Electronic implementation of multi layer neural networks including back propagation training algorithm

Kuijpers, M.J.

Award date:
1992

Electronic implementation of Multi Layer Neural Networks including Back Propagation training algorithm

Masters thesis of M.J. Kuijpers
id.nr.: 255408

Supervisor	: Prof.dr.ir. W.M.G. van Bokhoven
Coach	: Dr.ir. J.A. Hegt
Period	: Sept. 1 1991 - Apr. 23 1992
At	: Electronic circuit design group (EEB) Faculty of electrical engineering Eindhoven University of Technology

Abstract

Artificial Multi Layer Neural Networks can be used for problems for which an algorithm is not known or extremely complex. Examples are speech- and image-recognition. A possible algorithm used to train the network is the Back Propagation algorithm. This algorithm is mostly used with the Multi Layer Perceptron structure.
Most neural networks are implemented in software on a computer. Drawback of this method is that the fully parallel structure of the network is computed sequentially, which makes these implementations very slow. Designing an electronic implementation of neural networks speeds them up considerably. A digital implementation would occupy too much area on a chip so an analog approach is followed. It's also desired that the Back Propagation training algorithm is integrated on chip.

In earlier work, a structure was designed where two building blocks were proposed for implementation: a synapse-chip and a neuron-chip. Also an overview of possible useful circuits for implementing these building blocks was presented. This work is a logical extension to this already conducted research. The most important part of a neural network is the weight-storage and weight-update. For these parts a layout has been made on the basis of a refresh-circuit in earlier work and an update-circuit from the literature. After fabrication the chips have been tested as well. The design of the synapse- and neuron-chips is slightly altered after the representations of the signals have been defined. Of all the elements that are needed for the new designs, circuits are constructed. Also the communication of the chips and the control has been described.

It is believed that the results of the realised weight-storage and weight-update do satisfy the needs. The inaccuracies in storage and update are assumed to be canceled out by the adaptive operation of the network. To be sure about this, simulations on system-level have to be carried out. Of all elements in the network, one or more circuits are available. These circuits have to be optimised yet and layouts must be made to test their behaviour in practice. Also simulations on system-level must be done to analyse the performance of the network if the non-idealities in the electronic circuits are considered.

Contents

1 Introduction

Today computers are used extensively in solving a wide variety of problems. Before a computer can be put to work, an algorithm must be known for solving the particular problem. However, there are problems for which an algorithm is not known or extremely complex. Examples of such problems are speech and image recognition, robot control, routing, etc. For such problems artificial neural networks could be used. These networks consist of simple computational elements, which are densely interconnected. In this respect, artificial neural net structures are based on our present understanding of biological nervous systems. Also the operation of the class of neural networks that is payed attention to here, has similarities with the functioning of the human brain; if the output of the networks doesn't resemble the desired output, the network is adapted and tested again.

A structure of the neural network that works according to this "trial and error" method is the Artificial Multi Layer Neural Network (AMLNN) with the Back Propagation training algorithm for adapting the network.

There are different ways of implementing artificial neural networks. Mainly this is done by software on computers. A disadvantage of this method is that the parallel structure of the network is computed sequentially by a computer making these networks very slow. Electronic implementation can be useful in this respect because fully parallel operation speeds up these networks considerably. Electronic implementation can be divided in analog and digital implementation. It is widely accepted that analog implementation is the most promising option because of the physical size and speed of these networks compared to their digital counterparts. Recently a lot of papers in the literature dealt with analog electronic implementations of artificial neural networks. Very few of them however integrate the training algorithm in the electronic implementation. The adaptation of the network is still controlled by microprocessors. In the implementation that is designed at Eindhoven University of Technology, it's the intention that the training algorithm (Back Propagation) is integrated on the chip (on-chip learning).

In earlier work relevant literature was examined for possible implementations of analog electronic AMLNN including the Back Propagation algorithm. Also an overview of possible suitable subcircuits was presented.
This report is a logical proceeding of already conducted research. First a few theoretical aspects of neural networks including the back propagation algorithm are discussed. The rest of the report explains the AMLNN that is designed following a top-down approach. It starts with a system overview and goes down to circuits on transistor-level or layouts of designed circuits.

2 Theory AMLNN and BP

2.1 Working Principle

An AMLNN consists of a number of computational elements (neurons) connected by links with variable weights (synapses). Such an element is a simplified model of a biological nerve-cell. It sums N weighted inputs and passes the result through a non-linearity. In this case the non-linearity is a sigmoidal function. The multipliers together with the weight-factors form the synapses of the nerve-cell. Fig. 2.1 shows a model of such a computational element and the non-linearity.

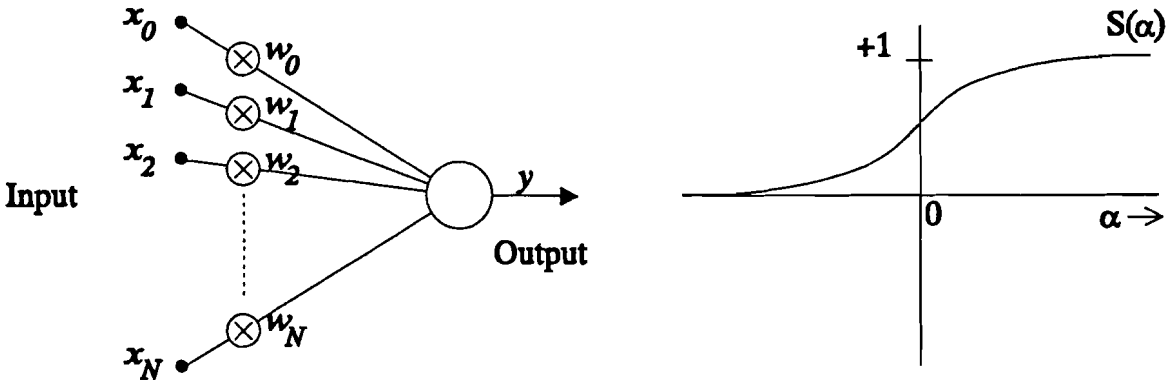


Fig. 2.1 Neuron model

With these nerve-cells different neural net structures with different properties can be built. In this implementation the Multi-Layer Perceptron structure is chosen because this structure has been shown to be successful for many problems of interest (refer to [16]).

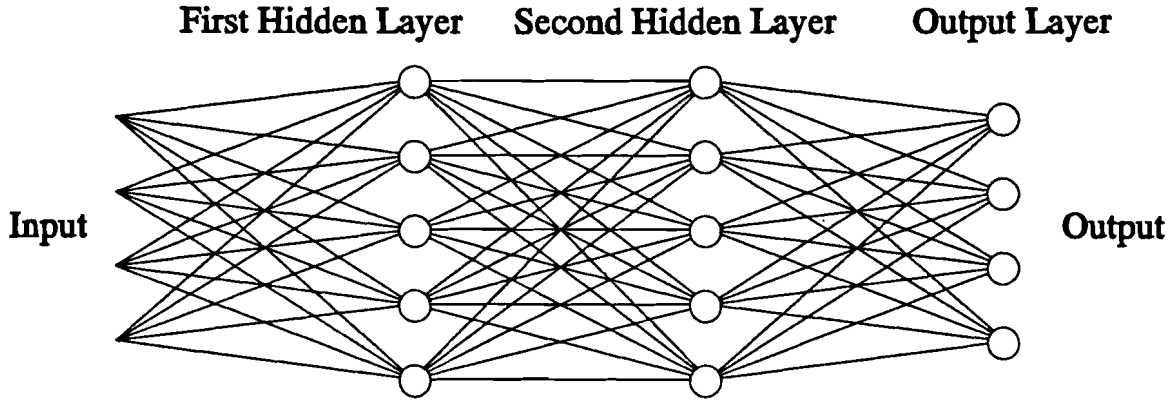


Fig. 2.2 Three-layer perceptron

Multi-layer perceptrons are feed-forward nets with one or more layers of nodes between the input and output nodes. These additional layers of nodes are called hidden layers. A three-layer perceptron with two layers of hidden units is shown in fig. 2.2. The number of nodes in each layer can be chosen freely.

2.2 Back Propagation training algorithm

A neural network has to be trained to function properly. This training of a neural network means adjusting the weights between nodes in order to decrease the error of the network. This is done by a training algorithm. One of these training algorithms is the Back-Propagation Training Algorithm. The back-propagation training algorithm is an iterative gradient algorithm designed to minimize the mean square error between the actual output of a multi layer neural network and the desired output. The difference between the actual output and the desired output determines the error. This error is propagated backwards from the output-layer to the first layer adapting the weights according to their contribution to the total error. Below a detailed step by step description of the back-propagation algorithm is given.

Step 1. Initialize weights
Set all weights to small random values.

Step 2. Present input and desired outputs
Present a continuous valued input vector $x_0, x_1, \ldots x_{N-1}$ and define the desired output vector $d_0, d_1, \ldots, d_{M-1}$.

Step 3. Calculate actual outputs
Calculate the output of the network starting with the first layer and propagating the result of each layer forward to the next layer. The output of a node is given by the following formula:

$$y_j = S(\sum_i w_{ij} x_i) \qquad 0 \le i \le N\text{-}1,\ 0 \le j \le M\text{-}1 \tag{2.1}$$

N : the number of inputs of the layer
M : the number of neurons in the layer
x_i : input i of the layer
y_j : output j of the layer
w_{ij}: weight from input i to neuron j

The term $\sum_i w_{ij} x_i$ is referred to as the activity of neuron j and denoted as a_j.

For layer 1 the input is the specified input vector $x_0, x_1, \ldots x_{N-1}$. For all other layers the output of the current layer serves as input of the next layer.

Step 4. Adapt weights
Starting at the output nodes the errors are calculated. The error of the last layers depends on the difference between the desired output and the actual output. The errors of the other layers are determined according to their contribution to the

total error:

$$\delta_j = S'(a_j)(d_j - y_j) \qquad \text{for the last layer.} \tag{2.2}$$

$$\delta_j^l = S'(a_j) \sum_k (\delta_k^{l+1} w_{jk}^{l+1}) \qquad \text{for the other layers.} \tag{2.3}$$

δ_j : error term for neuron j in the output-layer

δ_j^l : error term for neuron j in layer l

$S'(x)$: sigmoid derivative (fig. 2.3)

d_j : desired output for neuron j

y_j : actual output of neuron j

δ_k^{l+1} : error term of neuron k in next layer

w_{jk}^{l+1} : weight between neuron j in current layer and neuron k in next layer

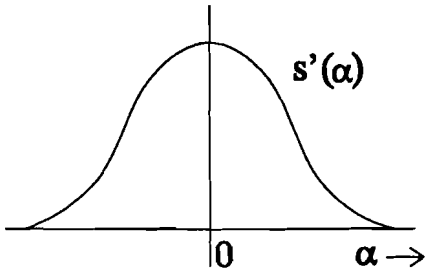


Fig. 2.3 Sigmoid derivative function

If all the error terms are calculated the weights are updated according to:

$$w_{ij}(t+1) = w_{ij}(t) + \eta \delta_j x_i \tag{2.4}$$

In this equation x_i is either the output of node i or an input. η is a gain term which determines the learning rate.
The order in which the weights are updated doesn't matter.

Step 5. Repeat by going to step 2

The steps 2 to 4 are repeated until the error $(d_j - y_j)$ is smaller than a maximum error ε for every input vector.

For better performance of the neural network it must be possible to adapt the thresholds in the nonlinear functions as well. The back-propagation algorithm has no possibility to adapt these thresholds. However there is a way to overcome this problem: add an extra weight to each neuron. The input to this weight is fixed to a non-zero value. Adapting this weight is mathematically equal to adapting the threshold of the non-linearity.

In this chapter a very brief description of the theoretical aspects of the Multi Layer Neural Network and the back-propagation algorithm is given. For more information about the theory of neural networks and training algorithms is referred to the literature a.o. [16].

3 Electronic Implementation

3.1 Organization of the Neural Network

As already mentioned in the introduction implementing neural networks in hardware speeds them up considerably in respect to software implementations. Because of the immense amount of chip area a digital implementation would take, an analog approach is followed. The main problem that arises with this analog approach (lack of accuracy) is believed to be compensated for by the adaptive behaviour of the network.

To be flexible in building different sizes of networks in [32] and [12] a one-layer neural networks is designed as building block. With this one-layer network larger networks can be built by paralleling and/or cascading this one-layer structure. Fig. 3.1 represents a schematic of a three-layer network built with these one-layer neural network structures.

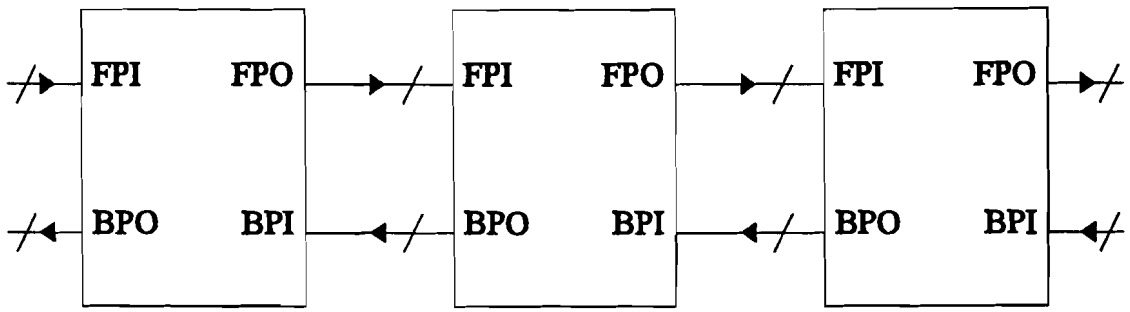


Fig. 3.1 Three-layer neural network

The meaning of the signals is as follows:

- FPI: Forward Propagating Inputs, $FPI_i = x_i$
- FPO: Forward Propagating Outputs, $FPO_j = y_j$
- BPI: Backward Propagating Inputs, $BPI_j = d_j - y_j$ for the last layer, $BPI_j^l = \sum_k \delta_k^{l+1} w_{jk}^{l+1}$ for the other layers
- BPO: Backward Propagating Outputs, $BPO_i^l = \sum_j \delta_j^l w_{ij}^l$

With these signals the BP formulas (2.1) to (2.4) can be rewritten to:

$$A_j = \sum_i W_{ij} \cdot FPI_i \tag{3.1}$$

$$FPO_j = S(A_j) \tag{3.2}$$

$$BPO_i = \sum_j (W_{ij} \cdot S'(A_j) \cdot BPI_j) \tag{3.3}$$

$$\Delta W_{ij} = \eta \cdot S'(A_j) \cdot BPI_j \cdot FPI_i \tag{3.4}$$

By cascading or paralleling a number of one-layer neural networks more layers or more neurons per layer can be created. However, the number of inputs per layer can't be expanded with this one-layer neural network. A solution to this problem is splitting the one-layer structure in a synapse-part and a neuron-part. With these sections as building blocks all sizes of networks can be built in principal.
A few examples of creating larger networks with synapses and neurons as building blocks can be seen in fig 3.2. In this example a synapse-section has 4 inputs and 3 outputs, a neuron-section has 3 inputs and thus 3 outputs. In the left configuration the number of neurons in the layer is enlarged from 3 (available with one synapse- and neuron-section) to 5. In the configuration on the right the number of inputs is expanded to 6.
The forward propagating inputs to the synapses must be represented by voltages so they can be distributed to another synapse-chip as in the left drawing of fig. 3.2. The outputs of a synapse-chip (the activation signal A) must be a current so they can be added as in the right example of fig. 3.2. For the backward propagating signals the same considerations hold: the output of a neuron-chip must be a voltage as it is distributed to several synapse-chips (right drawing). The backward propagating output of a synapse-chip must be a current because the contributions must be summed before applying them to the neuron-chip in the layer before (left example).

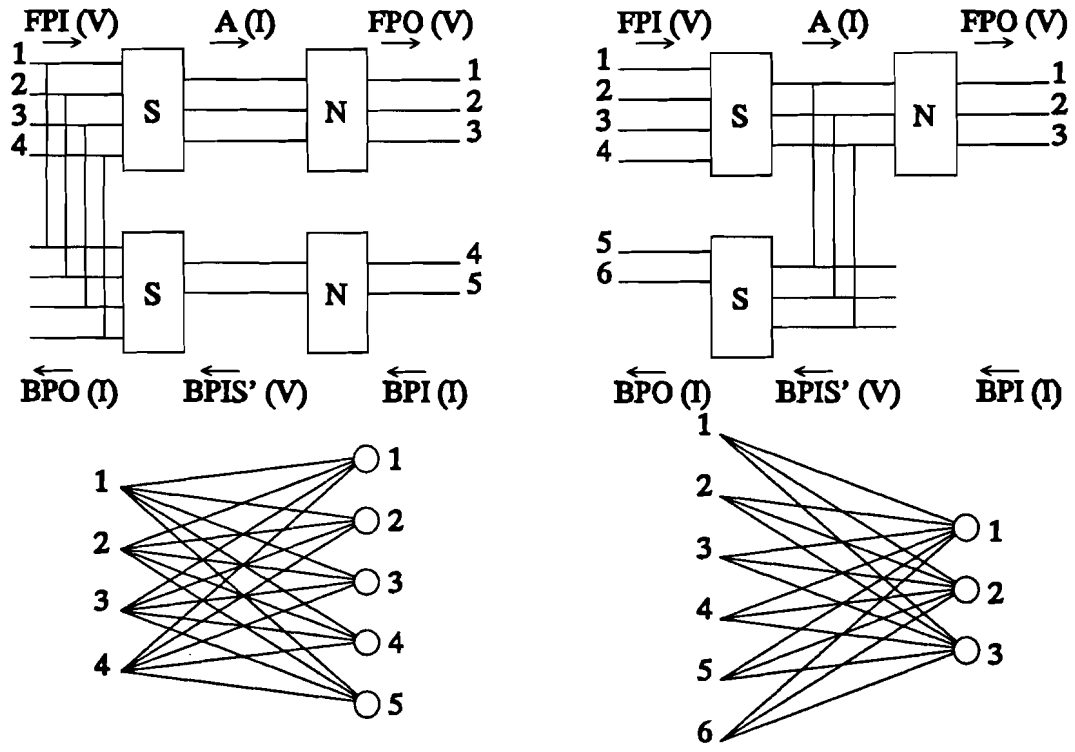


Fig. 3.2 Creating larger networks with synapse- and neuron-sections

The purpose of the neural network project is to implement these synapses and neurons as separate chips that can easily be connected together to form the desired neural network.

3.2 Building blocks

3.2.1 Synapse-chip

A schematic of the synapse-chip is given in fig. 3.3. The inputs of the synapse-chip are the forward propagating inputs FPI and the backward propagating inputs S'(A)·BPI abbreviated to BPIS'. This BPIS' is the last part of equation (3.3). The outputs are the forward propagating activation signal A and the backward propagating error signal BPO as in equations (3.1) and (3.3). The BPIS'-signals are used for the calculation of both the BPO-signals and the calculation of the weight-updates.

