On Hardware Implementation of Discrete-Time Cellular Neural Networks

Ph.D. thesis

Suleyman Malki

LUND UNIVERSITY

Department of Electrical and Information Technology,
Faculty of Engineering, 2008

Circuits and Systems
Department of Electrical and Information Technology
Lund University
Box 118
S-221 00 Lund, Sweden
http://www.eit.lth.se/
e-mail: suleyman@eit.lth.se, suleyman.malki@gmail.com

ISSN 1654-790X
NR 11

Printed by Tryckeriet i E-huset, Lund 2008.

To my parents
Gabriel and Suad
and my wife
Abeer

Abstract

Cellular Neural Networks are characterized by simplicity of operation. The network consists of a large number of nonlinear processing units; called cells; that are equally spread in the space. Each cell has a simple function (sequence of multiply-add followed by a single discrimination) that takes an element of a topographic map and then interacts with all cells within a specified sphere of interest through direct connections. Due to their intrinsic parallel computing power, CNNs have attracted the attention of a wide variety of scientists in, e.g., the fields of image and video processing, robotics and higher brain functions.

Simplicity of operation together with the local connectivity gives CNNs first-hand advantages for tiled VLSI implementations with very high speed and complexity. The first VLSI implementation has been based on analogue technology but was small and suffered from parasitic capacitances and resistances leading to undesired behaviour. Later implementations focus on larger network and higher level of robustness. Mixed full-custom chips are most famous and widely considered as a roadmap for advanced realizations. The digital counter parts have focused on emulating the functionality of the CNN rather than providing real-time performance. Furthermore, they are totally dependent on a host PC to function properly. In spite of being less sensitive to parasitic noise and fabrication artefacts beside providing a quasi-infinite accuracy, fully digital implementations are, however, still not available. In other words, the exploitation of a stand-alone fully-digital approach is highly desired, which this thesis aims to tackle.

Macro enriched Field-Programmable Gate-Arrays (FPGAs) are used to realize such systems on silicon. At first glance a pipelined approach, based on

circuit switching, seems promising. Two different approaches are investigated; Spatial and Temporal, of which the former is to prefer. Later on, in order to overcome design limitations and thus enhance performance, the benefits of packet-based switching have been explored. Although circuit switching is still employed, the enhancement is achieved by adopting the concept of Network-on-Chip (NoC), where packets are transmitted in a predefined communication pattern. The choice is between Serialized and Switched broadcasting schemes. The digital implementation of the Switched broadcasting is performed using Xilinx Virtex-II Pro P30 and the advantages over the pipelined approach are discussed by means of clock rate, area utilization and memory considerations. A serial communication approach shows, however, that network size can be increased further by a clear decrease in the size of communication interface. The thesis illustrates the power of the different implementations experimentally. It is shown how the digital CNN can be used to estimate velocity from images or to facilitate authentication by means of vein feature extractions. Furthermore, the issue of robustness is discussed from a different point of view. Here, the limited accuracy is compensated by gradual adjustment of the operative parameters, i.e. template coefficients. Finally, the thesis discusses main ingredients in system architecture to achieve the goal of a stand-alone fully-digital design.

Keywords

Cellular Neural Network, Discrete-Time Cellular Neural Network, Field-Programmable Gate-Array, Circuit switching, Network on Chip, Serialized broadcast, Switched broadcast, Velocity measurement, Vein feature extraction, Image processing.

Contents

Preface

This thesis aims on presenting the results of my research at the Department of Electrical and Information Technology at Lund University. The different parts of the material have been published in the publications listed below. References to the publications throughout the thesis are made using numbering convention adopted in the list.

[I] S. Malki. "Discrete-Time Cellular Neural Networks Implemented on Field-Programmable Gate-Arrays to Build a Virtual Sensor System." Lic. Thesis, Lund University, Lund, ISBN 91-7167-040-8, 2006, 98 pages.

[II] S. Malki, G. Deepak, V. Mohanna, M. Ringhofer and L. Spaanenburg. "Velocity Measurement by a Vision Sensor," in *Proc. IEEE International Conference on Computational Intelligence for Measurement Systems and Applications (CIMSA'06)*, La Coruna, Spain, 2006, pp.135-140.[1]

[III] S. Malki, Y. Fuqiang and L. Spaanenburg. "Vein Feature Extraction Using DT-CNNs," in *Proc. 10th International Workshop on Cellular Neural Networks and Their Applications (CNNA'06)*, Istanbul, Turkey, 2006, pp. 307-312.

[1] Best Student Paper Award

[IV] S. Malki and L. Spaanenburg. "Efficiency Considerations for DT-CNN Hardware," in *Proc. IEEE Northeast Workshop on Circuits and Systems (NEWCAS'07)*, Montréal, Canada, 2007, pp. 1038-1041.

[V] S. Malki and L. Spaanenburg. "Design Space Exploration for a DT-CNN," in *Proc. 11th International Workshop on Cellular Neural Networks and Their Applications (CNNA'08)*, Santiago de Compostela, Spain, ISBN 978-1-4244-2090-2, Jul. 2008, pp. 69 -74.

[VI] S. Malki and L. Spaanenburg. "A DT-CNN Data-flow Implementation," in *Proc. 11th International Workshop on Cellular Neural Networks and Their Applications (CNNA'08)*, Santiago de Compostela, Spain, ISBN 978-1-4244-2090-2, Jul. 2008, pp. 17 - 22.

[VII] S. Malki and L. Spaanenburg. "Soft DT-CNN core implementations," in *Proc. 15th IEEE International Conference on Electronics, Circuits, and Systems, (ICECS 2008)*, ISBN 978-1-4244-2182-4, Aug. 2008, pp. 1183 – 1186.

[VIII] S. Malki and L. Spaanenburg. "Design Space Exploration for the integrated digital CNN camera," in *Proc. 1st Int. Conference on Information Technology (IT2008)*, Gdansk, Poland, 2008, pp. 107-110.

[IX] S. Malki and L. Spaanenburg. "A CNN-Specific Integrated Processor." under review, *EUROSIP Journal on Advances in Signal Processing*, 2008.[2]

[X] S. Malki and L. Spaanenburg. "Algorithmic optimization of CNN computational hardware." under review, *Journal of Embedded Computing*, IOS Press.

[2] This paper is invited for submission in the special issue of the stated journal. The paper is composed of publications [V] and [VI].

Acknowledgements

First and foremost, my gratitude goes to my supervisor *Lambert Spaanenburg*, not only because of his efforts in guiding me but even for being understanding and patient. It is through his knowledge and kindness I stand at this stage of my research.

Further, I wish to express my appreciation to the many students of VLSI courses that, in different ways, helped to carry out the research work presented in this thesis.

I would also like to express my gratitude to staff and colleagues from the Department of Electrical and Information Technology, especially *Koraljka Golub* for being such a good and supporting friend.

My parents *Gabriel* and *Suad,* your have always supported and helped me, in different and creative ways, to have my dreams come true. Without your love, care, understanding and belief in me throughout the years, I would never be able to overcome the hindrances in my way.

Great parents cannot give you less than wonderful brothers. Thank you, *Jan* and *Michael*, for your support and love throughout my life. I'm also thankful for your families, *Maria* & *Nicole*, and *Iman* & *Gabriel*, for being around and bringing joy to our gatherings.

My mother in law, *Samira*, thank you for all the preyers, they certainly helped! *Haneen*, thanks for filling the atmosphere with joy when you visit us. *Mousa* and *Majed*, I found two new brouthers in you.

Special thank to all my friends who helped to make my time in Lund as master student and researcher enjoyable. I mention with great gratitude *Ahmad*,

Duja, *Abeer*, *Nabaz*, *Faten, Hafez*, *Malek, Rola, Khalil, Enas, Majed, Chafik*, and *Esma*.

Finally, there are no words that can express my gratitude to the woman of my life, *Abeer*, for her continuous encouragement, understanding and belief in me, and for her endless love.

Lund September 22, 2008,

Suleyman Malki

List of Abbreviations

ACE	Analogic CNN Emulator Engine
APR	Analogue Program Register
ASIC	Application-Specific Integrated Circuit
BRAM	Block Select RAM
CLB	Configurable Logic Block
CMOS	Complementary Metal Oxide Semiconductor
CNN	Cellular Neural Network
CNN-HAC	CNN Hardware Accelerator
CNN-UC	CNN Universal Chip
CNN-UM	CNN Universal Machine
CPA	Cellular Processor Array
CT-CNN	Continuous-Time CNN
CU	Control Unit
DDR	Double Data Rate
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor
DT-CNN	Discrete-Time CNN
ECAD	Electronic computer-aided design

FIFO	First-In First-Out
FPGA	Field-Programmable Gate-Array
FSR	Full Signal Range
GACU	Global analogic control unit
GAPU	Global Analogic Programming Unit
HIB	Host Interface Bus
HIU	Host Interface Unit
ILP	Instruction Level Parallelsim
IMB	Image Memory Bus
IMU	Image Management Unit
IOMMU	Input/Output Memory Management Unit
ISA	Instruction Set Architecture
ISU	Instruction Store Unit
LAM	Local Analogue Memory
LAOU	Local Analogue Output Unit
LCCU	Local Communication and Control Unit
LED	Light-emitting diod
LLM	Local Logic Memory
LLU	Local Logic Unit
LPR	Logic Program Register
LUT	Look-up table
MAC	Multiply-Accumulate
MOS	Metal Oxide Semiconductor
NI	Network Interface
NoC	Network on Chip
OoI	Object of Interest
PAL	Phase Alternating Line
PC	Personal Computer
PCI	Peripheral Component Interconnect
PE	Processing Element
RAM	Random Access Memory
RISC	Reduced Instruction Set Computing
RoI	Region of Interest
SCR	Switch Configuration Register
SCSI	Smaller Computer System Interface
SDRAM	Synchronous Dynamic Random Access Memory
SIMD	Single Instruction Multiple Data

SoC	System on Chip
SPE	Synergetic Processor Entity
SRAM	Static Random Access Memory
VHDL	Very high-speed integrated circuit Hardware Description Language
VLIW	Very Long Instruction Word

List of Figures

List of Tables

Chapter 1

Introduction

Although different aspects of computational complexity have given rise to different complex computer architectures, the concept of scientific computing has not changed during the last 50-60 years. A computer is still built as a Turing machine with stored programmability, i.e. with the algorithm as the underlying mechanism [40]. When Alan Turing introduced his abstract machine in 1936 it was meant to consist of a tape of symbols from a finite alphabet, a header to read/write the symbols, a state register and finally an action table that tells the machine what to do next. About ten years later, the foundation that has been established by Turing is adopted in von Neumann's computer architecture. In general, a von Neumann machine stores both the program and the data in a memory that can be unified as in a Princeton architecture or separate as in a Harvard architecture. A control unit features a program counter and keeps track of how instructions are executed on the arithmetic and logic units. The program is executed sequentially in line with human thinking, which is the main reason for von Neumann machine to gain worldwide acceptance and to quickly become the fundament of future digital computing devices [2].

Being sequential, architectures based on von Neumann machine are characterized by low utilization of the computational components. As the execution of each instruction is divided into a number of stages, only those components belonging to the current stage are active while all other units in the architecture remain idle! This is partially remedied in Harvard architectures by introducing the concept of *instruction level parallelism* (ILP), where the different stages are combined into a single pipeline. The maximum throughput

is, however, still dictated by the impact of hazards in the computation due to memory access conflicts [81].

Actually, nowadays engineering tasks are characterized by the high complexity of the underlying algorithms. Here, large amounts of information are handled in real-time and therefore require a 'close to perfect' memory management. In order to achieve that, a number of enhancement techniques have seen the daylight, where both hardware and software approaches have been tested. The focus of hardware developers have been on filling the performance gap between processor and memory which still dominates classical computer architectures [81]. Through intense utilization of the pipelining technique and advances in micro-electronic fabrication technology, the speed of processors has increased far more than the speed of semiconductor memory. This has caused the Reduced Instruction-Set Architecture (RISC) to reduce the amount of memory access per instruction and caching to raise virtual memory performance. Still the execution of data-intensive algorithms suffered and new architectures for image processing have been proposed [85]. Moreover, most algorithms overcome the intrinsic complexity of a certain problem through parallel execution of sub-operations, which opens for actual real-time performance. In light of the performance that software high-level languages provide, especially in real time applications, specialised hardware architectures are unavoidable.

The popularity of Digital Signal Processors (DSPs) illustrates the need for domain-specific processors with reduced memory access. Here, the data path is tailored for an optimal execution of a common set of repetitive and numerically intensive operations. However, DSPs still incorporate the von Neumann approach and remain, thus, sequential machines [2]. Consequently, moving toward parallel computing has become a dominant approach in computer architecture, mainly in the form of multi-core processors such as IBMs Cell [3]. Like all other coarse-grain parallel processing systems, such architectures come, however, with a large and complex instruction set [4]. Apparently, an increased granularity level will help to reduce the complexity. Beside, it is well-known that the smaller the granularity, the greater the potential for parallelism and hence speed-up. Cellular processor arrays (CPAs), that implement data processing at a fine-grain level of parallelism, are often comprised of simpler processors, with specific computational ability [4]. In one of the popular paradigms, Single Instruction Multiple Data (SIMD), each processor executes the same instruction, but operates on data residing in local memories [4]. Locality of storage removes most data hazards that are usually connected to access of common memories. This is preferable as most CPA architectures find application in the field of image processing and are usually considered as "vision chips". Here, low-level image processing tasks are executed on a processor-per-pixel arrangement. The intrinsic pixel-parallelism is inherited which enables real-time processing speeds without wasting any resources on long-distance transfers [5]. Together with the relaxed I/O demands comes the reduction in size, cost and power dissipation. It is reported that the power consumption is several orders of magnitude lower than for an equivalently

performing sequential system [4]. Due to the clear benefits, the analogue SIMD approach is incorporated in many vision chips, such as ACE16k [28]. What makes this chip interesting in our case is that it combines SIMD with the paradigm of *Cellular Neural Network* (CNN) [7]. The ACE16K has been introduced as the most promising vision chip that implements a CNN as parallel computing core.

This thesis focuses on the implementation of CNNs on hardware. But before digging deeper into technical details, section 1.1 explains the importance of image processing as a target for many CPAs in general and CNNs in particular. Subsequently, the objectives of the thesis are presented in section 1.2 where the concept of Cellular Neural Networks is briefly introduced. The choice of realization platform is discussed as well. Finally, the chapter is closed with outlining this thesis in section 1.3.

1.1 WHY IMAGE PROCESSING?

The focus of CPA architectures on image processing tasks is well-founded. One of the features distinguishing humans from other creatures is the dependence on our vision as main source of acquiring information. Indeed, vision is our most important sense as we rely on it for more than 99% of the information about our surrounding [1]. In contrast to a wide range of animals, humans have, apart from eyes, poorly developed sense organs. Consequently, it's not surprising that scientific instruments commonly communicate their results to the user by producing images, rather than generating audible tones or emitting smells. Even waves beyond the visible portion of the electromagnetic spectrum are presented visually, usually displayed with false colours to emphasise suitable variations in signal brightness [1].

Computer-based image processing applications usually apply algorithms based on human vision methods, but are not confined to it. Important differences between human vision and imaging devices implies considering other methods. The ability of a human judging colour or brightness of features within images depends on the possibility of comparing adjacent details. Furthermore, humans deal with images as a whole, rather than by breaking them down to constituent parts, which usually gives rise to many visual illusion phenomena, e.g. parallel lines appear to diverge if they cross different sets of parallel lines with different gradients (Figure 1.1). Hence, grouping parts within images is central to our understanding of images [1].

Each image can be considered as a container of signals that change value over time. These signals, seen as conveyors of data, are nothing else than all pixels constituting the image. Consequently, understanding the meaning of an image requires a mechanism for retrieving knowledge from the pixels within the image. A good example is in the on-line quality control of production streets. Here, we find high-end cameras with off-line high-performance computers to detect defects in the production and diagnose the probable cause. The challenge is to replace these by large amounts of cheap, virtual sensors [58] that can capture images but also feed back knowledge about the imaged situation; this

should lower the cost of inspection, improve the quality of production and provide reliable support. The advent of such vision sensors is made possible by the rapid decrease in price and size of the camera and the ongoing increase in performance and capacity of modern microelectronics. A vision sensor is based on the standard camera but extended with intelligent hardware and software to alleviate the communication demands that originate from full image transfer to a central computing service. Typical examples are (a) the remote temperature sensor that finds the flame within an image and checks colour, size and movement to quantify the burning conditions, (b) the microphonic imager to locate and analyse sound sources, and (c) the intelligent pen, that produces the equivalent character string.

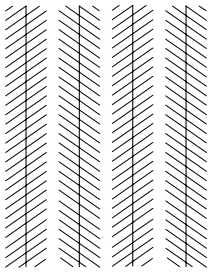


Figure 1.1 The vertical lines are actually parallel but appear to diverge.

Current applications range from velocity measurement to product inspection and are based on software personalization of Commercially-of-the-shelf (COTS) microprocessors. A migration toward vision sensors on basis of dedicated hardware is already established. The Xetal processor from Philips Research Laboratories [105] is a clear example. As the focus is on the *Region of Interest* (ROI), there is a natural clustering of data dependencies that can be utilized by introducing local operations on the locally stored ensemble of data. Very Long Instruction Word (VLIW) architectures are implemented to achieve the desired utilization, but other approaches are still demanded. In coming is, e.g., the Eye-RIS vision sensor [106] that employs a bio-inspired architecture where image acquistion and the fully parallel processing are combined. The key component of the Eye-RIS vision system is the retina-like front-end, which is a continuation from predecessing CNN-based chips, e.g. ACE4k [27] and ACE16k [28].

1.2 OBJECTIVES

In line with the previous section, it is not surprising that most experimental CNN systems have been proposed in image processing, in spite of the general nature of the CNN paradigm. CNNs have been introduced as a novel class of information-processing systems for solving complex real-time problems in space, like partial differential equations (PDEs). Due to their inherent potential, CNNs have attracted the attention of a wide variety of scientists. Over the years, the concept of CNN has shown to be multi-disciplinary: it has found application in robotics, bio-inspired vision issues and higher brain functions in addition to

image and video processing. Further, CNNs have been used to generate static and dynamic patterns, autowaves and spiral waves [11].

The paradigm is built on reformulating of many complex computational problems into well-defined tasks characterized by the fact that the information necessary to compute the solution at a certain point in space is within finite distance to that point. A CNN is made of a regular geometric 2-D or 3-D grid of cells that are connected locally.

After the introduction of the CNN model by Chua and Yang in 1988, different considerations for cell complexity, cell dynamics and network topology have led to the emergence of different generalized models. The reason has been to enhance both the capability and efficiency of the original CNN model. A list of the most common models includes: (a) Nonlinear CNNs where template coefficients are nonlinear functions and (b) Delay-type CNNs where cell dynamics are dependent on previous input/output pairs [15]. In order to reflect features found in neurobiological structures, Non-uniform CNNs with more than one type of cell and Multiple Neighbourhood Size CNNs have been studied [11]. Of all generalizations, this thesis focuses on employment and implementation of Discrete-Time CNNs (DT-CNNs) only [39],[41]. These different models have delivered a sound basis for the design of algorithm-specific analogue implementations. Even the discrete-time version has been introduced as analogue realization [39], in spite of the intrinsic favour for digital designs.

Obviously, CNNs give first-hand advantages for VLSI implementations due to their powerful parallelism and strict locality of operation. But the need for large numbers of multiplications has precluded efficient digital hardware realizations, leaving the stage to either analogue realizations or software implementations on highly pipelined hardware. Actually, the first CNN hardware has been almost completely in analogue. This has probably to do with that the first conceptual design proposed Chua and Roska, i.e. the CNN-Universal Machine (CNN-UM) [20], is analogue. So far impressive advances have been made in analogue realizations only [23]-[30], while the best attempt toward a digital realization *emulates* the functionality of a CNN rather than providing real-time performance [42]. Hence, the potential of a fully digital approach has never been exploited, which this thesis aims to change.

Digital CNN emmelators have followed the same development path as in classical computer architectures. The first publication [42] uses pipeline techniques to improve performance. The network is operated in step with the provision of image map elements, and the network is tuned such that it works exactly at the speed of the image stream. As such the architecture resembles that of a stream processor [86], a vector processor on images. Such architectures do not support the intense interaction that is required for the less trivial CNN operations.

Of late, the Network on Chip (NoC) architecture has been proposed to get away from the pipelining harness [9]. It is stated that a cellular architecture will be the way of the future. In general, a NoC consists of a number of switches and

network interfaces (NI). Network interfaces translate the view that components attached to the NoC have on communication, and the internal view switches have. By using multiple switches a NoC scales both in the number of components (such as cells) that can be attached to the NoC, and in the performance the NoC can deliver. NoCs are therefore modular, scalable interconnects [87]. A switch receives data on its inputs and sends it to its outputs, taking care that each output is used by only one input at any point in time. Data can be moved around a NoC in two ways: circuit switching [88], and packet switching [83].

Overall NoCs fit well with digital implementations (or models) of CNNs because they allow an arbitrary (programmable) neighbourhood of cells. Moreover, NoCs decouple the communication from computation, i.e. rates of computation of the individual cells may differ from each other, as well as the rate of inter-cell data transport between the cells. Hence no global notion of time or synchronization is required between the system components (cells/CNNs and NoC), taking any global interconnections out of the critical path. Still the system as a whole converges to a well-defined output for a given input if the components are continuous functions.

A major issue in application-specific hardware design is the time-consuming and costly fabrication process. As different architectures are to be built and tested in a relatively short time, there is need for a realization platform that provides a close-to-full-custom performance while retaining a high degree of flexibility and reusability. Furthermore, such a platform must allow for decreased granularity at least to the fine-grain level employed in SIMD-based CPAs. In this sense, Field-Programmable Gate-Arrays (FPGAs) seem the only choice (Figure 1.2). Actually, one of the greatest advantages of using FPGAs is the ability of using spatial computing rather than temporal or sequential computing. Higher throughput is then achieved as more parallelism per time unit is exploited.

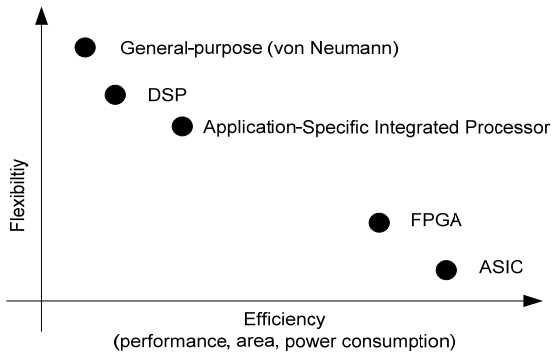


Figure 1.2 Efficiency versus performance of different implementation platforms [6] .

FPGAs have been commercially introduced in 1985 by Xilinx to replace standard gate arrays such as programmable logic arrays (PLA), programmable array logics (PAL) and complex programmable logic devices (CPLD) [2]. Over

time, FPGAs gained increased popularity as they allowed developers to bypass the costly fabrication process of application-specific chips [6]. Rapid prototyping is certainly the most common – but not the only – attraction of FPGAs. This allow for in system customization of non-accessible systems [2]. Another, and maybe the most important, feature of modern FPGAs is the ability for partial reconfiguration. Swapping modules into and out of the device without the need of a complete reset brings the FPGA a level of adaptability that reconfigurable devices never reached [2]. This innovation is unfortunately less utilized, though the potential benefits have been already illustrated early in [45] .

The ongoing improvements in modern FPGAs have led them away from being application-specific containers for logic circuitry to the algorithm-specific integrated circuit. An over-mass of flip-flops and logic-mapped memory is supplemented by high-density, multifunctional macros, such as Block Select RAM (BRAM) and Multiplier, while the supplementary handlers are easily accommodated in the microprocessor cores. Moreover, though being slower than Application-Specific Integrated Circuits (ASICs), FPGAs are gaining a foothold in speed. The newest devices for Xilinx, e.g., break the 500 MHz barrier [38], which theoretically paves the way to reach higher speed than what most CNN chips achieve.

Having all that in mind, the following questions need to be answered:

- ✓ Is a fully digital realization possible?
- ✓ Are FPGAs able to satisfactorily host such a realization?
- ✓ Which communication patterns are needed to meet the connectivity requirements?
- ✓ Can we accommodate already known approaches, or have new concepts to be developed from scratch?
- ✓ Is the ‘limited’ accuracy provided by digital implementation enough for real-life tasks?

Throughout the thesis, full digital approaches are explored by introducing a number of design implementations. Such implementations focus on the pattern of communication as the main consideration. The functionality and efficiency of the proposed designs are validated by means of different applications. The applications vary in the degree of difficulty from simple cases that test basic functionality to more advanced problems where the complex behaviour of the whole system is verified. In all cases, the implementations rely on the FPGA, more precisely on Virtex – II and Virtes – II Pro 30 from Xilinx [38].

1.3 THESIS OUTLINE

In Chapter 2, the concept of Cellular Neural Network is introduced. This thesis is restricted to the discrete-time version, where a number of basic examples are treated in detail, as the focus is on hardware implementations, Chapter 3 gives a brief overview of state-of-the-art of CNN chips. First of all, DSP-based emulators are covered, before the conceptual CNN-UM is discussed. Furthermore, both full-custom mixed-signal designs and pure digital emulators

of the CNN-UM are briefly described. It is meant that provided information serves as a solid base for the understanding of design approaches introduced later on.

Subsequently, the first digital implementations of DT-CNNs on FPGAs, as carried out by the author of this thesis, are discussed in Chapter 4. Here, two different unrollment schemes, temporal and spatial, are presented. Both employ pipelining with different degrees of success. The spatial scheme is discussed in detail as it serves as a start-up for later implementations. With the CNN hardware realization come the demands on inter-modular connectivity. Incorporating the concept of Network on Chip takes the hardware architecture one step further. The hard-wired communication is replaced by a packet-based communication pattern. The path is still pre-defined but the packets belonging to two different communication cycles (different source-target pairs) share one or more communication channels (inter-node connections). In this sense, we mix circuit switching with packet-switching techniques. Even here, two different implementations exist. One of them employs the idea of pipelining with moderate modifications on the internal design. In the other, the benefit of packet switching comes to full blossom in a broadcast architecture. Here, the CNN is divided into sets of active nodes with a totally different inter-node communication pattern.

A different approach is presented in Chapter 5 to overcome the enormous demands of internal communication. The approach is thought of as a revision of communication patterns already discussed in the previous chapters.

Chapter 6 takes the discussion one step further and covers memory considerations for the two main architectures. Chapter 7 shows how the variety of design implementations, presented throughout the thesis, is of benefit to different applications. It starts with a simple realization of the famous Game-of-Life, and moves to more advanced problems where the basics for a velocity meaturement system are verified. The power and suitability of performing biometric measures is then demonstrated by means of vein feature extraction.

One of the disadvantages of currently available analogue CNN chips is parameter deviation. The robustness of the system is easily disturbed due to noise in the electrical components as well as to parameter scattering introduced during the fabrication process. This leads to misbehaviour and often requires a laborious effort to tweak the parameter to the desirable range of operation. Chapter 8 tackles the problem from a different point of view. The precision of internal signals is gradually reduced while the system is guaranteed to perform well. The idea is that less bits in the internal representation compensates for the artefacts found in analogue chips, which allows finding a set of system parameters that guarantee the desired degree of robustness in all chips. Though inspired by the problems in analogue systems, it also has relevance to digitial ones. Pruning the internal representation helps to reduce word width and therefore reduce the size of the CNN nodes and the width of the communication paths

Chapter 9 proposes a methodology for design automation starting from a problem description and ending in a system architecture.

In Chapter 10 other design alternatives are introduced to the benefit of larger networks. The different architectures presented throughout the thesis are compared by means of area utilization and frequency. The chapter is closed with a concluding discussion.

Chapter 2

Cellular Neural Networks
The Concept

In 1988, Chua and Yang introduce a new architecture to efficiently perform large time-consuming tasks in real-time by using an array of simple, non-linearly coupled dynamic circuits. A novel class of information-processing systems is then born, and carries the name of Cellular Neural Network (CNN) [7].

The concept rests on two major sources of inspiration. The architecture possesses some of the key features of Neural Networks [8], such as continuous-time dynamics and global interaction of the network elements, which allows for real-time signal processing. On the other hand, it inherits the feature of local interconnectivity from the world of Cellular Automata [10], which makes it suitable for VLSI implementations.

In this chapter a brief description of two models is given: Chua and Yang model that is sometimes referered to as Continuous-Time CNN (CT-CNN) and the counterpart Discrete-Time CNN (DT-CNN). The aim is to give an intuitive understanding of the concept, rather than discussing the theory in detail. In section 2.1 the network structure is introduced as it eases the understanding of CNNs basic equations in section 2.2. Consequently, sections 2.3 discusses the effects of different parameter set-ups, while the importance of boundary cell handling is illustrated in section 2.4. The Discrete-Time CNN is presented in section 2.5, while section 2.6 shows that both presented models gain in power when more than one layer is used. Section 2.7 reintroduces the first analogue realizations of the two models. In order to increase the understanding for the functionality of DT-CNN model, a number of illustrative examples are presented in section 2.8. The chapter is closed with a summary in section 2.9.

2.1 SPHERE OF INFLUENCE (NEIGHBOURHOOD)

The CNN is a massive aggregate of regularly spaced processing units, called *cells*. Similar to Cellular Automata [10], any cell is connected only to its neighbour cells, where direct interaction only occurs among adjacent cells. Other cells are, however, indirectly affected due the propagation effect of the continuous-time dynamics. Theoretically, a cellular neural network of any dimension can be defined, as illustrated in Figure 2.1, which allows a CNN to handle spatial relations such as topographic maps. As the focus of this thesis is on 2-dimentional image processing, the discussion will be restricted to the 2-dimensional case.

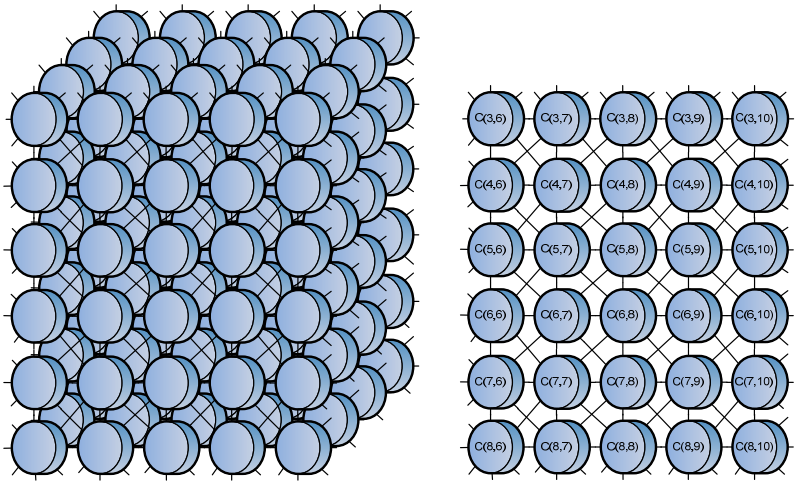


Figure 2.1 Cellular neural networks with different dimensions, where the globes represent cells and the links represent direct coupling. Far from all interconnections are seen in the 3-dimensional case (left). In the 2-dimensional finite-size case (right) each cell C(i,j) is indexed according to row i and column j.

Considering a finite-size two-dimensional CNN, cells are arranged in *M* rows and *N* columns. Each cell is identified by its position in the grid, denoted *C(i, j)*, and communicates directly with its *sphere of influence* $S_r(i,j)$ of *radius* r , also called *r-neighbourhood*. Such a neighbourhood is defined as the set of cells within a certain distance *r* to *C(i j)*, where $r \geq 0$ (Eq. (2.1)).

$$S_r(i,j) = \{C(k,l) | \max(|k-i|, |l-j|) \leq r; 1 \leq k \leq M, 1 \leq j \leq N\} \quad (2.1)$$

For instance, if $r = 1$ we have a 1-neighbourhood. It is also common practice to talk about 3×3 neighbourhood when $r = 1$, and 5×5 neighbourhood when $r = 2$ and so on. In general, for certain $r \geq 0$ a neighbourhood of size $(2r+1)^2$ is obtained. Different neighbourhood examples, with $r = 1,2$ and 3 are shown in Figure 2.2. Observe that when $r > N/2$, and $M = N$, a fully connected CNN is obtained, i.e. $S_r(i,j)$ is the entire network.

This extreme case, that is apparently impractical to build in a VLSI chip for large N, corresponds to the classical Hopfield Net [8].

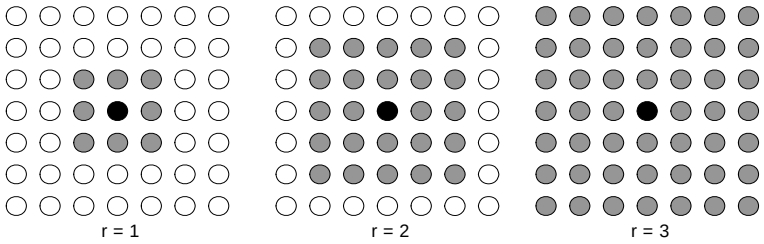


Figure 2.2 Different r-neighbourhoods for the centre cell (black circle). To avoid clutter all interconnections are dropped.

2.2 STANDARD CNN EQUATIONS

Let's first consider a cell with no coupling to any other cell in the grid. Such a cell, called an *isolated cell*, is associated with four variables: input $u_{ij} \in \mathbb{R}^u$, threshold $z_{ij} \in \mathbb{R}^z$, state $x_{ij} \in \mathbb{R}^x$, and output $y_{ij} \in \mathbb{R}^y$, which are, in general, functions of the continuous time *t*. The cell consumes the input value together with the threshold in order to produce the output value, which depends on the current state. Assuming further a given initial state $x_{ij}(t_0)$ at $t = t_0$, a threshold $z_{ij}(t_0)$ and an input $u_{ij}(t_0)$, the state $x_{ij}(t)$ evolves via the state equation given in Eq. (2.2) where the "dot" denotes the time derivate and $\mathcal{F}$ is an ordinary non-linear differential function.

Recall that an unknown function $\mathcal{H}: \mathbb{R} \to \mathbb{R}$ is ordinary differential if the n*th* derivative of $\mathcal{H}$ with respect to a variable $\hbar$ is a function of the lower-order derivatives, i.e. $\mathcal{F}\left(\hbar, \mathcal{H}, \mathcal{H}', \mathcal{H}'', \cdots, \mathcal{H}^{(n-1)}\right) = \mathcal{H}^{(n)}$. Furthermore, if the differential function is not dependent on the variable $\hbar$, it is then considered autonomous. In this sense, Eq. (2.2) is simply a non-autonomous system of ordinary differential equations [11]. In general, different non-linear functions $\mathcal{F}$ can be used for different cells, but in almost all known applications the cells are identical and therefore employ the same function.

$$\dot{x}_{ij} = \mathcal{F}(x_{ij}(t), z_{ij}(t), u_{ij}(t)) \tag{2.2}$$

The operative description of a cell is concluded by the determination of the output $y_{ij}(t)$ by means of a nonlinear function. This function may depend on $y_{ij}(t), x_{ij}(t)$ and $z_{ij}(t)$, but in this thesis, as in most literature, it is assumed to depend only on the state of the cell, as depicted in Eq. (2.3).

$$y_{ij}(t) = g\left(x_{ij}(t)\right) \tag{2.3}$$

The choice of function g is crucial for the quality of the obtained output and the speed it is achieved. Three different types of nonlinear functions are frequently used [41]: (a) threshold, (b) hyperbolic tangent and (c) piece-wise linear functions. The *threshold* function, commonly referred to as *Heaviside (or*

hardlimiter) function, is only binary-valued and performs a binary decision. The *hyperbolic tangent* function, shown in Figure 2.3.a and mathematically described in Eq. (2.4), is a special case of the *sigmoid* function that is generally defined as a strictly increasing continuous s-shaped function. By varying the *slope* parameter δ, different sigmoid functions are obtained. An important observation from Eq. (2.4) is that the sigmoid function becomes simply a threshold function as the slope parameter approaches infinity. Even though, the sigmoid function maintains, opposing to the threshold function, the characteristic of being differentiable. Finally, the most widely used discrimination function is the *piece-wise linear* function that is totally linear with positive slope within a certain interval $[-a, a]$ and saturates outside this interval as illustrated in Figure 2.3.b. The function is mathematically described in Eq. (2.5).

$$f(x) = \tanh(\delta x) \tag{2.4}$$

$$f(x) = \begin{cases} 1, & x \geq a \\ \delta x, & |x| < a \\ -1, & x \leq -a \end{cases} \tag{2.5}$$

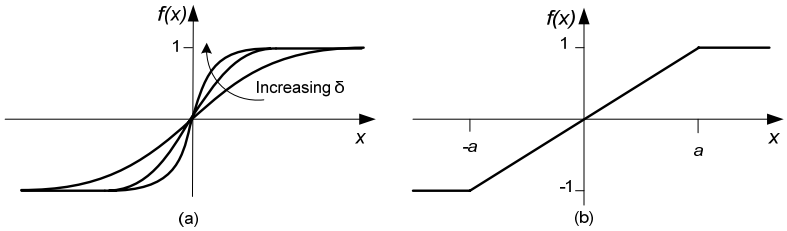


Figure 2.3 Sigmoid function (a) and piece-wise linear function (b).

The contributions of state and input variables are achieved by means of two weightings coefficients, a_{ij} and b_{ij}, while the threshold is simply assumed to be a constant scalar [11]. The coefficient a_{ij} mirrors the effect of the previous output value, while b_{ij} only scales the current input value. Hence, they are called *feedback* and *control* coefficients respectively. A threshold Zij is used to adjust the obtained state value into a desired range. This introduces the *standard isolated CNN cell*, claimed to be the most widely used in the literature. State equation of a standard isolated cell is given in Eq. (2.6); while the output is usually obtained by using the piece-wise linear function introduced in Eq. (2.5) with the interval $[-1, +1]$, with slope $\delta = 1$, resulting in Eq. (2.7). Assuming all coefficients are linear, the dynamics of the isolated CNN cell are due to the non-linear output function only.

$$\dot{x}_{ij} = -x_{ij} + a_{ij}y_{ij} + b_{ij}u_{ij} + z_{ij} \tag{2.6}$$

$$y_{ij} = f(x_{ij}) \triangleq \frac{1}{2}\left(|x_{ij} + 1| - |x_{ij} - 1|\right) = \begin{cases} 1, & x_{ij} \geq 1 \\ x_{ij}, & |x_{ij}| < 1 \\ -1, & x_{ij} \leq -1 \end{cases} \tag{2.7}$$

Equation (2.6) explains how the state of the cell evolves over time and is therefore commonly referred to as 'cell dynamics'. These dynamics are dependent on two constraints: *initial condition constraint* where the state variable is assumed equal a certain value upon start, and *input constraint* where input value $u \in [-1, +1]$.

In a general CNN architecture, each cell is directly coupled to all other cells within the sphere of influence. Both input u_{kl} and output y_{kl} of all neighbouring cells are available and therefore consumed to produce the new output. Similar to the isolated cell, inputs and outputs from cells belonging to S_r of the cell are weighted as b_{kl} and a_{kl} respectively. By simply summing the contributions of all cells in the sphere of influence, the state equation of a *standard CNN cell* can be written as in Eq. (2.8). The output value is still obtained according to Eq. (2.7).

$$\dot{x}_{ij} = -x_{ij} + \sum_{kl \in S_r(ij)} a_{kl}\, y_{kl}(t) + \sum_{kl \in S_r(ij)} b_{kl}\, u_{kl} + z_{ij} \qquad (2.8)$$

$$i = 1, 2, \cdots, M,\ j = 1, 2, \cdots, N$$

Almost all theorems and numerical techniques for solving ordinary differential equation systems are formulated in vector form [12]. Hence, it is desirable to express the state equation given in Eq. (2.8) in vector form. For a $M \times N$ CNN, $n = MN$ vector systems are obtained as depicted in (2.9), where the new indexing of state, output, input and the coefficients is obtained by a row-wise packing of the original matrices. The matrices $\boldsymbol{A}$ and $\mathbf{B}$ are $n \times n$ matrices whose nonzero entries are the weighting coefficients a_{kl} and b_{kl} respectively. As the coefficients are placed in a band along the main diagonal (Figure 2.4), each matrix is quite sparse where most of entries are zero. The vector form of the state equation is given in Eq. (2.10).

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_n \end{bmatrix} = -\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \boldsymbol{A}\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} + \mathbf{B}\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} + \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} \qquad (2.9)$$

$$\dot{\boldsymbol{x}} = -\boldsymbol{x} + \boldsymbol{A}\boldsymbol{y} + \mathbf{B}\mathbf{u} + \mathbf{z} \qquad (2.10)$$

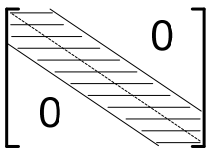


Figure 2.4 Band structure of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$.

One may conclude that each CNN cell in the mesh is a dynamic system whose state evolves according to a prescribed *state equation*. The dynamics of a cell are coupled to neighbouring cells lying within the *sphere of influence* that is centred at the location of the cell itself. The behaviour of the entire CNN is, however, highly sensitive to the dynamics of cells located at the boundary as will

be discussed in section 2.4. But first, a concise form of the state equation is introduced in the following section.

2.3 CLONING TEMPLATE

In general, all feedback and control coefficients in Eq. (2.8) can be represented by time-dependent nonlinear operators of the coupled values, but in this thesis they are assumed to be time-invariant and real-valued scalars. Furthermore, these coefficients are identical for all cells in the grid, which provide the CNN with one of its important features, i.e. *space invariance*.

In order to simplify the notation, the state equation (2.8) is written in a more compact form by using the two-dimensional convolution operator *, defined in [7],and reintroduction below.

***Definition*:** For any 3×3 matrix $\mathcal{M}$ that, the convolution operator * is defined by (2.11), where $\mathcal{M}(m,n)$ denotes the entry in the *m*th row and the *n*th column of the matrix, and $m,n \in \{-1,0,+1\}$.

$$\mathcal{M} * v_{ij} \triangleq \sum_{kl \in S_r(i,j)} \mathcal{M}(k-i, l-j) v_{kl} \tag{2.11}$$

Now, the weighting coefficients can be grouped in two square matrices: $\mathcal{A}$ and $\mathcal{B}$. The former holds all feedback coefficients and is accordingly called *feedback template*, while the latter is called *control template*. Together with the real-valued threshold (even called *bias*), they constitutes a so-called *cloning template* $\mathcal{T} = \langle \mathcal{A}, \mathcal{B}, z \rangle$. The latter term is commonly used to emphasize the property of space-invariance [14]. The compact form of the state equation is introduced in Eq. (2.12). Observe that Eq. (2.10) looks similar to Eq. (2.12), but the meaning of the involved parameters do differ, as the former deal with vectors while all parameters are scalars in the latter. The obtained result should, however, be the same. It is now obvious that cloning template $\mathcal{T}$ in addition to given input and the initial conditions, completely determine the dynamic behaviour of the cell.

$$\dot{x}_{ij} = -x_{ij} + \mathcal{A} * y_{ij} + \mathcal{B} * u_{ij} + z \tag{2.12}$$

The matrices in Eq. (2.13) show the common notation of feedback and control templates respectively, for the case of 1-neighbourhood. This notation is adopted later on (section 2.8) to index all input and output values of a cell. Furthermore, it is worth mentioning that the term *kernel* is widely used instead of template in image processing applications; see e.g. [13].

$$\mathcal{A} = \begin{bmatrix} a_{-1,-1} & a_{-1,0} & a_{-1,1} \\ a_{0,-1} & a_{0,0} & a_{0,1} \\ a_{1,-1} & a_{1,0} & a_{1,1} \end{bmatrix}, \mathcal{B} = \begin{bmatrix} b_{-1,-1} & b_{-1,0} & b_{-1,1} \\ b_{0,-1} & b_{0,0} & b_{0,1} \\ b_{1,-1} & b_{1,0} & b_{1,1} \end{bmatrix} \tag{2.13}$$

The centre entry of the feedback template, also called *self-feedback*, is of significance importance for the stability of operation of a CNN. In this sense, it is, in many cases, desired to decompose the $\mathcal{A}$ template in Eq. (2.13) as shown in

Eq. (2.14). Matrices $\mathcal{A}^0$and $\bar{\mathcal{A}}$ are called *centre* and *surround* feedback template respectively [12].

$$\mathcal{A} = \mathcal{A}^0 + \bar{\mathcal{A}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & a_{0,0} & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} a_{-1,-1} & a_{-1,0} & a_{-1,1} \\ a_{0,-1} & 0 & a_{0,1} \\ a_{1,-1} & a_{1,0} & a_{1,1} \end{bmatrix} \tag{2.14}$$

The number of the real-valued template coefficient is dependent on the neighbourhood. We have 19 coefficients for 1-neighbourhood and 51 coefficients for 2-neighbourhood. Hence, the space of CNN templates consists of an infinite number of templates. Three simple classes are, however, of special importance and are worth mentioning [12]. These classes are briefly introduced below.

- ***Zero-feedback template:*** All feedback coefficients are zero. The dynamics of each cell of a zero-feedback CNN is described by Eq. (2.15).

$$\dot{x}_{ij} = -x_{ij} + \mathcal{B} * u_{ij} + z \tag{2.15}$$

- ***Zero-input template:*** All control coefficients are zero. The dynamics of each cell of a zero-input CNN is described by Eq. (2.16). Zero-input CNNs, also called autonomous CNNs, are widely used in pattern formation applications and autowave generation.

$$\dot{x}_{ij} = -x_{ij} + \mathcal{A} * y_{ij} + z \tag{2.16}$$

- ***Uncoupled template:*** All surround control coefficients are zero, i.e. $\mathcal{A} = \mathcal{A}^0$. The dynamics of each cell of uncoupled CNN is described by a *scalar* nonlinear ordinary differential equation as shown in Eq. (2.17)

$$\dot{x}_{ij} = -x_{ij} + a_{0,0} f(x_{ij}) + \mathcal{B} * u_{ij} + z \tag{2.17}$$

2.4 BOUNDARY CONDITIONS

The observant reader must have noticed that no restrictions have been imposed on the size of the CNN grid. Actually, the conceptual discussion carried out so far is valid for infinite CNN grids, but it suffers from a number of complications when CNNs of finite size are considered. Equations (2.8) and (2.12) are not completely defined for cells whose sphere of interest $S_r(i,j)$ extends outside of the boundary of the grid. In this sense, CNN cells can be divided into two different categories: *regular* and *boundary* cells. For a certain neighbourhood, *r*, a *regular* cell has $(2r+1)^2$ neighbour cells. All other cells with less than $(2r+1)^2$ neighbours are called *boundary* cells. Note that not all boundary cells are *edge* cells if $r > 1$ (Figure 2.5). Edge cells are the outermost boundary cells, i.e. they lie on the perimeter. The absence of neighbouring cells doesn't affect the boundary cells only, but it has, due to the nature of indirect propagation, a great impact on the dynamic behaviour of the entire network, which calls for different interpretation of boundary cell employment. Traditionally, this problem is remedied by introducing virtual CNN cells around the grid, which completes the sphere of influence of all boundary cells. Each virtual cell is associated with a virtual state, a virtual input, a virtual output and a virtual threshold [12]. These virtual

parameters are specified via various *boundary conditions*. In the following, three of the most commonly used boundary conditions for 1-neighborhood, as described in [14], are rephrased.

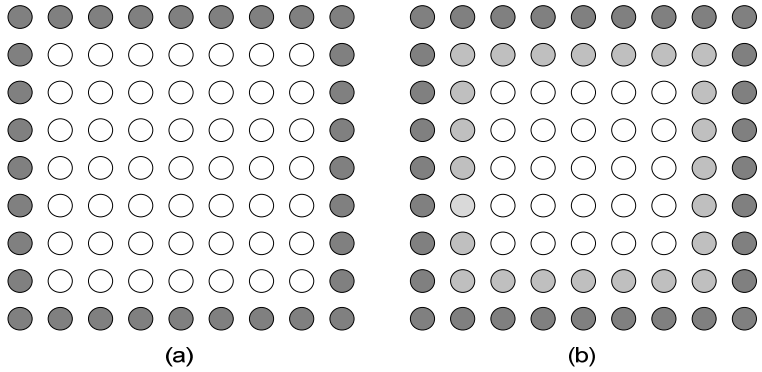


Figure 2.5 When $r = 1$, boundary cells coincide with edge cells (a) but for $r > 1$ boundary cells (light grey) are not located on the edges only (b).

- ***Fixed (Dirichlet) boundary condition:*** The boundaries of the network are tied to fixed values. In other words, virtual state and input of each virtual cell are assigned predefined constant values. This approach has been used in the first analogue realization of the basic CNN cell, which will be presented later, where the boundary is uniformly at ground.
- ***Zero-flux (Neumann) boundary condition:*** In this case virtual cells are considered to have the same state and input values as their direct neighbouring boundary cells. This condition applies usually to CNNs with no input, i.e. $u_{ij} = 0$. In principle, this corresponds to the class of autonomous CNNs (Eq. (2.16)).
- ***Periodic (Toroidal) boundary condition:*** Here the first and last rows (resp., columns) of the network are identical, as shown in Figure 2.6. Thus, the CNN behaves as if it is joined onto itself forming a torus.

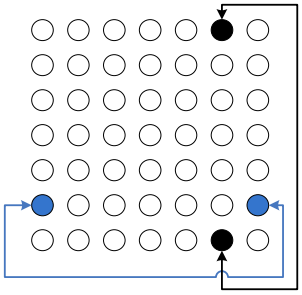


Figure 2.6 In periodic boundary condition the CNN is joined onto itself.

2.5 DISCRETE-TIME CNN

A DT-CNN poses a regular grid of locally connected cells. Once again, this grid may, theoretically, have any dimension, but in this thesis the focus is on the 2-dimensional case only. Contrary to CT-CNNs, the DT-CNN is a clocked system; whose dynamics are described by a set of discrete equations. This enforces the introduction of slightly different notations; the notations used in [40] are adopted in this thesis. It is important to emphasize that the feature of space invariance is assumed here as well. Furthermore, the size of the grid is assumed to be finite, unless it is explicitly pointed out not being the case.

A cell c is identified by the coordinate of its position in the grid, i.e. row c_i and column c_j and communicates directly with all the neighbour cells belonging to the r-neighbourhood. The definition of r-neighbourhood given in Eq. (2.1) is slightly modified to reflect the new notation of the cell but the relation remains unchanged, as depicted in (2.18). The character d represents any cell belonging to the neighbourhood of cell c, including c itself.

$$N_r(c) = \{d \in \mathbb{Z}^2 | \max(|d_i - c_i|, |d_j - c_j|) \leq r\} \tag{2.18}$$

The state of a cell c, denoted x^c, depends mainly on the contribution of the time-independent input u^d and the time-variant output y^d. Equation (2.19) depicts these dependencies at a discrete time k.

$$x^c(k) = \sum_{d \in N_r(c)} a_d^c \, y^d(k) + \sum_{d \in N_r(c)} b_d^c \, u^d + i^c \tag{2.19}$$

The real-valued coefficients a_d^c, b_d^c and i^c represent the *feedback* coefficients, the *control* coefficients and the *threshold*/*bias* respectively. While feedback coefficients a_d^c reflect the contribution from the output of all cells in the neighbourhood, control coefficients b_d^c describe the dependency on the inputs of the neighbours. The bias i^c is added to adjust a cell's threshold. Similar to CT-CNN, coefficients are commonly expressed in a compact form by means of matrices. Spatially invariant DT-CNNs are thus specified by the *cloning templat*e $\mathcal{T} = \langle \mathcal{A}, \mathcal{B}, i \rangle$ that is often thought of as an elementary DT-CNN program or DT-CNN instruction [40].

