Microchips convolucionadores AER para procesado asíncrono neocortical de información sensorial visual codificada en eventos

Memoria presentada por

LUIS ALEJANDRO CAMUÑAS MESA

para optar al título de doctor.

Sevilla, Marzo 2010.

Director

Dr. Bernabé Linares Barranco

Codirectores

Dra. Teresa Serrano Gotarredona **Dr. Antonio José Acosta Jiménez**

Tesis realizada en el Instituto de Microelectrónica de Sevilla, Centro Nacional de Microelectrónica (IMSE-CNM), perteneciente al Consejo Superior de Investigaciones Científicas (CSIC)

UNIVERSIDAD DE SEVILLA

Departamento de Electrónica y Electromagnetismo

Agradecimientos

Cuando el presente documento empezó a tomar forma, supe que con él culminaba una etapa importante de mi vida, e inmediatamente después tuve claro que no había llegado hasta aquí solo, sino que había mucha gente detrás de mí a lo largo de todos estos años. Así comenzó la sección de agradecimientos.

Al principio pensé englobar toda esta sección dando gracias a la vida, que me ha dado tanto, pero para ahorrarme pagar derechos de autor a los herederos de Violeta Parra decidí recurrir a mis propias palabras.

Resulta impensable empezar a hablar sin mencionar en primer lugar a las personas que me han dirigido durante estos años, sin los cuales esta tesis no existiría. Por eso quiero comenzar dando las gracias a Bernabé por haberme guiado pacientemente hasta aquí, por supuesto con la ayuda permanente de Teresa y Antonio, que siempre han tenido un momento para resolver mis dudas y corregir mis errores.

Muchas otras personas han sido importantes en el desarrollo de este trabajo, tantas que es difícil nombrarlas a todas. Por una parte, todos los compañeros que han formado parte de mi grupo de investigación, especialmente Rafa, que con su trabajo marcó el camino que yo continué. Y por otra parte, los compañeros del Departamento de Arquitectura y Tecnología de Computadores, por facilitar siempre mi labor en el laboratorio.

Es inevitable recordar también a los demás grupos con los que tuve la suerte de colaborar en el proyecto CAVIAR, tanto de la Universidad de Oslo como de la Universidad de Zürich, donde tan bien me acogieron durante mi estancia en tierras helvéticas.

No puedo dejar de mencionar a tantos compañeros que han pasado por el Instituto de Microelectrónica de Sevilla a lo largo de este tiempo, de los que tanto he aprendido, especialmente a los que han estado compartiendo despa-

cho conmigo cada día, y que han dejado de ser solamente compañeros para convertirse en amigos.

Todos estos agradecimientos han estado referidos al ámbito laboral, pero lógicamente por encima de todo nunca podré terminar de darles las gracias a mis padres, porque todo lo que soy se lo debo a ellos, y tienen el enorme mérito de haber conseguido darme todas las oportunidades que ellos nunca tuvieron.

En el ámbito formativo, esta tesis supone una meta importante tras toda una vida estudiando, así que no puedo olvidar que la primera persona que ejerció de profesora conmigo fue mi hermana. Por eso y por haber estado siempre a mi lado le doy las gracias, sin olvidarme de mi cuñado Rafa, cuyo apoyo en los últimos años ha sido muy importante para mí.

A la hora de referirme a los amigos, ni siquiera es necesario que los nombre, porque ellos ya saben perfectamente quiénes son y cuánto les agradezco que hayan estado conmigo siempre que lo he necesitado.

Mucho más que agradecimiento le debo a Carmen, por hacer que todo tenga sentido, por ser mi única certeza en un mundo de incertidumbres, por jugar todos los días con la luz del Universo.

Obligado ya por la excesiva extensión de esta sección, doy por cumplida la ración de agradecimientos insistiendo en que todos los aquí nombrados tienen una parte de “culpa” en la finalización de este trabajo.

Índice

CAPÍTULO 1

Introduction

1.1. Antecedents

In recent years, computer processing capabilities have been increased to reach levels unbelievable not long ago. This evolution has allowed to design artificial systems which are able to perform more and more complex tasks. However, when working with sensorial perception there is an important limitation, in particular for some applications like object recognition performed by the brain from visual information. This limitation shows that biological systems carry out these kind of tasks much faster that artificial systems, although they are based on much slower processing units (electronic systems work with time ranges of nanoseconds, while neurons deal with milliseconds, which corresponds to a difference of 6 orders of magnitude).

The main reason for this advantage in biological systems is related to its architecture, massively parallel, the coding schemes used for sensorial information, and the processing techniques. While conventional computers are programmed to follow sequential algorithms executing one instruction after the other, the brain follows a completely different processing scheme. It is estimated that there are around 10^{11} [82] neurons inside the brain, with

10^{15} [83] connections between them. This structure makes the brain specially efficient to perform tasks which require a parallel computation.

Conventional artificial vision systems carry out a sequential processing of the information, but this is not the only drawback, as they operate by capturing and processing sequences of frames. With a certain sampling rate, traditional vision systems obtain a sequence of images, and each one of them is processed to extract some feature which will be combined with all the different features to produce a final result for recognition operation. Nevertheless, biological systems are not based on frames, but on events, which are the electrical spikes generated by the retina and sent to the brain cortex each time its activity reaches a certain threshold. This activity can be referred to different visual properties, like intensity, contrast or movement. Hence, the more active pixels will produce a larger number of events, so that when these events are processed by the cortex, it can detect patterns immediately due to the spatial correlation of the events, produced by an object moving in front of the retina. This operation is performed as the events are generated, without waiting for an artificial sampling time like frame-based systems do, so it can produce very fast results.

In Fig. 1.1 we can see an example of how an event-based multilayer vision processing system works. There is a motion retina which generates events corresponding to the pixels in the object contour. These events are sent to a first processing layer. Each processing layer consists on several processing units that extract a certain feature. These features are extracted from the spatial correlation between the events, and this operation is implemented as a bidimensional convolution. Thus, the first layer in Fig. 1.1 performs some parallel convolutions, each one of them to find a certain feature, and produces more output events. The output events generated by the convolution systems in the first layer are sent to the parallel convolution systems in the second processing layer. This second layer will combine the results received to produce more output events. Following this processing structure, the brain cortex performs object recognition tasks with 8 or 10 layers of neurons.

A very interesting property of this kind of systems is the high-speed to process visual information, as is represented in Fig. 1.1. As an event is gen-

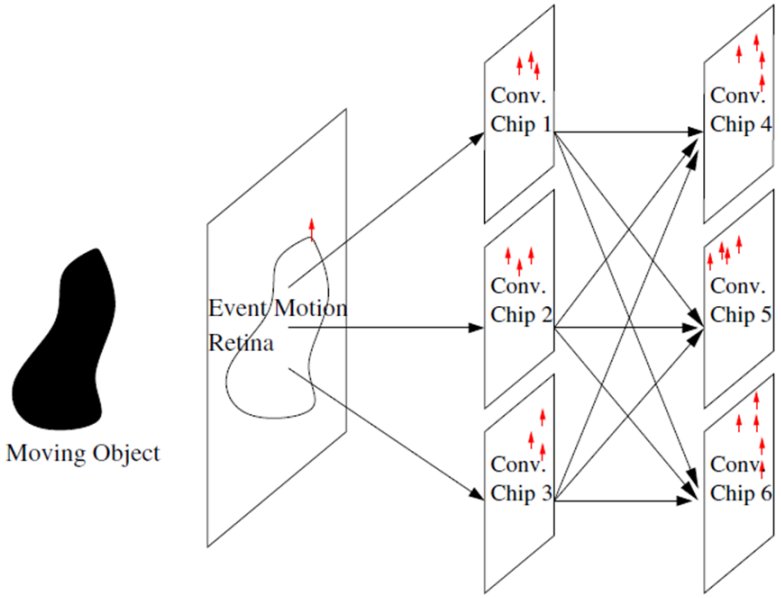


FIGURA 1.1. Example of an event-based multilayer vision processing system.

erated by the retina, it is propagated through the different processing layers, and produces results almost immediately.

When emulating the behavior of the brain with a structure based on convolution systems like the one in Fig. 1.1, we find an important limitation. Each convolution system is formed by a large amount of processing units (pixels), and each one of these processing units needs to communicate with the pixels in the next layer. However, electronic devices have some physical imitations for interconnecting pixel populations in the order of 10^3 which are integrated in different chips. To overcome this limitation we use the Address Event Representation protocol (AER). With this protocol, a large number of neurons integrated in one chip can communicate with the neurons in a different chip by multiplexing connections in a single digital asynchronous bus. Hence, the events generated by each neuron can propagate through the different layers of the processing systems using AER buses.

1.2. Objectives

In this work, we present two different versions of fully digital convolution chips based on the AER protocol for event-based vision processing systems. This chips are the main processing unit to build complex multilayer systems by interconnecting a large number of them. Each one of them calculates the convolution of the input with a programmable kernel of size up to 32×32.

The first version called Conv1 includes a 32×32 array of pixels, although it is designed to build higher resolution systems by connecting several samples in parallel. The second version called Conv2 includes an array 4 times larger with 64×64 pixels, and also implements the multikernel capability. This capability allows to program several kernels in a single chip (up to 32) so that it can receive events from different chips in the previous layer, and apply a specific kernel depending on the origin of the event. This is very useful to build large multilayer systems, emulating the brain cortex, and following the structures known as “Convolutional Neural Networks”.

This dissertation describes in detail the architectures of both proposed versions of convolution chips, and shows some experimental results obtained.

1.3. Structure of this document

The document is structured as follows. First, in Chapter 2, the main advantages of event-based vision processing systems against traditional frame-based structures is discussed. This chapter also includes a description of the AER protocol, essential to implement event-based systems. Chapter 3 reviews bio-inspired multilayer structures for vision processing, motivating the use of the convolution operation as the basis of these systems, and describing the proposed architecture for the AER convolution chips. In Chapter 4, we describe in detail the convolution pixel as the main block of the designed chips, analyzing both versions, while in Chapter 5 we present the rest of the peripheral circuits included in the convolution chips. Chapter

6 shows exhaustive experimental results obtained for both Conv1 and Conv2. Finally, Chapter 7 draws some conclusions.

CAPÍTULO 2

Event-based processing systems

2.1. Introduction

When performing real-time vision processing, traditional frame-based systems present important limitations. The main reason for these limitations is the way they sense and process visual information frame by frame. If we compare artificial systems with the biological ones, we find an important conclusion: the basic processing cell inside the brain (the neuron) is much slower than computers for implementing simple operations. However, the brain is much more efficient due to its behavior based on parallelism.

While conventional frame-based vision schemes work in a sequential way (once they finish performing a certain operation to a frame, they can start with the next task), biological systems are based in the parallel execution of many tasks. To implement this parallelism, each neuron must send information coded in spikes to many other neurons at the same time. This way, information is not processed on a frame basis, but on spikes, also called events.

On the other hand, to make this event-based scheme efficient, it is necessary to implement a massive interconnection between neurons, so that

information can be processed in parallel. To reach this high connectivity we use AER (Address Event Representation) protocol, which allows for two large populations of neuron to communicate by multiplexing their connections through a single common digital bus.

In this chapter we explain why we use event-based processing systems. Hence, Section 2.2 describes the traditional frame-based systems, while Section 2.3 shows the details of the event-bases structure, illustrating its main advantages. Finally, Section 2.4 describes the AER protocol.

