TKK Dissertations 60
Espoo 2007

A VLSI ARRAY PROCESSOR ARCHITECTURE FOR EMULATING RESISTIVE NETWORK FILTERING

Doctoral Dissertation

Asko Kananen

Helsinki University of Technology
Department of Electrical and Communications Engineering
Electronic Circuit Design Laboratory

TKK Dissertations 60
Espoo 2007

A VLSI ARRAY PROCESSOR ARCHITECTURE FOR EMULATING RESISTIVE NETWORK FILTERING

Doctoral Dissertation

Asko Kananen

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Department of Electrical and Communications Engineering for public examination and debate in Auditorium S4 at Helsinki University of Technology (Espoo, Finland) on the 2nd of March, 2007, at 12 noon.

Helsinki University of Technology
Department of Electrical and Communications Engineering
Electronic Circuit Design Laboratory

Teknillinen korkeakoulu
Sähkö- ja tietoliikennetekniikan osasto
Piiritekniikan laboratorio

Distribution:
Helsinki University of Technology
Department of Electrical and Communications Engineering
Electronic Circuit Design Laboratory
P.O. Box 3000
FI - 02015 TKK
FINLAND
URL: http://www.ecdl.tkk.fi/
Tel. +358-9-451 2271
Fax +358-9-451 2269
E-mail: asko.kananen@prh.fi

ISBN 978-951-22-8622-5
ISBN 978-951-22-8623-2 (PDF)
ISSN 1795-2239
ISSN 1795-4584 (PDF)
URL: http://lib.tkk.fi/Diss/2007/isbn9789512286232/

TKK-DISS-2264

Otamedia Oy
Espoo 2007

HELSINKI UNIVERSITY OF TECHNOLOGY P.O. BOX 1000, FI-02015 TKK http://www.tkk.fi	ABSTRACT OF DOCTORAL DISSERTATION

Author Asko Kananen

Name of the dissertation

A VLSI Array Processor Architecture for Emulating Resistive Network Filtering

Date of manuscript 29.1.2007	Date of the dissertation 2.3.2007
☑ Monograph	☐ Article dissertation (summary + original articles)

Department	Department of Electrical and Communications Engineering
Laboratory	Electronic Circuit Design Laboratory
Field of research	Microelectronics
Opponent(s)	Dr. Ricardo Carmona Galán
Supervisor	Prof. Kari Halonen
(Instructor)	Prof. Ari Paasio

Abstract

This thesis deals with silicon implementations of an all-transistor analogue parallel processor that emulates the functionality of a resistive network. The problems related to VLSI -implementations of parallel processors are the main concern of this thesis. These problems are first discussed and then to overcome these problems, a new system design is introduced, namely Reduced Cell-row System (RCS).

The work started from a resistive network -type spatial filter that was part of a video image compression algorithm. The functionality of this algorithm, as well as the filter, was described in Cellular Neural/Nonlinear Network (CNN) notations and they will be used throughout this thesis in describing the filters and processing operations. In addition to the resistive network array processor, a gradient calculation block was included on the chips to fulfil the original algorithm requirements.

Two different array processors were manufactured and measured. The processors had different objectives for their implementation: in the first implementation, the objective was to test the developed Reduced Cell-row System, while in the second implementation the goal was to obtain information on the large-scale implementation of such an array. During the research, a method to include some level of programmability in this type of filters was also developed. For the possible future implementation of such a system, system-level simulations were performed to locate the critical parts that have the most effect on the accuracy of the network.

Keywords analogue parallel processor, VLSI, resistive network

ISBN (printed)	978-951-22-8622-5	ISSN (printed)	1795-2239
ISBN (pdf)	978-951-22-8623-2	ISSN (pdf)	1795-4584
ISBN (others)		Number of pages	125

Publisher Helsinki University of Technology, Electronic Circuit Design Laboratory

Print distribution Helsinki University of Technology, Electronic Circuit Design Laboratory

☑ The dissertation can be read at http://lib.tkk.fi/Diss/ 2007/isbn9789512286232/

TEKNILLINEN KORKEAKOULU PL 1000, 02015 TKK http://www.tkk.fi	VÄITÖSKIRJAN TIIVISTELMÄ

Tekijä Asko Kananen

Väitöskirjan nimi

A VLSI Array Processor Architecture for Emulating Resistive Network Filtering

Käsikirjoituksen jättämispäivämäärä 29.1.2007	Väitöstilaisuuden ajankohta 2.3.2007
☑ Monografia	☐ Yhdistelmäväitöskirja (yhteenveto + erillisartikkelit)

Osasto	Sähkö- ja tietoliikennetekniikan osasto
Laboratorio	Piiritekniikan laboratorio
Tutkimusala	Mikroelektroniikka
Vastaväittäjä(t)	Dr. Ricardo Carmona Galán
Työn valvoja	Prof. Kari Halonen
(Työn ohjaaja)	Prof Ari Paasio

Tiivistelmä

Tämä työ käsittelee vastusverkkojen toiminnallisuuden toteuttamista analogisena rinnakkaisprosessoritoteutuksena VLSI-piirillä käyttäen ainoastaan MOS-transistoreja. Työssä on ensin käsitelty rinnakkaisprosessorien toteutukseen liittyviä ongelmia, minkä jälkeen esitellään uusi toteutusmenetelmä ongelmien ratkaisuksi. Menetelmää kutsutaan nimellä "Reduced Cell-row System" (RCS).

Työn lähtökohta oli vastusverkko -tyyppinen spatiaalisuodatin, joka oli esitetty osana videopakkausalgoritmia. Algoritmin toiminnallisuus, kuten myös suodattimen, oli kuvattu alkuperäisessä algoritmissa epälineaaristen soluverkkojen "Cellular Neural/Nonlinear Network" (CNN) merkintöjä käyttäen ja näitä merkintöjä tullaan käyttämään koko kirjan ajan kuvattaessa suodattimien toiminnallisuutta. Vastusverkkototeutuksen lisäksi myös gradientinlaskentalohko on lisätty piireihin alkuperäisen algoritmin toteuttamiseksi.

Kaksi tällaista rinnakkaisprosessoriverkkoa suunniteltiin, valmistutettiin ja mitattiin. Näiden kahden toteutuksen tavoitteet olivat erilaiset: ensimmäinen toteutettiin, jotta voitaisiin todentaa kehitetyn verkon rivien lukumäärää vähentävän menetelmän (RCS) toimivuus, kun taas toisen toteutuksen tavoitteena oli tutkia laajamittaisen toteutuksen ongelmia. Työn aikana kehitettiin myös menetelmä, miten prosessoriverkkoon voidaan lisätä säädettävyyttä ja tällaisen prosessoriverkon toteutukselle suoritettiin simuloinnit, jotta voitaisiin selvittää toteuksen kannalta kriittiset kohdat.

Asiasanat analoginen rinnakkaisprosessori, VLSI, vastusverkko

ISBN (painettu)	978-951-22-8622-5	ISSN (painettu)	1795-2239
ISBN (pdf)	978-951-22-8623-2	ISSN (pdf)	1795-4584
ISBN (muut)		Sivumäärä	125

Julkaisija Teknillinen korkeakoulu, Piiritekniikan laboratorio

Painetun väitöskirjan jakelu Teknillinen korkeakoulu, Piiritekniikan laboratorio

☑ Luettavissa verkossa osoitteessa http://lib.tkk.fi/Diss/ 2007/isbn9789512286232/

Preface

The research for this thesis was carried out in the Electronic Circuit Design Laboratory (ECDL) of Helsinki University of Technology during the years 1999-2006. The thesis was funded by the Academy of Finland (projects “Integrated Parallel Processors for Future Multimedia”, “Medical Imaging and Communication Systems” and “Integrated Parallel Processors for Future Data Processing and Analyzing Systems”). The work was also supported by the Research Foundation of Helsinki University of Technology, the Foundation of Electronics Engineers and Nokia Foundation.

After almost ten years in the laboratory, I would like to thank the whole staff at ECDL for the relaxed working atmosphere, in which the ten years did not feel long at all. All the answers I got to my questions on anything work related have helped to get to this point, and all the strictly non-work related coffee table discussions have been equally as important. I would also like to thank professors Veikko Porra and Kari Halonen for their guidance and support during this work.

I express my gratitude to Professor Ari Paasio for the close guidance in the early stages of this work as well as the time slots he was able to use for this work after moving to Turku. Discussions with Dr. Mika Laiho have been invaluable in finishing the thesis and I would like to thank Mika for that. The work as a part of the “CNN-team” has been fun all along and all the members of the team, namely Lauri Koskinen, Mikko Talonen and Jacek Flak, in addition to Ari and Mika, are all responsible for that.

Luckily for my sanity, my world has not been solely spinning around this thesis during the last ten years. Therefore, I would like to thank my friends for all the things and moments I have been able to share with you. Special thanks go to my band-mates Marjo & Jussi of Short Cuts and Ville & Petteri of our special Friday-night band.

Finally, I would like to thank my parents and my wife Iitu for all their support and, especially, for not letting me to give up on this work.

Helsinki, January 2007
Asko Kananen

This page is intentionally left blank.

Contents

Symbols and abbreviations

$2NEIGH$	feedback connections to neighbouring cells
$\hat{u}$	positive range CNN input value
$\hat{x}$	positive range CNN state value
$\hat{z}$	positive range CNN constant bias
λ	the resistor ratio R_2/R_1 in a LRN
λ_0	channel length modulation parameter
μ_0	surface mobility of the channel
σ	Variance
ε_0	permittivity of the oxide
$A, A(i,j;k,l)$	CNN interaction coefficients from the outputs of the neighbourhood to cell $C_{i,j}$, feedback coefficients
$a[m,n]$	value of a pixel located in coordinates m,n
$A_{00,new}$	new value for central term of the A-template
A_{00}	central term of the A-template
a_{ana}	value of the pixel represented in analog domain
a_{dig}	value of the pixel represented in digital domain
$B, B(i,j;k,l)$	CNN interaction coefficients from the inputs of the neighbourhood to cell $C_{i,j}$, feed-forward coefficients
$b[m,n]$	output value of a pixel after processing
C	,

$C_{i,j}$	an array processor cell located in coordinates i, j
C_{MSB}, C_{LSB}	AD-conversion control signals
$CELL\_OUT$	cell output node
$CURR_{IN}$	input node of the A-template realisation
$D1, D2$	delay elements
$d_{m,n}$	LRN node input value
DR_a	Analogue dynamic range of the pixel value a_{ana}
E_{full}	energy consumption of the full size network
$E_{reduced}$	energy consumption of the RCS-network
G	connection weight
G, G_1, G_2	LRN Conductances
$G_{1-D}(x)$	One-dimensional Gaussian function
$G_{2-D}(x,y)$	Two-dimensional Gaussian function
i, j, k, l	index denoting cell placement
$I_e(n)$	input current to node n
I_{dyn}	the dynamic range of the low-pass network
I_{in}	Input current
i_{max}	maximum number of cells
I_{tr_meas}	the threshold current used in the gradient block measurements
IN_CTRL	signal controlling writing in to the low-pass network
K	connection weight
k	number of parts the image has to be divided into
L	Length of a CMOS transistor
L_{eff}	effective channel length
$LINE123$	signal controlling output line selection for the 16th row in the low-pass network

M	the number of columns
m	coordinate value $\{m = 0, 1, 2, ..M-1\}$
m_{full}	number of pixels in full-size network in horizontal direction
m_{full}	number of pixels in full-size network in vertical direction
$M_{i,j}$	Convolution mask
$m_{reduced}$	maximum width of the RCS when number of cells is limited
MEM_SW, $\overline{MEM\_SW}$	signal controlling writing in to the low-pass network
N	the number of rows
n	coordinate value $\{n = 0, 1, 2, ..N-1\}$
n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_{x-1}, n_x	nodes in resistive network chain
$N_r(i, j)$	neighbourhood of the CNN-cell $C_{i,j}$
n_{in}	input node of the low-pass cell
$NEIGH\_CTRL$	signal controlling the neighbourhood of the first and the last low-pass cell row
OUT_CTRL	signal controlling writing out to the low-pass network
P	Neighbourhood of a pixel
P_{NW}	power consumption of a single cell of the network during processing
r	aspect ratio
$r(n_m)$, $r(n_{m+1})$, r_{tot}	multiplying term in when calculating resistance in resistive network chain
R_1	vertical resistor in a LRN network
R_2	horizontal resistor in a LRN network
R_{in}	CNN-cell state resistor

R_{nl}	nonlinear resistor
$READ\_ROW$	a shift register control signal
$rn_{\lambda=1}$	resistive network kernel for $\lambda = 1$
S_i	sphere of interaction, defines the maximum distance between two connected cells
t	common settling time
t_{AD}	time the AD-converters require to reach their final output
t_{DA}	unit settling time of the cell input and DA-converters
T_{full}	processing time of the full-size network
t_{NW}	unit settling time of the network
t_{ox}	thickness of the oxide
$T_{reduced}$	processing time of the reduced network
tr_{255}	the gradient block threshold value used in simulations of the functionality of the gradient-block
$u_{i,j}$	CNN-cell input value
V	Voltage
V_T	threshold voltage
V_{DS}	Drain-to-source voltage of a CMOS transistor
V_{GS}	Gate-to-source voltage of a CMOS transistor
$V_{m,n}$	LRN node voltage
$VLSI$	Very Large Scale Integrated circuit
W	Width of a CMOS transistor
W_{eff}	effective channel width
$WRITE$	a shift register control signal
$WRITE\_IN$	a shift register control signal
$WRITE\_ROW$, $\overline{WRITE\_ROW}$	a shift register control signal

X_4, X_5, X_F, SW, SEL	switches of realisation of the variable-λ cell
$x_{i,j}$	CNN-cell state value
$y_{i,j}$	CNN-cell output value
z, Z	constant bias in the CNN state equation
1-D	One-dimensional
2-D	Two-dimensional
AC	alternating current or voltage
AD	Analogue-to-Digital
AP	Array Processor
ASIC	Application Specific Integrated Circuit
B/W	black-and-white
CCCS	Current Controlled Current Source
CDT	Code Density Test
CIF	Common Intermediate Format
CLK	Clock signal
CNN	Cellular Neural/Nonlinear Network
CNN-UM	CNN Universal Machine
CurMeter	current meter
DA	Digital-to-Analogue
DC	constant current or voltage
DigiCtrl	Digital Control
DIM	Digital Image Memory
DNL	Differential nonlinearity
DoG	Difference of Gaussians
ECDL	Electronic Circuit Design Laboratory
ENOB	Effective Number of Bits

FFT	Fast Fourier Transform
GAPU	Global Analogic Programming Unit
HDTV	High Definition TV
High-Z	high impedance
I/O	Input/Output
IC	Integrated Circuit
INL	Integral nonlinearity
LAM	Local Analog Memory
LLM	Local Logic Memory
LLU	Local Logic Unit
LoG	Laplacian of Gaussian
LRN	Linear Resistive Network
LSB	least significant bits in a binary word
MIRROR	a variable-λ block
MSB	most significant bits in a binary word
N/A	Not Available
NMOS	n-channel MOSFET
PCB	printed circuit boards
Pr	protractor
QCIF	Quarter Common Intermediate Format
QVGA	Quarter Video Graphics Array
RAM	Random Access Memory
RCS	Reduced Cell-row System
rms	root mean square
ROI	Region of influence
SAR	Successive Approximation Register

SFDR	Spurious-Free Dynamic Range
SigGen	AC voltage signal source
SNDR	signal-to-noise-and-distortion ratio
SRAM	Static Random Access Memory
SVGA	Super Video Graphics Array
UI-converter	Voltage-to-current converter
VGA	Video Graphics Array
XVGA	eXtended Video Graphics Array

This page is intentionally left blank.

Chapter 1

Introduction

1.1 Motivation

Even with the ever-increasing speed of digital processors there are areas where new and innovative processor structures are needed because of the stringent calculation requirements along with the limited availability of power. One such large field is image processing, where it is possible to have a need for real-time processing of image data with a hand-held device, for instance in video compression. In image processing tasks, many times the early-stage image processing and image analysis require most of the processing power. For a serial type digital processor, these tasks are quite difficult to handle because of the parallel nature of the data. Also in image processing, in many cases, the data is inherently analogue and is transformed to digital domain mainly for data processing or storage needs. Therefore, parallel analogue processor structures can be considered ideal for this type of processing.

Neural hardware has been developed over the last two decades to implement neural systems that are based on learning neural systems (e.g. [1]). In a PhD-thesis [2], published in 1995, over 40 bio-inspired neural hardware projects were listed, starting from a conventional PC with an acceleration board (e.g. IBM's Network Emulation Processor NEP [3]), to Application Specific Integrated Circuits (ASIC's) designed purely for neural computation, (e.g.[4]). Most of the presented projects were learning neural systems, but for the image processing algorithms, the learning neural system is not required because the input-output mapping is normally known a priori. However, some of the mentioned projects were based on the ideas presented in [5], where bio-inspired chips were proposed for various processing tasks. In this approach, the chips emulate the processing of a biological neural system in their calculation schemes. In [6], a silicon retina model was introduced and in [7], in a similar fashion, a silicon model for auditory localisation was proposed.

In article [6], a resistive network was used in averaging the photoreceptor output in the retina model. This work inspired several researchers and, as a result, many proof-of-concept integrated circuits (IC's) have been manufactured, for instance [8] and [9]. However, resistive networks can be used for spatial low-pass filtering on any image processing system that requires such an operation; therefore a resistive network IC that could be used in accurate filtering purposes would be an attractive device.

In article [10] by Chua et al. an analogue parallel processing paradigm, namely Cellular Neural Network (CNN), was introduced. The article suggested a parallel processor structure that could be programmed using two template sets. The tempting feature of this structure was the local connectivity of the individual synapses that suggested a feasible realisation on silicon. However, the reality has turned out to be more complicated than expected. Even with the latest chip [11], the functionality and accuracy of the chips are limited according to the measurements and the power consumption and used silicon area are still considerably large. In spite of that, the paradigm yields a powerful tool with which to analyse and simulate parallel structures and that has been used in several applications (e.g. [12], [13], [14]) as well as in this work.

As the starting point for the work, an article by Stoffels [14] was chosen. That was because the article showed an algorithm for video compression that was presented according to the theory in [10]. The algorithm included a grey-scale pre-processing part, grey-scale to black-and-white (B/W) image analysis part and several B/W processing steps. Because the implementation of the QCIF-size B/W chip had already shown the realisation of large scale bipolar processor to be feasible [15], the goal was set in the implementation of the grey-scale part. The grey-scale part performs an *Edge Enhancing Low-pass Faltering* that included a resistive-network-type low-pass part and a gradient calculation part. Because the algorithm was intended for video image compression, the accuracy requirements of the processor were set by the image processing standards, which are quite stringent for an analogue processing system. Here, as the approach was chosen to separate the different processing parts and to optimise each separately [16], this led to an implementation where the processor grid size also was optimised, depending on the calculation task.

1.2 Research Contribution

The work is concentrated on silicon implementations of the proposed system where the research results are tested in real silicon implementations. This included implementation of digital image memory (SRAM), column analogue-to-digital (AD)-- and digital-to-analogue (DA)- converters and the actual processor blocks, namely the low-pass and gradient calculation blocks.

The system-level approach was developed by the author along with Professor Ari

Paasio and presented in [17]. The idea of minimising the grid size was taken as a goal from the beginning. The actual method to perform the low-pass filtering in a Reduced Cell-row System was developed by the author. The gradient calculation cell was developed by Professor Paasio.

Two separate chips were implemented to test the method. In the first chip, the low-pass network cell was developed from the implementation presented in article [18]. Later it turned out that in [19] a rather similar realisation was also used. For the second version of the low-pass filter, the cell was modified by the author to suit the system better. These modifications simplified the peripheral circuitry and corrected certain systematic error sources. The gradient cell reminded the same in the both versions of the chip, only the technology changed. The AD- and DA-converters in the first chip were implemented by Professor Ari Paasio and in second chip the AD-converter was changed to a converter designed by Dr. Mika Laiho [20]. The digital parts of the system were designed by the author in both chips.

All the measurements of both chips were carried out by the author as was the research on programmable resistive networks.

1.3 Organisation of the Thesis

The thesis is organised into seven chapters. Following the Introduction, Chapter 2 shows the definitions used and shows some widely used image processing algorithms and reviews the theories of Cellular Neural/Nonlinear Networks (CNN)[10] and Linear Resistive Networks (LRN). Chapter 2 also shows how LNR's can be considered as a special case of CNN. Also in Chapter 2, a modification by Shi in [21] to the analysis of resistive networks by applying CNN theory is introduced.

Chapter 3 concentrates on designing a resistive network systems, starting with the analysis of the problems related in the design of large-scale Array Processors. After this, some previously reported resistive networks are presented and briefly analysed. Finally, the Reduced Cell-row System (RCS) developed in this thesis is introduced. First, the basis of the reduction of cell-rows is discussed, then the advantages and disadvantages of this.

In Chapter 4, the realisation of the proposed system is shown. First, the basic transistor level building block, namely the current mirror, is presented, and its relevant nonidealities are discussed. The presentation of the circuitry is divided into two sections: the first shows the analogue parts of the system and the second the digital parts and their implementation. Finally, the layout design is also discussed.

In Chapter 5, the measurement results of the implemented chips are presented. The results of the first version are briefly shown and the problems in the design are discussed. After this, the measurement results of the second chip are comprehensively

shown and further improvements are considered.

Finally, in Chapter 6, the future work of designing a programmable resistive network is described. A high-level system simulation method is presented to investigate the effect of the mismatch in the transistors on the system level performance.

Chapter 2

Array Processors: Definitions and Examples

The purpose of this chapter is to give a theoretical background of array processors and to show the definitions that are used throughout the thesis. In addition to that some application examples are given also for motivation.

At the beginning, the image and array processing terms and definitions are given. One important consideration when implementing array processors is how the processing task can be divided if the array is not sufficiently large to process it as one entity. The methods of dividing a processing task are discussed briefly in this chapter before the introduction of the proposed method in the following section.

Then, as an introduction to actual array processing, some generally used linear smoothing operations are introduced by showing the principles of Mean and Gaussian filtering. As it will be shown later, these operations are close to the operations that are the object of this work. For their analysis, the convolution kernels are calculated and are considered from the implementation point of view also. Also some examples, where the properties of this type of filters can be used in image processing is shown.

The main objective of this work, namely Linear Resistive Networks (LRN), are introduced here after some general properties are shown first. After their functionality is analysed in general, a convolution kernel is calculated also for one special case in order to compare its functionality to Gaussian filters. Because the LRN's have a similar type of transfer function as the Gaussian smoothing filter [9], their use in the applications mentioned before is discussed.

Cellular Nonlinear/Neural Networks (CNN) [10] will be presented next. First, the original theory from the [10] is introduced. After this, some previously presented modifications and additions to the theory that are useful in this work, are shown. The sim-

ilarities between the LRN and CNN topologies are discussed and it is shown that the state function of the resistive network can be expressed in CNN notations. The uniformity of these two topologies was shown originally in [21] by Shi. Here the calculations are shortly repeated as the basis of the further analysis. However, that presentation is not directly implementable on silicon, so in order to simplify the implementation some modifications are introduced to the CNN presentation of the resistive networks. These modifications are based on the properties of the linear resistive networks. Using the results, it is quite straightforward to come up with a transistor-level realisation, as will be shown in Chapter 4.

2.1 Definitions and Properties Related to Array Processors

There are several different types of approaches to increasing the parallelism in processing, starting from executing parallel operations in general purpose digital processors [22] to chip multiprocessing where the processing task is divided between several microprocessors, used nowadays in household PC's. Here, a special case is considered where the definition of an array processor is limited to single chip processors in which the processing core consists of identical processing elements. These elements perform processing in the analogue domain by interacting with each other.

In this section, the basic notations and definitions related to images and array processors are given. First, the notations that were used in describing images that are used as input and output are shown. After that, the definitions used to describe an array processor functionality and connections are described and different types of array processor operations are listed. Finally, definitions related to a division of the processing task are introduced.

2.1.1 Image Notations

The two dimensional data, which are used as an input of an array processor (AP), can be considered to be an image, independently of which phenomenon it describes or quantity it measures. An image is formed by a set of pixels that are organised in a rectangular shaped $M \times N$ grid, where M is the number of pixels in the horizontal direction, i.e. columns, and N is the number of rows. This is depicted in Fig. 2.1.

Each pixel has a value $a[m,n]$, where the coordinates m and n vary between $\{m = 0,1,2,..M-1\}$ and $\{n = 0,1,2,..N-1\}$. The value a can be either a continuous analogue value or it can have discrete integer values, presented using, for example an 8-bit digital word. This results in the digital value of a_{dig} can have discrete values in

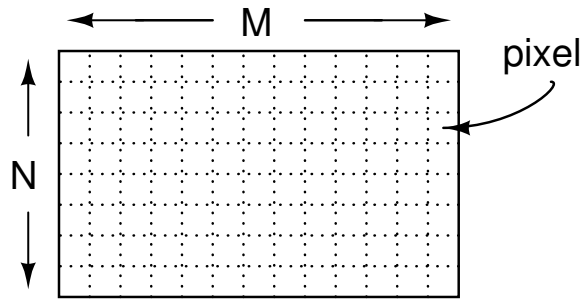


Figure 2.1 Definitions of an image.

the range $a_{dig}[m,n] \in [0,255]$ that linearly represent the sampled analogue value if the dynamic range of the analogue value a_{ana} is DR_a. If a represents the luminance of the pixel, the value can be illustrated as a grey-scale image. This type of representation is widely used in video sequence images. For clarity, some digital values and the grey-scale level that they represent are shown in Figure 2.2. The figure shows that, the larger the value a, the brighter the pixel.

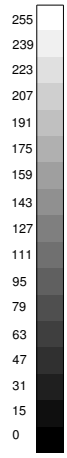


Figure 2.2 Some values of $a[m,n]$ and the grey-scale value they represent.

2.1.2 Array Processor Definitions

An array processor consists of processor units, referred to here as cells, where each represents one pixel of the input image. These processor units are here denoted with $C_{i,j}$ where i and j define its placement. For an $M \times N$ image, $i \in [0, M-1]$ and $j \in [0, N-1]$.

In the following, first the definitions of neighbourhood and connectivity are given for the AP. Then a distinction is made between different types of image processing operations and their suitability for an array processor is discussed. This leads to definitions of convolution-type processing.

For simplicity, in the definitions it is assumed that the shapes of the array processor and the images are rectangular.

2.1.2.1 Neighbourhood and the Connections between the Array Processor Cells

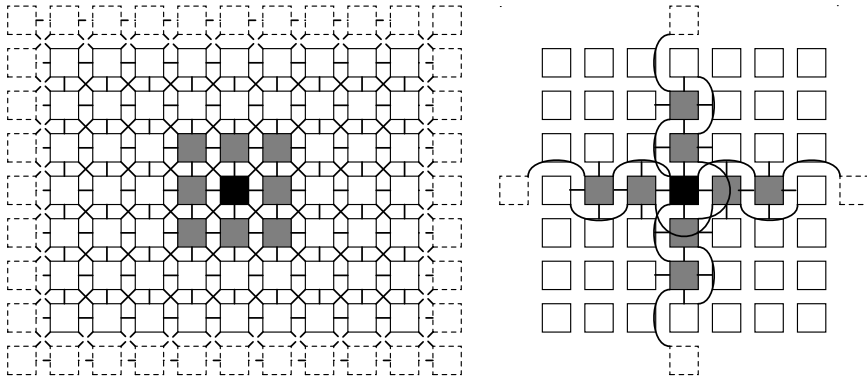
The cells in an array processor usually have a connection to a selection of cells in their neighbourhood. The interaction between the cells is possible using these connections. The maximum distance between two connected cells defines the sphere of interaction S_i. The S_i-neighbourhood of cell $C_{i,j}$ consists of cells $C_{k,l}$, where $k \in [i-S_i, i+S_I]$ and $l \in [j-S_i, j+S_I]$. The connection of the cell $C_{i,j}$ can vary arbitrarily within S_i, but the most common connections are symmetrical around the cell $C_{i,j}$. If the cell $C_{i,j}$ is connected to all the adjacent cells, the network is said to be 8-connected. In the case of a 4-connected network, the connections are to the cells in the same columns and rows as cell $C_{i,j}$.

In Fig. 2.3, two examples are shown of neighbourhood and connections. Cell $C_{i,j}$ is the black cell in the figures and the grey cells are the cells that belong to its neighbourhood. Figure 2.3(a) shows a 2-D network where the cells are first-neighbourhood and 8-connected, i.e. the cell is connected with all the cells surrounding it. In Figure 2.3(b), the array is second-neighbourhood and 4-connected. In the latter, only the connections of the cells that are within the S_i of the cell $C_{i,j}$ are shown.

As the figure suggests, the larger the S_i and neighbourhood, the more complex the silicon implementation becomes, because the number of routings between the cells increases.

2.1.2.2 Different Types of Image Processing Operations

If an image processing operation is considered as a transformation from the input image $a[m,n]$ to the output image $b[m,n]$, where the a and b are the input and output values of a given cell $C_{m,n}$, then the different types of image operations can be divided into three classes according to their complexity [23].



(a) 2-D 8-connected network with 1-neighbourhood

(b) 2-D 4-connected network with 2-neighbourhood

Figure 2.3 Neighbourhood and connections

1. *POINT* operations. In this class, the cell output value $b[m,n]$ in any coordinate is dependent on the cell input value $a[m,n]$ in the same coordinate. The calculation complexity for each pixel is constant. In this case, $S_i = 0$.

2. *LOCAL* operations. In this class, the cell output value $b[m,n]$ is dependent on the input values in certain surroundings of the cell in the same coordinate, sized $P \times P$. The complexity per pixel is P^2 and $S_i = P$.

3. *GLOBAL* operations. In this class, the cell output value $b[m,n]$ is dependent on all the input values. The complexity per pixel is $M \times N$. This case requires the cells to be connected to all the other cells, therefore $S_i = max\{M,N\}$.

If all these cases are considered from the implementation point of view and compared also to digital processor realisation, it can be stated that the *POINT* operations are the easiest to implement but have the least advantage over serial mode digital processors, because of the lack of interaction with the neighbouring cells. *LOCAL* operations are the most attractive operations to be implemented with an array processor because of the large amount of required surrounding information that is inherent to parallel processors, in contrast to the serial mode digital processors. However, as P becomes larger, the complexity of the wiring between the cells increases; *GLOBAL* operations are therefore practically impossible to implement on silicon with reasonably sized arrays.

2.1.2.3 Convolution Operations

The *LOCAL* operation can also be considered as a convolution operation. In convolution processing, the functionality and the dependency on the values of the neighbouring

cells can be presented using a convolution mask (template). This mask also shows the neighbourhood and the connection of the cells. Therefore, for instance for the network shown in Fig. 2.3(a), the convolution mask $M_{i,j}$ can be written as:

$$M_{i,j} = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{bmatrix}, \tag{2.1}$$

where $a_{k,l}$ are the interconnection weights and can be fixed or programmable in a hardware realisation. If $M_{i,j}$ is same for all the cells, the convolution mask is said to be space-invariant and the weights can be controlled globally.

2.2 Division of the Processing Task

The main advantage of parallel network processing would be the capability to process the whole input simultaneously. In the 2-dimensional case, it would require the network to be the same size as the input image. In some cases it is possible to build such a large grid, but in many cases it is not feasible. In these cases, the processing task itself has to be divided into smaller sub-images. Since the functionality of parallel processors is usually based on the interaction of the processing elements, the environment has to be maintained for the image pixels on the borders of the sub-image. The required neighbourhood that needs to be preserved is dependent on the processing task. For the analysis, the following definitions will be used:

Active cells	the cells from which the output is read out,
Overlapping cells	the cells that provide the required neighbourhood to the active cells in a divided network. These cells are similar to the Active cells.
Processing cells	Active and Overlapping cells together
Border cells	the cells surrounding the Processing cells providing required border condition, for instance zero-flux [24].
Region of influence (ROI)	the number of Overlapping cells required to obtain accurate enough result with a divided network, defines the number of Overlapping cells

The most common division is shown in Fig. 2.4, where the image is simply divided into parts and each part is separately processed. The figure shows the Processing cells and the Overlapping cells that are needed to form the correct surrounding for the Active cells.

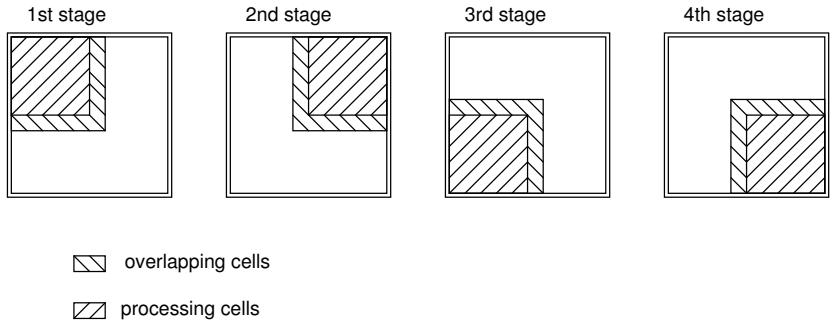


Figure 2.4 A traditional way to divide the processing task.

As it was defined above, the ROI-value of the processing task defines the number of overlapping cells. If we consider an algorithm, where there are several consecutive processing tasks, the ROI value differs from task to task. Because the algorithm requires storage of the previous result, there are two options for the processing. The first is by the method where the data is stored locally and the algorithm is run for the sub-images separately. Or, as proposed in [16], process each step of the algorithm in a separate processor block for the whole image and store the intermediate results to external memory outside the array processors. For the first method, the problem

is that the number of Overlapping cells has to be optimised according to the largest ROI-value of the algorithm steps. Naturally this decreases the number of Active cells and increases the number of sub-images. The implementation of inside-grid memories enlarges the cell size also. When using an outside of the grid memory, the downside is the read-out and write-in operations that have to be made for the intermediate results.

Even if the different tasks are divided into different array processor blocks, the size of one array processor block can yet become intolerably large. Therefore a system was designed where the number of cell rows is decreased drastically [25]. In this system, the number of cell rows is reduced to a fixed number, independently on the size of the input image, while the number of cell columns is the same as the width of the input. This way it is possible to handle the input and output image in a row-by-row manner and the writing in, processing and reading out of the result can be done simultaneously. This speeds up the processing when compared to the traditional way of writing first all the input, then processing and finally reading out the output.

