# Analog VLSI implementation of a feed-forward neural net

Oosse, J.M.C.

*Award date:*
1994

# Faculteit Electrotechniek

Technische Universiteit
Eindhoven

# Analog VLSI Implementation of a Feed-Forward Neural Net

Martin Oosse
June 1994

Electronic Circuit Design Group (EEB)
Faculty of Electrical Engineering
Eindhoven University of Technology

| | |
|---|---|
| Supervisor | Prof. dr. ir. W.M.G. van Bokhoven |
| Coach | Dr. ir. J.A. Hegt |
| Period | September 1993 - June 1994 |

# Abstract

This thesis discusses the implementation of a feed-forward neural network using dense analog circuits. For reasons of flexibility a seperate synapse and neuron chip were made, such that an arbitrary topology of a Multi-Layer Perceptron can be created. The synapse chip contains the weights and multipliers of the neural network and the neuron chip contains the neurons which perform a nonlinear operation on their input data. Each circuit part is discussed exhaustively. After a theoretical analysis, both simulation and measurement results are given.

To train the network a lot of algorithms can be used. This thesis presents two possible algorithms: Weight Perturbation and Parallel Stochastic Weight Perturbation. The latter is preferred, because this algorithm is easier to implement and much faster than the first one.

As a result of this graduation project, a paper was written, which comprises the import of this report [OWH, 1994]. The complete text of this paper can be found in Appendix D.

# Contents

# Chapter 1

# Introduction

*This chapter provides an introduction to this thesis. In the first section a brief overview of the field of neural networks is given, containing the history of artificial neural networks. Section 1.2, 8.2 contains a description of this graduation project, along with an outline of this thesis. Some recommended literature for more detailed information about neural networks are [Lippmann, 1987], and [Maren, 1990]. An overview of hardware implementations of neural networks is given by Hans Hegt in [Hegt, 1993].*

## 1.1 About Neural Networks

Because of the last decade's enormous progress of the field of artificial neural networks, it seems to be a recent development of modern science. However, modelling the brain already has a quite long history of about one century, in which one can distinguish three periodes: after a stage of early enthousiasm a lack of interest developped, because some fundamental limitations were discovered. After these limitations were overcome, the most recent period a real Neural Boom got going.

It all started in the Nineteenth Century when William James [Maren, 1990] wrote: *„When two brain processes are active together or in immidiate succession, one of them, on re-occurring tends to propagate its excitement into the other."* He was the first one who defined a neuronal process of learning. Inspired by James' work, the real field of artificial neural networks began in the 1940's. Since 1956 neural networks, along with some other processing and computer techniques, belong to the family of *'Artificial Intelligence'*. All of the techniques in this branch have one thing in common: they do not only calculate their results, but they are also able to 'learn' them.

Neural networks have this learning ability due to their 'brain-like' structure. A lot of simple processing units are densely connected together. In 1958 Frank Rosenblatt, a psychologist, developed and designed such a structure and called it *'Perceptron'*. The Perceptron was composed of two layers and was able to classify patterns.

A few years later (1960) Widrow and Hoff introduced their adaptive, learning processing unit: the *'ADALINE'*, which is trained using a more complex learning rule than the Perceptron technique. This rule is known as the *'Least-Mean-Squared (LMS)'* learning rule, still in use in todays adaptive systems. At that time researchers often focussed on the Perceptron, but it were Minsky and Papert who demonstrated the weakness of it. They proved that a Perceptron was unable to learn the XOR-problem. Their findings almost meant the end of the field.

In their book [Minsky, 1969] they also stated: *„... our intuitive jugdment that the extension (to multilayer systems) is sterile"*, but this assertion later demonstrated to be inadequate. Multilayered networks, using a more powerful learning rule, have solved the XOR-problem and many other nonlinear problems [Rumelhart, 1986].

With this breakthrough we are landed in the modern period, in which a lot of innovative neural network structures and learning rules are developed. One of the most famous artificial neural networks is the so-called *'Multi-Layer Perceptron'* of Rumelhart and McClelland, which will be treated exhaustively in section 2.1.

## 1.2 About This Thesis

The group EEB at the Eindhoven University of Technology (TUE) is working on the electronic implementation of neural networks. Although there are three ways to this, analog, digital, and mixed (analog-digital), they are putting their major effort on the analog implementations. Analog implementations have some slight advantages compared with the digital ones, for example a higher neurons-per-chip density. Chapter 2 gives a more closer look at the implemented network, a Multi-Layer Perceptron, and on the applied learning algorithm, Weight Perturbation. Chapter 3 discusses some general requirements and some specific aspects of a hardware implementation of a neural network.

In this thesis Paul Bruin's work [Bruin, 1993] is continued. His design had some imperfections, which are overcome now. In chapter 4 and 5 the new drafts are worked out. The simulation results of the total network are discussed in chapter 6. In chapter 7 an improved learning algorithm is discussed: Parallel Stochastic Weight Perturbation. This thesis ends with some conclusions and recommendations, which are written down in chapter 8.

# Chapter 2

# Multi-Layer Perceptron and the Weight Perturbation Algorithm

*In this chapter a more closer look at the Multi-Layer Perceptron will be given. The structure of this kind of neural networks will be reviewed in section 2.1. In the other part of this chapter a very straightforward learning algorithm will be described: Weight Perturbation. For further information about both subjects the reader is referred to [Lippmann, 1987] (Multi-Layer Perceptron), and [Jabri, 1992] (Weight Perturbation Algorithm).*

## 2.1 Structure of a Multi-Layer Perceptron

As stated in the introduction, one of the most well-known neural network models is the Multi-Layer Perceptron. The Multi-Layer Perceptron belongs to the family of feed-forward networks, because during operation all information flow is feed-forward. The Multi-Layer Perceptron consists of a set of neurons which are arranged in layers.

Three kinds of layers can be distinguished:

- the input layer, which doesn't contain neurons, but is a vector of input stimuli;
- several (maybe one) hidden layer(s) and
- an output layer.

Each neuron output in a layer is connected fully and only with all neuron inputs of the following layer. These connections are called synapses. Each connection is weighted. So the input of a neuron is the weighted sum of the outputs of the neurons in the previous layer. The layer structure of the Multi-Layer Perceptron is depicted in Figure 2.1a.

A Multi-Layer Perceptron performs a nonlinear operation on its input data, therefore each neuron has a nonlinear transfer function (occasionally, the output layer is composed of linear neurons). This function should be continuous and has to saturate for both infinitely large positive and negative input values.

Often a sigmoid-like[1] function is chosen, for which the nonlinear transfer function of the neuron is given by Eq. (2.1).

$$f(x) = \frac{1}{1 + e^{-\alpha \cdot (x - \theta)}} \tag{2.1}$$

where $\alpha$ = steepness
$\theta$ = bias

In Figure 2.1b the sigmoid-like transfer function of the Multi-Layer Perceptron is depicted.

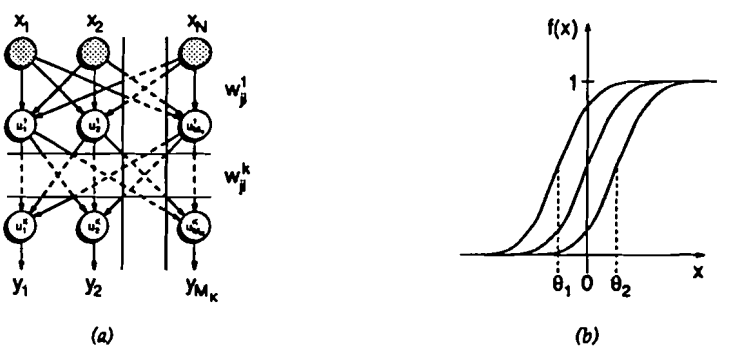



Figure 2.1 *(a) Structure and (b) transfer function of a Multi-Layer Perceptron.*

The outputs of the neurons are now in the 0-1 range. Because the outputs of a certain layer can be the inputs of another layer of neurons, some neuron inputs can become zero. Calculating the weighted sum of the outputs in the previous layer, these inputs are not counted. One can solve this problem by defining the sigmoid-like function between -1 and +1.

---

[1] The term "*sigmoid*" is usually used in reference to monotonically increasing "S-shaped" functions, such as the hyperbolic tangent. In this report, however, the term is generally used for any smooth nonlinear function at the neuron output. Formula (2.1) is just an example of a nonlinear sigmoid shaped transfer function.

The equation now becomes:

$$f(x) = 2\cdot\left(\frac{1}{1 + e^{-\alpha\cdot(x - \theta)}} - \frac{1}{2}\right) = \frac{e^{\frac{\alpha}{2}\cdot(x - \theta)} - e^{-\frac{\alpha}{2}\cdot(x - \theta)}}{e^{\frac{\alpha}{2}\cdot(x - \theta)} + e^{-\frac{\alpha}{2}\cdot(x - \theta)}} = \tanh(\beta\cdot(x - \theta)) \tag{2.2}$$

where $\beta = \alpha/2$ = steepness
$\theta$ = bias

In Figure 2.2 a single neuron with its incoming synapses is drawn.

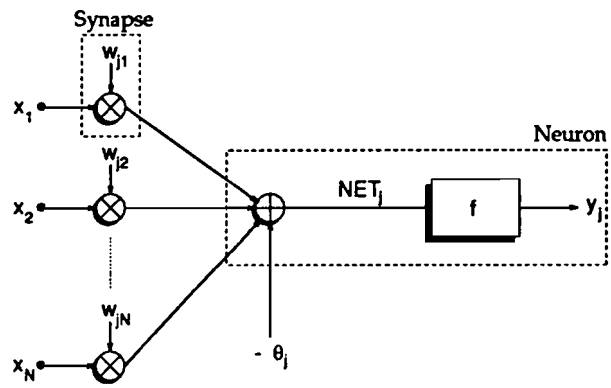



Figure 2.2 *Model of neuron j with N incoming synapses.*

The forward operation of the Multi-Layer Perceptron is as follows: after the inputs (which can be the outputs of another neuron layer) are multiplied with the synapse values (the weights), the outcomes of the multiplications are summed and fed into the neuron, which performs a nonlinear operation at the data. So the output of a single neuron can now be calculated according to Eq. (2.3).

$$NET_j = \sum_{i=1}^{N} w_{ji}\cdot x_i - \theta_j = \sum_{i=0}^{N} w_{ji}\cdot x_i = \sum_{i=0}^{N} net_{ji}$$
$$y_j = f(NET_j) \tag{2.3}$$

where $\theta_j = w_{j0}$ = bias
$x_0 = -1$

Hence, we can look upon offset $\theta_j$ as a synapse with input value -1.

## 2.2 Training with the Weight Perturbation Algorithm

The aim of the Multi-Layer Perceptron is to form a mapping from the input stimuli to a set of output nodes using features extracted from the input pattern. The hidden neurons are trained to respond to these features found in the input. It can be trained by a so-called 'Supervised Learning Algorithm'. These algorithms are called 'Supervised', because one has to 'tell' the network what the output at a particular input pattern should be. A special kind of 'Supervised Learning Algorithms' are 'Reinforced Learning Algorithms'. Now only an error function (a quantative measure of the comparison between the outputs $\underline{y}$ and the correct targets $\underline{d}$) is known, instead of the output and the desired output values itself, and the aim now is to minimize this error function. One of the advantages of using a 'Reinforced Learning' algorithm is the flexibility of chosing different critics. Two possible error functions $E(.)$ are defined in Eq. (2.4) and Eq. (2.5).

$$E(\underline{w}) = \sum_{p=1}^{P} \sum_{j=1}^{M_K} \left(d_{j,p} - y_{j,p}^{K}\right)^2 \tag{2.4}$$

$$E(\underline{w}) = \sum_{p=1}^{P} \max_{j} \left\{ \left| d_{j,p} - y_{j,p}^{L} \right| \right\} \tag{2.5}$$

where $y_{j,p}^{L}$ = actual output from neuron j for pattern p
$d_{j,p}$ = desired output from neuron j for pattern p
$M_L$ = number of neurons in the last layer L
$P$ = number of patterns

Using the error function of Eq. (2.5) the algorithm will find the so-called 'minimax' solution of a particular problem.

One of the algorithms that satisfies the goal of minimizing a certain error function is Weight Perturbation [Jabri, 1992]. This algorithm is chosen for its simplicity of operation and implementation. The weights are updated according to the gradient of an error function, a so-called gradient descend method. The most widely used gradient descent algorithm is the Back Propagation Algorithm, which calculates the weight updates according to the so-called steepest descent principle of Eq. (2.6).

$$\Delta w_{ji} = -\eta \cdot \frac{\partial E}{\partial w_{ji}} \tag{2.6}$$

where $\eta$ = learning rate
$E$ = error function (Eq. (2.4), (2.5) or another one).

The new weights' value will be calculated as:

$$w_{ji}^{new} = w_{ji}^{old} + \Delta w_{ji} \tag{2.7}$$

The Weight Perturbation Algorithm will not calculate the exact derivative of $E(\underline{w})$, but it will approximate the gradient. In Eq. (2.8) this approximation is given.

$$\frac{\partial E}{\partial w_{ji}} \approx \frac{E\left(w_{ji} + \delta w_{ji}\right) - E\left(w_{ji}\right)}{\delta w_{ji}} \tag{2.8}$$

If the perturbation $\delta w_{ji}$ is chosen small enough a good approximation of the gradient is obtained and the the updating of the weights is according to:

$$w_{ji}^{new} = w_{ji}^{old} - \eta \cdot \frac{E\left(w_{ji} + \delta w_{ji}\right) - E\left(w_{ji}\right)}{\delta w_{ji}} \tag{2.9}$$

Eq. (2.8) and (2.9) form the Forward Difference Method (*FDM*), resulting in a minimum error of $E_{opt} + O(\delta w^2)$. To get a minimum error which equals the optimum error $E_{opt}$, the Central Difference Method (*CDM*) can be used [Kate, 1993]:

$$\frac{\partial E}{\partial w_{ji}} \approx \frac{E\left(w_{ji} + \delta w_{ji}\right) - E\left(w_{ji} - \delta w_{ji}\right)}{2 \cdot \delta w_{ji}} \tag{2.10}$$

$$w_{ji}^{new} = w_{ji}^{old} - \eta \cdot \frac{E\left(w_{ji} + \delta w_{ji}\right) - E\left(w_{ji} - \delta w_{ji}\right)}{2 \cdot \delta w_{ji}} \tag{2.11}$$

Now the update rules are known, the Weight Perturbation Algorithm can be given:

**Step 1 : Initialize weights and offsets**

To start training all weights $\underline{w}$ and offsets $\underline{\theta}$ are given an initial value. Very often these values are chosen random within a particular range, but more sophisticated methods are also possible. In [Nguyen, 1990] Nguyen en Widrow suggest to choose the initial weights in such a way that none of the hidden neurons is starting in its saturation region. Simulation of this idea showed a faster convergence of most of the problems.

**Step 2 : Present all input vectors $\underline{x}$ and calculate error $E(w_{ji})$**

After presenting all continuous valued input vectors $\underline{x}$ the error $E(w_{ji})$ is calculated using the measured outputs $\underline{y}$ and the corresponding desired outputs $\underline{d}$ according to a certain error function (Eq. (2.4), (2.5) or another one). Instead of presenting the whole training set, where the patterns are presented sequentially, a subset can be presented. This subset can be chosen cyclically or randomly [Kate, 1993].

A forward relaxation of the network to obtain the measured outputs $\underline{y}$ is done in the following way: For the given input vector $\underline{x}$, the output vector of the first layer is calculated. This vector is the input vector of the second layer, and so on until the last layer is reached.Then the actual output is calculated. The output of neuron $j$ of a certain layer $l$ is calculated as (compare with Eq. (2.3)):

$$y_j^k = S\left(net_j^k\right) \qquad j=1,\ 2,\ \ldots,\ M_k \tag{2.12}$$

with

$$net_j^k = \sum_{i=1}^{M_{k-1}} w_{ji}^k \cdot y_i^{k-1} \qquad 1 \le k \le K \tag{2.13}$$

where $y_j^k$ = output of neuron $j$ in layer $k$ and $y_j^0 = x_j$ (input simulus)
$S(.)$ = sigmoid function
$net_j^k$ = activation of neuron $j$ in layer $k$
$w_{ji}^k$ = weight from neuron $i$ in layer $k$-1 to neuron $j$ in layer $k$
$M_k$ = number of neurons in layer $k$ and $M_0 = N$ (number of inputs)

**Step 3 : The error $E(w_{ji} + \delta w_{ji})$ is calculated**

One particular weight $w_{ji}$ is perturbed with an amount of $\delta w_{ji}$, and the error of the network is calculated in the same way as the error $E(w_{ji})$ of step 2.

**Step 4 : Adapt weight**
The weight update $\Delta w_{ji}$ is calculated according to Eq. (2.9), and the weight is updated according to Eq. (2.7).

**Step 5 : Stop criterion**
Repeat the sequence (step 2 to 4) until the error $E(w_{ji})$ is smaller than $\epsilon$, which is the maximum error that is allowed. The selection of the weights can be in increasing order, so start in layer 1 with $w_{11}$ and end in layer $L$ with $w^K_{M_L M_{L-1}}$.

The above mentioned procedure implements the FDM (only perturbing once). The procedure can be adapted for the CDM by adding step 3' and changing step 4:

**Step 3' : The error $E(w_{ji} - \delta w_{ji})$ is calculated**
The same weight which was perturbed at step 3 is now perturbed to the opposite direction with the same amount $\delta w_{ji}$. The network error is also calculated in the same way as the error $E(w_{ji})$ of step 2.

**Step 4 : Adapt weight**
The weight update $\Delta w_{ji}$ is calculated according to Eq. (2.11), and the weight is updated according to Eq. (2.7).

Using the Weight Perturbation Algorithm has a number of advantages [Bruin, 1993; Kate, 1993]:

- simple weight update calculation;
- flexibile with regard to the error definition;
- besides the net itself few additional hardware is needed;
- there is no backward path;
- the algorithm seems to be fairly immune for hardware imperfections.

To train the network many training cycles need to be done, because each weight is updated sequentially. This seems to be the main drawback of the Weight Perturbation Algorithm, but when a hardware implementation of the neural network is made the processing speed can be increased a lot. Another minus is the estimation of the gradient instead of the exact calculation, but this problem can be minimized by using very small perturbations. When perturbations are chosen too small other problems will arise, such as the transformation of small signals to electronic equivalents (signals can be below noise level). Therefore it is recommended to apply the central difference method, because to obtain an acceptable estimation of the gradient the perturbations do not have to be that small [Kate, 1993]. In chapter 7 an alternative Weight Perturbation Algorithm is discussed. Using this algorithm, all weights are updated "in parallel".

# Chapter 3

# Electronic Implementation

*In this chapter a universal approach of implementing a feed-forward neural network in electronics will be presented together with some requirements. Also the global structure of a synapse and a neuron chip is given.*

## 3.1 Requirements

Making an electronic implementation of an artificial neural network the following aspects has to be taken into account [Claassen, 1993]:

| | Aspects | Ideal |
|---|---|---|
| 1. | Power dissipation | Minimal |
| 2. | Synapse and neuron implementation area | Minimal |
| 3. | Number of required pins per chip | Minimal |
| 4. | Response time (speed) | Minimal |
| 5. | Range of input, output and weights' values | Maximal |
| 6. | Susceptibility to parameter variations | Minimal |
| 7. | Susceptibility to noise | Minimal |
| 8. | Flexibility and cascadability | Maximal |

Table 3.1 *Important aspects of electronic implementation.*

Ad. 1. One rule to minimize power dissipation is to make use of a small supply voltage range, which will be 0 to 5 V. It is also recommended to use low currents (in the μA-range).

Ad. 2. The implementation area of one synapse is more important than the chip area of one neuron, because the number of synapses is of order $N^2$, while the number of neurons is only of order $N$. The synapse's chip area is also a significant measure of comparing different hardware implementations of neural nets with each other.

Ad. 3. The number of pins in the experimental stage isn't that important. The more voltages and currents one can adjust, the better. So the experimental chips will contain more pins then the final ones.

Ad. 4. The speed of the implemented neural network is very important, because training the net many operations need to be done and the higher processing speed of the hardware implementation was the main reason of choosing a hardware instead of a software implementation. To make the delay times as short as possible, feedback loops (like buffers) have to be avoided.

Ad. 5. Using wide ranges of all input and output values, smaller quantizations steps can be used when converting the analog signals to digital (discrete) ones. The minimum quantization step $\Delta$ equals:

$$\Delta = \frac{R}{2^N - 1} \tag{3.1}$$

where $R$ = range of the input/output values
$N$ = number of bits used for quantization

The number of quantisationsteps increases with the range.

Ad. 8. Special attention has to be paid to the aspect of flexibility and cascadibility. This requirement has far-reaching consequences for the layout of the neural net chips. An implementation of a Multi-Layer Perceptron namely can result in the following configurations (depending on the application):

1. A Multi-Layer Perceptron with a fixed topology.
2. A Multi-Layer Perceptron with an arbitrary number of layers, but a maximum number of neurons per layer.
3. A Multi-Layer Perceptron with a complete arbitrary topology (arbitrary number of inputs, neurons and layers).

In this work the third configuration was chosen: a completely flexible structure with on-chip weight storage.

For this purpose one layer is split into two parts: a synapse and a neuron part. To obtain an arbitrary topology one simply has to cascade the synapse and neuron chips. For example synapse chips with 2 x 2 connections and neuron chips with 2 units can be combined in the following way:

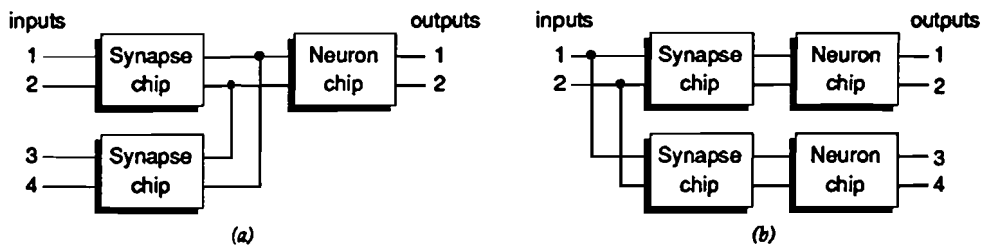



Figure 3.1 *Expanding (a) number of inputs and (b) number of neurons.*

Cascadibilty of the chips has some important electronic implications. When for example one connects 16 (2 x 2)-synapse chips in parallel, then a single neuron has 32 inputs. Suppose one synapse gives an output current between -2 μA and +2 μA, then 32 synapses can (maximally) generate a current between -64 μA and +64 μA. Because the output of a neuron is limited, the output now contains large flat areas. Also the sigmoid shifting range (θ in Eq. (2.3)), which is fitted to 1 synapse per neuron will be to small (4 μA instead of 128 μA). Figure 3.2 gives a graphical explanation of the above mentioned problem.

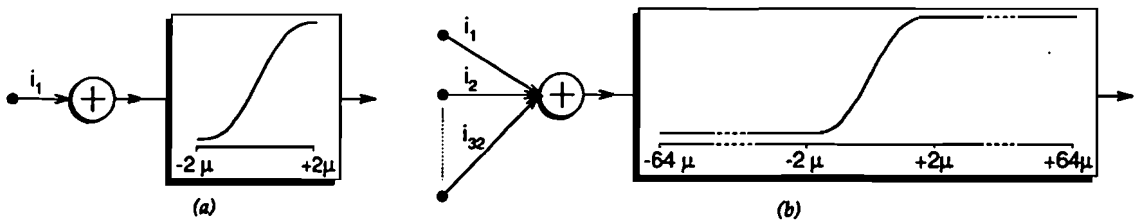



Figure 3.2 *(a) one synapse connected and (b) 32 synapses connected to one neuron.*

Because cascadability is desired to obtain an arbitrary network topology, scaling of the neuron input is necessary. The scaling factor must depend on the number of incoming synapses. Statistical analysis [Withagen, 1994] has shown that it's best to scale the input according to $\sqrt{N}$ (where $N$ is the number of inputs), because not all the weights have their maximum value at one time. Scaling is done after the synapse currents are summed, because this could be implemented easiest.

## 3.2 Global Internal Organization of the Synapse and Neuron Chip

A separate synapse and neuron chip are desired to obtain a complete flexible structure. The network only contains a forward path. In Figure 3.3 these forward paths of both the synapse and the neuron unit are presented. The subscripts $i$ and $j$ refer to the $i^{th}$ input and the $j^{th}$ neuron. The letters V and I refer to whether signals are voltages (signals to be distributed) or currents (signals to be summed).

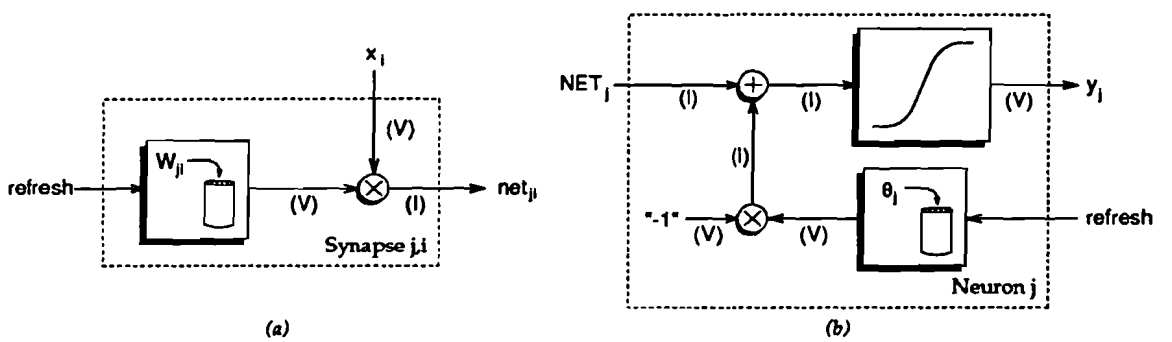



Figure 3.3 *The forward path of (a) the synapse and (b) the neuron unit.*

The Weight Perturbation Algorithm requires that the synapse weights should be accessible from outside, therefore a *refresh*-signal is added to the weight storage-block of the synapse unit. To select a particular weight some extra circuitry is needed, which performs a correct weight addressing. The weight storage in the neuron unit is added to realize a sigmoid function with a variable threshold $\theta_j$.

If N designates the number of inputs and M the number of neurons, then the minimum number of required pins ($P$) per chip is:

$$P_{synapse} > M_{synapse} + N + log_2(M_{synapse} \cdot N) + 2 \quad \text{synapse chip}$$
$$P_{neuron} > 2 \cdot M_{neuron} + log_2(M_{neuron}) + 2 \quad \text{neuron chip}$$

where $log_2(.)$ pins are needed for the addressing and
2 pins for the supply voltage and ground, respectively.

The following figure respresents the internal organization of the synapse and neuron chip. The synapse units are arranged in an array and neuron units in a row.

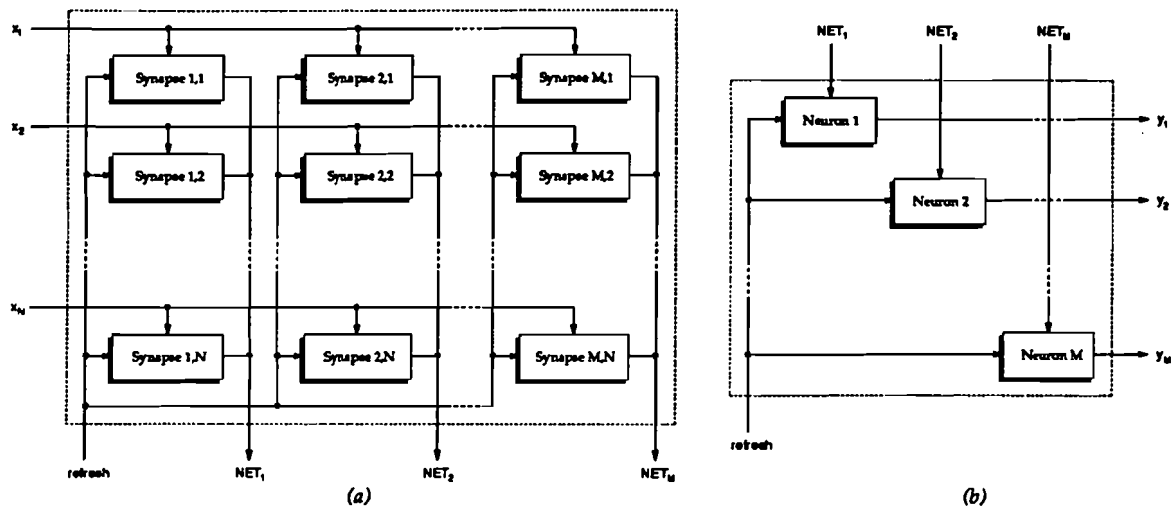



Figure 3.4 *The internal organization of (a) the synapse and (b) the neuron chip.*

## 3.3 Simulation

The simulations of the circuits are done with the simulation program *PSPICE* in combination with the graphic display tool *PROBE*. In PSPICE the elements are defined by their location in the circuit (node numbers) and their function (model). In Appendix A the PSPICE-file, containing all subcircuits of this design, is printed.

