantly. A reasonable and simple solution to this problem is to make extra copies of this particular synapse. The generated error for each of the synapses is then reduced to an acceptable magnitude, even though the total accumulated error will be the same. Extra controlling dynamics introduced with these synapses should also increase the networks capability of learning a specific problem.

In the generalized case, adding extra synapses to a neuron, means adding extra neurons to the previous layer. The conclusion is therefore that a network with enough hidden neurons should have a fair chance to solve a problem. The final conclusion must be that for large networks no problems should occur when ignoring the divisor.

On the other hand the $\left(1 - net_{pj}\right) / \left(1 - w_{ji}\right)$ can be computed by summing up the weighted inputs for all the synapses to the neuron, except the one to be changed. For each synapse we will need $n - 1$ (where n is the number of synapses to the neuron) extra weighting circuits of the type shown in figure 3.4. For each neuron this will give a total number of extra weighting circuits equal to $n(n-1)$. So the number of extra transistors is quadratically increasing in the number of synapses.

We also need extra wiring lines. For each synapse we need two extra wires, one for the positive and one for the negative part, to all the other synapses. It sums up to a total of $2n$ extra wires for each neuron.

It should be clear that this method is unacceptable for neurons with more than about 5-6 synapses. The extra area overhead will be to large.

If we are ignoring the divisor in equation 3.11, the positive weight change is modeled by

$$\Delta_p^+ w_{ji} \approx \eta \varepsilon_{pj}^+ o_{pi} \left(1 - net_{pj}^-\right) \left(\left(1 - net_{pj}^+\right) S_{w_{ji}} + net_{pj}^+ \left(1 - S_{w_{ji}}\right)\right) . \tag{3.12}$$

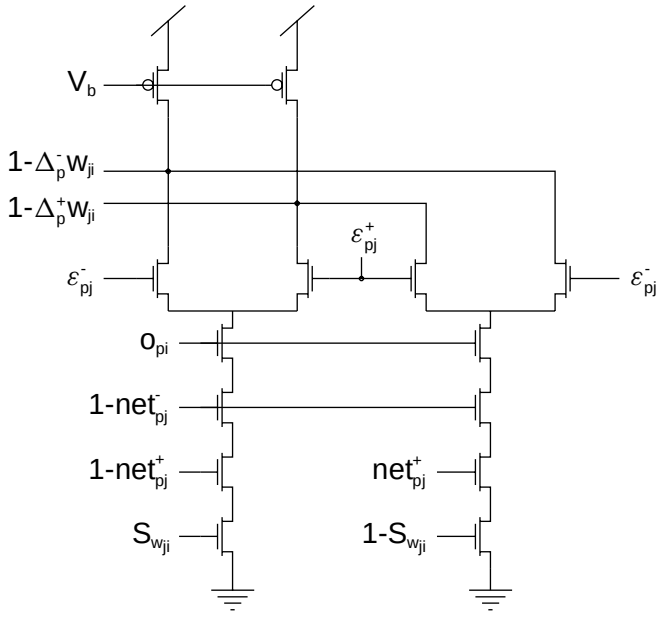


Figure 3.6: Weight-change circuit.

Using the same approach the negative weight change is modeled by

$$\Delta_p^- w_{ji} \approx \eta \varepsilon_{pj}^- o_{pi} \left(1 - net_{pj}^-\right) \left(\left(1 - net_{pj}^+\right) S_{w_{ji}} + net_{pj}^+ \left(1 - S_{w_{ji}}\right)\right) \ . \tag{3.13}$$

With this model a weight should be changed in positive direction when $\Delta_p^+ w_{ji} > \Delta_p^- w_{ji}$, and in negative direction when $\Delta_p^- w_{ji} > \Delta_p^+ w_{ji}$. The signal $\Delta_p^+ w_{ji}$ is larger than $\Delta_p^- w_{ji}$ when $\Delta_p^+ w_{ji}$ is on and $\Delta_p^- w_{ji}$ is off. The duration of the state decides the amount of change.

In these equations the learning rate is explicitly given as a pulse stream η. Its purpose is to reduce the amount the weight is changed for each pattern presentation. To much change will result in an divergent oscillation of the error along one of the axis in error space [Hertz *et al.*, 1991]. A small change results in very slow learning. The size of the learning rate giving the fastest programming is dependent of the exact form of the error space, which of course is dependent on the network topology and the problem presented. It will therefore be an advantage if the learning rate can be controlled.

The fact that the learning rate pulse stream must be supplied from off-chip, and distributed to all the synapses, which of course requires extra wiring and the dedication of one extra pad for the signal, calls for the investigation of other easier ways to implement the learning rate. In the implementation presented in this thesis the learning rate is controlled by the memory it self. Physical factors that determine the programming speed of the synaptic memory are used to adjust the learning rate. The explicit representation of the learning rate as the η part of equations 3.12 and 3.13, can therefore be removed.

Figure 3.6 shows a 14-transistor circuit implementing the inverted backpropagation weight change signals. As we can see there are five transistors in series in the pull-down chain. In most digital implementations such long series of transistors are avoided because of the extra switching delay they introduce. But considering that the pull-up devices in this circuit are subthreshold biased p-transistors, with switching delays normally in the range of micro seconds, the extra delay introduced by long transistor series can be neglected.

This implementation gives a fairly simple interface to the memory. It takes two inverted stochastic pulse streams as inputs, representing the change of the stored value. As output it gives two other stochastic pulse streams representing the stored value, and an extra signal line giving the sign of this value explicitly. One extra inverter is needed to generate the inverted sign.

3.5.2 Error propagation implementation

The implementation of the error generation and propagation circuitry is very similar to the implementation of the weight change circuit. But if we recall from section 2.1 the generation of the error signal is different for neurons in the output layer and neurons in hidden layers. Two different circuits must therefore be implemented. Let us start with the hidden layer circuit.

Hidden layer error propagation

The error for the output of a neuron was defined in equation 3.10. From equation 2.5 we have that the error to be generated by a hidden synapse is given by the weighted error from the subsequent layer scaled by the derivative of the output of the neuron the synapse is contributing input to. The total error to be propagated to a neuron in the previous layer is the sum of the generated errors from all the synapses which have the neurons output as their input.

Putting this into equations we get that the positive error part can be modeled by

$$\begin{aligned}
\varepsilon_{pj}^{+} &= -\frac{\partial E_p^{+}}{\partial o_{pj}} \\
&= \sum_k \left(-\frac{\partial E_p^{+}}{\partial o_{pk}} \frac{\partial o_{pk}}{\partial net_{pk}^{+}} \frac{\partial net_{pk}^{+}}{\partial o_{pj}} - \frac{\partial E_p^{-}}{\partial o_{pk}} \frac{\partial o_{pk}}{\partial net_{pk}^{-}} \frac{\partial net_{pk}^{-}}{\partial o_{pj}} \right) \\
&= \sum_k \left(\varepsilon_{pk}^{+} \left(1 - net_{pk}^{-}\right) w_{kj}^{+} \frac{1 - net_{pk}^{+}}{1 - w_{kj}^{+} o_{pj}} + \varepsilon_{pk}^{-} net_{pk}^{+} w_{kj}^{-} \frac{1 - net_{pk}^{+}}{1 - w_{kj}^{-} o_{pj}} \right) \\
&\approx \sum_k \left(\varepsilon_{pk}^{+} \left(1 - net_{pk}^{-}\right) w_{kj}^{+} \left(1 - net_{pk}^{+}\right) + \varepsilon_{pk}^{-} net_{pk}^{+} w_{kj}^{-} \left(1 - net_{pk}^{-}\right) \right) . \qquad (3.14)
\end{aligned}$$

The divisor is ignored in the same way as was done in the weight change model. When deriving equation 3.14 one must be very careful with the sign of the different equation parts. We must remember to get a positive error contribution when the error from the subsequent layer and the weight is positive. With this arrangement the negative error part is expressed by

$$\varepsilon_{pj}^{-} \approx \sum_k \left(\varepsilon_{pk}^{+} net_{pk}^{+} w_{kj}^{-} \left(1 - net_{pk}^{-}\right) + \varepsilon_{pk}^{-} \left(1 - net_{pk}^{-}\right) w_{kj}^{+} \left(1 - net_{pk}^{+}\right) \right) . \qquad (3.15)$$

An implementation of this model is shown in figure 3.7. The circuit generates the inverted signals. The summation of the signals are performed by wired-OR summation. The sum is then inverted, to get the correct signal.

The wired-OR summation is, as we know, not linear. It introduces an error in the error signal. This error is only significant for large error signals. When the error becomes small, ie. the network is about to converge to the right solution, it becomes accurate, and the fine tuning of the weights should be successful. The introduced error could even be regarded as added noise, and [von Lehmen *et al.*, 1988] and

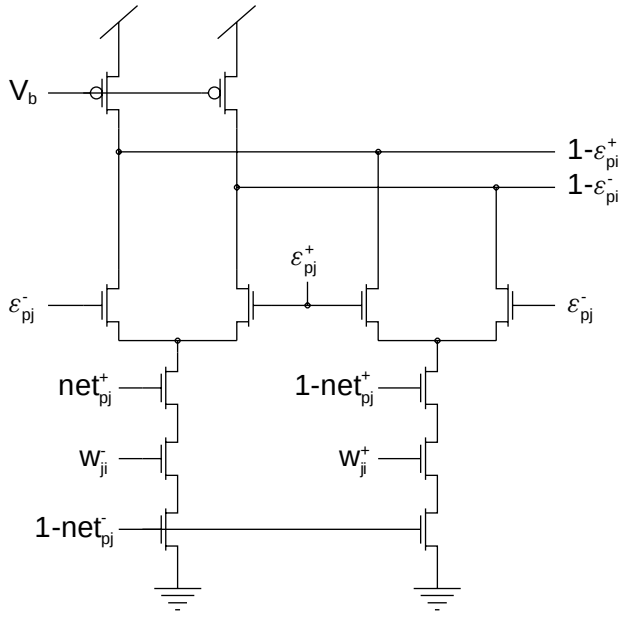


Figure 3.7: Error propagation circuit

[Murray and Tarassenko, 1994] reports that noise can make the backpropagation algorithm more efficient.

Output layer error generation

The error of an output neuron is the difference of the output and target value of that neuron. With a differentiated signal the error can be represented by

$$\begin{aligned} \varepsilon_{pj}^{+} &= \frac{\partial E_p^{+}}{\partial o_{pj}} = \begin{cases} t_{pj} - o_{pj} & ;\ t_{pj} \geq o_{pj} \\ 0 & ;\ \text{otherwise} \end{cases} , \\ \varepsilon_{pj}^{-} &= \frac{\partial E_p^{-}}{\partial o_{pj}} = \begin{cases} o_{pj} - t_{pj} & ;\ o_{pj} > t_{pj} \\ 0 & ;\ \text{otherwise} \end{cases} . \end{aligned} \tag{3.16}$$

It should now be clear that $t_{pj} \geq o_{pj}$ when t_{pj} is on and o_{pj} is off, and vice versa. The error could therefore be modeled by

$$\varepsilon_{pj}^{+} = t_{pj}\left(1 - o_{pj}\right) \quad ; \quad \varepsilon_{pj}^{-} = \left(1 - t_{pj}\right) o_{pj} \ .$$

The error of a output neuron could be generated by the simple circuitry shown in figure 3.8. But there is a catch. In the introduction to stochastic computation it was strictly pointed out that signals involved in the arithmetic operations had to be stochastically independent. With the implementation of the output error generation shown in figure 3.8, the error signal will be correlated with the output signal of the neuron, and then of course with the input, net-input and weight signals of the neuron, resulting in invalid calculation of the weight change and the error propagation signals.

[Dickson *et al.*, 1993] has done some research on this problem, and found that the simple implementation in figure 3.8 is sufficient for small networks with only a few neurons and synapses. But for larger systems the correlation effect becomes to large. The networks are unable to minimize the error and therefore learn a set of input-patterns

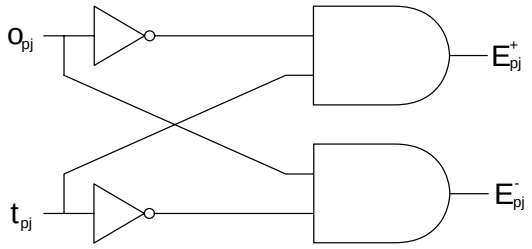


Figure 3.8: Simple output error generation circuit. Unfortunately it is useless due to the correlation of the error signals with the output signal.

correctly. So other solutions must be found, and the next chapter cover one alternative solution.

CHAPTER

4

Output error generation

To generate the output error we need a circuit capable of transforming the difference of two unipolar stochastic pulse streams into a bipolar stochastic pulse stream that is uncorrelated with the two input pulse streams. Several schemes can be used to solve the problem. Two methods springs to my mind. Either make the error signals with the simple circuit presented in figure 3.8, and then delay them sufficiently to make them uncorrelated with the output, or make a moving average of both the output and target signal, then compare the two integrated values and generate a bipolar stochastic pulse stream that expresses the magnitude of the difference.

The error delay approach

The error delay approach is the preferred approach to use in a clocked network. The error signals can be fed into two shift registers of sufficient size – 1 bit should actually be enough since individual pulse occurrences at different clock periods are statistically independent, [Ribeiro, 1967].

The fact that the output of a neuron is an asynchronous pulse stream introduces some problems when using this scheme. The time a signal has to be delayed depends on the average width of the pulses. The shortest average width of the pulses is dependent on the switching delay of the circuit elements. But the use of wider pulses can be decided statically through the design or it can be a user adjustable parameter. In both cases the average pulse width can vary over several orders of magnitude. So there must be possible to control the delay time.

One could think of using something like the follower delay line presented by [Mead, 1989b]. The basic problem with this circuit is its built in low-pass filter effect. It is a trade off between delay and cut off frequency. Larger delay results in lower cut off frequency. The result is that it is not suited for the application of delaying the error signals. The narrow error signals would have been filtered out, and much of the error signal lost.

A similar solution as used in a clocked system can be implemented. The input signal can be sampled and fed to a shift register, where the individual samples are delayed, but

the register must be capable of holding quite a large amount of samples to be able to generate a sufficiently large time delay. For this reason the silicon area needed to implement the circuit will be large, compared to the operation performed. Such a technique may be very attractive in a clocked system, but not at all useful in an asynchronous one.

4.1 Integration and regeneration circuit

The integration and regeneration approach is the one used in this thesis. The circuit must first perform a *time integration* of both the output and the target pulse streams.[1] The integration must be of a type where the contribution of the input to the present output decreases with time into the past. This makes the output a moving average of the input. The two integrated signals must then be compared, and the magnitude of the difference transformed to a bipolar pulse stream.

4.1.1 Moving average generation

In a electronic context integration is implemented as charging and discharging of a capacitor over time. The time it takes to charge the integration capacitor from a *zero* value to maximum value is usually referred to as the integrators time constant. The time constant can be looked upon as a scaling factor of the integration. The larger a time constant, the more the integration value is scaled down. By controlling the amount of charge a given input is supplying to the capacitor, the time constant can be varied. The possibility of controlling the time constant can be critical to make a circuit that is functional for inputs over a wide range of frequencies, pulse widths and/or rise/fall times.

To make a moving average, an integration circuit must also have the feature of leaking charge from the integration capacitor, which will make the output more dependent of the most resent input than of older input. The amount of leakage current depends on how fast the integrator is supposed to *forget* old input.

A circuit that implements these aspects is shown in figure 4.1 (a). For every pulse that is applied to the input V_{in}, a current I_{in} is fed through the current mirror. This current will charge the capacitance on the node V_{int}. The leakage of charge from the integration capacitance is contributed through the diode-coupled transistors Q_1, Q_2 and Q_3. To give the integration node a proper region of operation, both DC and AC, three diodes are cascaded. The bias voltage V_i controls the current I_i and therefore the time constant of the integration.

But even though the time constant can be set with the V_i bias, the characteristic of the integrator is just as much controlled by the integration capacitance, and conductance of the cascaded diodes. Averaging several wide pulses implies a long time constant, which results in a small I_i and a small AC operational region on the integration node. For narrow pulses the time constant must be shorter, i.e. the I_i must be greater, which leads to a greater AC swing on the V_{int} node, and faster discharging of the integration capacitance when the input is off. The result can be ripple on the integration node. To avoid to much ripple the integration capacitance and the W/L-ratio of transistor Q_1 must be scaled properly. The average width of the input pulses must therefore to

[1] In a strict sense it is not necessary to integrate the target signal. A DC voltage representing the integrated voltage of the target signal can be applied directly.