By substituting (2.11) into (2.19), a compact state equation is obtained in Eq. (2.20), which is obviously equivalent to Eq. (2.12). Because all cells in the DT-CNN have identical functionality, cell subscripts can be omitted as shown in Eq. (2.21).

$$x^c(k) = \mathcal{A} * y^d(k) + \mathcal{B} * u^d + i \tag{2.20}$$

$$x(k) = \mathcal{A} * y(k) + \mathcal{B} * u + i \tag{2.21}$$

In the case of non-zero feedback coefficients, an *initial output* $y^c(0)$ is of crucial importance for the dynamic behaviour of the network (compare with the initial condition constraint in section 2.2). On the other hand, if all feedback coefficients are equal to zero, the output of the system remains constant after the first time step.

In accordance to CT-CNN, the functionality of the system is defined by the cloning template $\mathcal{T}$ that, together with the activation pattern u and the initial output $y^c(0)$, completely determines the dynamic behaviour of a DT-CNN. Figure 2.7, that illustrates the functionality of a cell, explains schematically the influence of all involved parameters [40].

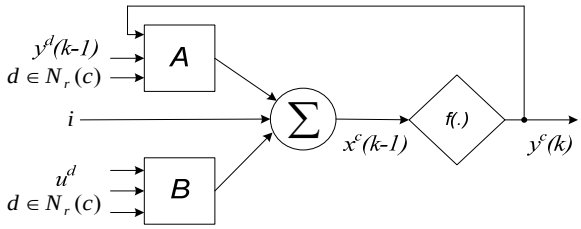


Figure 2.7 A schematic diagram illustrating a DT-CNN cell. The data comes in over the u^d input and is modified through the control template $\mathcal{B}$, while the interaction with the neighbouring cells is gathered through the y^dinput and modified through the feedback template $\mathcal{A}$. All modified input values are summed and discriminated after application of the bias i.

2.6 MULTILAYER CNN AND MULTIPLE LAYER DT-CNN

So far, only the single-layer CNN model has been considered. In this model each cell contributes with one state variable x_{ij} only. A multilayer CNN boosts the concept further where cells have several state variables, one for each layer. The emphasis is on the interaction between different state variables of the same cell. The cell-to-cell interaction is still restricted by means of the r-neighbourhood. Any layer may perform different processing tasks, whereas layers work in parallel. The set of different state variables enables the existence of concurrent multiple dynamic rules, which increases the flexibility of cellular neural networks and gives them the ability to tackle complicated signal processing problems [7].

The dynamic equations can be expressed in a compact vector form (Eq. (2.22)), where *m* denotes the number of layers, i.e. the number of state variables in each cell [14]. Here the operator $\circledast$ is to be interpreted as matrix multiplication but with the convolution operator $*$ (as defined in Eq. (2.11)) inserted between each entry of the (block triangular) matrices $\boldsymbol{A}$ and $\mathbf{B}$ and of the vectors $\boldsymbol{y}$ and $\boldsymbol{u}$ respectively.

CT-CNNs and DT-CNNs differ in the interpretation of the concept of multiple layers. Hence, we distinguish between the notations *multilayer* and *multiple layer* networks. In a multilayer CT-CNN, each cell has a number of state variables corresponding to the number of layers. In a multiple-layer DT-CNN each layer has different inputs, outputs and template coefficients. In addition to the outputs, both inputs and template coefficients are now time-variant. Obviously, employing multiple states for each DT-CNN cell is equally feasible, but this possibility is surprisingly never discussed in the literature!

$$\dot{\boldsymbol{x}}_{\boldsymbol{ij}} = -\boldsymbol{x}_{\boldsymbol{ij}} + \boldsymbol{A} \circledast \boldsymbol{y}_{\boldsymbol{ij}} + \mathbf{B} \circledast \boldsymbol{u}_{\boldsymbol{ij}} + \boldsymbol{z} \tag{2.22}$$

where

$$A = \begin{bmatrix} A_{11} & 0 & \cdots & 0 \\ A_{21} & & \ddots & \vdots \\ \vdots & & & 0 \\ A_{m1} & B_{m2} & \cdots & A_{mm} \end{bmatrix}, B = \begin{bmatrix} B_{11} & 0 & \cdots & 0 \\ B_{21} & & \ddots & \vdots \\ \vdots & & & 0 \\ B_{m1} & B_{m2} & \cdots & B_{mm} \end{bmatrix}, \boldsymbol{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_m \end{bmatrix}$$

$$\boldsymbol{x}_{\boldsymbol{ij}} = \begin{bmatrix} x_{1ij} \\ \vdots \\ x_{mij} \end{bmatrix}, \boldsymbol{y}_{\boldsymbol{ij}} = \begin{bmatrix} y_{1ij} \\ \vdots \\ y_{mij} \end{bmatrix}, \boldsymbol{u}_{\boldsymbol{ij}} = \begin{bmatrix} u_{1ij} \\ \vdots \\ u_{mij} \end{bmatrix}$$

Complex problems are divided into simpler subtasks where each layer of the network is allotted one subtask, which give rise to different coupling modes. In Figure 2.8 each layer is represented by a building block with two inputs, $u(k)$ and $y(0)$, and one output $y(k)$. The layers are interconnected in such a way that the output of one layer serves as an input or initial output to another layer, Figure 2.8.a-c. In the parallel mode, Figure 2.8.d, an additional logical function f_L combines the outputs of the layers and results in the overall network output. The function f_L is commonly realized using the AND or the EXOR operation. More complex multiple-layer systems can be constructed by combining these interconnection modes, Figure 2.8.e.

Since each layer in a multiple-layer DT-CNN corresponds to a single subtask, template coefficients are not changed during the processing. A question rises then: why are template coefficients time-variant? The only reason for introducing time-variant templates is hardware reduction and speed gain! The idea is easily understood when the output cascade mode, Figure 2.8.c, is considered. The system shown in the figure can be simply replaced by one layer, where output is fed back to the initial output, using time-variant templates to perform the operation of two layers. Using one layer with time-variant templates instead of two time-invariant layers reduces the exploited hardware and eliminates the overhead of transferring data between the layers [40].

2.7 ANALOGUE REALIZATIONS

In this section, the analogue realization of the standard CNN cell, as presented in [7], is reintroduced. Each cell, at location (i, j), consists mainly of linear circuit elements: a capacitor C, two resistors R_x and R_y, an independent current source I (that corresponds to the threshold of the cell) and a group of voltage-controlled current sources, e.g. $I_{xu}(i, j; k, l)$ and. $I_{xy}(i, j; k, l)$. A schematic view of the basic cell with all circuit elements is shown in Figure 2.9.

Voltages v_{uij}, v_{yij} and v_{xij} represent the input u_{ij}, the output y_{ij} and the state variable x_{ij} respectively. Initially, the magnitude of the state variable is

assumed to be less than or equal to 1; this is the *initial condition constraint*. Similarly, the magnitude of the input, that is obtained by the independent voltage source E_{ij} is assumed to be less than or equal to 1, but this value remains constant over time; this is the *input constraint*.

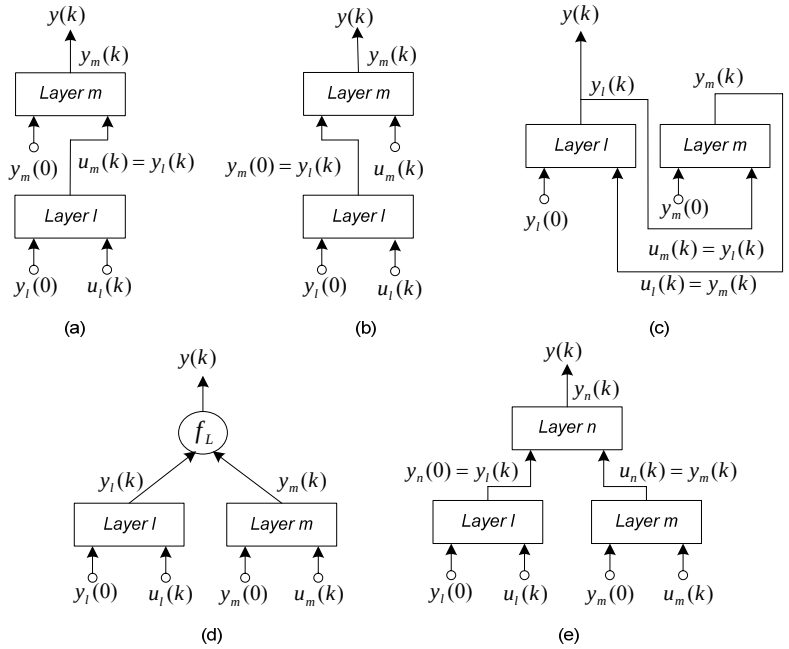


Figure 2.8 Basic interconnection modes for multiple layer DT-CNNs as presented in [40]. (a) input cascade (b) output cascade (c) feedback loop and (d) parallel. A more complex mode, parallel cascade, is presented in (e).

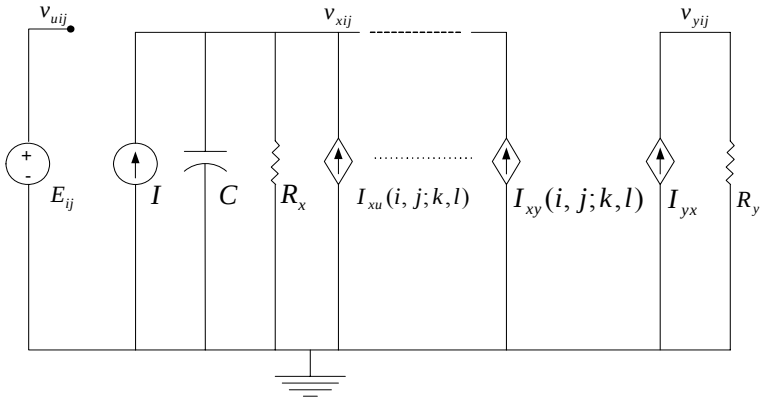


Figure 2.9 A schematic view of the standard CT-CNN cell.

The output voltage v_{yij} depends on the only non-linear element in the cell, i.e. the piecewise-linear voltage-controlled current source I_{yx} (Eq. (2.23)) with characteristic f as given in Eq. (2.7). In other words, the output of the cell is a non-linear function of the state voltage v_{xij} as depicted in Eq. (2.24).

$$I_{yx} = \frac{1}{R_y} f(v_{xij}) \tag{2.23}$$

$$v_{yij} = R_y I_{xy} \tag{2.24}$$

A cell is coupled to all cells belonging to its neighbourhood via the *controlling* voltage v_{ukl}, and the *feedback* from the output voltage v_{ykl}. In fact, the influence of any neighbouring cell on the state is obtained by means of two voltage-controlled current sources, defined by equations Eq. (2.25) and Eq. (2.26), where the coupling coefficients $A(i,j;k,l), B(i,j;k,l) \in \mathbb{R}$ correspond to *feedback coefficients* and the *control coefficients* respectively.

$$I_{xy}(i,j;k,l) = A(i,j;k,l) v_{ykl} \tag{2.25}$$

$$I_{xu}(i,j;k,l) = B(i,j;k,l) v_{ukl} \tag{2.26}$$

A formal description of the dynamics of a single cell is obtained by applying nodal analysis to the basic cell in Figure 2.9. This description, given in Eq. (2.27), represents the *state equation* of the analogue CNN cell.

$$\begin{aligned} C\frac{dv_{xij}}{dt} = -\frac{1}{R_x} v_{xij}(t) &+ \sum_{kl \in S_r(i,j)} A(i,j;k,l) \cdot v_{ykl}(t) \\ &+ \sum_{kl \in S_r(i,j)} B(i,j;k,l) \cdot v_{ukl}(t) + I \\ where\ 1 \le i \le M, 1 \le j \le N & \end{aligned} \tag{2.27}$$

In practice, the values of circuit elements C, R_x and R_y are conveniently chosen by the designer. R_x and R_y determine the power dissipation in the circuit and are usually chosen to $1k\Omega - 1M\Omega$. In fact, the dynamics of the circuit are simply scaled in time by changing the value of C only, as these dynamics are determined by $R_x C$, which is usually chosen to be $10^{-8} - 10^{-5}$ seconds [14] [7]. Currents and voltages are also scaled to fit the real design specifications. Equation (2.27) is then rewritten in order to describe the dynamics in a normalized and dimensionless manner. If the terminology of convolution described in Eq. (2.11) is adopted here as well, the resulting state equation is then identical to the one presented earlier in Eq. (2.12).

The DT-CNN cell, proposed in 1992 by Harrer and Nossek [39], is analogue as well (Figure 2.10). Similar to the basic cell introduced by Chua and Yang, it contains a number of linear circuit elements, such as capacitors, resistors and current sources. Voltages $v_u^c, v_x^c(kT)$ and $v_y^c(kT)$ correspond to variables $u^c, x^c(kT)$ and $y^c(kT)$ respectively, whereas kT represents time instances. Linear voltage-controlled current sources, such as v_u^d and $v_y^d(kT)$ are used to multiply the inputs and outputs of the neighbour cells by template coefficients.

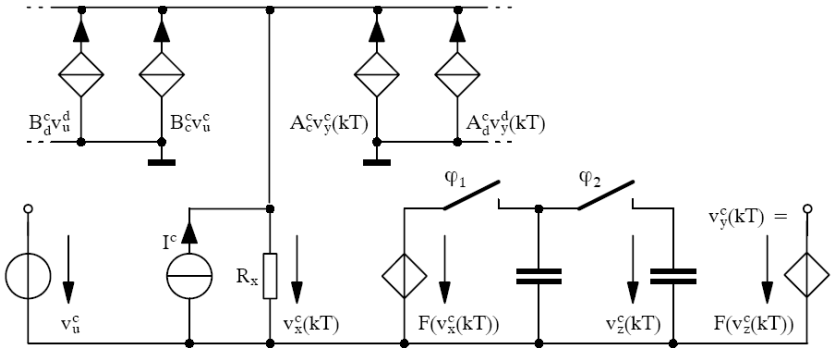


Figure 2.10 An analogue realization of a DT-CNN cell. The iterations are substituted by discrete-time instances kT and the variables $u^c, x^c(kT)$ and $y^c(kT)$ by voltages $v_u^c, v_x^c(kT)$ and $v_y^c(kT)$ respectively. T is the duration of one clock cycle.

2.8 ILLUSTRATIVE EXAMPLES

In this section, the functionality of DT-CNN is illustrated by using a number of simple templates. These, and other examples, have been presented in [40], but are described with the authors' own words. In the first two examples, the single-layered DT-CNN is used, while the third example involves the multiple-layered model. Due to the fact that grey-scale level is commonly used in image processing problems, an *input constraint* (compare section 2.2) is usually defined by restricting the input range of a cell that $u^d \in [-1, +1]$, where a value of –1 represents a white pixel, a value of +1 represents a black pixel and all other values represent grey levels in-between [40]. Here, the examples use binary images only, i.e. $u \in \{-1, +1\}$.

2.8.1 Isolated Pixel Removal

The iterative nature of the dynamic behaviour, as stated in Eq. (2.19) and Figure 2.7, is crucial to achieve the desired mapping of an input image onto an output image. However, there are few problems that can be solved by one step only. The simplest one is the *Isolated Pixel Removal*, where the aim is to remove all so-called *4-isolated pixels*: a black pixel whose orthogonal neighbours are white. In other words, the problem is characterized by a number of properties that can be summarized as: *A black pixel that has at least one black orthogonal neighbour remains black, otherwise it becomes white*. Because one picture tells more than a thousand words, the different situations are illustrated in Figure 2.11.

The aforementioned behaviour is achieved by applying the cloning template in Eq. (2.28) on a given input image u. As noticed, feedback coefficients are zero-valued, which eliminates the need of the initial output $y^c(0)$ in Eq. (2.19).

$$\mathcal{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \mathcal{B} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \qquad i = -1 \tag{2.28}$$

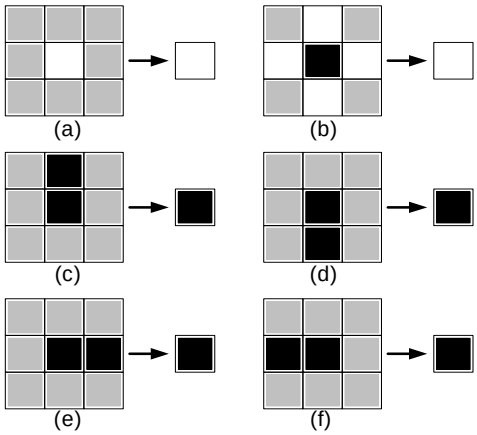


Figure 2.11 Properties of Isolated Pixel Removal applied on a centre cell with 1-neighbourhood. Grey-coloured squares represent don't-care pixels. The 4-isolated black pixel becomes white in (b), while in all other cases the presence of at least one black orthogonal neighbour helps the centre pixel to remain black.

If the orthogonal neighbours in Figure 2.11.c-f are indexed according to the convention in Eq. (2.13), the state equation of the centre cell at time $k = 0$ is obtained by substituting the coefficients from Eq. (2.28) as follows:

$$x^c(0) = 4u^c + S - 1 \qquad (2.29)$$

where $S = u_{-1,0} + u_{1,0} + u_{0,1} + u_{0,-1}$

Since input values are restricted to $\{-1, +1\}$, we have $S \in \{-4, -2, +2, +4\}$, which implies the following cases:

Case 1: When $u^c = -1$ (Figure 2.11.a) then $x^c(0) = S - 5$, which, regardless of S, implies that $x^c(0) \leq -1$. Assuming a threshold function is in use, the output $y^c(1) = -1$, which proves the case.

Case 2: For the 4-isolated pixel in Figure 2.11.b we have $S = -4$, Since $u^c = +1$ then $x^c(0) = S + 3 = -1$, which results in $y^c(1) = -1$, and thus the pixel becomes white as shown in the figure.

Case 3: If at least one of the orthogonal neighbours is black, e.g. $u_{0,-1} = +1$, we have $x^c(0) = S + 3$ where $S \geq -2$. Consequently, $x^c(0) \geq +1$ and $y^c(1) = +1$.

2.8.2 Hole Filling

One of the simplest image-processing problems whose solution still requires considering the contribution from cells beyond the r-neighbourhood, and thus involves iterative computations, is the problem of *Hole Filling*. Different definitions of a hole appear in literature. In [40], a hole is defined as the area that is completely enclosed by at least one 8-connected object, as shown in Figure 2.12.a-c. The aim is to make the pixels, belonging to the hole, black, by the end of the iterative procedure.

Obviously, the operation requires more than a set of local pixel transition rules as in Isolated Pixel Removal. A neighbourhood that may be as large as the whole image determines the colour of the pixel. A final solution is achieved by applying a set of local actions repeatedly, thus making use of the iterative nature of the dynamic behaviour of a cell, Eq. (2.19). The network is initialized with a black image, i.e. $y^d(0) = +1$ for all cells in the network, whereas the iterative process of local actions generates a wave of changing cell-outputs into white colour. The wave that propagates from the edges to the centre of the image is stopped by black input pixels, preventing it from penetrating enclosed objects, i.e. holes. Figure 2.13 illustrates the propagation of the wave step by step.

Based on the definition of the "8-connected object", terBrugge presents the following three properties for the hole filling behaviour [40].

Property 1: A white output remains white.

Property 2: A black output becomes white if the corresponding input is white and it has at least one 4-neighbour, i.e. an orthogonal neighour, whose output is white.

Property 3: In all other cases a black output will remain black.

Further, it is proposed that these properties are met by using the following cloning template:

$$\mathcal{A} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \qquad \mathcal{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad i = -1 \tag{2.30}$$

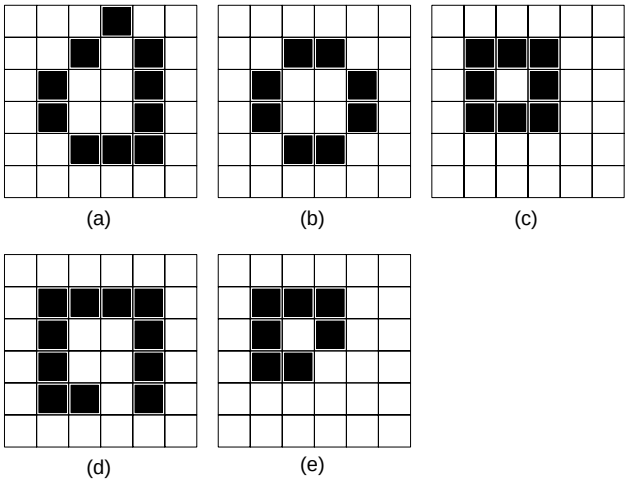


Figure 2.12 A number of holes with different sizes in (a) (b) and (c), while the absence of one black pixel makes a hole incomplete in (d) and (e).

Again, the cell state equation is obtained by substituting the template into Eq. (2.19) yielding:

$$x^c(0) = 4u^c + S + 2y^c(\mathrm{k}) - 1 \qquad (2.31)$$

where $S = y_{-1,0}(k) + y_{1,0}(k) + y_{0,1}(k) + y_{0,-1}(k)$

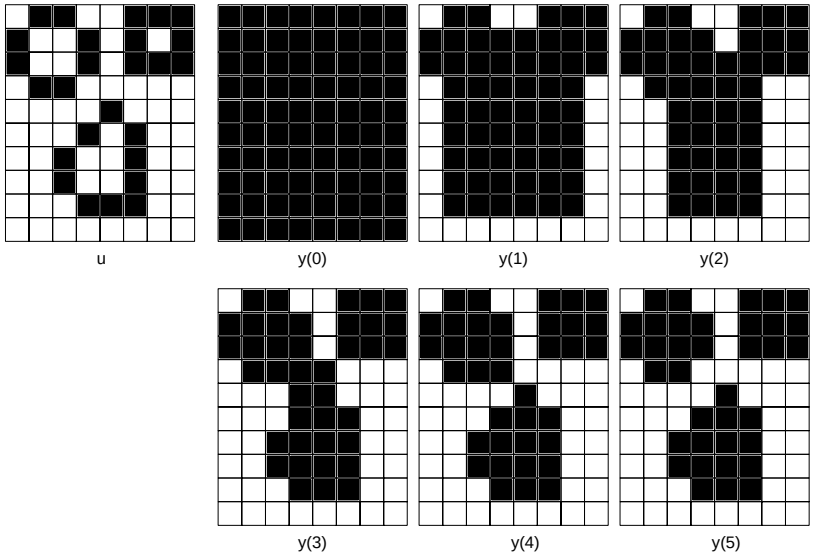


Figure 2.13 Given an input image u, the process of hole filling is initialized with a black output $y(0)$ and ends up, after five iterations, with filled holes in $y(5)$.

It remains to proof that Eq. (2.31) meets the three properties mentioned before, which is easily done by following the same approach presented in section 2.8.1. This is left for the interested reader to solve or look up at [40].

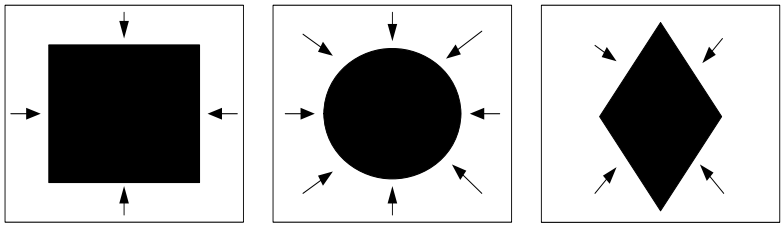
The behaviour of the template of Hole Filling presented above is coupled to the boundary condition in use. Simple tests show that the format of the white wave started at boundary cells depends on the boundary condition.

In the case of a fixed boundary condition, the following behaviour is observed with $u_{virtual} = -1$:

- ★ If $x_{virtual} = -1$, a black rectangle that is as big as the network upon start, shrinks toward the centre of the CNN (Figure 2.14.a).
- ★ If $x_{virtual} = 0$, the rectangle becomes an ellipse (Figure 2.14.b).
- ★ If $0 < x_{virtual} < 0.5$, the wave gives rise to a black rhomb that shrinks toward the centre. It is obvious that the wave starts from the four corners (Figure 2.14.c).

Starting from $x_{virtual} > 0$, the sides of the rhomb get more convex as $x_{virtual}$ increases until it equals 0.5. No wave is then generated and the template does not function at all, while changing input values of the virtual cells does not have any impact on the operation. Furthermore, the speed of convergence

decreases with increasing $x_{virtual}$. It seems that the states of the boundary cells become closer to the black value of $y(0)$. On the other hand, a Zero-flux boundary condition makes the template ineffective, regardless input and state values in use. The effect of boundary conditions is discussed in detail in Chapter 8.



(a) (b) (c)

Figure 2.14 Different state values of the fixed boundary condition force the white wave to propagate differently.

2.8.3 Hole Extraction

As the name implies, the problem consists of extracting the holes of an object. Using the input image u from Figure 2.13, the output of the *Hole Extraction* operation is given in Figure 2.15. The problem cannot be solved by a single-layer DT-CNN. The desired output is obtained by either using time-variant template coefficients or by adding a second layer with a time-invariant cloning template. Regardless the used approach, the problem demonstrates the expressive power of the concept of multiple-layer DT-CNN, presented in section 2.6.

In line with [41], the approach of time-variant template coefficients is used. The first phase of the solution consists of filling the hole by using cloning template (2.30). After the network has converged, the output contains the image in which all holes are filled. Applying another cloning template, presented in Eq. (2.32), a selection of all black pixels in the output image that are white in input image completes the task. The initial output, $y(0)$, equals the resulting image of hole filling template, while the input, u, is the original image in Figure 2.13.

$$\mathcal{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \mathcal{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad i = -1 \tag{2.32}$$

2.9 SUMMARY

A CNN is a regular array of many identical cells. Each cell has a simple function that takes an element of a topographic map and then interacts with all cells within a specified neighbourhood, each corresponding with neighbouring map elements. An isolated cell c at position (i, j) in the grid consumes an input u_{ij} together with a previous output $y_{ij}(t)$, where t represents time step, in order to produce a new output value. In a standard CNN, the current output of each cell depends on input and output values, denoted u_{kl} and y_{kl} respectively, from the

surrounding cells as well. This is achieved by simple additions of weighted contributions ruled by parameters called template. The key feature in a CNN is the direct interaction with the 8 neighbouring cells: value passing occurs in two directions. This defines the 1-neighborhood of the centre cell c, denoted $S_r(ij)$. The concept can be intuitively extended to the next level of neighbouring cells, which leads to 2-neighborhood (Figure 2.16). The union of all neighbourhoods gives the entire network. In DT-CNN, the notation is slightly different to reflect the discrete-time nature. Input and output values are denoted u^c and $y^c(k-1)$ for cell c, where k represents disrcete time step. The neighourhood is denoted $N_r(c)$.

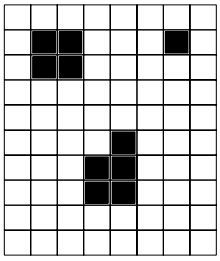


Figure 2.15 Resulting output after applying the operation of hole extraction on the input image of Figure 2.13.

The state of a cell, denoted x_{ij} or x^c, depends mainly on the time-independent input u to its neighbours and the time-variant output y of these neighbours. Equation (2.33) describes this dependence in a continuous time t, while Eq. (2.34) describes the discrete counterpart. In both equations a CNN of M rows and N columns is considered. The control coefficients b only "scale" the inputs, while the feedback coefficients a are responsible for the non-linear dynamical behaviour. A real valued cell bias z (or i) is added to adjust the threshold. These coefficients are usually combined to compose matrices, which results in a so-called cloning template $\mathcal{T} = \langle \mathcal{A}, \mathcal{B}, z \rangle$ (or $\langle \mathcal{A}, \mathcal{B}, i \rangle$.). In general, the template may differ for different cells in the network, but the majority of CNN applications use space-invariant templates.

$$\frac{dx_{ij}}{dt} = -x_{ij}(t) + \sum_{kl \in S_r(ij)} a_{kl}\, y_{kl}(t) + \sum_{kl \in S_r(ij)} b_{kl}\, u_{kl} + z \qquad (2.33)$$

$$where\ 1 \le i \le M\ ,\ 1 \le j \le N$$

$$x^c(k) = \sum_{d \in N_r(c)} a_d^c\, y^d + \sum_{d \in N_r(c)} b_d^c\, u^d + i \qquad (2.34)$$

The main advantage of both continuous-time and discrete-time CNNs relies on the local interconnectivity of the simple cells. This feature makes CNNs, in general, suitable for VLSI implementations. Though the elementary operations are very simple, this does not necessarily mean that the circuit should be correspondingly simplified. For instance, ter Brugge concludes in [40] that it could be more meaningful to use more complex operators.

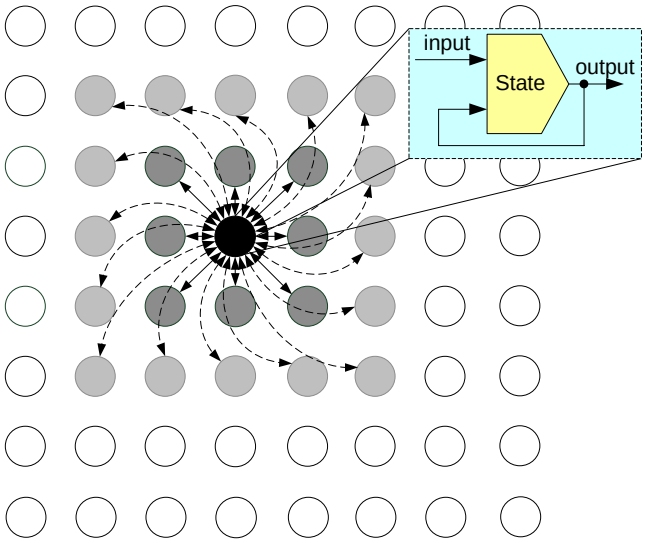


Figure 2.16 Dark gray cells along with the black cell constitute the 1-neighborhood, while adding light gray cells build the 2-neighborhood. The arrows represent the dual communication lines.

Finally, the feature of full parallelism (considered as one of the most important advantages of CNNs) is captured by means of hardware realizations only. Software implementations running on a standard PC, even those considered as real-time implementations, loose the benefits of real parallelism, and are, thus, used for simulation purposes only [40].

Chapter 3

Hardware Implementations
State of The Art

The ability of solving real-life problems has always been desired but not fully possible as these problems are characterized by being too complex and time-consuming tasks for classical digital computers. The partial success that regular analogue processing arrays face in a number of fields such as neural networks [8] has attracted interest worldwide and encouraged to take the step toward the implementation of programmable analogue arrays that can handle general real-time problems. A CNN is practically able to perform all types of convolutions/correlations due to its programmability in term of different cloning templates. Many physical phenomena can be translated into CNN algorithms and thus performed in a finite spatial window [17]. In this sense, the CNN seems to be an ideal framework for programmable analogue array computers. The first step has already been made by, e.g., the analogue implementation of the CT-CNN standard cell (section 2.7).

Fully connected analogue neural networks suffer from the number of connections and the distance that these connections need to propagate, which makes them very difficult to fabricate. In contrary to neural networks, CNNs are characterized by local connectivity. The adoption of the concept of nearest neighbour interactions found in Cellular Automata [10] allows for the arrangement of the cells in regular grid with equidistance. The routing and layout problems usually faced in traditional analogue circuits are then easier tackled in analogue CNN VLSI implementations. A cell is designed and replicated to form a regular network that is placed and routed rapidly. The first VLSI implementation of a CNN [16] has, naturally, been based on the analogue model of the standard cell as presented by Chua (section 2.7). Fabricated circuits

come usually with parasitic capacitances and resistances, which in many cases leads to undesired behaviour. In order to reduce the sensitivity of the cell to such fabrication deviations, the dynamics of the cell are dominated by large state capacitor. Furthermore, the state capacitors affect the initialization procedure. All cells cannot be loaded with initial states simultaneously, but a single row of state capacitors (cells) is loaded at a time. While initialized, the state capacitors have to be disconnected from the remainder of the cells in order to prevent their voltages from dropping to such a level that it may affect the processing. Another issue has to do with the degree of adaptability. The experience gained from neural network VLSI implementations shows that high flexibility is difficult to achieve. Hence, the CNN array is not 'programmable', i.e. the array is designed to perform one or a related set of processing functions using fixed coefficients. Complex tasks are proposed to be solved by cascading or paralleling multiple CNN VLSI devices! This is, apparently, not practical and removes most of the attraction of the CNN VLSI implementation as time and cost are then much higher in comparison to other established systems. Next, due to fabrication and the available VLSI technology issues, only small CNN chips (20×20) were realized, although larger and more sophisticated chips were expected to appear in due time.

Soon enough, advanced hardware technology allowed a wide range of concepts, models and architectures to see the daylight. Thus, introducing these forerunners and highlighting the pros and cons of their hardware implementations is highly desired. The aim is to open for better understanding of the deployment of the solutions that are discussed later on in the thesis. In this chapter, a brief description of the implementation of the most important concepts is given. First, CNN emulators that are built around of-the-shelf DSPs are introduced in section 3.1. CNN Universal Machine (CNN-UM) that provides a roadmap toward exploiting the intrinsic supercomputing features of the CNN is discussed briefly in section 3.2. Subsequently, a chronological review of the most known full-custom mixed-signal realizations of the CNN-UM is given in section 3.3. Section 3.4 emphasizes the need of fully digital implementations by presenting a number of digital CNN-UM emulators. Finally, the chapter is closed by a concluding summary in section 3.5.

3.1 DSP-BASED CNN EMULATORS

As mentioned above, the first attempt toward a VLSI implementation of a CNN has been presented by Yang and Chua [16]. Meanwhile, Roska et al. [17] have developed a hardware accelerator board (CNN-HAC), mainly for hardware simulation. The reported performance exceeds the one provided by any, at time, available software simulation due to the claimed ability of hosting around 1 million processing elements (PEs). Further, it is claimed that the flexible programmability enables handling of complex tasks. For instance, performing nonlinear and delay-type templates [15] is now possible. Thus, the proposed design provides an interest trade-off between speed, programmability and complexity. It is based on emulating the parallelism of operation using a number

of DSPs, each performing the functionality of a number of *virtual* digital processors corresponding to the actual cells in the grid.

The usage of digital processors requires a transformation of the analogue values into digital ones. This requires a discretization of the state equation (Eq. (2.8)) in the discrete time n, which is carried out using the forward Euler formula with a step size $h \geq 1.5$ as shown in Eq. (3.1). After analysis and extensive software experimentation a 16 bits fixed-point representation of values shows to be the best choice. A local memory per digital processor stores input and state values (v_{uij} and v_{uij} respectively) for all cells within the neighbourhood. Output values v_{yij} are not stored as they are easily obtained using the local piece-wise discrimination function.

$$\begin{aligned} v_{xij}(n+1) = (1-h)\cdot v_{xij}(n) + h\cdot \sum_{kl\in S_r(i,j)} A(i,j;k,l)\cdot v_{ykl}(n) \\ + h\cdot \sum_{kl\in S_r(i,j)} B(i,j;k,l)\cdot v_{ukl}(n) + h\cdot I \\ where\ 1\leq i\leq M, 1\leq j\leq N \end{aligned} \tag{3.1}$$

The focus is on using the accelerator for image processing purposes. Thus, the network has to contain a large number of cells, which is contrasted with the limited number of digital processors that a chip may contain. Each physical processing element, i.e. the DSP, performs the functionality of a large number of cells or more precise their corresponding virtual digital processors. This requires the local memory storage to be so large that it can hold the data needed for all virtual processors. An indirect effect is a separation of communication and calculation needs for the group of virtual processors that is mapped on a single DSP. The simultaneous two-ways communication between the virtual groups constitutes still a performance bottleneck. Therefore, the grid of cells is decomposed such that a vertical band of virtual processors are assigned to a single physical processor (DSP). In this case, communication is reduced from 8 down to 2 directions only, i.e. right and left.

The actual hardware implementation is based on a PC add-on board with 4 DSPs from Texas Instruments [18]. Each DSP has a dynamic memory to store processor program code in addition to the cloning templates and the appropriate data values as signed 16 bit numbers. All parts of the board are supervised and controlled through a PCI bus that is responsible for the interfacing to an IBM PC bus. Two FIFO blocks, each having 512×16 byte effective storage capacity, provide the communication path between each pair of DSPs. Next to components hardware, the accelerator consists of a software part. The software is allocated both to the host, e.g. data acquisition and visualization of the results, and to the board, e.g. iterating Eq. (3.1). Communication routines and interrupt handlers are necessary on both sides. The accelerator shows an average computing speed of 200 msec/iteration when images of size 20,000 pixels are processed.

The experience gained from building the CNN-HAC in addition to the fact that analogue VLSI implementations provide limited accuracy only has

encouraged to develop the Analogic CNN Emulator Engine (ACE) [22], a new emulator based on floating-point digital signal processors. Similar to CNN-HAC, the ACE accelerator is built as a PC add-on board that is connected via a SCSI (Small Computer System Interface) to a host PC, where data and results of the simulation are stored and displayed. The choice of a 16 bit wide SCSI interface is essential to guarantee a high speed, flexible and versatile communication. Further, the architecture is based on a number of built-in floating-point DSPs from Texas Instruments, where every two DSPs are grouped in a single computational unit with appropriate memory modules and control logic (Figure 3.1). This allows for emulation of large dynamical systems with a 4-bytes floating-point accuracy, with the ultimate goal of achieving quasi real-time performance. In line with CNN-HAC, the usage of digital processors requires here as well a discretization of the state equation as carried out in Eq. (3.1).

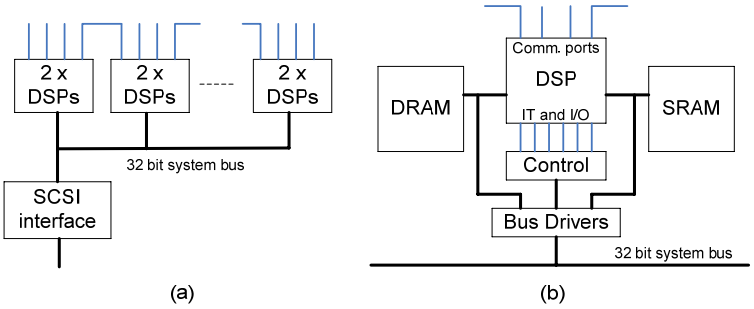


Figure 3.1 The architecture of the ACE board (a) and a single DSP with corresponding storage and control units (b).

The computation is made efficient through employment of two hierarchical levels of parallelism: the multiprocessor level and the operational level. The former exploits a data type parallelism where individual processors (DSPs) compute new values independent of the others. The computational power of this level is directly proportional to the number of processors and the inter-processor communication due to virtual processors mapping. A horizontal band decomposition of the 2-dimensional input array, similar to the one used in CNN-HAC, is adopted here to minimize the communication among the processors. The operational level is directly supported by the complex pipeline architecture of the used DSPs. Special attention should be paid to avoid address register and memory conflicts (most frequent pipeline conflicts [18]) during the internal loop of the calculation of the state equations.

Each physical processor (DSP) has two types of local storages: a large capacity DRAM, of size 4-32 MByte, that stores input and output data arrays, and a smaller SRAM module that serves as program memory and temporary storage for calculation results (Figure 3.1.b). During computation, the current data block is moved from the DRAM to a high speed internal memory (not shown in the figure) or to the local SRAM.

The ACE comes together with a library of CNN routines containing cloning templates, inter-processor communication routines, DRAM-SRAM-caching routines, floating number format conversions, and local and status report routines. The routines are written in C where the most time-sensitive ones are manually optimized and checked for pipeline conflicts in assembly level. With a user interface, a menu-controlled CNN-simulation environment and other monitoring features, the ACE board is easily controlled and accessed from the host PC. Due to the combined memory storage of maximum 512 MByte (16 32-MByte DRAM modules), the engine is able to store a CNN cube of size $512 \times 512 \times 512$ (virtual) cells, thus, providing a promising computation power that is able to handle large dynamical systems in quasi real-time [22].

3.2 CNN UNIVERSAL MACHINE

The CNN Universal Machine (CNN-UM) has been introduced in 1992 by Chua and Roska as the first algorithmically programmable analogue array computer having real time and supercomputer power on a single chip [20]. The system is universal in the sense of a *Turing machine*, with the stored programmability, i.e. the algorithm, as the underlying mechanism.

The concept of stored programmability is usually built on a number of assumptions. First, all the transients stabilize within a specific clock cycle, and all the signals remain within a prescribed range of dynamics [15]. This can, obviously, be assured in a CNN. The complete stability of the CNN has been guaranteed by most of the useful cloning templates found in literature. Even if the functionality is dependent on the size of the network, e.g. propagation type templates, the convergence time can be estimated independently of the input data. Furthermore, the area needed for storing an instruction is much less than the area occupied by the processing unit, while changing an instruction requires a negligible time compared to the instruction execution time [19]. For regular analogue arrays, e.g. neural networks, this is feasible only if the number of processing units is small. When the size of the array grows, the number of weights increases more than linear. A large-scale realization is, obviously, impractical from the storage area point of view. In contrary, the number of "weights" in a CNN is ruled by the size of the neighbourhood; 19 for $r = 1$ or 51 for $r = 2$; regardless of the size of the array. The capacity required by the global analogue program storage in a CNN-UM is therefore affordable.

A key feature in the CNN-UM is the "dual computing" paradigm. Analogue array processing is combined with logic operations that only involve the symbolic variables YES/NO; therefore denoted *analogic* computing. All signals and operators are either analogue or logic, which in principle removes the need for A/D and D/A conversions. In order to achieve this, the standard CNN cell that has been presented in section 2.2 is modified. Each cell is programmable by means of a number of switches that are, together with some logic elements, added to the analogue core resulting in a so-called *CNN nucleus*. The nucleus is conceptually divided into two parts: analogue and logic. These parts are connected by a binary converter (B/U) converting a bipolar analogue signal in

interval $\{-1,1\}$ into a unipolar signal in interval $\{0,1\}$. To keep the global I/O interaction at a minimum, local analogue memory (LAM) and local logic memory (LLM) components are added. Hence, intermediate analogue and logic values are locally stored, which facilitates the implementation of algorithms consisting of a sequence of cloning templates to be performed. Analogue values stored in different LAMs can be combined into a single value by means of a local analogue output unit (LAOU). In a multilayer CNN, the LAOU can be used to combine the results from the different layers. Similarly, the logic values are combined using a local logic unit (LLU). An illustrative view of the usage of an LLU is given in Figure 3.2 where the B/U converter is seen as well while Figure 3.3 illustrates the analogue part as presented in [19]. Apparently, the existence of local storages is the cause for the reported extreme computing power of the CNN-UM.

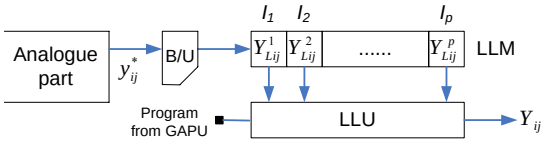


Figure 3.2 Local logic memory cells are combined using a local logic unit (LLU). The B/U converter converts a bipolar analogue signal into a unipolar signal. The small black square connected to the LLU indicates instruction path from a global controller.

The nucleus receives programming instructions through a local communication and control unit (LCCU). These instructions include the analogue values of the cloning template, logic function codes for the LLU and switch configuration specifying signal paths internally in each nucleus and the settings of thresholding function unit and the LAOU. The combined architecture of the nucleus and the surrounding components is called the *extended CNN-UM* cell. Table 3.1 summarizes the notation and functionality of the main components of the extended cell.

Table 3.1 The main components in the extended CNN-UM cell.

Acronym	Description
LAM	The local analogue memory stores intermediate analogue values locally.
LLM	The local logic memory stores intermediate logic values locally.
LAOU	The local analogue output unit combines the different local analogue values to a single output value.
LLU	The local logic unit combines the different local logic values to a single logic value.
LCCU	The local communication and control unit receive the global programming instructions and decode them. It is, e.g., responsible for keeping the switches opened/closed

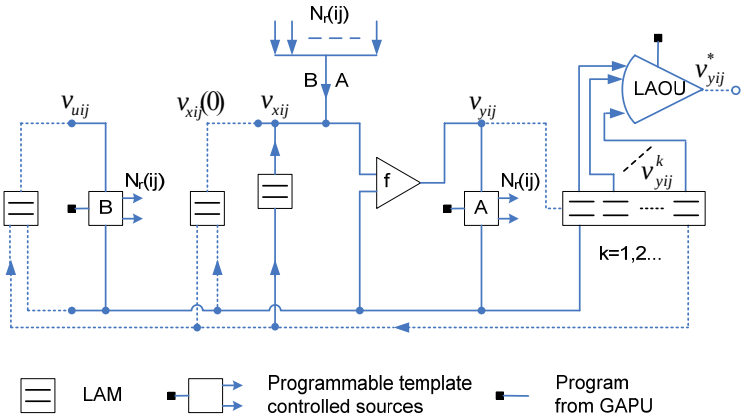


Figure 3.3 The analogue part of the extended cell. Dashed lines show the possible paths that are controlled by switches (not shown here) whose configuration is coded in LCCU.

Each extended cell in the grid is programmed and controlled by a global analogic programming unit (GAPU) as already has been indicated in Figure 3.2 and Figure 3.3. The GAPU consists of 4 sub-units: the analogue program register (APR) stores all cloning templates in an analogic algorithm, while the logic functions are stored in the logic program register (LPR). The switch configuration register (SCR) stores the configurations that are later coded by LCCU in each cell to control the internal switches. The LPR and SCR control the multi-input single-output units LLU and LAOU respectively. Finally, the sequence of analogic instructions, indicating in which order the different templates and logic functions are applied, is stored in the global analogic control unit (GACU) that constitute the fourth sub-unit of the GAPU.

Theoretically, any implementation of the CNN-UM will be characterized by an unprecedented computing power due to the inherited massive parallelism. The performance of the obtained CNN Universal Chip (CNN-UC) may however be degraded due to the distribution of global input and output signals. There is a one-to-one geometric correspondence between input (and output) signals and the cells. For a large-scale implementation, the time needed to bring input signals into the individual cells must be at minimum. Otherwise, the desired high throughput rate will be never achieved. The authors in [19] propose that a fully parallel input is possible by allowing each cell to have its own sensory input integrated on the chip. Light intensity, temperature and chemical properties are possibly captured by the sensors. Electromagnetic detection, e.g., can be used to detect certain output features captured by means of antennas connected to each cell.

Similar to the relationship between a classical digital processor and an operating system, the promised computation power of a (CNN-UC) is heavily dependent on the analogic CNN software. Not surprisingly, the implementation

of an analogic algorithm on a CNN-UC or any of its digital emulators follows then the traditional methodology in classical software. One starts with defining the algorithm by means of a flow diagram that makes use of an *analogic language*, e.g. the ACL [21]. Such a language must specify names and values of signals, instructions and parameters, subroutines, and programs. Now a CNN *analogic compiler* (e.g. Alpha Compiler [37]) produces codes for the target platform (emulators or simulators).

3.3 FULL-CUSTOM MIXED-SIGNAL CNN-UM CHIPS

Since the introduction in 1992, the concept of CNN-UM has been considered extremely attractive to realize electronically due to its universality and ability of implementing the most complex CNN applications. Several realizations have seen the light of day, some focusing on analogue only or mixed-signal implementation in CMOS, while others following the footprints of predecessor emulators CNN-HAC and ACE. In this section, the mixed-signal type is considered with focus on a specific chip series mainly developed by a group at Centro Nacional de Microelectónica at University of Seville in Spain [23]-[30], while the next section introduces two of the most famous fully digital emulators.

A number of drawbacks in previous CNN implementations have been reported [24]. One has to do with the difficulty of electrical cell design due to the various ranges for the internal voltages and currents. These ranges have to be considered in order to reduce the influence of MOS transistor nonlinearities. Another issue is that input signals are always voltages while internal signals may be voltages or currents. This is crucial in focal plane architectures where sensors provide the signals in form of currents. Incorporation of the sensory and the processing circuitry on the same semiconductor substrate is pretty common [33] as CMOS technologies offer good photo transduction devices [34]. A conversion into voltages is then needed, which complicates the CNN interface design. Finally, the combination of internal voltage and current signals leads to internal high-impedance nodes and, hence, large time constants. This results in a lower operation speed than desired.

Attempting to overcome these limitations, a new CNN model, i.e. Full Signal Range (FSR), has been introduced [31][32]. Here, all variables are in the form of currents, thus, eliminating the need of current-to-voltage conversion. The main difference compared to CT- and DT-CNNs is found in the way state variables evolve. State variables have the same variation range as input and output variables, i.e. $x^c \in [-1, +1]$ independently of the application (Eq. (3.2)). This results in a reduced cell complexity for both CT and DT cases and, thus, reduces area and power consumption in VLSI implementations. Stability and convergence properties are guaranteed and proven to be similar to the original models. It is further shown that uniform variations of the coefficients of the cloning template affect only the time constant of the network [32].

As has been mentioned before, the flexibility and generality of the CNN-UM lies in the ability to freely reprogram the system using distinct analogic parameters, i.e. different cloning templates and logic functions. This is

guaranteed in the proposed design in [23] through a synergy of analogue and digital programmability. Internally, all cells are equipped with an analogue-programmable multiplier, while digital control signals are provided externally, i.e. outside of the cell array. A specific interface circuitry is required to generate the internal weights from the corresponding external digital signals. The interface is located at the periphery of the cell array and behaves as a nonlinear D/A converter. The analogue weights are gradually adapted to the desired level and then used to control the analogue multiplier within the cells in the array. Each peripheral weight tuning stage consists of an analogue controlled multiplier and a digital controlled multiplier connected in a feedback loop through an integrator. Figure 3.4 illustrates the functionality of the tuning interface and shows the global lines. For this purpose, global routing channels are used. Notably, only 10 global channels are needed, 9 for control/feedback coefficients and 1 for the bias. The gained benefits are many: low area as only one tuning interface is needed for the whole array, fewer control lines because analogue weights require less lines compared to the digital weights with same accuracy, and the simplified realization of the APR (section 3.2) using a digital RAM memory. Consequently, the external management of the chip can be completely digital.

$$\frac{dx^c(t)}{dt} = \begin{cases} \sum_{d \in N_r(c)} \{a'^c_d y_d + b^c_d y_d\} + i^c & for\ all\ |x^c| < 1 \\ 0 & for\ all\ |x^c| = 1 \end{cases} \qquad (3.2)$$

$$with\ a'^c_d = \begin{cases} a^c_d & c \neq d \\ a^c_d - 1 & c = d \end{cases}$$

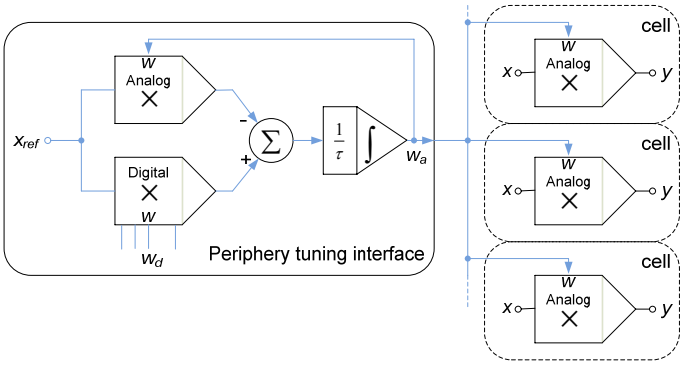


Figure 3.4 The tuning D/A interface is located at the periphery of the cell array. It uses the digital weights w_d to generate the corresponding analogue weights w_a that are brought into each cell in the array using global routing channels.

