UNIVERSIDAD DE SEVILLA

DOCTORAL THESIS

Neuromorphic Deep Convolutional Neural Network Learning Systems for FPGA in Real Time.

Author:
Ricardo Tapiador Morales

Supervisor:
Dr. Gabriel Jiménez Moreno
Dr. Alejandro Linares Barranco
Dr Ángel F. Jiménez Fernandez

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Robotics and Computer Technology Lab.
Departamento de Arquitectura y Tecnología de Computadores.

Declaration of Authorship

I, Ricardo Tapiador Morales, declare that this thesis entitled "Neuromorphic Deep Convolutional Neural Network Learning Systems for FPGA in Real Time." and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UNIVERSIDAD DE SEVILLA

Abstract

Escuela Técnica Superior de Ingeniería Informática.
Departamento de Arquitectura y Tecnología de Computadores.

Doctor of Philosophy

Neuromorphic Deep Convolutional Neural Network Learning Systems for FPGA in Real Time.

by
Ricardo Tapiador Morales

In this work, it is intended to advance on the knowledge of Deep Learning and pattern recognition and their hardware implementations, such as hardware architectures for frame-based and event-based processing paradigms.

First, frame-based vision systems are studied, along with Deep Learning algorithms for feature extraction in vision sensors, particularly Convolutional Neural Networks. Based on these studies, an FPGA OpenCL implementation for real-time frame-based CNN inference is proposed. Additionally, a second CNN accelerator based on a SoC-FPGA platform called NullHop is presented. Both designs are analyzed in terms of accuracy, latency, power consumption and area.

Next, biological neuron behaviour and neuromorphic vision sensors are studied, in order to adapt the CNN algorithm to the spiking domain. With the aim of inferring Spiking Convolutional Neural Networks, three different spiking convolution processors are presented. These spiking processors are inspired in leakage integrate and fire neurons and they perform the convolution operation with the arrival of events from a neuromorphic vision sensor. The three convolution processors proposed were tested with different stimuli to determine their behaviour and measure their performance..

Finally, a new event-based algorithm for feature extraction, called HOTS, was used to recognize gestures from a neuromorphic vision sensor. The HOTS algorithm uses a novel concept called time-surface, which represents the spatio-temporal activity of events to extract features from events. From this approach, a novel architecture for a VLSI system to infer the HOTS algorithm is presented with a testbench to characterize the system. In addition, a new memory model to reduce memory accesses and memory resources consumption was proposed and tested in a HOTS network.

Lastly, the performance and efficiency of these systems were evaluated. Then, the conclusions obtained are presented and new features and improvements are proposed for future works.

Agradecimientos

El desarrollo de esta tesis, ha sido sin lugar a dudas uno de los retos más difíciles a los que me he enfrentado. Han sido tres años llenos de experiencias, dificultades y altibajos, que me han hecho avanzar tanto a nivel profesional como personal. Sin embargo, he tenido la suerte de contar con personas que me han ayudado en el camino, motivándome, apoyándome y enseñándome. Estas páginas están dedicada a esas personas.

Me gustaría empezar agradeciendo a mis directores de tesis, Gabriel Jiménez Moreno, Alejandro Linares Barranco y Ángel Jiménez Fernández, por ayudarme en los problemas encontrados a lo largo de la tesis, aconsejándome y guiándome en cada paso que daba con sus sabios consejos, formándome como científico e investigador. Agradecer en particular la labor de uno de mis directores de tesis, Alejandro Linares Barranco por confiar y creer en mí, dándome la oportunidad de entrar y colaborar en el departamento de Arquitectura y Tecnología de Computadores donde he vivido incontables experiencias que jamás olvidare.

Agradecer a los miembros del grupo de Robótica y Tecnología de Computadores por estar a mi lado estos años haciendo de mi lugar de trabajo, un segundo hogar. Gracias a Manuel Domínguez Morales, Elena Cerezuela, Antón Civit Balcells, José Luis Sevillano Ramos, Daniel Cascado Caballero, Francisco Gómez Rodríguez, Lourdes Miró Amarante, Fernando Díaz del Río, Francisco Luna Perejón junto al resto de miembros del grupo de investigación.

Destacar la suerte de tener unos compañeros de trabajo a los que puedo llamar amigos, estas líneas van dedicadas a Antonio Ríos Navarro, Juan Pedro Domínguez Morales y Daniel Gutiérrez Galán. Con los cuales he tenido suerte de compartir muchas experiencias como estancias de investigación, viajes y alguna que otra barbacoa. Muchas gracias por vuestro apoyo, habéis sido de gran ayuda. Muchas gracias al equipo de electrónicos del departamento formado por Alberto Vázquez Baeza y Juan Manuel Montes, quienes siempre se han prestado a ayudarme y me han dado su apoyo en todo momento.

En el ámbito personal agradecer a mis amigos Pedro Castañeda Fernández, Juan Manuel Muñoz Noguera, María Jiménez Fernández, Cristina Santos Serra, Isabel Maria Beasley Bohorquez, Laura Cano García junto al resto del grupo por las partidas de juegos de mesa, conversaciones y múltiples salidas que me han ayudado a despejarme y animarme.

En lo familiar, agradezco a mis abuelos Manuel Morales Guerrero e Isabel Salguero Becerro, quienes me han cuidado desde pequeño viéndome crecer y convertirme en la persona que soy ahora.

Mención especial a Narciso Reguera Rodríguez y Dolores Valenzuela Díaz, a quien tengo el honor de llamarles mis segundos padres, estando junto a mí en los buenos y en los malos momentos tratándome como si fuera su hijo. Gracias de todo corazón.

A Natalia Martin Saborido, quien me ha acompañado durante gran parte de este camino dándome su apoyo, cariño, aportando su gran creatividad a mi vida. Muchísimas gracias.

Por último, las personas más importantes en mi vida a los que le dedico este trabajo, mis padres y mi hermana, Ricardo Tapiador Domínguez, María Isabel Morales Salguero y Cristina Tapiador Morales. Gracias por escucharme y aguantar mis altibajos, por educarme e inculcarme el valor del esfuerzo y la constancia, las cuales han sido clave en mi vida. Muchas gracias familia por todo.

Contents

List of Figures

List of Tables

List of Abbreviations

AER	**A**ddress **E**vent **R**epresentation
ALU	**A**rithmetic **L**ogic **U**nit
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit
ATIS	**A**synchronous **T**ime-based **I**mage **S**ensor
AVPT	**Av**erage **P**rocessing **T**ime
AXI	**A**dvanced e**X**tensible **I**nterface
BRAM	**B**lock **R**am **M**emory
CCM	**C**omputer **C**ore **M**odule
Chsum	**C**hannel **S**um
CMOS	**C**omplementary **M**etal **O**xide **S**emiconductor
CNN	**C**onvolutional **N**eural **N**etwork
ConvNet	**C**onvolutional **N**eural **N**etwork
CU	**C**omputer **U**nit
DSP	**D**igital **S**ignal **P**rocessor
DVS	**D**ynamic **V**ision **S**ensor
EDE	**E**uclidean **D**istance **E**stimator
EPSP	**E**xcitatory **P**ostsynaptic **P**otential
FPGA	**F**ield **P**rogrammable **F**ields **A**rrays
GPU	**G**raphics **P**rocessing **U**nit
HGCM	**H**istograms **C**omparator **M**odule (HGCM)
HPC	**H**igh **P**erformance **C**omputing
HOTS	**H**ierarchy **o**f **T**ime **S**urfaces
IDP	**I**nput **D**ecoding **P**rocessor
IE	**I**nput **E**vent
INI	**I**nstitute of **N**euroinformatics of **Z**urich
IPSP	**I**nhibitory **P**ostsynaptic **P**otential
LC	**L**eakage **C**ounter
LGN	**L**ateral **G**eniculate **N**ucleus
LIF	**L**eakage **I**ntegrate and **F**ire
LLM	**L**eakage **L**UTRAM **M**emory
LM	**L**eakage **M**ask
LTB	**L**eakage **T**imestamp **B**ank
LUTRAM	**L**ook**U**p **T**able **RAM**
LUT	**L**ook**U**p **T**able
MLS	**L**ayer **M**ask **S**elector
MPB	**M**embrane **P**otential **B**ank
NTH	**C**onvolution **I**dentification
NTH	**N**egative **T**hreshold
PCI	**P**eripheral **C**omponent **I**nterconect
PE	**P**rocessing **E**lements
PL	**P**rogrammable **L**ogic
PRE	**P**ooling **R**eLU **E**ncoding **M**odule
PS	**P**rocessing **S**ystem

PTH	**P**ositive **T**hreshold
PT	**P**rocessing **T**ime
RAM	**R**andom **A**ccess **M**emory
RCT	**R**efractory **C**ounter **T**ime
RC	**R**efractory **C**ounter
RLM	**R**efractory **L**UTRAM **M**emory
RM	**R**efractory **M**ask
ROI	**R**egion **o**f **I**nterest
RPV	**R**efractory **P**eriod **V**alue
RRAM	**R**esistive **R**andom **A**ccess **M**emory
RTB	**R**efractory **T**imestamps **B**ank
RTC	**R**obotics and **T**echnology of **C**omputers **L**ab
SCNN	**S**piking **C**onvolutional **N**eural **N**etworks
SC	**S**patial **C**ontrast
SIMD	**S**ingle **I**nstruction **M**ultiple **D**ata
SLICEMEM	**S**lice **Mem**ory
SoC	**S**ystem **o**n **C**hip
SoP	**S**um **o**f **P**roducts
SRDP	**S**pike **R**ate **D**ependent **P**lasticity
STDP	**S**pike **T**iming **D**ependent **P**lasticity
TC	**T**emporal **C**ontrast
TDV	**T**iming **D**iference **V**alue
UT	**U**pdate **T**ime
VLSI	**V**ery **L**arge **S**cale **I**ntegration

Dedicated to those who have supported and helped me throughout my life.

Chapter 1

Introduction

"Nothing is stronger than an idea
whose time has come."

Victor Hugo

Humans have searched for artificial solutions to solve different problems since their origin. From sea navigation in ancient times to the recent space conquest, humankind has adapted to the environment and evolved with it. Several inventions were inspired in how nature solves complex tasks in an efficient way, in order to use similar mechanisms as seen in nature with the aim of solving all sorts of problems. An example of this inspiration can be observed in inventions such as the wings of an airplane, which are inspired by birds. Curiosity for the unknown has opened new paradigms and developed smart solutions that make our life easier and better, bringing humankind to new areas of knowledge.

The development of sensors that collect data in real time from the environment has become one of the most important advances, since these sensors provide precise and useful information. Several research fields are focused on using the information given by image or audio sensors with the aim of providing machines with intelligence to solve complex problems efficiently, with high precision and in an autonomous way.

This kind of advances can be seen in factory robots that perform multiple risky tasks repetitively along an assembly line, with enough intelligence to work in an autonomous way. However, these advances are not limited to robotics. In recent years, several electronic devices, such as smartphones, can extract features from our photos with a low percent of error, or reduce the ambient sound when we receive a phone call. Currently, the tendency in the electronics market is the development of smart devices with any kind of intelligence that helps us with several daily tasks, from a smart speaker, which can place an order to a supermarket, to autonomous driving cars.

There exist several learning techniques in the literature; one of the most used is artificial neural networks (**ANN**). This kind of network consists of a group of neurons connected to each other in order to generate a specific output from an input stimulus. In recent years, these neural networks have become increasingly popular and several applications can be found in the literature. Most of the problems to be solved by this kind of algorithm are related to pattern matching and pattern recognition, i.e., predicting an output from input data. Within this processing paradigm, there are different algorithms which take inspiration from this concept, such as convolutional neural networks (**CNN**), which are ideal to extract semantic information from input data, and spiking neural networks (**SNN**), which take inspiration from the behaviour of biological neurons.

In the 1980s, a new research field called Neuromorphic Engineering emerged. The purpose of neuromorphic engineering is to study the biological and inner systems of neural processing with the aim of designing analog, digital or mixed-signal systems that can perform tasks, replicating similar behaviours and properties of biological neurons. One of the goals of Neuromorphic Engineering is to explain how our brain is able to learn and perform complex tasks with high efficiency. From this approach, several sensors and systems have been developed, such as vision sensors (Lichtsteiner, Posch, and Delbrück, 2008, Posch, Matolin, and Wohlgenannt, 2011, Serrano-Gotarredona and Linares-Barranco, 2013), auditory sensors (Yang et al., 2016, Jiménez-Fernández et al., 2017), robotics systems (Gomez-Rodriguez et al., 2007, Gómez-Rodríguez et al., 2016, Perez-Peña, Linares-Barranco, and Chicca, 2014, Rea, Metta, and Bartolozzi, 2013), neuromorphic processing chips (Akopyan et al., 2015, Furber et al., 2013, Schmitt et al., 2017, Moradi et al., 2018) and sensor fusion (John C Pearson, 1988, Chan, Jin, and Schaik, 2012), among others.

The present work is focused on neuro-inspired processing for artificial vision, and its aim was the development of bio-inspired systems capable of extracting patterns from input scenes in an autonomous and efficient way. The processing of images consists of two different phases: first, the nature of input stimuli is studied, analyzing and extracting the features; then these features are used to classify different images, searching for similarities between the features extracted in the previous phase. These phases can be found in the natural behaviour of living beings that learn patterns from an input stimulus, and apply this knowledge to recognize objects with the same properties.

In order to replicate this phenomenon, the aim of this work was to study different learning methods for frame-based processing and event-based processing. Regarding frame-based processing, the author studied the implementation of CNNs in embedded systems to perform pattern recognition in real time. Then, the same concept was adapted to event-driven processing. Although it can appear that these two concepts are similar, there is a great difference, since event-driven processing is inspired by the structure and functioning of the brain, where the information is encoded in spikes (events), which are processed in parallel by massive layers of neurons interconnected via synapses. In order to implement spiking convolution networks, a novel spiking convolution processor was developed in this work to infer this kind of network.

Finally, apart from CNNs in both paradigms, a novel pattern recognition algorithm for event-based processing was studied. These classifiers extract specific patterns from the input scene using different mechanisms and methods.

1.1 Motivation

Pattern recognition systems are needed for many applications, such as object tracking, text recognition, autonomous driving, speech recognition, helping people with disabilities, or in healthcare and medical applications, in order to detect different pathologies.

Real-time vision recognition is one of the most important challenges in recent years, since the number of frames per second to process must be high enough to detect changes in the actual scene. This fact is critical in some applications, such as autonomous driving, where a delay can cause an accident. Most of current pattern recognition algorithms for audio or image processing are deployed in huge server

racks since these algorithms perform a large number of operations. Therefore, the development of hardware systems capable of deploying this kind of algorithm with low latency and power consumption has recently become one of the main interest for the industry.

Currently, several pattern recognition accelerator systems for frame-based vision can be found in the literature. Most of these systems are focused on accelerating one kind of network called convolutional neural networks (**CNN**). These networks have become quite popular due to their high accuracy and easy unsupervised training. Although CNN accelerators have demonstrated good results in terms of latency and power consumption, the number of operations is still high compared to other processing paradigms where these algorithms can be adapted, such as Neuromorphic Engineering. In Neuromorphic Engineering, information is encoded in spikes, which represent pulses of a neuron. Using this concept, several neuromorphic vision sensors have been developed. The main characteristic of these sensors is that they only detect changes of luminosity. In other words, only those pixels whose luminosity have changed are reported. This fact can reduce the number of operations performed in CNNs, since only a few numbers of pixels are processed, instead of scanning all the pixels of a scene, even if the scene has not changed, as frame cameras do. Based on this kind of processing, the same concept of CNNs has been adapted to it. These networks are called Spiking Convolutional Neural Networks (**SCNN**).

The aim of this thesis was focused on the development of different digital systems capable of deploying this kind of algorithm in real time for frame-based and event-based processing paradigms. This work presents three different architectures for very large integration systems (**VLSI**). The first one is a CNN accelerator for frame-based processing, followed by an event-based convolution processor and its applications; the last architecture is a novel hardware implementation of a new event-based pattern recognition algorithm known as HOTS (Lagorce et al., 2017).

On the personal side, it is important to highlight the author's inquisitiveness for the development of new digital hardware architectures to deploy learning algorithms based on the biology of the brain in real time. The architectures developed can be used in several fields such as robotics navigation, or image recognition tasks.

This thesis is part of the research degree at the Robotics and Computer Technology group (RTC, TEP-108). This work is focused on and aligned with different tasks that are part of national and international research projects, which have served as funding. The work of the author has been supported by a "Formación de Personal Investigador" Scholarship from the University of Seville, to which the author expresses his gratitude. These projects are:

- NPP international project: Neuromorphic Processor Project.
- COFNET national project: Sistema Cognitivo de Fusión Sensorial de Visión y Audio por Eventos (TEC2016-77785-P).
- MINERVA excellence project: Mota-Infraestructura de Sensado y Transmisión Inalámbrica para la Observación y Análisis de la Pauta de Animales Salvajes o en Semilibertad (P12-TIC-1300).

The following sections present a detailed introduction about the history and the main scientific fields that this thesis comprises, starting with neuro-inspired systems,

followed by artificial vision and deep learning processing, and ending with spiking neural networks.

1.2 Neuro-inspired systems

Since the origin of life, living beings have adapted to different kinds of environments. Evolution throughout millions of years has allowed several species to change their habitat, colonizing new ones in order to survive. These species have adapted their physiology to a new environment in which they did not use to live. In addition, this evolution is intrinsic to their genetic code, evolving over the years to changes in their environment and transmitting these adaptation properties to their descendants. Thanks to this natural evolution, there is a great diversity of species.

Along history, engineers and scientist have studied the nature of living beings to take inspiration for developing efficient solutions to solve complex problems; this approach was the origin of bio-inspired systems. Nature has demonstrated to be one of the best approaches in several fields, such as professional swimsuits, which are based on shark skin, or in electronics, such as the sonar, which is inspired by the sounds produced by bats.

In recent decades, there has been a great revolution in society and in the industry, with the appearance of computer systems and robotics. In the industry, it can be observed that most of the manufacturing companies have incorporated robots that are capable of performing several complex and repetitive tasks efficiently. This fact has allowed satisfying the demand for products in society, since the time to obtain a product from the assembly line has been reduced. However, robots are limited in actions, since they only react to a limited number of instructions in a small controlled environment, with high power consumption and the need for supervision from a person. On the other hand, wildlife is capable of adapting to new environments, as was previously mentioned, learning from it, collaborating with other members, being conscious of their limitations, such as the need for resting and eating, without any kind of supervision; therefore, wildlife is intelligent and autonomous.

Nowadays, computer systems have evolved considerably, reaching a high computation rate that allows them to perform several complex tasks. This computer evolution was predicted by Gordon Moore (Moore, 2006) who said that the number of transistors will be duplicated every two years. However, the rate of progress in computers increases, whereas robots do not evolve at the same rate. Although algorithms to control robots are executed on high-end computers, this fact poses several problems: the power consumption of high-end computers reduces battery life; robots are limited to some instructions given an environment and an algorithm, making it impossible for them to learn and evolve with their environment; and the space limitation due to the fact that the connection latency between the robot and the computer must not affect the programmed task.

Maybe we should solve the problem with a different perspective or paradigm that allows robots to work in an autonomous way with low power consumption. For instance, we should wonder how nature is able to process information in such low latency without much power consumption; furthermore, we must wonder how humans are able to learn from their experiences. We should consider the possibility that the current algorithms or computer-based systems are not the best options to process information. Therefore, the field of robotics needs new computational paradigms that allow robots to evolve in such a way that they can learn and adapt

to new environments; in other words, provide them with similar cognitive properties as those observed in nature.

It is difficult to answer these questions, most of them are currently unanswered. However, there is one thing that most of the scientific community accepts: the human brain is the most complex processing system in the world and we should investigate the way in which it processes information. Taking inspiration from how the brain processes information, and by adapting the brain's behaviour in order to develop new solutions that can improve the performance of current processing systems, the concept of neuro-inspired system emerged. Neuro-inspired systems gather a set of devices that recreate how nervous systems encode information. This research field has grown in recent years, thanks to the work of neuromorphic engineers. Since general-purpose computers have come to a standstill nowadays (Elie, Forbes, and Strawn, 2017, Xiu, 2019), perhaps it is time to have a look at the brain to develop new specific solutions that can improve computer performance.

In this section, the concept of neuromorphic engineering is explained, from its history to its principles.

1.2.1 Neuromorphic engineering

The concept of neuromorphic engineering appeared in 1980, created by Carver Mead's group at the California Technology Institute (Caltech), with the aim of emulating and understanding the behaviour of neurons of the nervous system through their implementation in analog circuits (Mead, 1989, Mahowald and Douglas, 1991). Although the first development in neuromorphic engineering was implemented using analog technology (Liu et al., 2002), in recent years several hardware implementations of bio-inspired neural system architectures can be found in analog, digital or mixed signals. Apart from hardware implementations, the field of neuromorphic engineering has grown considerably in recent years, bringing researchers from other fields, such as biology, physics, mathematics or computer sciences, among others.

As a consequence of the growth of the neuromorphic community, several hardware platforms to train and deploy bio-inspired neural networks with hundreds and millions of neurons can be found (Furber, 2016b, Indiveri, Chicca, and Douglas, 2006). Additionally, several sensors with a high dynamic range based on this approach have been developed to be used in high-performance applications (Yang et al., 2016, Serrano-Gotarredona and Linares-Barranco, 2013, Posch, Matolin, and Wohlgenannt, 2011, Jiménez-Fernández et al., 2017).

Currently, there are two international workshops, one located in Telluride (United States) and another one for European researchers in Cappocaccia (Italy). These workshops allow researchers around the world to meet up in a common place, where they can share their advances with other researchers of the community, working together to develop and improve new neuro-inspired systems that process spiking information. In order to understand how neuromorphic platforms and devices work, it is necessary to understand the origin of neuroscience, which is the base of these solutions.

Continuously, our brain is processing information about the environment obtained thanks to our senses. The processing of this information allows us to do several actions, such as interacting with other persons, recognize food or identify a sound. These tasks are performed in a fast and simple way, although the structure of the brain is so complex at that point that currently there is a great part of it that is still unknown.

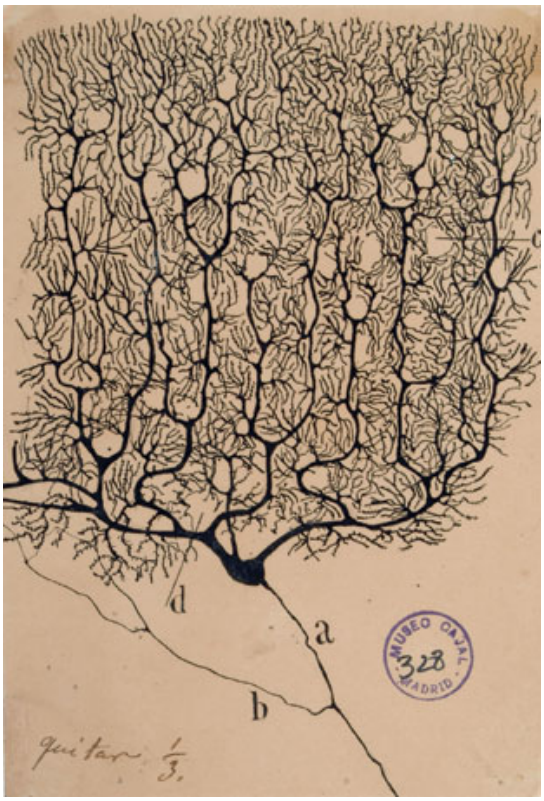


FIGURE 1.1: Neuronal model drawn by Santiago Ramón y Cajal.

The human brain is composed of a large amount of cells, called neurons. There are around 86 billion neurons of different types in our brain. The type of a neuron defines how it interacts with other neurons, the response against a stimulus and its purpose in a group. Camillo Golgi, an Italian scientist, stated that the nervous system had a structure of a fixed interconnected network of neurons; this theory is known as the reticular neuron theory. However, a Spanish scientist, Santiago Ramón y Cajal (Cajal, 1952), who received the Nobel prize in medicine, demonstrated that the reticular theory was not true, since there exists a small space between neurons of 20-30nm, and that they connect to each other through synapses. This theory is called the neural theory and it was demonstrated with the development of modern microscopes. Fig. 1.1 shows one of the drawings of Santiago Ramón y Cajal, where the structure of a biological neural network is represented.

The amazing fact of neural architecture is that it only consumes between 10 and 40 Watts of energy. The morphology of a neuron consists of three parts: soma, dendrites, and axon; and they are self-organized. The dendrites are the input of a neuron. They receive an input stimulus from another neuron and they send this stimulus to the soma. The soma is the central processing unit of a neuron; it processes an input from the dendrites in a non-linear way. The input stimulus modifies the membrane potential of the soma and, if a threshold is reached, the soma produces and output stimulus, sending it through the axon, which acts as the "output of a neuron". Fig. 1.2 shows the structure of a biological neuron.

The connection between two neurons is known as the synapse, and it has complex physiological characteristics (Johnston and Wu, 1995). In 1939, Hodging and Huxley (Hodgkin and Huxley, 1939) discovered that neurons communicate with each other by means of electric pulses, studying how their potassium and sodium channels behaved. The changes in the membrane potential of the dendrites

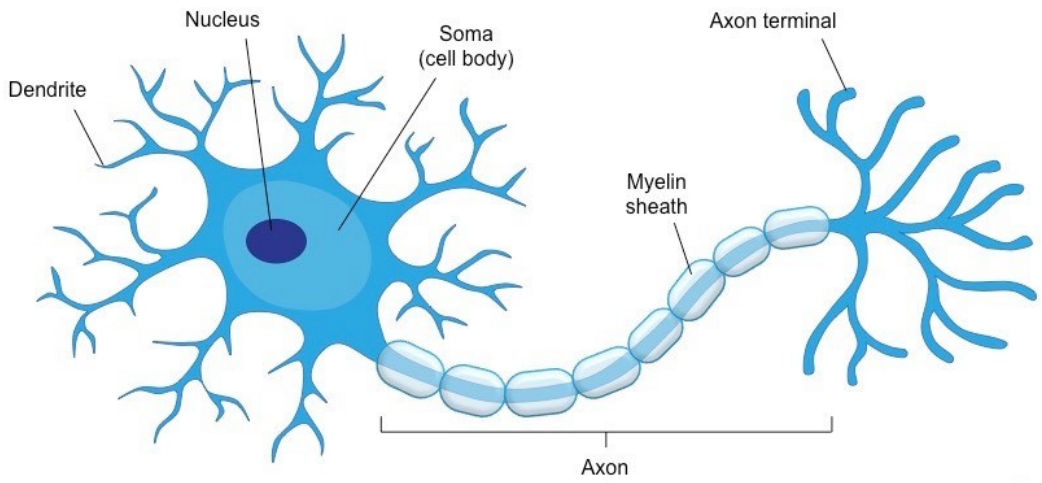


FIGURE 1.2: Biological neuron structure.

produce these electric pulses called action potential or spikes.

A neuron's membrane potential usually starts at a value of -70 mW, called resting potential. If the opening of the ion channel results in a net gain of positive charge across the membrane, the latter membrane is said to be depolarized, as the potential comes closer to zero. This process is called excitatory post-synaptic potential (**EPSP**), as it brings the neuron's potential closer to its firing threshold (about -55 mV). On the other hand, if the opening of the ion channel results in a net gain of negative charge, this moves the potential further from zero and is referred to as hyperpolarization. This process is called inhibitory post-synaptic potential (**IPSP**), as it changes the charge across the membrane to be further from the firing threshold. Finally, the neuron is not able to produce a new spike until its membrane potential returns to the resting potential; this period of time is called the refractory period. Fig. 1.3 shows the diagram of a spike generated by a neuron.

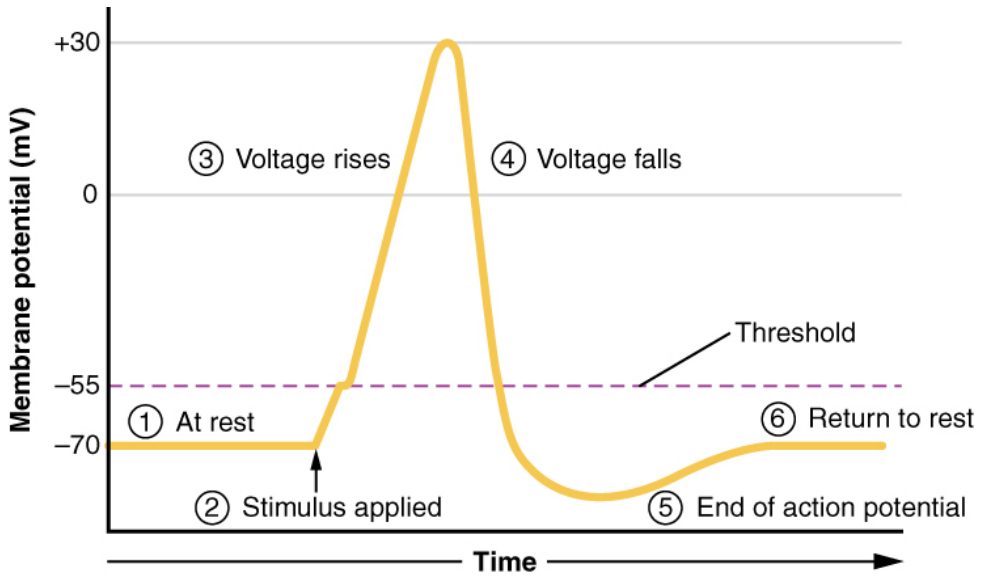


FIGURE 1.3: Diagram of a spike generated by a neuron.

Although the Hodgkin–Huxley membrane potential model of a neuron is accepted by most neuroscientists, there are several hypotheses on how neurons

represent information. One of the most accepted hypotheses in the neuromorphic engineering community is the one presented by Horace Barlow (Barlow, 1961), who proposed a model in which neurons communicate through a train of spikes using pulse frequency modulation (Maass and Bishop, 1999, Westerman, Northmore, and Elias, 1997). Other methods to encode information employ the inter-spike interval (Reich et al., 2000), or the reset time, where the most important events are those which were emitted first (Thorpe, Brilhault, and Perez-Carrasco, 2010).

One of the main properties of spiking representation is that information is obtained in a continuous way, instead of a discrete way. Since the information is not sampled, each neuron transmits a spike when it is needed, avoiding redundant information.

TABLE 1.1: Comparison between a computer system and the nervous system.

Computer	Neural System
High speed global clock signal.	Asynchronous without a global clock signal
Deterministic behaviour	Stochastic behaviour.
High resolution information sampled at a constant rate.	Low resolution, but adaptive. No sampling rate; the information is encoded within the frequency of the spikes.
Centralized computing, or slightly distributed.	Each neuron processes a small part of the information. The processing is highly distributed and massively parallel.
It needs memory for the algorithm and to store the data.	The information is encoded within the flow of spikes. The morphological characteristics and the interconnections of each neuron is the algorithm itself.

Table 1.1 shows a qualitative comparison between a computer system and the nervous system. A computer is synchronized by a global clock signal, which allows the computer to perform the task in each rising or falling edge of a clock; on the other hand, the nervous system does not have synchronized mechanisms, since neurons communicate and work in an asynchronous way. Furthermore, computers follow a deterministic sequence of instructions, whereas the nervous system has a stochastic behaviour, depending on their reaction to their dynamic probabilistic models. Current computer systems work with high-resolution information at a constant rate, whereas in neurons it is the opposite. Although neurons work with a low resolution compared with computers, the information is encoded in the frequency of the spikes transmitted, adapting this frequency to the input stimulus. The nervous system is a complex network, where each neuron processes a small part of the information, which allows neurons to not depend on other neurons, working on a massively parallel and distributed way. Although current computer systems allow parallel computing, there always exists an element which is in charge of synchronizing the execution, thus centralizing the computation. Furthermore, computer systems need memory to execute an algorithm and to store the data. However, in neural systems, the information is encoded in the flow of the spikes between neurons, thereby the morphological properties of neuron interconnections constitute the algorithm.

1.2.2 Address event representation

One of the goals of neuromorphic engineering is to take inspiration from the nervous system in order to find solutions to problems. However, the nervous system is composed of a vast number of neurons, where each of the $10^5/mm^3$ neurons could be connected to other 10 thousand, generating a density of connections of up to 4 km$/mm^3$ (Braitenberg and Schüz, 2013); due to the physical limitation of VLSI technology, it is not possible to implement such connectivity in a VLSI system. Although a single neuron has a firing rate of 1-10 Hz, the frequency of a group of neurons firing at the same time can increase to KHz or MHz. Current digital circuits are faster and can support that firing rate. In order to implement neuron communications, by making use of the high bandwidth of VLSI, a protocol that multiplexes information in time was developed, using one channel and assigning an address to each neuron. This protocol is called address-event representation (**AER**) and it was first presented in 1991 (Sivilotti, 1991, Lazzaro et al., 1993, Boahen, 2000, Liu et al., 2014). AER is an asynchronous and digital protocol to send/receive spikes between neuromorphic chips (Mahowald, 1992).

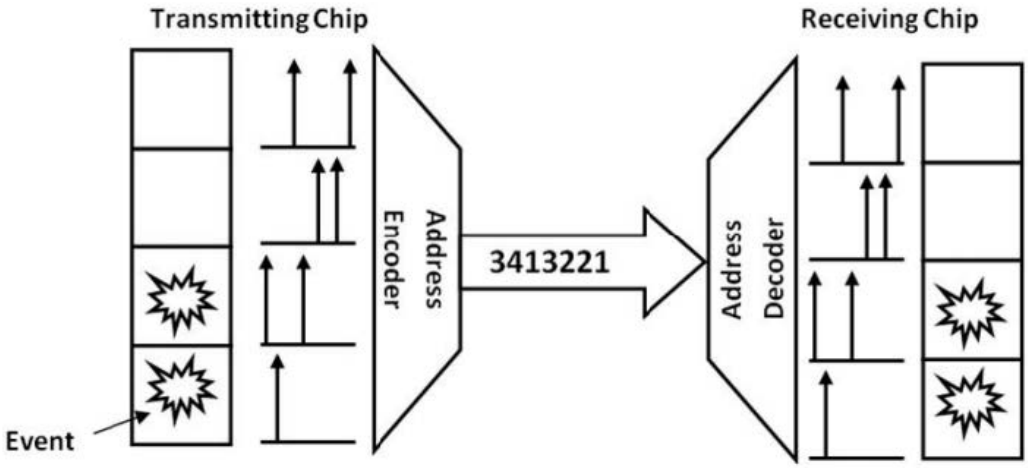


FIGURE 1.4: AER protocol diagram. Image taken from (Liu et al., 2014)

Fig. 1.4 shows a diagram of the AER protocol during the communication of two AER systems. AER proposes a unique multiplexed high bandwidth bus, where each neuron is represented by an address. When a neuron produces a spike, also known as AER event, an arbiter sends the address of that neuron. An address can represent the position of a pixel in a vision sensor or an audio channel in an auditory system. When the receptor sensor receives the AER event, the sensor is in charge of decoding the address, to send the spike to a group of receptive neurons. Thus, the neurons are connected virtually, through a common multiplexed bus.

There are two main ways of implementing the AER communication protocol: parallel and serial protocols. In general, the parallel protocol is widely used in neuromorphic engineering. In this work, the parallel implementation used in the European project CAVIAR: Convolution AER Vision Architecture for Real-Time (IST2001-34124) was used (Serrano-Gotarredona et al., 2009). On the other hand, serial protocols for AER communication are under research and being tested (Dorta et al., 2016, Fasnacht, Whatley, and Indiveri, 2008, Yousefzadeh et al., 2017), and

these will replace the parallel protocol in the future, as it happened in computer systems.

The most commonly used AER protocol consists of a four-step handshake protocol that guarantees the correct communication between the sender and the receiver. It is composed of three signals: data, request and ack. The sender asserts the request signals to inform the receiver that there is an address ready to be sent. In the case that the receiver is ready to read the event, the receiver asserts the ACK signal. Finally, the sender deactivates the request signal, then the receiver also unsets the ack signal to end the communication.

The AER protocol was the main communication protocol used throughout this work to receive and process information from vision sensors.

1.3 Artificial Vision

1.3.1 Vision in Biology

1.3.1.1 Visual System

Vision is one of the most important senses, and this is reflected by its complexity and by the large part of the mammalian brain cortex that it occupies. Neuroscientists discovered that 32 different parts of the brain process visual information (Cela-Conde et al., 2004), 25 of which work directly in vision processing tasks, whereas the other 7 perform sensory fusion and motor system tasks.

From the retina to the lateral geniculate nucleus, and from it to the successive layers of the brain cortex, there are neurons that process complex stimulius. For instance, in the primary visual cortex (V1), there is a set of neurons that reacts to the orientation of the input visual stimulus (Krug, 2012).

The visual system does not follow a sequential and hierarchical scheme. Information, such as the colour or shape of objects, is processed in parallel, i.e., routing the information through parallel routes to be processed by different parts at the same time.

1.3.1.2 Retina

The retina is the main part where the vision system starts. One of the first cell layers in the retina is the photo-receptor cell which detects spatio-temporal changes in brightness and darkness. Apart from the photo-receptor layer, inside the retina, there are several kinds of cells. Although, two types, called M and P, make up 90% of the retina. M cells are sensitive to high spatial frequencies, reacting slowly,. and they are the beginning of the parvocellular route, which is related to colour and details. On the other hand, P cells are sensitive to low spatial frequencies, reacting quickly to the input stimulus, and they are the beginning of the magno route, related to movement and contour.

1.3.1.3 Lateral geniculate nucleus

The lateral geniculate nucleus (**LGN**) is the intermediate point between the retina and the visual cortex. Axons from ganglion cells connect to each other, composing the optical nerve. Nerves from both eyes join together to make the optic chiasma. At this point, internal axons from retinas interconnect and they continue until they reach the LGN. Thus, both hemispheres have information from both retinas.