2.2. Frame-based visual representation

When working on image processing, one of the firsts matters to consider is how to represent visual information so that it can be possible to manage it. In general, the concept of frames is so strongly associated to moving images that it is often taken for granted.

Images in the real world are continuous in space and in time, so the firs step must lead us to sample in time, which means to capture static images at regular intervals [1]. Each one of these images is called a frame, and by capturing frames at regular intervals and reproducing them one after the other they recreate the movement impression to the human eye, as shown in Fig. 2.1. To achieve that, it is very important to chose a proper sampling time T_s. In general, it is used the frame frequency $f_s = 1/T_s$, expressed in frames per second (fps) or Hertz. For some of the most known systems, frame frequencies can be 16-18Hz for old silent films or 24Hz for present movies. Concerning television, there are several coding standards. Both PAL [3] *(Phase Alterning Line)* system and SECAM *(Séquentiel Couleur à Mémoire)*, used in Europe, Asia, Africa, Oceania and part of South America, sample images at 25 Hz, which is half of the electric current frequency in these countries (50Hz). However, NTSC system *(National Television System Committee)* [4], used in most of America and Japan, take a sampling frequency of 29.97 Hz, which is almost half of the electric current frequency of 60Hz extended in these countries.

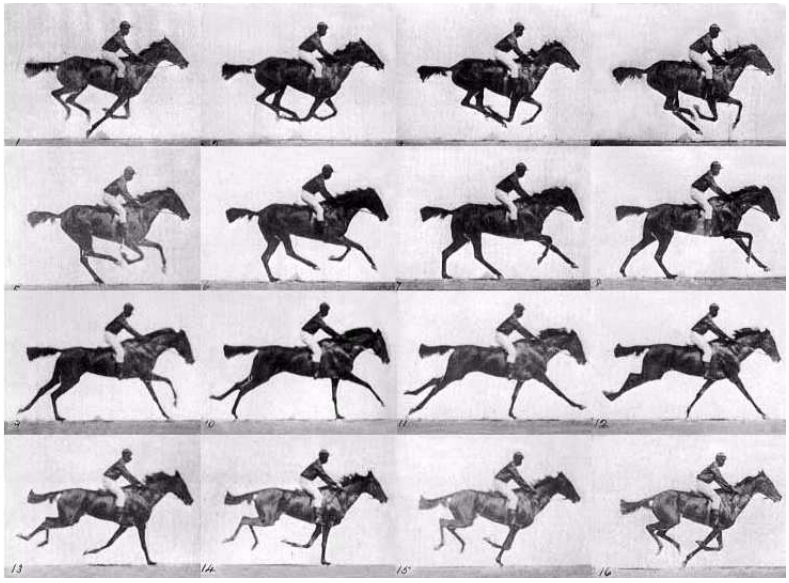


FIGURA 2.1. Sequence of a running horse published by Eadweard Muybridge in 1887 in Philadelphia [2]. These 16 frames reproduced one after the other give the impression of movement.

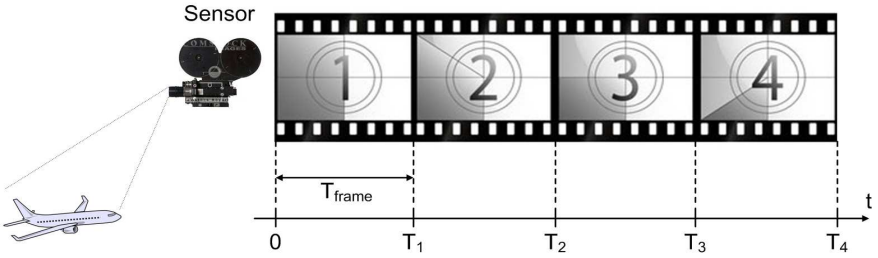


FIGURA 2.2. Description of frame-based vision processing.

Traditional frame-based vision processing systems behave as it is shown in Fig. 2.2. The camera or sensor captures a static image after T_{frame}, storing the information from all the pixels. When these frames are the input of a processing system, we face the following limitations. First, the information about any change that happens between two sample instants T_i is

lost, making it impossible to use this system for an application where an object is moving faster than the sampling frequency. Also, if most of the input image remains unchanged between two sampling instants (or even the whole image remains unchanged), the system will capture and process a whole frame, although no information will be obtained. This is why frame-based processing systems are not useful for high-speed applications, but they are also inefficient for low-speed applications. A consequence is that each sample time the whole image must be processed (a lot of information), which means large computational load, but everything must finish before the next sampling instant (when the following frame is captured). For this reason, it is not possible to implement a very complex computation based on this scheme.

2.3. Event-based visual representation

Biological brains do not process visual information frame by frame, they are based on events [5]. In a retina, each pixel sends a spike (also called event) to the brain cortex when its level reaches a certain threshold, so the information is transmitted as soon as it is produced, without waiting for an artificial sampling time. Retinae can sense different features, like intensity changes [6] or spatial contrast [7], and when a pixel detects a certain level in these features a spike is sent (in general, including the information about what pixel produced it). Thus, when an event is generated, the state of the whole system is updated, although only the pixels with some information are processed.

Fig. 2.3 illustrates the sensing and processing in an event-based system. While the camera on the top of the picture captures a whole frame after each regular interval T_{frame}, the sensor on the bottom generates output events continuously. These events represent the information about the input image. The computation system updates its state after each event, performing more simple processing tasks than the frame-based system, as only a certain part of the image is affected by an event.

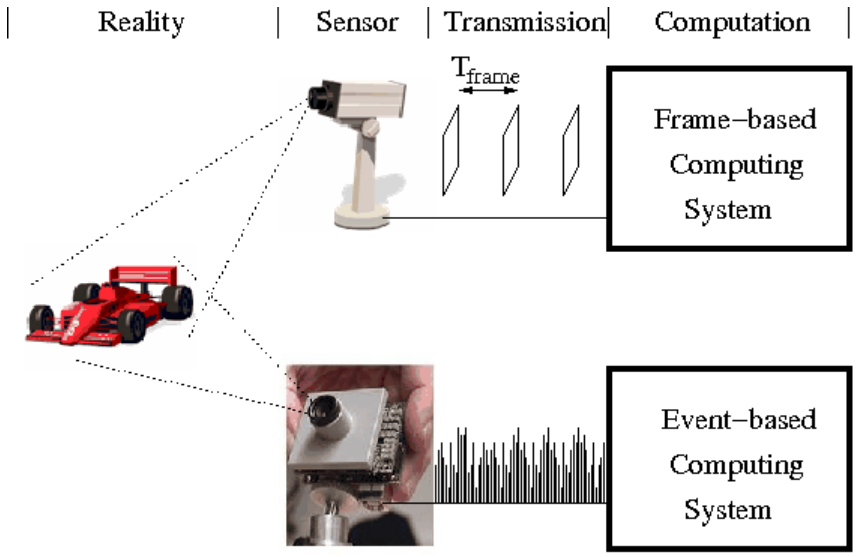


FIGURA 2.3. Comparison between frame-based sensing and processing and event based representation.

2.3.1. Advantages of event-based systems

The main advantage of event-based processing systems is related to the fact that the most relevant information is sent first (so it is also processed first). This happens for any different coding schemes [8], [9], as the events generated first will always correspond to the neurons with more intense inputs. As the most important information is processed first, an approximate result can be obtained very fast, with only the very first events. This is not possible with frame-based systems, as they need whole frames.

This advantage is illustrated in Fig. 2.4. On the top of the figure it is shown how a frame-based system behaves when something happens between instants 0 and T_1, represented with a red shape. Whenever it happens, the information produced will not be ready to be processed by the computing system until the whole frame is captured in T_1, with an extra delay Δ due to transmission. Only after this instant, the computing system can start processing the information received, what will take a relatively

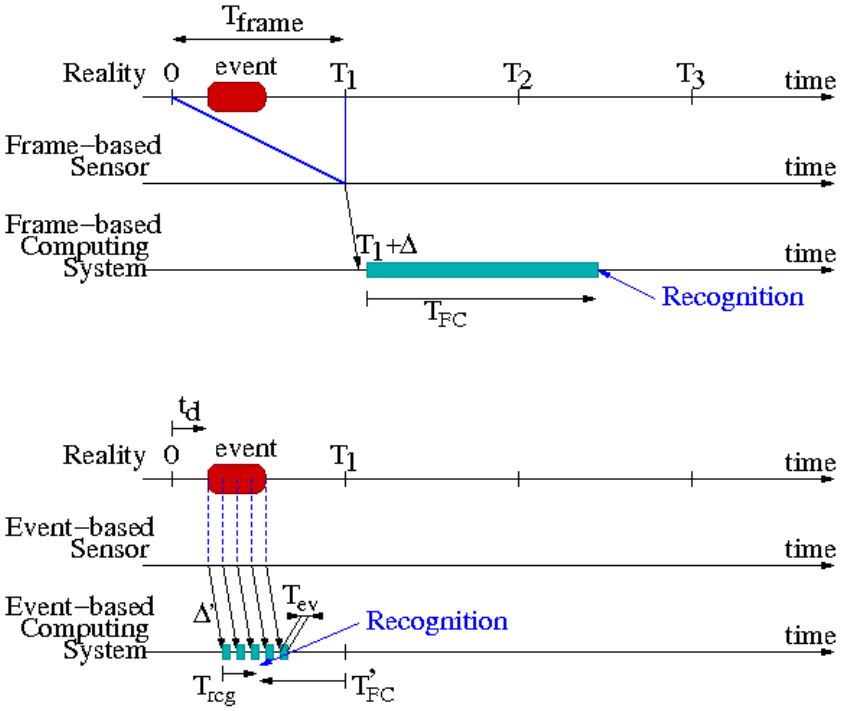


FIGURA 2.4. Comparison between the time response for a frame-based system and an event-based one.

long time T_{FC}, due to the large amount of unnecessary information included in the frame. After that, the desired recognition is obtained. However, as is described in the bottom of the figure for event-based systems, when a single pixel detects a change in reality, it sends immediately en event, with a transmission delay Δ'. Each event needs a very short time T_{ev} to be processed (in the range of nanoseconds). Also, as the first events carry the main information, for the computing system it is not even necessary to process all the events generated to obtain a response, so the recognition task can be performed much faster than the previous system, even before the sampling time T_1.

2.3.2. Coding schemes

Once described the event-based processing systems, and their main advantages, how is visual information coded into these events? There are several alternatives.

1. Event-frequency coding. Consider we have a sensor which detects light intensity. With this coding scheme, each pixel generates a train of events whose frequency is proportional to that pixel's intensity. The main drawback is that a lot of events must be integrated to decode the information.
2. Time-to-first-spike coding. After receiving a global reset, each pixel generates an event after a time interval which depends on that pixel's intensity, so the information is coded in the time spent between the global reset and the first event. More active pixels generate their event first. The main drawback is the global reset, which introduces some kind of "frame" like the frame-based systems.
3. Rank-order coding. Like the previous scheme, after a global reset, each pixel generates an event after a time interval given by its intensity level. However, the information is not coded in the event timing, but in the order. The events generated by all the pixels are put in order depending on the instant when they were produced. Again, the main drawback is the global reset, which resembles the concept of frame.
4. Intensity change coding. Each pixel generates an event if its intensity has changed above a certain threshold from the previous event.

2.4. Address Event Representation protocol (AER)

One of the main reasons for the great efficiency in the brain for visual processing is due to the massive interconnection between neurons. Each neuron is connected to thousands of neurons, so an event generated by one of them can be the input of a lot of neurons. However, for electronic systems, this interconnectivity could be an important limitation.