The cells can be initialized optically through photo-sensors or electrically, while output values are downloaded in electrical form only. In the first realized chip [23] that is fabricated in 1 μm CMOS technology and contains 32 × 32 cells, input or/and output values are downloaded and uploaded through 32 I/O bonding pads, on a row by row basis. Apart from that, the realization follows the

main lines defined in the conceptual CNN-UM. The digital circuitry at each cell includes a four-bit static memory LLM (storing 4 pixels belonging to 4 different images as each pixel is represented by a binary value), a completely programmable two-input digital LLU in addition to the LCCU. Notably, the LLM memory replaces the LAM memory in the general CNN-UM as only binary values are handled. The APR and LPR reside in a single on-chip RAM with a capacity of 8 instructions. Each instruction contains 19 8-bit template coefficients (7 bits plus sign), 2 2-bit boundary-condition values and one 4-bit local logic truth table.

It is worth mentioning that the computational part of the cell, performing the local convolutions in Eq. (3.2), is totally analogue and occupies about 70% of the cell area. It consists of different functional blocks: 9 programmable interconnection synapses (multipliers), an integrator, nonlinearity and a memory. The synaptic inputs are voltages, which eases the distribution of analogue template coefficients through the global lines, and the internal distribution of the cell's own state to all synapses. As synapses are connected to the integrator, their output is a current instead [24].

In time, CMOS technology allowed for accommodation of more cells and/or higher complexity on a single chip. In a chip fabricated in 0.8 μm CMOS technology [24], both global instruction memory and local data memory are made dynamic in order to increase the flexibility of operation. This leads however to a smaller array, where only 20×20 cells are available. Moving to 0.5 μm CMOS technology increased the size to 64×64 cells [25], in spite of the added ability of handling both analogue (greyscale) and digital (binary) inputs. Further improvements include simpler intracellular analogue synapses, i.e. multipliers, a more complex but highly accurate non-linearity device to obtain the cellular output, and a 4 fold larger global instruction memory. This chip has gotten the name CNNUC3 in [26], but is later on renamed to ACE4k in [27] to reflect the total number of accommodated cells. For same reason, the 0.8 μm chip is renamed ACE400 in [28]! The used notation is somewhat confusing as the architectures do not have any relation to the previously discussed ACE engine (section 3.1)!

Few years later, 2002, a new chip is fabricated using 0.35 μm CMOS technology. Following the latest naming convention, the chip is called ACE16k [28] because it accommodates 128×128 cells. The ACE16k (illustrated in Figure 3.5) is proudly introduced by the authors being a clear advance in a roadmap toward flexible vision systems on chips (VSoCs) [28]. The major improvements of the new chip compared to the previous ACE4k are:

- Digital buses are incorporated for greyscale input values, which allows for a fully digital interfacing.
- A hand-shaking protocol eliminates timing constraints.
- An internal bus, ACE-BUS, simplifies the communication among functional blocks within the cell.

- Two out of the four LLMs in ACE4k are replaced by 4 additional LAMs.
- Dynamic, instead of static, digital memories are used to store templates.
- The optical input module is reconfigurable and is flexible enough to operate under very different illumination conditions.
- The capacity of the chip is 4 times larger than the ACE4k with larger functional capabilities.
- Finally, blocks can be switched to idle in order to save power.

Moreover, ACE16k employs a modified interaction pattern among cells. The number of synaptic analogue multipliers in each cell is increased to 12 instead of 8 used in ACE4k. The additional four multipliers are introduced to increase robustness in templates where the central entry is much larger than off-centre coefficients. In this case, the synaptic multiplier corresponding to the centre coefficient has to be driven by a quite higher voltage, which will give rise to mismatch-induced errors. Each of the central multipliers is actually a parallel aggregation of two regular synaptic multipliers allowing for double strength and, thus, increasing the difference between weight voltages.

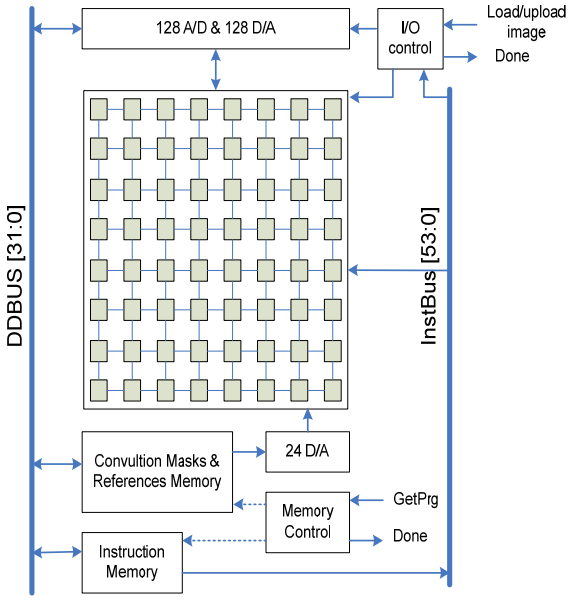


Figure 3.5 A conceptual architecture of ACE16k.

Table 3.2 presents a summary of the most relevant features of mixed-signal chip realizations. When it comes to the first two implementations, later publications of the same authors in Seville show slightly different figures (see e.g. [27] and [28]). This may depend on more exhaustive testing and more exact measuring of the different parameters of the fabricated chips. The table reveals

that all chips are far too small to handle a single frame. This drawback is important as all chips were developed with image processing capability in mind. The problem is solved by adopting the concept of windowing and time multiplexing, where large images are divided into sub-frames that are handled in sequence.

Table 3.2 Comparison of mixed-signal full-custom CNN universal chips. All chips use a modified CNN model, i.e. the FSR model.

		CNNUC1 [3] **[23]**	ACE400 **[24]**	ACE4k **[26]**	ACE16k **[28]**
CMOS technology		1 μm	0.8 μm	0.5 μm	0.35 μm
Density (cells/mm^2)		33	27.5	82	180
Array size		32 × 32	20 × 22	64 × 64	128 × 128
Input	Type	Binary	Binary	Binary & Greyscale	Binary & Greyscale
	Optical	√	√	√	√
	Electrical	√	√		√
Output	Type	Binary	Binary		
	Electrical	√	√		
Global instr. memory		Static	Dynamic	Static	Dynamic
# Ana. instructions		8	8	32	32
# Dig. instructions			0	64	64 × 64
Local memory	Type	Digital	Digital		Dig.&Ana.
	Dynamic		√		√
	amount	4 Binary (1-bit)	4 Binary	4 Binary 4 Grey	2 Binary 8 Grey
Ana. Acc.	Input	–		8 bits	8 bits
	$\mathcal{A}$ & B	7 bits + sign	7 bits + sign	7 bits	7 bits + sign
	bias	7 bits + sign	8 bits + sign	N/A	7 bits + sign
Ana. circuit area/cell		N/A	70%	N/A	
Cell array area/chip		N/A	53%	58%	
Cell area		180 × 170 μm^2	190 × 190 μm^2	120 × 102.2 μm^2	73.3 × 75.7 μm^2
Power	Entire chip	N/A	1.1W @ 5V	1.2W @ 3.3V	< 4W@ 3.3V [29]
	Per cell	N/A	N/A	370 μW	180 μW

[3] This architecture was not given any name in [23], but is called CNNUC1 here to emphasize that it was the *first* Universal Chip of the series.

It is worth mentionening that there are SIMD-based CPAs that are capable to compete with the ACE-serises in both size, accuaracy and power consumptions. For instance, the SCAMP vision chip [5] that is fabricated in $0.6\ \mu m$ technology accommodates 21×21 PEs. It has a peak power dissipation of $40\ mW$ at $3.3V$, while the maximum power per PE is as low as $85\ \mu W$. This is to be compared to the $180\ \mu W$ for the ACE16k chip that is implemented in $0.35\ \mu m$ technology. It is further claimed in [5] that 'future' chips are estimated to have a 256×256 array fabricated in $0.18\ \mu m$ technology. With a total chip area of $76\ m^2$ and power dissipataion of $2\ W$ per chip, this seems to make a milestone that all CNN chips have to beat!

3.4 DIGITAL CNN-UM EMULATORS

There is no doubt that the previously presented full-custom chips provide a powerful framework to handle CNN operations. The impressive computational speed of 330 GOPS in the ACE16k chip is comparable with the capability of modern supercomputers. But theses chips suffer from the 'limited' accuracy of the analogue signals (7-8 bits only). Another issue is the high cost as few chips only are fabricated. In addition, the development time window is wide which increases the cost further. Furthermore, analogue devices in general are known to be sensitive to fabrication artefacts, which in the case of CNNs may lead to complete failure. In this sense, fully digital architectures provide a good trade-off between computational speed on one side and versatility and cost on the other side. Such architectures have a much shorter design cycle as they utilize standard digital CMOS technology.

With confidence, one may consider the CASTLE architecture [35] as a representative of the class of fully digital emulators. The architecture is capable of performing 500 CNN iterations using 3×3 templates on a video stream with frequency of 25 fps taking 240×320 pixels each. This is valid for a system with 24 processing units (PEs) with precision of 12 bits. CASTLE makes use of the FSR model where the absolute value of the state variable is never allowed to exceed the value of +1. Recall that the value of the output and state values always coincide. The discretized state equation is obtained by applying the forward Euler formula as shown in Eq. (3.1).

Loading input pixels on-the-fly from an external memory into the processing array constitutes a performance bottleneck. On the other hand, storing the entire image on chip is impossible due to the limited resources. Instead, the image is divided into a number of belts with a height of $2r + 1$ pixels where r represents the neighbourhood. Each belt is then fed to a single PE (Figure 3.6 right). In this case, the I/O requirements of the processing entity, i.e. the cell, are reduced to two inputs and two outputs per cell update. Each pair consists of one state value and one constant value corresponding to the combined contribution of control template together with the bias (Figure 3.6 left). The main memory unit in the PE consists of 3 layers of equally sized circular shift-register arrays for the state input and 2 layers for each of constant

and template selection inputs. Inputs from left and right neighbouring PEs are directly connected to the corresponding ends of the shift-register arrays.

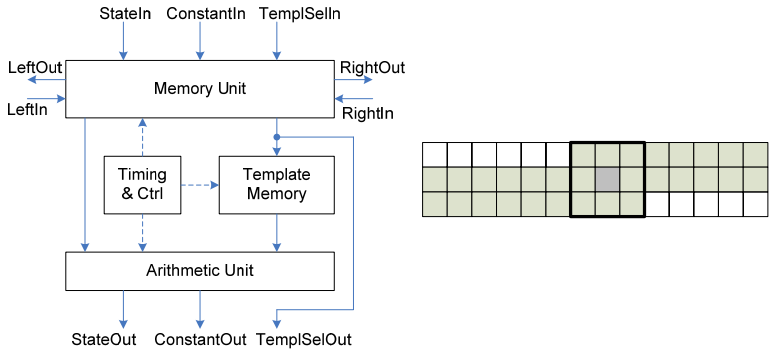


Figure 3.6 Left: a schematic view of the processing unit in CASTLE, where dashed lines represent control signals and continuous lines shows data path. Right: the belt of pixels stored on chip for 1-neighborhood where the black square indicates the current position of the convolution operation.

The high throughput of the system is due to the accommodation of 3 multipliers performing, in parallel, 3 multiplications that use pixels and corresponding template coefficients as operands. Multiplication results are shifted 1-bit in the LSB direction before they are forwarded to a tree of adders to accumulate the results with previous intermediate result. In order to improve the accuracy, rounding units are introduced between the shifters and the following adders. A limiter unit brings the final sum into the operational region. Figure 3.7 depicts the structural architecture of the arithmetic unit. It is obvious that the reduction of communication demands comes on the cost of larger arithmetic units with more functional blocks.

In line with the proposed approach in the CNN-UM, the functionality of the CASTLE architecture is ruled by means of a global control unit. One of the most important features of this unit is the selection of the employed precision. Data precision is variable and can be set to 1, 6 or 12 bits. The lower the accuracy the faster is the system.

An important issue is the amount of logic occupied by the register arrays constituting the internal memory units (Figure 3.6). In the first experimental chip that has been fabricated in 0.35 μm CMOS [36] technology, about 50% of the total area of a single PE is allocated to register arrays, while the arithmetic block occupies not more than 21% of the area (Figure 3.8). Furthermore, experiments show that a CASTLE emulator with 24 processors outperforms the DSP-based ACE engine (section 3.1) only when the rate of logic operations is high enough [36].

The CASTLE architecture suffers from a number of drawbacks. One has to do with the inability of emulating complex dynamic systems where operative parallelism is a key feature. The single layered architecture handles only one

operation at time. Other drawbacks include the limited template size, cell array size and accuracy. Hence, a new architecture called FALCON has been developed to provide higher flexibility and to allow for multilayer accommodation [42]. The implementation is based on the FSR model with discretized state equations. In contrast to all CNN-UM inspired implementation discussed so far, the design is hosted on a Xilinx Virtex series FPGA. This increases the ability for reconfiguration, brings down developing time cycle and decreases the overall cost.

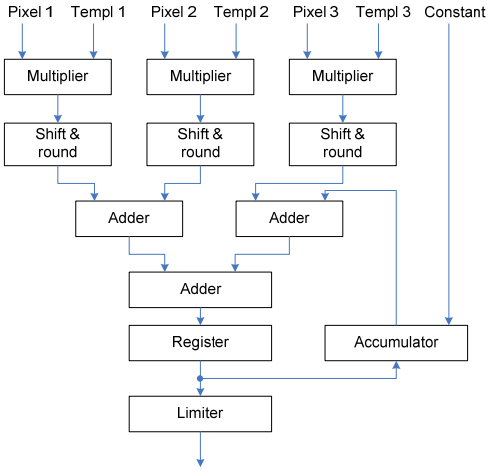


Figure 3.7 The arithmetic unit in CASTLE

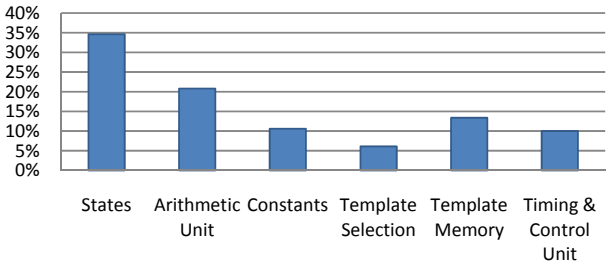


Figure 3.8 The amount of allocated logic for each of the blocks relative to the entire size of a single PE. Putting together the bars representing state values, constant values and template selection gives the total area of register arrays.

The arithmetic uses a fixed-point representation where both word width and displacement of the radix point for the state, constant and template values are configurable. Possible value widths are 2-64 bits. Other configurable parameters are: number of templates, neighbourhood size, size of the cell array and number of layers. Configurability is essential to allow accommodation of flexible precision when needed. But for the highest possible precision the cell array will

consist of not more than 4 processing cells! The configuration is unfortunately not dynamic but the entire design has to be re-synthesized and loaded on the FPGA every time a new configuration is required! Apparently, for algorithms with alternating operations of low and high precision the FPGA has to be reconfigured several times in order to provide accurate results. Moreover, the FALCON architecture comes with no possibility of algorithmic control on chip. All algorithmic steps, as well as local logical operations and programs, are executed on a host PC. This reveals that the system cannot stand alone, but is always dependent on the host PC! Obviously, all the benefits of performing complex tasks on the CNNs are lost. To remedy these problems, the architecture is extended with a global control unit GAPU [43] in line with the conceptual CNN-UM.

In addition to on-chip memories and some peripheral blocks, the GAPU is built using an embedded MircoBlaze processor core with 32-bit RISC architecture [38]. Most modern FPGAs provide at least one of these processor cores on chip. The extended FALCON architecture is implemented on a Xilinx Virtex-II 3000 FPGA. Apart from the embedded processor core, the GAPU occupies about 10% of the available logic, which can be compared to the area of a single CNN processor that requires about 2.8% of the logic. It is worth mentioning that the GAPU runs on lower clock frequency than the processing units (PEs), thus, setting a higher limit of the overall speed.

3.5 SUMMARY

In the light of previous advances made in design of analogue neural networks, researchers and developers have been encouraged to build analogue CNNs using VLSI implementation techniques. The first attempt yields in a too small chip with 20×20 cells only and lacks the feature of programmability. This is not enough to handle complex tasks, so developers have looked for use of available DSP technology to emulate the functionality of a CNN. The CNN-HAC shows a promising computing power but is mainly suffering from low accuracy. The successor ACE engine accommodates floating-point DSP to overcome the limitation. Both designs are dependent on a host PC where algorithmic programming is performed.

Meanwhile the conceptual CNN-UM is introduced to serve as a standard platform for real-time CNN realizations. The CNN-UM architecture contains a minimum number of component types. It provides stored programmable spatiotemporal array computing with real-time and supercomputer power. The stored programmability in form of cloning templates gives a minimal representation of a complex spatiotemporal dynamics. The machine supports both linear and nonlinear cloning templates and allows for implementation of multilayer CNNs. Many chips have been built to implement parts of the concept with varying success. Most interesting is the series of full-custom mixed-signal ACE-series. The characteristic drawback is again the limited analogue accuracy. In addition, the issue of sensitivity toward fabrication artefacts attracts special

attention and requires careful parameter tweaking to achieve the desired functionality. This time-consuming approach raises the overall cost per chip.

Naturally, the focus has moved toward fully digital implementations instead, as these provide an acceptable trade-off between computation power, accuracy, versatility and cost. Mainly, two architectures are available: full-custom CASTLE and FPGA-based FALCON. Both provide quasi real-time performance through a pipelining of the processed input values and are therefore considered as CNN-UM emulators. The former is, however, only single-layered whereas handling multi-tasks in parallel is not possible. Moreover, it has a limited, although flexible, accuracy of 12 bits. The FALCON architecture is extended to enable multilayer accommodation, and allow higher accuracy up to 64 bits. One of most important drawbacks is that FALCON cannot stand alone as is totally dependent on the host PC to perform the different steps of a certain algorithm. Moreover, the computational speed exceeds certainly the one provided by general-purpose computers, but is far less than the efficiency experienced in mixed-signal chips.

An important issue is the usage of the FSR model in both mixed-signal and fully-digital approaches. Here, the need of discrimination is removed as state and output values coincide. This saves a considerable amount of logic and makes the design simpler and smaller than general.

Furthermore, all architectures are heavily dependent on global control instruction in order to perform properly, which is affordable for small networks. For larger networks, long global control wires affect logic unitization negatively and slow down the system to such a level that the benefits of a CNN are lost. In [84] it is stated that a cellular architecture will be the way of the future, but that the performance advantages will soon dwindle “in the presence of global interconnections”. This seems to indicate the need for local connections only.

Finally, the question of communication with external storage units to bring in/move out values to/from the CNN array has never been answered satisfactorily. In the conceptual CNN-UM the problem is solved by photo transduction for input values while the proposed electromagnetic detection approach remains theoretical only. Practical employment is still conspicuous by its absence.

Chapter 4

Unrolling CNN on FPGA

The strict local connectivity gives CNNs first-hand advantages for tiled VLSI implementations with very high speed and complexity. This is tightly coupled to the simplicity of operation as it allows for implementation of a large number of simple units performing the same simple operation in parallel. A single chip is then able to accommodate multiple CNN layers, where a complex and time consuming task is divided into much simpler subtasks that are performed simultaneously; one subtask per CNN layer. Paradoxically, the strength of a CNN, i.e. simplicity of operation and local connectivity, constitutes the main hindrance toward efficient hardware implementation. The simultaneous activity of cells requires an instantaneous availability of input and output pairs (u and y-values) for each of the neighbouring cells. Consequently, 8 pairs of values have to be communicated for the minimal 1-neighborhood, one pair for each neighbouring cell. This is affordable in an analogue realization as it will result in 16 wires only. In a digital counterpart, a value is represented by an arbitrary number of bits, each requiring a wire on which a signal is carried. Even in the simple case of 8-bit values, the simultaneous interconnection will need 64 wires to be routed. Obviously, the interconnection requirements are severely increased for larger neighbourhood. Actually, establishing the connections within an arbitrary neighbourhood is so area and/or time demanding that little research on large neighbourhoods is made. Almost all known CNN templates are for a 1-neighbourhood, and all realizations are effectively restricted to that.

Parts of this chapter have been presented in [I] and [V].

As the CNN architecture is so wiring dominated, most of the available logic is used to render the inter-cell communication possible, yielding in a smaller network and therefore decreased throughput. Smaller networks do not contain the amount of cells that is needed to satisfactorily perform complex tasks. For instance, in the domain of image processing, where most CNN systems find their target applications, a frame has far more pixels than it can be handled by the largest available CNN chip (ACE16). One way to remedy this is by moving from one-to-one mapping between the actual operating unit and the theoretical CNN cell into a one-to-many correspondence. We have seen in Chapter 3 that the functional units, denoted PEs or virtual processors, in many CNN chips operate in-order on a large number of pixels. This, in itself, is a widely used methodology among system developers. For instance, André de Hon [44] has posed that the archetypical phase of hardware design is characterized by severe limitations on computing resources, making it necessary to use every hardware element as much as possible. This is called *temporal computing* as the operation is unravelled in time where the computational process is scheduled to execute-in-order on the few computational elements. On the other hand, *spatial computing*, where the process is unravelled in space, is preferred as it reduces spurious latency. In fact, the more efficient full-custom mixed-signal chips (section 3.3) employ the spatial approach and limit thereby the size of the processed topography (mainly 2-dimensional images) effectively to the amount of available PEs. In this thesis, the processing unit that performs the operation of a CNN cell is called *node*, and consequently is the state equation of a cell sometimes called the *nodal* equation. A node may, however, correspond to a number of cells that each corresponds to one element in the topographic map. Consequently, a node that performs the nodal equation in-order in a temporal architecture can be exported to a spatial architecture with almost no modifications.

Thus, a CNN architecture is able to efficiently handle a complex general task only if the number of nodes is large enough. How many nodes are enough is dependent on the application domain, the size of the topographic map and the amount of operational parallelism that is required. Consequently, smaller inter-nodal interface is crucial to achieve the goal. In this sense, an analogue approach is preferred, which explains why so far impressive advances have been made in analogue realizations only. The best attempt toward a digital realization *emulates* the functionality of a CNN rather than providing real-time performance [42]. These digital emulators are dependent on a host PC in order to perform the algorithmic steps in the desired order (section 3.4). This dependence is decisive for the dominance of the analogue realizations so far. In other words, the exploitation of a stand-alone fully-digital approach is highly desired, which this thesis aims to tackle.

On the other hand, analogue implementations keep up with high throughput and low latency by using array of photo sensors for data acquisition. This works fine for image processing applications where pixels are captured and processed directly, but imposes high latency if the topographic maps have to be pre-stored. When pre-storing is involved, network capacity can easily become limited by

the available bandwidth, not only inside the network itself, but even towards the external memory that holds the topographic map.

This chapter starts with a brief discussion on how CNNs are mapped on FPGAs. In section 4.2 two abstract execution models are introduced in order to highlight the importance of different communication and computation styles on the overall performance of any CNN architecture. Then, a brief description of previously, by the author, published architectures is given in sections 4.3 and 4.4. Finally, the chapter is closed with a discussion of the main pros and cons of the presented architectures.

4.1 MAPPING CNN ON FPGA

The desired stand-alone fully-digital approach can benefit from the achievement of the digital emulators in mapping a CNN on FPGA. We recall that both digital CNN-UM emulators (section 3.4) are realized on a Xilinx Virtex, i.e. 2nd generation macro-less FPGA. As the major arithmetic blocks are mapped on logic blocks provided on the FPGA, it is concluded in [42] that a further increase in packing density can be achieved in future generations. Indeed, in modern FPGAs, the over-mass of flip-flops and logic-mapped memory is supplemented by high-density, multifunctional macros such as Block Select RAMs and Multipliers. However, the most important feature of an FPGA is its modular construction, where the physical placement of the different components simplifies bundling logic and macros to easily form CNN nodes. Additionally, the eventual existence of Embedded Processor cores, denoted PowerPC in the terminology adopted by Xilinx, increases the power of computation and allows for a suitable mixture of hardware and software. Hence, flexibility and parallelism, provided by the specialized macros together with the modular construction, and the facility of spatial computing have made FPGAs already very popular as hardware accelerators and efficiently equipped to map modular structures, i.e. CNNs. .

The thesis stresses the exploitation of the built-in macros to spatially unroll the local feedback. The presented implementations rely on Xilinx Virtex-II 6000 and Xilinx Virtex-II Pro 20/30 [38]. These FPGAs are characterized by the richness of multiplier and RAM built-in macros that are closely placed pair-wise in a number of columns, whereas logic blocks are almost equally spread between the columns (Figure 4.1) . The functional blocks are available in a matrix-style floor plan which simplifies the mapping of the CNN mesh. In Virtex-II Pro 30, part of the matrix is obscured by the insertion of one or more PowerPC cores, yielding in a non-regular floor plan (Figure 4.1 right). This demands a careful placement of the nodes and may therefore complicate the procedure of floor planning the design.

A proposed mapping of the arithmetic blocks in a cell is illustrated in Figure 4.2. The different template coefficients need to be locally available as they are used in the nodal computational procedure. Thus, the coefficients are stored in a BRAM whose adjacent multiplier macro performs the multiplications, while

remaining computational operations such as addition as well as thresholding are suitably mapped on Configurable Logic Blocks (CLBs).

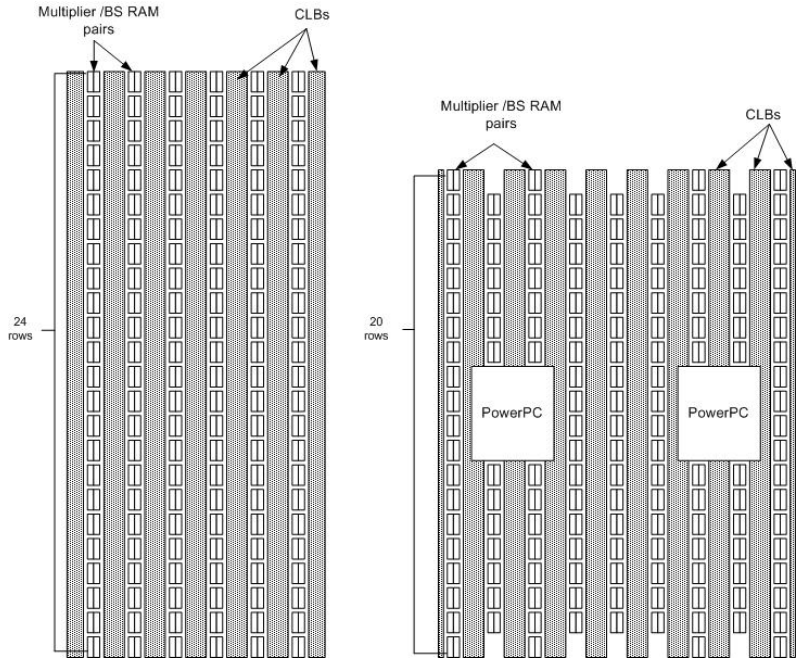


Figure 4.1 The configuration of a Virtex-II 6000 (left) and Virtex-II Pro P30 (right) from Xilinx. Grey columns represent bundling logic in form of CLBs, while the vertical boxes represent pairs of multiplier and BRAM macros. The placement of PowerPCs disturbs the matrix-style in the Pro P30 device.

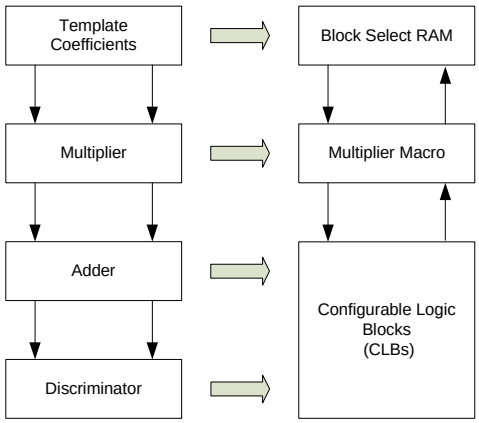


Figure 4.2 Mapping a CNN cell on FPGA primitives. Vertical arrows show possible data flow among different functional blocks/ FPGA primitives.

4.2 ABSTRACT EXECUTION MODELS

Looking back at the basic nodal equation of a single node (Eq. (2.19)), three contributions can be distinguished:

- *The feedback contribution*, $\sum a_d^c y^d(k)$, is involved in the iterations towards convergence.
- *The control contribution,* $\sum b_d^c u^d$, is valid for the current topographic map and does not depend on the iterations.
- *The offset contribution, i*, simply replaces the summed contributions to the right position for the final discrimination.

In this sense, the functionality of any node in the network is as follows. For each topographic map, the control contribution is first computed together with the bias, which results in a constant value that remains unchanged for the current map. This constant is preferably stored locally in the nodes. Then, the feedback contribution is calculated and added to the stored constant, resulting in a new nodal state that is discriminated to obtain the first iterative nodal output. For successor iterations, only the feedback contribution is computed, and the new state is discriminated and so on until convergence is reached or the iterative procedure is explicitly stopped. The calculation of control and feedback contributions is identical by means of number and nature of the performed computational operations. The series of multiply-and-add operations have, however, to be explicitly scheduled in order to guarantee correctness of functionality and achieve the desired performance. The need for explicit scheduling on nodal activities works out differently for different CNN to Network mappings. Two main categories can be distinguished:

- The *consumer node* is fully in accordance with the nodal equation. The discriminated output of a node is also the nodal output. It is broadcasted to all connected nodes, where it will be weighted with the coefficients of the applied template before the combined effect is determined through summation (Figure 4.3.a).
- The *producer node* discriminates the already weighted inputs and passes to each connected node a separate value that corresponds to the cell output but weighted according to the applied template (Figure 4.3.b).

Ideally all nodes are directly coupled and therefore bandwidth is maximal. In practice, the space is limited and the value transfer has to be sequenced over a more limited bandwidth. This problem kicks first in with the producer-type of network, where we have $2n$ connections for n neighbours. The network-on-chip approach is meant to solve such problems. However, as the Cellular Neural Network is a special case for such networks, being fully symmetric in the structure and identical in the nodal function, such a NoC comes in various disguises.

In the consumer architecture, scheduling is needed to more optimally use the limited communication bandwidth. Switches are inserted to handle the incoming values one-by-one. To identify the origin of each value, one can either

schedule this hard to local controllers that simply assume the origins from the local state of the scheduler (circuit switching, Figure 4.4.b), or provide the source address as part of the message (packet switching, Figure 4.4.a). The former technique is simple. It gives a guaranteed performance as the symmetry of the system allows for an analytical solution of the scheduling mechanism. The latter is more complicated, but allows also for best effort.

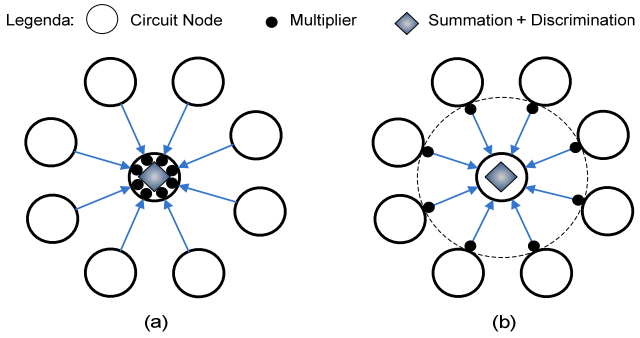


Figure 4.3 Consumer (a) and producer (b) cell to node mapping.

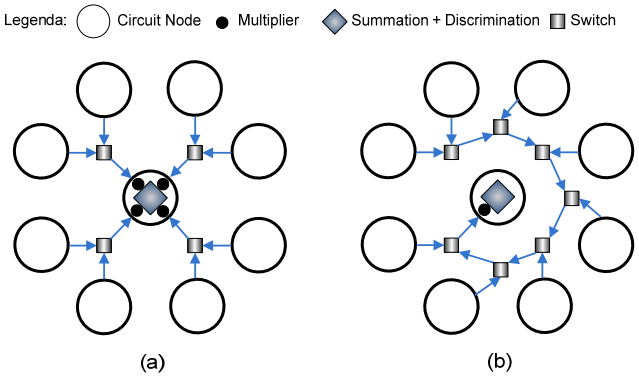


Figure 4.4 Value routing in the consumer node by multiplexing in space (a) and in time (b).

The counterpart of consumption is distribution. Every node produces values that have to broadcast to all the neighbours. Again where the communication has a limited bandwidth, we need to sequence the broadcast and this can be done in the same way as for the value consumption (Figure 4.5).

In a word-serial/bit-parallel approach, all nodes are broadcasting packaged values simultaneously over a set of 'rotating wheels' (Figure 4.5.b). For a 1-neighborhood, the cells execution time is $c + d$, where c is the amount of neighbouring cells and d is the core cell cycle. The packet that passed through the network is comprised by the values and for both the row and the column address 2 bits each. So, for an 8-bit value, a packet of 12 bits is needed. The network interface comprises of the packet switch, an input buffer and an output

register. The core node will iterate a parallel multiplication plus addition, followed by discrimination. Characteristic for this approach is the need for a parallel multiplier; furthermore it can only work on fixed-point integer.

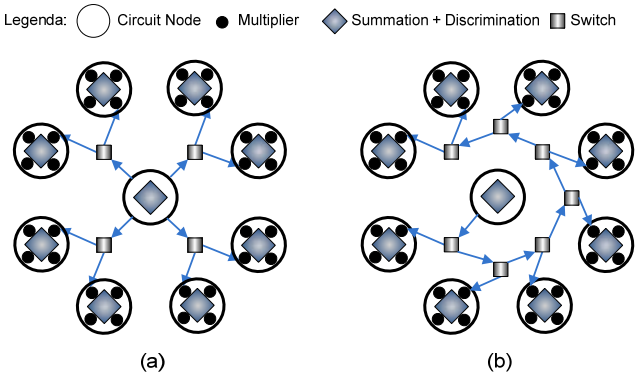


Figure 4.5 Another value routing in the consumer node by multiplexing in space (a) and in time (b).

The state of a cell is contained in the output register. For a multi-layer CNN implementation, the state is salvaged in the local memory. Therefore the overhead in performing the same operation on an image sequence or different operations on a CNN sequence is moderate.

In the case of producer architectures, the nodal output is already differentiated for the different target nodes. Each target node will combine such signals to a single contribution. This combining network is an adder tree that will reduce the n values to 1 in a pipeline fashion. Consequently, this tree can also be distributed, allowing for a spatial reduction in bandwidth. This can be seen from the simple re-write of the CNN equation as in Eq. (4.1). The content of the bracket is produced in neighbouring cells *d* before transmitted to cell *c*.

$$x^c(k) = \sum_{d \in N_r(c)} [a^c y(k)]_d + \sum_{d \in N_r(c)} [b^c u]_d + i^c \qquad (4.1)$$

The overall processing scheme as shown in Figure 4.6 is then similar to what has been discussed for the consumer architecture. The main difference is that the communicated values will be larger as they represent products and are therefore of double length. Where the consumer architecture is characterized by ‘transfer and calculate’, the producer architecture is more ‘calculate and transfer’. Furthermore they both rely on a strict sequencing of the communication, simultaneously loosing a lot of the principle advantage of having a cellular structure.

Also here, we have to look at the way values are broadcast. In contrast to the consumer architecture, we have as many output values as there are neighbours. This makes for an identical situation and no additional measures are needed, except for the fact that we will not be able to generate all the different products at the same and the sequencing issue pops up again.

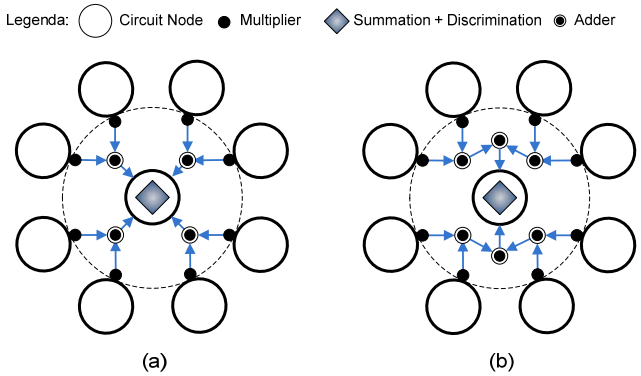


Figure 4.6 Different adder trees to obtain the state of the producer node

In word-parallel/bit-serial approach, all nodes are serially forwarding their values to all neighbours directly (Figure 4.6.b). Being circuit switched rather than packet-switched, no addresses are transmitted. For a 1-neighborhood, the cell execution time is given by $n + d + {}^{2}log(c)$, where n is the number of bits, d is the core cell cycle and c is the amount of neighbouring cells. There is no network interface. The local multiplications are done bit-wise and are followed by an adder tree that gradually increases in size. Characteristic for this approach is the reduction of the multiplier to a mere AND-gate; furthermore it can be easily adapted to scaled arithmetic and therefore allows a large dynamic range with limited precision.

It appears that the two architectural varieties differ mostly in the balance between wiring and logic, and are therefore dependent on the realization technology. They both show the ability to pass state and output data via the local memory, effectively mapping a levelled hierarchy of CNNs into a single implementation.

4.3 In The Footsteps of The Forerunners (Pipelining)

Analogue realizations have a larger capacity but suffer from limited accuracy, in contrast to digital realizations that have a smaller capacity but can in principle operate at a quasi-infinite accuracy. In fact, accuracy is limited due to amount of available resources. In the case of 8-bit precision for input values and template coefficients, the multiplicative adding of 19 contributions (1-neighbourhood) will lead to a 21-bit internal result; for a larger neighbourhood this will grow drastically. The amount of logic each node may occupy is not affordable!

The architectural characteristics emanate from a routing problem that occurs when information is sent to each of the 8 nodes in the direct neighbourhood. A local congestion can clearly not be avoided. This problem is attempted to be solved by not feeding all values simultaneously to the node. In the extreme case, values are fed in series creating a kind of systolic array as originally suggested by [42] (section 3.4). This is the *state-flow architecture* [50], where nodal

state/output values flows together with corresponding input data in the topographic map over array of cellular nodes coupled in series. In this way, FPGA resources such as multipliers, adders and other logic blocks are temporally exploited. Multiplicative additions are executed in-order on the limited computational elements. The architecture is developed with image processing in mind. As it will operate on images out of a stream, captured by a camera, it must be able to deal with many degrees of freedom in real-time: width and height of the image, the sequence of images in the stream, and the temporal dimension due to the iterative nature of the nodal equation. Apparently, the implementation medium, i.e. FPGA, offers only two dimensions and the others have to be masked away, which is easiest done by usage of local memory. The main architectural issue is then which two dimensions will rule the floor plan. Furthermore, the size of the frame is usually larger than the network, which implies partitioning of the image; passing the image in stripes over the system (Figure 4.7). This widely employed approach is called *windowing* (see e.g. [42]). Thus, the local operation is performed in 2-dimensional plane (width and length), of which one is masked away by stripe flow, and iterates in time. This is repeated over image slices and iterates over the surface to handle potential wave propagation. Finally, the operation is performed on a sequence of images.

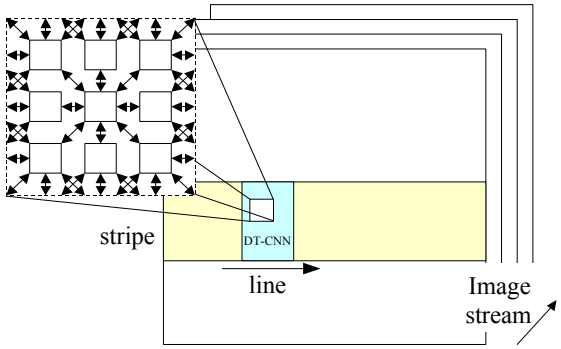


Figure 4.7 Dimensionality of DT-CNN image processing.

In a naive realization[4] of the state-flow architecture, employed for image processing applications only, data dependencies between scan-lines in an image are stretched over a pipeline of single multiply-accumulate units (Figure 4.8). Each performs only one operation on a single coefficient/input pair and then moves the result to the next unit. Pixels needed to perform the desired computation of the output for one node, are fetched from three series of registers connected to the pipeline. In this sense, each neighbouring pixel is evaluated separately in a pipelined fashion, doing in series as many multiply-accumulates as there are cells in the neighbourhood.

[4] A student project implemented and presented in the course VLSI Architecture at Dept. Information Technology, Lund Univeristy, 2003. For further details see [49].

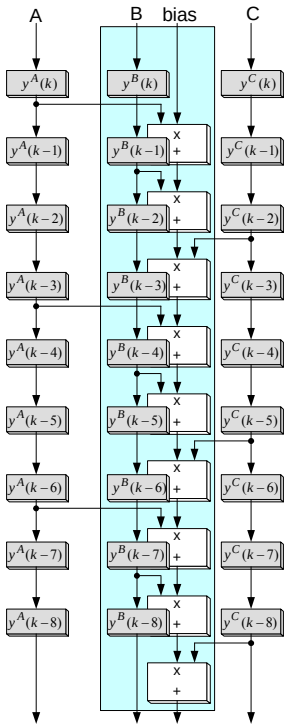


Figure 4.8 Data dependencies for a pipeline in a naive temporal state-flow architecture. Only the pipeline corresponding for the middle node is shown. White boxes represent functional blocks; consisting of a multiplier and an adder, while grey boxes represent registers. The middle node corresponds to a pixel sequence B. For sequences A and C, functional blocks are dropped for clarity. Identical architecture is used to calculate the contribution of pixel inputs.

Based on the observation that both multiplication sequences are independent, the desired network behaviour is implemented as two 9-stage pipelines per one DT-CNN node. The output is obtained by thresholding the sum. Thus, one pipeline, consisting of 18 multipliers and additional logic, is needed for every column of the image. Due to the organisation of multiplier macros in the target FPGA, i.e. Virtex-II 6000, in 6 columns and 24 rows (Figure 4.1), only six nodes can be mapped. Additionally, a node on each side of the image stripe is needed to eliminate boundary effects, which reduces the actual number of image columns processed in parallel to four only.

Apparently, the throughput of this, so-called temporal, approach is way lower than it can be accepted in spite of, or especially due to, using large amount or resources on the FPGA. To overcome this drawback, spatial and temporal elements are mixed by interweaving three pipelines corresponding to a row of three pixels (Figure 4.9). This reduces the latency and makes better

utilization of the available resources. The nodes are grouped in columns where each column holds a scan-line in the image stripe. The columns will then form iterations performed on the image. In this way, one dimension (width or length) of the image frame together with the number of iterations are implemented as columns of nodes while the other dimension of the frame is handled by slicing as illustrated in Figure 4.7. One of the resulting realizations is a design called ILVA [49].

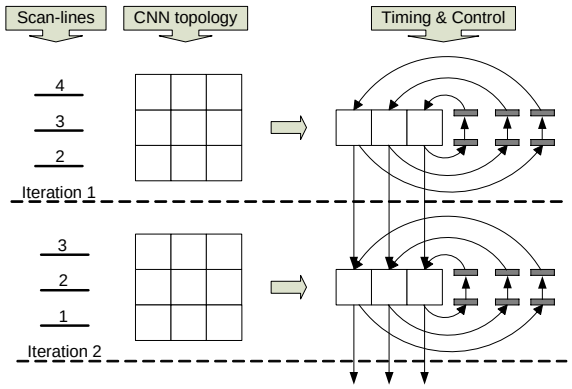


Figure 4.9 Mixed spatial-temporal state-flow architecture operating directly on the pixel pipeline.

The principle of operation is as follows. Each image stripe is entered to the CNN on scan-line-by-scan-line basis, as depicted in Figure 4.10.a. Here the sequence of scan-lines is numbered lexicographically using characters each representing a scan-line. Then, if a column in the CNN structure contains scan-line B, the column to the left will contain the next scan-line C and the column to the right will contain the previous scan-line A. Bringing the cell numbering (Figure 4.10.b) and the pixel numbering (Figure 4.10.a) together, we come to represent a pixel by triplets, where pixel A and C are orthogonal neighbours to pixel B but so are the upper B_u and the lower B_l (Figure 4.10.c). Consequently, at any given moment the network contains a part of the image as seen by viewing the picture.

Functionality of the architecture, as illustrated in Figure 4.11 can be algorithmatically described as:

for (a pixel line of limited length) do{
compute the constant contribution $B \times u + i$
pass $B \times u + i$ *and y to the next stage*
perform an iteration
while (there are more stages) do{
pass $B \times u + i$ *and the iteration result to the next stage*
perform an iteration
}
send the local outputs to the image store
}

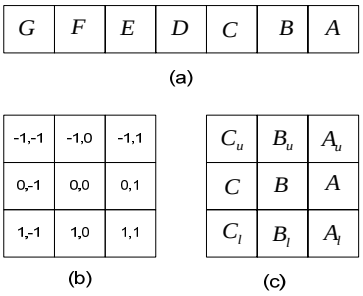


Figure 4.10 Numbering of CNN cells (b), lexicographically ordered pixels (a) and in combination (c).

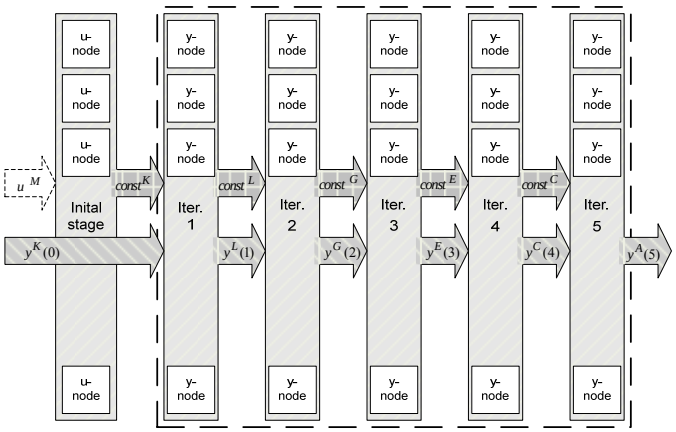


Figure 4.11 Snapshot of data flow between consecutive columns in ILVA. The design consists of six columns corresponding to one initial stage and five subsequent iterations. The notation of inputs u, outputs y and intermediate constants const follows the lexicographical ordering presented in Figure 4.10. The data flows from a node in a certain stage to a node, allocated in the same row, in the successor iteration stage. Arrows between two columns illustrate data flow originating from all nodes in a column.

The underlying idea is that a 2-dimensional computation of the local cell is flattened into a series of 1-dimensional computations by dropping intermediate results on the computational path. In this way, the requirement of each node to have data from eight neighbours for finding the output is met. In other words, we let every node in the network contain image data from three pixels, i.e. pixel values for the cell itself and for its left and right neighbours are stored in each node. A direct connection with the two nodes above and below completes the communication between a node and its neighbourhood. In short, one node contains three pixels and calculates the new value for *one* pixel and *one* iteration.

The prescheduled broadcasting in ILVA keeps the communication interface at minimum, which allows for a large number of nodes on chip. The performance is high as the system directly follows the line accessing speed, but the design suffers from a number of weaknesses. It supports 1-neighborhood only, where extension to larger neighbourhood requires, due to the hardwired communication interface, a total overhaul. The iterations are flattened on the pipeline, one iteration per pipeline stage, making the number of possible iterations not only restricted due to the availability of logic, but also fixed. Operations that require a single iteration only, have still to go through all pipeline stages. Output data has to be fed back to the pipelined system in order to perform additional iterations, making it far from trivial to handle larger iterations without accessing the external image memory. This requires additional logic for loading and uploading pixel data and therefore adds overhead for timing control and thereby severely slows down the system.

Though the architecture is very efficient for a single image operation, the handling of image streams (Figure 4.7) is less trivial. This is foremost because the pixel line flow does not support localized storage related to the original image. In effect, only the first array of nodes operates directly on pixel information. Consequently it becomes hard to store past information about more than a couple of pixel lines. Therefore this architecture seems unsuited for Wave Computing, i.e. manipulating streams of images.

The dilemma is resolved through packet switching techniques based on the concept of Network on Chip [9][87]. By splitting the node into a processor and a router, local timing becomes uncoupled. Actually, the path is still pre-defined as circuit switching with packet-switching techniques are mixed by replacing the hardwired communication with a packet-based communication pattern. By actively sending information to addressed nodes it becomes possible to create temporary storage out of line with the strict matrix topology. In this way, more iterations and more history may be accommodated. In the following, we explore two alternative architectures. The former serves as an extension to ILVA, where some of design limitations are removed. The latter explores a totally different approach of dealing with the problem.

4.4 NoC-based Implementations

Sleipner [49] is introduced as an improvement of ILVA architecture, where limitations experienced with the hardwired communication pattern are to be overcome. A generalization of the network system in order to support templates of an arbitrary size, i.e. neighbourhood size larger than 1, is the main issue. Similar to the flow pattern shown in Figure 4.11, sending a packet one column to the right means passing the value one iteration ahead. Pixel data is “kept” in each column for 3 iteration cycles before it gets modified and passed further. Consequently, data will never pass across more than one column at a time, regardless the neighbourhood in use. Neighbourhood size, however, dictates the number of rows the packet has to cross. Figure 4.12 illustrates transferring a packet in a 2-neighbourhood.

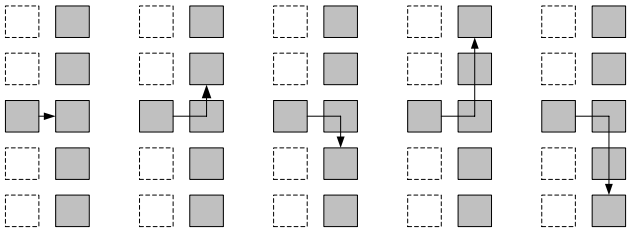


Figure 4.12 Packet transfer scheme in a 2-neighbourhood. A packet, originating in the middle cell in the left iteration column, is transmitted to all cells within the neighbourhood in the right iteration column.

In spite of the clear improvements Sleipner brings, it still inherits the limitation of 5-iteration steps from ILVA. This originates from the pipelined nature of both designs. An alternative architecture has simply to have a large network of simple nodes, each performing the entire iteration according to the CNN nodal equation. The equation is not unrolled in time but in space, and the nodes retain the result of the equation evaluation so that next iterations do not involve access to the external data memory. In this *state-scan architecture* [50], the neighbourhood is actively scanned for the input values. The coefficient/input pairs are sequentially fed through a single multiply-accumulate unit in a predetermined schedule. Such a schedule is totally decoupled from the inter-nodal communication scheme where nodes transfer their values within the neighbourhood in parallel. The aspect of being completely local is crucial to achieve high performance. Otherwise, a global communication scheme will lead to many bus conflicts and will therefore require additional bus arbitration. The local broadcasting scheme can be carried out in two different ways: word-serial or word-parallel. In the former, each node communicates its input/output value to a single neighbour, whereupon the message starts on a circular trip (Figure 4.13.a). After 8 time-steps, all nodes within 1-nieghbourhood have received a copy. Doing so for all nodes in the neighbourhood simultaneously, all u/y values become locally available in each node (This model is covered in Chapter 5). The word-parallel scheme (Figure 4.13.b) follows a more symmetric distribution, where the node passes its value first to the orthogonal neighbours at west, north, east and south. Next, the orthogonal neighbours duplicate this value in one sideway direction perpendicular on the former direction. Doing so for all nodes, at the same time, duplex orthogonal connections among all nodes have to be available (instantaneously). Instead, the broadcast is parallelized by simultaneously activating nodes at a knight-jump distance (think Chess!) as depicted in Figure 4.13.c, which brings the number of passes to 10 for the full 1-neighbourhood.

Both communication schemes make use of a simple router that consists of four switches on the orthogonal directions in line with well-known wormhole set-up. The router basically receives a data-packet originating from one of the neighbouring nodes, regularly refreshes a local buffer with new information, and eventually forwards the packet further while the local processor keeps on

computing. Figure 4.14 shows a schematic view of the architecture of a single node. The set-up is rather classical where the template memory takes the role of the program store.