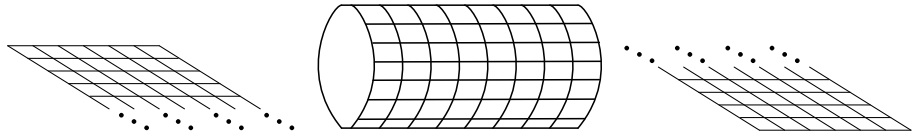
In the following, the basis for this reduction is presented first. The exact procedure of processing is described and the advantages and disadvantages are discussed in Section 3.3.2.

2.2.1 The Reduced Cell-row System (RCS)

In array processor systems, often the write-in and read-out operations are done in a row-by-row manner, for example, [15] because it maintains the parallelism. This was chosen as the approach in this work also. As a result, the Reduced Cell-row System was developed. The RCS processing can be divided into three stages: writing in, processing and reading out. In principle, the input is loaded to the network in a row-by-row manner until the required number of cell-rows is fed to provide the first row the neighbourhood for the correct result. The number of required cell-rows is defined by the ROI-value of the operation. After this, the processing can start in the first row; when it has reached its final value, the result can be read out from it. In this way, the whole image is processed in a row-by-row manner.

The network global connections can be divided into three stages depending on which part of the image is being processed. When processing the first image rows, the network is connected similarly as a full-size network. This is shown in Fig 2.5(a). After this, when the rows in the middle of the image are being processed, the network is connected to form a cylinder, shown in Fig 2.5(b). This is done by connecting the first processor cell-row with the last cell-row. Finally, at the end of the processing, the network is connected again as at the beginning, as shown in Fig. 2.5(c). A somewhat similar processing system was presented in [26] for a one-dimensional case with a fixed circular connection. The system can also be considered as a pipelined system,

which method was used, for instance, in temporal difference imager in [27]. There the photodiode values are stored in two storage elements. To avoid different sampling and holding times for the two elements, which could affect the accuracy of the difference evaluation, the operations are conducted in a pipelined row-by-row manner.



(a) Network connection for the first cell-rows

(b) Network connection for the cell-rows in the middle of the image

(c) Network connection for the first cell-rows

Figure 2.5 The different connection modes of the RCS.

2.3 Image Smoothing Operations

In this section, we will shortly discuss certain linear smoothing filters that can be used in image processing. These types of filters are used in image processing systems for improving the quality of the image or for preparing the image for further processing. Therefore these filters are usually on the lowest level of the image processing algorithm. However, by combining different filters, higher level image analysis algorithms can also be realised, for instance, the Difference of Gaussians (DoG) [28] or *Edge Preserving Image Smoothing* [14] that will be presented later. In the following, two filters, namely Mean and Gaussian, will be presented. The reason for taking these two filters is that their functionalities are close to the filters that were implemented in this work. Therefore they give a good comparison point when the implementability is being considered.

2.3.1 Mean Filtering

A Mean filter, as the name suggests, calculates the mean of the pixel values in a certain neighbourhood P. The calculation can be described by a convolution mask. For instance a case where the neighbourhood is 3×3, the mask $M_{i,j}$ can be given as shown in Equation (2.2).

$$M_{i,j} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.2}$$

In order to achieve a larger neighbourhood from where the mean of values is calculated, the mask has to be enlarged or more iteration rounds, where the result of previous round is used as the new input, have to be performed.

2.3.2 Gaussian Filtering

Gaussian filtering is a popular image smoothing method. The smoothing with a Gaussian kernel resembles defocusing a lens; this type of action is inherent in many biological systems. The usability is based on the fact that the smoothing with Gaussian effectively removes small sharp objects in the image that can be considered to be noise. When enhancing the edges in an image by differentiating, these noisy objects are also enhanced, unless an image smoothing operation is performed. It has been shown in [29] that when smoothing a noisy image, the best results in signal-to-noise ratio are obtained with a Gaussian kernel.

Gaussian filtering is based on Gaussian distribution. For a one dimensional (1-D) case, the Gaussian distribution G_{1-D} can be expressed as in Eq. (2.3)

$$G_{1-D}(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}, \tag{2.3}$$

where σ is the variance of the function.

The Gaussian function 1-D distribution is used as a 'point spread' function by calculating the convolution mask from the distribution. In practice, this is done by solving the impulse response with a certain σ in all coordinates and using the results in the convolution mask. In principle, the Gaussian distribution is non-zero everywhere, resulting in an infinite convolution mask. However, since the mask coefficients become smaller as the distance to the centre increases, the performance can be accurately approximated with a finite convolution mask where the smallest values are set to zero.

An example of a 1-D Gaussian convolution mask when $\sigma = 1$ is shown in Eq. 2.4. The values outside the 3-neighbourhood shown became so small that they could be left out.

$$a_1 = \frac{1}{256} \begin{bmatrix} 1 & 14 & 62 & 102 & 62 & 14 & 1 \end{bmatrix} \tag{2.4}$$

Since the image information is in most cases two-dimensional (2-D), is is also well worth considering a 2-D case of the Gaussian filter. A 2-D symmetrical Gaussian function $G_{2-D}(x,y)$ is of the form Eq. (2.5):

$$G_{2-D}(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{2.5}$$

In a similar fashion to the above, the convolution mask can be calculated using the equation. If, for example, the σ-value used is equal to one, by calculating the impulse response we get Eq. (2.6). Another way of obtaining the mask is to convolve the impulse two times with the 1-D mask in x and y directions. This is possible because the Gaussian function is separable.

$$a_2 = \frac{1}{273} \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 20 & 33 & 20 & 4 \\ 7 & 33 & 55 & 33 & 7 \\ 4 & 20 & 33 & 20 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \tag{2.6}$$

The mask is truncated to contain only the elements in 2-neighbourhood for simplicity. Usually the size of the kernel is limited to a certain size for easier computation and the elements outside it are simply left out. However, the size of the mask grows as the σ becomes larger. This is due to increased smoothing, which leads to a growing number of non-zero elements.

2.4 Using Linear Smoothing Filters

Here some applications are shown where these linear smoothing filters can be used in image processing. First, their use in error correction is shown. After this, a higher level image processing algorithm that preserves the edge information in a low-pass filtering operation is shown.

2.4.1 Correcting random errors

Since the basic operation that the smoothing filters perform is low-pass filtering, they can be used in early-stage error correction of the images. Random noise can be generated in the transmission lines, AD-conversion or capturing of the image. If such a disturbed image is low-pass filtered, the effect of these errors can be significantly reduced. Figure 2.6(a) shows the noiseless original image and 2.6(b) shows the same image with additional random noise.

The three figures 2.7(a)-2.7(c) show the simulation results after filtering the noisy image. In the first figure 2.7(a), the image was filtered with the 3×3 kernel mean filter

(a) Original image

(b) Image with additional random noise

Figure 2.6 The reference and the noisy input image used in the simulations

that was shown in Eq.(2.2). In the second figure, 2.7(b), the used filter was a similar mean filter but the kernel size was 5×5. In the last figure, 2.7(c), the situation after filtering the noisy image with the Gaussian kernel that was given in Eq. (2.6) is shown.

(a) Image filtered with 3×3 convolution kernel.

(b) Image filtered with 5×5 convolution kernel.

(c) Image filtered with Gaussian kernel shown in (2.6)

Figure 2.7 Simulation results with three different smoothing kernels.

2.4.2 Difference of Gaussians and Zero Crossing

A modern way to find edges in the image is to detect the zero-crossings of the Laplacian of the image. A zero-crossing refers here to the situation where adjacent pixels have a different sign as a result of calculating the Laplacian. As the Laplacian operation involves a second derivative, this means a potential enhancement of noise in the image at high spatial frequencies and therefore smoothing operation is desirable before the Laplacian operation. A suitable spatial filter is a Gaussian filter with an appropriate σ-value. Adding a Gaussian filtering before Laplacian operation is called Laplacian of Gaussians (LoG), suggested already in [30]. The Difference of Gaussians (DoG) [28] is in turn a good approximation of the LoG operation. In DoG, two Gaussian kernels with different σ-values are used in filtering the same image; then the resulting images are subtracted and the zero-crossing is detected from resulting image. This is shown in

Fig. 2.8, where figures 2.8(a) and 2.8(b) show the outputs after filtering with different σ-values. The last image, Fig. 2.8(c), contains the result of their subtraction. In the last image, to the resulting 8-bit image, an additional offset of 128 is added to shift the result to visible grey-scale levels.

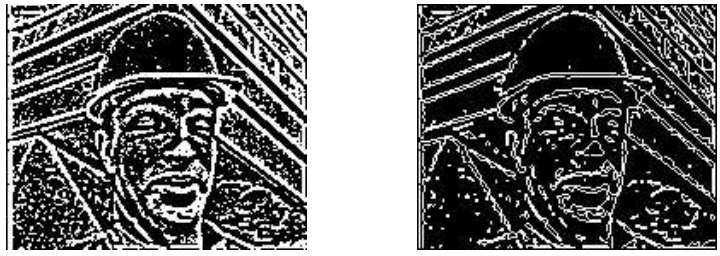
(a) Image filtered with 5×5 Gaussian convolution with $\sigma = 0.9$.

(b) Image filtered with 5×5 Gaussian convolution with $\sigma = 1.6$.

(c) The difference of the filtered images. An offset of 128 is added to each pixel.

Figure 2.8

When locating the edges from Fig. 2.8(c), they are assumed to be in the zero-crossings. The two regions where the pixels have the same sign can be detected by thresholding the subtraction image. That is shown in Figure 2.9(a), where all values greater than zero are marked with a white pixel and values equal to zero or less are marked with a black pixel. For finding the actual zero-crossing pixels, there are several methods proposed, but in this application a simple threshold logic algorithm is used for simplicity. In the method, all the white pixels that have 4, 5 or 6 white neighbours in 8-connected neighbourhood are marked as white and the rest are marked black. The result is shown in Figure 2.9.



(a) The thresholded image.

(b) The image after threshold logic operation.

Figure 2.9

2.5 Linear Resistive Networks (LRN)

Resistive Networks were used as analogue computers in solving complex boundary value problems in electro-magnetics [31] and [32] before the dawn of the integrated transistor. Along with the ever increasing calculation power of digital processors the interest was gradually lost for years until the 80's. Then resistive network-like processing was shown to occur in the early vision in the retina [5]. This resulted researchers becoming interested in resistive networks again, for instance [33], [19]. Here, the principle of resistive network calculation is first presented and then a short analysis is shown, using the properties of resistors and circuit theory.

2.5.1 LRN in principle

A 4-connected resistive network is shown in Fig. 2.10. If resistive network processing is considered in image processing then each node $n_{m,n}$ represents a pixel in the image. Each node has a voltage $V_{m,n}$, which represents the pixel output value $b[m,n]$ after processing and a voltage source $d_{m,n}$ that represents the input value of each pixel.

The connections to the neighbouring nodes through resistors R_2 and a connection to the ground through the vertical resistor R_1 define the transfer function. Linear resistive network performs low-pass filtering to the input and the amount of smoothing is controlled by the resistor ratio R_2/R_1, denoted here with λ.

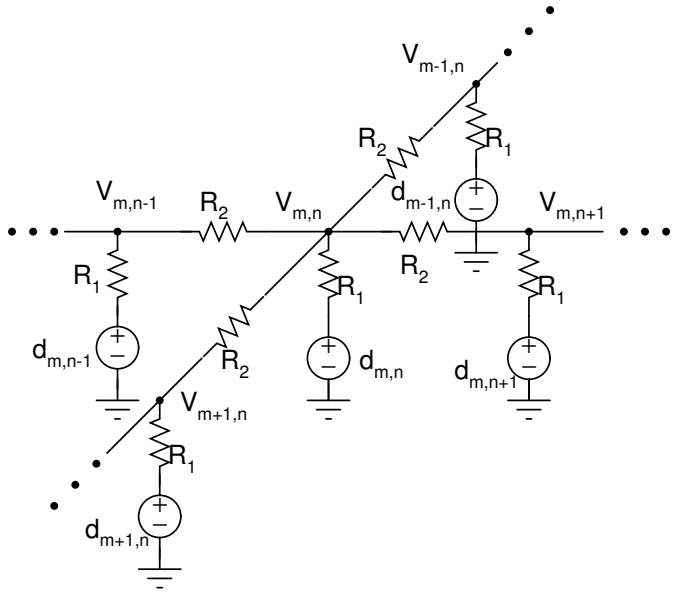


Figure 2.10 A resistive network with input voltage $d_{m,n}$, output voltage $V_{m,n}$ and connections to the neighbouring nodes through resistor R_2 and to the ground through R_1.

2.5.2 Analysis of the LRN's: Calculation of ROI

The resistive networks that are in the scope of this work are linear. This is important, because it leads to the fact that for any given input, there is only one possible output, toward which the circuit settles when the processing starts. This feature can be used in optimising the grid size, which from the beginning was a goal of this work, as it was mentioned in Chapter 1. However, optimisation requires the analysis of the ROI for resistive networks.

In the calculation of the ROI, a 1-dimensional network is used first because of its simplicity. For the analysis, the network is considered to be large yet finite, as shown in Fig. 2.11, and the input d_{in} to the network is fed only to one cell, located almost infinitely far away from the end of the network. The resulting output values at each node n_m, $m \in [0,1,2,..,x]$ are monitored; if the output value is over a threshold value, the cell is considered to be part of ROI. The used input is 255 and the threshold value is 1. It is obvious that these values are taken from the digital world and digital image processing where 8-bit values are commonly used. The circuit shown in Fig.2.11 was used in the calculations.

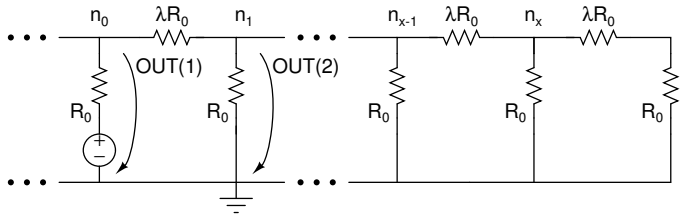


Figure 2.11 A 1-dimensional infinite resistive network.

In the analysis, the total resistance seen by node n_0 is calculated. The calculation starts from the node n_x where the resistance is reduced to one resistor connected between the ground and the node n_x. From there the resistance seen by the node n_{x-1} is calculated.

The circuit shown in Fig.2.12 shows the used calculation procedure. In the figure, the two consecutive nodes are shown as n_m and n_{m+1} and the resistance seen in the node n_m is calculated using the previously calculated resistance seen in the node n_{m+1}, marked as $r(n_{m+1})R_0$. The multiplying term $r(n_{m+1})$ is simply the total resistance of all the previous branches divided by R_0.

The multiplying term $r(n_m)$ can be calculated recursively from the previous result by using Eq. (2.7) for any number of nodes. For simplicity, the resistance value R_0 is set to one. In the equation, $r(n_{m+1})$ represents the resistance of the previous stages and $r(n_m)$ is the new value for the next calculation step.

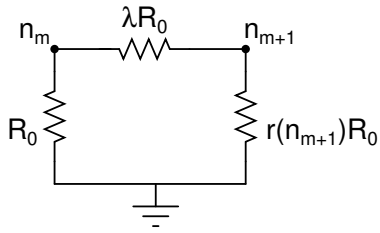


Figure 2.12 Circuit used to calculate the $R(n_m)$ value.

$$r(n_m) = \frac{\lambda + r(n_{m+1})}{\lambda + r(n_{m+1}) + 1} \tag{2.7}$$

Combining with the resistance $r(n_x)R_0$ of the first node n_x, that can be calculated using Eq. (2.8), the total resistance of a finite network can be calculated recursively using Eq. 2.7. It should be noted that also in Eq. 2.8 $R_0 = 1\Omega$

$$r(n_x) = \frac{\lambda + 1}{\lambda + 2} \tag{2.8}$$

The equations show that, since the λ is always greater than zero, as is the resistance scale $r(n_{m+1})$, then the left hand side of Eq. (2.7) is $0 < r(n) < 1$. If the values are calculated it can be noticed that the total resistor scaling value approaches a certain constant value for each λ. This scaling value, denoted here as r_{tot}, can be used in the calculation of the nodal voltages because, if we consider the network to be infinite, the same resistance value is seen from all nodes.

In the final step of the calculation, the symmetry of the network is also taken into account and both sides of node n_0 are reduced to a single resistor. The circuit representing this situation is shown in Fig.2.13.

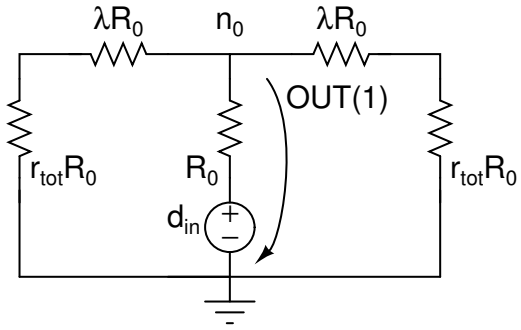


Figure 2.13 The resistance of the node n_0.

The resistance of the network is shown in Eq. 2.9 and the output of node n_0 can be calculated using this result with Eq. 2.10.

$$r(n_0) = \frac{r_{tot} + \lambda}{2} \tag{2.9}$$

$$OUT(1) = \frac{r(n0)}{(r(n0)+1)} d_{in} \tag{2.10}$$

Since the network is assumed to be almost infinite, all the following nodal voltages can also be calculated using the same equation and the output of the previous stage with Eq. (2.11).

$$OUT(N+1) = \frac{r_{tot}}{r_{tot} + \lambda} OUT(N) \tag{2.11}$$

As a result, Table 2.1 shows the output values of the nodes with different lambda values for the one dimensional case. The input $d_{m,n}$ was the aforementioned 255.

λ / node	n_0	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9
2	147	60	24	10	4	0	0	0	0	0
1	114	51	23	10	5	2	1	0	0	0
2/3	96	46	22	10	5	2	1	1	0	0
1/2	85	43	21	11	5	3	1	1	0	0
1/3	71	38	20	11	6	3	2	1	0	0
1/4	62	35	20	11	6	3	2	1	1	0

Table 2.1 ROI-values for different λ-values

The table shows, as expected, that the larger the λ-value, the smaller the ROI. The smallest λ-value shown here is 0.25 and with that the radius of influence is 8 nodes from the input node in both directions.

The results obtained here can also be interpreted as the transfer function of the 1-D resistive networks with different λ's. If a convolution kernel is formed in a manner similar to that of the Gaussian kernel, the following kernel $rn_{\lambda=1}$ (2.12) can be obtained for λ=1:

$$rn_{\lambda=1} = \frac{1}{256} \begin{bmatrix} 1 & 2 & 5 & 10 & 23 & 51 & 114 & 51 & 23 & 10 & 5 & 2 & 1 \end{bmatrix} \tag{2.12}$$

If the obtained LRN-kernel is compared to the Gaussian kernel with $\sigma = 1$ shown

in 2.6, it can be seen that, in the resistive network case the central term of the kernel is larger relative to other elements in the kernel. This results in the noisy objects not being removed as effectively as with the Gaussian kernel. Another remarkable thing is that the kernel decreases slower than the Gaussian kernel and this increases the kernel size. If the 2-D kernel is calculated, assuming that the function is separable, the resulting kernel for the LRN case can be written in the form of Eq. (2.13) when $\lambda = 1$.

$$a_2 = \frac{1}{315}\begin{bmatrix} 0 & 1 & 2 & 5 & 2 & 1 & 0 \\ 1 & 2 & 5 & 10 & 5 & 2 & 1 \\ 2 & 5 & 10 & 23 & 10 & 5 & 2 \\ 5 & 10 & 23 & 51 & 23 & 10 & 5 \\ 2 & 5 & 10 & 23 & 10 & 5 & 2 \\ 1 & 2 & 5 & 10 & 5 & 2 & 1 \\ 0 & 1 & 2 & 5 & 2 & 1 & 0 \end{bmatrix} \tag{2.13}$$

The shown kernel is limited to 3-neighbourhood and it is assumed that the effect of entries that were left out is negligible. Even with this assumption, the number of connections is reaching to the limits of a feasible implementation. Even more so if the calculation is considered to be performed in analogue domain; the dynamic range of the templates and the accuracy then becomes a problem.

2.6 Cellular Neural/Nonlinear Networks (CNN)

A Cellular Neural/Nonlinear Network theory was presented in [10] by Chua et al. The basic configuration was a 1-neighbourhood connected network but the main difference from the basic convolution processors was the introduction of a feedback term. This enables the Radius of Influence (ROI) to be larger than the S_i. This means that the interaction between cells is not restricted to the connected cells, but by using feedback the influence can spread out.

In this work, the basic CNN theory is used as a theoretical starting point and a way to model parallel array processors; therefore the basic theory will be presented here. Also the CNN-theory that led to the used cell implementation will be clarified in the following sections.

2.6.1 The Continuous-Time CNN

For each CNN-cell, an input, a dynamic state and an output value is defined. As mentioned above, the processing of the network is based on the interaction between the connected cells. This interaction affects the cell state and is proportional to the input and

output values. This relation is described using weight matrices, which are here called the templates. The interaction matrices are referred as A-template and B-template, where A-template is used to multiply the output value and B-template the input value. The contributions from the neighbouring cells are summed and the new state value is calculated from there. Then output is formed from the state using a sigmoid-function that limits the value of the output to -1 to 1. In between the limitation values, the output follows the state linearly, according to the original theory.

In a mathematical form, the state function can be presented as shown in Eq. (2.14).

$$\begin{aligned} C\frac{dx_{i,j}(t)}{dt} = &-\frac{1}{R_{in}}x_{i,j}(t) + \\ &+ \sum_{C_{k,l}\in N_r(i,j)} A(i,j;k,l)y_{k,l}(t) \\ &+ \sum_{C_{k,l}\in N_r(i,j)} B(i,j;k,l)u_{k,l}(t) + Z, \end{aligned} \tag{2.14}$$

where the $A(i,j;k,l)$ and $B(i,j;k,l)$ are the above mentioned weights and $N_r(i,j)$ defines the neighbourhood. The cell input, state and output are denoted with $u_{i,j}$, $x_{i,j}$ and $y_{i,j}$ respectively. Also shown are the constant terms CCNN-cell state capacitor representing the CNN-cell state capacitor, R_{in}, which is the input resistance of the summing node and Z, that is the constant biasing term used in setting the cell operating point.

The output of the cell is obtained from the state using Eq. (2.15). A graphical interpretation of the function, called sigmoid function, is shown in Figure 2.14

$$y_{i,j} = f(v_{i,j}) = \frac{1}{2}(|x_{i,j}(t)+1| - |x_{i,j}(t)-1|) \tag{2.15}$$

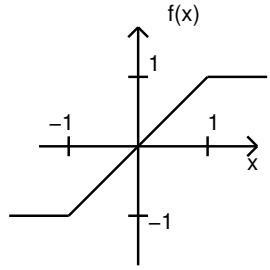


Figure 2.14 The sigmoid function that is used to limit the output between certain levels.

The template sets are space invariant, i.e. all the cells have similar templates. This leads to a very compact representation of the functionality of the network because it

can be described by using only two matrices and one constant. The template matrices for 1-neighbourhood are shown in Eq. (2.16)

$$A = \begin{bmatrix} a_{-1,-1} & a_{-1,0} & a_{-1,1} \\ a_{0,-1} & a_{0,0} & a_{0,1} \\ a_{1,-1} & a_{1,0} & a_{1,1} \end{bmatrix} B = \begin{bmatrix} b_{-1,-1} & b_{-1,0} & b_{-1,1} \\ b_{0,-1} & b_{0,0} & b_{0,1} \\ b_{1,-1} & b_{1,0} & b_{1,1} \end{bmatrix} Z = z \qquad (2.16)$$

Because of the small number of connections to the neighbouring cells when compared to some other parallel processing schemes, the CNN is considered to be more suitable for large-scale VLSI realisation. As Eq.(2.16) shows, the derivation of the output requires maximum of 18 multiplications and their summing in each cell. If this is transferred to a circuit that in principle realises the original state function, it can be presented as in Figure 2.15.

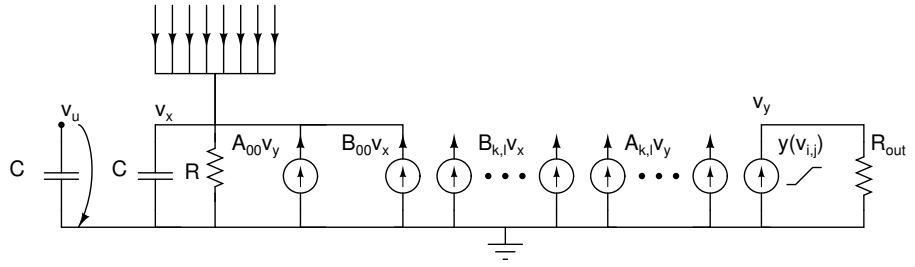


Figure 2.15 A CNN cell

In the figure, the interaction multipliers are shown as voltage-controlled current sources and the state value is formed from currents flowing to the summing node, from where they are fed through the state resistor to give the state voltage value. From this value, the output is formed by limiting the output with the sigmoid function.

As can be seen in Fig. 2.15, the realisation of the state equation is in principle very simple. However, since the multiplication is the dominant arithmetic operation and there are supposed to be 18 multipliers working in parallel in each cell, it is obvious that the cell area and the accuracy of the results is dictated by the implementation of the multipliers.

2.6.2 Positive Range CNN

In this section, the positive range CNN [34] is briefly described. As the name suggests, in this model only positive values are used in the input, state and output, which allows the multipliers to be two quadrant instead of four quadrant as required by the original theory.

The positive range operation is achieved by a linear transformation from the original equations (2.14) and (2.15). This maps the cell input and output according to Eq.(2.17)

$$\hat{x} = \frac{1}{2}(x+1) \qquad \hat{u} = \frac{1}{2}(u+1), \tag{2.17}$$

where the $\hat{x}$ and $\hat{u}$ denote the new transformed state and input values and x and u respectively denote the same values that are in equations 2.14 and 2.15.

The output nonlinearity can now be expressed as:

$$f(\hat{x}) = \begin{cases} 0 & \text{if } \hat{x} < 0; \\ \hat{x} & \text{if } 0 \leq \hat{x} \leq 1; \\ 1 & \text{if } \hat{x} > 1. \end{cases} \tag{2.18}$$

And the same is graphically shown in Fig. 2.16.

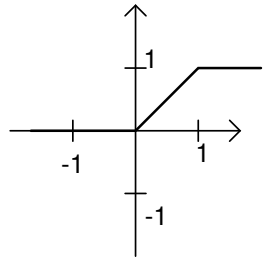


Figure 2.16 The sigmoid function in positive range CNN.

In [35], the positive range CNN was thoroughly investigated. As a result, it was shown that, in order to maintain the same input-output mapping with the original cell dynamics, the coefficients A and B were left untouched, but the new constant term $\hat{z}$ had to be re-calculated using Eq. 2.19.

$$\hat{z} = \frac{1}{2}\left[z + 1 - \sum_{C_{k,l} \in N_r(i,j)} (A_{k,l} + B_{k,l}) \right] \tag{2.19}$$

2.6.3 CNN Universal Machine (CNN-UM)

In article [36], some additions to the original cell were introduced to improve the possibilities of processing complex tasks and algorithms. The additions included a Global Analogic Programming Unit (GAPU), Local Logic Memory (LLM) and Local Logic Unit (LLU) for calculating with values of the LLM's. Also Local Analogue Memories (LAM) were introduced for storing grey-scale processing results. With these additions,

the results of the previous processing tasks can be used again as an input for a new task. A CNN including these additions was named a CNN Universal Machine (CNN-UM).

2.7 Resistive Networks as a Special Case of CNN

By looking at the topologies that were presented in previous sections, it is easy to see the similarities between the connections of the grids. However, resistive networks can not be directly transformed to a CNN-template set, since the connections between the nodes are not multipliers, the resistor to the ground also affects the transfer curve and there is no direct counter part for it in the CNN-cell. However, it is possible to write the state equations of both systems and through that to find the templates that result resistive network-like behaviour. This analysis is done here in fashion similar to that in [21] by Shi.

2.7.1 Comparison of the CNN and LRN as Shown by Shi

First in the analysis, a capacitor C is introduced to the resistive network between the node $V_{m,n}$ and the ground. It does not affect the steady state voltage; rather it represents the parasitic capacitances that are always present in circuit realisations. Using Kirchhoff's current law, the state equation can be written in the form shown in Eq. (2.20).

$$\begin{aligned} C\frac{dV_{m,n}}{dt} &= (V_{m,n-1}-V_{m,n})G_2 + \\ &+ (V_{m,n+1}-V_{m,n})G_2 + (V_{m-1,n}-V_{m,n})G_2 + \\ &+ (V_{m+1,n}-V_{m,n})G_2 + (V_{in}-V_{m,n})G_1 \end{aligned} \tag{2.20}$$

This equation can be rewritten by substituting $\lambda = \frac{G_1}{G_2}$ and separating the different nodes and the voltage source as shown in Eq. (2.21).

$$\begin{aligned} C\frac{dV_{m,n}}{dt} &= -(4+\lambda)V_{m,n} + V_{m,n-1} + \\ &+ V_{m,n+1} + V_{m-1,n} + V_{m+1,n} + \lambda V_{in} \end{aligned} \tag{2.21}$$

If the resulting equation is compared to the CNN state function, it can be seen that the RN equation can be written using the CNN templates, as shown in Eq. (2.22).

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & 0 \end{bmatrix} z = 0 \tag{2.22}$$

The above template set requires the state resistor to be $1/\lambda$. The linear nature of the resistive network processing results in the sigmoid function is not being needed to limit the output. This is because the output can neither exceed the maximum, nor go below the minimum, of the input.

2.7.2 Modifications to the Template Set

One of the good features of the CNN presentation is the fact that a complex phenomenon can be described with two matrices and one constant. However, in the above presentation of a resistive network as a special case of the CNN, the variable λ appears also in the state resistor. Therefore, it would be beneficial if the variable could be moved over to the self-feedback term. This way any CNN-UM chip could be easily programmed to process images with different λ-values. In order to achieve this, there are certain modifications that can be made to the original cell realisation.

As mentioned before, there is no need to limit the output and the sigmoid-function-forming circuitry can therefore be left out. Along with that, the output resistor is also unnecessary. This leads us to a simplified circuit for a cell that can be used in moving the λ to self-feedback. The circuit is shown in Fig. 2.17.

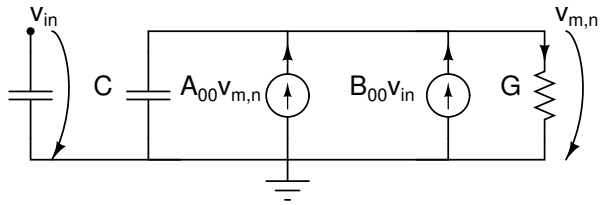


Figure 2.17 Circuit used to calculate the new A-template value.

In the figure, the centre elements of the A- and B-template sets, namely A_{00} and B_{00}, are shown along with the state resistor and the input and state capacitances. The two current sources in the figure are the voltage controlled current sources of the original CNN-cell. If the current equations are written for the circuit with the old A-template central term $A_{00} = -4$ and the original conductance value λ shown in Eq. (2.23), and similarly with the new central term $A_{00,new}$, shown in Eq. (2.24). Assuming conductance value G=1, we can solve $A_{00,new}$. This produces the new central term for the feedback template that is shown in Eq. (2.25).

$$A_{00}v_{m,n} + B_{00}v_{in} - \lambda v_{m,n} = 0 \tag{2.23}$$

$$A_{00,new}v_{m,n} + B_{00}v_{in} - v_{m,n} = 0 \tag{2.24}$$

$$A_{00,new} = -3 - \lambda \tag{2.25}$$

The resulting template set is shown in Eq. (2.26).

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -(3+\lambda) & 1 \\ 0 & 1 & 0 \end{bmatrix} B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & 0 \end{bmatrix} z = 0 \tag{2.26}$$

As the results show, the λ was transferred from the resistor value to the templates, which gives us the opportunity to control the smoothing with the templates. Another achievement comes from the required connections. When Eq. (2.26) is compared to the convolution kernel, shown in Eq. (2.13), it can be seen that the required S_i is shrunk from three to one. This simplifies the implementation of the array processor considerably.

2.7.3 All Current CNN Cell

As the original cell structure shows, the state and the output are linearly dependent on the currents flowing through the state resistor. If this state resistor is replaced with any, even a nonlinear, resistor R_{nl} and assuring that the current sources are dependent on the current flowing through the resistor, it is possible to achieve similar functionality in most cases, even if the dynamics of the cell are different. In an article [37], the current-mode approach was thoroughly investigated and also some experimental B/W results were shown. A grey-scale implementation published in [18] exploited the current mode approach as well. In this work, a current mode cell was also chosen in the approach. The current mode cell for realising CNN functionalities is pictured in Figure 2.18. The capacitor C in the summing node has been left out from the figure because, in principle, it is not required in this case. However, there is always parasitic capacitances in the circuit.

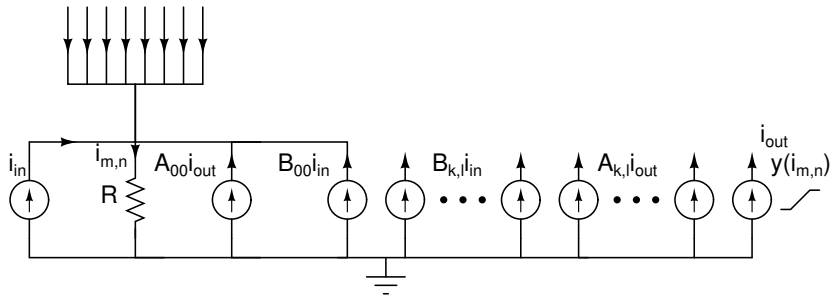


Figure 2.18 An all current CNN cell in principle.

2.8 Using Resistive Networks: Low-pass Filtering and Edge Detection

This section shows an example of an application where resistive network processing can be used in an efficient way and where the required computing speed is difficult to achieve using the traditional digital processors. In the article by Stoffels, [14] a CNN-based video compression algorithm was introduced, where the whole processing algorithm was carried out using CNN. Since the implementation of the pre-processing part was the starting point of this research the original idea will be presented here first.