There are 4 MOST models, denoted by level 1-4. In this work level 2 parameters are used, because they compare most with measurements on realised circuits. Unfortunately level 2 does not adapt the channel length modulation parameter $\lambda$ to the length of the channel L. For every MOST with a different channel length, a seperate model must be defined with its own $\lambda$. $\lambda$ depends on the Early voltage $V_{Early}$ in the following way:

$$V_{Early} = \frac{1}{L \cdot \lambda} \Leftrightarrow \lambda = \frac{1}{L \cdot V_{Early}} \tag{3.2}$$

where $V_{Early}$ is assumed to be constant (NMOS: 6.6 V/µm and PMOS: 11 V/µm)

The chips were processed at IMEC in Leuven, Belgium. The level 2 parameters, determined during the fabrication process, are printed in Appendix B (N.B. This is a parameter list of 1990!, the list of May 1993 was given after the chips were processed)

Remark: The parameters are printed for minimum sized transistors. For other transistor lengths recalculate $\lambda$ according to Equation (3.2).

For the so-called AC- and transient analysis in PSPICE the layout of the ciruits is very important, because the parasitic capacitances depend on the sizes of the transistors and the mutual positions of the transistors. To calculate the parasitic capacitances between the source and the body, and between the drain and the body of a MOST, the values of *AS* (*A*rea of *S*ource), *AD* (*A*rea of *D*rain), *PS* (*P*erimeter of *S*ource), and *PD* (*P*erimeter of *D*rain) are important. With most circuits the exact values of the parasitic capacitances are not that important for the operation, so only the sizes of the transistors are taken into account, and not their mutual position. In Figure 3.5 the geometry of a minimum sized transistor is given. The values of *AS*, *AD*, *PS*, and *PD* are now calculated according to:

$$\left.\begin{aligned} AS = AD &= (4.8\ \mu m)^2 + W{\cdot}2.4\ \mu m \\ PS = PD &= 24\ \mu m \end{aligned}\right\} \text{ if } W \leq 4.8\ \mu m \qquad (3.2)$$

$$\left.\begin{aligned} AS = AD &= W{\cdot}7.2\ \mu m \\ PS = PD &= 2{\cdot}(W{+}7.2\ \mu m) \end{aligned}\right\} \text{ if } W > 4.8\ \mu m \qquad (3.3)$$

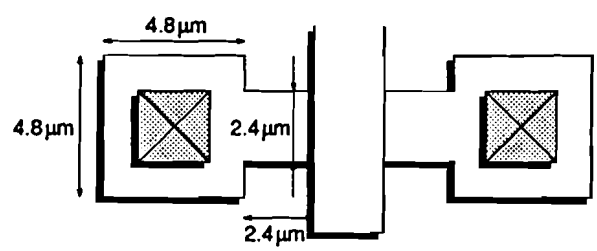



Figure 3.5 *Geometry of a minimum sized MOST.*

## 3.4 The Integrated Circuit Process

After a design was made and proved to be adequate, an implementation in MOS was made. Therefore the designed circuits have to be converted to a layout. For this purpose the program *DALI* is used. *DALI* is an acronym for the *D*elft *A*dvanced *L*ayout *I*nterface.

The final layout was manufactured at IMEC in Leuven, Belgium, using a standard double polysilicon N-Well CMOS process. This means:

- A 2.4 μm technique will be used. The minimum lengths and widths of the transistors are 2.4 μm. Increasing the transistors length and width can only be done in discrete steps of 0.4 μm.
- Ion-implant is impossible, so only enhancement MOSTs can be used.
- Floating gates devices are not available.

Three chips have been processed:

- a component chip (3805-01): to test all circuit parts
- a neuron chip (3805-02): containing 8 neurons
- a synapse chip (3805-03): containing 8 × 8 synapses

The numbers in parenthesis are the chip numbers. The bonding diagrams of all three chips can be found in Appendix C. Of each chip 10 copies are available.

# Chapter 4

# Synapse Chip

*In this chapter the layout of the synapse chip is worked out. The synapses are aimed at storing the values of the weights and multiplying these weights with the input values. The circuits parts which provide these functions are treated in sections 4.1 to 4.3. In these sections a theoretical analysis of the circuit parts is given, together with the simulation and measurement results, which are also compared with each other. Because of the algorithm the weights should be accessible from outside and one has to be able to select them one at a time. So the weight addressing is also important, which is described in section 4.4. In the final section the working of the total synapse chip is discussed.*

*This chapter only discusses the methods that are chosen, although a lot more approaches are possible. For these approaches the reader be referred to the graduation report of Peter Teulings [Teulings, 1991].*

*The internal organization of one synapse is given in Figure 4.1.*

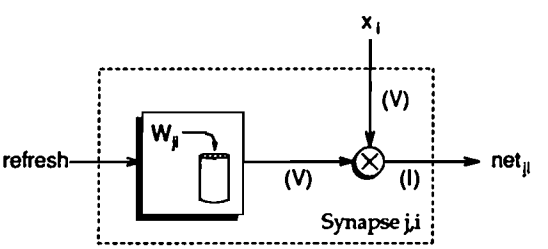



Figure 4.1 *The forward path of the synapse.*

## 4.1 Two-Transistor Multiplier

**Theoretical analysis**

One of the goals of the synapse chip is providing a linear multiplication of the input values with the values of the weights, which both can be positive and negative. Therefore a so-called four-quandrant multiplier is needed.

In [Teulings, 1991] two possible implementations of this type of multipliers are mentioned: the two-transistor multiplier, and the Gilbert multiplier. In this design the two-transistor multiplier is chosen for reasons of little chip area and superb linear behavior. The main characteristics of the two-transistor multiplier will be enumerated next.

The two-transistor multiplier is a very elegant electronic circuit, because it only uses two transistors of the NMOS- or the PMOS-type. The PMOS version is depicted below:

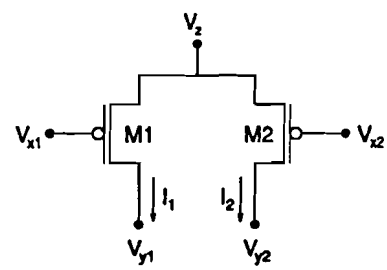



Figure 4.2 *A two-transistor multiplier.*

When both M1 and M2 are working in their linear range, Eq. (4.1) and (4.2) apply:

$$I_1 = K_P' \cdot \left(\frac{W}{L}\right)_1 \left\{\left(V_{x1} - V_{y1} + |V_{T1}|\right)\left(V_z - V_{y1}\right) - \frac{1}{2}\left(V_z - V_{y1}\right)^2\right\} \tag{4.1}$$

$$I_2 = K_P' \cdot \left(\frac{W}{L}\right)_2 \left\{\left(V_{x2} - V_{y2} + |V_{T2}|\right)\left(V_z - V_{y2}\right) - \frac{1}{2}\left(V_z - V_{y2}\right)^2\right\} \tag{4.2}$$

Assuming transistors M1 and M2 are identical, so $|V_{T1}| = |V_{T2}| = |V_{Tp}|$ and $(W/L)_1 = (W/L)_2 = (W/L)$ and $V_{y1} = V_{y2} = V_y$ then the multiplication becomes:

$$I_1 - I_2 = K_P' \cdot \left(\frac{W}{L}\right)\left(V_{x1} - V_{x2}\right)\left(V_z - V_y\right) \tag{4.3}$$

Using the multiplier as follows:

| | | |
|---|---|---|
| input value | : | $V_z - V_y$ |
| weights' value | : | $V_{x1} - V_{x2}$ |
| output value | : | $I_1 - I_2$ |

an excellent multiplication is obtained, see Figure 4.4b.

To get such an excellent multiplication a few things have to be taken into account:

1. Its linearity is splendid due to the fact that both transistors are identical and working in their linear range. The voltages $V_{y1}$ and $V_{y2}$ also have to be the same. To keep the transistors work in their linear range the voltage ranges of $V_{x1}$, $V_{x2}$, and $V_z$ are chosen in such a way that the following conditions are satisfied:

   (a) $\text{Max}(V_{x1}) < \text{Min}(V_z - |V_{Tp}|)$
   (b) $\text{Max}(V_{x1}) < \text{Min}(V_{y1} - |V_{Tp}|)$
   (c) $\text{Max}(V_{x2}) < \text{Min}(V_z - |V_{Tp}|)$
   (d) $\text{Max}(V_{x2}) < \text{Min}(V_{y2} - |V_{Tp}|)$

   If condition (a) is fulfilled, conditions (b) to (d) are fulfilled too.

2. Power dissipation is quite high, because of the relative high currents. One can reduce these currents by using longer transistors and small ranges of $V_{x1}$, $V_{x2}$, and $V_z$. A compromise between chip area and power dissipation is made by chosing *(W/L)* of both transistors (2.4µ/10µ). The implementation area of one two-transistor multiplier, plus capacitors and selection transistors (used for weight storage and access, respectively), will be 5,200 $\mu m^2$. For a more fair measure of the synapse implementation area the reader be referred to section 4.5.

   SPICE-simulations showed that suitable ranges of $V_z$, and $V_{x1}$ are:

   $V_z$ : 2 - 3 V
   $V_{x1}$ : 0 - 1 V

   and the values of $V_{y1}$ and $V_{y2}$ should be both 2.5 V, while $V_{x2}$ should be set to 0.5 V. The maximum current through one multiplier will be 6.5 µA (see Figure 4.4a and sum the currents $I_1$ and $I_2$ at an input voltage $V_z$ of 3 V) resulting in a maximum power dissipation per multiplier of about 20 µW, but a more typical value will be 10 µW.

3. The linearity of the multiplier may be disturbed by the channel length modulation parameter λ. The multiplication equation now becomes:

$$I_1 - I_2 = K'_P \cdot \left(\frac{W}{L}\right)\left(V_{x1} - V_{x2}\right)\left(V_z - V_y\right)\left\{1 + \lambda \cdot \left(V_z - V_y\right)\right\} \tag{4.4}$$

   The quadratic term due to the channel length modulation will be small if λ is small. In the case of PMOS-transistors with $L$ = 10 µm, λ is $7.5 \cdot 10^{-3}$ $\mu V^{-1}$ and the disturbance will be negligible.

4. There is always an undesired positive charge $Q_{ox}$ present at the interface between the oxide and the bulk silicon. This charge is due to impurities and/or imperfections at the interface. It changes the threshold voltage as follows:

$$V_T = V_{T0} + \gamma\left(\sqrt{|-2\cdot\phi_F + V_{SB}|} - \sqrt{2\cdot|\phi_F|}\right) \tag{4.5}$$

The parameter $\gamma$ is termed the *body-effect coefficient*, or *body-factor*. In the PMOS-case $\gamma$ equals 0.6 $V^{\frac{1}{2}}$. The body-effect can be reduced by chosing $V_{SB} \approx 0$ V. While $V_{SB}$ may not become negative, $V_B$ is chosen Max($V_S$) = Max($V_z$) = 3 V.

The chosen multiplier with the applied voltage ranges is depicted in Figure 4.3.

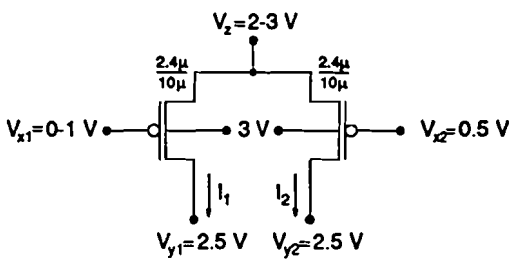
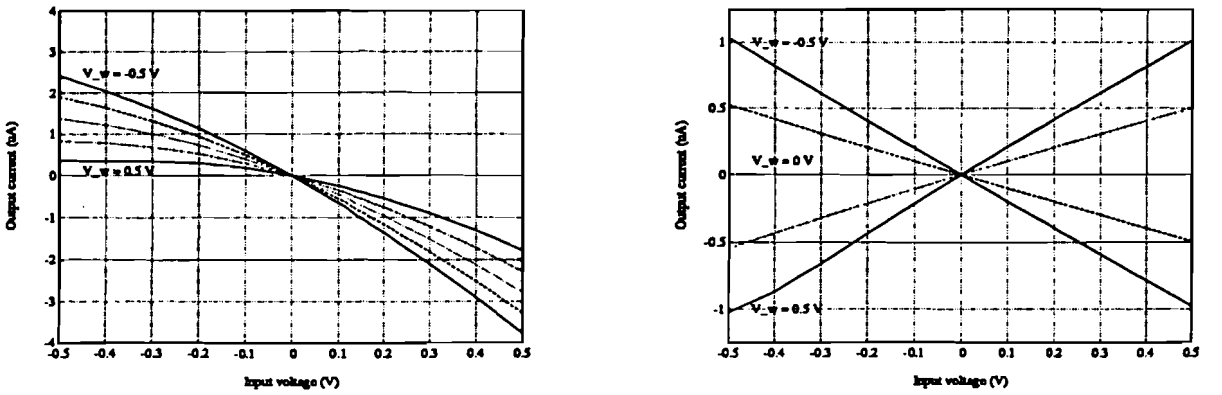



Figure 4.3 *The implemented multiplier.*

**Simulation results**

The two-transistor multiplier was simulated with the (input) values of Figure 4.3, so both the input voltage $V_i = (V_z - V_y)$ and the weights' voltage $V_w = (V_{x1} - V_{x2})$ are in the -0.5 V to 0.5 V-range. In Figure 4.4a the current $I_1$ flowing through M1 is depicted. The current through M2 equals the current through M1 at a weights' value of 0 V (the middle line). As can be seen the maximum current flows at an input voltage of 0.5 V and a weight voltage of -0.5 V: $I_{1,max} \approx$ -3.75 µA and $I_{2,max} \approx$ -2.75 µA.

Figure 4.4b depicts the difference between $I_1$ and $I_2$, the real multiplier output. The multiplier output range equals -1 µA to 1 µA.

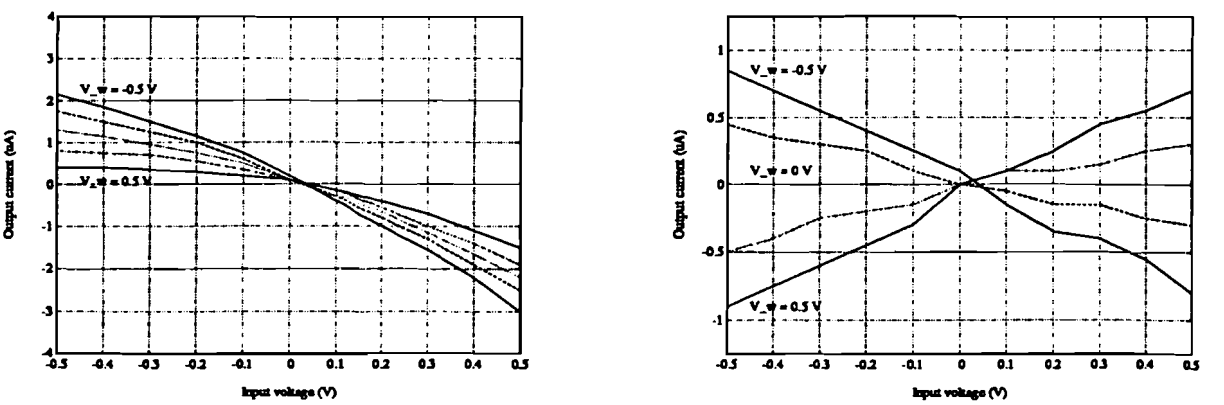



(a) (b)

Figure 4.4 *Simulation results of the two-transistor multiplier.*

**Measurement results**

To check the working of the synapse in practice the same experiments as simulated before were carried out. So the output current of the synapse for different input and weight values were determined. The ammeters are placed between $V_{y1}$ and $V_{ref}$ and $V_{y2}$ and $V_{ref}$ respectively. These meters have an input impedance of 10 kΩ, when used in the 10 μA-range. The output current of single synapse-transistor is in the -3.75 μA to 2.4 μA-range, resulting in a few mV's offset of $V_{y1,2}$. Therefore a small deviation of about 10% compared with the simulated case is possible. Figure 4.5a depicts the measurered current through M1.




(a) (b)

Figure 4.5 *Measurement results of the two-transistor multiplier.*

The precision of the measured values is about ±0.10 μA, therefore the measured curves aren't straight lines. The current of transistor M1 is now between -3.0 μA and 2.15 μA, which coincides with a maximum deviation of 20% compared with the simulated circuit. A part of this deviation (about 10%) is caused by placing the ammeters to measure the currents through M1 and M2, PSPICE simulations proved this assretion. The other part (also about 10%) can be imputed to the parameters variations and other unknown factors, which were not put into the simulation model.

## 4.2 Current Subtraction

**Theoretical analysis**

According to Eq. (4.3) the multiplication output equals the difference of $I_1$ and $I_2$. The voltages $V_{y1}$ and $V_{y2}$ also have to be kept constant. Therefore in [Bruin, 1993] so-called current conveyors were used (CCII$^+$ and CCII$^-$) to subtract the currents. This idea is borrowed from [Eberhardt, 1989]. The circuitry of a CCII$^+$ configuration is depicted in Figure 4.6. To obtain a CCII$^-$ circuit two additional complementary current mirrors were added to the output stage, which inverts the output current. The current conveyor CCII$^+$ keeps the voltage at node X constant and conveys the current at node X to node Z, so $V_X = V_Y$ and $I_Z = I_X$. If $I_X$ is positive it flows through the lower part of the circuit, if it is negative it flows through the upper part. Current conveyor CCII$^-$ also keeps the voltage at node X constant, but $I_Z = -I_X$. Connecting nodes Z of both CCII$^+$ and CCII$^-$, the currents are summed and the resulting current is equal to $I_{X+} - I_{X-}$.

Applying the current conveyor in the two-transistor case (Figure 4.2), one has to connect node *y1* of the multiplier to node X of the CCII$^+$ and node *y2* to node X of the CCII$^-$. This means that $I_{X+}$ equals $I_1$ and $I_{X-}$ equals $I_2$. The resulting current is now equal to $I_1 - I_2$, as desired. The current conveyors also keep the voltages $V_{y1}$ and $V_{y2}$ constant, which is one of the conditions for an appropriate working of the two-transistor multiplier mentioned in section 4.1, topic 1.

The current conveyor configuration was simulated exhaustively. In the DC-case no problems arose. The circuit worked according to the explanation given on the previous page. During transient analysis some combinations of the weights' value and the input value caused an oscillatory output of the current conveyor. Exploration of the open loop configuration of the current conveyor showed that the phase margin was too small. The phase margin was enlarged by adding small, so-called, Miller capacitances to the standard CMOS-opamp (the dashed part in Figure 4.7), but no significant changes occurred.

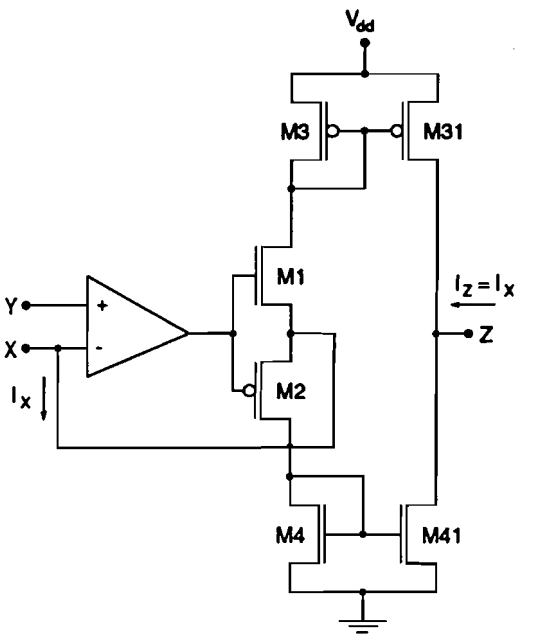



Figure 4.6 *Current conveyor (CCII⁺).*

The operational amplifier used, is a standard CMOS-opamp:

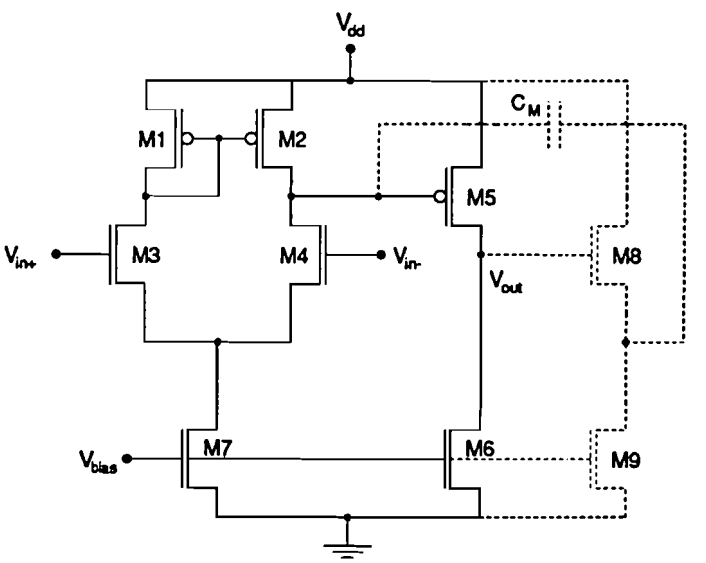



Figure 4.7 *Standard CMOS opamp.*

Since the idea of the current conveyors didn't work, a new approach was worked out: a straightforward subtraction. In the previous section it was stated that the $V_y$-values should be equal and constant. Giving up this requirement it is possible to subtract the currents by using small resistors as loads, which convert the currents to small proportional voltages. Because both sides of the multiplier do not have the same $V_y$ anymore, the multiplication now becomes nonlinear. A nonlinear behavior of the synapse output doesn't matter that much, because later on the output of the synapses will be fed into a neuron, which has a nonlinear transfer function.

Using a standard Operational Transconductance Amplifier (OTA) the $V_y$-voltages can be subtracted easily. Figure 4.8 illustrates the idea. The main difficulty is making good linear resistors in MOS. A very simple idea is to take MOS-transistors in their linear range. Although the $V_{ds}$-$I_d$ characteristic is parabolic, the subtraction of these parabolic curves will be quite linear.

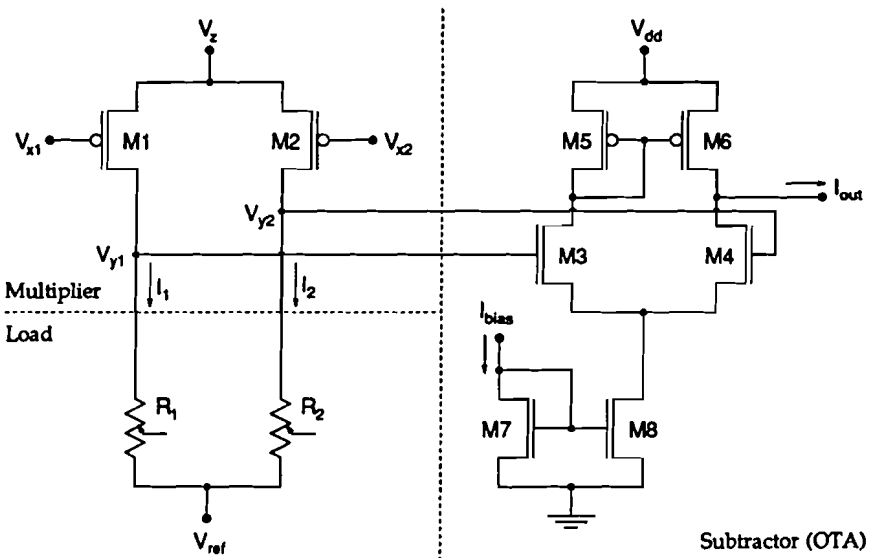



Figure 4.8 *Subtraction of 2 voltages by using small resistors and a differential stage.*

In Figure 4.9 these so-called active resistors are depicted. When they are working in their linear range and $V_{ds}$ is very small compared to $(V_{gs} - V_T)$, the average 'resistance' is

$$R_{ds} = \frac{L}{K_N' \cdot W \cdot (V_{gs} - V_T)} \tag{4.6}$$

Using NMOS transistors $K_N'$ is about $50 \cdot 10^{-6}$, $(W/L)$ is chosen to be $(14.4\mu/2.4\mu)$, and $V_{gs} - V_T$ is about 0.35 V. According to Eq. (4.6) the resistance will then be about 10 kΩ.

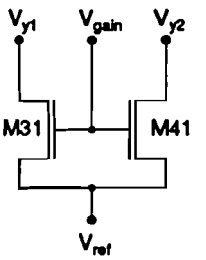



Figure 4.9 *Load of linear transistors.*

The conditions to keep transistors M31 and M41 in their linear range are:

(a) $V_{gain} - V_{ref} \geq V_{Tn}$
(b) $V_{gain} - V_{y1,2} \geq V_{Tn}$

Because the values of the voltages at $V_{y1}$ and $V_{y2}$ should be about 2.5 V as mentioned in section 4.1, $V_{ref}$ equals 2.5 V. If the currents through M31 and M41 and the resistances of M31 and M41 are small enough, then the values of $V_{y1}$ and $V_{y2}$ will be approximately 2.5 V. The information of the multiplication is concealed in the the difference between $V_{y1}$ and $V_{y2}$. This difference will not be linear, but nonlinear. Assume $V_{y1} = V_y + \Delta V_y$ and $V_{y2} = V_y - \Delta V_y$ then $I_1 - I_2$ equals

$$I_1 - I_2 = K_P' \cdot \left(\frac{W}{L}\right) \cdot \left\{ \left(V_{x1} - V_{x2}\right) \cdot \left(V_z - V_y\right) + \Delta V_y \left(V_{x1} + V_{x2} - 2V_y + 2\left|V_{Tp}\right|\right) \right\} \tag{4.7}$$

If $\Delta V_y = 10$ mV, then $\Delta(I_1 - I_2)$ can maximally be 10 % (using the values of the implemented multiplier of section 4.1).

The advantage of using this type of resistors is that it has a scaling possibility, because according to Eq. (4.6) the value of $R_{ds}$ depends on $(V_{gs} - V_T)$ and $V_g = V_{gain}$. The smaller $V_{gain}$ the greater $R_{ds}$ will be. If one synapse is connected to the load, $V_{gain}$ will be set minimal ($R_{ds}$ maximal). The minimum value of $V_{gain}$ is 3.85 V (keeping the loadtransistors just in their linear range). Connecting more than one synapse to the load, results in a higher $V_{gain}$.

During simulation of the prototype version (a matrix of 8 x 8 synapses) the following values of $V_{gain}$ were found, Table 4.1.

| # synapses | $V_{gain}$ |
|---|---|
| 1 | 3.85 V |
| 4 | 4.65 V |
| 8 | 5 V |

Table 4.1 *Gain factors of the synapse chip.*

In Table 4.1 a scaling according to $\sqrt{N}$ ($N$ = number of synapses) is assumed as proposed in section 3.1. If, for some reason, a scaling factor according to $N$ has to be obtained, the values of the various $V_{gain}$'s have to be chosen higher. For large numbers of synapses $V_{gain}$ exceeds the supply voltage (5 V). This doesn't matter, because $V_{gain}$ of the synapse chip is connected to a pad of the synapse chip. One only needs to adjust the protection voltage of this pad.

The main 'disadvantage' is the nonlinear behavior of both the load and the multiplier. This behavior can be made more linear by putting PMOS-transistors in parallel with the NMOST-load. The idea is that the PMOS-transistors, which have an inverse $V_{ds}$-$I_d$ characteristic, can counterbalance the nonlinearity introduced by the NMOS-transistors and the resulting dissimilarity of $V_{y1}$ and $V_{y2}$. The gate-voltages of these PMOS-transistors can be set to a fixed value, which keeps the transistors in their linear range. If the gates are connected to ground (0 V), this condition will be satisfied. Figure 4.10 reflects the resulting circuit.

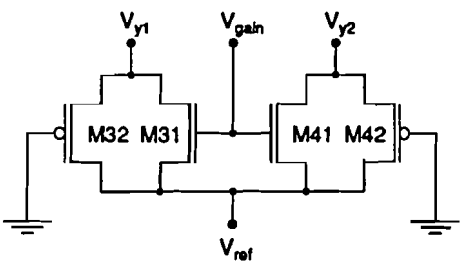



Figure 4.10 *Improved load of linear transistors.*

Choosing ($W/L$) of all transistors (2.4µ/2.4µ) and connecting one synapse to the load the resulting output is depicted in Figure 4.13. The values of $V_{gain}$ for different numbers of connected synapses can be found in Table 4.1.

The voltages $V_{y1}$ and $V_{y2}$ are now subtracted from each other. A standard differential stage with a current mirror as load is used (Figure 4.11).