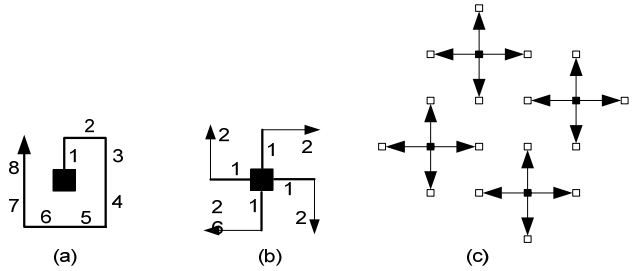


Figure 4.13 Switched broadcasting schemes: word-serial (a) and word-parallel (b). Nodes are activated at knight-jump distance in word-parallel broadcasting (c).

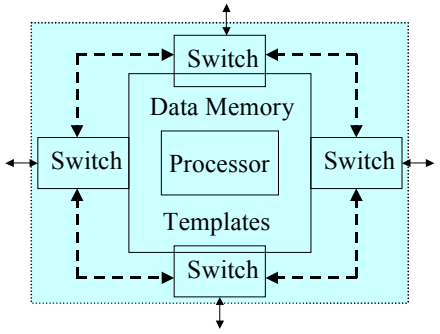


Figure 4.14 A node communicates with the neighbourhood through four switches.

The state-scan approach with word-parallel scheme has been embodied in a design called Caballero [49]. The inner architecture of the node is similar to the one in ILVA with common sub-units such as data memory, template memory, multiplier-accumulator (MAC) and local controller. A Caballero node is further equipped with a FIFO element to bring in global data, i.e. input and initial output values. The principle of operation depicted in Figure 4.15 is as follows. Pixel lines come into the FIFO till it is fully filled. Then these values are copied into the CNN nodes that subsequently start computing and communicating. Meanwhile new pixel lines come in over the FIFO. When the FIFO is filled again and the CNN nodes have completed all local iterations, the results are exchanged with the new inputs. This leaves the CNN nodes with fresh information to work on and the FIFO can take new pixel lines while moving the results out.

Upon start, nodes are provided with pixel values over the FIFO-structure. The first set of active nodes then start delivering u-values within the neighbourhood. Activation is moved among cells until all cells contain complete information about neighbours' u-values. Next time a node gets active, it will

eventually have the calculated value ready to be transmitted. Once a node completes the transmission cycle, the successor node in the activation group (Figure 4.16) is turned on. Hence, a mechanism that assures a smooth exchange of activation is required.

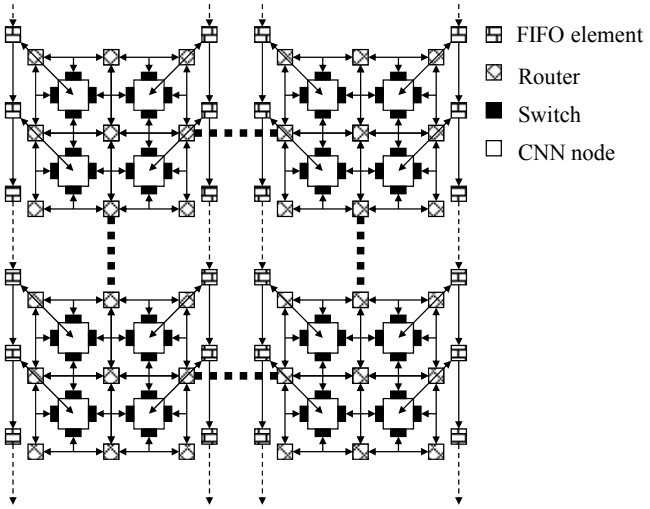


Figure 4.15 The state-scan architecture uses a network of CNN nodes with a Network-on-Chip, while the pixels are transported over a distributed FIFO.

A global control algorithm that groups the CNN into active and non-active nodes seems easy to implement. But the impact of a global control unit on wiring and timing overhead is not acceptable as the overall performance is negatively affected. A better solution is making the desired controlling local within each activation group. The activation pattern consists of 5 steps for the case of 1-neighbourhood. Once the node completes transmitting a packet, it notifies the successor node to get active by asserting an activation signal. Note that the activation pattern is incomplete for boundary nodes, i.e. edge and corner nodes, as these nodes lack certain neighbours. For further details about Caballero the reader is asked to look at [49].

4.5 DISCUSSION

Mixed (ILVA) and pure temporal architectures differ in a number of ways. First, the modular structure of the spatial design offers a better usage of the distributed memory than the sliced structure of the temporal design. Second, the temporal design with pipelines in the succession of multiplying-adder operations for a single DT-CNN node needs a frequent access to the external memory, while the spatial design allows unrolling the design for the required computational iterations. This eases the demands on external memory access and therefore leads to an intrinsically better performance. Furthermore, a single

node in the temporal design occupies at least 18 multiplier-adder pairs, which leads to imperfect floor plan and thus decreases the degree of resource utilization, while the modular nodes of the spatial design allows 24 nodes per column. The maximum capacity of a spatial architecture in terms of parallelized pixels is about 5-6 times higher than that of a temporal architecture (Table 4.1). The low clock rate of 17 MHz, mainly caused by the complexity of the pixel address generation, is a major drawback for the temporal architecture. There is of course room for more optimization, but the gap to the implementation in the spatial architecture, i.e. at least a factor of 40, is too large to bridge. This fact, together with the observations listed above, leads to the conclusion that the mixed architecture brings clear benefits.

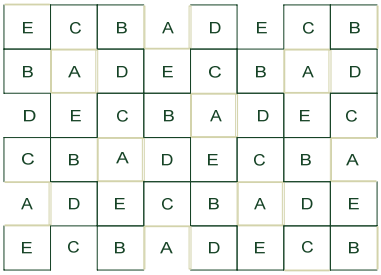


Figure 4.16 Caballero nodes are divided into active and non-active nodes in accordance with the knight-jump distance. Each activation group consists of 5 nodes that are activated in sequence A-B-C-D-E-A.

The most important disadvantage of the pipelined design, ILVA, is the restriction of the number of iterations into 5, which leads to decreased convergence and thus contributes with a tangible loss in performance. The packet-based distribution scheme overcomes this limitation and allows for user-defined level of convergence, which makes it preferable.

Table 4.1 Comparison of the two state-flow architectures. Logic counts are obtained after synthesis with Synplify, while throughput is obtained by simulating the designs using ModelSim. In ILVA, different depths (i.e. number of rows) yield different throughputs.

	ILVA	Temporal
Slice utilization	37%	13%
LUT utilization	28%	12%
Multipliers	132	78
BRAMs	132	0
Max. frequency (MHz)	100	17.3
Cycles per iteration	10	17
Throughput Mpixel/sec	205-220	4.1

Word-serialized broadcasting scheme (Figure 4.13.a) can be easily expanded to a higher neighbourhood than 1 without significant modifications in the strategy. Active nodes are simply chosen with equidistance depending on the

neighbourhood in use. This ability is not valid for the switched broadcasting scheme (Figure 4.13.b). On the other hand, the switched approach provides a symmetric distribution of packets in the neighbourhood (Figure 4.17).

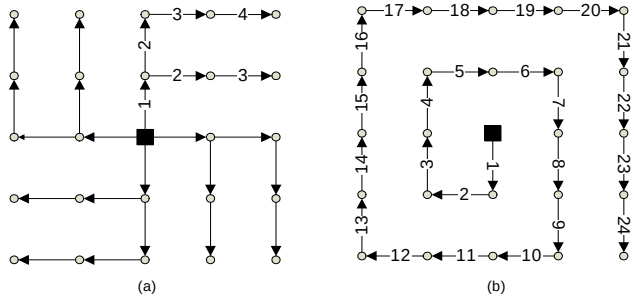


Figure 4.17 Distribution time for 2-neighbourhood in KJL (a) and SSL (b)

Due to the use of packet-based switching, communication and computation needs are decoupled, which removes the hard timing constraints found in ILVA. This allows for fetching new input data independently of the operational status of the nodes. While the nodes compute and communicate, new template coefficients can be sent using the FIFO structure. This releases a considerable amount of storage logic that can be used for other purposes, e.g. keeping a number of nodal values. Each node corresponds then to different pixels in different image frames in a stream. By storing a pixel value from successive frames, Wave Computing is within reach.

Chapter 5

Stretching The Communication

In a switched broadcasting, as introduced in the state-scan architecture, all nodes send their own values to the orthogonal neighbours that copy the data and forward it in a perpendicular direction to the received one. Theoretically, all nodes will have access to the values of the entire neighbourhood after two steps only but the group-based scheduling adds a latency as large as a 10 clock cycles. Hence, the actual communication cycle, during which a node is idle, is coupled to the number of cells in each subgroup. In other words, the short communication pattern of two steps does not boost the performance. On the contrary, it affects the final throughput negatively due to larger routing units and thereby smaller network. By stretching the communication cycle of a 1-neighborhood to 10 clock cycles, the routing demands are reduced, which in turns leads to simplified control. This is the semi-parallel broadcasting scheme (Figure 5.1.a), where the possible directions are always: North, East, South and West. Received packets are labelled in accordance to the position of the source node with respect to the current (destination) node. Obviously, the computation needs can be plaited together with the communication cycle. Table 5.1 shows how this can be done with sending and forwarding packets.

In this chapter a serial scheme (Figure 5.1.b) is proposed. The values are sent out in one direction only, but are forwarded to all nodes within the neighbourhood serially. Table 5.1 and Table 5.2 show that stretching the

Most of the material in this chapter has been published in [VI].

broadcasting of packets yields the same sequence of computation calculation, regardless of the broadcasting scheme. The received packets are consumed directly and overridden by subsequent packets. Consequently, the need of a local memory to hold the values of all neighbouring nodes is removed. A single register is used to hold the current packet before it is multiplied by corresponding template coefficient that resides in a local memory. Traditionally, the same memory is used to hold a look-up table representing the discrimination function.

Table 5.1 Semi-parallel broadcasting scheme

Clock cycle	Send	Receive	Forward	Hold	Calculate
1				y_{own}	$a_{own} \cdot y_{own}$
2	N	S	E	y_s	$a_s \cdot y_s$
3		W		----	$a_{sw} \cdot y_{sw}$
4	E	W	S	y_w	$a_w \cdot y_w$
5		N		----	$a_{nw} \cdot y_{nw}$
6	S	N	W	y_n	$a_n \cdot y_n$
7		E		----	$a_{ne} \cdot y_{ne}$
8	W	E	N	y_e	$a_e \cdot y_e$
9		S		----	$a_{se} \cdot y_{se}$
10					$f(\cdot)$

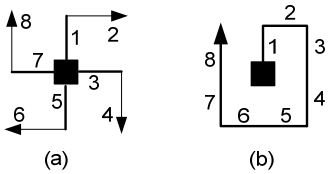


Figure 5.1 Switched broadcasting schemes: Semi-parallel (a) and Serial (b).

The chapter is organised as follows. First, a simple network interface that can remove the need for global synchronization is introduced in section 5.1. As the design is realized on a Virtex-II FPGA from Xilinx, the internal design of the node aims on a best utilization of the available functional units. Section 5.2 describes how this is carried out. Complications that rise in connection to handling boundary conditions are demonstrated in section 5.3. Finally, the chapter is closed with a discussion on how the performance can be boosted further.

5.1 KEEPING THE CONTROL LOCAL

Looking back at Eq. (2.34), we see that the part involving u^d-values together with the bias remains unchanged during the iterative process of computing the new nodal state and thereby the new output. Thus the broadcast will first handle

the inputs u^d and the bias and the resulting constant is locally stored. On every next iteration, the result of broadcasting the cell outputs will be added to the stored constant to give the new cell output. There is no need anymore for a global control and the network interface is very simple.

Table 5.2 Serial broadcasting scheme

Clock cycle	Send	Receive	Forward	Hold	Calculate
1				y_{own}	$a_{own} \cdot y_{own}$
2	N	S		y_s	$a_s \cdot y_s$
3		W	E	y_{sw}	$a_{sw} \cdot y_{sw}$
4		N	S	y_w	$a_w \cdot y_w$
5		N	S	y_{nw}	$a_{nw} \cdot y_{nw}$
6		E	W	y_n	$a_n \cdot y_n$
7		E	W	y_{ne}	$a_{ne} \cdot y_{ne}$
8		S	N	y_e	$a_e \cdot y_e$
9		S	N	y_{se}	$a_{se} \cdot y_{se}$
10					$f(\cdot)$

In order to simplify the control demands, the addressing of template coefficients is obtained through a base-address register that holds the higher address part, and indexing of the lower address part that is carried out by the nodal controller itself. As the BRAM has the configuration of a 2K entries memory, the base-address register does not need to be wider than 6 bits. The address space is arranged as shown in Figure 5.2. For a 1-neighbourhood 19 coefficients need to be stored for each template: 9 control coefficients, 9 feedback coefficients and a bias. As the control coefficients and the bias are used in the first iteration to compute the constant, they are stored sequentially and can be addressed by 4 bits. A u/y-flag, set by the nodal controller, allows the addressing of the corresponding feedback coefficients. The base address picks out the correct template.

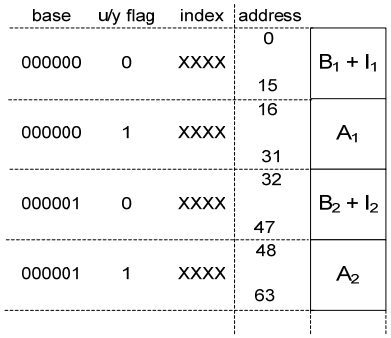


Figure 5.2 Address space of the nodal template memory

Also a number of templates are pre-stored in the local memory. But other templates can be sent by the user to every node in the network through the FIFO-elements. These FIFO-elements have served originally to bring the external inputs u into the nodes, but their functionality can easily be extended to cover the handling of template transmission. At first glance, this additional mechanism seems to add on the complexity of the nodal controller, but a proper usage of information stored in the header of the received FIFO-packets keeps the complexity at a minimum.

Two main types of FIFO packet do exist. These can be divided further into subtypes:

- Value packet
  - U packet
  - Y packet
- Template packet
  - Coefficient packet: used to store template coefficients properly in the BRAM.
  - Base-address packet: indicating the starting address from which the coefficients of the currently used template are fetched.

A FIFO packet contains two main fields: DATA and CTRL. The former is always 8 bits wide. It holds u/y value in the Value packet or coefficients and base address in the Template packet. The control field is further divided into 4 fields of different widths. These are, starting from the most significant bit: VALID, TYPE, SUBTYPE and INDEX fields (Figure 5.3). The first three fields are 1 bit wide each, while the width of the INDEX field varies depending on subtype and size of the network. FIFO elements are arranged in a grid, one element per CNN node. These elements are numbered row-wise, i.e. all FIFO elements that are aligned along row r are labelled with the row index r. In other words, in a Value packet the INDEX field will contain CNN-row number of the corresponding node to which the packet is intent. In a Coefficient packet, this field holds the sequential index of template coefficients, but it does not have any significance in a Base-address packet. The base address occupies normally the DATA field. Apparently, the size of the INDEX field varies with the number of rows in the network and the neighbourhood size. Packets are distinguished by using TYPE and SUBTYPE fields. The VALID field indicates the validity of the FIFO-packet.

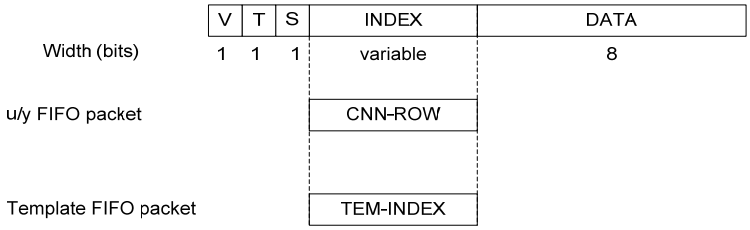


Figure 5.3 A FIFO packet is divided into 5 fields of different widths. V,T and S stand for VALID, TYPE and SUBTYPE respectively.

The importance of packet division into fields is obvious when the FIFO packet is used to control the functionality of a node. In the following, this is demonstrated by a simple sequence of actions performed on a certain topographic map.

(1) *Use pre-stored template*: The first action is to compute a new output by applying a pre-stored template. As templates are pre-stored in a BRAM locally in each node, a base address serves as a pointer to the template to be used. This starting address is set explicitly through a base address packet. Each node computes new y values iteratively until a new u packet is received. The content of the FIFO-element is then swapped with the current y-value that is flushed out to the user.

(2) *Receive new template*: For 1-neighbourhood 19 coefficients have to be sent which yields in a TEM-INDEX field of at least 4 bits (Figure 5.3). This is carried out using a set of template coefficient packets as demonstrated below:

a. The procedure starts with sending a Base-address template in order to point out the position in the template memory where the new template is to be stored.
   TYPE = 1
   SUBTYPE =1
   INDEX = XXXX
   DATA= base address

b. The nodal controller waits for the current iteration to be completed before the computation is halted. Most important is that the u/y register is disabled to prevent the obtained y-value from being overridden. One node of the last row sends then a ready signal out to the user.

c. When all nodes have completed the current iteration and updated the base address, a Coefficient-packet containing the value of the bias is sent. The index field is used to address the BRAM where the coefficients are stored. This packet is forwarded to all the nodes through the FIFO-structure.
   TYPE = 1
   SUBTYPE = 0
   INDEX = bias index
   DATA = bias value

d. The template coefficients are sent one by one. These will gradually reach all the nodes in the system.

e. The procedure ends when last Coefficient packet is received. The nodal controller waits for the new u-value to be received before new calculation round is initiated.
   TYPE = 0
   SUB-TYPE = 0
   INDEX = CNN-ROW

5.2 THE NODAL DESIGN

In principle, control demands are reduced down to a mux-enable signal and addressing of the template memory. A single register is used to hold one value only according to Table 5.2. The content of the register is overwritten as a new value is received or locally produced. The schematic design of the node is shown in Figure 5.4. Here the local memory is merged with the discriminator, as it also holds a table of pre-computed values to map the state onto a certain output.

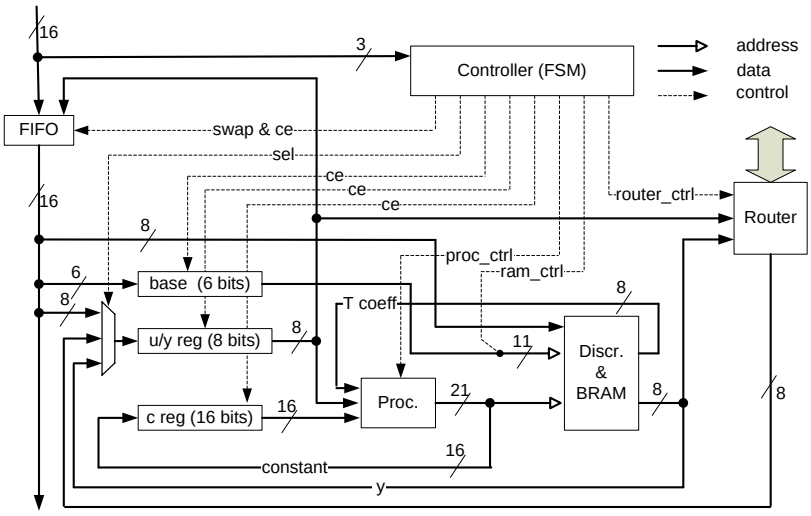


Figure 5.4 A schematic view of the serial CNN node.

The nodal controller has the full responsibility for all computation and communication activities. It is built as a simple state machine consisting of 3 main states as shown in Figure 5.5. Upon start the controller is IDLE and awaits a FIFO-packet. From here it can take two different paths: it may ITERATE on the received input value or LOAD new template coefficients. In the latter state, the controller enables the writing to certain addresses in the local memory. The addressing of the memory is combined using the content of the base address register, the u/y flag and template indexing field in the received FIFO-packet. Both computation and communication are performed in the ITERATE state. Here, the same sequence of sub-states perform constant- and new y-value calculation flattened with distribution of data packets within the neighbourhood.

Apart from the multiplexer in front of the u/y register, there is a need to use 3 other 2-to-1 multiplexers internally in some components: 2 in the processor and 1 in the discriminator. In the former (Figure 5.6), one multiplexer provides the accumulator with the proper data, constant value on one side and $u \cdot b$, $y \cdot a$ or bias values on the other side. The second multiplexer is used when the bias is

loaded into the accumulator. Here the bias is multiplied by 1 and send to the accumulator.

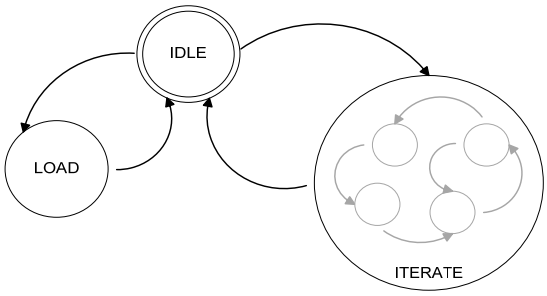


Figure 5.5 The nodal controller is built as a simple FSM. The ITERATE state tself consists i of a number of states.

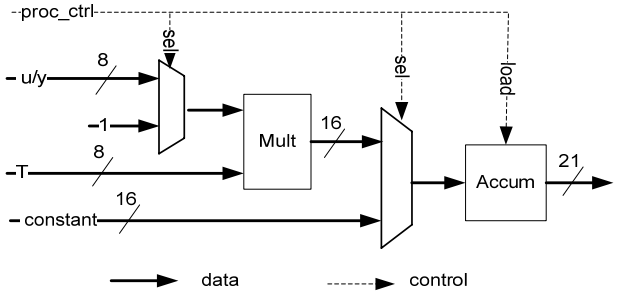


Figure 5.6 A schematic view of the nodal processor.

In the Discriminator (Figure 5.7), template coefficients are addressed differently. In the ITERATE state the controller indexes the lower part of the address, while this is obtained directly from the FIFO-packet during the LOADing of the new template.

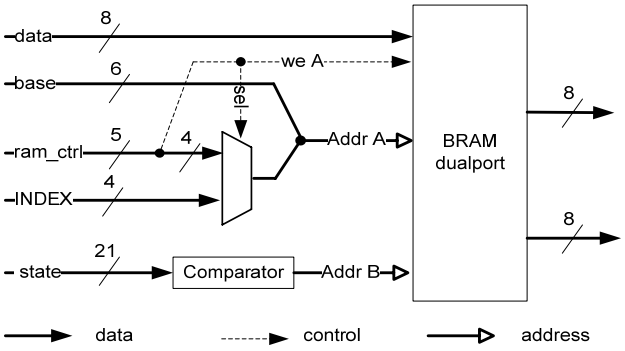


Figure 5.7 A schematic view of the nodal discriminator

5.3 BOUNDARY NODES

The functional correctness of any CNN system depends on the handling of the boundary nodes as these lack complete neighbourhood. Traditionally, the effect of boundary conditions is modelled by adding virtual nodes on the edge of the network. The problem here is further complicated by the asymmetry of the pre-scheduled communication pattern: boundary nodes experience different needs depending on their position in the network. Figure 5.8 illustrates the disturbed communication cycle for edge boundary nodes. The situation is even worse for the corner nodes (Figure 5.9). Actually, not only boundary nodes are affected by the incompleteness of broadcasting but even close-to-boundary nodes as well (Figure 5.10 left).

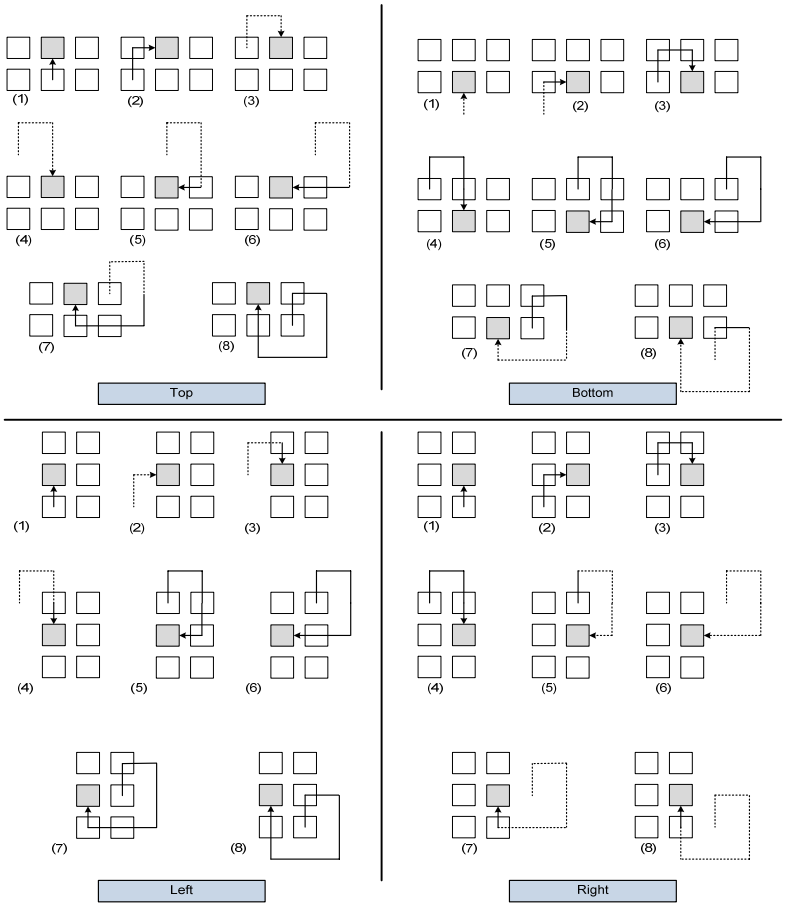


Figure 5.8 Boundary nodes have an incomplete communication cycle (from step 1 to 8). Squares represent nodes while the dotted lines show which part of the packet path is missing. The receiveing node is shaded.

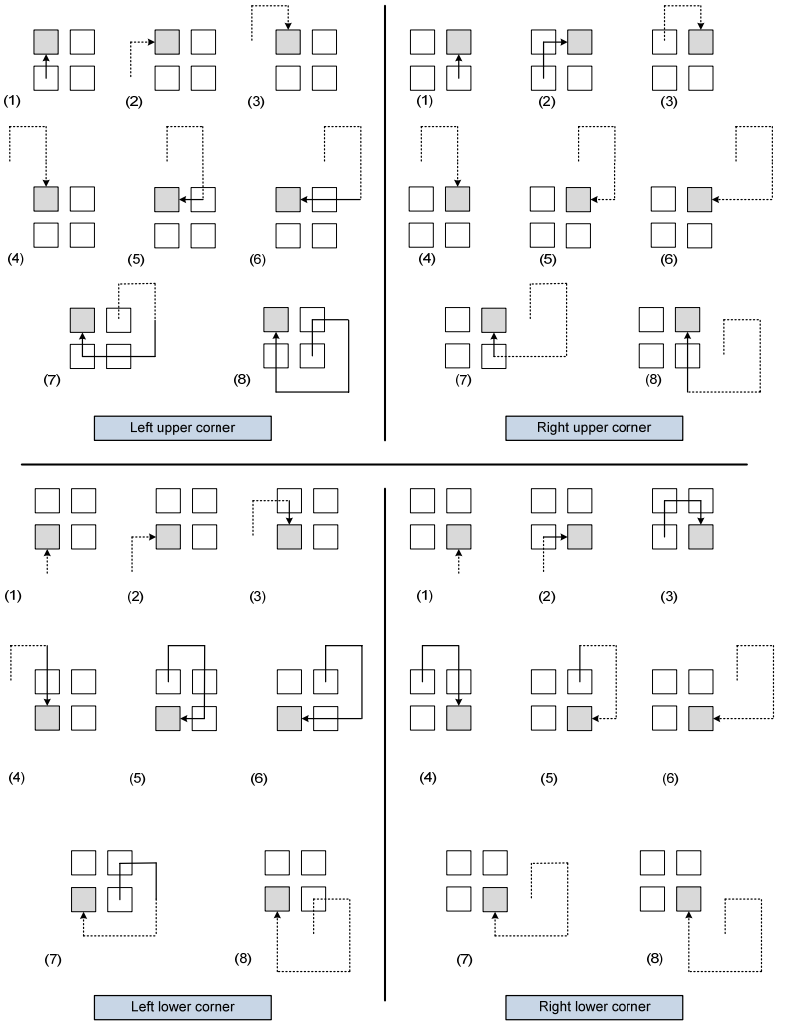


Figure 5.9 Boundary nodes located at the corners suffer more of the incomplete communication pattern.

Employing the traditional approach of adding virtual nodes is not as simple as it may seem. Besides being unable to solve the problem completely, it adds on the network size. In any prescheduled communication scheme, virtual nodes should follow the sequence of sending (and eventually forwarding) of values that is accommodated by all regular nodes in the network. This works fine for close-to-boundary nodes (Figure 5.10 right), but the communication path is still

incomplete for boundary nodes. It is clear from Figure 5.11 that top boundary nodes will not receive any data in steps (4), (5) and (6), even when virtual nodes are added. In other words, the partially asymmetric transfer cycle necessitates the existence of two (!) layers of virtual nodes to achieve completion. This holds for all boundary nodes except non-corner left edge nodes (Figure 5.8 and Figure 5.9). Hence, for an $M \times N$ CNN, the number of virtual nodes is equal to $4N + 3M + 12$. Each virtual node needs a router to send and forward packets, a local register and a simplified controller, which will affects area utilization negatively! We aim here for a total removal of the need for virtual nodes. This is possible by slightly changing the communication pattern of boundary nodes. Let's consider top and bottom boundary nodes. Then, the actions listed in Table 5.3 have to be performed in addition to the regular functionality of the node, mainly when a zero-flux boundary condition is used. For fixed boundary condition most of the sending/forwarding is redundant as all boundary nodes will need to store a single fixed value only that can be used instead of the received value.

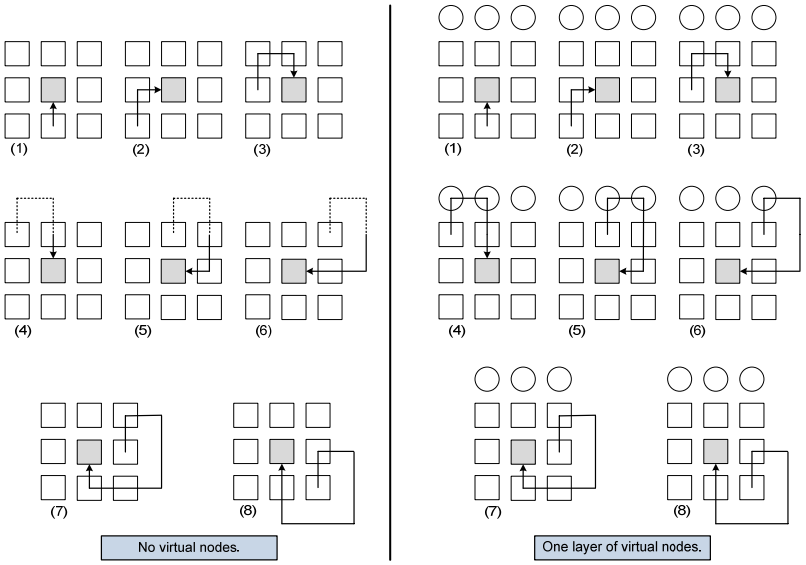


Figure 5.10 Broadcasting scheme of close-to-boundary nodes is incomplete (left), but the situation is salvaged by adding a single layer of virtual nodes (right). Virtual nodes are shown as circles.

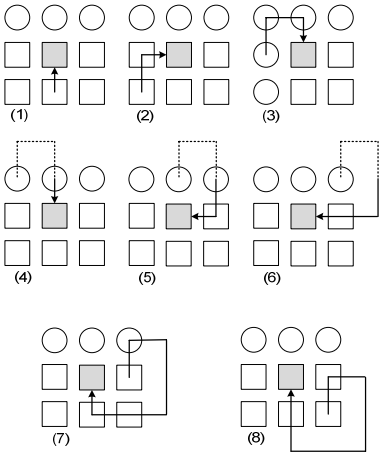


Figure 5.11 One layer of virtual nodes does not complete the broadcasting scheme of top boundary nodes.

Table 5.3 Additional actions in boundary nodes remove the need of virtual nodes.

Step	Top boundary node	Bottom boundary node
(1)	Send E (instead of N). Store W-value locally.	Use own value. u/y register shouldn't be updated
(2)	----	-----
(3)	Use W-value (instead of u/y-register value)	-----
(4)	Use W-value	-----
(5)	Use own value	-----
(6)	Forward own value W	-----
(7)	Forward own value S	Forward W; Receive E (instead of S).
(8)	-----	Forward own value W; Receive E (instead of S).

Implementing the actions in Table 5.3 introduces the need for boundary nodes to, sometimes, send or receive two packets simultaneously, which requires a remarkable redesign of the nodal controller and the router. Furthermore, there is need for an additional register that keeps one value (W-value in the table). Once again, different boundary nodes will require different refinements. This is of course better than the virtual nodes approach, but still increases the area considerably. A better solution makes use of the existing routing mechanism to forward boundary conditions. It is here denoted *swing boundary broadcasting* as each boundary node will send its own value to one

neighbouring boundary node and then to the other boundary node in the opposite direction. Due to the use of duplex lines between the nodes, the inter-nodal connections have to be idle for one time step in between (Figure 5.12). In this case, all boundary nodes will have the value of their neighbouring boundary nodes available locally. This requires two additional buffering elements to store the values, but the effect on area utilization is kept at a minimum. Overall, 3 time steps are introduced for each newly calculated y-value.

5.4 DISCUSSION

The moral of the serialized broadcasting approach is that the transfer needs to be sequenced when the communication bandwidth is limited due to area restrictions. The nodal control demands are kept at a minimum by interlacing communication and computation needs. Local storage needs are reduced as well, due to the need of holding one value only locally at any time step. A realization of the serial broadcasting approach, hosted on a Xilinx Virtex-II FPGA, shows reduced area utilization (Figure 5.13). Special attention should be paid to the smaller network interface of the switched serial approach compared to the two-steps approach employed in Caballero. The simplicity of the serial scheme eliminates the complexity of the router, which affects the total size of the node! Note that the serial architecture occupies fewer slices than the almost-interface-less state-flow architecture in spite of that the latter requires less flip-flops and equal number of LUTs. This has probably to do with a more balanced logic usage among the functional components (Figure 5.14).

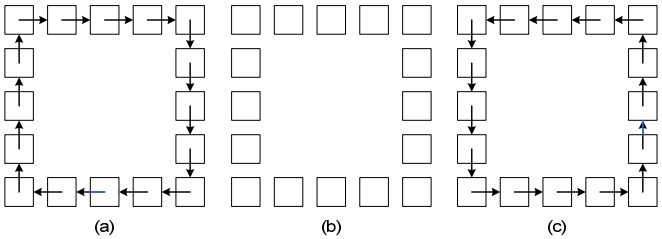


Figure 5.12 Swing broadcasting allows distributing of boundary conditions in two steps clock-wise (a) and anti-clock wise (c). For proper functionality on the duplex lines a separating idle step is introduced (b).

It is also found that by doing so by state machines not only leads to architectural rigidity but also to degraded performance. For instance, 30% of the utilized area in the serial broadcasting scheme is occupied by the controller, i.e. state machine, (Figure 5.14). One way to eliminate the need of the nodal controller, at least partially, is by transferring all values in a source-addressed packet. The original data-only packet used previously is padded with a small header containing the position of the source node in the grid. Hence, the packets carry their addressing information in the header, which can be exploited in two different ways.

In a traditional approach the packets will be stored in distinct destination registers accordingly. In this case, as many registers as there are neighbouring nodes are required. For the minimal 1-neighborhood, this means 9 registers. This is not as bad as it sounds. Registers are mapped on Flip-flops only and no LUTs are used. The present imbalance in the number of LUTs and Flip-flops, shown in Figure 5.14, allows for more Flip-flops without affecting the overall number of slices. In this way, an eventual architectural rigidity is removed with no impact on area utilization.

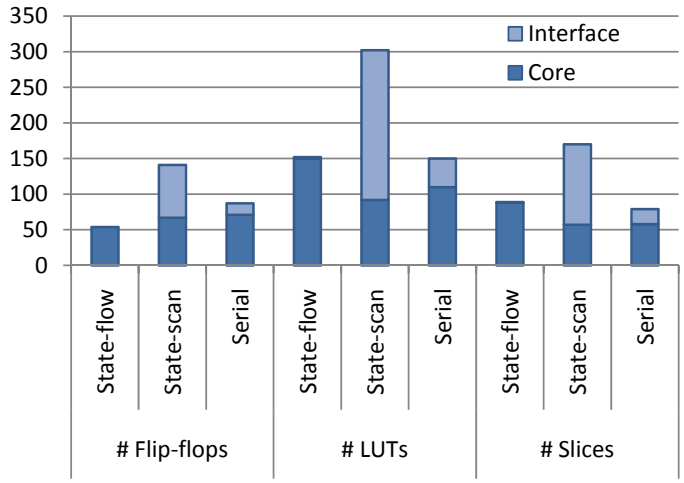


Figure 5.13 Area utilization per node compared to state-flow and state-scan architectures shows that nodal interface is kept at minimum which improves the overall logic utilization.

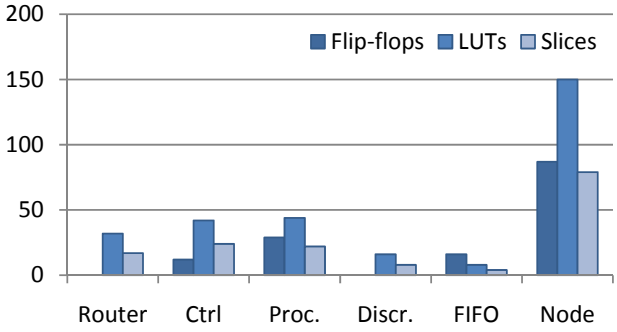


Figure 5.14 Area utilization of the different components with serial broadcasting scheme.

A better approach makes use of the intrinsic positioning information carried in the header to address the local template memory of the current node. The nodal equation, as given in Eq. (2.34), is then performed in the manner the

packets are received. The logic required for the addressing of the value/coefficient pairs is greatly reduced through the use of a mirrored binary numbers of both the rows and the columns. In case of a 1-neighborhood only 2 bits for the row and 2 bits for the column address are required. In general we need only $2(r+1)$ bits, where r is the neighbourhood.

It is also possible to merge the local controllers. The network is divided into small groups with each a single controller (Figure 5.15). This semi-global control approach does not affect the guaranteed performance but will lead to logic optimization. It adds some wiring overhead and therefore slow down the system but the gained amount of logic from reducing the number of nodal controllers is far much larger. Attention has to be paid so the average wire length is not increased to such a limit that the potential benefits of the CNN are lost [84]. The rate of one controller per neighbourhood seems to be a good trade-off.

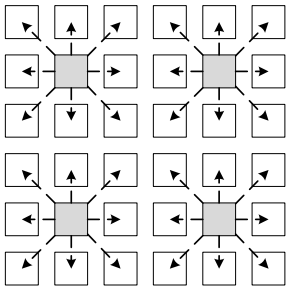


Figure 5.15 Semi-global control requires one controller per group of nodes.

The complexity of communication control is reduced in the serial scheme due to the pre-scheduled sequential arrival of neighbouring nodes' packets. A side effect is a simplified switching mechanism in the router, which saves in area. An additional register is, however, needed to store the constant obtained from computing the contribution of the control template and the bias. This constant is kept unchanged for all subsequent iterations. Finally, the usage of BRAM to implement the discrimination look-up table constitutes a main hindrance for accommodating more nodes on FPGA. It couples the number of nodes tightly to the availability of BRAM components on chip, even if area utilization allows for more functional units.

In a typical vein feature extraction application [51] we find that different templates need to be applied to the same image, and the two results need to be used in a next dyadic operation to bring a single result on which again two different templates are applied, and so on. With the current implementation, we can reduce the amount of external memory access, as each frame only has to be loaded once. Additional registers are simply added for each sub-operation. Factually, in this application we need to go through 7 subsequent CNN layers and never have to re-load from external memory. This provides us with an amazing $20\times$ higher performance, making real-world, real-time and real-power product applications possible.

Chapter 6

Memory Considerations

After the introduction of Cellular Neural Networks as a generic solver of non-linear differential equations, most of the work has been in image processing. An image-processing task applies a sequence of simple templates to each image in succession. Assuming that a single iteration of a given template requires t_{templ} time units, the question is how this relates to the time needed to complete the whole task t_{task}. Apparently, it depends on the CNN architecture. In the Bi-i camera [89], the core of system is an analogue chip supported by a DSP for non-CNN functionality and embedded in a digital programming environment [28]. In line with the earlier discussion in Chapter 3, the employed core is analogue in order to achieve the high network density that is required to handle image of sufficient size. Digital implementations were simply disregarded as the massive amount of multiplications in a typical CNN computation would otherwise be too area consuming.

As we already have seen, even if this is true for current techniques and technologies, the situation is far from impossible to change. The series of digital implementations discussed in Chapter 4 shows that a notable increase in network density can be obtained by careful handling of the internal communication. The problem remains to make choices in the wealth of architectural alternatives.

However, three bottlenecks have to be removed for the digital CNN camera to become reality. The first issue is the memory bandwidth. In contrast to the

Major parts of this chapter have been published in [V] and [VIII].

focal-plane approach, digital implementations are bounded by availability of, at least, one frame of the image stream on external storage. The second issue is the on-chip storage requirements as such storage alleviates the effect of processor-to-memory band gap. Lastly, the computational efficiency of the network implementation needs attention. The chapter presents these aspects and points out what makes the digital CNN camera viable. In section 6.1, two formulas that express the influence of data fetch from memory are derived. Then section 6.2 looks into the processor/memory band gap and how this works out for two of the digital implementations, i.e. ILVA and Caballero. The main issue here is the effect of slicing (windowing) on the overall performance. Subsequently, the situation when slicing is not required is discussed in section 6.3. Finally, the chapter is closed with a discussion.

6.1 OFF-CHIP AND ON-CHIP STORAGE

Nowadays most development boards use Double Data Rate (DDR) SDRAMs for main memory. Their name is derived from the fact that they transfer the data on both rising and falling edges of the bus clock. The DDR2 standard adds to that a doubling of bus versus memory clock so that effectively four data words are transferred per memory cell cycle. This is collectively measured in "data transfers per second per pin", which means that the bandwidth is related to both the data rates per pin and the width of the data bus (Table 6.1). Memory bandwidth is calculated as data transfers per second multiplied by the number of bits in a data word.

Table 6.1 DDR/DDR2 SDRAM JEDEC standards [90]

SDRAM Standard	Memory clock (MHz)	Cycle time (ns)	I/O Bus clock (MHz)	Mega data transfers per second
DDR-200	100	10	100	200
DDR-266	133	7.5	133	266
DDR-333	166	6	166	333
DDR-400	200	5	200	400
DDR2-400	100	10	200	400
DDR2-533	133	7.5	266	533
DDR2-667	166	6	333	667

If memory bandwidth (in bits) and speed grade are denoted w_{mem} and s_{mem} respectively, then the time required to fetch a frame of values is given by Eq. (6.1). Here, w_d stands for the width in bits of input/output values in the CNN, while r_{cnn} and c_{cnn} represent the number of rows and columns in the CNN respectively.

$$t_{fetch} = \frac{w_d \cdot r_{cnn} \cdot c_{cnn}}{w_{mem} \cdot s_{mem}} \tag{6.1}$$

From Eq. (6.1), one may conclude that the wider the memory is the smaller the data fetch time, which reduces the overall execution time as will be seen

later. Figure 6.1 illustrates how the choice of DDR memory affects data fetch time. The figure is valid for a CNN with 100 nodes and with $w_d = 8$ bits.

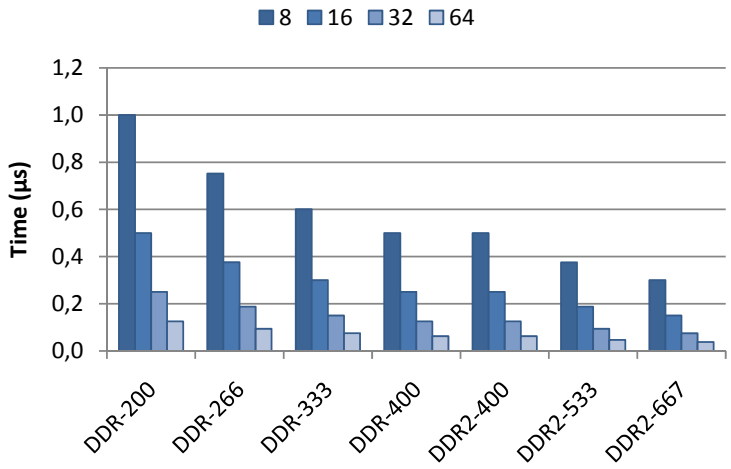


Figure 6.1 Data fetch time versus memory bandwidths.

Where the image is entered line-by-line, there are two basic approaches: the lines are buffered (state-scan approach), or the network is fitted to consume the lines as they come in (state-flow approach). In the former approach, the relation in Eq. (6.1) can be used straight forward, but it needs modification when the state-flow approach is considered. Here, a fetched scan-line is consumed directly, which has great influence on the overall performance of the system as will be seen soon. In this sense, if a scan-line is mapped on a column of nodes (as in ILVA), the time needed to fetch one line from the external memory is obtained according to Eq. (6.2).

$$t_{line\_fetch} = \frac{w_d \cdot r_{cnn}}{w_{mem} \cdot s_{mem}} \tag{6.2}$$

Intuitively, the basic constraint that every CNN realization should take into consideration is that memory latency becomes a bottleneck of the performance if the inequality given in Eq. (6.3) is valid. In other words, the time needed to fetch the desired amount of data from the external memory should not exceed the time it takes to perform the entire sequence of templates on data currently available in the CNN. Otherwise, the system must be halt waiting for the fetching procedure to be completed. In fact, this is valid for state-scan architectures only. For the state-flow approach, memory latency is taken care of intrinsically by the pipeline itself. Here, the requirement is as given in Eq. (6.4).

$$t_{mem} > t_{task} \tag{6.3}$$

$$t_{line\_fetch} \leq t_{stage} \tag{6.4}$$

For the state-scan architecture this enforces the need for on-chip buffering. The idea is to fetch the next image from memory and store it locally on chip, while the CNN processes the current image. This has to be replicated for the resulting output as it has to be buffered before it is sent to the external memory. This will clearly bind a large amount of the available logic on chip as storage elements.

The NoC-based approach, Caballero, resolves the need of pre-buffering partially through the employment of a FIFO-structure. Once the input values are brought in, the template is iteratively applied. The number of iterations, n_{iter}, is in principle infinite. The time needed to apply a certain template for a number of iterations on one frame depends on time to exchange information between the nodes in addition to template execution time. The time needed to perform the whole task is assumed enough to bring in the data of the next image from the external memory locally to each node. The number of templates and the complexity of each template, i.e. the number of iterations, are crucial to overcome the memory-band gap. The basic performance constraint given in Eq. (6.3) is then fulfilled.

This is however true if and only if a whole image can be accommodated on the CNN. This memory limitation is crucial for certain choices of FPGA chips. For XC2VP30, the maximum available BRAM is 306 KB. The additional storage provided as distributed memory is of marginal importance as it can accommodate 53 KB maximum! Assuming a pixel is represented by 8 bits (greyscale), accommodating one PAL image on CNN will need $720 \times 576 = 414720$ bytes to be available on chip, i.e. at least 405 KB of RAM is required. This assumes that RAM is not needed for storing any kind of data apart from pixel information; one pixel per node.

We may therefore rightfully assume that the CNN network can handle only a part of the image at a time. It has been suggested earlier, that *striping* the image may solve this problem. Now, a smaller part of the image is fetched from memory which decreases the latency, but not more than one template out of the given sequence can be applied before the next slice of the stripe has to be fetched. Furthermore, handling the edges of slices adds to the complexity as a certain degree of overlapping is required.

In the following, a frame execution formula is derived to evaluate the effect of slicing for each of the digital realizations. We aim for a unified notation and make the following assumptions:

- ★ Input values are brought per pixel line into a CNN column. Subsequent pixel lines will take subsequent columns.
- ★ Inter-nodal broadcasting is instantaneous, i.e. it doesn't add any delay to the system.

6.2 COMPUTATIONAL EFFICIENCY

In general, the nodal output execution time, t_{templ}, can be further divided into 2 parts:

- t_{const} : the time needed to calculate the control contribution along with the bias, i.e. $\sum Bu + i$, once per input pattern.
- t_y: the time needed to calculate the iterative part of the state equation, i.e. $\sum Ay$, followed by discrimination.