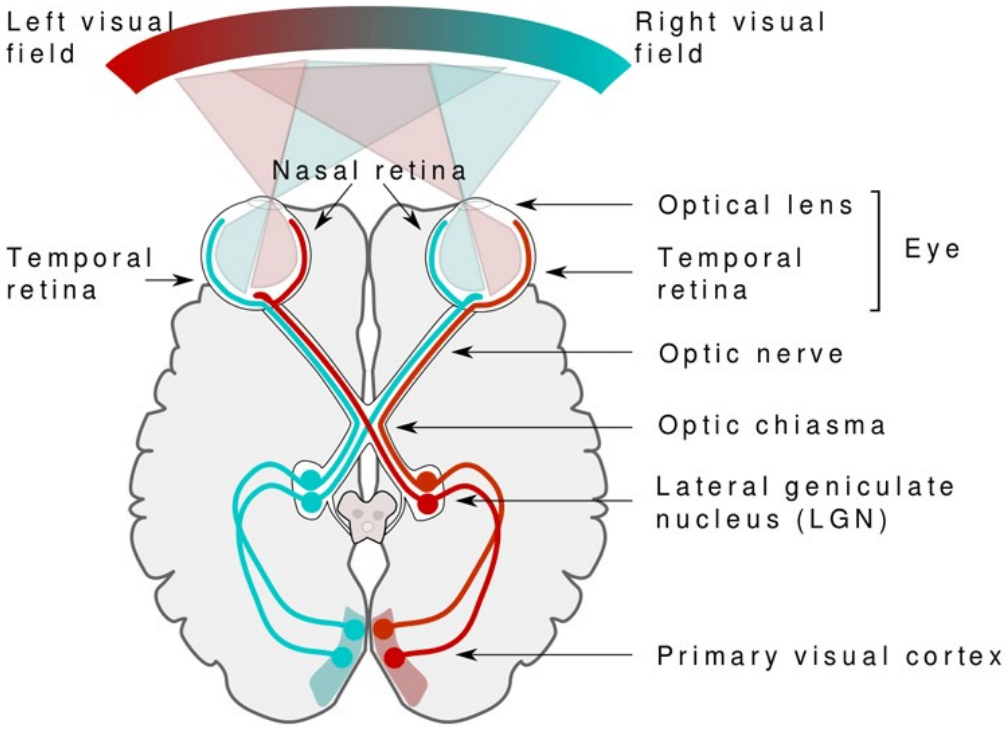


FIGURE 1.5: Neural vision system structure.

The LGN is composed of sets of neurons distributed in 6 layers that process information from both retinas. Layers 2,3,5 process the information of the retina at the same position as the LGN, whereas layers 1,4,6 process the information of the retina of the other side. The LGN follows the same organization as the ganglion cells in the retina. The LGN follows the same organization as the ganglion cells in the retina. M cells are projected in layers 1 and 2; on the other hand, P cells are projected in layers 3,4,5,6. This is the principle of binocular vision (Khan, Wadhwa, and Bijlani, 1994).

1.3.1.4 Visual Cortex

The cells of the LGN are projected into the visual cortex. The visual cortex is the area of the brain in charge of processing the visual stimuli, and it is mainly composed of 5 interconnected areas: V1, V2, V3, V4 and V5. In these areas there are two routes: "*what*" and "*where*"; the aim of the first route is to recognize the objects from the scene and it can be subdivided into two ways, i.e. colour and form, whereas the second route detects the position of objects in the scene. The neurons of the V1 layer belong to both paths, with areas V1->V2->V3->V4 being in charge of the "*what*" path; and V1->V2->V5 intervene in the "*where*" path. However, the two paths are not isolated, as there are neurons that interconnect them. A common property of the neurons of the different layers is that they only react to small areas of the visual field, which is called the receptive field. The V1 layer is common to both paths, and it is also known as the primary visual cortex. The primary visual cortex receives pre-processed information from the LGN, and separates this information in order to perform complex analyses in further layers. In the primary visual cortex, several fundamental tasks for visual perception are performed, as was demonstrated by

David H. Hubel and Torsten Wiesel in 1962. There are several neurons which are sensitive to some visual aspects (Wiesel, 1963), such as movement, orientation and colour. Furthermore, it was also discovered that the receptive field of the neurons of V1 changes dynamically, demonstrating that these neurons change their responses to the same stimuli after a period of time. In other words, neurons are able to learn from unknown stimuli and change their behaviour.

The visual system has inspired several approaches in several fields, such as artificial vision or deep learning. Fig. 1.5 shows the entire vision system.

1.3.2 Acquisition of digital images

The first photography was taken in 1826 by French engineer Nicéphore Niépce. However, the greatest contribution to photography was the development of the first charge-coupled device by Willard Boyle and George E. Smith in 1969 at Bell labs. It was the first digital image that could be stored in a computer, in order to capture information of the current scene and process it. The evolution of digital image vision systems has helped in other fields, such as medicine, where we can obtain a 3D image of our tooth (Aichert et al., 2012); or astronomy where the universe can be studied thanks to the use of modern telescopes, such as the Hubble telescope (Whitmore and Schweizer, 2002).

Within the evolution of computer systems and digital imaging, the concept of artificial vision emerged. The purpose of artificial vision is to mimic the way in which living beings process visual information, replicating this behaviour through computer algorithms, in order to perform tasks such as object tracking and image classification, among others. Nowadays, there are several systems that, thanks to artificial vision with other sensors, are able to perform different tasks, such as autonomous driving (Matthaei et al., 2015) or face unlock in cell phones (Crouse et al., 2015).

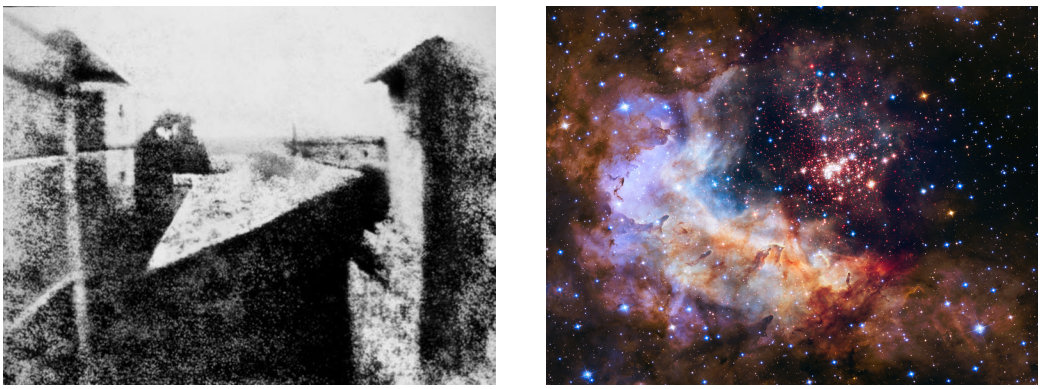


FIGURE 1.6: The world's first photograph (left) and a photograph of the Hubble telescope (right).

The first step in artificial vision is the acquisition and digitization of images captured by an image sensor. There exist many different types of images; three of the most used types are listed below:

- Intensity images: these are based on the luminous intensity obtained by the sensor for all points of the captured scene. This kind of image is the most common one, due to the low-cost manufacture of these sensors.
- Thermal images: these are captured with sensors that detect the radiation in the long-infrared range, creating a thermal map of the objects.

- 3D images: pixel value corresponds to the value of z, which determines the depth of that point in the scene.

Regarding intensity images, due to their great usage, after the acquisition of the image, a dimensional matrix is obtained. Each value of this matrix corresponds to the intensity of that point in the scene. The concept of these images follows a bidimensional function $f(x,y)$, where the result of the function corresponds to the luminosity of the pixel at x,y position. The value of the function depends on two factors: the luminosity of the scene ($i(x,y)$); and the amount of luminosity reflected ($r(x,y)$) by the objects of the scene. These factors are called luminosity and reflection.

$$f(x,y) = i(x,y) * r(x,y) \tag{1.1}$$

The digitized function shown above is related to the sampling. The sampling converts the spatial coordinates to a matrix, where the values of each cell represent the grayscale of the image. For instance, for a codification of 8 bits for the grayscale (256 levels of grey), a 0 would correspond to a pixel that has absorbed all the incident luminosity (black); on the other hand, the maximum value of 255 would represent a pixel with the highest luminosity (white).

Finally, the last concept that defines an image is resolution. The resolution of a digital image is the number of pixels captured by a vision sensor, and it is measured in pixels per inch.

1.3.3 Silicon retinas

In 1970, Fukushima presented the first prototype of an electronic retina (Fukushima et al., 1970). Although it had a great impact on the community, it was not as successful in the industry, unfortunately. Nevertheless, this first attempt posed a change in the state-of-the-art in vision sensors. In the last decades, it has been a great challenge in neuromorphic engineering to improve and implement these first prototypes in industrial applications.

Pixels of an AER retina behave as an independent neuron that works in an asynchronous way, sending their addresses through an AER bus when they detect a significant signal. The "*significant*" signal depends on the implementation of the pixel. AER retinas are commonly divided into two groups:

- Spatial contrast (**SC**) sensors reduce spatial redundancy based on intensity ratios, unlike spatial difference sensors, which use intensity differences. SC sensors are useful in applications whose purpose is to extract features or classify objects from a static scene.

- Temporal contrast (**TC**) sensors reduce the temporal redundancy based on relative insensitive changes, unlike temporal difference sensors, which use absolute intensity changes. TC sensors are useful for dynamic scenes where illumination is not constant, such as object tracking and navigation.

Currently, there are several AER retinas: the dynamic vision sensor (**DVS**) (Lichtsteiner, Posch, and Delbrück, 2008, Serrano-Gotarredona and Linares-Barranco, 2013), the asynchronous time-based image sensor (**ATIS**) (Posch, Matolin, and Wohlgenannt, 2011), the biomorphic Octopus image sensor (Culurciello, Etienne-Cummings, and Boahen, 2003), the Magno-Parvo spatial (Zaghloul and Boahen, 2004a, Zaghloul and Boahen, 2004b), and the spatial contrast and orientation sensor (VISe) (Rüedi et al., 2003). Two of these

neuromorphic vision sensors were used in this work: the DVS sensor and the ATIS sensor. These sensors are inspired by the photo-sensitive cells of biological retinas, as they capture a dynamic reality with each pixel triggering an event when the change in luminosity exceeds a threshold. This luminosity change is encoded in the polarity (p) of the visual event, which can be ON ($p = 1$) when the luminosity increases, and OFF ($p = 0$) when luminosity decreases. Therefore, static visual scenes will not produce any events, since there are no changes in them. This prevents the redundancy of data, obtaining precise information. Fig. 1.7 shows the operation principle behind the ATIS sensor.

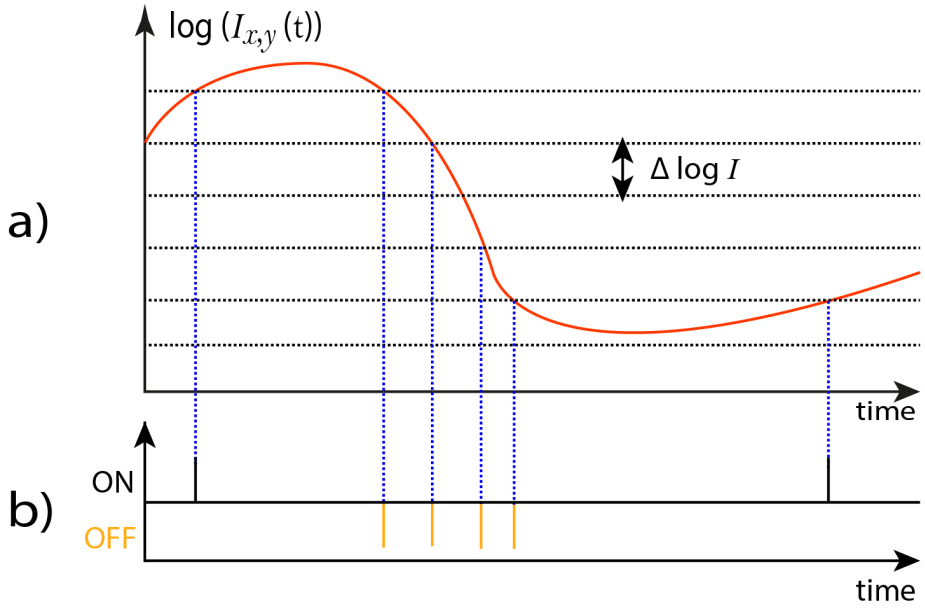


FIGURE 1.7: ATIS operation principles. When a pixel's luminosity change reaches a given threshold (a), it produces a visual event with an (x, y) address and a polarity, which is either ON or OFF (b).

In literature, there are several applications of these sensors, such as object tracking (Linares-Barranco et al., 2018, Moeys et al., 2016), motor controlling (Conradt et al., 2009, Perez-Peña et al., 2015, Rios-Navarro et al., 2015) or robotics (Mueggler et al., 2015, Perez-Peña et al., 2013). The output of these sensors is commonly processed by some kind of neuromorphic software framework.

1.3.4 JAER

Most of the tasks mentioned before were implemented using a well-known neuromorphic software framework, called JAER (Delbrück, 2007). JAER is an open-source software framework for real-time or event-based recorder visualization, and for the development of filters or algorithms for event-based processing. The algorithms and filters are developed in Java language for PCs. JAER is composed of a "*jAERviewer*", which allows the connection of multiple kinds of sensors, such as vision sensors or auditory system; and it allows performing multiple tasks, such as viewing the events of the sensor in real time, logging the stream of events or applying a filter.

These sensors emit the events in an asynchronous way with a timestamp of 1μs. The connection between the sensor and the computer is performed through a USB (Berner et al., 2007, Paz-Vicente et al., 2006), where several events are sent in packets.

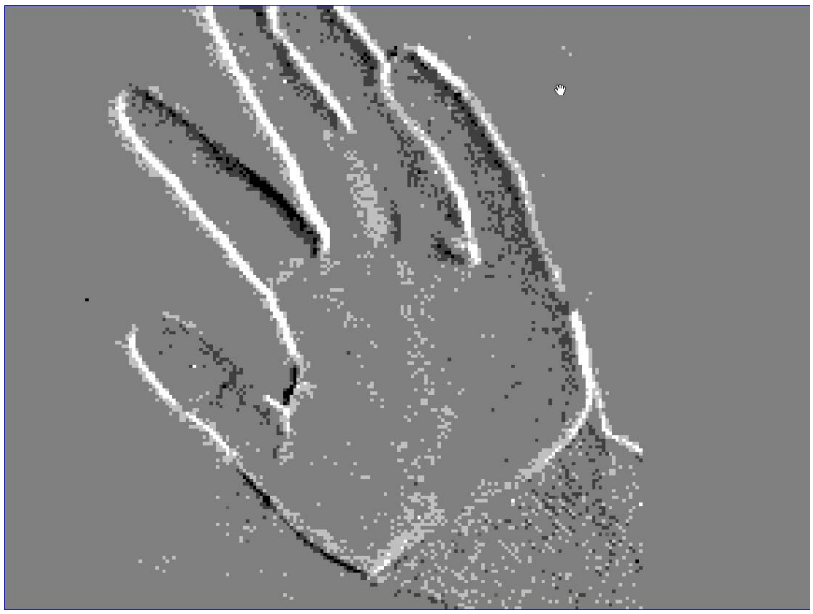


FIGURE 1.8: jAER capture of DVS sensor events.

jAERviewer can display the events in real time. Apart from visualizing events from a sensor, a jAERviewer can log the output of a sensor, reproduce a recorded stream of events, or apply a filter to the input stream.

In this work, jAER was used to receive events from a vision sensor and send events to the systems designed.

1.4 Deep Learning

During the search for mechanisms of artificial intelligence, several mathematical algorithms have been developed; one of the most popular approaches is deep learning. The purpose of deep learning is to use the full computation capabilities of computers to extract further and more detailed information compared to human capacity. Computer technology has evolved, obtaining the computational power required to deploy these algorithms, through which pattern matching or recognition tasks can be solved quickly.

These algorithms need vast labelled data sets for training and testing. On the other hand, these kinds of algorithms need several training iterations of complex computation. Therefore, deep learning algorithms typically learn and classify a whole dataset using a specific algorithm. Due to the number of iterations and the complex operations performed by these algorithms, they are usually used when there is a large/labeled dataset and when there is not another simple method to apply.

1.4.1 History of Deep Learning

In 1975, Kunihiko Fukushima proposed the Cognitron neural network (Fukushima, 1975), which mimics how the visual system works for visual pattern recognition. This network was extended in 1982 with a model called Neocognitron (Fukushima and Miyake, 1982), which was still very slow and, like its predecessor, was limited to visual pattern recognition.

These neural network architectures are not considered to be part of the field of deep learning, although they served as a basis for convolutional neural networks, which are the ones that were used as deep learning architectures in this work. In 1985, David Rumelhart et al. showed that neural networks with multiple hidden layers could be effectively trained by a relatively simple procedure, called back-propagation (Rumelhart, Hinton, and Williams, 1986). This would allow neural networks to get around the weakness of the perceptron, since the additional layers endowed the network with the ability to learn nonlinear functions.

Within deep learning, there are several algorithms for pattern recognition; convolutional neural networks (**CNN**) are among the most popular. CNNs are inspired by the first layers of the visual cortex. Furthermore, the work developed by Yann LeCun demonstrates that CNNs can be used to solve problems related to artificial vision. In 1989, Yann LeCun aggregated convolutions to a 5-layer model with backpropagation, decreasing the processing time and also obtaining more characteristics from input data (LeCun, 1989). These algorithms require a long training time, which can be shortened through the use of high performance computing (**HPC**).

CNNs have been demonstrated to be a good approach in several fields apart from artificial vision, such as audio processing (Domínguez-Morales et al., 2018, Abdel-Hamid et al., 2014) or robotics (Browne and Ghidary, 2003, Ran et al., 2017). Therefore, the tendency is to implement this kind of algorithm in embedded devices, in order to deploy CNN models locally and in real time, avoiding the usage of the cloud technology. The next section presents the state-of-the-art in the different hardware implementations of CNN accelerators.

1.4.2 State-of-the-art in CNNs accelerators

The vast number of computational resources within the high power consumption forced CNNs to be deployed in high-end servers at the beginning. However, the high parallelism obtained through graphical processor units managed to reduce the time to train and it also allows to infer models faster. This kind of platform can be used remotely from any portable device, thanks to the cloud technology, which is commonly used nowadays by most people. However, the hardware industry is looking for VLSI solutions to implement this kind of network in real time with low power consumption. This eliminates the need for an internet connection, which allows performing pattern vision recognition tasks in real time locally; this is critical in applications such as autonomous driving. Several VLSI solutions for application-specific integrated circuit (**ASIC**) or field-programmable gate array (**FPGA**) technologies have been implemented in recent years, in order to infer CNNs even further. In this section, three of the most known and important ones will be described. These architectures are: Origami, Snowflake and Eyeriss.

1.4.2.1 Origami accelerator

The Origami accelerator (Cavigelli and Benini, 2017) was one of the first implementations of the CNN hardware accelerator for ASIC. The main property of the Origami accelerator is the fact that pixels are received in streams. When there are enough pixels to process, the convolution operation is performed in sum-of-products (**SoP**). The circuit iterates over the channels of the input image, storing the partial sums in a channel sum module (**ChSum**), in order to send the partial sum out.

composed of three memory banks: 1) filter bank, which stores the weights and parameters of the filters, 2) the image windows SRAM, and 3) the image bank. The last two are in charge of receiving the input pixel and feed the SoP modules.

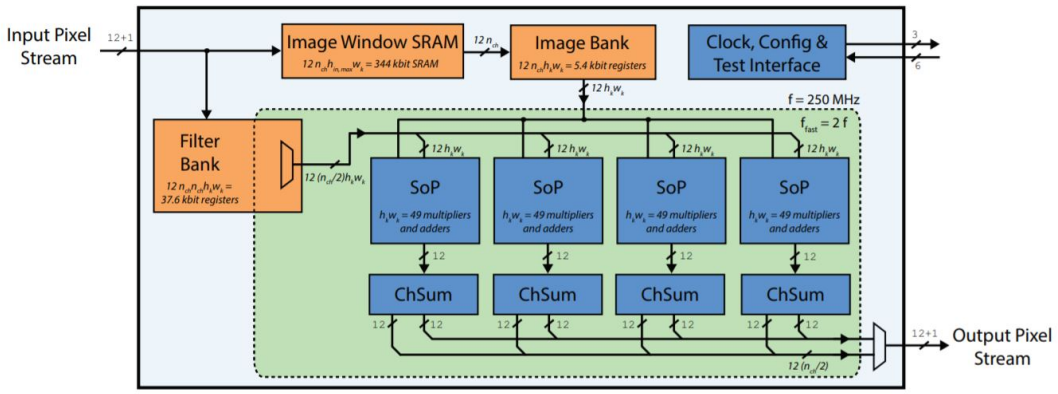


FIGURE 1.9: Origami accelerator architecture. Image taken from (Cavigelli and Benini, 2017).

The Origami architecture was a novelty in the way that data is processed, receiving the pixels in streams and creating pipeline stages to the process. However, since it is one of the first implementations, this architecture is limited, as it is only capable of performing a small number of the convolution operations of a CNN, and other layers, such as pooling of ReLU, have to be performed in software. Fig. 1.9 shows the complete structure of the Origami CNN accelerator.

1.4.2.2 Snowflake

The Snowflake accelerator (Gokhale et al., 2017) is the evolution of a previous CNN accelerator called NN-X (Gokhale et al., 2014). The first difference from this accelerator is how input data are organized, in order to obtain the maximum performance. The data are organized in "*traces*", which are defined as contiguous regions of memory that must be accessed as part of the computation necessary to produce a single output pixel. In addition, the data from a trace are accessed from on-chip buffers at 256-bit granularity. These smaller blocks of data are called vectors and they feed the vector MAC units (vMAC). This data organization is what makes Snowflake the most efficient accelerators, since that data are processed in vectors in a similar way as GPU units. The Snowflake architecture is composed of 5 modules: the control core, which is in charge of configuring the traces; the compute core, which contains the vMACs; the maps and weights buffers, where traces and weights are stored; the comparators, which perform the maxpool and ReLU operations; the data distribution network (DDN), which is responsible for forwarding data returned from the memory to the correct on-chip buffers and forwarding results produced by the coprocessor back to the memory; and, lastly, the memory interface, which contains four load-store units to move results or parameters into the memory.

1.4.2.3 Eyeriss

Eyeriss (Chen et al., 2017) is a CNN accelerator developed by the Massachusetts Institute of Technology (MIT). The concept behind Eyeriss is the re-utilization of resources to decrease the number of memory accesses, thus reducing memory

bottleneck, which is the main problem in Deep Learning systems. Eyeriss consists of a matrix of processing elements (**PE**), connected to a memory, called SRAM buffer. Each PE is composed by a register file memory, an arithmetic logic unit (**ALU**) and a small control unit. PEs process data in rows, performing the convolution operation, accumulating the partial sums or performing another kind of operation, such as pooling, through the communication between them. The results are sent to the SRAM buffer, which configures the PEs for the next layer and comprises the output feature maps for the next layer. The compression reuses the result of the ReLU layer, which filters negative pixel values, setting them to zero. This allows avoiding the multiplication of zero pixels, reducing the computation and, decreasing the power consumption; this technique is known as zero-skip. Fig. 1.10 shows the architecture of the Eyeriss accelerator.

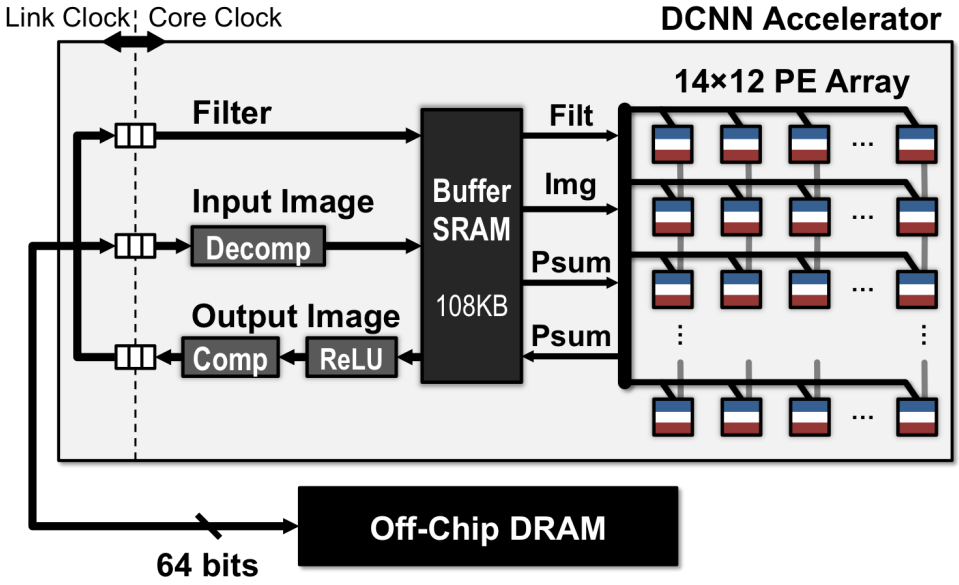


FIGURE 1.10: Eyeriss accelerator architecture. Image taken from (Chen et al., 2017).

1.5 Spiking Neural Networks

The spiking neural network (**SNN**) is a type of artificial neural network that mimics biological neurons (Vreeken, 2002, Maass, 1997). The idea behind SNNs is that neurons of this kind of network fire when their membrane potential reaches a threshold, following the same mechanism as that observed in biology, which was previously explained. In spiking neural networks, the activation level is usually considered to be the neuron's state, increasing with the incoming spikes from other neurons, and then either firing or decaying over time. This approach is the most used in neuromorphic engineering for pattern recognition, and several hardware platforms have been developed to train and deploy SNNs.

1.5.1 IBM-TrueNorth

The IBM TrueNorth (Akopyan et al., 2015) chip is a very large CMOS chip that incorporates 4096 neurosynaptic cores, where each core comprises 256 neurons

each with 256 synaptic inputs. The architecture is fully digital and operates asynchronously.

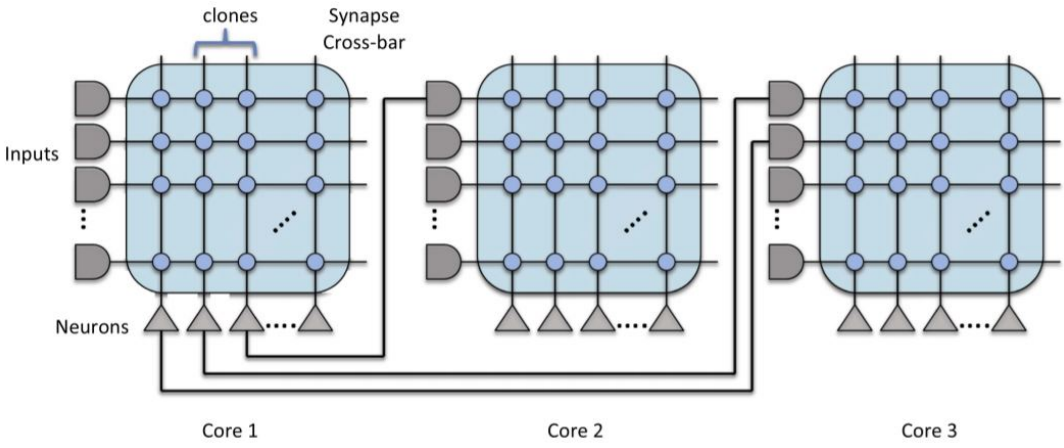


FIGURE 1.11: IBM TrueNorth architecture. Image taken from (Akopyan et al., 2015).

The architecture has a clock of 1 KHz, which determines the basic time step. The design is deterministic, since it executes software models that are easy to predict; therefore, IBM TrueNorth can be used for application development or the implementation of learning systems. The TrueNorth neurosynaptic core consists of a 256x256 crossbar that selectively connects incoming neural spike events to outgoing neurons. Neurons are connected following point-to-point connections to outgoing neurons inside or outside the chip. Fig. 1.11 shows the architecture of IBM TrueNorth.

1.5.2 NeuroGrid

Neurogrid (Benjamin et al., 2014) is a neuromorphic platform designed in Stanford University, for the simulation of biological brain neurons. It is based on analog technology emulating the ion channel activity of neurons with softwire structured connectivity patterns. Neurogrid is able to simulate a maximum of one million neurons and six billion synapses in real time. The Neurogrid board contains sixteen Neurocores, each of which has 256 x 256 silicon neurons per chip. An offchip RAM and an on-chip RAM (in each Neurocore) softwire horizontal and vertical cortical connections respectively, creating a grid of interconnected neurons.

1.5.3 SpiNNaker

SpiNNaker (Spiking Neural Network Architecture) (Furber, 2016b) is a massively parallel multicore computing system for modelling very large spiking neural networks in real time, optimized for neuromorphic applications. Both the system architecture and the design of the SpiNNaker chip have been developed by the Advanced Processor Technologies Research Group (APT). Each SpiNNaker chip consists of eighteen 200 MHz general purpose ARM968 cores, each with 64 kB of tightly-coupled data memory and 32 kB of tightly coupled instruction memory. The chip contains a Globally Asynchronous Locally Synchronous (GALS) architecture with an asynchronous packet-switching network that is highly optimized for neuromorphic applications. The communication between them is done via packets carried by a custom interconnect fabric. Only 16 ARM cores are involved in the

neuromorphic process, one of the rest is used for communication and the other is reserved. Fig. 1.12 shows the layout of the whole SpiNNaker board.

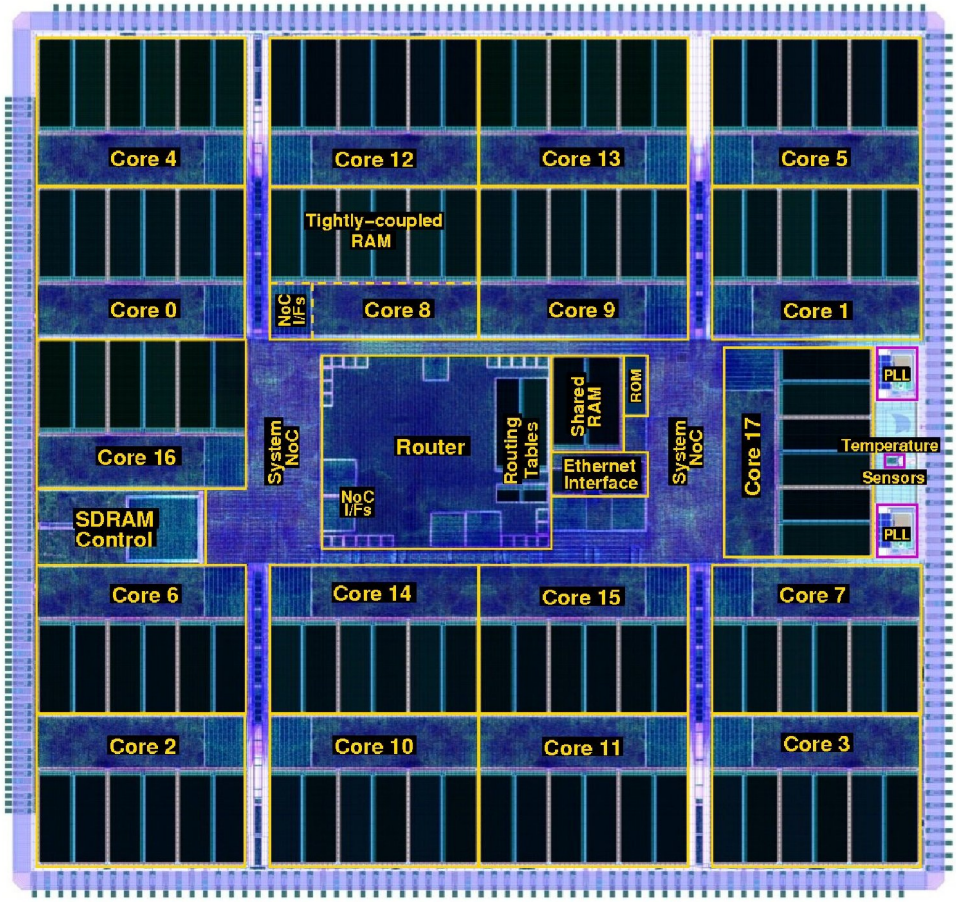


FIGURE 1.12: Spinnaker platform layout. Image taken from (Yousefzadeh et al., 2018).

1.5.4 Intel Loihi

The Intel Loihi (Lin et al., 2018) architecture consists of a digital asynchronous architecture composed of a mesh of 128 neuromorphic cores, three embedded x86 processor cores, and an off-chip communication interface that allows the chip to scale out to many other chips in the four planar directions. Communication between cores is performed asynchronously through packetized messages with write, read request, and read response messages for core management, spike messages, and barrier messages (for synchronization).

Each neuromorphic core implements 1024 primitive spiking neural units based on leaky-integrate-and-fire neurons, grouped into tree-like structures in order to simplify the implementation. Fig. 1.13 shows the structure of an Intel Loihi core. Each of those groups shares the same fan-in and fan-out connections, configuration, and state variables in ten architectural memories. A unique property of Loihi's cores, with respect to other neuromorphic platforms, is their integrated learning engine, which allows full-programmable on-chip learning.

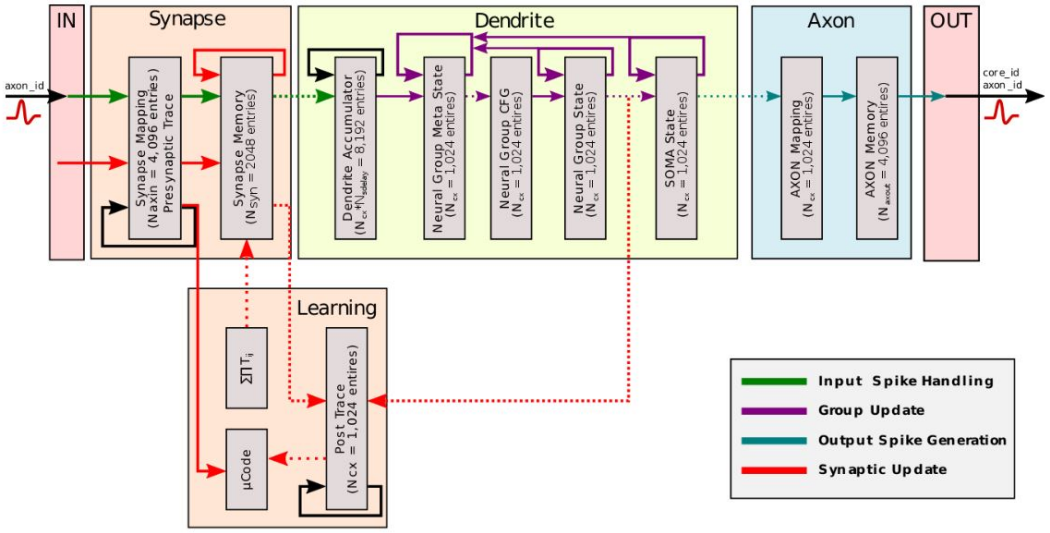


FIGURE 1.13: Intel Loihi core structure. Image taken from (Lin et al., 2018).

1.5.5 DYNAP

DYNAP (Moradi et al., 2018) is a digital asynchronous architecture that comprises four cores; each core comprises 256 mixed-signal neurons based on leaky-integrate-and-fire; and each neuron has a fan-out of 4k, being able to perform a maximum of 64k synapses. DYNAP architecture contains a novel routing methodology that employs both hierarchical and mesh routing strategies and combines heterogeneous memory structures for minimizing both memory requirements and latency, while maximizing programming flexibility to support a wide range of event-based neural network architectures, through parameter configuration. Fig. 1.14 shows the layout of the whole DYNAP chip.

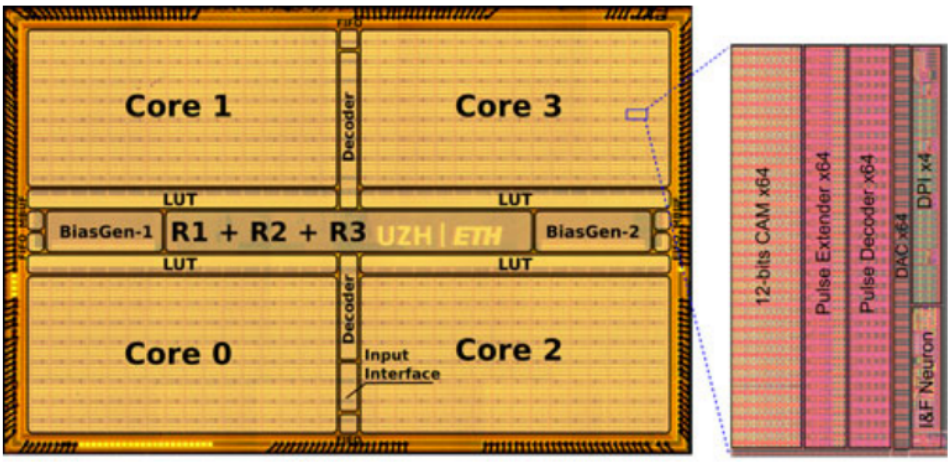


FIGURE 1.14: DYNAP platform layout. Image taken from (Moradi et al., 2018).

1.5.6 Spiking Convolutional Neural Networks

The spiking convolutional neural network is a type of SNN based on CNNs. SCNNs implement the convolution operation, generally using leaky integrate-and-fire neuron models or similar, such as the integrate-and-fire neuron model. This kind of network uses the membrane potential of neurons, increasing it with incoming spikes until a neuron reaches the threshold and fires, generating a spike. This accumulation is the equivalent of multiplying the intensity of a pixel a number of times in frame-based convolutions.

This concept of spiking neural networks is studied and explained in detail along this work, with the implementation and development of several architectures to accelerate the spiking convolution.

Chapter 2

Objectives and Thesis structure

"Those who can imagine anything, can create the impossible."

Alan Turing

This chapter sets out the main objectives of this thesis. First, the general and specific objectives are presented. Then, the structure of the thesis is presented with a brief description of each chapter.

2.1 Objectives

The purpose of neuromorphic engineering is to solve complex tasks emulating the processing paradigm of the nervous system through the implementation of analog, digital or mixed signal hardware architectures. The work presented is focused on real-time pattern recognition VLSI architectures, particularly on convolutional neural networks for both frame-based and event-based paradigms.

Initially, two types of objectives were proposed: general objectives, with the aim of analyzing and studying the viability of event-based algorithms in the context of vision pattern recognition and their implementation on VLSI systems, and specific objectives, related to solving problems regarding the topic introduced previously and the development of the architectures.

General objectives: to study pattern recognition algorithms based on spiking processing and their hardware implementation for feature extraction of visual information.

1. Study pattern recognition techniques and hardware architectures to extract features from neuromorphic vision sensors.

2. Develop real-time VLSI architectures to accelerate and infer pattern recognition algorithms in real-time.

The purpose of these general objectives is to mimic the behaviour of neural systems, understanding how the brain works, in order to develop high-performance hardware architectures and make use of spiking information. With the aim of fulfilling these general objectives, a set of specific objectives were proposed.

Specific objectives: analysis and implementation of different techniques in order to perform vision classification in VLSI for frame-based and event-based processing based on pattern recognition algorithms.