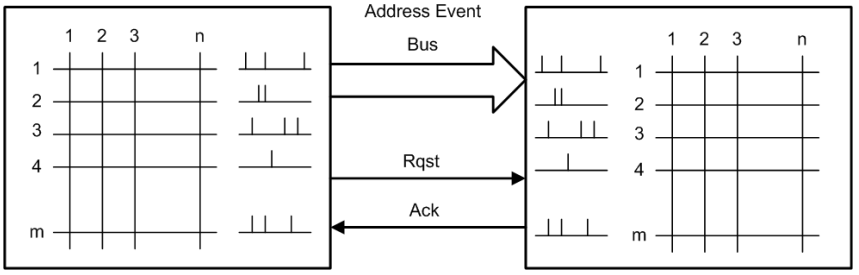


FIGURA 2.5. AER communication scheme between two populations of neurons.

In general, for artificial neurons, we can consider two different situations: we want to connect two populations of neurons inside a single integrated circuit, or we want to connect two populations of neurons integrated in different chips. With present day technologies, thousands of neurons can be integrated on a single chip (or event millions, depending on the neuron complexity). However, the number of metal layers to implement connections is not higher than 7 or 8, which makes it impossible for the neurons to communicate. On the other hand, if we consider populations of neurons in different chips, the number of pins available in each package is limited to a maximum of some hundreds, so it is impossible for all the integrated neurons to communicate directly with something outside of the chip. Nevertheless, there is an important advantage in the integrated circuits over the biological systems: the speed. While biological neurons generate spikes separated by milliseconds, electronic circuits produce spikes with a characteristic time of nanoseconds. This capability allows for multiplexing the outputs from many neurons into a single physic connection, using the AER protocol.

AER protocol was first proposed in Caltech in 1991 by Sivilotti [10] and in 1992 by Mahowald [11]. It allows for massive interconnection point-to-point between two populations of neurons, as is shown in Fig. 2.5. We can see two arrays of $m \times n$ neurons connected through a single digital bus with $\log_2(m \times n)$ bits and two more handshaking signals to implement the asynchronous communication. This way, each neuron inside the emitter system will produce events at a very reduced rate like biological neurons (in

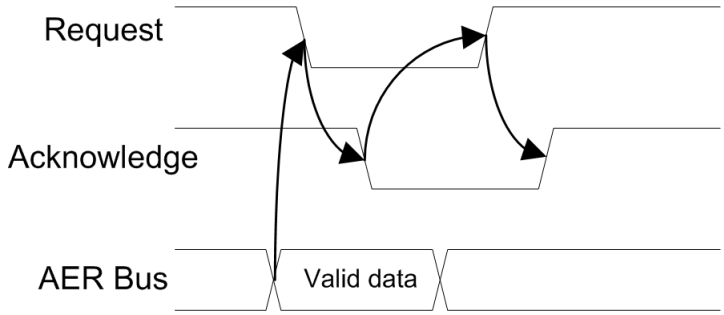


FIGURA 2.6. Timing diagram of AER protocol

the order of milliseconds), but the digital bus common for the $(m \times n)$ neurons handles much higher events rates. Each event includes the address of the emitter neuron, so that the receiver system will be able to reproduce the state of all the neurons in the emitter system in real time.

AER asynchronous communication follows a four-phase handshaking protocol, like shown in Fig. 2.6. Once the valid data is available in the digital bus, the emitter chip activates the Request signal, so the receiver chip stores the data and activates the Acknowledge signal. To finish the communication, the emitter deactivates Request and the receiver deactivates Acknowledge.

Using this AER protocol, a great number of applications can be developed with different ways of coding information on events. AER has been used for vision sensing and processing systems, for light intensity to frequency transformations [6], time-to-first-spike coding [12]-[15], foveated or spatial contrast sensors [16]-[18], or motion sensing and computations systems [19]-[23]. AER has also been used for auditory systems [24]-[27], competition and WTA (winner-takes-all) networks [28]-[30], or even for systems distributed over wireless networks [31]. Some generic systems like [32] process AER events for any coding scheme used, being applied for events processing on many bio-inspired architectures.

AER protocol sends events generated by different neurons through a common digital bus, so collisions can happen. When there is a collision between two events generated at the same time by different neurons, an

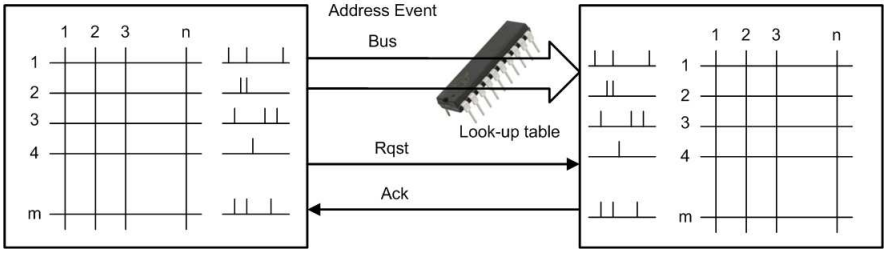


FIGURA 2.7. Image transformation in real time with AER protocol.

AER system can discard the events [33]-[35], or arbitrate between them [36]-[43]. Systems that discard colliding events are faster (as no extra processing is added to the event transmission), and do not modify the events timing (as they are transmitted as soon as they are generated), but some events are lost. However, systems that arbitrate between colliding events assure that no event will be lost, but they slow down the communication by introducing an arbiter stage, and modify slightly the events timing. Each application must indicate what option is better. For instance, it we have a very high event rate, the probability of collision increases, so the number of lost events in an unarbitrated system will be high. However, the importance of this event loss in the global behavior must be analyzed to choose the best alternative.

2.4.1. Advantages of AER

AER protocol presents several advantages for its application on multi-chip vision processing systems. First of all, we can implement simple image transformations as they are being transmitted between two chips, as shown in Fig. 2.7. By inserting a PROM memory in the AER bus, a look-up table can be programmed to transform each event address, implementing rotations and translations in real time. S

Some advantages of AER are related to the capability of implementing multi-sender and multi-receiver systems, with many chips sharing the digital bus, which is very useful for multi-layer systems.

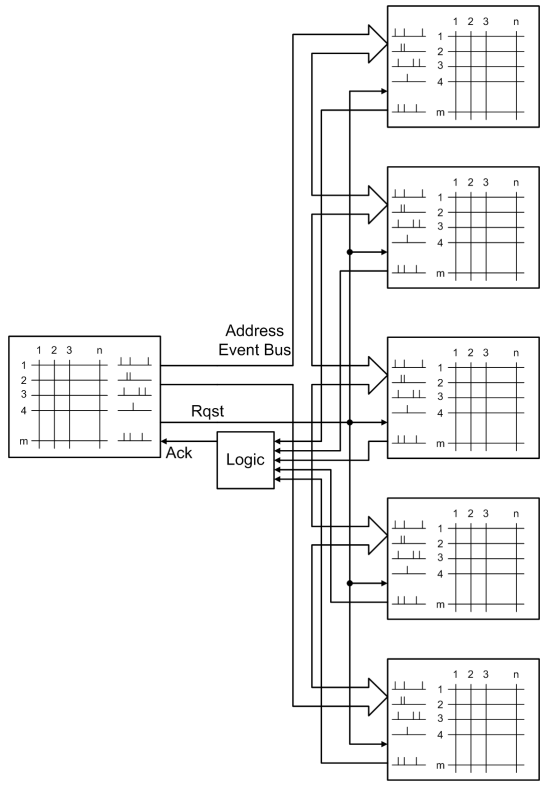


FIGURA 2.8. Architecture of a multi-receiver AER system.

Fig. 2.8 shows the architecture of a multi-receiver system, with several receiver chips share the AER bus. If no extra circuitry was added, the sender would remove the data from the bus once the faster receiver had acknowledged, so the other receivers would latch a wrong data. To avoid this problem, the sender must wait until the slowest receiver has acknowledged to deactivate the Request signal. Also, we must be sure that the sender will not send another Request until all the receivers have deactivated their Acknowledge signals. That is why a logic operation must be added for all the Acknowledge signals before being sent to the sender, like illustrated in Fig. 2.8. The logic operation is a C-element block, so the output will not change while the inputs have different values, and the sender will only see a change when all the receivers agree.

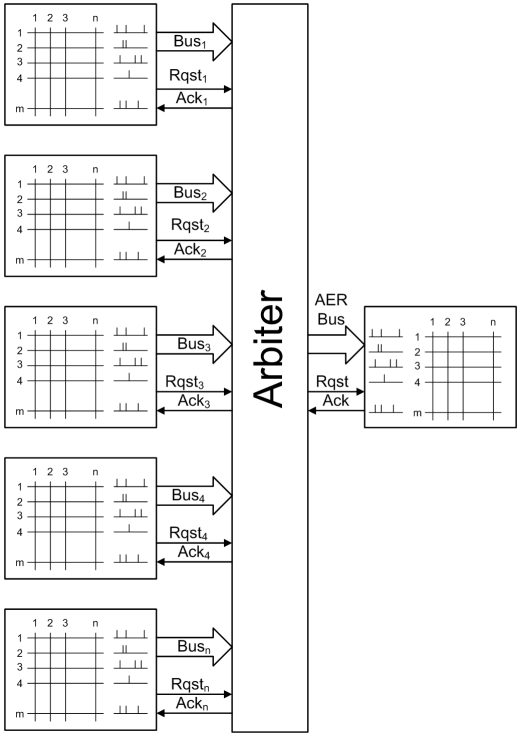


FIGURA 2.9. Architecture of a multi-sender AER system.

When there are several senders sharing a common bus, it is not so easy. If all the senders were allowed to write in the bus, collisions would be produced, so some kind of arbitration must be added. In [44] two different alternatives are proposed. The first one, shown in Fig. 2.9, consists on adding an external arbitration circuit, which controls the access to the bus. When a chip wants to send an event, it activates its Request signal $Rqst_i$ ($i = 1, \ldots n$), and when the arbiter acknowledges it copies the event address in the common bus, activating Rqst signal for the receiver. Once the receiver answers, the arbiter is ready to send another event generated by any sender.

The other alternative consists on modifying slightly the AER protocol so that all the sender can be connected in a tree-like structure, so that the

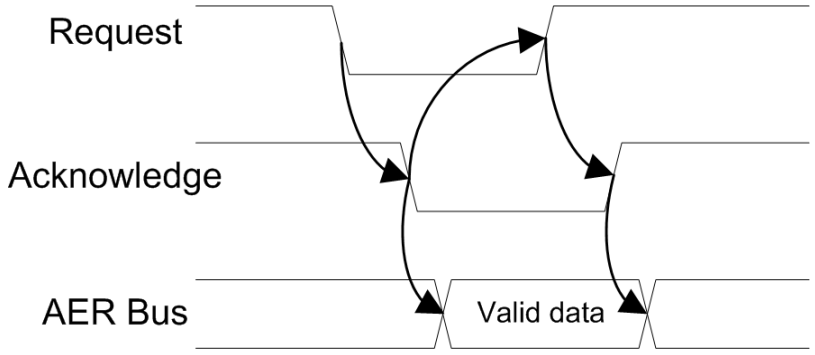


FIGURA 2.10. Timing diagram of AER protocol modified for multi-sender systems.

arbitration can be implemented by the senders internally. This way, all the AER buses can be connected, but some complexity must be added to the senders.