The first part needs to be performed only once for the given input-pattern, while the second part is repeatedly performed depending on the required number of iterations. Obviously, t_{const} and t_y depend on the r-neighbourhood, and so does t_{templ} as well. For all digital realizations carried out so far it shows that $t_{const} = t_y$. Therefore the common notation t_{comp} is used when no ambiguity rises. Hence, template execution time is basically obtained according to Eq. (6.5)

$$t_{templ} = t_{const} + n_{iter} \cdot t_y = (1 + n_{iter}) \cdot t_{comp} \tag{6.5}$$

In a state-scan architecture with a 1-to-1 mapping between digital nodes and CNN cells, the time needed to initially fill the network with data depends on the total number of columns in the design, c_{cnn}, and the time needed to fetch one line of the frame, t_{line_fetch} , as illustrated in Eq. (6.6). Hence, frame execution time is calculated according to Eq. (6.7).

$$t_{fetch}^{cab} = c_{cnn} \cdot t_{line\_fetch} \tag{6.6}$$

$$t_{frame}^{cab} = t_{templ}^{cab} + t_{fetch}^{cab} = (1 + n_{iter}) \cdot t_{comp} + c_{cnn} \cdot t_{line\_fetch} \tag{6.7}$$

This is, however, true only if the size of network is large enough to accommodate the whole frame. Slicing the frame introduces a number of complications. The number of slices depends on the size of both frame and CNN as shown in Eq. (6.8), where r_{frame}, c_{frame} and r_{cnn} stand for the number of rows and columns in the processed frame, and the number of rows in the CNN respectively.

$$n_{slice}^{cab} = \frac{frame\ size}{CNN\ size} = \frac{r_{frame} \cdot c_{frame}}{r_{cnn} \cdot c_{cnn}} \tag{6.8}$$

Two cases may rise depending on the relation between template execution time and data fetch time:

- ★ If $t_{fetch}^{cab} \leq t_{templ}$, frame execution time is then dependent on the number of slices and template execution time. All output values corresponding to the inputs of the entire frame have to be available before next iteration is performed. In other words, a single iteration has to be completed on each slice until the whole frame is processed before the next iteration is performed on the first slice of the next frame and so on. As the procedure of fetching overlaps with the computational part, due to the usage of FIFO-structure, Caballero is idle only when the first slice is brought in and the last slice is moved out. In Eq. (6.9), frame execution time is given as function of frame size, CNN size, number of iterations, and data fetch time. Note that the obtained formula differs from the one in Eq. (6.7).

$$t_{frame}^{cab} = n_{slice}^{cab} \cdot n_{iter} \cdot \left(t_{const} + t_y\right) + 2 \cdot t_{fetch}^{cab} = \\ = 2\left(\frac{r_{frame} \cdot c_{frame}}{r_{cnn} \cdot c_{cnn}} \cdot n_{iter} \cdot t_{comp} + c_{cnn} \cdot t_{line\_fetch}\right) \quad (6.9)$$

★ If $t_{fetch}^{cab} > t_{templ}$, frame execution time depends only on data fetch time as shown below.

$$t_{frame}^{cab} = n_{slice}^{cab} \cdot n_{iter} \cdot t_{fetch}^{cab} = \\ = \frac{r_{frame} \cdot c_{frame}}{r_{cnn} \cdot c_{cnn}} \cdot n_{iter} \cdot c_{cnn} \cdot t_{line\_fetch} \quad (6.10)$$

In contrast to Caballero, ILVA has an implicit bound on the number of iterations by the size of the implementation. As the nodes are arranged in pipeline stages, and the iterations are mapped on the pipeline stages; the maximum number of performed iterations is one shorter than the number of pipelines n_{pipe}. The first pipeline stage is used to calculate the constant part, while each of the following stages completes the computation of state and corresponding output. In all stages, the operation is accomplished during time t_{pipe}. Therefore, ILVA's template execution time (Eq. (6.11)) differs from the one previously obtained for Caballero (Eq. (6.5)). The calculated time is precise in Caballero, while it is on average in ILVA.

$$t_{templ}^{ILVA} = \frac{n_{pipe} \cdot t_{pipe}}{n_{pipe} - 1} \quad (6.11)$$

The pipelining mechanism requires only one (sub-) line of the frame to be present prior to computation start. ILVA consumes the fetched line directly but still experiences a latency that equals three times t_{comp}. An overall latency rises from the fact that the pipeline has to be filled before the first output values are produced. This is reflected in Eq. (6.12). However, when the pipeline is filled, a new output value is produced each t_{comp}. In other words, pipeline execution time t_{pipe} can be replaced by t_{comp} without loss of generality.

$$t_{frame}^{ILVA} = c_{frame} t_{templ}^{ILVA} + t_{line_{fetch}} + latency = \\ = c_{frame} \frac{n_{pipe} \cdot t_{comp}}{n_{pipe} - 1} + t_{line\_fetch} + 3\, t_{comp} \cdot n_{pipe} \quad (6.12)$$

Slicing of the processed frame is required when $r_{frame} > r_{cnn}$. Number of slices is then given as:

$$n_{slice}^{ILVA} = \frac{r_{frame}}{r_{cnn}} \quad (6.13)$$

In line with the earlier discussion, two different cases are distinguished:

★ $t_{line\_fetch} \leq t_{templ}$, frame execution time depends mainly on the t_{comp} and n_{slice}^{ILVA} as shown in (6.14).

$$t_{frame}^{ILVA} = n_{slice}^{ILVA}\left(\frac{c_{frame} \cdot n_{pipe} \cdot t_{comp}}{n_{pipe} - 1}\right) + t_{line\_fetch} + 3\, t_{comp} \cdot n_{pipe} \quad (6.14)$$

$$= \frac{r_{frame}}{r_{cnn}} \left(\frac{c_{frame} \cdot n_{pipe} \cdot t_{comp}}{n_{pipe} - 1} \right) + t_{line\_fetch} + 3\, t_{comp} \cdot n_{pipe}$$

★ $t_{line\_fetch} > t_{templ}$, frame execution time depends mostly on data fetch time:

$$t_{frame}^{ILVA} = n_{slice}^{ILVA} \cdot t_{line\_fetch} + 3\, t_{comp} \cdot n_{pipe} = \\ = \frac{r_{frame}}{r_{cnn}} \cdot t_{line\_fetch} + 3\, t_{comp} \cdot n_{pipe} \tag{6.15}$$

Due to the different mechanisms employed in state-flow and state-scan architectures, a straightforward comparison of frame execution times, as given in equations (6.9) and (6.14) respectively, is not feasible. A key factor is the number of iterations a given template is performed. In ILVA, this number is tightly coupled to the number of realized columns, i.e. $n_{iter} = n_{pipe} - 1$. Allowing more iterations will render the comparison unfair as it violates the intrinsic limit of functionality in ILVA. However, if less iterations are required, i.e. $n_{iter} < n_{pipe} - 1$, the superfluous pipeline stages should be removed and replaced, if possible, by nodes in such a way that the total number of rows in ILVA is increased. Equation (6.16) explains the relation between the number of rows in ILVA and Caballero.

$$r_{cnn}^{ILVA} = \begin{cases} r_{cnn}^{cab} & if\ c_{cnn}^{cab} \le n_{iter} \\ r_{cnn}^{cab} \cdot \left\lfloor \frac{c_{cnn}^{cab}}{n_{pipe}} \right\rfloor & otherwise \end{cases} \tag{6.16}$$

In the following, the comparison is arranged such that first a single iteration, $n_{iter} = 1$, and then several iterations, up to $n_{iter} = c_{cnn} - 1$, are performed on both architectures. This will, with respect to Eq. (6.16), yield the different settings given in Table 6.2.

Table 6.2 The actual number of rows in ILVA as a function of the number of pipelines and number of columns in Caballero. Parameter r represents the total number of rows in Caballero.

Iter	# Pipelines	Number of rows in ILVA							
		$r_{cnn}^{caballero}$	6	7	8	9	10	11	12
1	2		3r	3r	4r	4r	5r	5r	6r
2	3		2r	2r	2r	3r	3r	3r	4r
3	4		r	r	2r	2r	2r	2r	3r
4	5		r	r	r	r	2r	2r	2r
5	6		r	r	r	r	r	r	2r
6	7		r	r	r	r	r	r	r

Given a DDR-200 SDRAM with $w_{mem} = 16$ bits, Figure 6.2 illustrates the difference in data fetch time between ILVA and Caballero when $w_d = 8$. We know from Eq. (6.6) that data fetch time for Caballero depends on the number of columns as well, which is reflected in the figure.

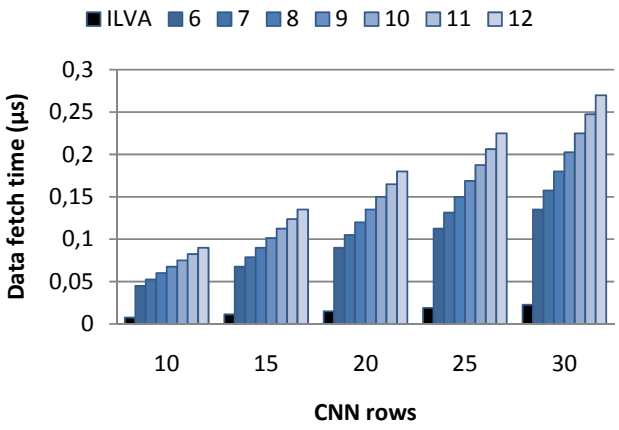


Figure 6.2 Data fetch time as function of the number of CNN rows when DDR-200 is used. The time increases linearly with the number of columns in Caballero while it is independent of pipeline depth in ILVA.

In order to express frame execution times in seconds, both ILVA and Caballero are assumed to run on 100MHz, resulting in $t_{comp} = 10^{-7}s$ in both realizations. We assume further that a PAL frame of size 720×576 is stored on an external storage of type DDR266 with $s_{mem} = 266 \times 10^6$ and $w_{mem} = 16$ bits. With respect to equations (6.16) and (6.2), Figure 6.3 and Figure 6.4 illustrate frame execution times with different sizes of the realized CNN. To fulfil the condition $t_{fetch}^{cab} \leq t_{templ}$, a DDR-266 or higher should be used.

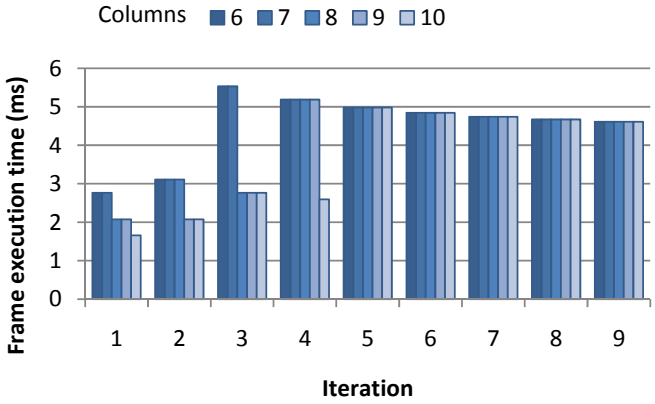


Figure 6.3 Frame execution time for ILVA with different CNN sizes, when slicing is required. The legends, 6 to 10, represent the number of pipelines, i.e. the number of columns in the design.

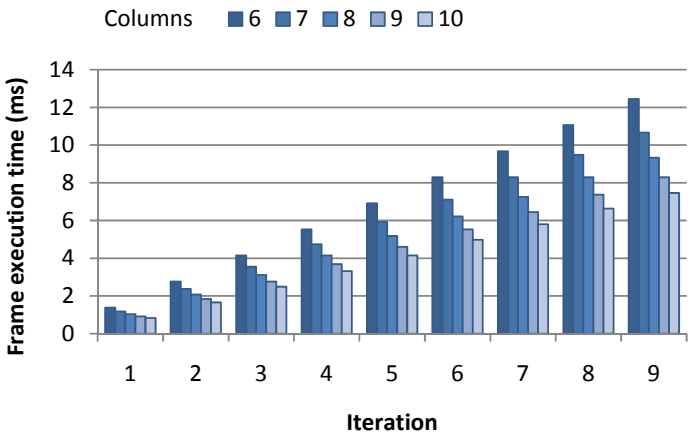


Figure 6.4 Frame execution time for Caballero with different CNN sizes, when slicing is required.

The figures show clearly that the state-flow architecture outperforms the state-scan architecture for all CNN-sizes when larger number of iterations per template is required. Caballero is better when 1 or 2 iterations are needed. This is caused by the need to swap all slices in and out for each iteration. On the other hand, if a sequence of iterations is allowed on the same slice before next slice is brought in, a different situation arises. This requires a slight modification of Eq. (6.9) as shown in Eq. (6.17). The resulting execution time is reflected in Figure 6.5. Here, it is noticed that for more accommodated columns Caballero performs better, almost regardless of the number of iterations.

$$\begin{aligned} t_{frame}^{cab} &= n_{slice}^{cab} \cdot t_{templ} + 2\, c_{cnn} \cdot t_{line\_fetch} \\ &= \frac{r_{frame} \cdot c_{frame}}{r_{cnn} \cdot c_{cnn}} \cdot (n_{iter} + 1) \cdot t_{comp} + 2\, c_{cnn} \cdot t_{line\_fetch} \end{aligned} \tag{6.17}$$

6.3 Moving Away from Slicing

The alternative to slicing is *pixel sampling*, where each CNN cell will correspond to the average of a pixel block rather than just one pixel. This can initially be done for the entire image and then repeated for smaller parts thereby gradually focusing in to the region of interest. Template sequencing is not a problem, nor have boundary conditions to be communicated between the handling of different image parts. To make this work requires a reasonable CNN network size to limit the overhead in reaching the region of interest. On the other hand, it has only to be done once in the task as the region of interest will be the same for all templates in succession.

The conclusion is that a Caballero-like architecture overcomes memory latency if and only if

- the size of the CNN allows for a rapid determination of the region of interest, on which the succession of templates is applied. In this sense, a number of approaches can be used, such as pixel averaging and texture analysis algorithms.
- the task consists of a number of templates, with a total number of iterations such that the total elapsed time exceeds, or at least, equals the time needed to fill the FIFO-structure.

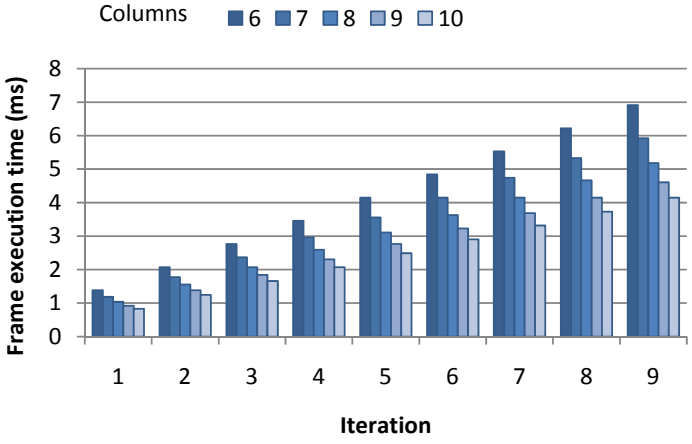


Figure 6.5 Frame execution time of Caballero is reduced when all the iterations are performed on a slice before next slice is brought in!

Having that in mind, the overall task execution time depends partially on how fast the region of interest is found, but mostly on the template set and the clock frequency of the digital design itself. Equation (6.18) provides a simple formula to calculate task execution time.

$$t_{task} = t_{region-of-interest} + \sum t_{templ} \tag{6.18}$$

The first part of the formula is independent of the architectural approach. The template computational part is, however, dependent on the efficiency of implementation and requires further attention. In the following, we focus on the contribution of this part only.

Consequently, the time required to perform a single frame is expressed in Eq. (6.19). The CNN is idle while the frame is brought into chip and moved out to memory, therefore the contribution of t_{fetch}. Apparently, frame execution time increases linearly with iteration count for a given network size, but data fetch time is dominant for larger networks (Figure 6.6)

$$t_{frame} = 2 \cdot t_{fetch} + t_{templ} \tag{6.19}$$

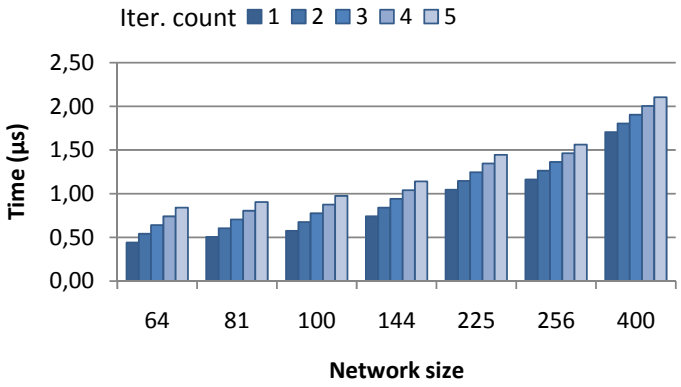


Figure 6.6 Frame execution time using DDR-266.

A task consists of a number of templates that are applied sequentially. In the extreme case a new frame need to be fetched from memory for each applied template. The overlapping between template execution and data fetching, which rises when two or more templates are sequentially performed, may complicate the derivation of task execution time formula. Intuitively, execution time of a task that consists of n_{templ} templates is expressed as:

$$t_{task} = 2 \cdot t_{fetch} + \sum_{i=1}^{n_{templ}} t_{templ}(i) \tag{6.20}$$

Substituting Eq. (6.5) in Eq. (6.20) gives

$$t_{task} = 2 \cdot t_{fetch} + t_{comp} \cdot n_{templ} \sum_{i=1}^{n_{templ}} n_{iter}(i) \tag{6.21}$$

Equations (6.20) and (6.21) are valid if and only if the condition in Eq. (6.22) is satisfied.

$$t_{fetch} \leq t_{templ}(k) \quad \forall \; k = 1, 2, \cdots, n_{templ} \tag{6.22}$$

On the other hand, if the inequality given in Eq. (6.22) is not valid for any of the sequentially performed templates, term $t_{templ}(i)$ in Eq. (6.20) has to be replaced by term t_{fetch}, and task execution time becomes:

$$t_{task} = (2 + n_{templ}) \cdot t_{fetch} \tag{6.23}$$

Figure 6.7 illustrates the effect of Eq. (6.22) for different DDR standards with memory word width of 8 bits. The task is assumed to consist of 6 templates that are performed sequentially with different iteration count; 2, 2, 1, 2, 2, 1 and 4 respectively. Obviously, the slower the memory the more dominant is data fetch time when the network gets larger.

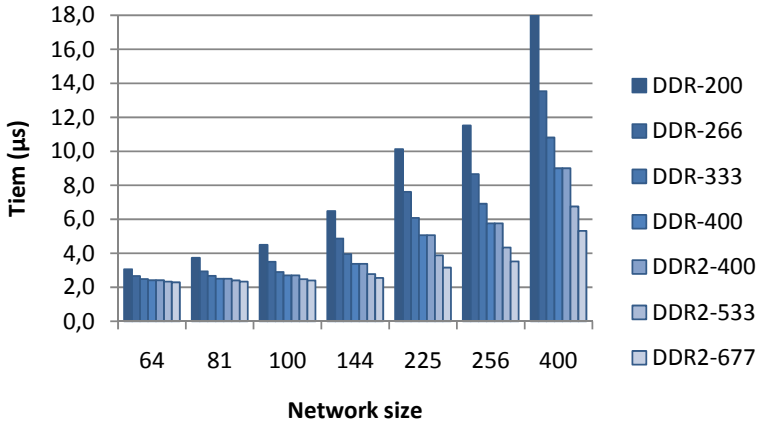


Figure 6.7 Task execution time for different SDRAMs according to Eq. (6.22).

For a given digital CNN implementation t_{comp} is fixed for a certain neighborhood. As most simple templates require a single iteration only, Eq. (6.20) and (6.21) are more likely to be considered as representative for task execution time. It is worth mentioning that, in most applications, the result of one template in the task serves as input for the subsequent template. The relation between t_{fetch} and t_{templ} is not important then, and Eq. (6.21) is always valid.

6.4 DISCUSSION

The development of classical computer architecture has shown an emphasis on computing acceleration by pipelining the central processing unit [81]. More and more the memory bottleneck becomes a concern. Of late, has attention moved to more spatially distributed methods, such as networked tiles, which offer inherent parallelism and local storage. The underlying assumption is that sequencing instructions over the local node takes the pressure away from the memory access by the many parallel executing tasks.

We see the same principle back in the research reported in this thesis. On one hand, we aim to have as much nodes executing in parallel as possible. This poses a severe burden on the memory bandwidth. Therefore it is required to do more locally. From inspection of existing CNN applications, one finds that data is accessed in memory more than once. Therefore bringing the amount of accesses down to 1 or less will easily pay the bill.

Actually, the question of communication with external storage units to bring in/move out values to/from the CNN array has never been answered satisfactorily in all available hardware realizations of CNN. In the conceptual CNN-UM [20] the problem is solved by photo transduction for input values while the proposed electromagnetic detection approach remains theoretical only. Factually, most famous mixed-signal architectures [24]-[30] incorporate the

optical input approach to overcome the memory access bottleneck and provide the promised computational speed when greyscale 8-bit input values are used.

Supporting more operations on the same data has two consequences. Firstly we need to maintain local copies. Then data can be used from the local store rather than from the external memory. Secondly we need to have a CNN cell that can easily change state and therefore implement a multi-level structure. A time-multiplexed node is such a cell.

The benefits will appear most clearly when the application is designed to optimize data re-usage. In [40] it is discussed how a CNN implementation can be derived by a morphological specification, such that operations are either in sequence (passing results from operation to operation) or in reconvergent parallel paths (using the same data and combining the individual outcomes into a single result). For instance, the typical image understanding algorithm, such as used for velocity measurement [52], runs all templates on the same set of frames (RoI) or on the result of a previous template. In slight modification of Eq. (6.22) fetching 2 frames must be done before a set of 7 templates is completely executed. As shown in Figure 6.8, this reduces the task execution time considerably for larger networks, read images.

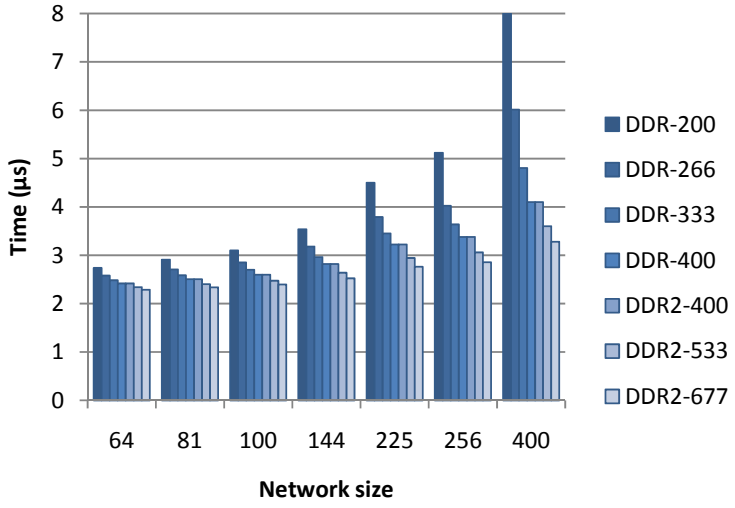


Figure 6.8 Task execution time with reduced data fetch. Compared to Figure 6.7, time reduction is obvious for larger networks.

Chapter 7

Applications

This chapter aims to discuss the different systems and to present some testing results. As image processing has always been a popular field of CNN applications, it feels natural to verify the significance of the different digital implementations presented in the previous chapters, by realizing a number of experimental systems to solve certain image processing tasks. Some of these tasks require, however, slight modifications in the system design to make the testing feasible. The popular Game of Life is implemented for the first time on FPGA in section 7.1. The interest in this game comes from the fact that it is especially this game that has brought glamour to the Cellular Automata [10]. In section 7.2 the link between picture enhancement and *Object Oriented Image Analysis* scheme is stressed, where the latter is employed to measure velocity of an OoI in a video sequence. Finally, a concluding discussion is given in section 7.4.

The hardware platform is an FPGA from Xilinx, particularly Virtex-II Pro P30, which is installed on a development board from Memec [46] that provides 4 external SDRAM memory blocks with a size of 32 MB. Additionally, the board is equipped with both serial and parallel communication ports, allowing for different communication schemes with a PC.

Parts of this chapter have been presented in [I], [II] and [III].

7.1 GAME OF LIFE

The Game of Life is not just an example of artificial life, but also an abstraction of a typical predator/prey situation. The Cambridge mathematician John Horton Conway, who spent a lot of time in manually finding the proper rules, has originally proposed it in 1970 but the game is popularised by Martin Gardner [53].

The game is played on an arbitrary board. The cells can be either populated or not (black or white). According to specific rules, a cell can change the population under influence of the neighbours. For instance:

A cell that is populated dies if at most one or at least four of the neighbours are populated (respectively loneliness and overpopulation), while it will become populated when three of its neighbours are.

Important is therefore the initial situation. For a number of starts, the effect has been recorded in literature. In Figure 7.1, one of the many ways to get into pattern oscillation is shown. As the Game of Life is a cellular automaton, it ought to be possible to accelerate the game by means of a CNN. The purpose of the project is therefore to demonstrate this acceleration. This is not a first in general, as the Game of Life has been demonstrated in analogue hardware [54] and in software, both on general-purpose as in vision hardware [55]. There are also conventional solutions implemented on FPGAs [56]; therefore the project aims to be the first System-on-Chip (SoC) on FPGA, taking the other solutions as quantitative references.

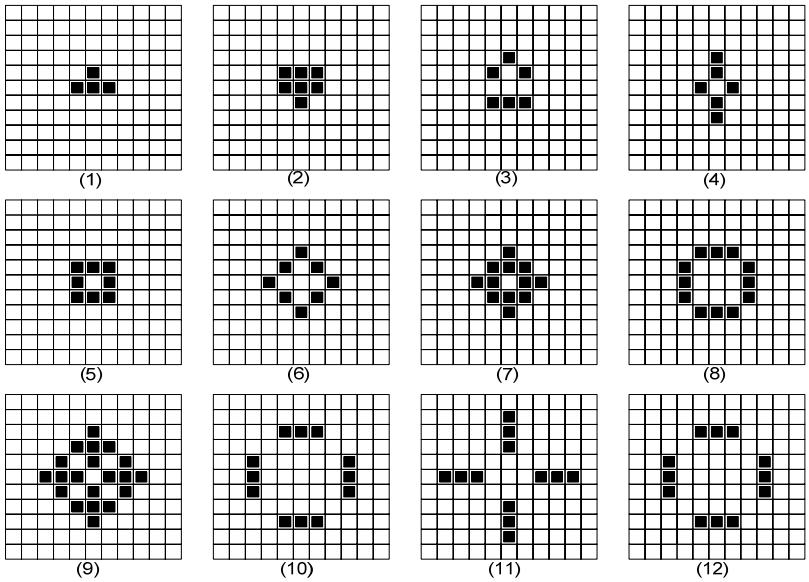


Figure 7.1 A Game of Life that never stops. A black cell is alive and turns white when it dies.

In the black-and-white version, only one convergence iteration is necessary per game step. The state of a cell changes between white (-1) and black (+1), with no other grey-levels in between. As the operation is local, it can, similar to e.g. the template of Logic Not, be performed by calculating the nodal formula only once. This pleads for the pipelined architecture, configured with a minimal depth. On the other hand, when the game is played with grey-level population, the choice of architecture is not straightforward and additional experience is required.

The actual operation is performed on the input values only. Because no feedback occurs, the need of feedback template $\boldsymbol{\mathcal{A}}$ is eliminated. As the contribution of neighbouring cells inputs is already normalized, a simple summation suffices for making a decision whether a cell is to be populated or not. Hence, the bias term is not needed either. Consequently, based on the cloning template, given in (7.1), a necessary condition for a cell to become/remain populated is that the state of the cell, x^c, equals 2 or 3, otherwise it is considered dead. However, the current state of the cell is decisive to obtain the proper output (Table 7.1).

$$\boldsymbol{A} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \boldsymbol{B} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \quad i = 0 \tag{7.1}$$

Table 7.1 'Truth table' for the game of life where all values follow the binary representation.

Input value u^c	State x^c	Output value y
0	0010	0
1	0010	1
-	0011	1
-	Otherwise	0

7.1.1 Implementation

A slimmed-down version of the Sleipner architecture has been chosen to perform the game. It has one column only, containing nodes that perform the control contribution, $\sum Bu$, only. The internal design per node is slightly modified to enable the realization of the truth table (Table 7.1). The nodal input that is one bit only is buffered and concatenated together with the computed state. The obtained word is then used to address a small memory providing the final output. Pixel data of the entire game board is saved on chip, using two BRAMs. This eliminates the need of communication with external SDRAM units and, thus, simplifies timing control. An input buffer is used to bring the pixels into the network from one RAM, and an output buffer to send the results back to the other RAM.

For simplicity, the procedure of testing the final hardware is replaced with a simple visualization mechanism. The target board is equipped with 8 light-emitting diods (LEDs) for verification purpose only [46]. Instead of sending the outputs of the network to a PC, the LEDs are fed with output values. A turned-on LED indicates a populated cell (black pixel value in the network). As only 8 LEDs are available the size of the network is decreased to consist of one column with 8 nodes as shown in Figure 7.2.

The initial state of the game enters the network according to the scheme described in section 4.3, i.e. a scan-line-by-scan-line. The LEDs show then a sequence of scan-lines, where each light combination corresponds to one output line. After all, the purpose of this project is to demonstrate the acceleration a CNN implementation can provide. Thus, replacing the procedure of testing with the simple visualization mechanism is acceptable.

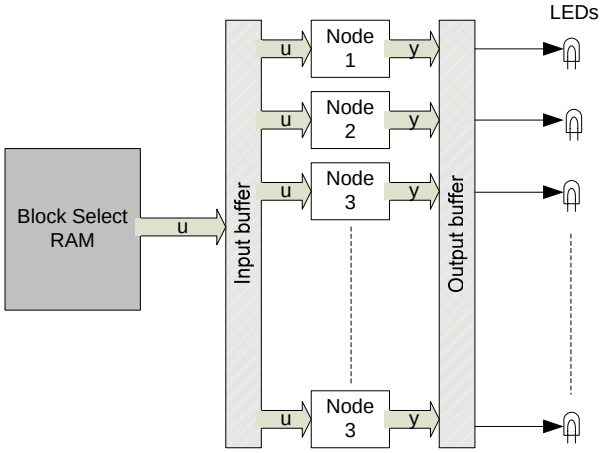


Figure 7.2 A schematic view of final design testing.

7.2 VELOCITY MEASUREMENT

In a typical CNN-based system as Vehicle License Plate Recognition (VLPR) [48], a character string of a well-defined composition is available on a clean background. Factors such as limited resolution and dirty surfaces complicate the situation. However, It is of increasing interest to determine and read text on arbitrary locations, i.e. not only license plates. Figure 7.3 gives an easy but common example. The same template set as in VLPR can be used for this purpose, but the consistency check is a bit more complicated. Once the image is extracted by means of a pre-processing system, a standard optical character recognizer (OCR) package in the loop will generally suffice for performing the desired recognition.

In professional VLPR systems, complications arise due to the speed of the car versus the angle, and quality of the camera. But such complications can also

appear in other situations, as exemplified in Figure 7.4. Here one wants to read the text on a passing train, as required for train management systems.

Figure 7.3 Reading the text from the E-building at Faculty of Engineering (LTH), Lund University (Sweden).

The problem has different aspects, each with their own importance. The first is movement detection: which object in the image actually moves, and with which speed and in which direction. Once such objects are found and labelled, the next issue is to track such an object, while diagnosing the reason for and character of the movement. A related aspect, for which we want to check the feasibility in this experiment, is whether the rendering of the moving object can be improved by using knowledge about the actual movement.

Figure 7.4 After edge detection on an image of Lund Railway Station, the text on the moving train can still not be read.

There are of course other (and maybe better) ways to measure object velocity than by smart vision. The subject was chosen because velocity estimation is crucial to dynamic face recognition [107]. A good CNN system must be stream-oriented: all the processing needs to be on the single image flow while reconfiguring the hardware. As literature only shows mixed hardware/software approaches, the focus is on showing how velocity measurement can be efficiently realized in hardware only.

For the application described in this section we use the NoC-based CNN implementation Caballero. The core of the design is a grid of regularly spread cells building the DT-CNN. In order to provide the core with image pixels, a serial-to-parallel unit makes use of an SDRAM controller to read data from one

of the external SDRAMs, where image frames are "pre-loaded ". The actual size of a frame, delivered by a PAL video camera, is 720x576 pixels. Hence, with 8-bits pixel width, each SDRAM is able to hold up to 80 frames, which is sufficient to complete the task of velocity estimation. However, two different SDRAM units are used; one for input image frames and one for storing resulting image frames. Output pixels are directed to the SDRAM by a parallel-to-serial unit that captures these pixels from several FIFO units aligned between the columns of the CNN. Figure 7.5 shows a schematic view of the complete design.

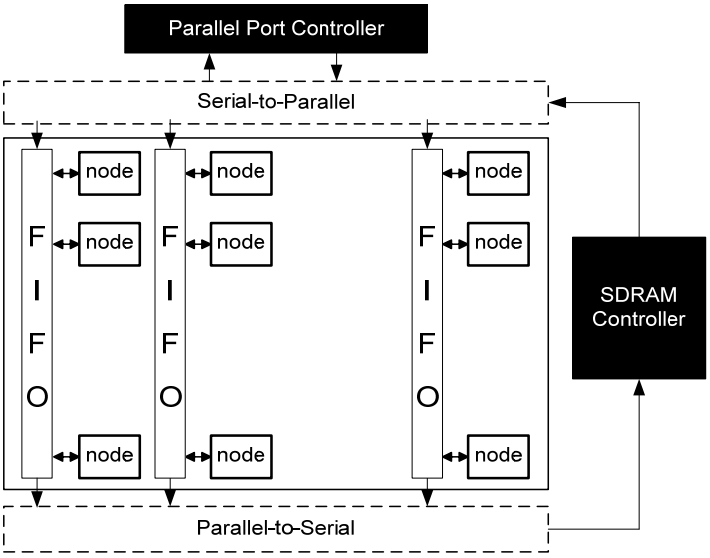


Figure 7.5 A schematic view of the design. Arrows represent data transmission between few units, but far from all data lines are shown in the figure.

In the following, the concept of velocity estimation is first introduced in 7.2.1 before the implementation of the algorithm using CNN basic operations is presented in 7.2.2. In subsection 7.2.3, the reliability of the algorithm is ensured by performing MATLAB verification, before the hardware design is tested.

7.2.1 Considerations on the Velocity Estimation Algorithm

Motion detection is a central scheme in various areas of vision sensing, both in industrial as in consumer applications. The concept is based on the simple observation that moving objects carry the most important features, in comparison to other details contained in background and still parts. Thus, detecting and coding the moving objects is essential for image understanding. The typical image-analysis algorithm consists of four main steps: 1) segmentation 2) parameter (motion) estimation 3) image synthesis and 4) consistency observation [107]. The most important and computationally most complex one of these steps is the task of segmentation, which is accomplished by cutting a scene into different moving objects (regions). These objects have to

be labelled and measured. Furthermore, consistency of object segmentation is an essential aspect to guarantee quality of the result. Therefore, maintaining a global uniform velocity is crucial to recognize the segmented objects [109].

The object-oriented image analysis scheme is widely accepted as an interesting and sophisticated approach for future video coding systems with very low bit-rate. By transferring the moving objects only, the transmission rate is greatly reduced [107]. In [109] the emphasis is on achieving object-oriented image compression for videoconferencing purposes on a CNN-UM hardware platform. The modelling of the motion can be eased from the understanding that labelled objects can only move within a Region of Interest: the remarkable features of a human face are the areas containing eye, nose, mouth and ear. However, any deterioration of facial expressions decreases image quality drastically. Thus different "quality enhancing" steps are needed, which makes the segmentation algorithm rather complex.

In our work, the nature of the problem is fundamentally different. A moving object is also changing location in its RoI and needs therefore also to be distinguished in every frame from a sequence of consecutive images. Once the object is segmented into a number of frames, the displacement of the object between two image frames must be extracted. Wrapping the moving object in an encapsulation box, eases the computation of the displacement. The displacement of the box corresponds to the distance covered by the object in reality. The establishment of this correspondence is a problem by itself and often forces a need for calibration. When the whole object is observed in one of the frames, the difference between two consecutive frames shows two leftovers instead of one. Additional effort is then required to relate these two as belonging to a single object.

The process of velocity measurement depends on a number of parameters, mainly coupled to the camera in use, such as image resolution, frame frequency f_f and view angle θ. Another parameter of importance is the distance between the camera and the moving object, d_p. Given f_f in frames/seconds, d_p in meters and θ in degrees, the width of the captured scenery is $d_a = 2d_p \cdot tan\,(\theta/2)$. Figure 7.6 illustrates a camera where the involved parameters are pointed.

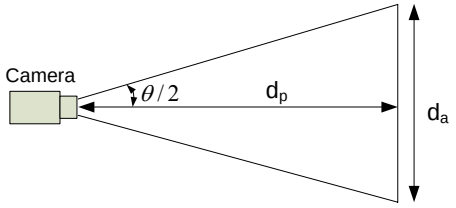


Figure 7.6 Mapping of the image on the pixel map.

An object with velocity v (meter/seconds) will cover the distance d_a (meters) in $t = d_a/v$ (seconds). During this time, the camera takes $N = t \cdot f_f = (d_a \cdot f_f)/v$ frames. In other words, if all the frames are super-imposed, there will be *N* instances of the moving object on a single frame. If W, in pixels,

denotes the width of frames delivered by the camera, the movement of the object corresponds to a displacement in pixels given by Eq. (7.2)

$$n_p = W/N = (W \cdot v)/(d_a \cdot f_f) \tag{7.2}$$

The minimum velocity that can be detected corresponds to a single pixel displacement of the object. The nature of the applied template complicates the calculation of maximum speed when the object is close to the edge of the frame. In order to overcome this limitation, a 5% margin of the total frame-width is provided on both vertical edges. Obviously, the maximum displacement by means of pixels is correlated to the maximum object-velocity that can be detected. Typical PAL camera specifications like: horizontal view angle of 60˚, 720 pixel wide frames, and frame-rate of 25 frames/s are utilized in Figure 7.7, where the displacements of a 3 meter long object are shown for different speeds. Obviously, the displacement depends on the distance d_p of the camera from the captured scenery in which the object moves. The size of the blob is dependent on d_p as well.

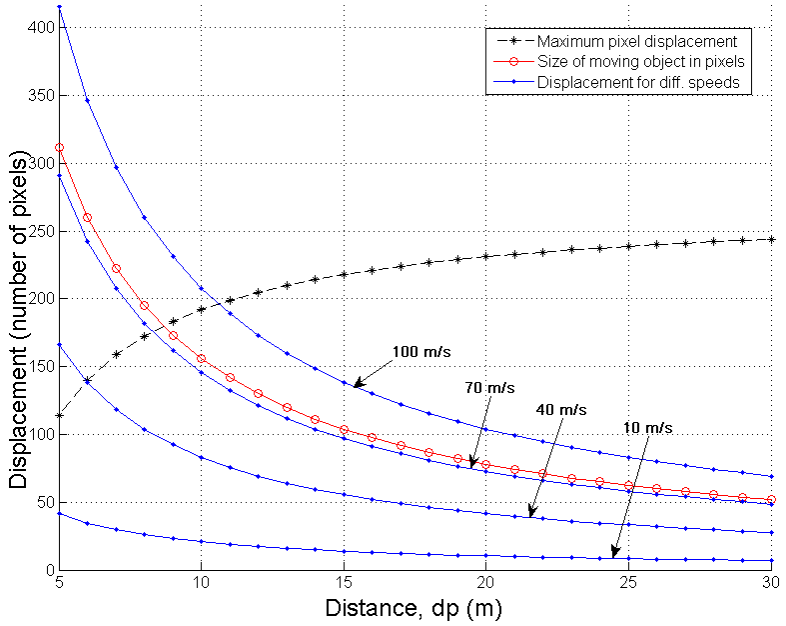


Figure 7.7 Pixel displacement versus observation distance for several object velocities.

7.2.2 The algorithm in basic CNN operations

The segmentation quality is significantly increased by pre-processing image-frames in order to remove disturbing noise. Noise filtering is achieved by iteratively applying the averaging template (Eq. (7.3)) to the image. The next step is to create a mask around the Object of Interest (OoI), i.e. the moving object. In order to extract the RoI, all background information has to be

removed. Calculating the absolute value of the subtraction of two consecutive frames in the image sequence completes the task. In other words, if f_1 and f_2 are first and second frames respectively, the result is given by $|f_1 - f_2|$. The resulting output map has its darkest pixels where both frames differ, while background information is covered in grey.

$$\mathcal{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \mathcal{B} = (1/8) * \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \qquad i = 0 \tag{7.3}$$

Although the output map includes all necessary contour pixels for segmentation, further contour enhancement is needed to achieve sufficiently exact segmentation. Contour lines vary in thickness due to the sharpness of object edges. An appropriate choice of the gradient threshold reduces the effect of sharp/smooth edges, but we follow the proposition made in [109] and apply the approach of *skeletonization* (Table 7.2). It performs iteratively in 8 subsequent steps, where each step peels one layer of pixels in a certain direction. As one iteration is accomplished, the pattern is one pixel thinner in all directions. The algorithm stops when no difference between the input and the output is obtained. Applying this powerful line-thinning algorithm iteratively reduces lines of arbitrary and varying thickness to their centre pixels.

Table 7.2 Different skeletonization templates corresponding to the direction of "peeling".

Direction	$\mathcal{A}$	$\mathcal{B}$	i
North	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 7 & 0 \\ -0.5 & -1 & -0.5 \end{bmatrix}$	-3
Northeast	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 1 \\ -1 & 7 & 1 \\ 0 & -1 & 0 \end{bmatrix}$	-3
East	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -0.5 & 0 & 1 \\ -1 & 7 & 1 \\ -0.5 & 0 & 1 \end{bmatrix}$	-3
Southeast	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 7 & 1 \\ 0 & 1 & 1 \end{bmatrix}$	-3
South	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -0.5 & -1 & -0.5 \\ 0 & 7 & 0 \\ 1 & 1 & 1 \end{bmatrix}$	-3
Southwest	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 7 & -1 \\ 1 & 1 & 0 \end{bmatrix}$	-3
West	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & -0.5 \\ 1 & 7 & -1 \\ 1 & 0 & -0.5 \end{bmatrix}$	-3
Northwest	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 7 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	-3

Quality of the intermediate result is however still reduced due to the presence of some isolated pixels that might even inhibit the creation of the encapsulation box around the moving object. These pixels are easily removed by applying the template of *Isolated Pixel Removal* (Eq. (7.4))

$$\mathcal{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \mathcal{B} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \qquad i = -1 \tag{7.4}$$

The resulting output map is now free of disturbing information and the task of segmentation is accomplished by thresholding the value of all pixels using a hard-limiter. A binary mask is then created and combining the corners of that mask easily creates a black box covering the object of interest. Figure 7.8 summarises all steps of the creation of the encapsulation box starting from two consecutive video frames. As the differences between the frames are not necessarily co-located, they must be linked in order to establish the OoI.

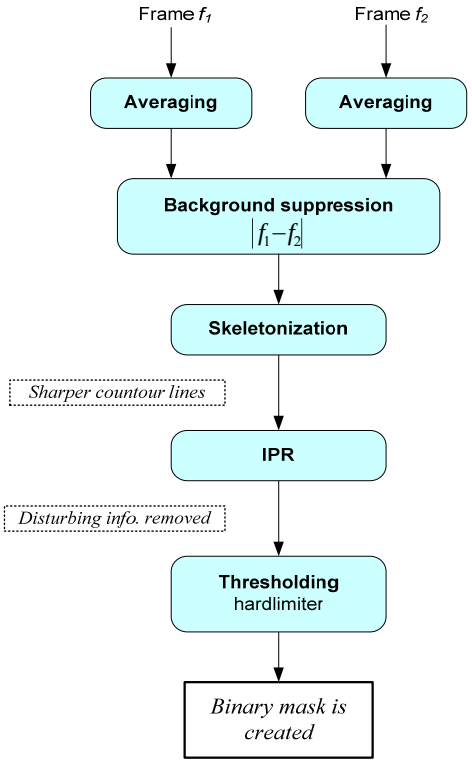


Figure 7.8 Template flow diagram in velocity measurement approach.

In the general case any blob that appears in both frames must be checked. The problem is eased when the direction of movement can be predicted from the past or by hypothesis. For instance a train will move along a track and its direction will be clear from the outset. As the early movement will show co-

located blobs (Figure 7.9.a and b), the direction can be predicted when the speed rises and object splitting occurs. Verification of the hypothesis can be achieved by enlarging the blob in the frames in the direction of the movement, followed by an AND operation. This supplies us already with a box at length of the displacement. We will see this extended algorithm in section 7.4.

When the blob has not split, we have to repeat the same segmentation procedure with the following pair of frames, i.e. f_3 and f_4, creating another black box. Two different facts are extracted by comparing the position of the boxes: motion direction and, most important, displacement of the moving object between frames f_3 and f_4. As time between consecutive frames is known, we only need metric information about one of the details in the scenery (preferably the moving object itself) to determine the velocity of the moving object. In Figure 7.9.c, the displacement of the moving object is illustrated with the difference Δ of the two black boxes.

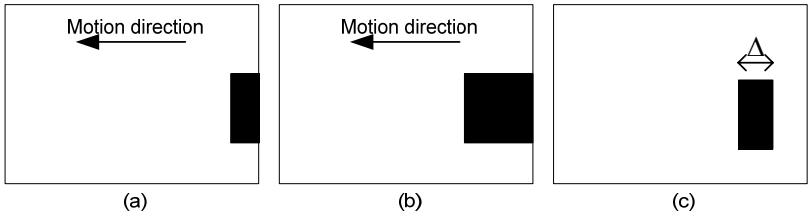


Figure 7.9 Measuring the displacement of an object moving from right to left in the scenery. Displacement (shown in (c)) of the moving object is the difference between the black boxes in (a) and (b).

7.2.3 Verification and Test

MATCNN is one of many software tools for CNN simulation provided in [59]. It is a flexible and easy-to-use test environment for single-layer CNNs. Although basically built to simulate analogue CNN implementations, it provides a very good toolbox (for MATLAB) to verify the approach. The toolbox is equipped with a library of many 3×3 templates.

In the following, all steps in the flow diagram in Figure 7.8 are applied to a number of frames captured from a video sequence of a locomotive of type SJR6. First of all, the averaging template is iteratively applied on the first two frames of the video sequence. Figure 7.10 shows the results after 25 iterations with a time step of 0.019τ.

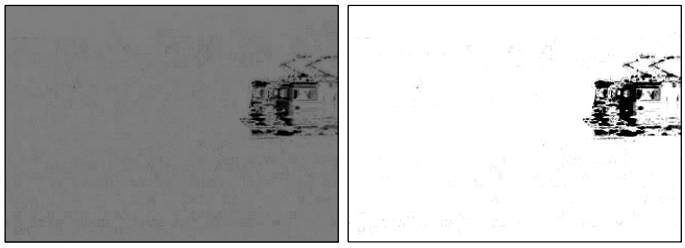
The background is then faded in order to outline the moving object, which is achieved by calculating $|f_1 - f_2|$. As seen in Figure 7.11.a, the background is replaced with grey pixels.

Assuming the object moves horizontally only, applying the templates of skeletonization for two directions only, i.e. west and east (Table 7.2), is sufficient. The templates are applied iteratively for 25 iterations each, with time step of $0.019\,\tau$. The result is shown in Figure 7.11.b, where isolated pixels are easily noticed. Isolated Pixel Removal is applied iteratively for 25 iterations

with time step 0.04τ to remove these pixels. Figure 7.12.a depicts the resulting image.

f_1 f_2

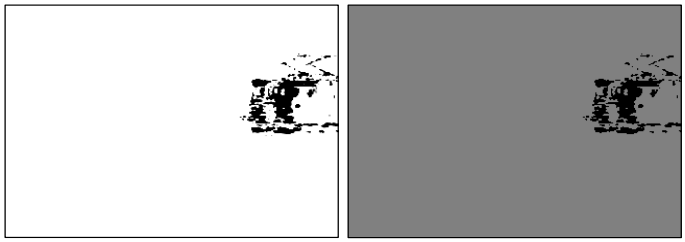
Figure 7.10 First two frames (f_1 and f_2) of the video sequence after applying the averaging template for a number of iterations.



(a) (b)

Figure 7.11 (a) Resulting image of $| f_1 - f_2 |$. Darkest pixels are observed where the two frames differ as most. (b) Intermediate result after skeletonization, where the isolated pixels can easily be noticed.

Segmentation of the moving object is accomplished by creating a binary mask using the hard-limiter function. Figure 7.12.b depicts the final result with the binary mask. Finally, the object of interest is covered with a black box, as shown in Figure 7.9.b.



(a) (b)

Figure 7.12 Applying the template of IPR removes all isolated pixels (a). Procedure of segmentation is completed once the binary mask is created (b).

Caballero performs all steps of the algorithm of velocity estimation from averaging to isolated pixel removal through skeletonization. Subsequent steps of the algorithm, i.e. creation of both the binary mask and the black box covering

the moving object, are performed on a PC by, preferably, using MATLAB toolbox. The task of post-processing is completed by calculating the value of Δ (see Figure 7.9.c) and, thus, estimating the velocity of the moving object. This design maintains the regularity of a CNN and paves the way for future modification, e.g. usage of two-layered CNN [108] needed for implementation of steps 2, 3 and 4 in image analysis algorithm mentioned before (section 7.2.1).

The design performs on the same video sequence used for verification. The intermediate result obtained after skeletonization (Figure 7.13) differs, however, from the expected result noticed in Figure 7.11.b. Quality reduction is due to the use of another squashing function compared to MATCNN. This indicates, however, the need for adjustment of the parameters in the table look-up. Instead, the error is attempted to be overcome by using the template of averaging (Figure 7.8) again between the operation of skeletonization and IPR. Figure 7.13 shows the intermediate result after each step. It is obvious that the quality of images is still much lower than obtained in software simulation, but as the aim of the experiment is to test the basic functionality of the design no further actions are taken.

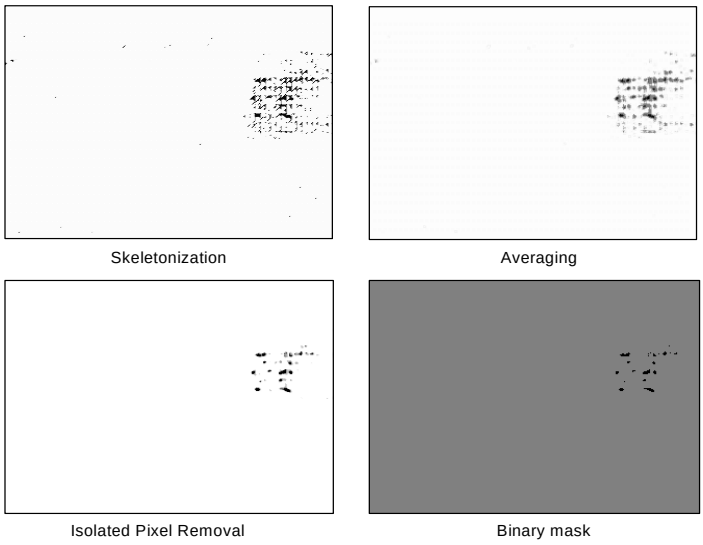


Figure 7.13 The intermediate results of all steps as obtained from the post place and route simulation.

7.3 VEIN FEATURE EXTRACTION

Modern security systems have to provide fast, accurate and robust personal identification, which implies moving away from traditional and unreliable methods such as PIN codes and smart cards. The use of electronically stored records of human biometrics features seems promising. The US Department of Defense started an experiment to replace existing ID-badges for 4.3 million

employees by using fingerprint readers from Precise Biometrics already in 2001 [92]. Recently, some European states have accepted a biometric signature as legally binding, and the UK government has placed in November 2005 biometrics identification technology on the short list of its Science and Innovation Strategy [93].

As the identification process is based on the unique patterns of the users, biometrics technologies are expected to provide highly secure authentication systems. However, the existing systems are very vulnerable. One's fingerprints are accessible as soon as the person touches a surface, while a high resolution camera easily captures the retina pattern. Thus, both patterns can easily be "stolen" and forged [93]. Beside, technical considerations decrease the usability for these methods. Due to the direct contact with the finger, the sensor gets dirty, which decreases the authentication success ratio. Aligning the eye with a camera to capture the retina pattern gives an uncomfortable feeling. On the other hand, vein patterns of either a palm of the hand or a single finger offer stable, unique and repeatable biometrics features.

Already in 2001, an experiment was reported where hand vein images were recognized with 99.45% success [94]. Images were cleaned and compared within 150 msec. The main bottleneck was the cost and performance of the sensor. Meanwhile Fujitsu has built a biometric palm vein scanner, while Hitachi presents a finger vein identification system [95]. In both cases, a thermal imager acquires vein images. Near-infrared rays generated by means of LEDs penetrate the hand and are absorbed by the hemoglobin in the blood. Thus, the veins (where the blood flows) appear as dark areas in an image taken by a CCD camera (Figure 7.14.b). Then image processing reconstructs a hand-vein pattern from the camera image. Finally, appropriate processing extracts the vein patterns from the images and performs a feature matching against reference images.

(a) (b) (c)

Figure 7.14 Typical biometric patterns; (a) fingerprint, (b) hand vein [96] and (c) human retinal angiograph [97].

In [97], it has been concluded that a Gaussian model for feature extraction is fairly successful; here we check the quality of a CNN-based feature extraction that has previously been demonstrated by Gao for fingerprints [98]. This section will go in phases through the feature extraction algorithm of Gao, while making modifications for handling veins. The pre-processing [99] is handled in subsection 7.3.1, and the extraction [98] & matching [100] in subsection 7.3.2. Subsequently we discuss the quality of the vein feature extraction (subsection 7.3.3) and give some details on an experimental realization (subsection 7.3.4).

7.3.1 Image Pre-processing

Normally, the captured vein pattern is greyscale and subject to noise. Noise Reduction and Contrast Enhancement are crucial to ensure the quality of the subsequent steps of feature extraction [101]. This is achieved by means of three operations: *Binarization* that transforms the gray-scale pattern into a black and white image, *Skeletonization* (Table 7.2) that reduces the width of lines to one pixel and finally *Isolated Pixel Removal* (Eq. (7.4)) that eliminates the unwanted isolated points. These three steps constitute the procedure of image pre-processing (Figure 7.15). Upon start, the original image is fed as input u, and the initial output $y(0)$ equals 0, while the intermediate results constitute the input of the the templates in the successive steps.