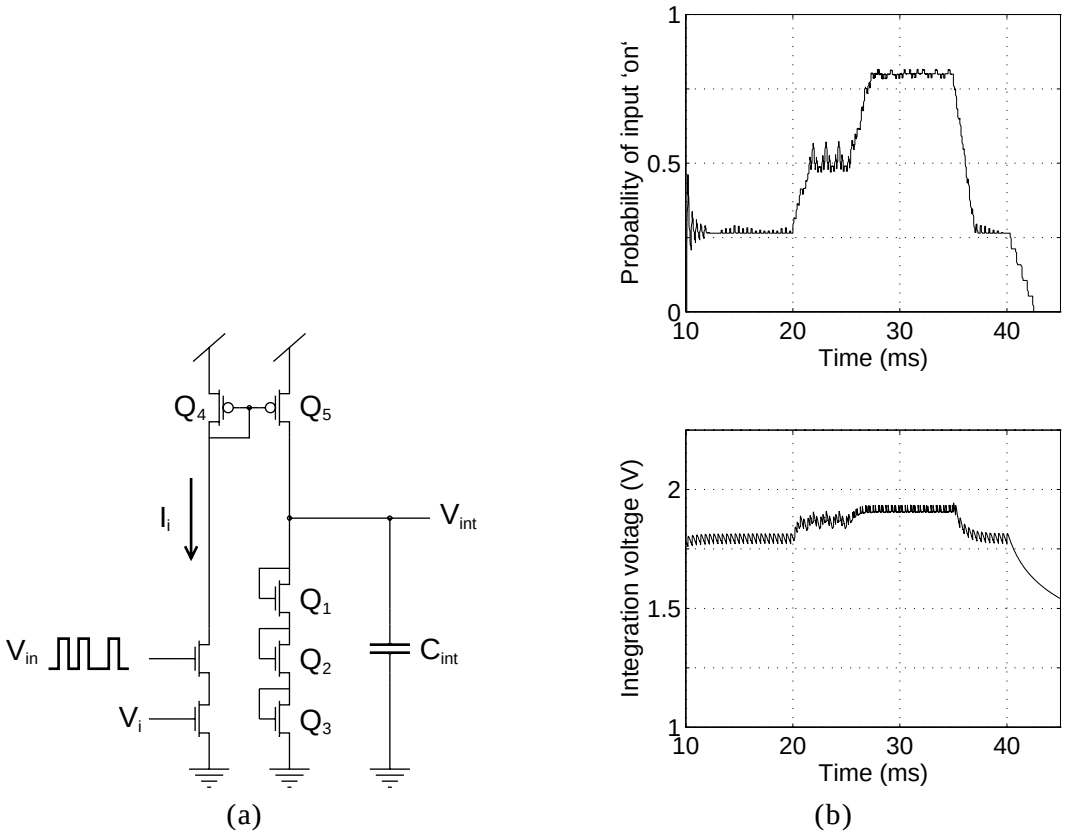


Figure 4.1: Pulse stream integration circuit: (a) The actual implementation, and (b) simulated time integration. Upper plot is a theoretical calculated moving average of the input pulse stream and lower shows the integration voltage. For the simulation the W/L ration of Q_1 was approximately 0.95, and the total integration capacitance $295\,fF$. V_i was set to $0.45V$.

some degree be decided at implementation time, so the discharge diode and integration capacitance can be scaled to fit the time constant and acceptable ripple.

Figure 4.1 (b) show a simulated characteristic of the integration circuit and a theoretical calculated moving average of the input for comparison. For the simulation the W/L-ratio of Q_1 was approximately 0.95, the same as in the actual implementation, and the whole integration capacitance was formed by the gate capacitance of transistor Q_1, which was approximately $295\,fF$.

The circuit in figure 4.1 (a) has both advantages and disadvantages. On the positive side it is small, it has few bias voltages and it suits its purpose in the application. Disadvantages are its highly non-linear integration and the fact that its region of operation must be defined statically through the integration capacitance and W/L-ratio of Q_1.

4.1.2 Comparator circuit

The moving average of the output and target pulse stream is to be transformed to a two signal line bipolar stochastic pulse stream. One strategy is to make a comparator that generates a rectified signal representing the absolute value of the difference between the

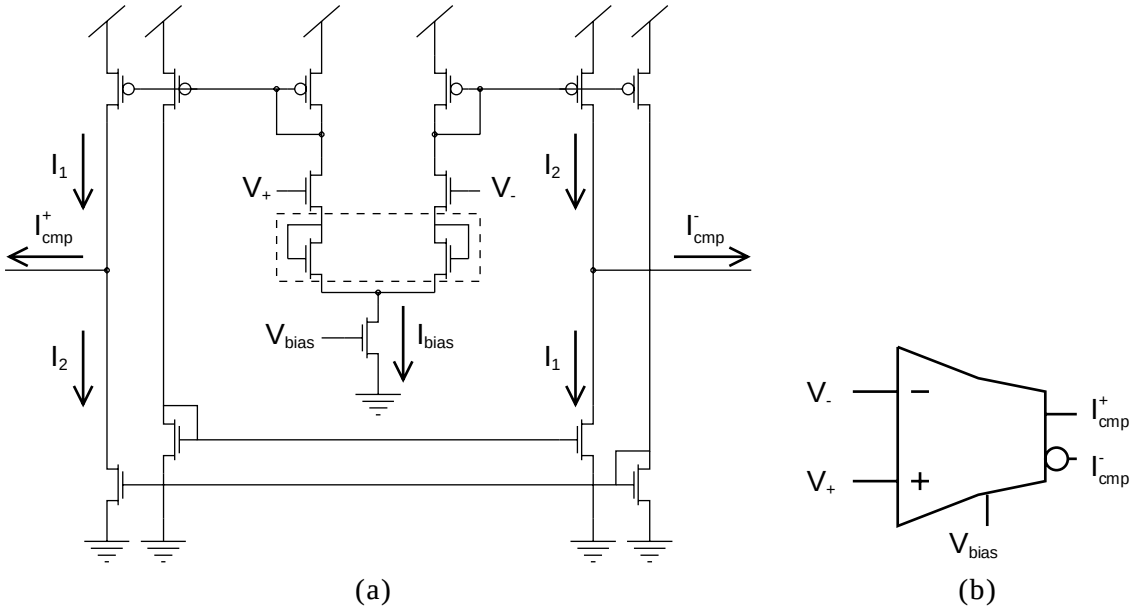


Figure 4.2: Comparator: (a) A wide-range OTA with an extra inverting output stage. The diodes shown in the dashed box can be added to extend the linear region. (b) The symbol representing the circuit.

target and output, and to let this signal control the pulse stream generating circuitry. The comparator must also generate a signal that can be used to select the line which the pulse stream is to be sent out on. Another implementation is to let the comparator circuitry generate both a positive and a negative output, and then let the outputs control a pulse stream generation circuit for the positive and negative error part respectively. Both strategies are equally good. But the latter one surprisingly enough turns out to occupy less silicon area than the first, even though it incorporates two pulse stream generators. The reason is the extra space occupied by the rectifier circuitry.

In the implementation presented in this thesis the second implementation have been chosen. A circuit that generates an output that is dependent on the difference of two inputs, is an operational amplifier. In low power analog VLSI design, a common amplifier is the Operational Transconductance Amplifier (OTA) presented by [Mead, 1989b]. A wide-range OTA with an extra inverting output stage, as shown in figure 4.2 (a) and (b), is just what we need. For a bias voltage (V_{bias}) that make the bias transistor operate in the weak inversion regime, the output current is given by the equation

$$I^{+}_{cmp} = I_1 - I_2 = I_{bias} \tanh \frac{\kappa \left(V^{+} - V^{-}\right)}{2U_T} \ ,$$

where U_T is the thermal voltage given in units of $\frac{q}{kT}$. This value approximates to $25.6mV$ at room temperature.

A plot of measured DC-responses for both the positive and negative output is shown in figure 4.3. As we can see from the plot this amplifier has a narrow linear region of operation. To extend the linear region, extra diodes can be introduced in the differential pair as indicated in the transistor diagram. These diodes will introduce source degeneration in the differential pair transistors, and it gives a proper region of operation for

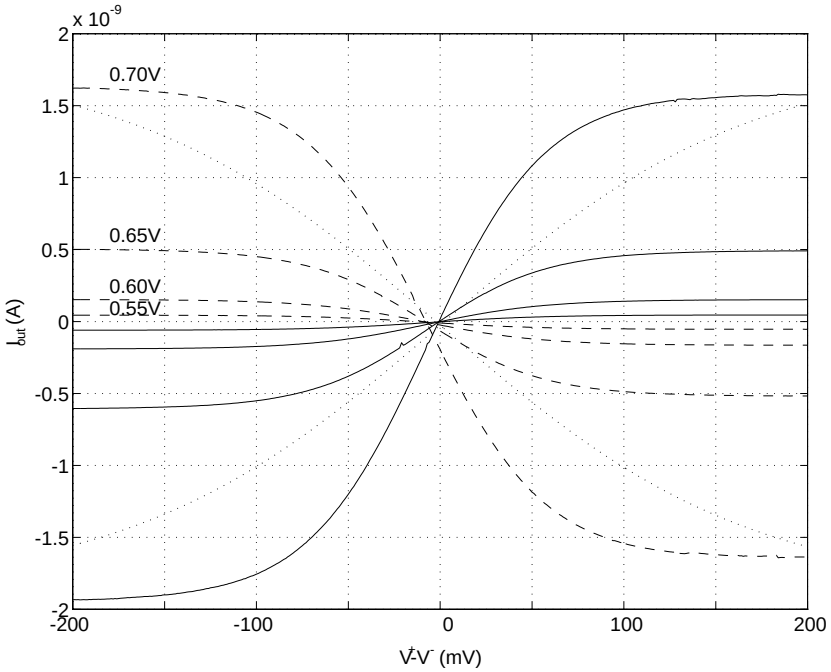


Figure 4.3: Comparator characteristic: Measured response of the positive (solid lines) and negative (dashed lines) output of the comparator, for different bias-voltages. The dotted lines show a simulated response for a comparator with extended linear region.

the differential inputs from 1.6–4.9 volt for $V_{bias} = 0.6V$, $\kappa = 0.7$ and $V_{ds\,sat} = 100mV$ [Watts *et al.*, 1992]. The result is shown for a simulated response in the same plot as the measured response for a traditional OTA.

Theoretically all equally sized transistors of the same type have an equal drain-source current for the same gate-source and drain source potentials. But the real world is different. Effects as variations in doping densities, lithographic variations, edge effects and striation[2] effects results in differences in drain-source currents of typically ± 20 percent for physically adjacent and equally sized small transistors. For the comparator these effects results in offsets in the DC characteristic. As we can see from figure 4.3 the negative output slope is offset approximately $5mV$ for this particular circuit, as the positive output offset is close to zero. This is a bit less than $25mV$ which is a typical offset for such transconductance amplifiers [Mead, 1989b].

Further, the transistor mismatch results in a difference in positive and negative saturation current. For the positive output the negative saturation asymptote is about 15% larger than for the positive asymptote. Transistor mismatch effects can be reduced by increasing the physical size of the devices. Larger transistors result in smaller relative variations. More on this topic, including an excellent visualization of transistor mismatches, can be found in [Andreou *et al.*, 1991].

Another effect that have influence on the characteristic of the OTA-circuit, and other analog circuits, is the *Early-effect* or *drain conductance*. The Early-effect is the result of increased depletion regions surrounding the drain and source terminals, for increased terminal potential (relative to the substrate or well potential). Increased depletion re-

[2]This is a rather peculiar effect. See [Andreou *et al.*, 1991] for further details.

gions imply shorter effective channel lengths, and therefore larger drain-source currents. To reduce the Early-effect the physical length of the transistor can be increased to reduce the effective channel length reduction.

4.1.3 Pulse stream generator

The output of the comparator is current. The error pulse streams should have a probability of being on equal to the magnitude of the current from the respective output of the comparator circuit. From section 3.2 we know that the actual value of a bipolar stochastic pulse stream is the difference of the *on* probabilities of the two signal lines, which implies that only one signal line need to be *active* at a time, ie. only one line pulses, the other one is off. Which line depends on the sign of the error. In the context of the implementation of the pulse stream generator, it should output a pulse stream equal to the magnitude of a positive current flowing into it.

The pulse stream generated should preferably be random, to ensure as little 'correlation' between different pulse streams as possible. But as long as it is not directly correlated with any of the other pulse streams it is supposed to interact with, it will suffice. A current controlled pulse generator is therefore all that is needed. The output is only dependent on the input current, and although this current is dependent of the output pulse stream of the neuron, it should be obvious that the output and error pulse streams are uncorrelated.

I have chosen to use the pulse generator presented by [Lazzaro, 1992a] shown in figure 4.4 (a) in my implementation. This is a low-power version of a similar circuit presented by [Mead, 1989b].

If we assume that V_{out} is off, ie. transistor Q_1 is open and Q_2 is closed, a positive current I_{cmp} will charge the C_1 and C_2 capacitances. When the potential of the V_1 node reaches the point where transistor Q_4 conducts more current than Q_5, the node V_2 is turned off and V_{out} on, which will close transistor Q_1 and open Q_2. The capacitive coupling through C_2 will pull the potential of V_1 equal to:

$$\Delta V_1 = \frac{V_{dd} C_2}{C_1 + C_2} \,. \tag{4.1}$$

The current I_{width} will discharge the capacitor C_1 to the potential of V_1 again reaches the switching potential of the inverter buffer. The output will be turned off, and the capacitive coupling through C_2 will lower the potential of V_1 by ΔV_1. The time from the output comes on until it turns off again, is dependent of the current I_{width} and the size of the capacitor C_2, and it can be expressed as

$$t_{width} = \frac{\Delta V_1}{dV_1/dt} = \frac{(C_1 + C_2)\,\Delta V_1}{I_{width}} = \frac{C_2 V_{dd}}{I_{width}} \,.$$

The charging time, that is the time the output is off, is expressed as

$$t_{off} = \frac{C_2 V_{dd}}{I_{cmp}} \,.$$

As expressed in equation 4.1 the quantity that V_1 switches when the output switches is given by the capacitive division between C_1 and C_2. It is therefore important that the ratio between C_1 and C_2 is carefully selected, to make sure that the potential of V_1 is not

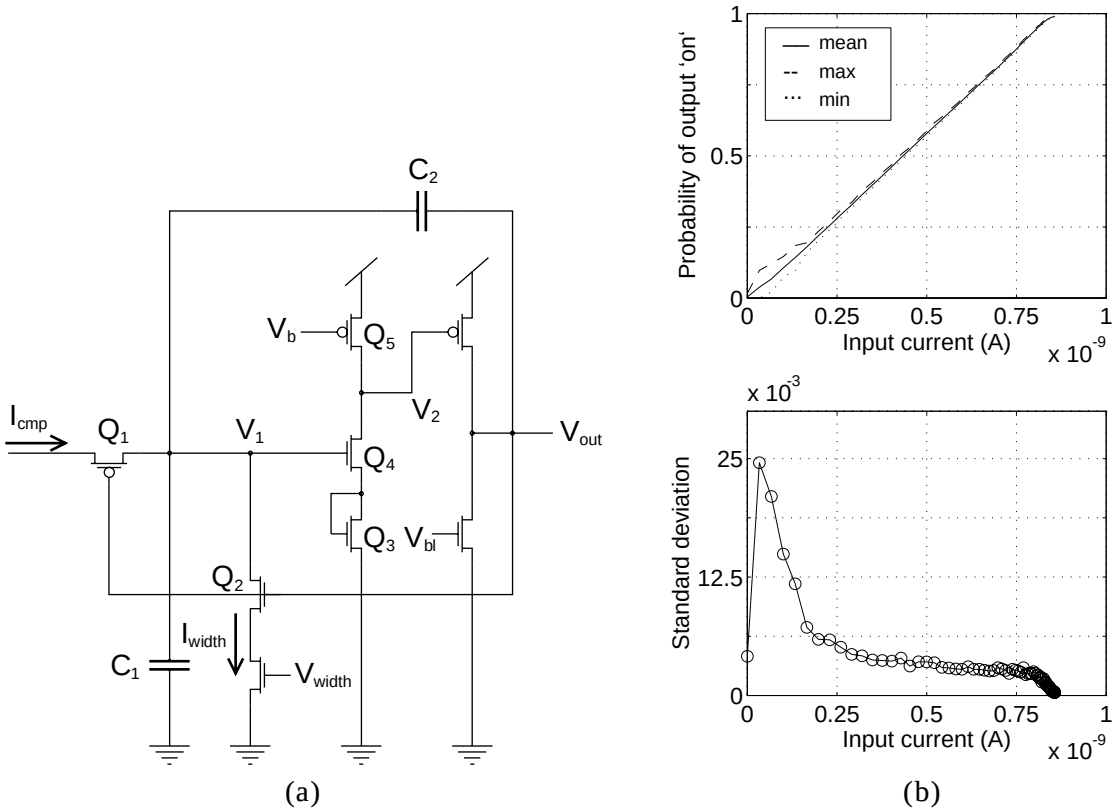


Figure 4.4: Pulse stream generator: (a) Its implementation. (b) measured response to a ramp input current. The V_{width} bias is equal 0.32 volt, which gives a pulse width of $200\mu s$. The V_b and V_{bl} biases was set to 4.2 and 0.8 volt respectively.

pulled all the way to ground when V_{out} switches low. A rule of thumb is to keep the C_2/C_1 ratio in the range of 1/10 to 1/5.