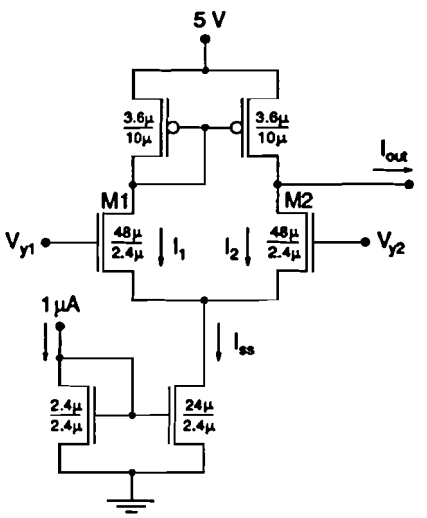



Figure 4.11 *Operational transconductance amplifier.*

Assuming M1 and M2 (Figure 4.11) are always in saturation, the large-signal characteristics can be derived. The pertinent relationships describing large-signal behavior are given as

$$V_{Diff} = V_{gs1} - V_{gs2} = V_{y1} - V_{y2} = \sqrt{\frac{2I_1}{\beta}} - \sqrt{\frac{2I_2}{\beta}} \tag{4.8}$$

$$\beta = K'_N \cdot \left(\frac{W}{L}\right)$$

$$I_{SS} = I_1 + I_2 \tag{4.9}$$

where it has been assumed that M1 and M2 are matched.

Substituting Eq. (4.9) in Eq. (4.8) and forming a quadratic allows the solution for $I_1$ and $I_2$ as

$$I_1 = \frac{I_{ss}}{2}\left(1 + \sqrt{\frac{\beta V_{Diff}^2}{I_{ss}} - \frac{\beta^2 V_{Diff}^4}{4I_{ss}^2}}\right) \tag{4.10}$$

$$I_2 = \frac{I_{SS}}{2}\left(1 - \sqrt{\frac{\beta V_{Diff}^2}{I_{SS}} - \frac{\beta^2 V_{Diff}^4}{4I_{SS}^2}}\right) \tag{4.11}$$

The output current $I_{out}$ equals the difference of $I_1$ and $I_2$, so $I_{out}$ becomes

$$I_{out} = I_{SS} \cdot \sqrt{\frac{\beta V_{Diff}^2}{I_{SS}} - \frac{\beta^2 V_{Diff}^4}{4 I_{SS}^2}} \tag{4.12}$$

If $(\beta \cdot V_{Diff}^2 / I_{ss})$ is substituted by $C^2$, Eq. (4.12) becomes

$$I_{out} = I_{SS} \cdot \sqrt{C^2 - \frac{C^4}{4}} \tag{4.13}$$

Providing C is very small, $C^4/4$ can be neglected and the output current $I_{out}$ will be proportional to $V_{Diff} = V_{y1} - V_{y2}$.

**Simulation results**

First the subtractor with a simple NMOS-load was simulated. In this case transistor sizes of (14.4µ/2.4µ) were used, as a trade-off between implementation area and power dissipation. Using smaller transistors the nonlinear behavior of the subtractor influences the subtraction, because $V_{Diff}$ isn't small enough anymore which conflicts with the condition that $\sqrt{(\beta \cdot V_{Diff}^2 / I_{ss})}$ has to be very small. $V_{ref}$ is set to 2.5 V, as argued before. The set up current $I_{bias}$ of the OTA is the same as mentioned in the *theoretical analysis*-part: -1 µA (see Figure 4.11), because $I_{out}$ has to be zero when the input voltages $V_{y1}$ and $V_{y2}$ are equal. During simulation one synapse was connected to the subtraction circuit, as in the multiple-synapse-connection-case the 'resistance' will be adjusted according to Table 4.1. So $V_{gain}$ will be set to 3.85 V. Figure 4.12 gives the results of using the simple NMOST-load to subtract $V_{y1}$ and $V_{y2}$, for different values of $V_w$ (-0.5 V to 0.5 V) and sweeping the input voltage $V_i$ from -0.5 V to 0.5 V.

As can be seen the output current of the synapse part will be in the -1.75 µA to 1.75 µA-range when NMOST-loads are used. The nonlinear behavior is imputed to the un-equality of $V_{y1}$ and $V_{y2}$ as formulated in Eq. (4.7). To improve the linearity of the subtraction an extension of the NMOST-load with PMOS-transistors was proposed, resulting in the CMOS-load of Figure 4.10. The sizes of all these transistors are (2.4µ/2.4µ). Using this load with one synapse and the same OTA as before, the output current of the subtraction circuit will be more linear and cover a wider range of -3.5 µA to 3.5 µA as can be seen in the simulation results depicted in Figure 4.13. This subtractor is therefore chosen to be implemented on chip.

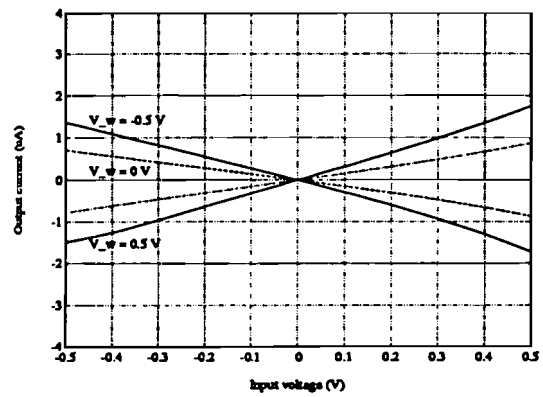



Figure 4.12 *Simulation results of the subtractor with a NMOS-load.*

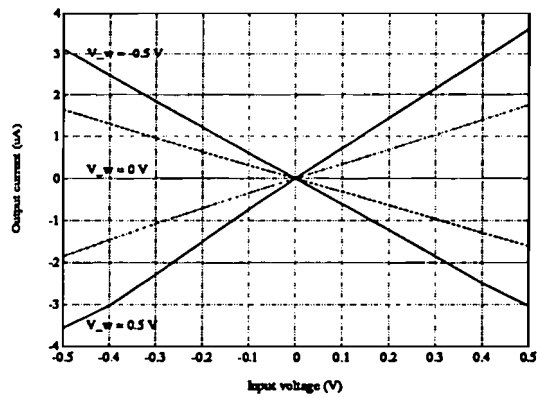



Figure 4.13 *Simulation results of the subtractor with the improved load.*

**Measurement results**

Only the CMOS-version was implemented on a chip (the implementation area equals 8,500 $\mu m^2$), so the measurement results will only be about this implementation. The synapse of the component chip was connected to the subtraction circuit of the same chip. For different weights' and input values the output current was measured. First the bias current $I_{bias}$ was settled in such a way that for zero input values ($V_z$ = 2.5 V, and $V_{x1} = V_{x2}$ = 0.5 V) the output is also 0 μA.

The mean value of $I_{bias}$ equals -0.75 µA (chip #5: -0.9 µA; #6: -0.6 µA; #7; -0.75 µA). This deviation of about 25% is imputed to the variations in the transistorsizes of the current mirrors (M5, M6) and (M7, M8) of the OTA (Figure 4.8), because all transistors are minimum sized. The mirror ratio was measured by connecting $V_{y2}$ to 0 V, and $I_{out}$ equals $I_{ss} \approx 10$ µA, so the mirror ratio is 1 : 13⅓ instead of 1 : 10 according to the (W/L)-ratios of the mirrors (M3, M4) and (M5, M6). To obtain more accurate current mirrors, larger and wider transistors have to be used.

In Figure 4.14 the output current of chip #7 as a function of the input voltage $V_i = V_z - V_y$ for different weight values ($V_{x1} - V_{x2}$) is depicted. $V_{gain}$ was set to 3.85 V.

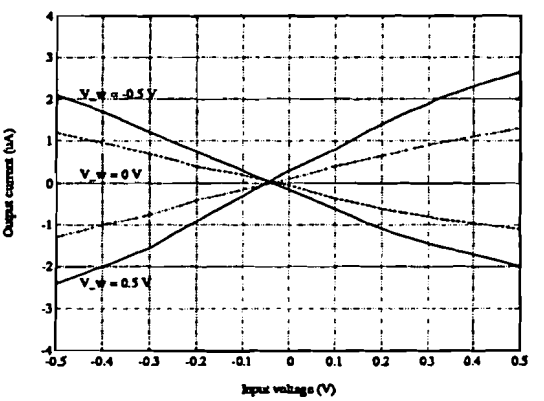



Figure 4.14 *Measurement results of the subtractor with the improved load.*

The maximum output range equals -2.45 to 2.75 µA, so this means a reduction of about 30% when compared with the simulations. 20 % can be imputed to the multiplier part (see the measurement results of section 4.1), the other 10 % is caused by imperfections of the subtractor part by setting $I_{bias}$ to its mean value for all copies.

Finally, the offsets of both inputs and the output were determined. The input offsets were determined by sweeping one of the inputs (input value or weight input) over its total range and setting the other input value to 'zero', so no output change occured. In theory the 'zero'-values of both inputs were, $V_z$= 2.5 V and $V_{x1}$ = 0.5 V, respectively. The difference between these values and the measured ones were defined as the input offsets. The output offset equals the 'zero' output value, which has to be 0 µA. The offsets are enumerated in Table 4.2. In this table $\Delta x = x_{measured} - x_{simulated}$, $V_i = V_z - V_y$, and $V_w = V_{x1} - V_{x2}$.

| Chip | $V_i$ (mV) | $\Delta V_w$ (mV) | $\Delta I_{out}$ (μA) |
|---|---|---|---|
| #5 | -30 | -23 | -0.25 |
| #6 | -30 | -58 | 0.45 |
| #7 | -30 | 0 | 0.10 |
| #8 | -30 | -4 | -0.35 |
| #9 | -30 | 22 | -0.20 |
| #10 | -30 | -50 | 0.25 |

Table 4.2 *Multiplier offsets of the synapse unit.*

The maximum offsets are:

$V_i$ : 3%
$V_w$ : 6%
$I_{out}$ : 10%

## 4.3 Weight Storage

**Theoretical analysis**

The weight storage is implemented on-chip. This can be done in several ways. The most commonly used techniques are storage on floating gates and on capacitors. Only capacitors remain left, because floating gate devices are not available in the integrated circuit process used (see section 3.4). Using capacitors as weight storage elements, two problems arise: leakage and charge injection. Sample-and-hold circuits are used to select a weight. The most simple sample-and-hold circuit is depicted in Figure 4.15 and consists of only one switching transistor (MS1) and one capacitor ($C_{w1}$). When the switch is turned off, a small leakage current flows, which is equal to the reverse-biased diffusion-to-bulk current of the source-bulk diode of the transistor. This effect would be cancelled if the substrate voltage $V_{bulk}$ exactly equals the stored voltage $V_{x1}$.

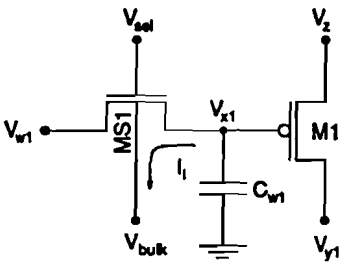



Figure 4.15 *Leakage current of the selection transistor.*

In literature [Vittoz, 1989] a lot of techniques are developped to reduce the leakage of a capacitor, but all of these techniques use a lot of chip area and a small chip area is one of the main goals of the design of a synapse chip. Because of the differential character of the two-transistor multipliers, another approach is possible. Both of the weight inputs of the multiplier ($V_{x1}$ and $V_{x2}$) can be accessed by a sample-and-hold circuit. At both sides a leakage current arise. In theory, leakage currents do not depend on the voltage of the capacitor, so $I_{l1} = I_{l2}$. The idea is reflected in Figure 4.16. After summation the currents are subtracted, and so are the leakage currents.

The difference between leakage currents $I_{l1}$ and $I_{l2}$ can be kept minimal by placing both sides of the multiplier close together on the chip, because the differences between the used transistors and capacitors will then be as small as possible. Chosing capacitor values of about 0.1 pF, one can calculate an effective leakage of approximately 3 mV/s (see [Bruin, 1993], where it is assumed that there's a difference of only 1% between the leakage currents $I_{l1}$ and $I_{l2}$). Leakage will be small if the updating procedure of a single weight will take a few μs.

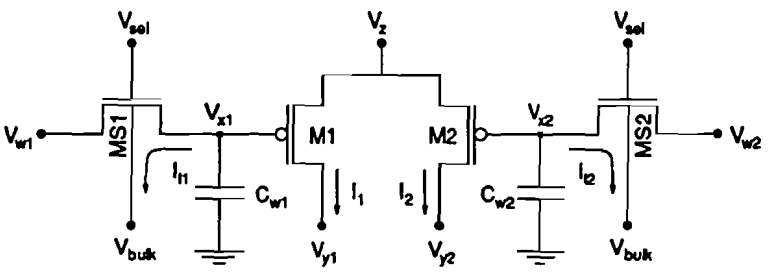



Figure 4.16 *Differential weight storage.*

**Measurement results**
The assumption of a 1%-difference between $I_{t1}$ and $I_{t2}$ is not so realistic. A more typical value will be 10%, resulting in an effective leakage of about 30 mV/s. The leakage at the output of the subtractor will be approximately 0.1 μA/s, what can already be seen on the ammeter (used in the 10 μA-range). Refreshing the weights once per second the measured output current leakage is about 0.1-0.2 μA, corresponding with the expected value at a 10-20%-difference between $I_{t1}$ and $I_{t2}$. To reduce the leakage-effect, one has to refresh the weight with about 1 kHz (or higher values), what can be realized easily, and results in a leakage of at most 30 μV per weight.

## 4.4 Weight Addressing

**Theoretical analysis**
The weight addressing discussed below was suited to a synapse matrix of 8 x 8 synapses. Because the synapses were arranged in a matrix, a row and a column address both being 3 bits were used ($a_0$, $a_1$, $a_2$, and $a_3$, $a_4$, $a_5$). The synapses were numbered according to zero-index numbering.

In this prototype version only one chip at a time will be connected to a neuron chip. So no Chip Enable (*CE*) signal was needed, only a Write Enable (*WE*) signal to write a particular weights' value. The updating procedure will now be as follows: select the intended weight by offering the correct weights' address to the address lines, offer the proper weights' value to the net and activate the *WE* line.

To save chip area, also the inverse signals of the address lines ($\neg a_0$, etc.) were extended to the net, so a total amount of 12 address lines are used. Each weight is characterized by its row- and column number, Row Select (*RS*) and Column Select (*CS*). For each combination of $a_0$, $a_1$, $a_2$, and $a_3$, $a_4$, $a_5$, an AND-gate is used to activate the Column Select (*CS*) and Row Select (*RS*) line, respectively. For example: to select weight $w_{32}$, *CS* will be equal to $\neg a_0 \cdot a_1 \cdot \neg a_2 \cdot WE$ and $RS = \neg a_3 \cdot a_4 \cdot a_5 \cdot WE$. The weight is accessed when both signals are high ("1"), which is done by an AND-gate. This prototype version will use simple logic gates, NMOS instead of CMOS. A disadvantage of using NMOS logic gates is that in the AND-case (series connection) the output voltage decreases with the number of inputs, because each transistor has its own $V_{ds}$ and the supply voltage is limited. In the OR-case (parallel connection) the current increases with the number of inputs. In practical situations 4-input AND's and OR's can be made. The AND-gates are chosen because of their low power dissipation, only if a weight is selected current flows through the circuit. In Figure 4.17 the used AND-gates are depicted.

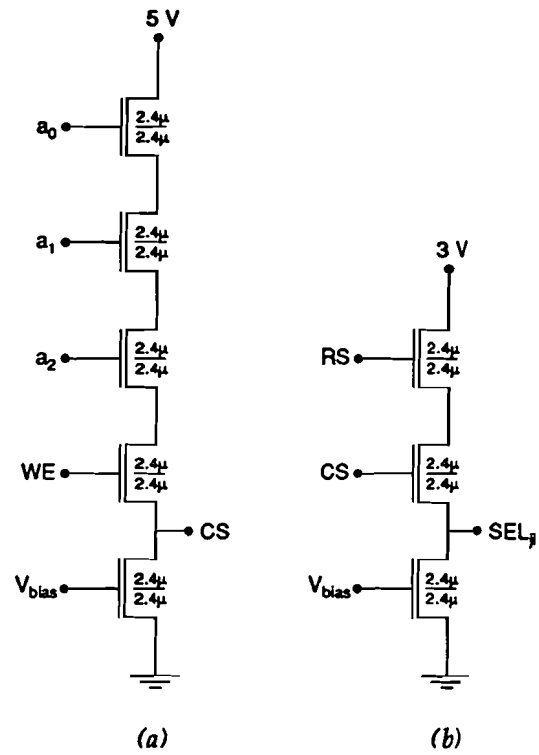



Figure 4.17 *The selection circuits.*

The chip area of the selection equals:

- 80,000 µm² for both row and column selection
- 4,200 µm² per 2 AND-gates (64 AND's needed on the synapse chip)

So, the implementation area of the selection circuitry on the synapse chip is about 32 × 4,200 + 2 × 80,000 ≈ 300,000 µm², while on the neuron chip only 80,000 µm² was needed.

In Figure 4.17a the column selection of column 7 is represented. To select another column, one has to connect the inputs $a_0$, $a_1$ and $a_2$ to the address lines which belong to the desired column number. The row selection is carried out in the same way, $a_0$, $a_1$ and $a_2$ have to be replaced by $a_3$, $a_4$ and $a_5$. The output will then be *RS*. The *WE* input is physically placed under the address line inputs, because during the selection of another weight, no spikes may be observed at the output. To access a particular weight the 2-input AND of Figure 4.17b is used, which uses a supply voltage of 3 V. This value was chosen, because in the layout these AND-gates were placed near the multipliers, which need a bulk voltage of 3 V (see Figure 4.3).

For the circuit's functioning it is all the same to choose 3 V instead of 5 V, only a distinction between "0" and "1" is needed. PSPICE simulations confirm this assertion.

The bias voltage $V_{bias}$ of both circuits was set to 1.35 V, which keeps the bottom transistor in saturation. In Figure 4.18 the resulting addressing structure of the synapse chip is given.

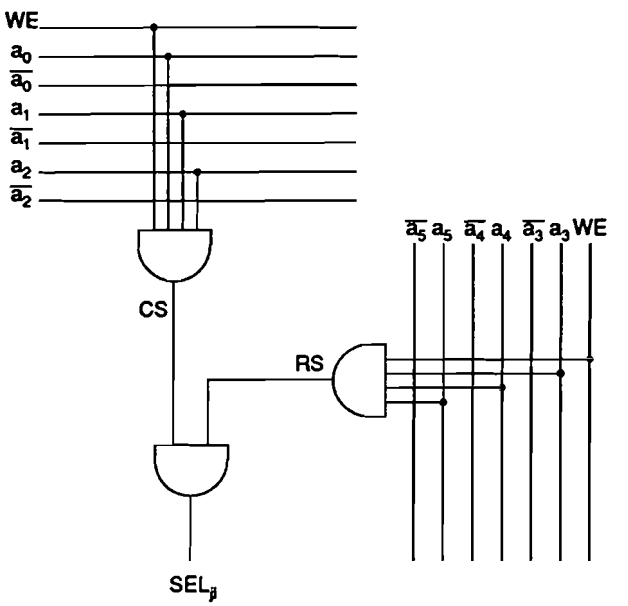



Figure 4.18 *Addressing structure of the synapse chip.*

If the synapse chips are expanded to, for example, 32 x 32, or 64 x 64 synapses the addressing of the weights can be done more efficient. The expansion, for instance, from 5 bits to 32 outputs can be done with less than 32 AND-gates.

**Measurement results**

The main disadvantage of using NMOS instead of CMOS logic AND-gates was the voltage drop over the transistors. In practice this appeared to be a real problem. The voltages at the output of the 4-input AND's, *RS* and *CS*, only became about 3 V. Because the input voltages of the 2-input AND's were only 3 V, the selection voltage of a particular weight, $SEL_{ji}$, didn't exceed 2 V and consequently the selection transistors MS1 and MS2 (Figure 4.16) could not be turned on properly. The selection voltage was increased a bit (to just above 2 V) by decreasing $V_{bias}$ to the 0.95 V to 1.00 V-range, but in this case one can not be sure that the bottom transistors of Figure 4.17a/b are still working in their saturation region. To solve this problem the 4-input AND-gates must be carried out as CMOS- instead of NMOS-types.

## 4.5 Total Synapse Chip

**Theoretical analysis**

The synapse chip contains a matrix of 8 x 8 synapses (a structure analogous to Figure 3.4a) and 8 subtractor circuits. One column of 8 synapses and one subtractor circuit are depicted in Figure 4.19. The chip area of one column of synapses is about 80,000 $\mu m^2$, so per synapse about 10,000 $\mu m^2$ of effective chip area is occupied.

Expanding the synapse chip to $N \times N$ synapses, the number of the relatively small two-transistor multipliers is increased with a factor $N^2$, as the number of the (larger) subtractor circuits (including the loads) is increased with only a factor $N$, so the effective chip area of the synapse is decreased.

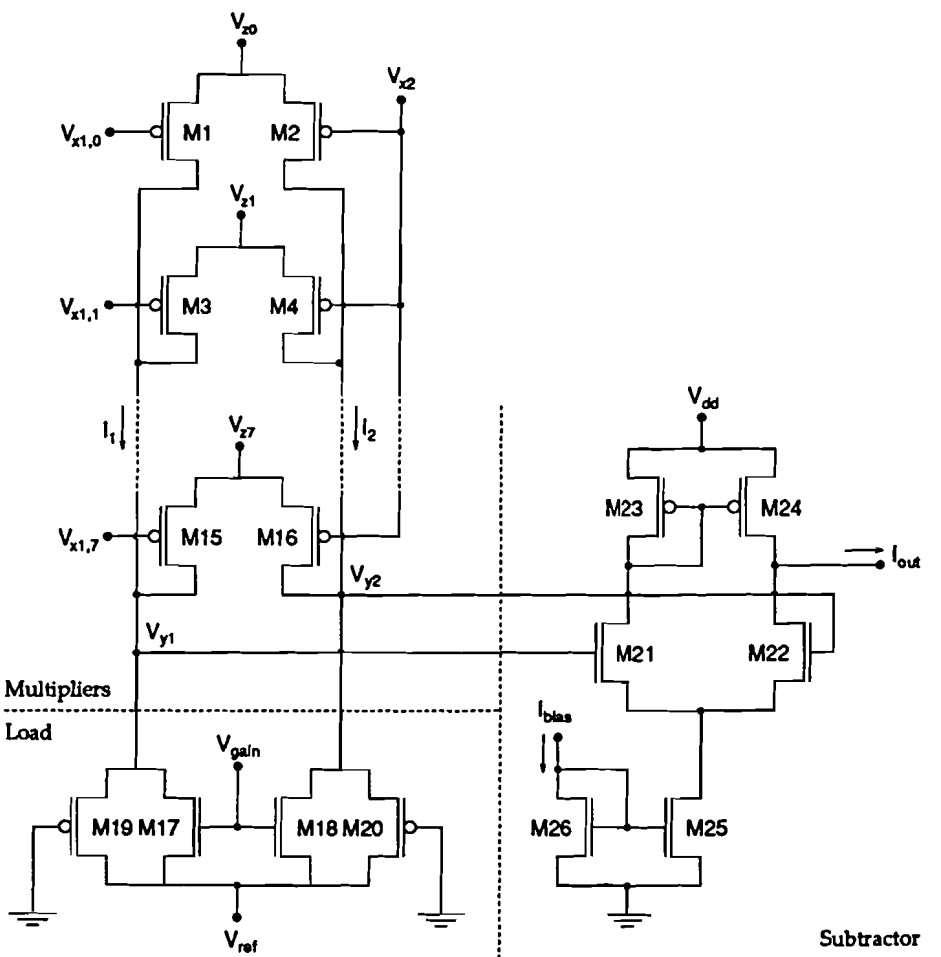



Figure 4.19 *One column of the synapse chip.*

**Simulation results**

To simulate the effect of placing more than 1 synapse in parallel, a $N$-times larger transistor was taken (or $\sqrt{N}$ if the results of the statistical analysis performed in [Withagen, 1994] are taken into account). Later on (after the first measurements) this appeared to be a wrong approach, because the tacit assumption that the unselected synapses in one column didn't have a contribution to the output current wasn't true at all. For instance, the upper synapse (#0) was selected solely, and therefore the inputs of the other synapses (#1 to #7) were all set to zero ($V_z$ = 2.5 V, and $V_{x1} = V_{x2}$ = 0.5 V), then a particular current flew through synapse #0, resulting in voltages $V_{y1}$ and $V_{y2}$, which were subtracted from each other. If the input values of synapse #0 are non-zero then $V_{y1} \neq V_{y2}$ and so the current through the left part of the unselected synapses will slightly differ from the current through the right part, resulting in a contribution to the output current.

The above mentioned example was simulated and the results are depicted in Figure 4.20.

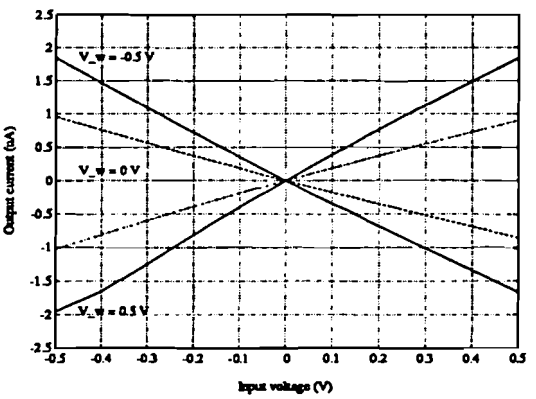



Figure 4.20 *Simulation results of the feedforward path of the synapse chip.*

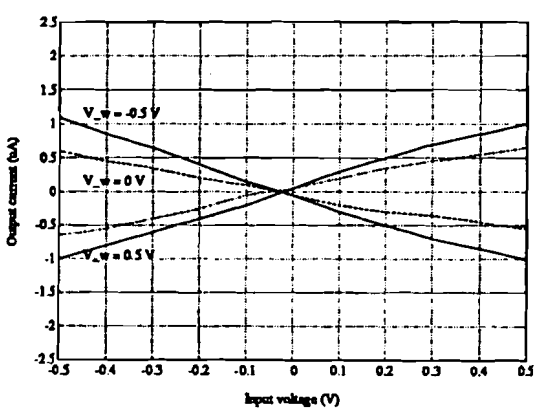
The current through an unselected synapse caused a reduction of the output current of 6%, so in this example (1 synapse was selected, the other 7 synapses were unselected) the output current was reduced by 42% (compare with Figure 4.13). If a synapse chip of 16 x 16 synapses is made and only one synapse is used, then the output current of this synapse will be reduced by almost 100%! This problem can be solved by using another refresh scheme. The unused synapses will be refreshed with the supply voltage 5 V instead of 0.5 V. If both $V_{x1}$ and $V_{x2}$ are 5 V, then no current will flow through the synapse transistors, because $V_z$ and $V_{y1}$ (or $V_{y2}$) are both about 2.5 V. Now, selecting one synapse out of 8 will give the same results as a single synapse, so the simulation results are the same as depicted in Figure 4.13.

**Measurement results**

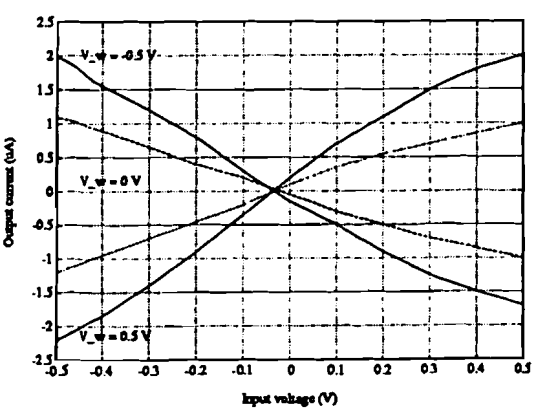
The 'diodes' of the current mirrors producing the bias currents $I_{bias}$ of all columns were erroneously connected together, so the total bias current has to be 8 times the bias current of one subtractor. The bias current fed to the synapse chip equaled -5.1 μA, not exactly 8 times the -0.75 μA of a single subtractor, because of the parallel connection of the minimum sized current mirrors. The main disadvantage of making this mistake is the uncertainty of a good distribution of the current to all subtractor circuits, which depends on the matching of all current mirrors.

In practice three experiments were performed: selecting 1, 2 and 4 synapses, respectively. If there was more than 1 synapse selected, these synapses were put in parallel. To select a particular column the right combination of bits $a_0$ to $a_2$ was activated. To select a particular synapse within a column a selection circuit was built, being composed of a 8-to-1 multiplexer, a counter and a clock. This selection circuit refreshed every weight within a column with a refresh rate of 1 kHz, so each second each weight was refreshed a 1000 times (the clock was set to 8 kHz).