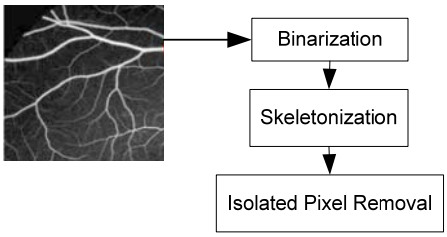


Figure 7.15 Image Pre-processing.

7.3.2 Feature Extraction

Blood vessels are characterized by means of length, thickness, shape and distribution of the veins. Only the length and distribution are taken into consideration as this enables a feasible matching of the overall pattern. As the operation of skeletonization masks out the shape and thickness, the thinned vein pattern has, similar to fingerprints, two main features: ending and bifurcation (Figure 7.16). The former is the end point of a thinned line, which reflects the length of the veins, while the latter is the cross section of three lines, which reveals the distribution of the veins.

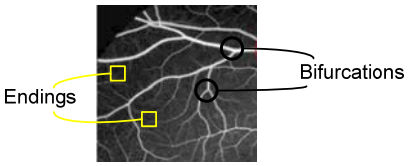


Figure 7.16 Vein features: endings and bifurcations

It is important to point out the existence of false features due to the noise in the original image and artefacts that may be introduced during the procedure of image pre-processing. As two false features are normally close to each other, they are handled in pairs. Actually, three different types exist: a pair with two false endings, a pair with two false bifurcations and a pair with one false ending and one false bifurcation [98]. Figure 7.17 depicts one of the cases that may arise during bifurcation detection.

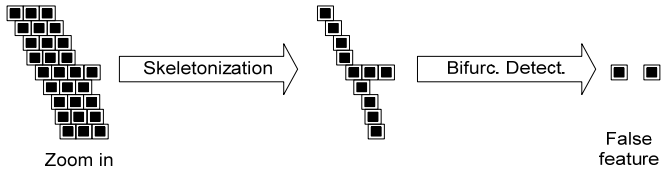


Figure 7.17 Bifurcation detection may give rise to false features.

The algorithm consists mainly of 4 different operations (Figure 7.18). First of all both bifurcations and endings in the pre-processed image are detected. This can be carried out in parallel. The intermediate results are added together by means of a simple logical OR operation [102] that is given in Eq. (7.6). In order to remove all pairs of false features the operation of *False Feature Elimination* is applied. Furthermore, two new bifurcation and ending images are created by subtracting the false features from the images originating from the previous steps of bifurcation and ending detection. This is simply achieved by applying the operation of logical AND [102] that is given in Eq. (7.5). These new images are target of the final operation, *Figure Reconstruction*, where two instances of the operation are applied in parallel. The final result consists of two images containing the placement and direction of endings and bifurcations.

$$\mathcal{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \mathcal{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad i = -1 \tag{7.5}$$

$$\mathcal{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \mathcal{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad i = 1 \tag{7.6}$$

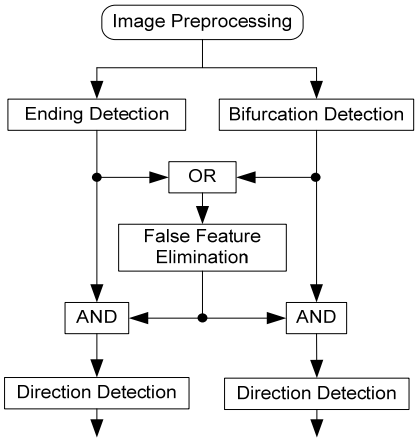


Figure 7.18 Block diagram of the vein feature extraction.

The end of a thinned line has only one black pixel within its neighbourhood. As all isolated pixels are already removed during the pre-processing, ending points are easily extracted by applying the template of *Ending Detection* (Eq. (7.7)) once. The input image u is the pre-processed picture, while initial output values, $y(0)$, are set to zero.

$$\mathcal{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \mathcal{B} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & -1 \end{bmatrix}, \qquad i = -7 \tag{7.7}$$

Similarly, *Bifurcation Detection* extracts all points that have at least 3 black pixels within the neighbourhood. Three different types of junctions do exist: "real" points, T- and Corner-forms (Figure 7.19). Extracting real bifurcations from the T- and Corner-forms needs further treatment. To do that, the approach introduced in [98] is employed. The template of *Junction Point Extraction* (Eq. (7.8)) that extracts the real junction points but keeps the T- and Corner-forms. Once again, the initial output values, $y(0)$, are set zero.

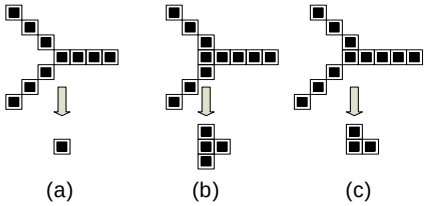


Figure 7.19 Different types of Junction Points: regular bifurcation (a), T-form (b) and Corner-form (c)

Junction points in T- and Corner-forms are extracted by means of the template given in Eq. (7.9), which removes all real bifurcations that have been detected using Eq. (7.8). The template of *Isolated Point Extraction* (Eq. (7.8)) is applied in parallel and the result is added to the outcome of Eq. (7.9) by means of a logical OR operation. Obviously, the initial output values equal zero here as well. The order of operation in the procedure of bifurcation detection is depicted in Figure 7.20.

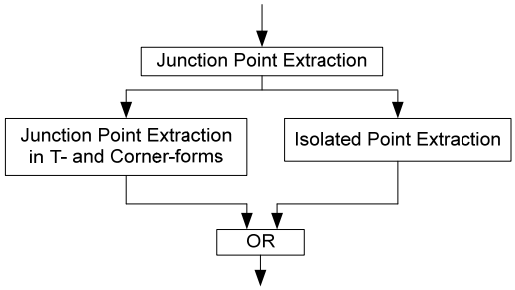


Figure 7.20 Bifurcation detection uses three different templates in addition to a Logic OR operation

$$\mathcal{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \mathcal{B} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 6 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad i = -3 \qquad (7.8)$$

$$\mathcal{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \mathcal{B} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad i = -3 \qquad (7.9)$$

$$\mathcal{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \mathcal{B} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & -1 \end{bmatrix}, \quad i = -8 \qquad (7.10)$$

In order to remove all false points that are separated by a distance $d \leq n$, it performs the dilation and erosion operations for n/2 iterations each. The dilation operation connects all features, with distance $d \leq n$ in between, together. As conventional erosion operation will bring the disconnected objects in the dilated image back to the original size, the erosion has to be applied in two diagonal directions. Thus the templates "Erosion \" and "Erosion /" are employed. The former, Erosion \, erodes all pixels inserted in the dilated image except those belonging to the centre of diagonal lines with direction "\". Erosion / works similarly for all diagonal lines with direction "/". The block diagram in Figure 7.21 shows the sequence of the different operations. The applied templates of Dilation, Erosion / and Erosion \ are given in equations (7.11), (7.12) and (7.13) respectively.

$$\mathcal{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \mathcal{B} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad i = 8 \qquad (7.11)$$

$$\mathcal{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \mathcal{B} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad i = -2 \qquad (7.12)$$

$$\mathcal{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \mathcal{B} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad i = -2 \qquad (7.13)$$

The fact that two false features are usually close to each other [98], implies the use of a low value of n. Actually, experiments show that n=2 is sufficient in our case. Thus, one iteration is enough for each of the operations, which explains all feedback templates being equal to zero. Consequently, the values of initial output are all set to zero as well.

So far, the extracted bifurcations and endings are represented as single points. Thus, only the location of every ending and bifurcation is obtained so far. In order to perform the procedure of Feature Matching, the direction of each feature needs to be known. The template of Figure Reconstruction (Eq. (7.14)) takes the original image as input and the intermediate image (with the extracted feature) as initial output in order to reconstruct the feature to the limit that makes it comparable. The number of iterations determines the number of pixels

that are restored of the three lines leaving a bifurcation and the only line leaving an ending.

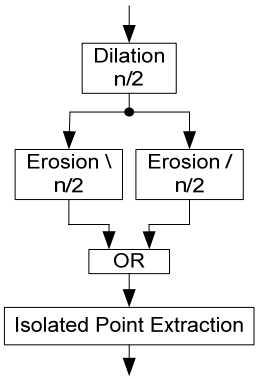
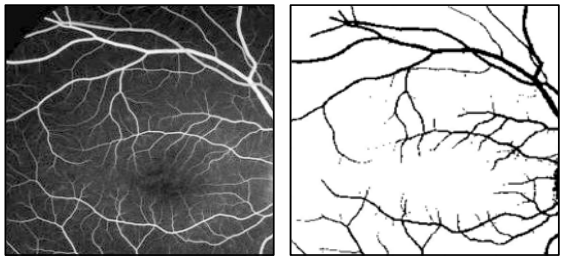


Figure 7.21 Operations involved in False Feature Elimination. Number of iterations, n/2, depends on the distance, n, between two false features.

$$\mathcal{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \mathcal{B} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \qquad i = 0 \tag{7.14}$$

7.3.3 Analysis and Verification

MATLAB provides an easy-to-use and feasible environment, on which verification of the aforementioned algorithm is carried out. We start with the image in Figure 7.22.a, where a pattern of veins is captured. Applying the first operation of pre-processing, i.e. *Binarization*, yields in a black and white image (Figure 7.22.b). The binary image serves as input to the sequence of *Skeletonization* templates (Table 7.2) that is applied iteratively 7 times to get the line-thinned image shown in Figure 7.23.a. As this image undergoes the operation of *Isolated Pixel Removal* (7.4), all unwanted isolated points are removed, which is depicted in Figure 7.23.b.



(a) (b)

Figure 7.22 Original image containing vein pattern (a) and a black and white image after binarization (b).

As the stage of pre-processing is accomplished, we move on to the first stage of feature extraction. Ending Detection produces the image shown in Figure 7.24.a, while Figure 7.24.b is obtained by means of Bifurcation Detection. The subsequent stages from eliminating false feature in the ORed image to the reconstruction of bifurcations and endings result in the images shown in Figure 7.25.

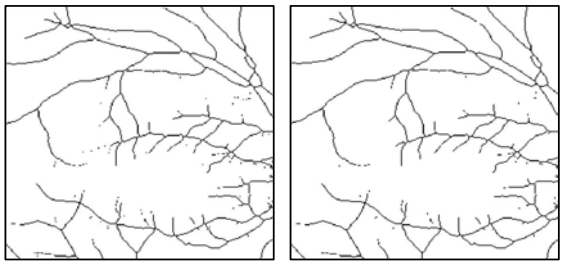


Figure 7.23 Result of skeletonization (a) and Isolated Pixel Removal (b)

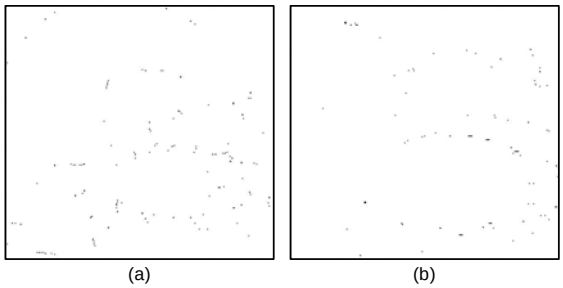


Figure 7.24 Endings (a) and bifurcations (b).

7.3.4 Experimental Set-Up

The project adopts the NoC-based architecture, Caballero. All templates introduced previously are preloaded in the BRAM internally in each node, which also serves as temporary storage for the intermediate outputs. Apparently, this is not feasible if the entire greyscale image (Figure 7.22) is used. The size of the image is 261×261 and will require about 67 kB of memory storage, which occupies almost 22% of the available on-chip memory (see section 6.2). Therefore, only a smaller portion of the original frame that equals the size of the network itself is used. As the feedback coefficients in all templates, except Eq. (7.14), equal zero, the contribution of y-values in the calculation of the nodal equation is removed. The accumulator is initialized with the value of the bias, whereas the subsequent control contributions performed on the multiplier are accumulated. The need of multiplication by 8 in Eq. (7.14) is resolved by a simple 3-bits left-shift. Thus, the computational stage of the nodal operation is brought down to 10 clock cycles only instead of 19 cycles originally.

Only 78 nodes are realized. Thus, 78 pairs of Multiplier/RAM out of 136 are used. The utilization of the logic shows to be 64% of the available slices, which opens for accommodating additional functionality. The design runs on a clock frequency of 100 MHz.

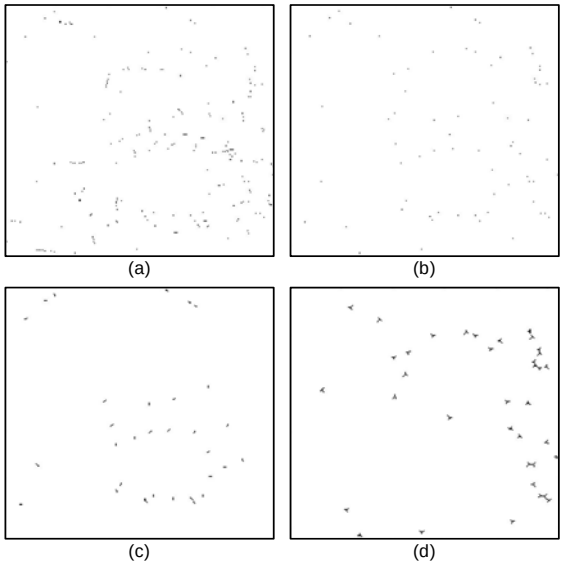


Figure 7.25 Adding the images with ending and bifurcation points by applying the operation of Logical OR (a) before eliminating the false features (b). Reconstruction of endings (c) and bifurcations (d).

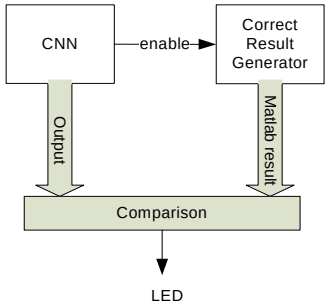


Figure 7.26 FPGA test set-up

The experiment is kept simple by removing the need for interaction with MATLAB. The original image and the MATLAB result are included in the programming file as BS-RAM content. Then the CNN will work on the image and the result is compared with the stored MATLAB result. If these results are in agreement, a LED on the Memec board is lighted (Figure 7.26).

7.4 DISCUSSION

In the previous sections a number of CNN-based experimental systems have been discussed to get guidance about the practical significance of the FPGA implementations that have been developed so far. The key questions are whether there is a need for different architectures and to which degree a generic architecture is helpful in creating such systems.

The Game-of-Life seems a first step in the direction of Artificial Life, but it is actually a very simple game. The accuracy requirements are not very high; the conventional game can even be played by a binary implementation. It then becomes questionable to go for a digital implementation: analogue will be faster and the gap with a software solution (using bit parallelism) is unusually small. In turn, a grey-level implementation may open new possibilities when available, and also closes the gap to predator/prey situations.

A careful look at Figure 7.7 tells that displacements captured with parts of the blob (object) outside the scenery (Figure 7.10) guarantees obtaining one solid box after the step of background suppression. On the other hand, separation between blobs occurs for all displacement values above maximum blob size. Furthermore, two leftovers are observed when one of the frames contains the whole object and the displacement is above the maximum size of the blob (Figure 7.27). The size of the obtained blobs is highly dependent on the speed of the object and the frame frequency f_f. In the extreme case, one of the ends of the object is seen in one frame while the other end only appears in the successive frame. This case is discarded, as no measurements are possible. The alternative is to operate so fast that co-location still exists between subsequent images.

Taking the difference between frames will not suffice, when there is separation between blobs. The separation between the blobs has to be bridged. This case can be taken care of in the algorithm by including more steps.

The templates for thickening only the right hand side (Eq. (7.15)) and left hand side (Eq. (7.16)) are applied on the first and second image frames respectively. The logical AND template (Eq. (7.5)) is applied to the resulting image frames to get an intermediate image frame. Now the operation is performed on the other sides i.e. the templates for thickening left hand side and right hand side are applied on the first and second frames respectively. The resulting image frames are again logical ANDed to get another intermediate image frame. These two intermediate frames are logical ORed (Eq. (7.6)). Hence, the image frame derived from this operation will be free of pixel separation between the two blobs. The value of Δ can be calculated by taking the difference between the separation-free frame and the first frame. An illustrative diagram of the different steps is shown in Figure 7.28.

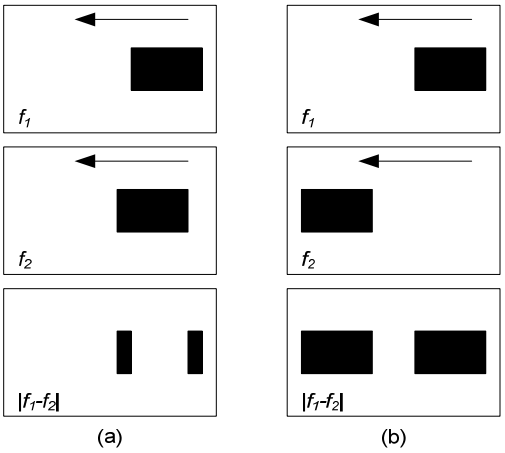


Figure 7.27 Separation between blobs due to different speeds: "slow" object in (a) and a "fast" one in (b). The arrows indicate the direction of the movement.

$$\mathcal{A} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \mathcal{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad i = 0 \tag{7.15}$$

$$\mathcal{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \mathcal{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad i = 0 \tag{7.16}$$

In the case of velocity measurements, hardware requirements differ with the objective. Initially a single template needs to be applied to the entire image to find objects of interest. This favours the pixel pipeline. For more complex operations such as the matching of moving objects, the focal plane approach is more effective. It seems appropriate to reconfigure the board, but preferably not for the total design. In other words, more product-level experience is required to make sensible decisions.

It has been illustrated that the algorithms described in [99][100] can also be used for vein identification. In comparison with the algorithm presented in [98], the operation of False Feature Elimination is applied only once instead of 3 times. Furthermore, the design is simplified by restricting the number of iterations to 1 for all used templates. This allows for comparison with solutions, based on layers of feed-forward networks [103]. Where this paper performs detection based on pre-learned physical features, here such features are pre-defined through the template application.

Pre-processing is achieved by applying the template of skeletonization that masks out the features of shape and thickness of the veins. The order in which the templates of skeletonization are applied influences the type, the number and the direction of extracted features. Figure 7.29 shows the output of bifurcation

detection as the templates in Table 7.2 are applied in the order: NW, N, NE, E, SE, S, SW and W.

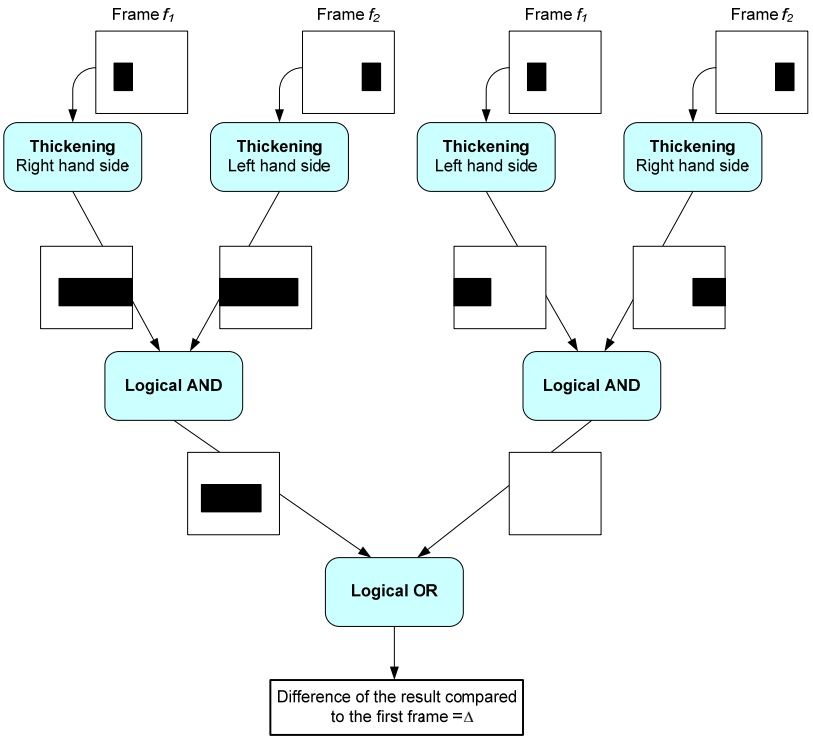


Figure 7.28 Extended algorithm for handling fast moving objects. The direction of movement is from right to left.

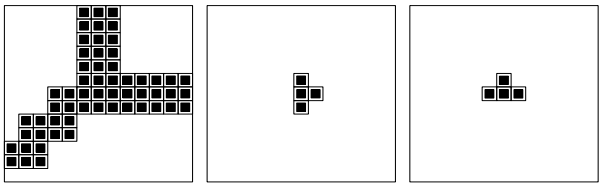


Figure 7.29 A certain order of skeletonization templates applied on (a), results in a false feature (b) instead of the real one (c).

The algorithm is restricted to 2-dimensional black and white images. This limitation increases unfortunately the rate of false detection, as vessels passing over each other in reality will be treated as a cross-section in the 2-dimensional image (Figure 7.30). The operation of *False Feature Elimination* is crucial for the accuracy of the overall algorithm, as the number of false features as well as

the total number of extracted features is affected. Unfortunately, the current algorithm proves to be sensitive for image resolution.

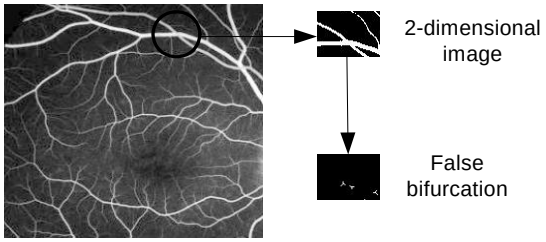


Figure 7.30 The non-crossing veins (marked with circle) give rise to false bifurcation in the 2-dimesional image.

The functionality of vein feature extraction approach is demonstrated on a human retinal angiographic image used in [97], as this enables comparison between the methods. As the algorithm in [97], a Gaussian model, does not distinguish between true junctions and corners (points of high curvature), many of the detected junctions are neither bifurcations nor end-points. It is reported that the Gaussian algorithm extracts 60 out of the actual 65 junctions in the original image. In addition, it has a false-positive rate of 80 junctions! A manual count shows the existence of 43 real bifurcations in the binarized image. The CNN-based approach presented here extracts 30 bifurcations of which 2 shows to be false features (Figure 7.25.d). This is caused by the 2-dimensional mapping that has already been discussed (Figure 7.30). By mean of comparison with the Gaussian extraction model, we focus on the rate of detection rather than the number of extracted junctions. Here, two rate numbers are of importance: the detection rate of 65% and the false-positive rate of 5%. The former is much lower, while the latter is a bit higher than in the Gaussian model. By removing the False Feature Elimination, the detection rate is raised to 100% but the false-positive detection rate is also increased. It appears therefore that this algorithm is not fully adequate, although it easily matches the Gaussian model (Table 7.3). The reason seems to be a lack in image resolution. As claimed in [103], this can be solved by re-introducing feedback to adapt the resolution before skeletonization is performed, similar to the bio-inspiration claimed in [104]. This confirms the setting of different block sizes in [97] and the variety of 2^{nd} layer networks in [103]. Another issue that needs more attention is the accuracy of the binarization procedure as a simple visual inspection reveals a difference in the number of bifurcations and ending between the original and the binarized images.

Feature matching of two vein patterns depends on the type, the location and the direction of endings and bifurcations. It is debatable, whether with all the existing variety it is really required to find all the existing bifurcations and endings to limit the images that need to be inspected on per pixel basis. A larger experiment seems required to decide how much is enough.

For realistic image databases, the many images require more pixels to be discriminated, leading to a more than quadratic increasing search time. One may

conclude that image comparison is accurate [94], given a repeatable capture mechanism [95], but the number of images to be compared is simply too large. One way to ease the problem is by providing a content-based selection mechanism. The automatic provision of such 'features' allows determining the small number of images to search through. For this purpose the reduction of False Acceptance Rate (FAR) will be dominant.

Table 7.3 A comparison of the Gaussian model and the CNN-based approach when applied on a human retinal image. FFE stands for False Feature Extraction.

		Detected junctions	**False-positive**	**False-negative**
Gaussian Model [97]		92%	123%	8%
CNN-based approach	With FFE	65%	4.7%	35%
	Without FFE	100%	21%	0%

The implementation stresses the exploitation of the FPGA as realization target. We have aimed at the best detection using few resources, as a realistic product will be based on bi-spectral imaging. The merging of features from two sources will definitively raise the performance figures but poses additional computational demands. Hence, the computational need of the feedback contribution is removed. This provides for a good starting position to extend the hardware with variable resolution and 3-dimensional modelling.

In all, generic VHDL descriptions are feasible in the sense that parameters can be scaled. But experience shows that the diversity is even larger and a deliberate customisation of the original generic set-up will be inescapable. For instance, the Game-of-Life can be implemented at least 6 times denser than the Sleipner template on the target FPGA as only one pipeline is required. Consequently the designs are derived from the same base, but made unique in implementation.

Chapter 8

Template Optimization

The functionality of both continuous- and discrete-time CNNs is defined by the cloning template $\mathcal{T}$ that, together with the input pattern u and an initial output pattern $y(0)$, completely determines the dynamic behavior of the system. In a 1-neighbourhood, a template consists of 19 free parameters, while 51 free parameters constitute the template in a 2-neighbourhood. Any deviation in template parameters will have a tangible effect on the dynamic behaviour and may lead to malfunctioning. Thus, any template design method has to guarantee the robustness of the CNN. A CNN is said to be robust if it operates as desired even when subjected to implementation inaccuracies [61]. More attention has to be paid to parameter deviation when designing templates for analogue CNN chips. Template robustness is easily disturbed due to the noise in the electrical components as well as parameter scattering introduced during the fabrication process [80].

The simplicity of CNN operation, given in Equations (2.33) and (2.34) is deceptive. The dynamics can be extremely complex, even for relatively small networks [11]. The feedback coefficients, i.e. template $\mathcal{A}$, give rise to a non-linear dynamic behavior that leads to the existence of different interesting phenomena such as oscillation and chaos. Most CNN applications require, however, complete stability and strive to eliminate the chaotic tendency instead. In his book [11], Chua presents the mathematical criteria that guarantee such complete stability, which is rephrased below.

Major parts of this chapter are published in [X].

Theorem 7.1: Complete Stability Criterion

For a standard CNN with constant inputs, constant bias and an arbitrary neighbourhood, all trajectories converge to an equilibrium state, which in general depends on the initial states, if the following three conditions are fulfilled:

- The feedback template, $\mathcal{A}$, is symmetric with respect to the center of the template
- The squashing function f(·) is differentiable with positive slopes, and bounded.
- All equilibrium points are isolated, i.e. there exists an open set around any equilibrium point that contains no other equilibrium point. ■

The piece-wise function in Eq. (2.7) does not fulfil condition (ii) above, but can be approximated by an injective function that does fulfil it. Furthermore, the three conditions are sufficient but not necessary. For instance, many CNNs with non-symmetric templates are completely stable [11].

By increasing the slope of the piece-wise function in Eq. (2.7) such that it approaches infinity, a step function is obtained. This leads to two distinct output values for each cell, +1 or -1. The cells are then *bistable*. Below, we reintroduce the bistability theorem as stated in [11].

Theorem 7.2: Bistability Criterion

The output of every cell at any stable equilibrium point of a completely stable standard CNN is equal to either +1 or -1, if the centre element of the $\mathcal{A}$ template satisfies $a_{00} > 1$. ■

8.1 DESIGN OF ROBUST TEMPLATES

Different design methods can be applied to find the desired template [62]. The most difficult one is design by intuition as it requires a long experience in the applicable field. For the experienced designer this method is fast, but it does not guarantee a satisfactory result. The second method is design by learning, where classical neural network training techniques are employed. Both local and global learning algorithms have been tried [63][64], where the idea is to design the desired template by gradual enhancement of the robustness. The problem is that for some application the template either exists or does not exist. The gradual enhancement approach is then not possible and the template may never be found! The third method requires the desired function to be exactly determined; this is the direct template design. According to [62], the popularity of the method lies in the fact that it finds a template class rather a single or few working templates. The obtained template class is guaranteed to contain the most robust template, while time and computational power needed are much smaller than in the two other methods.

In his strive to give a practical survey of template design in bipolar CNNs, i.e. $y(t) = sign(x(t))$, Zarándy [62] divides CNN templates from the interconnection point of view into two sets: uncoupled and coupled. In the uncoupled templates, a cell is not at all affected by the current output of the neighbouring cells but only of the input pattern, while coupled templates takes the contribution of neighbouring cells' output values into account. In other words, all the entries in the $\mathcal{A}$ matrix of an uncoupled template are set to zero apart from the self-feedback coefficient a_{00}. Table 8.1 illustrates the idea for a 1-neighborhood.

Table 8.1 Feedback matrix $\mathcal{A}$ in coupled and uncoupled CNN templates.

Uncoupled Template	Coupled Template
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & a_{00} & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} a_{-1-1} & a_{-10} & a_{-1+1} \\ a_{0-1} & a_{00} & a_{0+1} \\ a_{+1-1} & a_{+10} & a_{+1+1} \end{bmatrix}$

The simplicity of uncoupled templates lies in the fact that analyzing the dynamic behaviour of a single cell is enough to understand the functionality of the whole network. The state equation of a single cell is merely a simplification of Eq. (2.33), as shown in Eq. (8.1) below.

$$\dot{x}(t) = -x(t) + a_{00}y(t) + w \qquad (8.1)$$
$$where\ w = \sum_{kl \in S_r(ij)} b_{ij}u_{kl} + I$$

The robustness of uncoupled templates is mainly dependent on the value of a_{00}. For a zero valued self-feedback coefficient, the final output is independent of the initial state and depends only on the contribution of the input. On the other hand, when $a_{00} = 1$, the CNN acts as an integrator with the state in the linear region ($|x| < 1$). Here two options are possible: (i) $w \neq 0$, the final output is then only binary, i.e. $y(\infty) = sign(w)$ and (ii) $w = 0$, the final output depends on the initial state, i.e. $y(\infty) = x(0)$. In a third case, $a_{00} > 1$, the final output is always binary regardless of the initial state and the contribution of control template (Eq.(8.2)).

$$y(\infty) = sign((a_{00} - 1) \cdot x(0) + w) \qquad (8.2)$$

Similarly, the dynamics of coupled templates are described in Eq. (8.3). In line with Theorem 2, the self-feedback coefficient a_{00} is assumed to always be larger than 1. The contribution of neighbouring cells in the feedback loop gives coupled templates their characteristic feature, i.e. propagation phenomenon. Propagation occurs due to the interaction between active and inactive cells. An active cell is always in the linear region of the activation function, i.e. the output of an active cell changes over time. An inactive cell remains in the saturation region unless it is explicitly activated. At the beginning of propagation, some cells are active. These cells might activate neighbouring inactive cells, a procedure that continues for a while. At the end of the operation all the cells are inactive.

$$\dot{x}(t) = -x(t) + w_{coupl} \tag{8.3}$$

where $w_{coupl} = \sum_{kl \in S_r(ij)} a_{ij} y_{kl} + \sum_{kl \in S_r(ij)} b_{ij} u_{kl} + I$

Equation (8.3) reveals that stability of an output is only possible in the saturation region. For instance, if $x(t) = +1$ and $w_{coupl} > +1$ then the stable equilibrium point $x = w_{coupl}$ is in the positive saturation region. The state x reaches equilibrium without changing the output y. On the other hand, if $x(t) = +1$ and $w_{coupl} < +1$ then the current output resides in the positive saturation region, but it moves gradually toward the linear region as y becomes smaller than +1, and hence w_{coupl} decreases. This is due to the decreasing value of term $a_{00}y$. The positive feedback brings the state for the cell to the negative saturation region. Similar discussion is valid when the state initially resides in the negative saturation region.

Now the design of the robust template can begin. Zarándy gives in [62] a detailed description of the entire procedure (Figure 8.1) accompanied with a number of examples. We will reintroduce the methodology briefly.

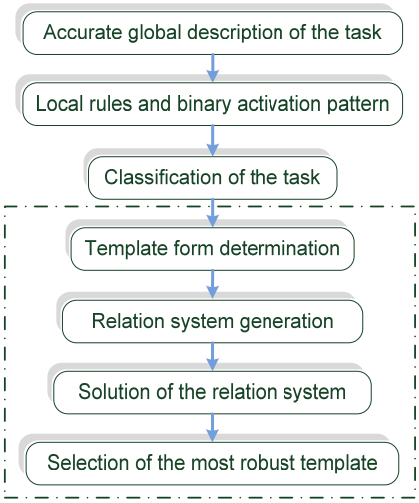


Figure 8.1 Flowchart of the design steps of coupled templates. The dashed box marks the steps of uncoupled templates [62].

In the first step, the global task is described verbally with some input-output pairs. The global description eases the derivation of local rules (on the pixel level) of the propagation. A local rule prescribes if the output of a bipolar cell has to remain constant or to change sign, depending on the input and output values of the neighbouring cells [65]. Furthermore, a 3 x 3 binary activation pattern is generated against which a matching of the input image is performed. This completes the second step. The classification step determines whether the propagation is *constrained*/*unconstrained* and *symmetric/asymmetric*, which helps to perform the next step. The propagation is symmetric if the activation

condition is symmetric to the sign of the cell, i.e. in image processing applications, black and white cells are affected in a similar (but opposite) way. If the input contains a mask that limits the propagation, the template is constrained (e.g. hole filling). In an unconstrained template $\mathcal{B} = 0$, while a symmetric template has $I = 0$. Now, the number of free parameters is known and the template form can be determined. In this step, the aim is to reduce the free parameters in the searched template as much as possible, e.g. from 11 down to 3-4 parameters in the uncoupled templates. This is crucial to successfully design a robust template. Depending on the task and with help of the previous discussion about Eq. (8.2) and Eq. (8.3) a number of inequality relations are generated. This is simple since input-output pairs are already known. Each relation defines a hyperplane that divides the template space into two halves, where the relation is satisfied in one half and not in the other. The intersection of all satisfying halves gives the subspace in which all correct templates are found. See Figure 8.2 for an illustrative description. What remains is only to select the most robust template that resides in the centre of the specified template subspace. In the case of uncoupled templates, the design procedure consists only of the last four steps in Figure 8.1.

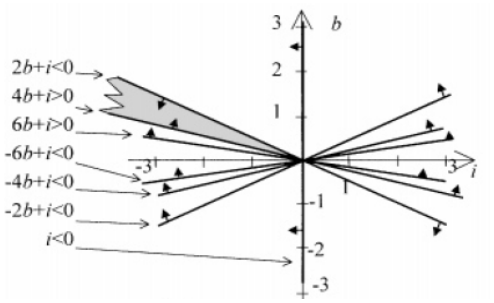


Figure 8.2 Graphical example of the Solution of the Relation System step. Here, only two free parameters, b and i, are involved. The arrows indicate in which half of the space a relation (the line) are satisfied [62].

A rule of thumb in template design is that the larger the template values are the faster the transient is (it reaches stability faster). But the analogue realization of the CNN limits the maximal absolute value to 3 for template coefficients and 6 for the bias [62], which bounds the inifite subspace obtained in Figure 8.2. In another modularly extendable g_m-C implementation of the CNN-UM [66], the maximal absolute values are a bit larger but have more strict precision characteristics (Table 8.2). Thus, the values of control template $\mathcal{B}$ and bias i in Figure 8.2 are chosen from the middle of the subspace (dashed lines in Figure 8.3). This yields the so-called nominal template. Due to parameter deviations in the analogue realization, each cell is considered to have its own template. These real templates are located in a circle around the nominal template, but the circle should be completely within the obtained subspace.

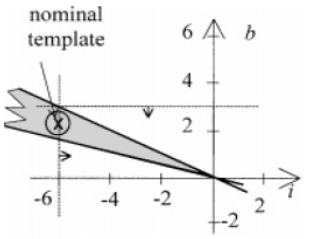


Figure 8.3 The nominal template is the origin of a circle containing all real templates. Dashed lines mark the technical limitation of the employed analogue CNN chip.

Table 8.2 Range of template values in the g_m-C implementation of the CNN-UM [67].

Parameter	Interval
$\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$ off-center entries	$\pm\{0, 1, 2, 3, 4\}$
$\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$ center entries	$\pm\{0, 0.5, 1, 1.5, 2, \dots, 4\}$
Bias	$\pm\{0, 0.5, 1, 1.5, 2, \dots, 7.5\}$

8.2 CHIP-INDEPENDENT TEMPLATE OPTIMIZATION

Analogue CNN-UM chips, such as ACE4k [27] and ACE16k [29] perform image-processing tasks with extremely high throughput data rates in the order of tera operations per second. However, non-ideal functionality of the chip may occur due to erroneous behaviour in some cells. These analogue implementations can guarantee only a rough accuracy of 5%-10% in relation to ideal parameter values [68][67]. Additionally, template parameters have a discrete range of implementable values of about 7 bits for the actual chips. Having this in mind, fabrication imperfections may lead to undesirable and unavoidable parameter variations. Other sources of erroneous behaviour are: noise in the electrical components of the cells, imperfect or noisy loading of the input and initial state from off-chip to on-chip memory, and temperature variations. Beside, saturation at exactly ± 1 cannot be guaranteed which affects the characteristics of the piece-wise function and, hence, leads to erroneous behavior. The best way to overcome these restrictions is by adjusting template parameters, which makes the CNN more tolerant against inherent chip-parameter deviations and noise in the analogue implementation. At first, chip-independent template design methods have been developed, as they are thought to generate templates robust enough to be employed on all chips. We discuss two methods: one is targeting continuous-time CNN implementations and the second covers discrete-time CNN models.

8.2.1 Discrete-time implementations

In the finite iteration DT-CNN approach [69], the network performs a fixed number of iterations. The standard feature of convergence is not taken into account and the achievement of a steady state is not required. The main feature is that different templates are used for different iterations; the DT-CNN is then called *non-stationary*. In this way all the demands on symmetry and stability,

discussed in Theorem 1 and Theorem 2, are removed. Each cell in the network will have a set of states according to the applied number of iterations. In line with the standard CNN, the state corresponding to the current template is tightly coupled to the previous state through the feedback of previous output. For iterations $t = 0, \dots, T-1$ the states evolve according to Eq.(8.4). R stands for the number of rows in $\mathcal{A}$ and $\mathcal{B}$ matrices.

$$x_{ij}^{t+1} = \sum_{k,l=-\frac{R-1}{2}}^{\frac{R-1}{2}} \left(A_{k,l}^{t} y_{i+k,j+l}^{t} + B_{k,l}^{t} u_{i+k,j+l}^{t} \right) + i^{t} \tag{8.4}$$

The template is designed through a constructive learning strategy based on back-propagation. Input and output pairs (y_j, u_j) are presented to the network upon which the template weights are adjusted. The method is concerned with using the DT-CNN as a classifier. The training process aims on minimizing the training error that depends on the training data (input/output pairs) and the template. For each template, a certain absolute loss function simply measures the deviation of the actual CNN output from the desired output. An ϵ-insensitive absolute loss function [71] is used due to the higher accuracy of resulting derivatives. The actual output has zero loss and zero gradient if it lies inside the ϵ-margin of the desired output. In this way, the algorithm learns misclassified training patterns rather than adjusting the weights with gradient steps of already correctly classified patterns

For verification purposes, the system is used to classify normalized handwritten digits, i.e. recognition of digits $0, 1, 2, \dots, 9$, where 16 x 16 grayscale images have served as inputs. Classification rates increased from 80% for a single template to around 97% for 10 templates. In a later work [70], the effects of finite-word length have been studied, where the design of templates with limited precision is demonstrated. Input images are represented as double-precision floating-point values with an effective precision of eight bits (covering 256 greyscale levels). The template weights by the training algorithm are then reduced to two decimal digits (at least 8 bits). The algorithm is adjusted to the limited precision through truncation of the weights after each training epoch. The final weights fall into the interval [-2,+2] and are 9 bits wide; 8 bits for the value and 1 sign bit. In both [69] and [70], the simulation is confined to $K = 5$ only, i.e. 2-neighborhood, as adequate classification performance cannot be achieved for $K = 3$, while $K = 7$ leads to overfitting. It is interesting that both stationary and non-stationary DT-CNNs lead to the same classification rate!

8.2.2 Continuous-time implementations

The proposed analytical method in [61] and [67] considers robustness with respect to template errors only. Approximation errors of the piece-wise function are taken into account by choosing sufficiently large fractional errors for the template parameters; larger than the errors that the parameters actually experience. The analysis are further restricted to the case of binary initial state and input values.

The approach provides robust templates for accomplishing different tasks, e.g. horizontal line detection and shadowing, but the obtained template parameters are derived through an intensive mathematical analysis of the proposed solution for each of the tasks. Additionally, for tasks requiring a high degree of connectivity, e.g. edge detection, the approach leads to relatively large template values. The CNN system has then to be able to accommodate larger template values than needed in other templates. One way to remedy this is to use other algorithms to permit template values remaining in a small range, which adds to the already high complexity of the approach! Furthermore, the proposed analytical approach guarantees template robustness only for so-called locally regular CNNs. Locally regular CNNs are a subclass of bipolar CNNs [65]. They contain two subgroups: uncoupled and a subclass of the coupled CNNs, i.e. propagation-type CNNs. The template set consists of not more than 11 non-zero coefficients; otherwise the template will not function properly. One of the strong features of the proposed method is that the optimal robust template is obtained directly without any need for iterative enhancement.

8.3 CHIP-SPECIFIC TEMPLATE DESIGN

The main drawback in chip-independent optimization methods is that even if some templates show to be robust enough, the degree of robustness for different operations is obviously not the same. Actually, noise of the electrical components as well as parameter scattering introduced during the fabrication process, lead to space-variant differences in template parameters from the ideal values. There is no guarantee that two cells within a single chip will react identically on the same stimuli, even when the most robust template is used! If the tolerance range of the template is smaller than the inherent parameter deviation of a given chip, the template works improperly. Consequently, the functionality of a robust template is not guaranteed on different chips of the same type. In this case improvements can only be achieved by readjusting the templates [80]. Errors of actual chips are then eliminated or at least minimized for a certain operation. The ultimate solution in this case would be to manually and empirically tune the template for a given chip, which should be avoided as it tend to be tedious and require very long time. An automatic approach that tunes the template parameter for a specific chip is needed. Two adaptive template optimization methods are presented in [80][68], where the template is not *redesigned* but *optimized*. In the following subsections we describe these two methods briefly.

8.3.1 LMS-based approach

The approach starts with the theoretically most robust template for the given task and ends with the *optimal* template for the given chip. Thus, the obtained templates are *optimal* but not necessarily the most *robust* ones for the given chip and task

The method assumes uncoupled CNNs with space-invariant templates and binary output, where the self-feedback entry is always larger than 1 to guarantee stability (Theorem 1). Both binary and analogue inputs can be handled. For

simplicity, the initial state of all cells is set to zero, i.e. $x_{ij}(0) = 0$. Then, any task is defined by the control vector $\boldsymbol{B}$ (corresponding to control $\mathcal{B}$ template) and the bias z.

The optimization aims on eliminating, or at least minimizing, the average error of the entire CNN, denoted $E(\boldsymbol{B}, z)$, when presented an input vector $\boldsymbol{u}$. Usually, the average error is obtained by calculating the normalized sum of the mean square errors of all cells in the grid. Then the gradient descent approach is employed to minimize the error and thus find the optimum solution on the error surface. The main hindrance is that the exact form of $E(\boldsymbol{B}, z)$ is not known for a given chip. This makes the attempt to find a solution analytically very difficult. Instead, the proposed method is based on a cumulative single-cell model of the chip for which an optimum is found through iterative optimization. The gradient descent approach is still employed, but the average (or cumulative) response of the entire chip to input $u_k \in \boldsymbol{u}$ is obtained according Eq. (8.5) instead. N is the number of rows and M is the number of columns of the CNN, while $\bar{y}_k$ represents the desired output value.

$$\breve{y} \equiv \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} \bar{y}_{k,ij} \tag{8.5}$$

Accordingly, the average error of the entire chip is given by Eq.(8.6), where K is the number of input vectors, i.e. $K = 2^9$ for 1-neighborhood and $K = 2^{25}$ for 2-neibhborhood.

$$\breve{E}(\boldsymbol{B}, z) = \frac{1}{K} \sum_{k=1}^{K} (y_k - \breve{y}_k)^2 \tag{8.6}$$

The cumulative output carries information about the number of erroneous cells rather than about their exact location. This is of course desirable as the changes in template parameters should not be cell dependent due to the space-invariant nature of the template. The minimum on the error surface is found by means of gradient descent using Least Mean Square (LMS) learning. Furthermore, a piece-wise linear output function is chosen to model the noisiness instead of the sharp threshold function. The linear model is advantageous due to its simplicity, but for more accurate results, using the sigmoid function should be considered [68].

Figure 8.4 illustrates the template optimization set-up. The ideal values $(\mathbf{B}^*, \mathrm{z}^*)$ are used to initialize the CNN-UM chip and are used by the simulator to produce the desired values. New template values are adapted gradually as they are calculated by the LMS component.

There is no guarantee that the global minimum will be found since the LMS method will find the local minima on the error surface. Actually, the optimization method may fail in finding any template for which $E(\boldsymbol{B}, z) = 0$. In this case, the template is decomposed into more robust child templates through iterative replacement of nonzero template coefficients by zeros. It is well-known that increasing the number of zero-valued entries enhances the robustness [72].

A logical combination of all child templates yields an expression that is functionally equivalent to the original non-robust template. The conclusion is that templates having a robustness lower than approximately 0.5 require decomposition. The approach, illustrated in Figure 8.5, is fully automated in the sense that it stops nt before all obtained child templates are robust enough.

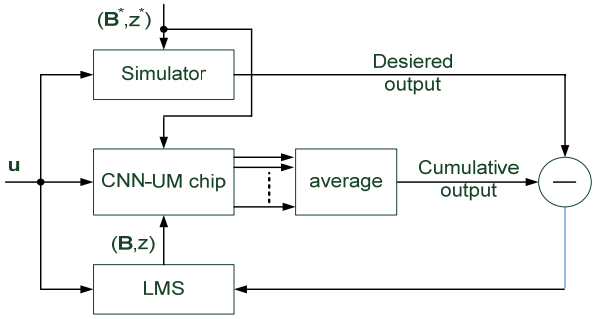


Figure 8.4 Template optimization set-up [68].

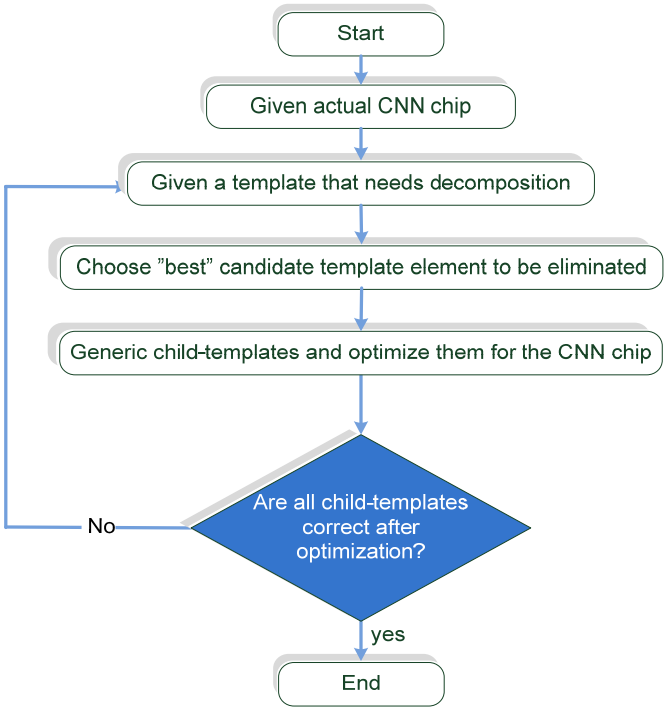


Figure 8.5 Block diagram of fault-tolerant template decomposition [68].

8.3.2 ASA-based approach

The main critique to the previous method is that the local optimization approach needs information on the gradient of the cost function, which leads to a poor minimum and forces to decomposition into simpler templates. Applying a global optimization method such as the adaptive simulated annealing algorithm (ASA) [73] overcomes this limitation and enables template optimization for coupled CNNs as well. This brings the most interesting feature of CNNs, i.e. global interaction, into play, which compensates for the main disadvantage of the ASA algorithm of being slower than local optimization methods. In [68], the ASA algorithm constitutes the cornerstone in the proposed template optimization method. The method is composed of two steps: in the first step the optimal tuning of the nominal template is found, while the second step aims on finding the *robust* optimum starting from the tuned template. The target is, as in the previously discussed method [80], the two analogue chips ACE4k and ACE16k.

For a given *training set* consisting of input $\boldsymbol{u}$, initial state $\boldsymbol{x}$ and desired output $\bar{y}$ (all values belonging to range [0,1]), the cost function of certain, randomly generated, template parameter vector $\boldsymbol{p} = (p_1, p_2, \ldots, p_D)$ is given by Eq. (8.7) where k is the number of cells in the chip and $y_i(\infty)$ is the value of the steady-state output of the ith cell. As the objective of the approach is tuning, not learning, an initial approximation p_{init} is imposed to set the boundaries of the search (these initial parameters compose the initially proposed template for the given operation under assumption that this template is fully correct on a simulator). The allowed search boundaries should be larger than parameter deviations of the chip. The search space can be minimized, which makes the approach faster, by applying additional constraints on the template, such as symmetry or dependence between the values.

$$g(p, u, x, \bar{y}) = \frac{1}{\sqrt{k}} \sqrt{\sum_{i=1}^{k} (\bar{y}_i - y_i(\infty))^2} \tag{8.7}$$

The ASA algorithm is performed recursively where the constraints are gradually removed and the boundaries are made narrower between subsequent iterations. The procedure is considered successful and stopped when the cost function becomes smaller than a certain tolerance value, otherwise the enforced constraints are relaxed to the next level and another ASA optimization is initiated. The result of this new optimization round is considered to be optimal. In other words, the wider boundaries and the harder constraints upon start allows for a fast and rough localization of a global optimum that is iteratively refined.

In the LMS-based approach, the optimization is concerned only with minimizing the error, while here robust improvement is involved as well. As discussed earlier (Figure 8.3), robust templates have their parameters in the middle of a correct operation interval. Similarly, the tuned template of a given template resides in the middle of an interval that is shifted together with the nominal values due to parameter deviations. It is clear that a more robust version

of the tuned template may be found only in this interval. To do that, a method similar to statistical circuit design is employed. A Gaussian noise vector of random variables with zero mean and small variance, denoted $\boldsymbol{e} = (e_1, e_2, \ldots, e_D)$, represents chip parameter deviations. The cost function now contains several different embedded measurements instead of only one (Eq. (8.8)). The parameter r denotes the number of runs executed.

$$G(p, u, x, \bar{y}, \mathrm{e}) = \frac{1}{r}\sum_{j=1}^{r} g\left(p + e_j, u, x, \bar{y}\right) \tag{8.8}$$

The optimizations are performed using the Aladdin system in connection with a MATLAB environment where the main features of the ASA algorithm are run. Target chips are: ACE4k of size 64×64 cells, and ACE16k of size 128×128 cells. The input and initial state for each optimized template operation have been generated randomly, except for few operations such as binary edge detection. The desired output values are obtained from simulators of ideal CNN-UM using robust templates. No slicing of the images is needed as those are always chosen to have the same size as the given chip. Each optimization round requires in average tens of thousands iterations, where each iteration takes about 50 ms. Performance bottleneck resides on the execution of the MATLAB part. Be aware that the obtained templates are robust and/or optimal for the given chip only. For use in a different chip, a proper repetition of the whole procedure must be performed.