2.8.1 Image Pre-processing According to Stoffels

In article [14], an all CNN object based video compression algorithm was proposed. The algorithm was divided into a pre-processing part and object-based motion estimation part. The motion estimation was achieved using black-and white images obtained from the video images by the grey-scale pre-processing part. The grey-scale processing part was chosen as the starting point for this work, since the black-and-white processing had already been shown to be implementable in a large scale, [38] and [15].

The processing starts with low-pass filtering of the image. The idea is to smoothen the image in order to minimise the high-frequency components in the output of the Fast Fourier Transform (FFT) and to suppress the effect of the errors that have occurred during the transmission from, for example, the sensors to the processor. The proposed smoothing operation can be expressed using the CNN template (2.27).

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} B = \begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.2 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{bmatrix} z = 0 \tag{2.27}$$

It can be seen from the equation that the smoothing operation is resistive-network-

style, because, when it is compared to the general case of resistive networks shown in Eq. (2.26), it can be seen that, in this case, the λ-value is 1 and the λ in the B-template has been spread to the neighbouring cells to obtain more smoothing by performing Mean-type filtering. Another way of seeing the equation is that it first does Mean Filtering with the B-template and then it is followed by resistive network filtering.

In the smoothing operation, some information is always lost. Therefore an edge restoration operation that attempts to preserve the edge information follows the filtering. This operation was performed in the original paper by calculating from the low-pass filtered image, the sum of absolute differences between a cell and the cells in its 1-neighbourhood and then comparing the sum of these absolute values to a threshold value. The method is based on the assumption that, if an edge exists, the sum of these differences is high, even if the image is low-pass filtered. The method is referred to here as a gradient calculation. This way, a black-and-white mask was obtained and was used for restoring the details in the image by replacing all the pixels in the low-pass filtered image with the original pixel values, where the mask indicated the existence of an edge. The whole operation was named in the original paper as *Edge-enhancing low-pass filtering*. In the following we will further discuss the method and show an alternative method to obtain similar results that is fully based on the resistive networks.

2.8.2 Realising an *Edge-enhancing Low-pass Filter*

In addition to the algorithm described above, there are other ways to implement similar functionality. In this section, a resistive network method will be discussed and compared to the original method.

The approach is here implementation oriented and the two possibilities do not have the same input/output-mapping; it is just that, similar results can be obtained. The main difference rests, in principle, in how the black-and-white mask is formed.

2.8.2.1 Using the Original Templates: Separate Low-pass and Gradient

If the original algorithm is to be realised, two different types of grey-scale operations have to be implemented because, in the algorithm, first the low-pass filtering is performed and after that the summation of the absolute values is performed. Even though it is possible to present the algorithm using a CNN-template, it is impossible to realise it with the CNN-UM that was presented in [36] or with the implemented and reported CNN-chips [39] or [11] without separating the gradient calculation into smaller parts. Therefore, a system where two application-specific networks were implemented was chosen as the approach [16]. The resulting system that was implemented on silicon, will be presented in Chapter 4.

2.8.2.2 Using Resistive Networks Only

As mentioned previously in this chapter, it is possible to obtain different levels of smoothing using different values of λ in resistive network filtering. This feature can be used also in edge-detection applications by comparing the results of two different λ-values. The approach was used in the Difference of Gaussians (DoG) calculation circuit presented in [8]. In this method, the same image is filtered twice with different λ-values. Then a pixel-wise comparison is made for all pixels in the image between the two filtering results, and if the difference is larger than a threshold value, the pixel is marked. Here, this result will be used as the mask for the *Edge-enhancing low-pass filtering* procedure.

The two methods, namely the Stoffels method and the resistive network method, are compared visually in the following images. The calculations are made with Matlab. The image in Fig. 2.19 is used as the input. The image is taken from the Foreman sequence widely used in video image processing analysis. The image size was reduced to 56×64 because it was also the size of one of the implemented chips in this work. In both methods the low-pass filtering was performed first. In the resistive network approach, the image was filtered two times with different λ-values and then the difference of the results was calculated pixel-wise. After this, the value of each pixel was compared to a threshold value. In the case of Stoffels method, the low-pass filtered result was used in the calculation of the gradient, as described in Section 2.8.1.

Figure 2.19 Original image used in the simulations.

The method by Stoffels is shown in Fig. 2.20. In the first image, the low-pass filtered image is shown. The image shows that the used template causes quite a bit of smoothing to the image and all the details are lost. The image in the middle shows the mask obtained from the gradient calculation. In the image the pixels that are considered to form an edge, are marked here with a black pixel and where the gradient calculation does not exceed the threshold value the pixels are white. The threshold value was in this case 30 using 8 bit accuracy. The mask image was used to obtain the image on

the right from the original and the low-pass filtered image. The output shows that the algorithm preserves most of the original details.

Figure 2.20 The low-pass filtered image, resulting mask and masked image using the method proposed in [14].

In Fig. 2.21 the results of the new method are shown. In this example, the used λ-values are 2 and 0.25. As in the image sequence above, the first image is the low-pass filtered image. The shown image is filtered with $\lambda = 2$. In the threshold calculations, the value of 7 was used to obtain the image in the middle. The threshold value is, in this method, significantly smaller than in the first method because here the two images are compared pixel-wise, whereas, in the other method, the sum of differences is compared to the threshold. The result after masking is shown in the image on the right.

Figure 2.21 The low-pass filtered image, resulting mask and masked image using the resistive network only.

If the two results are compared, it can be seen that similar results are obtained if the images are judged visually. Obviously, the results show that the edge detector realised with Stoffels algorithm gives a much coarser result than the resistive network. In this application this does not do any harm because the binary result is used as a mask to preserve the original edges. However, if the Stoffels method were used for finding the edge, it would be difficult, because the B-template averaging of the image loses much of that information.

When the two systems are compared implementation-wise, the main difference is the number of absolute value calculations, where as in the original system the required number is 8 and in the resistive network system it is 1. Therefore, if the goal is to implement an array processor comprising one calculation array, the first method requires

considerably more silicon area than the latter method. Naturally, the first method is also possible to implement with one absolute value block and multiplex the calculation of the absolute values in time. This, however, requires considerably more analogue memory and control logic and in addition to those also the calculation speed decreases by a factor of 8.

Finally, if the processing time is taken into account, the processing of the image in the latter case requires to low-pass filter the image twice, when, in the first method, just one filtering is needed. Since the low-pass filtering is a grey-scale to grey-scale operation it can be assumed that it is the most time consuming operation of the algorithms and therefore it can be estimated that the time required for the second method is almost twice the time required for the first.

This page is intentionally left blank.

Chapter 3

Designing Resistive Network Systems

The aim of this work was to implement a resistive-network-type array processor on silicon that could be used as a part of an image processing system. This meant that the size of the processor should be small compared to the maximum possible size of a silicon chip that can be manufactured with the used process. The interface should also be directly connectible to any system. This meant that the interface was chosen as digital. In this chapter, the system-level design of the array processor will be shown. However, first some previously published implementations of similar processors are briefly presented and their usability as a stand-alone processor in a larger system is discussed.

After that, the problems related to implementing array processors are considered before going to the proposed design. Here the problems are presented in more detail than in the previous section, where they were considered mainly from the division point of view. In addition to silicon size, processing speed and power consumption, which are problems found in any silicon implementation, the analogue array processors have design issues where trade-offs have to be made between the silicon size and accuracy of the processing. Also, the holding times of the analogue values come into consideration when the processor becomes so large that there are considerable differences in the required holding times between the cells in the network.

Finally, the system-level design that was devised for the work is presented in detail. The proposed system is based on reduction of the implemented cell-rows and separation of the processing tasks so that their designs can be better optimised, as already briefly mentioned in Section 2.2.1. This way, it is possible to save in silicon area and improve the speed and accuracy/silicon area ratio. Two separate array processor blocks

were chosen to be implemented, namely the resistive-network type low-pass filtering (presented in 2.8.1) and the gradient calculation block. The functionality of the system is shown in detail and the proposed system is compared to a full-size network using the silicon size, processing time and power consumption as parameters. Finally, using the same parameters as above, the effect of limited silicon area is taken also into account in the analysis. This way, it is possible to get a better understanding of how this type of system improves the performance.

3.1 Previous Implementations

Before going into the proposed system, we will take a look at some previously presented silicon implementations of resistive networks, as well as CNN's, and discuss their suitability for implementing large-scale resistive network filtering. First, some of the implemented processors that were intended for resistive network filtering and their reported results are shown. Their suitability for processing images as a part of an image processing system is then discussed. After this, those of the implemented CNN-UM's that are suitable for this type of processing are similarly presented and compared.

3.1.1 Implementations of Resistive Networks

In the pre-transistor era, the resistive networks were implemented with discrete components but after the findings from the use of resistive networks in the 80's, implementations have emerged, most notably [6], in silicon retina architectures. Later on different types of approaches have been used in these architectures, for instance [40] and [41]. In addition to silicon retina implementations, orientation selective networks use the same type of connections, as an example [42], [43] and [44] can be given. However, in this chapter the implementations shown in [8] and [19], when a one-dimensional network had been implemented on silicon, and a 2-dimensional network of [9] will be shown in more detail.

There are many ways to implement resistors in a silicon process. The most straightforward would be to use the poly resistor, but normally the resistance values obtained using these are relatively small compared to the area, in the range 100 to 300 kΩ per square, they require. Also their matching is rather difficult in a resistive network system because the connections to neighbouring cells are made using these resistors, therefore their orientations and placements can not be optimised for the matching. Also the resistors cannot be isolated from the control lines and power dissipating devices that degrades their performance. This method was used in [42]. Another way is to use transistors in the triode region. That approach was chosen in [9] in the cases where the resistance value had to be adjustable. In the same article, the constant resistors were

implemented using a p-well diffusion, which have even poorer matching properties than poly resistors. In article [19], a totally different approach was chosen: instead of voltages as the stimuli and response, a current mode operation was used. This way, the actual implementation of the resistor could be ignored, allowing the replacement of the resistors by Current Controlled Current Sources (CCCS). In the proposed system, we ended up with a similar realisation through CNN theory, as will be shown in the later sections.

In the following, the three previously published implementations will be briefly described and compared.

3.1.1.1 Network by Bair and Koch

In article [33] four different methods, that were developed by the same group to compute motion, were compared. One of the methods was finding motion from Zero-crossings [8]. In a way similar to that of the DoG method, the input was processed with two smoothing filters and the difference was calculated and compared to a threshold value. Instead of Gaussian filters, two resistive networks with different λ-values were used. The circuit implementing this is shown in Fig. 3.1.

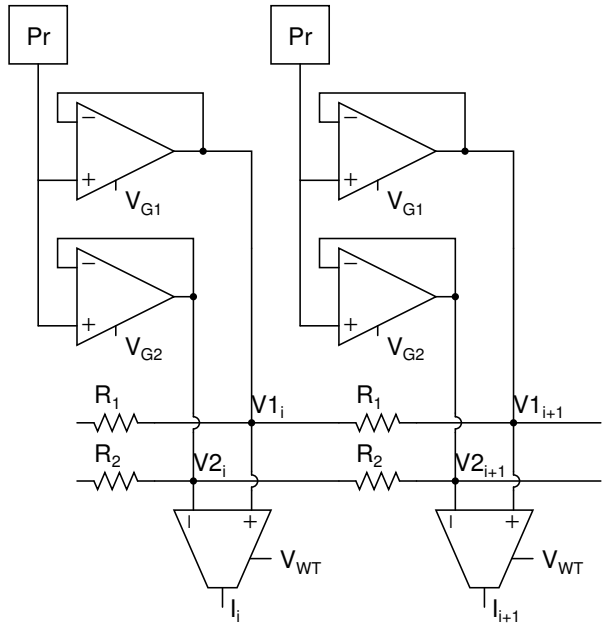


Figure 3.1 Resistive network realisation by Bair and Koch

In this realisation two 1×64 separate resistive networks were implemented. In both networks, each node was connected to the input, through conductances G_1 and G_2, and to neighbouring nodes through resistors R_1 or R_2. The input was a photo-receptor,

shown as Pr in the figure. The realisation of the resistors R_1 and R_2 was as Mead's saturating resistors [5] and the conductances were formed with transconductance amplifiers connected as followers. The conductance was controlled with the voltages V_{G1} and V_{G2}. The filtered images were subtracted by wide-range transconductance amplifiers in each node, producing a current I_i, proportional to the voltage difference across the input of the amplifier. This value is then compared to the threshold value. As an output, a 63-bit wide word results, where a logic high indicates the existence of a zero-crossing.

The measurements showed that the circuit was functional and the zero-crossings were detected. However, the functionality of the resistive networks itself was not commented on.

3.1.1.2 Network by Kobayashi et al.

In article [9], the use of resistive networks instead of Gaussian filters was proposed. It was shown that, in order to obtain a transfer function close to that of Gaussian filter, the resistive network requires ways to smooth the impulse response in the middle to avoid emphasising the noisy features in the images. Therefore the network shown in Fig. 3.2 was proposed.

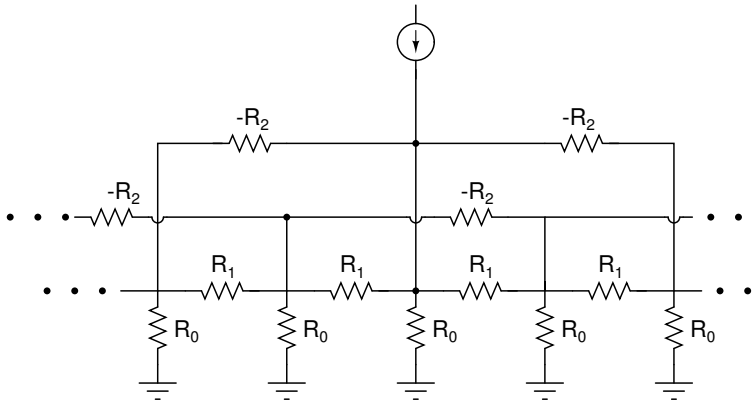


Figure 3.2 Resistive network realisation by Kobayashi

The figure shows a 1-D version of the circuit. Resistors R_0 and R_1 function as normal resistive network resistors and the negative resistor $R2$ brings the additional smoothing to the transfer curve. To be able to control the amount of smoothing, resistor R_0 was designed to be controllable and therefore a triode region transistor was used to implement it.

When moving on to a 2-D realisation another problem arose. The circular symmetry was hard to achieve if a rectangular network was to be used. Therefore, a hexagonal

structure, which is inherently circular, was adopted. As a result, a 45×40 network was implemented using 2μm CMOS process. The cell size became $170 \times 200\mu$m and each cell also included photo-receptor. Measurement results showed a Gaussian-type response, but it was not possible to obtain accurate measurement of the response.

The use of this type of processor as a part of an image processing system is quite difficult just because of the shape of the 2-D network, which is not used in any standardised image processing system.

3.1.1.3 Network by Raffo et al.

The circuit designed in the article by Raffo [19] was not intended to be a resistive network but rather a Gabor-type filter. However, because of the close relation between the resistive networks and Gabor filters [45], and especially because of the similarities between the implementations presented in the paper and in this work, the circuit will be briefly shown here.

In the paper, it was chosen to implement the network using current controlled current sources; in this way, a current mirror could be used in the implementation. The current equation of each node in the 1-D network was Eq. (3.1).

$$I_e(n) = I_s(n) + GI_e(n-1) + GI_e(n+1) - KI_e(n-2) - KI_e(n+2), \tag{3.1}$$

where $I_e(n)$ is the input current fed to node n and G and K are connection weights between the cells. Transferring the equation into a CNN template set, it can be written in the form:

$$A = \begin{bmatrix} -K & G & 0 & G & -K \end{bmatrix} \quad B = 1 \quad Z = 0 \tag{3.2}$$

The current mirror realisation is shown in figure 3.3. It actually shows just half of the required processing circuitry because the incoming current can have positive and negative values. Therefore, a similar circuitry is required, only the PMOS transistors are replaced by NMOS transistors and vice versa. In addition to that, a switching configuration is needed to steer the current to the correct part.

The implemented circuit had 17 nodes and was manufactured using a 2μm process. The circuit was by no means designed for large-scale implementation, rather only to demonstrate the feasibility of the approach. The measurements included comparison between the simulated response and the measured response. However, no quantitative error was given in the article.

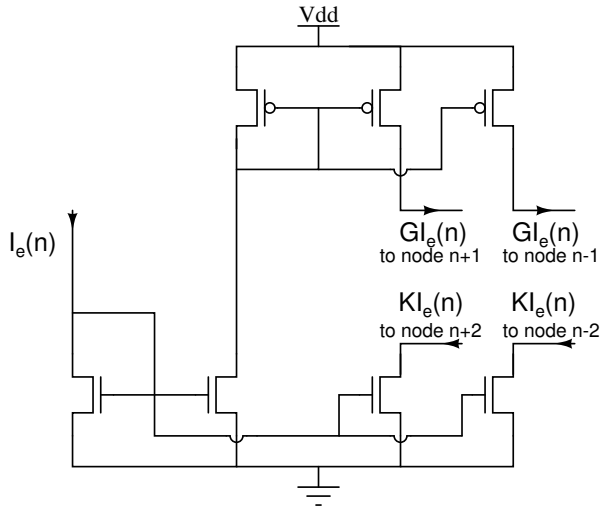


Figure 3.3 Single cell of the Gabor-filter presented by Raffo et al.

3.1.2 Implementations of CNN-UM's

There have been several implementations of the CNN-UM scheme, for instance [35], [15], [39], [11] and [20]. Also resistive-network-type Gabor filter based on CNN theory [46] have been presented and also implementations, [47]. But if the implementations are limited by their ability to perform resistive network processing and the grid size is sufficient for image processing tasks, basically two implementations are left. The two realisations are the CNN-UM by the Sevilla group, published in [11] and a CNN-based processor that was designed for epilepsy detection at Electronic Circuit Design Laboratory (ECDL) in Espoo, [20], which can be used also as a programmable network processor. The first design was completed as an attempt to realise the original continuous-time CNN-UM processor as in [36]. In contrast to that, the processor in [20] was the foremost designed to be application-specific for analysing brain activity in epilepsy and the calculation scheme was based on discrete-time iteration. The features that were required by the algorithm included second and third order templates; these requirements dictated the design, but it was not optimised for traditional image processing tasks.

If the published measurement results are considered only from the resistive network implementation point of view, in [20] the same template set as in Eq. (2.27), only without the B-template part, was used to demonstrate processing of data when there are many nonzero states. Due to this, the analysis of the filtering itself was omitted. However, the resulted output showed that the network had performed low-pass filtering. The presented measurement results of [11] did not have an example of grey-scale operation with feedback and the presented low-pass filtering was done using a 3×3

convolution kernel.

It can be claimed that with both of these chips the resistive network type filtering can be performed but the processing has to be performed in an iterative way and therefore the original processing power of CNN paradigm is partly lost.

3.1.3 Comparison of the Implementations

As the previous sections showed, the resistive network implementations have been for the most part proof-of-concept type realisations and the accuracy of the processing has not been at the top of the check list. Also, in the two cases [19] and [8] the implemented network was 1-D, which is not suitable for the image processing purposes and, in the 2-D case [9], the choice was a hexagonal network, which is rarely used in image processing and non-existent among standardised images. However, these implementations set a reference point for our design; Table 3.1 has therefore been compiled to show the features these chips include.

CHIP	Kobayashi [9]	Bair [8]	Raffo [19]
Processor type	Gaussian type Resistive network	Resistive Network	Gabor-filter
Grid size	hexagonal 45×40	1×9	1×9
Cell size (μm^2)	170×200	N/A	N/A
Settling time	20μs	100μs-10ms	N/A
Reconfigurability	R_0 variable	G variable	fixed
Used process	2μm	2μm	2μm

Table 3.1 Comparison of the reported parallel processors.

3.2 Problems Related to the Implementation of an Array Processor

There are various issues that relate to the implementation of a parallel array on silicon and that limit the achievable spatial resolution. Here, three such issues are discussed. These matters have to do with a large cell size, holding the analogue values and the accuracy requirements. In describing them, previously implemented chips are used as examples.

3.2.1 Large Cell and Array Size

Because the goal was to realise a processor block that could be implemented to be a part of a larger processor chip, the size of the individual cell became one of the main design issues. Considering that the largest processor chip reported to date is $435mm^2$ using $65nm$ digital process [48], and assuming that the same die size could be achieved with any process, to implement even a VGA-size array processor the size of the cell should be reduced to $37.6 \times 37.6\mu m^2$. When this is compared to recently achieved cell size of grey-scale array processor [11], the required size is one-fourth of the cell reported in the paper. The processor was implemented using a $0.35\mu m$ process, but even moving on to smaller processes does not improve the situation very much. The problem with the smaller line-width processes is that the supply voltage drops and along with that the available voltage swing. Also, the matching properties do not improve along with the decrease of the line width and the analogue devices cannot be scaled directly.

In most cases, however, it is possible to process images larger than the resolution of the processor array. That can be done by using the division of the image into smaller parts, and processing them separately. In section 2.2 an overview of the traditional method to perform the division was done. Later in this chapter a method will be proposed that can effectively be used in the resistive network implementation.

3.2.2 The Accuracy Requirements

When designing analogue circuits there is always some mismatch present between transistors. The easiest and most straightforward ways to minimise the effects of the mismatch are to increase the size of the transistors or to design some kind of a compensation circuit. When dealing with array processor realisations both lead to a larger cell size, which is not desirable. But if different template sets are investigated and their accuracy requirements are calculated, as was achieved for the B/W-templates in [49], it can be concluded that there is a large variation in the requirements between different templates. If then the requirements for grey-scale templates are considered, normally an 8-bit accuracy is expected in many applications. However, in analogue processing, even achieving 6 to 7 bits accuracy is a challenge with reasonably sized transistors. This means that the accuracy of the multipliers should be around 1% of the nominal value. These different requirements lead to a realisation where the accuracy is optimised for the strictest requirements. As mentioned before, this easily leads to large transistors and therefore to a larger cell size.

If we take the CNN algorithm in [14], it can be divided into grey-scale and black-and-white processing parts. For the grey-scale part, the accuracy requirements are the tightest and at least 6 bits of accuracy at the output is preferred. On the other hand, in the B/W part, the coefficient accuracy requirements vary between 5-20 %, depending

on the evaluation model. There is a big difference in the requirements and this opens up possibilities for optimisation if the different parts are implemented separately.

3.2.3 Holding the Analogue Values

In the proposed CNN-UM processor [36], there are also analogue memories included in the structure. These memories are used for storing the image in analogue domain to the network. The values to be stored may be obtained from a sensor, from an external memory or from the results of previous processing steps. This local memory structure is one of the main advantages of the CNN-paradigm, because it reduces considerably the number of read/write operations outside the array.

The problem with the analogue memories is in keeping with their physical size, i.e. small enough to be included into every cell while still maintaining the accuracy and holding time requirements. If we think of implementing large networks there is an obvious conflict between the requirements: the larger the grid size is, the longer the values have to be kept accurately in the memories while the input image is being loaded into the network or the processing result is written out after processing. In this case, we consider a common row-by-row image loading to the array and a network that does not include the sensor array. To maintain the accuracy, the memory capacitance has to be increased and, in that way, the size of the cell also increases. In [50], an analogue RAM was presented and measurement results were given. This memory structure was not intended to be used inside every cell as such. The results, anyhow, give us some insight into the properties of an analogue memory. The holding time obtained with that design was around 200 ms, but to get this result, the obtained density was only 637 memories per mm^2. If we compare that with a digital SRAM that was processed using a 0.25μm standard process [15], the density of an 8 bit per pixel digital memory was 12238 pixels per mm^2. Even by using the scalability rule of the digital processes and calculating the density for a 0.5μm process, it would result in 3060 8-bit pixel memories per mm^2. This leads to the conclusion that for some cases, it could be worth thinking of on-chip digital memories outside the cell grid, instead of implementing analogue memories on every cell. This, however, means that there will be a need for A/D- and D/A-converters.

3.3 The Implemented Array Processor System

This section deals with the proposed system-level design of a resistive network array processor. The processor that was implemented performs the *Edge-enhancing Low-pass Filtering* that was published in [14] and presented in Section 2.8.1. In order to achieve the required functionality, it a low-pass filtering part and a gradient calcula-

tion part had to be implemented. The first decision was to separate the two different processing parts and implement them in two different array processor networks, as suggested in [16]. The system level design started with considering the optimum solutions for both blocks and their interaction.

3.3.1 Optimising the Processor Size

The processing task can be divided first into the low-pass and gradient part. Then the low-pass part can be further divided into two parts as well. First is the B-template part, which averages the pixel value with its neighbouring pixels, and then the resistive network part with $\lambda = 1$, which performs additional smoothing. The input and output of both parts of the low-pass filtering are grey-scale, while in the gradient part the input is grey-scale and output B/W. From the beginning it was obvious that a designed system had to be implementable for large image sizes also, therefore the different processing parts could not be full image size networks.

The gradient calculation is obviously a 1-neighbourhood feed-forward operation and therefore S_i =ROI=1. This means that the minimum network size would be 3×3 if the gradient were to be calculated pixel-by-pixel. When considering the low-pass filtering part, it is somewhat more complicated. The B-template part could be calculated separately with a processor having only 1-neighbourhood and minimum size processor would be again 3×3, but the resistive network operation requires at least the information of 4 neighbouring pixels away from the pixel to be processed, as the convolution kernel Eq. (2.13) shows. This would lead to a minimum size of 9×9. In considering this the input and output formats also had to be taken into account.

As a result, the Reduced Cell-row System, already briefly described in Section 2.2.1, was designed for the low-pass processing part. Both sub-tasks of the low-pass part were implemented on the same cell because, otherwise, the operation would have required two analogue memories and an additional transferring of analogue values. To maintain the parallelism of the processing, row-by-row transfer of pixel values was chosen and therefore the width of the processor was the same as the width M of the image. The length of the low-pass network had to be more than 9 because of the ROI-requirement: it was chosen to be 16. This was because the row-wise operations could now be controlled by 4 bits. Also, a network of the same size could be used for smaller λ-values, which require larger neighbourhoods. After this, the size of the gradient block was chosen to have the same width as the low-pass part and the length to be the minimum 3. The system is depicted in Fig. 3.4 and the operation of the proposed system is described in detail in the following section.

The analogue processor structure and standardised representation of the images in digital format requires that there is a DA-conversion before the image is fed to the

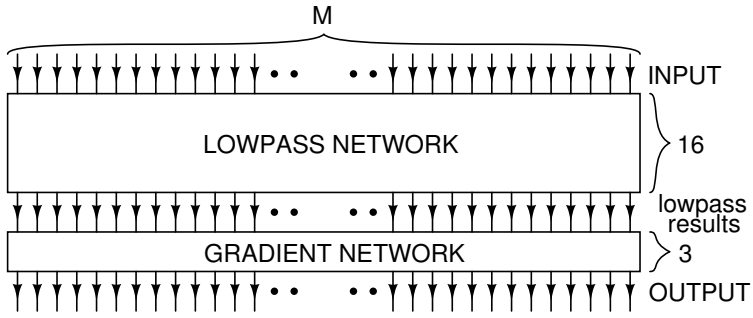


Figure 3.4 Low-pass filtering and gradient calculation blocks.

network and AD-conversions after processing when the results are written in to the digital memory. In our design, it was chosen to implement a digital image memory where the input and the output results could be stored. This was chosen because now a serial mode digital transfer was possible. Also the system would be directly compatible with any digital image processing system. For the transformation to analogue domain, it was decided to have column-wise converters used in a row-by-row manner. A similar approach was taken to the inverse transformation as well. Figure 3.5 shows the low-pass network from the reduced side.

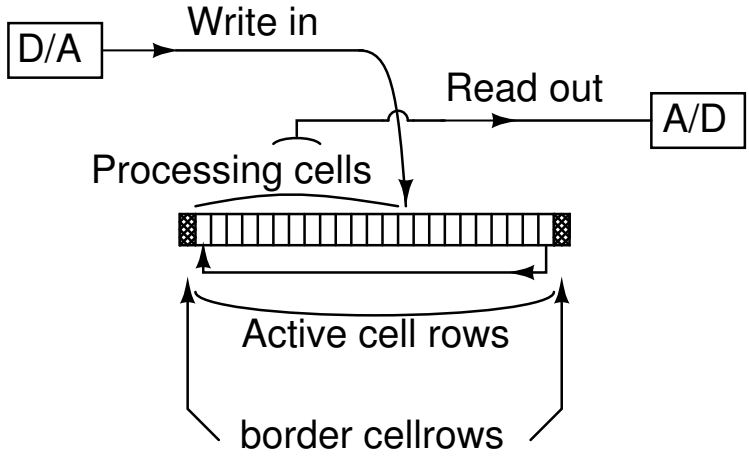


Figure 3.5 Side view of the low-pass network.

The figure shows also how the simultaneous write-in, read-out and processing operations occur. The write-in operation advances row-by-row and, after it, the processing cells follow. After a large enough neighbourhood is reached, the processing reaches its final value and the result can be read out from a cell row. A more detailed description is given in the following section.

3.3.2 Processing Flow to Process an Image Using RCS

This section describes the processing flow of the system that was implemented on the two test chips that were manufactured. There were some simplifications to the method that were done between the chips [25], but the basic operation remained the same. First, the processing flow and how the image is processed with the system is described in detail. After this, a comparison is made between the used method and a full image size processor. The comparison is made assuming that it is possible to implement an array processor of any size. The variables in the comparison are processing speed, energy consumption and silicon area. Finally, the limited silicon area is also taken into account and the comparison is made between the traditionally divided image processing and the proposed method with the same variables as in the previous case.

The processing of an image starts with loading the input to the image memory. The memory is 9 bits for each pixel out of which 8 bits are reserved for the grey-scale value of the pixel and one bit is for the calculated gradient result. Since the controlling of the memory is done row-wise, the loading of the image is carried out by first loading one row of the image to the shift register in serial mode and then writing it to a memory row. This is carried out until every image row is written to the memory.

When the image is ready in the memory, the processing can start. The process flow for the low-pass part was described in [51] in principle, and, in [17], in more detail. In Fig. 3.6, the processing flow is pictured. There the boxes with CLK inside refer to the clock cycle where the number indicates how many clock cycles the processing has carried out. WRITE IMAGE ROW refer to writing an input row from the DA-converter to the network, the CONVERT AND READ OUT refer to the AD-conversion of the processing result and, finally, the EVAL GRADIENT refer to the evaluation of the GRADIENT calculation and reading it out. In these boxes, the number refers to the image row in question.

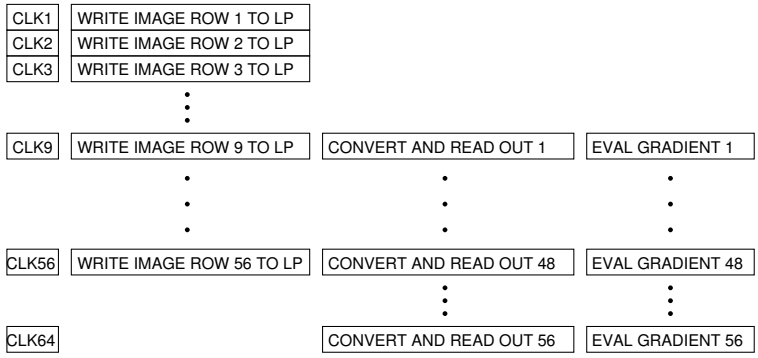


Figure 3.6 Timing of the parallel operations in the analogue part during the processing.

The figure shows the processing flow of an image that has length $N = 56$. As can be seen, the processing is ready after 64 clock cycles. It can therefore be concluded that the processing of one image of size $M \times N$, where M is the width of the image and N is the length, takes $N + 8$ processing cycles.

The system presented here was used in the implemented chips [25] and [52] and the input image size was 4×48 and 64×56 respectively. However, the processor grid sizes were reduced to 4×16 and 64×16 in the same order.

3.3.3 Advantages and Disadvantages of the Proposed System

In this section, the presented system is compared to a full image size, used in the implementation of a similar system. The comparison is made of silicon size, processing time and power consumption. Both of the considered systems are to be constructed using a similar cell. It is also assumed that, in both systems, the input is fed through digital-to-analogue converters in a row-wise manner and that the results are read from the array in a similar fashion using analogue-to-digital converters.

At the beginning of the analysis it is assumed that a full image size network can be implemented on a single silicon chip. After that, the effect of limited silicon area is also taken into account in the analysis. Finally, the comparison is also shown graphically, using standard video image sizes.

3.3.3.1 Silicon Area

Naturally, savings in silicon area is the main advantage of the proposed system and that was the aim of the design. The savings in silicon area are dependent on the size of the image if it is considered that the optimum would otherwise be a full image size processor.

If the image size is $m \times n$ and the size of the reduced network $m \times 16$ as previously presented, the ratio between the network sizes can be written in the form of Eq. (3.3):

$$\frac{A_{full}}{A_{reduced}} = \frac{m \times n}{m \times 16} = \frac{n}{16}, \qquad (n > 16) \tag{3.3}$$

m number of pixels in the image in horizontal direction,
n number of pixels in the image in vertical direction,

As the equation shows the saving is linearly proportional to the vertical measure of image. If, as an example, just the smallest standardised video image size QCIF, which is 176×144, is considered, the full-size network would require an area that is nine times the size of the reduced network.

3.3.3.2 Processing Time

When analysing the processing time, the settling time of the DA- and AD-converters and the network has to be taken into account. Here it is assumed that the settling time for the full-size network to reach its final value is the same as for one row of the reduced network to settle. If the proposed system is considered from the processing time aspect, it can be noticed that the processing speed is dictated by the slowest of the settling times mentioned above. This is because all operations occur simultaneously and each of them has to settle to its final value before the processing can move on to the next line. These assumptions lead to the equations for the total processing times: for the full-size network (T_{full}) Eq. (3.4) and for the reduced network ($T_{reduced}$) Eq. (3.5).

$$T_{full} = n \times t_{DA} + t_{NW} + n \times t_{AD} \tag{3.4}$$

$$T_{reduced} = (n+8) \times max(t_{DA}, t_{NW}, t_{AD}) \tag{3.5}$$

t_{NW} unit settling time of the network,
t_{AD} time the AD-converters require to reach their final output,
t_{DA} unit settling time of the cell input and DA-converters

If the time constants are further considered and an assumption is made that all the time constants t_{NW}, t_{AD} and t_{DA} are of the same magnitude, because all are analogue operations, the constants are assumed here to be equal. If this common settling time is denoted with t, the relation between the times $T_{full}/T_{reduced}$ can be written, using the equations (3.4) and (3.5), in the form:

$$\frac{T_{full}}{T_{reduced}} = \frac{2n+1}{n+8} \tag{3.6}$$

As the equation shows, the time ratio asymptotically closes to two as n increases towards infinity.