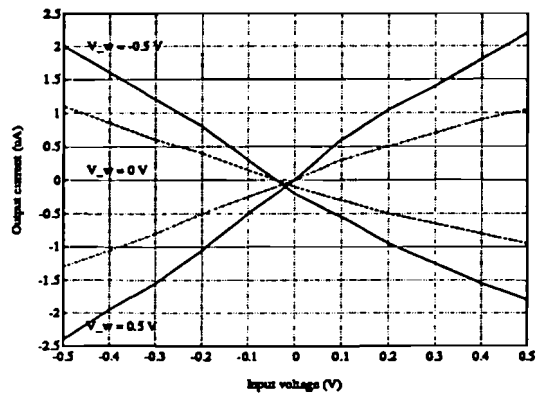
The reference voltage $V_{ref}$ was set to 2.5 V, the bias voltage $V_{bias}$ to about 1 V, and the $V_{x2}$-part of the multipliers was connected to 0.5 V. The input voltage(s) $V_z$ of the selected synapse(s) was/were sweeped from 2 to 3 V, and the input voltages $V_z$ of the unselected synapses were connected to the reference voltage $V_{ref}$ = 2.5 V. The $V_{x1}$-part of the selected synapse(s) was sweeped from 0 to 1 V, while the unselected synapses were set to $V_{x1} = V_{x2} = 0.5$ V. In Figure 4.21a the results of selecting synapse (0, 0) of synapse chip #2 are depicted, the gain voltage $V_{gain}$ was set to 3.85 V. Figure 4.21b shows the results of putting synapses (0, 0) and (0, 1) of synapse chip #2 in parallel ($V_{gain}$ = 3.85 V). Finally, Figure 4.21c shows the results of the parallel connection of synapses (0, 0) to (0, 3) (4 synapses), where $V_{gain}$ was set to 5 V.




(a)




(b)




(c)

Figure 4.21 *Measurement results of the feed-forward path of the synapse chip.*

Comparing Figure 4.21a with Figure 4.21b the output current of putting in parallel 2 synapses is two times higher than the output current of 1 synapse (as expected). The amplitude of the output current of one synapse (selected within a column of 8 synapses) is only half the output current of a single synapse (see measurement results of section 4.2), which is imputed to the non-zero contribution of the unselected synapses. In Figure 4.21c the output current of the parallel connection of 4 synapses is depicted, which is in the same range as the output current of Figure 4.21b, due to the higher gain voltage.

No measurements were carried out with the improved refresh scheme (connecting the gates of the unused synapses to the supply voltage), but it can be expected that the results of selecting one synapse out of 8 will give the same results as depicted in Figure 4.14.

# Chapter 5

# Neuron Chip

*This chapter describes the design of the neuron chip. The electronic circuits needed for the generation of a sigmoidal nonlinearity with a shifting possibility are discussed in sections 5.1 to 5.4. In section 5.5 the complete feed-forward path of the neuron unit is given. In Figure 5.1 the internal organization of one neuron is depicted.*

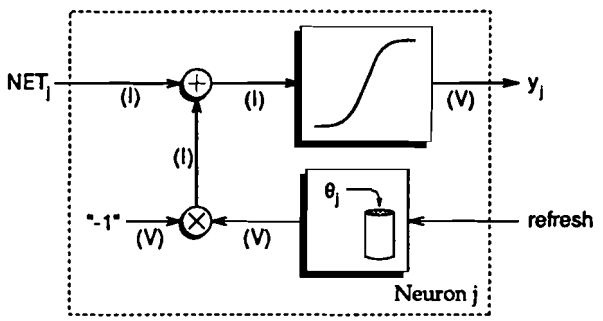



Figure 5.1 *The forward path of the neuron.*

## 5.1 Sigmoid

**Theoretical analysis**

The main aspect of the neuron unit is its nonlinear behavior. In section 2.1 the main characteristics of a sigmoid shaped nonlinearity were treated, which is used by the Weight Perturbation Algorithm. To create a sigmoid-like function in electronics a differential stage can be used, which is depicted in Figure 5.2.

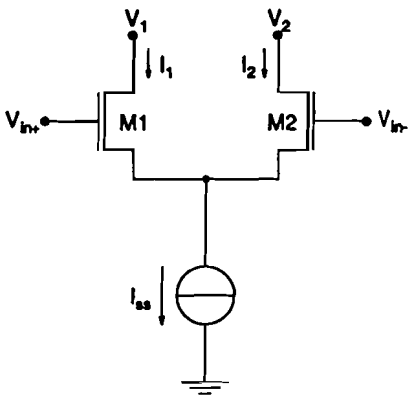



Figure 5.2 *Differential stage.*

Assuming M1 and M2 are always in saturation, the same equations as used for the current subtraction of section 4.2 apply:

$$V_{Diff} = V_{gs1} - V_{gs2} = V_{in+} - V_{in-} = \sqrt{\frac{2I_1}{\beta}} - \sqrt{\frac{2I_2}{\beta}} \tag{5.1}$$

$$\beta = K'_N \cdot \left(\frac{W}{L}\right)$$

$$I_{SS} = I_1 + I_2 \tag{5.2}$$

If transistors M1 and M2 are matched and substituting Eq. (5.2) in Eq. (5.1), the solution for $I_1$ and $I_2$ becomes

$$I_1 = \frac{I_{SS}}{2}\left(1 + \sqrt{\frac{\beta V_{Diff}^2}{I_{SS}} - \frac{\beta^2 V_{Diff}^4}{4I_{SS}^2}}\right) \tag{5.3}$$

$$I_2 = \frac{I_{SS}}{2}\left(1 - \sqrt{\frac{\beta V_{Diff}^2}{I_{SS}} - \frac{\beta^2 V_{Diff}^4}{4I_{SS}^2}}\right) \tag{5.4}$$

These relationships are only valid for $V_{Diff} < (2I_{ss}/\beta)^{1/2}$. Figure 5.3 shows a plot of the normalized drain current ($f(x) = I_1/I_{ss}$) of M1 versus the normalized differential input voltage ($x = (\beta/I_{ss})^{1/2} \cdot V_{Diff}$). The transfer function of Figure 5.3 looks almost like the sigmoid shape of Figure 2.1b.

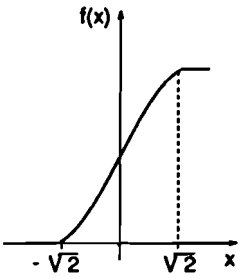



Figure 5.3 *Large-signal transfer characteristics of a CMOS differential amplifier.*

Using linear resistors as loads, the accompanying voltages $V_1$ and $V_2$ will also have a sigmoidal shape. As with the current subtraction in section 4.2, linear resistors in MOS are made by taking MOS-transistors in their linear range and providing $V_{ds} << 2(V_{gs} - V_T)$. In this case PMOSTs are chosen instead of NMOSTs. In Figure 5.4 the resulting circuit is depicted.

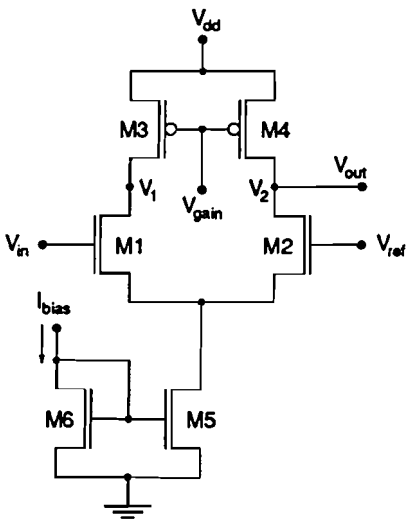



Figure 5.4 *Sigmoid function.*

Transistors M3 and M4 can be kept in their linear range if

$$V_{DS} \geq V_{GS} + |V_{T_p}| \tag{5.5}$$

The currents through M3 and M4 now become:

$$I_3 = K_p' \cdot \left(\frac{W}{L}\right)_3 \left\{\left(V_{gain} - V_{dd} + |V_{T3}|\right)\left(V_{dd} - V_1\right) - \frac{1}{2}\left(V_{dd} - V_1\right)^2\right\} \tag{5.6}$$

$$I_4 = K_p' \cdot \left(\frac{W}{L}\right)_4 \left\{\left(V_{gain} - V_{dd} + |V_{T4}|\right)\left(V_{dd} - V_2\right) - \frac{1}{2}\left(V_{dd} - V_2\right)^2\right\} \tag{5.7}$$

where $V_2 = V_{out}$

The only parameters which can be changed are $V_{gain}$ and $(W/L)$. Making $V_{gain}$ low, the output range will become wider. To get the same voltage drop over M3 and M4, the currents through M3 and M4 should be increased. So a larger tail current $I_{bias}$ is needed, which means an augmented power dissipation. An increased current $I_{bias}$ leads to increased widths of transistors M1 and M2. The other way to enlarge the output range is taking wider transistors M3 and M4. Therefore it was chosen for a compromise between power dissipation and chip area, which leads to the configuration depicted in Figure 5.5. The implementation area of the sigmoid stage is equal to 5,400 $\mu m^2$, and the power dissipation equals 0.14 mW.

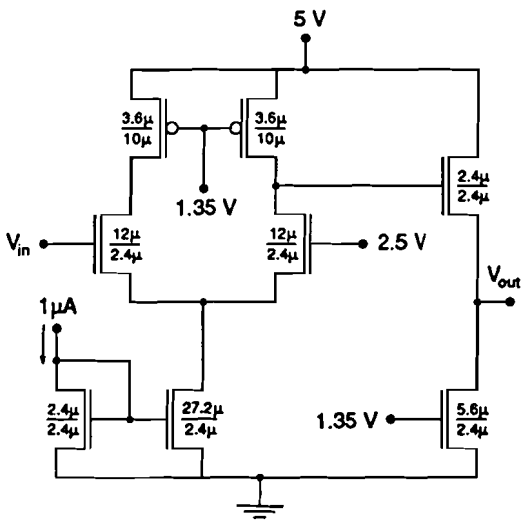



Figure 5.5 *The chosen sigmoid.*

The level shifter at the output stage of the sigmoid circuit was added to transform the output voltage to the 2 to 3 V range, because the output of the neuron unit can be connected to the input of one or more synapse units, which desire a 2 to 3 V input range (see section 4.1).

The input of the sigmoid is also 2 to 3 V, so the reference voltage $V_{ref}$ was chosen to be 2.5 V. The other voltages ($V_{gain}$ and $V_{bias}$ of the level shifter) are 1.35 V. In the final version of the neuron chip these nodes can be connected together, which reduces the number of pins.

The output of the neuron unit must be able to feed at least 32 neurons, so the output current must be high enough to do so (about +80 µA to -120 µA). Two approaches are possible:

(a) increasing the sizes of the output transistors (of the level shifter), or
(b) adding a buffer stage to the output of the sigmoid unit,

both resulting in an increased output current. The main disadvantage of both approaches is an increased implementation area of the neuron unit. Approach (a) results in an output stage with transistors widths of several hundreds of µm's, which is unacceptable. So only approach (b) remains left. The circuitry of the buffer is described in the next section.

**Simulation results**

The sigmoid circuit was simulated with PSPICE using the set ups mentioned above. Figure 5.6 represents the resulting sigmoid shape.

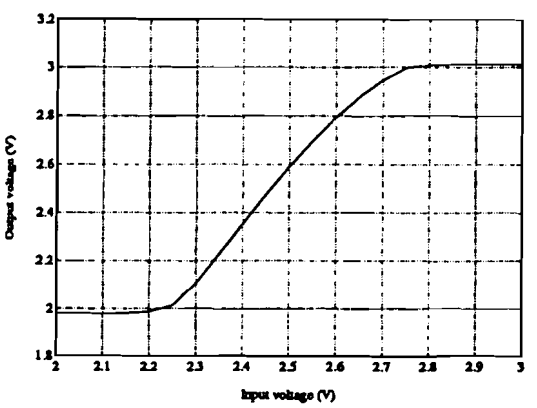



Figure 5.6 *Simulation results of the sigmoid stage.*

**Measurement results**

In the layout of the chips, the pins of the chips which were connected with a (MOS-)diode were not protected. So the bias current inputs had to be protected manually by putting diodes to the supply voltage and ground. Measuring the single sigmoid stage, this has been forgotten, so almost all chips were broken. Only chip #4, and #5 worked properly.

The bias current $I_{bias}$ of the simulations is -1 μA. To get a working sigmoid-stage the bias current was set to -0.4 μA. A part of this deflection is probably owing to the deviations in the transistor sizes of the current mirror, which consists of minimum sized transistors. The set ups of both chip #4, and #5, are enumerated in Table 5.1.

| Set up | Chip #4 | Chip #5 |
|---|---|---|
| $I_{bias}$ (μA) | -0.4 | -0.4 |
| $V_{gain}$ (V) | 1.35 | 1.40 |
| $V_{bias}$ (V) | 1.32 | 1.30 |
| $V_{ref}$ (V) | 2.50 | 2.30 |

Table 5.1 *Set ups of the measured sigmoid stage.*

Using the set ups of Table 5.1 the transfer function of the sigmoid stage becomes:

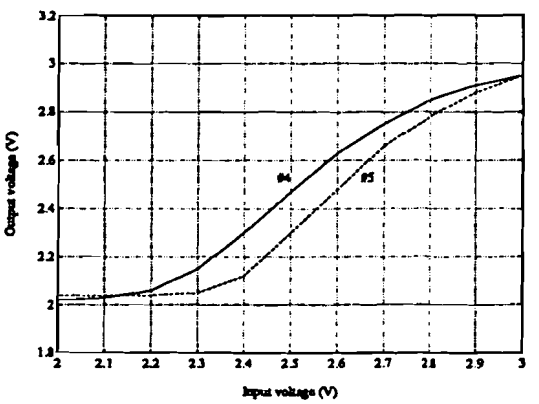



Figure 5.7 *Measurement results of the sigmoid stage.*

Comparing the simulation results with the measurement results one thing catches the eye. The slope of the measured sigmoid function is more smooth then the simulated one (about 2 times). This implies that the output current of the synapse chip has to be larger (also 2 times). PSPICE simulations showed that the sigmoid stage is very sensitive to parameter variations, in particular to the bias current $I_{bias}$. Attempts to steepen the slope of the sigmoid by decreasing the bias current $I_{bias}$ and increasing the gain voltage $V_{gain}$ failed, because $V_{gain}$ becomes to large, and transistors M3 and M4 are no longer working in their linear range.

## 5.2 Buffer

**Theoretical analysis**

To increase the output current of the neuron unit a CMOS operational amplifier is used as buffer. In this case a CMOS Operational Transconductance Amplifier (OTA) is used, with an extra source-follower, which connects a Miller capacitor to the OTA. This Miller capacitor $C_M$ is needed to increase the phase margin of the OTA. Now the same circuit as depicted in Figure 4.7 results. The only difference is the connection of the output $V_{out}$ with the negative input $V_{in-}$, which provides a unity feedback to form a buffer configuration. The buffer, with its chosen transistor sizes, is represented in Figure 5.8. The buffer requires an implementation area of about 12,500 $\mu m^2$ and the power dissipation is 0.71 mW.

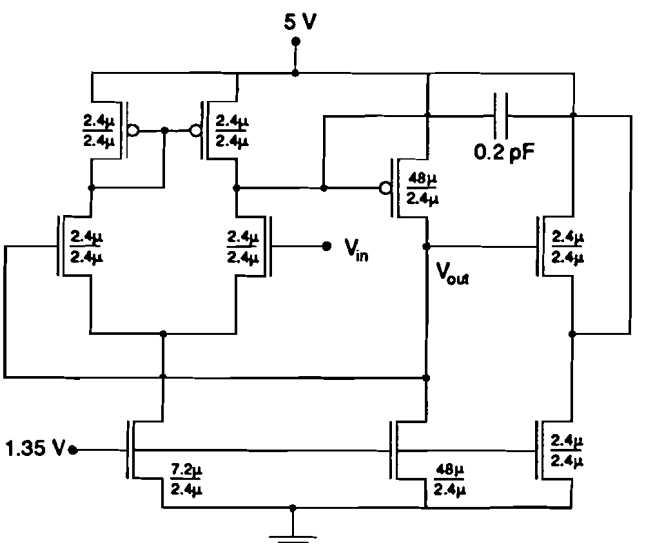



Figure 5.8 *The chosen buffer (OTA).*

**Simulation results**

The transfer characteristic has to be linear in the 2 to 3 V range, and its offset has to be (almost) zero. Simulating the buffer of Figure 5.8 results in the following transfer characteristic.

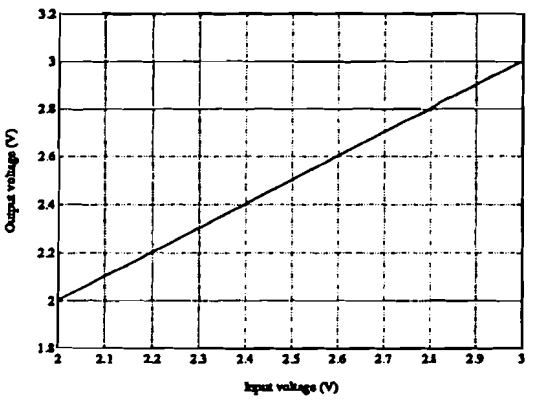



Figure 5.9 *Transfer charateristic of the output buffer.*

Very important in this case is the transient behavior of the buffer, because its delay time must be small to obtain a high feed-forward speed of the neural network. SPICE simulations were done to determine the transient behavior of the buffer. At the input a block voltage in the 2 to 3 V range was presented with several transitions from "high" to "low" and from "low" to "high". At the output a load of 20 pF and 1 MΩ were connected, simulating the practical situation of connecting a probe to the output of the neuron part (the buffer). The results are depicted in Figure 5.10.

The maximum delay time arises when a transition from "high" to "low" is made and its value is about 150 ns.

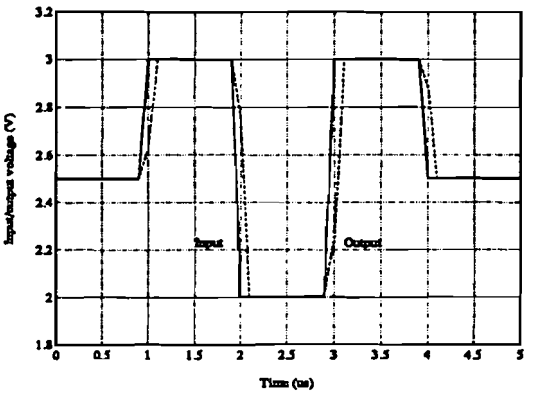



Figure 5.10 *Transient behavior of the output buffer.*

**Measurement results**

The input of the buffer was given a voltage sweep from 2 to 3 V. The bias-voltage $V_{bias}$ was set to 1.35 V. The measured static transfer characteristic of the buffer is depicted in Figure 5.11.

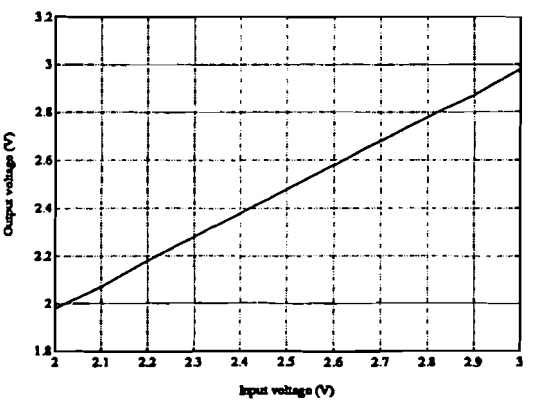



Figure 5.11 *Measured transfer characteristic of the output buffer.*

The offset of all buffers is -20 mV (2 % of the total range), which is very acceptable.

## 5.3 Biascontrol

**Theoretical analysis**

In section 2.1 it was stated that the synapse output (neuron input) has to be shifted over the sigmoid range. In Figure 5.1 this is represented by the block "$\theta_i$" with the "*refresh*"-input. It is also known from section 2.1 that the shifting of the sigmoid function can be considered as an extra synapse with input value "-1" (or "+1", with an inverse weights' value). This principle is worked out directly in an electronic circuit. Therefore we make use of the circuits described in the previous chapter: the two-transistor multiplier, NMOST loads, and the subtraction circuit.

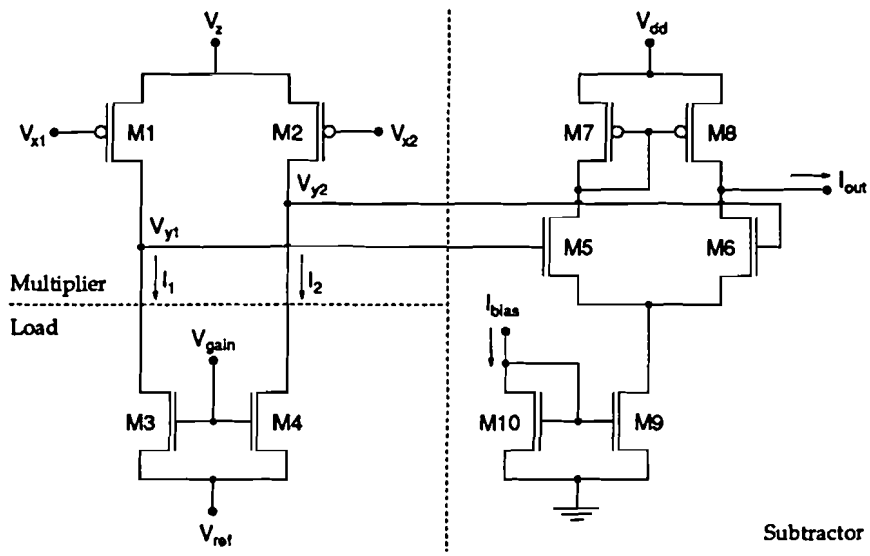



Figure 5.12 *Shifting the sigmoid.*

Because the shift has to cover the total sigmoid shape, its range is twice the sigmoids' range, so the sigmoid can be totally shifted to the left and to the right. The output of the synapse chip (the neuron activity) ranges from -3.5 μA to 3.5 μA (Figure 4.13), so the output of the shifting circuit must be twice this range, -7 μA to 7 μA. The two-transistor multiplier (single synapse) and the subtraction circuit can be kept the same as in the synapse case, but the load (scaling factor) has to be changed. Linearity of $V_z$-$I_{out}$ doesn't play a main part in this case, because only one input value ("-1" or "+1", $V_z$ = 2 or 3 V) of the multiplier is used. The output of the subtraction circuit $I_{out}$ will now be observed as a function of the weights' value ($V_{x1}$ - $V_{x2}$). So to obtain the right shifting range of -7 μA to 7 μA, the resistance of the NMOST-load must be increased. This was done by decreasing the (*W/L*)-ratio of the NMOST's. Therefore the sizes of both NMOS-transistors were chosen (7.2μ/2.4μ).

The implementation area of the bias control circuitry equals the implementation area of a two-transistor multiplier and the area of the subtraction circuit with the NMOST loads, so the chip area is equal to 14,500 $\mu m^2$ and the power consumption is maximally 82.5 $\mu W$ (a more typical value is 72.5 $\mu W$, using the multiplier on average power dissipation).

**Simulation results**

The output $I_{out}$ of the shifting circuit is depicted in Figure 5.13. The applied set ups are:

$V_z$ = 3 V (corresponding with "+1")
$V_{x1}$ : 0 - 1 V
$V_{x2}$ = 0.5 V
$V_{ref}$ = 2.5 V
$V_{gain}$ = 3.85 V (minimum value)
$I_{bias}$ = -1 $\mu A$

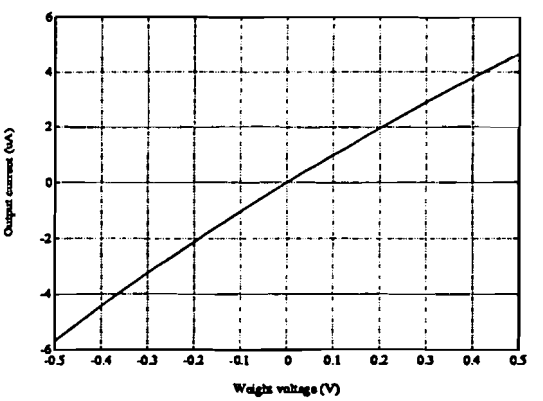



Figure 5.13 *Simulation results of the sigmoid shifting.*

**Measurement results**

On the component chip both circuit parts (a single synapse and the subtractor circuit including the NMOST-load) were connected together. So the deviations of a single synapse were also present in these results. The bias current $I_{bias}$ appoints the offset in the output current of the subtractor. The bias currents of three different copies were determined, in such a way that the output offsets were zero. The bias current of all three copies was now set to the mean value (= - 0.8 $\mu A$). The other parameters ($V_z$, $V_{x1}$, $V_{x2}$, $V_{ref}$, and $V_{gain}$) were set to the same values as used with the simulations, mentioned above.

The results of chip #7 are depicted in Figure 5.14.

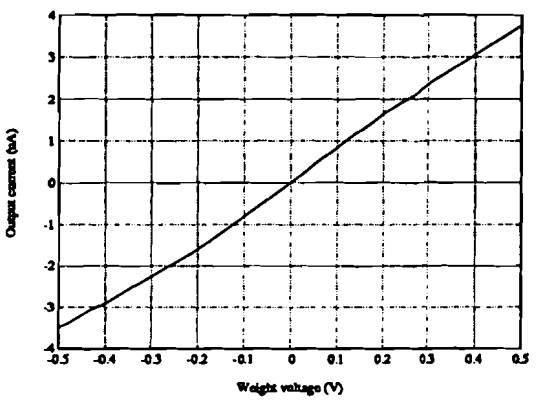



Figure 5.14 *Measurement results of the sigmoid shifting.*

The output current is also halve its expected range: -3.5 to 3.5 μA instead of -7 to 7 μA. See for a possible explanation the review of the measurement results of the subtraction circuit of the synapse chip (section 4.2).

In this case the range of the output current can be enlarged by increasing the input voltage $V_z$, and also the bulk voltage $V_{bulk}$ to 4 V instead of 3 V. This is allowed, because simulations showed that for positive input values of the synapse ($V_z > 2.5$ V) the multiplication was still linear, whereas for negative input values the multiplication became nonlinear. The results of a shifted sigmoid are mentioned in section 5.5.

## 5.4 Current-to-Voltage Converter

**Theoretical analysis**

The sigmoid stage needs a voltage input (see section 5.1), so the current resulting from the summation of the synapse output and the output of the shifting stage has to be converted to a proportional voltage. The Current-to-Voltage Converter (abbreviation: *IV-converter*) addresses itself to this transformation. A simple IV-converter is an Operational Transconductance Amplifier (OTA) with unity feedback. An OTA with feedback in its simplest form [Nauta, 1991] is depicted in Figure 5.15.

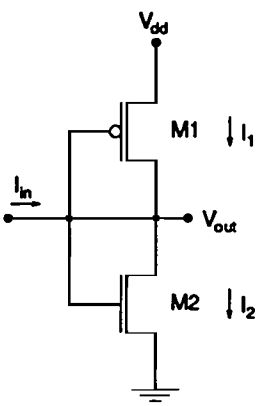



Figure 5.15 *IV-Convertor (simple OTA with feedback).*

Since gate and drain are connected, both transistors M1 and M2 are forced to operate in their saturation region. So the following current equations can be written down.

$$I_1 = \frac{K'_P}{2}\cdot\left(\frac{W}{L}\right)_1\left(V_{gs} + |V_{T1}|\right)^2 = \frac{\beta_1}{2}\cdot\left(V_{out} - V_{dd} + |V_{T1}|\right)^2 \tag{5.8}$$

$$I_2 = \frac{K'_N}{2}\cdot\left(\frac{W}{L}\right)_2\left(V_{gs} - V_{T2}\right)^2 = \frac{\beta_2}{2}\cdot\left(V_{out} - V_{T2}\right)^2 \tag{5.9}$$

The input current $I_{in}$ equals $I_2$ - $I_1$ and if the transistors M1 and M2 are matched, so $\beta_1 = \beta_2 = \beta$ and $|V_{T1}| = V_{T2} = V_T$, then a linear relation between $I_{in}$ and $V_{out}$ exists.

$$V_{out} = \frac{I_{in}}{\beta\cdot\left(V_{dd} - 2V_T\right)} + \frac{V_{dd}}{2} \tag{5.10}$$

A few conditions apply, first, the input range equals the sum of the synapse output (-3.5 μA to 3.5 μA) and the output of the shifting stage (-7 μA to 7 μA), resulting in -10.5 μA to 10.5 μA. The IV-converter should be linear in this range. Second, the sigmoid stage requires an input range of 2 to 3 V, as mentioned in section 5.1 so the output of the IV-converter must also be in this range. If no shifting current is added, the synapse output of -3.5 μA to 3.5 μA should be mapped onto the sigmoid input of 2 to 3 V. In the third place, the IV-converter must have a low impedance input, because this input is connected with a pad of the neuron chip, which has a relatively high parasitic capacitance of a few pF's.

This condition conflicts with the mapping of the IV-converter, because the input impedance of the IV-converter is about 150 kΩ, which is not quite low (the resulting time constant will be in the order of a few hundred nano seconds). To overcome this problem an extra (inverting) stage is added to the IV-converter, so the first stage can be composed of smaller transistors, resulting in a lower input impedance.

The first stage equals the one depicted in Figure 5.15 and converts the input current to a proportional small output voltage (the input impedance now becomes about 3 kΩ). The second stage will be a CMOS inverter, which 'converts' this small voltage range to the sigmoid input range of 2 to 3 V. In Figure 5.16 the resulting circuitry is represented. The implementaion area of the IV-converter equals about 4,200 $\mu m^2$ and the power dissipation is equal to 1.2 mW. The inverting character of the IV-converter doesn't bother the algorithm, because the weights will automatically be updated in the opposite direction.