1. Frame-based feature extraction through convolutional neural networks
   (a) Study of CNNs theory for image classification.
   (b) Train and test CNNs using Caffe and ADaPTION deep learning frameworks.
   (c) Implementation using FPGA technology to accelerate CNN inference using OpenCL and SoC technologies.
   (d) Analysis and performance study of FPGA and SoC-FPGA platforms for CNN inference.
2. Event-based spiking convolution accelerators.
   (a) Study of spiking convolution theory for event-based processing.
   (b) Analysis of the state of the art of current spiking convolution architectures.
   (c) Design of spiking convolution processors architecture for FPGA.
   (d) Frame-based to Event-based dataset generation and conversion.
   (e) Test, analysis and evaluation of the architectures developed to implement SCNNs.
3. Pattern recognition based on Hierarchy of time surfaces.
   (a) Study of the theory of time surfaces for feature extraction.
   (b) Implementation of a VLSI architecture to inference a hierarchy of time surfaces network.
   (c) Verification of the architecture classifying event-based image vision data sets.
   (d) Evaluation and analysis of the design implemented on a FPGA platform.
   (e) Development of new memory models to reduce memory bottlenecks in event-based processing systems.
   (f) Study and analysis of memory model in hierarchy of time surfaces networks.

2.2 Thesis structure

The thesis presented is structured in four chapters, following the specific objectives previously detailed, within a conclusion chapter. Each chapter is briefly described below:

- Chapter 3: Convolutional Neural Network Accelerators
  This chapter is focused on the implementation of CNNs accelerators on FPGA platforms, and it explains two different FPGA implementations: the first one consists of a CNN accelerator based on Altera OpenCL technology; then, the chapter ends with the description of CNN accelerators based on FPGA-SoC architecture, called NullHop.

- Chapter 4: Spiking Convolution Accelerators
  In this chapter, three different spiking convolution processor architectures for FPGA are described. The spiking processor is based on LIF neuron behavior, processing data row-by-row, which reduces memory bottlenecks and, consequently, the latency.

- Chapter 5: Pattern Recognition Based on Time Surfaces in Real Time
  This chapter describes an FPGA implementation of an event-based algorithm, called HOTS, which uses a novel concept of time surfaces to extract features from visual events. Furthermore, a new memory model for event-based processing systems is presented; this memory stores the most recent events and it is tested under a HOTS network.

- Chapter 6: Conclusions and Future works

  This chapter presents the conclusions from the research of this thesis, with the main articles and future works.

Chapter 3

Convolutional Neural Network Accelerators

> "The two most powerful warriors are patience and time."
>
> Leo Tolstoy

Convolutional neural networks (**CNN/ConvNet**) have become, in recent years, one of the most popular solutions to solve complex problems in several fields, such as audio processing or machine vision, for detection or classification. The major advantages of this kind of networks are their simple supervised training using backpropagation and their high classification accuracy. On the other hand, CNNs are computationally expensive, due to the large number of complex operations performed. CNNs consist of two phases: training, where weights are adjusted during several iterations with the aim of obtaining a better accuracy in each step, and inference, where the model is deployed to perform a pattern recognition task. The training step requires several iterations; therefore, CNN frameworks, such as CAFFE (Jia et al., 2014) or Theano (Bastien et al., 2012) support the use of graphics processing units (**GPUs**), since their massive parallelism can reduce the computation time to train large network models.

Although GPUs parallelism can be used to reduce the time of the training and inference steps, their power consumption and physical size do not make these devices ideal for embedded systems. CNN operations can be divided into partial operations that can be performed in a parallel way. According to this idea, the tendency in recent years is to design CNN accelerators, based on application-specific integrated circuit (**ASIC**) or field-programmable gate array (**FPGA**) technology, to deploy these networks.

This chapter is divided into three parts. The first part explains the fundamentals of CNNs, followed by a second part, which presents a CNN inference using OpenCL language for an Altera FPGA. This solution is similar to a GPU inference, although it uses the reconfigurable property of FPGAs to improve the performance. The results for different implementations are analyzed and discussed in terms of latency, area and power consumption. Finally, the last section explains the implementation of a CNN accelerator, called NullHop, in a Xilinx FPGA. This accelerator reduces the number of operations skipping the computation of null pixels and it proposes a novel real-time hardware solution inside the state-of–the-art in CNNs.

CNNs are inspired by biological neurons of the primary visual cortex of the brain, where each neuron from layer i is connected only to a subset of neurons in layer i + 1, known as projective field. CNNs follow the same structure of other neural networks, which have an input layer, an output layer and several hidden layers in

between. The main difference of CNNs is that convolution operations are applied to extract features from images. Apart from this layer, this kind of network is composed of other layers that simplify the computation, reducing the amount of data. CNNs also implement other layers to classify the output. This section describes the most frequently used layers in CNNs.

3.0.1 Convolution layer

The convolution layer performs the convolution operation over an input, sending the result to the next layer. The convolution operation uses a filter to process the input data. Those filters, also known as convolution kernels, are matrices, whose elements are called weights. The characteristics of the input data are grouped in feature maps. In a CNN, the convolution layers process these input feature maps by convolving their data (Equation 3.1), thus generating new output feature maps for the next layer. Equation 3.2 shows the operations for one convolution layer, where R and Q represents the input and output feature maps, respectively.

$$W_{rq} * x_q = z \tag{3.1}$$

$$z(m,n) = \sum_{k=0}^{K-1} \sum_{l=0}^{L-1} W_{rq}(k,l) * x_q(m+k, n+l) \tag{3.2}$$

Filter weights are computed during the training phase using an algorithm known as backpropagation (LeCun et al., 1989). The backpropagation algorithm propagates the error calculated at the output to the previous layers, adjusting weights coefficients and minimizing the error.

3.0.2 Pooling layer

The purpose of the pooling layer is to reduce the spatial dimension, in order to decrease the number of parameters and computation in the network. This shortens the training time and controls overfitting. This kind of layer is not affected during the training phase, as it always performs the same operation without any kind of parameter. The pooling layer combines neurons of an output feature map into a single neuron of the next layer. The most common pooling functions are:

3.0.2.1 Max-pooling

It is the most commonly used function in this layer. This function obtains the most representative value of a region of interest, reducing its size. Fig. 3.1 shows an example of this function, where an output feature map is divided into groups of 2x2 and, then, these groups are replaced with the maximum value inside them.

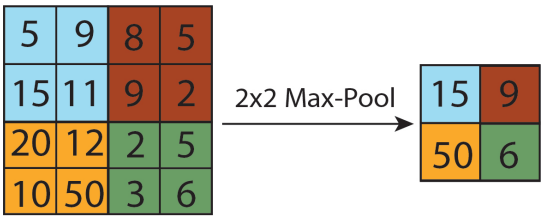


FIGURE 3.1: Max-pooling.

3.0.2.2 Average-pooling

Average pooling computes the arithmetic average of the values inside each region of interest, instead of the maximum value used in max-pooling. Fig. 3.2 shows an example of how average pooling is computed.

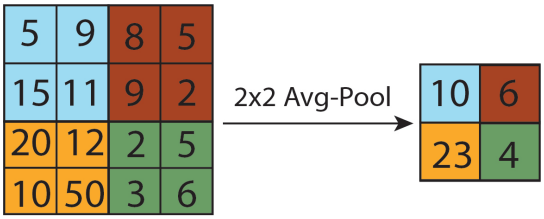


FIGURE 3.2: Avg-pooling.

3.0.3 Dropout layer

The dropout function was firstly introduced by Srivastava et al., 2014 with the aim of reducing the overfitting. Overfitting happens when a model works extremely well with the training set. This implies that the model can lose accuracy with other images that differ from the ones of the training set. The dropout layer suppresses a group of neurons during training, thus other neurons will have to step in and handle the representation required to make predictions for the missing neurons. The effect is that the network becomes less sensitive to the specific weights of neurons. This results in a network that can generalize more efficiently and which is less likely to overfit the training data. It is usually employed in very deep CNNs with a large number of input and output feature maps. They were not used in the CNN models presented in this work.

3.0.4 Fully-connected layer.

The fully connected layer works as a classifier, and consists of two interconnected layers of neurons that follow an all-to-all pattern. This kind of layer is the last one in a CNN structure, since they classify the resulting features obtained from the previous convolution layers. The last fully-connected layer output matches with the number of classes to be classified by the network.

3.0.5 ReLU layer.

The ReLU layer is a rectifier layer that applies the function shown in Equation 3.3. This layer filters negative values, transforming them into 0. This layer is commonly used in CNNs deployed in hardware, since, in further layers, the computation of pixels whose value is 0 is avoided, reducing the number of operations and, consequently, the power consumption.

$$f(x) = max(0, x) \tag{3.3}$$

3.1 OpenCL

As part of the Neuromorphic Processor Project (**NPP**), a CNN accelerator based on OpenCL for FPGA was developed. OpenCL is a free standard API for the parallel

programming of diverse computing platforms, such as personal computers, high-end servers, GPUs and FPGAs. The standard API provides an easy programming model and an elegant low-level abstraction that allows programmers to develop parallel applications for different hardware platforms, ignoring the knowledge of the internal architecture.

The abstraction mentioned before is based on a hierarchy of models: platform model, memory model and execution model. The knowledge of these models allows programmers to optimize the application by making use of computing resources for the target problem. The following sections explain the OpenCL hierarchy in detail.

3.1.1 Platform model

OpenCL considers each non-host processor as "devices" that are divided into one or more computer units (**CU**), which are further divided into one or more processing elements (**PE**), with the latter being the basic unit where computation is performed. An OpenCL application is implemented as a host code and a device kernel code. The application runs in the host processor (here in after referred to as "host"), which executes the commands to prepare global memory buffers, communicating with devices in order to send or receive data.

On the other hand, an OpenCL *device* executes the computation commands into the processing elements. Fig. 3.3 represents an OpenCL platform model for one *host* with multiple *devices*.

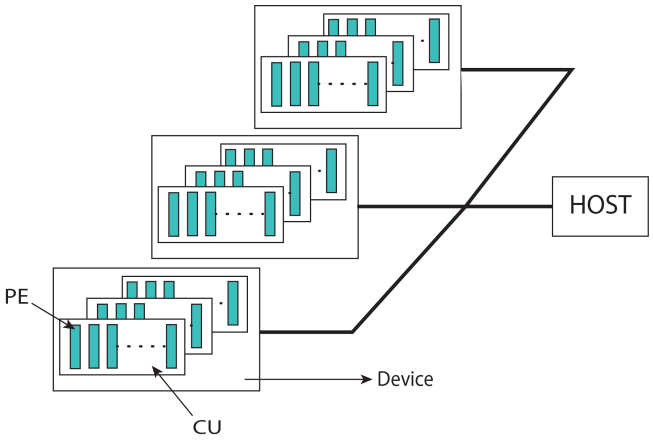


FIGURE 3.3: OpenCL platform model.

3.1.2 Execution model

The OpenCL execution model is divided into two units of execution: kernels, which are executed in OpenCL *devices*, and a host program, which runs in the *host*. In an OpenCL application, the computational work is performed by the kernels; therefore, the key of the OpenCL execution model is how kernels are executed. When an enqueued kernel command submits a kernel for execution, an index space is defined. The function defined in the kernel is executed for each point in the index space. Each of these functions is called a work-item and it is computed in the PEs. However, these work-items are grouped in work-groups, which are managed by the *device*.

The space index mentioned before is called the NDRange. The NDRange is a N-dimension index space of up to 3 dimensions. The NDRange (global_size) is divided into work-groups (local_size) and the work-items to be processed. An example of

how the NDRange is decomposed into work-groups for kernel execution is shown in Fig. 3.4. In the example of Fig. 3.4, there is a 3D NDRange with a global_size of 16x16x16 and a local_size of 4x4x4; this computation is divided into 64 work-groups of 64 work-items each.

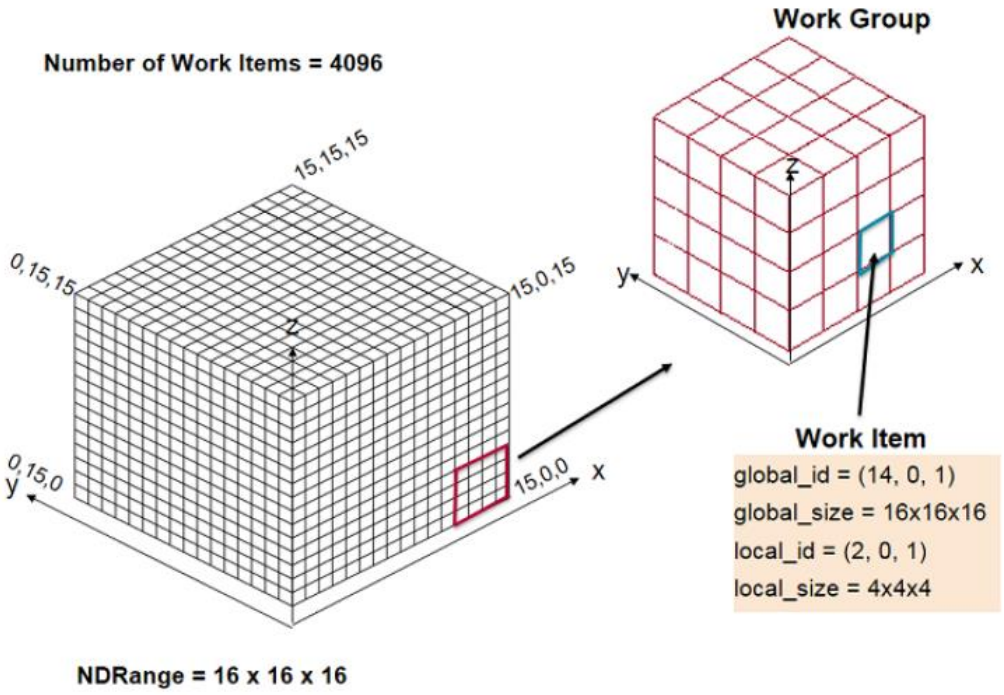


FIGURE 3.4: OpenCL index space.

3.1.3 Memory model

In OpenCL, there are two different types of memory: the *host memory* and the *device memory*. *Host memory* is the memory of the *host*, the behaviour of this is defined out of the OpenCL context. On the other hand, the *device memory* corresponds to the available memory for kernels in OpenCL devices. However, the *device memory* is composed of four memory regions:

- **Global memory:** A memory region is shared by the *host* and the *device*, in order to send or receive data. However, this memory is blocked for the *host* during the computation until the kernel finishes and the data are ready to be read by the *host*.
- **Constant memory:** A region of the global memory that remains constant along the kernel execution. The *host* initializes and allocates the parameters in this region.
- **Local memory:** A memory region local to a work-group. It is used to allocate variables shared by all work-items of that work-group.
- **Private memory:** Private memory space for a work-item. The variables of this memory region are not visible to another work-item.

An example of how memory is organized in an OpenCL device is shown in Fig. 3.5.

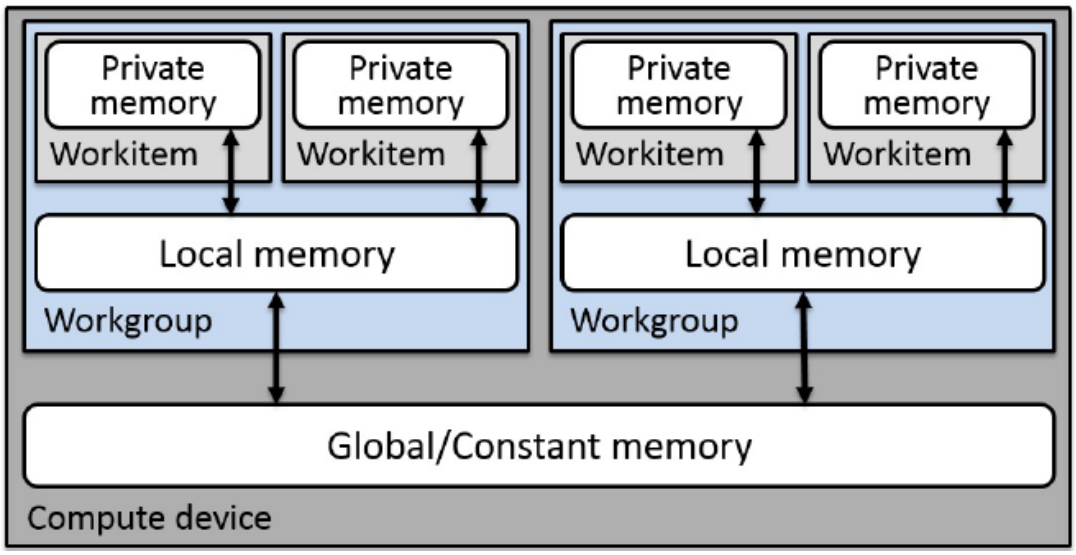


FIGURE 3.5: OpenCL device memory model.

Although OpenCL is a standard, the memory model is highly dependent on the platforms where the OpenCL application will be deployed. In this work, the OpenCL device was a Terasic-DE5 board donated by Intel-Altera. DE-5 contains a Stratix-V FPGA, which has two different kinds of memory: off-chip memory, which is composed of an 8GB DDR memory that is external to the programmable logic of the FPGA, and on-chip memory, which consists of block ram memories (**BRAM**) of the FPGA.

The main memory resources inside an FPGA are BRAM memories. The BRAM memories are commonly used to store a large set of data in FPGAs. The two types of BRAMs available in a device can hold either 18k or 36k bits, and the available amount of these memories is device specific. The global memory is stored in the off-chip memory, whereas the local memory will be allocated in BRAM.

3.1.4 CNN inference using Altera OpenCL

In this work, the Caffe framework (Jia et al., 2014) was used to train the network, since it is the most used framework in the literature, as it describes the network layers easily. Moreover, after the training step, the weights can be extracted easily from the trained model to be used in any application.

The network deployed in this work is a modified version of the LeNet-5 (Lecun et al., 1998), which was developed to recognize handwritten digits. Fig. 3.6 represents the whole network, which has a first convolution layer of 20 filters of 5x5 with 1 input feature map of 28x28 that corresponds to the input image, followed by a max-pooling layer. The second convolution layer has an input of 20 feature maps, generating 50 output feature maps and also followed by a second max-pooling layer with ReLU. Finally, there are two fully-connected layers: one with 500 neurons connected to another fully-connected layer of 10 neurons, matching the number of digits to be recognized.

The CNN is implemented, converting each layer in an OpenCL kernel. The *host* is in charge of receiving the data of one layer and sending it to the following one. This is due to the fact that, in OpenCL, the host application has the control of the execution, allocating in the memory the input and output buffers of each kernel. During the execution, each kernel reads and writes data into the global

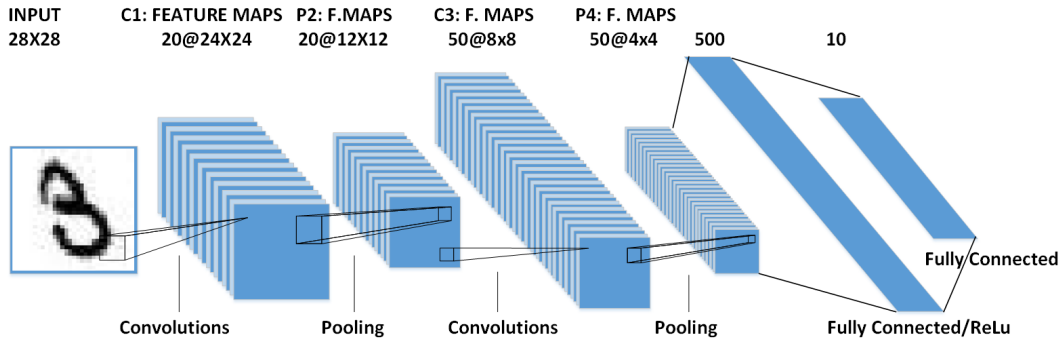


FIGURE 3.6: LeNet5 network architecture.

memory, to be read by the *host* after each layer's processing. In order to reduce the latency, the kernels can obtain a better performance through some directives, such as increasing the number of CUs for a kernel, unrolling complex loops (**UNROLL**) or vectorizing data access (single instruction multiple data) (**SIMD**). A diagram of OpenCL kernels execution for LeNet5 is shown in Fig. 3.7. In this implementation, the first convolution and pooling layer are merged into one kernel, whereas the other layers are separated into different kernels.

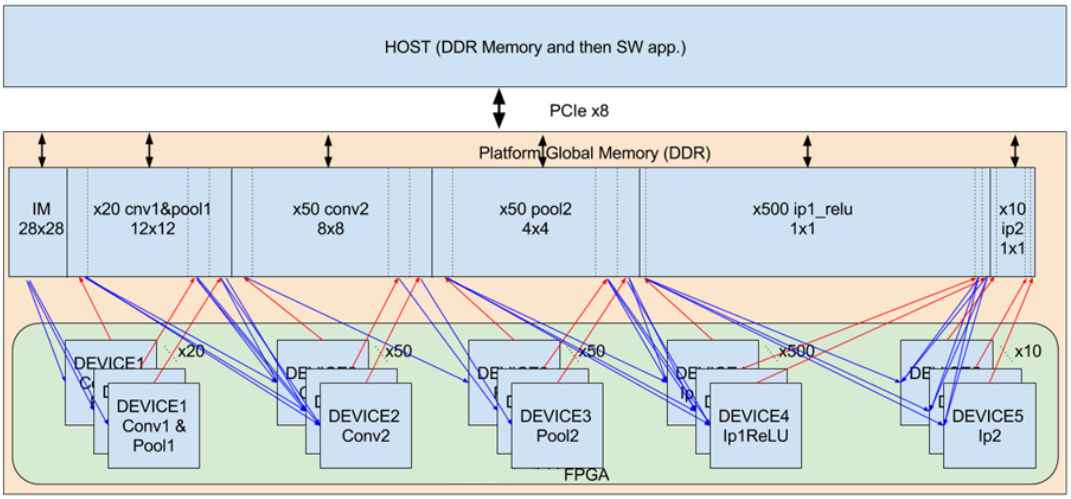


FIGURE 3.7: LeNet5 OpenCL implementation.

3.1.5 Results

With the aim of testing the behaviour of the OpenCL FPGA implementation, a real-time test was performed. The test consisted in processing frames captured from a camera, measuring the average processing time of each layer. Each frame was processed in three different scenarios with different parallelism directives:

- **No parallelism:** The kernel executes without any kind of parallelism.
- **Unroll:** Loops are unrolled, increasing parallelism, thus the input throughput is higher.
- **SIMD:** The kernel architecture is adapted to access data in parallel, such as vector processors. This solution increases parallelism as much as possible.

The execution time and FPGA resources for each scenario are shown in Table 3.1. When parallelism increases, logic cells, digital signal processors (**DSP**) and BRAM resources also increase, since more FPGA resources are needed to unroll loops or vectorize data processing. However, the execution time also increases with parallelism, due to the fact that the memory bandwidth generates a bottleneck when multiple kernels access the global memory. This slows the kernels down, increasing the execution time. Therefore, this solution does not allow deploying real-time CNNs, since the memory bandwidth is too low, even for a small CNN as the one used in this chapter.

Altera OpenCL provides a mechanism that allows communications between each kernel, called pipes. Kernels can send and receive data from one kernel to another using pipes, creating a pipeline processing flow that reduces the time wasted in the host communication after each kernel's execution. Although this solution reduces the global memory bottleneck, pipes do not belong to the OpenCL standard; therefore, this mechanism is only valid for Altera OpenCL platforms. In addition, it would require a communication protocol between kernels, thus more FPGA resources would be consumed.

In addition to the real-time latency problem, this kind of platform depends on the peripheral component interconnect (**PCI**) interface and they also have a high power consumption. Therefore, these hardware solutions are not appropriate for embedded systems.

TABLE 3.1: Altera OpenCL results for each scenario; no parallelism, unroll and simd.

Convolution Layer	Exec Time (ms)	Logic Cells /Elem.(K)	DSP slices	BRAM(Kb)
Conv_pool1	1.01 / 1.01 / 0.98	145.7 /42.3 / 73.7	8 / 31 / 57	5225 /6205 / 11200
Conv2	3.95 / 3.96 / 4.27	300.5 / 34.0 / 34.0	8 / 31 /31	3207 /4882 / 4900
Pool2	0.06 / 0.07 / 0.13	6.9 / 6.9 / 6.8	2 / 2 / 2	279 / 273 / 279
Ip1_ReLU	1.01 / 1.81 / 2.02	5.8 / 5.8 / 5.8	4 / 4 / 4	1471 / 1470 / 1500
Ip2	0.15 / 0.14 / 0.13	5.7 / 5.7/ 5.7	4 / 4 / 4	1471 /1470 / 1500

3.2 NN-X architecture

Altera OpenCL presents a huge memory bottleneck between *host* and *device*, since the kernels have to wait for all the data from previous layers to perform the computation. On the other hand, other FPGA/ASIC solutions improve the number of operations per second by performing the convolution when there are enough pixels, as is shown in Fig. 3.8.

In order to reduce the impact of memory bandwidth to implement a real-time convolution accelerator, other CNN accelerators, such as NN-X or Origami, were studied and implemented. The RTC group was in charge of simulating and implementing the NN-X architecture, whereas the Institute of Neuroinformatics (INI) from the ETH of Zurich developed the Origami accelerator.

The NN-X accelerator, later known as Snowflake, is a system-on-chip (**SoC**)/FPGA implementation proposed by Gokhale et al., 2014. NN-X is composed of two parts: an ARM processor (Berger, 2007, Joshi Vaibhav Vijay, Balbhim Bansode and ARM Limited, 2013) which is in charge of configuring the accelerator and controlling transfers of input data; and the NN-X coprocessor, where

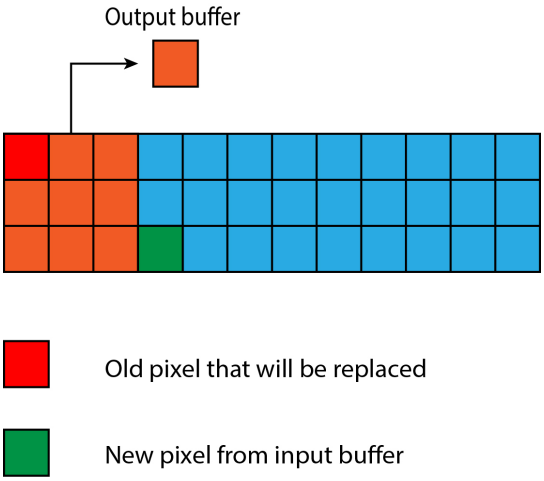


FIGURE 3.8: Input/output data sequencing by rows in a CNN accelerator.

convolutions are performed among other operations. The NN-X coprocessor is composed of the following three elements:

- **Collections:** where the computation of CNN is performed. These modules also perform different operations apart from convolution, such as pooling and non-linear operations. In addition to computation submodules, it also contains a router to communicate with other collections or with the memory router.
- **Memory router:** this module is in charge of transferring data between the external memory and the collections.
- **Configuration bus:** it connects the ARM processor to the collections in order to configure the different parameters.

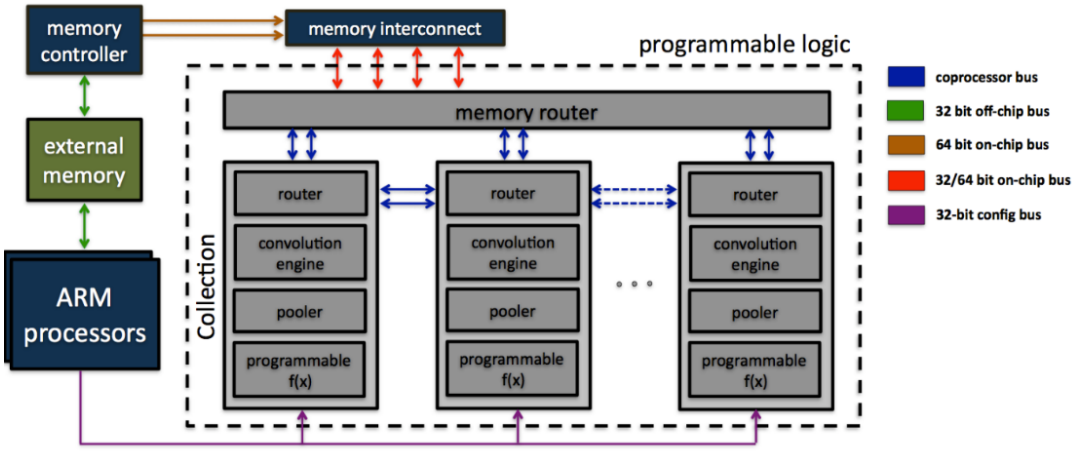


FIGURE 3.9: NN-X architecture. Image taken from (Gokhale et al., 2014)

The NN-X coprocessor was simulated using a mixed signal architecture simulator, called Chimera-SIM. Chimera SIM is an evolution of the discrete event simulator developed at Cornell University during the IBM TrueNorth

development. The simulator gives a high-level programming tool in C++/python that allows for rapid prototyping.

Although some components, such as a single collection with a memory router, were implemented in a correct way, both NN-X and Origami implementations were discarded for two reasons: the low number of convolutions that both architectures were able to perform and the fact that they do not use novel techniques, such as zero-skip, to reduce the number of operations.

3.3 NullHop Accelerator

In collaboration with the INI group from the ETH of Zurich, a CNN accelerator called NullHop (Aimar et al., 2018) was developed. The main property of the NullHop accelerator is that it skips the computation of pixels whose value is zero, making use of the ReLU layer, whose resulting feature-maps activation is sparse, which means that the resulting feature-maps contain several 0 values. Fig. 3.10 represents the sparsity of two of the most popular CNNs in the literature: the GoogleNet (Szegedy et al., 2015) and VGG19 (Simonyan and Zisserman, 2014).

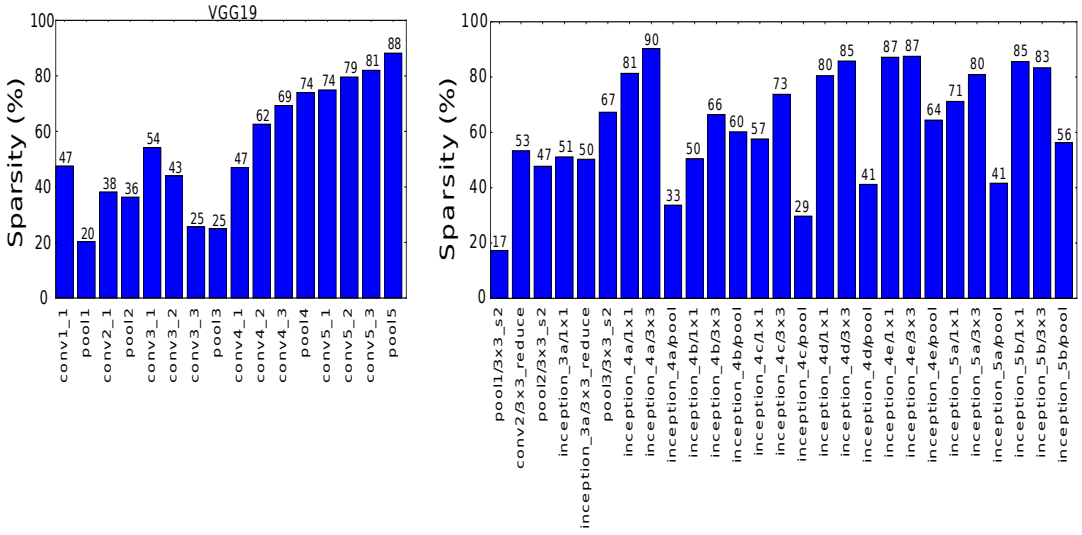


FIGURE 3.10: Sparsity VGG19 (left) and GoogleNet (right).

3.3.1 NullHop Architecture

The current version of the NullHop accelerator is able to accelerate a convolution layer with max-pooling and ReLU layers. Most deep neural network frameworks represent weights in floats of 32 bits. However, the current tendency in CNNs is to reduce bit precision, since it reduces the number of DSPs used in computation and, thus, the power consumption. The NullHop accelerator uses a precision of 16 bits codified in Q8.8 format.

After performing an entire convolution layer, NullHop is configured with the weights and parameters of the next convolution layer; then, non-zero pixels are sent. However, a mechanism to distinguish valid pixels from non-valid pixels is needed. Input feature maps are analyzed in order to detect non-zero pixels, creating a sparsity map that indicates with 1 bit which pixels are valid pixels ('1') and which ones are not ('0'). Fig. 3.11 shows an example of how compression is performed.

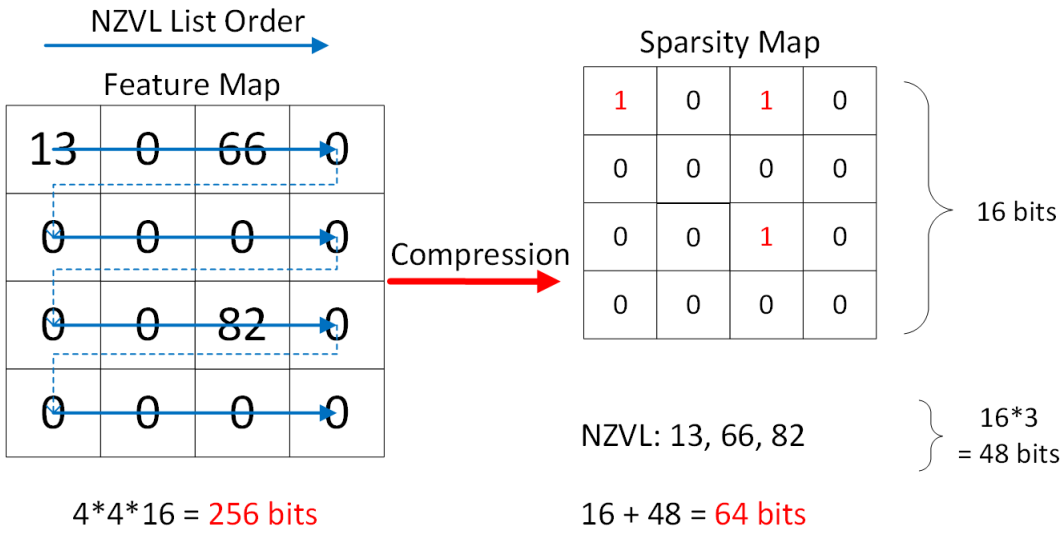


FIGURE 3.11: Sparsity map generation. Image taken from (Aimar et al., 2018).

Sparsity maps are sent to the accelerator as 16-bit words. Each sparsity map represents a row of 16 pixels, when a group of pixels is valid ('1'). The next words sent to the accelerator are the values of those valid pixels; otherwise, if all the pixels of a sparsity map are null pixels ('0'), the next word to be sent is the sparsity map of the next row of 16 pixels to be processed, as is shown in Fig. 3.12.

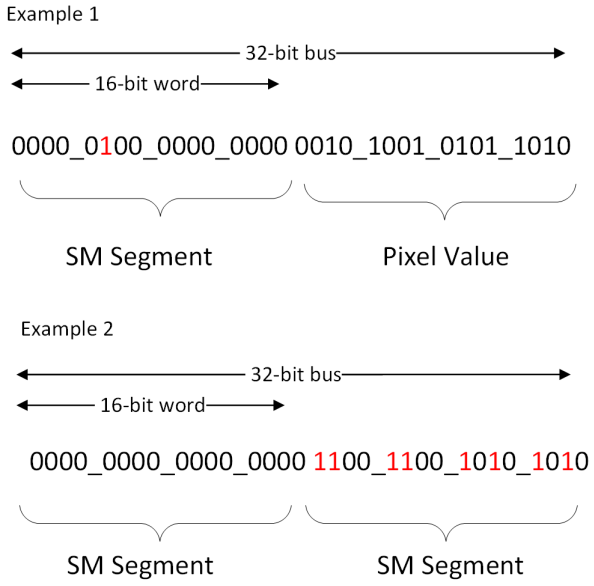


FIGURE 3.12: Sparsity map streams. Image taken from (Aimar et al., 2018).

The NullHop architecture consists of three modules: input decoding processor (**IDP**), compute core module (**CCM**), and pooling-ReLU-encoding module (**PRE**). The IDP module decodes the sparsity maps and stores the pixels in the memory. The computation is performed inside the CCM, which is composed of 128 MAC units. Each MAC has a controller, which is in charge of sending the pixels and the kernels' weights to perform the multiplication and accumulation operations, generating one

part of the output feature maps. The last module, called PRE, performs both pooling and ReLU layers, if these are enabled during configuration. In Fig. 3.13 is shown the NullHop architecture and its components.

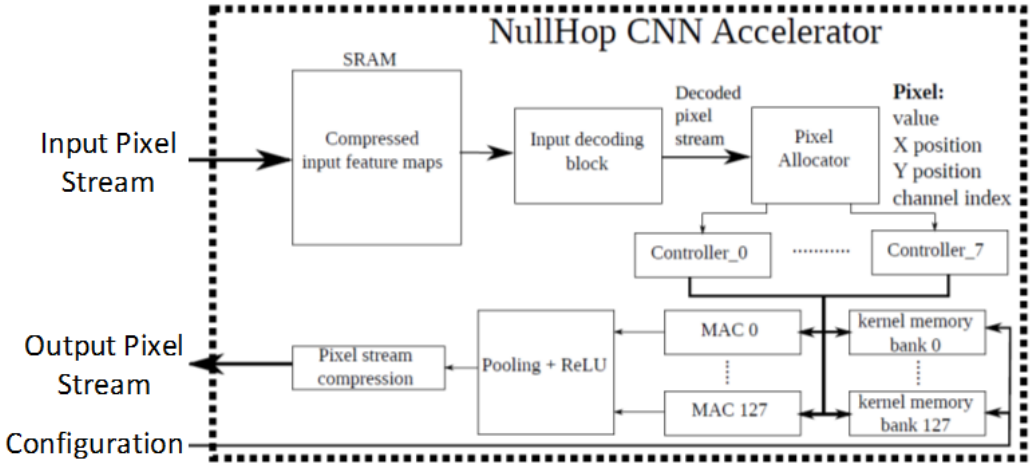


FIGURE 3.13: NullHop architecture.

As was previously mentioned, the main advantage of this kind of accelerator, in comparison with the one implemented in OpenCL, is that the computation can be started while pixels are coming, since pixels are received as streams. The OpenCL implementation does not allow processing data in a pipeline way, as all the kernels need to wait for all the data to be sent by the *host*. Current CNNs accelerators, such as Origami (Cavigelli and Benini, 2017), Snowflake (Gokhale et al., 2017) or Eyeriss (Chen et al., 2017), make use of this mechanism to increase the OP/s, creating pipeline stages between convolution layers. In Table 3.2, the NullHop FPGA and ASIC versions are compared with other implementations of CNNs accelerators in terms of efficiency and power consumption.

TABLE 3.2: Comparison of CNN accelerators.