8.4 Optimization of Digtial Implementations

The noise in digital circuits is primarily of numerical origin, where the finite word width causes a degree of value crisping that gradually impacts the intended behaviour [60]. The main difference with respect to analogue realizations is that the word width is a design aspect rather than a chip parameter aberration. For instance, internal values in [74] use just a single bit, while most applications need 8 bits and only few functions take considerably more than 8 [75][76]. Word size translates immediately into system size. Computational precision that is theoretically unlimited becomes rapidly a design bottleneck as the digital circuits grow rapidly and dissipate more power with larger words.

In his attempt to improve the design of a DT-CNN and lift it to a higher and more formal level, ter Brugge [40] has used the rules of Mathematical Morphology to systematically derive complex templates. Actually, the approach flows in the opposite direction of what other developers and researchers try to achieve; simpler templates! This results in a highly efficient CNN architecture as it simplifies the overall structure but it introduces a number of complications when a digital realization is considered. One has to do with the increasing internal word width. An 8-bit wide data word will lead to 21-bits wide internal word (Figure 8.6), which takes an appreciable amount of processing capacity away. Obviously, the problem is even more severe when the number of multiplicative additions increases, i.e. when a larger neighbourhood is employed. However, an iterative optimization approach presented by Fang et al.

[60] shows that the internal word width can be reduced down as low as 7 bits with no effect on the overall functionality. In this way, employment of composed complex templates can be accomplished with limited impact on the hardware implementation.

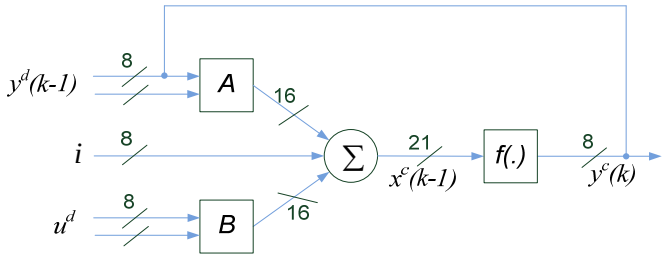


Figure 8.6 Block diagram of a single DT-CNN cell. The numbers represent the width of each line in a 1-neighborhood digital implementation.

The effectiveness of the approach is demonstrated by applying the template of hole filling (Table 8.4) on a small black and white image (Figure 8.7). In line with the digital implementations introduced previously, a fixed-point representation is assumed for all values, internally and externally. Table 8.3 shows the placement of the decimal comma for the different values. The proposed approach reduces the internal values (after the multiplication) systematically from 16 to, at least, 7 bits. In this way, word size of the state is brought down to 12 bits only, of which 7 bits are used to address a table representing the final discrimination.

Table 8.3 Typical data representation of a digital DT-CNN. The notation <n:m> means that the number consists of n-bits integer part and m-bits fractional part.

Value	Fixed-point notation
$\boldsymbol{u}$ -value and $\boldsymbol{y}$ -value	<1:7>
$\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{B}}$ coefficients	<4:4>
Bias	<5:3>
Multiplication results	<5:11>
State	<10:11>

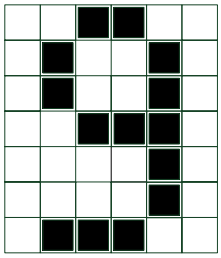


Figure 8.7 Input image used in template optimization algorithm.

Upon start the range, in which the search is performed, has to be defined. This is heavily dependent on the number of free parameters that the initial template has. For instance, the template shown in Table 8.4 that is usually used to perform hole filling operation has 4 degrees of freedom only. These are denoted a_1 for the center $\mathcal{A}$-coefficient, a_2 for the non-zero off-center $\mathcal{A}$-coefficients, b for the centre $\mathcal{B}$-coefficient and i for the bias.

Table 8.4 Hole filling template.

$\mathcal{A}$ Template	$\mathcal{B}$ Template	bias
$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	-1

In [60] the search of robust templates is ruled by the tuning ranges given in Table 8.5. Obviously, the range a_2 is not symmetrically spread around the nominal value of 3, while the value of 4 is not at all included in the adjustement range of b! These tuning ranges are, however, employed here as well to make the comparison with the approach in [60] possible. The number of robust templates is dependent on the performed operation, word size and the employed representation of values. For the hole filling template, and with the value representation given in Table 8.3 , the smallest step between two consecutive values of $\mathcal{A}$- or $\mathcal{B}$-entries is 0.0625, while it is 0.125 for the bias. In all cases, 11 levels for each of the parameters do exist, which results in a total number of 11^4 = 14 641 templates to be tested.

Table 8.5 Tuning ranges for the Hole filling template.

	a_1	a_2	b	i
Decimal	[2.625, 3.25]	[1, 1.625]	[3.125, 3.75]	[-1.375, -0.125]
Binary	[0010.1010, 0011.0100]	[0001.0000, 0001.1010]	[0011.0010, 0011.1100]	[11110.101, 11111.111]

The method evolves as follows. At the beginning, the system runs on full precision without any truncation (16 bits for the multiplication results with notation <5:11>). All templates obtained for the given tuning ranges are tested, where each template executes until one of the stopping conditions is fulfilled: equilibrium is established or a predefined upper bound of iteration count is reached. If the obtained output matches exactly a desired output, the template is considered robust for the tested truncation level. The set of robust templates is then tested on the next truncation level, i.e. <5:10> for multiplication results. In other words, the robust templates from a certain optimization round are tested in the successor round where additional bit of the decimal part is truncated. This continues until truncation level <5:2> is reached. The methodology is schematically illustrated in Figure 8.8.

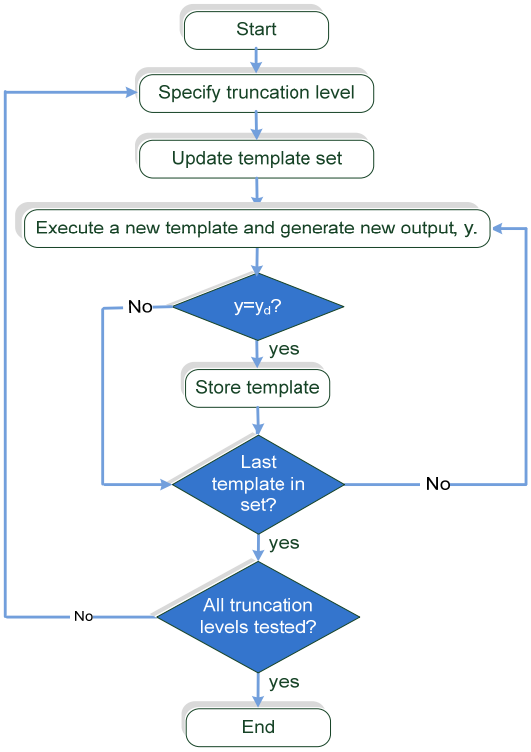


Figure 8.8 Template optimization through truncation.

The algorithm is executed on a CNN model that is first built in software, before it is digitally emulated on an FPGA. The software model is written in pure MATLAB functions, while the digital emulator is implemented using VHDL. Due to hardware limitations on the FPGA, mainly memory space, the algorithm is slightly modified. Instead of checking the entire set of templates for robustness on one truncation level as in Figure 8.8, one template is tested with all truncation levels for which the template shows to be robust before the next template is tested. If a template passes through all truncation levels and still provides the desired output, it is then considered fully robust. Both software and hardware models find the same final set of robust templates consisting of 466 templates, i.e. about 3.2% of the entire template space. An interest observation is that changing the order of optimization, i.e. starting from heavily truncated values <5:2> and gradually ending in full precision <5:11>, still provides the same robust templates. For clarity of the following discussion, the approaches are here named as *descending* and *ascending*. The former employs the intuitive understanding of truncation and starts from higher precision, i.e. <5.11>, and ends with lowest precision, i.e. <5:2>, while the latter follows the opposite order. While both approaches find the same 466 robust templates, they differ, however, greatly in the number of robust templates found in the intermediate

optimization steps (Figure 8.9). It is further observed that far more iterations per template are generally required in the ascending approach. Recall that the number of iterations toward convergence is mainly application dependent. Some applications, e.g. the Logical NOT, require a single iteration, independent of the network size. Other applications, e.g. hole filling, require more iterations dependent on the number of rows/columns in the network. However, the combination of template coefficients is desicive for how soon convergence is reached, i.e. how many iterations are required before the network converges. Thus, it is not surprising that the obtained robust templates differ in iteration count, making them less or more 'suitable' to perform the operation. On average, the ascending approach requires 26.3991 iterations for the robust templates in the last optimization level while the descending approach requires 12.5601 iterations. This advantage of the descending approach disappears quickly when other aspects are taken into account as will be discussed in section 8.7.

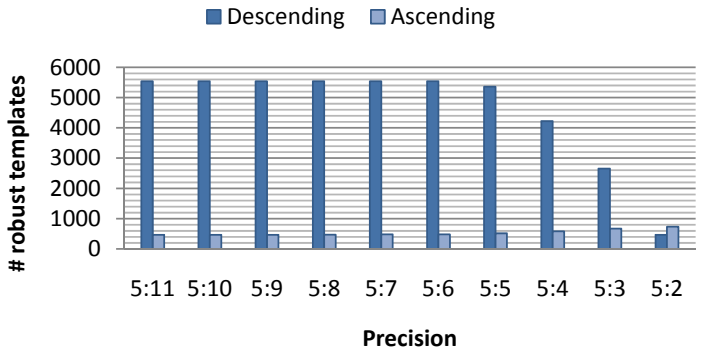


Figure 8.9 The number of robust templates remains unchanged at the beginning of the descending approach before it decreases strongly at then end. In the ascending approach, the number of robust templates is already very low and decreases slightly until it reaches the same value as for the descending approach.

8.5 INFLUENCE OF BOUNDARY CONDITIONS

The effect of boundary conditions on the quality of the applied approach requires special attention. It is well known that the choice of boundary condition is essential for the overall functionality of most templates. Mainly three conditions are used in literature: zero-flux, fixed (with different values) and periodic. Which one to use is operation dependent and is, thus, coupled to the derived template. In [77], the influence of boundary conditions on space-invariant coupled templates is discussed in detail. Only 1-neighborhood templates with zero $\mathcal{B}$-coefficients and bias are considered. The width of the added frame of boundary cells is equal to the neighborhood size. It is here proved that for any two-dimensional infinite CNN that has a stable equilibrium

point, there are boundary conditions such that the resulting finite CNN, defined by the same template, also has a stable equilibrium. The conclusion is that CNNs can be divided into three different groups with respect to the influence of boundary conditions on their stability.

- ★ CNNs that *always have stable equilibrium*, e.g. such that $a_{00} > 1 + \sum_{m,n\neq 0,0}|a_{mn}|$ because there is a stable equilibrium in every point of the state space where all states are saturated. Loss of stable equilibrium is independent of the boundary conditions.
- ★ *Always completely unstable* CNNs regardless the boundary conditions. The instability is "intrinsic" and is easily unveiled by analyzing the templates locally. The size of neighbourhood is at least 2.
- ★ *Stable with some boundary conditions but not with others:* for instance some linear finite one-dimensional CNNs with opposite-sign templates are unstable if the boundary conditions are set to zero and stable if they are set to ±1. The instability of this group of CNNs depends more on boundary conditions than on the template that defines them. The instability is therefore "extrinsic", which force to examine the whole CNN globally before discovering the instability.

It is then natural to take the accuracy of boundary condition into consideration in an effort to obtain the robust templates. Hence, a revision and modification of the approach of template optimization in section 8.4 is desired. The influence of boundary conditions is the main target of a Master thesis [78] that the author of this thesis has co-supervised. The algorithm shown in Figure 8.8 is extended with a loop that takes into account all possible boundary condition values between -1 and +1 with step size of 0.1. Section 8.6 discuss the implementation of the extended algorithm and presents the obtained results.

8.6 Extended Template Optimization Algorithm

In line with the original work presented in the previous section, the algorithm is implemented in two models: a software model using both Java and MATLAB (Figure 8.10), and hardware model that digitally emulates the algorithm on an FPGA. In the former, an indexed set of all possible templates is generated by means of Java classes. The number of templates in the set depends on the number of free parameters in the model, i.e. non-zero template coefficients (Table 8.3), their data representation (Table 8.4) and the tuning ranges to be covered (Table 8.5). The MATLAB cluster performs the actual optimization and provides a table with the robust templates corresponding to each truncation level. One of the distinguishing characteriscs of the software model is that it is consuming. For the nominal set of templates (14641 templates), the MATLAB cluster requires more than 130 hours for the ascending approach and 223 hours for the descending approach before all options are checked. The search is performed on a PC with Pentium 4, 2.4GHz and 1 GB RAM, equipped with Microsoft XP Professional service pack 3. This is mainly caused by using fixed-point objects to perform the actual truncation in MATLAB. Implementing the model in hardware seems the only way to salvage time constraints.

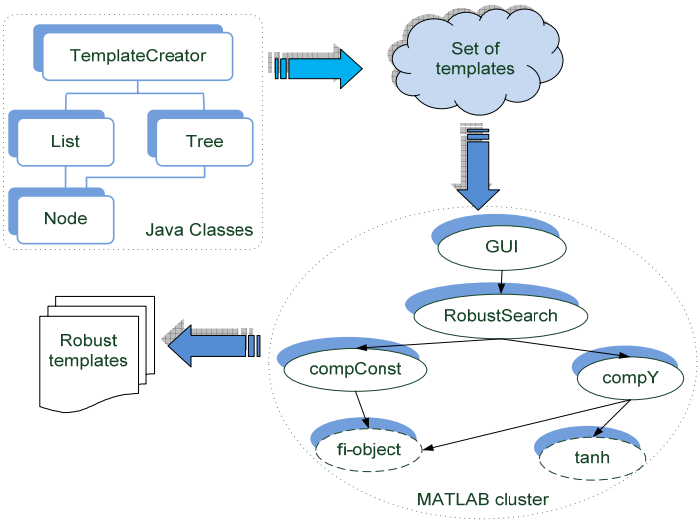


Figure 8.10 Software model of template optimization approach, where only most important classes and functions are shown. Dashed ellipses indicate MATLABs own functions. The function compConst computes the constant corresponding to control and offset contribution as stated in section 4.2, while compY computes the feedback contribution.

The hardware model is based on the scan-architecture embodiment Caballero. Here, the size of network is, however, less important as the input image is small, 7×6 pixels only (Figure 8.7), which opens for denser macro utilization as more components can be accommodated per node. Two multipliers per node allow for handling two input values in parallel, which reduces the clock count per iteration and boosts the overall throughput. This is desirable in order to shorten the execution time of the software model. The node is further equipped with a dynamic truncation unit to perform the actual optimization (Figure 8.11).

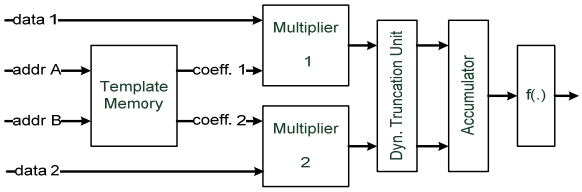


Figure 8.11 Structure of the modified Caballero node. Communication interface and nodal controller are not shown.

As two values need to be available simultaneously, the communication scheme is modified. Figure 8.12 shows how data is first send to East and North neighbours, of which each forwards it perpendicularly on a counter-clockwise manner within the neighbourhood. Then, data is broadcasted South and West and further forwarded West and North respectively. Doing so, all nodes in the

neighbourhood receive their neighbours' values in 4 steps and produce the output in the fifth step. This reduces the communication time-overhead by 50% compared to the original approach employed in Caballero.

8.7 DISCUSSION

Computing reliability of a circuit is usually decreased due to a number of errors. The errors originate from different sources: during the manufacturing process, internal noise, e.g. thermal noise, and external noise such as electrostatic discharge. Therefore, the existence of error sources is accepted, and the focus in fault-tolerant computing is on minimizing the influence of these errors on the processing results [79]. Thus, it is not at all surprising that existing CNN chips suffer from parameter scattering. Different approaches have been tried to tackle the problem for both continuous-time and discrete-time systems. Of these approaches, some are targeting a specific chip, while others are chip independent. The focus has been on retaining stability of the system with a solid mathematical analysis of the proposed algorithms. However, the influence of boundary conditions seems to have sunk into oblivion, which this chapter tries to salvage. The proposed approach is general and is adaptable to any digital design.

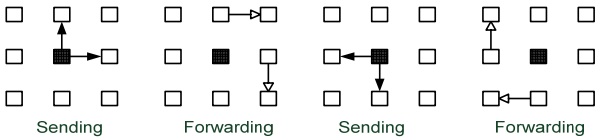


Figure 8.12 The inter-nodal communication is modified to allow the usage of two multipliers. Two values are received /submitted simultaneously.

A completely software-based model is compared to software-aided HW model by means of number of robust templates each model finds and the time needed to achieve that. Both software and hardware models find the same robust templates, where both ascending and descending approaches are tried. None of the approaches finds any robust template in the range [0.6, +1] at any precision level, while at least one truncation level results in robust templates in the range [0.4, 0.5]. Figure 8.13 and Figure 8.14 give an illustrative view of the situation while the complete tables are given in Table A.1 and Table A.2 in Appendix A.

For the descending approach, far more robust templates are found with boundary condition 0 than with boundary condition -1 for all truncation levels. About 6 times more robust templates are found for full truncation <5:2>.

In order to gain a better understanding of the dependencies between boundary conditions, precision and template robustness, 3-D views are shown in Figure 8.15 and Figure 8.16. Here, the absence of robust templates for boundary conditions in the range $[+0.6, +1]$ is clearly seen. A visual inspection of the 3-D diagrams reveals two significant differences in how the approaches evolve. In general, the ascending approach finds less robust templates for all intermediate steps, but finds exactly the same number of robust templates as the descending

approach in the final step. The peak of robustness is shifted a bit compared to the descending approach and is located at boundary condition -0.1 instead of 0 for all precision levels.

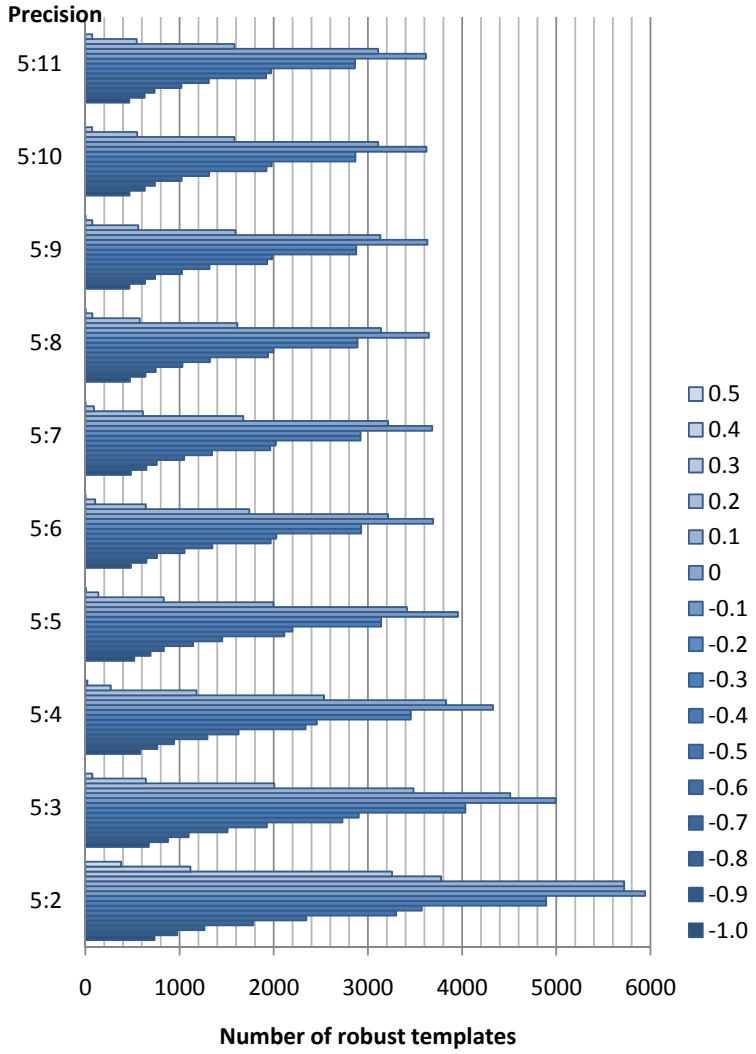


Figure 8.13 Number of robust templates for different boundary conditions in the ascending approach. No robust templates are obtained for boundary values in the range [0.6, +1] for all precisions.

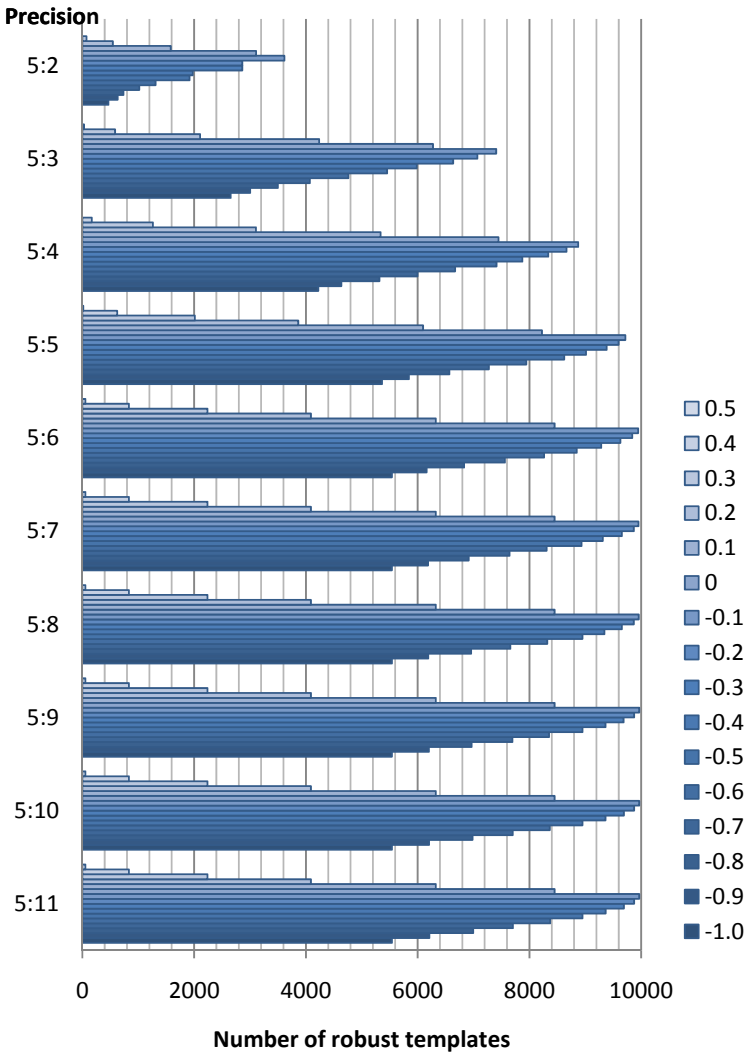


Figure 8.14 Number of robust template for different boundary conditions in the descending optimization approach. No robust templates are obtained for boundary values in the range [0.6, +1] for all precisions.

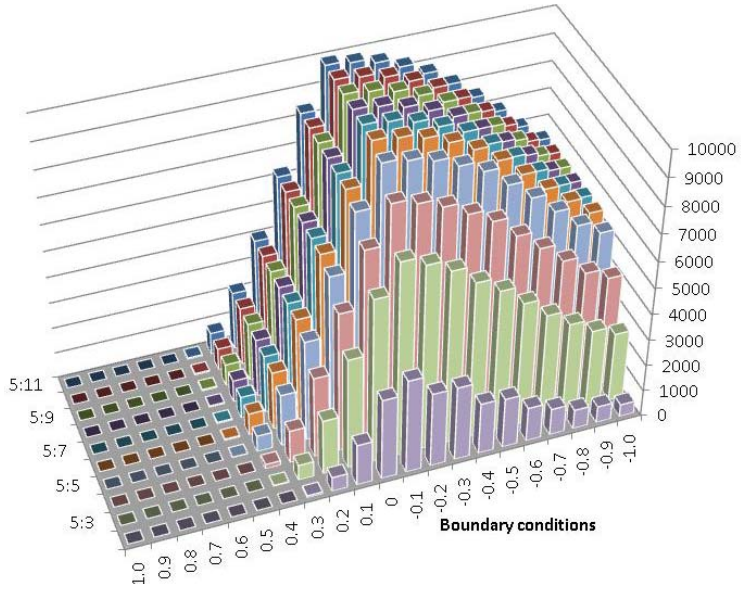


Figure 8.15 A 3-D view of the outcome of the descending approach. First line of columns represents obtained robust templates for each boundary condition on the final optimization step.

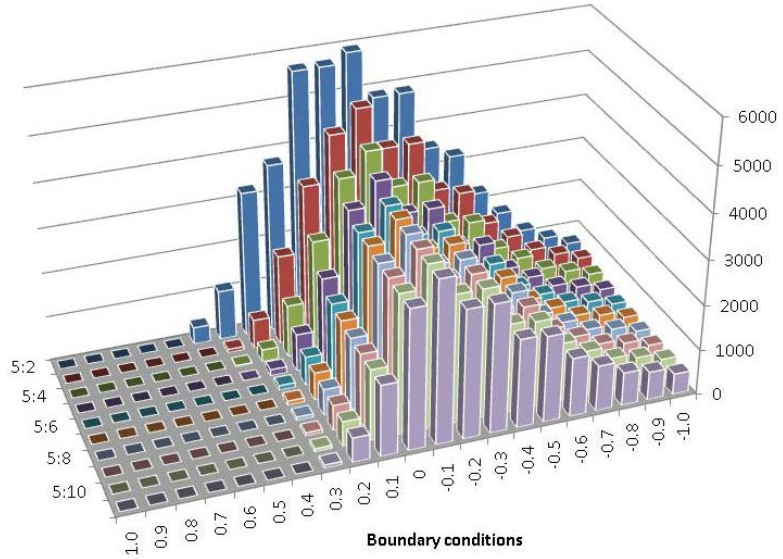


Figure 8.16 A 3-D view of the outcome of the ascending approach. . First line of columns represents obtained robust templates for each boundary condition on the final optimization step.

An important issue has to do with the number of iterations required for achieving convergence for each of the robust templates. Looking at Figure 8.17 we can see that the iteration count increases for each precision level in the ascending approach, while it decreases successively in the descending approach. Notably, some of the robust templates require far more iterations than the input image theoretically needs due to its size (Figure 8.7). However, as more templates are found in the descending approach in the intermediate truncation levels, the overall require time is longer than for the ascending approach. In other words, the ascending approach is preferred as it is faster and provides the same result at the last optimization level.

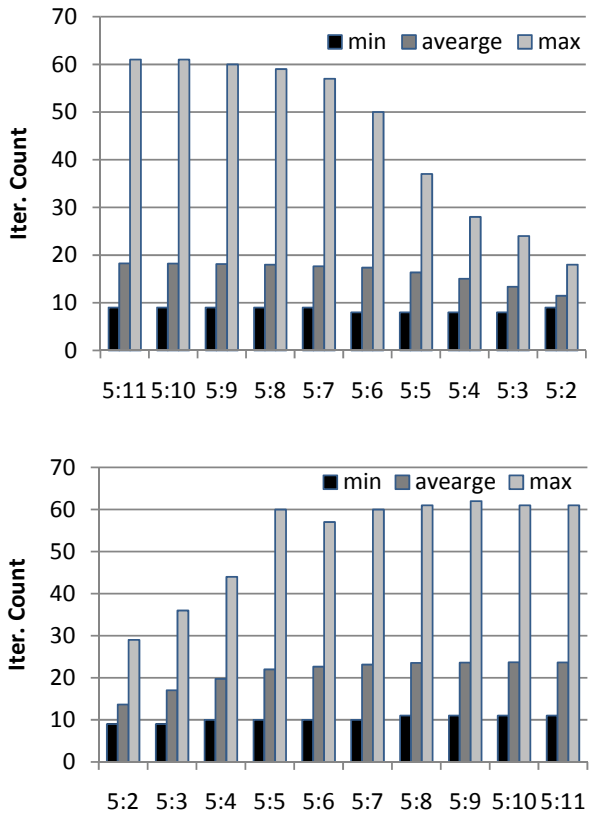


Figure 8.17 Iteration count of robust templates obtained in the descending approach (top) and the ascending approach (bottom) for boundary condition -0.1. Other conditions show a similar behaviour. Note that the horizontal axis is flipped to emphasize the direction of optimization.

The question is whether both approaches find the same templates to be robust at least in the last optimization level. To ease the comparison, notions

from set theory, such as intersection, union and symmetric difference of two sets, are employed. The definitions of these set operations are given below.

Definition 7.1: Laws of Set Theory[110]

For a given universe $\mathcal{U}$ and for $A, B \subseteq \mathcal{U}$:

- The union of A and B : $A \cup B = \{x | x \in A \vee x \in B\}$.
- The intersection of A and B : $A \cap B = \{x | x \in A \wedge x \in B\}$.
- The symmetric difference of A and B : $A \triangle B = \{x | x \in A \cup B \wedge x \notin A \cap B\}$.
- The cardinality of A: $|A|$ = number of elements in A. ■

Let's first consider the ascending approach and focus on the final optimization level, i.e. precision <5:11>. A set of robust templates is denoted S_i, where i $\in \{-1, +1\}$ stands for the boundary condition in use. Looking at Table A.3- Table A.5, the following is observed for sets $S_{-1}$and $S_{-0.9}$

$$\left.\begin{array}{c} |S_{-1} \cup S_{-0.9}| = 630 \\ |S_{-1} \cap S_{-0.9}| = 466 = |S_{-1}| \\ |S_{-1} \triangle S_{-0.9}| = 164 \end{array}\right\} \Rightarrow S_{-1} \subset S_{-0.9} \tag{8.9}$$

Performing the same comparison reveals the relation in Eq. (8.10). In other words the templates that show to be robust for boundary condition g are also robust for boundary conditions $g + 0.1, \forall g \in \{-1, -0.2\}$. The relationships among robust template sets $S_{-1} - S_{-0.1}$ are illustrated in Figure 8.18 left.

$$S_{-1} \subset S_{-0.9} \subset S_{-0.8} \subset \cdots \subset S_{-0.3} \subset S_{-0.2} \subset S_{-0.1} \tag{8.10}$$

For positive boundary conditions, i.e. $\{0, +0.3\}$ as other conditions do not result in any robust template for precision level <5:11>, a different story holds. Eq. (8.11) depicts the relation between $S_{0.1}$ and $S_{0.2}$. Similar discussion leads to the situation illustrated in Figure 8.18 right.

$$\left.\begin{array}{c} |S_{0.1} \cup S_{0.2}| = 1658 \\ |S_{0.1} \cap S_{0.2}| = 469 \neq |S_{0.1}| \neq |S_{0.2}| \\ |S_{0.1} \triangle S_{0.2}| = 1189 \end{array}\right\} \Rightarrow S_{0.1} \not\subseteq S_{0.2} \wedge S_{0.2} \not\subseteq S_{0.1} \tag{8.11}$$

It remains to examine the relation between the template sets corresponding to positive and negative boundary conditions. Comparing the sets that are closest to each other from both groups, i.e. S_1 and $S_{-0.1}$, seems a good idea (Eq. (8.12)). The chain of proper subset relations as given in Eq. (8.10) is broken at the transition between negative and positive boundary conditions.

$$\left.\begin{array}{c} |S_{-0.1} \cup S_0| = 4296 \\ |S_{-0.1} \cap S_0| = 2427 \neq |S_{-0.1}| \neq |S_0| \\ |S_{-0.1} \triangle S_0| = 1869 \end{array}\right\} \Rightarrow S_0 \not\subseteq S_{-0.1} \wedge S_{-0.1} \not\subseteq S_0 \tag{8.12}$$

A careful look at Table A.3, Table A.4 and Table A.5 shows further that only a small amount of robust templates found with positive boundary conditions work properly with negative boundary conditions. Eq. (8.13) makes a good example.

$$\begin{aligned} |S_{0.3} \cap S_{-0.1}| &= 45 \\ |S_{0.3} \cap S_{-0.2}| &= 44 \\ |S_{0.3} \cap S_{-0.3}| &= 44 \\ |S_{0.3} \cap S_{-0.4}| &= 1 \\ |S_{0.3} \cap S_{-0.5}| &= 1 \\ |S_{0.3} \cap S_{-0.6}| &= 0 \end{aligned} \tag{8.13}$$

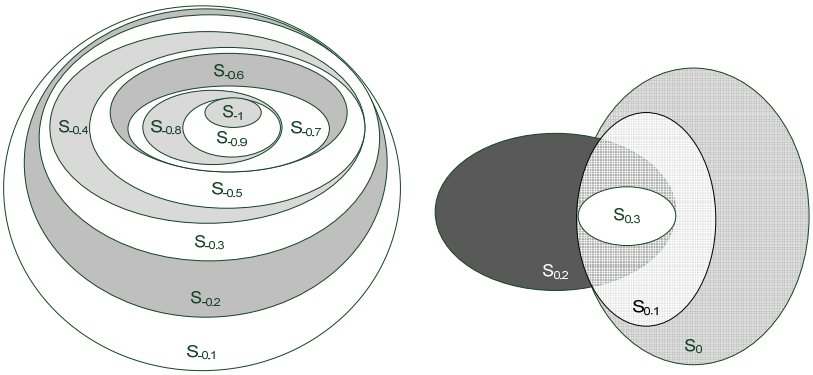


Figure 8.18 Complete overlapping of sets of robust templates is found from boundary condition -1 down to -0.1 (left) while positive boundary conditions give rise to a different situation (right).

To summarise, all templates in S_{-1} show to be robust for all boundary conditions down to -0.1, but none of the templates $\in S_{-1}$ is robust for boundary conditions that equal or are larger than 0. On the other hand, a subset of robust templates with boundary condition $+0.3$ are also robust for some conditions less than 0. The overlapp between positive and negative boundary conditions decreases further when we move away from 0 toward -1. The same observation is completely valid for the descending approach as exactly the same sets of robust templates are obtained for all boundary conditions.

Chapter 9

System Architecture
A proposal

It is well known that strongly nonlinear systems give rise to chaotic oscillation. In fact, , the universal CNN is inspired by Chua's earlier work on chaotic oscillators [7], which gives the clear advantage of employing CNNs to handle nonlinear systems. Obviously oscillation effects easily hamper typical dynamic behaviour of a CNN when feedback is activated, while feedback-less settings such as for erosion and dilation will not be affected. A CNN is considered completely stable if all cells converge to equilibrium states. But complete stability is sensitive to parameter settings, mainly in feedback template A. This sensitivity leads to, e.g., the occurrence of reaction/diffusion phenomena in higher levels of integration, i.e. when 2 CNN layers are coupled.

Due to parameter sensitivity, hardware implementation of a CNN turns to be application dependent. We have seen in Chapter 8 that a comprehensive evaluation in MATLAB is crucial to achieve a reliable digital design that performs the basic operation of hole filling. This experimental approach tends to be harder and, thus, requires more time to perform if template complexity is higher. In many other cases, an analytical approach is considered. For instance, in [61], template parameters are first mathematically derived and then checked for robustness through extensive simulation. The conclusion is that templates with high connectivity, e.g. edge detection, result in large template values, which implies the need of further optimization by using algorithmic approaches. Obviously, design automation will reduce time and save effort while reliability

Parts of this chapter have been published in [IV] and [VII].

is clearly increased. In other words, there is a need for an automated system that extracts information from a user-supplied processing recipe and then builds the CNN system. In such a system, robustness is a key issue.

The newest digital implementations, e.g. word-serial approach, pave the way for improvement of network capacity by merging temporal distribution of many cells inside a single node and spatial distribution of many nodes within the network. This allows entire CNN programs to be handled with minimal memory access. This invites to the definition of a system architecture and an appropriate application programming interface.This chapter proposes such an architecture in sections 9.2 and 9.3, but first section 9.1 discusses how an automated system for CNN implementation is generated starting from the basic elements.

9.1 DESIGN AUTOMATION

Ter Brugge discusses the front-end in [40], where algebraic expressions are interpreted and converted to a normalized notation, from which different CNN architectures can be derived. Technology mapping takes care of various system optimizations, notably in the processor/memory balance, in different ways. The amount of parallelism is reduced as operations are performed on the previous results without saving and reloading data. Furthermore, transformations produce templates in an arbitrary (also larger than 1) neighbourhood. Overall, the front-end produces a CNN architecture that is efficient by large but has not necessarily taken the technological restrictions of an eventual hardware realization into account.

The back-end has therefore to be extended to take care of hardware requirements. For instance, data representation has great impact on both computation and communication schemes of the different values within a CNN. A proper choice of arithmetic will, thus, largely affect both area and time overhead. Another issue is the size of CNN needed to facilitate a proper information processing systems. In image processing, e.g., an image is therefore sliced into sub-frames that are small enough to be accommodated on the network. The subsequent snapshots of the image have to be overlapped over a pixel thickness equal to the size of the neighbourhood. Unfortunately, this works only for templates with locality of operation. Furthermore, it takes an appreciable amount of processing capacity away. The larger the neighbourhood is, the harder are the demands.

In short, an automated design system will start with algebraic expressions and end in a fully functional full custom design on ASIC. The overall system development procedure is depicted in Figure 9.1. The algebraic expressions allow for separating the processing steps from the flow control and are easily described using MATLAB instructions. The aim is to bring as much of the expressions together in single functions as this provides a basis to generate an efficient set of CNN templates This step is independent of hardware implementation needs such as data representation, word-length and communication schemes. By now, the obtained result constitutes a minimal solution of the processing application in terms of CNN operations, i.e. a

minimal sequence of consecutive templates. The number of templates in use has a direct impact on the amount of external memory access, as reloading of intermediate results may be required. In this sense, the MATLAB model provides us with optimal performance in terms of operation latency and memory access overhead.

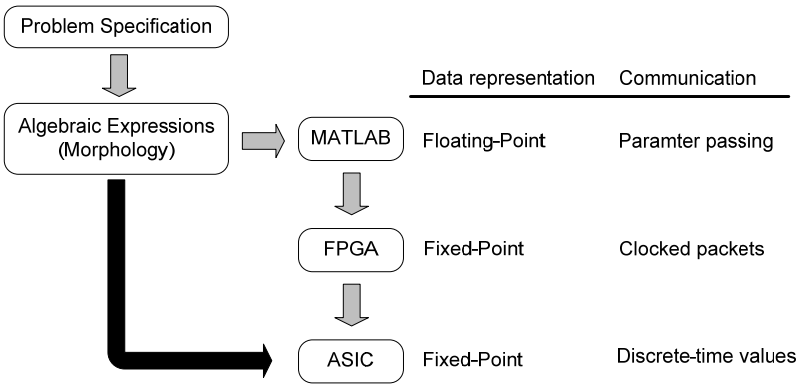


Figure 9.1 Moving from algorithm to hardware.

The next development stage uses an FPGA platform, for a number of reasons. Firstly, the step toward a full custom ASIC becomes shorter when technology and design changes are easily accommodated. This reduces both cost and time. Secondly, modern FPGAs combine the over-mass of flip-flops with high-density and multifunctional macros such as multipliers and memory blocks. Moreover, modular construction is simplified due to the physical placement of the functional units. This allows bundling logic and macros to easily form CNN nodes and makes the FPGA technology to first-hand choice of programmable devices.

MATLAB works with double-precision floating-point numbers, the largest number representation supported on nowadays general-purpose computing platforms. This is not feasible for resource-critical digital platforms. Hence, float-to-fix conversion is needed. On the FPGA platform, functions are applied on basis of fixed-point values represented as arbitrary long bit-strings. The limited resources on an FPGA force, however, to focus on making the functional components as small as possible. One way is to accommodate shorter word-length for the internal numbers. This is achieved through gradual decrease of the internal precision in line with the approach used in section 8.6. This is not only aimed to make the functional macros smaller, but also to evaluate whether a precision can be achieved that is as low as inherently coupled to analogue implementations. MATLAB is very useful here as well.

The question on the required transfer characteristics that next needs to be answered cannot be handled in MATLAB anymore. The aim is to reduce the traffic density as much as possible. This, in combination with the allowance of medium-size precision, shows that also a realization of analogue function macros, embedded in a digital network-on-chip, can be afforded. Through

combining small cores with small inter-nodal communication interface, a larger network can be accommodated on single FPGA.

At the last stage, we further question the value transfer by the digital network that was originally introduced to allow for a smooth design flow. Having subsequently reduced the computation and the communication requirements, we may find ourselves in the situation that a fully digital realization proves to be feasible.

9.2 ARCHITECTURAL OVERVIEW

The CNN Instruction Set Architecture (ISA) defines the exterior of the CNN Image Processor in terms of signals and visible memory locations. The overall CNN ISA is depicted in Figure 9.2. Overall we find four modes of operation and their respective instructions using two separate bus systems: the Image Memory Bus (IMB) and the Host Interface Bus (HIB), both with a R/W signal and strobed address and data bus.

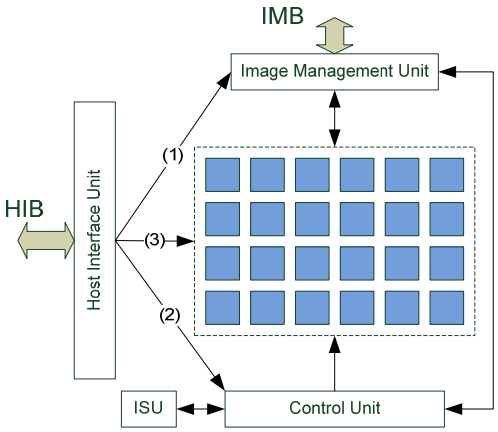


Figure 9.2 External View of the CNN Architecture

Window

The window operations influence the image management unit only. It converts physical into virtual pixels and will autonomously fill the CNN with pixel information with respect to the designated Region of Interest (RoI) for any frame format using the Image Memory Bus (IMB). Using the window settings it is possible to repeat the CNN program on a steadily smaller part of the image while increasing the resolution.

Frame Size: the width and height of a frame in pixels

Centre coordinate: the non-sampled centre of the first frame to be handled.

Configuration

The internal operation is governed by a number of tables, downloaded over the HIB. They all start with a preamble that gives the general table information

and then subsequently provides the table entries. The template and discrimination table will be distributed to all nodes, while the program table is saved in the Instruction Store Unit (ISU).

Discrimination: table for discrimination function

Program: Instruction Store (opt.)

Template: label and content of a template

The *discrimination function* lists the transformation from internal node status to external data result. The length of the table is therefore given by the table size divided by the table step.

The *program* tells the successive applications of pixel operations that can be either templates or hard-coded linear instructions. It implicitly relates the use of various layers and how they are combined either in time or in space. A *template* gives each CNN function. Templates can be downloaded and stored in every CNN node for use later on. The *pixel operations* can be selected from a number of linear (hardwired) and non-linear (downloadable) options. The instructions will be placed into a separate ISU.

Logical: NOT, AND, OR, EXOR.

Arithmetic: Sum, Minus per pixel or horizontal or vertical

CNN: refers to downloaded templates

Run

Run: none, per clock, per iteration, per template till a specified breakpoint in the program.

Boundary: the boundary conditions as stated in the templates can be overwritten for debug purposes.

Sample Size: the amount of physical pixels represented by one virtual (CNN internal) pixel as implied by the window can be overwritten for debug purposes.

Mode: only this window, or a stripe of the entire image

Debug

The ISA makes the CNN network architecture invisible to the host program and therefore allows a late binding of the actual structure to an application at hand. More often than not, the development network is different from the production network. Starting from a MATLAB model with values represented in a double floating-point format, a gradual conversion into fixed-point numbers is needed (section 9.1). The length of the internal words is application dependent, though accuracy can be easily guaranteed by block-based scaling with a factor derived by inspection of the templates. In practice we have not seen the need for more than 8 bits precision, but for simple templates a smaller length can be accepted.

In line with this, we have inserted a number of in-line debug facilities. The system can be run in various time step size, inspected for network data, while allowing to overwrite the network status and to continue from the existing status.

9.3 SYSTEM COMPONENTS

In our reference system we assume that the network is configured separate from the rest. Consequently we have to ensure that the system components can handle appropriate network architectures.

9.3.1 Host Interface Unit (HIU)

A host must be able to control the overall functionality of the system by sending instructions and cloning templates and by setting a number of configuration parameters. The communication is handled by the HIU that receives the requests from the host over the HIB and forwards them to the system using a wishbone bus. The HIU is as well responsible for data delivery to the host. Figure 9.3 shows the main components. Two different FIFOs are used, one for acquiring host requests and one for putting out data to the host.

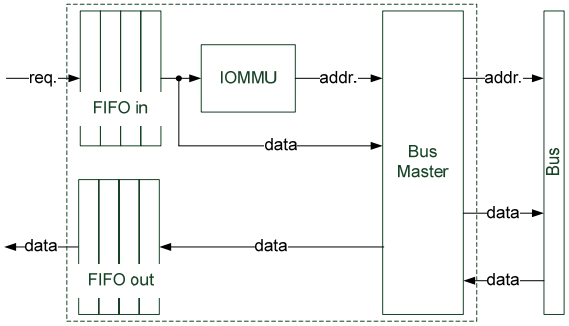


Figure 9.3 The HIU consists of two FIFOs for communication with the host, IOMMU for address translation and a bus master to communicate with other units in the system.

A host request is 25 bits long and is divided into 3 fields: a Read/Write flag that determines the type of the request, a virtual address field and a data field that is of interest only in write-requests (Figure 9.4). Once a request is captured by the FIFO, the virtual address is translated into a system memory address by the Input/Output Memory Managemet Unit (IOMMU). This address will serve as a base address for all incoming data as long as the virtual address field in the subsequently received requests remains unchanged. The bus master acts partially as a Direct Memory Access (DMA); it generates the proper addresses from the base address and put it on the address port of the wishbone bus. In case of a read request, once data are available, the wishbone bus raises an acknowledgement signal notifying the bus master that reads the data and put it on the output FIFO. Write requests are handled similarly. Here the acknowledgement signal notifies the bus master that the writing of data is accomplished so next pair of address/data can be handled.

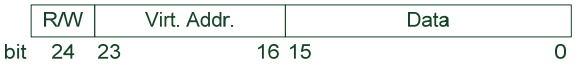


Figure 9.4 A host request is subdivided into flag, address and data fields.

Looking into area utilization for the different components in HIU, Figure 9.5 gives an impression of the incurred overhead. We take here a FIFO of only 1 deep. Most of the logic is hence devoted for IOMMU and Bus Master only. It can be clearly seen that the bus master requires more slices and FFs than the HIU itself! This is due to the fact that some of the signals in the bus master are not used at all and therefore optimized away.

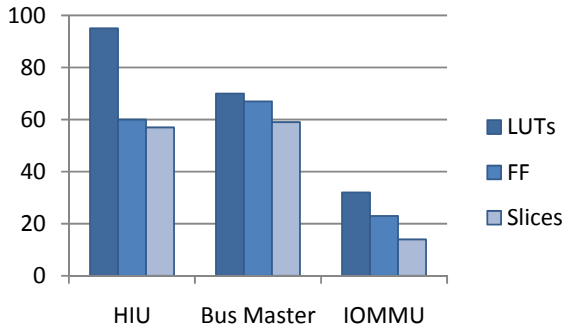


Figure 9.5 Area Utilization for HIU and two of the sub-components.

9.3.2 Image Management Unit (IMU)

The camera captures images and stores them in an external memory. The 8-bit greyscale pixels are then retrieved and converted by the IMU to a signed fixed-point notation with a precision of 7 bits for the fractional part. One of the main operations of the IMU is the windowing operation. As the size of the network is far much smaller than the processed image frame, a gradual zooming toward the RoI is required. At the beginning the RoI covers the entire frame, where each CNN node on the chip is mapped onto a virtual pixel that corresponds to a group of real pixels in the image. The virtual pixel is suitably obtained through a conventional averaging of all pixels in the corresponding group. In a next round the RoI covers a smaller part of the frame depending on the output of the previous round.

9.3.3 Control Unit (CU)

The unit has direct communication to the CNN core and the HIU through wishbone buses. It is built with the concept of pipelining in mind and consists of two main components: Instruction Fetch and a Controller (acts as instruction decoder). The naming convention is somehow misleading as the former pipelining stage generates two additional signals; control (used by the Controller pipeline stage) and iteration; in addition to the instruction that is fetched from a dual-port RAM. The controller consists of two major components. One is the actual instruction decoder and provides the proper template, while the other generates CNN-enable and instruction-done signals depending on the number of iterations and whether equilibrium is reached or not. Figure 9.6 shows a schematic view of the control unit.

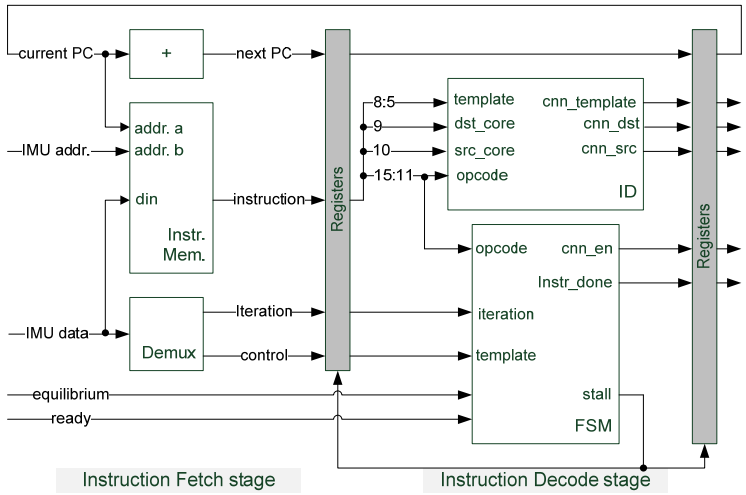


Figure 9.6 Control Unit schematic view.

The instruction memory (ISU) is arranged as shown in Figure 9.7 with space for 64 instructions, while Figure 9.8 illustrates the area utilization for the main components. Taking a Xilinx Virtex-II 6000 as reference which accommodates 34,000 slices, we find from Figure 9.5 and Figure 9.8 that the overhead incurred by turning a CNN network into a system ranges from 1% for a limited edition to 5% for a complete one with large buffers.

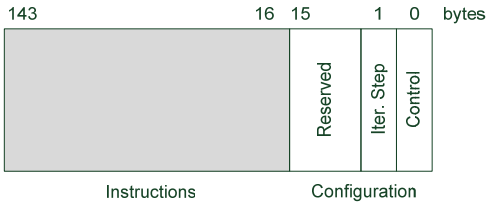


Figure 9.7 Memory address space as used by the control unit.

9.4 DISCUSSION

Design automation tools such as ECAD (electronic computer-aided design) have rapidly gained popularity with the continuous scaling in semiconductor technology. The aim is to make the translation from graphics to electronics smoother, more reliable and less time consuming. We have seen in this thesis how application may steer the implementation of a CNN on hardware. Furthermore, it has been proved that pruning of internal signals is possible without any effect on the obtained result. Obviously, selection of a proper architecture already at the beginning is important, but performing the desired adjustments is also critical. Doing so manually is not feasible, which implies the

need to automate the search for the perfect set-up. The belief is that such an automated system will be highly appreciated.

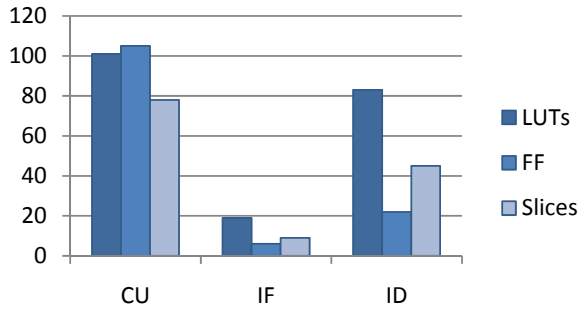


Figure 9.8 Area utilization for the Control Unit and the sub-components Instruction Fetch and Instruction Decoder.