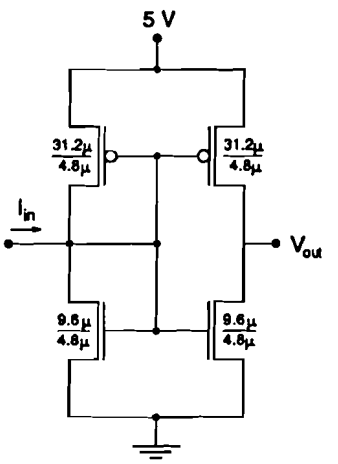



Figure 5.16 *Implemented IV-Converter.*

**Simulation results**

The conversion from current to voltage is depicted in Figure 5.17. The output voltage at an input current between -3.5 µA and 3.5 µA is in the 2 to 3 V range.

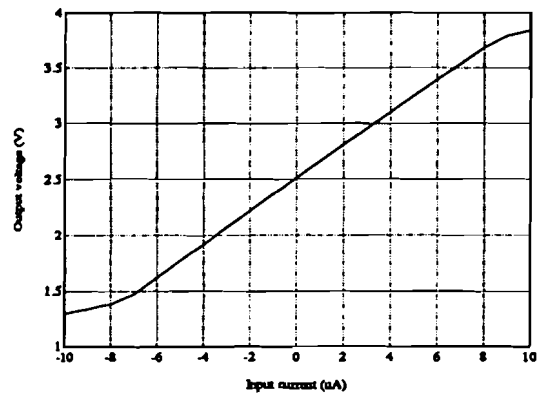



Figure 5.17 *Simulation results of the IV-converter.*

**Measurement results**

In the IV-converter case also chip #5, #6, and #7 were measured. A current between -5.5 μA and 5.5 μA is offered at the input, because in practice this is the maximum input current of the IV-converter. The results are depicted in Figure 5.18.

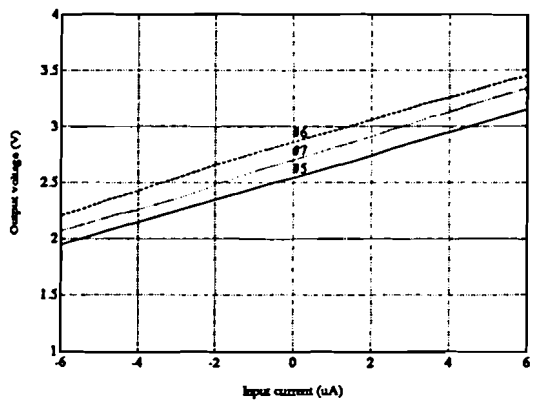



Figure 5.18 *Measurement results of the IV-converter.*

The maximum deviations between the different copies of the IV-converter are about 0.3 V, which will result in a horizontal (bias-weights') shift of the sigmoid. This shift can be compensated by the learning algorithm.

## 5.5 Total Neuron Chip

The circuits described in the previous chapters were now connected together shaping the neuron part resulting in Figure 5.19. The total amount of chip area needed for one neuron is 41,000 $\mu m^2$. Adding the power dissipation of all circuit parts leads to a total power consumption of 2.12 mW per neuron. In the prototype version, 8 neurons were put into one chip. So the chip area of the neuron chip is 8 x 41,000 + 80,000 (selection circuitry) + 72,000 (wires) = 480,000 $\mu m^2$ and the power consumption of the neuron chip equals 17 mW. All communal signals, like the bias currents of the biascontrol and the sigmoid part, the bias voltage, etc., were joined together. Connecting together the bias currents of all 8 copies the same mistake as in the synapse chip-case was made: the current mirror providing the bias current was carried out 8 times, so an 8 times higher current has to be offered to these current inputs which has to be distributed correctly to all 8 neurons. The latter can not be checked, there can only be hoped for.

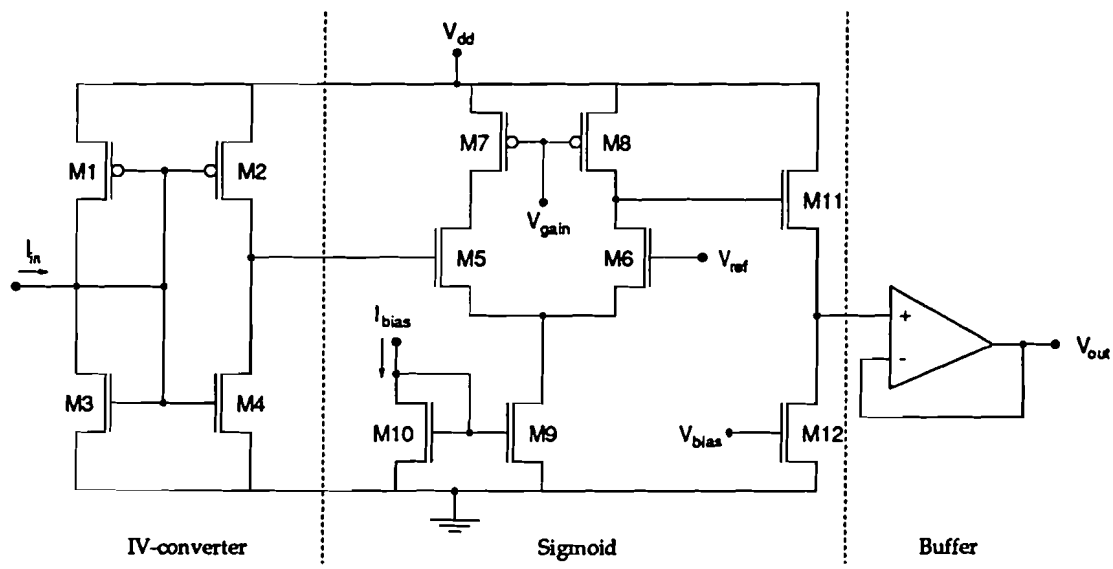



Figure 5.19 *Complete feedforward path of one neuron.*

The main thing that has to be simulated and measured in this section is the shifting ability of the sigmoid. So for different weight values the output curve has to be shifted totally to the left and to the right.

**Simulation results**

For a weights' value ($V_{x1}$ - $V_{x2}$) of 0 V, and an input current range which equals the output current range of the synapse unit (-3.5 μA to 3.5 μA), the output of the neuron unit has to be a sigmoid shape between 2 and 3 V, because it has to be possible to connect (more than) one synapse chip to the neuron chip to build a whole network. The saturation of the sigmoid curve has to take place near the extreme values of the input range (i.e. -3.5 μA to 3.5 μA). By making the weights' value -0.5 V and +0.5 V, the output of the neuron has to become 3 V and 2 V, respectively.

Using the proper set ups for the accompanying circuits mentioned in section 5.1 to 5.4 the results of shifting the sigmoid are depicted in Figure 5.20.

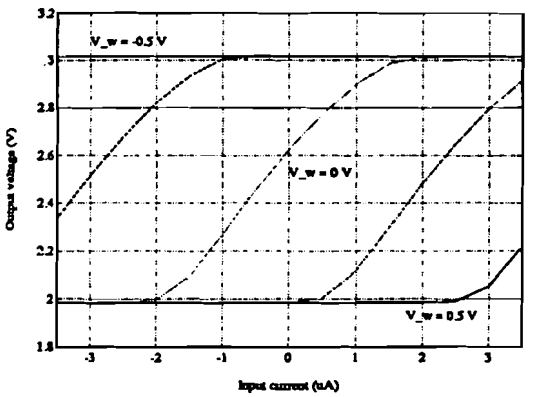



Figure 5.20 *Simulation results of the shifted neuron output.*

**Measurement results**

The practical situation deviates a lot from the simulations, because the output current of the synapse unit is only half the expected value (see section 4.2), and the slope of the sigmoid is two times lower than the foreseen slope. It was impossible to steepen the sigmoid slope (see the measurement results of section 5.1), and it wasn't also possible to increase the output current of the synapse unit by changing the scaling factor $V_{gain}$ ($V_{gain}$ is already set to its minimum value of 3.85 V). The only possibility to increase the output current of the synapse unit is to put two synapses in parallel, reducing the number of synapses to 8 x 4 (this is only a makeshift solution, in the new version of the chips this problem must be corrected by changing the design of the particular circuits).

To enable a larger shift of the sigmoid the input value of the synapse (of the neuron unit) was increased to 4 V (the bulk voltage $V_{bulk}$ of the neuron unit was also increased from 3 V to 4 V).

First the sigmoid shape of the neuron was determined while no shift was added. Second, the sigmoid was shifted by changing the weight input from 0 V to 1 V (step: 0.25 V). To obtain the proper set ups of the neuron unit, the mean values of the parameters of all 8 neurons were appointed. The results are enumerated in Table 5.2.

| | $\#3^0$ | $\#3^1$ | $\#3^2$ | $\#3^3$ | $\#3^4$ | $\#3^5$ | $\#3^6$ | $\#3^7$ | mean |
|---|---|---|---|---|---|---|---|---|---|
| $V_{bias}$ (V) | 1.23 | 1.20 | 1.22 | 1.22 | 1.20 | 1.23 | 1.19 | 1.19 | 1.21 |
| $V_{ref}$ (V) | 2.44 | 2.61 | 2.50 | 2.63 | 2.70 | 2.60 | 2.40 | 2.40 | 2.54 |
| $V_{gain}$ (V) | 1.34 | 1.32 | 1.34 | 1.34 | 1.32 | 1.35 | 1.42 | 1.43 | 1.36 |

Table 5.2 *Set ups of all neurons of the neuron unit.*

The spread in the reference voltages $V_{ref}$ of about 0.3 V can be mainly ascribed to the deviations of the various copies of the current-to-voltage-converter (the IV-converter of section 5.4), which also had a spread of 0.3 V resulting in a horizontal shift of the sigmoid shape. This shift can only be caused by parameter variations, like the threshold voltage, lenghts and widths of the transistors, because no set up voltages or currents are used. $V_{ref}$ has to compensate this shift. The gain voltage of $\#3^6$ and $\#3^7$ is slightly higher than the gain voltage of the other neurons in the chip (about 0.1 V), probably caused by the parallel connection of the current mirrors of $I_{bias}$. The higher its number, the more diodes were put in parallel and this may lower the bias current of the single neuron.

To compensate for this effect the gain voltage has to be increased. The mean values of $V_{bias}$, $V_{ref}$ and $V_{gain}$ were used to measure the various sigmoid shapes. The results are depicted in Figure 5.21.

What catches the eye most is the horizontal shift of the various sigmoid shapes and the varying tales of the sigmoid shape at 2 V. This is caused by taking the mean value for all parameters instead of the exact value.

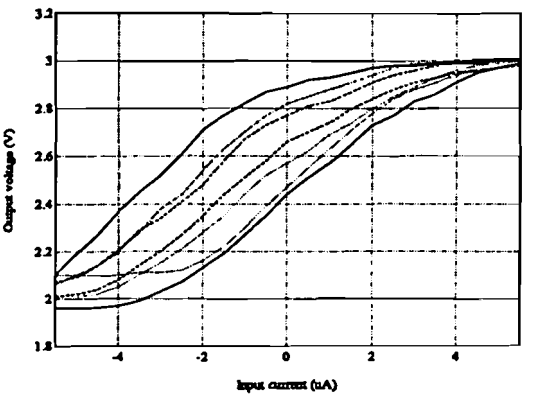



Figure 5.21 *Measurement results of the neuron output of chip #3.*

Because neuron #3[2] is the best one, this neuron was taken to be shifted by sweeping the weight input of the biascontrol from 0 V to 1 V and setting the other input to 4 V (!). The bulk voltage was also connected to 4 V, while the gain voltage of the biascontrol part was set to 3.85 V. The reference voltage $V_{ref}$ equaled still 2.5 V. The results can be seen in Figure 5.22.

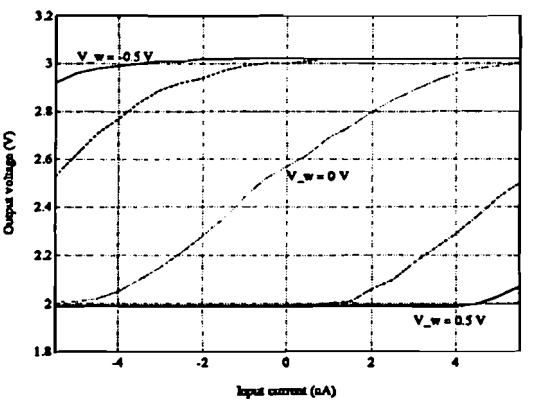



Figure 5.22 *Measurement results of the shifted neuron output.*

The sigmoid is only used in the -2.0 μA to +2.0 μA range, because that's the output of the synapse part. As can be seen in Figure 5.22 it is possible to shift the sigmoid totally to the left and to the right by changing the bias-weight.

# Chapter 6

# Complete Feed-forward Path

*Connecting together synapse and neuron chip, which are described in the previous two chapters, a lot of neural net configurations can be made. In this chapter the connection of one synapse and one neuron chip is discussed. The resulting complete feed-forward path is simulated and tested exhaustively. The results are stated in section 6.1 and 6.2, respectively. In Figure 6.1 a block scheme of this feed-forward path is given.*

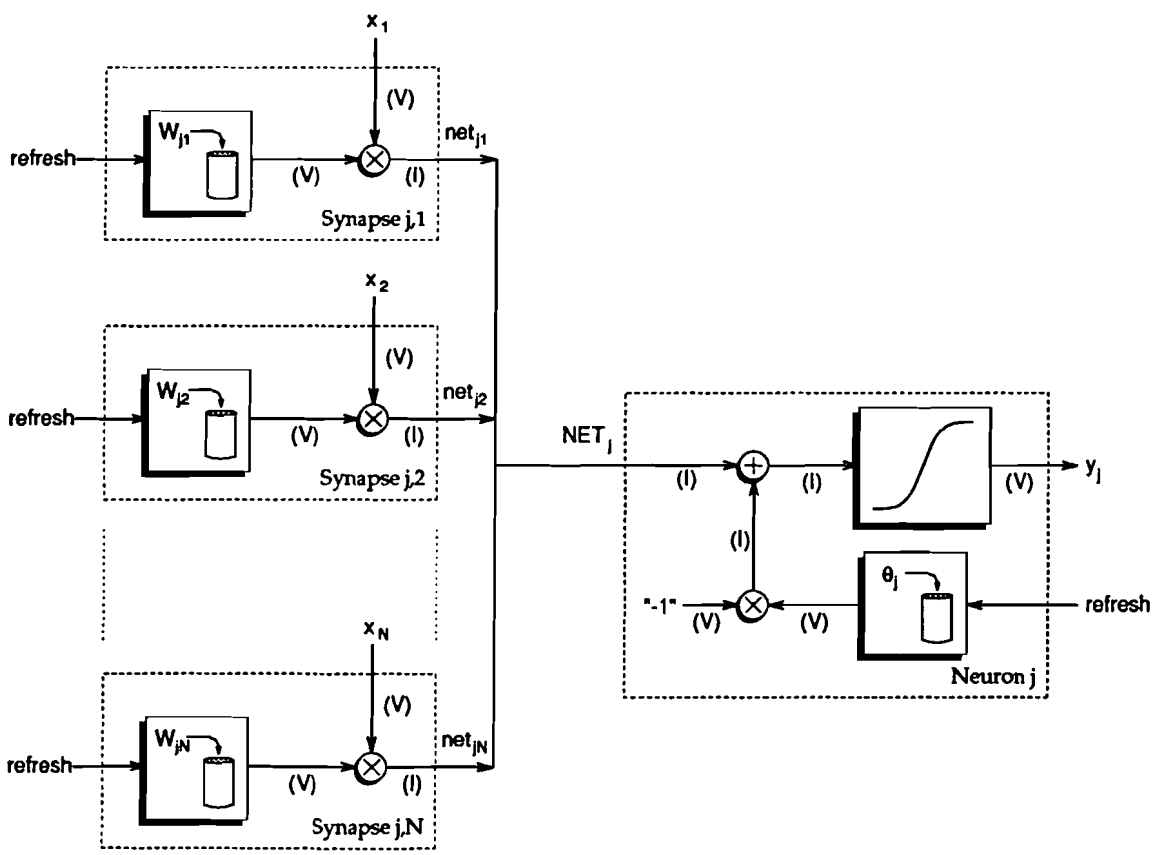



Figure 6.1 *Schematic representation of the connection between synapse and neuron chip.*

## 6.1 Simulation Results

By connecting a synapse and a neuron chip one can observe one neuron with its incoming synapses as already depicted in Figure 2.2. In chapter 4 and chapter 5 the static properties of the synapse and neuron part were described. In this chapter the dynamic properties will be discussed. For reasons of simplicity only one synapse will be connected to the neuron. The problems which arose connecting more then one synapse to the neuron chip are already dealt with in section 4.5. In Figure 6.2 the simulation results of changing the input value from 2 to 3 V (maximum change) for 3 different weight values (-0.5, 0, and 0.5 V) are depicted. The first 3 µs the weights' value is -0.5 V, the second period of 3 µs (from 3 µs to 6 µs) its value is 0 V, and the last 3 µs (from 6 µs to 9 µs) 0.5 V.

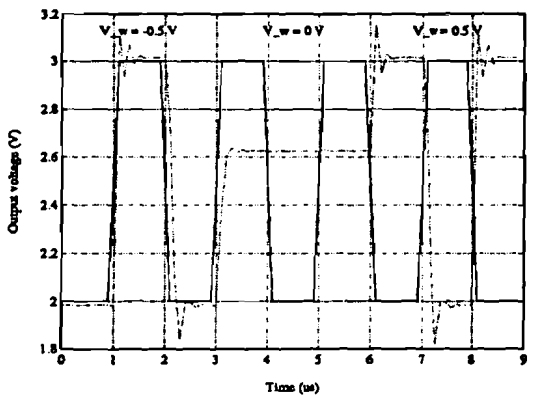



Figure 6.2 *Simulation results of the complete feedforward path.*

The output of the neuron part is connected to a load of 20 pF and 1 MΩ, corresponding with the input impedance of an oscilloscope. So this is the representation of the neurons of the output layer of a neural network. Between the synapse and neuron part a capacitive load of 10 pF is added to simulate the parasitic effect of the synapse-to-neuron-part-connection. To take into account the connection of hidden layers a load of 10 pF and a synapse were added to the output of the neuron part. The worst case situation took place when the output layer was observed. This situation is therefore depicted in Figure 6.2. The maximum propagation-'delay'-time is ≤ 1 µs, which corresponds with a learning rate of about 1 million patterns per second.

## 6.2 Measurement Results

In practice, the synapse and neuron were also connected together. Taking synapse chip #2 and neuron chip #3 the propagation delay is measured for different weight values and sweeping the input from 2 to 3 V. In the worst case situation the delay time was less than 1 μs, corresponding with the simulation results. This experiment was repeated with other copies of both the synapse and neuron chip. The results were almost the same, so the delay time of one layer was measured to be less than 1 μs. The maximum number of patterns per second is more than 1 million using a pipe line technique. If one will not use such a technique the total feed-forward delay time equals the number of layers times 1 μs. In most cases the network will only have 2 layers, resulting in a feed-forward delay time of 2 μs.

## 6.3 Characteristics of the Complete Design

The main results of the static and dynamic measurements of both the synapse and neuron chip are summarized in Table 6.1.

| | | Synapse | Neuron |
|---|---|---|---|
| Size | (single) | 7,300 $\mu m^2$ | 41,000 $\mu m^2$ |
| | (total chip) | 640,000 $\mu m^2$ | 480,000 $\mu m^2$ |
| Power diss. | (single) | 0.01 mW | 2.12 mW |
| | (total chip) | 1.14 mW | 17.0 mW |
| Input range | | 2 - 3 V | -2 - 2 μA |
| Weight range | | 0 - 1 V | 0 - 1 V |
| Output range | | -2 - 2 μA | 2 - 3 V |
| Input offset | | ≤ 3% | ≤ 10% |
| Weight offset | | ≤ 6% | ≤ 6% |
| Output offset | | ≤ 10% | -- |
| Output shift | | -- | ≤ 0.3 V |
| Weight drift | | ≈ 50 mV/s | ≈ 50 mV/s |
| Layer propagation delay | | ≤ 1 μs | |

Table 6.1 *Characteristics of the complete design.*

The size of a single synapse comprises the two-transistor multiplier, the weight storage capacitors and the selection 2-input AND. Dividing the chip area of the total synapse chip by 64 (= 8 x 8), the effective size of one synapse becomes 10,000 $\mu m^2$. In this figure the size of the selection circuitry (a 4-input AND per row and per column) together with the subtraction circuitry (one subtractor per column) were also taken into account. If the synapse chip will be expanded, the overhead of the selection and subtraction circuitry will decrease and so will the effective synapse area.

Calculating the power consumption of the total synapse chip, it was assumed that each synapse of the chip was used and that each synapse consumed average power. In the neuron case the same assumption was done, so each neuron was assumed to be used.

The input range of a neuron equals the output range of a synapse, because the neuron chip is connected to one (or more) synapse chip(s). If there is more than one synapse chip connected to a neuron chip the output range of a single synapse chip will be scaled according to the number of connected synapse chips. The output range of the parallel connection of these synapse chips will now be in the -2 to 2 $\mu A$-range. While a neural network can contain more than one layer, the output of a neuron chip can be the input of a synapse chip. Therefore the output range of the neuron chip equals the input range of the synapse chip.

The range of one side of the weights' input equals 0 to 1 V, while the other side is connected to 0.5 V. So the effective weights' value is equal to -0.5 to 0.5V.

# Chapter 7

# Parallel Stochastic Weight Perturbation

*In this chapter an improvement of the standard Weight Perturbation Algorithm will be discussed: Parallel Stochastic Weight Perturbation. This algorithm is described by Gert Cauwenberghs in his paper 'A Fast Stochastic Error-Descent Algorithm for Supervised Learning and Optimization' [Cauwenberghs, 1993b]. Section 7.1 provides a description of the algorithm in mathematical terms. In section 7.2 the algorithm itself is given, while in section 7.3 the results of some computer simulations are represented. In section 7.4 a comparison with the pure Weight Perturbation Algorithm is discussed and in the final section some algorithm accelerators are given.*

## 7.1 Formulation and Properties

Like the standard Weight Perturbation Algorithm described in chapter 2, Parallel Stochastic Weight Perturbation is also a gradient descent method, so the weights $\underline{w}$ will be updated according to

$$\Delta \underline{w} = -\eta \cdot \frac{\partial E}{\partial \underline{w}} \tag{7.1}$$

Iteration of Eq. (7.1) leads asymptotically to a local minimum of network error $E(\underline{w})$, provided $\eta$ is strictly positive and small. The main disadvantage of the pure Weight Perturbation Algorithm was the huge amount of iterations, because a computation cycle is required for each weight in the neural network. To increase the convergence speed a combination was investigated of pure Weight Perturbation and Model-Free Distributed Learning (supplying time-varying perturbations $\underline{\pi} = \sigma \cdot \underline{b}$ in parallel to all weights in the neural net and correlating these perturbations with the network error $E(\underline{w} + \underline{\pi})$ to form an incremental update $\Delta w$). Actually, this means that for every epoch the algorithm chooses a random direction in the weight space. For each weight only the perturbation direction has to be stored, so just one bit per weight extra is sufficient for the updating of the weights.

The update rule for each weight in the neural net now becomes:

$$\Delta w_{ji} = -\eta \cdot \frac{E(\underline{w} + \sigma \cdot \underline{b}) - E(\underline{w})}{\sigma \cdot b_{ji}} \tag{7.2}$$

where $b_{ji} = random(\{-1, +1\})$ (with equal probability)

The new weights' value will be calculated as:

$$w_{ji}^{new} = w_{ji}^{old} + \Delta w_{ji} \tag{7.3}$$

A set of three properties can be stated for this algorithm [Cauwenberghs, 1993b]:

1. The algorithm performs gradient descent on average
2. The error $E(\underline{w})$ always decreases under an update (7.2) for any $\underline{\pi}$, provided that $|\underline{\pi}|^2$ is "small", and $\eta$ is strictly positive and "small"
3. The maximum attainable average speed of the algorithm is a factor $W$ slower than that of pure gradient descent, but a factor $W$ faster than pure Weight Perturbation (where $W$ is the number of weights in the neural net).

The proofs of these assertions can be found in [Cauwenberghs, 1993b].

## 7.2 Procedure

Using the update rule of Eq. (7.2) a pseudo-Pascal procedure can be formulated as follows:

1. After presenting all the input training patterns $\underline{x}$, the error $E(\underline{w})$ is calculated using the measured outputs $\underline{y}$ and the target patterns $\underline{d}$ according to a certain error function (Eq. (7.6), (7.7) or another suitable function).
2. Generate a random binary vector $\underline{b}[k]$, with $b_i[k] \in \{-1, +1\}$ and the lenght of $\underline{b}$ equals the number of weight in the neural network, so for each weight a random perturbation sign is determined. Add a perturbation $\sigma \cdot b_i[k]$ to each weight ($k$ = iteration number).
3. The resulting error $E(\underline{w} + \sigma \cdot \underline{b})$ is calculated.
4. The weight update $\Delta w_{ji}$ is calculated for each weight according to (7.2).
5. Stop when an acceptable error $\varepsilon$ is reached. If not go to step 1.

The above-mentioned procedure implements the Forward Difference Method (*FDM*) of estimating the gradient.

It is also possible to implement the Central Difference Method (*CDM*), which estimates the gradient by adding a "positive" and a "negative" perturbation to all weights, respectively, and calculating the update according to

$$\Delta w_{ji} = -\eta \cdot \frac{E(\underline{w} + \sigma \cdot \underline{b}) - E(\underline{w} - \sigma \cdot \underline{b})}{2\sigma \cdot b_{ji}} \tag{7.4}$$

where $b_{ji}$ = *random*({-1, +1}) (with equal probability)

So step 4 will be changed to:

4. Subtract a perturbation $\sigma \cdot b_i[k]$ from each weight ($k$ = iteration number).

4'. The resulting error $E(\underline{w} - \sigma \cdot \underline{b})$ is calculated.

4''. The weight update $\Delta w_{ji}$ is calculated for each weight according to (7.4).

## 7.3 Computer Simulations

### 7.3.1 Benchmarks

In order to compare this algorithm with pure Weight Perturbation computer simulation were performed. The program was written in *Turbo Pascal 6.0*. Richard ten Kate's simulations of pure Weight Perturbation [Kate, 1993] were used as a guideline, so the same benchmarks were used: XOR (extended to PARITY-N) and the SINE-problem.

Using *tanh*-neurons (analogous to the neurons of the implemented neural net) the maximum value of the desired outputs (targets) will be 0.9 instead of 1, because otherwise the output neurons should be learned completely in the tail of the sigmoid-function and a target value will never be reached exactly.

**XOR-problem**

This is probably the most used and studied benchmark of comparing different training algorithms. The network structure is tiny and fairly simple, because it consists of only two layers: the output layer which is composed of a single neuron (and 3 weights), and a hidden layer of 2 neurons (and 6 weights). The network has 2 inputs: $x_1$ and $x_2$.

The network was trained with the Parallel Stochastic Weight Perturbation Algorithm (*PSWP*) to 'learn' the XOR-function ($y = x_1 \oplus x_2$) according to:

| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| -1 | -1 | -0.9 |
| -1 | +1 | +0.9 |
| +1 | -1 | +0.9 |
| +1 | +1 | -0.9 |

Table 7.1 *The XOR-function.*

To extend the XOR-problem (also called the PARITY-2 problem) to a more complex PARITY-N problem (with $N = 3, 4, ...$) the network will be expanded to a *N-N-1* structure and the number of patterns to be learned will be $2^N$. The target output value of the network belonging to a certain input pattern will be -0.9 if the number of +1-inputs is even, else (odd +1-inputs) the output will be +0.9. All of the PARITY-N functions are digital, so they can be solved ('learned') more efficiently by other methods and therefore these digital examples are very academic.

**SINE-problem**

The more serious problems are analog ones, of which the SINE-problem is very often chosen as a benchmark. The network has to 'learn' the function $y = 0.8 \cdot sin(x)$ for values of $x$ between $-\pi$ and $+\pi$. 19 equidistant sample points are taken, which are calculated in the following way:

$$x^p = \left( \frac{(p-1)}{9} - 1 \right) \cdot \pi, \; p = 1..19 \qquad (7.5)$$

$$d^p = 0.8 \cdot \sin(x^p)$$

The network which is suitable to train this function is a 1-5-1 MLP containing 6 neurons and 16 weights. The neurons all use *tanh*-neurons. In [Kate, 1993] a 1-3-1 MLP is used with a linear output neuron, but these kind of neurons are not available on the neuron chip which is described in chapter 5. Because different networks are used, the results can not be compared with each other.