	Eyeriss	Origami	Caffeine	Snowflake	NullHop
Technology/Platform	ASIC 65nm CMOS	ASIC 65nm CMOS	Ultrascale KU060	Zynq XC7Z045	GF 28nm Zynq XC7Z100
Frequency (MHz)	200	250	200	250	500/60
Precision	16-bit fixed	12-bit fixed	16-bit fixed	16-bit fixed	16-bit fixed
MAC units	168	196	1058	256	128
Performance (Gops/s)	46.1	145	310	116.5	471/17.2
Theoretical Max (Gops/s)	67.2	196	423.2	128	128/15
Consumption (W)	0.28	0.45	25	9.48	0.155/0.851
Computational Efficiency	68.6%	74%	73.3%	91%	368/143%

3.3.2 NullHop FPGA implementation

In order to develop a prototype, a Zynq 7100 FPGA board was used. This kind of platform has a Zynq processor, which contains a dual-core ARM-Cortex-A9, called processing system (**PS**), and a Kintex-7 as programmable logic (**PL**), in the same chip. This kind of platform allows executing software applications in the PS and communicates with the PL. In our implementation, Nullhop is programmed in the

PL, whereas PS is in charge of configuring and communicating with the accelerator, configuring it, sending frames and receiving the results of each CNN layer. This FPGA implementation was developed as part of the collaboration between INI and the Robotics and Technology Lab (**RTC**) of the University of Seville (to which the author belongs) as part of the NPP project.

3.3.2.1 AXI-INTERFACE

The PS communicates with the PL through an Advanced eXtensible Interface (**AXI**), which is a standard open-source protocol of ARM. AXI is a master-slave protocol, where the master sends the address of the slave with a control signal. The master answers sending data to be written (write operation) or the slave answers sending data (read operation). However, this protocol is designed for communications with devices where the data traffic is low.

In recent years a new version of this protocol was developed to increase the input throughput, called AXI-stream (AMBA, 2014). This new protocol allows transferring huge streams of data, ignoring the address step of the AXI protocol. In order to make as fast as possible the transfers between PL and PS this protocol was used, which is implemented by the AXI-DMA IP.

AXI-DMA IP has two different channels: MM2S, which reads from the memory and send streams of data to the PL, and S2MM, which receives streams from the PL and stores them in the memory. The connection between AXI and DMA is performed by two adaptation modules: MM2S2NH and NH2S2MM. Both modules consist of a state machine with a FIFO, which adapt the AXI protocol to the NullHop protocol.

3.3.2.2 Implementation Results

The design was synthesized at 60 MHz, due to the limitation of the critical path to MAC blocks. The NullHop implementation has 128 MAC and can store 512 input/output feature-maps of a maximum size of 512x512. Table 3.3 shows the total resource summary of the implementation. As can be observed, the design is limited by the consumption of LUT resources of the FPGA.

TABLE 3.3: NullHop FPGA resources.

Resource	LUT	FF	BRAM	DSP
IDP	**7.55%**	**0.41%**	**17.22%**	**0%**
MAC	**50.19%**	**16.99%**	**33.9%**	**6.30%**
PRE	**16.41%**	**1.88%**	**0%**	**0%**
AxiSTREAM	**1.25%**	**0.73%**	**1.46%**	**0%**

Regarding the power consumption, the whole system, including the PS and PL, has a total static power consumption of 317 mW and a dynamical consumption of 2006 mW. The PS consumes 76% (1532 mW) of the total dynamic power. The resulting 24% (474 mW) corresponds to the PL, distributed in its components: logic (2%), clocks (13%), signals (7%), BRAM (2%), and DSP and I/O (both < 1%). Fig. 3.14 shows the power consumption summary obtained from the syntheses tool.

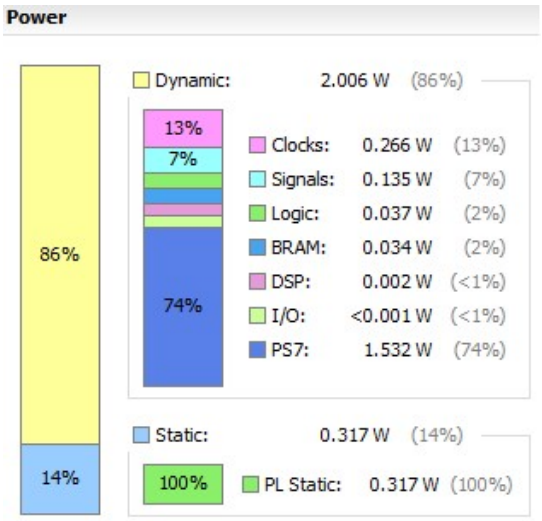


FIGURE 3.14: NullHop FPGA power consumption.

3.3.2.3 Real-time CNN inference

In order to test the behaviour of the architecture, a real-time demo was developed to test the performance. In this test, NullHop runs a CNN in real time, called Roshambo, which recognizes the symbols of the Roshambo game (rock, paper, scissors). This network implements 5 convolution layers of different kernel sizes, 1 of 5x5, 3 of 3x3 and 1 of 1x1, each followed by a max-pooling layer of 2x2 and a ReLU layer. The network performs 18 MOp, and it was developed to exploit the NullHop computation resources to the maximum. Fig. 3.15 is shows the structure of this network.

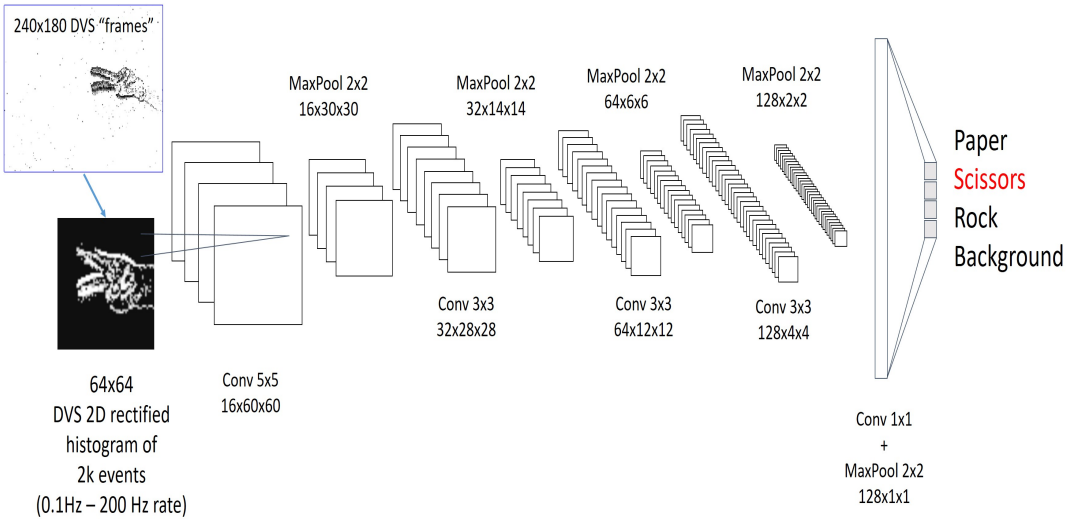


FIGURE 3.15: Roshambo CNN.

Zynq PS runs an embedded OS, called Petalinux, which is open-source and fully configurable. In this test, a Petalinux runs a software application called CAER. CAER collects events from a DAVIS-240C (Brandli et al., 2014) sensor connected to the Zynq board and integrates them as frames that are sent to the accelerator. Apart from the integration, CAER implements the NullHop and the AXI-DMA controller to configure each layer of the accelerator, send input feature maps and collect the

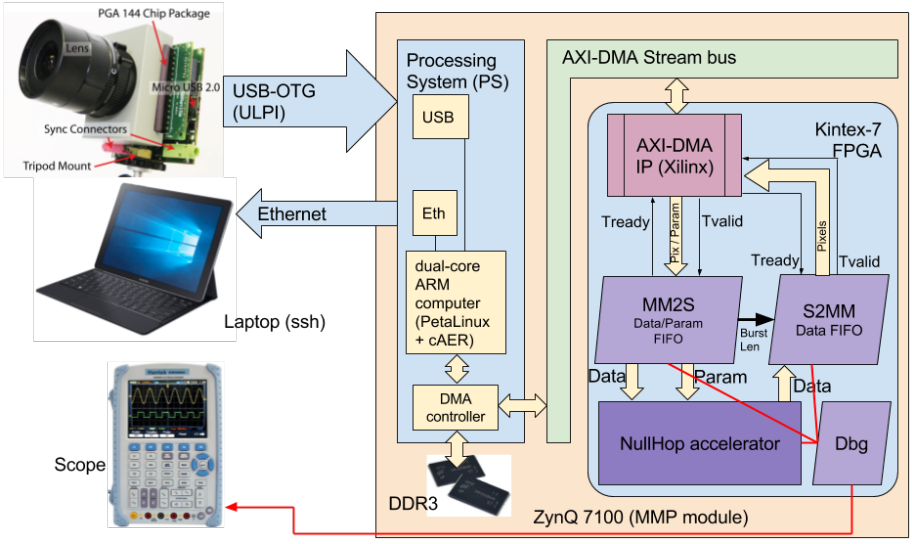


FIGURE 3.16: Block diagram of the NullHop test scenario.

results. Fig. 3.16 shows the block diagram of the SoC system implemented for the test proposed. The ARM processor configures each layer, sending the parameters and the input feature maps of the corresponding layer.

Each convolution and pooling layer are performed inside NullHop;, on the other hand, the fully-connected layer is implemented in software running in the ARM core.

The performance of the accelerator was obtained by measuring the time to process one frame. This experiment allows measuring the time spent by both the software and the hardware for each layer. The software execution time was measured using CAER, whereas the hardware execution time was analyzed using an oscilloscope, measuring the time that the MACs took to compute the feature-maps. As is shown in Fig. 3.17, the time to process one frame is 8 ms, with a frame-rate of 125 fps and a total of 2.25 GOp/s. The time consumed by the slowest layer is due to the the integration of frames of the CAER software. A possible solution to reduce the latency is to connect the sensor directly to the PL, integrating the events through hardware and reducing the computation load of the PS.

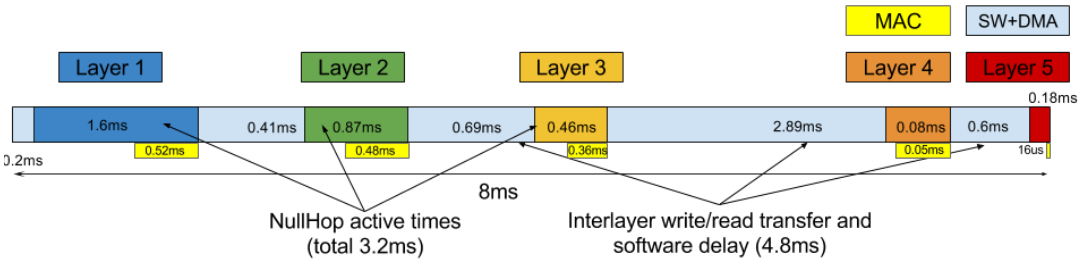


FIGURE 3.17: Roshambo timing analysis.

This kind of accelerators can perform a vast number of operations with a low power consumption. The tendency of CNNs is to be deeper, which allows them to recognize more patterns, thus implying more computation. Although this kind of accelerators are able to decrease the number of operations to be computed, it is possible to reduce the number of operations (therefore the power consumption) by

using other kinds of processing paradigms, such as neuromorphic event-based ones, which will be explored in further chapters.

Courbariaux, Bengio, and David, 2015 proposed a new approach to train CNNs using binary weights. This implementation consists of limiting the weights values to -1 or +1. This new approach can reduce the number of operations and memory resources. However, it is an extreme case of low-precision technique. The main problem in CNNs is the multiplication and accumulation operations and their high hardware cost in terms of DSPs. Although NullHop makes use of the sparsity representation of DVS to perform less computation, an integration of events into frames is needed, slowing down the processing flow, as was previously mentioned. In the next chapter, a spiking convolution processor based on neuromorphic processing paradigm is explained.

Chapter 4

Spiking Convolution Accelerators

> "Any man could, if he were so inclined, be the sculptor of his own brain."
>
> Santiago Ramón y Cajal

Spiking convolutional neural networks (**SCNN**) are inspired by ConvNets, and they are a type of spiking neural network. Although Convnets were developed for frame-based processing, several methods have been recently developed to transform from frame-driven to event-driven networks, as proposed by Pérez-Carrasco et al., 2013, Diehl et al., 2015, Rueckauer et al., 2017, whereas other methods directly train the event-driven networks with spikes (Orchard et al., 2015a, Stromatias et al., 2017a). Farabet et al., 2012 and Lee, Delbrück, and Pfeiffer, 2016 demonstrated that SCNNs provide a better solution in terms of accuracy, power consumption and speed than CNNs.

These spike techniques do not follow the all-to-all pattern connection of CNNs; the processing can start with the first arrival event from the sensor, generating results while the sensor is producing new events. This kind of network is implemented in several neuromorphic hardware platforms, such as IBM TrueNorth (Esser et al., 2016) or Spinnaker board (Serrano-Gotarredona et al., 2015). Although these neuroinspired platforms are large-scale modular computing systems with a vast number of neurons and synapses, these hardware solutions cannot inference this kind of network in real time, due to the fact that they cannot handle the high event traffic produced in spiking convolution layers.

This chapter presents three different FPGA designs that are capable of performing event-based convolutions in real time based on leaky integrate–and-fire neurons. The first version performs event-based convolutions by firing events with positive and negative polarities, using two different thresholds for both positive and negative membrane potentials. The neuron model implemented in this version does not implement the refractory period. The next design versions, apart from adding the refractory period of leaky integrate-and-fire neurons, discard the negative threshold, adding new computation mechanisms to improve the performance and the input throughput. The behaviour of each version is explained in the following sections in detail, with some performance tests using different input event traffic, in order to characterize the convolution processor in terms of speed, power consumption and performance.

4.1 Spiking convolution layers

4.1.1 Spiking convolution

Spiking convolution replicates the same concept of frame based convolution in the spike-domain. However, in event-based processing not all the pixels of a frame are processed; this is due to the fact that neuromorphic vision sensors, such as DVS (Lichtsteiner, Posch, and Delbrück, 2008) or ATIS (Posch, Matolin, and Wohlgenannt, 2011) only detect changes in the luminosity of a scene. Those changes are called events, and they are represented by an address (x,y), which, in turn, represents the pixel position and a polarity; the latter indicates whether that pixel's luminosity is higher than the previous scene (ON) or lower (OFF). Unlike frame-based cameras, where all the pixels of a scene are scanned, even if there is no activity, neuromorphic sensors represent a moving reality, where only those pixels that have changed are detected, reducing pixel computation in machine vision processing.

In a spiking convolution an input image X is coded in such a way that each pixel X(i,j) is represented by a number of events of a visual source output (DVS or ATIS sensors). The results of convolution operations are stored in a Y matrix (capacitors for analog circuits or registers or RAM cells for digital circuits). When an input event arrives, the corresponding pixel and its neighbors are modified in Y by adding the convolution kernel. The following Equation (4.1) shows the operation for computing each incoming event with address (i,j):

$$\begin{gathered} Y\left(i+a, j+b\right) = Y\left(i+a, j+b\right) + K\left(a, b\right), \forall a, b \\ a\epsilon\left[-\frac{N}{2}, \frac{N}{2}\right], b\epsilon\left[-\frac{M}{2}, \frac{M}{2}\right], N, M = dim\left(K\right) \end{gathered} \tag{4.1}$$

Once all the events of pixel X(i,j) have been received and calculated, the integrator value of the corresponding address Y(i,j) accumulates X(i+a,j+b) $\forall(a,b)$, times the value of the kernel. In other words, we are adding the kernel value to a neighbor of outputs as many times as the number of input events. This continuous addition is equivalent to multiplying the intensity of a pixel by a kernel value in frame-based convolutions. The output of the convolution operation, at this point, is stored in a matrix of integrators Y. The resulting matrix Y can be sent out in several ways. In this work, it is inspired by the leaky integrate-and-fire (**LIF**) neuron model. In order to accelerate the spike convolution, three different FPGA architectures were developed. Two of them have different LIF implementations and they will be explained in detail with the designs.

4.2 Subsampling layer

As was explained in the previous chapter, in a frame-based CNN, a convolution layer is usually followed by a pooling layer, reducing the spatial size of output feature maps and, thus, the computation in the next layers. In SCNNs, the pooling layer is easier to implement than in frame-based CNNs, since reducing the spatial size requires dividing the (x,y) address by two. It has been demonstrated in other works (Pérez-Carrasco et al., 2013) that this mechanism is valid for the implementation of a subsampling layer in a SCNN. An advantage of this implementation is that the hardware implementation only needs a shift register to perform the subsampling, as is shown in Fig. 4.1.

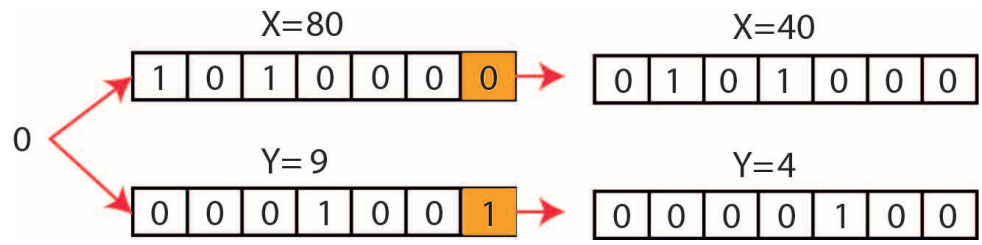


FIGURE 4.1: Example of event sub-sampling. The input event is shifted to the right, dividing its value by 2.

4.3 Leaky integrate-and-fire convolution processor V1

4.3.1 Leaky integrate.and-fire neuron model for V1 convolution processor

As was previously mentioned, the neuron model implemented in this work is the LIF model. The implemented LIF neuron model increases or decreases its membrane potential with the addition of kernel coefficients. When the membrane potential reaches a positive (**PTH**) or negative threshold (**NTH**), a spike is generated with the corresponding (x,y) address of the neuron and its polarity, resetting its membrane potential afterwards. The LIF neuron is inspired by the biological neuron, which reduces its membrane potential when it does not receive any excitation during a period of time. This property is called leakage. LIF neuron is inspired in biological neuron, which reduces its membrane potential if it does not receive any excitation during a period of time. This property is called leakage.

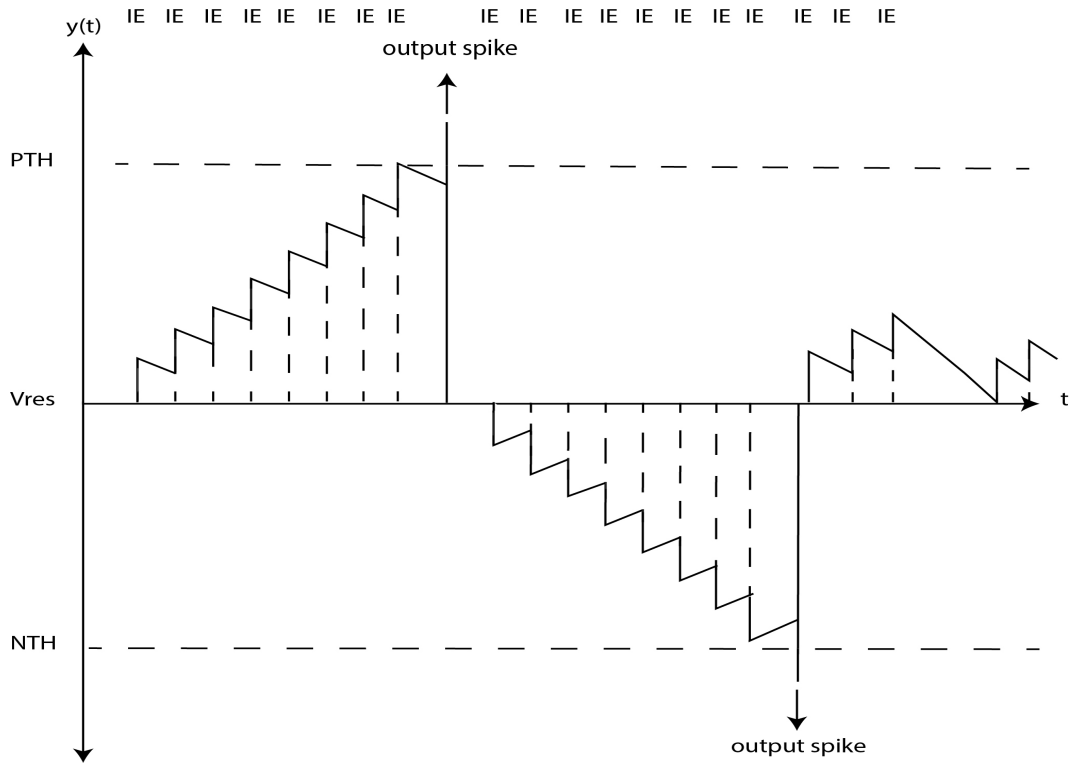


FIGURE 4.2: LIF behaviour with several input events (IE).

Leakage is implemented in the processor, since neuron that do not receive any excitation do not give information about the actual scene. In this implementation, leakage time, which is the time to apply the leakage, and its reduced potential value are fully configurable parameters that allow controlling the event traffic. This fact implies that a small leakage time would reset neurons more often. On the other hand, larger values for decay time lead neurons to reset their potential rarely, increasing the number of output spikes. Fig. 4.2 represents the behaviour of the implemented neuron.

4.3.2 Spiking convolution processor V1 architecture

In the literature there are several implementations of event-based convolution processors for FPGA (Linares-Barranco et al., 2010, Camuñas-Mesa et al., 2018). These implementations access data pixel-by-pixel, increasing the number of memory accesses and memory bottlenecks, thus increasing the latency. The solution proposed improves the performance, as the data are accessed row-by-row instead of pixel-by-pixel, reducing the time to perform a convolution with a higher input throughput. There were several attempts to implement a row-by-row convolution processor in FPGAs. However, there was an important limitation, as the BRAM memory was completely consumed by few convolution processors, since the synthesis tool provides all the memory available to a small number of convolution modules. The main idea of the architecture presented in this chapter is to use a memory arbiter that shares all the memory with the convolution modules, minimizing the consumption of memory resources.

Fig. 4.3 shows a high-level schematic of the event-based row-by-row processor architecture. The processor has three interfaces: two address event representation (**AER**) interfaces, which allow the processor to connect other neuromorphic sensors to receive and send events, and a 32-bit digital interface to connect with a host microcontroller to configure the system.

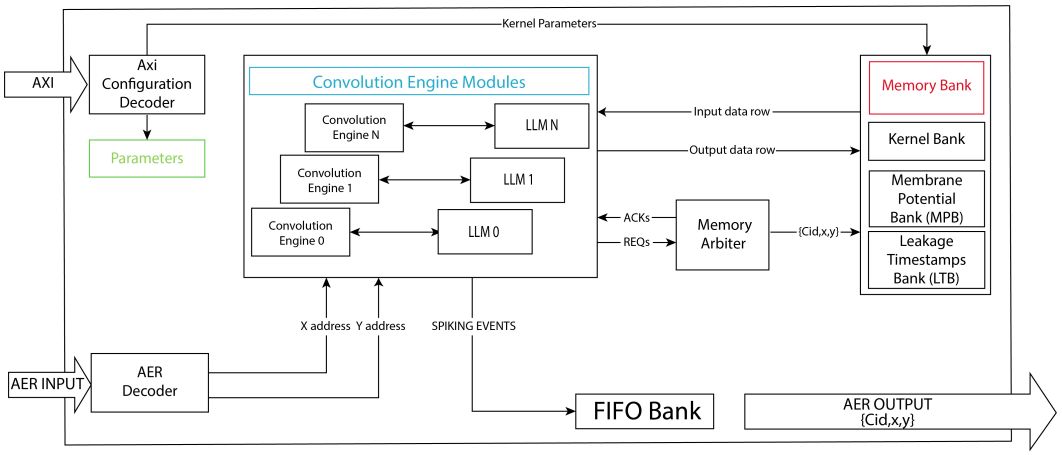


FIGURE 4.3: Convolution processor V1 schematic.

The system is able to compute a maximum number of 64 convolution operations in parallel with kernel sizes from 1x1 to 7x7. These convolutions are computed row-by-row in a fully shuffled way, where one complete row is computed per clock cycle. The engine is able to perform the spiking pooling operation, decreasing the spatial size of the representation to reduce computation in a SCNN.

First of all, let us provide a high-level overview of the processing pipeline. When an input event arrives, each convolution engine reads the membrane potential and leakage timestamps from the memory. These values correspond to neurons around the address of the received event and to the size of the programmed kernel, and they are read row-by-row. The convolution engine compares the timestamps with a global counter for leakage, in order to verify whether leakage must be applied. Then, the convolution engine convolves a membrane potential row with a kernel row. The neurons that reach the threshold fire, producing an event with (x,y) address and their polarity. This process is repeated until all rows are convolved by all kernel rows.

The following subsections describe the functional blocks of the accelerator and the processing pipeline in detail.

4.3.2.1 Membrane potential, leakage timestamps and kernel BRAM banks

In previous chapters, it was mentioned that BRAMs are the main memory resources to store large amounts of data in FPGAs. In this design, BRAM is divided into three different memory banks: one bank stores the membrane potential, the second one stores the leakage timestamps and the last bank stores the values of the kernels.

Memory banks for membrane potential (**MPB**) and leakage timestamps (**LTB**) are organized in multiple blocks that store values in rows of 8 pixels. When a convolution engine accesses memory data, it transmits three (x,y) addresses of the corresponding row to be convolved and the convolution ID (**CID**). The x address selects the memory blocks to be accessed, using a decoder to enable or disable the banks.

Memories store values for a maximum image size of 128x128 for all convolution engines. Since the memory banks are shared by all the convolution engines, the CID specifies the memory region for the corresponding engine and the y address selects which pixel row is read/written.

Each BRAM memory row stores 8 pixels with a resolution of 8 bits, thus for a 128-pixel row, 16-BRAM blocks are needed. The depth of this memory is related to the number of convolution engines (**N**) multiplied by the number of rows of an image, which is 128.

During the processing of an event, the convolution engine always reads one row of two consecutive BRAMs: the one where the input event belongs and the neighbor row. The reason for this multiple read is that neighbor pixels of the input event can be involved in the convolution operation, but they can be stored in a different bank. Fig. 4.4 shows an example of how MPB and LTB are read/written.

Following the same concept, the kernel memory is organized in one shared block, where KVs are stored row-by-row. When a kernel row is read, a mask (**CM**) is generated in order to let the convolution engines know which pixels of the combined membrane potential row will be convolved.

The memory is accessed using a memory arbiter, which grants access to the corresponding convolution module. The memory arbiter receives request petitions from convolution modules, answering with an acknowledgment which indicates that the corresponding module can access the shared memory. The arbiter implemented is a priority arbiter, which means that lower convolution modules access the memory earlier. However, when memory access is granted to a convolution module, it will not be granted again until the request from other modules is checked. This prevents one convolution engine from owning the memory access.

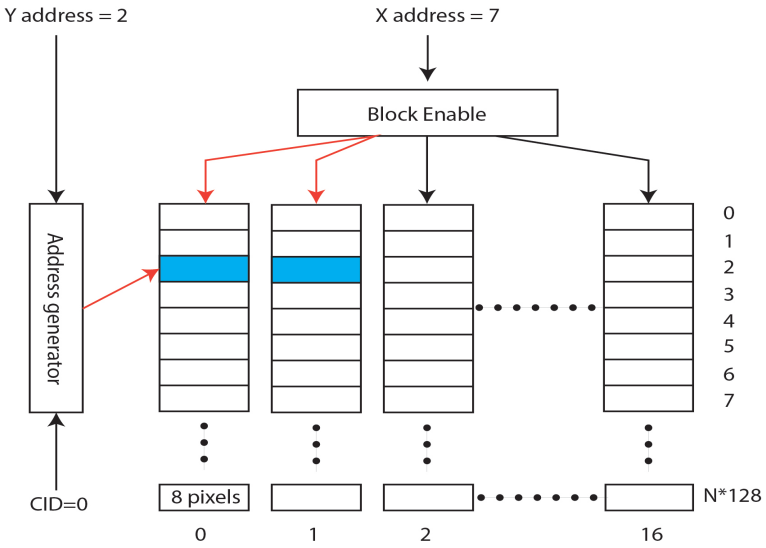


FIGURE 4.4: Example of BRAM access.

4.3.2.2 Leakage system

One of the main properties of LIF neuron is leakage. The purpose of implementing leakage is to store, in BRAM for each neuron, a timestamp (leakage timestamp) that corresponds to the last time the neuron was accessed. During convolution, the leakage timestamp is compared with the counter of a global counter, called leakage counter (LC). If the temporal difference is higher than a configurable value, leakage is applied; otherwise, the convolution is performed without leakage. LC has a resolution of 32 bits, since storing 32 bits in BRAM for each neuron for all convolution engines would consume a large amount of resources; therefore, only 8 bits of the counter are stored. Those bits are selected using a sliding window that allows calibrating the leakage time resolution. Fig. 4.5 shows the counter mechanism implemented.

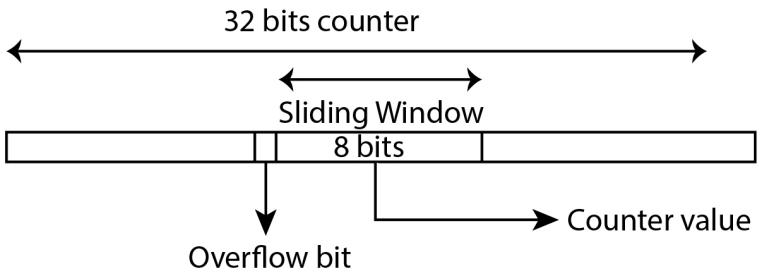


FIGURE 4.5: Leakage counter mechanism.

However, an overflow can occur in digital counters and leakage would be applied wrongly. To solve this problem, a mechanism based on distributed ram was developed.

Multiple lookup tables (**LUT**) in a slice memory (**SLICEM**) can be combined in diverse ways to store large amounts of data (**LUTRAM**). LUTRAM, or distributed

RAM, is crucial to many high-performance applications that require relatively small embedded RAM blocks, such as FIFOs or small register banks.

The solution proposed in this section consists of a leakage memory composed of a bank of arrays of 128x128 LUTRAM cells of 2 bits each, with one bank for each convolution engine. Fig. 4.6 shows how the leakage LUTRAM memory bank (**LLM**) works for a 3x3 kernel size. Initially ($t=t_0$), LLM cells have a value of 0 (Fig. 4.6.a). When LC produces an overflow ($\mathbf{L_{ov}}$), each cell adds 1 to its content (Fig. 4.6.b). When an input event (IE) is convolved, the content of those rows that have been accessed during the convolution, are reset (Fig. 4.6.c). If another overflow occurs, the cells increment their values again. Those cells that have reached a value of 2 indicate that a long time has passed since the last access to that neuron. Therefore, those neurons would have decreased their membrane potential to 0; thus, the next time one of those neurons is accessed, their membrane potential is reset (Fig. 4.6.d).

LLM is inside each convolution engine and it is also read/written row-by-row. When the leakage counter produces an overflow, the convolution engine cannot update the membrane potential memory until the content of each leakage memory has been updated, since the membrane potential could be reset. However, a convolution engine can read from the memory during the update operation, reducing the waiting time.

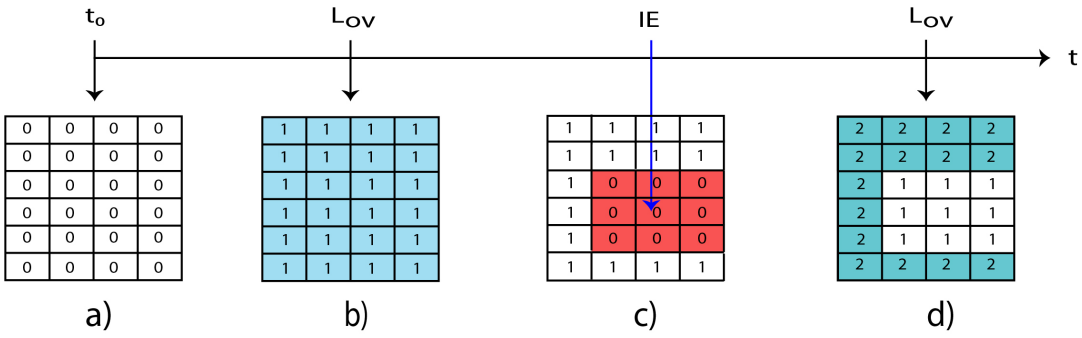


FIGURE 4.6: Leakage LUTRAM mechanism.

4.3.2.3 Convolution Engine

The convolution engine module is the most important module, since it is where the convolution is performed. This module consists of a state machine that communicates with the memories by reading membrane potentials, leakage timestamps and LLMs. Convolution engines are coordinated by a memory arbiter that gives them access to the memory. This arbiter solution allows all convolution engines to work in parallel by sharing BRAM, saving FPGA memory resources in an efficient way. When rows from the memories are read, the convolution starts checking the content of the leakage LUTRAM memories. If a long time has passed, the membrane potential is reset; otherwise, the convolution is performed. Before starting the convolution operation, LC and leakage timestamps are compared to apply decay, then the convolution is performed. If a membrane potential reaches a threshold, an event with the (x,y) of the neuron and its polarity are generated and sent to an output FIFO. The advantage of this implementation is that it decreases the number of memory accesses, since the data are read and convolved row-by-row.

Although convolution is performed correctly by the processor presented, the output event rate is quite high and needs to be stabilized in order to implement

a SCNN. Although the leakage can control the output event rate, when the activity of some neurons is higher, the information given by them can be redundant and also saturate the system.

4.4 Leaky integrate-and-fire convolution processor V2

As was previously mentioned, version 1 of the event-based convolution processor had a high output traffic rate to deploy a SCNN. CNNs implement the ReLU layer, which decreases the computation. However, some works implement ReLU in the spiking domain, implementing it as a group of neurons whose firing rate works as a LIF neuron with no refractory period (Cao, Chen, and Khosla, 2015), obtaining good accuracy compared with ConvNets (Diehl et al., 2015). In spite of its accuracy, this solution does not solve the event traffic problem in hardware platforms, since, if a group of neurons are excessively active, they will generate a large amount of spikes that can saturate FIFOs, creating bottlenecks. Another mechanism that can stabilize the traffic is the refractory period of LIF neurons. The refractory period does not allow neurons to fire until a period of time has been meet, reducing the firing rate, avoiding redundant information of overactive neurons and stabilizing the output.

The new version adds a new mechanism to implement the refractory period in the FPGA. In addition to this mechanism, some architectural features were developed to optimize the computation in convolution engines.

4.4.1 Leaky integrate-and-fire neuron model for V2 convolution processor

The LIF behavior has two different features compared to V1. Firstly, the refractory period has been added, which is a property of biological neurons that guarantees a separation time between two spikes generated by the same neuron. If a neuron fires and generates a spike, it should wait for a period of time before receiving any kind of excitation. A long refractory period implies that the neuron will wait longer before firing, reducing the firing rate. On the other hand, a short refractory period will let the neuron fire more often.

The other feature of this implementation affects the membrane potential values. Since the results of convolution can be interpreted using either positive or negative events, one part was removed. Thus, in this work, only positive membrane potentials can fire. Fig. 4.7 represents the change of membrane potential for this neuron implementation.

4.4.1.1 Refractory period mechanism

The refractory period uses another counter with the same sliding window mechanism as the leakage counter, called refractory counter (**RC**). Another BRAM bank for refractory timestamps (**RTB**) was also used.

During the convolution process, the refractory timestamp is compared with the refractory counter time (**RCT**). If the refractory timestamp is higher than the RCT, the refractory period is met and the neuron can fire if the threshold is reached. Otherwise, this neuron must wait until the period is met. If during convolution, a neuron fires, then it updates its refractory timestamp with the result of the RCT plus a programmable refractory period value (**RPV**). The resulting refractory timestamp indicates the next time that a neuron will be able to fire. Otherwise, if its refractory timestamp is 0, the neuron can fire the next time, and thus it will be accessed.

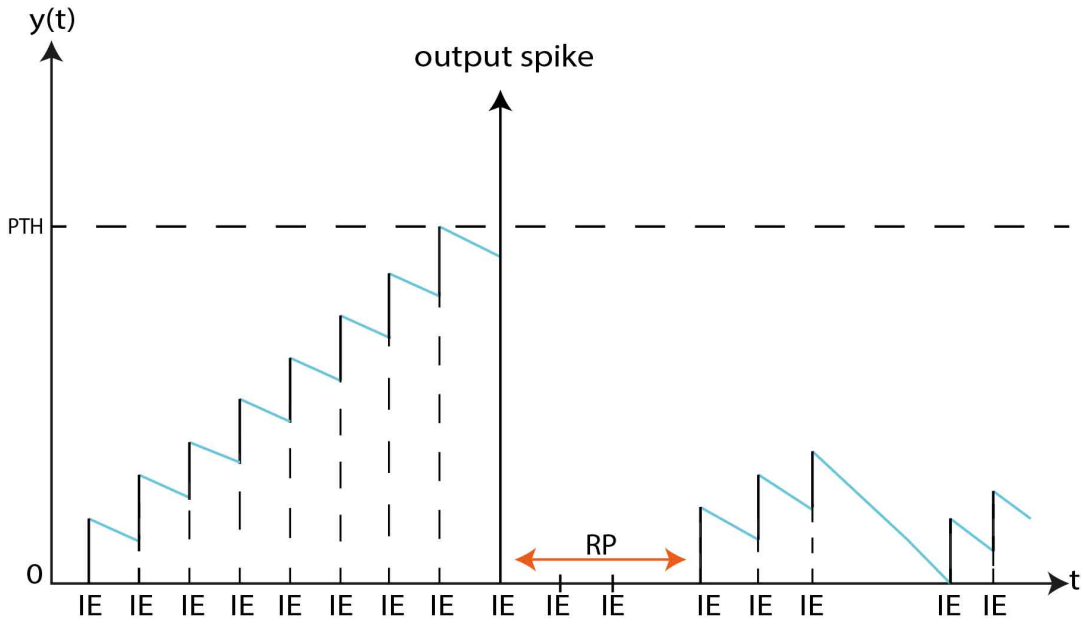


FIGURE 4.7: LIF behaviour with Refractory period.