The V_b and V_{bl} biases limit the static current consumption in the inverter buffer. The diode coupled transistor Q_3 is applied to raise the switching threshold of the buffer. This makes it possible to use a larger range of capacitive division ratios, ie. the layout of the capacitors is more straightforward.

Figure 4.4 (b) shows a measurement of the duty cycle of the output signal as a function of the input current. The duty cycle is equivalent of the probability of the signal being on. For each input current 100 measurements was performed, and the mean, maximum and minimum values were calculated. The lower plot shows the standard deviation. As we can see there is some variation, and this variation is only beneficial. The V_{width} bias was adjusted for a pulse width of approximately $200\mu s$.

4.1.4 The complete circuit

The assembling of two integrators, one comparator and two pulse generators into a complete output error generation circuit is shown in figure 4.5. In figure 4.6 and 4.7 measurements of the probability of the positive and negative error parts for the complete output error generation circuit are shown. The circuit was implemented with source degeneration in the differential pair of the comparator. For the first measurement the target

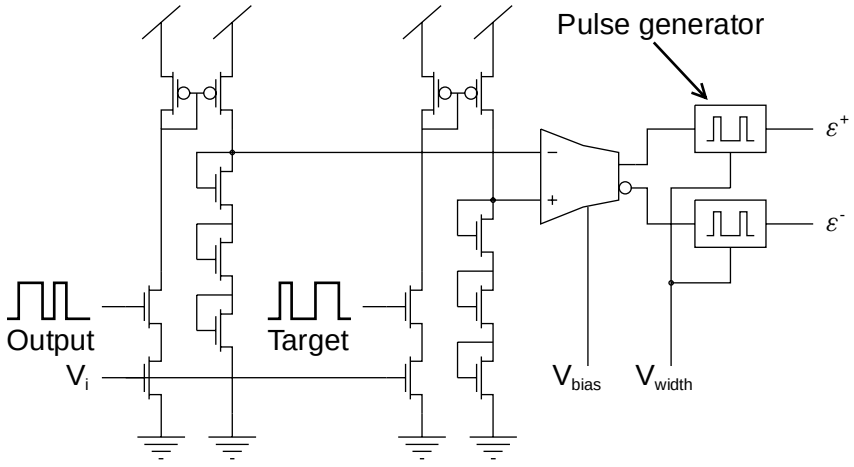


Figure 4.5: Complete output error generation circuit.

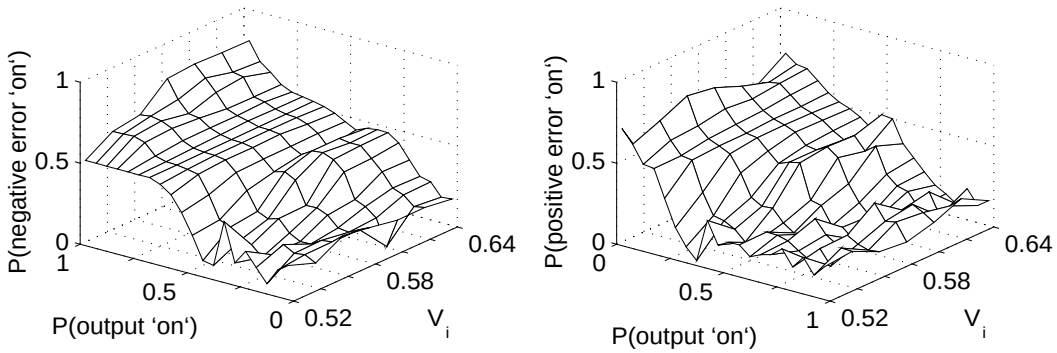


Figure 4.6: Output error generation response for different integrator biases: Measurement of the response of a complete output error generation circuit. The duty cycle of the target signal was kept at 50%. The plotted results are the mean result of 100 measurements.

input was kept at one value, a $2.5kHz$, $5V$ pulse height, 50% duty cycle signal which gives a probability of 0.5 for the target being on. For an output signal with the same frequency and pulse height the duty cycle was swept. This was done for different values of V_i in the range from $0.52V$ to $0.64V$, which was the biases that gave the best response. It is slightly larger than for the simulation of the integrator, but the actual threshold voltage was larger than specified in the transistor parameters used in the simulation.

As we can see from figure 4.6 the best shift in output signals is obtained for a bias value of approximately $0.56V$. For this bias a new measurement was done, were signal frequency of both the target and output, and dutycycle of the output was swept. The result is shown in figure 4.7.

In the measurements of figure 4.6 and 4.7 we can see that the positive and negative error is not totally off when the output duty cycle is less than or greater than the target duty cycle respectively. The reason can be traced back to ripple on the output of the integration circuits, and problems getting the universal counter used in the measurement to trigger on the actual output, and not on small amplitude noise.

Since the differential pair of the comparator is equipped with source degenerative

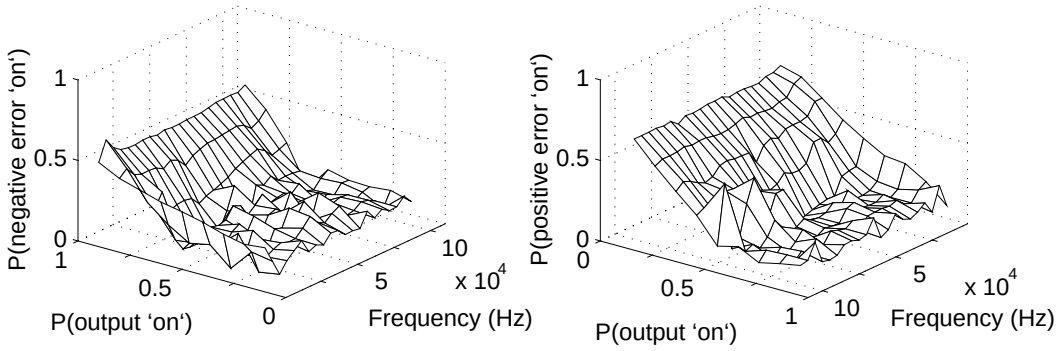


Figure 4.7: Output error generation response for different pulse frequencies: Measurement of the response of a complete output error generation circuit. The duty cycle of the target signal was kept at 50%. *Mark the axis!*

transistors the comparator may be responsive to twice as much offset in the differential input as a standard comparator. With a small AC swing on the integrator nodes such high offsets may result in erroneous outputs. It is therefore important that the differential pair transistors are scaled properly, to reduce probability of large offsets.

CHAPTER

5

Long term synaptic weight storage

One of the greatest challenges faced during the development and implementations of a VLSI artificial neural network, is the implementation of the synaptic weight storage circuits. Several of the key aspects of weight storage are in opposition to each other. The memory should be physically small, but have a large resolution. The retention time of the stored value should be infinitely, but changing it should be fast and easy. Two analog memory schemes with a potential of fulfilling all of these aspects are investigated in this chapter. One of the techniques is then utilized in the implementation of a synaptic memory block with an interface that fits the stochastic neural network.

An absolute necessity in any adaptive circuit or system is the possibility to store values, and change the stored values, to making the adaptation possible. In the neural network context it is the adaptation of the synaptic weights that makes the network able to solve problems. There are several aspects of the synaptic memory influencing the adaptation potential of a network. The memory should preferably have an infinite resolution, but noisy discrete memories with an resolution of about 10 bits show good results, [von Lehmen *et al.*, 1988]. Accurate digital memories on the other hand tends to need larger resolution for a network to achieve the same adaptation potential, as much as 16 bits may be necessary. This excludes digital memories from being used in very dense networks, since every synapse needs one memory unit and the units physical size therefore must be reduced to an absolute minimum. A 16-bit register consumes way to much silicon area, considering the need for transforming the stored value to the signal representation used in the rest of the network. In some applications they can be an interesting approach though, since they are fast, easy to program and come in standard cells. But for a scalable network implementation as the one presented here, it is not of great interest.

The synaptic memory must also have the ability to store values over long time. Preferably no change in the stored value should occur, except through explicit programming. There should be no need to relearn a problem due to synaptic forgetfulness.

Most artificial neural network training algorithms assumes that the synaptic weights are randomly initialized. If not the training process may be greatly hampered, and convergence of the training set may be impossible. The possibility to initialize the memory before training starts must be present, and the initial value should be different for all memories, and not the same for each initialization.

5.1 Analog storage techniques

Analog storage of values imply storing charge on a capacitor. This gives a resolution which is only limited by the signal to noise ratio. Even if the programming circuitry is only capable of charging and discharging minimum fixed amount of charge, which signifies that the memory have a discrete resolution dependent on the amount of the charge, the iterative programming algorithms used in neural networks together with the noise present in any analog VLSI system, introduces the possibility of hitting any intermediate state. The potential of implementing a memory with sufficiently large resolution is therefore present.

In VLSI CMOS basically two approaches to charge the capacitor can be used. One technique uses a transistor as a voltage controlled current source to apply charge on the capacitor. The most significant problem with such a device is short time charge retention. The leakage current of the reversed-biased diffusion to bulk junction in the programming transistor only allows storage times in the amounts of a few seconds. The charge retention time can be increased by either introducing feedback sample and hold circuitry or a off-chip controlled refresh scheme [Vittoz *et al.*, 1991]. The sample and hold circuit only introduces a mediocre charge retention at the cost of extra circuitry, and the memory with a refresh scheme will only allow synchronous access to the stored value, which make their potential as storage device in an asynchronous neural network small.

The other arrangement is to use a floating circuit node to store charge on. This model utilizes the excellent insulating capability of the silicon dioxide (SiO_2) that separates the different layers of a CMOS chip. The floating node is totally trapped inside the SiO_2, and only a part of it constitutes the gate of a transistor and makes it possible to read out the stored value. The insulating capabilities of the silicon dioxide makes charge retentions for up to 10 years with only 0.1% charge loss possible, [Carley, 1989]. This type of non-volatile storage have been used in commercial EPROM and EEPROM digital memories for years. Several methods are available to make charge conduct through the insulating material when programming. The two oldest are *avalanche injection* and *hot carrier injection*. These methods are very power-consuming and makes it impossible to implement dense circuit designs. The last two approaches are ultraviolet-light (later referred to as UV-light) exposure and Fowler-Nordheim tunneling, which draw very small quantities of current, and can be implemented in even the cheapest standard CMOS process.

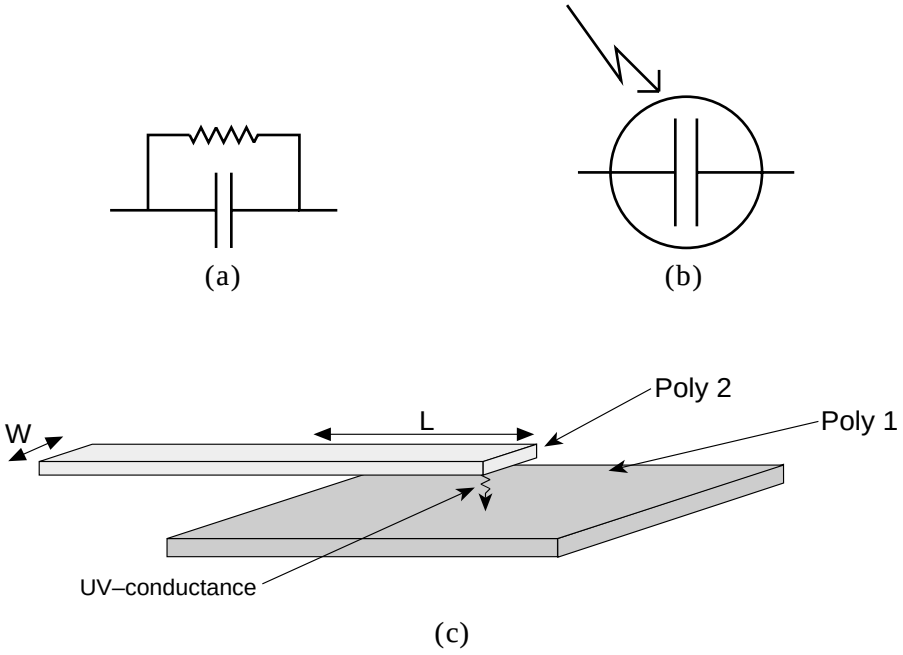


Figure 5.1: UV-structure: (a) A simple first order circuit model for a UV-structure. (b) Symbol representing the structure. (c) Simple silicon layout of a poly1-poly2 UV-structure.

5.1.1 UV-light memory

The UV-memory structures are Si-SiO_2-Si layered structures were the SiO_2 insulator introduces a $4.25eV$ energy barrier to electrons. Exposing short wave (less than $290nm$) UV-light to the structure will introduce enough kinetic energy in the electrons in the silicon valence band, to make them surmount the energy barrier and enter the silicon dioxide conduction band [Williams, 1965]. If there exists an electric field in the dioxide these electrons will introduce an electric current through it. The amount of current is dependent of UV-light intensity and the size of the electric field in the dioxide [Kerns *et al.*, 1991, Abusland, 1994]. The insulating properties of the dioxide are not lost due to UV-light exposure. When light is turned off, the conducted electric charge are trapped inside the floating node for years.

This simple physical model leads to the first order circuit model shown in figure 5.1 (a). A simple silicon structure utilizing two separate poly layers to form a UV-capacitor is shown in 5.1 (c) and figure 5.1 (b) shows a symbol representing a UV-structure. The conductance is the UV-induced dioxide current. Since the upper poly layer is hiding parts of the lower poly layer, the UV-conductance is mainly created along the edges of the upper poly layer.

For a long time it was assumed that there was a linear dependence between the UV-conductance and the electric field in the dioxide. Later investigations have shown that the dependence is non-linear [Kerns *et al.*, 1991, Benson and Kerns, 1993]. This non-linearity is most dominant for very low electric fields. As the electric field in the dioxide drops below a certain constant value (E_0), the conductance is significantly reduced as the electric field is reduced, [Benson and Kerns, 1993]. For electric fields larger than E_0 the UV-induced current is more or less linearly dependent of the electric

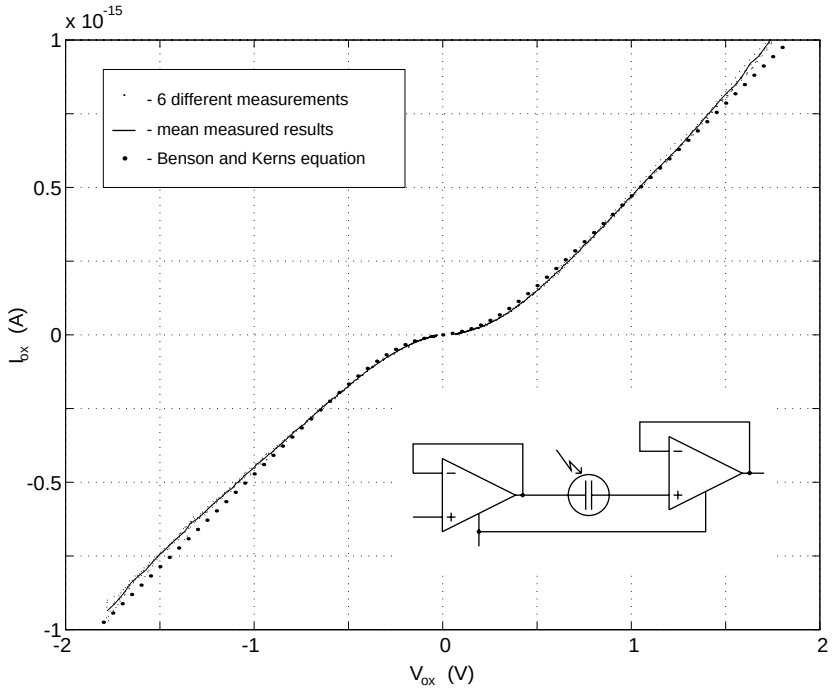


Figure 5.2: UV-conductance characteristic: The plot shows both the measured characteristic and a theoretical fitted curve. For the plot of the theoretical curve $I_{leak} = 0.0A$, $G = 6.3 \cdot 10^{-16} A/V$, $g = 1 \cdot 10^{-16} A/V$ and $V_0 = 0.3V$. The circuit setup used in the measurements is shown in the inset.

field. The reason for the non-linear behavior is not known, but Poole-Frenkel conduction and charge trapping in the SiO_2 have been used as explanations, [Kerns *et al.*, 1991, Benson and Kerns, 1993]

No accurate analytical model of the UV-light induced current exists. [Benson and Kerns, 1993] presents an empirical model that fits to measured data. In this model the dioxide current is expressed as

$$I_{ox} = I_{leak} + GV_{ox} - V_0 (G - g) \, tanh\left(\frac{V_{ox}}{V_0}\right) , \tag{5.1}$$

where I_{leak} accounts for parasitic leakage currents in the UV-structure, and the *tanh* and V_0 parts models the non-linear behavior for small electric fields. A problem with this model is the I_{leak} term. The major leakage currents in a UV-structure is due to parasitic UV-conductance to other nodes than the control node. The I_{leak} term is therefore not constant, but dependent on the potential on the floating node, but for most practical needs this is not important. Careful design can reduce leakage currents to negligible quantities, and the I_{leak} part can be ignored. Figure 5.2 shows a plot of measured I-V characteristic of a poly1-poly2 UV-structure, and a theoretical plot of equation 5.1 for comparison. The measurement was obtained through the circuit setup shown in the inset of the plot. To make the circuits operating environment as close to a practical environment as possible, the circuit was isolated between two OTA's connected as followers. The floating node voltage as a function of time as it approached the control

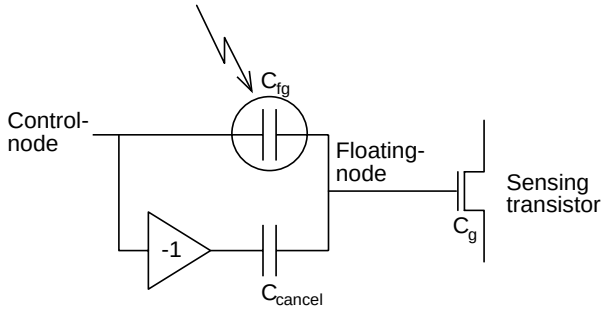


Figure 5.3: Capacitive canceling structure: Canceling the UV-capacitance allows fast programming and stable floating node

node voltage was measured. The current was then obtained by equation:

$$I_{ox} = C_{fn}\frac{dV_{ox}}{dt} .$$

The floating node capacitance was extracted from the layout geometries and interlayer capacitance parameters supplied with the chip.