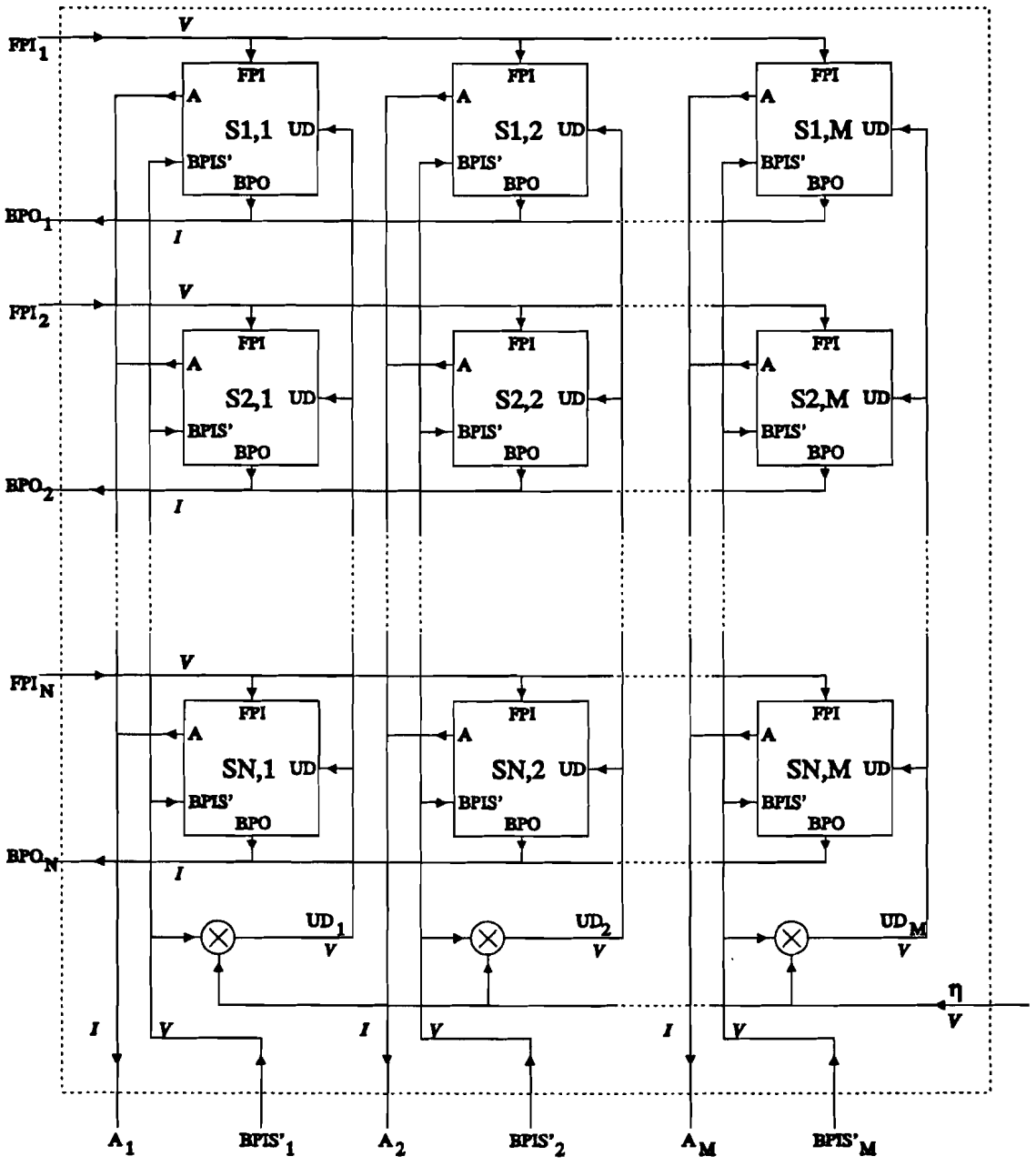


Fig. 3.3 Block-diagram of synapse-chip

For the calculation of the weight-updates BPIS' is multiplied by η resulting in the UD-signal. This signal is distributed to all synapses belonging to one neuron for multiplication by FPI (see formula (3.4)). The synapses in the chip are numbered from $S_{1,1}$ to $S_{N,M}$ with each row of synapses belonging to the same input and each column belonging to the same neuron. In designing the chips a useful representation of the signals must be chosen. According to fig. 3.3 the FPI-signals are distributed to several synapse-cells. Therefore these signals are represented by voltages. The same holds for the BPIS'-signals and UD-signals. The A-signals and BPO-signals are summations of several contributions and thus can best be represented by currents. The summation is than carried out by simply wiring together the outputs of the synapse-cells. The internal organization of a single synapse-cell $S_{x,y}$ is shown in fig. 3.4.
The representation of the signals (current I or voltage V) that results from the considerations mentioned above are displayed as well.

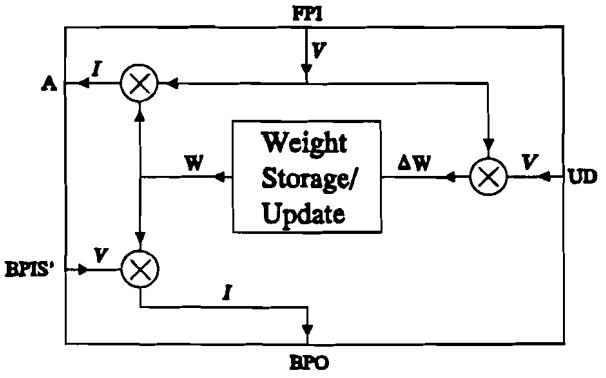


Fig. 3.4 Internal organization of synapse-cell

The weight-storage and update block in fig. 3.4 includes a mechanism to store and update the weight. Details about the implementation of the synapse-chip and especially the storage and update parts are given in chapter 4.

3.2.2 Neuron-chip

A block diagram of the neuron-chip is given in fig. 3.5. This chip has as inputs the activation signals A from a synapse-chip and the error-signals BPI. These BPI-signals can come from a synapse-chip (BPO-signal in that synapse-chip). However, if the neuron-chip forms the neurons for the last layer of the network, the BPI-signal is the difference between the output of this neuron-chip and the desired output $d_0,d_1,...,d_{M-1}$. This BPI-signal is calculated by a simple resistor network that is placed on top of the last neuron-chip. The output of the neuron-chip are the FPO-signals and BPIS'-signals according to equations (3.2) and the last part of (3.4).

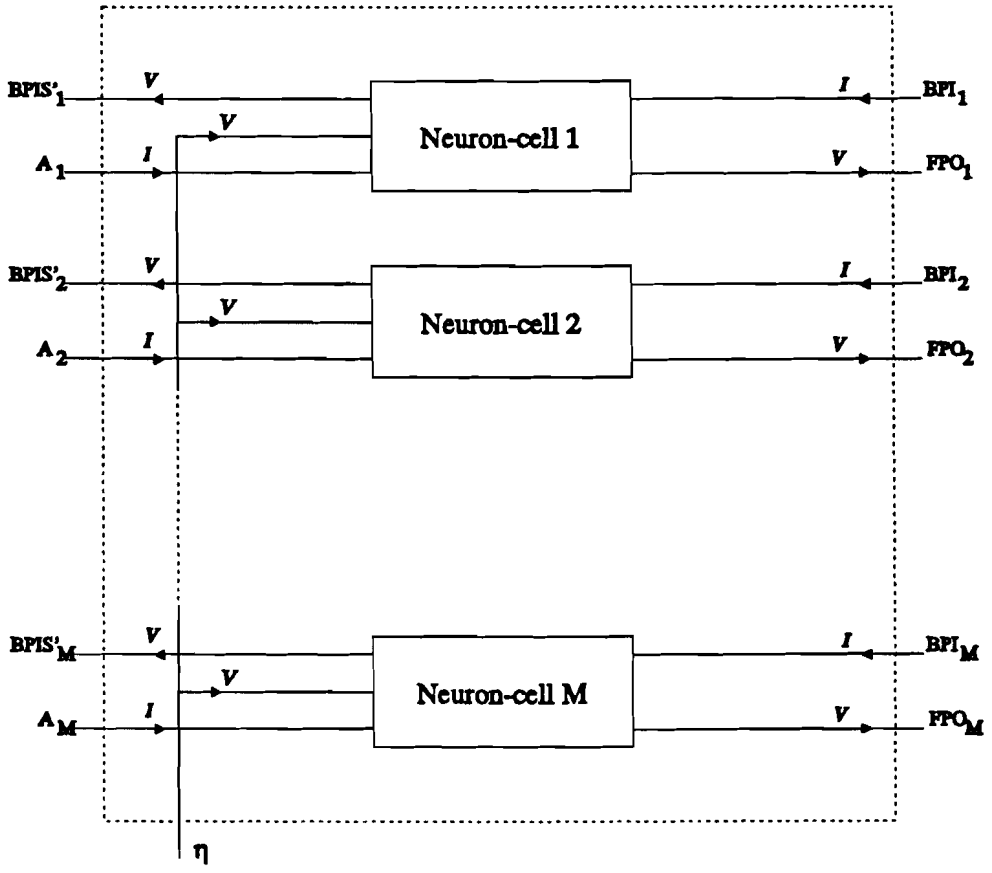


Fig. 3.5 Block diagram of neuron-chip

In fig. 3.6 the internal organization of one neuron-cell is shown. The weight in the neuron-cell represents the extra weight with fixed input (in this case this input is 1) to implement the variable threshold in the sigmoid-function as explained in chapter 2.2. Also in this chip signals that have to be distributed to several blocks (BPIS' and FPO) are represented by voltages, signals that must be summed are represented by currents (the output of the "threshold-weight" must be added to the activation-signal A). Further information about the implementation of the neuron-chip is given in chapter 5.

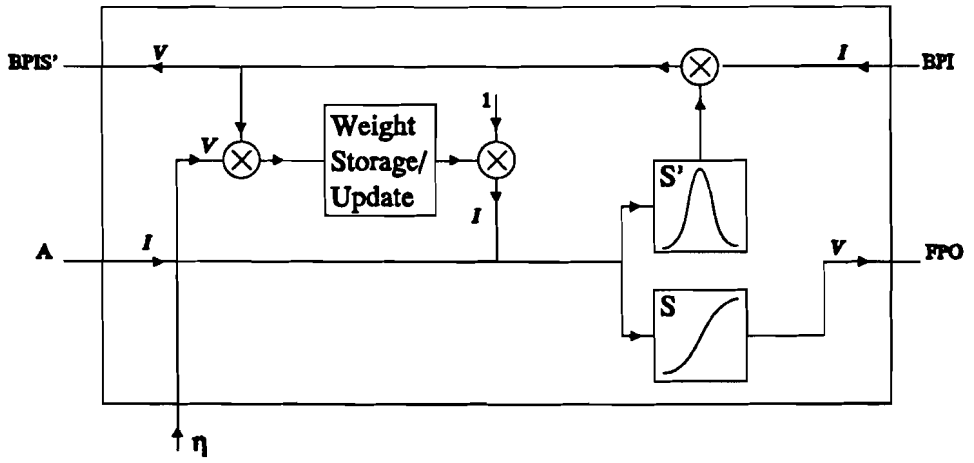


Fig. 3.6 Internal organization neuron-cell

3.3 Communication and control

The synapse- and neuron-chip are designed such that they can be combined without further hardware between them. One exception is the neuron-chip of the last layer where the backward error-signals are calculated from the actual and desired output with extra circuitry as mentioned in the previous paragraph. The number of synapses in a chip is limited because of area-considerations. However, also the number of IC-pins can be a limiting factor. An N by M synapse-matrix (see fig. 3.3) needs already 2(N+M) signals just for the forward- and back-propagating signals.
For now every signal has its own IC-pin but in the future it is possible that the chips must accommodate more synapses. A solution to this pin-problem is multiplexing. A number of forward propagating signals are multiplexed in a chip and transported on a single line to the receiving chip. There the signals are demultiplexed and distributed to their final destination. This multiplexing-demultiplexing occurs in both synapse- and neuron-chip for both forward- and backward-propagating signals (fig. 3.7).

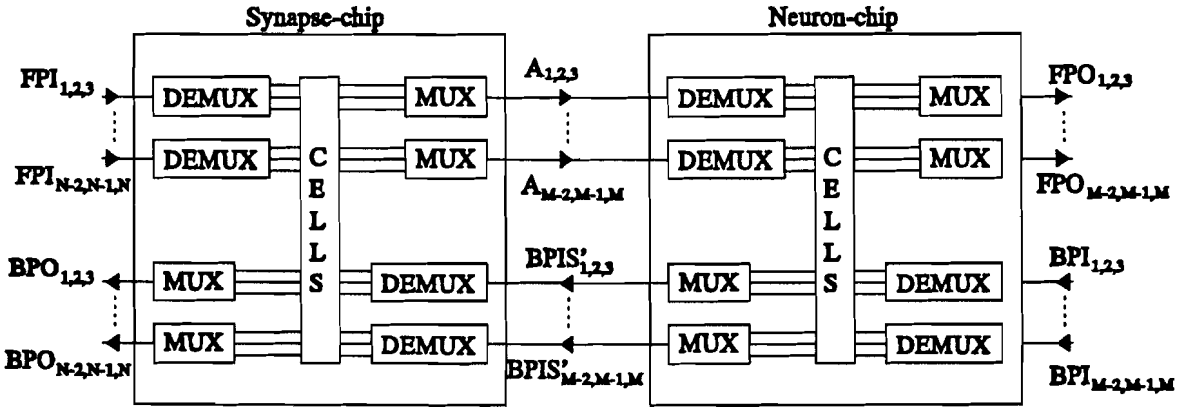


Fig. 3.7 Multiplexing/Demultiplexing for pin-reduction

In the literature two methods of multiplexing are used: frequency division multiplexing (FDM) and time division multiplexing (TDM). These methods are not worked out for this design but detailed information can be found in [4],[7],[8],[9],[10], and [27].

From the description of the back propagation algorithm it is clear that there must be a mechanism to control when the weights are updated and when new inputs and outputs can be presented to the network. This is done by means of delays that are implemented on each chip as shown in fig. 3.8. At the same time an input/output-pair is presented a signal FPI_VALID is generated. This signal acts like a token which is passed from chip to chip until it eventually comes back to the first chip (BPO_VALID). Then all errors are calculated (step 4 in the back propagation description) and the weights can be updated. The delay in a chip is equal to the delay of the "real" forward and backward propagating signals in each chip.

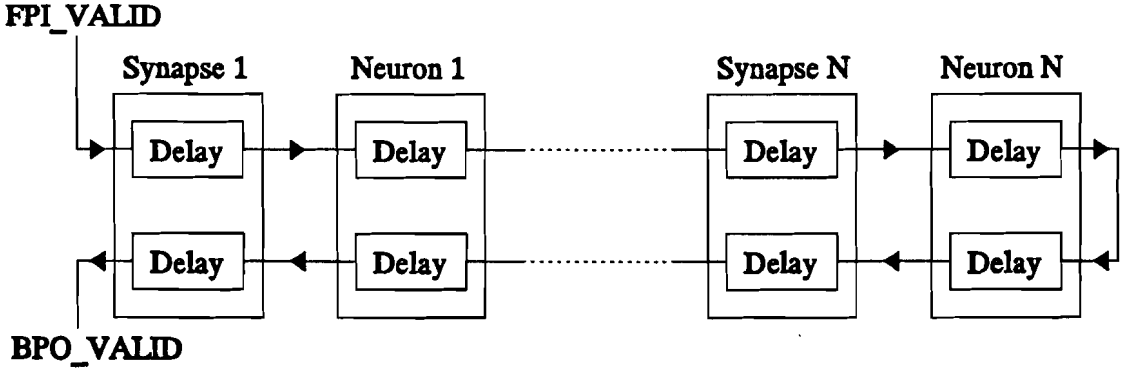


Fig. 3.8 Token passing for controlling the network

It's the intention that the neural-network is controlled by a computer for easy operation. This computer generates the pairs of inputs and desired outputs and the FPI_VALID signal. On detection of the BPO_VALID-signal the computer can take action to update the

weights and present a new pair of input/output signals. Another possibility is that the computer determines the delay and generates the signals for weight-update by itself.
More information about the implementation of the delay-circuit and the computer-controlled operation is given in chapter 6.

4 The Synapse-chip

From the block-diagram of the synapse-chip (fig. 3.3) and the synapse-cell (fig. 3.4) it is clear that the following components must be designed:
- weight storage and update
- multipliers

All circuits have to operate with a power supply of Vdd=5V and Vss=0V.

4.1 Weight Storage

The weight storage block in fig 3.4 consists of an update-part and a refresh-part. The weight is stored as a charge on a capacitor. This capacitor leaks its charge because of leakage currents in transistors attached to the charge capacitor. That's why a refreshing technique is used to keep the weight at its value. The update-part takes care of the adaptation of the weight.

4.1.1 Refresh circuit

The refresh circuit is shown in fig 4.1. It works according to the principle of locking the voltage on a sampled ramp: the voltage on the capacitor is periodically compared with quantized values and regenerated to these levels. In [13],[25],[34] and [32], this method is already described and the circuit which is expected to have the best properties is presented here.

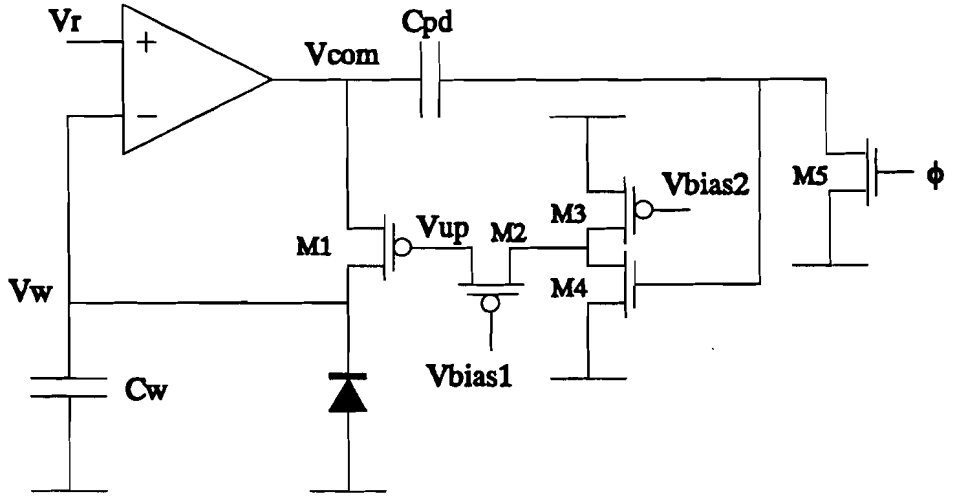


Fig. 4.1 Refresh circuit

In fig. 4.1 Vw is the weight-voltage, stored as a charge on capacitor Cw. Vr is a signal in the form of a staircase, representing the predefined quantized levels and ϕ is a clock.