However, during this update, the sum operation of RCT and RPV may produce an overflow. Therefore, this neuron would be able to fire before meeting the refractory period. In that case, the solution proposed uses another LUTRAM bank as the leakage system. Using a similar concept as the LLM bank, a refractory LUTRAM memory bank (**RLM**) is instantiated. This bank is composed of 128x128 LUTRAM cells of 1 bit. When adding the actual time to the refractory period, an overflow occurs, which means that the neuron must wait for a counter overflow and some time before it can fire. Therefore, when the refractory counter RC produces an overflow, instead of adding 1, as for leakage memories, it subtracts 1 to all cells in the RLM, indicating that an overflow has occurred, and the next time that the refractory period is met the neuron can fire. Fig. 4.8 shows how the refractory memory system works. In this example, we suppose a 3x3 kernel, where each coefficient is higher than the threshold. Therefore, after applying the kernel, 9 events would fire, and the sum operation of RCT and RPV always produces an overflow. When $t=t_0$ (Fig. 4.8.a), the refractory memories are empty, since no event has been fired yet. When an input event (**IE**) arrives and is convolved, the sum of RCT and RPV for each neuron produces an overflow, updating the content of the refractory memories (Fig. 4.8.b). Regardless of the amount of IEs that arrive, the neurons cannot fire, since they have to wait for an overflow from the RC and some time (t_{lim}) to meet refractory period. When an overflow occurs, the cells are updated, subtracting 1 from their content and indicating that an overflow ($t_{R_{ov}}$) has occurred (Fig. 4.8.c). In spite of the memories update, the neurons have to wait some time until the refractory period is met ($t=t_{lim}$). When the refractory period is met, the neurons can fire, as is shown in Fig. 4.8.d).

In a similar way as the leakage memory bank, this bank is inside convolution engine units. Since the 8-bit resolution for all parameters consumes all memory the resources available in the FPGA, the leakage and refractory timestamp resolutions were reduced to 7 bits.

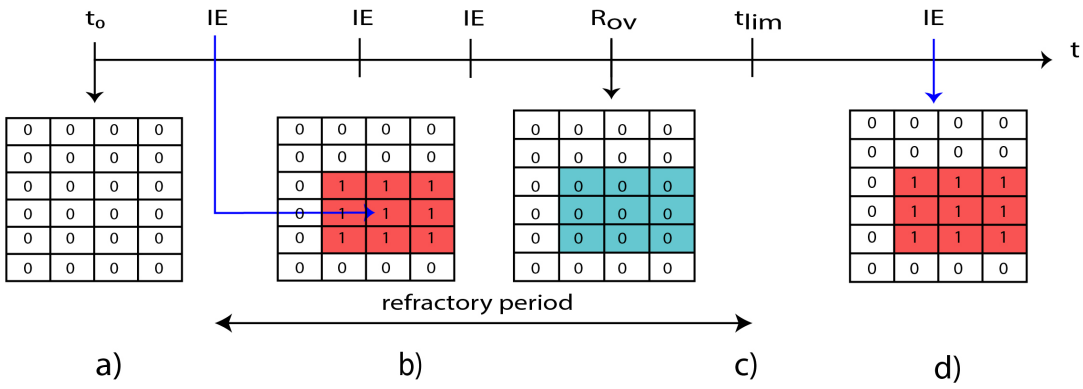


FIGURE 4.8: Refractory LUTRAM mechanism.

4.4.1.2 Convolution Engine V2

The convolution engine module is a state machine that communicates with the memory through a memory arbiter to perform the convolution. However, in the previous version, the convolution engine performs the convolution in one clock cycle, checking all the LLM memory content and performing the convolution. This mechanism had a high resource consumption, which also reduced the latency. Furthermore, the addition of RLM and RTB banks would consume more logic. In order to reduce the resources, the convolution operation was divided into two phases for each processing row: masks step and convolution step.

In previous sections, it was explained how the refractory period and leakage work. In the masks step, timestamps are verified to either apply decay or determine whether the refractory period has been met.

In SIMD processors, conditional branching is used to solve many problems, since the threads have to stop their executions in order to check the condition and execute the correct path. However, recent SIMD architecture threads execute all possible conditional branches and then the correct branch operations are filtered using a mask. Inspired by how SIMD processors work over arrays of data (Hwang, Su, and Ni, 1981, Chen and Kaeli, 2016), the **mask step** generates two binary masks, one for leakage and the other for the refractory period. The leakage mask (**LM**) indicates to each neuron whether decay must be applied during convolution (logical 1) or not (logical 0). On the other hand, the refractory mask (**RM**) indicates whether the refractory period of a neuron has been met or not.

The **convolution step** updates the timestamps, leakage memories and refractory period memories. Before convolution, the corresponding row values of the leakage memory are checked, as was previously mentioned. If the leakage value is 2, the membrane potential is reset; otherwise, the convolution operation is performed for that neuron. During the convolution operation, the LM is multiplied by DV in order to apply leakage to the corresponding neurons. Therefore, the convolution operation consists in adding MP to KV and subtracting DV, if it must be applied. However, the LIF neuron cannot fire if the refractory period is not met; thus, the convolution operation result is multiplied by the refractory period mask. The RM is a binary mask, which implies that, if the refractory period is not met, the result of the convolution operation is 0, since the neuron cannot receive any kind of excitation. Fig. 4.9 shows the computation workflow for a row convolution.

Although this implementation increments in one clock cycle the processing

flow, the next chapters demonstrate that both the area and latency of V2 are lower than those of the V1 version. Fig. 4.10 describes the convolution process. This example highlights that membrane potential values are kept positive and below the threshold. This version manages to implement all the properties of the LIF neuron, which properties allow implementing SCNNs, as the output event rate can be stabilized, thus an SCNN can be inferred.

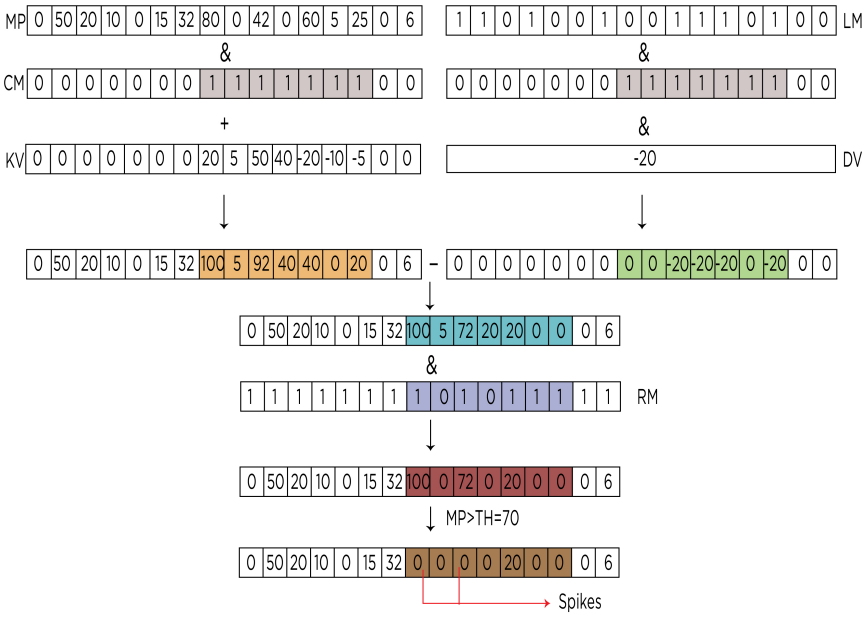


FIGURE 4.9: Kernel memory structure and row generation for convolution operation.

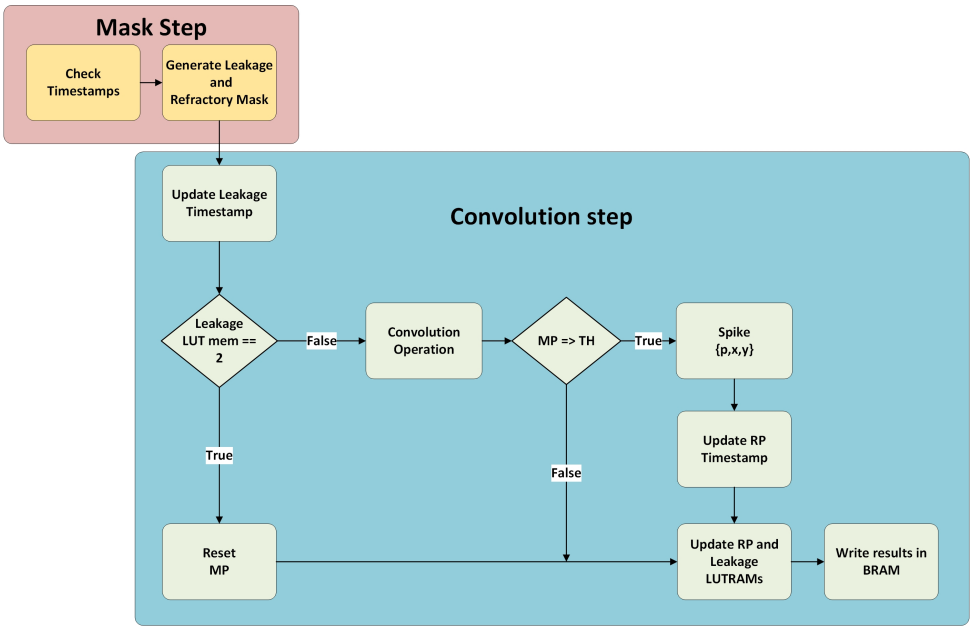


FIGURE 4.10: Diagram of the whole computation, with both mask and convolution steps.

4.5 Leaky integrate-and-fire convolution processor V3

Convolution processor V2 is able to perform 64 convolutions of one configured kernel. Although it implements the LIF properties needed for a SCNN, it would require to configure the processor between each layer or duplicate the hardware to implement several layers in parallel. In addition, if less than 64 convolution engines are used, the unused ones cannot perform convolutions of different kernel sizes, wasting processing resources. As was previously mentioned, each convolution engine sends output events with a CID.

However, the CID is never used for the next computation in previous designs. The new architecture divides the convolution engines into layers, thus each group can perform convolutions of different layers, routing the output events to the next layer depending on its CID. This new solution allows performing multiple convolution layers using one chip. The new architecture changes the way in which convolution engines are configured and how output events are routed. This new version does not change the neuron model behavior or the way in which the computation is performed.

4.5.1 Layer mask selector module

The architecture allows configuring the maximum number of layers to be implemented before synthesizing the design. The micro-controller configures a layer mask, which indicates to each convolution engine which layer they belong to. Through this mask, the convolution engines are able to select the parameters and send output events to the next layer. Having multiple layers implies that the size of memory registers (to store parameters) and multiplexers increases with the number of layers. Fig. 4.11 represents an example of parameter selection for 2 convolution engines of different layers. In this figure there are two register banks that store the leakage and refractory period values for both layers; through layer mask (**MLS**), they select the channel to read the data of the corresponding layer.

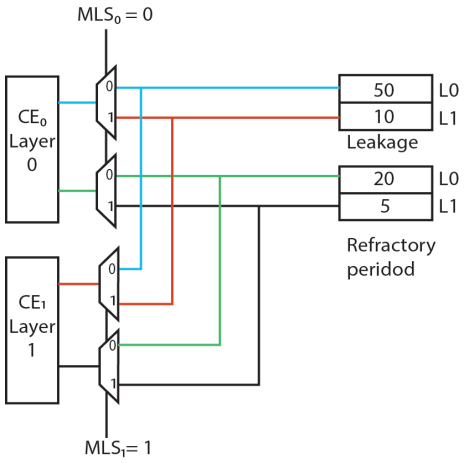


FIGURE 4.11: Parameter selection using the Layer Mask for two convolution engines of different layers.

4.5.2 Cycle output system

Since the convolution can be configured in groups with different kernel sizes and the output events contain information about the convolution engine that generates it, it is possible to perform multiple convolutions in one chip. When an event goes out, its convolution ID is sent, indicating which convolution engine generated it. The convolution ID with the layer mask allows the output multiplexer to route the output event to the next layer or to the output AER bus. Following the previous example of two convolution engines from different layers, Fig. 4.12 represents the whole architecture schematic for two convolution engines of different layers. This new design keeps the same convolution engine, BRAM and LUTRAM distribution of the V2 design. However, this new version adds the parameters bank for each layer and the output router, which sends events to the corresponding layer. Fig. 4.12 represents how events are routed between two convolution layers with pooling enabled. This new characteristic is a novelty in this kind of processor, as other processors need to reconfigure the system between convolution layers to load the weights of the next layer, whereas other designs remove this reconfiguration step by duplicating the hardware with loaded kernels, although this solution increases the power consumption.

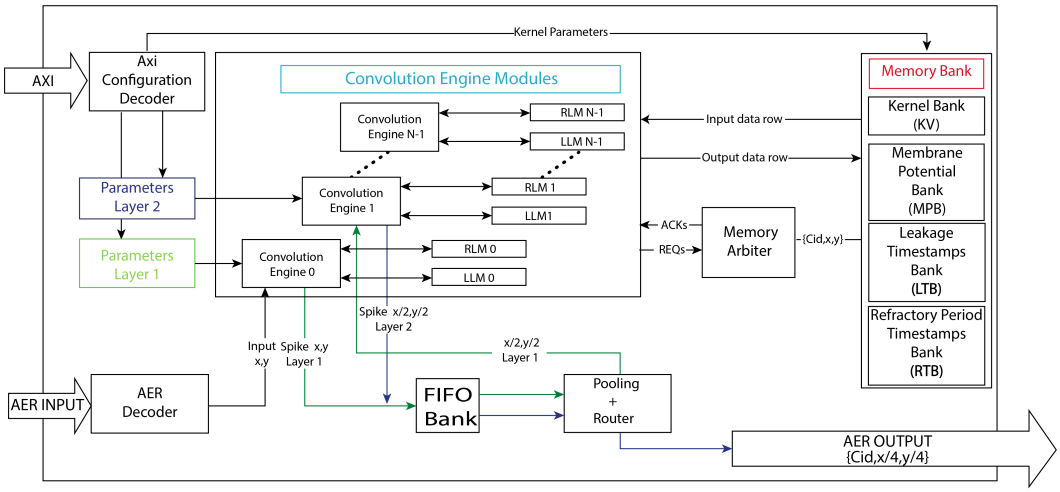


FIGURE 4.12: Convolution processor V3 schematic.

4.6 Frame-based to event-based image conversion

Before introducing the hardware implementation and the different performance tests for each architecture, it is important to mention that a tool was developed while the architecture was being designed, with the aim of converting frame-based images to event-based images. Currently, there exist many software tools to simulate spiking neural networks, such as NENGO (Bekolay, 2014) or BRIAN (Goodman, 2008), and others like JAER (Delbrück, 2007) or NAVIS (Dominguez-Morales, 2017) to visualize neuromorphic sensors output and develop custom filters or processing layers. However, these software tools, for training learning algorithms for image recognition, such as spiking convolutional neural networks (SCNN), require vast datasets to obtain a good accuracy. Although there are several event-based datasets, such as NMNIST (Orchard et al., 2015b) or

Poker-DVS (Stromatias et al., 2017b), generating new datasets is a hard task. In addition, during the development of the architectures presented in this chapter, it was necessary to use synthetic AER images, such as diagonal events or a single centered event to check the neurons activation during the convolution process. In the literature, several methods have been proposed to convert frame-based images to event-driven (Linares-Barranco et al., 2006, Gomez-Rodriguez and Paz, 2005), most of which were implemented in hardware, which hindered their use in the neuromorphic community. In order to make these methods easily accessed, they were implemented in software by joining them in one tool, called frame to event-based (**F2EV**).

Three algorithms were implemented to convert frames to event-based representation. Each algorithm was implemented using 8-bit resolution for pixels intensity. The inter-event-time can be configured. In Fig. 4.13, the results of the conversion methods are shown as histograms of events.

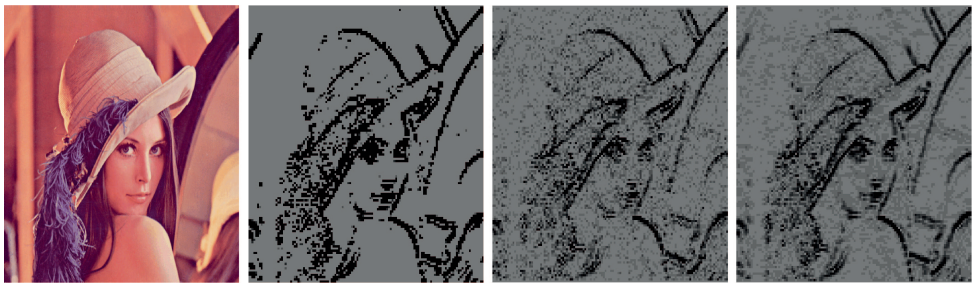


FIGURE 4.13: From left to right: Original image and converted event-based images using Scan, Random and Bitwise algorithms.

4.6.0.1 Scan algorithm

The Scan algorithm goes over the image as many times as the highest gray level using a counter and checking all pixels' intensity for each iteration. When the intensity of the pixel is greater than the current counter value , an event is fired with the (x,y) address of the current pixel.

4.6.0.2 Random algorithm

The Random algorithm consists of a random number coded in 22 bits, where the 14 most significant bits correspond to the position of the pixel address and the other 8 bits to the gray scale. If the gray scale of the pixel in the position given by the random number is lower than the random gray scale generated, an event is produced. The method finishes when the amount of random numbers generated is equal to the number of pixels in the image times the maximum gray level. The event distribution generated by this conversion method is the most bio-inspired among the proposed methods, since the events follow the Poisson distribution (Blei, 2018) presented in the human brain (Heeger and Heeger, 2000, Doya et al., 2007).

4.6.0.3 Bitwise algorithm

The Bitwise algorithm uses a counter of 22 bits that is order-inverted in each iteration. As in the Random algorithm, 14 bits are used for the position of the pixel

and 8 bits for the gray scale. This algorithm compares the gray scale given by the order-inverted value of the counter with the value of the gray scale of the pixel at the position given by the order-inverted counter. An event is produced if the gray scale of the pixel is lower than the gray scale of the order-inverted counter. This method provides a uniform distribution of events.

The F2EV tool helped to verify the convolution processor architecture during the behavioral test phase. In the next section, the architectures implemented in FPGA are tested with real-time event-based recordings.

4.7 Hardware implementation

All the designs presented were described as RTL level in System Verilog language and synthesized with Vivado 2016.4 for a Zynq-7100 MMP platform, which is the same platform used in the NullHop implementation. Although the NullHop Zynq processor used both DMA channels, in this hardware implementation one channel is used to configure the accelerator with the kernel weights and leakage parameters. When the accelerator is configured, the events are received through AER interfaces.

Comparing the three architectures presented, V1 convolves with negative and positive threshold values and implements the leakage property with a working frequency of 90 MHz; however, it cannot implement a SCNN, due to the absence of the refractory period. On the other hand, the V2 implements the refractory period and improves the performance through a mask mechanism that reduces the FPGA resources and increments the frequency to 100 MHz. However, it implements the refractory period mechanism, which has a great impact on the memory resources. The last version presented is an update of V2 that can perform the convolution of multiple kernel sizes, implementing several layers in the same chip. Although this design, called V3, is more flexible for deploying a SCNN than the previous version, it needs more LUTs resources, which reduce its latency to 90 MHz, due to the routing congestion. Table 4.1 presents the FPGA resources consumed by the three designs.

TABLE 4.1: Convolution processor architecture resource consumption.

	V1			V2			V3		
Resource	**Utilization**	**Available**	**Utilization %**	**Utilization**	**Available**	**Utilization %**	**Utilization**	**Available**	**Utilization %**
LUT	264660	277400	0.95	247472	277400	0.89	257503	277400	0.93
LUTRAM	36788	108200	0.34	50851	108200	0.51	50851	108200	0.47
FF	122056	554800	0.22	172607	554800	0.31	179925	554800	0.32
BRAM	514	755	0.68	710	755	0.94	710	755	0.94

4.8 Benchmark scenarios

For the scenario proposed to test the different architectures, two test were developed. The first test checks the correct behavior of the processors convolving event-based images from a DVS recorded dataset, called Slow-Poker (Serrano-Gotarredona and Linares-Barranco, 2015) dataset. The second test checks the behavior of V2 and V3 in response to a fast stimulus using a fast propeller. The purpose of this experiment was to demonstrate that the convolution processors are able to perform convolution in real time with a high speed input from a neuromorphic vision sensor, demonstrating that this could improve the performance for SCNN inference.

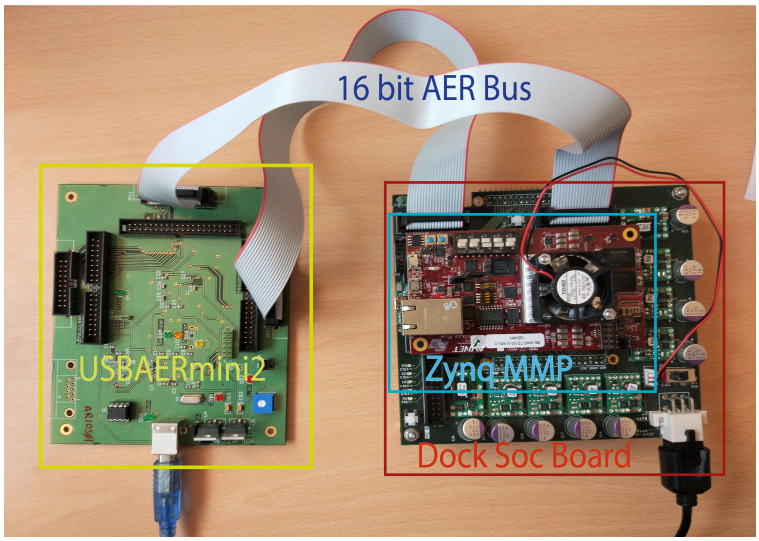


FIGURE 4.14: Experimental hardware setup.

In Both scenarios, a Dock-SoC board was been used. This board was developed by the RTC group and adapts the interfaces of ZYNQ-7100 to AER bus. In this experiment, Dock-SoC is connected to a USB AER mini (Berner et al., 2007) board, which is in charge of sending and receiving events from the FPGA and showing the result of the convolution in JAER software (Delbrück, 2007). Fig. 4.14 shows the setup for both experiments.

4.8.1 Slow-Poker processing test

For the initial test, the FPGA was configured to use the 64 convolution engines with different kernel sizes in order to measure the latency from the minimum kernel size 1x1 to the maximum size 7x7. Although there are several changes from V1 to V3, those changes only affect one clock cycle in the processing pipeline, since both designs work with the same clock frequency (90 MHz), and thus they have almost the same processing time. On the other hand, V2 works with a clock frequency of 100 MHz.

As was mentioned above, there exists one case that stops the system to refresh the leakage memory. Updating a row in the memory takes 2 clock cycles (one to read and another one to write), taking a time of 2.3 μs for V1/V3 and 2.56 μs for V2. Although this situation occurs sporadically, the update could coincide with the convolution operation step. In order to get a more precise processing time of the system, the average time to process an event was calculated after processing 10000 events from each image with different kernel sizes.

Fig. 4.15 shows the resulting processing time obtained from the test. The blue bars (**PT**) represent the processing time obtained with different kernel sizes without any leakage memory delay, which is the best case. The red bars (**PT+UT**) show the worst case, which is the result of processing time plus leakage memory delay. The orange bars (**AVPT**) are the average time obtained after processing 10000 events from different poker images of the Slow-Poker-DVS dataset (Serrano-Gotarredona and Linares-Barranco, 2015). The differences obtained between the best case and the average time are insignificant. This demonstrates that the worst case does not have

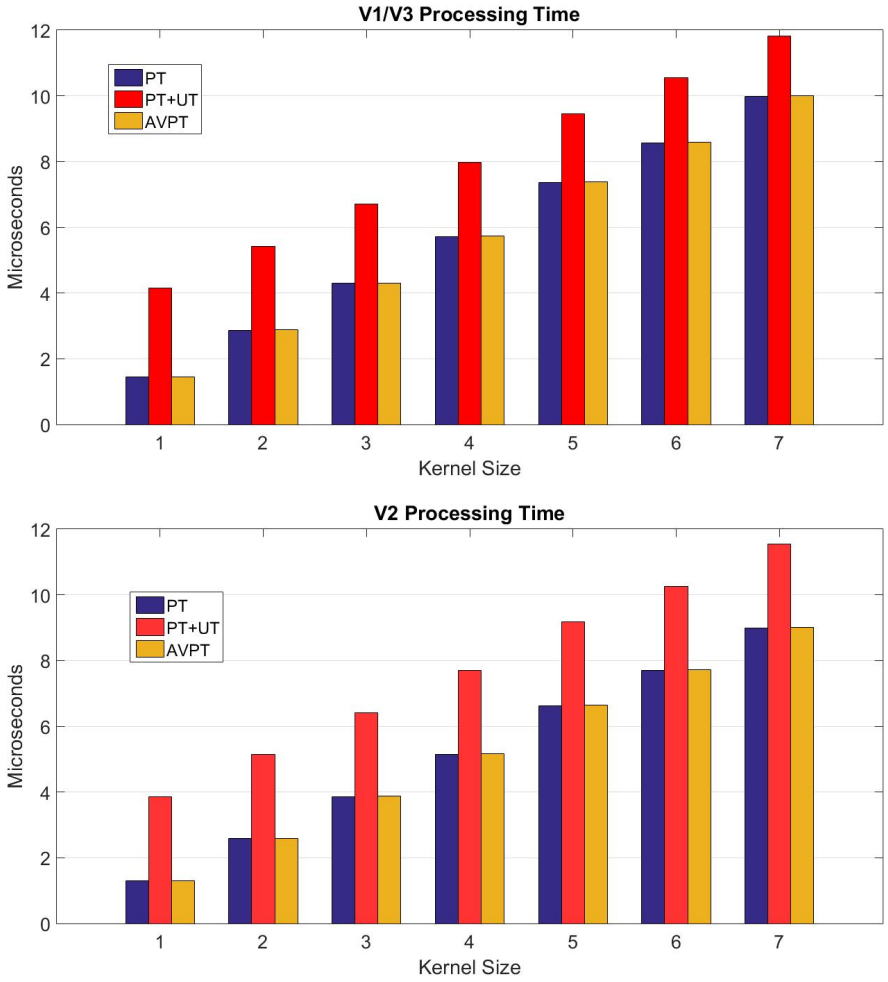


FIGURE 4.15: Processing time of each architecture with 64 convolution engines enabled.

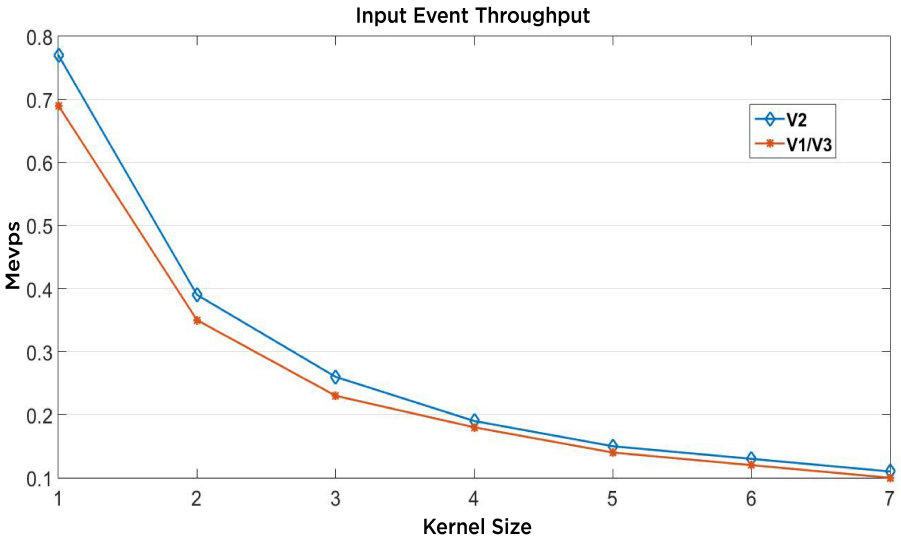


FIGURE 4.16: Input event rate of each convolution processor.

an impact on the processing time and that the behavior of the system has a tendency towards the best case.

The results show that the worst case would be a leakage counter overflow when processing the maximum kernel size (7x7). Comparing the time obtained with the latency of the neuromorphic sensors, such as the DVS retina, the system should be able to perform convolutions in real time. Fig. 4.16 represents the input event rate as a function of kernel size. Fig. 4.17 represents the output of convolution processor V1 after applying two Sobel filters (a)) and the output of both V2 and V3 after applying the Sobel and Gabor filters to multiple poker images (b)).

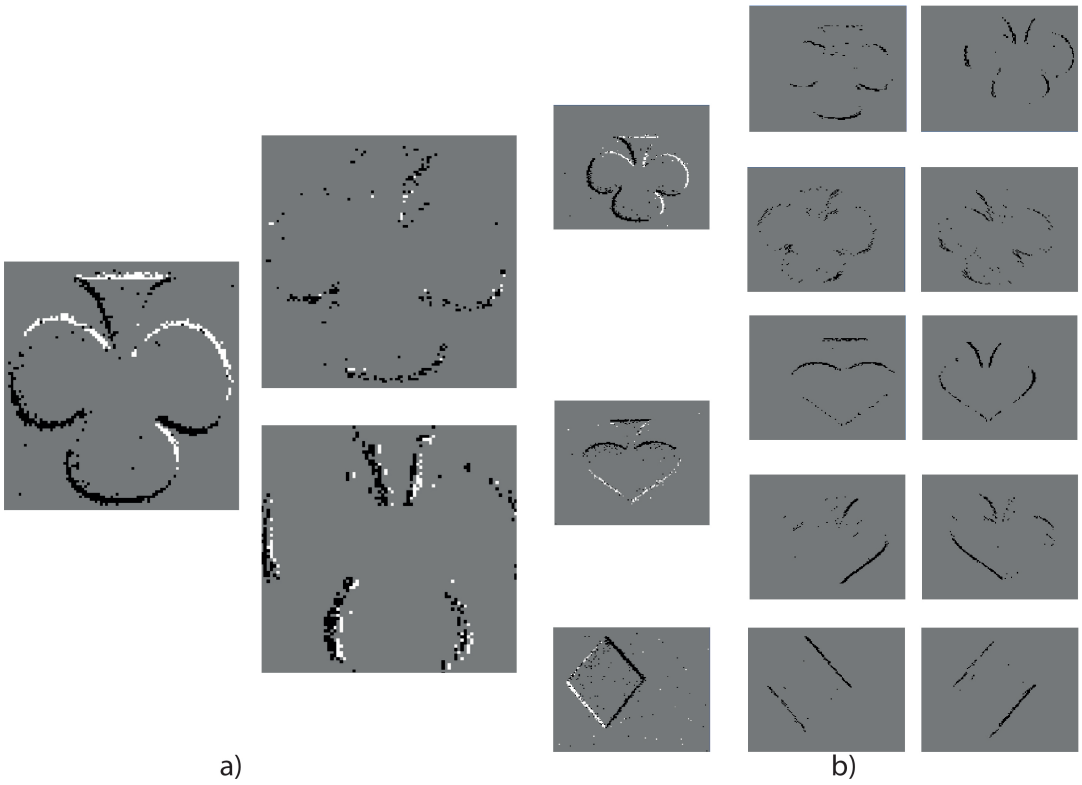


FIGURE 4.17: Slow-Poker convolution integrated during a period of 5-10 ms.

4.8.2 Fast-dot processing test

The second experiment proposed consists in detecting a dot inside a disk in a propeller spinning at 2000 rpm. In this experiment, only processors V2 and V3 were tested. Each processor was programmed with 64 convolution engines with a 7x7 kernel, with the aim of detecting the internal dot. The purpose of this experiment was to characterize the behavior of the system in response to a real-time input stimulus.

Fig. 4.18.a represents the input traffic to be processed, which has two circles: an external circle, which corresponds to the circumference of the disk, and an internal circle, which is the dot to be detected. Fig. 4.18.b shows a histogram of the output events from the convolution processor, where the external circle is filtered and only the events of the internal dot are shown, following its trajectory.

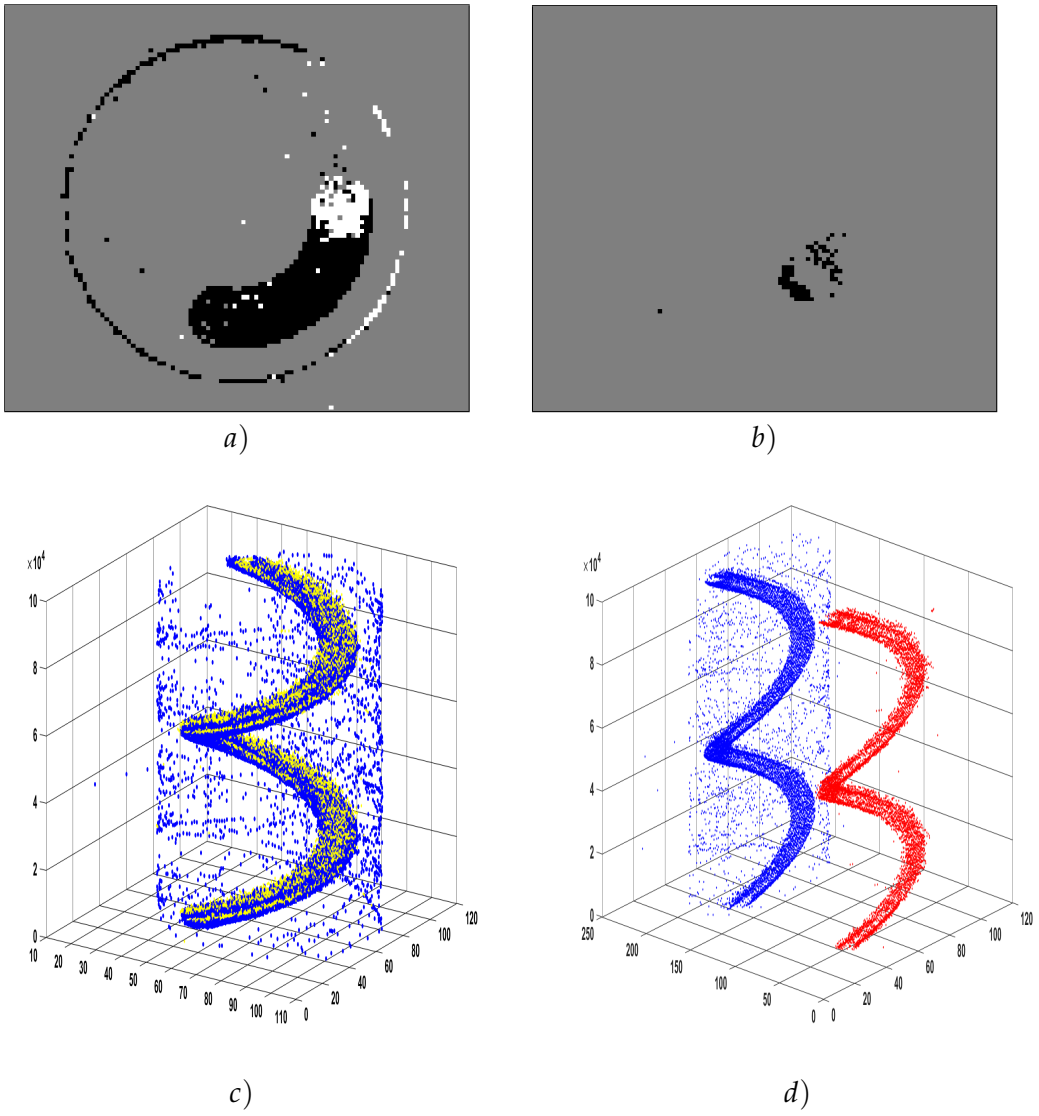


FIGURE 4.18: Fastdot input events comparison with the output from convolution processor.

In order to determine the latency between input and output events, the event traffic behavior was analyzed. Fig. 4.18.c represents a merge between input events (blue) and output events (yellow). As can be observed, the yellow events, which correspond to convolved events, follow the blue events of the internal dot. A similar representation is shown in Fig. 4.18.d, where the blue events represent the input and the red events represent the output. The results show that the convolution operation filters the events that do not correspond to the internal spinning dot.

Both figures shows that there is no significant delay between the input and output events, thus it can be determined that the convolution processor can perform convolutions in real time in response to the traffic of a DVS (Lichtsteiner, Posch, and Delbrück, 2008) or ATIS (Posch, Matolin, and Wohlgenannt, 2011) sensor. In order to have an equivalence with frame-based cameras, a DVS sensor provides timing resolutions above 100 kFrames/s (Serrano-Gotarredona and Linares-Barranco, 2013).

4.8.3 Analysis and comparison

Regarding other implementations of event-based convolution processors in the literature, the work presented by Serrano-Gotarredona et al., 2008 and Camuñas-Mesa et al., 2012 also updates the neurons row-by-row. Although these ASIC solutions are able to perform a single convolution with low latency and power consumption, these designs have two important limitations to implement a SCNN. The first limitation is the reduced number of convolutions they can perform; the second limitation is that their neuron model does not implement the refractory period, which is needed to deploy SCNNs. Another solution, such as the one presented by Camuñas-Mesa et al., 2012, can perform up to 24 multi-kernel convolutions in parallel. However, it still has the same problem as the previous ASIC solutions: the refractory period is not implemented. Apart from the ASIC implementation, other FPGA implementations, such as the one proposed by Camuñas-Mesa et al., 2018, presents a convolutional node implemented in a Spartan-6 FPGA based on the design presented in Linares-Barranco et al., 2010. This implementation is able to perform 22 convolutions in parallel with the refractory period. However, the memory is accessed pixel-by-pixel, producing many bottlenecks and, thus, reducing the system latency.

TABLE 4.2: Comparison between event-based convolution processors.

	LIFCP-V2	**LIFCP-V3**	Camuñas-Mesa et al., 2018	Camuñas-Mesa et al., 2011	Camuñas-Mesa et al., 2012
Convolution Modules	64	64	22	1	24
Max Kernel Size	7x7	7x7	10x10	32x32	32x32
Muti-Kernel	No	Yes	Yes	No	Yes
Platform	Zynq 7000 FPGA	Zynq 7000 FPGA	Spartan 6 FPGA	0.35um CMOS	0.35um CMOS
Latency in-out μs	1.3-9.01	1.44-9.98	0.5-32	0.05-0.14	0.06-0.68
Input Event Throughput (Meps)	0.11-0.77	0.10-0.69	0.05-3	1.77-20	1.47-16.6
Frequency (MHz)	100	90	50	120	100
Refractory Period/ Leakage	yes/yes	yes/yes	yes/yes	No/yes	No/yes
Weights Resolution (bits)	8	8	8	8	4
Leakage/Refractory resolution (bits)	7/7	7/7	8/8	-/8	-/4
Adders	7	7	22	1	24
Mop/s	348.06	314.06	68.75	7314.29	36000
Power Consumption per convolution module(mW)	0.92	100	0.35	200	8.3

Table 4.2 represents a comparison between the designs presented in this work with other designs mentioned previously. The V2 and V3 implementations perform a number of 348.06 and 314.06 MOp/s, respectively, with a power consumption per convolution engine of 0.92 and 1.12 mW.