UV-memory structure implementation

As seen from the UV-structures I-V characteristic in figure 5.2, the dioxide current is very small, which indicates that the main goal of the memory structure implementation must be to achieve as high speed as possible. Gaining sufficiently large resolution should not be a problem. [Maher, Unpub] propose a memory scheme where the floating node is capacitively coupled through an equally sized capacitance as the UV-capacitance, to a node containing the inverted control signal as shown in figure 5.3. This capacitive canceling technique ensures that the potential across the UV-capacitance can be high, and therefore reduces the programming time. The canceling effect also have the property of keeping the floating node potential stable during fast changes in the control node potential. This is very important since the fastest programming is achieved by driving the control node to either of the supply rails when programming. Without the capacitive canceling a change in programming direction would have resulted in a dramatic change in floating node potential through the capacitive coupling of the control node and floating node. A condition that must be fulfilled for the floating node to be stable is that the relationship between the control node voltage change and the UV-capacitance is the same as the relationship between the *inverted* control node voltage change and the canceling capacitance.

To achieve as fast programming as possible the total floating node capacitance must be reduced to an absolute minimum, which implies that the UV-capacitance and the canceling capacitance must be small. But as we know the UV-induced conductance is created along the edges of the upper layer forming the UV-capacitance, and the total amount of UV-induced current is therefore determined by the length of the UV-light exposed edge of the upper layer. [Kerns *et al.*, 1991] shows that it is a linear dependence between exposed edge and conductance, except for exposed edges less than $5\mu m$. To maximize the conductance and minimize the capacitance the upper poly node should

be long and narrow. The programming time constant of a memory structure with a canceling capacitor can be expressed as

$$\begin{aligned}\tau_{prog} &= \frac{C_{fn}}{G_{ox}} = \frac{C_{fg} + C_{cancel} + C_g}{G_{ox}} = \frac{2 \cdot W \cdot L \cdot C_{ox/\mu m^2} + C_g}{(2L + W)\, G_{ox/\mu m}} \\ &= \frac{W}{1 + W/(2L)} \cdot \frac{C_{ox/\mu m^2}}{G_{ox/\mu m}} + \frac{1}{2L + W} \cdot \frac{C_g}{G_{ox/\mu m}}\,, \end{aligned} \tag{5.2}$$

where $C_{ox/\mu m^2}$ is the capacitance per unit area of the dioxide, $G_{ox/\mu m}$ is the UV-induced conductance per unit length, C_g is the gate capacitance of the sensing transistor, and W and L are the width and length of the exposed top node. One end of the strip must be used to connect the node to either the sensing transistor, if it is the floating node, or the programming logic, and is therefore not contributing anything to the exposed edge. As can be seen from equation 5.2, increasing the length of a long and narrow device will increase the capacitance with approximately the same factor as the conductance and only reduce the effect that the gate capacitance of the sensing transistor have on the time constant. In an adaptive synaptic environment transistor mismatch, and offsets in the sensing transistor environment, are not introducing any problems at all, and the size and capacitance of the sensing transistor can be minimized. So large UV-devices should only make small improvements in the programming time constant.

Another aspect that must be considered when designing a UV-structure is the fact that UV-induced conductance arises between all silicon structures separated by SiO_2. Therefore all other circuits on a chip except the UV-structures must be shielded against UV-light. This is accomplished by using a metal layer to cover all the computation circuitry, making holes in the shield above the UV-structures. It is also important to reduce the leakage currents from the floating node to a minimum. The layout of the structure should shield the floating node from other silicon structures than the control node. [Benson and Kerns, 1993] also reports that the UV-light can be reflected between the metal shield and underlying structures for distances up to $30\mu m$ away from the holes in the shield. Even though the conductance decreases significantly as the distance from the hole increases, special care must be taken to reduce the effect of this light reflection.

Figure 5.4 shows a layout that incorporates many of the discussed aspects. The control node is formed by poly1 and the floating node by poly2. The poly1 control node shields the floating node from underlying silicon (well or substrate). A *guard ring* formed by metal1 and a poly1/metal1-contact protects the floating node from forming unwanted UV-conductance with silicon structures under the metal2 shield. This guard ring also protect surrounding circuitry from indirect UV-light exposure. Some light will still reach the computational circuitry, but the amount is greatly reduced. The guard contact violates the design rules of many environments, and not all silicon foundries may allow you to incorporate such structures on a chip, but for the $2\mu m$ P-well and N-well processes from Orbit Semiconductor Inc. no problems have ever occurred. It is also important that your layout tool does not split the guard contact in several uniformly spaced quadratic contacts.

The floating node is formed by a poly2 piece that is just big enough to allow a contact to metal1. The rational for using metal1 to connect the floating node part of the UV-capacitance to the outside world, is to reduce leakage to the poly2 floating node and reduce the poly2 edge not exposed to UV-light to a minimum.[1] If poly2 is used

[1] Poly2 area not exposed to UV-light contribute capacitance, but no conductance except from reflected

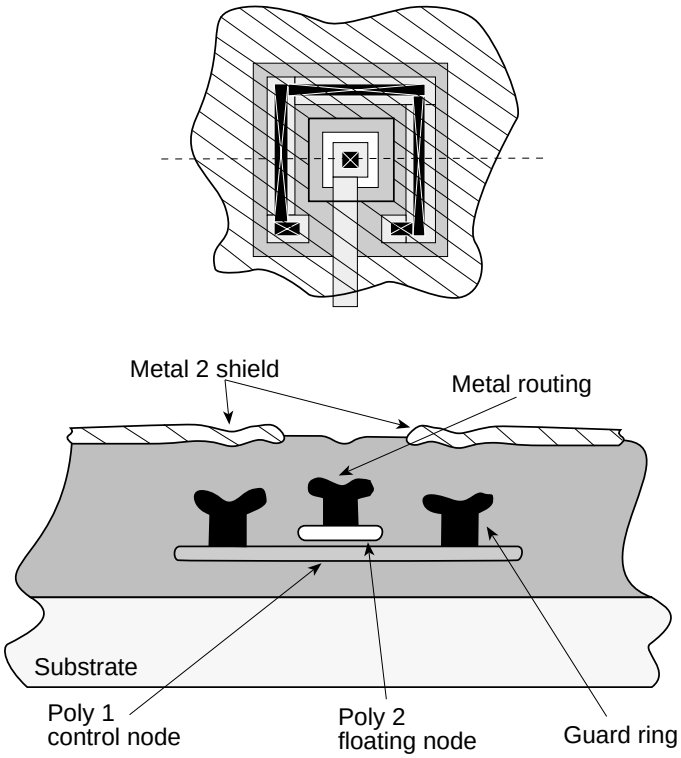


Figure 5.4: Shielded UV-structure: A UV-structure layout that reduces unwanted UV-light exposure to a minimum. Upper figure is a lithographic layout view, while the lower shows a cut view along the dashed line.

as connecting material there must either be quite an amount of unexposed edge or an increased leakage current to the well or substrate. The structure shown in figure 5.4 is the one used in the current measurement shown in figure 5.2.

Alternative implementation utilizing poly1-diffusion thinoxide

As an alternative to the poly1-poly2 structure discussed, a similar structure which uses the thinoxide of a transistor gate to form the insulator between the control node (diffusion) and floating node (polysilicon) may be implemented. The reduced thickness of the dioxide separating the two nodes will increase the electric field in the dioxide compared to the poly1-poly2 structure. The increase of the electric field will increase the dioxide current as well. The layout of the structure can be made as for the poly1-poly2 structure, except that the control node is formed by n-active or p-active, the floating node by poly1 and the guard ring by an active-metal1 contact. This layout will introduce a highly illegal stacked gate contact on the poly1 floating node. Early investigations, though, indicate that this may not be a problem.

Other problems introduced in this structure is the variable UV-capacitance and current leakage to the substrate. The first problem is due to the fact that the gate-capacitance of the transistor like structure vary significantly for different potentials across the oxide. The substrate leakage current is a result of the channel formed under the poly

light conductance, which is negligible compared to the conductance of directly exposed edges.

floating node. There will be a substantial current between the floating node and the substrate. This may be compensated for by arranging the structure in a well, where the well potential is the same as the diffusion control node potential.

5.1.2 Fowler-Nordheim tunneling memory

The main reason for searching for alternative long term memories, is the UV-light schemes need for a rather awkward external light source. There are two alternatives. Make very small UV-light sources that can be mounted on top of the chips, or use a different approach to charge the floating node. Fowler-Nordheim tunneling may be one such approach.

Fowler-Nordheim tunneling is the standard method to erase, or both program and erase, commercial digital EEPROM's. But the last few years several researcher have investigated the possibilities of using Fowler-Nordheim tunneling to implement analog memories for trimming analog circuits [Säckinger and Guggenbühl, 1988, Carley, 1989], and for synaptic memories in neural networks [Vittoz *et al.*, 1991, Pasero, 1991, Sin *et al.*, 1992, Yang *et al.*, 1992] .

The tunneling mechanism

The tunneling memory must cope with the same problem as the UV-light memory, to get the electrons to cross the dioxide separating the floating node from the control node. There is an energy barrier of approximately $3.2eV$ that prevents electrons in the silicon conduction band[2]to enter SiO_2 [Carley, 1989]. The kinetic energy of the electrons are only large enough to give them a probability to tunnel a small distance into the dioxide. If the potential in the dioxide at this point is less than $3.2V$ higher than in the silicon material, the electrons will return to the silicon. But if the potential is large enough, the electrons will be carried away with the electric field. The result is a small current of electrons away from the silicon material. The distance an electron is capable to tunnel, is at a maximum $5nm$. As the distance decreases, the probability of an electron making the distance increases. Increasing the electric field in the dioxide decreases the distance the electrons must tunnel initially, and therefore increases the electron current.

Theoretically the current density through the SiO_2 can be expressed as

$$J = A \cdot E^2 \cdot e^{-B/E} \ , \tag{5.3}$$

where E is the electric field in the dioxide, and A and B are constants that can be worked out to various degrees of sophistication, [Lenzlinger and Snow, 1969, Sin *et al.*, 1992, Sze, 1981, Concannon *et al.*, 1993]. Of practical interest is the fact that both A and B are functions of the effective mass of free electrons in the forbidden gap between the material the electrons are leaving and the dioxide. If the two nodes separated by the dioxide is of different doping degrees, the result is that for the tunneling current for a positive electric field is different from the tunneling current for a negative electric field, even for the same field strengths.

From the description above and equation 5.3 we can conclude that there are two major factors that influence the amount of electron current. Both the thickness of the

[2]It is a $1.1eV$ difference in the barrier voltage for electrons in the valence band and the conduction band, which is the reason for the difference from the barrier height for UV-light induced electrons.

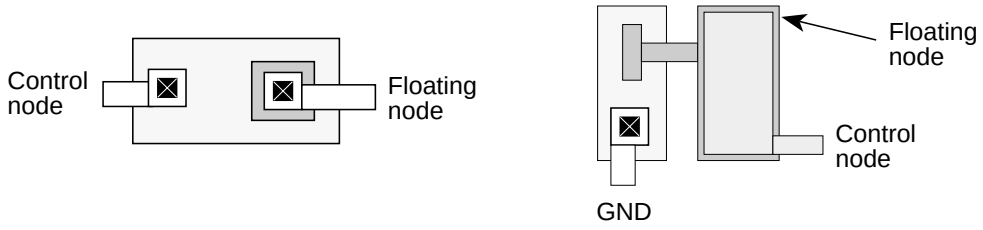


Figure 5.5: Standard CMOS tunneling device: The corners in the poly gate overlap introduces local enhancement of the electric field. (a) structure with diffusion as control node. (b) Alternative arrangement with poly2 as control node.

dioxide and the potential across it determines the electric field. Thinner dioxide or larger potential yields higher electric field and more current. In a standard $2\mu m$ process the thinnest dioxide separating two layers is the gate-oxide of a transistor. With a gate-oxide thickness of about $40nm$, programming voltages of about 25 volt or more are needed to get a current to flow. With a gate-oxide breakdown voltage of about 28 volt there is a significant chance of destroying the device if fast programming in such a device is to be achieved.

In some commercial EEPROM's the use of an extremely thin tunneling dioxide in the range from 8–$10nm$ is introduced, which makes it possible to get fast programming with only $5V$ programming pulses. Another technique used to reduce the programming voltage is to utilize the fact that it is not the electric field in the bulk dioxide that matters, but the electric field $5nm$ from the Si/SiO_2 intersection. This together with the fact that electric fields can be enhanced locally in sharp edges or corners, can reduce the programming voltage. Some special processes makes use of spikes or other nonuniformities on the silicon surface, referred to as textured surface, to gain local field enhancement in the tip of the spikes, which can increase the electric field by a factor of 4 to 5.

The ultra thin dioxide and the textured surfaces need special processing, which implies lower processing yield and higher costs. On the other hand there is possible to make tunneling structures with relatively low programming voltages in a standard CMOS process. [Carley, 1989] uses photolitographic possibilities of any MOS structure to locally enhance the electric field. By introducing corners in the poly gate of a poly-diffusion structure as shown in figure 5.5 (a), the electric field is concentrated at the corners of the poly rectangle. With this scheme one can achieve appreciable tunneling currents for programming voltages in the range of 14–17 volt, which is well below the breakdown voltage. The poly gate of the structure is serving as floating gate, while the diffusion makes the control node. The stacked gate contact is not necessary, and should probably be avoided, but to reduce the amount of test structures on the chip a device that could both function as a poly1–diffusion UV-light and tunneling memory was implemented.

The main problem with the use of diffusion as control node, is that the same control node can only be used to program in one direction. A negative (relative to the substrate) programming voltage on a n^+-active control node will create a forward biased diode-junction between the diffusion and substrate, which prohibits one from applying sufficiently large programming voltages. To allow programming in both directions, two programming devices can be applied. One with a n^+-active control node for pro-

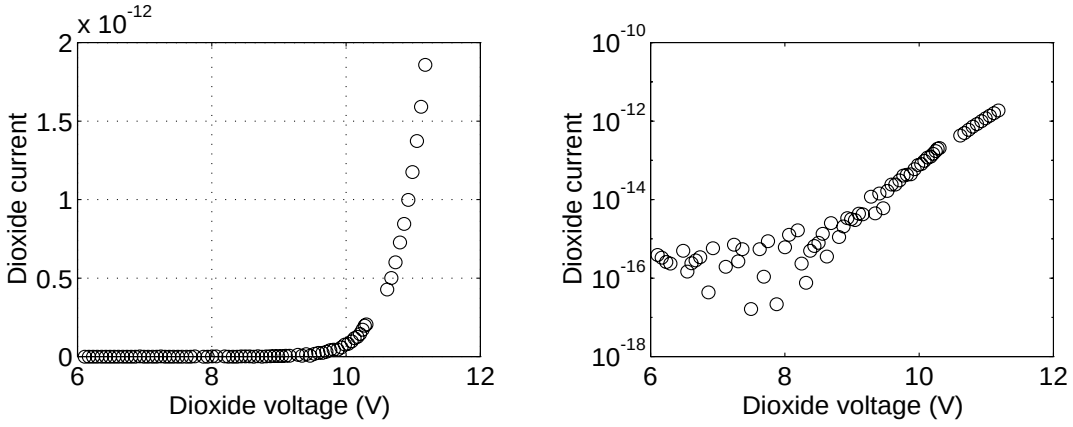


Figure 5.6: Tunneling device characteristics: The I-V characteristics for the poly1-diffusion tunneling device showed in figure 5.5 (a). Linear plot left and logarithmic replot right.

gramming up, and one p^--active for programming down. Alternatively the device can be implemented as shown in figure 5.5 (b), where a capacitively coupled poly2 node makes the control node. With this arrangement the same control node can be used to program in both direction. The tunneling still takes place through the *transistor* gate dioxide.