3.3.3.3 Energy Consumed to Process One Image

The energy consumption is related to both the array size and the time the processor is processing the result. This is because it is assumed that each cell consumes energy only while processing. The energy required by the converters is similarly dependent

on the time they are active. However, since the same number of conversions has to be done independently on the network size, the energy consumption of the converters is the same in both full-size and reduced networks. Therefore it can be left out of this analysis.

It is assumed that the power consumption is linearly dependent on the the number of pixels and the processing time. Therefore, the full-size processor energy (E_{full}) consumption can be given as shown in Eq. (3.7).

$$\begin{aligned} E_{full} &= A_{image} \times P_{NW} \times t_{NW} \\ &= m \times n \times P_{NW} \times t_{NW} \end{aligned} \tag{3.7}$$

For the reduced system, the worst case of the energy consumption is that all the cells of the processor are active while processing the image. This time was denoted above as $T_{reduced}$. Therefore, the energy consumption of the reduced network ($E_{reduced}$) can be given as shown in Eq. (3.8).

$$\begin{aligned} E_{reduced} &= A_{NW} \times P_{NW} \times T_{reduced} \\ &= (m \times 16) \times P_{NW} \times (n+8) \times t_{NW} \end{aligned} \tag{3.8}$$

P_{NW} unit power consumption of a single cell of the network during processing

The equation for calculating the relative power consumption is shown in Eq. (3.9).

$$\frac{E_{full}}{E_{reduced}} = \frac{m \times n \times P_{NW} \times t_{NW}}{(m \times 16) \times P_{NW} \times (n+8) \times t_{NW}} = \frac{n}{16 \times n + 128} \tag{3.9}$$

If the energy equations are compared, it can be seen that, in principle, in the Reduced Cell-row System the power consumption is 16 times larger than for a full-size network, if $n >> 8$. However, for this linear type of processing the settling toward the final value can begin before the full neighbourhood is loaded to the network and therefore the actual time constant for the latter case is smaller than for the full-size network. It is fair to say, however, that the reduced cell row system consumes considerably more power than a full size network.

In order to visualise the results, different image sizes are considered. In Table 3.2, some standardised image sizes are shown. The first column shows the name of the standard and its official abbreviation, then, in the next column, the size of the image in pixels is given and finally the abbreviation that will used in the following figures are

shown.

Name	Size	Abbrev.
Quarter Common Intermediate Format (QCIF)	144×176	q
Common Intermediate Format (CIF)	288×352	C
Video Graphics Array (VGA)	640×480	V
Super Video Graphics Array (SVGA)	800×600	S
eXtended Video Graphics Array (XVGA)	1024×768	X
Quarter Video Graphics Array (QVGA)	1280×960	Q
High Definition TV (HDTV)	1920×1080	H

Table 3.2 Some standardised image format sizes

In Figure 3.7, the results are presented graphically. The total number of pixels was chosen as the x-axis in the figures. The figure first shows the relation between the area of the full-size network divided by the area of the reduced network. The second and third plot show in similar fashion the relative time difference and power consumption between the two systems. As expected, the last two relative values remain almost constant independently of the image size and the Relative Area difference increases as the N of the images increases.

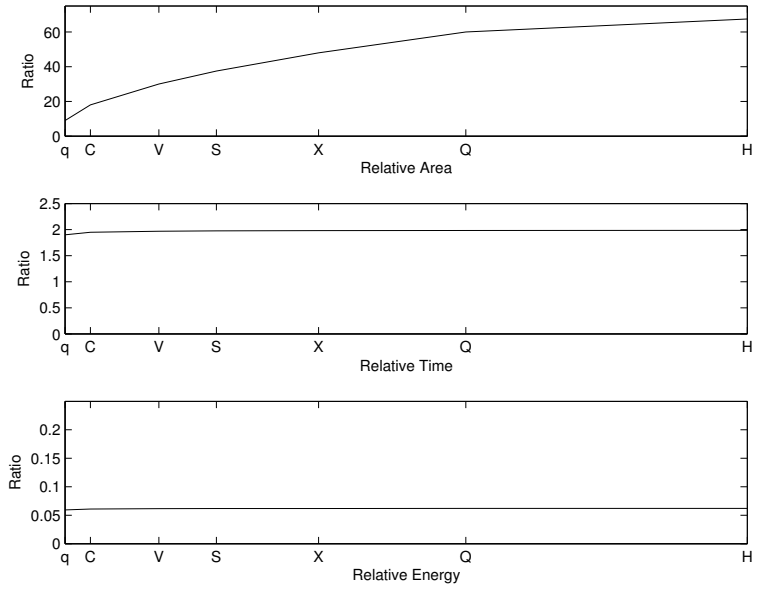


Figure 3.7 Comparison of the systems when the number of cells is not limited.

3.3.3.4 The Effect of the Limited Silicon Area

In the previous calculations it was assumed that it is possible to implement a full image size network. However, when moving on the larger standardised image sizes, this as-

sumption is not valid anymore. Here, the effect of the limited silicon area is considered with the same assumptions of similarity of the used cell in the area, power consumption and settling time. The two systems to be compared are here referred to as the full-size network and the reduced network.

The analysis starts with the definitions of aspect ratio r and the maximum number of cells i_{max}. The aspect ratio is defined here as shown in Eq. (3.10) and i_{max} is the number of cells that can be implemented on a single silicon chip.

$$r = \frac{m}{n} \tag{3.10}$$

The aspect ratio can be used in the division of the image for the full size processor. By designing the processor to have the same aspect ratio as the image, the number of divisions can be minimised in many cases and the same processor can be used in processing different size images with the same aspect ratio. There are other ways to design the network that can result in better performance with certain image sizes, but, for simplicity, only the division based on the aspect ratio will be used here.

If the maximum number of cells is i_{max} and the image is $m \times n$, the maximum width $m_{reduced}$ of the Reduced Cell-row System network is:

$$m_{reduced} = \frac{i_{max}}{16} \tag{3.11}$$

Since the full-size processor is limited by the same maximum number of cells as the reduced network, its maximum grid size is defined as:

$$\begin{aligned} m_{full} \times n_{full} &= i_{max} \\ \rightarrow m_{full} &= \sqrt{r \times i_{max}} \end{aligned} \tag{3.12}$$

m_{full} number of pixels in full-size network in horizontal direction
n_{full} number of pixels in full-size network in vertical direction

Using these values, it is possible to calculate to how many parts, denoted here by k, the image has to be divided in order to be processed with the full-size processor. The equation is given in (3.13).

$$k = \ulcorner \frac{m}{m_{full}} \urcorner^2 = \ulcorner \frac{m}{\sqrt{r \times i_{max}}} \urcorner^2$$

$$k = \ulcorner \sqrt{\frac{m \times n}{i_{max}}} \urcorner^2 \tag{3.13}$$

The k in Eq. (3.13) is the minimum of the needed divisions, it does not include the possible need for overlapping of the parts, which would increase it. Because of the definition of k and the assumption that the full-size network has the same aspect ratio as the image, the number of movements is the same along both axes to cover the whole image and it is equal to $\sqrt{k}$.

When these are taken into account when calculating the the equation for T_{full}, Eq. (3.4) becomes:

$$\begin{aligned} T_{full} &= k(2n_{full}+1) \times t_{NW} \\ &= k(\frac{2n}{\sqrt{k}}+1) \times t_{NW} \\ &= (2n\sqrt{k}+k) \times t_{NW} \end{aligned} \tag{3.14}$$

Because the processing time $T_{reduced}$ remains the same as in Eq. (3.5), the relation between processing times is:

$$\frac{T_{full}}{T_{reduced}} = \frac{2n\sqrt{k}+k}{n+8} \tag{3.15}$$

If the energy consumption is considered the effect of division is seen as a multiplier in the energy equation (3.16). Again the energy consumed by the converters is omitted because the number of conversions remains constant in both systems.

$$E_{full} = k(A_{full} \times P_{NW} \times t_{NW}) = k \times m_{full} \times n_{full} \times P_{NW} \times t_{NW} \tag{3.16}$$

The relative energy consumption is therefore:

$$\frac{E_{full}}{E_{reduced}} = \frac{k(m_{full} \times n_{full} \times P_{NW} \times t_{NW})}{m \times 16 \times P_{NW} \times (n+8) \times t_{NW}} \tag{3.17}$$

Remembering that $m_{full} = m/\sqrt{k}$ and $n_{full} = n/\sqrt{k}$, we get the same equation as in Eq. 3.9 and the relation between the energy consumption of the two systems remains:

$$\frac{E_{full}}{E_{reduced}} = \frac{1}{16} \quad (n >> 8) \tag{3.18}$$

So the division to subtasks does not change the relative power consumption. Of course, if the need for overlapping is taken into account, the relation increases.

Figures 3.8, 3.9 and 3.10 show how the limited silicon area changes the relations between the full-size network and reduced network. In Figure 3.8, the maximum number of cells is limited to 100×100. In this case, when using the RCS, the processing also has to be divided for the five largest image sizes. In these cases, the division was made using the smallest integer that resulted in the number of cells in the reduced network being smaller than the maximum number of the cells. Again, in the division, the overlapping issues were not taken into account.

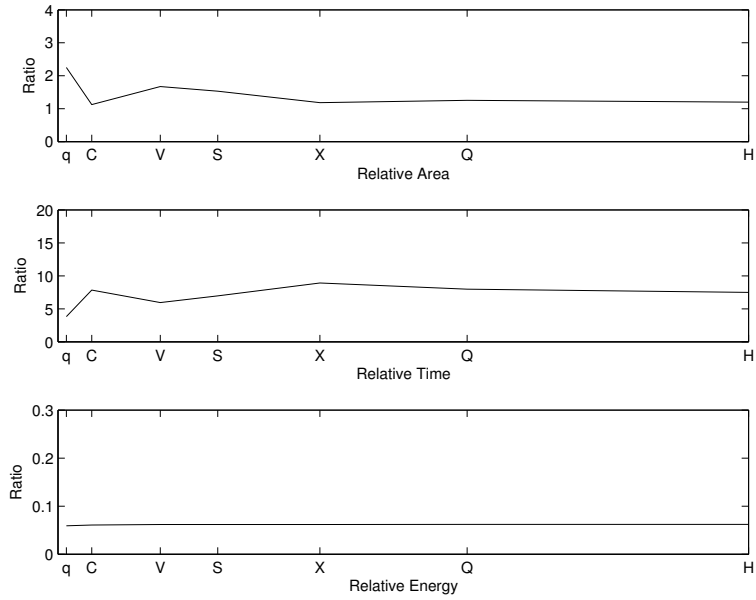


Figure 3.8 Comparison of the systems when maximum cell number is 100×100.

Most notably, the limitation of the number of cells is seen in the relative area and the processing time. The relation between the areas is close to one when the image size becomes large, because in both systems the number of cells is close to maximum. The increased time ratio is, in turn, directly from the number of divisions that has to be made in order to process the whole image. One common thing in the figures is that the curves are not monotonic. That is simply a result of the calculation method used, where the image ratio is kept also for the processor and therefore the obtained grid does not result in an optimum result for every image size. Especially for small image sizes, the processor grid can be better optimised.

Fig. 3.9 shows the same situation with the maximum number of cells equal to 200×200 and Fig. 3.10 for 300×300 case.

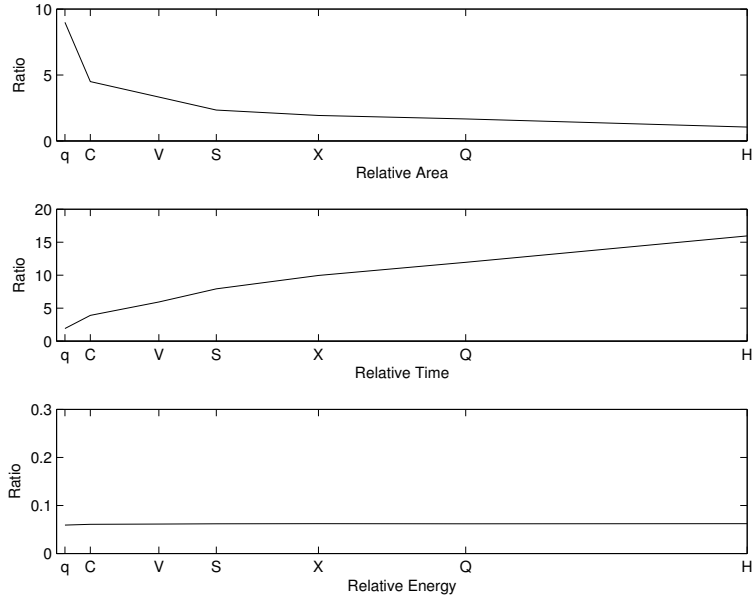


Figure 3.9 Comparison of the systems when maximum cell number is 200×200.

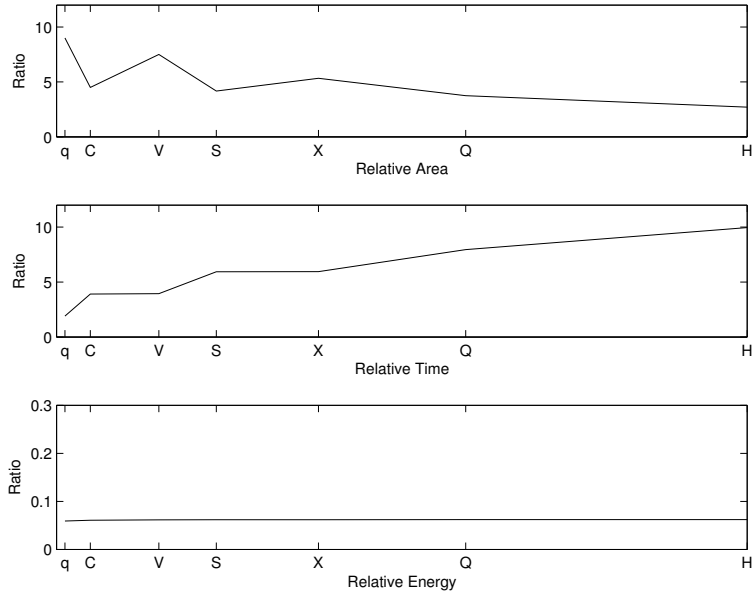


Figure 3.10 Comparison of the systems when maximum cell number is 300×300.

Chapter 4

The Implemented Resistive Network Array Processors

This chapter shows the two realisations that were implemented to verify the proposed array processor structure where the processing task was implemented using task specific sub-processors. The realisation was based on the *Edge Enhancing Low-pass Filter* algorithm proposed by Stoffels [14], which was presented in Section 2.8.1. The realisation of the low-pass part of the processor is close to that of a CNN-structure [10] with the connections to the 1-neighbourhood. However, the simplifications that were presented in Chapter 2 were used in the design. Therefore, the output limiting sigmoid function was left out and the current mode CNN cell was used.

Fig. 4.1 shows the block diagram of the realised chips. The different blocks are Digital Image Memory (DIM), AD/DA-converters and the Array Processor (AP) itself. AP consists of the two parts discussed in the previous chapter, namely the low-pass filtering and gradient calculation parts. For all the blocks, the controlling is done through the DigiCtrl-blocks that contain the required digital circuitry to control the operation. The I/O to the chip is performed through the DIM.

The basic structure of both chips was the same, while the main difference was the image size that the DIM could store. For the first chip, the image size was 4×48 and, in the second version, it was enlarged to 64×56. Naturally, this meant that the array processor needed to be wider in the latter case. The first version was manufactured using a 0.25μm process and the latter using a 0.18μm process; therefore all the blocks had to be re-dimensioned for the new process.

In the following sections, the realisations of the different blocks, shown in Fig. 4.1, will be presented. Naturally, there was some evolution inside the blocks in addition to dimensioning; that will also be mentioned in the coming sections. First, however, the

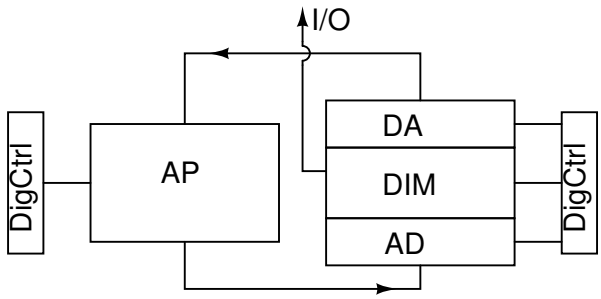


Figure 4.1 Block diagram of the implemented systems.

general specifications used are given.

4.1 General Specifications

Before starting the transistor-level design, certain specifications were made. First of all, it was decided that the digital input image was fed to the chip pixel-wise, resulting in serial mode I/O. This would naturally reduce the throughput of the processor, but that was not considered as a problem, since the main interest was on the internal processing speed. However, this requires conversion from digital domain to analogue domain before processing can start and a reverse conversion when reading the output values after processing. As a result of this, column-wise DA/AD-converters were implemented on the chip.

A positive range, current mode processing, presented in Section 2.6, was chosen as the approach to implement the processor. The dynamic range of the pixel value in the analogue domain was set to be 10μA. It was also chosen that a controllable offset was added to the pixel value. The aim of this was to speed up the loading of small pixel values when transferring them between different parts of the chips.

Due to the mixed-mode operation, three different supply voltages were used in the chips, namely analogue, digital and SRAM. As the analogue supply voltage, the maximum permitted voltage of the process was chosen for better analogue operation. For the first chip with 0.25μm process that was $2.5V$ and for the 0.18μm process it was decreased to $2.1V$. The digital supply voltage was $1.8V$ and SRAM $1.5V$ in the first design. In the latest design, these were lowered to 1.5 and 1.2 volts, respectively.

In the first version, the pixel value presentation was chosen as 6 bits for easier design of the converters and also for reducing the size of the DIM. In the succeeding version, the 8 bit presentation was chosen since it is normally used in digital images to represent the luminance of a pixel.

4.2 The Current Mirror

The basic building block of the whole analogue part of the design is the current mirror. It was used in the design of the low-pass cell, gradient calculation block, converters and in the bias circuitry of the converters, basically everywhere where the signal was in the analogue domain. Therefore it is good to analyse the basic operation and the error sources that are related to the implementation of current mirrors.

The current mirror NMOS-transistor configuration is shown in Fig. 4.2. There the DC input current I_{in} flows through the diode-connected transistor M1. Because of the capacitive load of the gates, it can be assumed that all the current flows through the transistor M1 from drain to source.

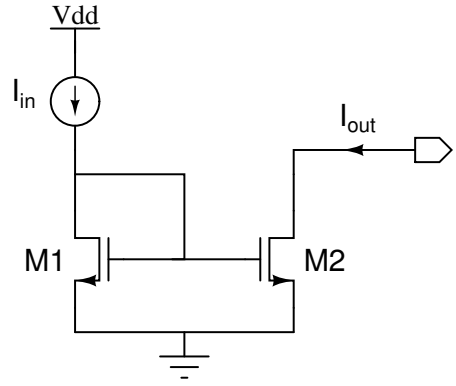


Figure 4.2 Basic configuration of a current mirror.

The current equation of a CMOS transistor in saturation can be written in the form shown in Eq. (4.1) [53]. The equation defines the drain current I_d through a transistor to be a function of aspect ratio W/L, gate-to-source voltage V_{GS} and drain-to-source voltage V_{DS}. K', V_T and λ_0 are process constants.

$$I_d = K' \frac{W}{2L} (V_{GS} - V_T)^2 (1 + \lambda_0 V_{DS}) \tag{4.1}$$

If the equation is considered the other way around, a current through a transistor results in a voltage from gate to source. If the current mirror structure of Fig. 4.2 is considered, the gates of the two transistors are connected together. Therefore they share the same voltage, which in turn results in a current flowing through the transistor $M2$ according to the Eq. (4.1).

The exact analysis of a current mirror has been presented in several publications, for instance in [54]. The basic, simplified output equation can be easily shown to be Eq. (4.2), where W is the width of the given transistor and L is the length.

$$I_{out} = I_{in}\frac{(W/L)_2}{(W/L)_1} \tag{4.2}$$

The equation shows that, in the ideal, case the output current is linearly proportional to the relation of the aspect ratios. Therefore, a current mirror can also be considered as an inverting current amplifier whose gain is controlled by changing the transistor sizes. In [55], its use as an amplifier was thoroughly investigated and the error analysis was conducted for AC input. However, in this work, it is sufficient that its error analysis is performed using DC currents. In the following section, the nonidealities are included in the analysis and the mismatch they cause on the output current in Eq. (4.1) is discussed.

4.2.1 Mismatch in the Current Mirrors

In order to analyse the effects of the nonidealities mentioned above, their origins first have to be pinned down. Equation (4.2) is calculated using the current equation of a CMOS transistor, shown in Eq. (4.1). If the definition of K' [53] is substituted to Eq. (4.1) the equation becomes (4.3), where also the width and length of the transistor are replaced with their effective values, W_{eff} and L_{eff}, respectively.

$$I_d = \frac{\mu_0 \varepsilon_0}{t_{ox}}\frac{W_{eff}}{2L_{eff}}(V_{GS} - V_T)^2(1 + \lambda_0 V_{DS}) \tag{4.3}$$

The equation shows that the current of a transistor is dependent on several parameters that are listed here in more detail:

μ_0 surface mobility of the channel,

ε_0 permittivity of the oxide,

λ_0 channel length modulation parameter.

t_{ox} thickness of the oxide

V_T threshold voltage

W_{eff} effective channel width,

L_{eff} effective channel length,

If the origins of the parameters are further considered, out of the seven parameters, only the last two can be chosen by the designer; with these two parameters the

functionality of the transistor is fully defined.

Due to the inaccuracies in the lithographic processing of the silicon chips, there is random error in the width and the length of the transistor, in addition to systematic errors that are included in the equation with the terms L_{eff} and W_{eff}. The thickness of the oxide t_{ox} also varies due to processing, independently from any other variation. But variation of the V_T is different because it is not an independent process parameter itself, rather it is dependent on several parameters [56]. The three parameters left, namely μ_0, ε_0 and λ_0 are either physical constants (μ_0, ε_0) or a constant dependent on transistor dimensions (λ_0); therefore they do not contribute any variation to the current mirror output.

The variation of V_T, t_{ox}, L_{eff} and W_{eff} has been researched widely, for instance in [57], [56], [58] and [59]. The variation of the three last parameters is considered to be normally distributed and independent and the variation of V_T inversely proportional to the area of the gate $\sqrt{W_{eff}L_{eff}}$ and normally distributed. Even though in [56] this V_T relation was said to be inaccurate, especially for short channel or thin transistors, the assumption was used in the simulations because it was used by the simulation parameters at the time of design.

4.2.2 *Monte Carlo*-simulations

The effect of the variations on the circuits was simulated using ®Hspice level 50 parameters when designing the first version with 0.25μm process [25] and ®Eldo level 59 parameters when designing the second version [52].

The simulation parameters and their variation are given by the foundry to the designers so that the effect of the random variation can be taken into account. The parameters that are usually included in these simulations are the V_T, t_{ox}, L_{eff} and W_{eff}, mentioned above. The variation can be included separately in each transistor inside the circuit. This way it is possible to model the effect of variation on the circuit performance.

At the time of designing the chips, both processes were quite immature and the process parameters were not defined sufficiently accurately for advanced analogue design, at least as far as universities were concerned. The problem was that, as the amount of variation was not given with the parameters accurately, for its value a sophisticated guess had to be given. However, since the first version was started from scratch, the mismatch simulations were conducted using the given parameters. In designing the second version, the results from the previous design were also taken into account in estimating the possible amount of variation.

In the first step of the simulations the transistor W/L-ratio was chosen so that the desired input levels could be handled. To improve the accuracy it is desirable to have

as large a gate overdrive voltage as possible [57], but the fact that the following load mirror was maintained in saturation had also to be taken into account.

After the W/L-ratio was found, the mismatch was included into the circuit. In the mismatch simulations, for instance 100 rounds of simulations were run and for each time new values were calculated for the parameters. This way, it was possible to obtain a distribution of the output. At this point of the simulations, the only way to increase accuracy of the circuit is to increase the size of the transistors.

Since it was possible to include the variation only in the transistors at the top level of the design, it was not possible to simulate the effect of random variation on the system-level performance of the low-pass network. Therefore the simulation of one cell was conducted so that the connections of the cell were connected to its summing node where normally the connections of the neighbouring cells are connected. Because the cell is supposed to perform low-pass filtering it should maintain the input value when connected this manner. However, this simulation does not take into account the possible different offset values of the neighbours or the effect of the neighbours that are not directly connected but which are inside the ROI. To overcome this shortcoming, the simulation system, that will be shown in Chapter 6, was developed. Also, the exact mismatch parameters were available at that time.

4.3 Analogue Circuitry

This section shows the transistor level implementation of the fixed template array processors chips. The presentation of the used circuitry is roughly divided into two parts, analogue and digital. In this presentation the AD- and DA-converters are included in the analogue part.

The analogue part of the chips consists of the low-pass filtering network and the gradient calculation block that follows it. The core of the design is naturally the cells of these blocks. Their size define the size of the array processor, their settling time partially defines the operation speed and their accuracy limits the accuracy of the processing.

In the following, the system description starts from the cell and its basic building blocks. Similar blocks were used in both implemented chips as well in the realisation of programmable resistive network, that will be later shown in Chapter 6. The transistor sizes are given for the most critical parts of the system for both processes used.

4.3.1 Fixed Template Low-pass cell

As was shown in [35] and again in section 2.6.2, in the positive range operation, the CNN templates remain the same, as with the full range CNN, but the constant term Z

has to be re-calculated using Eq. (2.19). By substituting the required template values to the equation it can be calculated that the constant term is zero and there is no need for an additional biasing term. Therefore the template set (2.27) can be directly used.

Considering the current mode CNN cell shown in Fig. 2.18 in Chapter 2, the definition of R_{nl} and the CCCS's can be fulfilled with the properties of the current mirror: the diode connected input works as a nonlinear resistor R_{nl} and the output follows the incoming current. As the same figure shows, all the incoming currents are summed in the same node and those currents are marked in the figure as $i_{m,n}$.

Fig. 4.3 shows the structure of the low-pass filtering cell that is based on the above mentioned figure. The main parts of the cell are the analogue memory, which is pictured as the capacitor in the figure, and the current scalers, that form the currents to the neighbouring cells according to the templates, are also shown the figure.

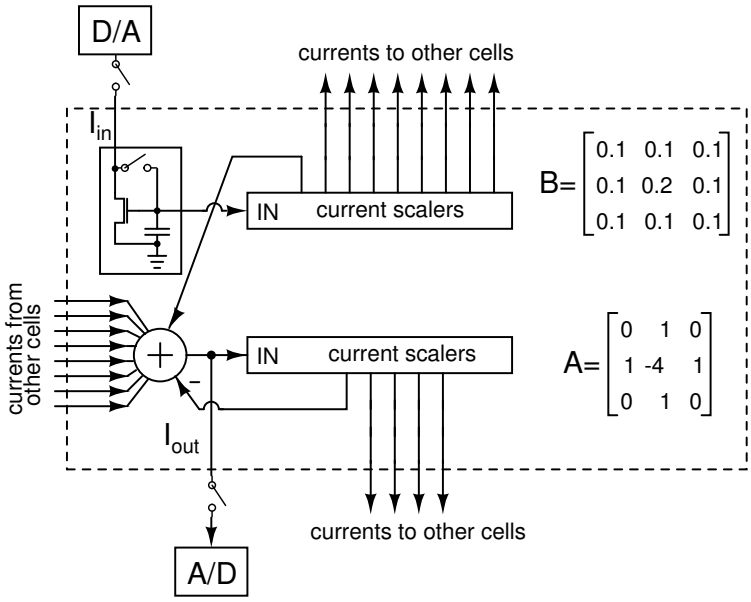


Figure 4.3 Block diagram of one low-pass cell.

Fig. 4.4 shows the transistor-level realisation of the input circuitry and the B-template. The current input to the cell is written through the *IN_CTRL* and *MEM_SW* switches to the current mirror, where the division by five is carried out and the value is stored in the memory transistor. Since the B-template has as its smallest value 0.1, the value stored into the memory is further divided by two in the PMOS current mirror and the value is then mirrored to the output transistors.

There were two reasons originally for this structure of two current mirrors. First of all the structure can be implemented with two transistors less than with an all-NMOS

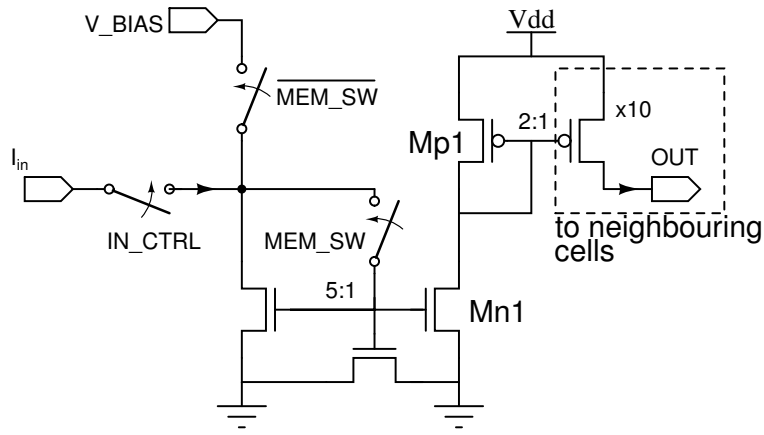


Figure 4.4 Realisation of the input and the B-template.

solution, because to obtain the required 0.1 template value, the current mirror would have required 11 transistors. The structure also enables the controlling of the evaluation because in the first version of the processor there was a switch transistor included in between the transistors $Mn1$ and $Mp1$ of the figure. By switching the transistor to conducting mode, the evaluation of the processing was started. It was left out from the second version, because if the evaluation started immediately after the input value is read in, it would speed up the settling time. Also the control circuitry was simplified considerably because the generation of the signal controlling the switch was quite complicated. However, with respect to the calculations of the power consumption of this type of network, shown in Chapter 2, the biggest disadvantage of the system is that all the cells in the network consume power during processing. This could be controlled by controlling the evaluation because there would not be any current flowing through the cells where the switch is open.

The voltage V_BIAS was added to the cell for the second version. The aim of the voltage is to keep the 5 parallel transistors, marked as single transistor $Mn1$, in saturation after write-in procedure to maintain the capacitance of the memory node constant. If the voltage in node n_{in} drops close to the ground level and the transistor is no longer in saturation mode, the capacitance of the mirror transistors increase [53] and the voltage stored in the memory drops, because the charge remains the same.

The A-template realisation is pictured in Fig. 4.5. There the currents from the neighbouring cells and the self-feedback are summed in the $CURR_{IN}$-node. The currents flowing through this node correspond to the current $i_{m,n}$ of Fig. 2.18. Simultaneously with the input current, the feedback, shown in the figure as $2NEIGH$, to the other cells is formed and the settling for the final output value starts. When the network has settled to its final value, the result can be written to the output line through the switch

OUT_CTRL.

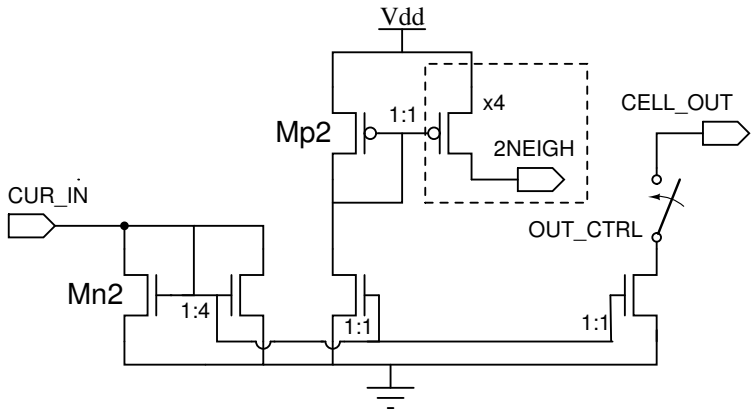


Figure 4.5 Realisation of the A-template and output of the cell.

Figures 4.6 and 4.7 show the simulation results of the low-pass cell. The simulation was conducted as described in Section 4.2.2. As the input to the cell, a DC current was fed to the input node I_{in}. The value of the current varied between 3μA to 18μA with a 1μA step. Figure 4.6 shows the evolution of the current flowing from *CELL_OUT*-node of Fig. 4.5. The input is written in the simulations to the analogue memory during the first 200ns, after which the MEM_SW, $\overline{MEM\_SW}$, IN_CTRL-switch change their state in the same order. As the figure shows, the output has almost reached its final value when the write-in operation has finished.

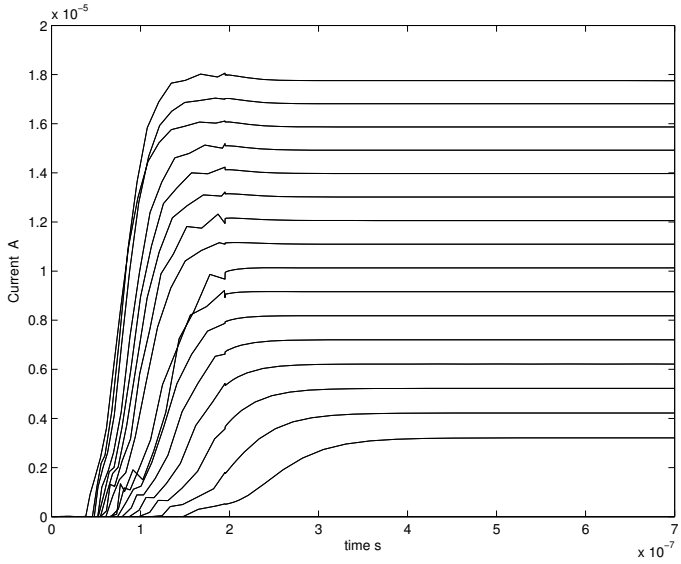


Figure 4.6 Simulation results with different input currents.