With respect to the number of operations, the designs presented in this work are below ASICs, 36 GOp/s (Camuñas-Mesa et al., 2012) and 7.3 GOp/s (Camuñas-Mesa et al., 2011). However, they surpass the FPGA implementation presented in Camuñas-Mesa et al., 2018, which performs 68.75 Mop/s. On the other hand, due to the fact that each convolution processor implements a different number of convolution units, the power consumption was measured as the power consumed per each convolution unit. In this comparison, the power consumption is lower in FPGA implementations, which was 0.92 mW for V2, 100 mW for V3, and 0.35 mW for Camuñas-Mesa et al., 2018.

4.9 Summary and discussion

In this chapter, three different spiking convolution processor architectures are presented. The main difference with other implementations seen in the literature is the row-by-row access through a memory arbiter, which allows reducing the memory consumption and implementing several convolution modules. The row-by-row memory access increments the input throughput; therefore, a faster input event traffic can be processed in real time, preventing a possible congestion, which can slow down the system. In spite of the fact that the latency is higher than in other FPGA processors, such as the one presented by Camuñas-Mesa et al., 2018, the number of logical resources is higher and limits the implementation of several convolution modules in small FPGAs, such as MiniZed or ZedBoard.

The aim of the architectures presented is to allow the inference of SCNNs in the future. Although there are several methods to train SCNNs, such as the one presented by Liu and Furber, 2016 and Kheradpisheh et al., 2018, and it is also necessary to study how to scale the trained weights to fit in 8-bit resolution without losing much accuracy. Apart from accuracy, the complexity of inferring a SCNN lies in the configuration of the parameters, such as the refractory period and leakage. One possible innovation to be added in the designs of this work would be to incorporate a mechanism to configure those parameters automatically as a function of the input and output event rates. The next chapter describes an alternative technique for event-based vision feature extraction that can be implemented in small VLSI systems with low power consumption.

Chapter 5

Pattern Recognition Based on Time Surfaces in Real Time

> "Imagination is more important than knowledge. Knowledge is limited. Imagination encircles the world."
>
> Albert Einstein

Previous chapters described spiking convolution as one approach to pattern recognition for the event-based processing paradigm. However, this kind of approach emulates the behavior of LIF neuron models, which require a large number of resources to be implemented and does not directly make use of rich timing information provided by the events. Although most event-based pattern recognition systems are based on spiking neural networks (Lee, Delbruck, and Pfeiffer, 2016, Shrestha and Orchard, 2018, Stromatias et al., 2017a), other methods integrate dynamical information of neuromorphic vision sensors to recognize patterns (Jhuang et al., 2007, Sironi et al., 2018) or combine frame-based and event-based information (Liu et al., 2016), fusing both data. Recent works, such as the one presented by Lagorce et al., 2017, propose a new concept, called time surfaces, which takes into consideration the past activity of incoming event neighborhoods. This approach works directly over the timing and spatial information of events, recognizing features from input events. This algorithm is called hierarchy of time surface (**HOTS**), since time surfaces can be organized in a hierarchical way to extract patterns. Currently there are several neuromorphic platforms, such as Spinnaker (Furber et al., 2013), BrainScale (Schmitt et al., 2017), IBM TrueNorth (Akopyan et al., 2015) or Intel Loihi (Lin et al., 2018), to deploy this kind of technique. These hardware platforms are able to implement scalable networks by connecting multiple boards with a maximum of 460 million neurons and 460 billion synapses, with a power consumption between 100 mW and 50 kW (Furber, 2016a). However, these solutions are not ideal for embedded systems, due to their size, latency and power consumption.

This chapter presents a novel architecture that implements the HOTS algorithm for FPGAs. This implementation poses a new hardware approach in event-based pattern recognition systems. Furthermore, a new event-based memory model for neuromorphic systems is presented; this memory works in a similar way as cache memories, storing the most recent events, and reducing memory resources consumption and latency.

5.1 Hierarchy of the time surface algorithm

5.1.1 Time surface

The HOTS algorithm is based on time surfaces, which extract a pattern from the events of a scene. A time surface describes the spatio-temporal activity of a neighborhood around the last received event e_k. As is explained in previous chapters, an event consists of a tuple of four values: an (x,y) address, which corresponds to the spatial location; a timestamp (ts), which is the time in ticks of microseconds when the event was received; and a polarity (p), which is ON if luminosity increases or OFF if luminosity decreases. An event can be described as follows:

$$e_k = [x, y, p, ts] \tag{5.1}$$

The first step to create a time surface is to compute the time difference between the input event e_k and its neighborhood. This time difference is known as time context. A time context $\mathcal{T}_k$(u,p) of the event e_k represents the square region of interest (**ROI**) centered in e_k, of spatial coordinate x_i=[x_k,y_k], and it is the difference between e_k timestamp and the timestamps of the most recent neighbors. The square ROI has a dimension of (2R + 1) x (2R + 1). Mathematically, the time context can be expressed as:

$$\mathcal{T}_k(\mathbf{u}, p) = \{t_k - t | t = max\{t_j | x_j = (\mathbf{x_i} + \mathbf{u}), p_j = p\}\} \tag{5.2}$$

Where u = [u_x,u_y] is such that $u_x \in$ [-R,R] and $u_y \in$ [-R,R].

Time surface S_k(**u**, p) associated with the event e_k, is obtained by applying an exponential or lineal decay kernel of time-constant τ to the time context $\mathcal{T}_k$. There are several ways to apply a decay kernel to the time-context, the most common ones are exponential decay (Equation 5.3) and linear decay (Equation 5.4).

$$S_k(\mathbf{u}, p) = e^{(-(t_i - \mathcal{T}_i(u,p))/\tau)} \tag{5.3}$$

$$S_k(\mathbf{u}, p) = \begin{cases} 1 - \frac{\mathcal{T}_i(u,p))}{\tau}, & \text{if } \mathcal{T}_i(u,p) < \tau \\ 0, & \text{otherwise} \end{cases} \tag{5.4}$$

The resulting time surface represents the spatio-temporal activity of the incoming events. However, since a time surface is computed for each new incoming event, there is a possibility that the computed time surface does not have enough neighbor timestamps; therefore, the resulting time surface would not contain useful information and it would waste resources. In order to limit this effect, time surfaces are discarded if they do not contain enough information, since this information will be part of a later time surface as soon as a new event is emitted in the spatio-temporal neighborhood. A time surface will be processed if it satisfies the next condition:

$$\sum_{\mathbf{u},p} S_k(\mathbf{u}, p) > 2R \tag{5.5}$$

The condition shown in Equation 5.5 ensures that there will be enough neighbors before processing the time surface, with a minimum number of neighbors equal to the size of the square ROI.

Information from time surfaces is not useful unless it is compared with well-known patterns. The next section explains how a network is built in order to extract patterns from an input scene. In this work, linear decay was implemented, since the operations to be performed are simpler, as it only consists of a division and a subtraction.

5.1.2 Hierarchical time surface network

Time surfaces can be trained and organized in a hierarchical way to extract features. Trained time surfaces are called prototypes and each one represents a pattern. Prototypes are organized in banks, composing a HOTS layer. The layer properties are listed below:

- N: the number of learned prototypes in a layer
- R: the radius size of the time surface
- τ: the time constant kernel decay applied to the events

When a time surface is created from an input event, it is compared with the bank of prototypes of a layer, in order to obtain the matching prototype. A distance algorithm, such as the Euclidean distance or the cosine distance can be used. The prototype with the shortest distance is the matching one.

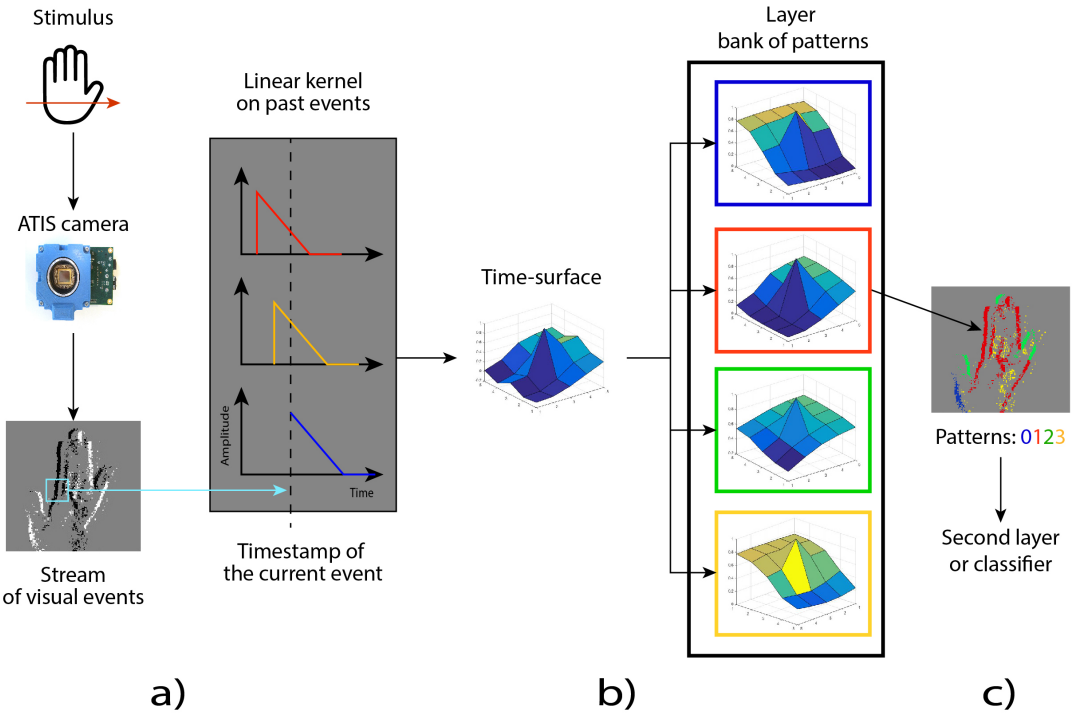


FIGURE 5.1: HOTS layer processing workflow.

Then, an event is generated with the same (x,y) address and timestamp. However, the event polarity p_k is modified so that $p_k = c$, where c is the ID of the matching prototype. Thus, the event encodes a pattern instead of ON/OFF

polarity. The output event can be used to feed a second layer that processes the event in a similar way, although the second layer combines the features of the previous layer. Otherwise, the output events can be integrated over time, generating a histogram of activated features that can be used by a classifier.

Fig. 5.1 shows an example of a processing pipeline for a one-layer HOTS implementation used in this work. The sensor sends a stream of events with (x,y) addresses and an ON/OFF polarity from a captured stimulus (Fig. 5.1.a). The time surface is generated as a result of applying a decay kernel to the time context of the ROI (Fig. 5.1.b). A comparison between the generated time surface with the bank of prototypes is performed to determine the closest pattern. The closest prototype will send out an event with the same (x,y) but with its corresponding ID. Finally, events are sent out to another layer, or integrated over time to generate a histogram to be processed by a classifier (Fig. 5.1.c), as was previously mentioned.

5.2 FPGA HOTS accelerator

This chapter describes a novel FPGA architecture to infer the HOTS algorithm in real time. The design has a similar configuration system as described in previous chapters for spiking convolution accelerators; it has an AXI connector of 32 bits embedded to an ARM processor, which is in charge of configuring the different modules. On the other hand, there are also two AER buses of 16 bits that receive events from a sensor and send out the result of the classification.

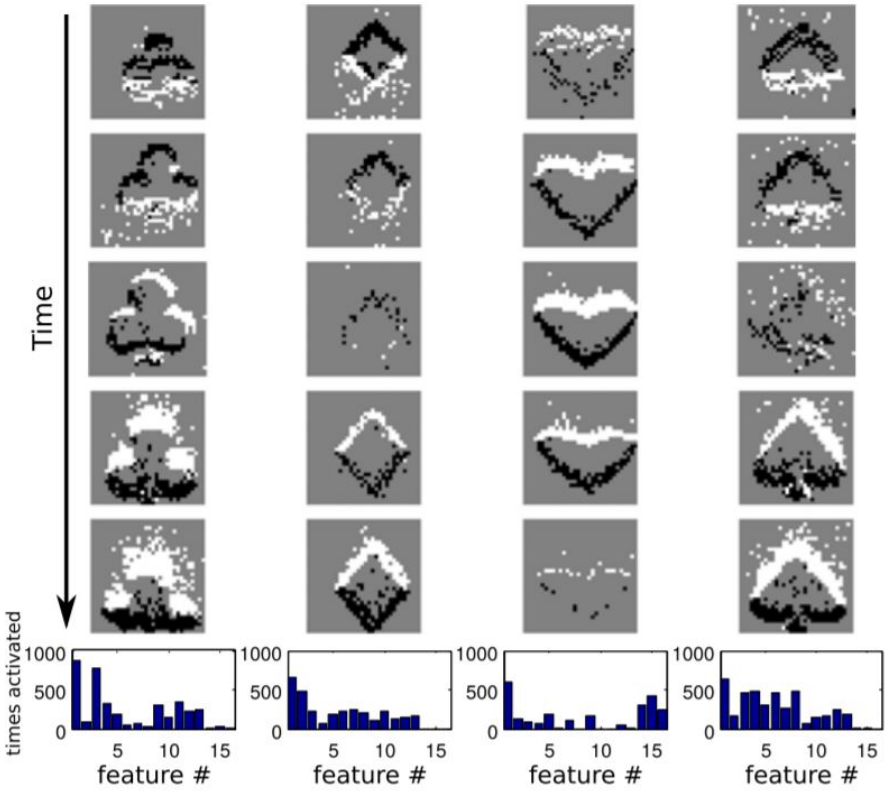


FIGURE 5.2: Example of HOTS histograms. Image taken from (Lagorce et al., 2017).

In previous sections, it was explained that two types of decay kernels can be applied to the events: linear or exponential decay. Although both solutions are

valid to implement the HOTS algorithm, exponential decay requires more hardware resources, which implies more power consumption, due to the fact that the computation operations performed are complex. Therefore, in this architecture, a linear decay was implemented to generate the time surfaces from input events.

When an event is received from a sensor, the time surface generator module creates the time surface from an incoming event and its neighbors. The resulting time surface is sent pixel-by-pixel to the Euclidean estimator module, which computes the Euclidean distance with the prototypes, accumulating the partial differences. When all partial differences for all pixels in a radius R are compared, the square root module computes the square root, obtaining the Euclidean distance for a given prototype. Then, all distances are compared, in order to determine the minimum one that corresponds to the matching pattern. Finally, an event is generated with the same (x,y) address and timestamp as the input event. However, the polarity is replaced by the ID of the matching prototype. Output events are accumulated in a histogram, which, over an integration period, is compared with histograms obtained during the network training, classifying the input. Fig. 5.2 shows an example of histograms obtained during the event processing of four poker symbols.

The second purpose of the time surface generator is to create the time surface to be compared with prototypes. Incoming event timestamps are stored in a BRAM memory, in order to characterize the timing behavior. The (x,y) addresses of incoming events are used with radius R to determine the spatial neighborhood. Each pixel timestamp is read, obtaining the time context as a result of the difference between input event timestamp and the timestamp of the corresponding neighbor, then the linear decay is applied using a configurable τ value, following Equation 5.4.

5.2.1 Time surface generator module

The time surface generator module is in charge of two processes: the first one is to assign a timestamp to each incoming event. Most neuromorphic vision sensors, such as ATIS or DAVIS, send events and timestamps through USB, this fact makes it easier to read information from these sensors, facilitating the connection between different devices.

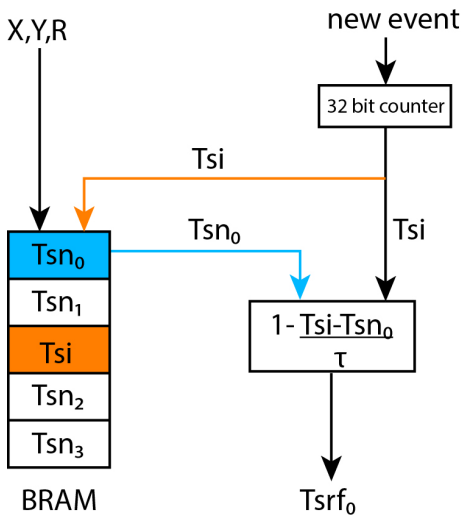


FIGURE 5.3: Time surface generator module.

However, it also hinders the integration in embedded systems, since USB needs drivers and software frameworks, such as CAER[1], that increment the latency to read events. Apart from this problem, since these sensors use 32-bit timestamps, a large connector would be needed to receive the information. Assigning timestamp values in hardware allows connecting the sensor directly to the FPGA, reading events directly from the sensor, thus reducing the latency and the size of the connector. Fig. 5.3 shows a diagram of the time surface generator module.

5.2.2 Euclidean distance estimator

The Euclidean distance is defined as the length of the segment that joins two points in the plane. With p = (p1, p2,..., pn) and q = (q1, q2,..., qn), two points in Euclidean n-space, distance (d) from p to q, or q to p, the Euclidean distance is mathematically described as follows:

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(p_1-q_1)^2+(p_2-q_2)^2+(p_n-q_n)^2} = \sqrt{\sum_{i=1}^{n}(p_i-q_i)^2} \quad (5.6)$$

This distance equation has been applied in several classification algorithms, such as K-means or support vector machines, to determine the closest pattern. However, although the Euclidean distance equation is easy to implement and execute by a computer, the square root is hard to implement in an embedded solution. In spite of this problem, most square root hardware implementations are developed using high level tools, such as Vivado High level synthesis or Intel High level synthesis tools. Although these tools are able to implement the square root function, the resulting implementation usually consumes a large amount of FPGA resources. In the architecture designed, two implementations based on Babylonian square computation and non-restoring square root computation were studied. The Euclidean distance estimator (**EDE**) consists of two modules to compute the Euclidean distance between time surface and prototypes, the difference distance computation and the square root modules. Fig . 5.4 shows the whole architecture of the Euclidean distance estimator module.

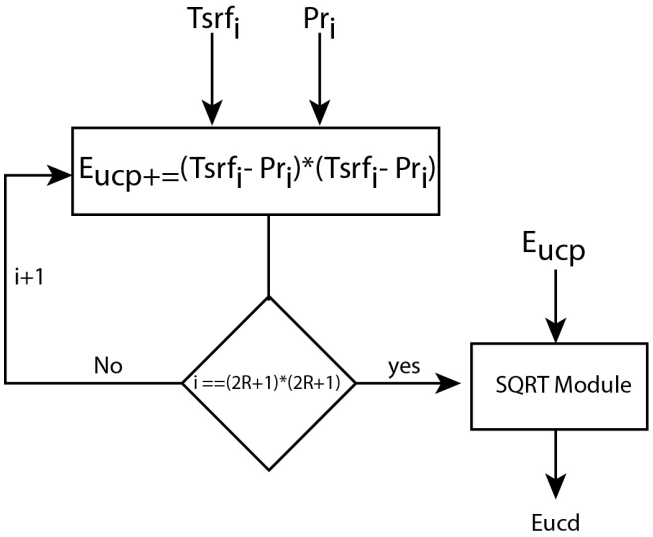


FIGURE 5.4: Euclidean distance estimator module.

[1]CAER is an adapted version of JAER software written in C language for embedded systems.

5.2.2.1 Difference distance computation module

The main problem in digital design is how the computation is performed. Although floating point arithmetic usually gives less errors than fixed-point arithmetic, it is more complex to implement in HDL language and it usually needs more hardware resources. The architecture proposed works with fixed-point arithmetic, which can be configured from 16 to 64 bits before synthesizing.

The difference distance computation module computes the square difference between the incoming time surface of a pixel and the corresponding time surface prototype pixel stored in the LUTRAM memory of the FPGA, following the Euclidean equation (Equation 5.6). Each partial square difference is accumulated for all pixels; when all pixels in the square neighborhood of size (2R+1)x(2R+1) have been processed, the accumulated value is sent to the square root module to obtain the Euclidean distance.

5.2.2.2 Babylonian square root module

The square root modules compute the square root of the total sum of the time surface difference with the prototypes. During the development of the architecture, two different mechanisms were studied to implement the square root computation: the Babylonian method and non-restoring square root implementation.

The Babylonian method (Fowler and Robson, 1998) is an algorithm that is easily implemented in VLSI circuits. Given a number N, whose square root will be found, the algorithm will iterate a number of times following the next equation:

$$L_i = \frac{L_{i-1} + N/L_{i-1}}{2} \tag{5.7}$$

Where L is an initial value, which usually is half of N, and N is the number whose square root will be computed. The Babylonian method iterates a number of times until the algorithm converges. A larger number of iterations implies that better values of the square will be obtained. Although this algorithm does not implement complex operations, its accuracy depends directly on the number of iterations performed. Therefore, this algorithm was discarded, as it computes an approximation of the resulting square root, and it can be far from the original solution, giving imprecise accuracy results. Apart from the problem mentioned previously, the implementation of the division operation of N and L_{i-1} in fixed-point is complex and it needs a large number of hardware resources, unless the division is computed using a constant value as the divider.

5.2.2.3 Non-restoring square root module

The other evaluated mechanism to compute the square root of a number is the non-restoring square root (Li and Chu, 1996), which computes the square root of a number represented in 2's complement.

The non-restoring algorithm is an iterative square root algorithm that generates one bit of a result in each iteration using the two most significant bits of the input data. The partial dividend used in each iteration is determined with the remainder of the previous iteration and appending the two most significant bit to the least significant position of the remainder. The stage quotient is computed by shifting one bit the quotient of the previous iteration, and it is appended to the new bit value, the subtrahend is determined as the previous iteration quotient shifted by two and appended to the new predicted quotient bit value.

If the value of the new dividend is greater than the subtrahend, then the predicted value is accepted and the difference is forwarded as the remainder for the next iteration; otherwise, the new quotient bit is assumed to be zero and the constructed dividend is directly forwarded as the remainder of the current iteration. An iteration of the algorithm generates one bit of the square root, and thus it is needed to perform multiple iterations storing the partial results.

Algorithm 1 Non-restoring square root algorithm

```
1:  D ← 0;
2:  Q ← 0;
3:  R ← 0;
4:  for i ∈ {0, ..., 15} do
5:     if R >=0 then
6:        R←(R << 2) or (D >> (i+1)&3);
7:        R←R-((Q<<2) or 1);
8:     else
9:        R←(R<<2) or (D >> (i+1)&3);
10:       R←R-((Q<<2) or 3);
11:    end if
12:    if R >=0 then
13:       Q←((Q<<1) or 1);
14:    else
15:       Q←((Q<<1) or 0);
16:    end if
17: end for
```

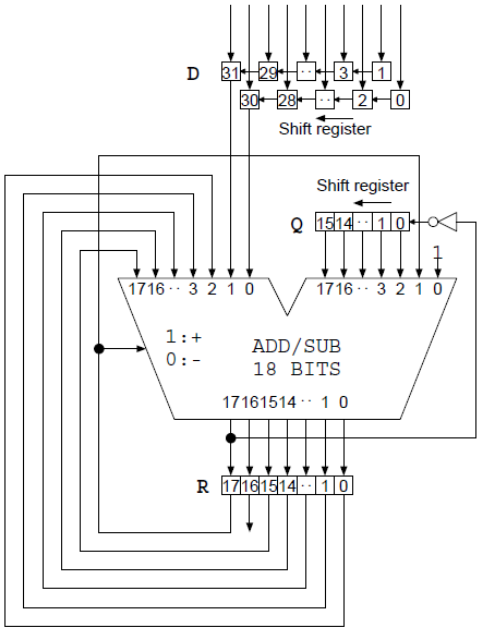


FIGURE 5.5: Non-restoring schematic.

Although this algorithm has multiple implementations, there are efficient implementations in the literature, such as the one presented by Sutikno et al., 2014 and Piromsopa, Arporntewan, and Chongstitvatana, 2001, who proposed an efficient way to implement the algorithm in VLSI systems. As can be observed in Algorithm 1, the operations performed by the algorithm only consist in bit shifting, bit masking, bit comparison, addition and subtraction operations. In other words, these operations can be easily performed using an arithmetic logic unit (ALU). Fig. 5.5 shows the hardware schematic of the square root system; the operations performed are simple, its result does not depend on the number of iterations, and it does not need a huge amount of hardware resources. This algorithm is the one implemented for the square root module.

When the Euclidean distance is computed, the minimum Euclidean distance value of each Euclidean distance estimator module is selected, obtaining the matching pattern. The number of Euclidean distance estimator modules depends on the number of prototypes of each layer and they work in parallel, processing the input time surfaces. Each module in this design works in a pipeline way, whereas the time surface generator module computes the next time surface of a pixel, and the Euclidean distance modules compute the distance of the previous pixel time surface

5.2.3 Histograms integration and compare modules

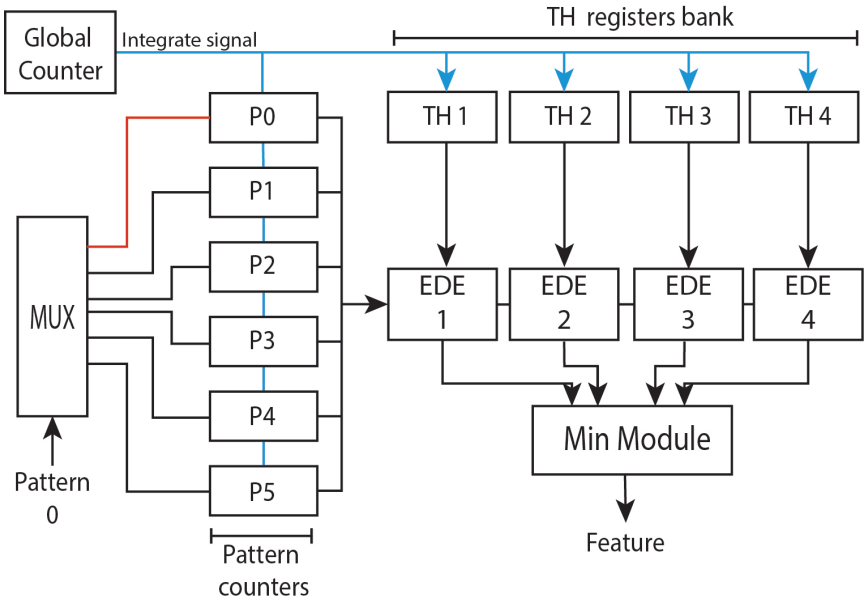


FIGURE 5.6: Histograms generator and comparator module (HGCM).

The output events encode a pattern within their (x,y) address, instead of an ON/OFF polarity, from here onward they will be called pattern events, being integrated for a period of time that matches the τ value of the last layer. In the design presented, the output events histogram is generated through several counters that increment their value with the arrival of a pattern event. These counters are called pattern counters. Pattern counters match the number of possible patterns that an event can encode in the HOTS network. The resulting histogram stored in the pattern counters is compared with trained histograms. These histograms are stored in a bank of registers. After the integration period is met, a

global counter asserts the integrate signal, computing the nearest neighbor algorithm (**K-NN**) between histograms using the Euclidean distance through EDE modules, which were previously described, and resetting the counters for a new histogram integration while the system is still receiving. However, the previous EDE takes the data pixel-by-pixel, as it needs the time-surface value. The EDE implemented in this module processes the square differences for all the columns of the histograms in one clock cycle, reducing the latency, then the square root is computed. After the Euclidean distance is computed, the classification result corresponds to the closest histograms and it is sent out through AER bus. Fig. 5.6 shows an example of the histograms generator and comparator module (**HGCM**) for 5 different patterns and 4 features to be classified.

5.2.4 Hardware implementation

The name of the architecture implemented is F-HOTS and it has been described as RTL with System Verilog language and synthesized for a Zynq-7100(xc7z100-2) MMP and Zedboard (xc7020clg482) platforms using Vivado 2016.4. The implementation works with a maximum clock frequency of 100 MHz. The whole platform architecture, which is shown in Fig. 5.7, including the PS and the PL requires a power consumption of 1.6 W. For our implementation, the ARM processors need 1.533 W and the remaining 77 mW are consumed by the FPGA logic. These power consumptions were measured with Xilinx power tool after the implementation, assuming a toogle rate of 50% of the signals, what is higher than normal operation.

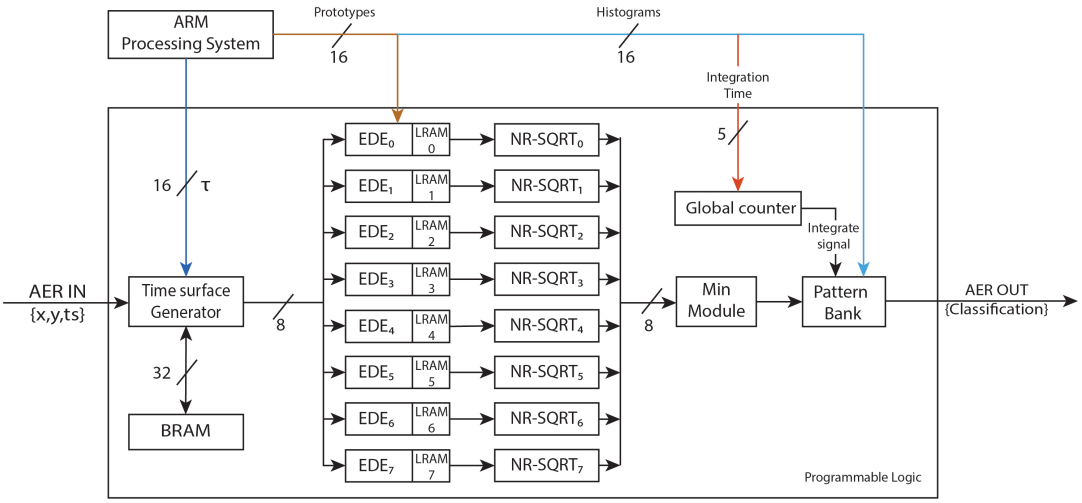


FIGURE 5.7: F-HOTS global architecture.

The ARM has the same function as in convolution spiking processors: it is in charge of configuring the layer with the different prototypes and the parameters (Radius, tau), whereas AER interfaces communicate with neuromorphic sensors, sending or receiving events. In this design, the dock SoC board has also been used (Aimar et al., 2018, Rios-Navarro et al., 2018).

Although the design was tested in a large platform, due to the AER interface adaptation, the system presented can fit in a smaller FPGA, such as the one available at the Zedboard. Table. 5.1 shows the percentage of resources used in Zynq-7100 and Zedboard FPGAs. The next section presents some tests to characterize the system in terms of accuracy and performance.

TABLE 5.1: PS + PL resource utilization for 16-bit resolution.

	Zedboard (xc7020clg482)	Zynq7000 (xc7z100ffg2)
LUT	8313 / 53200 (15.6%)	8351 / 277400 (3%)
LUTRAM	2879 / 17400 (16.5%)	2872 / 108200 (2.6%)
FF	5627 / 106400 (5.2%)	6092 / 554800 (1.1%)
DSP	46 / 220 (20%)	46 / 2020 (2%)
BRAM	18 / 140 (12.8%)	18 / 755 (2%)

5.2.5 Experimental results

5.2.5.1 Pattern recognition test

The F-HOTS architecture was tested with the processing of events from the NavGestures-sit database, in order to compare the accuracy error produced between the software implementation and the F-HOTS implementation. The NavGestures-sit dataset has 6 hand gestures: Right, Left, Up, Down, "Hello-hand" and Select, as shown in Fig. 5.8. This dataset was first used to test this network, whose parameters are: τ = 10ms, R = 2 and N = 8.

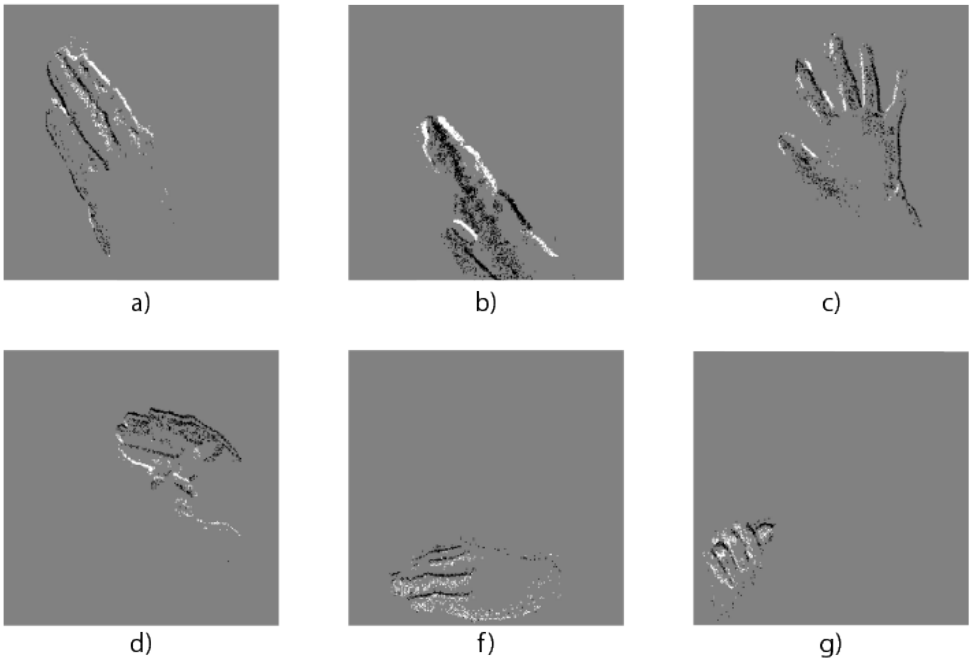


FIGURE 5.8: Hands gestures from a to g: Left, Right, Hello Hand, Up, Down, Select.

This network was previously tested on this dataset in a mobile phone implementation (Maro and Benosman, 2018), obtaining an accuracy of 94.5%. The experimental setup was the one used during the development of the spiking convolution processors. It consists of an AERtool, which receives events from the computer through USB packets and sends them using an AER interface to the Zynq, where the events are processed. Output events are collected by the USBAERmini2 board Berner et al., 2007 through JAER software (Delbrück, 2007). The purpose of this experiment was to characterize the behavior of the system with the NavGestures-sit dataset mentioned previously, measuring the accuracy against

the software implementation. The addresses of the events from NavGestures-sit were scaled to 128x128 resolution to fit in the AER connector. As was already mentioned, the computation error depends directly on two factors: the bit resolution and the operations performed.

In fixed-point operations, it is quite common to lose some precision due to several factors, such as bit truncation or resolution. In this work, the accuracy loss obtained by the architecture after processing the dataset using different fixed-point resolution was measured. The computation resolutions used in this experiment were 16, 32 and 64 bits in $\mathbf{Q_{n.m}}$ notation, where n bits were for the integer part and m for the decimal part. In this work, n corresponds to the bits of the upper half of the resolution, whereas m is the lower-half bits; e.g., for 16 bits, n is the 8 most significant bits, and m is the 8 least significant bits. The average accuracy loss obtained for NavGestures-sit for each resolution was 1.2%, 0.78% and 0.4%, respectively, with respect to the classification obtained in software implementation presented in Maro and Benosman, 2018. Table 5.2 presents the accuracy results obtained for each bit resolution.

TABLE 5.2: Accuracy comparison with different numerical precision.

	Maro and Benosman, 2018	**Q8.8**	**Q16.16**	**Q32.32**
NavGestures-sit	94.5%	93.3%	93.72%	94.1%

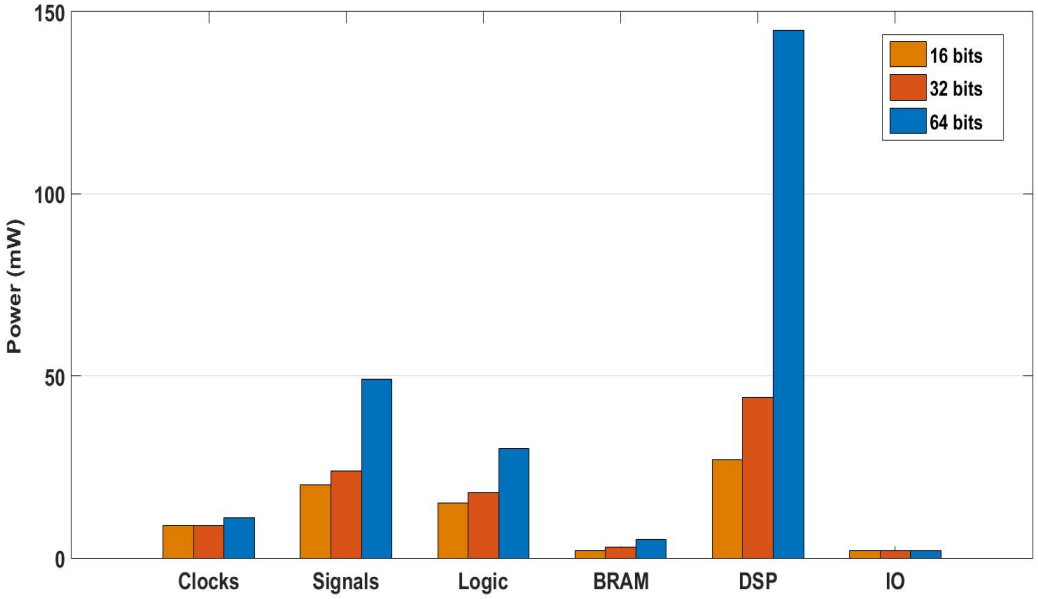
The computation error produced is due to fixed-point arithmetic. Multiplication operations have a loss of accuracy. In the design presented, two multiplications are performed, one in the time surface generator module and another one in the Euclidean distance estimator module. However, time surface multiplication is not the main problem. The Euclidean estimator module performs several multiplications depending on the R value. Therefore, the error is accumulated. Furthermore, after each multiplication step, words are truncated to match with the bit resolution, and bit truncation carries a loss in the resolution, as some decimals are lost. Table 5.2 shows that increasing the bit resolution does not have a significant effect on the accuracy.

TABLE 5.3: Programmable logic resources as a function of numerical precision.

Zynq7000 (xc7z100ffg2)			
Resolution	Q8.8	Q16.16	Q32.32
LUT	3%	4.22%	4.68%
LUTRAM	0.31%	0.36%	1.67%
FF	0.34%	0.38%	0.48%
DSP	2%	3.23%	6.92%
BRAM	2.12%	2.12%	2.12%
Zedboard (xc7020clg482)			
Resolution	Q8.8	Q16.16	Q32.32
LUT	15.6%	16.7%	22.01%
LUTRAM	7.43%	8.41%	10.37%
FF	1.62%	1.88%	2.41%
DSP	20.2%	34.09%	64.45%
BRAM	11.43%	11.43%	11.43%

Additionally, even if bit resolution increases, the accuracy loss does not decrease significantly and it would imply more consumption of hardware resources. Table 5.3 shows the hardware consumption for each bit resolution. Regarding the table results, LUT and LUTRAM resources increase as a function of bit resolution, due to the fact that buses and memories increase their size. On the other hand, there is a critical resource that is limited in small devices: the DSPs. As was previously mentioned, the accuracy obtained from resolution over 16 bits is insignificant. However, bit resolutions higher above 16 bits have a great impact on DSP consumption for small devices, making it difficult to implement a multi-layer version in a small FPGA, as can be seen in Zedboard implementation, where more than 50% of the DSPs are used for 64-bit resolution. Therefore, the ideal implementation, according to these results, is a 16-bit resolution implementation, since the DSP consumption is ideal for future multi-layer implementations, obtaining good accuracy results that do not differ significantly with the other bit resolutions.