The functioning of the circuit is as follows. If Vr becomes greater than Vw, Vcom goes high. If ϕ is low at that moment Vup will drop from supply-voltage to a voltage equal to $Vbias_1+V_T$. Here V_T is the threshold voltage of the transistor M2. Now M1 is turned on with a small gate to source voltage and Cw is charged raising voltage Vw. If Vw approaches Vr, Vcom will decrease and cut off M1. Now Vw is refreshed to approximately the present value of Vr. After that ϕ goes high and causes Vup to rise to supply-voltage again. Besides, Vr is incremented to the next quantization value and becomes higher than Vw causing Vcom to go high again. Because the refresh-transistor is off when ϕ goes high, clock feed-through is reduced so a small capacitor can be used for Cw (a few hundreds of fF's). The diode assures that the capacitor will leak to ground lowering voltage Vw.
If ϕ is high at the moment Vcom goes high, Vup stays at supply-voltage so the refresh-transistor stays off and the capacitor Cw isn't charged.
Fig. 4.2 shows the results of a PSPICE-simulation of the refresh-circuit. Used values are:

Cw = 400 fF
Cpd = 25 fF
$Vbias_1$ = 2.4V
$Vbias_2$ = 3.6V

Furthermore all transistors are minimum sized (W/L = 2.4µ/2.4µ).

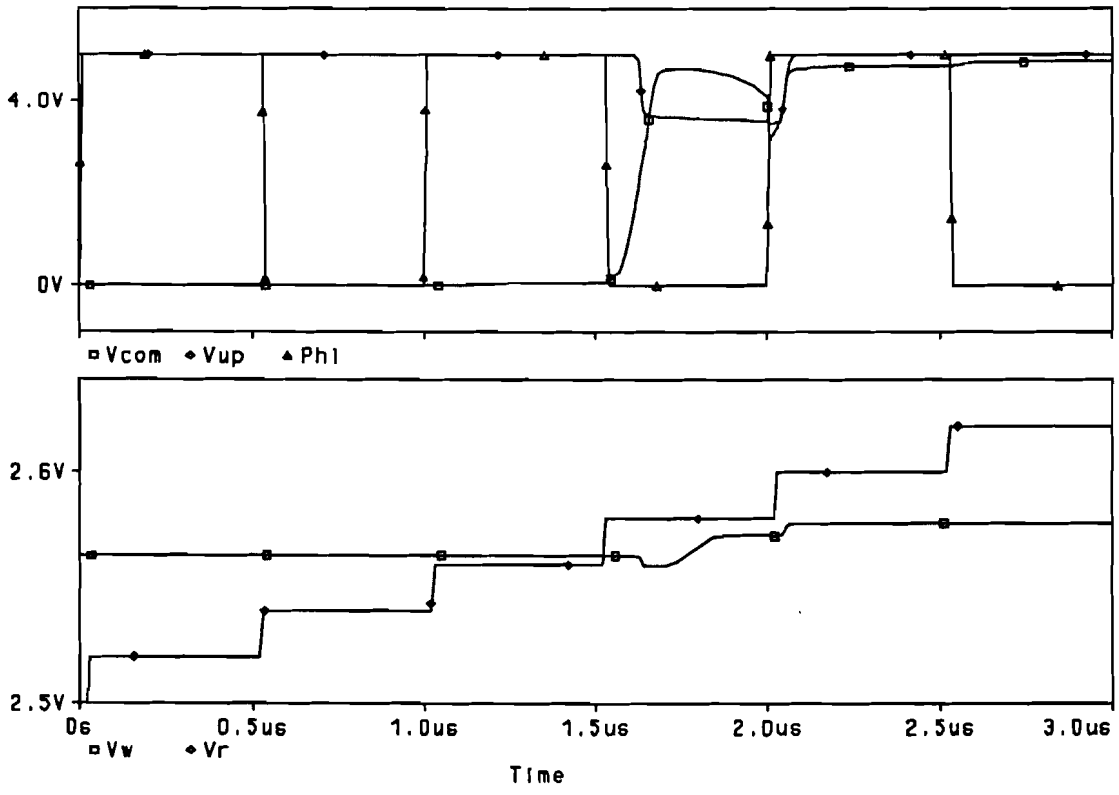


Fig. 4.2 PSPICE-simulation of refresh-circuit

4.1.2 Update circuit

As already mentioned, training the network means adjusting the weights. The amount of adjustment is given by equation (2.4). So a mechanism has to be found where the stepsize is controllable. It is believed that it's not necessary that equation (2.4) is exactly implemented because of the adaptive operation of the network.
The principle of the applied update circuit is moving charge packets to or from the charge capacitor Cw like a Charge Coupled Device ([14],[29],[30] and [31]). The circuit is given in fig 4.3.

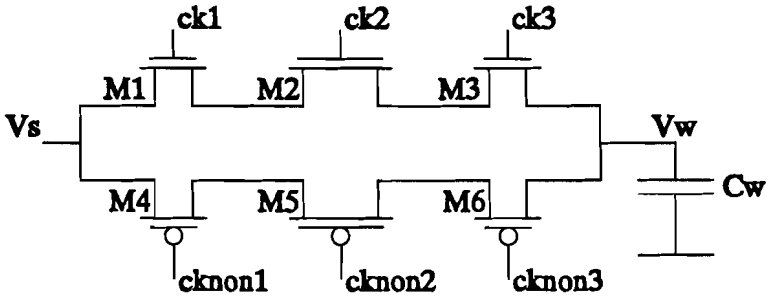


Fig. 4.3 Update circuit

The control-signals ck1, ck2 and ck3 used for the update-circuit are shown in fig. 4.4. The signals cknon1, cknon2 and cknon3 are not shown. These signals are simply inverted versions of the signals ck1, ck2 and ck3.

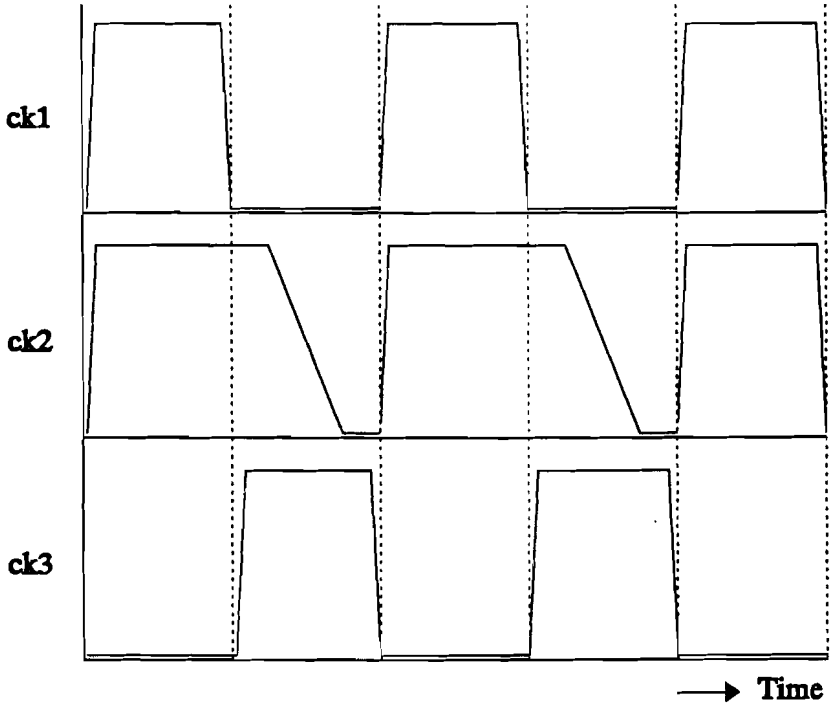


Fig. 4.4 Control-signals for update circuit

The circuit works as follows. First the voltage Vs, which represents the weight change, is sampled by the n-channel and p-channel transistors M1 and M4. This results in a charge packet used to form the channel of the transfer transistors M2 and M5. If M1 and M4 are turned off again a charge packet

$$\Delta Q \approx C^*(V_s - V^*)$$

is transferred onto the transfer transistors. V^* is a reference voltage which determines the "zero-level" of the charge-capacitor. Negative changes are represented by voltages lower than this V^*, positive changes by voltages higher than V^*. V^* is defined by the characteristics of the transfer-transistors but it's not completely independent of the voltage on the charge capacitor. C^* is determined by the geometries of the transfer transistors: a transistor with a large channel length takes up a lot of charge to create the conductive channel when it's turned on. This results in a relative large C^*.

After the transistors M1 and M4 are turned off the transistors M3 and M6 are turned on. By slowly turning off the transfer transistors almost all of the charge in the channels of these transistors is transferred to the charge capacitor Cw resulting in the desired weight change. One clock cycle is defined by the time-range from the moment ck1 goes high until the moment ck3 goes low.

PSPICE-simulations show that V^* and C^* are optimal if the transfer-transistors have a W/L-ratio of 2.4/4. This results in a V^* of approximately 2.2 V and the most symmetric weight changes of Vw when applying voltages Vs in the range of 0 to 5V. The charge capacitor used has a value of 800 fF to limit the clock-feedthrough and to keep the weight changes small for being able to make fine adjustments. Fig. 4.5 shows a PSPICE-simulation of this update-circuit.

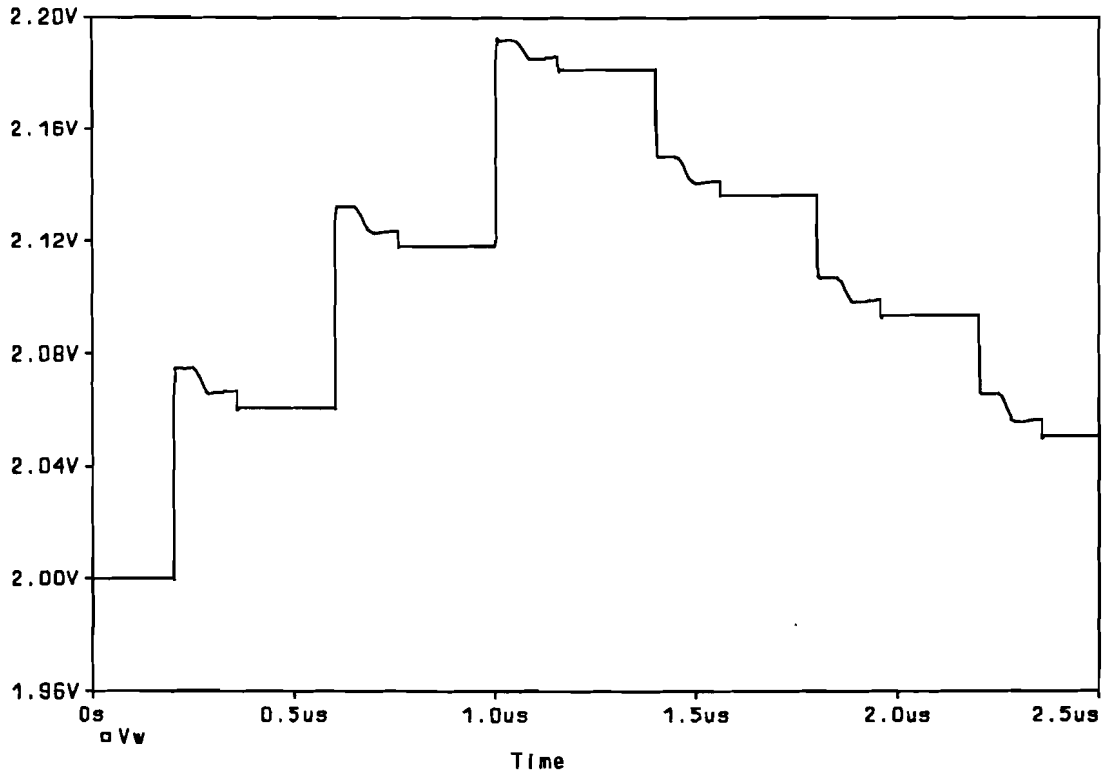


Fig. 4.5 PSPICE-simulation update-circuit

During the first three clock cycles the weight Vw is updated in the upward direction using a voltage Vs of 5V. The next three cycles show a downward adaptation of the weight with Vs = 0V. The control-signals ck1, ck2 and ck3 are displayed in fig. 4.6.

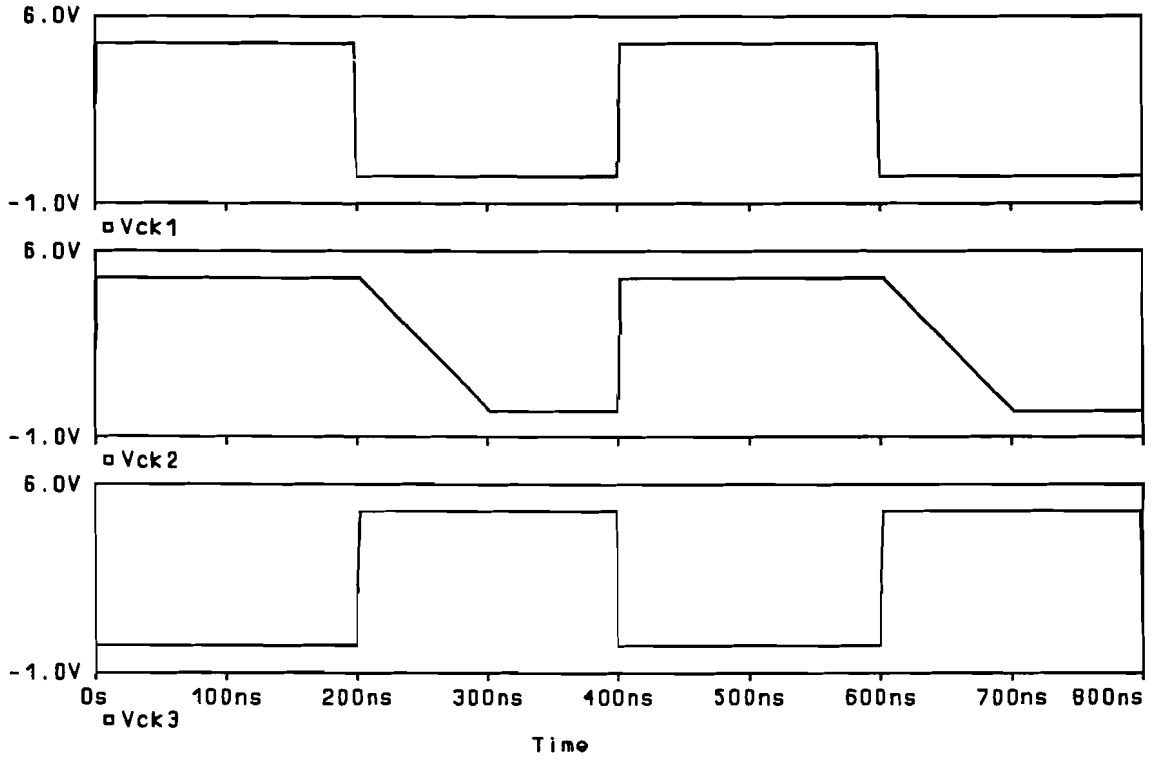
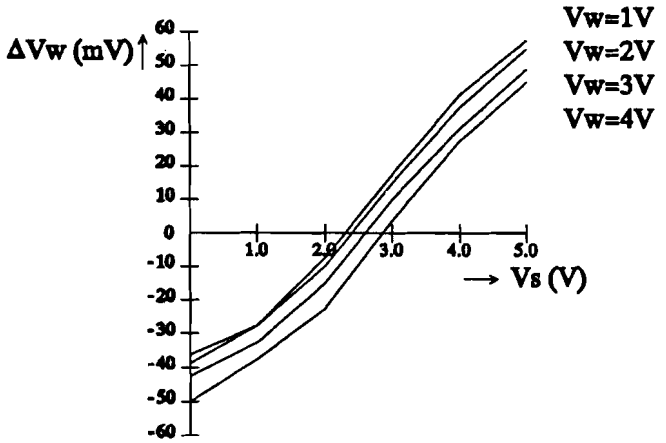


Fig. 4.6 Control-signals in PSPICE-simulation

In table 4.1 the weight changes per clock-cycle are shown as a function of Vs and Vw. Only values from 1V to 4V are considered for Vw as the range used to represent the weights is approximately from 1 to 4.5V. This range is determined by the voltage range the refresh-circuit is able to refresh adequately. Vs ranges from 0V to 5V to be able to update the weights in different controllable step-sizes. All weight-change values are in mV.

Table 4.1 Weight changes ΔVw [mV] per cycle of update-circuit



Vs (V) \ Vw (V)	1	2	3	4
0	-36	-39	-43	-50
1	-28	-28	-33	-38
2	-7	-10	-15	-22
3	+18	+15	+10	+4
4	+41	+37	+31	+29
5	+57	+54	+49	+45

From table 4.1 and the figure it's clear that the weight changes for a certain voltage Vs are not completely independent of the weight-voltage Vw that is currently stored. Also a kind of saturation occurs for small (<1V) and large (>4V) values of Vs.
However it is believed that these results are good enough to be used in the design: the adaptive operation of the networks cancels out wrong or less accurate adjustments.

4.1.3 Layout

Because the weight-storage is the most important and difficult part of a neural network a layout of the proposed weight-storage was made. Another reason for implementing these parts on a chip is that the operation of these circuits depends heavily on parasitic effects and simulations for such circuits are not very reliable.
The technology used for fabricating the chips is the 2.4μ double poly double metal n-well CMOS process of IMEC in Leuven. The layout-design consists of two parts: an update-part and an update-part including the refresh. This is done for being able to test both circuits separately. Also added are two buffers between the output of both circuits and the pad. This is done only for being able to perform measurements on the chip. In the real design these buffers can be omitted. The total circuit the layout is made for is shown in fig. 4.7.

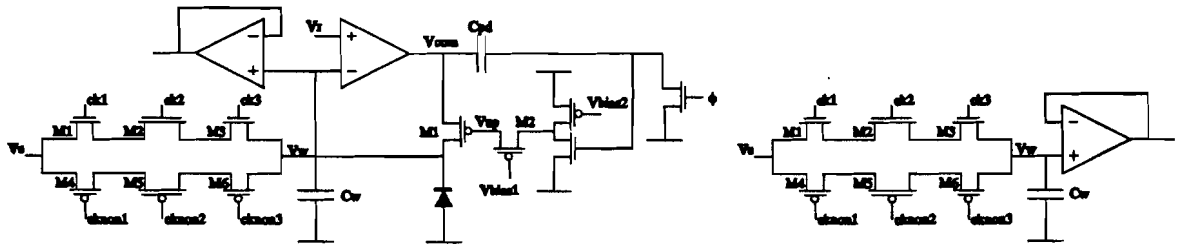


Fig. 4.7 Total circuit for the layout

In fig 4.8 the opamps for the comparator in the refresh-part and for buffering are shown on transistor-level. The left circuit is of the comparator-opamp. All transistors in this circuit are minimum-sized (2.4µm over 2.4µm). The dimensions of the transistors in the buffer-opamp are shown in the circuit. Cm is the Miller-capacitor used for frequency compensation. The value of this capacitor is 1.8 pF.