With these ideas, several alternatives could be developed. One of them consists on avoiding any external component modifying the AER protocol as shown in Fig. 2.10. This way, when a sender chip activates the Request signal the valid data is not ready in the AER bus. Only when the receiver acknowledges it has permission to write on the common bus. Thus, in this multi-sender scheme the Acknowledge signal is sent to only one sender, so the receiver must generate as many Ack signals as senders in the system. In [45], [46] SCX (Silicon Cortex) is proposed as a configurable AER communication infrastructure that can be used to test inter-chip communication in neuromorphic systems with different connectivities.

Another alternative could be to add an internal arbiter to each AER chip to connect all of them in cascade. Hence, when the first chip wants to send an event, the Request must be processed by the whole chain, and only if all the senders agree, the event can be sent. The main advantage is that not all the senders have the same priority, as it depends on their position inside the chain. This asymmetry could be compensated by including a mechanism that makes each chip wait in case of collisions a time proportional to its position inside the chain.

In this work, some auxiliary boards have been used to implement multi-sender or multi-receiver systems. These boards follow the schemes described in Fig. 2.8 and Fig. 2.9. With this approach, each link is handled point-to-point [47], [48].

CAPÍTULO 3

Multilayer convolution systems

3.1. Introduction

Biological vision systems work with events, as was described in previous chapter, and they present a highly interconnected multilayer structure. Visual information (sensorial information, in general) is captured by the retina and sent to the brain cortex as spikes, where it is processed. The cortex is formed by several layers of neurons, each one of them with a large number of neurons massively interconnected. Each neuron is connected to many neurons in the next layer, so the events it generates are received by a projection field.

Thus, information captured in a specific region of the retina propagates very fast through the different layers of the cortex, which detects the shape of the objects in front of the retina. Each connection between a neuron and its projection field has specific weights, which are different for each neuron. However, in the first layers of the cortex these weights are quite similar for neurons belonging to the same layer. This is why the whole system can be approximated by a multilayer convolutional operation. The aim of this work is to develop convolutional blocks that can be interconnected to build a mul-

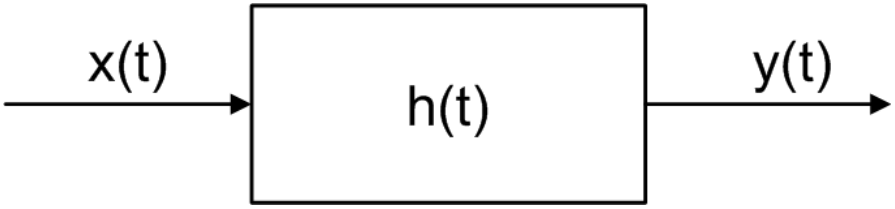


FIGURA 3.1. Representation of a linear time-invariant system (LTI).

tilayer system, and emulate the behavior of biological vision processing systems.

Convolutional Neural Networks use this multilayer structure to perform different applications [49], like character recognition for vision systems [50], [51], phonemes [52], [53] or spoken words recognition [54] for acoustic systems. These systems are based on interconnected convolutional layers with configurable weights.

Section 3.2 describes the basic concept of convolution operation, and how to implement it with an AER-based system. Section 3.3 presents bio-inspired systems for multilayer vision processing, with special emphasis on a software tool for simulation of these structures. Finally, Section 3.4 describes the architecture of the proposed convolution chip.

3.2. 2-D Convolution

3.2.1. The mathematical operation

For a lineal and time-invariant system (LTI) like shown in Fig. 3.1 with impulse response $h(t)$, being $x(t)$ the time dependent input signal, the output signal $y(t)$ can be expressed as:

$$y(t) = \int_{-\infty}^{\infty} h(\tau)x(t-\tau)d\tau \qquad \textbf{(EQ 3.1)}$$

This operation is called convolution of $x(t)$ and $h(t)$, and is expressed as $y(t) = x(t) \otimes h(t)$.

If we are working with signals discrete in time, the integral becomes a summatory, and the convolution is expressed as [55]:

$$y(n) = x(n) \otimes h(n) = \sum_{m=-\infty}^{\infty} h(m)x(n-m) \quad \textbf{(EQ 3.2)}$$

If the signals are not only discrete in time, but also finite, the result of the convolution will also be a finite signal, expressed by the equation (3.3):

$$x(n) \otimes h(n) = \sum_{m=0}^{n} h(m)x(n-m) \quad \textbf{(EQ 3.3)}$$

Thus, for an input signal $x(n)$ with N_x values (it has non-zero values for $0 \le n \le N_x - 1$), and an impulse response $h(n)$ with N_h values (it has non-zero values for $0 \le n \le N_h - 1$), the convolution $y(n)$ will have $N_y = N_x + N_h - 1$ non-zero values, as is illustrated in the example of Fig. 3.2.

Once described the 1-D convolution, it is easy to extend it for 2 dimensions, considering signals depending of 2 variables (x,y) [56]. The bidimensional convolution is applied for images, considering that variables x and y correspond to the horizontal and vertical coordinates. We call $I(x,y)$ the input image, and $O(x,y)$ the output signal. Being $K(x,y)$ the impulse response (also called the convolution kernel), we obtain the expression:

$$O(x,y) = I(x,y) \otimes K(x,y) = \sum_{m}\sum_{n} K(m,n)I(x-m, y-n) \quad \textbf{(EQ 3.4)}$$

This expression presents a horizontal inversion in the kernel positions with respect to the input image for calculating the convolution. However, we omit that inversion for implementing the operation, as we can simply

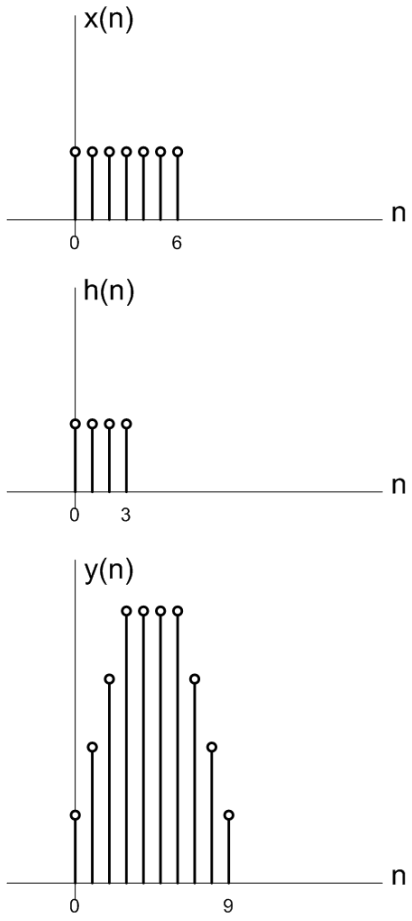


FIGURA 3.2. Example of a 1-D convolution with an input sequence $x(n)$ with $N_x = 7$ and an impulse response $h(n)$ with $N_h = 4$, obtaining an output $y(n)$ with $N_y = N_x + N_h - 1 = 10$.

define the kernel properly to make the expression equivalent. Also, for many applications the kernels are symmetric, so expression (3.4) is equivalent anyway.

The meaning of this expression is illustrated in Fig. 3.3, where there is an input image with $X \times Y$ pixels and a kernel sized $M \times N$. The output

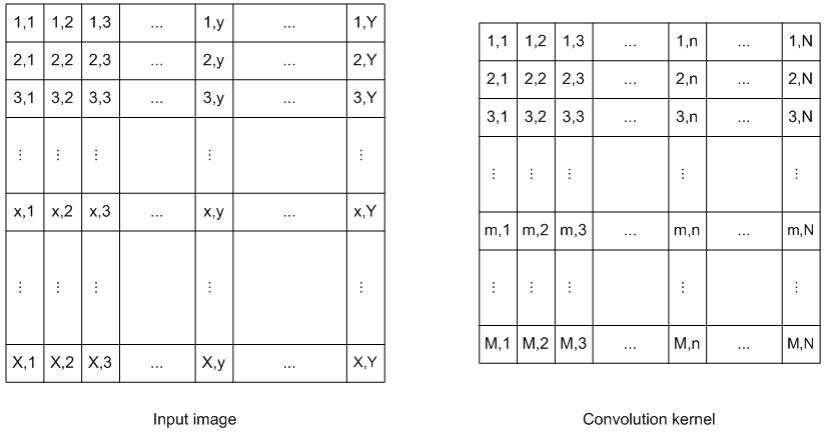


FIGURA 3.3. Description of a 2-D convolution between an input image with (X,Y) pixels and an (M,N) kernel.

image that represents the convolution is obtained applying the kernel centered around each input pixel, and calculating the weighted addition of the pixels in the neighborhood with the kernel weights. The example in Fig. 3.4 shows an input image with 5×5 pixels and a 3×3 kernel. The output image shows the result obtained for a single pixel $(4, 2)$. The weighted addition is calculated as:

$$O(4, 2) = 1 \cdot 0 + 2 \cdot 2 + 3 \cdot -1 + 1 \cdot 1 + 2 \cdot 3 + 3 \cdot 0 + 1 \cdot -2 + 2 \cdot 0 + 3 \cdot 1 = 9$$

This way, the output value for each pixel is obtained centering the kernel around its position in the input image, and calculating the equivalent operation.

3.2.2. AER-based convolution

Once described the mathematical convolution, this operation must be implemented with an AER-based system, going back to the event-based image processing structures described in previous chapter.

With traditional frame-based systems, each image is represented by a matrix, and the convolution operation can be calculated applying the equa-

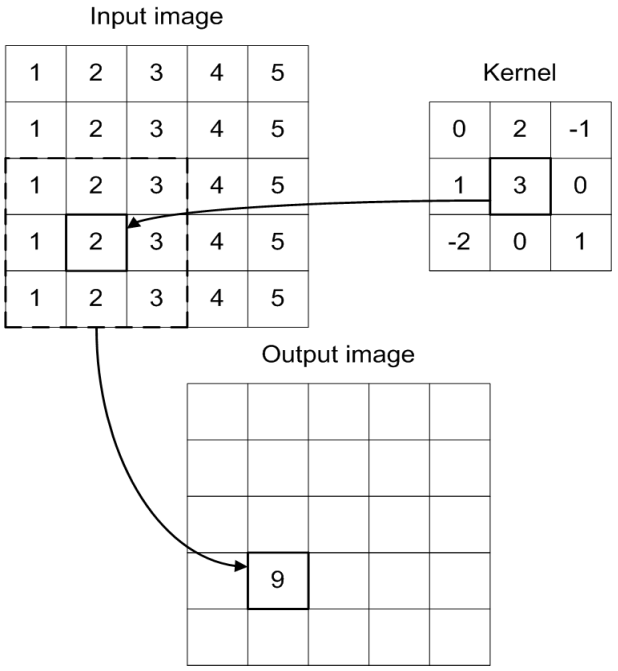


FIGURA 3.4. Example of how to calculate the convolutional operation for a single pixel. The rest of the pixels are calculated equivalently.

tion (3.4). However, in event-based systems there is no input image, as the events are produced in real time representing somehow the state of the pixels. Fig. 3.5 shows how the convolution is implemented. When an input event is received with an address (x, y), the operation affects the whole neighborhood around the pixel indicated by this address. Then, the kernel is applied to the neighborhood, making each pixel update its state with the corresponding weight, so that the pixel array includes the state of the convolution operation.