FIGURE 5.9: Power consumption of FPGA components for each bit resolution.



Regarding the power consumption, as was already mentioned, the increment of bit resolution implies an increment of DSP and signals. This affects the power consumption directly, as can be seen in Fig. 5.9, which shows the power consumption for each different component of the FPGA. The total power consumption for each different resolution is: 77 mW for 16 bits, 99 mW for 32 bits, and 199 mW for 64 bits.

5.2.5.2 Performance test

The purpose of the second experiment was to characterize the response of the accelerator to fast stimuli. In the previous chapter, a spinning dot was used in convolution spiking processors; here, the same experiment was performed with the processing of events with different radius. In this experiment, the processing time was measured with different R values, from 1 to 8. The latency of the design

depends directly on the square radius ((2R+1)*(2R+1)) to be processed, since large R values imply that more memory accesses are performed. On the other hand, small R values imply less memory access and, consequently, a small latency. In neuromorphic systems, the supported throughput (Ev/s) from sensors is also important, since it indicates the throughput that the system is able to compute, as was done with spiking convolution processors. Therefore, the spinning dot image is ideal, since it presents a high input throughput to test the limit of the F-HOTS architecture.

In this experiment, the processing pipeline until the generation of the histograms was measured, since the complex computation is focused on event pattern extraction and classification. This occurs sporadically when the integration period is met, taking around 0.5 μs, does not affect to the pattern extraction. Fig. 5.10 shows how the R value affects both the processing time and the input throughput. Furthermore, regarding the plot shown in Fig. 5.10, the smallest radius to be processed is for R = 1. This scenario represents the best case, due few pixels are processed. The latency obtained for this scenario is 1 μs and an input throughput of 1 Mev/s. On the other hand, the worst scenario is for the maximum radius of 8, as several memory accesses are performed, increasing the latency to 6.6 μs with an input throughput of 0.15 Mev/s.

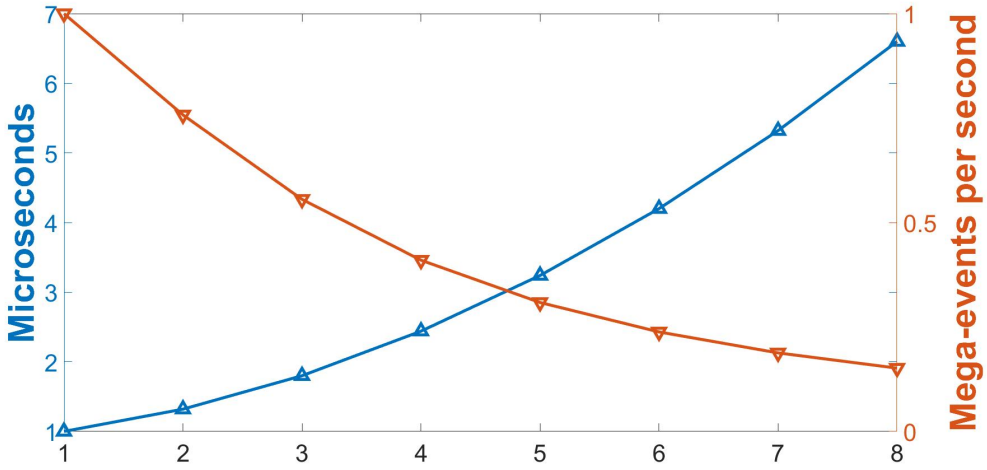


FIGURE 5.10: Left axis: Processing time per event with different radius. Right axis: Mega events per second of evolution for each different radius.

Another important factor to consider in hardware accelerators is the number of operations per second. The time surface generator module (**TSG**) computes a division and a subtraction for each pixel in the square neighbors of radius R, whereas each Euclidean estimator module (**ESM**) computes two subtractions, one multiplication and one addition for each pixel in the square neighbor and number of prototypes (**N**) in parallel. Finally, the nonlinear square module (**NLS**) computes one addition/subtraction and a shift operation. The total number of operations performed by F-HOTS architecture is expressed in Equation 5.8, where **R** is the radius, **N** is the number of prototypes and **T** is the time to process an event:

$$MOp/s = \frac{(\overbrace{(2R+1*2R+1)}^{\text{Square neighborhood}} * (\overbrace{2}^{\text{TSG}} + \overbrace{4}^{\text{ESM}} * N)) + \overbrace{2}^{\text{NLS}} * N}{T} \tag{5.8}$$

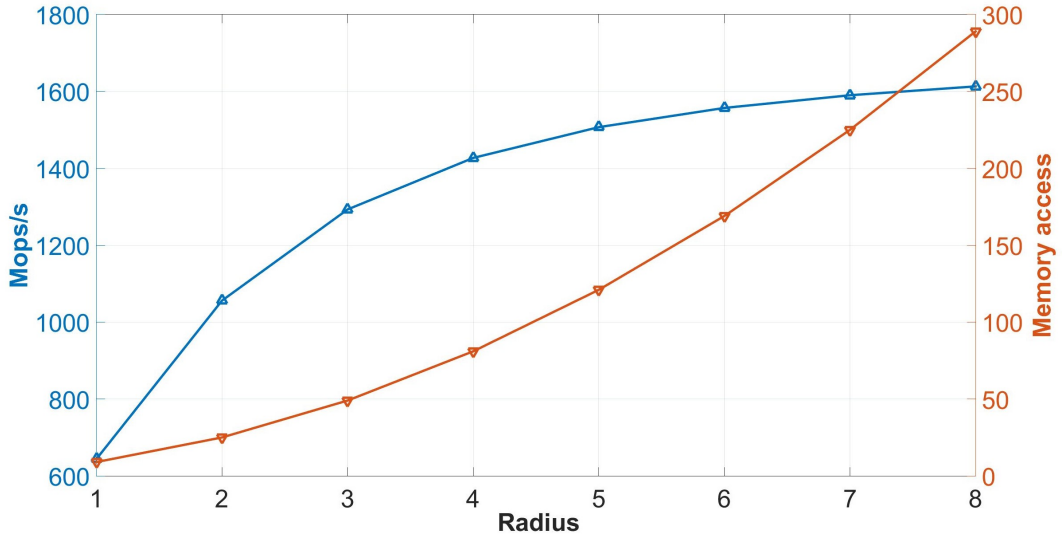


FIGURE 5.11: Left axis: Mops/s performed with different radius, with frequency of 100 MHz. Right axis: memory accesses performed.

Fig. 5.11 shows the number of operations per second (Ops/s) with the number of memory accesses for different kernel radius. As shown in Fig. 5.11, the Mops/s increases significantly for each different R value up to 6. At that point, the performance does not depend on memory bandwidth (memory access), but on computation units. In other words, the performance reaches its limit at approximately 1600 Mops/s, due to the number of computation elements.

5.2.6 Comparison and discussion

The F-HOTS implementation poses a novelty in the state-of-the-art of pattern recognizing event-based processing, due to the novel HOTS algorithm and the low power consumption. there are other pattern recognition systems in the literature, such as the one presented in Amir et al., 2017 where an event-based gesture recognition is implemented in IBM TrueNorth using spiking convolutional neural networks. The approach presented works over the DVS gesture dataset, obtaining accuracy results of 94.59% for 10 categories, and 96.49% for 11 categories. However, the implementation requires a vast amount of neurons, which implies a power consumption of 178.8 mW. Although the accuracy obtained is higher and the system classifies more categories than the system presented in this chapter, its power consumption is higher than that of F-HOTS.

Other works, such as the one presented by Camuñas-Mesa et al., 2018, also implement a spiking convolutional neural networks in an FPGA platform, obtaining an accuracy of 96%. However, the solution presented is not able to maintain that accuracy, which decays to 63% when processing events in real time.

The F-HOTS architecture memory usage can be upgraded to perform a multilayer version. Although the BRAM memory is not a limitation in the design presented, a large part of the memory is unused, due to the sparse output of neuromorphic vision sensors. In most neuromorphic vision systems, the memory is as large as the spatial resolution of the sensor, in order to store all possible incoming pixels. However, some algorithms, such as HOTS, only consider recent events. Therefore, a memory model that only stores recent events and can access them in parallel would reduce the memory size and latency. The next section presents a memory model for event-based systems with the previously described properties.

5.3 Event-based time configurable memory

In neural networks, one of the critical aspects is memory access, as it reduces the latency to deploy them in real time. Within deep learning, CNN architectures use new compression methods and techniques, to reduce both memory size and accesses. On the other hand, in neuromorphic engineering, there are several memory topologies focused on improving the performance of spiking systems in order to increase the number of neurons and synapses (Indiveri and Liu, 2015). On the other hand, other works focus in the new promising Memristor (Li et al., 2015, Zhang et al., 2018) or resistive switching memory (**RRAM**) (Ielmini, 2018,Wu et al., 2017) technology, which can emulate neuron behavior, such as spike-timing dependent plasticity (**STDP**) (Shouval, Wang, and Wittenberg, 2010, Dan and Poo, 2004) and spike-rate dependent plasticity (**SRDP**) (Rachmuth et al., 2011, Dong et al., 2016), which is better than the current complementary metal-oxide-semiconductor (**CMOS**) technology (Smith, McDaid, and Hall, 2014). However, a large number of event-based processing systems mentioned previously need vast memories to store the state for all the pixels of a vision sensor. However, due to the sparse output given by this kind of sensor, part of the memory is wasted, as is shown in Fig. 5.12.

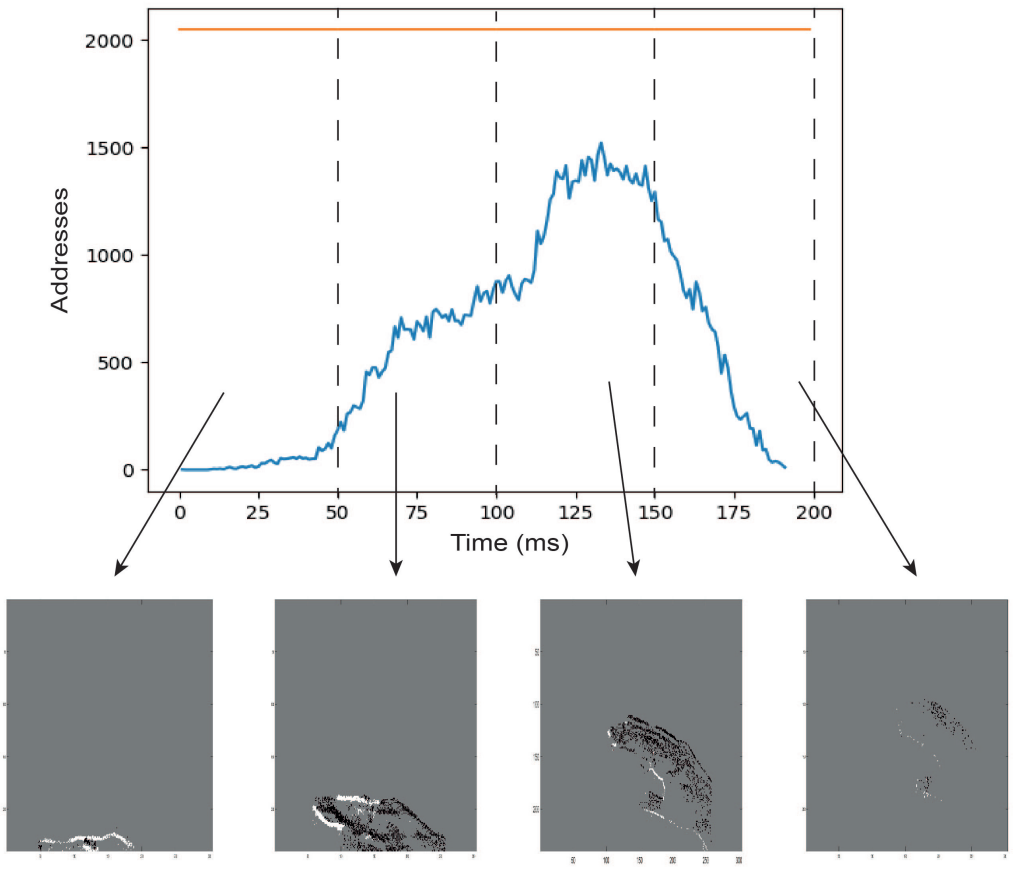


FIGURE 5.12: Number of events in windows of 5ms (blue) and maximum stored events in a memory of 2048 addresses (orange).

The previous HOTS implementation was based on RAM memory, where each pixel had an address, consuming a lot of memory space. From here onward, RAM memory will refer to this kind of memory topology.

Recent works, such as presented by Khodamoradi and Kastner, 2018, propose a memory model to reduce utilization, updating the rows and columns of the memory with the new incoming event, and thus reducing the memory resources in an FPGA. In the previous section of this chapter, a one-layer HOTS was implemented in FPGA, since a one-layer HOTS does not require many hardware resources and there were enough BRAM memories. However, in multi-layer implementations of HOTS algorithm, apart from DSPs, memory resources are critical, due to the availability and latency. Regarding the availability, each intermediate prototype has to store its output events, in order to generate the time surfaces for the next layer; therefore in multi-layer implementation with several prototypes, as shown in Fig. 5.13, the lack of memory resources is a great problem. Additionally, in HOTS algorithm, there are many memory bottlenecks that increase the latency, as several memory accesses are performed to read the data of the input event neighborhood.

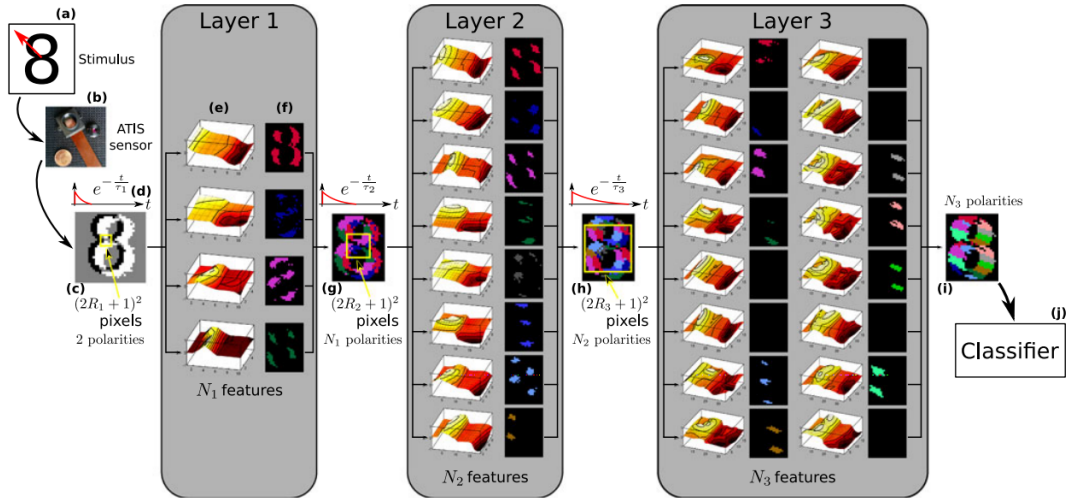


FIGURE 5.13: Multi-layer HOTS implementation composed of 3 layers. Image taken from (Lagorce et al., 2017)

In this section, a new memory model is proposed, which stores the most recent events, reducing the amount of memory resources, and sends the neighbors directly in one clock cycle, reducing the latency.

5.3.1 Memory Model

In computer architecture, cache memories store the recently accessed data following the principles of temporal and spatial localities, in order to avoid accessing the global memory. The spatial and temporal localities are defined below:

- **Temporal locality**: if a memory location is accessed, it will be probably referenced in the near future since there is a temporal relation with the current location.

- **Spatial locality**: if an accessed memory position is referenced, it is likely that its neighbor positions will be accessed.

Applying the same concept of cache memories to event-based processing, a memory model, which stores recent events, can be developed. An analogy between events and data can be done, since an event generated from a pixel whose luminosity has changed recently will have to be processed in a short period of time by an algorithm, and the recent stored events are probably neighbors of the next incoming event. The memory model presented in this chapter is based on cache memories, which compare the timestamps of the stored events with the timestamp of the incoming event, in order to update the memory content, storing the most recent events. The event-based cache memory stores the (x,y) address with its corresponding timestamp. When a new event arrives, each event's timestamp is compared with the incoming one. If the difference between the timestamp of the incoming event and the one stored in the memory is greater than a configurable value, named timestamp difference value (TDV), older events are erased and the memory is shifted, replacing the erased events and updating a memory pointer that points to the next available memory address (Fig. 5.14.a). The TDV allows controlling the number of events to be stored and the time difference between them, greater TDVs imply that several events will be stored, whereas lower values will store only the most recent events. Then, a square neighborhood focused on the incoming event is sent out to be processed by another module (Fig. 5.14.b).

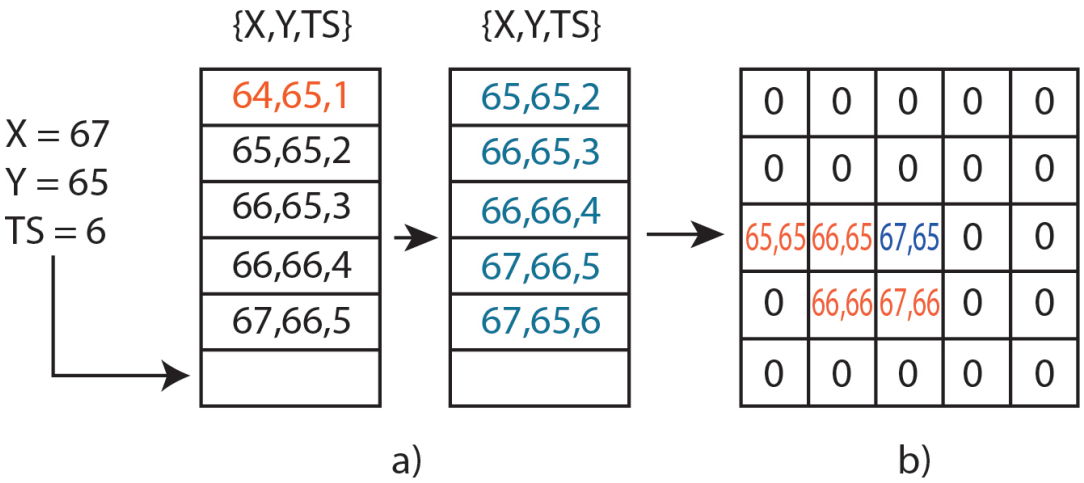


FIGURE 5.14: Event-based timing memory workflow.

5.3.2 VLSI implementation

This chapter presents two different implementations of this memory for VLSI systems. The first implementation reads all the data of the memory cycle–by-cycle, updating the memory content. On the other hand, the second implementation accesses all the data, updating the memory content in two clock cycles. The working pipeline of both designs is explained in detail in the following subsections.

5.3.2.1 Sequential memory model

The sequential implementation reads events from the memory one by one and compares them with the new incoming event. If the read event is older than the incoming one, it is erased from the memory. When all the memory content has been read, the memory is shifted to replace the blank spaces of the erased events. This memory model can be implemented using either register or BRAM memory

technology. Although memory resource consumption is low, this implementation poses a great problem, since the memory reads all its content, even if it is almost empty.

5.3.2.2 Parallel read memory model

The parallel implementation reads all the events from the memory in one clock cycle and compares them with the incoming event. The behavior of the memory is the same as in the previous case: events older than the incoming event are erased and the memory content is shifted to leave all blank spaces at the end of the memory. For the proposed implementation, parallel read memory takes two clock cycles, one cycle to read and group the square neighborhood, and a second cycle to update the memory. In order to access all the data in parallel, each memory position is a register that can be read in parallel. This version has the opposite problem with respect to the sequential version, since it has a fixed latency regardless of memory usage. However, it requires several logical resources to implement a small memory, since a large amount of data has to be accessed in parallel.

An algorithm such as HOTS can make use of this kind of memory, as this algorithm considers the timing activity of neighbors. In order to test the viability of the proposed memory, some theoretical tests were developed, as described in the next subsections.

5.3.3 Experiments and Results

With the aim of testing the memory model, two different experiments were performed. The first experiment consists of replacing memories of the FPGA implementation previously presented with the parallel read memory to determine its behavior and measure its performance. Then, the memory model was tested under a multi-layer hots implementation with different τ values.

5.3.3.1 One-layer experiment

The gesture-recognizing HOTS network used in previous sections, implements RAM-based memory to store the incoming events and generate the time surfaces. In this experiment, RAM-based memory is replaced with the parallel read memory model, in order to compare the error produced with different TDVs.

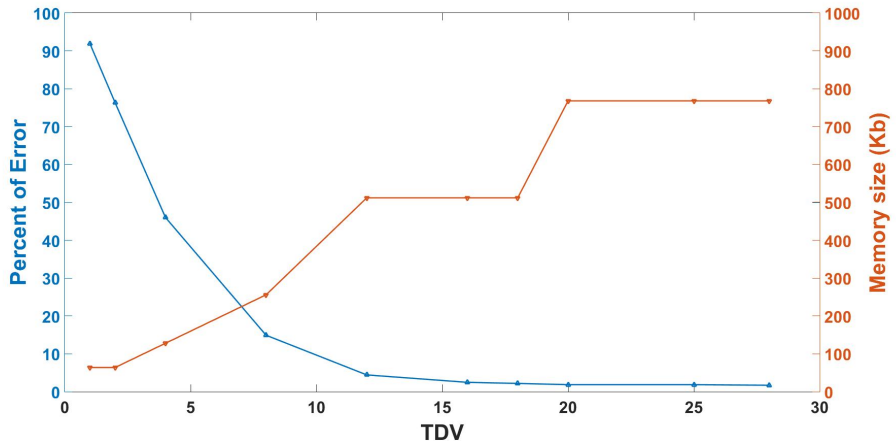


FIGURE 5.15: Parallel read memory size vs classification error.

As is shown in Fig. 5.15 the accuracy error is around 0 for a memory of 756 Kb with a TDV greater than 18 ms. High TDVs imply a greater memory size to store events without erasing many events. RAM-based HOTS requires a memory of 3562 Kb. On the other hand, parallel read memory model reduces the memory consumption for this implementation to 79%.

Apart from memory consumption, another important factor is the memory accesses. Regardless of the radius, the parallel read memory requires two clock cycles, which reduces considerably the latency. Fig. 5.16 shows the latency comparison between an FPGA implementation using RAM-based memory with the implementation that uses parallel read memory using different kernel radius.

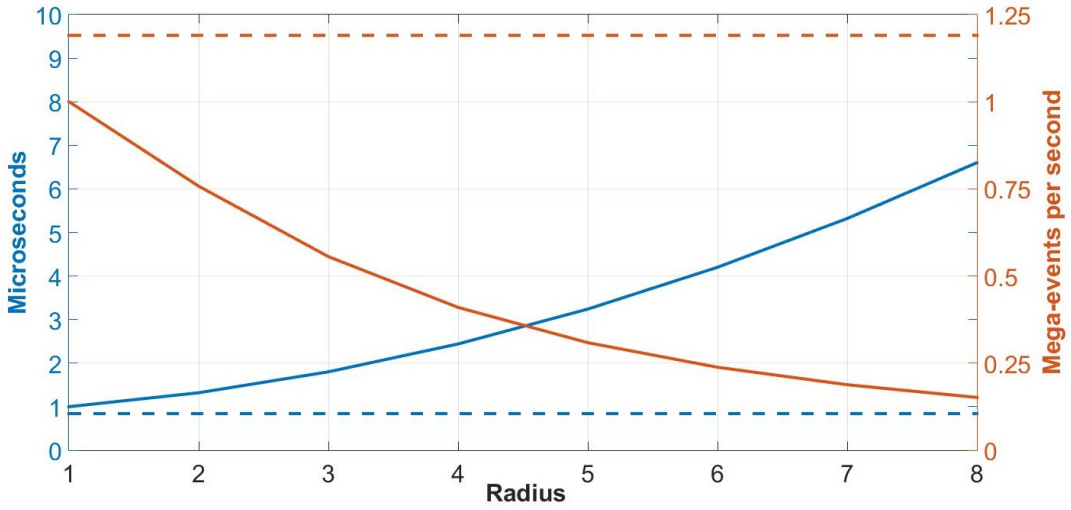


FIGURE 5.16: FPGA latency and throughput with parallel read memory (discontinued line) and RAM memory (continuous line).

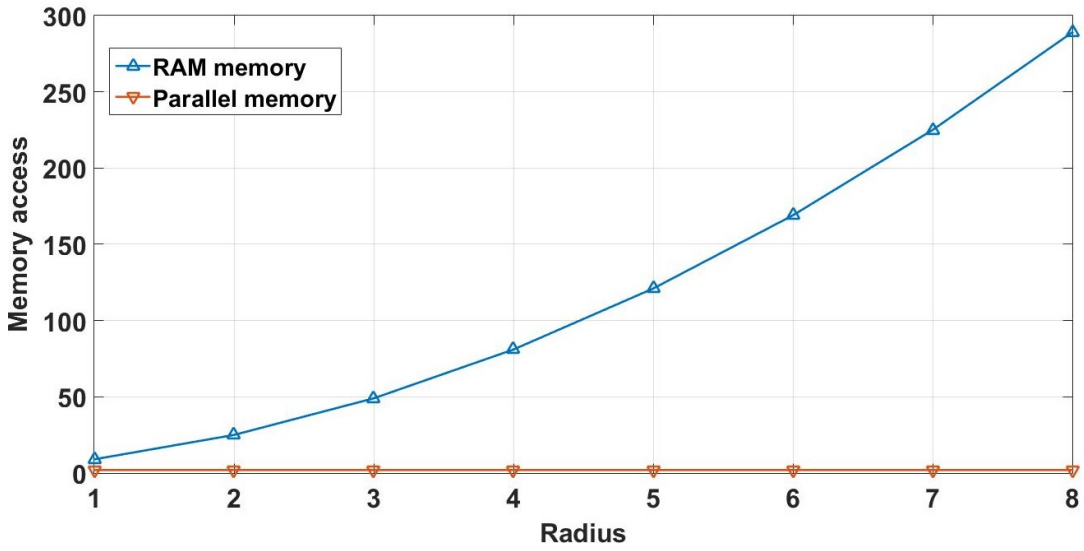


FIGURE 5.17: FPGA memory access of parallel read memory (orange line) and RAM memory (blue line) .

Since the number of cycles is constant, there are no variations in the latency or input event throughput. Fig. 5.17 shows the memory accesses with different radius

of both memories; it can be observed that the number of memory accesses of the parallel read memory is constant regardless of the radius.

5.3.3.2 Multi-layer experiment

The memory resource reduction does not seem to have a great impact, since only one memory is used in one-layer implementation. However, in a multi-layer HOTS implementation, the memory increases with the number of prototypes, thus the memory consumption can decrease using the parallel read memory model. Table 5.4 shows the properties of the multi-layer HOTS network used in this experiment. The purpose of this experiment was to determine the minimum memory required to implement these networks in order to obtain the same surfaces as RAM-based HOTS.

TABLE 5.4: Properties of HOTS layers used in this experiment.

Network	Layer 0			Layer 1		
	N	R	Tau	N	R	Tau
A	8	2	10ms	8	2	100ms
B	8	2	10ms	8	2	200ms
C	8	2	10ms	8	2	400ms
D	8	2	10ms	8	2	800ms

As was previously mentioned, larger values of tau imply longer integration time, thus larger memories are needed to avoid the loss of significant events. In the one-layer HOTS, only one memory is needed to store the timestamps of the input events, generating a time surface and comparing it with the prototypes. Although in one-layer HOTS implementation pattern events from prototypes are used to feed a classifier, in a multi-layered HOTS the pattern events are stored in memories, grouping the events by patterns. Therefore, these networks require one memory for input events and eight memories for the output patterns of the first layer. Although the first layer of multi-layered networks does not require much time to integrate events, the second layer needs longer time periods to integrate events. Therefore, the TDVs of the memories are set to the τ value of the second layer.

Apart from latency, which was demonstrated in the previous section, the size of each memory used in these networks to avoid losing accuracy is 1024 Kb, assuming 32 bits for event addresses and 32 bits for timestamps. Since in **A** and **B** layers the memory is never full, this memory is not forced to erase events. In layers **C** and **D**, since the tau value is higher (400 ms and 800 ms), the memories become full and older events stored are erased. However, since the erased events are too old, they do not affect the current processing, as the obtained time surfaces are the same as those obtained by RAM-based HOTS networks. Therefore, a total memory of 9216 Kb is enough to run this network without losing precision. Fig. 5.18 shows the memory needed to implement this network for an ATIS and Davis 240C sensor with 32 bits for timestamps and their corresponding bits for event addresses and polarities.

Regarding the memory used by each sensor, the parallel memory presented in this chapter reduces the memory used to store the whole spatial resolution of DAVIS 240C and ATIS sensors to 50% and 72%, respectively, using 64 bits, which is above the requirement for these sensors.

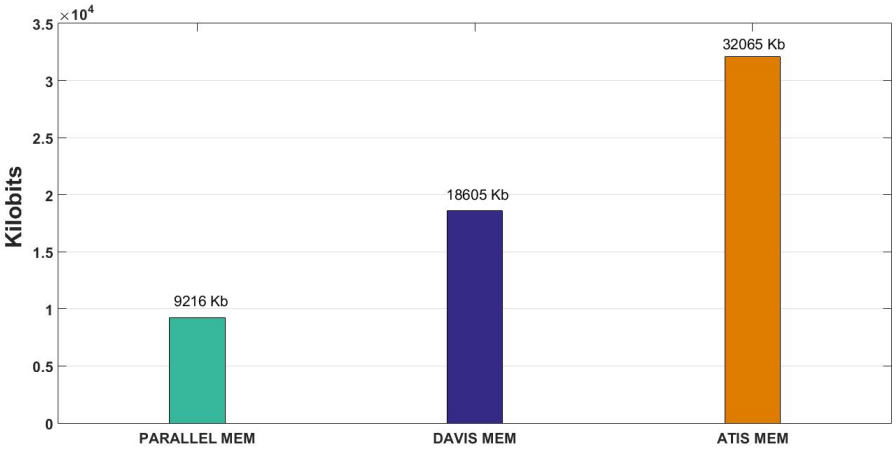


FIGURE 5.18: Total memory consumption of parallel memory with memory consumed by RAM memory for DAVIS 240c and ATIS sensors.

5.4 Summary and Discussion

This chapter presents a novel architecture to infer the HOTS algorithm. Although the system is able to compute a single-layer HOTS network in real time without significant accuracy loss, it cannot infer a multi-layer network. One possible solution would be to implement several processing units in the FPGA, in order to use more prototypes to compare, and thus more patterns. Additionally, this would require a router mechanism as the one seen in chapter 4 for convolution processors to route events from one layer to other. However, a multi-layer implementation would imply more hardware resources and memory accesses, due there are more prototypes to store in memory.

This chapter also presents a novel memory model to store the most recent events, with the aim of reducing latency and memory consumption. The development of this model arises, due to the memory bottlenecks produced in neuromorphic processing systems. The results of the memory model presented in this chapter correspond to a theoretical study prior to synthesis, since it is currently not possible to implement several memories of vast capacity in digital platforms with the current technology. However, the tendency in neuromorphic engineering is to develop new memory models such as the one presented in this chapter, in order to increase the high input throughput of new neuromorphic vision sensors, which have a vast input throughput, such as the one developed by Samsung (Son et al., 2017), which improves the performance of DVS (Lichtsteiner, Posch, and Delbrück, 2008) with a spatial resolution of 640x480 and an input throughput of 300 Mevps, which is difficult to process in real time

TABLE 5.5: Zynq-7100 MMP FPGA (xc7z100-2) resources of each memory model for a 128 position and 64 bit width.

	Cycle-per-Cycle	Parallel read	RAM memory
LUT	9%	23%	1%
LUTRAM/BRAM	2%/1%	4%/-	10%/1%
FF	1%	3%	1%
Memory accesses	128	2	25

Table 5.5 shows the memory resources needed for each type of memory with 128 positions and 64-bit width. The main problem of the memory model developed in comparison with RAM-based memories is that LUT resources are too high to implement a single memory, as can be seen in the cycle-by-cycle and parallel read versions, which require 9% and 23% of LUT, respectively, for a large FPGA platform such as Zynq-7100 MMP.

Memristors, have been demonstrated to be a good technology approach to parallel processing systems (Haron et al., 2016). This makes them ideal for neuromorphic systems spike grids, where neurons have to communicate with their neighbors in parallel (Y. Kim and Li, 2014). This kind of technology could allow implementing the presented memory model in the future, using fewer resources and reducing the memory bottlenecks, thus increasing the latency, as demonstrated in this chapter.

Chapter 6

Conclusions and Future works

> "Imagination means nothing without doing."
>
> Charlie Chaplin

This chapter presents the conclusions of the thesis, its main contributions and future works. Before introducing the conclusions, it is important to highlight the main differences between the two processing paradigms studied in this thesis to assess the implementations of the convolution accelerators.

In Farabet et al., 2012 compared the two processing paradigms for frame-based and event-based convolutions; however, this comparison can be applied to other event-based systems, such as HOTS.

Event-driven processing makes use of sparse information of neuromorphic vision sensors, starting the processing with the arrival of the first event, thus event-based processing is pseudo-simultaneous. On the other hand, frame-based processing requires all the pixels of the frame to start the computation. Furthermore, regarding the speed of both systems, frame-based systems latency depends directly on the speed of hardware resources, regardless of the input stimulus. However, event-based processing speed depends on the number of events received at the input.

Table 6.1 summarizes the main differences between the hardware systems developed in this thesis.

TABLE 6.1: Event-driven vs frame-driven. Table taken from (Farabet et al., 2012).

	Event-based	**Frame-based**
Data processing	Per-event, resulting in pseudo-simultaneity	Per frame/patch
Hardware multiplexing	Not possible	Possible
Hardware up-scaling	By adding modules	Ad hoc
Speed	Determined by statistics of input stimuli	Determined by the number of operations, type of operations, available hardware resources and their speed
Power consumption	Determined by module power per-event and inter-module communication power per-event	Determined by the power of the processor(s) and memory fetching requirements
Feedback	Instantaneous. No need to iterate	Need to iterate until convergence for each frame

6.1 Conclusions

This section highlights the conclusions extracted from the work presented throughout this document:

- A study of the human visual system and the most relevant neuromorphic vision sensors were carried out.
- Study and comprehension of biological neurons properties and characteristics, through an analysis of spike-encoded information.
- Deep learning algorithms were studied in depth, particularly convolutional neural networks, to extract features from input images. This algorithm was studied to be applied in frame-based images.
- Convolutional neural network accelerators for real-time processing were developed for a co-design platform (FPGA + host computer) using OpenCL. The FPGA accelerator was tested with a Lenet-5 network, which classifies the MNIST dataset.
- A SoC-FPGA convolutional neural network accelerator called NullHop was developed as part of the NPP project. NullHop's main property is that it is able to avoid null pixels computation, reducing the latency and power consumption. A convolutional neural network called Roshambo was trained and deployed on NullHop, in order to test the performance of the architecture.
- An in-depth study of the theory behind spiking convolutional neural networks and the most relevant hardware implementations.
- A software tool to convert images from frame-based to event-based algorithm was developed. The tool applies three well-known methods, called Scan, Random, and Bitwise, to convert pixels into a stream of events to be used in event-based processing systems.
- An FPGA spiking convolution processor to infer spiking convolutional neural networks was designed and implemented. This architecture implements several spike convolutions based on leaky integrate-and-fire neurons, firing events with negative or positive polarity. However, the number of spiking events at the output is too high to infer a spiking convolutional neural network, since the only implemented mechanism to stabilize the firing rate is the leakage of the neurons.
- A new version of the spiking convolution processor was developed. In this new version, negative values of membrane potential are not considered for firing events. On the other hand, the convolution processor implements the refractory period property to control the output firing rate. In addition, it also has a new mask mechanism to implement the computation efficiently. Finally, new upgrades were developed for both convolution processor versions to implement several convolution layers using a single chip.
- During a three-month research internship in the Neuromorphic Vision and Natural Computation of the Institut de la Vision, which is headed by Ryad Benosman, a new pattern recognition algorithm for event-based images, called

HOTS, was studied. As a result of this study, a novel VLSI implementation of HOTS algorithm was developed for real-time processing. The architecture was tested to classify six gestures from a novel event-based dataset, called NavGestures-sit; we believe this is the first implementation of HOTS algorithm on a VLSI system.

- A new memory model for event-based processing systems was designed. The main property of this memory is that only recent events are stored, similar to a cache memory in computer systems. This memory was studied and tested under simulation over a HOTS network, reducing considerably the memory access; however, it requires a large amount of LUT resources. For a single memory of 128 positions and 64-bit width, 23% of LUT resources are used in a large FPGA platform, such as Zynq-7100 MMP FPGA.

6.2 Future works

After the tests and results obtained from the Altera OpenCL accelerator, it can be seen that there is a memory bottleneck produced by the different kernels accessing the memory. There exist two possible improvements to reduce latency. The first one consists in changing the kernel structure to allow kernels to store their data in local memory, avoiding memory accesses to global memory. On the other hand, using the pipes mechanism given by Altera OpenCL would allow the kernels to communicate between them, creating pipeline stages without accessing the global memory.

Regarding the spiking convolution processors, they were tested to convolve event-based images with different event streams, although convolution processors are able to process events in real time. As future works, the different trained mechanisms for spiking convolutional neural networks will be studied, in order to use the convolution processor V2 to infer these networks, as it can compute multiple layers of a network. In addition, the development of a mechanism to configure automatically the different parameters of LIF neurons will also be developed.

FPGA HOTS implementation was tested with gesture recognition. Although the accuracy obtained by the architecture does not differ significantly from the software implementation, the architecture cannot infer a multi-layer HOTS network. In the future, the architecture will contain several processing units within a routing module to send pattern events through the layers.

Finally, the event-based memory model proposed in this work was analyzed and studied under post-synthesis simulation for the VLSI and pre-synthesis simulation of a multi-layer network, due to the limitation of logical resources of current hardware platforms. In the future, the memory model will be implemented in order to verify its behavior in an event-based processing hardware platform. In addition, new technologies will be studied to implement this kind of memory, using less logical resources.