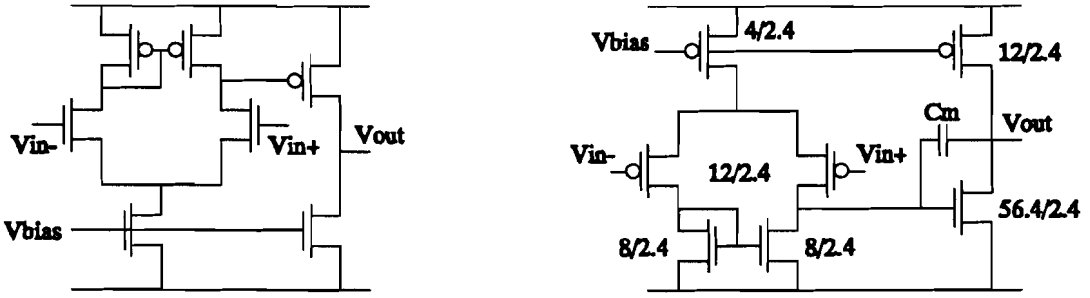


Fig. 4.8 Comparator-opamp and buffer-opamp

The layout is designed using DALI (Delft Advanced Layout Interface). The layers used in the design are:

- two poly-silicon layers (poly1 and poly2)
- two metal layers (metal1 and metal2)
- an active layer
- an n-well used to create p-type transistors
- a p-diffusion layer also for p-type transistors

There are two ways of creating capacitors with these layers. In this design capacitors are constructed using one poly-silicon layer on top of the other (poly2 on top of poly1). If two metal tracks have to cross each other without making contact the metal2-layer is used.
The layout is shown in appendix 1. On the left the update including refresh is designed; the right part only consists of the update-circuit. The large square on both sides is the charge-capacitor Cw (800 fF). Just below the charge capacitor the six transistors of the update-circuits are placed. In the middle the two opamps for buffering the outputs can be seen. These opamps were already designed in [15]. The little squares connected together form a large capacitor. This is the Miller capacitor in the opamp. In the top left corner the refresh circuit is situated with on the left the opamp used as comparator and on the right the rest of the refresh circuit. The location of these transistors and the capacitor Cpd corresponds with the drawing of fig. 4.1. The dark metal line at the bottom is the ground-line. This line moves up as ground for the refresh-circuit and the charge-capacitors. The metal line at the top is the Vdd-line. This line extends to the n-wells (dotted regions) for connecting the bulks of the p-transistors to Vdd.

In appendix 2 the layout is shown with the pads and the lines connecting these pads with the desired points in the circuit. All lines going to the gate of a transistor are connected to a special gate-protection pad. These gate-protections need a power supply which must be connected to the pads PFVDD and PFVSS. In appendix 2 also the number of the pin which is connected to each pad is given. Pad-names with an extension of _1 (ck1_1 etc.) are pads for the update-circuit without refresh. The _2 extension is for the refreshed

circuit. Signals that cause no confusion because they're only used in one circuit (such as Vbias1) are denoted without _2. The pad VBIAS_buffer is used for the bias-voltage for the buffer-opamps. This pad is connected to both buffers.

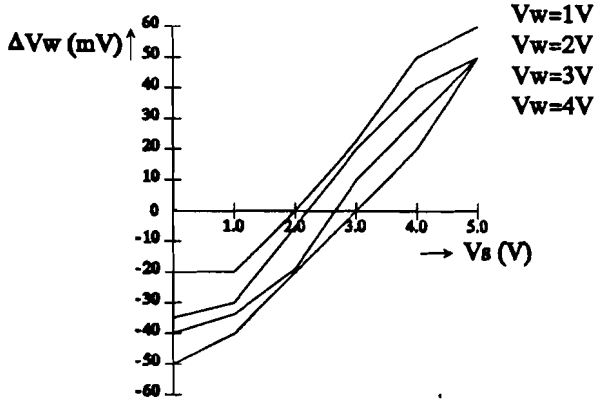
In appendix 3 the taping of the pads to the pins is shown. Pin number 1 is (standard) connected to the bottom of the silicon wafer and must be connected to the lowest potential in the circuit. Pin 28 is not connected.

As it is only a test chip for verifying wether the update and refresh are functioning properly, area considerations don't have the highest priority. The size of the circuit (i.e. of the layout in appendix 1, thus without pads) is now approximately 195µ x 400µ. The buffers, which are not needed in the final chips, occupy most of this area. The area the update+refresh-circuit without buffers would occupy is roughly estimated on 120µ x 120µ. Circuit extractions from the designed layout show that it corresponds with the circuit of fig. 4.7. The circuit extractor can extract capacitors as well. These values cannot be used directly in PSPICE as they are determined just by looking which layers lay on top of each other and by computing what capacitance this overlap causes. Most of these capacitances are included in the PSPICE-models as well. Only the capacitances caused by interconnecting transistors are omitted in PSPICE. After correction of the capacitors PSPICE accommodates for in its models, most of the capacitors that remain are negligible.

4.1.4 Test results

First the update-circuit without refresh is tested. Therefore three clockpulse generators with controllable delay, rise-time, fall-time, pulse-duration and frequency are used. They also have both a positive and negative output. The first generator produces ck1 and cknon1, the second ck2 and cknon2 and the third ck3 and cknon3. The first generator triggers the others so by a proper delay for the generators 2 and 3 the control-signals of fig. 4.4 can be built. By manually triggering the first generator one clock-cycle is generated and the voltage-changes are measured with a digital volt-meter. The results of this test are displayed in table 4.2.

Table 4.2 Test results update-circuit (weight-changes ΔV_w [mV] per cycle)



Vs (V) \ Vw (V)	1	2	3	4
0	-20	-35	-40	-50
1	-20	-30	-35	-40
2	0	-5	-20	-20
3	+20	+20	+10	0
4	+50	+40	+30	+20
5	+60	+50	+50	+50

From these test results it can be concluded that the weight changes are even more dependent on the current stored values than appeared from the PSPICE-simulations. Because the phenomenon of the charge-transfer depends heavily on parasitic capacitances and therefore is not well defined the differences between the PSPICE-simulations and the test-results are hard to explain. However also these results seem useful because the direction of the weight-change is well defined for voltages of Vs below 2V and above 3V. Voltages between 2V and 3V show a relative small weight-change (no more than one quantization-level) so fine-tuning of the weights is also possible. These results are measurements on only one test-chip. To know whether these results are representative, more chips have to be tested on their behaviour.

The update with refresh is tested with the test-configuration of fig. 4.9. A master-clock increments a 12 bit counter on each pulse. The output of this counter is connected to a 12 bit DAC. This DAC delivers a current at its output so an opamp is used to convert this current into a voltage resulting in the staircase Vr. The variable resistor R_r is used to trim the maximum output voltage to 5V. The staircase is incremented on both the rising edge and the falling edge of the clock ϕ. That's why the master-clock is divided by two to form the desired clock ϕ. The voltages Vbias1, Vbias2, the control-clocks ck1 to ck3 and the power-supply are as in the PSPICE-simulation. VBIAS_buffer is set to 3.6V, Vbias_-opamp to 1.4V. Vs_2 and the clocks ck1 to ck3 are used to set the voltage Vw (Vout_2) to a specific value.

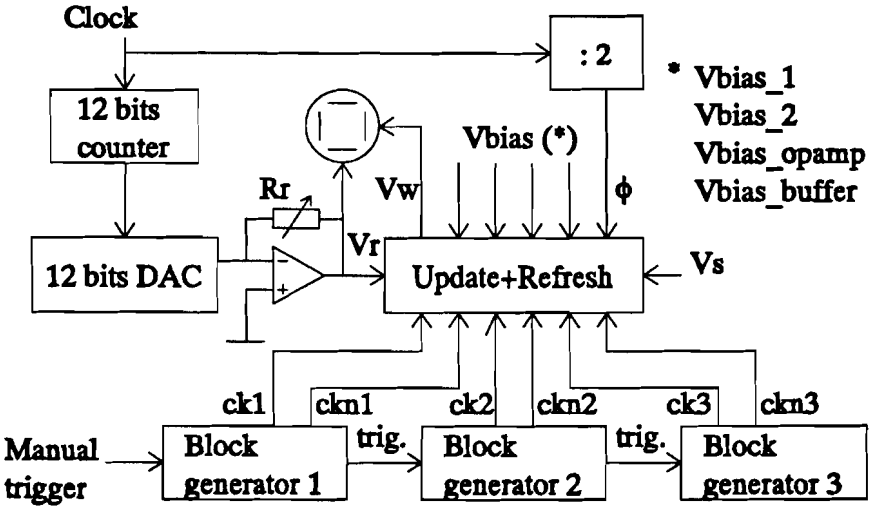


Fig. 4.9 Test configuration update+refresh

In appendix 4 a drawing of the signals Vw and Vr as a function of time is plotted. In this situation for clarity only 4 bits of the DAC are used resulting in 16 levels. At time t of approximately 3 seconds Vw is refreshed to the quantization level at 2.45V. After that the voltage decreases to 2.25V at time t=16.75 sec. Then Vr rises above Vw and brings Vw back to 2.45V. The frequency of the masterlock is now approximately 2.4 Hz.
The small offset between the time Vw jumps from 2.10V to 2.45V and the moment Vw goes up is caused by the method of plotting. Because only an X-Y-recorder with one channel was available, first the staircase was plotted. Then the signal Vw was plotted by

manually starting the X-Y-recorder at the same time as with the staircase. This was done a little too late resulting in the small difference in time between the two signals.

The number of bits used for the DAC was expanded to 8, resulting in 256 steps in the staircase. Only when ϕ is low, a refresh can occur so the number of quantization levels is 128. The frequency of the masterclock is increased with a factor four when one bit is added. This is because the leakage of the capacitor must be the half value of the leakage with one bit less. So the time of one complete period of the ramp must be halfed, which requires a frequency of four times the previous frequency (the counter counts to twice the previous value and the time required for this must be halfed). This results in a masterclock frequency of approximately 600 Hz with 8 bits and 40 mV between two consecutive quantization levels. In this situation the refresh-circuit works properly for a voltage-range of 1.0 V to 4.20 V.
When the number of bits is increased to 9, no improvement in storage-resolution is achieved: 40 mV between two quantization levels. The frequency of the master-clock is now 2400 Hz. The timing of the clock-signals is quite critical: in fig. 4.2 it can be seen, that the transistor is shut off by the output of the opamp (Vcom) just before ϕ goes high. If ϕ goes high before Vcom shuts off the transistor M1, Vup causes M1 to stop charging Cw instead of Vcom. That's why the timing is changed: Vr is incremented on the falling edge of the masterclock, which is also the clock ϕ. The duty-cycle of this clock is less than 50% so the time for the voltage on the charge-capacitor Vw to reach Vr is increased. In this case, all the steps in the staircase Vr are possible quantization levels. If 8 bits are used for the DAC, in this way the quantization levels lie approximately 20 mV apart. Testing with this configuration yields the following: some steps in the staircase are skipped so the quantization levels lie alternating 20 mV and 40 mV apart. Also when the number of bits is increased to 9 or 10, this skipping of levels occurs resulting in quantization levels lying 10 mV to 40 mV apart.

Another thing is, that the leakage of the charge-capacitor Cw differs from chip to chip, so different clock-frequencies ϕ must be chosen to prevent the capacitor from leaking to zero volts.

4.2 Multipliers

In every synapse cell (fig. 3.4) three multipliers are required. Therefore it's very important to keep these multipliers as small as possible. Another very important topic in designing the synapse-chip is power consumption. Representing all signals by DC voltages and currents would consume too much power; the power consumption depends (approximately) linearly on the number of synapses, while this amount of synapses should be as large as possible. Therefore the signals are represented by current- and voltage-pulses. The frequency and/or value of these pulses determine the value of a signal. Because of this representation a dynamic multiplier can be used. The circuit of the multiplier is given in fig. 4.10 (refer to [5] and [32]).

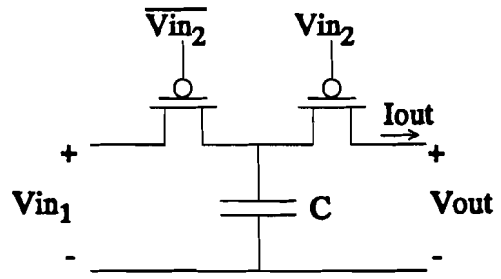


Fig. 4.10 Switched capacitor multiplier

It's a switched capacitor multiplier in which the capacitor C is constantly charged and discharged. The circuit works as follows. Vin_1 is a voltage that can be positive or negative. Vin_2 is the pulsed input; the frequency of the pulses determines its value. First when Vin_2 goes high the capacitor is charged to Vin_1. After that Vin_2 goes low again and the capacitor is discharged over the output delivering an average output current:

$$Iout_{avg} = C \cdot (Vin_1 - Vout) \cdot freq.(Vin_2)$$

The choice of the value of C is a trade-off between minimisation of power consumption (small C) and clock feedthrough (large C). [32] uses a 100fF value of C resulting in an average current $Iout_{avg}$ of 5μA with $(Vin_1 - Vout) = 5V$ and $freq.(Vin_2)$ is 10 MHz.
Advantages of this circuit are the small size and low power consumption. If the frequency-range of the used signals is known, effects of clock-feedthrough must be examined to see if this multiplier is still usable.

4.3 Integrators

Using switched-capacitor (SC) multipliers has implications on the total design of the chips. As the output of a SC-multiplier are current-pulses an integrator must be used to convert these pulses to a voltage that can be distributed to several blocks. In the synapse-chip this holds for the multiplication of BPIS' with η to form UD (see fig. 3.3) and the generation of voltage Vs in the synapse-cell to update the weight. A simple integrator made with an opamp and a capacitor is used to convert UD to a voltage which is distributed to one column of synapse-cells. In the synapse-cell only a capacitor is used to integrate the incoming current-pulses creating the voltage Vs to keep the synapse-cell as small as possible. Fig. 4.11 shows an overview of the synapse-chip with this new representation of signals and integrators. A p-subscript means that the signal is pulsed.