In Fig. 4.7, the steady-state currents are plotted against the input current. As the figure shows, the simulations predict that the cell itself would function quite linearly within the given dynamic range. The slight error in the gain can be compensated for by adjusting either DA- or AD-converters.

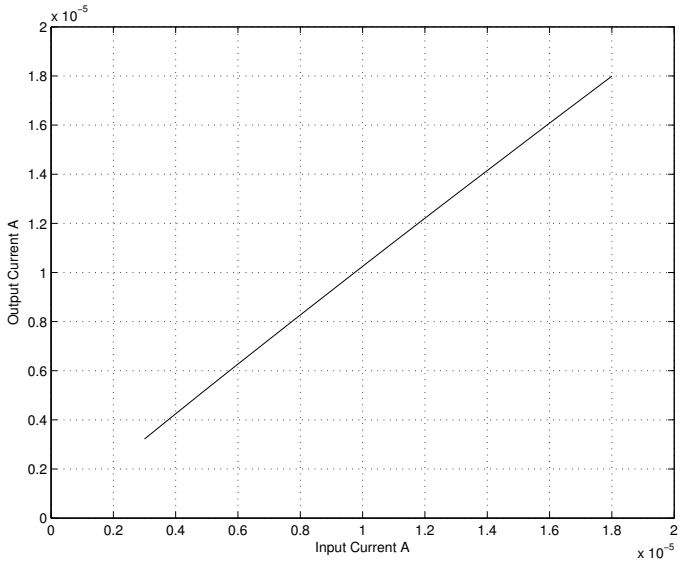


Figure 4.7 Simulation results with different input currents.

Finally, the transistor sizes are shown in Table 4.1:

Transistor	4×16 chip ($0.25\mu m$) W/L μm	64×16 chip ($0.18\mu m$) W/L μm
Mn1	1.45/3	1/3
Mp1	1.5/3	2.1/2.4
Mn2	1/2	2.6/3.3
Mp2	1/2	3.2/1.3

Table 4.1 Transistor sizes of the low-pass network.

4.3.2 Gradient Calculation Cell

The gradient calculation in this realisation is based on thresholding the sum of absolute differences of the pixels in 3×3-neighbourhood with programmable threshold value. This is a B-template operation and therefore, to calculate the gradient value of the centre pixel in the grid, the minimum grid size is 3×3.

The calculation is done in a row-by-row manner and this leads to a realisation of the $M \times 3$ network, where M in this case was first 4 and then, in the latter version, 64. The block diagram of the realisation is shown in Fig.4.8. The functionality is described below for one column.

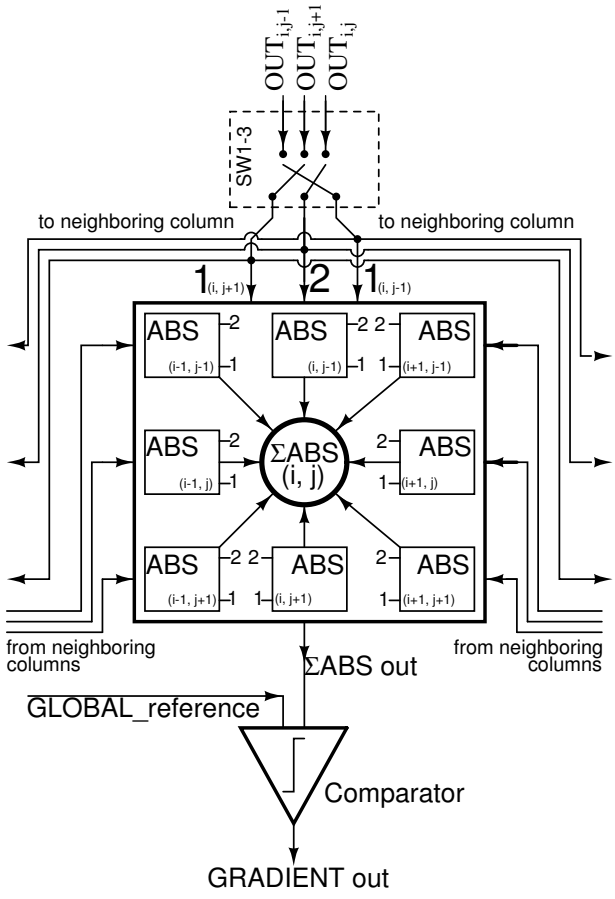


Figure 4.8 Block diagram showing the functionality of the gradient cell.

In the figure, at the top are the incoming currents from 3 consecutive cell-rows of one of the columns in the low-pass network. On the actual chip, each output of the low-pass cells is connected to one of the three output lines, except for the 16th cell row, where the output line can be chosen. This leads to a need to be able to steer the

currents from the three output lines to any of the needed absolute value calculation blocks of the gradient network. This operation is performed in the $SW1-3$ block in the figure.

For each column, there are eight absolute value calculation blocks, one for each neighbouring pixel, marked in the figure as *ABS*. These blocks calculate the absolute value of the difference between the two current inputs, marked as 1 and 2 in the figure. Input 2 is the same for all the absolute value blocks of the same column, and it is the value of the pixel in the location (i, j) in the image, whose gradient is being calculated. The other input, marked with 1, is fed with the value of the corresponding neighbouring pixel. In the figure these incoming currents are marked with 1 and the source pixel address.

After calculating each separate absolute value, all the results are summed and the sum is compared to a threshold current ($GLOBAL\_reference$) that is controlled outside the chip. As a comparator, an inverter is used. After the output of the inverter ($GRADIENT\_out$) has settled, it is read to the memory array.

The realisation of the absolute value block is a current rectifier, presented in [60]. The transistor realisation is shown in Fig. 4.9.

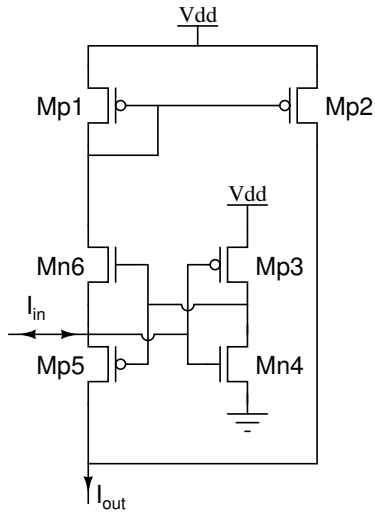


Figure 4.9 Single absolute value calculation block.

There the sum of the currents I_{in} is conducted to gates of the transistors $Mp3$ and $Mn4$, connected as an inverter, which acts as an amplifier. The inverter controls the current switches $Mp5$ and $Mn6$, which steer the input current either to the current mirror formed by the transistors $Mp1$ and $Mp2$ or directly to the output, depending on the direction of the current. The transistors of the current mirror are equal in size and the only operation it performs is the change of direction of the current.

4.3.3 Digital-to-Analogue converters

Since there was a converter for each column in the image, the width of the converter was dictating the design. In the first version, the goal was to implement a 6-bit current mode converter with a dynamic range of $10\mu A$ with controllable offset to keep the output in the positive range and larger than 0 all the time. In the second version, the goal was to design a similar 8-bit current mode converter. A binary weighted DA-converter was chosen for the implementation. Figure 4.10 shows the block diagram of the realised 8-bit converter.

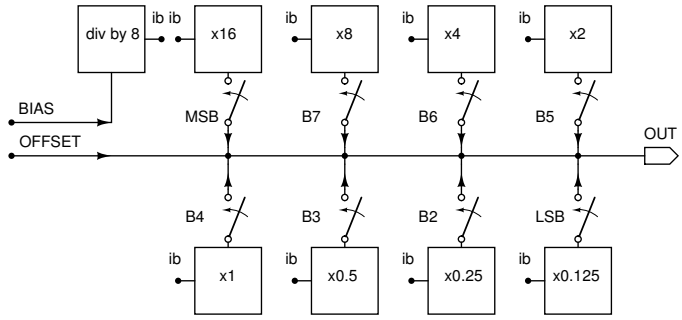


Figure 4.10 Block diagram of the used DA-converter.

As seen in Fig. 4.10, the dynamic range is defined with the *BIAS*. This current mode reference is first divided by eight inside the converter to form a local reference *ib* for the binary weighted current mirrors. By multiplying this *ib*, the different branch currents are obtained, and depending on the desired digital code, the switches conduct the branch currents to the output. The five most significant bits (MSB) are formed by parallel mirror transistors and the three least significant bits (LSB) are obtained by increasing the length of the mirroring transistor. The OFFSET input shown in the figure is directly connected to the converter output and its current summed with the branch currents. For faster output settling, the branch currents are directed to a dummy load transistor when their switch is open.

4.3.4 Analogue-to-Digital converters

Between the two designs, one of the major modifications was introduced to the realisation of the AD-converters, or rather to their control. In the first version of the converters, an outside clocking signal was required to control the conversion. This meant that a fast digital signal had to be brought close to the analogue processing blocks. This realisation was changed to the asynchronous Successive Approximation Register (SAR) converters [20] when the larger network was designed.

The asynchronous converter is shown in Fig.4.11. There the control signals for storing each bit and controlling the DA-converter are generated using a chain of delay elements in the *SIGNAL GENERATOR* block, which also includes the conversion result buffers. The DA-converter of the AD-converter (AD DA in the figure) is similar to the DA-converter described above.

The conversion starts with setting the $CONVERT$ signal high and while it is high the chain of delay blocks form the control signals for all the bits, starting from the MSB. After all bits are converted and the result is written to output buffers, the CONVERT signal is set to zero and the *ZERO*-signal is set to high, which erases the conversion result from the *SIGNAL GENERATOR* block. Because it has to be possible to write to the image memory from the input shift register also, the output buffers have to have high impedance (High-Z) mode. The 3-state buffer block is controlled with the $WRITE$-signal, which sets the buffers to High-Z mode when down.

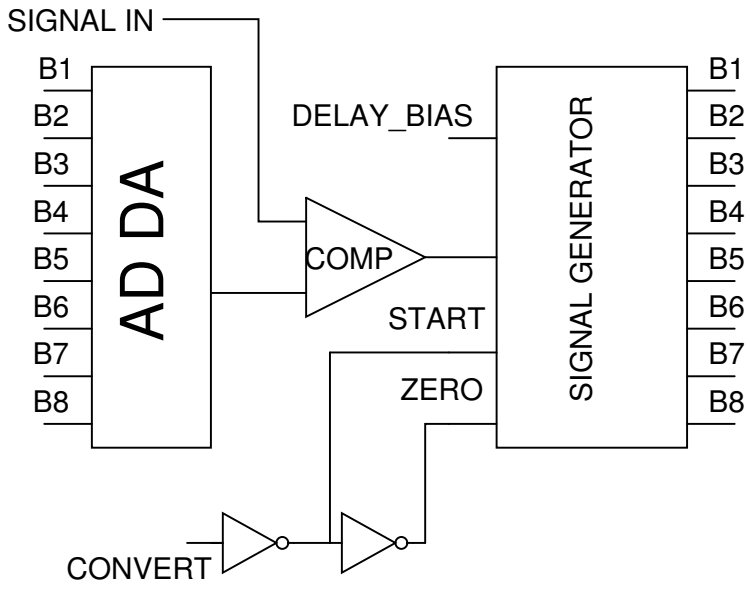
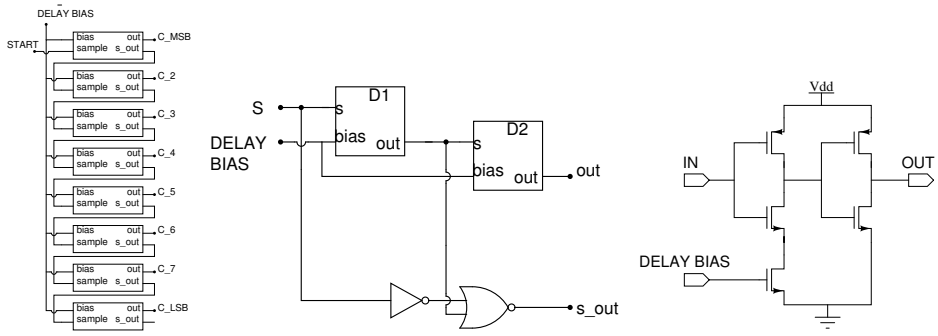


Figure 4.11 AD-converter realisation.

Some of the contents of the *SIGNAL GENERATOR* block are shown in Fig. 4.12. The chain of delay elements is shown in Fig. 4.12(a) and one delay block from the inside is shown in Fig. 4.12(b). Figure 4.12(c) shows the implementation of one delay element $D1$ or $D2$ of Fig. 4.12(b). The difference of the two delays is that D2 has a shorter delay. By adjusting the *DELAY_BIAS* voltage, the current flowing through transistor $M1$ can be controlled and this way the propagation of the pulse, fed to the IN-node, can be controlled. As a result, 8 non-overlapping signals are generated to nodes C_{MSB}-C_{LSB}.



(a) The chain of delay blocks.

(b) One delay block.

(c) Realisation of $D1$ and $D2$.

Figure 4.12 Clock generation for the AD-converter.

In addition to the parts shown, the *SIGNAL GENERATOR* also includes cache memory implemented with flip-flops that stores the conversion result before it is transferred to output buffers. Some logic circuits are also included to control the operation.

4.3.5 The Bias and Offset Distribution for the Converters

As the previous sections showed, both the actual DA-converters and the DA-converters of the AD-converters need two biasing signals, *BIAS* and $OFFSET$. With the first chip, respective currents were brought directly to two current mirrors, which copied the currents to the four column converters. In the case of the bigger network, it was decided that, in order to avoid the effects of the supply voltage drop, the mirroring was to be performed in two stages. For the first time, the current coming outside the chip was copied to eight branches, each of which was further copied to eight more, totalling the required 64 currents for both *BIAS* and $OFFSET$ signals.

4.4 Digital Circuitry

Even if all the calculation is performed in the analogue domain, half of the silicon area of the chips was digital circuitry. The main area was occupied by the image memory (*DIM*), but also the I/O circuitry and the control of the analogue part had their share. In the following, the different parts of implemented digital circuitry is presented.

4.4.1 Control of the Analogue Circuits

As the previous sections showed, quite a few signals are required to steer the processing of the processor. Figure 4.13 shows the external and internal controlling signals of

analogue processing circuits. The dotted line separates the signals that are fed from the outside of the chip from the signals that are generated inside the chip.

One of the reasons why the chosen number of cells rows in the low-pass network was 16 was the fact that it was possible to generate all the controlling signals using only four bits. These four bits are named A, B, C, D in Fig. 4.13. Gray-coded binary codes [61], i.e. only one bit changes at a time, were used in order to avoid glitches in the controlling signals. With these four bits, 67 control signals were generated to steer the operation during the processing. The *LP_CTRL*-block signals *IN_CTRL*, *MEM_SW*, $\overline{MEM\_SW}$ and *OUT_CTRL* were shown in Figs. 4.4 and 4.5. *LINE*123 -signal controls in which, of the three output lines, the output of the 16th row is written to, as was described in the presentation of the gradient calculation. One additional bit was needed to control the neighbourhood connections of the first and last Active Cell-rows, as it was described in Section 2.2.1. It is called *NEIGH_CTRL* in the figure. The signals on the left side of the dotted line are controlled from the outside of the chip, while the rest of the signals are generated from these and are internal only.

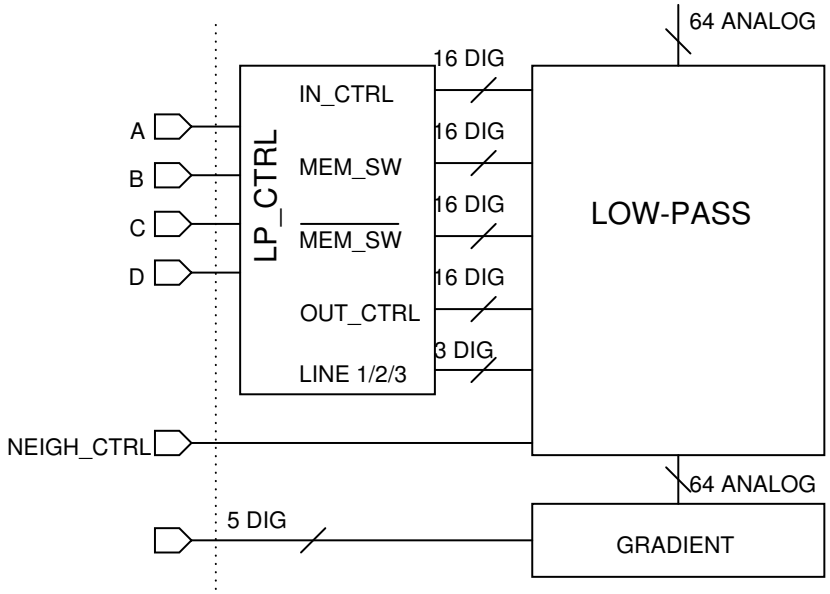


Figure 4.13 Control signals for the analogue processing.

4.4.2 I/O-circuitry

The data input to the chip was implemented as a chain of shift registers, depicted in Fig. 4.14 as *SH_REG*. The same shift registers were used also in reading out the processing results. There are 64 shift registers, one for each column, and they have

9 parallel bits, out of which 8 are used for the pixel information and one is for the gradient calculation result. Therefore the writing in is done pixel-by-pixel; after one row of the image is loaded in the *SH_REG*, it is written to the SRAM. Figure 4.14 shows the block diagram of the implementation. In the figure, the dashed line again represents the boundary surface of the chip and the outside world.

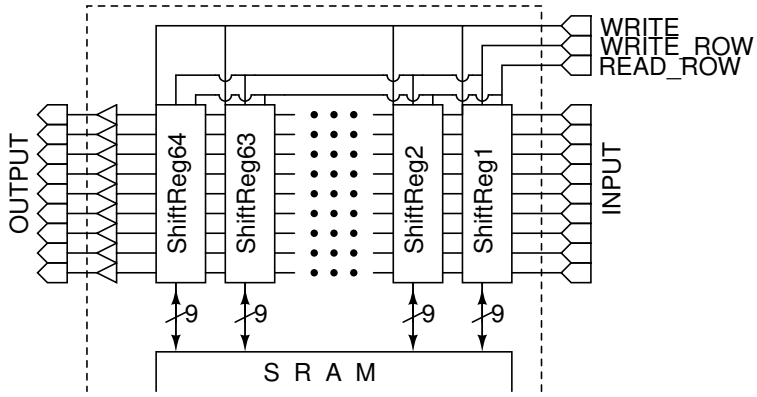


Figure 4.14 The chain of shift registers used as I/O-circuit.

Figure 4.15 shows the realisation of a single shift register element. The four controlling signals seen in the figure, namely $WRITE$, $WRITE\_ROW$, $\overline{WRITE\_ROW}$ and $READ\_ROW$, are used for controlling the block. The fifth signal, $WRITE\_IN$, is formed as a logic OR-function of the $WRITE$ and $READ\_ROW$ signals. This is because it has to be possible to write from the previous register in the chain of registers or from the digital image memory to the shift registers.

With the $WRITE$, signal the contents of the shift register blocks are transferred along the register line. The column operations are controlled with the $WRITE\_ROW$ and $READ\_ROW$, where the preceding is used when writing the content of the shift register to one memory row and the latter is used when reading a single memory row to the shift register. The writing to the digital memory is performed using the 3-state buffer because, as mentioned in reference to AD-converters, the shift registers and AD-converters share the write-line to the SRAM-cell. The buffer is controlled with the $WRITE_ROW$ signal and its inversion.

4.4.3 SRAM Image Memory

The building blocks of the image memory (*IM*) are shown in Fig. 4.16. *IM* consists of six transistor SRAM cells, shown in Fig.4.16(a), and the sense amplifiers, Fig. 4.16(b). The SRAM library cells were not available with the process used, therefore they had to be designed and realised using normal digital process parameters and design rules. This

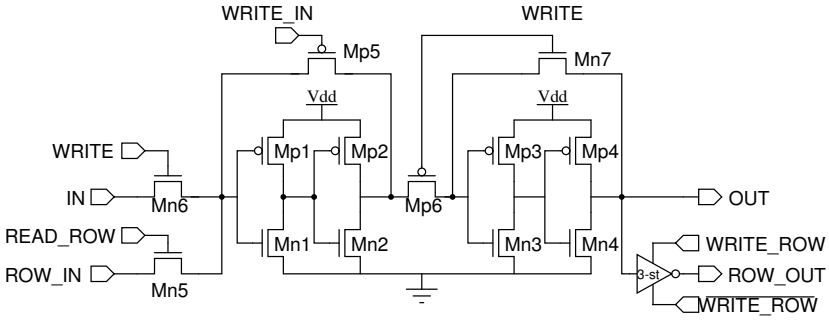
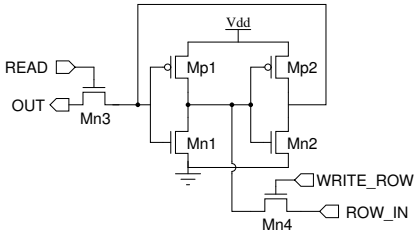


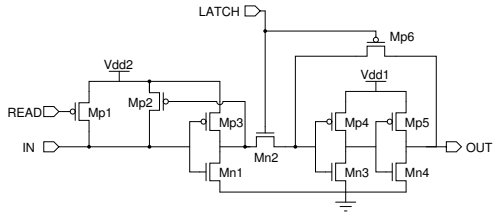
Figure 4.15 One shift register element.

resulted in over a two-times larger silicon area than a SRAM memory cell designed with design rules optimised for SRAM design, for instance [62].

The sense amplifier is shown in Fig. 4.16(b). The first stage is the pre-charge stage, formed by the transistors $Mp1$ and $Mp2$ and the inverter, consisting of the transistors $Mp3$ and $Mn1$, where the first transistor is the pre-charge and latter the keeper transistor [63]. The second stage latches the read value and stores it until a new value is read. The other functionality of the second stage is transforming the SRAM logic level $Vdd2$ to the digital part logic level $Vdd1$.



(a) A SRAM cell.



(b) The sense amplifier.

Figure 4.16 The image memory building blocks

4.5 Layout Design

The design of the layout has two main aspects when designing mixed-mode array processors: the silicon area and accuracy of the analogue processing parts. The grey-scale analogue parallel processor transistors are quite large when compared to the transistors that were used in the B/W array processors [35] and [15]. Therefore, the layout design does not offer that many possibilities of minimising the cell area, but instead, by carefully placing the transistors, the accuracy can be increased.

In the first version of the processor not so much emphasis was given to the effect

of the layout design on the accuracy of the processor. As the result will show in the following section, this had an impact on the measured results. In the second version this aspect was also taken into account by using dummy devices where possible and by designing the layout of border cells based on the layout of the basic cell. This provided the cells on the edges of the array the similar surrounding as the rest of the cells.

The original idea of the system-level floor-plan is shown in Fig. 4.17. The layout design was started from the analogue part and it was considered possible to draw the rest of the circuitry to the same pitch as the analogue part. However, it turned out that the 9 bit SRAM could not be squeezed to the same pitch. Therefore the floor-plan of the layout is similar that of Fig. 4.1 and the analogue values had to be transferred using the 64-bit wide analogue buses. This resulted in approximately 20% more silicon area. Fortunately, the limiting factor was the number of pads that required the chip to be a certain size.

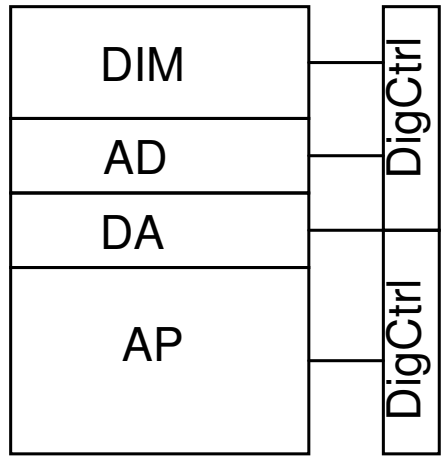


Figure 4.17 The original floor-plan of the chip.

In Table 4.2, the areas of the different processor parts are collected. The first is the size of the whole block and then comes the size of an individual device. There are differences between the sizes of an individual device if it is calculated from the block size. That is because the individual devices were designed for the pitch of the analogue processors.

Appendix A shows the implemented chip. The total size of the chip without pads is $2384 \times 852 \mu m^2$ resulting in $2.03 \mu m^2$.

BLOCK	TOTAL AREA (mm^2)	UNIT AREA (μm^2)
AP (Low-pass)	0.49	478
AP (Gradient)	0.081	421
DA	0.055	696
AD	0.212	2655
DIM (SRAM)	0.354 ($64 \times 56 \times 9$bits)	11 (1 bit)
DIM (sense amp)	0.054	837

Table 4.2 The layout size of the different blocks in the designs.

Chapter 5

Measurements of the Implemented Chips

Two chips were designed and measured to investigate the implementability of the proposed system. The first chip was a 4×16 network with 4×48 image memory. The main purpose of that chip was first to test the functionality of the Reduced Cell-row System and the interconnections between the analogue and digital parts. The design was made using a $0.25\mu m$ process. According to its measurements, a larger 64×16 network was designed with a 64×56 image memory. The aim of that chip was to test the large scale implementability of the design. This chip was designed using a $0.18\mu m$ process and therefore the whole design had to be re-dimensioned. In the first chip, the converters were designed to be 6-bits in accuracy when in the latter chip that was changed to the normal image processing accuracy of 8-bits.

In this section the measurement results are presented for both chips and also the encountered problems are presented. Due to the similarity of the designs, the measurement setup was similar for both chips and it is presented first.

5.1 Measurement setup

Both chips had an all-digital interface, meaning that the input image was loaded in digital form to the chip's image memory and the control of the processing was handled using digital signals. The only analogue controls were the bias currents for the converters and the threshold value to the gradient block. This made the control of the chip quite easy, but from the measurement point of view, this caused a problem in measuring the analogue parts, because there was no direct way to measure the analogue values. Therefore, the measurement of the supporting circuitry had to be conducted first.

The basic measurement setup is shown in Figure 5.1. The controlling signals and the input image are fed to the PCB-board using a computer-controlled pattern generator, shown as PatGen in the figure. The results of the processing are read to the logic analyzer, which is also computer controlled. It is pictured as LogAnalyzer in the figure. Since there were over 40 parallel control signals, they were first generated with ®Matlab and then imported to the pattern generator. The supply voltages and the current biases were delivered from DC-sources shown in the figure.

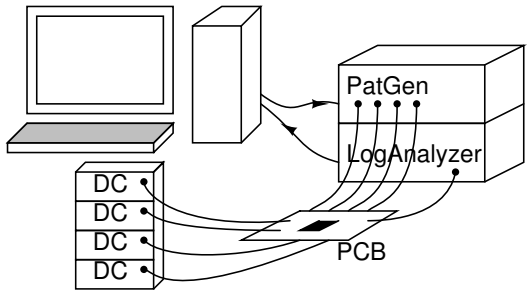


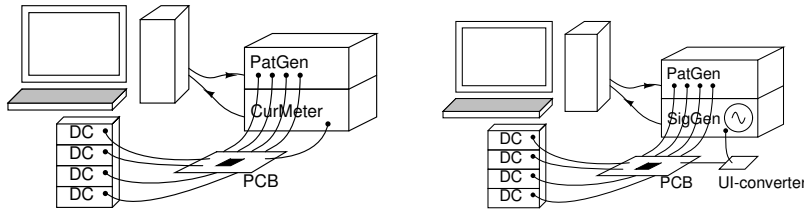
Figure 5.1 Basic setup in the measurements.

In the cases of measuring each of the DA- and AD-converters separately, the setups shown in figures 5.2(a) for DA and 5.2(b) for AD were used. In both measurements, the control signals were fed from the pattern generator. In the DA-measurement it was possible to measure each of the converters separately through a test pin. The input digital code to the converter to be measured was fed through the image memory. The output of the converter was steered with a 6-bit selection signal, which controlled switches inside the chip, to the test pin. From the test pin, the current was steered to a current meter (CurMeter).

When measuring the AD-converters, an AC current source was necessary to produce the input current to the converters. This was done by using a voltage signal source (SigGen) and a voltage-to-current converter (UI-converter) [54], which was built onto a separate PCB. The current from the UI-converter was fed through the same test pin as above to the converters. This time, the same 6-bit control was used in selecting the AD-converter.

5.2 4×48 Chip Measurements

The main purpose of the 4×48 chip was to be a proof of concept for the proposed system. Naturally, the accuracy of the processing was also of interest. However, during the measurements it became evident that the stability and the repeatability of the measurements was not sufficient for any accuracy measurements. There were quite large



(a) DA-measurement (b) AD-measurement

Figure 5.2 The converter measurement setups.

differences between two consecutive measurements. It turned out that the three-state buffer of the AD-converters driving the SRAM cells was not strong enough to change the state of the SRAM to one in all the occasions. This led to a random variation to the images that were read out of the chip. This can be seen in Figure 5.3, where the same simulation is repeated 2500 times. On the left side are the histograms of the different values of four cells in the same row of the network. The right hand side shows the consecutive outputs of the same cells.

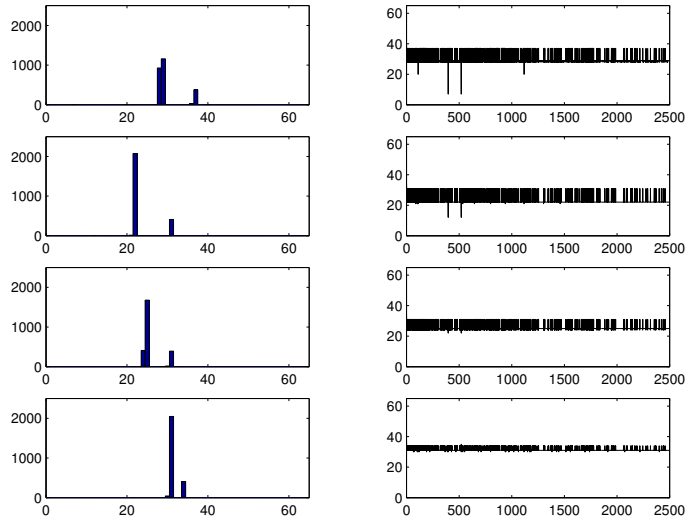


Figure 5.3 Measured variation of four cells in the same row.

However, it was possible to obtain the result that the Reduced Cell-row System itself was functional. Figure 5.4 shows a case where the input, shown in Fig. 5.4(a), is fed to the network. The ideal output is shown in Fig. 5.4(b) and the output of the measurement is shown in Fig. 5.4(c). In the measurement result, the maximum of each pixel was used.

As the figure shows, there are no discontinuation points visible where the network has been circularly connected. This can be seen more precisely from Figure 5.5, where

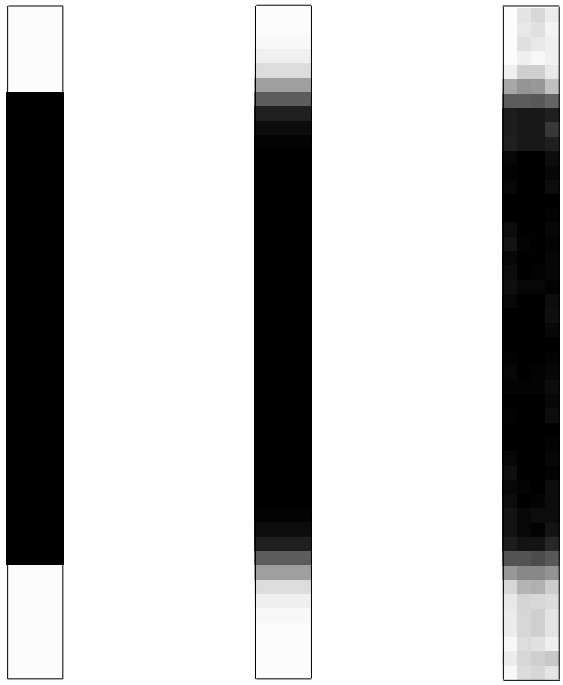


Figure 5.4 The measured output of the 4×48 network compared to the output of an ideal network.

the left column of Fig. 5.4(c) is shown.

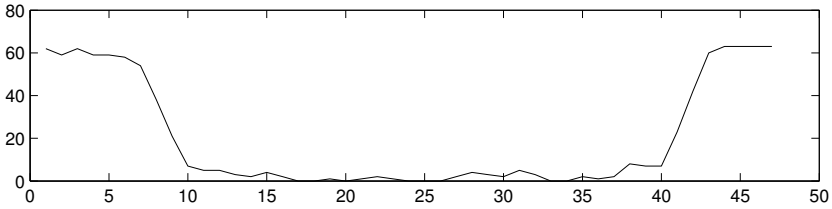


Figure 5.5 Output of the one image column after processing.

Because of the problems in reading out from the converters to the SRAM, it was not possible to make any accuracy or processing speed measurements.

Another thing that can be seen from the obtained results is seen in both Figures 5.3 and 5.4(c). There is a level drop in the central columns in comparison to the columns on the sides. No direct source of it can be pointed to, but it can be assumed that the matching between the cells in the middle and the cells on the edges and their bias circuitry was not sufficient. The difference was probably due to the design of the border cells, which consisted only the required mirror transistors to provide the zero-flux bor-

der condition. This was improved in the design of the 64×56 chip by introducing dummy transistors to the bias circuitry and by making the border cells from the actual cells.

5.2.1 Conclusions from the measurements

Even with all the shortcomings that were found in the measurements, the chips provided the main result they were designed for, i.e. the system itself was functional and worked as it was supposed to be. Naturally, all the information that was obtained from these measurements was used in the design of the 64×56 chip.

5.3 64×56 Chip Measurements

In the measurements, the main goal was to investigate the accuracy and the functionality of the analogue processing networks. Here the analysis is divided into two different parts, the low-pass filtering part (LP) and the gradient calculation part (GRAD). In both cases, it can be considered that there are independent error sources for the errors in the output. In the case of LP, the error sources are the DA-converters, the analogue network processor itself and the AD-converters. For the GRAD-calculation, similarly the sources are the LP-network, which work as the input to the GRAD-block, and the processor itself. In Fig. 5.6 the different error sources are shown for the low-pass filtering case.