Nevertheless, the convolution result must also be coded in events. That is why the convolution pixels are implemente with Integrate&Fire neurons that update their state until reaching a specific threshold, when they generate an output event. Then, an output flow of events is produced, which corresponds to the result of the convolution. AER convolutions can be

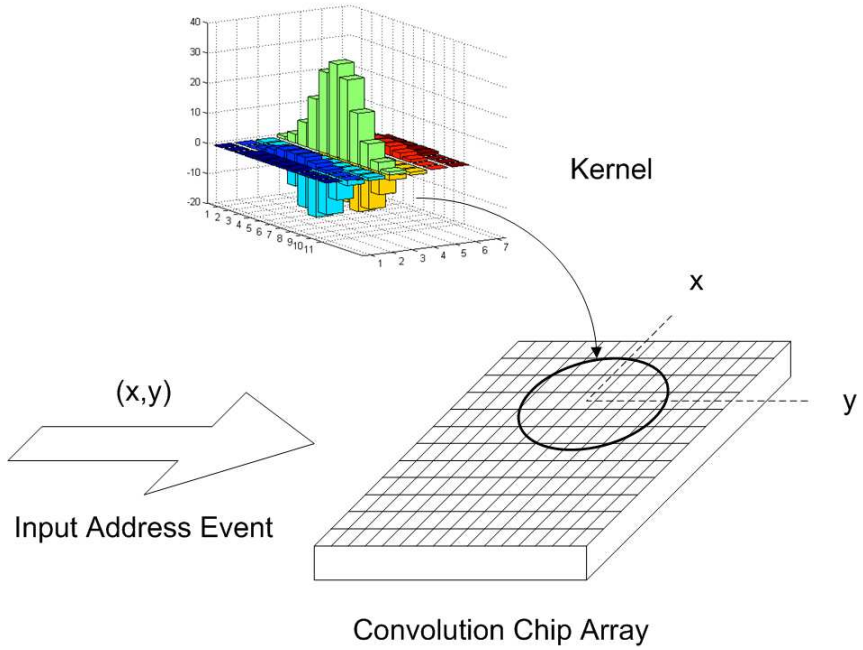


FIGURA 3.5. Implementation of a 2-D convolution with an AER event-based system.

cascaded, or they can be connected in parallel, building complex multilayer systems.

3.3. Bio-inspired multilayer systems

3.3.1. Biological inspiration

From a biological point of view, vision is performed in a region inside the brain called visual cortex. This cortex is structured as several layers (between 8 or 10 for humans [57], [5]), receiving events from the retina (which also performs some preprocessing of the captured information), and is specialized for processing objects information (both static or moving) and for pattern recognition. There is a massive interconnection among the neu-

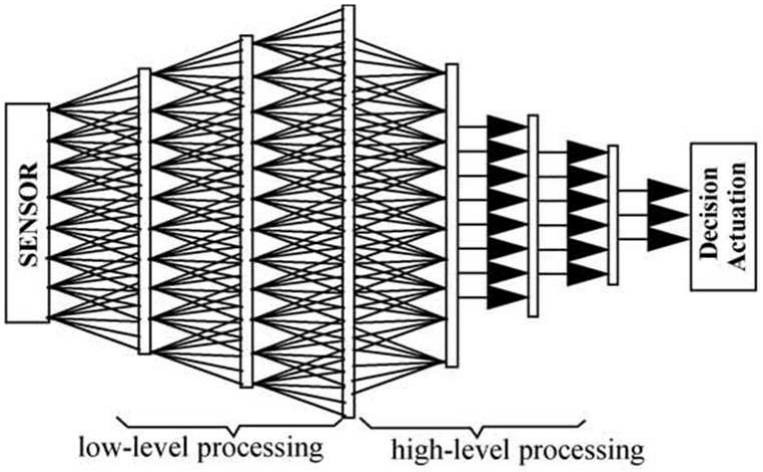


FIGURA 3.6. Structure of a bio-inspired multilayer system.

rons in the different layers of the visual cortex, with an estimated number of 10^{11} neurons in the whole human brain [58]. However, these connections do not follow a random pattern, as each neuron inside one layer is connected to a set of neurons inside the next layer. This set of neurons in the next layer is called projection field [59].

The concept of projection field implies that when a neuron generates an event, this event is received by a set of neurons in next layer, so a relation can be established between this concept and the 2-D AER convolution. In the brain, each connection between a neuron and the neurons in the projection field has a different weight, so the event does not contribute the same way to all the neurons. This effect is implemented by the kernel values.

The structure of the visual cortex is shown in Fig. 3.6. The sensor in the left corresponds to a retina, and the neurons in each layer are connected to a projection field in each layer. If we consider that each layer is performing a convolution (or many in parallel), an artificial multilayer network of convolution systems can implement complex tasks like object recognition, emulating the behavior of the brain cortex.

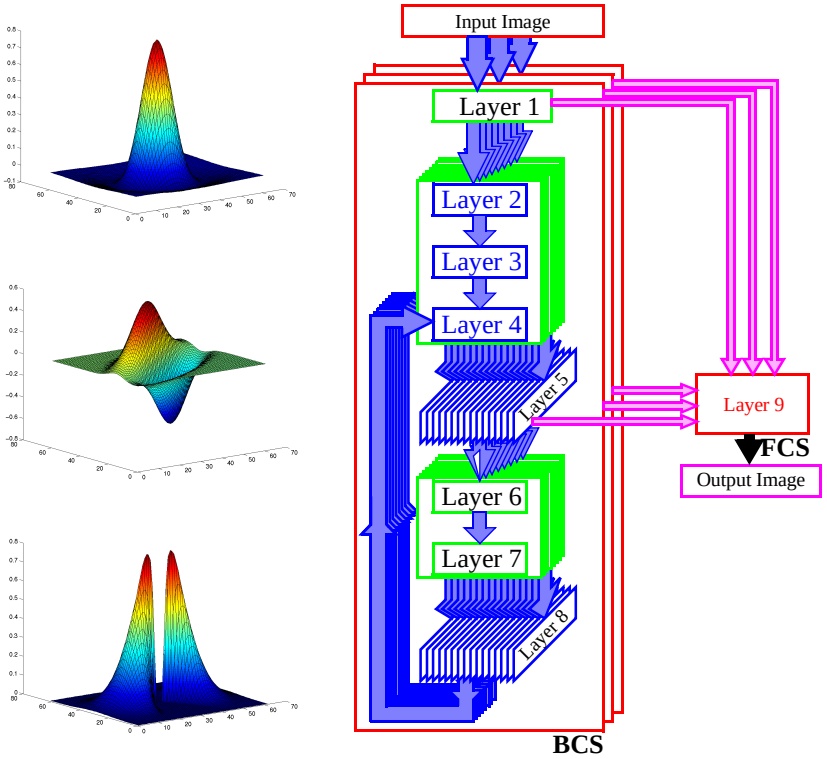


FIGURA 3.7. Block diagram of the BCS-FCS system.

One example of artificial vision model based on the structure of the brain cortex is the BCS-FCS algorithm (Boundary Contour System- Feature Contour System) for image segmentation [60]-[64]. This model includes 9 layers, and extracts edges for different orientations and scales, so that the contours of the objects in the input image can be identified.

The block diagram that implements this algorithm is shown in Fig. 3.7. The BCS system consists of several identical subsystems, each of which is tuned for a different spatial scale. In the example of the figure we have three subsystems. Each BCS spatial subsystem consists of 8 layers, with consecutive layers connected by thick shaded arrows. These connections represent a convolution (or filter) operation applied to the state of the previous layer

and resulting in the state of the next layer. For instance, the 2-D input image suffers three different filtering operations, each of which is the starting point of a BCS subsystem, which operates autonomously.

Layers 1, 2 and 3 implement only feedforward filtering operations, while layers 4 to 8 are connected in a feedback loop configuration, which means the system will reach a steady state after a certain number of iterations (if the system is implemented sequentially on a computer) or after a certain time constant (if the system operates asynchronously and fully parallel, like in biological brains). The outputs of layers 1 and 5 of the three BCS subsystems are fed to the FCS. Now we will briefly describe the behavior of the different layers.

Layer 1 applies an "on-center off-surround" kernel (like the one shown on the top of Fig. 3.7) to the input image, with positive weights for the center region of the kernel and negative weights for further pixels. The result of such convolution is local illumination normalization and contrast enhancement. This filtering is applied by the three layers 1 in parallel to the same input image, with different widths for the positive region of the kernel.

Layer 2 applies several convolutions in parallel, as many as different orientations we want to detect. This is performed by edge-detection kernels like the one represented on the middle of Fig. 3.7, each one of them rotated a certain angle to detect the specific orientations. If the input image to this layer presents a positive change in contrast with respect to the indicated orientation axis, a high positive output value will be generated. A negative change in contrast will produce a negative output value. Then, layer 3 rectifies the output of the previous stage, as it is not necessary to distinguish between positive and negative values.

Another filtering like the one implemented in layer 1 is applied in layer 4, in order to enhance the contrast of the image at this point. Once obtained the images from layer 4 for all possible orientations, they are filtered with 1-D "on-center off-surround" kernels so that the contrast is improved for orientations with higher pixel values.

Layer 6 applies a set of filters with bipolar kernels like the one represented on the bottom of Fig. 3.7, in order to identify contours, which can be

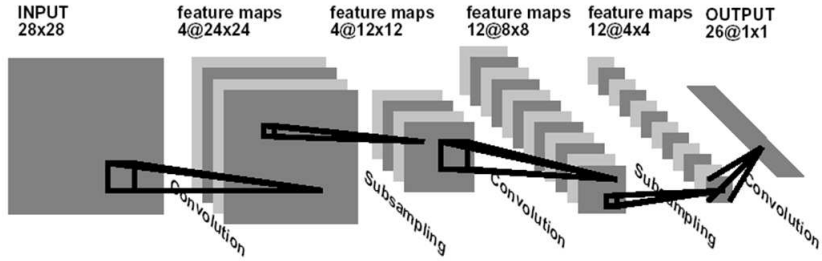


FIGURA 3.8. Convolutional Neural Network proposed by Y. LeCun.

defined as edges that remain consistent over larger space ranges. After that, layers 7 and 8 performs the same operations as layers 4 and 5, respectively. The output of layer 8 is combined with the output of layer 3 to form the input image of layer4, making a feedback loop.

Finally, layer 9 receives the contours information generated by layer 5 (once the feedback loop has settled) for all computed orientations, and with the original image produced by layer 1 performs a selective diffusion operations between pixels to obtain a clean noise-free image with clear and consistent long range contours.

This kind of multilayer algorithms for image processing has been traditionally limited by the concept of frame, that made it difficult its real-time implementation. Each layer in Fig. 3.7 performs several parallel convolutions, with a large computational load. For this reason, if we want to obtain a real-time processing with this model, it is fundamental to adapt it to event-based AER vision systems [65].

As was described in the previous chapter, in event-based processing systems the more relevant pixels send spikes that propagate through all the layers in a very short time (in the range of microseconds), performing very fast recognitions in multilayer systems [66], [8].