In figure 5.6 an I-V characteristic for the tunneling structure in figure 5.5 (a) is shown. The characteristic was obtained in the same way as for the UV-structure, by measuring floating gate voltage change and calculate the current. For each measurement with low programming voltages a $100ms$ programming pulse was used. For the largest programming voltages the pulse width was reduced to $10ms$. The dioxide voltage was calculated on the basis of the tunneling structure capacitance[3] and the sensing transistor capacitance. As we can see, the tunneling electron current can be several orders of magnitude larger than a UV-induced current, which results in faster programming. To obtain a $10V$ drop over the dioxide for an initial floating node potential of $1.5V$, a programming voltage in the amounts of $18V$ had to be applied. If the size of the sensing transistor had been increased (larger gate-capacitance), the programming voltage could be reduced, since a change in the control node potential had resulted in less change in the floating node potential.

Disadvantages with tunneling memories

The high voltages needed to program a tunneling memory is not a major problem. As already stated the currents introduced by these high voltages are small, so there should be no need to take any precautions in the layout of the high voltage power supply lines. The circuitry generating the control voltage potentials must be made to endure the high voltages. This can be achieved by distributing the high voltage over several cascaded diode connected transistors, which will prevent damage of the tranistor gateoxide.

The real problem in tunneling devices are the degradation of the tunneling dioxide. For each programming pulse applied on the control node that introduce tunneling, some

[3]For this measurement the tunneling capacitance and the sensing transistor capacitance was assumed constant for simplicity, even though they vary slightly with the floating node potential.

off the electrons that enters the dioxide will be trapped [Witters *et al.*, 1989]. The charge of these trapped electrons will reduce the electric field in the dioxide, and therefore reduce the tunneling electron current. The result is that the programming time constant increases more and more for each change of the floating node potential. In the literature this is referred to as a closing of the programming window, where the programming window is constituted by the maximum and minimum floating node potential achieved by two fixed width and fixed height up and down programming pulses.

The amount of charge trapped is a function of the maximum electric field in the tunneling dioxide. Fast transitions of the control node potential introduces short time high electric fields in the dioxide, so the trapping effect can be reduced by using programming pulses with longer raise times [Thomsen and Brooke, 1990].

The degradation process is not a failure mechanism in a neural network memory. The iterative weight adaptation algorithms should still have a potential to teach the network a set of patterns, even though the training time will increase for each relearning of a new pattern set. There is also the possibility to increase the programming voltage with a small amount each time the degradation problem become a nuisance.

Another effect that arises in tunneling devices with diffusion as control node is that the large voltage applied results in a substrate hole current [Witters *et al.*, 1989]. This diffusion-substrate current involves a deep depletion band-to-band current, which even can be amplified by avalanche impact ionization. It leads to the generation of *hot carriers*, which is responsible for trapping of positive charges in the dioxide and an opening of the programming window for the first cycles of programming pulses [Witters *et al.*, 1989]. The opening of the programming window is small, and is soon canceled out by the electron trapping.

As for the UV-light memory, it is important that the floating node is kept stable during programming. Achieving a stable floating node for a tunneling device is a bit more difficult than for a UV-device. A canceling approach is very difficult to implement. The canceling capacitor can not be made of the same material combinations as the tunneling device, since it results in a tunneling current in the canceling device. The current will be in the opposite direction of the programming current, and both the capacitive coupling and the tunneling current will be canceled out. Implementing the canceling capacitor in a different material combination is difficult, since matching capacitances of physically small VLSI devices are nearly impossible.

5.1.3 The memory of choice

The tunneling memory is very attractive since it do not need any external equipment except a high voltage power supply source. On the other hand much work is still to be done before memory devices for fully asynchronous neural networks that works in practice is developed. The amount of research necessary to achieve this goal was the main reason that I did not choose a tunneling programming scheme for the neural network implementation developed in this thesis. Even though I seriously considered using a tunneling memory device, it would have taken to much time to investigate and develop the memory circuits. It is my belief though that Fowler-Nordheim tunneling memories are the choice of the future for neural networks with on-chip storing of weights.

Excluding tunneling as programming method leaves the UV-light memory as the only interesting choice. The memory structure is fairly simple and its behavior is adequately proven. UV-light memories have also been used with success in both trim-

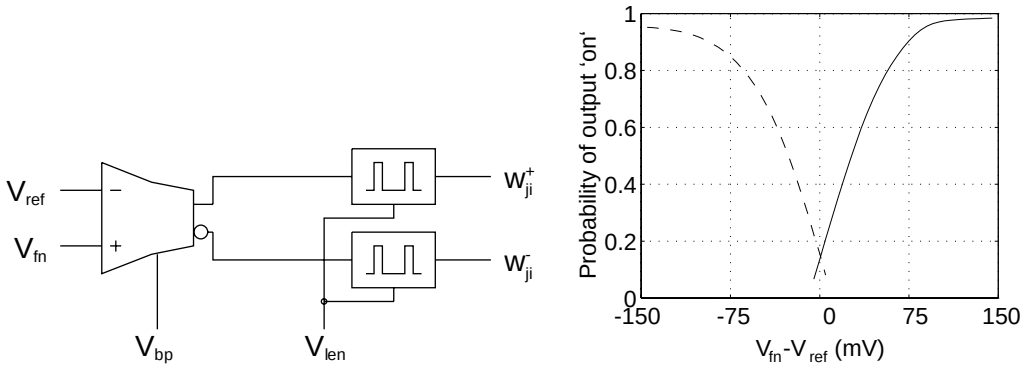


Figure 5.7: Voltage to bipolar pulse stream transformation: (a) The circuitry, and (b) the output pulse probabilities as a function of input voltage change relative to the reference voltage. Solid line is positive output and dashed line is negative.

ming offsets in amplifiers, [Mead, 1989a], and neural networks, [Soelberg *et al.*, 1994, Abusland, 1994].

5.2 UV-programmable synaptic weight memory

To achieve positive and negative synaptic weights from a voltage stored on a floating node, some kind of differential approach must be used. The two most common methods are either to make a differential implementation with two memories, or an implementation with one memory and let the value of the stored voltage be relative to a reference voltage. Since two UV-structures is needed in the first method it is space consuming, and therefore the use of it was not considered.

5.2.1 Voltage to pulse stream transformation

As defined in section 3.4.2 the output from the synaptic weight is a differential bipolar stochastic signal. There was also a constraint that only one of the differential signal lines can be on at the same time. A set of circuits that can implement such an output on the basis of a differential input was discussed in the previous chapter. One comparator circuit with an inverting and non-inverting current output drives two pulse generators to make the bipolar stochastic outputs. Figure 5.7 (a) shows the connection of the comparator and pulse generators that transform the stored floating node voltage to a differential pulse stream representation. A plot of the pulse probabilities for the two output signals for different floating node voltages relative to a reference voltage of $V_{dd}/2$ is shown in figure 5.7 (b). There was not possible to measure lower pulse rates than shown because of trigger problems in the universal counter used for the measurement. If the outputs are examined on a oscilloscope one will see that the outputs are zero for zero or negative input current. At the other extreme the characteristic saturation of an OTA is reflected in the output pulse probability. By increasing the V_{bp} bias this saturation characteristic can be avoided. But then of course we get a smaller region of operation for the floating node.

From the plot we can also see that there is a small region where both of the outputs pulses at the same time. This is a result of offsets and transistor mismatch in the comparator, and violates the constraints set for the memory. But the pulse rates are as we can see so low that a or-sum will be almost linear (see figure 3.3), and pulse overlaps is therefore negligible and no problems should occur.

For the final memory circuit an extra copy of the non-inverted output of the comparator will be added, which represents the sign of the stored weight.

5.2.2 Programming the UV-memory

To program the UV-memory with capacitive canceling we need a circuit capable of generating both a non-inverting and inverting output. The shift in the outputs as input changes must be symmetrical to avoid other changes on the floating node than those introduced by UV-light induced charging or decharging. The output characteristics must therefore be both symmetric along the voltage axis and the time axis. A mismatch in voltage changes will result in unwanted floating node change due to capacitive coupling. If the change of one of the outputs is delayed compared to the change of the other output, a short transient change of the floating node potential will occur. For a memory that is supposed to be adapted by pulse signals, which implies rapid changes of the control node, none of these problems can be tolerated.

To achieve as fast programming as possible the inverting and non-inverting outputs should be pulled to the supply rails when an adaptation pulse arrive. If we recall from section 3.5.1 the weight adaptation is represented as a bipolar stochastic pulse stream. To be more specific the floating node potential should be increased when the signal representing positive weight-change is on, and the signal representing negative weight-change is off. A decrease in floating node potential should occur for the complementary signal values. For the situations were both lines are on or off at the same time the floating node should be kept stable.

The comparator circuit comes to our rescue once again. It is a circuit with a very symmetrical characteristic for the outputs. Both the inverted and non-inverted outputs are equally delayed in the circuit, and with negligible current drawn at the outputs they may be pulled almost symmetrically to both supply rails. A comparator with a wide range output stage also have a high voltage gain, usually about 1000–2000 [Mead, 1989b], so large output transitions for small input changes are achieved. Figure 5.8 shows measurements of the voltage characteristic of a comparator. We can see that there is symmetry, but the cross point, that is the output when the inputs are equal, ignoring any offsets, is not at $V_{dd}/2$. This is the result of Early-effects in the current mirrors in the OTA. From the earlier discussion we know that this is not acceptable voltage output values, since a change in input will make the *raising* output to change significantly more than the *falling* output. Extra circuitry that stabilize the control nodes at other values when the inputs are equal must be introduced.

The best solution would be to drive the control signal to the same potential as the floating node potential. No electric field would be present in the dioxide and no charge would enter or leave the floating node. But if the non-inverted control node is driven to the floating node potential the inverted control node potential should be driven to the inverted potential relative to $V_{dd}/2$ to keep the floating node stable when a programming pulse arrives at the input. This is not an easy task to accomplish, and a simple circuit capable of performing such task is unkown to me.

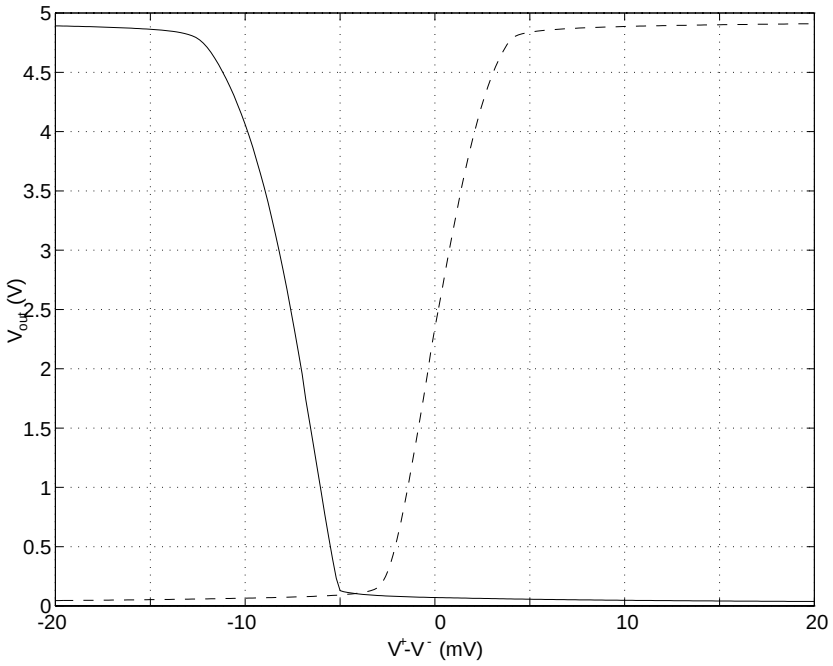


Figure 5.8: Comparator voltage characteristics: The two outputs are symmetrical but the cross point is not at $V_{dd}/2$ due to Early effects in the current mirrors.

The second best solution utilizes the fact that the only thing that matters is that the control nodes are driven to inverted potentials relative to $V_{dd}/2$ and that the electric field in the UV-structure dioxide is kept at a minimum to make the floating node voltage change as small as possible.

The natural choice for a reference voltage for floating node voltage comparison is $V_{dd}/2$. For a comparator based on the standard wide range OTA, that is without differential source degeneration, the area of operation for the floating node potential is approximately $(V_{dd}/2) \pm 150mV$. For floating node potentials outside this region the output current of the comparator will saturate at the bias current. The result is that one of the pulse outputs will be stuck at an on signal, that is a pulse probability of 1. A slow drift of the floating node potential towards the reference voltage for this situation should only be beneficial, since it will help the the floating node stay outside the regions were the output saturates. The effect can be fulfilled by driving both control nodes to the reference voltage. For a reference voltage of $V_{dd}/2$ the control nodes will be changed with equal amounts in the opposite direction when a programming pulse arrive at the input.

The current through the dioxide in the case of no programming and the floating node potential inside the $\pm 150mV$ range can be calculated with equation 5.1 and the constant values extracted from the measurements, and one will obtain that the current for a $150mV$ drop across the dioxide is approximately 35 times less than for a $2.5V$ drop across the dioxide. For a $5V$ supply voltage the forgetting time constant is at least 35 times larger than the programming time constant.

The forgetting dynamic should not be regarded as a bug. It not only help to bring a floating node that is out of range back into its region of operation, but also makes it very simple to initialize synaptic weights. By driving the control nodes to the reference

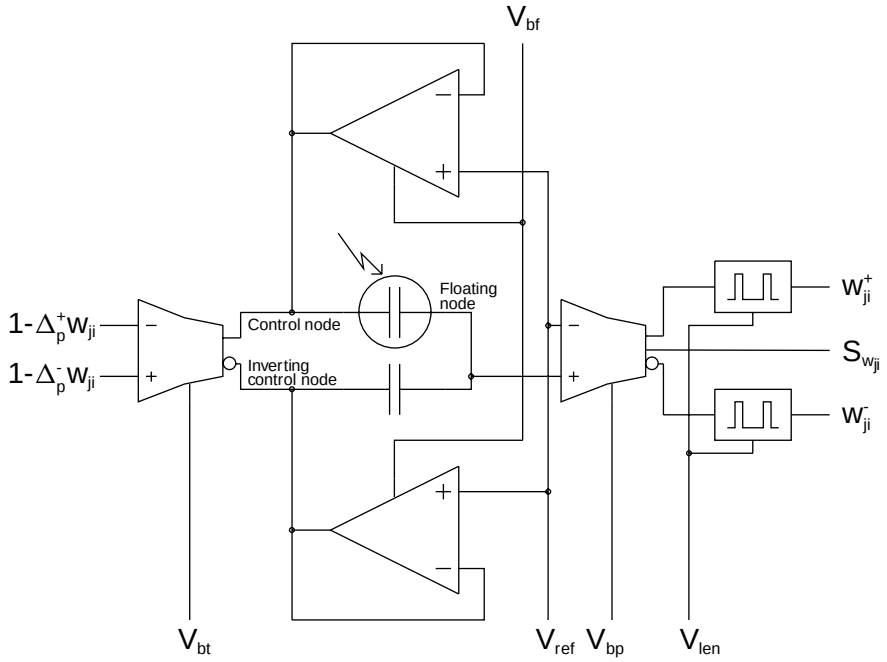


Figure 5.9: Synaptic memory circuit: This memory allows for continually adaptation of the stored weight as the weight signal is read out. The floating node voltage is only changed by UV-light induced charge. No voltage change due to capacitive coupling occur, and only a small forgetting dynamic is introduced to ensure this.

voltage and turn on the UV-light source and wait, the floating node potential will soon reach a voltage inside the $\pm 150mV$ range of operation.

One may argue that the forgetting dynamics will hamper network learning, because weights will change when they are not supposed to do so. But on the other hand the forgetting dynamics may also improve learning. One of the main problems with the backpropagation of error learning algorithm is that networks can be stuck at local minima during learning. That is the network is stuck in a local *valley* of the error space, and is not capable of getting out. For a local minima that does not lead to the right output patterns for all inputs, the consequence is failure in the learning process. The introduction of forgetting dynamics on the other hand may help the network getting out of the local minima and into an other part of the error space, where further learning can proceed.