It has been assumed that the behavior of PSWP in a ideal and non-ideal MLP is comparable with the behavior of WP in these environments, although this has to be checked as this algorithm is chosen to be implemented on chip (what is left out of consideration in this report). Only the ideal case is simulated, so an ideal MLP is taken.

### 7.3.2 Simulation Results

Since this is an introductory investigation of PSWP, only PARITY-2 and PARITY-3 are trained during simulations, because the other benchmark (SINE-problem) took too much time. If the algorithm seems to be a promising one, then other (more complex) benchmarks can be studied. At first sight, there was no difference in convergence speed between FDM and CDM, therefore FDM is implemented, because FDM needs less perturbations than CDM. First, general observations can be made with regard to the (learning-)parameters: $\eta$ (the learning rate) and $\sigma$ (the magnitude of the perturbations).

The aim of the algorithm is to decrease the network error $E(w)$ at each iteration, according to an estimation of the gradient. The accuracy of this estimation depends on the perturbation stepsize $\sigma$. If $\sigma$ is choosen too small, too much bits will be needed to represent it as a digital signal or in the analog case its value can be below noise level. Therefore it is advisable to choose $\sigma$ large enough, but also not too large, because then the estimation of the gradient becomes worse. This manifests itself in a very noisy error function, which results in a noisy network output. So the optimal perturbation step is the largest value for which the error function is still smooth. This value also has to be chosen as large as possible, because it accelerates the convergence speed a lot. The optimal value is the one for which the error function contains no spikes, because when $\eta$ is chosen too large the weight updates are also too large, which can cause an increase of the network error instead of a decrease. If $\eta$ is chosen extremely large the algorithm gets a kind of random search behavior, which can be very fast for simple problems like the XOR-problem, but the number of converged trials will decrease tremendously and therefore these solutions are not useful.

One of the advantages of using this kind of algorithms is that it is possible to define more than one error criterion. Eq. (7.6) renders the most commonly used error criterion for update algorithms, the mean-squared error.

$$E(\underline{w}) = \frac{1}{P \cdot M_K} \cdot \sum_{p=1}^{P} \sum_{j=1}^{M_K} \left(d_{j,p} - y_{j,p}^{K}\right)^2 \tag{7.6}$$

This error evaluation function can be used for any kind of problem (digital, analog and classification). In digital cases it is also possible to make use of more complex error functions, like the *log*-criterion of Eq. (7.7). The error landscape belonging to this error criterion is more slanting than the surface belonging to the mean-squared error criterion [Unnikrishnan, 1992].

$$E(\underline{w}) = -\frac{1}{P \cdot M_K} \cdot \sum_{p=1}^{P} \sum_{j=1}^{M_K} \ln\left(\frac{y_{j,p}^K + 1}{2}\right), \qquad IF\ d_{j,p} = 1$$

$$E(\underline{w}) = -\frac{1}{P \cdot M_K} \cdot \sum_{p=1}^{P} \sum_{j=1}^{M_K} \ln\left(1 - \frac{y_{j,p}^K + 1}{2}\right), \qquad IF\ d_{j,p} = -1 \tag{7.7}$$

where $y_{j,p}^K$ = actual output from neuron j for pattern p
$d_{j,p}$ = desired output from neuron j for pattern p
$M_K$ = number of neurons in the last layer K
$P$ = number of patterns

Both error criteria were used for the digital problems PARITY-2 and PARITY-3.

**PARITY-2 (XOR-) problem**

First of all the *M*ean-Squared Error (*MSE*) criterion of Eq. (7.6) was applied in the PARITY-2 case. The optimal learning rate $\eta$ was determined by taking $\sigma$ small enough ($10^{-4}$) and raising $\eta$ with a small amount every 20 runs until an unstable learning process was reached. A problem is learned when the network error equals $10^{-3}$, so every output differs only about $3 \cdot 10^{-2}$ from its desired value of -0.9 or +0.9 for each learned input pattern.

The number of epochs was obtained from 20 runs of which 1 or 2 did not converge within 2000 epochs. Only the converged runs were used to calculate the mean number of epochs. The optimal learning rate for the XOR-problem using the mean-squared error criterion is 0.18. The mean number of epochs for convergence using the optimal learning rate, equals 257.

The same experiments were done for the XOR-problem with a *log*-error criterion. A run was stopped when the network error was less than $10^{-2}$, corresponding with a maximum deviation of $1 \cdot 10^{-2}$ between each learned output value and its desired output value for each weight. The desired output values were now ±1 instead of the ±0.9 in the mean-squared case.

The optimal learning rate is about 0.3. Of each 20 runs 4 or 5 did not converge within 2000 epochs, so the mean number of epochs was obtained from 15 or 16 runs. The mean number of epochs using the optimal learning rate, equals 338.

**PARITY-3 problem**
Performing the same experiments as mentioned above the optimal learning rates for the PARITY-3 problem are $\eta_{MSE} \approx 0.15$ and $\eta_{log} \approx 0.3$. The mean number of epochs belonging to the MSE- and the log-error are 322 (1 not converged) and 411 (all converged), respectively.

**PARITY-4 problem**
It was also tried to 'learn' the neural net the PARITY-4 problem, but the number of converged runs (number of epochs < 5000) decreased dramatically (< 50%), while the mean number of iterations per converged run increased tremendously (> 2000). The learning rate has to be made very small ($\leq 10^{-2}$), which is one of the causes of the increased number of iterations per converged run.

**Optimal perturbation step**
After the optimal learning rates were determined, the perturbation step was increased in such a way that the converge speed of each problem didn't decrease substantially. The maximum allowed perturbation stepsize for all problems equals $10^{-2}$.

**Comparison between MSE- and log-error**
The results of using the *MSE*- or *log*-error can not be compared directly, because in the *MSE*-case the desired outputs are only ±0.9, while in the *log*-case the outputs must be learned into the tail of the sigmoid, namely ±1. The other difference concerns the inequality of both maximum allowed errors. Training the net using the *MSE*-error the maximum allowed difference between actual and desired outputs may be $3 \cdot 10^{-2}$, while in the *log*-case this difference may only be $1 \cdot 10^{-2}$. Because of these two differences it can be assumed that it will take less training cycles to learn a particular function using the *MSE*-error than using the *log*-error. The above mentioned experiments confirm the assertion, because the problems trained with the *MSE*-critic used about 80 epochs less than the ones trained with the *log*-critic.

## 7.4 Comparison with Weight Perturbation

In [Kate, 1993] the minimum number of epochs required for convergence of the PARITY-2 (XOR-) problem using Weight Perturbation (WP) equals 143, by using a learning rate of 0.1 and the stop criterion equals $10^{-3}$.

PSWP required about 300 epochs for convergence, using a learning rate of about 0.15 and the learning process was also stopped when the network error reached the value of $10^{-3}$. So the PSWP algorithm needs approximately twice as much epochs to converge compared with the pure WP algorithm. However, an epoch of the WP algorithm means a presentation of the input patterns and an update procedure for each separate weight in the network, as the PSWP algorithm needs only one presentation of the input patterns and an update procedure for all weights at one time. In the XOR-case this means that the number of epochs of PSWP may be 9 times the number of epochs of WP, because the MLP used for the XOR-problem holds 9 weights. Knowing this and taking it into account, PSWP converges about 4 times faster than WP in the XOR-case. The error criterion used in [Kate, 1993] is not the same as used in this report. In this report the error is normalized by dividing it by the number of output neurons ($M_K$) and the number of patterns ($P$). So to compare both stop criteria, the one used in this report should be multiplied by $M_K \cdot P$. The mean number of epochs needed to converge in this case is about 250. This makes PSWP about 5 times faster than WP.

## 7.5 Modifications of the algorithm

**Simplifying the implementation**

To simplify the implementation of the algorithm, it is tried to remove one of the parameters $\sigma$ or $\eta$. This is done by throwing away some of the gradient information, namely the steepness of the gradient. The update procedure is now according to:

$$\begin{aligned} \Delta w_{ji} &= +\sigma \cdot b_{ji} \quad IF\ E\left(w_{ji} + \sigma \cdot b_{ji}\right) - E\left(w_{ji}\right) < 0 \\ \Delta w_{ji} &= -\sigma \cdot b_{ji} \quad IF\ E\left(w_{ji} + \sigma \cdot b_{ji}\right) - E\left(w_{ji}\right) \geq 0 \end{aligned} \tag{7.8}$$

where $b_{ji} = random([-1, +1])$ (with equal probability)

So, if the network error belonging to the perturbed weights is smaller than the one belonging to the unperturbed weight set, the new weights are the perturbed ones, else the weights are updated in the opposite direction. The algorithm can not be accelerated in the way PSWP can be done, because it lacks a learning rate parameter. The only parameter that can be changed is the perturbation step $\sigma$. By making this parameter smaller or larger the convergence speed can be influenced. The main advantage of this modification of the PSWP algorithm is the easier implementation: only a comparison between two network errors (the perturbed ($E(\underline{w} + \underline{\pi})$ and unperturbed error ($E(\underline{w})$) has to be done. The difficult multiplication of the learning rate (divided by the perturbation step) (a relatively large number) with the difference between the perturbed and unperturbed network error (a very small signal) is now removed.

The perturbation step has to be chosen (very) small, what is automatically the main 'disadvantage' of this approach. In the beginning the algorithm will not converge so fast, but at the and the convergence will be faster, because it uses a constant stepsize. The optimal perturbation step will be the maximum one for which the algorithm can converge to the maximum allowed error $\varepsilon$, because when $\sigma$ is taken to large it can not reach the required accuracy. The simulation results of using this modification are printed in Table 7.2:

| Problem | Error-criterion | $\sigma_{opt}$ | #Epochs | | | | |
|---|---|---|---|---|---|---|---|
| | | | Mean | S.D. | Min | Max | N.C. |
| PARITY-2 | MSE | 0.03 | 269 | 32 | 198 | 325 | 1 |
| | log | 0.03 | 318 | 31 | 291 | 421 | 4 |
| PARITY-3 | MSE | 0.02 | 273 | 35 | 228 | 373 | 1 |
| | log | 0.02 | 329 | 58 | 251 | 543 | 0 |

Table 7.2 *Simulation results of the modified algorithm (#trials = 20).*

Comparing these results with the standard Parallel Stochastic Weight Perturbation Algorithm, the convergence behavior is quite the same. The mean number of epochs until convergence is reached is approximately the same, while the number of non-converged runs are also comparable.

**Speeding up the algorithm**

Since the above mentioned modification converges faster than the original PSWP algorithm, an accelerator for this algorithm was designed. To improve the convergence behavior of the algorithm the perturbation step for the first epochs has to be chosen larger, therefore the perturbation will be multiplied by $(1 + \lambda \cdot E(\underline{w}))$. The network error $E(\underline{w})$ for the first epochs is relatively high, so the the weights will be updated a lot, while at the end the network error decreases and so does the update of the weights. The simulation results of using this accelerator on the XOR-problem are enumerated next.

| Problem | Error-criterion | $\sigma_{opt}$ | $\lambda_{opt}$ | #Epochs | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | Mean | S.D. | Min | Max | N.C. |
| PARITY-2 | MSE | 0.03 | 3 | 162 | 25 | 127 | 217 | 2 |
| | log | 0.03 | 5 | 148 | 39 | 100 | 268 | 3 |
| PARITY-3 | MSE | 0.02 | 3 | 319 | 77 | 226 | 530 | 0 |
| | log | 0.02 | 5 | 305 | 90 | 212 | 586 | 0 |

Table 7.3 *Simulation results of the accelerated modified algorithm (#trials = 20).*

Determining $(1 + \lambda \cdot E(\underline{w}))$ is quite a heuristic operation, as can be seen in the PARITY-3 case. In this case the results became even worse using the multiplication factor $(1 + \lambda \cdot E(\underline{w}))$. Perhaps, choosing $\sigma$ slightly smaller and $\lambda$ a little larger will increase the convergence speed.

In the XOR-case the number of epochs is decreased with almost 100 epochs using both the *MSE*- and *log*-error criterion.

# Chapter 8

# Conclusions and Recommendations

*This chapter contains the conclusions, which are drawn on the basis of the electronic aspects of chapter 3. The errors made during the design of the chips, are translated into recommendations.*

## 8.1 Conclusions

In this report the design of a neural network chip is described. Two chips were processed:

- an 8 x 8 synapse chip and
- a neuron chip containing 8 neurons.

Recalling the aspects of an electronic implementation of chapter 3 the following conclusions can be drawn:

1. Power dissipation: each single synapse has a power consumption of only 10 μW, and each single neuron needs about 2 mW of power.
2. Implementation area: the effective synapse implementation area equals 10,000 $\mu m^2$, while the effective chip area of a single neuron is equal to 60,000 $\mu m^2$.
3. Number of required pins: these prototype versions need about 40 pins per chip.
4. Response time: the delay (response) time of the complete feed-forward path (synapse + neuron part) is less than 1 μs, so the pattern throughput is about $10^6$ patterns/sec.
5. Input-, output ranges: both inputs of the synapse can be swept 1 V (input: from 2 to 3 V, and weight: from 0 to 1 V). The input of the neuron chip has a range of -2 to 2 μA.
6. Susceptibility to parameter variations: comparing the simulated with the measured figures of both synapse and neuron part, it is clear that the neuron part is most sensitive to parameter variations. The slope of the sigmoid function is about 2 times smoother than the simulated one. This deviation can be partly imputed to the parameter variations of the circuit parts of the neuron unit.

8. Flexibility and cascadability: using a separate synapse and neuron chip the flexibility is maximal (an arbitrary network configuration can be made) and by scaling the synapse output according to the number of connected synapses to one neuron, cascadability is also maximal.

The difference between simulation results and measurement results is about 50 %. This is quite a lot, but for reasons of a maximum speed, feedback loops are avoided. The only feedback loop that is present in the design is the output buffer of the neuron unit. The algorithm which will train the net has to provide the network feedback, so the algorithm must be able to handle with the 50 % difference between simulations and measurements.

To train the neural network the modified Parallel Stochastic Weight Perturbation Algorithm is recommended, because this algorithm is much faster than standard Weight Perturbation and the implementation is much easier. It also requires less memory to store the weight updates.

## 8.2 Recommendations

Designing a chip, a lot of things can go wrong. The things that went wrong during this design are enumerated next:

- During the layout phase, all connections between different layers (poly 1 or 2, metal 1 or 2, active, etc.) were made by using only one contact. To get a better connection, as much as possible mimimum sized contacts must be used.
- The number of pins determines the final implementation area of the chip, because the chip area occupied by the pads, which are used to connect a pin to the chip, is generally much larger than the chip area occupied by the actual design. The empty space between the pins and the actual design, in which the connections are made, can be filled with metal 1. Connecting this surface to ground, bulk contacts can be created easily around the actual design.

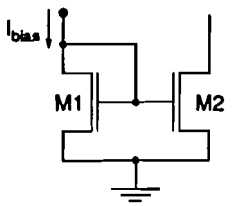



Figure 8.1 *Current mirror.*

- The pins which were connected to a diode of a current mirror (M1 in Figure 8.1), were not protected, because they were connected both to a gate but also to a drain/source. This appeared to be wrong. These pins were also connected to the gates of the other transistor of the current mirror (M2 in Figure 8.1) and these gates will break down when there's a charge on the particular pin.
- As mentioned a few times before in this report, the diode parts of the current mirrors of a particular signal (for instance $I_{bias}$) used in this design were erroneously connected together. In the new design only one diode must be used.
- The current mirrors were all composed of minimum-sized transistors. Small deviations in the width and/or length of the transistors that form the current mirror caused a large deviation of the mirror ratio.
- The selection of the weights of the synapse chip was done by 4-input NMOS-AND's. The voltage drop over the various transistors appeared to be too large. Therefore, it is recommended to use CMOS-types of the 4-input AND's instead of NMOS-types, although these circuits require a lot more chip area. Chip area isn't the main point in the column-and-row-selection-case, because only one 4-input AND per column and one per row are required. The 2-input AND's still have to be very small (NMOS-types), because these gates are carried out once per synapse (weight).
- Particularly the neuron part (IV-converter and sigmoid stage) has to be made less sensitive to parameter variations. Maybe, this problem can partly be solved by choosing larger sizes of the transistors used in these critical circuit parts.
- Finally, it is very important to use the most recent parameter list of the fabrication process. This design was made with a list of May 1990! At this moment of list of May 1993 is available at the group. This isn't still recent enough. Before starting a new design, one has to claim the most recent list of IMEC in Leuven, because one is only able to make a proper design, when one can perform realistic simulations!

# Literature

**[Bruin, 1993]**
Bruin, P.P.F.M.
*A Weight Perturbation Neural Net Chip Set*
Master Thesis, Faculty of Electrical Engineering, Eindhoven University of Technology, 1993

**[Cauwenberghs, 1993a]**
Cauwenberghs, G.
*A Learning Analog Neural Network Chip with Continuous-Time Recurrent Dynamics*
Available via FTP

**[Cauwenberghs, 1993b]**
Cauwenberghs, G.
*A Fast Stochastic Error-Descent Algorithm for Supervised Learning and Optimization*
Available via FTP

**[Claassen, 1993]**
Claassen-Vujčić, T.
*Implementation of a Multi-Layer Perceptron Using Pulse-Stream Techniques*
Master Thesis, Faculty of Electrical Engineering, Eindhoven University of Technology, 1993

**[Eberhardt, 1989]**
Eberhardt, S. et al.
*Design of Parallel Hardware Neural Network Systems from Custom Analog VLSI 'Building Block' Chips*
Int. Joint Conf. on Neural Networks (Washington, DC), June 18-22 1989, Vol. 2, pp. 183-190

**[Hegt, 1993]**
Hegt, J.A.
*Hardware Implementations of Neural Networks*
Available via FTP

**[Jabri, 1992]**
Jabri, M.A. and B. Flower
*Weight Perturbation: An Optimal Architecture and Learning Technique for Analog VLSI Feedforward and Recurrent Multilayer Networks*
IEEE Trans. on Neural Networks, Vol. 3 (1992), No. 1, p. 154-157

**[Kate, 1993]**
Kate, R.E. ten
*A Study of the Weight Perturbation Algorithm Used in Neural Networks*
Master Thesis, Faculty of Electrical Engineering, Eindhoven University of Technology, 1993

**[Lippmann, 1987]**
Lippmann, R.P.
*An Introduction to Computing with Neural Nets*
IEEE ASSP Magazine (1987), pp. 4-22

**[Maren, 1990]**
Maren, A.J. et al.
*Handbook of Neural Computing Applications*
London: Academic Press, 1990. - XIX, 448 p.

**[Minsky, 1969]**
Minsky, M. and S. Papert
*Perceptrons: An Introduction to Computational Geometry*
Cambridge: M.I.T. Press, 1969. - 258 p.

**[Nauta, 1991]**
Nauta, B.
*Analog CMOS Filters for Very-High Frequencies*
Ph.D. Thesis, pp. 95-96, September 1991

**[Nguyen, 1990]**
Nguyen, D. and B. Widrow
*Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights*
Proc. Intl. Joint Conf. on Neural Networks, San Diego, CA, June 1990

**[OWH, 1994]**
Oosse, J.M.C., H.C.A.M. Withagen and J.A. Hegt
*Analog VLSI Implementation of a Feed-Forward Neural Network*
To be published at ISCECS 1994, Cairo

**[Rumelhart, 1986]**
Rumelhart, D.E. et al.
*Learning Internal Representations by Error Propagation*
In: *Parallel Distributed Processing*
Vol. 1: Foundations, pp. 318-362
Ed. by J.L. McClelland and D.E. Rumelhart
Cambridge, MA: MIT Press, 1986

**[Teulings, 1991]**
Teulings, P.M.W.
*Analog Electronic Implementation of Multi-Layer Neural Networks Including Back-Propagation Training Algorithm*
Master Thesis, Faculty of Electrical Engineering, Eindhoven University of Technology, 1991

**[Unnikrishnan, 1992]**
Unnikrishnan, K.P. and K.P. Venugopal
*Learning in Connectionist Networks Using the Alopex Algorithm*
Proc. Intl. Joint Conf. on Neural Networks, Vol. 1, pp. 926-931

**[Vittoz, 1991]**
Vittoz, E.A. et. al.
*Analog Storage of Adjustable Synaptic Weights*
In: *VLSI Design of Neural Networks*
E. by U. Ramacher and U. Rückert
Dordrecht: Kluwer Academic Publishers, 1991
The Kluwer Int. Series in Engineering and Computer Science

**[Withagen, 1994]**
Withagen, H.C.A.M.
*Reducing the Effect of Quantization by Weight Scaling*
Accepted at ICNN 1994, Orlando, Florida

## Epilogue (In Dutch)

# De dingen hebben hun geheim...[1]

*"Our little systems have their day*
*They have their day and cease to be*
*They are but broken lights of Thee*
*And Thou o Lord art more than they."*

-- Alfred Tennyson

Ik wil dit verslag besluiten met een persoonlijke bijdrage. Gedurende mijn afstudeerperiode bij de vakgroep EEB heb ik me niet alleen verdiept in de technische (elektronische) kant van de kunstmatige neurale netwerken, maar is mijn interesse ook gewekt voor de meer filosofische kant van dit vakgebied. Nadenkend over zoiets als kunstmatige intelligentie, komt men vrij snel uit bij vragen als: *"Kunnen computers, machines echt denken?"* en *"Wat is eigenlijk intelligentie?"*. Nu pretendeer ik niet dat ik de antwoorden op deze (en meer van dit soort) vragen heb gevonden, maar ik wil u mijn mening hierover niet onthouden.

Vanaf het begin van de ontwikkeling van de computer heeft de mens deze machine vergeleken met zijn eigen kwaliteiten. Dit resulteerde in een zeer suggestieve naamgeving voor tal van processen in de computer. Zo werd bijvoorbeeld de opslagfunctie "geheugen" genoemd. Daarnaast ging men op een antropomorfe manier (als betrof het een mens) over de computer spreken: "Hij is even bezig...", "Hij doet het niet...", "Hij zegt..." (dus is het zo!), enz. Door van begin af aan de mens met de computer te vergelijken, is er een computeromorfe visie (als betrof het een computer) op de mens ontstaan. De mens is gereduceerd tot een 'rekenmachine' met een opslagcapaciteit. Via in- en outputs wisselt de mens informatie met zijn omgeving uit, op dezelfde wijze als een computer. De mens: een informatieverwerkend systeem!?

Deze manier van denken werd versterkt door de ontwikkeling van de kunstmatige intellligentie, die in de jaren vijftig haar opgeld deed. Een typisch menselijke eigenschap als intelligentie (wat dat dan ook mag zijn) werd gevangen in een technisch systeem, nadat het meetbaar gemaakt was in de vorm van het Intelligentie Quotiënt (IQ).

Ook hier werden weer suggestieve namen gebruikt, zoals 'expertsystemen': systemen die beslissingen overnemen van experts en die 'redeneren' als experts en 'neurale netwerken': wiskundige modellen van de hersenen die dezelfde struktuur hebben als de hersenen en daarmee toch wel intelligent moeten zijn. Ze 'leren' zelfs!

Men vraagt zich misschien af: "Wat is hier dan mis mee? De mens denkt toch en als we willen weten hoe de mens denkt, kunnen we toch machines maken die dat denken nabootsen. Door nu deze simulaties te vergelijken met het denken van de mens komen we misschien wel tot een model van de hersenen. Dat zou toch fantastisch zijn. Wat kunnen we dan allemaal wel niet doen!"

Waartoe dit alles kan leiden, wil ik demonstreren aan de hand van een voorbeeld dat ik vond in Joseph Weizenbaum's 'Computerkracht & Mensenmacht'[2]. De schrijver, zelf programmeur, heeft in de jaren zestig een programma geschreven waarmee het mogelijk was om als mens tot op een bepaalde hoogte met een computer te communiceren. Het programma werd een bepaald script gegeven, waardoor de computer op een bepaalde manier reageerde op hetgeen de gebruiker via zijn toetsenbord intypte. De computer was zo bijvoorbeeld in staat om een zogenoemde Rogeriaanse psycho-therapeut te imiteren. Hierbij bestaat de strategie van reageren uit het terugkaatsen van datgene wat de gebruiker intypt. De computer reageerde bijvoorbeeld volgens dit script op de opmerking van de gebruiker: "Alle mannen zijn eender" op de volgende manier: "In welk opzicht?" Op deze wijze werd de gebruiker uit zijn tent gelokt.

Op dit programma, ELIZA genaamd, kwamen tal van enthousiaste reakties, ondermeer van een aantal praktizerende psychiaters die vonden dat het programma een eerste begin was om te komen tot een computer-geleide psycho-therapie. Zij zagen het als dé oplossing van het tekort aan psycho-therapeuten. De psychiater: een informatieverwerker!?

Wat is dan het verschil tussen een mens en een computer? Ik zal niet ontkennen dat beide informatie verwerken, maar het verschil is dat de informatie die de computer verwerkt geen betekenis heeft voor de computer zelf. John Searle[3] heeft dit geprobeerd duidelijk te maken met zijn beroemd geworden 'Chinese-kamer-experiment'. Bij dit gedachtenexperiment wordt een persoon opgesloten in een kamer. Door de brievenbus worden vellen papier geduwd met daarop allerlei onbegrijpelijke Chinese karakters. In een instructieboek moet de persoon die tekens opzoeken en door nieuwe tekens vervangen en daarna de vellen met de nieuwe tekens terugsturen. Later blijkt dat de tekens die door de brievenbus binnenkwamen vragen in het Chinees waren en dat de tekens die terug werden gestuurd werden opgevat als de antwoorden op die vragen. Sterker nog, de teruggestuurde antwoorden waren niet te onderscheiden van de antwoorden die iemand zou geven die echt Chinees kan lezen.

Is dit niet precies wat een computer doet, formele, betekenisloze tekens omzetten in andere formele, betekenisloze tekens zonder te begrijpen waarom het zo moet.

Bovenstaand argument gaat ook op voor meer geavanceerde systemen, zoals 'kunstmatige neurale netwerken', hoewel nog niet iedereen daarvan overtuigd is, getuige een artikel in de Automatiseringsgids van 29 oktober 1993[4]. Hierin beweert een promovendus bij het Gerechtelijk Laboratorium het volgende:

> "De mystiek waarmee neurale netwerken omgeven zijn wordt mede veroorzaakt door de gedistribueerde representatie van informatie. Neurale netwerken zijn 'pattern crunchers'. In tegenstelling tot de componenten in een symbolische of numerieke representatie zijn de componenten van een patroon niet abstract. Het patroon *heeft* een betekenis, een symbolische of numerieke representatie *verwijst* altijd naar een betekenis."

Maar ook in dit geval betekenen de patronen natuurlijk niets voor het neurale netwerk zelf. Het maakt voor het neurale netwerk niet uit welke patronen het moet 'leren', daarnaast blijft ieder patroon natuurlijk een symbolische of numerieke representatie van het werkelijke gegeven. Als het neurale netwerk bijvoorbeeld een onderscheid moet aanbrengen tussen appels en peren, zullen deze vruchten moeten worden geabstraheerd tot meetbare grootheden, zoals de vorm van de vrucht, de kleur van de vrucht, etc. Het neurale netwerk kan dan aan de hand van deze grootheden bepalen of de vrucht een appel of een peer is.

Uiteraard heb ik bovenstaande niet opgeschreven om hiermee het onderzoek naar kunstmatige neurale netwerken te diskwalificeren of als waardeloos af te schilderen. Integendeel, maar ik heb willen duidelijk maken dat er een essentieel verschil is tussen een mens en een computermodel/-simulatie: informatie heeft voor de mens een intrinsieke betekenis, voor een machine ('neuraal netwerk') heeft ze dit niet. Dit komt doordat iedere machine een creatie van mensen is en de mens zichzelf nooit zal kunnen herhalen, of om het met Wolfgang Stegmüller te zeggen: "Wir werden niemals über ein vollständiges Erklärungsmodell für den Menschen verfügen, andernfalls hätten wir uns in eine neue Spezies transformiert"[5]. Alleen als men oog krijgt voor dit geheim, kan het onderzoek op het gebied van de kunstmatige intelligentie zich zinvol ontwikkelen.

## NOTEN

1. Deze titel is ontleend aan het gelijknamige boek van Prof. A. van den Beukel, waarin hij de lezer op een zeer openhartige wijze laat zien dat er achter de wetenschapper 'Prof. van den Beukel" een mens schuilgaat. Een mens die gevormd is door zijn omgeving, maar vooral door zijn geloof. Met name dit laatste aspect heeft grote gevolgen voor zijn denken als wetenschapper.

2. In zijn boek 'Computerkracht en Mensenmacht' (een vertaling van 'Computer Power and Human Reason', 1976) bespreekt Joseph Weizenbaum de herschepping van de wereld naar het evenbeeld van de computer. Hij verdedigt ook op dit punt zijn ethische grondhouding op het gebied van de computertechniek: "er zijn bepaalde taken die men computers niet *behoort* te laten doen, ongeacht het feit of men computers die *kan* laten doen."