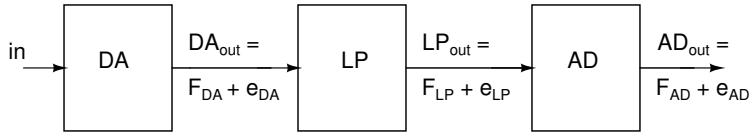


Figure 5.6 The different error sources in the processing chain.

In the figure, for each block, its output is shown as a sum of the ideal transfer function and a error function. Here we are mainly interested in the transfer and the error function of the LP block. Therefore the analysis of the converters had to be carried out before the performance of the cell array could be obtained. This way, by eliminating the errors in the conversions, it was possible to also analyse the errors in processing caused by the analogue array.

In this section, we start with the measurements of the digital-to-analogue (DA) converters, then moving on to analogue-to-digital (AD) converters. After analysing the results, we are able to get our hands on the errors in the analogue processors. The analogue processing part measurements were conducted using an offset that provided

the best results in the system-level measurements, namely 1.5μA. That point was not necessarily the optimum for the converter measurements, as also the results showed, because they were designed with 5μA offset. The dynamic range of the converters was the 10μA used in the system simulations; this was kept for all the measurements.

5.3.1 Measurement Results of the DA-converters

As mentioned in the previous chapter, the design of the DA-converters was based on binary weighted current sources. The measurements of the DA-converters were made so that each of the binary weighted sources for all the converters were measured separately and then all the possible codes calculated using ®Matlab.

To verify that the summation of the currents do not cause error, the calculated maximum output was compared to a measured maximum output. This is shown in Fig. 5.7, where the measured output of all the 64 column converters is shown in the same figure with the calculated sum of separately measured weighted bit outputs for one of the chips measured.

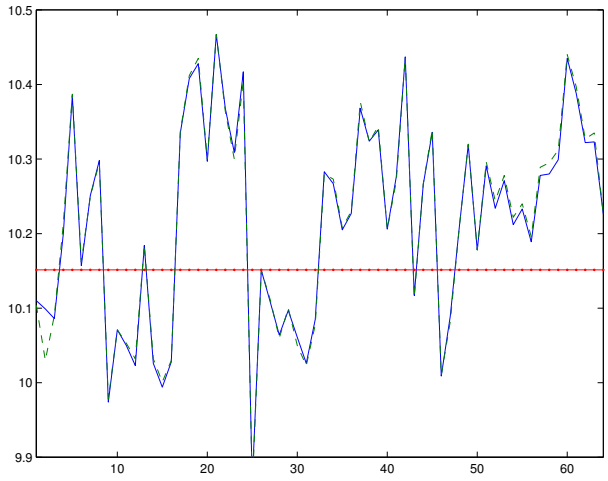


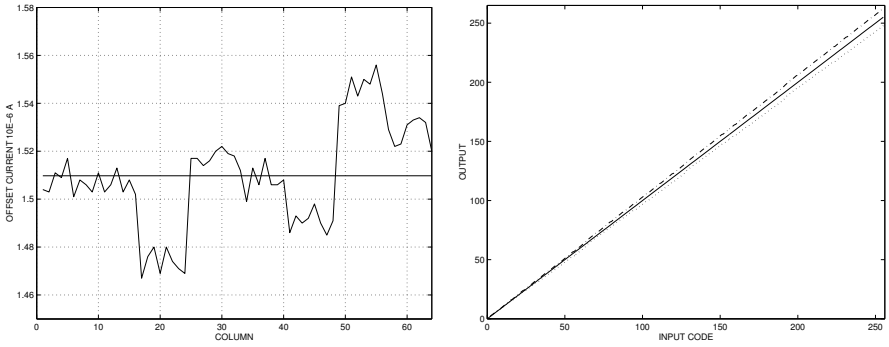
Figure 5.7 Calculated DA-converter outputs vs. measured maximum output.

The figure shows that error between the calculated maximum and measured one is negligible.

In the following, first the measurements of the offset and dynamic range are shown. After this the INL and DNL are calculated for each converter separately. Finally, the matching of the converters is considered by calculating the INL and DNL for all the converters using a common curve where the converter output is compared.

5.3.1.1 Offset and dynamic range

The system was designed to be operating on current levels above zero, resulting in all the DA-converter output values sharing the same offset. This offset was generated by mirroring the offset current to all the outputs of the converters. Naturally, this causes matching errors between the outputs due to the mismatch in the current mirrors. In these measurements the offset current $1.5\mu A$. The resulted offsets are shown in Fig. 5.8(a).



(a) Measured offset of the DA-converters. (b) DA-converter maximum outputs without offset.

Figure 5.8 Offset and dynamic range measurements of the DA-converters.

The construction of the offset distribution circuitry is clearly visible in the measurement results, where the level is same on the blocks of eight converters. The mean of the offset is also shown as the straight line in the figure. The value for it is $1.51\mu A$. The standard deviation of the outputs was $0.0214\mu A$, which is 0.6 LSB if the dynamic range is the used $10\mu A$. If the blocks of eight are considered the deviation inside a block is at the maximum $0.006\mu A$, which is 0.15 LSB.

The realisation of the distribution circuitry was chosen to avoid the errors caused by a possible voltage drop in the power supply voltage. The results show that it would have been more accurate if all the current mirrors would have had the same reference because there was no significant voltage drop in the power supply.

The dynamic range of the converters was produced similarly as the offset, as the reference current $2.5\mu A$ was used to produce the dynamic range of $10\mu A$. Figure 5.8(b) shows the measurement results of the converters producing the maximum and the minimum outputs without offset. The solid line shows the median curve. The plots are scaled to the common LSB, which was calculated using the median curve by setting the maximum output of it to respond to 255 LSB.

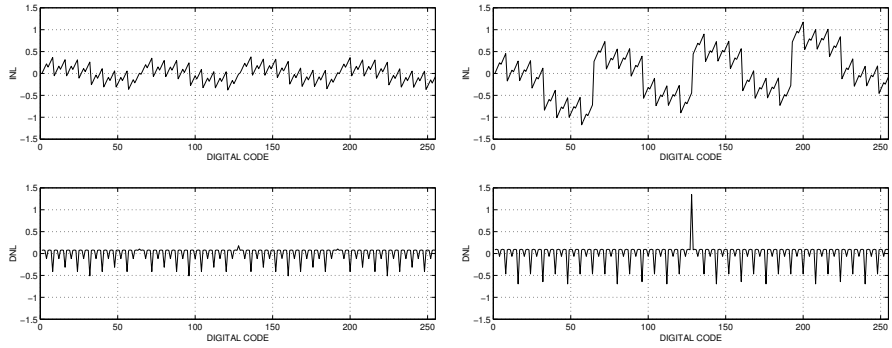
Differences as large as 7 LSB in both directions are visible in Fig. 5.8(b). If the measurements of the distribution circuitry are considered and the results are transferred

to dynamic range measurements, it can be concluded that the distribution circuitry can cause an error of 2-3 LSB. This is because the reference current is multiplied in the converters by two in order to form the current representing the MSB.

5.3.1.2 INL and DNL

For each of the converters the static Integral Nonlinearity (INL) and Dynamic Nonlinearity (DNL) [64] curves were calculated where the reference curve was defined for each converter separately. The results showed that 61 of the 64 converters were working with at least seven bits. The mean of the INL was 0.773 LSB with a standard deviation of 0.188 LSB. Similarly, the mean of the DNL was 0.788 LSB with a standard deviation of 0.186 LSB.

Figures 5.9(a) and 5.9(b) show the DNL and INL curves of the best case and worst case converters, respectively.



(a) Measured INL and DNL curves for the best column DA-converter.

(b) Measured INL and DNL curves for the worst column DA-converter.

Figure 5.9 INL and DNL curves of the best and the worst DA-converters.

5.3.1.3 Matching of the DA-converters

Since the column converters feed the image information to the analogue processor array, their individual accuracy is not of interest as much as their matching together. When analysing the column converters together a common ideal conversion result had to be defined. Here it was defined by minimising the absolute error in INL-curve for all the converters. This is because, as it was shown in last paragraph, individually all converters work reasonably well and the main cause of error between the converters is the gain. In practice the ideal curve was obtained by finding the LSB that resulted in minimum absolute error in INL when all the converters were concerned.

When the results were analysed using a common ideal curve and the DNL and INL were calculated using this curve as a reference, the results showed that, in the worst

case, the INL of the single converter was 7.38LSB. This results in an accuracy of 4 bits for all the converters combined. The INL and the DNL curves of the worst-case converters are shown in Fig. 5.10.

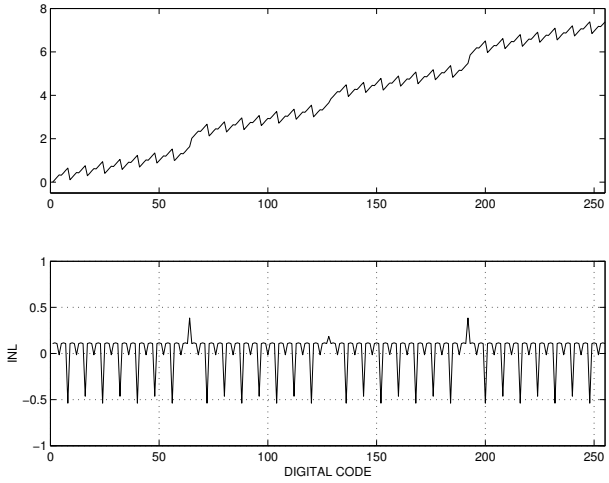


Figure 5.10 Measured INL curve for worst case converter when using common ideal curve.

As the plot of the INL curve shows, the error increases quite linearly as the digital code increases and since the DNL-curve is at worst just a little over $1/2$LSB, the origin of the error is the gain difference between ideal curve and the output of the measured column converter.

5.3.2 AD-converter Measurements

Each of the 64 column converters were measured separately. As with the DA-converters, the main interest of the measurements was the behaviour of the converters relative to each other rather than the absolute result of each converter separately. In order to get this information all the converters were measured separately using the same input.

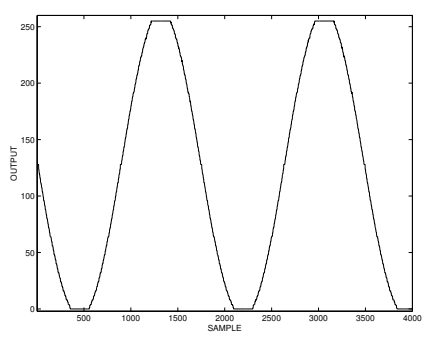
As the figures of merit the INL, DNL and Effective Number of Bits (ENOB) were calculated. These values were obtained by using the code density test (CDT) calculation defined in [65]. In addition to this, the offset level and value for the LSB for each converter were also calculated to be able to evaluate the differences between the converters. The different conversion speeds were also tested in order to find the optimum processing speed for the analogue part of the processor. This resulted from the fact that, for each analogue step, a conversion has to be made, and it turned out to be the bottleneck of the processing speed.

5.3.2.1 Calculation of the Figures of Merit

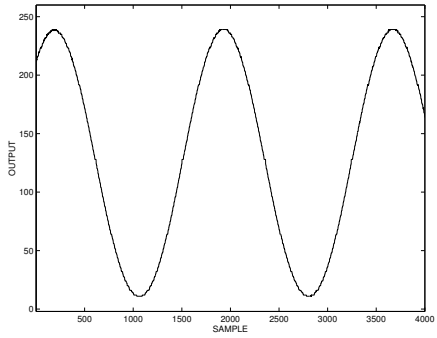
The calculation of the INL and DNL was based on a histogram method, also known as the code density test (CDT) method. This test is performed in the amplitude domain of a data converter. During the test, a repetitive sine-wave signal with a large-enough amplitude to make the output clip is applied to the converter, generating a corresponding distribution of digital codes at the output of the converter. Any deviation from the corresponding output code distribution results in various errors that may be estimated with the histogram method. DNL and INL are among those calculations.

When calculating the signal-to-noise-and-distortion ratio SNDR, and from there the effective number of bits, a similar setting was used; only the input signal was in the limits of the converter dynamic range.

The measured results of the two different sinusoidal input signals are shown in Fig. 5.11, whereas the Fig. 5.11(a) shows the distorted signal and Fig. 5.11(b) the clean signal.



(a) The distorted input signal for INL and DNL measurements.



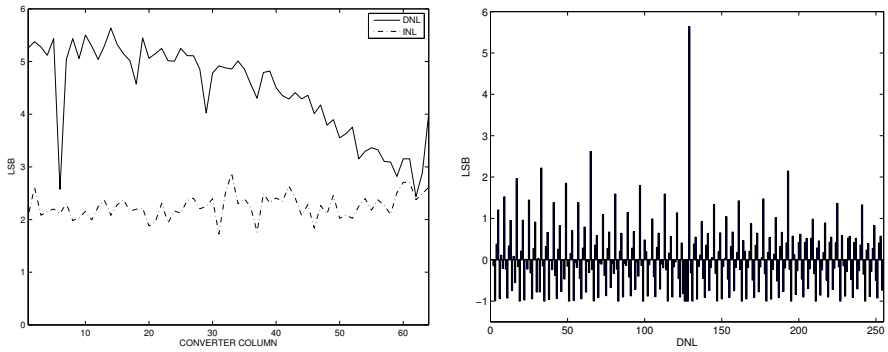
(b) The clean input signal for INL and DNL measurements.

Figure 5.11 The input signals that were used in the AD-converter measurements.

The used signal frequency was 200Hz because there was no sample-and-hold circuitry in the setup and the signal had to be constant during the conversion.

The maximum values of the INL and DNL are 2.880 and 5.64, respectively, in the worst-case column converters. What is interesting is that the DNL results especially are better on the other side of the converter row. This is shown also in Fig. 5.12(a), where the maximum values of the INL and DNL results are pictured as the function of the converter placement. The DNL curve of the worst-case converter is shown in Fig.5.12(b). The largest error is in the MSB transition point, and this is the case for all the converters. This could be due to the DA-converter of the AD-converter, which is the same as previously presented, if the MSB outputs of all converters were giving too small a current. However, the measurements of the DA-converters did not indicate any

systematic error in the MSB and therefore the systematic error could be caused by a drop in the power supply.



(a) The maximum values of DNL and INL as the function of placement on the chip.

(b) The worst-case DNL curve.

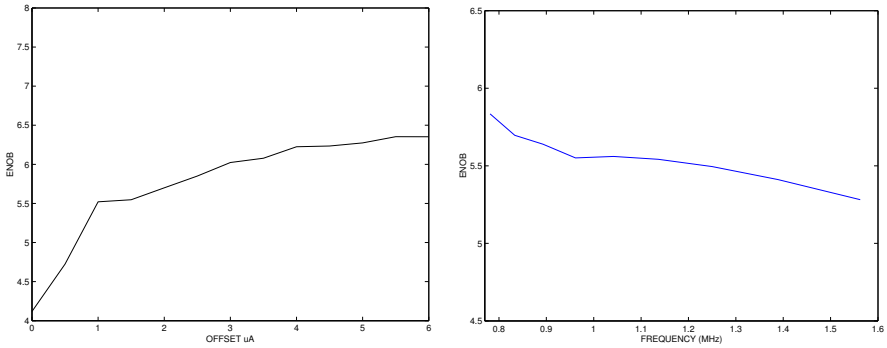
Figure 5.12 DNL and INL measurements of the AD-converters.

In the dynamic measurements, first the SNDR was calculated from the measured data. In this calculation, over 260000 samples were used; using these, an estimate of the signal was calculated with the 4-parameter fit [66] and using this, the result root mean square (rms) error was first calculated and from it the SNDR. Effective number of bits (ENOB) was obtained from the SNDR using Eq.5.1.

$$ENOB = \frac{SNDR - 1.72}{6.02} \tag{5.1}$$

When ENOB was calculated for each converter, the mean accuracy of the converters turned out to be 5.53 bits with a standard deviation of 0.09 bits in the dynamic measurements using the offset of $1.5\mu A$. However, since originally the converters were designed to work with $5\mu A$ offset, the effect of the offset was also tested for the ENOB. The results are shown in Figure 5.13(a). The figure shows that using the original $5\mu A$ offset it is possible to increase the ENOB by one bit.

Since the AD-conversion speed determines the speed of the processing, it is crucial to know the maximum speed of the conversion with a reasonable accuracy. In Section 4.3.4, it was shown that the conversion speed of the AD-converters is adjusted with a single global voltage (DELAY_BIAS) and the conversion time is controlled by a global signal CONVERT, which is a multiple of the clock frequency. Here these signals are independently tuneable and they have to be set to correspond to each other by observing the conversion results. This is achieved by setting the conversion time to the desired conversion speed and then speeding up the conversion by increasing the DELAY_BIAS-voltage until all the output bits change their values during multiple conversions. This is because, if the chain of delays do not reach the last bits, the LSB-



(a) ENOB as a function of offset.

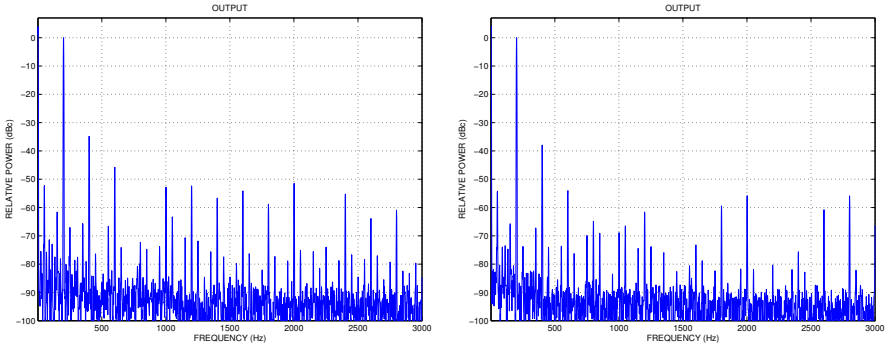
(b) ENOB as a function of conversion speed.

Figure 5.13 AD-converter measurements for one converter using different offsets and conversion speeds.

bit will remain constant during series of conversions.

Figure 5.13(b) shows the results of the speed test for one column converter. The figure shows that, under $1.3MHz$, the accuracy of the conversion is over 5.5 bits and after that, it starts to gradually decline.

Figures 5.14(a) and 5.14(b) show Spurious-Free Dynamic Range (SFDR) curves for the worst-case and best-case converters. The second harmonic is dominant in both cases as it is for all the measured converters.



(a) The best result for an AD-converter in SNDR measurements.

(b) The worst result for an AD-converter in SNDR measurements.

Figure 5.14 SNDR measurement results for two column converters

5.3.2.2 Offset and Dynamic Range Measurements

To be able to compare the converters between each other and to define the error caused by the mismatch between the converters is somewhat more difficult than with DA-

converters. This is mainly because of the measurement setup, where it was difficult to measure the performance of the designed UI-converter. To overcome this problem, the measurements were conducted so that first the offset and amplitude of the UI-converter current was set according the output of the AD-converters so that either distorted or clean output was obtained. Then, using these settings, all the converters were measured. Finally, maintaining the offset and bias settings of the AD-converters, a DC-current source was connected to the input pin and five linearly increasing DC-currents were fed to the input of one of the converters. This way it was possible to obtain five different input levels and their digital output values. Using the linear curve of these five points, the current corresponding to one LSB was possible to calculate. Finally, this current value could be used in calculating the input current signal in both distorted and clean cases.

When the input signal was known, it was possible to calculate the LSB for all the rest of the converters by using the information of the clean input current and the fact that, for sinusoidal signal, the derivative is at its minimum in the minimum or maximum of the amplitude, and that, therefore, the minimum and maximum codes are the ones that have the most 'hits' when looking at the conversion results.

Figure 5.15 shows the variation of the offsets relative to the median offset, which is shown as a dotted line in the figure. The results are normalised to the median LSB. The figure shows that the worst-case errors are around $+/-5LSB$. The standard deviation of the error is $2.3LSB$.

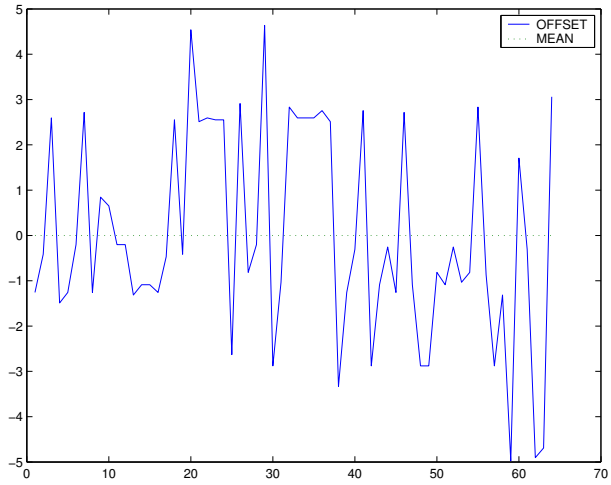
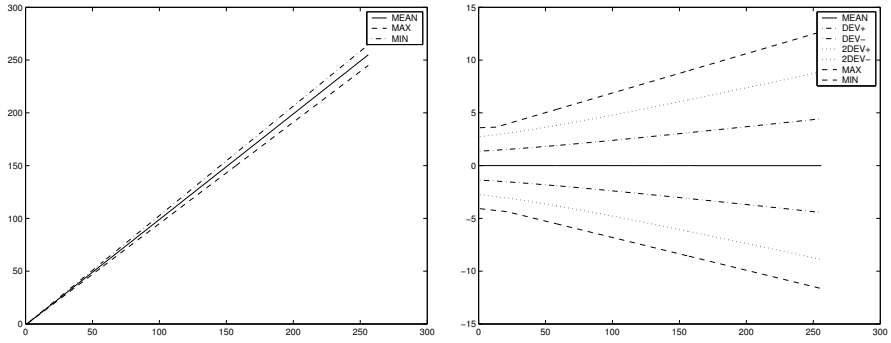


Figure 5.15 The variation of the offset, normalised to median of the LSB.

The second error source is the mismatch in the BIAS of the DA-converters (shown in Fig. 4.10). This results in differences in the dynamic range of the converters. In Fig.5.16(a), this error is shown by plotting the maximum, minimum and the mean

dynamic range, and results in errors as large as $+/-10LSB$ and with a deviation of $4.67LSB$



(a) The gain error of the AD-converters, normalised to median of the LSB.

(b) Offset and gain error combined, normalised to median of the LSB.

Figure 5.16 The effect of the errors in offset and gain in AD-converters.

In Figure 5.16(b) the above mentioned error sources are combined and the worst-case results are shown. The figure also shows the standard deviation of the error and two times the deviation under which should be the errors of 95% of the converters. The results show that the global accuracy of the AD-converters is a little over 4 bits, since the worst-case errors are less than 16 LSB.

5.3.3 Low-Pass Measurements

After analysing the converters, the low-pass and gradient blocks were to be measured. First the measurements of the low-pass block will be shown with and without the results obtained from the converter measurements. This way the error sources can be isolated and an estimate of the accuracy of the analogue network can be calculated. Finally, also the repeatability of the processing is considered and the behaviour of the chips in different phases of the processing is discussed.

There were four different input images used in these measurements. Those were the zero-level input, i.e. all black image, and all white image, i.e. input to all the cells was 255. Then an image where a black square, sized two-times-two pixels, was in all white background. This is shown in Fig. 5.17(a), where the black border was not part of the image but is shown for clarity. The fourth input image was part of one image in the Foreman sequence, used in many cases when analysing image or video processing algorithms. The used part is shown in Fig. 5.17(b). The latter two input images were also used as a sequence of images where the black square was moving or the video frames were changing.

In the measurements, both the errors during one processing round in one image and

(a) A single black dot in all white surrounding. (b) One frame of the Foreman sequence.

Figure 5.17 The image with single dot before and after correction.

errors when repeating the same input image several times were investigated. In the following, the measurements are divided into three parts, where in the first section the raw measurement results are shown and then, in the second section, the systematic errors are taken into account and cancelled out. Finally, the repeatability of the processing is considered by feeding the same input to the network and monitoring the differences between outputs of consecutive measurements in the output of the cells.

The measurements were conducted with the same frame rate of 4768 with I/O and frame rate 16276 when only the processing itself was taken into account. The limiting factor turned out to be the AD-conversion speed which therefore defined the used processing cycle time for one row.

5.3.3.1 Measurement Results without Correction

Already in the preliminary tests it was noticed that a mistake had been made in the layout of the border cells that provide the zero-flux environment for the cells on the edges of the grid. This had happened when changing the border cell used in the simulations to the actual border cell which is modified from the processor cell, including all the transistors of the actual cell with different connections. The error caused a division of the output of the edge cells; this error naturally spreads through the low-pass filtering array. The effect of the error can be seen in Fig. 5.18, where a constant level image is pictured.

The errors can be seen on the edges of the image as darker pixels that gradually become brighter towards the center of the image. The horizontal stripes that are repeated three times in the image are also caused by the border cells. Even though there was not supposed to be any connection between the edge cell and the border cells when

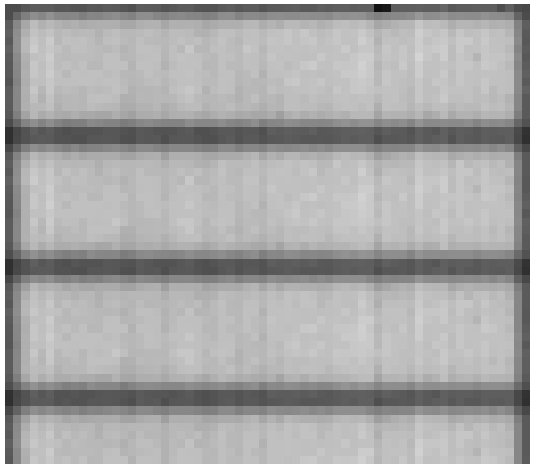


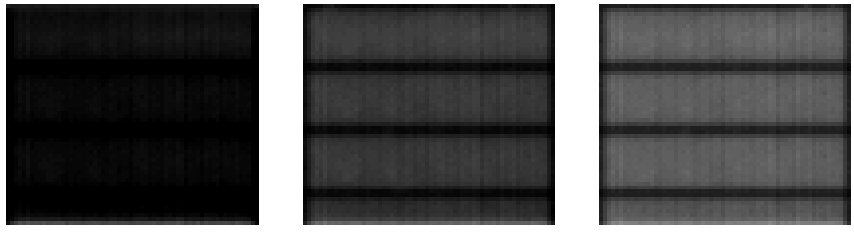
Figure 5.18 Output of constant level input to the network.

processing the middle parts of the image, the erroneous connection conducted part of the incoming currents to the A-template summing node to the border cell also in this phase of the processing. However, it was yet possible to analyse the accuracy of the network itself, as it will be later shown.

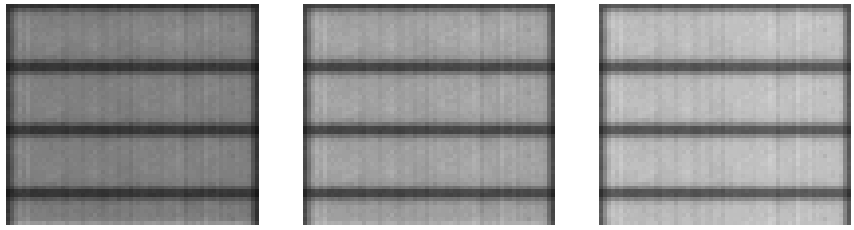
The analysis starts from the errors in a single image output. At the beginning, the output levels were measured using different constant input levels. It was chosen to be done with writing in the same digital code to each converters and, from there, to the low-pass cells. The used codes were from zero to 250 with 25 LSB steps. The outputs of these measurements are shown in Fig. 5.19, where the input level is shown under each result.

As the figure shows, the column-wise errors of the converters were left to the low-pass input. However, this error can be cancelled afterwards as a systematic error. The output for one cell in this measurement is shown in Fig. 5.20 for two different cells placed on opposite sides of the network.

The figure shows quite linear performance but a significant error in gain, causing the levels to drop compared to the input. Another measurement was performed to investigate if the reason for this was in the DA-converters. In this measurement, the input current level to the network was controlled with the DA-converter offset only. Then the results were compared to the results obtained from a level measurement, similar to the measurement above. The results are shown in Fig. 5.21, where the solid line shows the output of the level measurement and dashed line the same in the offset controlled measurement. The input and the output is shown in current; the output current values were obtained from the AD-converter measurements by transforming the digital code back to current using the measured LSB for the used column.

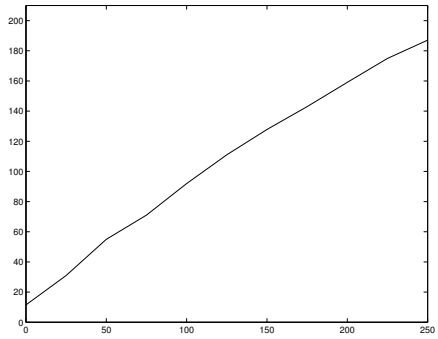


(a) 0 LSB (b) 50 LSB (c) 100 LSB

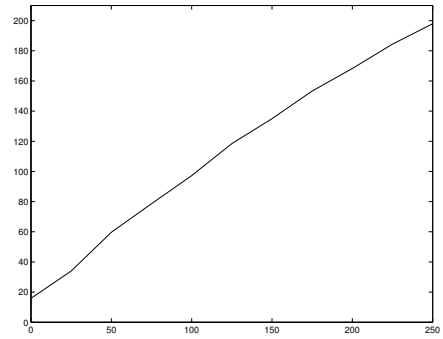


(d) 150 LSB (e) 200 LSB (f) 250 LSB

Figure 5.19 Output of the low-pass-network with different input levels



(a) Cell $C_{i,j}$, where $i = 3$ and $i = 7$



(b) Cell $C_{i,j}$, where $i = 3$ and $i = 55$

Figure 5.20

The figure shows similar behaviour in both cases; it can be concluded therefore that DA-converters are not the source of the error. Again, there was no direct way to measure the effect of the AD-converters, but since similar DA-converters, which determine the offset and gain, were used in the AD-converters, it was concluded that the network itself causes the gain error. The reason for it cannot be seen directly, but the result that was visible in all the measurements was that when the full network is not operating at the beginning of an image, the drop is not that significant. That indicates that the supply voltage limits the dynamic range. Another possibility is that the temperature inside the processor causes the gain error.

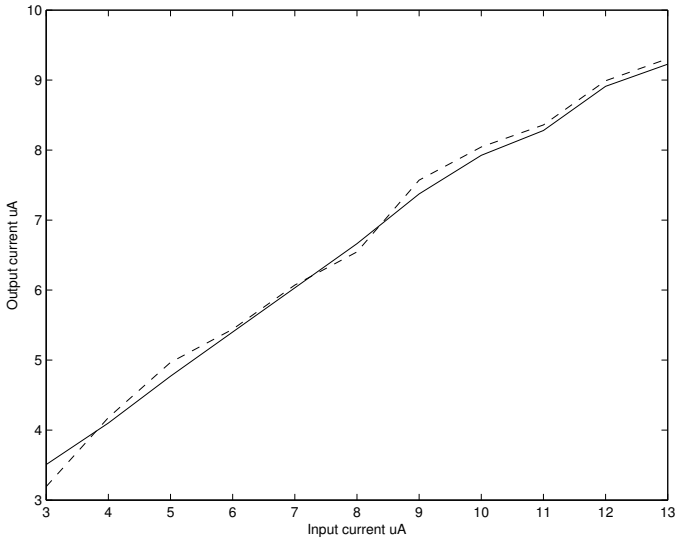


Figure 5.21 Outputs of the same cell when controlling the input with bias and offset currents.

5.3.3.2 Linear Correction of the Measurement Results

From the above measurements, it was possible to analyse the accuracy of the cells by limiting the analysis to only the central cells in the processor grid because they are not affected by the defective border cells. From these cells, it is possible to eliminate the systematic error caused by the converters. Since the effect of the low-pass filtering is not significant after four cells, the cell rows 5-12 and cell columns 5-61 can be used. This leaves us 448 cells for the accuracy analysis.

The calculation of the correction is based on the assumption that the systematic error of the column converters is linear and it can be separated to two parts, the offset and the gain. That can be assumed from the converter measurements where the main error source was either gain or offset. The idea was to calculate a column-wise gain correcting multiplying term and a subtraction term that should correct the column-wise offset errors.

The error correction terms are obtained by calculating first the mean of the all cells separately for different levels. After this, again for each cell, gain and offset correcting terms are calculated by fitting the outputs of the different levels to the corresponding input. Therefore, at this point we have for each cell two terms that correct the output gain and offset error. Now these terms are considered column-wise by calculating the mean of both the terms for each of the columns. This way we get the desired multiplying and subtraction terms for each column.

Figure 5.22 shows first the output of the cell-rows 7-12 without any correction. Then below are shown the outputs of the same cell-rows with the above described

correction to the full dynamic range. As it can be seen, the column-wise similarities are no longer visible and the output can be considered to represent the differences between the cells. In this figure, the columns 1-4 and 60-64 are also shown. However, because their outputs are affected by both the border cells on the sides of the network as well as the top and bottom border cells, these columns are left out from the accuracy analysis below.