The circuitry necessary to drive the control nodes to $V_{dd}/2$ when no programming pulses are active on the input is simple. Two follower connected simple amplifiers with a bias current slightly less than for the programming comparator, with the outputs connected to the inverted and non-inverted control node respectively and the input connected to the reference voltage, as shown in the circuit diagram for the complete synaptic weight circuit in figure 5.9. When the programming inputs are equal the follower output currents will *overpower* the programming comparator output current, and drive the control nodes to the reference voltage, as the comparator will overpower the followers when a programming signal is active at the input. If not both control nodes are driven all the way to the supply rails, the reference voltage can be adjusted to a voltage half way between the upper and lower control node voltages, and the floating node will still be kept stable during operation.

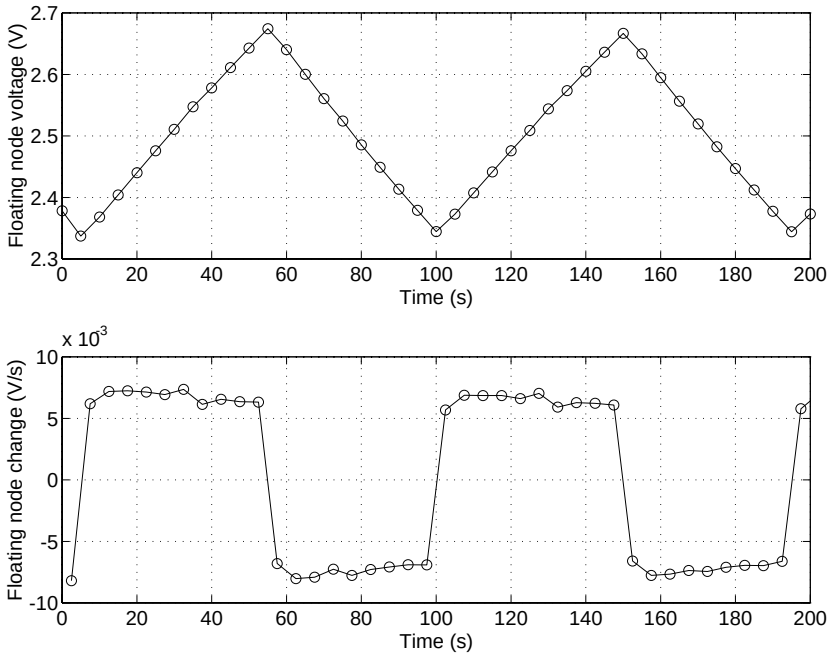


Figure 5.10: Synaptic weight programming characteristics.

In figure 5.10 and 5.11 the programming characteristic and forgetting dynamics of of the memory is shown. The floating node voltage as it is programmed up and down a couple of times, and its derivative is plotted. As we can see there is no fast floating node voltage *shift* when the programming changes sign.

The floating node change is approximately $7.5mV/s$ when programming and $175\mu V/s$ when forgetting. That is a difference of about 43, which is a bit better than theoretically calculated.

5.2.3 Controlling the learning rate

In section 3.5.1 it was pointed out that the backpropagation learning rate should be controlled by the memory itself. The learning rate expresses the maximum programming speed of the memory, and it is important that the speed is not to high. The result can be an oscillating network. With the low conductance of the UV-structures it is assumed that oscillation is not likely to occur.

On the other hand it is easy to adjust the weight change rate. If we recall from the introduction to the UV-light memory, the UV-conductance is a function of the UV-light intensity. By adjusting the intensity of the light source the learning rate can be adjusted. Usually it is not possible to adjust the intensity of the light source directly, but for prototype testing the light intensity can be regulated by adjusting the distance between the light source and the induced silicon surface and/or placing a UV-light filter between the light source and the chip. UV-light has a fast decay rate in air, so a small change in the distance between the light source and the chip results in significant change in programming speed.

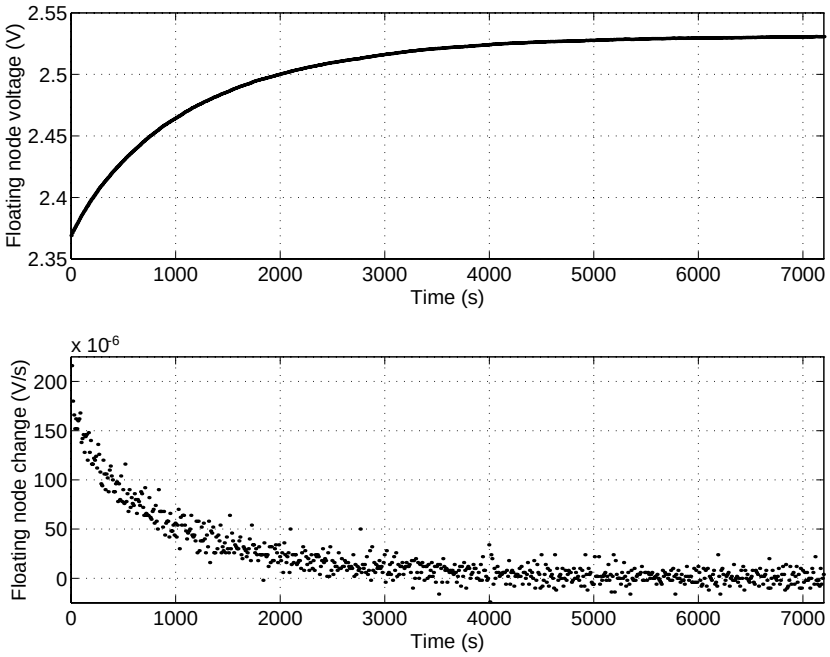


Figure 5.11: Synaptic weight forgetting dynamics.

5.2.4 Synapse layout

In figure 5.12 a layout of a complete synapse with weight, weight change, error propagation and memory circuitry is shown. The leftmost 2/3 of the layout contains the weight programming, storing, and voltage to pulse stream transformation circuitry discussed in this chapter, as the rightmost 1/3 contains the rest of the synaptic circuitry presented in chapter 3.

The UV-structure and the canceling capacitor are almost identical. The only difference is that the canceling capacitor is completely covered with metal2. It was done to ensure that the two capacitances was as equal as one possibly can obtain.

The metal2 top shield is connected to ground, and serves as the ground potential supply distribution 'line'.

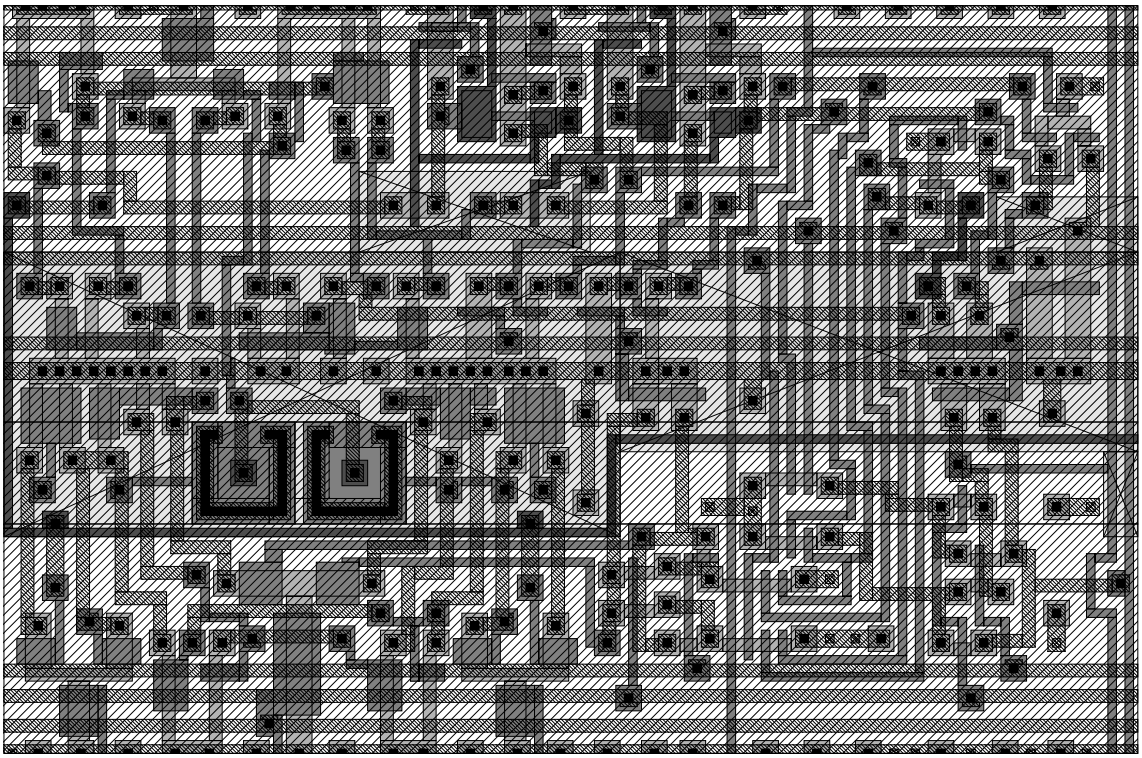


Figure 5.12: Layout of complete synapse circuit: The whole synapse circuit occupies $264\mu m \times 176\mu m$ in a $2\mu m$-process.

CHAPTER

6

Composition and performance of a complete network

Even though the neural network building blocks work perfect in theory, and when tested as single circuits, it is when the different building blocks are assembled into a complete neural network that their real performance can be proven. Unexpected behavior can usually be observed when a circuit is moved from a test environment to its actual environment of operation. This chapter covers the performance of a complete network. Problems that can be solved by the network are presented, as is the limitations of the network implementation.

6.1 Assembling the network parts into a 3-6-2 network

The reason for choosing a network with 3 inputs, 6 hidden neurons and two output neurons was simply that this was the largest network which could be fit on the available silicon area. Of course another network topology could have been chosen, but I considered it important to have a rather large number of hidden neurons relative to the number of inputs, to yield large adaptation dynamics.

During the work with the layout little effort was done to minimize area consumption. A well arranged layout was considered more important than a dense implementation. Simple and symmetrical building blocks that could easily be put together to a complete network was implemented. A well arranged synaptic circuit was most important, since the synaptic weight matrices usually represent most of the circuitry of a complete network.

6.1.1 The threshold synapses

As mentioned in section 2.1.4 the threshold of a neuron can be controlled by a synapse with its input clamped at -1 or 1. Since the outputs of neurons in the stochastic arith-

metic implementation used in this thesis are always positive, the input 'pulse stream' of a threshold synapse should always be on, which implies that it is connected to the positive supply voltage.

Since the input of a threshold synapse is stuck at a constant value there is no need for these synapses to propagate any error. The error propagation circuitry of figure 3.7 can therefore be removed from a threshold synapse.

With the input always stuck at 1 the threshold will also have the potential of getting a larger influence on the total net-input. A change in the threshold weight will result in a change in the net-input of the same quantity. The input of a synapse also control the amount of weight change relative to the error (equation 3.12 and 3.13). For a synapse with the input clamped in an on position the weight will change with a higher rate than for a synapse with the input fluctuating between on and off. The total result is that the threshold have a considerable impact on the total change on the output of a neuron during learning. It can be large enough to make the output totally dominated by the threshold.

To ensure that the threshold do not dominate the net-input, a method to control the threshold learning rate must be introduced[1]. Two possible approaches exists. Either the learning rate can be reduced statically at implementation time by lithographical changes in the UV-structure of the threshold weight, or the possibility of controlling the signal at the o_{pi} node of the weight change circuitry (figure 3.6) must be introduced. The last method was utilized in the implementation presented in this thesis. The o_{pi} node of the *weight* circuit (figure 3.4) was connected to V_{dd}, while the o_{pi} node of the *weight change* circuit (figure 3.6) was controlled by an off chip supplied signal. The learning rate of the threshold synapses can then be reduced by the desirable quantity by applying an appropriate pulse stream signal from off chip.

6.1.2 Complete floorplan

By assembling synapse circuits as shown in figure 5.12 side by side in a row the total dendritic tree of a neuron can be composed. One of the synapses must be of the threshold type. To construct synaptic weight matrices equal rows are placed *back to back* on top of each other as shown in figure 6.1. The back to back arrangement implies that adjacent placed rows are mirrored along the longitudinal axis. The synapses of adjacent rows also share bias voltage lines, so two slightly different synapse cell layouts must be prepared.

At one end of the synapse row the neuron circuit of figure 3.5 is placed. Both ends are equally well suited. The neuron circuits for hidden layers must also incorporate two inverters for the propagated error signals. As we remember it was the inverted signals that were generated.

In feed-forward networks the input neurons are only mapping the input value to the appropriate signal representation. There is no change in the input magnitude. With pulse stream signals as inputs it is not necessary to implement any circuitry to form the input neurons. The input neurons are therefore not represented by any circuitry, only by the wires that the input signals are connected to, which results in the fact that no error propagation circuitry is necessary in the synapses of the hidden layer.

[1]The input pulse stream of a threshold synapse must still be constantly on, to ensure that the weighted input (both positively and negatively) can have a probability of being on over the whole range from 0–1.

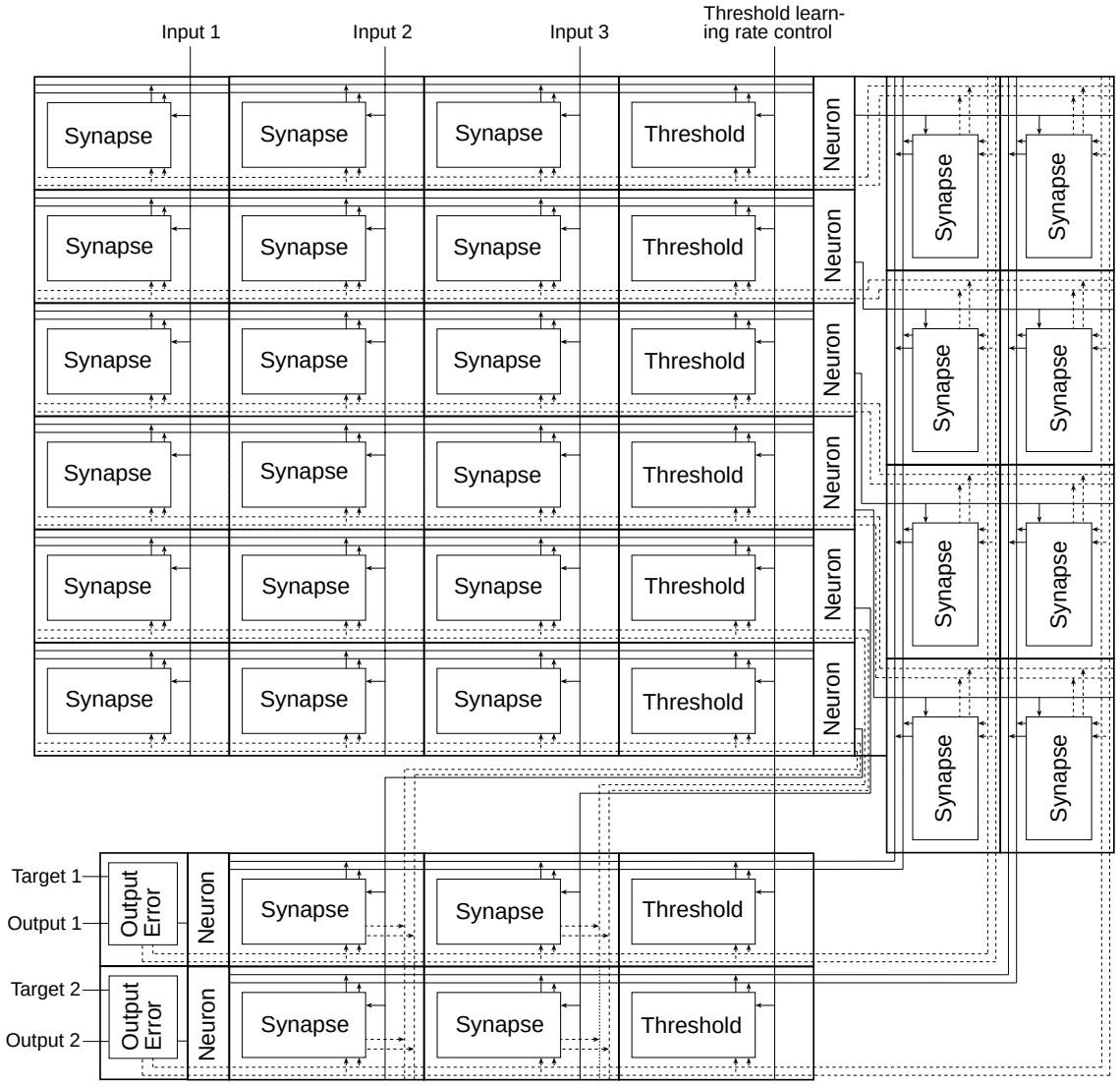


Figure 6.1: Floorplan of a 3-6-2 network: The complete network covers an area of approximately $1500\mu m \times 1500\mu m$ and consists of about 3400 transistors. Solid lines show feed-forward signal flow, while dashed lines show backpropagated error signals.