3. Recentelijk stond er nog een interview met John Searle in de Volkskrant (14 mei 1994), waarin hem, naar aanleiding van zijn laatste boek 'The Rediscovery of the Mind', naar zijn mening werd gevraagd over kunstmatige intelligentie. Hij is hierover zeer negatief en zegt ondermeer dat kunstmatige intelligentie een uit de hand gelopen leugen is, ooit bedoeld om geld voor onderzoek te verwerven. Het is volgens hem namelijk ondenkbaar dat computers ooit zullen denken. Het artikel eindigt met: "Maar onze geest is geen computerprogramma en ons brein is geen computer, dat is zo overduidelijk, dat zal iedereen tenslotte wel gaan inzien. Niemand kan ongestraft tegen zo'n simpel inzicht van het gezond verstand ingaan."

4. In deze uitgave van de Automatiseringsgids waren een aantal artikelen opgenomen over kunstmatige intelligentie. Hierbij ging het vooral om praktische toepassingen van neurale netwerken. In het artikel waaruit het citaat is overgenomen werd het neurale netwerk van Eurocard besproken. Eurocard maakt namelijk sinds medio december 1993 gebruik van een neuraal netwerk voor het bestrijden van fraude met creditcards.

5. Zie Wolfgang Stegmüller, *Neue Wege der Wissenschaftsphilosophie*, Berlin 1980.

# Acknowledgements (In Dutch)

De resultaten die zijn beschreven in dit verslag waren nooit bereikt zonder de hulp van mijn begeleider Hans Hegt. Hans, bedankt voor alle tijd die je voor me vrijmaakte en nog bedankt voor 'de Chinees'.

Ook Heini wil ik bedanken voor al zijn 'support' en belangstelling en bovenal voor het 'in elkaar draaien' van het paper.

# Appendix A

# MOS-parameters

| Parameter | Value | Parameter | Value |
|---|---|---|---|
| Level | 2 | Cjsw | 201p |
| L | 2.4u | Mjsw | 0.333 |
| Tox | 40.6n | Rsh | 33.36 |
| Kp | 51.2u | Nfs | 1.26E11 |
| Vto | 0.899 | Dl | -0.04u |
| Nsub | 1.21E15 | Dw | -0.065u |
| Gamma | 0.236 | Ld | 0.22u |
| Phi | 0.651 | Cgso | 180p |
| Uo | 614 | Cgdo | 180p |
| Delta | 1.05 | Cgbo | 220p |
| Cj | 75.8u | Uexp | 0.065 |
| Mj | 0.37 | Lambda | 63.1E-3 |
| Pb | 0.502 | Vearly | 6.6E6 |

Table A.1 *NMOS parameters.*

| Parameter | Value | Parameter | Value |
|---|---|---|---|
| Level | 2 | Mjsw | 0.495 |
| L | 2.4u | Rsh | 32.8 |
| Tox | 42.5n | Nfs | 3.927E11 |
| Kp | 19.2u | Dl | -0.156u |
| Vto | -0.834 | Dw | -0.139u |
| Nsub | 9.713E15 | Ld | 0.35u |
| Gamma | 0.699 | Cgso | 280p |
| Phi | 0.637 | Cgdo | 280p |
| Uo | 234.2 | Cgbo | 340p |
| Delta | 0.951 | Uexp | 0.1 |
| Cj | 310u | Vmax | 1.5E5 |
| Mj | 0.5 | Ucrit | 1E4 |
| Pb | 0.751 | Lambda | 37.9E-3 |
| Cjsw | 328.3p | Vearly | 11E6 |

Table A.2 *PMOS parameters.*

# Appendix B

# PSPICE-file

```
Feed-forward path

X1    3 4 5 6 1     Synapse_1
X2    4 3 7         Subtractor1

X3    13 14 5 11 2 Synapse_1
X4    14 13 7       Subtractor

*CL1   7 0           10pF
X5    7 8           IV_Conv
X6    8 9           Sigmoid

X7    9 10          Buffer
*X8   61 61 5 5 1    Synapse_32
*V61   61 0          DC 2.5V
*RL2   10 0          1MEG
*CL2   10 0          20pF

* Voltage sources
Vx1     6   0 DC 0.5V
Vx11   11   0 DC 0.5V
Vx2     5   0 DC 0.5V
Vz1     2   0 DC 3V
Vz      1   0 DC 2.5V

.SUBCKT Synapse_1 3 4 5 6 8
   * 1 Synapse
   m1    3   5   8   1 P10_0 w=2.4u AD=27p AS=27p PD=24u PS=24u
   m2    4   6   8   1 P10_0 w=2.4u AD=27p AS=27p PD=24u PS=24u

   * Voltage sources
   V1   1   0 DC 3V
.ENDS

.SUBCKT Synapse_8 3 4 5 6 8
   * 8 Synapses; Factor = Square_Root(N=8)
   m1    3   5   8   1 P10_0 w=6.8u AD=77p AS=77p PD=35u PS=35u
   m2    4   6   8   1 P10_0 w=6.8u AD=77p AS=77p PD=35u PS=35u
   * 8 Synapses
   *m1    3   5   8   1 P10_0 w=19.2u AD=216p AS=216p PD=66u PS=66u
   *m2    4   6   8   1 P10_0 w=19.2u AD=216p AS=216p PD=66u PS=66u

   * Voltage sources
   V1   1   0 DC 3V
.ENDS
```

```
.SUBCKT Synapse_32 3 4 5 6 8
   * 32 Synapses; Factor = Square_Root(N=32)
   m1   3  5  8  1 P10_0 w=13.6u AD=153p AS=153p PD=52u PS=52u
   m2   4  6  8  1 P10_0 w=13.6u AD=153p AS=153p PD=52u PS=52u
   * 32 Synapses
   *m1   3  5  8  1 P10_0 w=76.8u AD=864p AS=864p PD=210u PS=210u
   *m2   4  6  8  1 P10_0 w=76.8u AD=864p AS=864p PD=210u PS=210u

   * Voltage sources
   V1  1  0 DC 3V
.ENDS

.SUBCKT Subtractor1  3 4 6
   * Transistors
   m1  11 11  8  1 P10_0 W=3.6u AD=41p AS=41p PD=27u PS=27u
   m2   6 11  8  1 P10_0 W=3.6u AD=41p AS=41p PD=27u PS=27u

   m3   4  5  7  2 N2_4  W=2.4u AD=27p AS=27p PD=24u PS=24u
   m4   3  5  7  2 N2_4  W=2.4u AD=27p AS=27p PD=24u PS=24u
   m31  4  0  7  1 P2_4  W=2.4u AD=27p AS=27p PD=24u PS=24u
   m41  3  0  7  1 P2_4  W=2.4u AD=27p AS=27p PD=24u PS=24u

   m5  11  4 10  2 N2_4  W=48u AD=540p AS=540p PD=138u PS=138u
   m6   6  3 10  2 N2_4  W=48u AD=540p AS=540p PD=138u PS=138u
   m7  10  9  0  2 N2_4  W=24u AD=270p AS=270p PS=78u PD=78u
   m8   9  9  0  2 N2_4  W=2.4u AD=27p AS=27p PD=24u PS=24u

   * Voltage sources
   V1  1  0 DC 5V
   V2  2  0 DC 0V
   V5  5  0 DC 3.85V
   V7  7  0 DC 2.5V
   V8  8  0 DC 5V

   * Current source
   I9  0  9 1uA
.ENDS

.SUBCKT Subtractor 3 4 6
*   3 V_y2          4 V_y1          5 V_gain        6 I_out
*   7 V_ref        99 vdd           9 I_bias
*
   m1  11 11  8  1 P10_0 W=3.6u AD=41p AS=41p PD=27u PS=27u
   m2   6 11  8  1 P10_0 W=3.6u AD=41p AS=41p PD=27u PS=27u

   m3   4  5  7  2 N2_4  W=7.2u AD=81p AS=81p PD=36u PS=36u
   m4   3  5  7  2 N2_4  W=7.2u AD=81p AS=81p PD=36u PS=36u
   m5  11  4 10  2 N2_4  W=48u AD=540p AS=540p PD=138u PS=138u
   m6   6  3 10  2 N2_4  W=48u AD=540p AS=540p PD=138u PS=138u
   m7  10  9  0  2 N2_4  W=24u AD=270p AS=270p PS=78u PD=78u
   m8   9  9  0  2 N2_4  W=2.4u AD=27p AS=27p PD=24u PS=24u

   * Voltage sources
   V1  1  0 DC 5V
   V2  2  0 DC 0V
   V5  5  0 DC 3.85V
   V7  7  0 DC 2.5V
   V8  8  0 DC 5V

   * Current source
   I9  0  9 1uA
.ENDS
```

```
.SUBCKT IV_Conv 4 5
   M1    4  4  3  1 P4_8  W=31.2u AD=338p AS=338p PD=93u PS=93u
   M2    4  4  0  2 N4_8  W=9.6u AD=135p AS=135p PD=48u PS=48u

   M3    5  4  3  1 P4_8  W=31.2u AD=338p AS=338p PD=93u PS=93u
   M4    5  4  0  2 N4_8  W=9.6u AD=135p AS=135p PD=48u PS=48u

   * Voltage sources
   V1  1  0 DC 5V
   V2  2  0 DC 0V
   V3  3  0 DC 5V
.ENDS

.SUBCKT Sigmoid 3 12
   * Differential stage with linear load
   M1    6  3  5  2 N2_4  W=12u AD=135p AS=135p PD=48u PS=48u
   M2    7  4  5  2 N2_4  W=12u AD=135p AS=135p PD=48u PS=48u
   M3    6  8  9  1 P10_0 W=3.6u AD=41p AS=41p PD=27u PS=27u
   M4    7  8  9  1 P10_0 W=3.6u AD=41p AS=41p PD=27u PS=27u

   * Current mirror
   M6    5 11  0  2 N2_4 W=27.2u AD=306p AS=306p PD=86u PS=86u
   M61 11 11  0  2 N2_4 W=2.4u AD=27p AS=27p PD=24u PS=24u

   * Level shifter
   M8    9  7 12 2 N2_4  W=2.4u AD=27p AS=27p PD=24u PS=24u
   M81 12 10  0 2 N2_4  W=5.6u AD=63p AS=63p PD=32u PS=32u

   * Voltage sources
   V1  1  0 DC 5V
   V2  2  0 DC 0V
   V4  4  0 DC 2.5V
   V9  9  0 DC 5V
   V8  8  0 DC 1.35V
   V10 10 0 DC 1.35V

   * Current source
   IBias   0  11    1uA
.ENDS

* Buffer
.SUBCKT Buffer 5 1
     * Opamp
   MO1  6  1  7  0 N2_4 W=2.4u AD=27p AS=27p PD=24u PS=24u
   MO2  8  5  7  0 N2_4 W=2.4u AD=27p AS=27p PD=24u PS=24u
   MO3  6  6 99 99 P2_4 W=2.4u AD=27p AS=27p PD=24u PS=24u
   MO4  8  6 99 99 P2_4 W=2.4u AD=27p AS=27p PD=24u PS=24u
   MO5  1  8 99 99 P2_4 W=48u  AD=540p AS=540p PD=138u PS=138u
   MO6  1  9  0  0 N2_4 W=48u  AD=540p AS=540p PD=138u PS=138u
   MO7  7  9  0  0 N2_4 W=7.2u AD=81p AS=81p PD=36u PS=36u

   * Miller compensation
   MM1 99  1  2  0 N2_4 W=2.4u AD=27p AS=27p PD=24u PS=24u
   MM2  2  9  0  0 N2_4 W=2.4u AD=27p AS=27p PD=24u PS=24u
   CM          8  2 200fF

   * Voltage sources
   Vsup   99  0 DC 5V
   Vbias   9  0 DC 1.35V
.ENDS
```

```
* Models

* NMOST with L=2.4um and Vearly=6.6V/um: Lambda=1/(L*Vearly)=63.1E-3
.MODEL N2_4 NMOS(Level=2 L=2.4u Tox=40.6n Kp=51.2u Vto=0.899 Nsub=1.21E15
+ Gamma=0.236 Phi=0.651 Uo=614 Delta=1.05 Cj=75.8u Mj=0.37 Pb=0.502
+ Cjsw=201p Mjsw=0.333 Rsh=33.36 Nfs=1.26E11 Dl=-0.04u Dw=-0.065u
+ Ld=0.22u Cgso=180p Cgdo=180p Cgbo=220p Uexp=0.065 Lambda=63.1E-3)

* NMOST with L=4.8um and Vearly=6.6V/um: Lambda=1/(L*Vearly)=32.6E-3
.MODEL N4_8 NMOS(Level=2 L=4.8u Tox=40.6n Kp=51.2u Vto=0.899 Nsub=1.21E15
+ Gamma=0.236 Phi=0.651 Uo=614 Delta=1.05 Cj=75.8u Mj=0.37 Pb=0.502
+ Cjsw=201p Mjsw=0.333 Rsh=33.36 Nfs=1.26E11 Dl=-0.04u Dw=-0.065u
+ Ld=0.22u Cgso=180p Cgdo=180p Cgbo=220p Uexp=0.065 Lambda=32.6E-3)

* NMOST with L=6um and Vearly=6.6V/um: Lambda=1/(L*Vearly)=25.2E-3
.MODEL N6_0 NMOS(Level=2 L=6u Tox=40.6n Kp=51.2u Vto=0.899 Nsub=1.21E15
+ Gamma=0.236 Phi=0.651 Uo=614 Delta=1.05 Cj=75.8u Mj=0.37 Pb=0.502
+ Cjsw=201p Mjsw=0.333 Rsh=33.36 Nfs=1.26E11 Dl=-0.04u Dw=-0.065u
+ Ld=0.22u Cgso=180p Cgdo=180p Cgbo=220p Uexp=0.065 Lambda=25.2E-3)

* NMOST with L=24um and Vearly=6.6V/um: Lambda=1/(L*Vearly)=6.31E-3
.MODEL N24_0 NMOS(Level=2 L=24u Tox=40.6n Kp=51.2u Vto=0.899 Nsub=1.21E15
+ Gamma=0.236 Phi=0.651 Uo=614 Delta=1.05 Cj=75.8u Mj=0.37 Pb=0.502
+ Cjsw=201p Mjsw=0.333 Rsh=33.36 Nfs=1.26E11 Dl=-0.04u Dw=-0.065u
+ Ld=0.22u Cgso=180p Cgdo=180p Cgbo=220p Uexp=0.065 Lambda=6.31E-3)

* NMOST with L=48um and Vearly=6.6V/um: Lambda=1/(L*Vearly)=3.15E-3
.MODEL N48_0 NMOS(Level=2 L=48u Tox=40.6n Kp=51.2u Vto=0.899 Nsub=1.21E15
+ Gamma=0.236 Phi=0.651 Uo=614 Delta=1.05 Cj=75.8u Mj=0.37 Pb=0.502
+ Cjsw=201p Mjsw=0.333 Rsh=33.36 Nfs=1.26E11 Dl=-0.04u Dw=-0.065u
+ Ld=0.22u Cgso=180p Cgdo=180p Cgbo=220p Uexp=0.065 Lambda=3.15E-3)

* PMOST with L=2.4um and Vearly=11V/um: Lambda=1/(L*Vearly)=37.9E-3
.MODEL P2_4 PMOS(Level=2 L=2.4u Tox=42.5n Kp=19.2u Vto=-0.834 Nsub=9.713E15
+ Gamma=0.699 Phi=0.637 Uo=234.2 Delta=0.951 Cj=310u Mj=0.5 Pb=0.751
+ Cjsw=328.3p Mjsw=0.495 Rsh=32.8 Nfs=3.927E11 Dl=-0.156u Dw=-0.139u Ld=0.35u
+ Cgso=280p Cgdo=280p Cgbo=340p Uexp=0.1 Vmax=1.5E5 Ucrit=1E4 Lambda=37.9E-3)

* PMOST with L=4.8um and Vearly=11V/um: Lambda=1/(L*Vearly)=19E-3
.MODEL P4_8 PMOS(Level=2 L=4.8u Tox=42.5n Kp=19.2u Vto=-0.834 Nsub=9.713E15
+ Gamma=0.699 Phi=0.637 Uo=234.2 Delta=0.951 Cj=310u Mj=0.5 Pb=0.751
+ Cjsw=328.3p Mjsw=0.495 Rsh=32.8 Nfs=3.927E11 Dl=-0.156u Dw=-0.139u Ld=0.35u
+ Cgso=280p Cgdo=280p Cgbo=340p Uexp=0.1 Vmax=1.5E5 Ucrit=1E4 Lambda=19E-3)

* PMOST with L=6um and Vearly=11V/um: Lambda=1/(L*Vearly)=15.2E-3
.MODEL P6_0 PMOS(Level=2 L=6u Tox=42.5n Kp=19.2u Vto=-0.834 Nsub=9.713E15
+ Gamma=0.699 Phi=0.637 Uo=234.2 Delta=0.951 Cj=310u Mj=0.5 Pb=0.751
+ Cjsw=328.3p Mjsw=0.495 Rsh=32.8 Nfs=3.927E11 Dl=-0.156u Dw=-0.139u Ld=0.35u
+ Cgso=280p Cgdo=280p Cgbo=340p Uexp=0.1 Vmax=1.5E5 Ucrit=1E4 Lambda=15.2E-3)

* PMOST with L=10um and Vearly=11V/um: Lambda=1/(L*Vearly)=9.1E-3
.MODEL P10_0 PMOS(Level=2 L=10u Tox=42.5n Kp=19.2u Vto=-0.834 Nsub=9.713E15
+ Gamma=0.699 Phi=0.637 Uo=234.2 Delta=0.951 Cj=310u Mj=0.5 Pb=0.751
+ Cjsw=328.3p Mjsw=0.495 Rsh=32.8 Nfs=3.927E11 Dl=-0.156u Dw=-0.139u Ld=0.35u
+ Cgso=280p Cgdo=280p Cgbo=340p Uexp=0.1 Vmax=1.5E5 Ucrit=1E4 Lambda=9.1E-3)

* PMOST with L=12um and Vearly=11V/um: Lambda=1/(L*Vearly)=7.6E-3
.MODEL P12_0 PMOS(Level=2 L=12u Tox=42.5n Kp=19.2u Vto=-0.834 Nsub=9.713E15
+ Gamma=0.699 Phi=0.637 Uo=234.2 Delta=0.951 Cj=310u Mj=0.5 Pb=0.751
+ Cjsw=328.3p Mjsw=0.495 Rsh=32.8 Nfs=3.927E11 Dl=-0.156u Dw=-0.139u Ld=0.35u
+ Cgso=280p Cgdo=280p Cgbo=340p Uexp=0.1 Vmax=1.5E5 Ucrit=1E4 Lambda=7.6E-3)
```

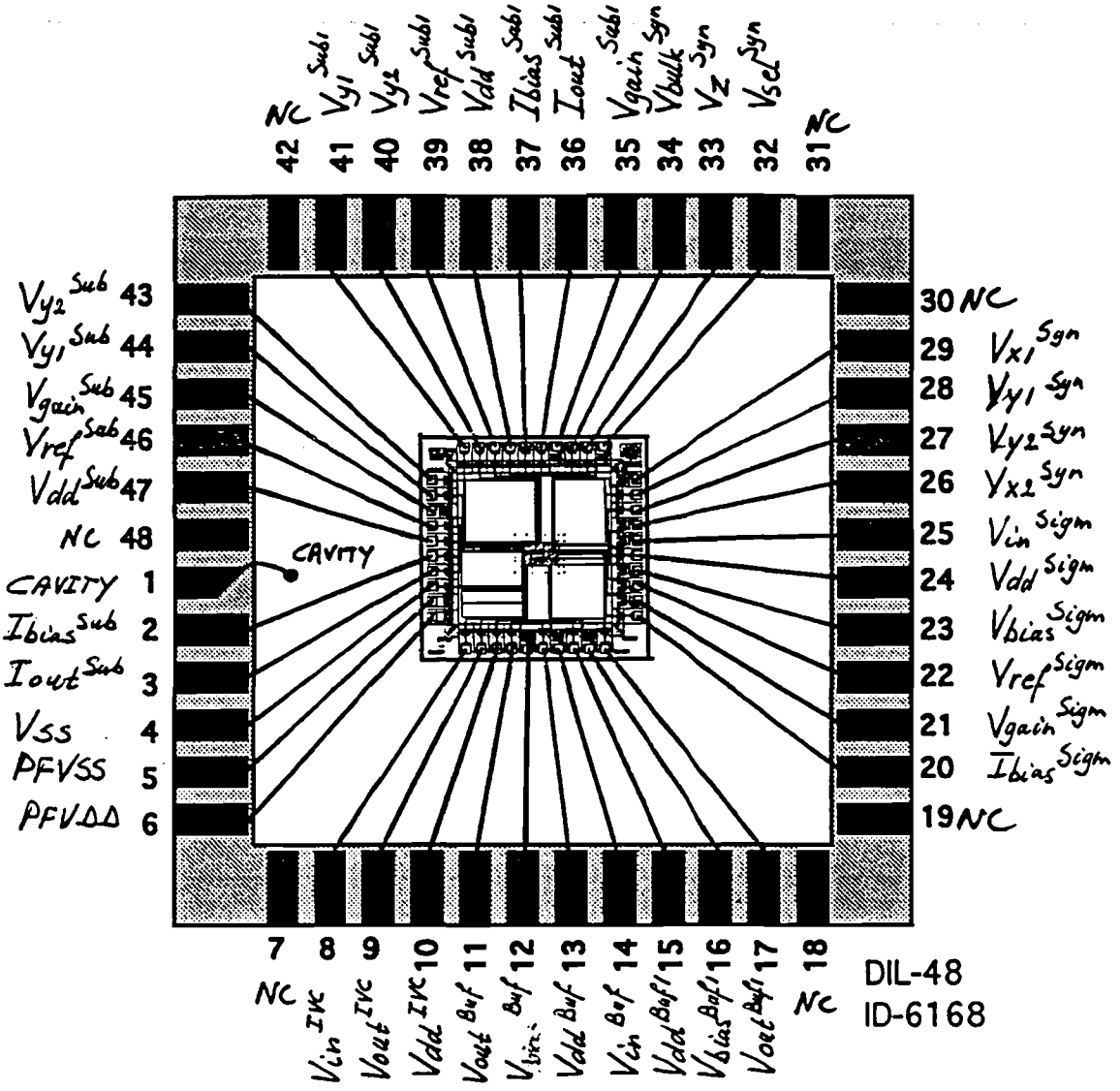
```
* PMOST with L=24um and Vearly=11V/um: Lambda=1/(L*Vearly)=3.79E-3
.MODEL P24_0 PMOS(Level=2 L=24u Tox=42.5n Kp=19.2u Vto=-0.834 Nsub=9.713E15
+ Gamma=0.699 Phi=0.637 Uo=234.2 Delta=0.951 Cj=310u Mj=0.5 Pb=0.751
+ Cjsw=328.3p Mjsw=0.495 Rsh=32.8 Nfs=3.927E11 Dl=-0.156u Dw=-0.139u Ld=0.35u
+ Cgso=280p Cgdo=280p Cgbo=340p Uexp=0.1 Vmax=1.5E5 Ucrit=1E4 Lambda=3.79E-3)

* PMOST with L=48um and Vearly=11V/um: Lambda=1/(L*Vearly)=1.9E-3
.MODEL P48_0 PMOS(Level=2 L=48u Tox=42.5n Kp=19.2u Vto=-0.834 Nsub=9.713E15
+ Gamma=0.699 Phi=0.637 Uo=234.2 Delta=0.951 Cj=310u Mj=0.5 Pb=0.751
+ Cjsw=328.3p Mjsw=0.495 Rsh=32.8 Nfs=3.927E11 Dl=-0.156u Dw=-0.139u Ld=0.35u
+ Cgso=280p Cgdo=280p Cgbo=340p Uexp=0.1 Vmax=1.5E5 Ucrit=1E4 Lambda=1.9E-3)

* DC-analysis
*.STEP Vx11 0 1 .25
.STEP Vx1 0 1 .25
.DC Vz 2 3 .01
.OP

.OPTIONS RelTol=1E-7
.PROBE
.END
```

Appendix C

# Bonding Diagrams

EUR330

# BONDING DIAGRAM




DIL-48
ID-6168

Comp_Chip

REMARKS: Not connected pins: 7-18-19-30-31-42
48

Pin 1 with Cavity

SCALE:10

3805-01

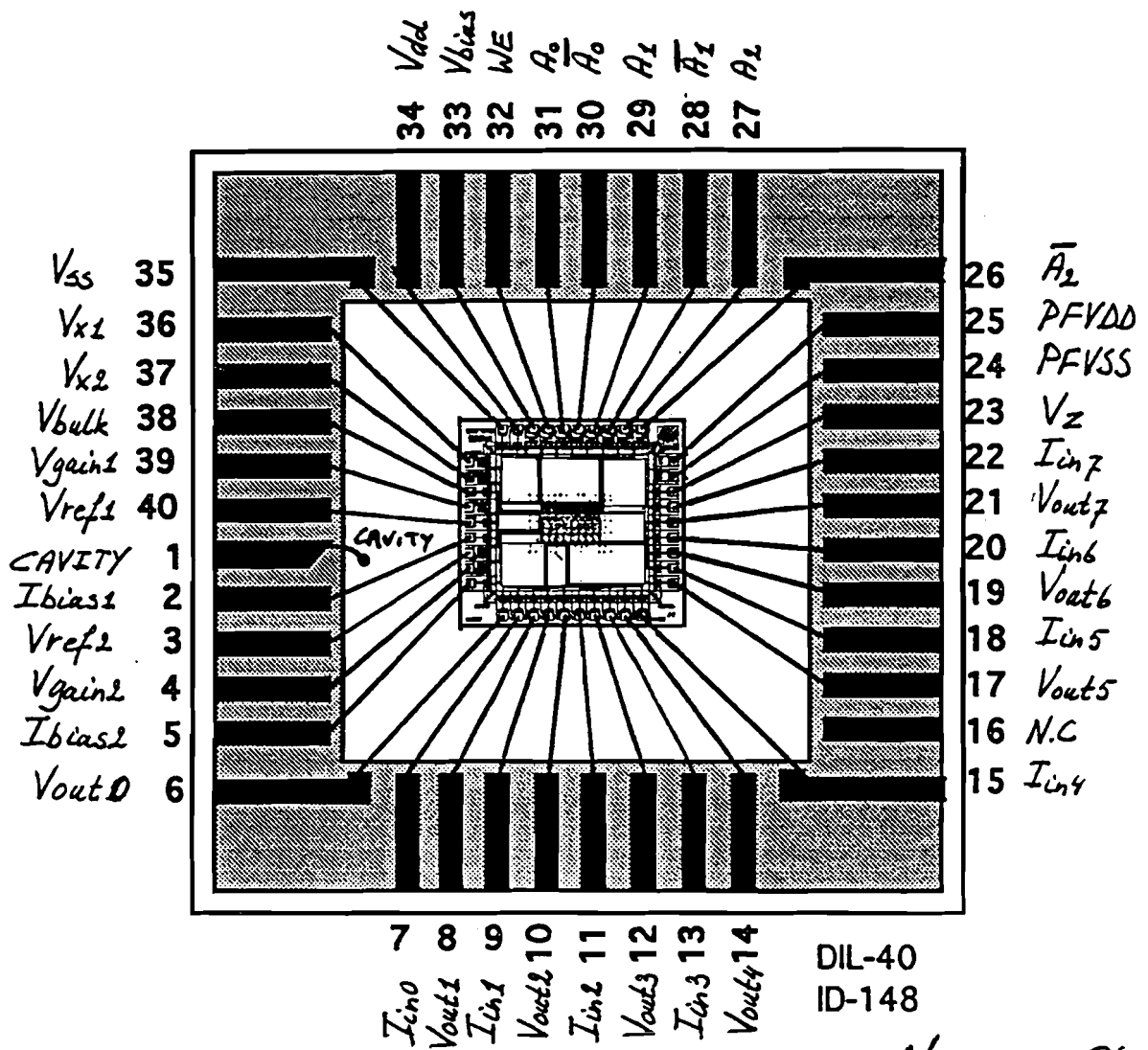
DIE size : 3168µ x 3202µ
MAX. WIRE LENGTH: 4mm

SEE ALSO LAY-OUT RULES FOR PACKAGING
OF DEVICES IN DOCUMENT NUMBER 40001

IMEC

EUR 331

# BONDING DIAGRAM




DIL-40
ID-148

Neuron_Chip

REMARKS: Not connected pin : 16
Pin 1 connected with CAVITY

SCALE:10

3805-02

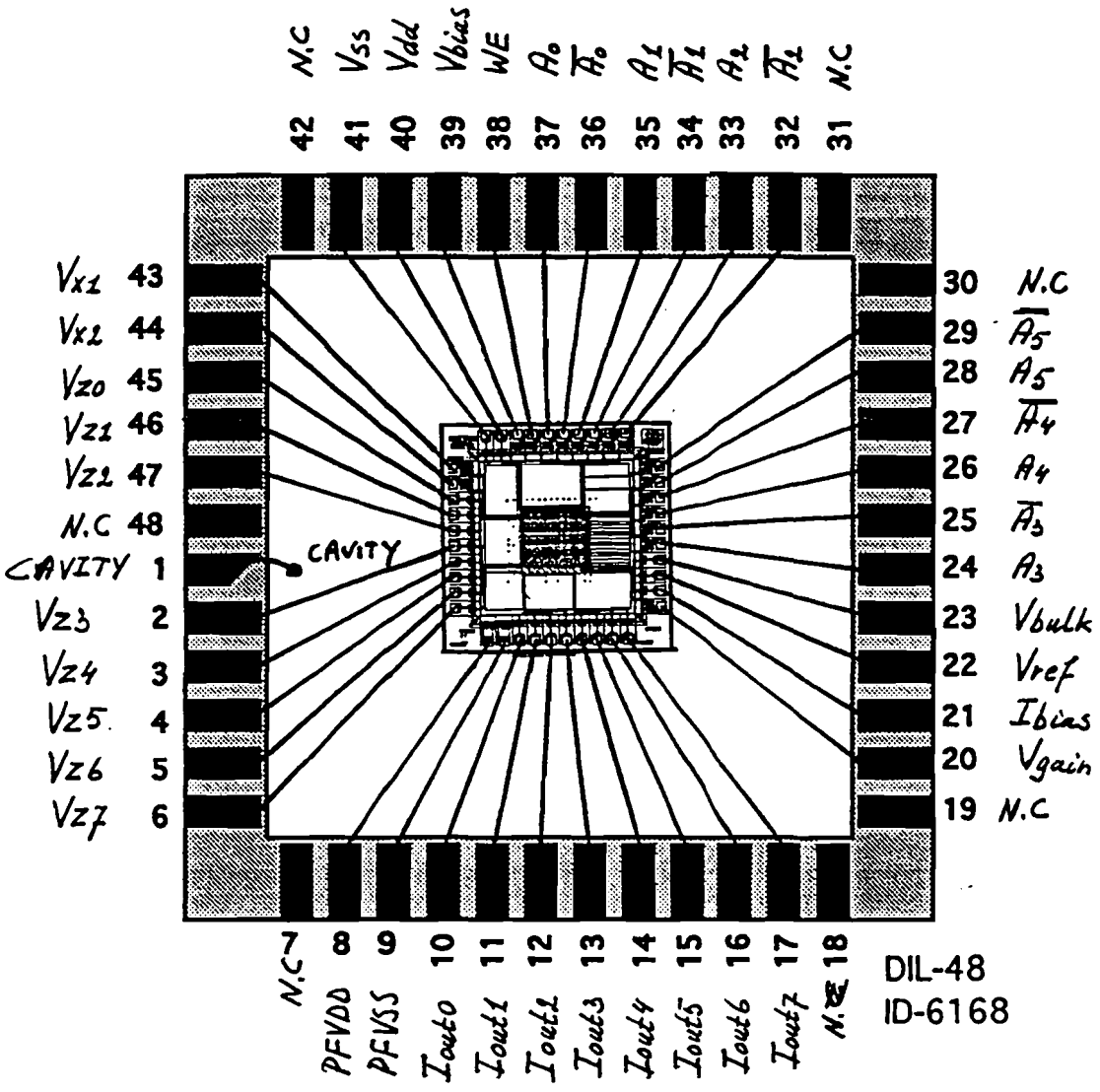
DIE size : 3168µ x 2955µ
MAX. WIRE LENGTH: 3mm

SEE ALSO LAY-OUT RULES FOR PACKAGING
OF DEVICES IN DOCUMENT NUMBER 40001

IMEC

EUR 332

# BONDING DIAGRAM




DIL-48
ID-6168

Synapse_Chip

REMARKS: Not connected pins: 7-18-19-30-31-42-48
Pin 1 connected with CAVITY

SCALE:10

3805-03

DIE size : 3163µ x 3202
MAX. WIRE LENGTH: 4mm

SEE ALSO LAY-OUT RULES FOR PACKAGING
OF DEVICES IN DOCUMENT NUMBER 40001

IMEC

## Appendix D

# Paper

# Analog VLSI Implementation of a Feed-Forward Neural Network

J.M.C. Oosse, H.C.A.M. Withagen, J.A. Hegt
Department of Electrical Engineering, Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
Tel: +31 40 472366, Fax: +31 40 455674, E-mail: heiniw@eeb.ele.tue.nl

**Abstract**

Implementing neural networks using analog electronics has the advantage that common neural operations (multiplication, addition) can be realized with small circuits. Here, the implementation of feed-forward multilayer perceptrons using simple analog circuits will be discussed. Both simulation results and measurements on fabricated chips will be presented.