6.3 Articles

The main articles and scientific contribution are listed below:

- Journals
  - Ricardo Tapiador Morales, Alejandro Linares Barranco, Angel Jiménez Fernandez and Gabriel Jiménez Moreno, "Neuromorphic LIF Row-by-Row Multiconvolution Processor for FPGA," in IEEE Transactions on Biomedical Circuits and Systems, vol. 13, no. 1, pp. 159-169, Feb. 2019. doi: 10.1109/TBCAS.2018.2880012
  - Alessandro Aimar , Hesham Mostafa, Enrico Calabrese, Antonio Ríos Navarro, Ricardo Tapiador Morales, Iulia Alexandra Lungu *et al*. "NullHop: A Flexible Convolutional Neural Network Accelerator Based on Sparse Representations of Feature Maps" in IEEE Transactions on Neural Networks and Learning Systems, vol. 30, no. 3, pp. 644-656, March 2019.doi: 10.1109/TNNLS.2018.2852335
- Congress
  - Ricardo Tapiador Morales, Antonio Ríos Navarro, Alejandro Linares Barranco, Minkyu Kim, Deepak Kadetotad, and Jae-sun Seo, "Comprehensive Evaluation of OpenCL-Based CNN Implementations for FPGAs". International Work-Conference on Artificial Neural Networks (IWANN), Cadiz, 2017, pp. 271–282.
  - Ricardo Tapiador Morales, Antonio Ríos Navarro, Juan P. Dominguez Morales, Daniel Gutierrez Galán et al., "Event-based Row-by-Row Multi-convolution engine for Dynamic-Vision Feature Extraction on FPGA". International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, 2018, pp. 1-7.doi: 10.1109/IJCNN.2018.8489449
  - Ricardo Tapiador Morales, Antonio Ríos-Navarro, Juan P. Dominguez Morales, Daniel Gutierrez Galán, Alejandro Linares Barranco. "Spiking row-by-row FPGA Multi-kernel and Multi-layer Convolution Processor". International Conference on Field-Programmable Logic and Applications (FPL), Barcelona, 2019.
  - Ricardo Tapiador Morales, Juan P. Dominguez Morales, Daniel Gutierrez Galán, Antonio Ríos Navarro, Angel Jimenez Fernandez, Alejandro Linares Barranco ."Live Demonstration: Neuromorphic Row-by-Row Multi-Convolution FPGA Processor-SpiNNaker Architecture for Dynamic-Vision Feature Extraction". IEEE International Symposium on Circuits and Systems (ISCAS), Sapporo, 2019.

Bibliography

Abdel-Hamid, O. et al. (2014). "Convolutional Neural Networks for Speech Recognition". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 22.10, pp. 1533–1545. ISSN: 2329-9290. DOI: 10.1109/TASLP.2014.2339736.

Aichert, André et al. (2012). "Image-Based Tracking of the Teeth for Orthodontic Augmented Reality". In: *MICCAI (2)*. Vol. 7511. Lecture Notes in Computer Science. Springer, pp. 601–608.

Aimar, A. et al. (2018). "NullHop: A Flexible Convolutional Neural Network Accelerator Based on Sparse Representations of Feature Maps". In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–13. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2018.2852335.

Akopyan, F. et al. (2015). "TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.10, pp. 1537–1557. ISSN: 0278-0070. DOI: 10.1109/TCAD.2015.2474396.

AMBA, ARM (2014). *AXI4-Stream Protocol Specification*.

Amir, A. et al. (2017). "A Low Power, Fully Event-Based Gesture Recognition System". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7388–7397. DOI: 10.1109/CVPR.2017.781.

Barlow, H. B. (1961). "Possible principles underlying the transformations of sensory messages". In: *Sensory Communication*. Ed. by W. A. Rosenblith. Cambridge, MA: MIT Press, pp. 217–234.

Bastien, Frédéric et al. (2012). "Theano: new features and speed improvements". In: *CoRR* abs/1211.5590. arXiv: 1211.5590. URL: http://arxiv.org/abs/1211.5590.

Bekolay, Trevor et al. (2014). "Nengo: a Python tool for building large-scale functional brain models". In: *Frontiers in Neuroinformatics*. ISSN: 1662-5196. DOI: 10.3389/fninf.2013.00048.

Benjamin, Ben Varkey et al. (2014). "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations". In: *Proceedings of the IEEE*. ISSN: 00189219. DOI: 10.1109/JPROC.2014.2313565.

Berger, Arnold S. (2007). "The ARM Architecture". In: *Hardware and Computer Organization*. DOI: 10.1016/b978-075067886-5/50035-5.

Berner, Raphael et al. (2007). "A 5 Meps $100 USB2.0 address-event monitor-sequencer interface". In: *2007 IEEE International Symposium on Circuits and Systems*. IEEE, pp. 2451–2454.

Blei, David M. (2018). "Technical Perspective: Expressive Probabilistic Models and Scalable Method of Moments". In: *Commun. ACM* 61.4, pp. 84–84. ISSN: 0001-0782. DOI: 10.1145/3186260. URL: http://doi.acm.org/10.1145/3186260.

Boahen, Kwabena A. (2000). "Point-to-point connectivity between neuromorphic chips using address events". In: *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*. ISSN: 10577130. DOI: 10.1109/82.842110.

Braitenberg, Valentino and Almut Schüz (2013). "Cortical Architectonics". In: *Cortex: Statistics and Geometry of Neuronal Connectivity*. DOI: 10.1007/978-3-662-03733-1_27.

Brandli, C. et al. (2014). "A 240x180 130dB 3μs Latency Global Shutter Spatiotemporal Vision Sensor". In: *IEEE Journal of Solid-State Circuits* 49.10, pp. 2333–2341.

Browne, Matthew and Saeed Shiry Ghidary (2003). "Convolutional Neural Networks for Image Processing: An Application in Robot Vision". In: *AI 2003: Advances in Artificial Intelligence*. Ed. by Tamás (Tom) Domonkos Gedeon and Lance Chun Che Fung. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 641–652. ISBN: 978-3-540-24581-0.

Cajal, S.R. (1952). *¿Neuronismo o reticularismo?: las pruebas objetivas de la unidad anatómica de las células nerviosas*. Consejo Superior de Investigaciones Cientificas, Instituto Ramón y Cajal.

Camuñas-Mesa, L. et al. (2011). "A 32×32 Pixel Convolution Processor Chip for Address Event Vision Sensors With 155 ns Event Latency and 20 Meps Throughput". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 4, pp. 777–790. ISSN: 1549-8328. DOI: 10.1109/TCSI.2010.2078851.

Camuñas-Mesa, Luis et al. (2012). "An event-driven multi-kernel convolution processor module for event-driven vision sensors". In: *IEEE Journal of Solid-State Circuits* 47.2, pp. 504–517.

Camuñas-Mesa, Luis A. et al. (2018). "A Configurable Event-Driven Convolutional Node with Rate Saturation Mechanism for Modular ConvNet Systems Implementation". In: *Frontiers in Neuroscience* 12, p. 63. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00063. URL: https://www.frontiersin.org/article/10.3389/fnins.2018.00063.

Cao, Yongqiang, Yang Chen, and Deepak Khosla (2015). "Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition". In: *International Journal of Computer Vision* 113.1, pp. 54–66. ISSN: 1573-1405. DOI: 10.1007/s11263-014-0788-3. URL: https://doi.org/10.1007/s11263-014-0788-3.

Cavigelli, Lukas and Luca Benini (2017). "Origami: A 803-GOp/s/W Convolutional Network Accelerator". In: *IEEE Transactions on Circuits and Systems for Video Technology* 27.11, pp. 2461–2475. ISSN: 10518215. DOI: 10.1109/TCSVT.2016.2592330. arXiv: 1512.04295.

Cela-Conde, Camilo J. et al. (2004). "Activation of the prefrontal cortex in the human visual aesthetic perception". In: *Proceedings of the National Academy of Sciences* 101.16, pp. 6321–6325. ISSN: 0027-8424. DOI: 10.1073/pnas.0401427101. eprint: https://www.pnas.org/content/101/16/6321.full.pdf. URL: https://www.pnas.org/content/101/16/6321.

Chan, Vincent, Craig Jin, and André van Schaik (2012). "Neuromorphic Audio-Visual Sensor Fusion on a Sound-Localising Robot". In: *Frontiers in Neuroscience* 6, p. 21. ISSN: 1662-453X. DOI: 10.3389/fnins.2012.00021. URL: https://www.frontiersin.org/article/10.3389/fnins.2012.00021.

Chen, Y H et al. (2017). "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks". In: *IEEE Journal of Solid-State Circuits* 52.1, pp. 127–138. ISSN: 0018-9200. DOI: 10.1109/JSSC.2016.2616357.

Chen, Z. and D. Kaeli (2016). "Balancing Scalar and Vector Execution on GPU Architectures". In: pp. 973–982. ISSN: 1530-2075. DOI: 10.1109/IPDPS.2016.74.

Conradt, J. et al. (2009). "A pencil balancing robot using a pair of AER dynamic vision sensors". In: *Proceedings - IEEE International Symposium on Circuits and Systems*. ISBN: 9781424438280. DOI: 10.1109/ISCAS.2009.5117867.

Courbariaux, Matthieu, Yoshua Bengio, and Jean-Pierre David (2015). "BinaryConnect: Training Deep Neural Networks with binary weights during propagations". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., pp. 3123–3131. URL:

http://papers.nips.cc/paper/5647-binaryconnect-training-deep-neural-networks-with-binary-weights-during-propagations.pdf.

Crouse, David et al. (2015). "Continuous authentication of mobile user: Fusion of face image and inertial Measurement Unit data". In: *Proceedings of 2015 International Conference on Biometrics, ICB 2015*. ISBN: 9781479978243. DOI: 10.1109/ICB.2015.7139043.

Culurciello, Eugenio, Ralph Etienne-Cummings, and Kwabena A. Boahen (2003). "A biomorphic digital image sensor". In: *IEEE Journal of Solid-State Circuits*. ISSN: 00189200. DOI: 10.1109/JSSC.2002.807412.

Dan, Yang and Mu ming Poo (2004). "Spike Timing-Dependent Plasticity of Neural Circuits". In: *Neuron* 44.1, pp. 23 –30. ISSN: 0896-6273. DOI: https://doi.org/10.1016/j.neuron.2004.09.007. URL: http://www.sciencedirect.com/science/article/pii/S0896627304005768.

Delbrück, T (2007). *jAER open source project*. URL: https://github.com/SensorsINI/jaer.

Diehl, P. U. et al. (2015). "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing". In: *2015 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. DOI: 10.1109/IJCNN.2015.7280696.

Domínguez-Morales, J. P. et al. (2018). "Deep Neural Networks for the Recognition and Classification of Heart Murmurs Using Neuromorphic Auditory Sensors". In: *IEEE Transactions on Biomedical Circuits and Systems* 12.1, pp. 24–34. ISSN: 1932-4545. DOI: 10.1109/TBCAS.2017.2751545.

Dominguez-Morales, Juan P. et al. (2017). "NAVIS: Neuromorphic Auditory VISualizer Tool". In: *Neurocomputing* 237, pp. 418 –422. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2016.12.046.

Dong, Wenshuai et al. (2016). "Sliding threshold of spike-rate dependent plasticity of a semiconducting polymer/electrolyte cell". In: *Journal of Polymer Science Part B: Polymer Physics* 54.23, pp. 2412–2417. DOI: 10.1002/polb.24152. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/polb.24152. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/polb.24152.

Dorta, Taho et al. (2016). "AER-SRT: Scalable spike distribution by means of synchronous serial ring topology address event representation". In: *Neurocomputing* 171, pp. 1684 –1690. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2015.07.080. URL: http://www.sciencedirect.com/science/article/pii/S0925231215010875.

Doya, K. et al. (2007). *Bayesian Brain: Probabilistic Approaches to Neural Coding*. The MIT Press.

Elie, Nancy Forbes, and George Strawn (2017). *The End of Moore's Law*. DOI: 10.1109/MCSE.2017.25.

Esser, Steven K. et al. (2016). "Convolutional Networks for Fast, Energy-Efficient Neuromorphic Computing". In: *CoRR* abs/1603.08270. arXiv: 1603.08270. URL: http://arxiv.org/abs/1603.08270.

Farabet, Clément et al. (2012). "Comparison between frame-constrained fix-pixel-value and frame-free spiking-dynamic-pixel ConvNets for visual processing". In: *Frontiers in neuroscience* 6, p. 32.

Fasnacht, Daniel B., Adrian M. Whatley, and Giacomo Indiveri (2008). "A serial communication infrastructure for multi-chip address event systems". In: *Proceedings - IEEE International Symposium on Circuits and Systems*. ISBN: 9781424416844. DOI: 10.1109/ISCAS.2008.4541501.

Fowler, David and Eleanor Robson (1998). "Square Root Approximations in Old Babylonian Mathematics: YBC 7289 in Context". In: *Historia Mathematica* 25.4,

pp. 366 –378. ISSN: 0315-0860. DOI: https://doi.org/10.1006/hmat.1998.2209. URL: http://www.sciencedirect.com/science/article/pii/S0315086098922091.

Fukushima, Kunihiko (1975). "Cognitron: A self-organizing multilayered neural network". In: *Biological Cybernetics*. ISSN: 03401200. DOI: 10.1007/BF00342633.

Fukushima, Kunihiko and Sei Miyake (1982). "Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position". In: *Pattern Recognition*. ISSN: 00313203. DOI: 10.1016/0031-3203(82)90024-3.

Fukushima, Kunihiko et al. (1970). "An Electronic Model of the Retina". In: *Proceedings of the IEEE*. ISSN: 15582256. DOI: 10.1109/PROC.1970.8066.

Furber, Steve (2016a). "Large-scale neuromorphic computing systems". In: *Journal of Neural Engineering* 13.5, p. 051001. DOI: 10.1088/1741-2560/13/5/051001. URL: https://doi.org/10.1088%2F1741-2560%2F13%2F5%2F051001.

— (2016b). "The SpiNNaker project". English. In: *Unconventional computation and natural computation : 15th International Conference, UCNC 2016, Manchester, UK, July 11-15, 2016, Proceedings*. Vol. 9726. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). United Kingdom: Springer U K. ISBN: 9783319413112.

Furber, Steve B et al. (2013). "Overview of the spinnaker system architecture". In: *IEEE Transactions on Computers* 62.12, pp. 2454–2467.

Gokhale, V. et al. (2014). "A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 696–701. DOI: 10.1109/CVPRW.2014.106.

Gokhale, V. et al. (2017). "Snowflake: An efficient hardware accelerator for convolutional neural networks". In: *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4. DOI: 10.1109/ISCAS.2017.8050809.

Gomez-Rodriguez, F., R. Paz, et al. (2005). "Two Hardware Implementations of the Exhaustive Synthetic AER Generation Method". In: *Computational Intelligence and Bioinspired Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 534–540. ISBN: 978-3-540-32106-4.

Gomez-Rodriguez, F. et al. (2007). "AER Auditory Filtering and CPG for Robot Control". In: *2007 IEEE International Symposium on Circuits and Systems*, pp. 1201–1204. DOI: 10.1109/ISCAS.2007.378268.

Goodman, Dan (2008). "Brian: a simulator for spiking neural networks in Python". In: *Frontiers in Neuroinformatics*. ISSN: 16625196. DOI: 10.3389/neuro.11.005.2008.

Gómez-Rodríguez, F. et al. (2016). "ED-Scorbot: A robotic test-bed framework for FPGA-based neuromorphic systems". In: *2016 6th IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, pp. 237–242. DOI: 10.1109/BIOROB.2016.7523630.

Haron, A. et al. (2016). "Parallel matrix multiplication on memristor-based computation-in-memory architecture". In: *2016 International Conference on High Performance Computing Simulation (HPCS)*, pp. 759–766. DOI: 10.1109/HPCSim.2016.7568411.

Heeger, David and Professor David Heeger (2000). *Poisson Model of Spike Generation*.

Hodgkin, A.L. and A.F. Huxley (1939). "Action Potentials Recorded from Inside a Nerve Fibre". In: *Nature* 144, pp. 710–711. DOI: 10.1038/144710a0.

Hwang, Kai, Shun Piao Su, and Lionel M. Ni (1981). "Vector Computer Architecture and Processing Techniques". In: *Advances in Computers*. ISSN: 00652458. DOI: 10.1016/S0065-2458(08)60497-0.

Ielmini, Daniele (2018). "Brain-inspired computing with resistive switching memory (RRAM): Devices, synapses and neural networks". In: *Microelectronic Engineering* 190, pp. 44 –53. ISSN: 0167-9317. DOI: https://doi.org/10.1016/j.mee.2018.01.009. URL: http://www.sciencedirect.com/science/article/pii/S0167931718300157.

Indiveri, G., E. Chicca, and R.J. Douglas (2006). "A VLSI array of low-power spiking neurons and bistable synapses with spike–timing dependent plasticity". In: *IEEE Transactions on Neural Networks* 17.1, pp. 211–221. DOI: 10.1109/TNN.2005.860850. URL: http://ncs.ethz.ch/pubs/pdf/Indiveri_etal06.pdf.

Indiveri, G. and S. Liu (2015). "Memory and Information Processing in Neuromorphic Systems". In: *Proceedings of the IEEE* 103.8, pp. 1379–1397. ISSN: 0018-9219. DOI: 10.1109/JPROC.2015.2444094.

Jhuang, H. et al. (2007). "A Biologically Inspired System for Action Recognition". In: *2007 IEEE 11th International Conference on Computer Vision*, pp. 1–8. DOI: 10.1109/ICCV.2007.4408988.

Jia, Yangqing et al. (2014). "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *arXiv preprint arXiv:1408.5093*.

Jiménez-Fernández, Angel et al. (2017). "A Binaural Neuromorphic Auditory Sensor for FPGA: A Spike Signal Processing Approach". In: *IEEE Trans. Neural Netw. Learning Syst.* 28.4, pp. 804–818.

John C Pearson Jack J Gelfand, W E Sullivan Richard M Peterson Clay D Spence (1988). *Neural Network Approach To Sensory Fusion*. DOI: 10.1117/12.946654. URL: https://doi.org/10.1117/12.946654.

Johnston, D. and S.M.S. Wu (1995). *Foundations of Cellular Neurophysiology*. A Bradford book. Mit Press. ISBN: 9780262100533. URL: https://books.google.es/books?id=f8JnQgAACAAJ.

Joshi Vaibhav Vijay, Balbhim Bansode, ARM Limited and ARM Limited (2013). "ARM Processor Architecture". In: *IJSETR*.

Khan, Aijaz Ahmed, Shashi Wadhwa, and Veena Bijlani (1994). "Development of human lateral geniculate nucleus: An electron microscopic study". In: *International Journal of Developmental Neuroscience* 12.7, pp. 661 –672. ISSN: 0736-5748. DOI: https://doi.org/10.1016/0736-5748(94)90018-3. URL: http://www.sciencedirect.com/science/article/pii/0736574894900183.

Kheradpisheh, Saeed Reza et al. (2018). "STDP-based spiking deep convolutional neural networks for object recognition". In: *Neural Networks* 99, pp. 56 –67. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2017.12.005. URL: http://www.sciencedirect.com/science/article/pii/S0893608017302903.

Khodamoradi, A. and R. Kastner (2018). "O(N)-Space Spatiotemporal Filter for Reducing Noise in Neuromorphic Vision Sensors". In: *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1. ISSN: 2168-6750. DOI: 10.1109/TETC.2017.2788865.

Krug, Kristine (2012). "Principles of function in the visual system". In: *Sensory Perception: Mind and Matter*. Ed. by Friedrich G. Barth, Patrizia Giampieri-Deutsch, and Hans-Dieter Klein. Vienna: Springer Vienna, pp. 41–56. ISBN: 978-3-211-99751-2. DOI: 10.1007/978-3-211-99751-2_3. URL: https://doi.org/10.1007/978-3-211-99751-2_3.

Lagorce, Xavier et al. (2017). "Hots: a hierarchy of event-based time-surfaces for pattern recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 39.7, pp. 1346–1359.

Lazzaro, John et al. (1993). "Silicon Auditory Processors as Computer Peripherals". In: *IEEE Transactions on Neural Networks*. ISSN: 19410093. DOI: 10.1109/72.217193.

LeCun, Y (1989). "Handwritten Digit Recognition with a Back-Propagation Network". In: *Advances in Neural Information Processing Systems*. ISSN: 1524-4725. DOI: 10.1111/dsu.12130. arXiv: 1004.3732.

LeCun, Y. et al. (1989). "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4, pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541.

Lecun, Y. et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. ISSN: 0018-9219. DOI: 10.1109/5.726791.

Lee, Jun Haeng, Tobi Delbrück, and Michael Pfeiffer (2016). "Training Deep Spiking Neural Networks Using Backpropagation". In: *Frontiers in Neuroscience* 10, p. 508. ISSN: 1662-453X. DOI: 10.3389/fnins.2016.00508. URL: https://www.frontiersin.org/article/10.3389/fnins.2016.00508.

Lee, Jun Haeng, Tobi Delbruck, and Michael Pfeiffer (2016). "Training Deep Spiking Neural Networks Using Backpropagation". In: *Frontiers in Neuroscience* 10, p. 508. ISSN: 1662-453X. DOI: 10.3389/fnins.2016.00508. URL: https://www.frontiersin.org/article/10.3389/fnins.2016.00508.

Li, Yamin and Wanming Chu (1996). "A new non-restoring square root algorithm and its VLSI implementations". In: *Proceedings International Conference on Computer Design. VLSI in Computers and Processors*, pp. 538–544. DOI: 10.1109/ICCD.1996.563604.

Li, Z. et al. (2015). "An overview on memristor crossabr based neuromorphic circuit and architecture". In: *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 52–56. DOI: 10.1109/VLSI-SoC.2015.7314391.

Lichtsteiner, P, C Posch, and T Delbrück (2008). "A 128 x 128 120 dB 15 us Latency Asynchronous Temporal Contrast Vision Sensor". In: *IEEE Journal of Solid-State Circuits* 43.2, pp. 566–576. ISSN: 0018-9200. DOI: 10.1109/JSSC.2007.914337.

Lin, C. et al. (2018). "Programming Spiking Neural Networks on Intel's Loihi". In: *Computer* 51.3, pp. 52–61. ISSN: 0018-9162. DOI: 10.1109/MC.2018.157113521.

Linares-Barranco, A. et al. (2006). "On algorithmic rate-coded AER generation". In: *IEEE Transactions on Neural Networks* 17.3, pp. 771–788. ISSN: 1045-9227. DOI: 10.1109/TNN.2006.872253.

Linares-Barranco, A. et al. (2010). "On the AER convolution processors for FPGA". In: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pp. 4237–4240. DOI: 10.1109/ISCAS.2010.5537577.

Linares-Barranco, Alejandro et al. (2018). "Approaching Retinal Ganglion Cell Modeling and FPGA Implementation for Robotics". In: *Entropy* 20.6. ISSN: 1099-4300. DOI: 10.3390/e20060475. URL: https://www.mdpi.com/1099-4300/20/6/475.

Liu Shih-Chii, Delbruck T. et al. (2014). *Event-based neuromorphic systems*. Wiley. DOI: 10.1002/9781118927601.ch6.

Liu, H. et al. (2016). "Combined frame- and event-based detection and tracking". In: *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2511–2514. DOI: 10.1109/ISCAS.2016.7539103.

Liu, Qian and Steve Furber (2016). "Noisy Softplus: A Biology Inspired Activation Function". In: *Neural Information Processing*. Ed. by Akira Hirose et al. Cham: Springer International Publishing, pp. 405–412. ISBN: 978-3-319-46681-1.

Liu, Shih-Chii et al. (2002). *Analog VLSI: Circuits and Principles*. Cambridge, MA, USA: MIT Press. ISBN: 0262122553.

Maass, Wolfgang (1997). "Networks of spiking neurons: The third generation of neural network models". In: *Neural Networks*. ISSN: 08936080. DOI: 10.1016/S0893-6080(97)00011-7.

Maass, Wolfgang and Christopher M. Bishop, eds. (1999). *Pulsed Neural Networks*. Cambridge, MA, USA: MIT Press. ISBN: 0-626-13350-4.

Mahowald, M. (1992). "VLSI Analogs of Neuronal Visual Processing: A Synthesis of Form and Function". In: *Technology*.

Mahowald, Misha and Rodney Douglas (1991). "A silicon neuron". In: *Nature*. ISSN: 00280836. DOI: 10.1038/354515a0.

Maro, Jean-Matthieu and Ryad Benosman (2018). "Event-based Gesture Recognition with Dynamic Background Suppression using Smartphone Computational Capabilities". In: *CoRR* abs/1811.07802. arXiv: 1811.07802. URL: http://arxiv.org/abs/1811.07802.

Matthaei, Richard et al. (2015). "Autonomous driving". In: *Handbook of Driver Assistance Systems: Basic Information, Components and Systems for Active Safety and Comfort*. ISBN: 9783319123523. DOI: 10.1007/978-3-319-12352-3_61. arXiv: 1704.03952.

Mead, Carver (1989). *Analog VLSI and Neural Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0-201-05992-4.

Moeys, Diederik Paul et al. (2016). "Live demonstration: Retinal ganglion cell software and FPGA implementation for object detection and tracking". In: *Proceedings - IEEE International Symposium on Circuits and Systems*. ISBN: 9781479953400. DOI: 10.1109/ISCAS.2016.7527528.

Moore, Gordon E. (2006). "Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff." In: *IEEE Solid-State Circuits Newsletter* 20, pp. 33–35.

Moradi, Saber et al. (2018). "A Scalable Multicore Architecture with Heterogeneous Memory Structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs)". In: *IEEE Transactions on Biomedical Circuits and Systems*. ISSN: 19324545. DOI: 10.1109/TBCAS.2017.2759700.

Mueggler, Elias et al. (2015). "Lifetime estimation of events from Dynamic Vision Sensors". In: *Proceedings - IEEE International Conference on Robotics and Automation*. DOI: 10.1109/ICRA.2015.7139876.

Orchard, G. et al. (2015a). "HFirst: A Temporal Approach to Object Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.10, pp. 2028–2040. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2015.2392947.

Orchard, Garrick et al. (2015b). "Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades". In: *Frontiers in Neuroscience* 9, p. 437. ISSN: 1662-453X. DOI: 10.3389/fnins.2015.00437. URL: https://www.frontiersin.org/article/10.3389/fnins.2015.00437.

Paz-Vicente, R. et al. (2006). "PCI-AER interface for neuro-inspired spiking systems". In: *2006 IEEE International Symposium on Circuits and Systems*, 4 pp.–. DOI: 10.1109/ISCAS.2006.1693296.

Pérez-Carrasco, José Antonio et al. (2013). "Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing - Application to feedforward convnets". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.11, pp. 2706–2719. ISSN: 01628828. DOI: 10.1109/TPAMI.2013.71. arXiv: 1602.06925.

Perez-Peña, F., A. Linares-Barranco, and E. Chicca (2014). "An approach to motor control for spike-based neuromorphic robotics". In: *2014 IEEE Biomedical Circuits*

and Systems Conference (BioCAS) Proceedings, pp. 528–531. DOI: 10.1109/BioCAS.2014.6981779.

Perez-Peña, F. et al. (2015). "Event-based control system on FPGA applied to the pencil balancer robotic platform". In: *2015 International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*, pp. 1–5. DOI: 10.1109/EBCCSP.2015.7300699.

Perez-Peña, Fernando et al. (2013). "Neuro-Inspired Spike-Based Motion: From Dynamic Vision Sensor to Robot Motor Open-Loop Control through Spike-VITE". In: *Sensors* 13.11, pp. 15805–15832. ISSN: 1424-8220. DOI: 10.3390/s131115805. URL: http://www.mdpi.com/1424-8220/13/11/15805.

Piromsopa, Krerk, Chatchawit Arporntewan, and P Chongstitvatana (2001). *An FPGA Implementation of a Fixed-Point Square Root Operation*.

Posch, C, D Matolin, and R Wohlgenannt (2011). "A QVGA 143 dB Dynamic Range Frame-Free PWM Image Sensor With Lossless Pixel-Level Video Compression and Time-Domain CDS". In: *IEEE Journal of Solid-State Circuits* 46.1, pp. 259–275. ISSN: 0018-9200. DOI: 10.1109/JSSC.2010.2085952.

Rachmuth, Guy et al. (2011). "A biophysically-based neuromorphic model of spike rate- and timing-dependent plasticity". In: *Proceedings of the National Academy of Sciences* 108.49, E1266–E1274. ISSN: 0027-8424. DOI: 10.1073/pnas.1106161108. eprint: https://www.pnas.org/content/108/49/E1266.full.pdf. URL: https://www.pnas.org/content/108/49/E1266.

Ran, Lingyan et al. (2017). "Convolutional Neural Network-Based Robot Navigation Using Uncalibrated Spherical Images". In: *Sensors* 17.6. ISSN: 1424-8220. DOI: 10.3390/s17061341. URL: http://www.mdpi.com/1424-8220/17/6/1341.

Rea, Francesco, Giorgio Metta, and Chiara Bartolozzi (2013). "Event-driven visual attention for the humanoid robot iCub". In: *Frontiers in Neuroscience* 7, p. 234. ISSN: 1662-453X. DOI: 10.3389/fnins.2013.00234. URL: https://www.frontiersin.org/article/10.3389/fnins.2013.00234.

Reich, Daniel S. et al. (2000). "Interspike Intervals, Receptive Fields, and Information Encoding in Primary Visual Cortex". In: *Journal of Neuroscience* 20.5, pp. 1964–1974. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.20-05-01964.2000. eprint: http://www.jneurosci.org/content/20/5/1964.full.pdf. URL: http://www.jneurosci.org/content/20/5/1964.

Rios-Navarro, A. et al. (2015). "Real-Time motor rotation frequency detection with event-based visual and spike-based auditory AER sensory integration for FPGA". In: *Proceedings of 1st International Conference on Event-Based Control, Communication and Signal Processing, EBCCSP 2015*. ISBN: 9781467378888. DOI: 10.1109/EBCCSP.2015.7300696.

Rios-Navarro, Antonio et al. (2018). "Performance evaluation over HW/SW co-design SoC memory transfers for a CNN accelerator". In: *CoRR* abs/1806.01106. arXiv: 1806.01106. URL: http://arxiv.org/abs/1806.01106.

Rueckauer, Bodo et al. (2017). "Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification". In: *Frontiers in Neuroscience* 11, p. 682. ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00682. URL: https://www.frontiersin.org/article/10.3389/fnins.2017.00682.

Rüedi, Pierre François et al. (2003). "A 128x128 Pixel 120-dB Dynamic-Range Vision-Sensor Chip for Image Contrast and Orientation Extraction". In: *IEEE Journal of Solid-State Circuits*. DOI: 10.1109/JSSC.2003.819169.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). "Learning representations by back-propagating errors". In: *Nature* 323.6088, pp. 533–536.

ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://doi.org/10.1038/323533a0.

Schmitt, Sebastian et al. (2017). "Neuromorphic hardware in the loop: Training a deep spiking network on the BrainScaleS wafer-scale system". In: *IJCNN*. IEEE, pp. 2227–2234.

Serrano-Gotarredona, R. et al. (2008). "On Real-Time AER 2-D Convolutions Hardware for Neuromorphic Spike-Based Cortical Processing". In: *IEEE Transactions on Neural Networks* 19.7, pp. 1196–1219. ISSN: 1045-9227. DOI: 10.1109/TNN.2008.2000163.

Serrano-Gotarredona, Rafael et al. (2009). "CAVIAR: A 45k neuron, 5M synapse, 12G connects/s AER hardware sensory-processing-learning-actuating system for high-speed visual object recognition and tracking". In: *IEEE Transactions on Neural Networks*. ISSN: 10459227. DOI: 10.1109/TNN.2009.2023653.

Serrano-Gotarredona, T and B Linares-Barranco (2013). "A 128x128 1.5% Contrast Sensitivity 0.9% FPN 3 μs Latency 4 mW Asynchronous Frame-Free Dynamic Vision Sensor Using Transimpedance Preamplifiers". In: *IEEE Journal of Solid-State Circuits* 48.3, pp. 827–838. ISSN: 0018-9200. DOI: 10.1109/JSSC.2012.2230553.

Serrano-Gotarredona, T. et al. (2015). "ConvNets experiments on SpiNNaker". In: *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2405–2408. DOI: 10.1109/ISCAS.2015.7169169.

Serrano-Gotarredona, Teresa and Bernabé Linares-Barranco (2015). "Poker-DVS and MNIST-DVS. Their History, How They Were Made, and Other Details". In: *Frontiers in Neuroscience* 9, p. 481. ISSN: 1662-453X. DOI: 10.3389/fnins.2015.00481. URL: https://www.frontiersin.org/article/10.3389/fnins.2015.00481.

Shouval, Harel, Samuel Wang, and Gayle Wittenberg (2010). "Spike Timing Dependent Plasticity: A Consequence of More Fundamental Learning Rules". In: *Frontiers in Computational Neuroscience* 4, p. 19. ISSN: 1662-5188. DOI: 10.3389/fncom.2010.00019. URL: https://www.frontiersin.org/article/10.3389/fncom.2010.00019.

Shrestha, Sumit Bam and Garrick Orchard (2018). "SLAYER: Spike Layer Error Reassignment in Time". In: *CoRR* abs/1810.08646. arXiv: 1810.08646. URL: http://arxiv.org/abs/1810.08646.

Simonyan, Karen and Andrew Zisserman (2014). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556. arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

Sironi, Amos et al. (2018). "HATS: Histograms of Averaged Time Surfaces for Robust Event-based Object Classification". In: *CoRR* abs/1803.07913. arXiv: 1803.07913. URL: http://arxiv.org/abs/1803.07913.

Sivilotti, Massimo Antonio (1991). "Wiring Considerations in Analog VLSI Systems, with Application to Field-programmable Networks". UMI Order No. GAX91-37292. PhD thesis. Pasadena, CA, USA.

Smith, A.W., L.J. McDaid, and S. Hall (2014). "A compact spike-timing-dependent-plasticity circuit for floating gate weight implementation". In: *Neurocomputing* 124, pp. 210 –217. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2013.07.007. URL: http://www.sciencedirect.com/science/article/pii/S0925231213007418.

Son, B. et al. (2017). "A 640x480 dynamic vision sensor with a 9μm pixel and 300Meps address-event representation". In: *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 66–67. DOI: 10.1109/ISSCC.2017.7870263.

Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15, pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

Stromatias, Evangelos et al. (2017b). "An event-driven classifier for spiking neural networks fed with synthetic or dynamic vision sensor data". In: *Frontiers in Neuroscience*. ISSN: 1662453X. DOI: 10.3389/fnins.2017.00350.

Stromatias, Evangelos et al. (2017a). "An Event-Driven Classifier for Spiking Neural Networks Fed with Synthetic or Dynamic Vision Sensor Data". In: *Frontiers in Neuroscience* 11, p. 350. ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00350. URL: https://www.frontiersin.org/article/10.3389/fnins.2017.00350.

Sutikno, Tole et al. (2014). "Strategies for FPGA Implementation of Non-Restoring Square Root Algorithm". In: *International Journal of Electrical and Computer Engineering*. ISSN: 20888708. DOI: 10.11591/ijece.v4i4.6008.

Szegedy, Christian et al. (2015). "Going Deeper with Convolutions". In: *Computer Vision and Pattern Recognition (CVPR)*. URL: http://arxiv.org/abs/1409.4842.

Thorpe, Simon J., Adrien Brilhault, and José Antonio Perez-Carrasco (2010). "Suggestions for a biologically inspired spiking retina using order-based coding". In: *ISCAS 2010 - 2010 IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems*. ISBN: 9781424453085. DOI: 10.1109/ISCAS.2010.5537898.

Vreeken, Jilles (2002). "Spiking neural networks , an introduction". In: *Computing*.

Westerman, Wayne C., David P. M. Northmore, and John. G. Elias (1997). "Neuromorphic Synapses for Artificial Dendrites". In: *Analog Integrated Circuits and Signal Processing*.

Whitmore, Bradley C. and Francois Schweizer (2002). "Hubble space telescope observations of young star clusters in NGC-4038/4039, 'the antennae' galaxies". In: *The Astronomical Journal*. ISSN: 00046256. DOI: 10.1086/117334.

Wiesel, T (1963). "Single-cell responses in striate cortex of kittens deprived of vision in one eye". In: *J Neurophysiol*.

Wu, H. et al. (2017). "Device and circuit optimization of RRAM for neuromorphic computing". In: *2017 IEEE International Electron Devices Meeting (IEDM)*, pp. 11.5.1–11.5.4. DOI: 10.1109/IEDM.2017.8268372.

Xiu, Liming (2019). *Time Moore: Exploiting Moore's Law from the Perspective of Time*. DOI: 10.1109/MSSC.2018.2882285.

Y. Kim, and and P. Li (2014). "Architectural design exploration for neuromorphic processors with memristive synapses". In: *14th IEEE International Conference on Nanotechnology*, pp. 962–966. DOI: 10.1109/NANO.2014.6967962.

Yang, M. et al. (2016). "A 0.5 V 55 μW 64 $\times$ 2 Channel Binaural Silicon Cochlea for Event-Driven Stereo-Audio Sensing". In: *IEEE Journal of Solid-State Circuits* 51.11, pp. 2554–2569. ISSN: 0018-9200. DOI: 10.1109/JSSC.2016.2604285.

Yousefzadeh, A. et al. (2017). "On Multiple AER Handshaking Channels Over High-Speed Bit-Serial Bidirectional LVDS Links With Flow-Control and Clock-Correction on Commercial FPGAs for Scalable Neuromorphic Systems". In: *IEEE Transactions on Biomedical Circuits and Systems* 11.5, pp. 1133–1147. ISSN: 1932-4545. DOI: 10.1109/TBCAS.2017.2717341.

Yousefzadeh, A. et al. (2018). "Performance Comparison of Time-Step-Driven versus Event-Driven Neural State Update Approaches in SpiNNaker". In: *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4. DOI: 10.1109/ISCAS.2018.8350990.

Zaghloul, Kareem A. and Kwabena Boahen (2004a). "Optic Nerve Signals in a Neuromorphic Chip I: Outer and Inner Retina Models". In: *IEEE Transactions on Biomedical Engineering*. ISSN: 00189294. DOI: 10.1109/TBME.2003.821039.

— (2004b). "Optic Nerve Signals in a Neuromorphic Chip II: Testing and Results". In: *IEEE Transactions on Biomedical Engineering*. ISSN: 00189294. DOI: 10.1109/TBME.2003.821040.

Zhang, X. et al. (2018). "An Artificial Neuron Based on a Threshold Switching Memristor". In: *IEEE Electron Device Letters* 39.2, pp. 308–311. ISSN: 0741-3106. DOI: 10.1109/LED.2017.2782752.