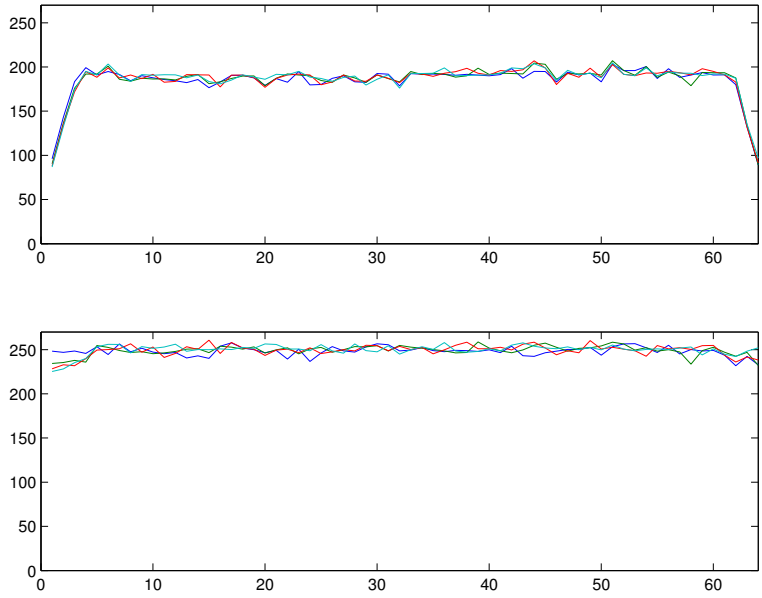


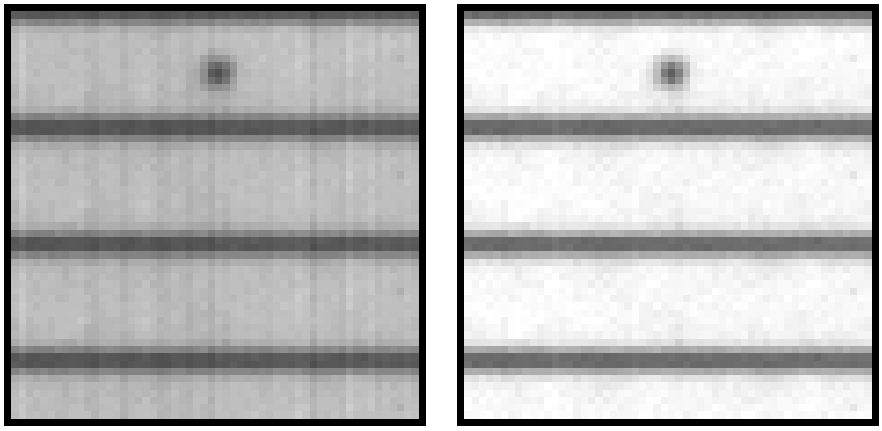
Figure 5.22 The outputs of the cell-rows 7-12 before and after linear correction.

In Figures 5.23 and 5.24, some examples of the processed images are shown together with the respective corrected results. The images are 56×56 pixels because the first and last four columns are not shown.

When the correction is employed to a constant-level image the standard deviation between the cells can be calculated with the 448 cells. This was performed for the image where the input was 250. The deviation of the original output image was 5.389 LSB with a mean value of 187 LSB; after correction, the deviation was 4.718 LSB and the mean value 247. Therefore, the accuracy of the cells can be considered to be somewhere four to five bits.

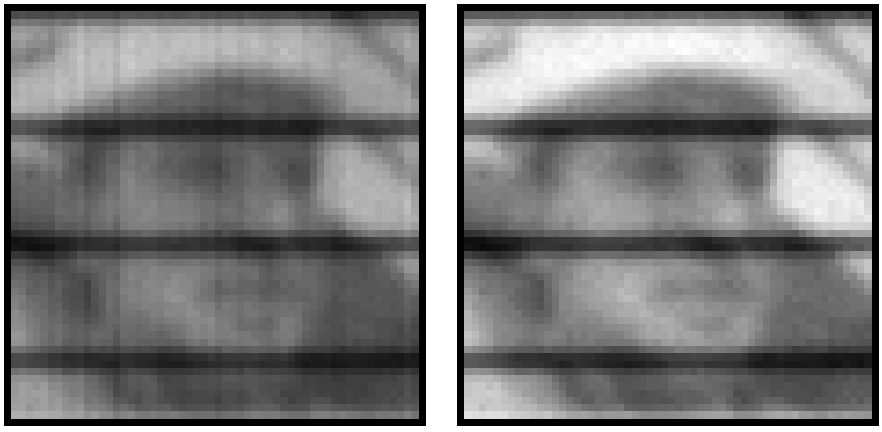
5.3.3.3 Repeatability of the Processing

The consistency of the low-pass network output was measured for two different cases with the same, maximum input. In the first case, the repeatability was investigated frame by frame and the deviation was calculated for each pixel in the above used 8×56



(a) Original image after processing. (b) Image after linear correction.

Figure 5.23 The image with single dot before and after correction.



(a) Original image after processing. (b) Image after linear correction.

Figure 5.24 Single frame in the Foreman sequence before and after correction.

sub-image, which was not affected by the border cells. The measurement was made by repeating the control code as long as the memory of the logic analyser, where the output was read, was full. This limited the number of consecutive frames to 146. Figure 5.25 shows the standard deviation of each of the pixels in the image. The pixels of the image are organised so that, starting from the left-most column of the processor, the deviation of the pixels is shown column-by-column.

As the figure shows, the deviation is, at worst, 2.0 LSB. When the variation of a single pixel output was more closely investigated, it was noticed that the outputs either kept constant or decreased, which was the case with most of the pixels. Figure 5.26

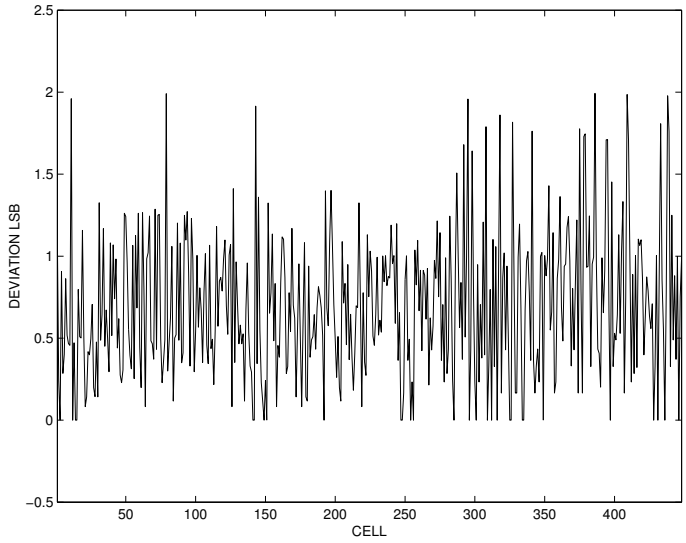


Figure 5.25 The deviation of each pixel of the sub-image

shows the outputs of four randomly picked pixels.

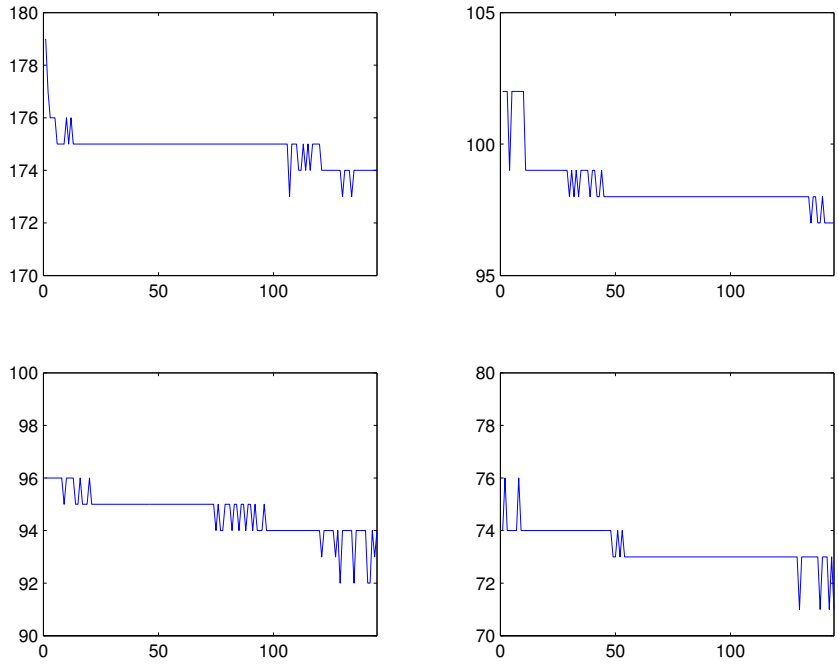


Figure 5.26 Four randomly chosen pixel outputs

At the maximum the drop was 9 LSB. The medium drop was 3.8 LSB. The reason for this phenomena is quite difficult to give, but one possible reason is the heating of

the chip during the processing. There was no cooling to the chip and the amount of current consumed was quite large compared to the size of the chip.

5.3.3.4 Differences Inside One Image

Since the processing is divided into blocks of 16 rows and there are differences in the physical implementation of the neighbourhood for the cells in the grid during processing of one image, it is of interest how much these differences effect the processing result. The differences are at their largest between the first 16 rows and the following blocks of 16 rows, because, when starting the processing, the first cell row is connected to the border cells and, when in the middle of the image, the first physical row is connected to the last physical cell-row.

For this measurement the image is divided into 16×64 blocks and the outputs of the pixels in these blocks are compared. Naturally, because the number of image rows is 56, the division results in three full-size blocks and one 8×64 block.

Figure 5.27 shows the differences of the mean values of the pixels between the blocks. The plot on the top of Fig. 5.27 shows the difference between the outputs of the first and second block pixel-by-pixel. The difference is scanned starting from the top left side of the image column-by-column. As expected, there are quite large differences. The main differences are between the first rows of the blocks; this is due to the totally different connections, as mentioned above.

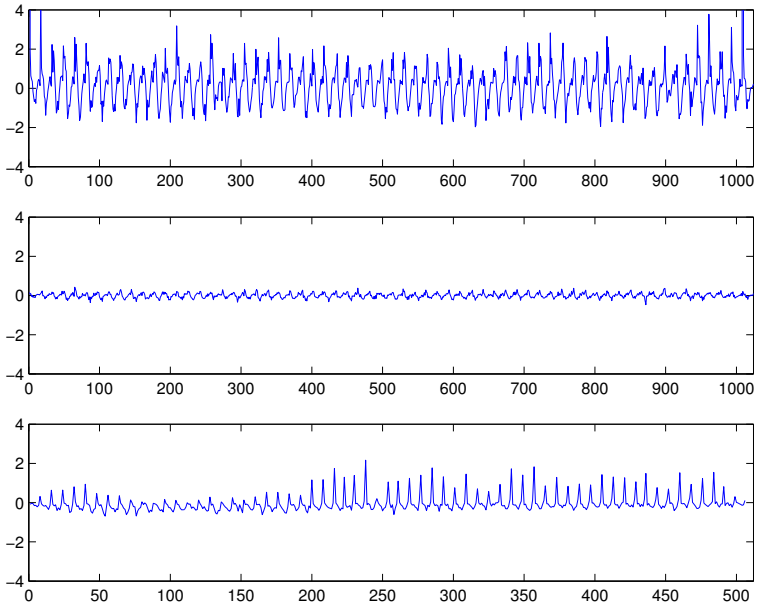


Figure 5.27 The differences between the outputs inside a constant image

The plot in the middle of Fig. 5.27 shows the difference between second and third block. Both of the blocks are in the middle of the image and they have exactly the same neighbourhood connections and the input. This results in only minor differences between the blocks.

The last of the plots shows the difference between the third and the last non-full block. There again the differences start increasing and the main differences occur with the last rows.

5.3.4 Gradient Measurements

As it was introduced in Section 4.3.2, the gradient block calculates the sum of absolute values of the difference between the central cell and its neighbours. Then this sum is compared a tuneable threshold value and if the sum is larger than the threshold value, it is considered that there is an edge.

Again there was no direct way to test the gradient block, instead the Low-pass (LP) network served as the input to the gradient-block. Therefore the errors in the LP also affect the gradient calculation results. In addition to that, the output from the LP-block is written to the input of the gradient block through current mirrors and an additional error is added to the input current of the gradient-block. If the actual calculation accuracy is to be measured, the exact input to the block should be available, but instead of that, the output of the low-pass block that is read out, is also affected by the AD-converters. However, in this section the gradient block is analysed the degree that it is possible with the given options.

In the measurements, an input image with different levels was used. In the ideal case, the gradient block should find the border cells where the level changes. Since the input image is smoothed with the low-pass filtering, the resulting border becomes thicker than if a non-filtered image was used. To obtain information on the accuracy of the calculation, the images were tested with different threshold values until no border was found. The input images used are shown in Fig. 5.28.

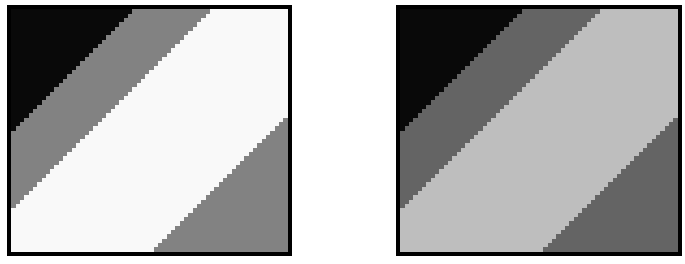


Figure 5.28 Input images to the gradient.

5.3.4.1 Measurement Results vs. Matlab Simulations

In the analysis, the non-corrected measurement results after low-pass filtering were used, despite the difference between this result and the analogue current value flowing to the gradient block. The results after processing the images in Fig.5.28 are shown in Fig.5.29.

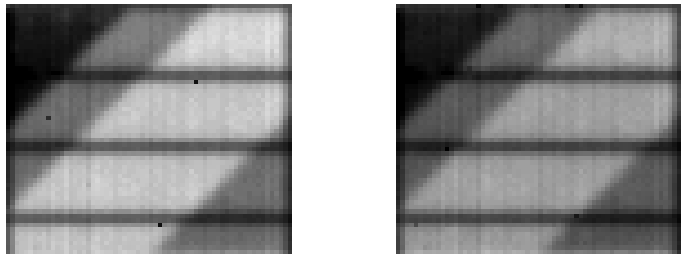


Figure 5.29 Measured outputs without any correction.

With these low-pass filtering results the output of the gradient calculation in an ideal case was calculated using ®Matlab. The threshold value for the simulations can be calculated from the current value used in the measurements with Equation 5.2, where the I_dyn is the dynamic range of the network and I_{tr_meas} is the threshold current used in the measurements. The tr_{255} is the threshold value for the simulation and 255 denotes just that 8-bit accuracy was used and the maximum of the dynamic range was 255.

$$tr_{255} = \frac{I_{tr\_meas}}{I_{dyn}} \cdot 255 \tag{5.2}$$

The measured result was then compared to the result of the ideal case with the measured low-pass result as its input. The comparison was made by calculating differences of the outputs when different tr_{255}-values were used. The calculation was achieved by performing a bit-wise XOR-function between the results.

In Figure 5.30, are shown the percentage of the correct pixels in the measurements compared with the calculated results with different threshold values. The comparison is made for the measurements where threshold values from $2.5\mu A$ to $5.5\mu A$ with $0.5\mu A$ steps were used. In the calculation, the threshold value tr_{255} was swept by steps that correspond to $0.1\mu A$'s. As it can be seen in the figure, the calculated output that gives the maximum of the correct pixels, the threshold value is close to the actual threshold current used in the measurement.

Figure 5.31 shows the measured output, the calculated output and their difference, respectively from left to right.

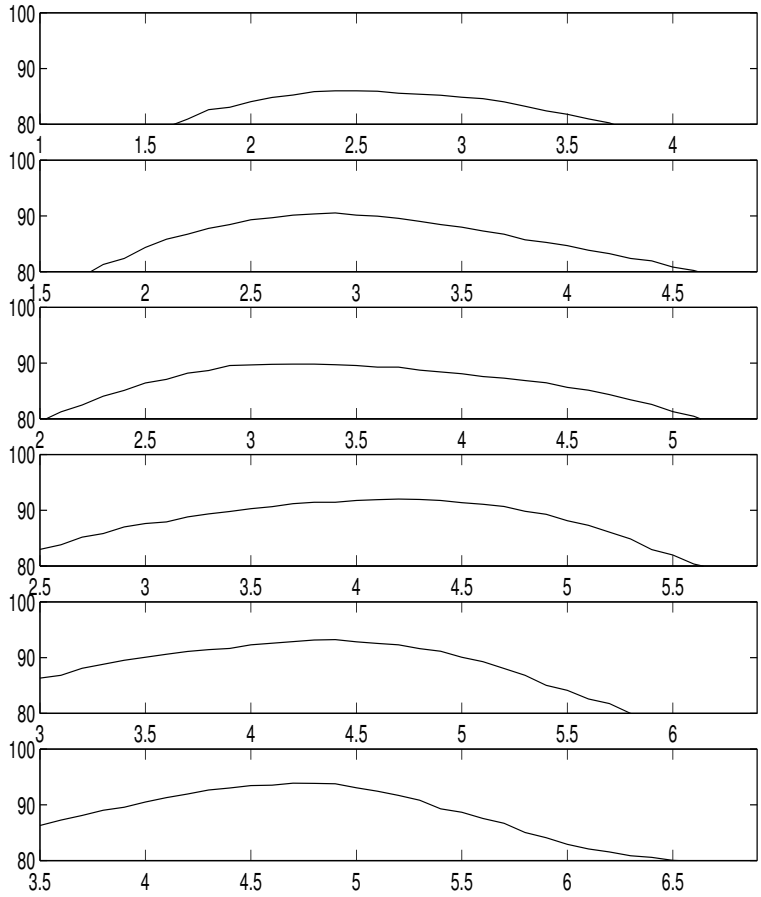


Figure 5.30 Percentage of correct pixels as a function of simulated threshold.

Figure 5.31 The measured and the calculated outputs and their difference.

5.3.5 Power Consumption of the Chip

The power consumption is one of most interesting figures of merit. This is especially interesting in this case because, as was shown in Section 2.2.1, the proposed system theoretically consumes considerably more power than an array processor that has a traditionally divided processing task.

With the implemented chips, it was possible to measure separately the power consumption of the analogue processor arrays together and the power consumption of the

DA- and AD-converters and the digital part. Again in these measurements the processing speed was 4768 f/s. This was obtained with the digital part working with a 62.5 MHz clock signal, while the analogue part was controlled with a 120 times 8 ns signal, resulting in a 0.96μs cycle for each image row. Since the processing takes, in this case, 64 cycles, the internal processing speed becomes the before-mentioned 16276 f/s.

There were three different supply voltages on the chip. The analogue parts had one common supply voltage and that was in the measurements 2.25 V, which meant that the suggested maximum voltage of the process was exceeded by 0.15V. The digital parts were working on a 2.1 V supply voltage and SRAM had its own supply voltage of 1.2 V. The results of the power consumption measurements are collected in Table 5.1. The figures shown for the analogue parts and the converters are those measured while processing, and those shown for the digital parts are those measured while writing in and reading out. The total power consumption is calculated from these figures by averaging them with the time the parts are active during the processing.

BLOCK	POWER (mW)
Low-pass + Gradient	119.6
DA-converters	2.4
AD-converters	16.3
Bias circuits for AD and DA	9.9
SRAM	0.3
I/O	3.4
TOTAL POWER @ 4768 f/s	62.9

Table 5.1 The power consumption of the different blocks in the 56×64 designs.

To give some perspective, in Table 5.2 the power consumption of the implemented chip is shown and then the power consumption if a CIF-size chip were implemented with the same realisation. In this case, the numbers are given if the required speed were the normal video processing speed of 30 f/s. In the case of CIF-size chip, the increased processing time is also taken into account. Because the used serial mode I/O is not feasible for CIF-size images, here it is assumed that the image is loaded already to the image memory and the internal processing speed is used in the calculations.

As the numbers show, the power consumption is yet reasonable small, even when considering the chip to be used in a system where the power is supplied from a battery. If the silicon area is considered also, the CIF-size chip would require approximately 4.6mm^2 for the analogue part including the converters.

	Implemented 64×56 (mW)	CIF (352×288) (mW)
Low-pass + Gradient	0.22	5.6
DA-converters	0.0044	0.112
AD-converters	0.030	0.76
AD/DA BIAS	0.018	0.46
TOTAL POWER @ 30 f/s	0.2724	6.93

Table 5.2 The power consumption if a CIF-size processor was implemented using realisation similar to that of the 56×64 design.

This page is intentionally left blank.

Chapter 6

Design of a Programmable-λ Network

The results in Chapter 2 showed that it is possible to obtain different λ-values for a resistive network processor simply by changing the CNN-template. For such a network there are several possible applications starting from image-size-dependent low-pass filtering to image analysis, which was presented in Section 2.8. However, the implemented CNN-UM chips have not been shown to be capable of such a task and therefore a special purpose processor structure may be feasible. In this chapter the transistor level design and system level simulations are given for such a processor.

6.1 Realisation of a Variable-λ Cell

The measured circuit and the Reduced Cell-row System can be used as a starting point for the design. Figure 6.1 shows the different building blocks of a variable-λ -cell. When it is compared to Fig. 4.3 in Chapter 4, the connections from the B-template are obviously missing and the λ-block is introduced.

For the B-template part, the design of the λ-block is quite straightforward since the incoming input is just multiplied by λ. If we have an input stage similar to that of the previous implementation, the λ can be realised using the circuit that is shown in Figure 6.2, where the different λ-values are obtained either by dividing or multiplying the input current. With the shown circuit, it is possible to obtain λ-values of 0.25, 0.33, 0.5, 0.66, 1 and 2.

For the A-template part, the realisation is somewhat more complicated because of the constant value of 3 in the feedback term. Figure 6.3 shows one possible realisation of the A-template.

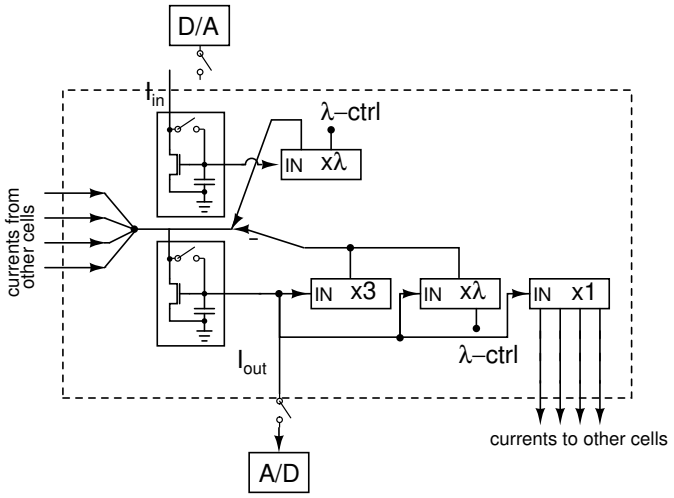


Figure 6.1 Block diagram of a variable λ cell

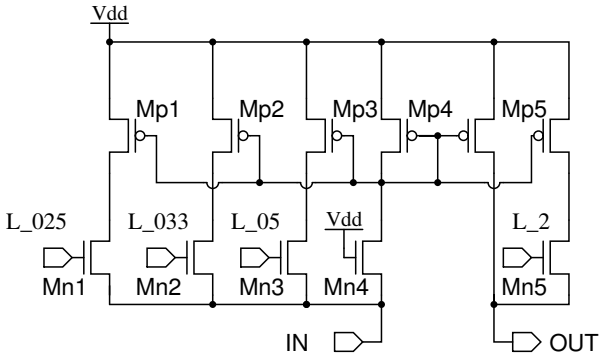


Figure 6.2 Realisation of the B-template

In the basic configuration, the λ is one, and from that value a current is subtracted to obtain the values smaller than 1. This subtraction current is formed with the PMOS transistors $Mp1$-$Mp7$ and switches $x1$-$x3$ and $d2$-$d4$. For the value 2, the transistor $Mn6$ is connected in parallel to the $Mn5$ using the switches $\overline{SW\_\lambda}$ and SW_λ. The switching configurations of all the switches to form the different λ-values are shown in Table 6.1.

An other way to implement the A-template part is simply by changing the W/L-value of the transistors. In this case, three transistors the same size as the mirror transistor would be needed to form the -3 term. In addition to that, the implementation would need 8 transistors, quarter of the size of the mirror transistor, that would be used in realising the λ-values 0.25, 0.5, 1 and 2, and then two transistors one third of the size of the mirror to form the λ-values 0.33, 0.66. However, an implementation will

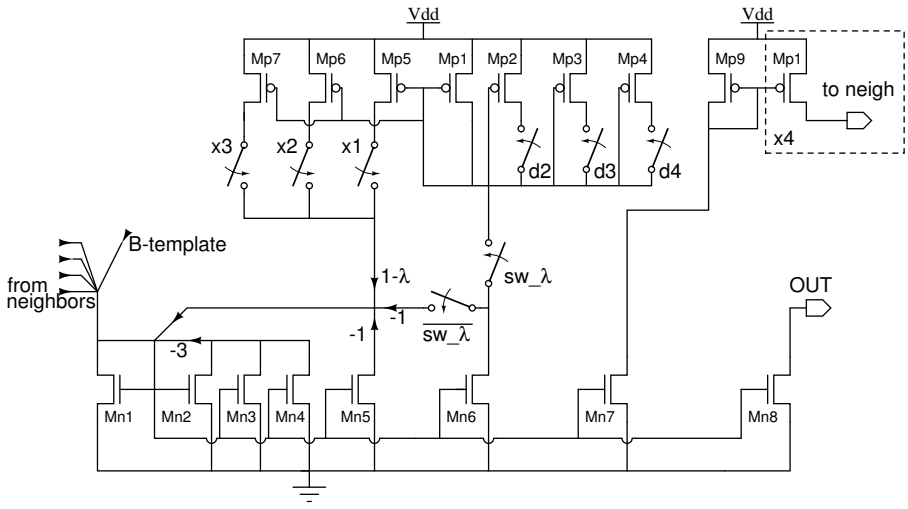


Figure 6.3 Block diagram of the A-template of a variable λ cell.

	sw_λ	$\overline{sw\_\lambda}$	$d2$	$d3$	$d4$	$x1$	$x2$	$x3$
$\lambda = \frac{1}{4}$	on	off	on	on	on	on	on	on
$\lambda = \frac{1}{3}$	on	off	on	on	off	on	on	off
$\lambda = \frac{1}{2}$	on	off	on	off	off	on	off	off
$\lambda = \frac{2}{3}$	on	off	on	on	off	on	off	off
$\lambda = 1$	on	off	off	off	off	off	off	off
$\lambda = 2$	off	on	off	off	off	off	off	off

Table 6.1 The switching configurations to form the required λ-values

be shown here where only 6 transistors, one quarter of the size of the mirror transistor, will be used. In Figure 6.4, the basic configuration is shown.

To clarify the functionality of the configuration, the circuits inside of the two blocks used in Fig. 6.4, namely MIRROR and FRACTION, are presented. MIRROR is shown in Fig. 6.5. The block consists of four identical transistors $Mn1$-$Mn4$ with gates and sources connected together. The drains of the three transistors through the dummy transistors $Md1$-$Md3$ and the drain of the transistor $Mn4$ through switch $Msw1$ are also connected together to node IN/OUT. When MIRROR-block is used as the block $B1$ as in Fig. 6.4, and where the connection results in the transistors being diode connected, if a current is fed to node IN/OUT it is divided either by three or four, depending on the state of the switch. This results in a voltage V_G that is distributed to the other MIRROR blocks and to the FRACTION-block. When considering the other MIRROR-blocks, if a voltage is applied to the gate node Vg it causes a current that is either three or four times the unity current of each branch in the block, assuming that the transistors remain in the saturation region.

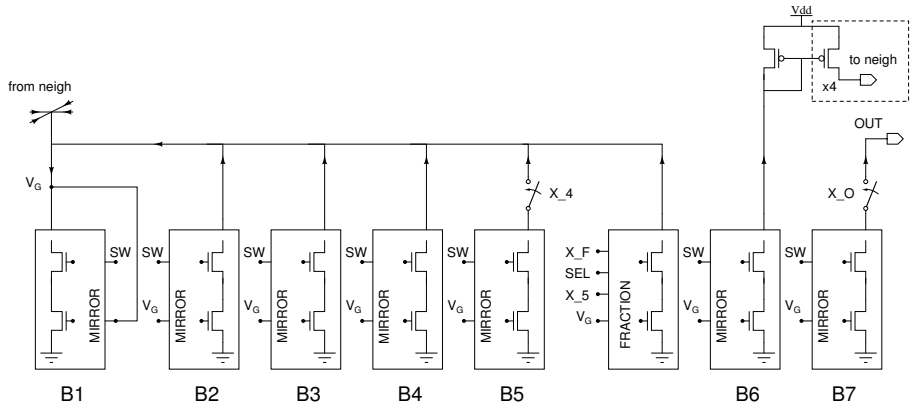


Figure 6.4 Another method to form an A-template realising circuitry.

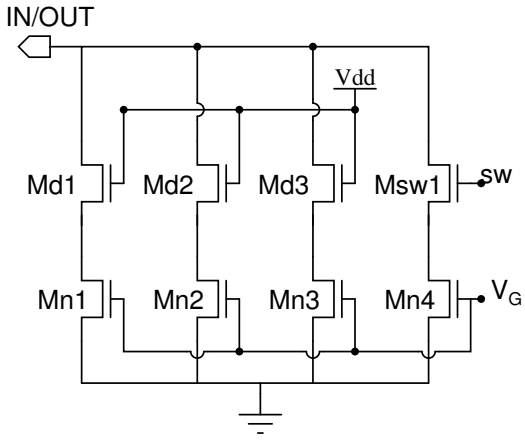


Figure 6.5 MIRROR-block of Fig. 6.4.

The FRACTION-block, shown in Fig.6.6 is similar to the MIRROR, only the number of branches is three. Here the transistors $Mn1$-$Mn3$ are identical to the transistors $Mn1$-$Mn4$ of the MIRROR-block. Using the switches SEL, X_F and X_5, different λ-values can be obtained.

As an example, to get λ-value 0.25, the central term of the A-template is required to be -3.25. In order to obtain this, we set the *sw*-voltage high so that the switches are conducting in all the MIRROR-blocks. This causes the current coming to *from neigh*-node to be divided by four and the same unity current flows through all the four branches of the input MIRROR-block and sets the gate voltage Vg to a certain value. This gate voltage is distributed to all blocks, in the rest of the MIRROR-blocks it causes the same current to the output as it flows to the first block. Therefore, blocks $B2$-$B4$ cause the constant term -3 of the central value. The switch X_4 at the output of $B5$ is in non-conducting mode and no current flows to the sum node from that block.

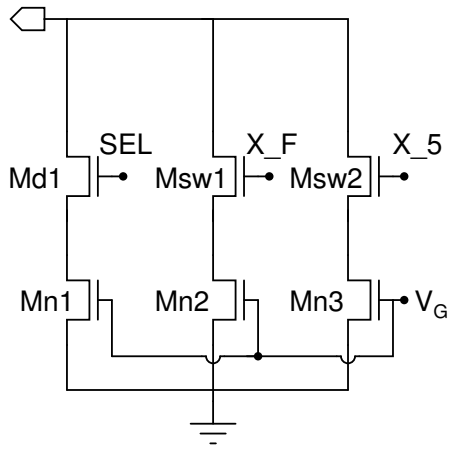


Figure 6.6 FRACTION-block of Fig. 6.4.

At the same time, the switch SEL of the FRACTION-block is also in non-conducting mode and the voltage Vg causes the output current of the FRACTION-block to be one fourth of the original incoming current to the B1-block. This way the desired output is reached.

In order to obtain a λ-value of 0.33, the procedure is identical, exept that the switches controlled by voltage *SW* are not conducting. The values 0.5 and 0.66 can, in turn, be obtained from the previous values by setting switch *X_F* in FRACTION-block in conducting mode. Finally, the value 2 is reached by setting *SW* off and *SEL*, *X_F* and *X_5* on along with the switch X_4. Again, the all the switching configurations are collected in Table 6.2.

	SW	*SEL*	*X_4*	*X_F*	*X_5*
$\lambda = \frac{1}{4}$	on	off	off	on	off
$\lambda = \frac{1}{3}$	off	off	off	on	off
$\lambda = \frac{1}{2}$	on	on	off	on	off
$\lambda = \frac{2}{3}$	off	on	off	on	off
$\lambda = 1$	on	off	on	off	off
$\lambda = 2$	off	on	off	off	on

Table 6.2 The switching configurations to form the required λ-values with the second realisation.

In the following both methods are simulated at the system-level and the effect of the transistor mismatch and the area of the implementation is investigated. First, however, the system-level simulation method is presented.

6.2 System Simulations of the Networks

As the measurement results showed the mismatch simulations that were completed for one cell were not sufficient. Therefore, in order to obtain system level results, a new method was used in system-level mismatch simulations. The goal was to be able to simulate the effect of the errors inside each cell on the processing results for differently sized transistors. The procedure was as follows:

1. Monte Carlo -simulations for differently sized transistors with a circuit simulator.
2. Read the Monte Carlo results to ®Matlab.
3. From the results, calculate the variance of the transistors.
4. For each cell and each mirror transistor inside a cell, calculate a mismatch affected value using the obtained variance.
5. Using the transistor values, calculate the effect of the mismatch to the template values.
6. Process the input figure using the mismatch affected templates.

The PMOS and NMOS transistors were simulated using the *Monte Carlo*-method for five different size transistors using the mismatch parameters given by the foundry. These parameter values were the accurate model parameters that were not available at the time of designing the chip presented in Section 5.3. The size of the active area of the transistors was always doubled when moving to a larger value and the W/L-ratio was kept constant. The smallest active area was half of the area of the transistors that were used in the measured implementation. This way, it is possible to get a rough estimate of the required area of the cell and information on the area/accuracy-ratio also.

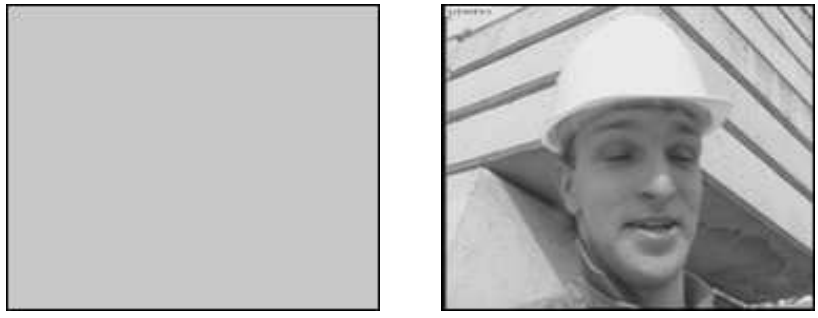
After the Monte Carlo results were calculated, the results were imported to ®Matlab, where the standard deviation of the simulation results was calculated. Using this value for each individual transistor in the current mirrors, a transistor mismatch value was calculated. This way, for transistors of each size, a certain deviation value was obtained to be used in the ®Matlab simulations. For simplicity, it was assumed that the mismatch is constant in the dynamic range of operation. Using the transistor values in different λ-configurations, it was possible to obtain the template sets that had a mismatch included. Because the same transistor values were used in the template calculations, when using two different λ-values the same mismatches affect the output as they would in a real silicon implementation. This makes it possible to also simulate the effect of the mismatch on the calculation of Difference of Gaussians, presented in Chapter 2.8.