## 1 Introduction

Because of the increasing popularity of neural networks to provide solutions for many real world problems, there is a growing demand for large, fast processing neural networks. The advantage of the implementation of neural networks in software is the large amount of available flexibility. Virtually any kind of neural network in any size can be implemented with the same underlying hardware. However, since most software neural networks run on sequentially operating architectures, it is impossible to really simultaneously compute multiple 'connections' or 'multiply-and-add's, which are so typical of neural networks. Therefore software implementations tend to be prohibitively slow. Only by the implementation in hardware of many devices that can each perform multiply-and-add operations, the connections can be calculated truly in parallel making these implementations orders of magnitude faster than their software equivalents.

One of the main reasons for using analog electronics to realize neural network hardware is that several of the operations in neural networks can be realized by simple analog circuits, eg. sigmoids, adders, simple multipliers. It is believed that the fault-tolerant nature of neural networks will compensate for the lack of accuracy in analog realizations.

Here we will concentrate on the implementation of fully connected multilayer perceptrons. Training of the resulting network will be done off-chip by a host-computer using the Weight Perturbation algorithm [1].

In section 2, some general points will be put forward which should be considered when trying to implement neural networks using analog electronics. In section 3 and 4, the synapse and neuron chips will be described, respectively.

## 2 General Considerations

### 2.1 Flexibility and Cascadability

Special attention has to be paid to this aspect as it has far-reaching consequences for the layout of the neural network chips. An implementation of a multilayer perceptron could be done in the following possible configurations:

- A multilayer perceptron with a fixed topology
- A multilayer perceptron with an arbitrary number of layers but a maximum number of neurons per layer.
- A multilayer perceptron with a completely arbitrary topology (arbitrary number of inputs, neurons, and layers).

Here we have chosen for the third configuration; a completely flexible structure with on-chip weight storage. To obtain this flexibility, a layer is split into two parts; a synapse and a neuron part.

An arbitrary topology can be created by cascading synapse and neuron chips. For example synapse chips with 2x2 connections and neuron chips with 2 units can be combined as shown in figure 1.

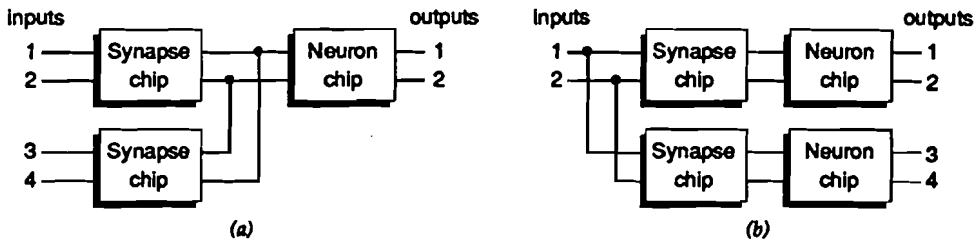



Figure 1: Expanding *(a)* number of inputs and *(b)* number of neurons

Cascadability of the chips has some important electronic implications. When for example one connects 16 (2x2)-synapse chips in parallel, then a single neuron has 32 inputs. Suppose one synapse produces an output current between -2$\mu$A and +2$\mu$A, then 32 synapses can (maximally) generate a current between -64$\mu$A and +64$\mu$A. The input range of the sigmoid will now be too small and the output of the neuron will contain large flat areas as illustrated in figure 2. Because cascadability is desired, scaling of the neuron input is necessary. The scaling factor

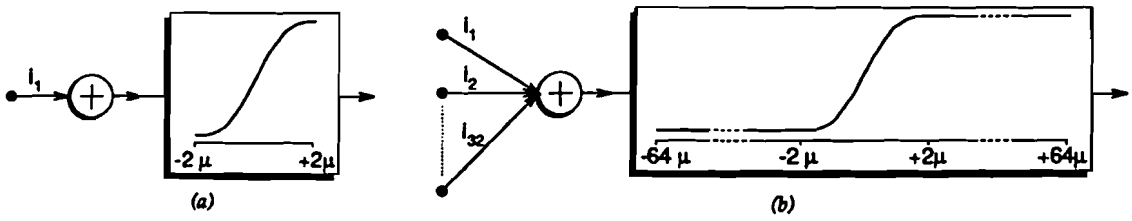



Figure 2: *(a)* one synapse connected and *(b)* 32 synapses connected to one neuron

must depend on the number of incoming synapses. Statistical analysis [2] has shown that it's best to scale the input according to $\sqrt{N}$, with $N$ the number of inputs.

## 2.2 Internal chip organization

In figure 3 the forward path of both the synapse and neuron unit are presented. The subscripts $i$ and $j$ refer to the $i^{th}$ input and the $j^{th}$ neuron. The characters V and I refer to whether signals

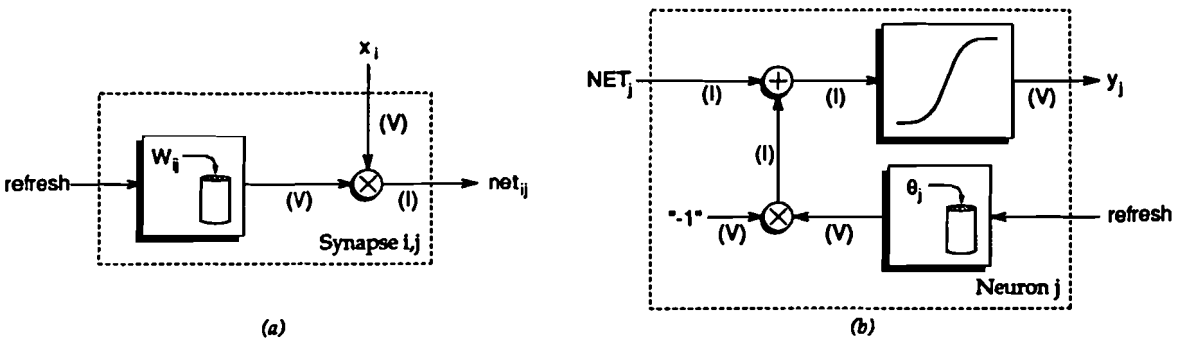



Figure 3: The forward path of *(a)* the synapse and *(b)* the neuron unit

are voltages (signals to be distributed) or currents (signals to be summed). The synapse unit consists of multiplier together with a unit to store the weight value. The neuron unit comprises a bias weight and a sigmoid block.

The high-level organization of both chips is shown in figure 4. The synapse units are arranged

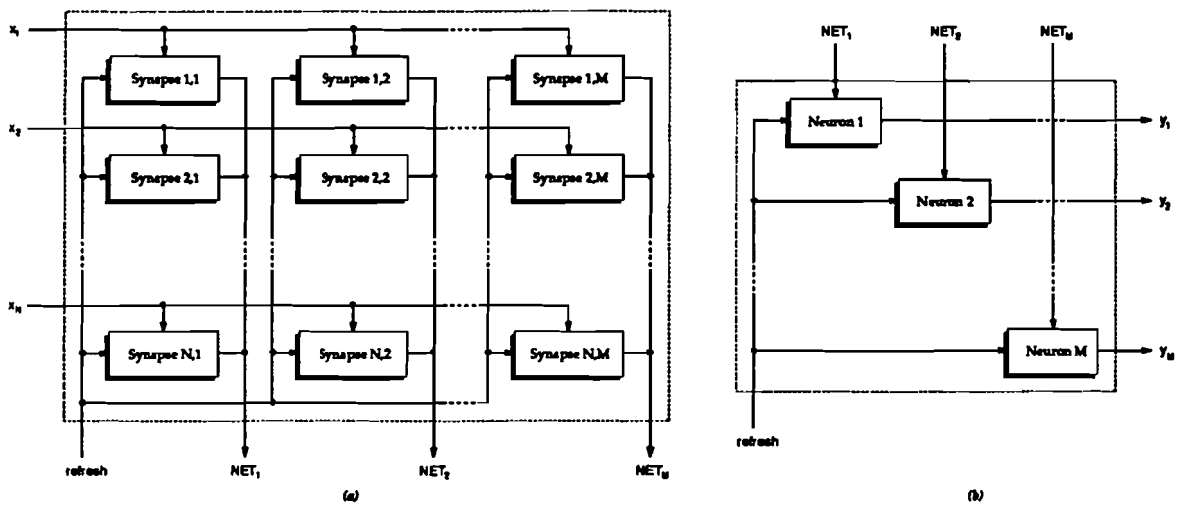



Figure 4: Internal organization of *(a)* the synapse and *(b)* the neuron chip

in an array and the neuron units in a column. Building larger networks with these building blocks is now straightforward.

# 3 Synapse Chip

Here we have chosen to keep the multiplier implementing the synapse as small as possible. The two-transistor multiplier [3] as shown in figure 5 will be used. It occupies minimal area while delivering superb linear behavior. Assuming transistors M1 and M2 are identical and $V_{y1} = V_{y2} = V_y$, the difference in current can be expressed as:

$$I_1 - I_2 = K'_p \left(\frac{W}{L}\right) (V_{w1} - V_{w2})(V_z - V_y) \tag{1}$$

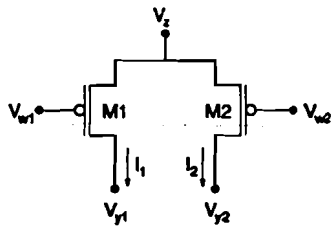



Figure 5: Two-transistor multiplier

The multiplier can then be used in the neural network implementation by applying the synapse input value $x_i$ as $V_z - V_y$ and the weight value $w_{ij}$ as $V_{w1} - V_{w2}$. To secure a linear multiplication, the following ranges are allowed for the different voltages; $V_{w1}, V_{w2}$: 0-1V, $V_z$: 2-3V, and $V_y$: 2.5V The difference between the output currents $I_1$ and $I_2$ represents the output of this simple multiplier under the condition that the voltages $V_{y1}$ and $V_{y2}$ remain equal and constant. This could be achieved by adding a current conveyor behind the multiplier. However, here we propose a straightforward subtraction implementation. Giving up the requirement that the $V_y$-values should be equal and constant, it is possible to subtract the currents by using small resistors as loads which convert the currents to small proportional voltages. This deviation from the ideal situation will cause the multiplier to become non-linear. However, if we keep the voltage swing small, the influence will be marginal. Furthermore, the loads can be shared by all the synapses connected to the same neuron which has two advantages:

- Area reduction because the circuit has to implemented only once per column of synapses, see figure 4.
- If we make the loads tunable, we will be able to control the neuron input and solve the cascadability problem mentioned in the previous section.

The small resistors are realized by using NMOS-transistors in their linear range. Although the $V_{ds} - I_d$ characteristic is parabolic, the difference between two of these curves is quite linear. Linearity can be improved by adding PMOS-transistors in parallel with the NMOS-load. The resulting circuit be seen in figure 6 in the left-bottom part. Note that more than one two-transistor multipliers can be connected to the same load. By adjusting the value of the *load-resistors* (through $V_{gain}$) the resulting output range can be set. The voltages $V_{y1}$ and $V_{y2}$ are now subtracted using a standard differential stage with a current mirror load (OTA), which can be also seen in figure 6 on the right side. The outputs of different subtractor circuits can be connected together and the resulting current will be used as the input of a neuron.

In figure 7, both simulated and measured static characteristic of one synapse (including load and subtractor) are shown. For different weight values, the output current versus input voltage relation is computed, measured respectively.

Dynamic weight storage has also been implemented on-chip by adding capacitors and switches which are used as sample-and-hold circuits. In figure 8, the additions to the original two-transistor multiplier are shown. The differential structure is chosen because it will reduce the effect (in the output of the multiplier) of charge injection when accessing the capacitors and leakage of the capacitors during the retention period. In a trade-off between accuracy and area, capacitor values of 0.1pf were chosen. Weight values will now be stored as follows. Input $V_{x2}$ will be kept fixed at 0.5V while the actual weight will be presented through $V_{x1}$. Note, that each time $V_{w1}$ is refreshed, $V_{w2}$ will also be refreshed. This way, first-order effects of charge injection

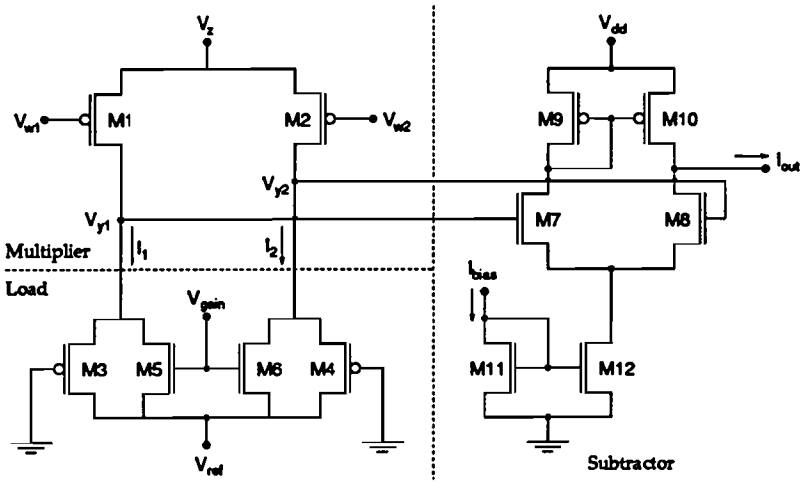



Figure 6: Synapse implementation

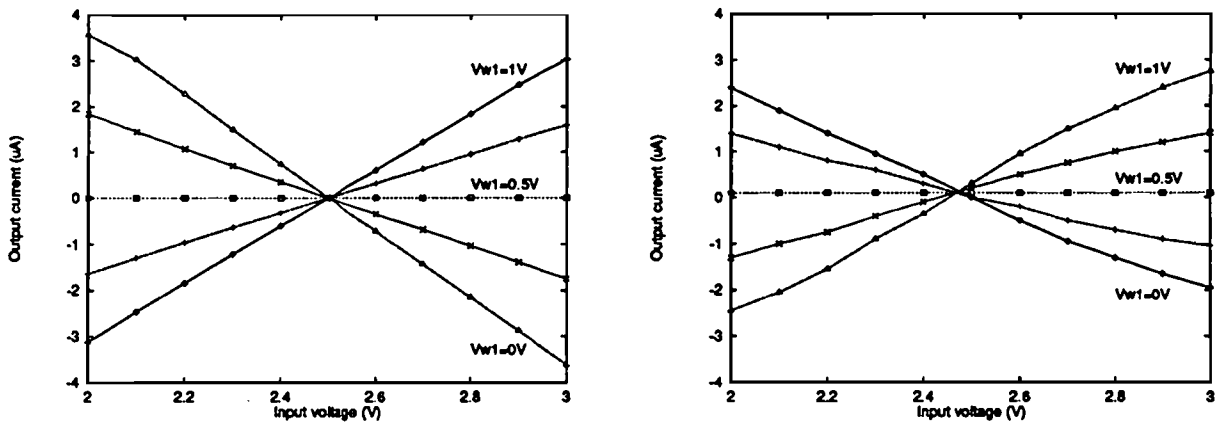



Figure 7: Simulated *(left)* and measured *(right)* synapse characteristic

and leakage will be greatly reduced.

As the number of synapses on one chips increases, it becomes impossible to access all weights simultaneously. Therefore, addressing circuitry has been added to access the weights sequentially. The number of synapses per chip, the leakage of the capacitors, and the refresh time per synapse will determine the final weight accuracy obtainable.

An 8x8 synapse chip has been implemented using the MIETEC 2.4$\mu$m double poly/double metal process. The area needed to implement one synapse i.e. the two-transistor multiplier plus the capacitors and access transistors amounts to 5,400$\mu m^2$. A more fair measure would be the total area (including the loads, subtractor and addressing circuitry) needed to implement all 64 synapses and divide this area by 64. This amounts to a synapse implementation area of 12,100$\mu m^2$. Note however that this last figure will decrease as larger synapse arrays will be implemented.

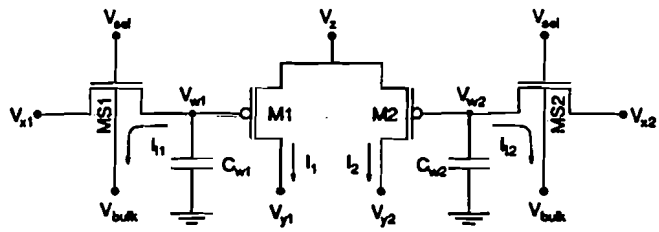



Figure 8: Weight storage

# 4 Neuron Chip

The main aspect of the neuron unit is its non-linear behavior. A sigmoid-like function can be realized in electronics by a simple differential stage as can been seen in the middle part of figure 9. The differential stage is formed by transistors M1 and M2. Through $V_{gain}$ (M3 and M4) and

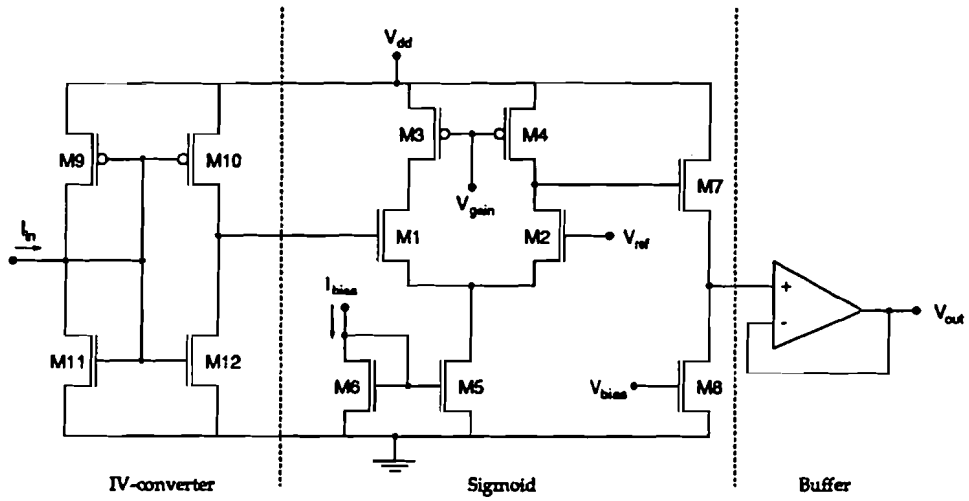



Figure 9: Neuron implementation

$V_{bias}$ (M8) the output range of the sigmoid can be controlled. As the output of the sigmoid is used as the input of synapses of the following layer, the two ranges should comply (2-3V). Because the neuron has to be able to source or sink the necessary current for all the synapses in the next layer a buffer has been added at the output.

As the synapses generate currents and the sigmoid requires a voltage input, a current-to-voltage (IV) converter is needed. It consists of two stages as be seen on the left side in figure 9. The first stage [4] converts the input current into a small output voltage proportional to the input current. The second stage is a CMOS inverter which 'boosts' this small voltage to the required input range of the sigmoid.

The bias weight of the sigmoid (see figure 3) is implemented through a circuit very similar to the synapse circuit shown in figure 6. In this case, the input voltage $V_z$ of the synapse is connected to a fixed voltage (3V). This bias weight is then able to shift the sigmoid over the whole input range.

In figure 10, the simulated and measured static characteristic of one neuron (i.e. the circuit shown in figure 9) are shown. The measurements have been performed on a chip containing eight

complete neurons. The implementation area of one neuron (including bias weight) amounts to approx. 41,000$\mu m^2$. In the right plot, the neuron transfer for several neurons is shown which

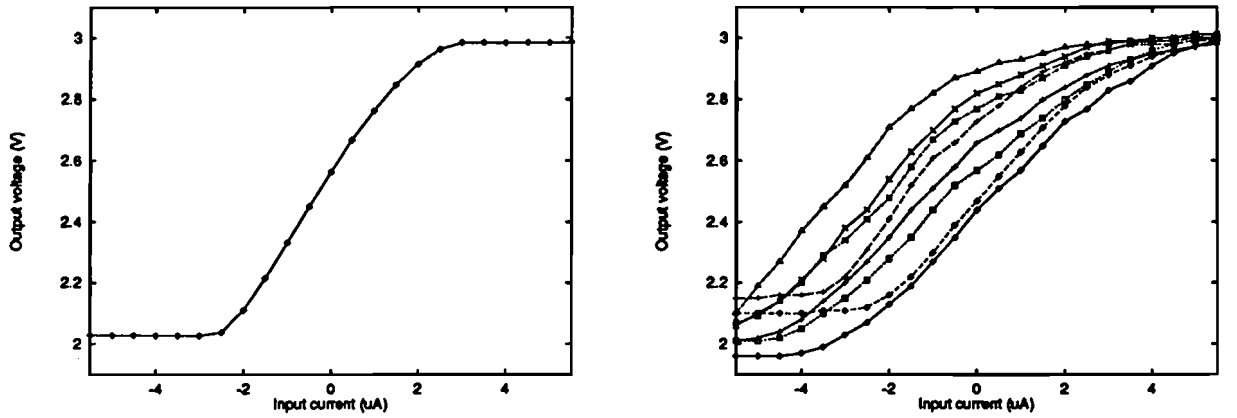



Figure 10: Simulated *(left)* and measured *(right)* neuron characteristic

were implemented on one chip. The differences between the neurons are caused by mismatches between transistors. However, the resulting error can be seen as a shift of the sigmoid and during learning the bias weight will be able to take this into account. Furthermore, with regard to the simulated transfer, the steepness of the measured transfer is lower. This is most probably caused by a problem in the current mirror of the sigmoid part.

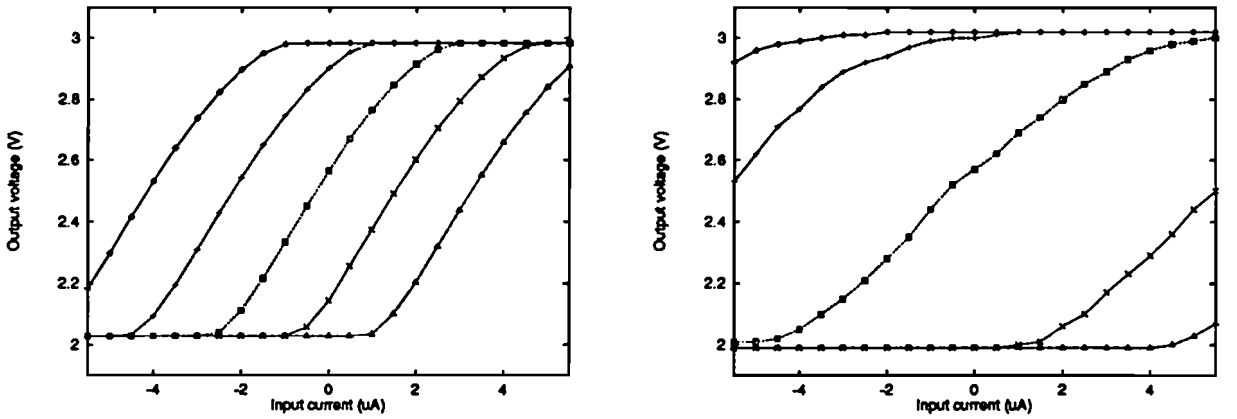



Figure 11: Simulated *(left)* and measured *(right)* neuron shift characteristic

For one neuron, we have tested the ability to shift the sigmoid over the whole input range. In figure 11, both ideal and measured neuron characteristic are shown for different values of the bias weight. Because of the lower steepness, with regard to the simulations, the bias synapse has to deliver more current than initially calculated to be able to shift the sigmoid over the whole input range. This was accomplished by adjusting the (fixed) input voltage $V_z$ of the bias synapse.

# 5 Conclusions

We have presented an analog VLSI implementation of a multilayer perceptron with very simple subcircuits. The actual synapse only comprises two transistors. Through the use of tunable loads for these synapses it is possible to connect several synapses to the same neuron. Furthermore, it is very easily possible to build larger networks by connecting together an arbitrary

number of synapse and neuron chips.

In the near future, we hope to be able to build a complete system including a multilayer perceptron connected to a host computer which will execute the Weight Perturbation algorithm. If this proves to be successful larger synapse and neuron chips will be implemented.

## References


[1] M. Jabri and B. Flower, "Weight Perturbation: An Optimal Architecture and Learning Technique for Analog VLSI Feedforward and Recurrent Multilayer Networks", *IEEE Transactions on Neural Networks*, vol. 3, no. 1, pp. 154-167, January 1992.

[2] H.C.A.M. Withagen, "Reducing the Effect of Quantization by Weight Scaling", *accepted at ICNN 1994, Orlando, Florida.*

[3] P.B. Denyer and J. Mavor, "MOST Transconductance Multipliers for Array Applications", *IEE Proceedings*, vol. 128, Pt. I, no, 3, June 1981.

[4] B. Nauta, "Analog CMOS Filters for Very-High Frequencies", *Ph.D. Thesis*, pp. 95-96, September 1991.