However, most of the multilayer convolution systems developed are frame-based, like the Convolutional Neural Networks. With these networks, many applications have been proposed for character recognition [67]-[69],

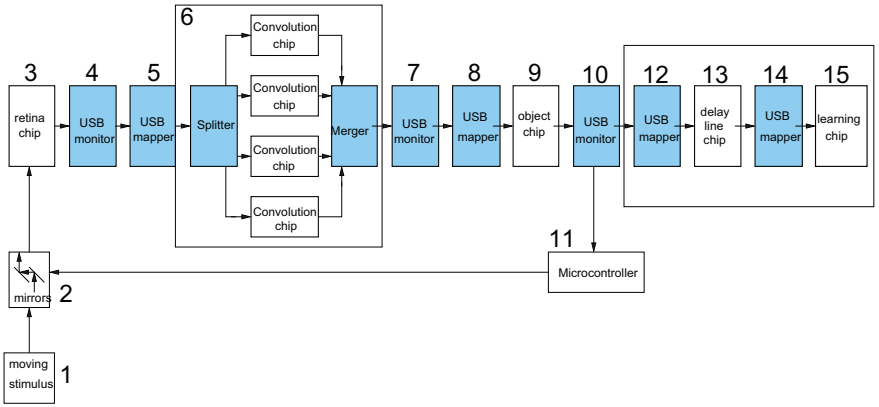


FIGURA 3.9. Block diagram of the experimental setup developed by european project CAVIAR for an AER vision system for objects tracking.

[71], objects detection [69], or face recognition [69]-[71]. Fig. 3.8 shows the multilayer architecture proposed by LeCun [71], with a large number of convolutions.

This example can be very useful to explain one of the main functionalities presented in one of the convolution chips designed for this thesis, the one called Conv2. Fig. 3.8 shows how each feature map in a certain layer applies a different kernel for each projection field from the previous layer. For our systems, this would correspond to apply a different kernel depending on the origin of each event, and we call it multikernel. Next chapters describe how it is implemented.

European project CAVIAR was developed to translate the experience with frame-based multilayer processing systems into the more efficient even-based processing. An AER architecture was proposed with a set of building blocks (some of them were convolution chips [32]), with the capability of implementing many applications for image processing and objects recognition and tracking [72], like the one shown in Fig. 3.9, described in the following paragraphs.

A mechanical rotor (1 in Fig. 3.9) holds a rotating piece of paper with two circles of different radius and some distracting geometric figures. The vision system follows the circles only, and discriminates between them. A pair of servomotor driven mirrors (2) changes the point of view of the AER retina (3), which sends outputs events to a monitor PCB (4) and to a mapper PCB (5) before reaching the convolution PCB (6) with four convolution chips. The outputs of these convolution chips are sent through another monitor PCB (7) and a mapper PCB (8) to the 2-D WTA (winner-takes-all) object chip (9), which sends its output to a monitor PCB (10) that sends them to a microcontroller (11) that controls the mirrors (2) to center the detected circle, and also to the learning system. The learning system consists of a mapper PCB (12), a delay line chip (13), another mapper (14) and a learning classifier chip (15), and learns to classify trajectories into different classes.

The temporal contrast retina provides an output event space of size 128×128 pixels. Each convolution chip has a 32×32 resolution, but the convolution PCB with four chips can process the whole 128×128 space, although it would compute the convolution output for only the central 64×64 pixels. To overcome this problem, a mapping PCB (5) is introduced between the retina and the convolution PCB to downsample from 128×128 to 64×64 pixels. This way, the convolution PCB will provide outputs for the complete retina visual space. The convolution chips use circular kernels with specific diameters to detect the position of the center of circular shapes with these diameters.

The mapper (8) downsamples again to the 32×32 pixels at the WTA object chip input, which provides cleaned up coordinates of the center of the target-size detected circle. These coordinates are sent to the control subsystem and make the microcontroller acts on the two servomotors holding two mirrors. One of the mirrors is in vertical position and is controlled by the y-coordinate of the object chip output, while the other one is in horizontal position an is controlled by the x-coordinate. Thus, the coordinate provided by the object chip indicates the deviation of the detected circle from the center of the field of view. The microcontroller is programmed to zero this deviation, keeping the target-size circle centered on the field of view of the WTA.

The delay line chip converts the temporal information of the events generated by the object WTA chip into spatial information, with a certain spatial pattern. The learning chip implements competitive learning to classify spatial patterns. The patterns may be spatial patterns, formed by coincident spikes at different spatial locations, or activity patterns, formed by coincident average spiking activity at different locations.

3.3.2. Software implementation: applications

The aim of this thesis is the development of convolution chips for implementing modular systems with large number of chips, so it is important to be able to study previously how to build, configure, program and train this kind of systems. Once designed the AER convolution chips, to perform a specific application it will be necessary to find the optimum hierarchical structure, choose the best-suited convolution kernels, and any other parameters that must be set. The AER behavioral simulator developed by J. A. Pérez-Carrasco in Visual C++ [73] allows to make a behavioral description of real AER modules (convolution chips, in particular) and build complex systems with large number of modules. Then, we can obtain a realistic estimation of the results provided by multilayer systems before implementing them in hardware.

Many architectures have been simulated with this system, emulating the behavior of the brain. For instance, in [74] it is presented the behavioral simulation of a system for character recognition, which could discriminate between different characters in less than $10\mu s$, even though it includes up to 52 convolutions. This is a good example to show how fast AER multilayer systems are. Fig. 3.10 shows the multilayer architecture implemented by the simulator. As is indicated in the figure, it can discriminate between 7 different hand-written characters. It has 7 outputs, so only one of them will produce events, indicating the result of the recognition processing.

Some others interesting results have been obtained implementing behavioral simulation models for texture recognition, like described in [75], [76]. Fig. 3.11 shows the proposed architecture for this example, where the

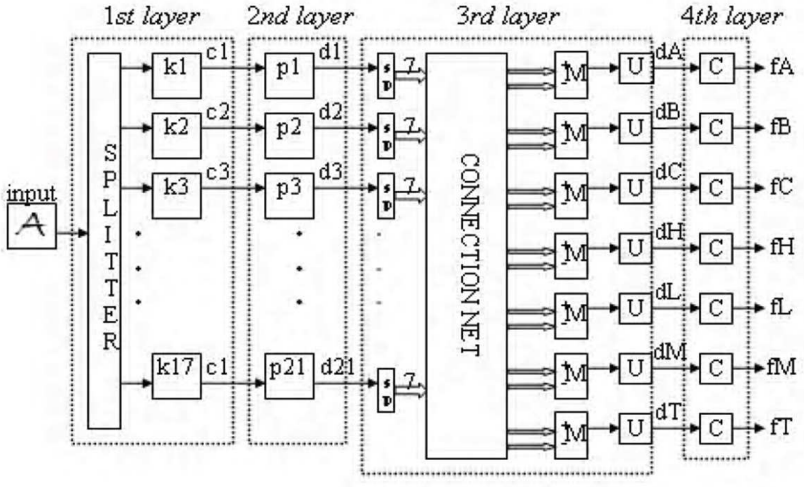


FIGURA 3.10. Structure of the AER system implemented with the behavioral simulator for character recognition.

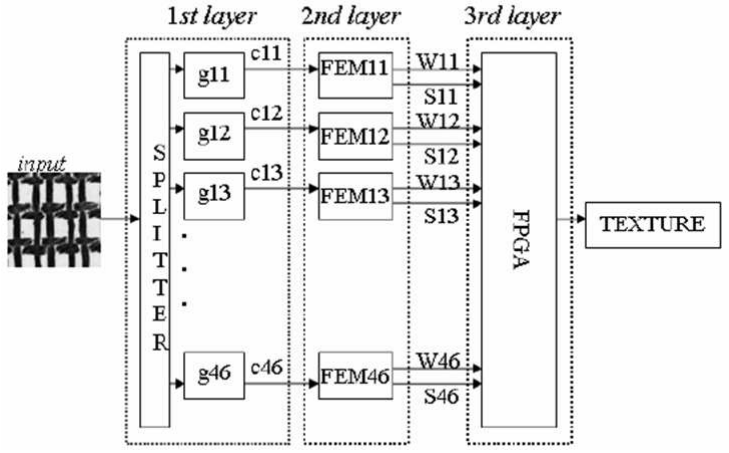


FIGURA 3.11. Structure of the AER system implemented with the behavioral simulator for textures recognition.

first layer includes 24 convolution modules in parallel, implementing a bank of Gabor filters with 4 scales and 6 different orientations.

3.4. The AER convolution chip

Once described the motivation of this work, it is the moment to start describing the AER convolution chip.

The convolution chip is the main module to implement complex neural systems with a configurable multichip architecture. In [77] an architecture is proposed to implement 2-D AER-based convolution chips in real time, with some limitations for the allowed kernels, as they must be decomposable in (x, y) coordinates (a kernel given by the expression $F(x, y) = H(x)V(y)$).

In [78] a convolution chip based on analog integrators is proposed. Its main limitations are:

1. the need for calibration to compensate mismatch between transistors,
2. a reduced 3-bit resolution (even after calibration), due to the low current transistor operation,
3. a high latency (around $1ms$) due to delays in analog components in the pixels, biased for low consumption.

These limitations have been overcome in the chips developed for this thesis. Digital pixels have been used, so there is no need for calibration (which reduces area and power consumption) and the precision is given by the size of the registers implemented in the pixels. Also, as there are no analog components, the operation is much faster, reaching event latencies as low as $150ns$ [79].

In this thesis, two different convolution chips are proposed, although both of the are based on fully digital pixels, and they can process arbitrary sized and shaped kernels, up to 32×32.

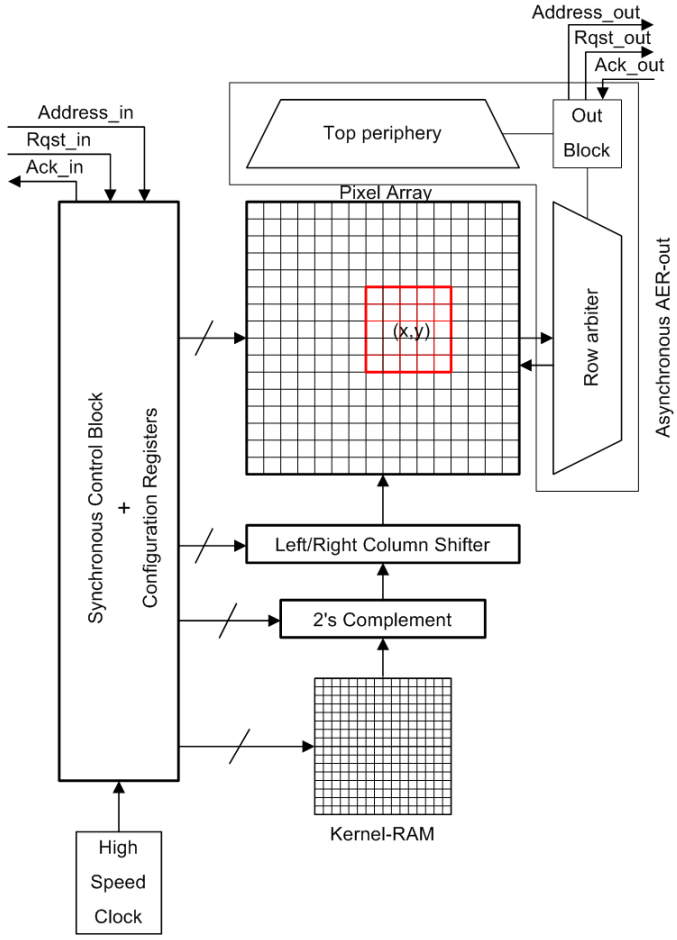


FIGURA 3.12. Architecture of both convolution chips Conv1 and Conv2.