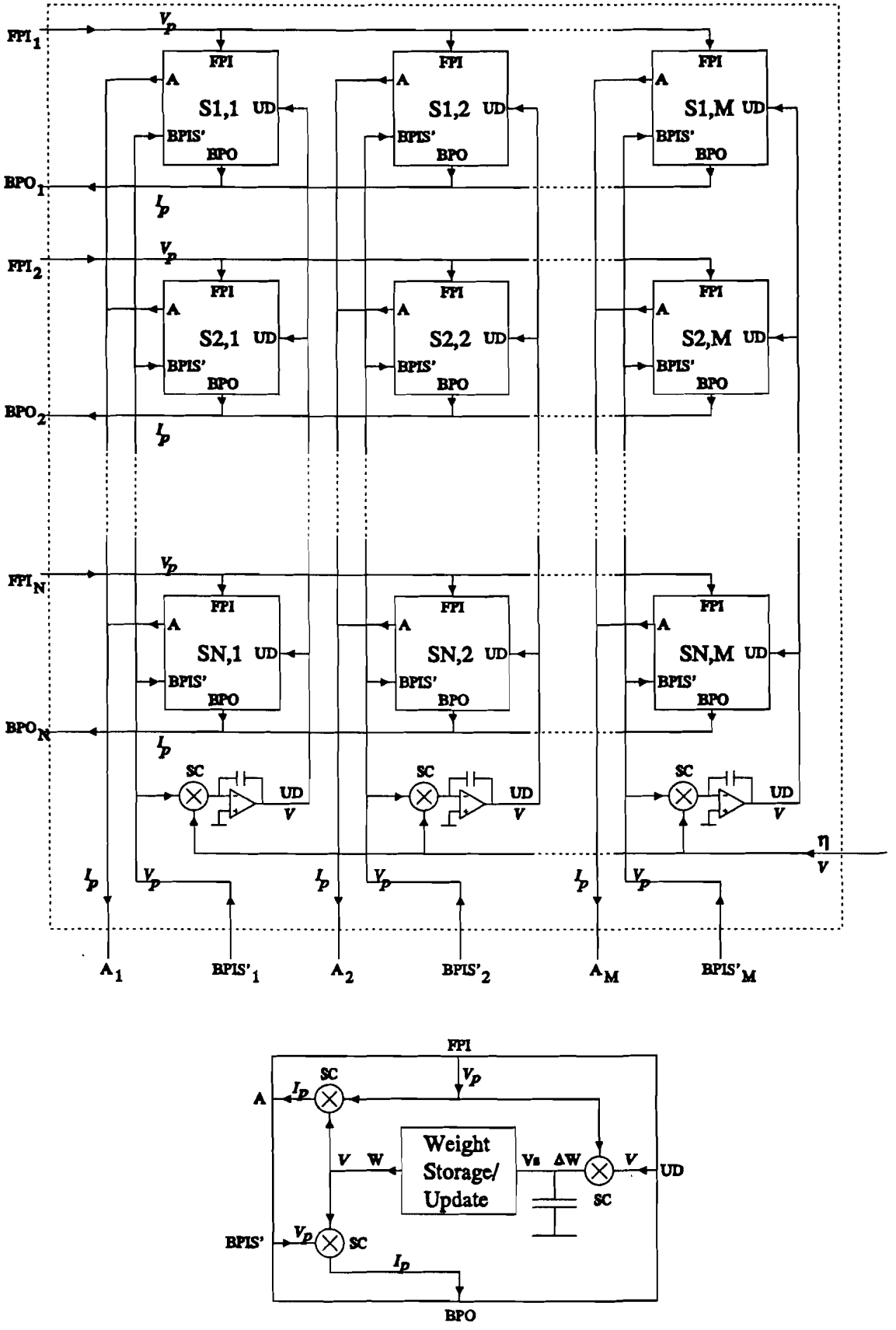


Fig. 4.11 Synapse-chip with new representation of signals

5 The Neuron-chip

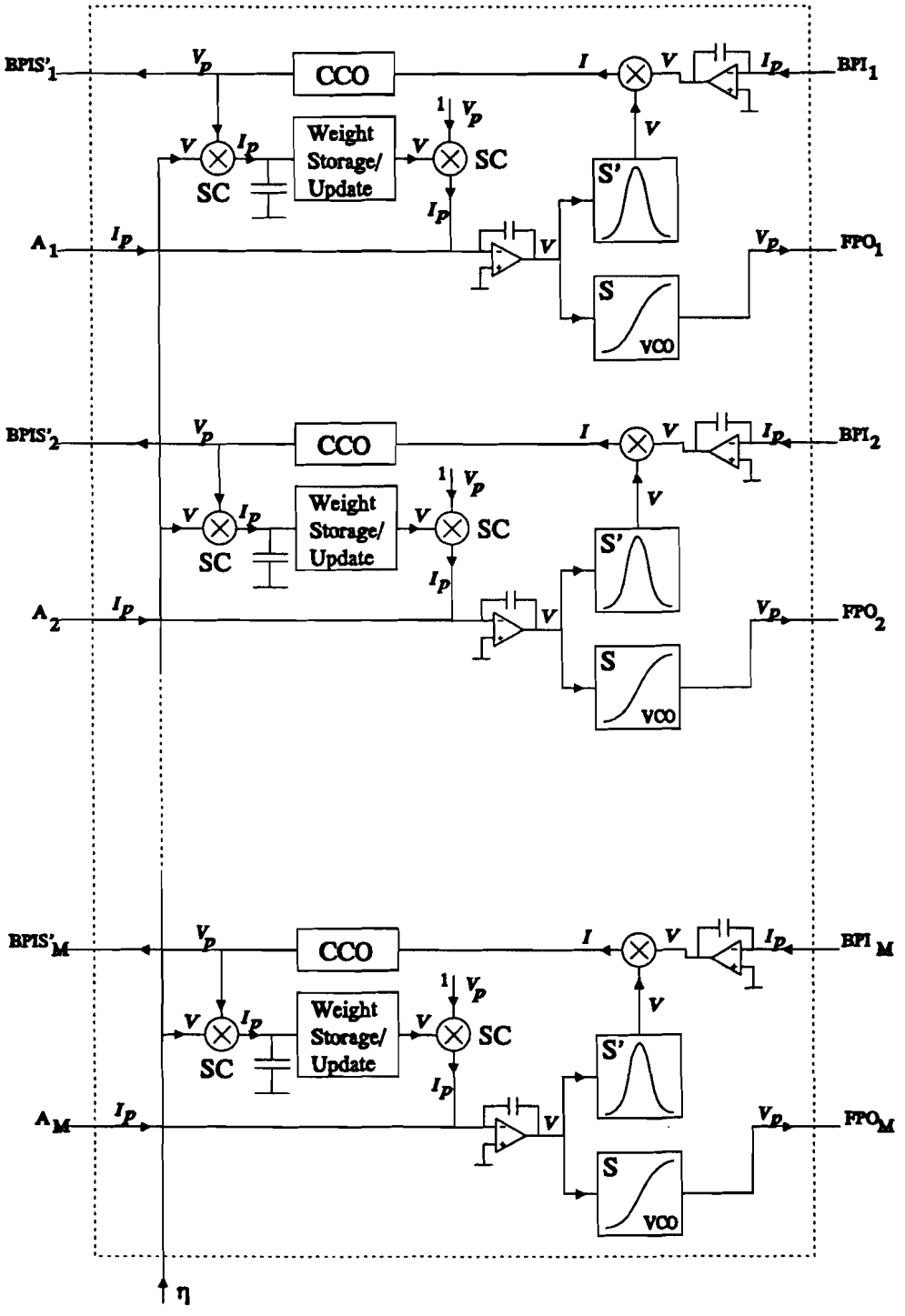


Fig. 5.1 Neuron-chip with new representation of signals

The representation of signals by pulses has also implications for the neuron chip. In fig. 5.1 an overview of the neuron-chip with the new representation is given.

5.1 Integrators

Three integrators per neuron are used in the neuron-chip. The first one converts the summed current-pulses of all synapse-cells connected to it (A) and the variable threshold contribution to a voltage. This voltage drives the circuits for the sigmoid-derivative S' and the VCO with sigmoid characteristic. The second converts the BPI-signal into a voltage. This is done because the BPI-signal is used as input for a multiplier and no multipliers are available with a pulsed current input. These two integrators are made using an opamp with a capacitor across.
The third integrator is the one used for the weight storage. This integrator is a single capacitor as the one in the synapse-cell: it creates the voltage Vs from incoming current-pulses for updating the weight.

5.2 Multipliers

Most of the multipliers are the same SC-multipliers as in the synapse-chip. However the multiplication of BPI with S'(A) needs a static multiplier with two voltage-inputs. In [32] the multiplier of this type with the best properties is the one shown in fig. 5.2.

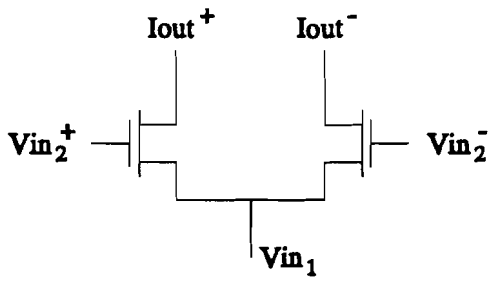


Fig. 5.2 Static multiplier

If $Vout^+ = Vout^-$ and M1 and M2 operate in the linear area the following formulas hold.
For $Vout > Vin_1$:

$$Iout^+ = \beta \cdot (Vin_2^+ - Vin_1 - V_T - 1/2 \cdot (Vout - Vin_1)) \cdot (Vout - Vin_1)$$
$$Iout^- = \beta \cdot (Vin_2^- - Vin_1 - V_T - 1/2 \cdot (Vout - Vin_1)) \cdot (Vout - Vin_1)$$
$$\Rightarrow \quad Iout = Iout^+ - Iout^- = \beta \cdot (Vin_2^+ - Vin_2^-) \cdot (Vout - Vin_1) \qquad (5.1)$$

If $Vout < Vin_1$ drain and source are exchanged. The above formulas now become:

$$Iout^+ = -\beta \cdot (Vin_2^+ - Vout - V_T - 1/2 \cdot (Vin_1 - Vout)) \cdot (Vin_1 - Vout)$$
$$Iout^- = -\beta \cdot (Vin_2^- - Vout - V_T - 1/2 \cdot (Vin_1 - Vout)) \cdot (Vin_1 - Vout)$$
$$\Rightarrow \quad Iout = Iout^+ - Iout^- = -\beta \cdot (Vin_2^+ - Vin_2^-) \cdot (Vin_1 - Vout) \qquad (5.2)$$

Combining equations (5.1) and (5.2) yields: $Iout = \beta \cdot (Vin_2^+ - Vin_2^-) \cdot (Vout - Vin_1)$

Fig. 5.3 shows a PSPICE-simulation of the multiplier. The following values are used:

$Vout = 1.5\ V$ $\quad -1.0\ V \leq Vin_1 - Vout \leq 1.0\ V$

$Vin_2^- = 5\ V$ $\quad -1.0\ V \leq Vin_2 \leq 1.0\ V$

$Vin_2 = Vin_2^+ - Vin_2^-$

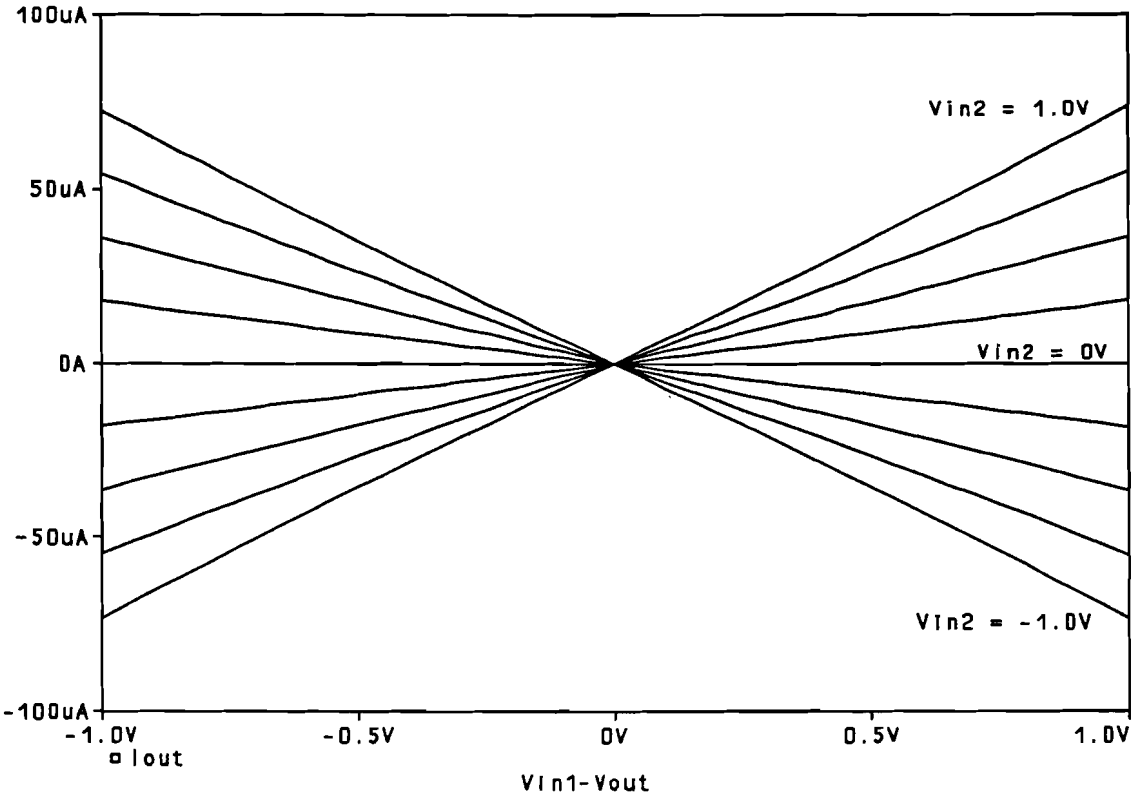


Fig. 5.3 PSPICE-simulation multiplier

Fig. 5.3 shows that the multiplier is excellently linear in all four quadrants. Drawback is the rather large power consumption. This is because Vin_2^+ and Vin_2^- need a high bias so large currents flow through input Vin_1. Another drawback is, that the output of the multiplier is a differential current, so the next circuit must accept this current as input or a differential to single current converter must be used.

5.3 Current Controlled Oscillator

The current controlled oscillator (CCO) is used to convert the current from the output of the static multiplier to a voltage pulse-train. The circuit is presented in fig. 5.4 (refer to [18] and [19]). If the input to the circuit is a steady current a train of voltage pulses will be produced at the output. The circuit consists of a non-inverting amplifier (M_3 to M_6) with positive feedback by capacitor C_2, a capacitor C_1 whose voltage serves as input to the

amplifier and a current source switched by the output voltage (M_1 and M_2). It works as follows. If the output is low the current I_{in} charges capacitors C_1 and C_2 raising voltage V_c. If V_1 reaches the trigger-value V_L (= unity-gain point) of the non-inverting amplifier the output changes from ground to V_{DD}. This causes the voltage V_c to jump from V_L to V_L + ΔV_c. Here ΔV_c is equal to $V_{DD} \cdot \frac{C_2}{C_1+C_2}$. Now Vout is high turning on transistor M_2. Hence the capacitors C_1 and C_2 are discharged by a current equal to I_{sink} - I_{in}.

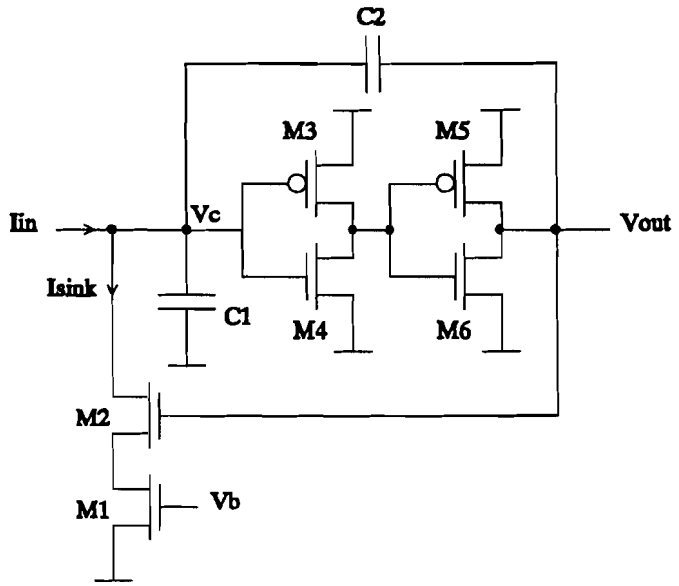


Fig. 5.4 Current Controlled Oscillator

If the voltage V_c has dropped below the high trigger-value of the amplifier V_H (second unity-gain point), Vout changes from V_{DD} to ground. The voltage V_c becomes equal to V_H - ΔV_c where ΔV_c is the same amount as before. This sequence is repeated delivering a voltage pulse train at the output. In fig. 5.5 a PSPICE-simulation of the pulse generator is shown. If it is assumed that the voltage-difference V_L - V_H is much less than ΔV_c, as will be the case normally, the following calculations can be made.

The time it takes to charge V_c up an amount of ΔV_c is the time between pulses (t_{low}). In this case I_{in} charges both C_1 and C_2 so the calculation of t_{low} becomes:

$$t_{low} = \frac{\Delta V_c}{dV_c/dt} = \frac{C\Delta V_c}{I_{in}} = \frac{(C_1+C_2)(V_{DD}C_2)}{(C_1+C_2)I_{in}} = \frac{C_2 V_{DD}}{I_{in}}$$

The high-time of the pulse is determined by the time it takes to discharge V_c by ΔV_c volts. The current to discharge the capacitors C_1 and C_2 is equal to I_{in} - I_{sink}. This results in the following formula for the pulse-width t_{high} of the pulses:

$$t_{high} = \frac{\Delta V_c}{dV_c/dt} = \frac{(C_1+C_2)\Delta V_c}{I_{sink}-I_{in}} = \frac{C_2 V_{DD}}{I_{sink}-I_{in}}$$

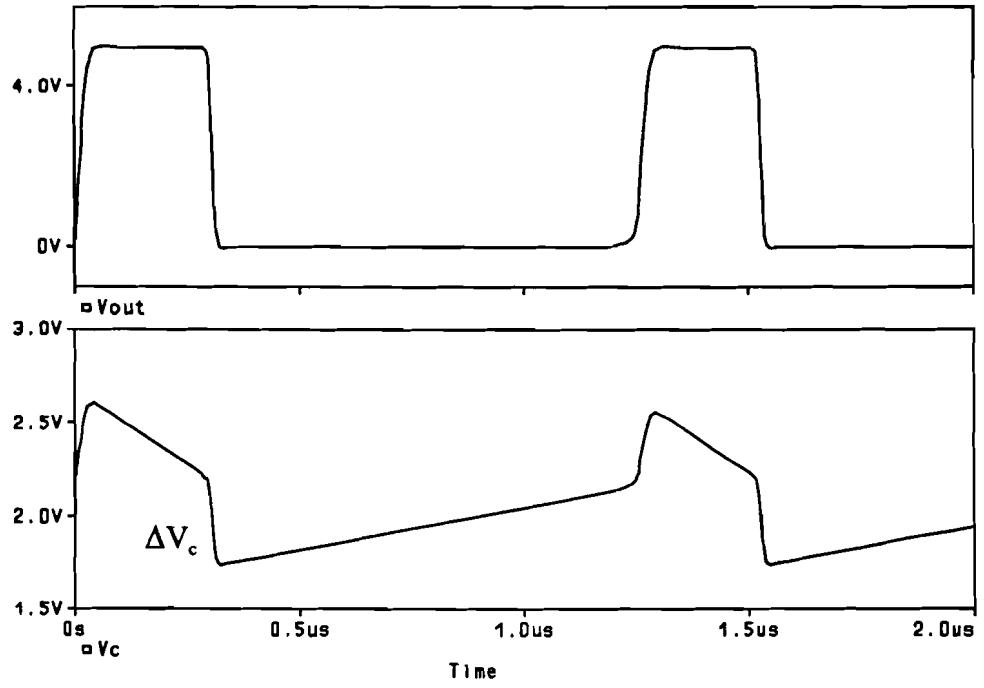


Fig. 5.5 PSPICE-simulation Current Controlled Pulse Generator

From the equations of t_{low} and t_{hi}, it is clear that t_{low} is inversely linear with I_{in}, while t_{high} is inversely linear with I_{sink}-I_{in}. So the frequency of the pulse generator is not linear with the input-current I_{in}. If the high-time t_{high} of the pulse is much lower than the low-time t_{low}, the pulses generated are completely determined by the low-time and thus the frequency of the pulses are linearly dependent of I_{in}.

5.4 Voltage Controlled Oscillator

The voltage controlled oscillator (VCO) with sigmoid characteristic is the last step in the neuron-chip for the forward propagating signals. It generates the output of the neuron-chip which must be a voltage pulse-train. The input of this VCO is the voltage representing the activation-signal. It is believed that the S-curve of the circuit doesn't have to implement the function $\frac{1}{1+e^{\theta(\tau-\alpha)}}$ exactly; a look-alike is enough.

The output-frequency ranges from a minimum frequency f_{min} to a maximum f_{max} with saturation at both ends. The circuit is given in fig. 5.6 (refer to [17] and [18]).

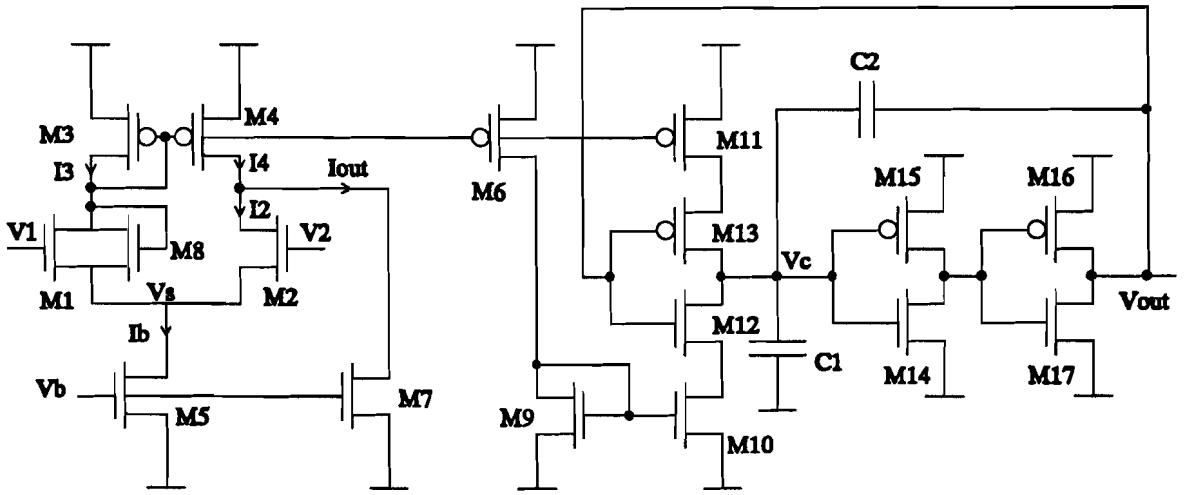


Fig. 5.6 Sigmoid-like pulse generator

It consists of two parts: the first part provides the sigmoid function by using a transconductance amplifier. The non-inverting terminal is provided with the input-voltage V_1 while the inverting input is fixed at a bias voltage V_2. This bias voltage V_2 determines the centre frequency of the sigmoid curve. The differential pair is made up by the transistors M_1, M_2 and M_5. For the saturated drain currents in the subthreshold region the following formulas apply:

$$I_2 = I_0 e^{\kappa V_2 - V_s}$$

$$I_3 = I_0 e^{\kappa V_1 - V_s}$$

$$I_b = I_2 + I_3$$

Here κ is a constant, that expresses the effectiveness of the gate in determining the surface potential (refer to [19] for the calculation of κ).