The choice of floating-point and fixed-point representations in hardware implementations has so far been ruled by the desired precision and accuracy in the target application. As one of the major goals in CNN hardware design is accommodation of as many nodes as possible on a single chip, fixed-point representation is prefered. Addition of two floating-point numbers is rather complex and requires additional control for alignment and postnormalization. The longer latency for each accumulation operation leads to performance degradation. In contrast, fixed-point addition is straightforward and requires minimal control, which even results in smaller needs of logic resources. In the proposed automated system a migration from float- to fixed-point is thus crucial.

It is worth mentioning that the first emulator ACE is based on a floating-point computation core. The decision is based on the observation that the 'limited' accuracy obtained in fixed-point representation is not enough to solve partial differential equations. But even in ACE16K, the obtained results of these equations are not accurate enough to be used in engineering applications [111]. Actually, moving from floating-point to fixed-point numbers is not merely a question of reducing the value scope, but a careful arbitration between precision and accuracy. Precision is addressed by the smallest step in the value space, such that small variations in the value space have little significance in the problem space. Accuracy on the other hand has to do with the degree by which the computation can achieve the desired result. Figure 9.9 tries to visualize the difference between these two notions. It is shown that if one is precise the result will be consistent but may still be off target. There is evidently enough discriminatory power in the value space. If one is accurate, the average over repeated calculations will be in order but the individual readings must be precise to get to the target. The representation range of floating-point numbers outperforms the one obtained in fixed-point numbers but it comes at the expense

of a smaller precision [112]. On the other hand, it is now proved that a digital CNN performs properly even with less accurate fixed-point internal signals.

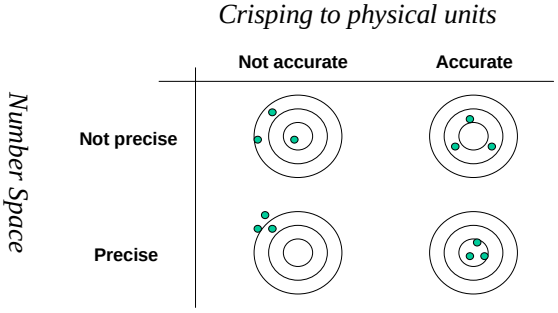


Figure 9.9 Precision versus accuracy

Furthermore, the large variation in digital CNN implementation shares a number of common principles. Such principles can be used to define an Instruction Set Architecture that interfaces host programs from CNN phenomena, allowing a soft core based implementation.

The system architecture is founded on the recognition that larger networks pose an increasing demand on memory access. Applying more templates to the once loaded data alleviates these demands. The more templates can be handled, the larger networks can be allowed. The development of an effectively small node by employment of time multiplexing has made this possible.

In the Bi-i system [89], we find a potentially large but analogue network. However, for a typical application we need both CNN and non-CNN operations. An example is motion detection, a central scheme in various areas of vision sensing, both in industrial as in consumer applications (section 7.2). For the non-CNN operations, the Bi-i system needs an additional digital signal processor. In a fully digital system, as presented here, such functionality can be integrated into the basic node. This relieves the Host Interface Bus from a lot of bandwidth problems.

This leaves the issue of the Program Store. This is solved by having a configurable Stored Program Architecture. The first reason is the separation between scope and function. With a Stored Program we cannot only operate on subsequent images but also on several sample sizes within the same image without burdening the host computer.

The ISA is especially helpful when the actual parameterization for the network is not clear during development, but should not influence the application at hand. It has become practical because the virtual network size has been raised from a meager 144 nodes to 4096, and can probably be raised even higher. This makes a digital CNN a practical alternative for image processing. As such networks do not have a global control, their intrinsic speed ought to be much higher than usual.

Chapter 10

Further Considerations

There is a lot of information concealed in the sequencing of frames, but it is not easy to get it out. A pixel-wise comparison is not easy to compute. It will be slow which defies the purpose of dynamic knowledge extraction or uses specialized hardware. Cellular Neural Networks can be used for this purpose, as shown in [47]. For the digital implementation, where the dimensions of the problem (Figure 4.7) are mapped on the two dimensions of a Field-Programmable Gate-Array, not all potential architectures permit such applications in an efficient way.

The key issue seems to be whether access to image information stored off-chip can be kept outside the inner loops of the computation. This is clearly exemplified in the original ILVA architecture, where the computation is unrolled on the nodal iteration dimension at the expense of the on-chip image salvage. The consequence is that image stream manipulations will involve a bandwidth problem with respect to the external image RAM.

The principle of broadcasting processing elements, loosely coupled through a NoC-based architecture retains the potential of image stream handling. Of course, in the present generation of FPGAs, the amount of on-chip memory is not large enough to store the desired number of frames. However, the ongoing increase of storage capability of modern FPGAs indicates that the newer generations will be the better platform for real Wave Computing.

Parts of this chapter have been published in [IV].

The fundamental critique on the implementations presented earlier is that the discrete-time formulation as given in the CNN nodal equation is not handled cycle-true as the implementations are based on the communication of converged results. For instance, simultaneous transfer of values within a neighbourhood gives rise to bus conflicts in the state-scan approach. In order to avoid such conflicts, nodes in Caballero are activated at a Knight Jump distance, which burdens the design with additional activity control and severely reduces the amount of potential parallelism (Figure 4.16). The special treatment of the activation pattern of edge nodes complicates the control further. Apparently, this adds heavily on the control and severely reduces the amount of potential parallelism. The amount of additionallt required logic is so big that a larger neighbourhood is basically precluded. Admittedly, all previous implementations emulate the functionality of CNN rather than providing real-time performance. One way to overcome interconnect limitations is to use a bit-serial communication scheme, which allows all nodes to immediately consume the values that are currently produced at the neighbouring cells. This will be part of the Network Interface (NI) that wraps any design part to become accessible through the network standard. It brings out the basic advantages of the time-multiplexed communication and is fully in-line with the original Æthereal systematic [83]. But it also presents a degree of overhead that needs to be minimized [91]. Therefore it demands investigation, how the concept of serial processing can be moved further into the node. Furthermore, as all nodes can be active simultaneously, the activation cycle employed in Caballero is not needed anymore, which saves a global controller. Such a scheme has a small footprint and scales well with increasing neighbourhood.

As communication schemes are strongly coupled with the usage of word-level arithmetic, arithmetical constraints are of crucial importance even when optimized communication schemes, e.g. bit-serial, are used. For instance, connecting the bit-serial approach to the existing Caballero node, whose computation performance is built on word-level arithmetic, requires buffering to create series/parallel conversion and vice versa. Apparently, this introduces both time and logic-overhead such that the benefits of a bit-serial communication scheme are lost. One way to remedy this is to use serial arithmetic together with bit-serial communication. The operation of a fully serial approach is illustrated in Figure 10.1. For every step, the coefficient bits multiply the single-bit input from each neighbour; the results are added and accumulated. It requires the coefficients to be locally available in a ring-buffer. This is not as bad as it seems, because a serial shifter can be implemented in a single LUT per 4 bits. For longer coefficients one may consider to build the ring-buffer in the Block RAM, but usually coefficients are not long. Together with the bit-register for the input and the bit multiplier, a 4-bits base unit takes just a slice. The outputs are tree-wise added and give a 4-bit result to be added to the shifting accumulator. This final addition has to be in parallel because long carry propagation may occur. Also the result has to be available in parallel, because a final table lookup is needed for the output discrimination.

As usual in bit-serial logic, the data-path becomes small but at the expense of a more complicated control. Furthermore we can expect a higher latency, as more clock ticks are needed to get to the result. But that is only true for the single node. The basic 10 clock cycles for a single node in Caballero have to be repeated for 5 neighbouring nodes due to bus contention. It does not seem likely that a serial solution that eliminates such bus contention problems will need more. As the small serial node allows for a larger network to be implemented on a single chip, it is worthwhile to evaluate its potential. This provides not only for high density but also supports a further higher density increase at the expense of a moderate deterioration in latency, which is usually affordable in commercial applications. .

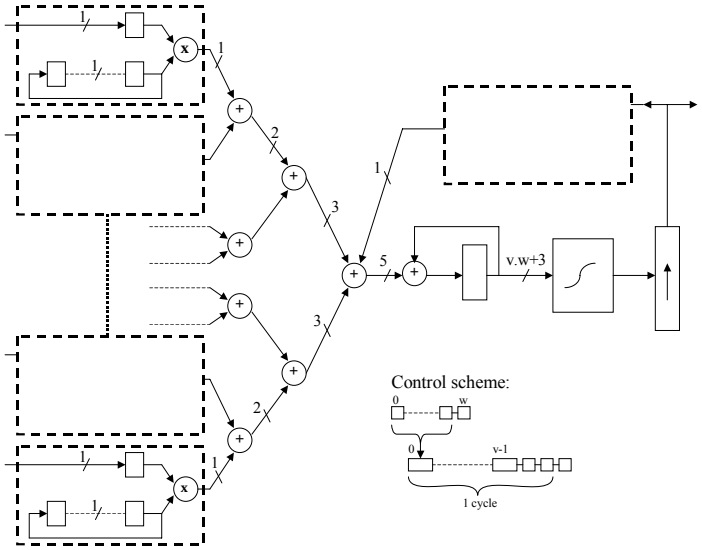


Figure 10.1 A serial architecture for bit-serial communication. Variables v and w represent the width of u/y-values the width of template coefficients respectively.

The overview of the alternative designs, presented so far, shows a rich variety of compromises between speed and area. Starting from the bit-serial structure, even more alternatives can be created by logic transformation. A typical example of such a derivative implementation is in series/parallel computation (Figure 10.2). Every single input bit multiplies the entire coefficient. The outputs are tree-wise added and give a $w + 3$-bit result to be added to the shifting accumulator, where w represents the width of template coefficients. This reduces the latency and the control significantly, but at the expense of wider adders. Connected to this comes the implementation of buffering. Where in the pure bit-serial approach, the buffers are directly created in hardware; in the derivatives it becomes worthwhile to implement the buffers in the local memory.

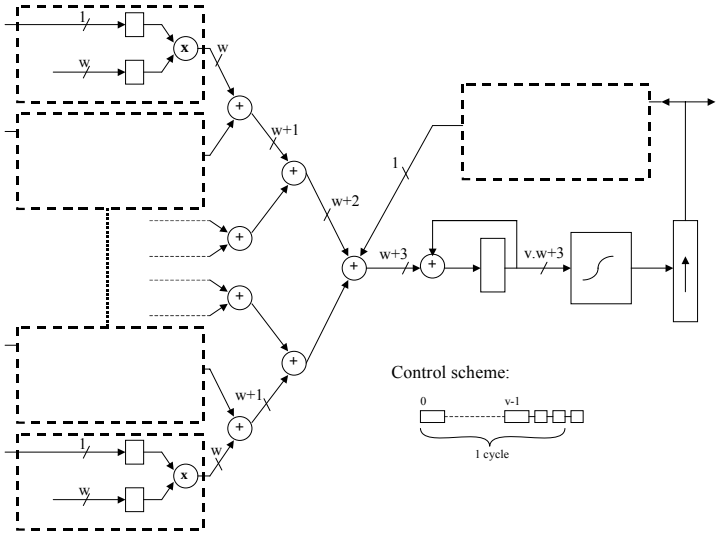


Figure 10.2 Series/parallel architecture for bit-serial communication. Variables v and w represent the width of u/y-values the width of template coefficients respectively.

An overview of the CNN implementation spectrum (VIND) is given in Figure 10.3. The similarity to Corporaal's 4-dimensional diagram about Architecture Design Spectrum [82] reveals the importance of optimizing control and data flows in order to achieve a well performing CNN system, as the case always is with hardware design. The temporal state-flow architecture can be found on the D-axis, while multithreaded improvements like ILVA and Sleipner, reside on the surface between D- and N-axes. ILVA covers the gap to memory by flattening a 2-dimensional computation of nodal equation into a 1-dimensional computation by dropping the intermediate results on the computational path (Figure 4.9). Pipelining is then, like in RISC architectures, a consequence rather than a target. Sleipner opens for larger neighbourhood through rearrangement of internal data flow to achieve better utilization of memory resources, which reminds of the concept of super-pipelining where extra pipeline stages come from decomposing the memory access.

At this stage, the state-flow architecture seems to reach the edge of its performance and a total overhaul is needed. Allowing larger numbers of iterations for each transfer of input values from the neighbourhood will boost the performance in the same way VLIW architecture increases the number of operations per instruction. The state-scan architecture, Caballero, brings this into reality by de-coupling intra-node computation from inter-node communication needs. The same sequence of multiply-accumulate operations is performed repeatedly in each node independently of the state of all other nodes. The state-scan architectures that are originally on the I-axis move with the bit-serial technique closer to the V-axis where the number of simultaneously transferred values is in focus.

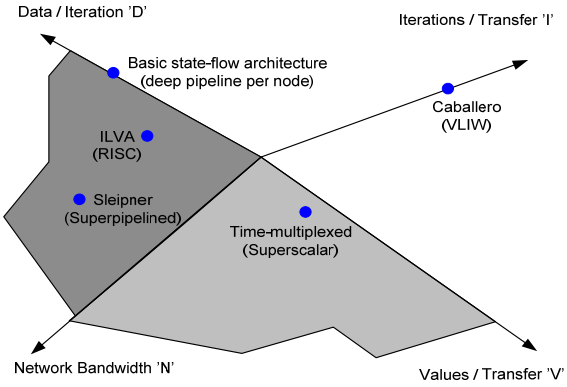


Figure 10.3 The 4-dimentional design space spectrum {V, I, N, D} of CNN architectures. The Time-multiplexed architecture employs the bit-serial technique.

In order to get a better feeling for the design trade-offs, the effects of different implementations are shown in Figure 10.4. The basic clock speed of these designs differs considerably. An improved ILVA architecture exhibits a clock frequency of 144 MHz, which sets the lower clock rate. This is taken into account by normalizing the performance of all other implementations, expressed in clock cycles (cc) per iteration. The figure illustrates the impact of inter-nodal communication as well, i.e. when network interface is considered. The pipelined approach (ILVA) has the larger core, but the communication interface is slim. On the other hand, the parallel approach (Caballero) uses a smaller core, but the benefit disappears quickly when the communication interface is taken into account. The bit-serial approach is superior with small core and little communication overhead.

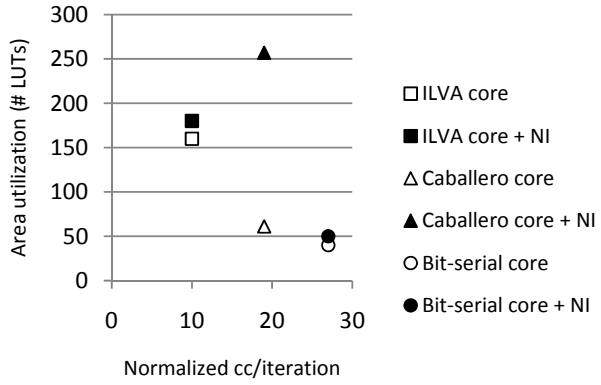


Figure 10.4 Design trade-offs in digital CNN implementations without (hollowed shapes) and with inter-nodal communication overhead in form of Network Interface (filled shapes).

For a proper evaluation, we also need to look into the impact of larger templates. Figure 10.5 shows the effect of larger neighbourhood on the required logic. Caballero exhibits the smallest increase in area (9%) due to the reuse of routing paths. Main impact is on the control mechanism. Unrolling the nodal behaviour into time, as done in ILVA, shows somewhat larger increase in area (30%). In contrast, the bit-serial approach tends to grow super-linear with template size, about 300% increase. Noteworthy is the version with serial communication and parallel nodes (denoted word-parallel/bit-serial), where the Caballero performance is merged with a slim-line communication. However, the bit-serial approach still has smallest footprint of all other implementations for both neighbourhoods. Going from 1- to 2-neighborhood has a marginal impact on the normalized performance for all approaches except Caballero, where the latency is almost 7 times higher!

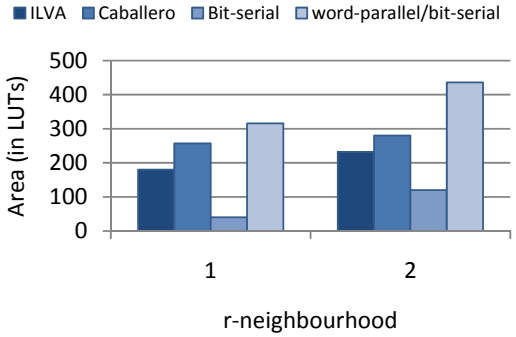


Figure 10.5 Area utilization for different neighbourhoods.

10.1 DISCUSSION

Connecting to the abstract execution models in section 4.2, we see that almost all architectures in the VIND spectrum benefits from the consumer node model only. In the producer node model, 8 distinct values are transmitted simultaneously equipped with different target addresses. This requires additional control compared to the consumer node where a single value is broadcasted with same source address. Apart from that, the bandwidth is doubled in the producer model which makes the interface eat up more logic. This kicks directly in Caballero but even Sleipner is in the danger zone. ILVA is the one that suffers less. Apparently, these architectures are typical consumers. The producer model makes sense only when the bandwidth is kept at absolute minimum, which is possible only in the bit-serial approach.

All state-flow and state-scan architecures are developed in VHDL using generic notations and configuration options. This creates in principle a high degree of portability, which unfortunately has not been exploited yet. Instead the parameters are instantiated for 8-bits. One reason is that the 18-bits multiplier

macros are tightly coupled to the Block RAM, which enforces the address space to be divided between the two macros. As the 8-bits parameters lead to larger internal values, a balance between input value size and fan-in per node is at stake. The other reason is that CNNs will in general work adequately with 8-bits parameters. Moving to a larger word width is feasible but it will not be easy without major structural changes. Most important, it will require a not fully parallel arithmetic. For the moment, it is not clear whether this can be done without consequences to the current packing density and this will therefore require additional investigation.

Another issue for future research is the effect of architectural choices in the problem space. Of the many degrees of freedom, mentioned in section 4.3, two selections only have been experimented with. Many more are possible and will probably have different consequences in the efficiency of exploiting the local storage potential through the BRAMs. In both pipelined designs, ILVA and Sleipner, the handling of the image stripe sequences does not require much support but has a fundamental limitation in the number of iterations to find locally stable solutions. In the state-scan architecures the limitation on iteration count is removed though at the expense of the ease of handling the image stripes.

The discussed applications have shown to be very diverse in their implementation requirements. Image processing is by its nature very demanding, as the desired 'locality of operation' in the geometry domain does not agree with the underlying principles of the superscalar computing architecture [57]. This is caused by the temporal character of such architectures, where a small number of large resources are scheduled in time for optimal usage. Spatial architectures, where the process is divided over many small resources, provide an alternative, as illustrated in this thesis.

A further boost in performance can be derived from the use of complex templates. TerBrugge gives an example of skeletonization, where the previously published hand-derived solutions can be mechanically improved to a really optimal, single template solution [40]. This is especially relevant to the pipelined approach, where the image is written back to external memory after every template application. Having fewer templates will then clearly raise performance.

One of the most important features of an FPGA is the innovation of partial reconfiguration of hard-wired modules. This innovation has always been kept in mind while designing the different approaches. Actually, the ability of dynamically reconfiguring (parts of) the FPGA is one of the main reasons for adopting the NoC-based architectures. For instance, in all implementations, as presented here, on-line programming can change the nature of the sensor. The integrated intelligence serves to extract knowledge from the image about physical conditions or objects that otherwise require a dedicated sensor. This virtualization of the sensory function is especially of advantage, where such measurements are only occasionally needed. Re-programming the sensor at need replaces the installation of a hardly used sensor, such as for diagnostics.

However, this ability has unfortunately never been tried! Future research is ought to take this feature into consideration.

Appendix A

Table A.1 Number of robust templates obtained for each boundary condition and precision level in the ascending approach. Boundary values [0.4, 1.0] are omitted as they do not result in any robust template

	5:2	5:3	5:4	5:5	5:6	5:7	5:8	5:9	5:10	5:11
-1.0	736	673	583	519	483	483	473	469	468	466
-0.9	976	877	764	692	648	648	638	634	632	630
-0.8	1264	1096	941	833	761	759	746	740	738	736
-0.7	1780	1509	1297	1145	1051	1049	1031	1025	1022	1018
-0.6	2344	1927	1627	1453	1347	1344	1323	1317	1314	1310
-0.5	3300	2730	2338	2113	1968	1963	1938	1929	1923	1919
-0.4	3572	2902	2457	2200	2028	2022	1997	1984	1978	1974
-0.3	4892	4034	3454	3140	2928	2921	2889	2876	2867	2861
-0.2	4892	4034	3454	3140	2928	2921	2889	2876	2867	2861
-0.1	5944	4993	4328	3955	3692	3682	3648	3631	3621	3615
0	5720	4512	3830	3414	3214	3214	3139	3131	3108	3108
0.1	5720	3484	2533	1997	1741	1677	1613	1593	1582	1582
0.2	3776	2006	1180	831	641	611	577	561	550	545
0.3	3256	642	268	137	104	91	73	73	72	72
0.4	1116	74	21	5	1	1	1	1	0	0
0.5	381	0	0	0	0	0	0	0	0	0

Table A.2 Number of robust templates obtained for each boundary condition and precision level in the descending approach. Boundary values [0.4, 1.0] are omitted as they do not result in any robust template.

	5:2	5:3	5:4	5:5	5:6	5:7	5:8	5:9	5:10	5:11
-1.0	466	2653	4221	5361	5538	5538	5538	5538	5538	5538
-0.9	630	3001	4634	5844	6162	6186	6187	6201	6205	6210
-0.8	736	3497	5314	6569	6833	6912	6957	6967	6983	6993
-0.7	1018	4071	5996	7275	7563	7646	7658	7696	7703	7705
-0.6	1310	4760	6671	7945	8262	8308	8320	8352	8366	8373
-0.5	1919	5453	7410	8623	8846	8932	8949	8949	8949	8949
-0.4	1974	5984	7875	9015	9287	9312	9342	9361	9361	9363
-0.3	2861	6636	8339	9382	9629	9655	9655	9685	9689	9690
-0.2	2861	7070	8663	9596	9841	9867	9867	9873	9873	9873
-0.1	3615	7406	8875	9716	9945	9951	9957	9966	9966	9966
0	3108	6274	7447	8224	8452	8452	8452	8452	8452	8452
0.1	1582	4238	5338	6097	6325	6325	6325	6325	6325	6325
0.2	545	2107	3105	3862	4090	4090	4090	4090	4090	4090
0.3	72	585	1262	2012	2240	2240	2240	2240	2240	2240
0.4	0	29	166	621	833	833	833	833	833	833
0.5	0	0	0	15	54	54	54	54	54	54

Table A.3 $|S_i \cup S_j|$ *in the ascending approache for precision <5:11> where* $i, j \in \{-1, +0.4\}$*. Same results are obtained for the descending approach with precision <5:2>.*

	-1	0.9	-0.8	-0.7	-0.6	-0.5	-0.4	-0.3	-0.2	-0.1	0	0.1	0.2	0.3	0.4
-1	466	630	736	1018	1310	1919	1974	2861	2861	3615	3276	1931	998	538	466
-0.9	630	630	736	1018	1310	1919	1974	2861	2861	3615	3350	2090	1162	702	630
-0.8	736	736	736	1018	1310	1919	1974	2861	2861	3615	3441	2196	1268	808	736
-0.7	1018	1018	1018	1018	1310	1919	1974	2861	2861	3615	3492	2365	1541	1090	1018
-0.6	1310	1310	1310	1310	1310	1919	1974	2861	2861	3615	3617	2630	1833	1382	1310
-0.5	1919	1919	1919	1919	1919	1919	1974	2861	2861	3615	3808	3024	2390	1990	1919
-0.4	1974	1974	1974	1974	1974	1974	1974	2861	2861	3615	3862	3079	2445	2045	1974
-0.3	2861	2861	2861	2861	2861	2861	2861	2861	2861	3615	4106	3568	3142	2889	2861
-0.2	2861	2861	2861	2861	2861	2861	2861	2861	2861	3615	4106	3568	3142	2889	2861
-0.1	3615	3615	3615	3615	3615	3615	3615	3615	3615	3615	4296	4063	3844	3642	3615
0	3276	3350	3441	3492	3617	3808	3862	4106	4106	4296	3108	3108	3184	3108	3108
0.1	1931	2090	2196	2365	2630	3024	3079	3568	3568	4063	3108	1582	1658	1582	1582
0.2	998	1162	1268	1541	1833	2390	2445	3142	3142	3844	3184	1658	545	545	545
0.3	538	702	808	1090	1382	1990	2045	2889	2889	3642	3108	1582	545	72	72
0.4	466	630	736	1018	1310	1919	1974	2861	2861	3615	3108	1582	545	72	0

Table A.4 $|S_i \cap S_j|$ *in the ascending approache for precision <5:11> where* $i, j \in \{-1, +0.4\}$*. Same results are obtained for the descending approach with precision <5:2>.*

	-1	0.9	-0.8	-0.7	-0.6	-0.5	-0.4	-0.3	-0.2	-0.1	0	0.1	0.2	0.3	0.4
-1	466	466	466	466	466	466	466	466	466	466	298	117	13	0	0
-0.9	466	630	630	630	630	630	630	630	630	630	388	122	13	0	0
-0.8	466	630	736	736	736	736	736	736	736	736	403	122	13	0	0
-0.7	466	630	736	101	101	101	101	101	101	101	634	235	22	0	0
-0.6	466	630	736	101	131	131	131	131	131	131	801	262	22	0	0
-0.5	466	630	736	101	131	191	191	191	191	191	121	477	74	1	0
-0.4	466	630	736	101	131	191	197	197	197	197	122	477	74	1	0
-0.3	466	630	736	101	131	191	197	286	286	286	186	875	264	44	0
-0.2	466	630	736	101	131	191	197	286	286	286	186	875	264	44	0
-0.1	466	630	736	101	131	191	197	286	286	361	242	113	316	45	0
0	298	388	403	634	801	121	122	186	186	242	310	158	469	72	0
0.1	117	122	122	235	262	477	477	875	875	113	158	158	469	72	0
0.2	13	13	13	22	22	74	74	264	264	316	469	469	545	72	0
0.3	0	0	0	0	0	1	1	44	44	45	72	72	72	72	0
0.4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table A.5 $|S_i \Delta S_j|$ *in the ascending approache for precision <5:11> where* $i, j \in \{-1, +0.4\}$*. Same results are obtained for the descending approach with precision <5:2>.*

	-1	0.9	-0.8	-0.7	-0.6	-0.5	-0.4	-0.3	-0.2	-0.1	0	0.1	0.2	0.3	0.4
-1	0	164	270	552	844	145	150	239	239	314	297	181	985	538	467
-0.9	164	0	106	388	680	128	134	223	223	298	296	196	114	702	631
-0.8	270	106	0	282	574	118	123	212	212	287	303	207	125	808	737
-0.7	552	388	282	0	292	901	956	184	184	259	285	213	151	109	101
-0.6	844	680	574	292	0	609	664	155	155	230	281	236	181	138	131
-0.5	145	128	118	901	609	0	55	942	942	169	258	254	231	198	192
-0.4	150	134	123	956	664	55	0	887	887	164	264	260	237	204	197
-0.3	239	223	212	184	155	942	887	0	0	754	224	269	287	284	286
-0.2	239	223	212	184	155	942	887	0	0	754	224	269	287	284	286
-0.1	314	298	287	259	230	169	164	754	754	0	186	292	352	359	361
0	297	296	303	285	281	258	264	224	224	186	0	152	271	303	310
0.1	181	196	207	213	236	254	260	269	269	292	152	0	118	151	158
0.2	985	114	125	151	181	231	237	287	287	352	271	118	0	473	546
0.3	538	702	808	109	138	198	204	284	284	359	303	151	473	0	73
0.4	467	631	737	101	131	192	197	286	286	361	310	158	546	73	0

Bibliography

[1] J. C. Russ. *The Image Processing Handbook*. Boca Raton, CRC, 4th edition, 2002.

[2] C. Bobda. *Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications*. Springer, e-ISBN 978-1-4020-6100-4, 2007.

[3] J. A. Kahle et al. (Jul./Sep. 2005). "Introduction to the Cell multiprocessor." IBM J. RES. & DEV. [online]. vol. 49, no. 4/5. Available: http://www.research.ibm.com/journal/rd/494/kahle.pdf [Sep. 18, 2008]

[4] D. R. W. Barr and P. Dudek "A Cellular Processor Array Simulation and Hardware Prototyping Tool," in *Proc. 11th IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA2008)*, Santiago de Compostela, Spain, 2008, pp. 213-218.

[5] P. Dudek and P. Hicks. "A General-Purpose Processor-per-Pixel Analog SIMD Vision Chip." *IEEE Transactions on Circuits and Systems – I: Regular papers*, vol. 52, no. 1, pp. 13-20, Jan. 2005.

[6] D. Soudris (editor). *Fine- and Coarse-grain reconfigurable computing*. Springer, e-ISBN: 978-1-4020-6505-7, 2007.

[7] L.O. Chua and L. Yang. "Cellular Neural Networks: Theory." *IEEE Transactions on Circuits and Systems*, vol. 35, no. 10, pp. 1257-1272, 1273-1280, 1988.

[8] S. Haykin. *Neural Networks: a comprehensive foundation*. Prentice Hall, second edition, 1999.

[9] F. Moraes. "A low area overhead packet-switched network on chip: architecture and prototyping," in *Proc. IFIP Conference on Very Large Scale Integration*, Darmstadt, Germany, 2003, pp. 318-323.

[10] J. Kari. "Theory of cellular automata: A survey." Elsevier, *Theoretical Computer Science*, vol. 334, pp. 3-33, 2005.

[11] L.O. Chua. *CNN: A Paradigm for Complexity*. World Scientific Series on Nonlinear Science, Series A, vol. 31, ISBN 981-02-3483-X, 1998.

[12] L. O. Chua and T. Roska. *Cellular neural networks and visual computing – Foundation and applications*. Cambridge University Press, ISBN 13 978-0-521-01863-0, 2002.

[13] A. Lundgren. "Design of a Co-processor that Implements Several Specific Smart Imaging Algorithms." M.Sc. thesis, Lund University, Lund, Sweden, 2004.

[14] G. Manganaro, P. Arena and L. Fortuna. *Cellular Neural Networks, Chaos, Complexity and VLSI Processing*. Springer-Verlag, Berlin Heidelberg, 1999.

[15] T. Roska and L. O. Chua. "Cellular Neural Networks with non-linear and delay-type template elements and non-uniform grids." *International Journal of circuit theory and applications*, vol. 20, pp. 469-481, 1992.

[16] L. Yang, L. O. Chua and K. R. Krieg. "VLSI Implementation of Cellular Neural Networks," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS-90)*, vol.3, 1990, pp. 2425-2427.

[17] T. Roska et al. "A hardware accelerator board for Cellular Neural Networks: CNN-HAC," in *Proc. IEEE Int. Workshop on Cellular Neural Networks and their Applications (CNNA-90)*, 1990, pp. 160-168.

[18] Second Generation TMS320 User's Guide. Texas Instruments Inc., 1989. Internet: http://www.ti.com [Sep. 18, 2008].

[19] T. Roska, L.O. Chua. "The CNN universal machine: An analogic array computer." *IEEE Transactions on Circuits and Systems—Part II: Analog and Digital Signal Processing*, vol. 40, no. 3, pp. 163-173, 1993.

[20] L.O. Chua and T. Roska. "The CNN universal machine – Part I: The architecture," in *Proc. 2nd IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA-92*, 1992*)*, pp. 1-10.

[21] T. Roska et al. "On a CNN chip-prototyping system," in *Proc. of 3rd IEEE International Workshop on Cellular Neural Networks and their Applications (CNN-94)*, Rome, 1994, pp. 378-379.

[22] B. Fehér et al.. "ACE: A digital Floating Point CNN Emulator Engine," in *Proc. Fourth IEEE International Workshop on Cellular Neural Networks and their Applications (CNN-96)*, Seville, Spain, 1996, pp. 273- 278.

[23] R. Domínguez-Castro, S. Espejo, A. Rodríguez-Vázquez and R. Carmona. "A CNN Universal Chip in CMOS technology," in *Proc. 3rd IEEE Int. Workshop on Cellular Neural Networks and their Applications (CNN-94)*, Rome, Dec. 1994, pp. 91-96.

[24] R. Domínguez-Castro et al. "A 0.8 μm CMOS Two-Dimensional Programmable Mixed-Signal Focal-Plane Array Processor with On-Chip Binary Imaging and Instructions Storage." *IEEE Journal of Solid-State Circuits*, vol. 32, no. 7, pp. 1013-1026, Jul. 1997.

[25] S. Espejo, R. Domínguez-Castro, G. Liñán and A. Rodríguez-Vázquez. "A 64x64 CNN Universal Chip with analog and digital I/O," in *Proc. IEEE International Conference on Electronics, Circuits and Systems (ICECS98),* Lisbon, 1998, pp. 203-206.

[26] G. Liñán, S. Espejo, R. Domínguez-Castro, E. Roca and A. Rodríguez-Vázquez. "CNNUC3: A Mixed-Signal 64 x 64 CNN Universal Chip," in *Proc. 7th Int. Conf. on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems (MicroNeuro-99*, 1999*)*, pp. 61-68.

[27] G. Liñán, S. Espejo, R. Domínguez-Castro and A. Rodríguez-Vázquez. "ACE4k: An analog I/O 64x64 visual microprocessor chip with 7-bit analog accuracy." *International Journal of Circuit Theory and Applications*, vol. 30, 2002, pp. 89-116.

[28] A. Rodríguez-Vázquez et al. "ACE16k: The Third Generation of Mixed-Signal SIMD-CNN ACE Chips Toward VSoCs" *IEEE Transactions on Circuits and Systems Part I: Regular Papers*, vol. 51, no. 5, pp. 851–863, 2004.

[29] G. Liñán, A. Rodríguez-Vázquez, S. Espejo, and R. Domínguez-Castro. "ACE16k: A128x128 Focal Plane Analog Processor with Digital I/O," in *Proc. 7th IEEE Int. Workshop on Cellular Neural Networks and their Applications (CNN-02*, 2002. *)*, pp.132-139

[30] G. Liñán, S. Espejo, R. Domínguez-Castro, E. Roca and A. Rodríguez-Vázquez. "A 0.5 μm CMOS 10^6 transistors Analog Programmable Array Processor for Real-Time Image Processing," in *Proc. 25th European Solid-State Circuits Conference (ESSCIRC-99)*, 1999, pp. 358-361.

[31] A. Rodríguez-Vázquez, S. Espejo, R. Domínguez-Castro and J. L. Huertas. "Current-Mode Techniques for the Implementation of Continuous- and Discrete-Time Cellular Neural Networks." *IEEE Transactions on Circuits and Systems—II: Analog and Digital Signal Processing*, vol. 40, no. 3, pp. 132-146, Mar. 1993.

[32] S. Espejo, A. Rodríguez-Vázquez, R. Domínguez-Castro and R. Carmona. "Convergence and Stability of the FSR CNN Model," in *Proc. 3rd IEEE Int. Workshop on Cellular Neural Networks and their Applications (CNNA-94*, Rome, 1994*)*, pp. 411-416.

[33] C. Koch and H. Li. *Vision Chips: Implementing Vision Algorithms Using Analog VLSI Circuits*. New York: IEEE Press, 1994.

[34] T. Delbrück and C. A. Mead. "Analog VLSI phototransduction by continuous-time, adaptive, logarithmic photoreceptor circuits," Cal. Inst. Technol., Computation and Neural Systems Program, Tech. Rep., CNS Memo, no. 30, May 1994.

[35] Á. Zarándy et al.. "An emulated digital architecture implementing the CNN Universal Machine," in *Proc. 5th International Workshop on Cellular Neural*

Networks and their Applications (CNNA-98), London, England, 1998, pp. 249-252.

[36] P. Keresztes et al. "An Emulated Digital CNN Implementation." *Journal of VLSI Signal Processing,* 23, pp. 291-303, 1999.

[37] S. Zöld. "CNN Alpha Language and Compiler," Report DNS-10-1997, Computer and Automation Research Institute, Budapest, 1997.

[38] Xilinx Inc., Xilinx Products Homepage, Xilinx products. [Online]. Available: http://www.xilinx.com [Sep. 18, 2008].

[39] H. Harrer and J.A. Nossek. "Discrete-Time Cellular Neural Networks," *International Journal of Circuit theory and Applications*, vol. 20, pp. 453-467, 1992.

[40] M.H. terBrugge. "Morphological Design of Discrete-Time Cellular Neural Networks." Ph.D. thesis, Rijksuniversiteit Groningen, Groningen, The Netherlands, 2005.

[41] H. Harrer. "Discrete-Time Cellular Neural Networks." Ph.D. thesis, Verlag Shaker, ISBN 3-86111-286-8, Aachen, 1992.

[42] Z. Nagy and P. Szolgay. "Configurable Multilayer CNN-UM Emulator on FPGA." *IEEE Transactions on Circuits and Systems –I: Fundamental Theory and Applications,* vol. 50, no. 6, pp.774 – 778, June 2003.

[43] Z. Vörösházi, Z. Nagy, A. Kiss and P. Szolgay. "An embedded CNN-UM Global Analogic Programming Unit Implementation on FPGA," in *Proc. 10th International Workshop on Cellular Neural Networks and Their Applications*, Istanbul, Turkey, Aug. 2006, pp. 1-5.

[44] A. deHon. "Reconfigurable Architectures for General-Purpose Computing," Ph.D. Thesis, MIT, Cambridge (USA), 1996.

[45] J. Villasenor, C. Jones and B. Schoner. "Video Communications Using Rapidly Reconfigurable Hardware." *IEEE Transactions on Circuits and Systems for Video Processing*, vol. 5, pp. 565-567, Dec. 1995.

[46] Memec Design. "Virtex-II Pro FF 1152." Internet: http://www.memec.com/uploaded/VirtexIIPro_FF1152_1.pdf, [Jun. 15, 2006].

[47] T. Roska "Computational and Computer Complexity of Analogic Cellular Wave Computers," in *Proc. 7th IEEE Workshop on CNNs and their Applications*, R. Tetzlaff (ed.), World Scientific (Singapore), pp. 323-338, 2002.

[48] M.H. terBrugge et al. "CNN Applications in toll driving." *Journal of VLSI Signal Processing*, vol. 23, no. 2/3, pp. 465-477, 1999.

[49] S. Malki. "Discrete-Time Cellular Neural Networks Implemented on Field-Programmable Gate-Arrays to Build a Virtual Sensor System." Lic. thesis, Lund University, Lund, ISBN 91-7167-040-8, 2006, 98 pages.

[50] S. Malki and L. Spaanenburg. "Efficiency Considerations for DT-CNN Hardware," in *Proc. IEEE Northeast Workshop on Circuits and Systems (NEWCAS'07)*, Montréal, Canada, 2007, pp. 1038-1041.

[51] S. Malki, Y. Fuqiang and L. Spaanenburg. "Vein Feature Extraction Using DT-CNNs," in *Proc. 10th International Workshop on Cellular Neural Networks and Their Applications (CNNA'06)*, Istanbul, Turkey, 2006, pp. 307-312.

[52] S. Malki, G. Deepak, V. Mohanna, M. Ringhofer and L. Spaanenburg. "Velocity Measurement by a Vision Sensor," in *Proc. IEEE International Conference on Computational Intelligence for Measurement Systems and Applications (CIMSA'06)*, La Coruna, Spain, 2006, pp.135-140.

[53] M. Gardner. "Mathematical Games: The fantastic combinations of John Conway's new solitaire game "life"." *Scientific American*, vol. 223, pp. 120-123, 1970.

[54] P. Julian. "Simplicial RTD-based cellular nonlinear networks." *IEEE Transaction on Circuits and Systems - part I*, vol. 50, no. 4, pp. 500-509, 2003.

[55] P. Dudek and P.J. Hicks. "A general-purpose CMOS vision chip with a Processor-per-pixel SIMD array," in *Proc. ESSCIRC'01*, Villach (Austria), 2001, pp. 228-231.

[56] Trenz Electronic. Internet: www.trenz-electronic.de, [Sep. 18, 2008].

[57] N. Tredennick and B. Shimamoto. "Go Reconfigure", *IEEE Spectrum*, vol. 40, no. 12, pp. 36-40, 2003.

[58] H.F. Durrant-Whyte. "Sensor Fusion: when more means better." In: K.T.V. Grattan (ed.) Sensors: technology, systems and applications, Adam Hilger, Bristol, 1991.

[59] Analogical and Neural Computing Laboratory. Computer and Automation Research Institute of the Hungarian Academy of Science, MTA-*SzTAKI*, Budapest, Hungary. Intenet: http://lab.analogic.sztaki.hu, 2003-2005 [2006-05-16].

[60] W. Fang, C. Wang, and L. Spaanenburg, "In Search for a Robust Digital CNN System," in *Proc. 10th IEEE Workshop on CNNA and their Applications*, Istanbul, Turkey, 2006, pp. 328 – 333.

[61] B. Mirzai, D. Lím and G. S. Moschytz. "Robust CNN Templates: Theory and Simulation," in *Proc. Fourth IEEE International Workshop on Cellular Neural Networks and their Applications*, Seville, Spain, 1996, pp. 393-398.

[62] Á. Zarándy. "The Art of CNN Template Design." *Int. J. Circuit Theory and Applications*, vol. 27, no. 1, pp. 5-23, 1999.

[63] J. A. Nossek. "Design and Learning with Cellular Neural Networks." *Int. J. Circuit Theory and Applications*, vol. 24, no. 1, pp. 15-24, 1996.

[64] T. Kozek, T. Roska and L.O. Chua. "Genetic Algorithm for CNN Template Learning." *IEEE Trans. on Circuits and Systems – I*, vol. 40, no. 6, pp. 392-402, 1993.

[65] M. Hänggi. "On Locally Regular Cellular Neural Networks." *IEEE Trans. on Circuits and Systems – I: Fundamental Theory and Applications*, vol. 48, no. 5, pp. 513-520, 2001.

[66] D. Lím and G. S. Moschytz. "A programmable gm-C CNN implementation." *IEEE Int. Workshop on Cellular Neural Networks and their Applications*, V. Tavsanoglu (Ed.), pp. 294-299, London, 1998, pp. 294-299.

[67] M. Hänggi and G. S. Moschytz. "Analytic and VLSI Specific Design of Robust CNN Templates." *J. VLSI Signal Processing*, 23, pp. 415-417, 1999.

[68] S. Xavier-de-Souza, M. Yalcin and J. Suykens. "Toward CNN Chip-Specific Robustness." *IEEE transactions on Circuits and Systems – I: Regular Papers*, vol. 51, no. 5, pp. 892-902, 2004.

[69] C. Merkwirth et al. "Finite Iteration DT-CNN – New Design and Operating Principle," in *Proc. ISCAS*, Vancouver, Canada, 2004, pp. 504-507.

[70] J. Wichard, M. Ogorzalek and C. Merkwirth. "Peformance of Finite Iteration DTCNN with Truncated Stationary Templates," in *Proceedings ISCAS*, 2005, pp. 4657-4660.

[71] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1999.

[72] J. M. Cruz, L. O. Chua and T. Roska. "A Fast, Complex, and Efficient Test Implementation of the CNN Universal Machine," in *Proc. IEEE Int. Workshop on Cellular Neural Networks and their Applications (CNNA-94)*, Italy, 1994, pp. 61-66.

[73] Ingber L. "Adaptive Simulated Annealing (ASA) version 24.1" Internet: http://www.ingber.com, [Sep. 17, 2008].

[74] V. Brea et al. "A One Quadrant Discrete-Time Cellular Neural Network Architecture for Pixel-Level Snakes," in *Proc. ISCAS*, Kobe, Japan, pp. 3922-3925, 2005.

[75] Z. Vörösházi, Z. Nagy and P. Szolgay. "An advanced emulated digital retina model on FPGA to implement a real-time test environment," in *Proc. ISCAS*, Kos, Greece, 2006, pp. 1949-1952.

[76] Chr. Niederhöfer and R. Tetzlaff. "Detection of a preseizure state in epilepsy," in *Proc. ISCAS*, Kos, Greece, pp. 165-168, 2006.

[77] P. Thiran. "Influence of Boundary Conditions on the Behavior of Cellular Neural Networks." *IEEE trans. on Circuits and Systems –I: Fundamental Theory and Applications*, vol. 40, no. 3, pp. 207-212, Mar. 1993.

[78] E. Planas. "CNN Template Optimization." M.Sc. Thesis, Lund University, Lund, Sweden. 2007.

[79] J. Flak, M. Laiho and A. Paasio. "Scalable fault-tolerant logic system based on regular array of locally interconnected gates," in Proc. 11th Int. Workshop on Cellular Neural Networks and their Applications (CNNA2008), Santiago de Compostela, Spain, Jul. 2008, pp. 116-119.

[80] P. Földesy et al. "Fault-Tolerant Design of Analogic CNN Templates and Algorithms — Part I: The Binary Output Case." *IEEE Transactions on Circuits and Systems – I: Fundamental Theory and Applications*, vol. 46, no. 2, pp. 312-322, 1999.

[81] J. L. Hennessy and D. A. Patterson. *Computer architecture: A quantitative approach*. Morgan Kaufmann, 4th edition, 2007.

[82] H. Corporaal. "Transport triggered architectures used for Embedded Systems." Internet: http://www.elis.rug.ac.be/ntca/announcement.html#Corporaal, Dec. 16, 1999 [Sept. 18, 2008].

[83] K. Goossens, J. Dielissen and A. Radulescu. "The Æthereal Network on Chip: concepts, architectures, and implementations." *IEEE Design & Test of Computers*, 22(5), pp. 21-31, 2005.

[84] V. Zhirnov, R. Cavin, G. Leeming and K. Galatsis. "An Assessment of Integrated Digital Cellular Automate Architectures." *IEEE Computer*, vol. 41, no. 1, pp. 38 – 44, 2008.

[85] D.A. Patterson et al., "A Case for Intelligent RAM: IRAM." *IEEE Micro*, vol. 17, no. 2, pp. 34 – 44, 1993.

[86] B. Khailany et al. "Imagine: media processing with streams." *IEEE Micro*, vol. 21, no. , pp. 35 – 462, 2001.

[87] L. Benini and G. De Micheli. "Networks on Chip: A new SoC Paradigm." *IEEE Computer*, 35(1), pp. 70-80, 2002.

[88] D. Wiklund and D. Liu. "SoCBUS: Circuit-switched Network on Chip for Hard Real Time Embedded Systems," in *Proc. Int. Parallel and Distributed Syposium,* 2003, pp. 78-85.

[89] A Zarandy and C. Rekeczky. "Bi-i: a stand-alone ultra high-speed cellular vision system." *IEEE Circuits and Systems Magazine*, vol. 5, no. 2, pp. 36 – 45, 2005.

[90] JEDEC Solid State Technology Association, "Double Data Rate (DDR) SDRAM Specification". JESD79E. Internet: http://www.jedec.org/download/search/JESD79E.pdf, 2005 [Sep. 21, 2008].

[91] Benny Åkesson. "An analytical model for a memory controller offering hard real-time guarantees." M.Sc. thesis, Lund University, Lund, Sweden, 2005.

[92] Press Release, Precise Biometrics. Internet: http://cws.huginonline.com/P/131387/PR/200112/844226_5.html, Mar. 2006 [Sep. 18, 2008]

[93] K. Munro "Biometrics: attack of the clones." *Infosecurity Today*, vol. 3, issue. 1, pp. 45, Jan./Feb. 2006.

[94] S.-K. Im et al. "A Biometric Identification System by Extracting Hand Vein Patterns." *Journal of the Korean Physical Society*, vol. 38, no. 3, pp. 268-272, Mar. 2001.

[95] Hitachi Engineering Co. Ltd. "About Finger Vein." Internet: http://www.hitachi-hec.co.jp/english/about_fv.htm, [Mar. 20, 2006]

[96] Jean-François Mainguet. "Vein, Vascular pattern." Internet: http://perso.orange.fr/fingerchip/biometrics/types/vein.htm, [Sep. 18, 2008].

[97] L. Wang and A. Bhalerao. "Detecting branching structures using local Gaussian models." in *Proc. IEEE Symposium on Biomedical Imaging*, 2002, pp. 161-164.

[98] Q Gao and G. S. Moschytz. "Fingerprint Feature Extraction Using CNNs," in *Proc. European Conference on Circuit Theory and Design*, Espoo, Finland, 2001, pp. 97-100.

[99] Q Gao, P. Förster, K. R. Möbus and G. S. Moschytz "Fingerprint Recognition Using CNNs: Fingerprint Preprocessing," in *Proc. IEEE International Symposium on Circuits and Systems*, vol. 2, 2001, pp. 433-436.

[100] Q Gao and G. S. Moschytz. "Fingerprint Feature Matching Using CNNs," in *Proc. ISCAS*, 2004, pp. 73-76.

[101] A. Jain, R. Bolle and S. Pankanti. *Biometrics: Personal Identification in Networked Society*, Kluwer Academic Publishers, 1999.

[102] Tamás Roska et al. "CNN software Library (templates and algorithms), vers. 7.3," *Tech. Rep. DNS-CADET-15*, Analogical and Neural Computing Laboratory, Computer and Automation Research Institute, Hungarian Academy of Science, 1999.

[103] C. Grunditz, M. Walder, and L. Spaanenburg. "Constructing a neural system for surface inspection," in *Proc. IJCNN*, vol. III, Budapest, July 2004, pp. 1881 - 1886.

[104] E. Alpaydin and P. Marchal. "Why an 'A' is an 'A'," in *Proc. Journées d'Électronique*, Lausanne, Switzerland, 1989, pp. 88-103.

[105] H. Fatemi et al. "Real-Time Face Recognition on a mixed SIMD VLIW Architecture," in *Proc. Progress*, Nieuwegein, The Netherlands, Oct. 2003, pp. 78-83.

[106] Á. Rodríguez-Vázquez et al. (2008) "The Eye-RIS CMOS Vision System," in *Analog Circuit Design*. [online]. Springer Netherlands, pp. 15-32 Available: http://www.springerlink.com/content/w717717wnltr63r3/ [Sep. 18, 2008].

[107] G. Grassi and L. A. Grieco. "Object-oriented image analysis via analogic CNN algorithms- part I: Motion Estimation," in *Proc. of 7th IEEE Int. Workshop on CNNs and Their Applications*, Frankfurt/M, Germany, 2002, pp. 172-179.

[108] P. Arena et al. "Complexity in Two-Layer CNN," in *Proc. Fourth IEEE International Workshop on CNNs and their Applications*, Seville, Spain, 1996, pp. 127-132.

[109] A. Stoffels, T. Roska and L. O. Chua. "An Object-Oriented Approach to Video Coding via the CNN Universal Machine," in *Proc. Fourth IEEE International Workshop on CNNs and their Applications*, Seville, Spain, 1996, pp. 13-18.

[110] R. P. Grimaldi. *Discrete and Combinatorial Mathematics: An Applied Introduction*. 3rd addition, Addison-Wesley Publishing Company Inc, ISBN 0-201-60044-7, 1994.

[111] Z. Nagy, Z. Vörösházi and P. Szolgay. "Emulated digital CNN-UM solution of partial differential equations." *Int. Journal of Circuit Theory and Applications*, no 34, pp. 445-470, 2006.

[112] I. Koren. *Computer Arithmetic Algorithms*. University of Massachusetts, Amherst, A K Peters Ltd., 2002, ISBN 1568811608.