3.4.1. Architecture

Considering a block diagram point of view, both chips (that we call Conv1 and Conv2) follow the architecture shown in Fig. 3.12. In both chips, AER input events arrive (Address_in bus, and the asynchronous protocol signals Rqst_in and Ack_in) which represent visual information pro-

duced by the previous layer, and the chip generates AER output events (Address_out bus, and the asynchronous protocol signals Rqst_out and Ack_out), which represent the result of the convolution. The blocks in Fig. 3.12 (which are described in detail in the following chapters) are:

1. Pixel array, of size 32×32 for Conv1 and 64×64 for Conv2, increasing four time the spatial resolution.
2. Static RAM integrated in the chip, where the kernel is stored in 2's complement representation. In both chips the RAM size is 32×32 data, though Conv1 can only write a single kernel and Conv2 allows for up to 32 different kernels, implementing the multikernel system.
3. Synchronous controller, which sequences the necessary operations to process each input event, and also to handle the forgetting mechanism, that does not depend on the arrival of events.
4. High-speed clock generator, with controlled frequency, is used by the synchronous controller.
5. Configuration registers, which stores some parameters loaded serially at startup.
6. 2's complement inverter. It is a block for inverting the sign of the kernel before being added to the pixels when the input event is negative.
7. Horizontal shifter block, to center properly the kernel stored in the RAM around the pixel indicated by the input event's coordinates.
8. AER generator, an asynchronous block that arbitrates between events generated by the pixels, and sends them to the next layer of the multichip system.

The chip works as follows: when the synchronous controller detects a falling edge in signal Rqst_in, the event's coordinates (x, y) are latched from the Address_in bus, and the asynchronous handshaking is completed. Then, the controller uses the information relative to the kernel size (which has been previously written in the configuration registers) to calculate the limits of the projection field associated to the event address. Three different situations are possible:

1. the projection field is completely inside the pixels array,
2. it is partially inside the array,

3. it is completely outside of the array.

If we are in case 3, the controller discards the event and waits for the next one. However, in any other situation, the controller calculates the left/right horizontal shift between the RAM columns where the kernel is stored, and the columns of the projection field in the pixel array. Next, it enables the adding row by row of the kernel values into the corresponding pixels. This way, after processing an input event the pixels inside the projection field are updated. If any of them reaches the programmed threshold, it resets its state and generate an event, that is arbitrated by the asynchronous AER generator, and then is sent off-chip with the corresponding handshaking. In parallel, there is a global forgetting mechanism which is common for all the pixels.

The asynchronous AER generator reads the events following the row-parallel technique [41]. Thus, the events are arbitrated by rows (for a single row, all the request signals implement a wired-or). Once the row arbiter answers, all the events generated by that row are latched at the upper periphery, so the row arbiter is free. Then, it can answer a new request while all the events from the previous row are emitted in burst mode.

In general, as convolution kernels can have either positive or negative values, output events must be signed. In a multilayer system, the convolution operations can be cascaded, so a convolution chip must be able to handle signed input events and produce signed output events. That is why the convolution chips developed in this work include a sign bit both in the input AER bus and in the output one, and also in the kernel values stored in the RAM (in 2's complement representation). The pixels must be able to perform signed addition, and produce positive or negative events. When a negative input event is being processed, the controller enables the 2's complement inverter to change the sign of the kernel values before being added to the pixels.

Concerning the forgetting mechanism, it is handled by the synchronous controller. The aim of this mechanism is that the values stored in the pixels (in absolute value) are decremented at a programmable rate, so that they can forget there state after some controlled time.

3.4.2. Proposed multichip structure

The aim of this work is not only to design two convolution chips Conv1 and Conv2, but also they must be designed to build complex multichip structures. First, we describe the tile configuration, with arrays of convolution chips, and the multikernel system, to finish describing the general proposed multichip structure.

1. Tile configuration.

Both convolution chips Conv1 and Conv2 have 14 bits in the address input AER bus, 7 bits for each coordinate, so their visual address space is 128×128. However, the dimensions of the pixel arrays are 32×32 for Conv1 and 64×64 for Conv2. Hence, the coordinates of both chips can be configured, so that several chips can be connected in parallel to emulate the behavior of a single ship of larger resolution. For Conv1, we could tile 4×4 chips together to build an equivalent 128×128 array, while for Conv2 it would be enough tiling 2×2 chips.

Fig. 3.13 shows an example of a tile configuration, with 4×4 convolution chips, each one of them with 32×32 pixels, building a large array with 128×128 pixels. When an input event arrives to this system, it is possible that the projection field is divided in 4 different chips. Then, each one of the 16 chips computes the coordinates of the event with its own coordinates, and checks if some part of the projection field corresponds to its array of pixels. If the answer is negative, it discards the event, an if it is positive it adds the kernel into the corresponding pixels. This way, the system behaves like a single large array of pixels. To make this work properly, all the chips must have the whole kernel programmed in its own RAM, so the kernel size will still be 32×32, as for an individual chip.

2. Multikernel system.

The aim of the multikernel system is that we can program several kernels in a single convolution chip, so that each event will be processed with the corresponding kernel. This system has only been included in chip Conv2, allowing up to 32 different kernels to be written in the RAM. The

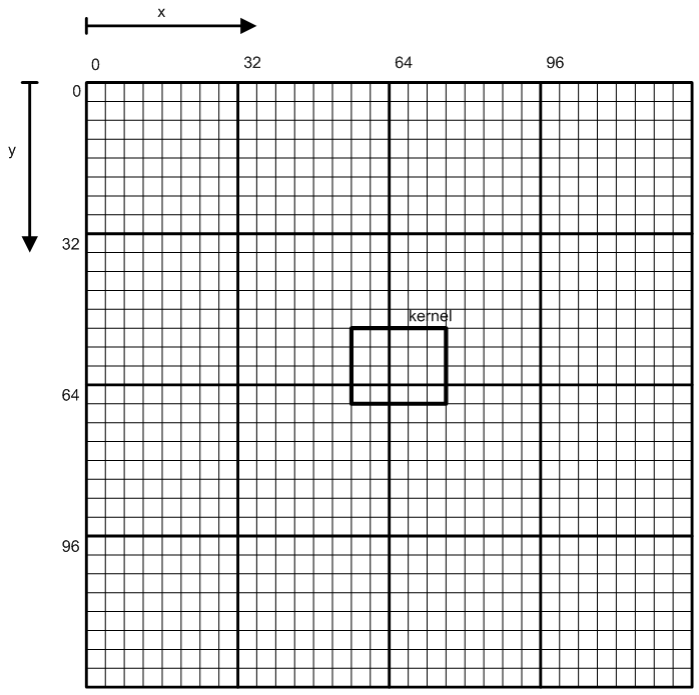


FIGURA 3.13. Tile configuration with several chips to process larger input images.

sizes of the kernels will be limited by the total size of the RAM, which is 32×32.

To implement this system, several modification had to be added, and everything is detailed in the following chapters, when all the blocks are described. First, the AER bus must include some extra bits (5 for the case of 32 possible kernels) so that each input event carries its coordinates (x, y) and also the kernel identifier k_i $(i = 0,\ldots, 31)$ to indicate which kernel must be used to process it. The aim of this system is to implement with a single chip the equivalent functionality of several different chips, reducing the number of elements in a multichip system. This is illustrated in Fig. 3.14.

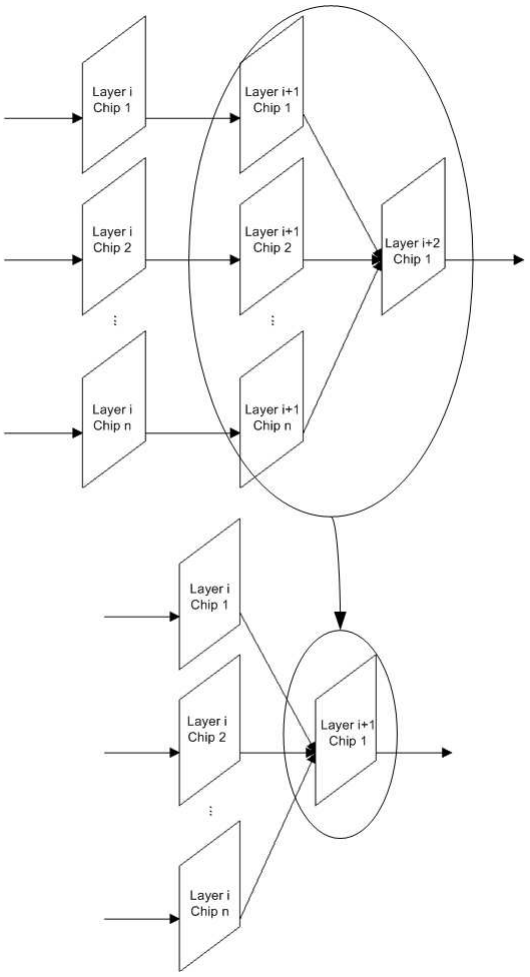


FIGURA 3.14. Description of the multikernel system: on the top, conventional system, and on the bottom, equivalent multikernel system.

On the top of Fig. 3.14 we can be a conventional multilayer system, with n convolution chips in parallel in layer i. The outputs of each one of the chips in layer i are connected to the inputs of the corresponding chips in layer $i + 1$. This structure is very common in neural networks, like the systems for character recognition [74] or texture recognition [75], [76] described in previous section. Finally, the n outputs from the convolution

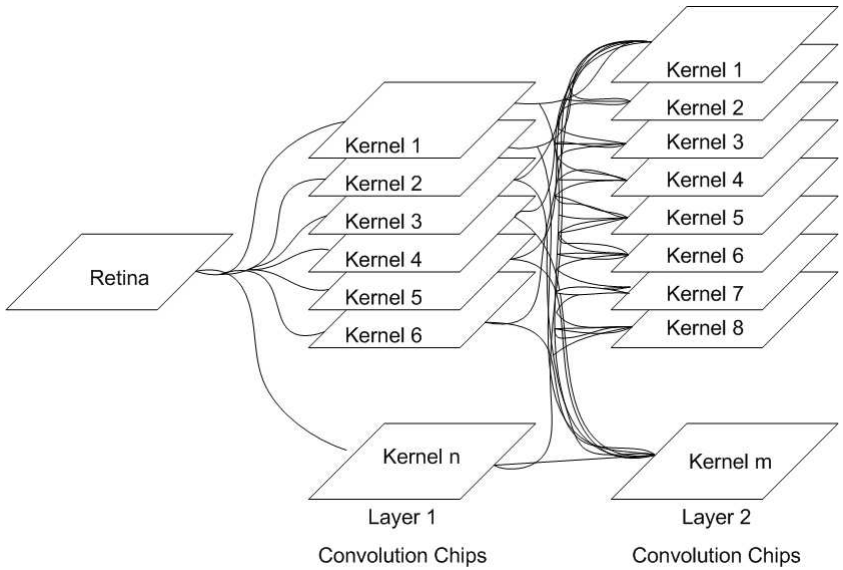


FIGURA 3.15. Multilayer system for vision processing.

chips in layer $i + 1$ are merged in a single chip in layer $i + 2$, which adds the contributions produced by all of them. This whole system is equivalent to the one shown on the bottom of Fig. 3.14. In this case, a single convolution chip Conv2 can replace the n chips from layer $i + 1$ and the chip in layer $i + 2$, making each event coming from layer i carry information about the chip that generated it, so that Conv2 can choose the corresponding kernel to process it.