Solving for I_3 yields: $I_3 = I_b \frac{1}{1 + e^{\kappa(V_2 - V_1)}}$; a sigmoid function.

This output current of the transconductance amplifier is mirrored to the second stage by the transistors M_6, M_9 and M_{11}. Transistor M_8 is added to mirror a minimum current to the second stage independent on the input-voltage V_1. This minimum current I_{min} determines the lowest frequency f_{min} and can be varied by the W and L of M_8. If no pulses are desired as zero-level ($f_{min} = 0$) transistor M_8 can be omitted. Transistor M_7 is an active load for the output current $I_{out} = I_4 - I_2$. This load determines the upper limit of the current and thereby f_{max}. By varying the aspect ratio of M_7 or the bias voltage V_b this maximum current can be controlled.

The second stage takes care of the pulse generation from the supplied current I_3. This is a circuit similar to the Current Controlled Oscillator from fig. 5.4. Via the current mirrors, I_3 is supplied as I_{in} by M_{11} and as I_{sink} by transistor M_{10} as well (refer fig. 5.4). The only difference is that here both currents are switched by the output voltage V_{out} via M_{12} and M_{13}.

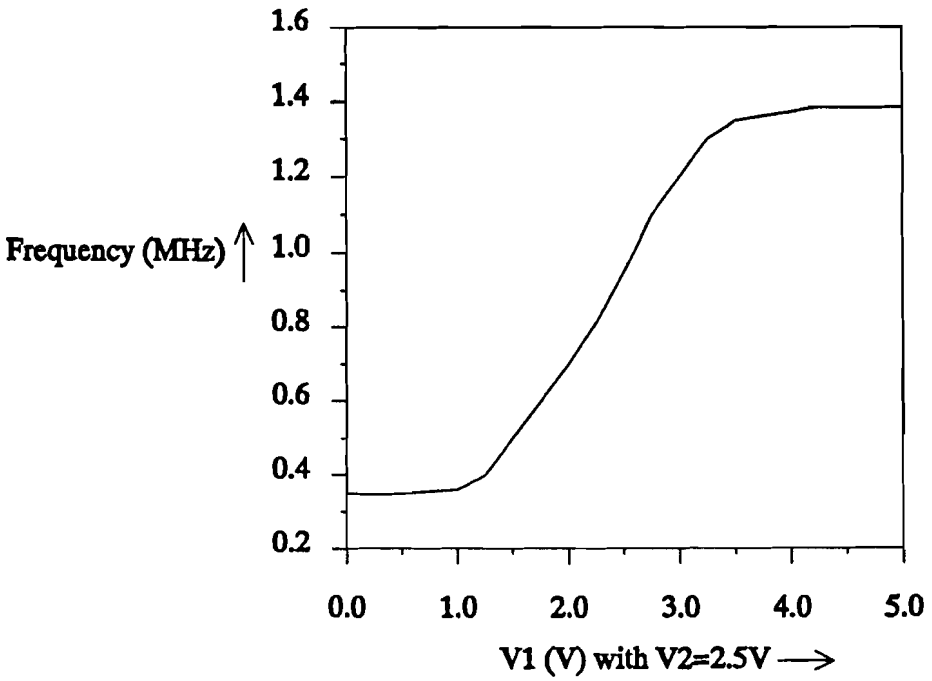


Fig. 5.7 Frequency-voltage relationship

The formulas for both t_{low} and t_{high} now become:

$$t_{low} = \frac{C_2 V_{DD}}{I_3}$$

so square waves are produced. If this is not convenient, the aspect ratio of transistors M_{10} and M_{11} can be changed so different amounts of currents of I_{in} and I_{sink} (fig. 5.4) will flow. In fig. 5.7 a plot of the output frequency as a function of input voltage V_1 is shown for V_2=2.5V and $C_1 = C_2 = 5$ pF.

5.5 Sigmoid derivative

The sigmoid derivative function S' is used in the backward propagation path to multiply the backward propagating input BPI with S'(A) as in the last part of equation (3.3). Also in this case the S'-function doesn't need to be perfect. Until now, the only circuit that appeared in the literature as an implementation of S'is the circuit in [12]. The question is whether the S'-approximation can be simplified further so an implementation with less transistors can be found. Maybe a triangular form or a quadratic approximation as substitute (fig 5.9) would suffice. For an answer to this question simulations on the system level must be carried out.

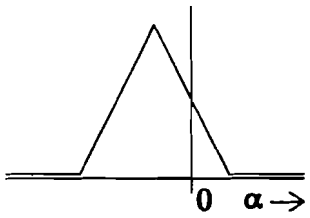


Fig. 5.9 Simplification of S'

But even if the form of fig. 5.9 would satisfy, whether a circuit exists to approximate this function with fewer transistors than the circuit of fig. 5.8 remains to be seen.

6 Control

In this chapter controlling the neural network is discussed. As mentioned in paragraph 3.3 this includes the delay-circuit used for the passing of tokens and the computer-control.

6.1 Delay-circuit

The delay-circuits take care of passing the token from one chip to the other. When a pulse arrives at the input at time t a pulse at the output must appear at time $t+\Delta t$. Here Δt is the time it takes to compute the output-signals when the inputs are known. In fact there are two different delays in each neuron- and synapse-chip: the forward propagating delay and the backward propagating delay. A possible delay-circuit is shown in fig. 6.1.

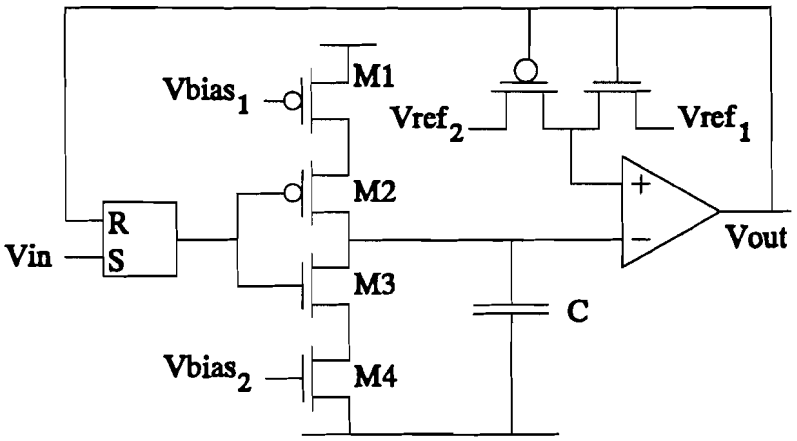


Fig. 6.1 Delay-circuit

The circuit works as follows. Assume the output Vout is low, the output of the flip-flop is low and the voltage on the capacitor C is 5V. When the input Vin goes high the flip-flop is set and the transistor M3 begins to conduct discharging the capacitor C. When the voltage on the capacitor becomes lower than $Vref_2$ the output of the opamp goes high. At that moment the flip-flop is reset again and the transistor M3 is cut off. Transistor M2 is conducting so the capacitor is charged. When the voltage on the capacitor becomes higher than $Vref_1$ the output of the opamp goes down. At last, at the moment the capacitor is charged to 5V the initial conditions are met again.
Fig. 6.2 shows a PSPICE-simulation of the delay-circuit. Used values are:

$Vref_1$ = 2.2V	$Vref_2$ = 2V
$Vbias_1$ = 3.7V	$Vbias_2$ = 1.2V
C = 500 fF	V_{DD} = 5V

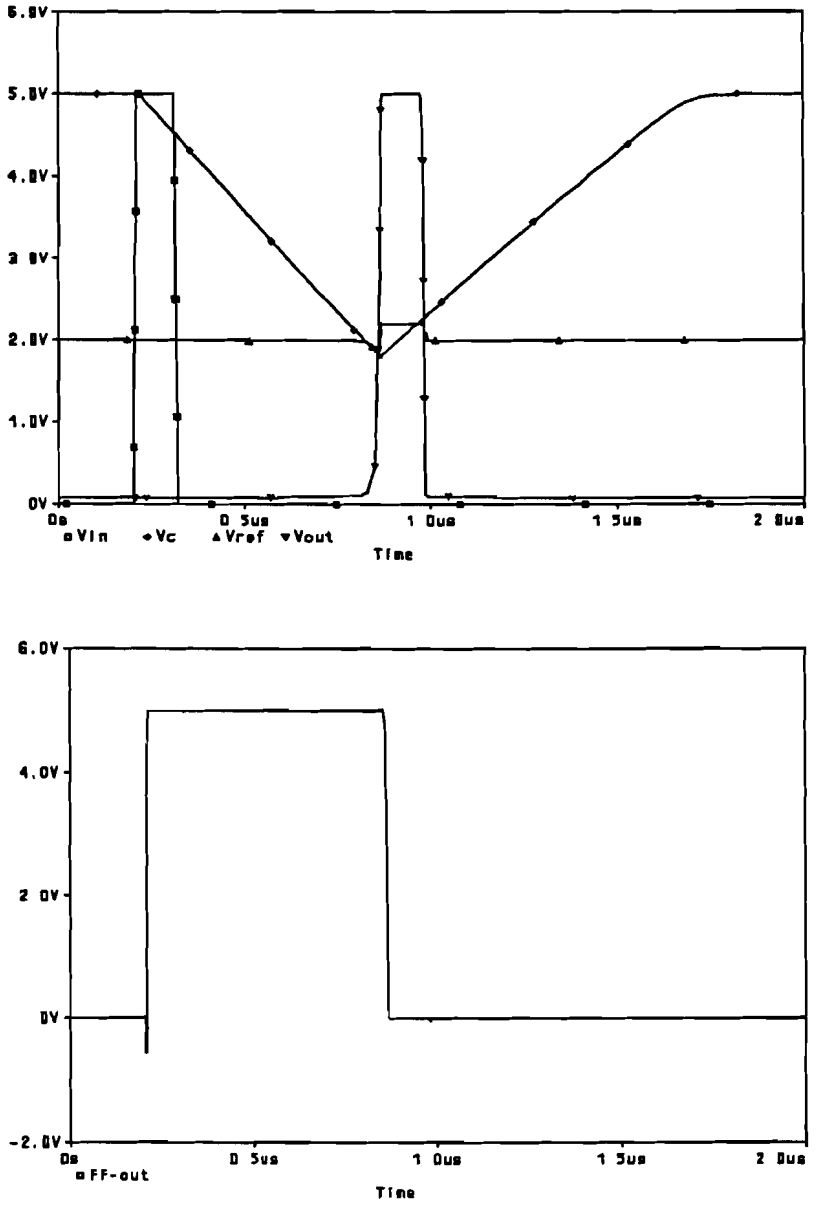


Fig. 6.2 PSPICE-simulation delay-circuit

The time Δt can be computed by the following formula:

$$\frac{5 - Vref_2}{\Delta t} = \frac{I_{sink}}{C} \quad \Rightarrow \quad \Delta t = \frac{(5 - Vref_2) \cdot C}{K \cdot (Vbias_2 - V_T)^2}$$

The voltages $Vbias_1$ and $Vref_1$ only have influence on the resulting pulse-width at the output.

Also a PSPICE-simulation is done in the situation of three cascaded delay-circuits as is the case when a number of synapse- and neuron-chips are connected. The results are shown in fig. 6.3.

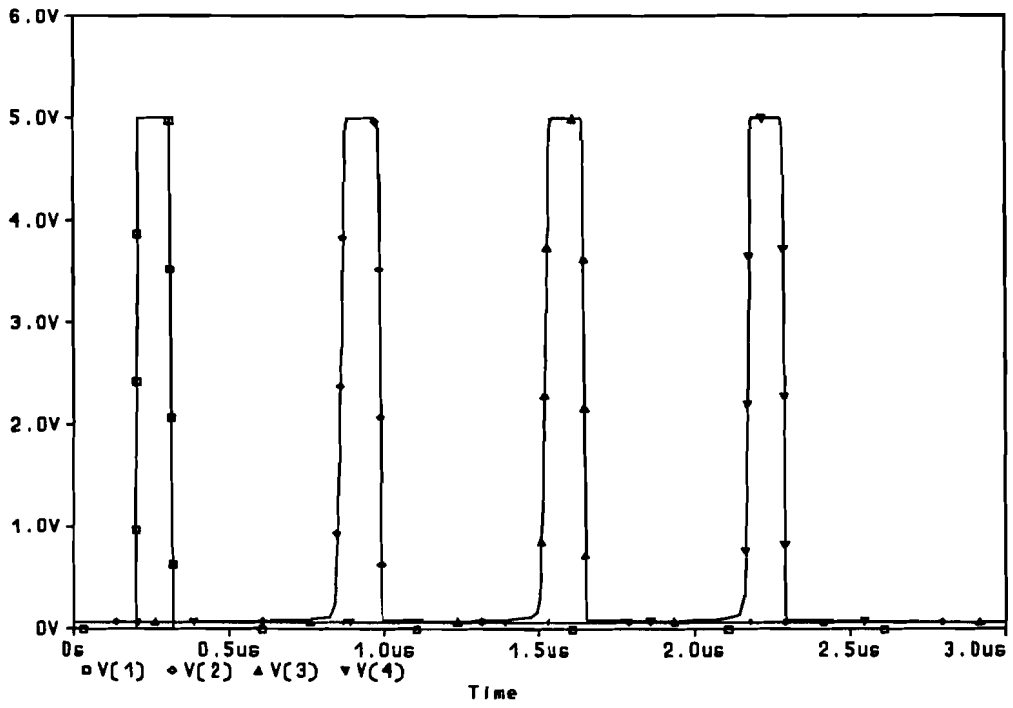


Fig. 6.3 PSPICE-simulation cascaded delay-circuits

Here three delay-sections are cascaded. The first pulse (V1) is the one at the input of delay-section 1. Each time a delay-block receives a pulse at its input it delivers a pulse to the next section 0.65 µs later.

6.2 Normal training and batch-training control

According to step 4 of the back-propagation algorithm, if all the errors are calculated the weights can be updated. Using the token-passing technique, if the signal BPO_VALID from fig. 6.4 goes high all errors are calculated. This signal can serve as a global Adapt_Enable signal for all chips which indicates that the weights can be updated. A possible implementation would be that the Adapt_Enable signal triggers the three pulse-generators for the clocks ck1/ckn1, ck2/ckn2 and ck3/ckn3 so one clock-cycle is generated each time Adapt_Enable goes high. Fig. 6.4 shows a diagram of this configuration.

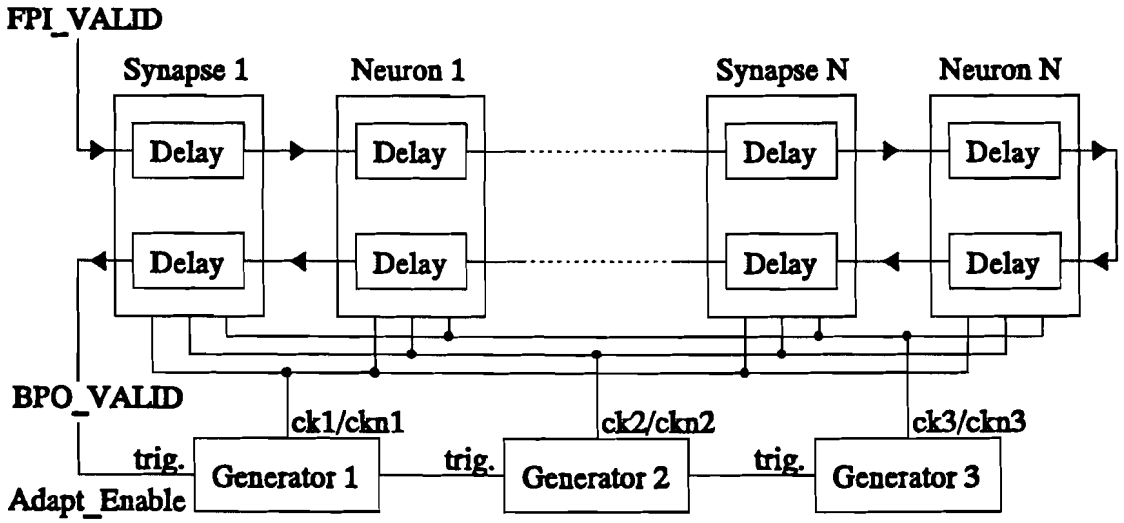


Fig. 6.4 Controlling weight-update

Above the "normal" training control was described; every time when all errors are calculated for a certain input/output-pattern the weights are updated. There is another training mode called batch-training that doesn't adapt the weights with each input/output-pair. With batch-training first the whole set of input/output-patterns (a batch) is presented. The resulting error terms for each element from the set are accumulated and after the complete set has been presented the weights are updated. This is repeated until the error drops below a predefined maximum error. To be able to do batch-training the integrator-circuit before the weight-storage must be changed; a new capacitor is added to accumulate the contributions of the error with each element from the training-set. The new circuit that results from these considerations is presented in fig. 6.5.

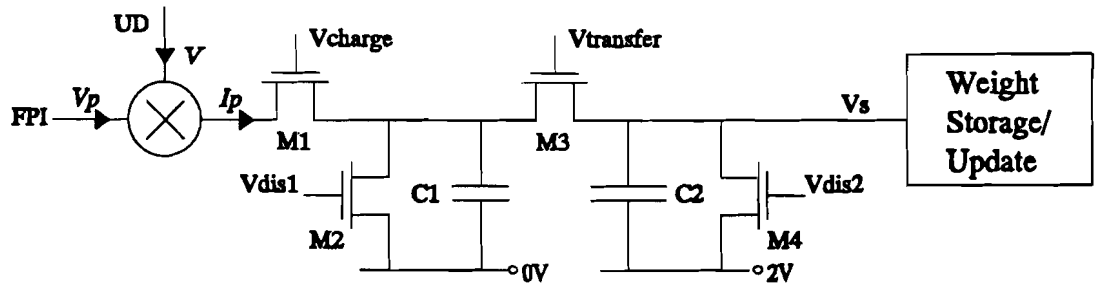


Fig. 6.5 Weight Storage Integrator for batch-training

In fig. 6.5 C1 is the capacitor that stores the temporary weight-changes. C2 is the accumulation-capacitor that sums up the weight-changes stored on C1. As shown in the update-circuit a voltage Vs of approximately 2V results in no weight-change so this voltage is the "zero-level" of C2. The control-signals Vcharge, Vtransfer, Vdis1 and Vdis2 are shown in fig. 6.6.
The dynamic multiplier calculates the multiplication of FPI with UD (refer fig. 4.11)

resulting in a train of current-pulses. This pulse-train is integrated on C1 during the time Vcharge is high. After Vcharge goes down Vtransfer goes high transferring the weight-change stored on C1 to C2. After that $Vdis_1$ goes high to discharge the capacitor C1.