The arrangement of the hidden layer in relation to the output layer is straight forward. One layer is rotated 90 degrees relative to the other, and the layers are placed adjacent to each other. A complete floorplan of the circuit layout is shown in figure 6.1. Since the synapse cells are not totally square sized, some extra space must be used for wiring inputs, outputs and error lines between the hidden and output layers together. To fit the output layer onto the chip die it had to be bent around the *corner* of the hidden layer weight matrix as shown in the floorplan. The actual appearance of the network is shown in the die photo in appendix A.

6.1.3 Network time constants

The backpropagation learning algorithm was in the first place developed for a discrete behavior. The feed-forward computation, and the backpropagation of error values, with weight adaptation, were supposed to be performed in discrete time steps. When this algorithm is extended to be used in a continuous system one must make sure that the different time constants embedded in the learning process are well related to each other. For proper learning to take place there are some restrictions on the choice of time constants. It is already pointed out that the weight change time constant, that is the learning rate, must be large compared to the other time constants, to keep the network from oscillating. A slow learning rate is just as important in a discrete learning approach as in a continuous one, but in the continuous domain several other time constants must be chosen correctly. [Pineda, 1988] sums it up in the equation:

$$\frac{\tau_w}{M} \gg \tau_P \gg \tau_{bp} \gg \tau_{ff} \ , \tag{6.1}$$

where τ_w is the weight change time constant, M is the number of patterns in the problem, τ_P is the characteristic time constant over which input and target patterns fluctuate, τ_{bp} is the error propagation time constant and τ_{ff} is the feed-forward time constant. It implies that the delay in the feed-forward computation should be shorter than the delay in the backpropagation computation. The backpropagation computation on the other hand must of course be much faster than the time between pattern changes, which again must be faster than the learning rate divided by the number of patterns in the problem presented. The last relation ensures that the network do not learn each pattern trivially as the subsequent pattern is forgotten.

The delay of the feed-forward computation in the network is only a few micro seconds, and with the delay of the integrator in the output error generation circuit it should be clear that the time constant of the error propagation is much larger than the feed-forward computation. For the appropriate integrator bias it is in the range of milliseconds. The delay between pattern changes can be selected before learning takes place. The delay depends on the number of patterns and the size of the weight change time constant. From the plot in figure 5.10 and the region of operation of the comparator used in the floating gate voltage to weight pulse stream conversion, it can be concluded that the weight change time constant at a minimum is in the range of 10–15 seconds. For the 3-6-2 network a maximum of eight binary patterns are possible, so the weight change time constant may be a bit to short for some problems. It can therefore be necessary to reduce UV-light intensity for a proper learning to take place.

6.2 Pattern presentation and training algorithm

The backpropagation of error learning algorithm is as stated before based on an iterative presentation of all the input and target pattern pairs in a sequential order. A presentation of all the pattern pairs is called an epoch. The total output error can be calculated after each epoch on the basis of the output error for each pattern. The *mean square error* – *MSE* – for each output is calculated by the equation:

$$MSE_j = \frac{1}{M}\left(\sum_p \left(t_{pj} - o_{pj}\right)^2\right) \cdot 100\% \ ,$$

which is a variation of the gradient decent error measure, with the only difference being that the total error for the output is expressed as a percentage of the maximum possible error. M represents the number of patterns presented.

If the patterns are presented in the same order for every epoch the learning algorithm is subject to the local minima limitation of backpropagation. Presenting the patterns in a random order on the other hand will expose the network for a sequence of random forces with a built in mechanism of helping the network climb out of a local minima [Pineda, 1988]. The randomness introduces noise to the system, and noise is as we already know helpful for the learning process.

Taking these reflections into consideration, a simple learning control routine can be expressed as:

```
/* Initialize the synaptic weights */

<set the programming comparator bias (V_bt) to 0V>
<turn on the UV-light source>
<wait a predefined period of time to let the floating >
<gate voltages approach the reference voltage>
<turn off the UV-light source>

<set the programming comparator bias to an appropriate value>
<turn on the UV-light source>

i = 0;
finished = false;

/* Present the patterns until the problem is learned or exhaustion occur */

while not finished and i < maxEpochs do
begin
    patternSet = <all patterns in the training set>

    for the number of elements in patternSet do
    begin
        <select a pattern from patternSet at random>
        <present the pattern for a predefined amount of time>
        <measure the output response for all outputs>
        <discard the pattern presented from patternSet>
    end

    <calculate MSE>

    if MSE < maximum tolerated error then
        finished = true;
    else
        i = i + 1;
```

Pattern	I_1	I_2	I_3	Target
Pattern 1	0	0	0	0
Pattern 2	0	0	1	1
Pattern 3	0	1	0	1
Pattern 4	0	1	1	0
Pattern 5	1	0	0	1
Pattern 6	1	0	1	0
Pattern 7	1	1	0	0
Pattern 8	1	1	1	1

Table 6.1: Truth table for the 3-parity problem.

end

<turn off the UV-light source>

Since the stochastic pulse streams representing the network outputs appears as random noise, the output signals probability of being on could not be measured by the universal counter used in the previous duty cycle measurements. The moving average of the pulse stream outputs was generated with an off-chip operational amplifier integrator. The integrator was implemented to reflect the duty cycle as a voltage between 0 and 5 volt. For more details about the moving average op-amp integrator see appendix B.2.

6.3 Network performance for the 3-parity problem

To benchmark the performance of the 3-6-2 network it was set up to solve the 3-parity problem, which is the double XOR-function:

$$O = I_1 \oplus I_2 \oplus I_3 \, . \tag{6.2}$$

Equation 6.2 is the feed-forward network benchmark XOR-problem generalized to three inputs. The problem is therefore of course linearly inseparable and a network with hidden neurons are necessary to solve it. For the problem the output should be on if an odd number of inputs are on, and off for an even number of inputs on. The truth table for equation 6.2 is shown in table 6.1.

For the attempt to teach the 3-6-2 network the 3-parity problem the off-chip programming parameters shown in table 6.2 was used. The training algorithm presented was then executed on a workstation controlling all the measuring instruments involved in the process. For this particular measurement the programming algorithm was slightly changed. The pattern presentation loop was modified to keep on programming until exhaustion even if the network gave a sufficiently good response. The exhaustion time was set to 600 epochs. The mean square error was also calculated separately for each of the outputs, and the result is shown in figure 6.2.

As seen the mean square error is not slowly reduced as the learning progress. At first the error is getting larger, before it goes into a period of fast changing in both directions. For the last half of the learning period the error very slowly progress to the better. The

Threshold learning reduction factor	UV-light source distance	UV-Filter	Presentation time for each pattern
1	3 cm	Yes	10s

Table 6.2: Of-chip training parameters for the 3-parity problem.

reason for the very fast changes of the error is not easy to explain. It may be that the learning rate is to large, and the network is forgetting what it has learned right after a major progression has been achieved. On the other hand the changes are so fast that this seems unlikely. Another explanation can be environmental noise that is wrecking the measurement.

With a mean square error of 20% at the end of the learning phase it was not assumed that a retrieval of the output response to the inputs would show any good results. It was therefore quite a surprise that one of the outputs responded with something that could be called half a success. Presenting the input patterns together with the target patterns gave an output response that was almost correct. The only failure being that the output integrator value is fluctuating with 0.5 volt around 1 volt, and not close to 2.5 volt around 2.5 volt. It implies that the network has not be capable of adapting the weights to give a maximum attainable difference for the binary output values. On the other hand it may be a bit too ambitious to believe that this is possible. The interesting point is that the output values are not at all difficult to distinguish.

A measurement of the output response, not presenting any target values was not promising. As we can see from the lower plot of figure 6.2 the result was a total failure. The reason for this may be traced back to the memory devices. Even though the programming circuitry was carefully designed to prevent the floating node from being disturbed by control node voltage switches, it is apparent that the network response is very much dependent on the control node switching in the same way during retrieval as during the end of the learning process. If on the other hand the network had been capable of gaining output values which had resulted in no propagation of error signals at the end of the learning process, the response would probably had been correct. With correct outputs the error signal lines would have been off and the control node stable.

From a practical point of view it is unacceptable that the solution to a problem must be presented together with a problem to make the network respond correctly. If you have got the answer to a problem you do not need a neural network to present it for you. On the other hand the result is quite interesting. It implies that the network is capable of learning even hard linearly inseparable problems, and further attempts to train the network should be performed. Sadly enough time did not allow further measurements to be accomplished, so the presented results are all that is available at the current time.

6.4 Improvements

During the network performance measurements it was observed that it was a difficult task to match the relationship between the follower bias (V_{bf}) and the programming comparator bias (V_{bt}) in the synaptic memory. Due to transistor mismatch it was difficult to get a sufficient number of synaptic memories to function correctly. Only small

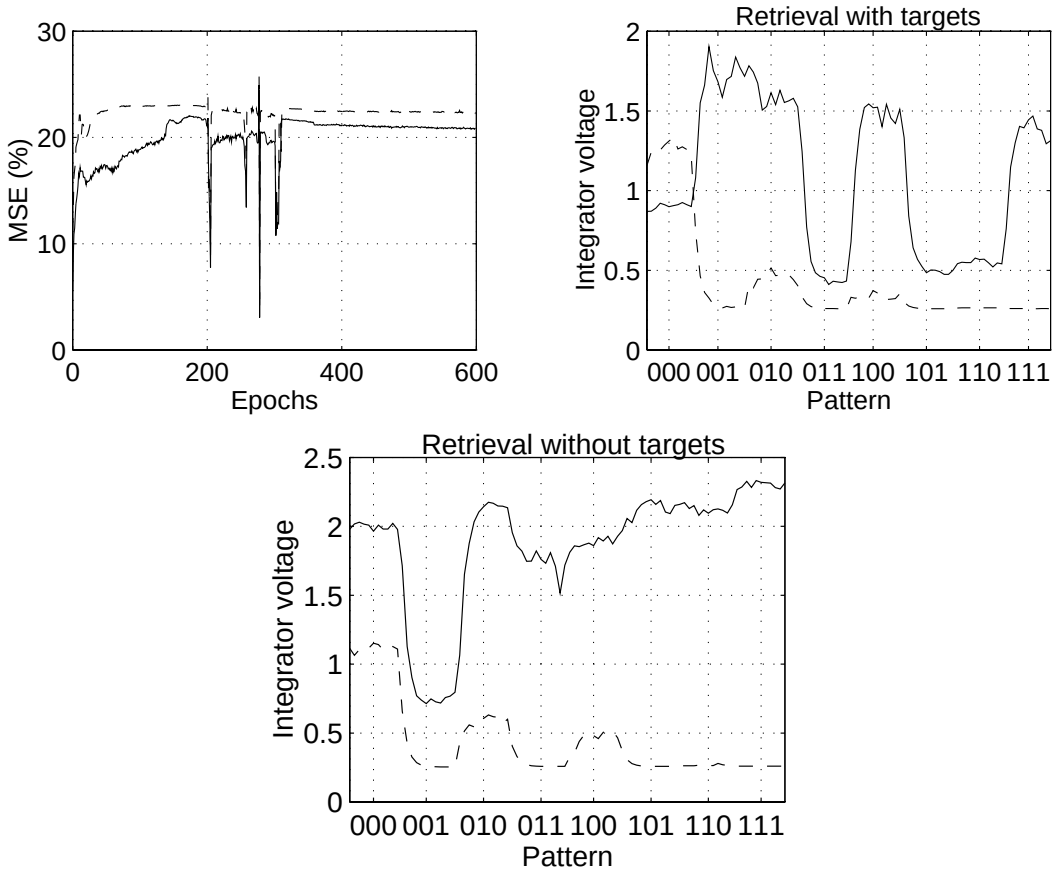


Figure 6.2: Results from the 3-parity learning process: The upper left plot shows the mean square error, while the right shows output retrieval with targets presented together with the inputs. The lower plot shows retrieval without target presentation. Solid lines are the results from output 1, while the dashed lines are the results from output 2.

variations in the bias voltages resulted in either that the followers overpowered the programming comparator, or the programming comparator overpowered one or both of the followers. The first of course results in a inability to program the floating node, as the latter imply that the control node is kept at one diode voltage above ground (see figure 5.8) when no programming is supposed to occur. For the last case the total result is that the synaptic weight is pulled maximally low. Improvements that reduces the transistor mismatch must therefore be performed to increase the number of synapses that functions correctly.

Further, an investigation of aspects that can improve the symmetry of the programming comparator should be preformed. The basis for a stable floating node is the fact that the inverting and non-inverting control node must have a symmetrical behavior. To achieve the symmetry the programming comparator and it's interaction with the followers must be improved. It was not difficult to achieve the necessary symmetry for a single cuircuit, but as we have seen this does not imply that it will work out for a complete network of synaptic elements.

CHAPTER

7

Conclusion

In the preceding chapters a low-power, fully scalable, feed-forward neural network with in situ learning, utilizing the advantage of pulse stream coding of quantities, has been developed in a standard CMOS process. The different building blocks constituting the different network parts have been implemented and tested, and finally assembled into a complete network with one hidden layer of neurons. The network was set up to learn the linearly inseparable 3-parity problem.

7.1 Conclusion

Taking advantage of the simplicity of arithmetic operations on stochastic pulse streams, very simple and dense circuits to implement the feed-forward computation and the back-propagation of error learning algorithm was developed. Even complex multiplications and summations are performed with a minimum number of transistors, and nearly all of the computational power embedded in both the feed-forward and backpropagation of error was implemented locally in each synapse circuit, ensuring a minimum of wiring and the possibility to implement very dense networks. An analog integration and re-generation circuit for output error generation was also implemented.

Further a relatively compact synaptic weight storage circuit[1] was developed. The memory make use of floating-gate UV-programmable analog memory devices for long term storage of synaptic weights, which introduces the rather infrequent combination of analog storage and 'digital' (stochastic arithmetic) computational elements. All the analog circuits were tested and their behavior was proven to be as expected.

All building blocks are developed in a low-power fashion, where subthreshold biased transistors limit the static currents flowing in the circuits. For typical bias voltages the complete network consumes power in the range of a few microwatts, which ensure full scalability.

The network building blocks were then assembled into a complete 3-6-2 network with a potential to solve the 3-parity problem. The training of the network was a partial success, but due to the networks incapability to converge to almost exact correct

[1] Compared to other possible implementation schemes.

response, and the fact that the synaptic memories did not work as good in a complete network context as opposed to their behavior when tested as single circuits, a successful training was not achievable. The network was capable of learning the 3-parity problem, but a correct response was only achieved under exactly the same conditions as used during training, which implies that the target patterns had to be presented together with the input patterns during the retrieval phase.

There should be no question about the network implementation's capability of solving even hard inseparable problems. The network concept is therefore in principle correct, but the implementation should be further improved. It can well be that only marginal corrections is necessary to gain large improvement in learning.

7.2 Further work

Further work of course include improvements of the memory circuit. An implementation that is even less responsive to noise introduced by control node switching should be developed. It is assumed that this may improve learning considerably. Methods to improve learning speed should also be investigated. As it stands now the time to train the network is too long, and only for implementations with very large networks, the programming time may compete with the more common off-chip learning schemes. The employment of variations of the backpropagation of error learning algorithm and the use of higher pulse frequencies can improve the network learning speed.

Bibliography

[Abusland, 1994] Aanen Abusland. A CMOS Analog Hopfield Net with Local Adaption and Storage of Weights. Main Subject Thesis (masters thesis), University of Oslo, Department of Informatics, 1994.

[Andreou *et al.*, 1991] Andreas G. Andreou, Kwabena A. Boahen, Philippe O. Pouliquen, Aleksandra Pavasović, Robert E. Jenkins, and Kim Strohbehn. Current-Mode Subthreshold MOS Circuits for Analog VLSI Neural Systems. *IEEE Transactions on Neural Networks*, 2:205–213, 1991.

[Banzhaf, 1988] W. Banzhaf. On a Simple Stochastic Neuron-Like Unit. *Biological Cybernetics*, 60:153–160, 1988.

[Benson and Kerns, 1993] Ronald G. Benson and Douglas A. Kerns. UV-Activated Conductances Allow For Multiple Time Scale Learning. *IEEE Transactions on Neural Networks*, 4(3):434–440, May 1993.

[Carley, 1989] L. Richard Carley. Trimming Analog Circuits Using Floating-Gate Analog MOS Memory. *IEEE Journal of Solid-State Circuits*, 24(6):1569–1575, December 1989.

[Concannon *et al.*, 1993] A. Concannon, S. Keeney, A. Mathewson, R. Bez, and C. Lombardi. Two-Dimensional Numerical Analysis of Floating-Gate EEPROM Devices. *IEEE Transactions on Electronic Devices*, 40(7):1258–1262, July 1993.

[Dickson *et al.*, 1993] Jeffery A. Dickson, Robert D. McLeod, and Howard C. Card. Stochastic Arithmetic Implementations of Neural Networks with In Situ Learning. *IEEE International Conference on Neural Networks*, 2:711–716, 1993.