The two methods that were presented in the previous section were simulated in the above mentioned manner for their suitability to a silicon implementation. For the simulations to be comparable, the size of the NMOS-transistors in the latter method of A-template implementation was chosen so that the size of a unity transistor was one fourth of the size of the NMOS-transistor in the first method. Therefore the variance of the transistor in the second method is twice the variance of the NMOS in the first method. This way, it was possible to maintain the comparability of the silicon areas of the two implementations.

To take the Reduced Cell-row System into account, the templates were calculated first for the 16 rows and then copied to comprise the full image size processor.

6.2.1 Simulation Setup

The simulations were conducted using two QCIF-size input images. The first image was a constant-level image where the input grey-scale value was 200 in an 8-bit system. This image was used in simulating the error the mismatch causes to the preservation of the input level. In the second case, an image from the widely used Foreman-video sequence was used. With that image, it is possible to investigate the effect on the algorithms where the difference between the processed images are used. Both images are shown in Figure 6.7. The images are referred to here as LEVEL and FOREMAN shown in Figures 6.7(a) and 6.7(b) respectively.



(a) LEVEL (b) FOREMAN

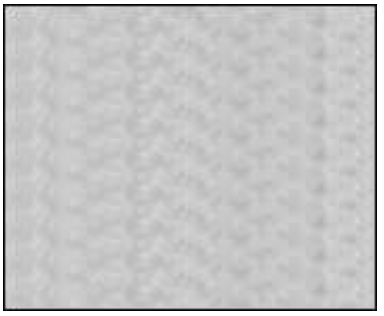
Figure 6.7 Simulation input images.

The actual simulations were made so that all the combinations of the different transistor sizes were calculated for both systems and for both input images. At the same time, the size of the implementation was also calculated for both systems and also for the implemented network that was described in the previous chapters. In the calculation of the area, only the active area of the analogue transistors was included. Naturally, this way it is only possible to compare areas of the two methods relatively. The quality of

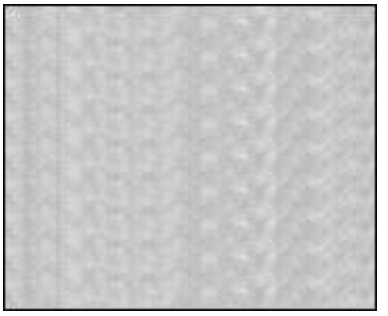
the processed images is observed here both objectively and by calculating the *Mean Square Error (MSE)* of the result with the ideal processing result. To compare the results with the measured chip of Section 5.3, also the standard deviation of the output is calculated for the case when $\lambda = 1$ and the input image is LEVEL. The realisation shown in Fig. 6.3 is denoted here as REAL1 and the alternative realisation shown in Fig. 6.4 in turn as REAL2.

6.2.2 Simulation Results

First the effect in the worst case situation is shown. Fig. 6.8 shows the results of both the systems in a simulation when $\lambda = 0.25$ and the transistors are the smallest used in the simulations. The input image here is the LEVEL image.



(a) Output of the simulation for LEVEL-image when using smallest size transistors and λ=0.25 and REAL1.



(b) Output of the simulation for LEVEL-image when using smallest size transistors and λ=0.25 and REAL2.

Figure 6.8 Simulation results

As the result shows, the mismatch causes patterns to the output image. These patterns are repeated every 16 rows due to the Reduced Cell-row System. When the FOREMAN is used as the input, the result shows the same patterns in the output, as it can be seen from Figures 6.9(a) and 6.9(b).

The MSE-values resulting from the worst-case settings can be seen in the first column of Table 6.3. For the REAL1 method, the MSE is 29.1 LSB's for the LEVEL input and for FOREMAN 20.7. Similarly for the REAL2 method, the MSE values are 41.1 LSB's and 29.5 LSB's, respectively, for the LEVEL and FOREMAN images. Table 6.3 shows the resulted MSE results of all the different λ-values with the input images LEVEL and FOREMAN. As the results, show the error is strongly dependent on the used lambda value; as the λ increases, the MSE-value decreases. This is due to the method of implementation, as it was investigated in [67], where it was shown that in transconductor or current mirror approaches the effect of mismatch increases as λ decreases in contrast to a true resistive network approach, where the effect decreases

(a) Output of the simulation for FOREMAN-image when using smallest size transistors and λ=0.25 and REAL1.

(b) Output of the simulation for FOREMAN-image when using smallest size transistors and λ=0.25 and REAL2.

Figure 6.9 Simulation results

with λ. However, there are two exceptions to this rule; these occur when using the REAL2 realisation and in the cases where $\lambda = 1/3$ or $\lambda = 2/3$. This is because there are only three active transistors in the MIRROR blocks and the effective active area of the transistor is therefore decreased.

	$\lambda = \frac{1}{4}$	$\lambda = \frac{1}{3}$	$\lambda = \frac{1}{2}$	$\lambda = \frac{2}{3}$	$\lambda = 1$	$\lambda = 2$
REAL1 LEVEL	29.07	22.98	16.38	12.85	9.43	6.90
REAL2 LEVEL	41.12	43.60	26.05	28.33	18.44	14.03
REAL1 FOREMAN	20.75	16.31	11.71	9.23	6.77	4.98
REAL2 FOREMAN	29.52	31.92	18.74	20.76	13.35	10.16

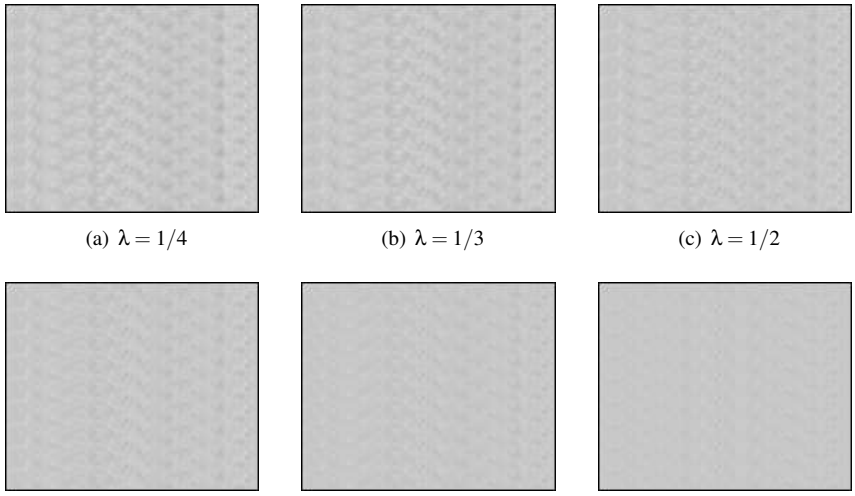
Table 6.3 Effect of the λ-value on MSE

The error and its visibility is also dependent on the used λ-value through the ROI-value. Figure 6.10 shows the outputs of the simulations for all six lambda values that were used here in the simulations. The corresponding λ-value is given in the caption of each sub-figure. The pictured simulations are made using REAL1.

The objective results follow the results obtained from the MSE-calculation; for the largest λ-values the errors are quite invisible.

6.2.2.1 Optimising the Transistor Sizes

Because each transistor is modelled separately in the simulation it is possible to investigate the effect of the transistor sizes for NMOS and PMOS separately. In both realisations, the NMOS transistors form the summing parts of the cell and the PMOS transistors are mainly used in the interconnections to neighbouring cells. As shown in Table 6.3, the λ-value 0.25 causes the largest values for the *MSE* and therefore it can be considered the worst-case situation. Tables 6.4 and 6.5 show the *MSE*-values of all



(a) $\lambda = 1/4$ (b) $\lambda = 1/3$ (c) $\lambda = 1/2$

(d) $\lambda = 2/3$ (e) $\lambda = 1$ (f) $\lambda = 2$

Figure 6.10 Processing the LEVEL image using different λ-values.

the simulated combinations of the transistor sizes when the input image is the LEVEL with both implementations. On each row of the table, the size of the PMOS-transistor is kept constant, and similarly on each column the NMOS-transistor size is constant. In the tables, the transistor sizes are marked with PMOS(#) and NMOS(#) where the # is the size of the transistor relative to the transistor size used in the measured chip.

	$PMOS(\frac{1}{2})$	$PMOS(1)$	$PMOS(2)$	$PMOS(4)$	$PMOS(8)$
$NMOS(\frac{1}{2})$	29.07	29.28	28.87	32.59	28.11
$NMOS(1)$	22.24	26.02	22.21	21.80	18.11
$NMOS(2)$	11.24	9.52	10.59	9.58	9.62
$NMOS(4)$	7.39	5.51	4.08	4.87	3.83
$NMOS(8)$	4.34	2.32	2.50	2.56	2.35

Table 6.4 *MSE* of the REAL1 with different size transistors.

	$PMOS(\frac{1}{2})$	$PMOS(1)$	$PMOS(2)$	$PMOS(4)$	$PMOS(8)$
$NMOS(\frac{1}{2})$	41.12	41.62	42.81	34.40	41.12
$NMOS(1)$	29.01	27.63	27.08	32.30	28.57
$NMOS(2)$	12.90	13.74	13.73	12.18	10.61
$NMOS(4)$	7.27	6.03	5.97	5.04	5.41
$NMOS(8)$	4.42	3.92	3.00	2.98	3.17

Table 6.5 *MSE* of the REAL2 with different size transistors.

Both tables show that the accuracy of the processing is strongly dependent on the

size of the NMOS transistors and almost independent on the size of the PMOS transistors. This can be explained by reference to the errors in the PMOS current mirrors that are averaged in the summing nodes of the network. The errors, therefore, do not emerge to the output as strongly as the errors caused by the NMOS-transistors, which function as dividers of the current coming from the neighbourhood.

The simulations show that, when using similar-size transistors, the REAL1 can be said to be more accurate than the REAL2. However, if the size of the realisation is also taken into account, the advances of the REAL1 are not as obvious. Tables 6.6 and 6.7 show the estimates of cell size, normalised to the cell size of the implemented cell, for both implementation options.

	$PMOS(\frac{1}{2})$	$PMOS(1)$	$PMOS(2)$	$PMOS(4)$	$PMOS(8)$
$NMOS(\frac{1}{2})$	0.4667	0.6333	0.9667	1.6333	2.9667
$NMOS(1)$	0.7667	0.9333	1.2667	1.9333	3.2667
$NMOS(2)$	1.3667	1.5333	1.8667	2.5333	3.8667
$NMOS(4)$	2.5667	2.7333	3.0667	3.7333	5.0667
$NMOS(8)$	4.9667	5.1333	5.4667	6.1333	7.4667

Table 6.6 Relative size of the REAL1 with different size transistors.

	$PMOS(\frac{1}{2})$	$PMOS(1)$	$PMOS(2)$	$PMOS(4)$	$PMOS(8)$
$NMOS(\frac{1}{2})$	0.3500	0.5167	0.8500	1.5167	2.8500
$NMOS(1)$	0.5333	0.7000	1.0333	1.7000	3.0333
$NMOS(2)$	0.9000	1.0667	1.4000	2.0667	3.4000
$NMOS(4)$	1.6333	1.8000	2.1333	2.8000	4.1333
$NMOS(8)$	3.1000	3.2667	3.6000	4.2667	5.6000

Table 6.7 Relative size of the REAL2 with different size transistors.

From the tables, it can be observed that the REAL2 implementation consumes considerably less silicon area when using same size transistors. This is because the number of transistors in the cell realisation is smaller. By combining the results from the accuracy simulations and the estimate of the silicon area, it can be concluded that, with the same amount of silicon area using the REAL2, better accuracy can be obtained.

6.2.2.2 Comparison to the Measured Output of the Implemented Chips

Naturally, it is interesting to see how the simulations relate to measured data from the implemented chips. Even if the template realisations were quite different, what both systems had in common would be that they maintain the input level, if the input for all the cells remained the same. With the measured data, we used the cells in the middle of the network that were not affected by the defective border cells and established the

standard deviation to be 4.7*LSB*'s after linear column-wise correction with the mean output of 247 *LSB*. If the case where $\lambda = 1$ is chosen from the simulations, it is possible to calculate the standard deviation for both REAL1 and REAL2 when the PMOS and NMOS sizes are the same as with the larger realised chip. Since the maximum input was 200 in the simulations, the results are multiplied with $247/200$. This way a standard division of 3.29 can be obtained for REAL1 and 4.32 similarly for REAL2. The simulated results show behaviour similar to the measured results for the REAL2 especially.

6.2.2.3 Effect on the DoG and *Edge-enhancing Low-pass Filter* methods

The simulation system allows us to also simulate the effect of the mismatch on the application that was shown in Section 2.8. This is possible because, when calculating the mismatch-affected templates with transistors of a particular size, each transistor in the circuit was given a mismatch-affected value and the different templates were calculated using these transistor values. This results that, in the calculations of the templates, when considering two different λ-values, the transistors that are in use when forming both values, affect the final value in the same direction in the both cases, as it would be in a real silicon implementation.

Figures 6.11(a), 6.11(b) and 6.11(c) show the simulated DoG outputs when the used transistor sizes are NMOS(2) and PMOS(1). The used method is the same as in Section 2.8.2.2.

(a) Ideal output of the DoG-method when using λ-values 2 and $\frac{1}{4}$

(b) Output of the DoG-method when using REAL1 in the implementation

(c) Output of the DoG-method when using REAL2 in the implementation

Figure 6.11 Processing the LEVEL image using different λ-values.

In visual comparison, the images seem to be quite similar, but if the images are compared pixel-to-pixel, the differences become quite significant. Tables 6.8 and 6.9 show the percentage of the pixels with values different from those in the ideal simulation for all the combinations of the transistor sizes. The used λ-values were 2 and $\frac{1}{4}$.

As the tables show, the percentage of different pixels is quite high, even for the

	$PMOS(\frac{1}{2})$	$PMOS(1)$	$PMOS(2)$	$PMOS(4)$	$PMOS(8)$
$NMOS(\frac{1}{2})$	10.1%	10.3%	9.8%	11.2%	9.6%
$NMOS(1)$	9.4%	10.1%	8.0%	8.6%	7.4%
$NMOS(2)$	6.5%	5.7%	5.8%	5.6%	5.5%
$NMOS(4)$	5.7%	4.4%	4.0%	4.1%	3.8%
$NMOS(8)$	4.7%	3.2%	3.3%	2.9%	3.1%

Table 6.8 Percentage of pixels having different values when ideal simulation and mismatch simulations are compared using REAL1.

	$PMOS(\frac{1}{2})$	$PMOS(1)$	$PMOS(2)$	$PMOS(4)$	$PMOS(8)$
$NMOS(\frac{1}{2})$	10.1%	10.3%	10.5%	9.4%	10.7%
$NMOS(1)$	8.5%	8.2%	8.4%	9.6%	8.7%
$NMOS(2)$	6.2%	5.8%	5.4%	5.8%	5.2%
$NMOS(4)$	5.5%	4.2%	4.1%	3.8%	3.6%
$NMOS(8)$	4.0%	3.8%	3.0%	3.2%	3.5%

Table 6.9 Percentage of pixels having different values when ideal simulation and mismatch simulations are compared using REAL2.

large transistors with the shown threshold value. Another thing, that can be seen from the tables is that the REAL2 implementation results better performance in general than the REAL1 version. This can be explained by the use of the same transistors with both λ-values in REAL2 in contrast to REAL1, where the $\lambda = \frac{1}{4}$ requires the use of subtracting circuit and the $\lambda = 2$ is calculated without it.

However,if the obtained DoG masks are used in *Edge-enhancing Low-pass Filtering*, as suggested in Section 2.8.2.2, and the resulting images are visually compared, the difference is quite invisible, as Figures 6.12(a)-6.12(c) show.

(a) Ideal output of the processor when using λ-values 2 and $\frac{1}{4}$

(b) Output of the processor when using REAL1 in the implementation

(c) Output of the processor when using REAL2 in the implementation

Figure 6.12 The results of the simulations of the *Edge-enhancing Low-pass Filter* algorithm.

If all the results are combined, it can be stated that by carefully optimising the sizes of the transistors, it would be possible to design a resistive network chip with limited amount of programmability that would be reasonable small in silicon size. The

accuracy of the processing would be sufficient for human visual system and yet the size of the processor would lie under $10mm^2$ for, for instance, CIF-size processor. It the accuracy is required to be the original 8 bits of video standards, the proposed systems are not suitable for implementation.

Chapter 7

Conclusions

In this thesis, the realisation of a resistive network was investigated through the theory of CNN. The work started from the implementation of the *Edge-enhancing Low-pass Filter*, presented by Stoffels, that was aimed to be used in a video compression system. To minimise the required silicon area, the Reduced Cell-row System was developed. To show the pros and cons of the RCS, it was first compared to other methods of processing an image in parts by comparing processing speeds, silicon sizes and power consumption.

The functionality of RCS was finally tested with two separate chips that were designed and manufactured using first a $0.25\mu m$ process and then a $0.18\mu m$ process. The both chips reached their goals despite some errors and difficulties in the implementations: the first version showed the system itself to be functional, even thought any accuracy measurements could not be performed, and the second version showed that large-scale implementation was feasible when considering the size of the array and processing speed.

The larger chip had a 64×16 low-pass array processor network with a 3×16 network for gradient calculation. The total chip area was $2.03\mu m^2$ without the pads, out of which the analogue processing parts covered 28% if the wiring was not taken into account. Even with the parallel I/O the processing speed was over 4000 frames per second; if only the internal operations are taken into account the speed was increased to over 16000 frames per second. If these are combined with the requirements of a normal video processing system, where the frame-rate is 30 frames per second, the result promise, that the original goal of building an application specific processor, that could be added to any image processor system, would be possible.

On the basis of the low-pass cell in the implemented fixed-template chips, a resistive network cell with a limited programmability was designed. With this cell, it is possible to implement a network for higher-level processing tasks than just low-pass

filtering. The simulations targeted the effect of the mismatch that was found to be crucial on the previous chips. As a result, it was found that by carefully designing it would be possible to implement also some programmability on a resistive network processor, without losing any of the good properties of the implemented chips.

Overall, the work showed that there might still be room for analogue parallel processors in dedicated tasks that require enormous processing power but are limited by the silicon size and/or power consumption. These types of tasks can be found in, for instance, image processing and pattern recognition, both of which are currently at the centre of research.

Bibliography

[1] T.Kohonen, *Selforganization and Associative Memory*. Berlin, Germany: Springer-Verlag, 1989.

[2] J. N. H.Heemskerk, "Neurocomputers for brain-style processing. design, implementation and application," Ph.D. dissertation, Unit of Experimental and Theoretical Psychology Leiden University, The Netherlands, 1995.

[3] C. Cruz-Young, W.A.Hanson, and J.Y.Tam, "Flow-of-activation processing: parallel associative networks (pan)," in *American Institute of Physics Conference Proceedings*, 1986, pp. 115–120.

[4] Y.Maeda, H.Hirano, and Y.Kanata, "An analog neural network circuit with a learning rule via simultaneous perturbation," in *Proc. International Joint Conference on Neural Networks*, 1993, pp. 853–856.

[5] C.Mead, *Analog VLSI and Neural Systems*. USA: Addison-Wesley Publishing Company, 1989.

[6] C. A. Mead and M. A. Mahowald, "A silicon model of early visual processing," *Neural Networks*, vol. 1, no. 1, pp. 91–97, 1988.

[7] J. Lazzaro and C. Mead, "A silicon model of auditory localization," *Neural Computation*, vol. 1, no. 1, pp. 47–57, Spring 1989.

[8] W.Bair and C.Koch, "An analog VLSI chip for finding edges from zero-crossings," in *Advances in Neural Processing Systems 3*, 1991, pp. 399–405.

[9] H.Kobayashi, J.L.White, and A.Abidi, "An active resistor network for gaussian filtering of images," *IEEE J. Solid-State Circuits*, vol. 26, no. 5, pp. 738–748, May 1991.

[10] L.O.Chua and L.Yang, "Cellular neural networks: Theory," *IEEE Trans. Circuits Syst.*, vol. 35, no. 10, pp. 1257–1272, October 1988.

[11] G. Linan, A. Rodriguez-Vazquez, R.Carmona-Galan, F.Jimenez-Garrido, S. Espejo, and R.Dominguez-Castro, "A 1000 FPS at 128×128 vision processor with 8-bit digitized I/O," *IEEE J. Solid-State Circuits*, vol. 39, no. 7, pp. 263–275, July 2004.

[12] R.Tetzlaff, R.Kunz, C.Ames, and D.Wolf, "Analysis of brain electrical activity in epilepsy with cellular neural networks (CNN)," in *Proc. European Conf. on Circuit Theory and Design*, 1999, pp. 1007–1010.

[13] A.Kananen, A.Paasio, S.Lindfors, and K.Halonen, "A cellular nonlinear network for digital error correction," in *Proc. Int. Symp. Circuits Syst.*, 1999, pp. 255–258.

[14] A.Stoffels, T.Roska, and L.O.Chua, "Object-oriented image analysis for very-low-bitrate video-coding systems using the CNN universal machine," *International Journal of Circuit Theory and Applications*, vol. 25, no. 4, pp. 235–258, July/August 1997.

[15] A.Paasio, A.Kananen, K.Halonen, and V.Porra, "A QCIF resolution binary I/O CNN-UM chip," *Journal of VLSI Signal Processing Systems*, vol. 23, no. 2-3, pp. 281–290, Nov.-Dec. 1999.

[16] ——, "Different approaches for CNN VLSI implementations," in *Proc. European Conf. on Circuit Theory and Design*, 1999, pp. 1347–1350.

[17] A.Kananen, A.Paasio, M.Laiho, and K.Halonen, "CNN applications from the hardware point of view: Video sequency segmentation," *International Journal of Circuit Theory and Applications*, vol. 30, no. 2-3, pp. 117–137, March-June 2002.

[18] M.Anguita, F.J.Pelayo, E.Ros, D.Palomar, and A.Prieto, "Focal-plane and multiple chip VLSI approaches to CNNs," *Analog Integrated Circuits and Signal Processing*, vol. 15, no. 9, pp. 263–275, September 1998.

[19] L.Raffo, S.P.Sabatini, G.M.Bo, and G.M.Bisio, "Analog VLSI circuits as physical structures for perception in early visual tasks," *IEEE Trans. Neural Networks*, vol. 9, no. 6, pp. 1483–1494, November 1998.

[20] M. Laiho, "Mixed-mode cellular array processor realization for analyzing brain electrical activity in epilepsy," Ph.D. dissertation, Helsinki University of Technology, Espoo, Finland, 2003.

[21] B.E.Shi and L.O.Chua, "Resistive grid image filtering: Input/output analysis via the cnn framework," *IEEE Trans. Circuits Syst. I*, vol. 39, no. 7, pp. 531–548, July 1992.

[22] G.Singer and S.Rusu, "The first IA-64 microprocessor," in *Proc. IEEE Intl. Solid-State Circuits Conf., Digest of Technical Papers*, 2000, pp. 422–423.

[23] R.S.Bajwa, R.M.Owens, , and M.J.Irwin, "Image processing with the MGAP: a cost effective solution," in *Proc. 7th International Parallel Processing Symposium*, 1993, pp. 439–443.

[24] L.O.Chua and T.Roska, *Cellular Neural Networking and Visual Computing*. Cambridge, United Kingdom: Cambridge University Press, 2002.

[25] A.Kananen, A.Paasio, M.Laiho, and K.Halonen, "An improved current mirror based approach for linear spatial filtering," in *Proc. European Conf. on Circuit Theory and Design*, 2001, pp. 137–140.

[26] K.Wiehler, R.Lembcke, R.-R.Grigat, J.Heers, C.Schnorr, and H.-S. Stiehl, "Dynamic circular cellular networks for adaptive smoothing of multi-dimensional signals," in *Proc. Cellular Neural Networks and their Applications*, 1998, pp. 313–318.

[27] V.Gruev and R. Etienne-Cummings, "A pipe-lined differencing imager," *IEE Electronics Letters*, vol. 38, no. 7, pp. 315–317, March 2002.

[28] D. Marr and E. Hildreth, "Theory of edge detection," *The Proceedings of the Royal Society, London*, vol. 207, pp. 187–217, February 1980.

[29] T.Poggio, V.Torre, and C.Koch, "Computational vision and regularization theory," *Nature*, vol. 317, pp. 314–319, September 1985.

[30] D.Gabor, "Theory of communications," *Journal of IEE*, vol. 93, no. 26, pp. 429–456, 1946.

[31] T.K.Hogan, "A general experimental solution of poisson's equation for two independent variables," *J. Inst. Eng. (Australia)*, vol. 15, pp. 89–92, April 1943.

[32] G.Liebmann, "Solution of partial differential equations with a resistance network analogue," *J. Inst. Eng. (Australia)*, vol. 1, no. 4, pp. 92–103, April 1950.

[33] C.Koch, A.Moore, W.Bair, T.Horiuchi, B.Bishofberger, and J. Lazzaro, "Computing motion using analog VLSI vision chips: An experimental comparison among four approaches," in *Advances in Neural Processing Systems 3*, 1991, pp. 312–324.

[34] M.Balsi, I.Ciancaglioni, and V.Cimagalli, "Optoelectronic cellualar neural network based on amorphous silicon thin film technology," in *Proc. Cellular Neural Networks and their Applications*, 1994, pp. 399–403.

[35] A. Paasio, "Integration of cellulalr nonlinear network universal machine," Ph.D. dissertation, Helsinki University of Technology, Espoo, Finland, 1998.

[36] T.Roska and L. Chua, "The CNN universal machine: An analogic array computer," *IEEE Trans. Circuits Syst. II*, vol. 40, no. 3, pp. 163–146, March 1993.

[37] A. Rodriguez-Vazquez, S. Espejo, R.Dominguez-Castro, J.L.Huertas, and E.Sanchez-Sinencio, "Current-mode techniques for the implementation of continuous- and discrete-time cellular neural networks," *IEEE Trans. Circuits Syst. II*, vol. 40, no. 3, pp. 132–146, March 1993.

[38] A.Paasio and V.Porra, "A CNN universal machine chip with 295 $cells/mm^2$," in *Proc. International Symposium on Nonlinear Theory and Applications*, 1997, pp. 221–224.

[39] G. Linan, P. Foldesy, A. Rodriguez-Vazquez, S. Espejo, R. Dominguez-Castro, and E. Roca, "A 0.5 μm cmos 10^6 transistors analog programmable array processor for real-time image processing," 1999, pp. –.

[40] C.-Y. Wu and H.-C. Jiang, "An improved BJT-based silicon retina with tunable image smoothing capability," *IEEE Trans. VLSI Syst.*, vol. 7, no. 2, pp. 241–248, June 1999.

[41] K. Zaghloul and K. Boahen, "Optic nerve signals in a neuromorphic chip II: Testing and results," *IEEE Trans. Biomedical Eng.*, vol. 51, no. 4, pp. 667–675, April 2004.

[42] D. Standley, "Object position and orientation IC with ebedded imager," *IEEE J. Solid-State Circuits*, vol. 26, no. 12, pp. 1853–1859, December 1991.

[43] P.-F. Ruedi, P. Heim, F. Kaess, E. Grenet, F. Heitger, P.-Y. Burgi, and P. Nussbaum, "A 128×128 pixel 120-dB dynamic range vision-sensor chip for image contrast and orientation extraction," *IEEE J. Solid-State Circuits*, vol. 38, no. 12, pp. 2325–2333, December 2003.

[44] T. Choi, B.E.Shi, and K. Boahen, "An ON-OFF orientation selective address event representation image transceiver chip," *IEEE Trans. Circuits Syst. I*, vol. 51, no. 2, pp. 342–353, February 2004.

[45] L.Raffo, "Resistive network implementing maps of gabor functions of any phase," *IEE Electronics Letters*, vol. 31, no. 22, pp. 1913–1914, October 1995.

[46] B.E.Shi, "Gabor-type filtering in space and time with cellular neural networks," *IEEE Trans. Circuits Syst. II*, vol. 45, no. 2, pp. 121–132, February 1998.

[47] ——, "A one-dimensional CMOS focal plane array for Gabor-type image filtering," *IEEE Trans. Circuits Syst. I*, vol. 46, no. 2, pp. 323–327, February 1999.

[48] S.Rusu, S.Tam, H.Muljono, D.Ayers, and J.Chang, "A dual-core multi-threaded xeon processor with 16MB l3 cache," in *Proc. IEEE Intl. Solid-State Circuits Conf., Digest of Technical Papers*, 2006, pp. 102–103.

[49] A.Paasio and A.Dawidziuk, "CNN template robustness with different output nonlinearities," *International Journal of Circuit Theory and Applications*, vol. 27, no. 1, pp. 87–102, March 1999.

[50] R.Carmona-Galan, A.Rodriguez-Vazquez, S.Espejo-Meana, R.-C. amd T.Roska, T.Kozek, and L.O.Chua, "An 0.5-μm CMOS analog random access memory chip for TeraOPS speed multimedia video processing," *IEEE Trans. Multimedia.*, vol. 1, no. 2, pp. 121–135, June 1999.

[51] A.Kananen, A.Paasio, and K.Halonen, "Overlapping issues in designing large CNNs," in *Proc. Cellular Neural Networks and their Applications*, 2000, pp. 321–324.

[52] A.Kananen, M.Laiho, A.Paasio, and K.Halonen, "Nx16 cellular test chips for low-pass filtering," in *Proc. Int. Symp. Circuits Syst.*, 2004, pp. 461–464.

[53] P. E. Allen and D. R. Holberg, *CMOS Analog Circuit Design*. USA: Holt, Rinehart and Winston, 1987.

[54] A.S.Sedra and K.C.Smith, *Microelectronic Circuits, Third Edition*. USA: Saunders College Publishing, 1991.

[55] K. Koli, "CMOS current amplifiers: Speed versus nonlinearity," Ph.D. dissertation, Helsinki University of Technology, Espoo, Finland, 2000.

[56] P.G.Drennan and C.C.McAndrew, "Understanding MOSFET mismatch for analog design," *IEEE J. Solid-State Circuits*, vol. 38, no. 3, pp. 450–456, March 2003.

[57] M.J.M.Pelgrom, A.C.J.Duinmaijer, and A.P.G.Welbers, "Matching properties of MOS transistors," *IEEE J. Solid-State Circuits*, vol. 24, no. 5, pp. 1433–1440, October 1989.

[58] G.Wegmann and E.A.Vittoz, "Analysis and improvements of accurate dynamic current mirrors," *IEEE J. Solid-State Circuits*, vol. 25, no. 3, pp. 699–706, June 1990.

[59] E.Bruun, "Analytical expressions for harmonic distortion at low frequencies due to device mismatch in cmos mirrors," *IEEE Trans. Circuits Syst. II*, vol. 46, no. 7, pp. 937–941, September 1999.

[60] I.Baturone, S.Sanchez-Solano, A.Barriga, and J.L.Huertas, "Implementation of CMOS fuzzy controllers as mixed-signal integrated circuits," *IEEE Trans. Fuzzy Systems*, vol. 5, no. 1, pp. 1–19, February 1997.

[61] F. Gray, "Pulse code communication," U.S. patent no. 2,632,058, Mar. 1953.

[62] H.Pilo, A.Allen, J.Covino, P.R.Hansen, S.Lamphier, C.Murphy, T.Traver, and P.Yee, "An 833-MHz 1.5-W 18-Mb CMOS SRAM with 1.67 Gb/s/pin," *IEEE J. Solid-State Circuits*, vol. 35, no. 11, pp. 1641–1647, November 2000.

[63] A.Alvandpour, R.K.Krishnamurthy, K.Soumyanath, and S.Y.Borkar, "A sub-130-nm conditional keeper technique," *IEEE J. Solid-State Circuits*, vol. 37, no. 5, pp. 633–638, May 2002.

[64] R. van de Plassche, *CMOS Integrated Analog-to-Digital and Digital-to-Analog Converters*, 2nd ed. Boston: Kluwer, 2003.

[65] *IEEE Standard for Terminology and Test Methods for Analog-to-Digital Converters, Standard, Measurements*, IEEE Standard 1241-2000, 2000.

[66] M. Waltari, "Circuit techniques for low-voltage and high-speed A/D converters," Ph.D. dissertation, Helsinki University of Technology, Espoo, Finland, 2002.

[67] K. Hui and B.E.Shi, "Distortion in analog vlsi networks for image filtering," *IEEE Trans. Circuits Syst. I*, vol. 46, no. 10, pp. 1161–1171, October 1999.

Appendix A

Chip Layout

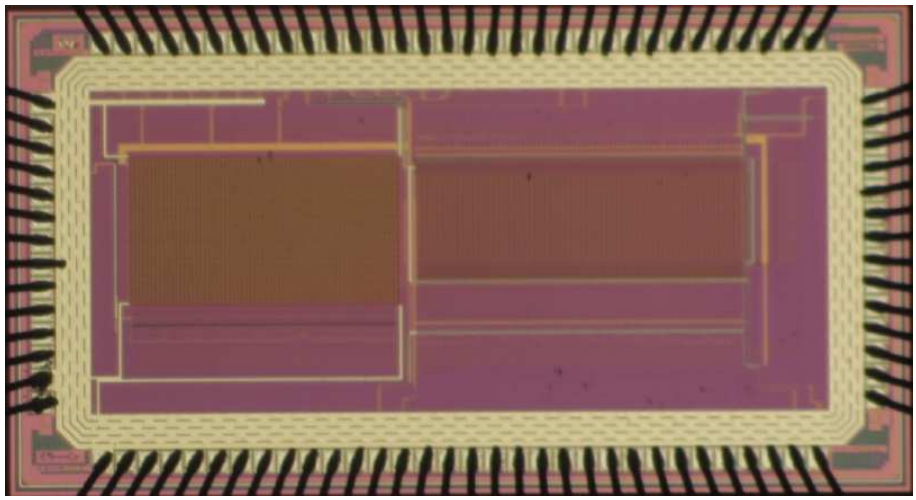


Figure A.1 Chip photography

ISBN 978-951-22-8622-5
ISBN 978-951-22-8623-2 (PDF)
ISSN 1795-2239
ISSN 1795-4584 (PDF)