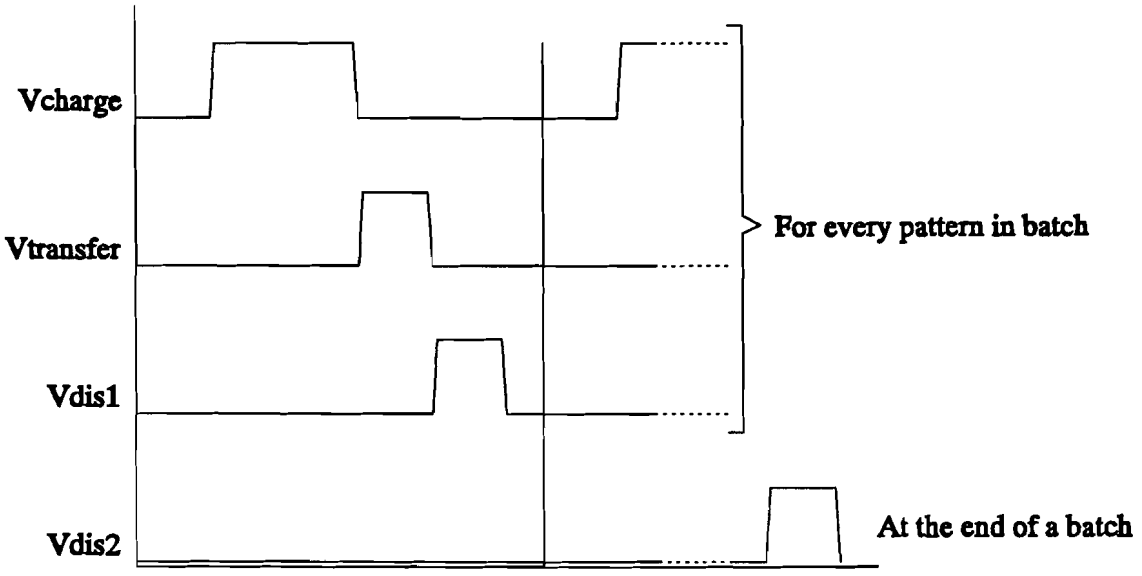


Fig. 6.6 Control-signals batch-training circuit

This is repeated for each pattern in the training set. If all patterns have been presented the final voltage Vs to change the weight is stored on capacitor C2. The weight is updated and after that $Vdis_2$ goes high to discharge the accumulator capacitor C2. Now the sequence can start all over again for the next batch. The ratio between the values of capacitors C1 and C2 determine the contribution of each temporary stored weight change. In fig. 6.7 a PSPICE-simulation is shown of this batch-training.

In this simulation UD has the following values:

0μs < t < 1μs: UD = 3V
1μs < t < 2μs: UD = 1V
2μs < t < 3μs: UD = 5V
3μs < t < 4μs: UD = 2V
4μs < t < 5μs: UD = 0V
5μs < t < 6μs: UD = 4V

The frequency of FPI is kept constant at 2 MHz, C1 = 400 fF, C2 = 800 fF.
The signals Vcharge and Vtransfer can be derived from the delay-circuit (fig. 6.1). The flip-flop output can be used for Vcharge while Vout coincides with Vtransfer. The other signals (Vdis1 and Vdis2) have to be generated separately.

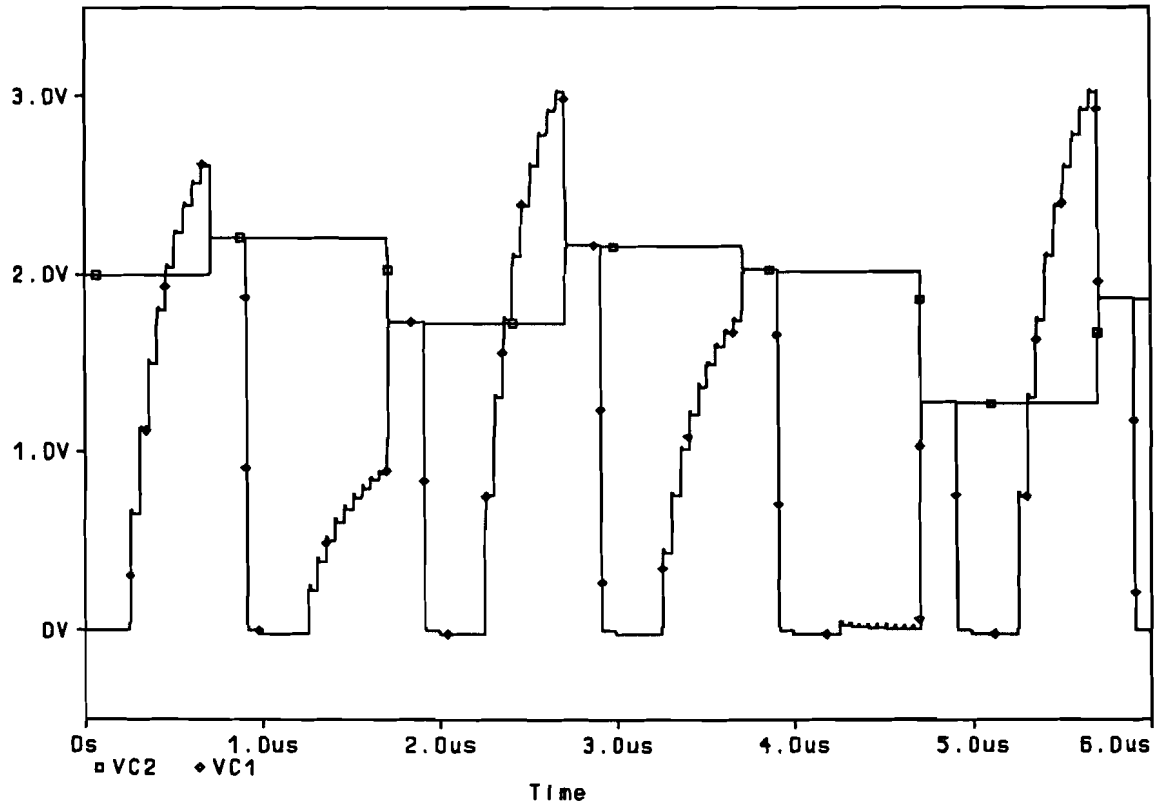


Fig. 6.7 PSPICE-simulation batch-training

6.3 Environmental topics

For easy operation it's the intention that the neural network is realized on a printed circuit board that can be plugged into a slot of a personal computer. This computer must take care of the following topics:

- make the connections between different synapse- and neuron-chips to implement the desired network.
From fig. 3.2 it's clear that the topology of the network determines which chips have to be parallelled and where the different lines have to be interconnected. Also unused synapse-inputs must be grounded, while unused outputs must be left floating. If all wires are already printed on the board, the computer must set switches to connect the wires and to ground them or make them floating.

- program predefined weight-values
If a network is trained before and the weight values are known, it must be possible to read in these values so the training phase can be omitted.

- present the input/output-patterns
The whole set of input-patterns with their correspondent desired outputs are stored in the computer and sequentially presented to the input and output of the network.

- control the FPI-VALID signal and ADAPT_ENABLE-signal
With each pattern that is presented the FPI_VALID signal is triggered so it can propagate through the delay-stages in each chip. On detection of the BPO_VALID signal the ADAPT_ENABLE line can be made high (normal training mode) or the next pattern can be presented until the whole set is handled. In the last case, after the last input/output-pattern the line ADAPT_ENABLE is made high (batch training). Another possibility is that the computer determines the total delay that is expected from the specific configuration and the ADAPT_ENABLE signal is controlled without using the FPI_VALID or BPO_VALID signals.

- read out the output and weight values
To use the network, of course the outputs have to be known. However also the weight values must be available to locate problems (local minima etc.) or to be able to program the weights after the network has been trained.

In fig. 6.8 an overview of the computer-control is presented. The blocks marked with an S or an N are sockets for plugging in a synapse- or neuron-chip. The horizontal lines between the chips are lines that connect all pins of one chip with the other. The mechanism to ground or disconnect one of the lines is not shown. The vertical lines with the switches between two chips in the same column connect every line of the first chip with every corresponding line of the second as in fig. 3.2. Implementing a certain configuration means setting these switches which is done by the Control_1 block.

Programming the weights in a synapse-chip is as follows. First a voltage of 5V (high level) is produced by the DAC. This voltage appears at the input of the left and right multiplexer. Then one capacitor is addressed in each MUX. There is one capacitor for each network-input at the left MUX and one for each network-output at the right MUX. The left capacitors are distributed to all backward propagating synapse-inputs (BPIS') via the switches controlled by the control_2-block. The same holds for the right capacitors. These however are distributed to the forward propagating inputs of all synapse-chips (FPI). If the control_2 block closes one switch of the lower line and one of the upper line which are pointing to the forward and backward propagating input of the same column of synapse-chips, this layer of chips is selected. If the capacitors were properly addressed, a voltage of 5V that was stored on one capacitor on the left and one on the right is now fed to the forward propagating input FPI and backward propagating input BPIS' of one chip.
Referring to fig. 4.11 this means that FPI selects a row of synapse-cells by turning on the switched capacitor multipliers in these cells, while BPIS' selects the SC-multiplier that calculates UD. Now the value to be programmed is produced by the DAC and is fed to all chips via the η/Vprog-line. Via the integrator (fig. 4.11) the value of UD of the desired column will be integrated for a fixed amount of time, such that it becomes equal to Vprog because the SC-multiplier is turned on by BPIS'. This value is fed to the weight storage and update by the SC-multiplier in the synapse-cell, which is turned on by FPI. If all transistors between this point and the storage capacitor Cw (update transistors and batch-circuit transistors) are turned on, Cw will eventually become equal to Vprog.

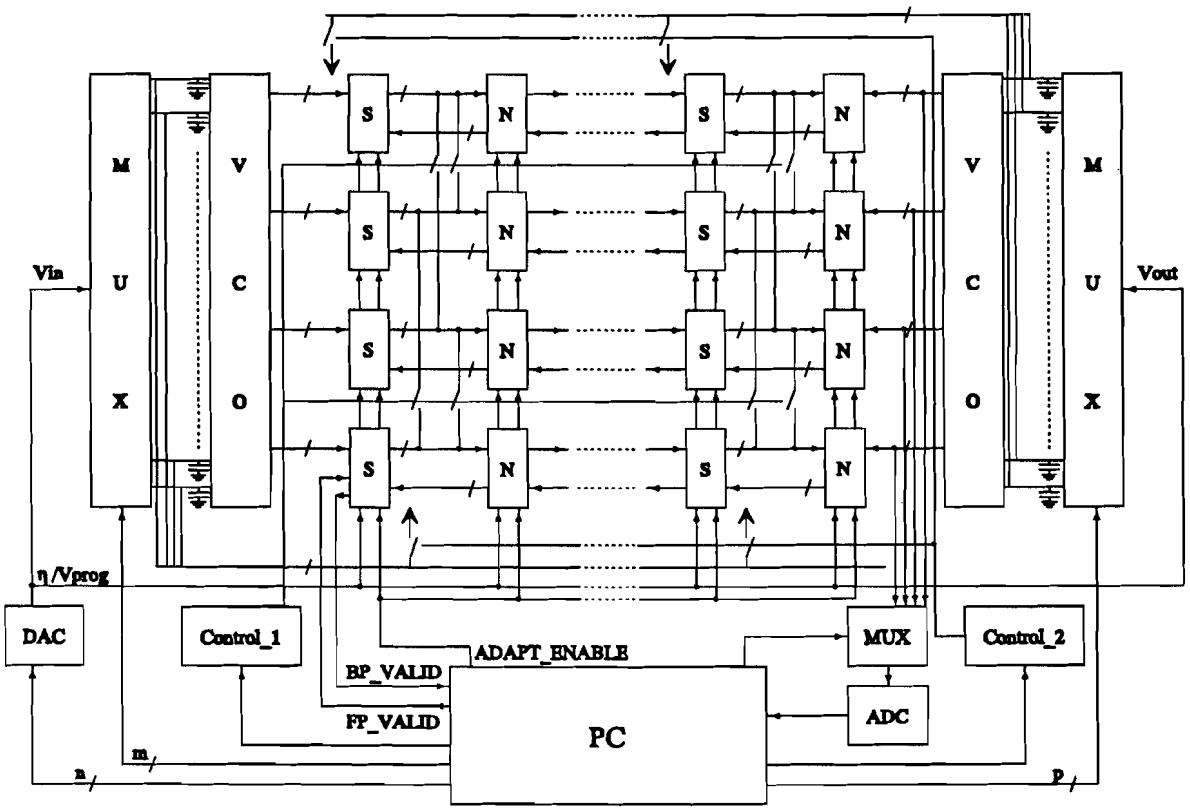


Fig. 6.8 Computer control

By addressing different capacitors in the multiplexers and choosing different layers with the Control_2 block all weights in the synapse-chip can be programmed.

Programming the weights in the neuron-chips is somewhat different to programming the synapse-weights. Here only a left capacitor is addressed and charged to a voltage of 5V. Then the Control_2 block closes a switch of the lower line going to the backward propagating **output** of the desired neuron. By this method a voltage of 5V is fed to the BPIS'-output of the neuron to be programmed. In fig. 5.1 can be seen that the switched capacitor before the weight storage/update is turned on now. By sending the desired weight-value to the DAC this value will appear on the η/Vprog-line and if the transistors in the weight-storage block (update transistors and batch-circuit transistors) are turned on Cw will be charged to Vprog.

Training the network is as follows. First the input- and output-vectors are stored on the capacitors by sequentially producing a value Vin/Vout and addressing the right capacitor. The large VCO-block producing the desired FPI- and desired FPO-signals contains a VCO for every capacitor and thus for every input and output of the network. If the vectors are stored, FP_VALID is made high (if the token-passing is used). The network computes its output and error terms and BP_VALID becomes high if this is done. Now the signal ADAPT_ENABLE can be made high to update the weights (normal training) or the next patterns can be presented (batch-training).

The output of the network is a vector of pulse-trains. These pulse-trains are integrated for a constant time resulting in voltages. The integrators at the output are not shown in the figure. These voltages can be read by selecting them by the multiplexer at the output. This multiplexer passes the voltages to an ADC resulting in a digital word that can be handled by the computer.

Here a first design of a possible implementation of the computer-control is given. It is believed that several improvements can be made. For instance reading out the weights is not implemented yet. Also the number of switches and wires to be used is tremendous. Further investigation has to reduce the total needed hardware to come to a realistic design.

7 Conclusions

For all components in the synapse- and neuron-chips an electronic implementation has been found.

For the most important element of a neural network, the weight storage and update, a test-chip is designed. From the test-results, it can be concluded that the used method of update is supposed to satisfy the needs. For the refresh-circuit, a voltage-range of 3.2 V with 40 mV between two consecutive values can be stored accurately. This results in 80 quantization levels, which might be not enough.

Representing the signals by voltage and current pulse-trains results in very small multipliers and a low power consumption, which is a necessity if chips with a large number of synapses and neurons are built.

8 Future work

The specific representation of signals must be determined (i.e. ranges of frequencies, voltages and currents). In this respect the electronic circuits for most of the components must be optimised for these specific signal-representations.

Layouts have to be made for certain elements in the design to test if they work properly in practice.

To test whether non-idealities in the behaviour of the electronic circuits degrade the performance of the neural network, simulations on system-level must be carried out. This holds especially for the non-idealities in the weight-update and quantization of the weights.

9 Literature

[1] Allen, P.E. & Holberg, D.R.
"CMOS Analog Circuit Design"
Dryden Press, 1987

[2] Andreou, A.G.
"Synthetic neural systems using current-mode circuits"
Int. Symp. on Circuits and Systems, New Orleans, Louisiana,
Vol.3, pp. 2428-2432, May 1-3, 1990

[3] Andreou, A.G. et al.
"Current-mode subthreshold MOS circuits for analog VLSI neural systems"
IEEE Tran. on Neural Networks, Vol. 2, no. 2, pp. 204-214, Mar 1991

[4] Bailey, J & Hammerstrom D.
"Why VLSI Implemenations of Assosicative VLCNs Require Connection Multiplexing"
Proc. IEEE Int. Conference on Neural Networks, San Diego, pp. II-173-180, 1988

[5] Brownlow, M.J. et al.
"Analogue computation using VLSI neural network devices"
Electronics Letters, Vol. 26, no. 16, pp. 1297-1299, Aug. 2, 1990

[6] Card, H.C. & Moore, W.R.
"Silicon models of associative learning in Aplysia"
Neural Networks, Vol. 3, pp. 333-346, 1990

[7] Craven, M.P. et al.
"Frequency Division Multiplexing in analogue Neural Networks"
Electronics Letters, Vol. 27, No. 11, pp. 918-920, May 23, 1991

[8] Curtis, K.M. et al.
"A novel neural network architecture VLSI implementation"
Micro-electronics for neural networks, Proc. 1st Int. Work. Dortmund,
pp. 120-128, Jun 25-26 1990

[9] Curtis, K.M. et al.
"An optimised VLSI implementation for analogue neural communications"
Micro-electronics for neural networks, Proc. 2nd Int. Work. Munich,
pp. 273-279., Oct 16-18, 1991

[10] Del Corso, D. et al.
"An artificial neural network based on multiplexed pulse streams"
Micro-electronics for neural networks, Proc. 1st Int. Work. Dortmund,
pp. 28-39, Jun 25-26 1990

[11] El-Leithy, N. et al.
"Implementation of pulse-coded neural networks"
Proc. of the 27th Conf. on Decision and Control, Austin, Texas
Vol.1, pp. 334-336, Dec 1988

[12] Furman, B. & Abidi, A.A.
"An analog CMOS back error-propagation LSI"
Twenty-second Asilomar Conf. on Signals Systems and Computers,
Vol. 2, pp. 645-648, Oct 13 - Nov 2 1988, Maple Press 1989

[13] Hochet, B. et al.
"Implementation of a Learning Kohonen Neuron Based on a New Multilevel Storage Technique"
IEEE Journal of Solid State Circuits, Vol. 26, No. 3, pp. 262-267, March 1991

[14] Ibrahim, F. & Zaghloul, M.E.
"Design of modifiable weight synapse CMOS analog cell"
Int. Symp. on Circuits and Systems, New Orleans, Louisiana,
Vol. 4, pp. 2978-2981, May 1-3, 1990