[Eguchi *et al.*, 1991] H. Eguchi, T. Furuta, H. Horiguchi, S. Oteki, and T. Kitaguchi. Neural network LSI chip with on-chip learning. In *Proceedings of International Joint Conference on Neural Networks*, volume 1, pages 453–456, New York, 1991. IEEE.

[Foty and Nowak, 1994] Daniel P. Foty and Edward J. Nowak. MOSFET Technology for Low-Voltage/Low-Power Application. *IEEE Micro*, 14(3):68–77, June 1994.

[Gaines, 1969] B. R. Gaines. Stochastic Computing Systems. In Julius T. Tou, editor, *Advances in Information Systems Science*, volume 2, chapter 2, pages 37–172. Plenum Press, New York, 1969.

[Hertz *et al.*, 1991] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the theory of neural computation*. Computation and Neural Systems Series. Addison-Wesley, 1991.

[Hopfield, 1982] J.J. Hopfield. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. In *Proceedings of the National Academy of Sciences*, pages 2554–2558, 1982.

[Hornik *et al.*, 1989] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, 2:359–366, 1989.

[Kandel and Schwartz, 1985] E.R Kandel and J.H. Schwartz. *Principles of Neural Science*. Elsevier, New York, 2nd edition, 1985.

[Kerns *et al.*, 1991] D.A. Kerns, J. Tanner, M. Sivilotti, and J. Jou. CMOS UV-writable Non-Volatile Analog Storage. In C.H. Séquin, editor, *Advanced Researches in VLSI*, pages 245–261. MIT Press, Cambridge, MA, University of California Santa Cruz, 1991.

[Lazzaro and Mead, 1989a] John Lazzaro and Carver Mead. Circuit models of sensory transduction in the cochlea. In Carver Mead and Mohammed Ismail, editors, *Analog VLSI Implementations of Neural Systems*, pages 85–101. Kluwer Academic Publishers, Boston,MA, 1989.

[Lazzaro and Mead, 1989b] John Lazzaro and Carver Mead. A Silicon Model Of Auditory Localization. *Neural Computation*, 1:47–57, 1989.

[Lazzaro and Mead, 1989c] John Lazzaro and Carver Mead. Silicon modeling of pitch perception. *Proc. Natl. Acad. Sci. USA*, 86:9597–9601, 1989.

[Lazzaro, 1991a] John Lazzaro. Biologically-based auditory signal processing in analog VLSI. *IEEE Asilomar Conference on Signals, Systems, and Computers*, 1991.

[Lazzaro, 1991b] John Lazzaro. A Silicon Model of an Auditory Neural Representation of Spectral Shape. *IEEE Journal of Solid-State Circuits*, 26:772–777, 1991.

[Lazzaro, 1992a] John Lazzaro. Low-power Silicon Spiking Neurons and Axons. *IEEE International Symposium on Circuits and Systems*, 5:2220–2223, 1992.

[Lazzaro, 1992b] John Lazzaro. Temporal Adaptation in a Silicon Auditory Nerve. In J. Moody, S. Hanson, and D. Tourestzky, editors, *Advances in Neural Information Processing Systems*, volume 4. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[Lenzlinger and Snow, 1969] M. Lenzlinger and E.H. Snow. Fowler-Nordheim Tunneling into Thermally Grown SiO_2. *Jornal of Applied Physics*, 40(1):278–283, January 1969.

[Maher, Unpub] M.A. Maher. New UV-Memory writing scheme. Unpublished.

[McCulloch and Pitts, 1943] W.S. McCulloch and W. Pitts. A Logical Calculus of Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, pages 115–133, 1943.

[Mead, 1989a] C. Mead. Adaptive retina. In C. Mead and M. Ismail, editors, *Analog VLSI Implementation of Neural Systems*, pages 239–246. Kluwer Academic Publisher, Boston, MA, 1989.

[Mead, 1989b] Carver Mead. *Analog VLSI and Neural Systems*. VLSI System Series and Computation and Neural Systems Series. Addison-Wesley, 1989.

[Meador *et al.*, 1991] Jack L. Meador, Angus Wu, Clint Cole, Novat Nintunze, and Pichet Chintrakulchai. Programmable Impulse Neural Circuits. *IEEE Transactions on Neural Networks*, 2:101–109, 1991.

[Murray and Tarassenko, 1994] Alan Murray and Lionel Tarassenko. *Analogue Neural VLSI. A pulse stream approach*. Neural Computing Series. Chapman & Hall, Boundary Row, London, 1994.

[Pasero, 1991] Eros Pasero. Floating gates as adaptive weights for artificial neural networks. In M. Sami and J. Calzadilla-Daguerre, editors, *Silicon Architectures for Neural Networks*, pages 125–135. Elsevier Science Publishers, B.V. (North-Holland), 1991.

[Pineda, 1988] Fernando J. Pineda. Dynamics and Architecture for Neural Computation. *Journal of Complexity*, 4:216–245, 1988.

[Ribeiro, 1967] S. Ribeiro. Random-pulse machines. *IEEE Transactions on Electronic Computers*, 16(3):261–276, June 1967.

[Rumelhart *et al.*, 1986] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Internal Representation by Error Propagation. In *Parallel Distributed Processing*, chapter 8, pages 318–362. MIT Press, 1986.

[Säckinger and Guggenbühl, 1988] Eduard Säckinger and Walter Guggenbühl. An Analog Trimming Circuit Based on a Floating-Gate Device. *IEEE Journal of Solid-State Circuits*, 23(6):1437–1440, December 1988.

[Shepherd, 1979] Gordon M. Shepherd. *The Synaptic Organization of the Brain*. Oxford University Press, New York, 2nd edition, 1979.

[Sin *et al.*, 1992] Chi-Kai Sin, Alan Kramer, V. Hu, Robert R. Chu, and Ping K. Ko. EEPROM as an Analog Storage Device, with Particular Application in Neural Networks. *IEEE Transaction on Electronic Devices*, 39(6):1410–1418, June 1992.

[Soelberg *et al.*, 1994] Knut Soelberg, Roy Ludvig Sigvartsen, Tor Sverre Lande, and Yngvar Berg. An Analog Continuous-Time Neural Network. *Analog Integrated Circuits and Signal Processing*, 5:235–246, 1994.

[Sze, 1981] S.M. Sze. *Physics of Semiconductor Devices*. Wiley, 2nd edition, 1981.

[Thomsen and Brooke, 1990] Axel Thomsen and Martin A. Brooke. A floating-gate MOSFET with tunneling injector fabricated using a standard double polysilicon CMOS-process. Technical Report 90–02, Analog Microelectronics Group, Department of Electrical Engineering, Georgia Institute of Technology, Atlanta, 1990.

[Tomlinson and Walker, 1990] Max Stanford Tomlinson and Dennis J. Walker. DNNA: A Digital Neural Network Architecture. *INNC 90 Paris. International Neural Network Conference*, 2:589–592, 1990.

[Tomlinson *et al.*, 1990] Max Stanford Tomlinson, Dennis J. Walker, and Massimo A. Sivilotti. A Digital Neural Network Architecture for VLSI. In *Proceedings of International Joint Conference on Neural Networks*, volume 2, pages 545–550, New York, 1990. IEEE.

[Vittoz *et al.*, 1991] E. Vittoz, H. Oguey, M.A. Maher, O. Nys, E. Dijkstra, and M. Chevroulet. Analog storage of adjustable synaptic weights. In U. Ramacher and U. Rückert, editors, *VLSI Design of Neural Networks*, pages 47–63. Kluwer Academic, Boston, MA, 1991.

[von Lehmen *et al.*, 1988] A. von Lehmen, E. G. Paek, P. F. Liao, A. Marrakchi, and J. S. Patel. Factors Influencing Learning by Backpropagation. *IEEE International Conference on Neural Networks*, pages 335–341, 1988.

[von Neumann, 1956] J. von Neumann. Probabilistic logics and synthesis of reliable organisms from unreliable components. In C. E. Shannon, editor, *Automata Studies*, pages 43–98. Princeton University Press, Princeton, NJ, 1956.

[Watts *et al.*, 1992] Loyd Watts, Douglas A. Kerns, Richard F. Lyon, and Carver A. Mead. Improved Implementation of the Silicon Cochlea. *IEEE Journal of Solid State Circuits*, 27(5):692–700, 1992.

[Williams, 1965] Richard Williams. Photoemission of Electrons from Silicon into Silicon Dioxide. *Physical Review*, 140(2A):A569–A575, October 1965.

[Witters *et al.*, 1989] Johan S. Witters, Guido Groeseneken, and Herman E. Maes. Degradation of Tunnel-Oxide Floating-Gate EEPROM Devices and the Correlation with High Field-Current-Induced Degradation of Thin Gate Oxides. *IEEE Transactions on Electronic Devices*, 36(9):1663–1682, September 1989.

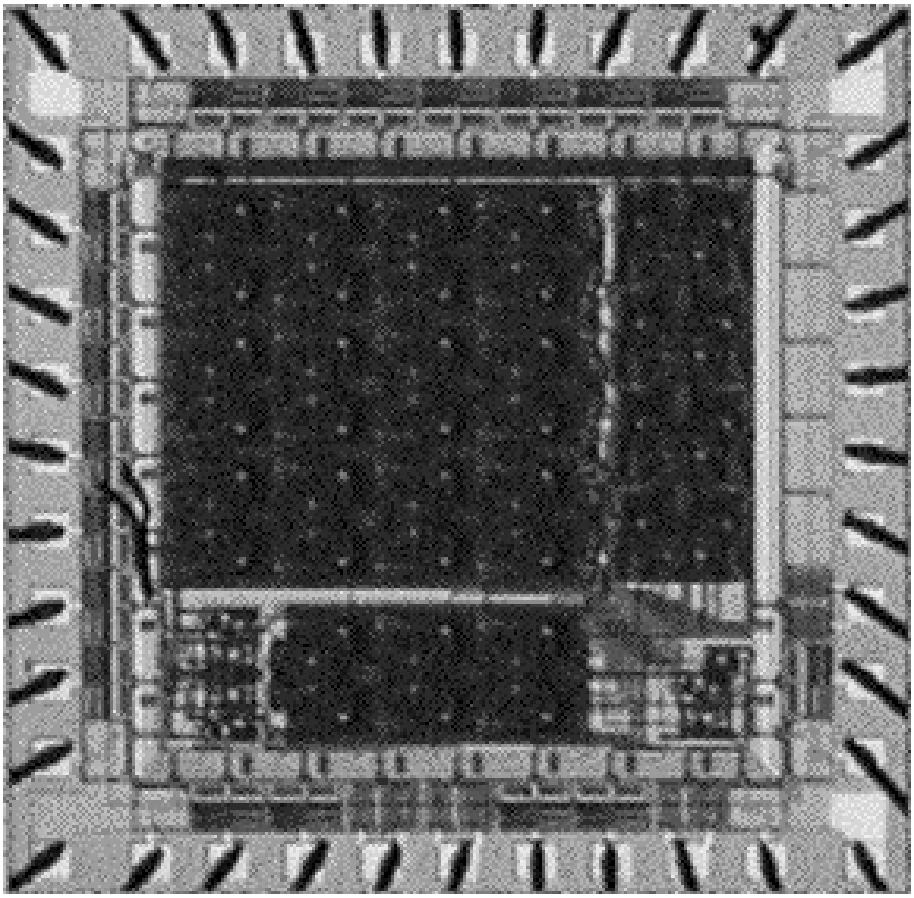
[Yang *et al.*, 1992] Han Yang, Bing J. Sheu, and Ji-Chen Lee. A Nonvolatile Analog Neural Memory Using Floating-Gate MOS Transistors. *Analog Integrated Circuits and Signal Processing*, 2:19–25, 1992.

APPENDIX

A

Die photo of the chip

A die photo of the chip containing the 3-6-2 feed-forward network is shown below. The total size of the chip is $2225\mu m \times 2220\mu m$, where the network consumes a total area of approximately $1500\mu m \times 1500\mu m$. The rest of the space is occupied by a standard TinyChip pad frame. The total number of transistors for the complete network is 3400.



APPENDIX

B

Miscellaneous

This chapter contains some material that did not fit in other parts of the text. The program used to calculate the neuron transfer curves in figure 3.5 (b) is listed, and the op-amp integrator used to measure the network outputs is also presented

B.1 Program for empirical calculation of the neuron transfer characteristic

```
/* A small C program that calculates the transfer characteristics for  */
/* neurons in a stochastic arithmetic neural network.                  */

/* Compile command: gcc average.c -o average -lm                       */

/* The format of the output file is matlab ASCII matrix format.        */

#include <math.h>
#include <stdio.h>
#include <time.h>

int i, j, k, it, numw;
float w, pnp, pnn, x[301], y[301];
FILE *F;
char fname[31];

void main() {

  for(i=0;i<301;i++) x[i]=y[i]=0.0;
  for(i=150;i<301;i++) y[i] = 1.0;

  printf("Enter number of iterations (>100000) : ");
  scanf("%d", &it);
```

```
  printf("Number of synaptic weights > ");
  scanf("%d", &numw);
  printf("Output file name (max 30 characters) > ");
  scanf("%s", fname);

  /* Calculate the average for it different possibilities          */

  for(i=0;i<it;i++) {

    /* numw weighted inputs to the neuron.                         */

    pnn = pnp = 0.0;

    for(j=0;j<numw;j++) {

      /* Calculate the weighted input with a probability of on in the  */
      /* intervall [0,1], with 3 decimal accuracy. A negative value    */
      /* yields a negative (inhibitory) weight.                        */

      w = ((float)((rand() % 301) - 150))/150.0;

      if(w < 0.0) pnn += w;
      else pnp += w;
    }

    k = (int)((pnp + pnn + 15.0)*10.0 + ((pnp+pnn>0.0)?0.5:-0.5));

    if(x[k] == 0.0)
      y[k] = (1.0 - (float)exp(-pnp))*exp(pnn);
    else
      y[k] += (1.0 - (float)exp(-pnp))*exp(pnn);

    x[k] += 1.0;
  }

  /* Flush out the result.                                         */

  F = fopen(fname ,"w");

  for(j=0;j<301;j++)
    if(x[j] == 0.0)
      fprintf(F,"%2.3f  %e\n",((float)(j-150))/(100.0/((float)numw)),y[j]);
    else
      fprintf(F,"%2.3f  %e\n",((float)(j-150))/(100.0/((float)numw)),y[j]/x[j]);
}
```

B.2 Op-amp integrator that implements a moving average

To generate the moving time integration of the stochastic pulse streams on the output of the network a circuit as shown in figure B.1 was supplied off-chip. The circuit is a standard operational amplifier integrator with an extra resistance in parallel with the integrator capacitance. The amplifier connection yields a unity gain, and the resistance and capacitance values were selected to give an integration time constant of 1 second. The negative feedback results in an output of 0 volt for an input where the signal is on all the time, and 5 volt output a signal that is off all the time, for on and off voltages of 5 and 0 volt respectively. For intermediate signals the circuit reflects the mean duty cycle for the last second. In the results presented in chapter 6 the integrator voltages are inverted, that is the presented integrator voltage is

$$5 - \textit{actual integrator voltage}\ .$$

This was done to ensure as simple presentation of the results as possible.

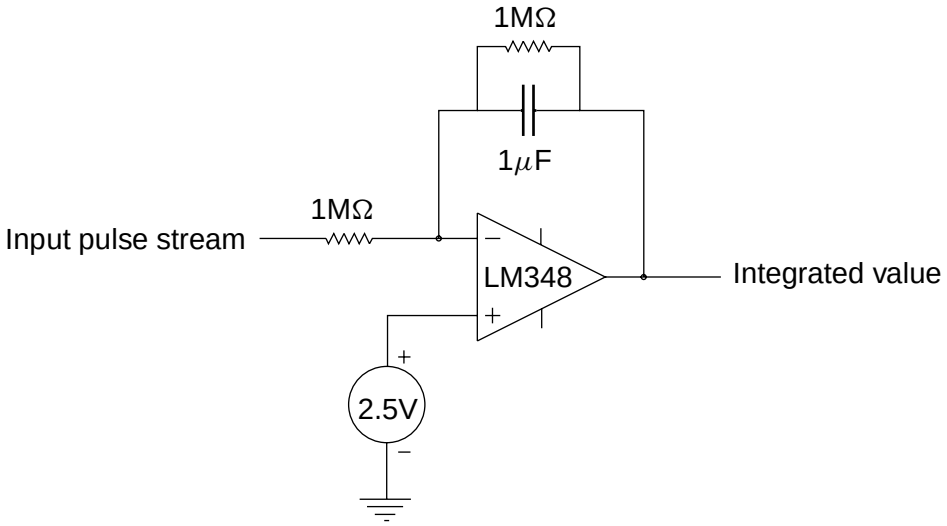


Figure B.1: Operational amplifier integrator.