[15] Leenaerts, D.M.W.
"TOPICS, a contribution to analog design automation"
Ph.D. dissertation, Eindhoven, januari 1992

[16] Lippmann, R.P.
"An introduction to computing with neural nets"
IEEE ASSP, pp. 4-22, Apr 1987

[17] Maundy, B. & El-Masry, E.
"Pulse Arithmetic in switched capacitor neural networks",
Proc. of the 33rd MidWest Symp. on Circuits and Systems, Calgary, Canada,
Vol. 1, pp. 284-288, Aug 12-15, 1990

[18] Maundy, B. & El-Masry, E.
"SC implementation of Asynchronous Pulse Arithmetic in Artificial Neural Networks",
Int. Symp. on Circuits and Systems,
Vol. 3, pp. 2510-2513, May, 1991

[19] Mead, C.
"Analog VLSI and neural systems"
Addison-Wesley Publishing Co., 1988

[20] Murray, A.F. & Smith A.
"Asynchronous VLSI Neural Networks Using Pulse-Stream Arithmetic"
IEEE Journal of Solid-State Circuits,
Vol. 23, no. 3, pp. 688-697, Jun 1988

[21] Murray, A.F.
"Pulse arithmetic in VLSI neural networks"
IEEE Micro, pp. 64-74, Dec 1989

[22] Murray, A.F. et al.
"Innovations in pulse stream VLSI: Arithmetic and communications"
Micro-electronics for neural networks, Proc. 1st Int. Work. Dortmund,
pp. 8-15, Jun 25-26 1990

[23] Murray, A.F.
"Analog VLSI and Multi-Layer Perceptrons - Accuracy, Noise and On-Chip learning"
Micro-electronics for neural networks, Proc. 2nd Int. Work. Munich,
pp. 27-34, Oct 16-18, 1991

[24] Nikodem, M. et al.
"Switched-capacitor (SC) simulation of Hopfield-type neural networks"
Int. Symp. on Circuits and Systems, New Orleans, Louisiana,
Vol. 1, pp. 507-509, May 1-3, 1990

[25] Peiris, V. et al.
"Implementation of a Kohonen Map with learning capabilities"
Int. Symp. on Circuits and Systems,
pp. 1501-1504, May, 1991

[26] Reed, R.D. & Geiger, R.L.
"A multiple-input OTA circuit for neural networks"
IEEE Trans. on Circuits and Systems,
Vol. 36, pp. 767-769, May 1989

[27] Reyneri, L.M. & Sartori, M.
"A neural vector matrix multiplier using pulse width modulation techniques"
Micro-electronics for neural networks, Proc. 2nd Int. Work. Munich,
pp. 269-272, Oct 16-18, 1991

[28] Schoemake P.A. et al.
"Back propagation learning with trinary quantization of weight updates"
Neural Networks, Vol. 4, no. 2, pp. 231-241, 1991

[29] Schwartz, D.B. & Samalam, V.K.
"Learning, function approximation and analog VLSI"
Int. Symp. on Circuits and Systems, New Orleans, Louisiana,
Vol. 3, pp. 2441-2445, May 1-3, 1990

[30] Schwartz, D.B. et al.
"A programmable analog neural network chip"
Artifical neural networks: Electronic implementations
IEEE Computer Society Neural Networks technology series
pp. 43-49, 1990

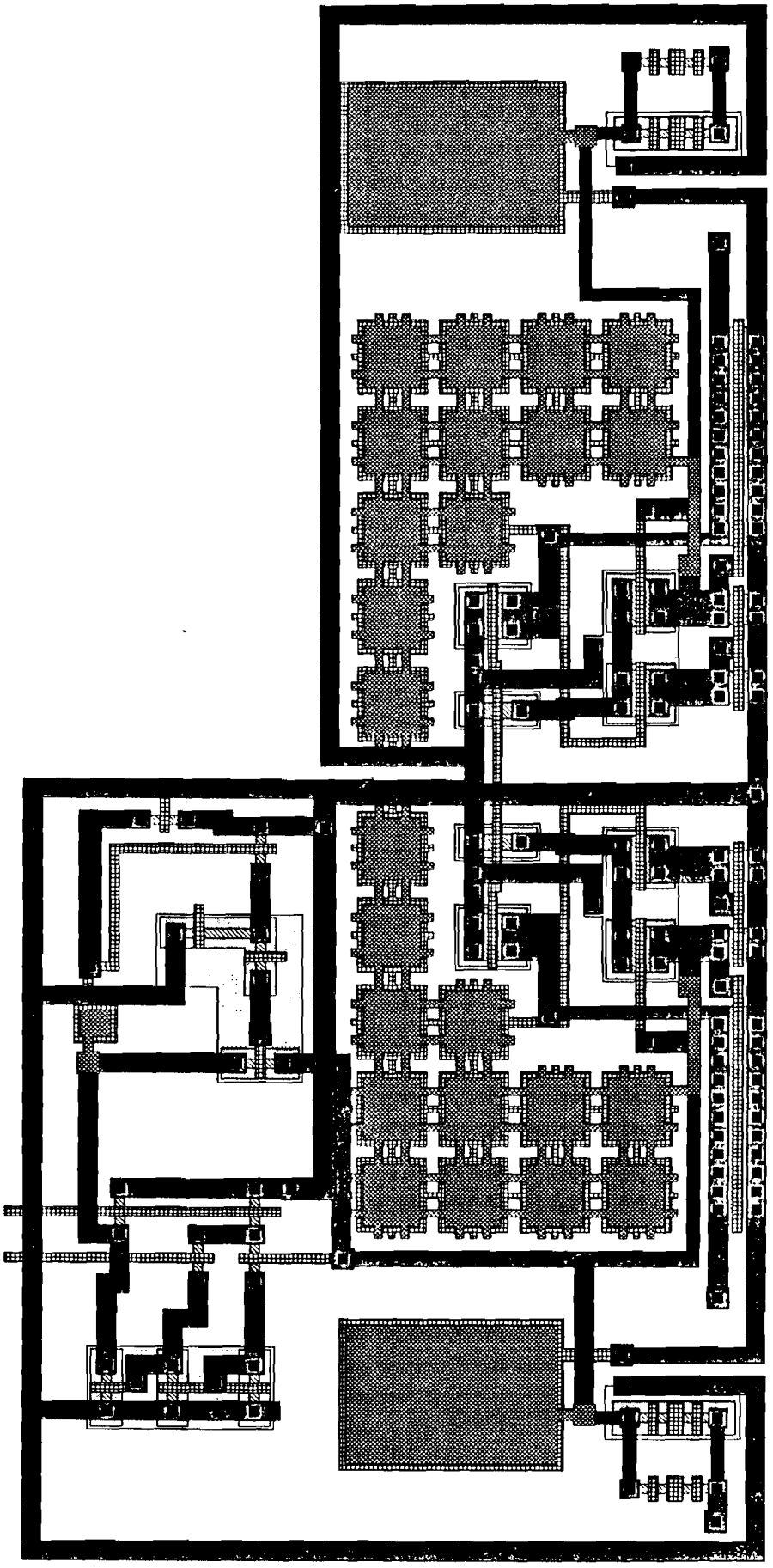
[31] Schwartz D.B. et al.
"A programmable analog neural network chip"
IEEE Journal of Solid State Circuits,
Vol. 24, No. 2, pp. 313-319, April 1989

[32] Teulings, P.M.W.
"Analog electronic implementation of Multi Layer Neural Networks including Back Propagation Training Algorithm"
Masters Thesis, Eindhoven, Aug 1991

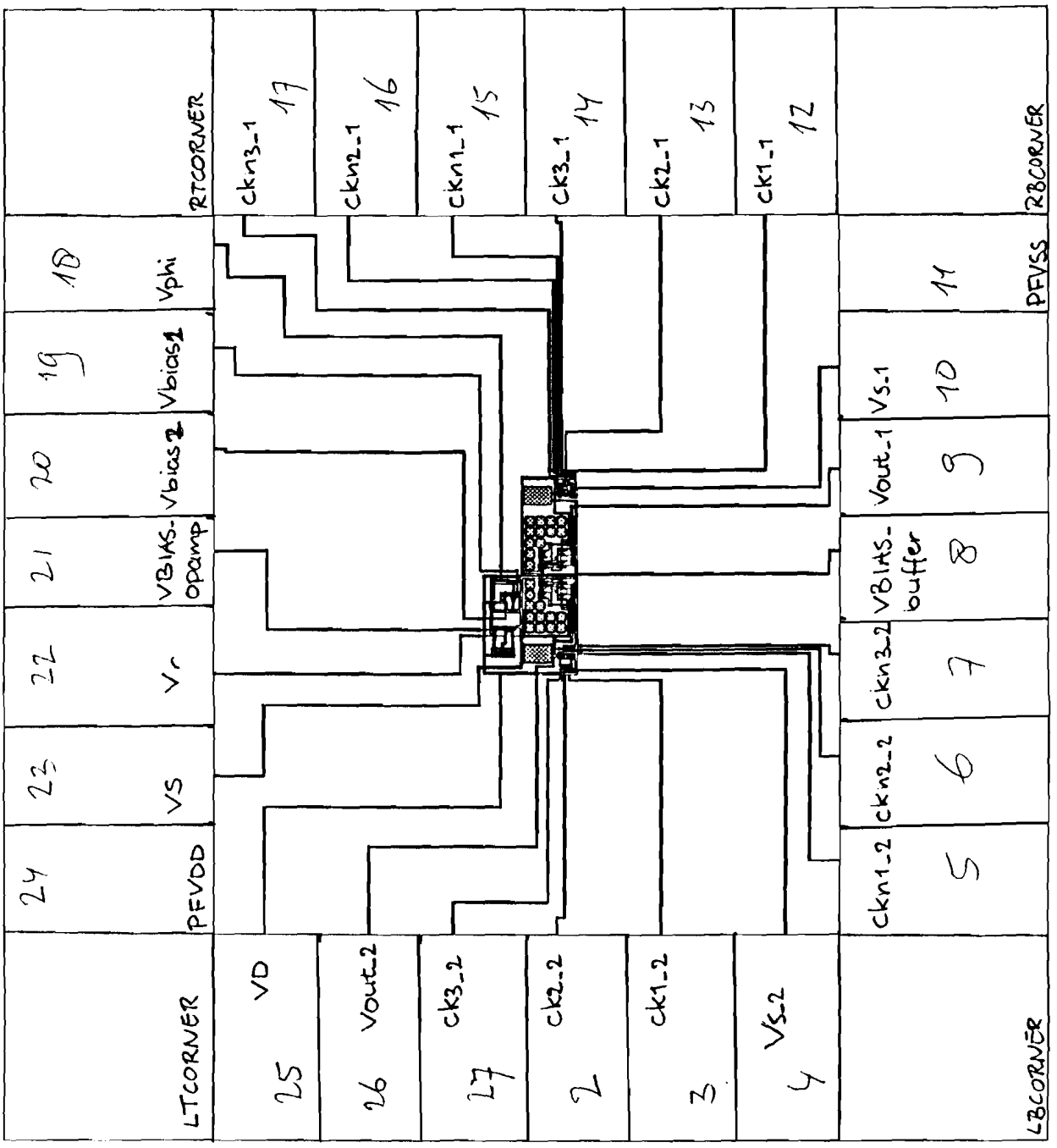
[33] Tomberg, J. et al.
"VLSI Implementation of pulse density modulated neural network structure"
Int. Symp. on Circuits and Systems, Portland,
Vol. 3, pp. 2104-2107, May 8-11, 1989

[34] Vittoz, E. et al.
"Analog storage of adjustable weights"
Micro-electronics for neural networks, Proc. 1st Int. Work. Dortmund,
pp. 69-79, Jun 25-26 1990

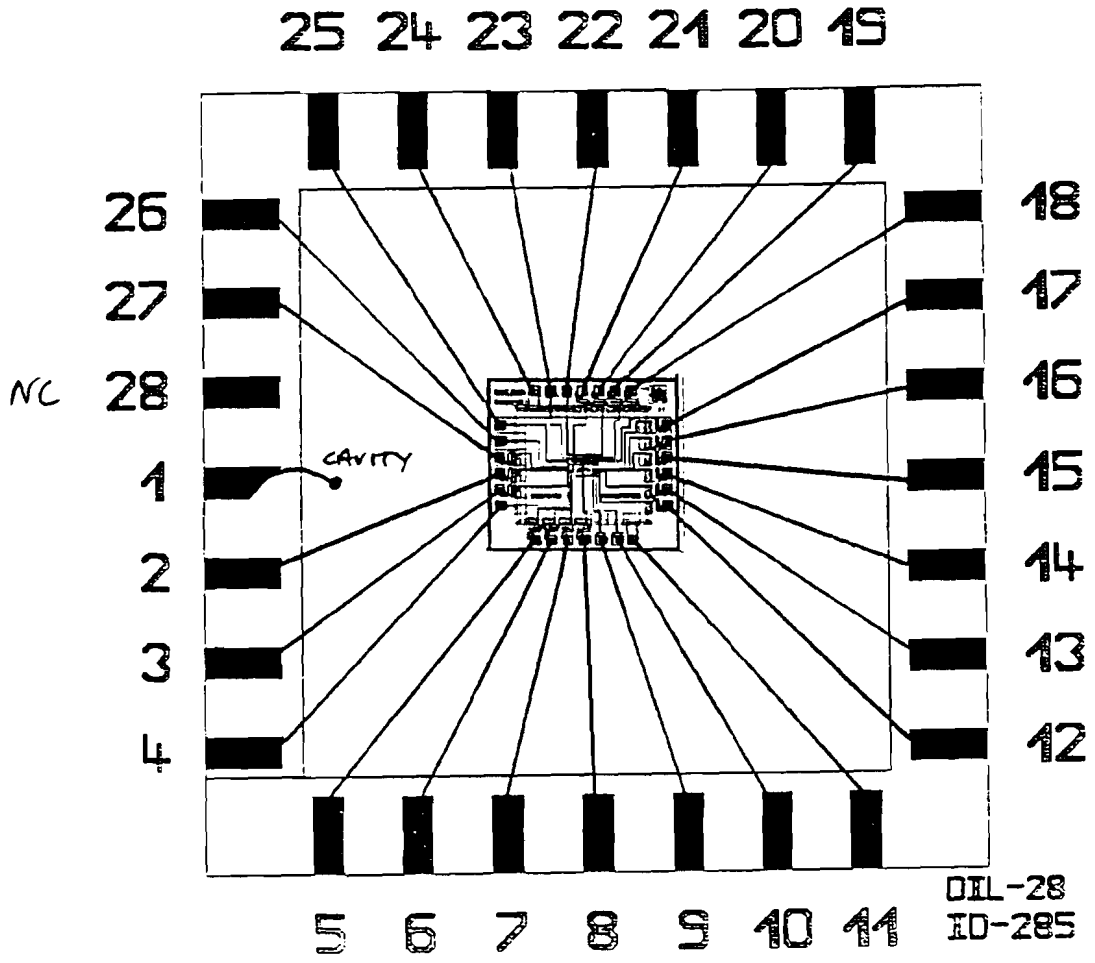
Appendix 1: Layout of update and update/refresh circuit



Appendix 2: Layout with pad configuration and pin numbers

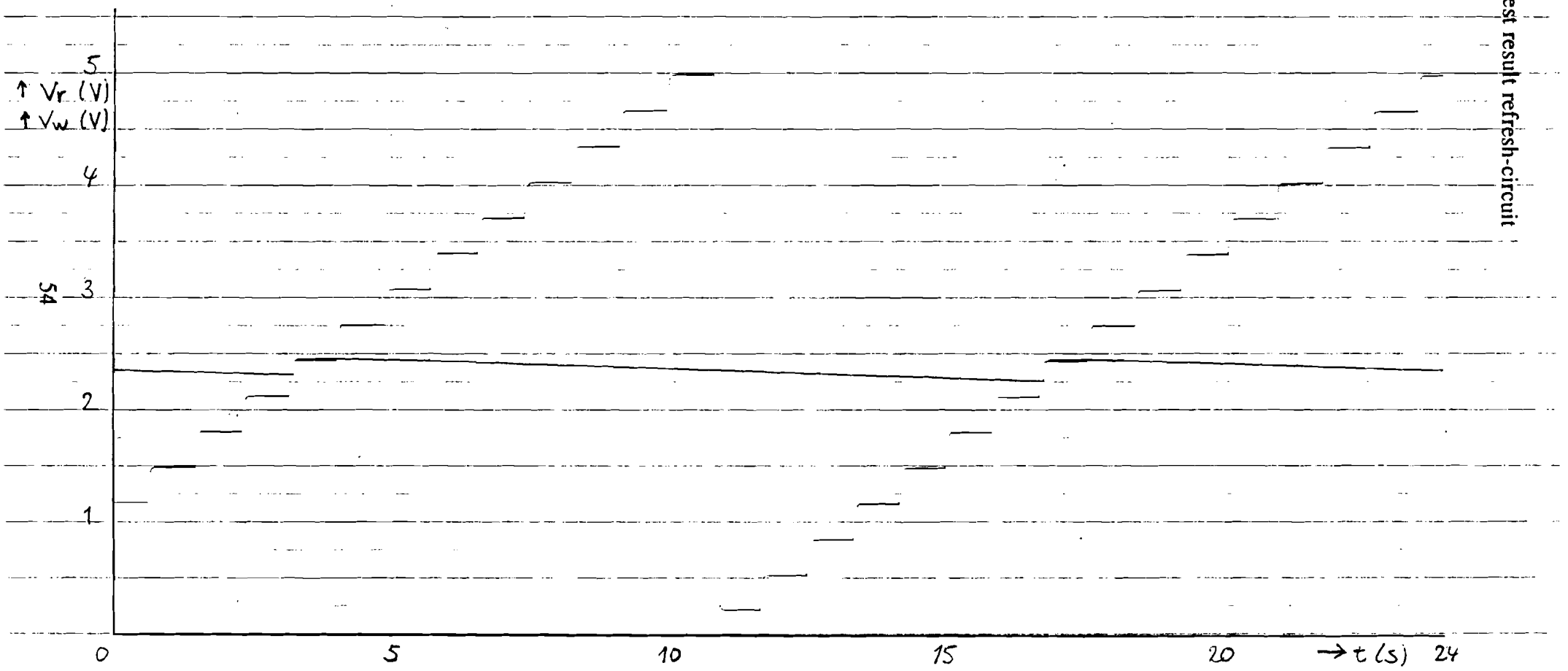


DIL-28



Remarks: Not connected: pin 28

Project: MPW-E 11-91	Circuit	EINDH-03	Scale = 10
Size 2549 x 2366	Die attach	CONDUCTIVE	
Pages : 1	Glue	:	imec
Number : 20	Lid	: TAPED (bodyline)	


5
↑ Vr (V)
↑ Vw (V)
4
3
2
1
0
5
10
15
20
→ t (s)
24