3. General multichip structure

In general, for both specific configurations described in this section, the aim of the convolution chips Conv1 and Conv2 developed in this thesis is to build multilayer systems like the one shown in Fig. 3.15, with many processing layers.

These chips are designed to communicate with many of them building networks with as many convolution chips as we want, and as complex as possible. Then, the utility of these chips does not finish with the applica-

tions shown in this thesis, but they are a basic module to continue developing more and more complex processing systems.

CAPÍTULO 4

The convolution pixel

4.1. Introduction

In previous chapters, the behavior of the human brain for vision processing has been analyzed, describing how it is structured as a multilayer system for events processing. As the aim of this thesis is to develop convolution chips that emulate the brain behavior for vision processing, in this chapter we describe the basic processing unit for these chips: the convolution pixel.

First, the biological neuron is described as the basic processing unit in the brain. Section 4.3 illustrates the analog pixel proposed in a previous work, showing its behavior and some drawbacks. Finally, Section 4.4 describes the digital pixel proposed in this work, with special emphasis on the two different versions: the initial version integrated in the convolution chip Conv1 and the improved version integrated in Conv2.

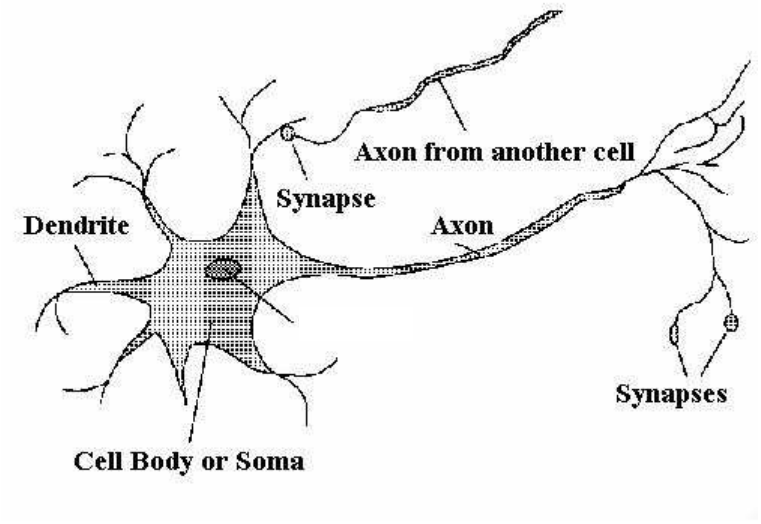


FIGURA 4.1. Basic structure of a neuron.

4.2. The biological neuron

For modelling and implementing an artificial neural network, it must be studied first the behavior of the biological nervous system, and specially its basic processing unit: the neuron. The brain is formed by a dense network of neurons, with more than 10^4 of the and several kilometers of connections per mm^3 [80], which makes a total estimation in the range of 10^{11} neurons in the human brain [82], with a number of connections around 10^{15} [83].

A common neuron is formed by three main parts, as is shown in Fig. 4.1: soma or neuronal body, axon and dendrites. In general, the signals coming from other neurons reach the dendrites. If the excitation level caused by an input is enough, an output signal is generated, which propagates through the axon and its ramifications to other neurons. The joint between a ramification of the axon and another neuron's dendrite is called synapse, and it is also shown in Fig. 4.1. It is common to refer to a sender neuron as pre-synaptic, and a receiver neuron as post-synaptic.

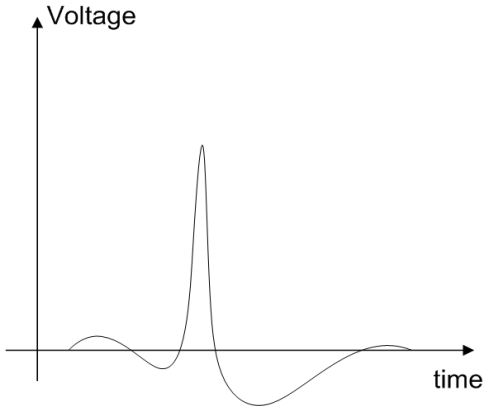


FIGURA 4.2. Appearance of an electric spike transmitted between neurons.

The process for transmitting signals between neurons is both chemical and electric, as the signal that propagates through the axon is an electric spike, but when it reaches the synapses there are some special substances called neurotransmitters that propagate.

The electric spike which propagates between neurons looks like the one shown in Fig. 4.2, with 1 or $2ms$ length. The generation of this spike is related to the composition of the neuron's membrane, which is semipermeable and achieves around $70mV$ of potential difference between both sides due to a different concentration of ions. The internal liquid has a potassium concentration 10 times larger than the external liquid outside the neuron. However, sodium concentration is 10 times larger outside. This is called resting potential.

When the neuron receives neurotransmitters through the dendrites, their effect accumulates producing a slow decrement in the membrane potential. This causes a change in the membrane permeability, allowing ions to go through it (sodium goes in and potassium out). This way, the membrane polarity inverts and an electric spike is produced as a consequence. The potential inversion produces another change in the membrane permeability and the neuron goes back to its resting state.

The potential inversion propagates through the axon, and causes the emission of neurotransmitters in the dendrites. As a result, the neurons produce trains of spikes whose frequencies are proportional to the number of neurotransmitters received at their input. One feature of this mechanism is that more active neurons generate spikes faster the other ones. Therefore, the most important information propagates much faster through the different layers of neurons inside the brain, producing very fast responses with only a small amount of spikes.

There are two different kinds of synapses: inhibitories, whose neurotransmitters tend to stabilize the membrane potential, and excitatories, whose neurotransmitter tend to decrease the membrane potential and favour spike production. Each neuron has input synapses of both kinds. These synapses have weights which regulate the influence of neurotransmitters, and learning capability is enabled by changing these weights.

When a neuron receives no inputs, the membrane allows a small number of ions to flow outside of the cell, making the membrane potential tend towards the resting potential. Then, older input events are less relevant in the neuron present state. This is what we call forgetting mechanism, and it is fundamental to detect spatio-temporal correlations, which is one way of coding information in the brain [81]. Without this mechanism, it would be impossible to distinguish between old and new information.

4.3. First approach: the analog pixel

There are several neuron models which try to reproduce the biological neuron behavior, and one of the most useful is the one proposed by Hodgkin and Huxley [84]. It is a conductance-based model extracted from the study of the behavior of the squid neuron giant axon. The main idea presented by this model is the reproduction of the flow of ions through the neuron membrane with conductances. This model is represented in Fig. 4.3, and it is based on three conductances: *Na* models the flow of sodium ions through the membrane, *K* represents the transit of potassium from inside the cell, and *R* models the leakage current that implements the forgetting mecha-

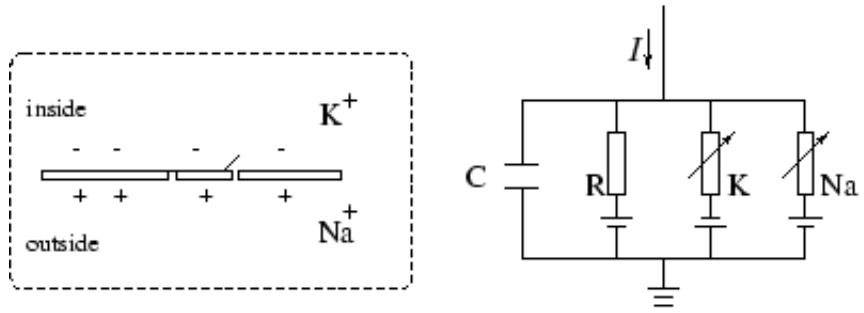


FIGURA 4.3. Neuron model scheme proposed by Hodgkin and Huxley.

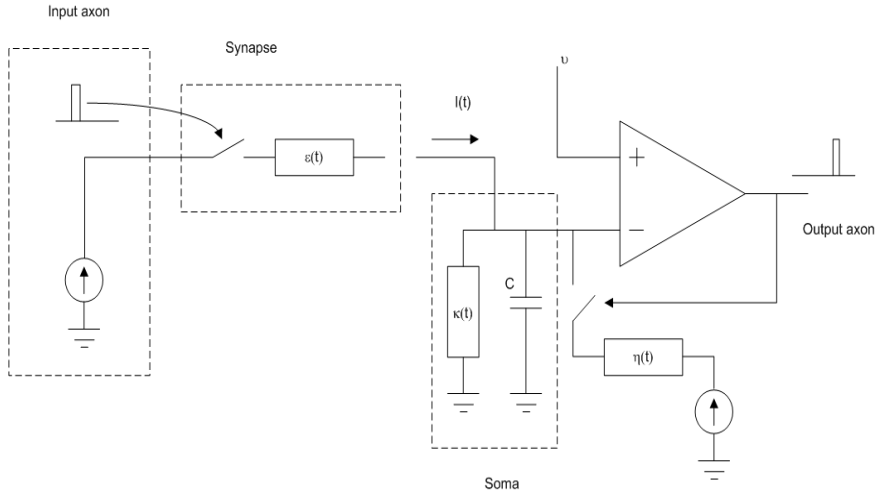


FIGURA 4.4. SRM neuron model scheme.

nism. As shown in the figure, the conductances related to flow of ions are variable, as they depend on the membrane potential. The current *I* represented in the figure models the input received by the neuron, produced by connections with other neurons.

This model describes precisely the membrane potential behavior, but it is quite complex in terms of computational load. For this reason, SRM model (Spike Response Model) was proposed [85]. This model is repre-

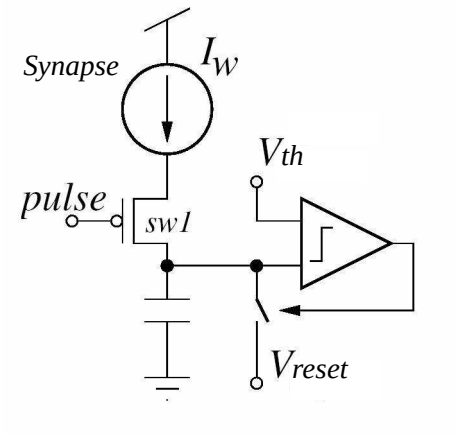


FIGURA 4.5. Simplified scheme of the Integrate&Fire analog neuron proposed by Serrano [78].

sented in Fig. 4.4. Each time a spike is received through the input axon, a certain amount of charge is injected into the soma, weighted by the corresponding synapse. This is modelled by function $\varepsilon(t)$. This charge is integrated in capacitor C, so that when the voltage reaches a threshold υ the neuron produces a spike through the output axon, and its state is reset to a resting value. This reset pulse affects the neuron potential through function $\eta(t)$. Any external input will have an effect on the membrane potential modelled by function $\kappa(t)$. All the interactions between different elements of the model are described by these temporal functions, which depend on the present state, but also on previous spikes, in order to make the model more realistic.

The analog Integrate&Fire neuron proposed by Serrano [78] is based on the SRM model, although more simplified, as it reduces the complexity of the functions $\varepsilon(t)$, $\eta(t)$ and $\kappa(t)$, using square pulses instead. Fig. 4.5 shows a reduced version of the analog neuron. Each time it receives an input spike the switch is closed, injecting current into the capacitor C. Then, the capacitor voltage is increased by the expression $\Delta V = \frac{I_w \cdot \Delta T}{C}$, being I_w the input current, ΔT the length of the pulse and C the capacitor. When this voltage reaches the threshold V_{th}, the comparator produces an output pulse that resets the capacitor to a voltage value V_{reset}.

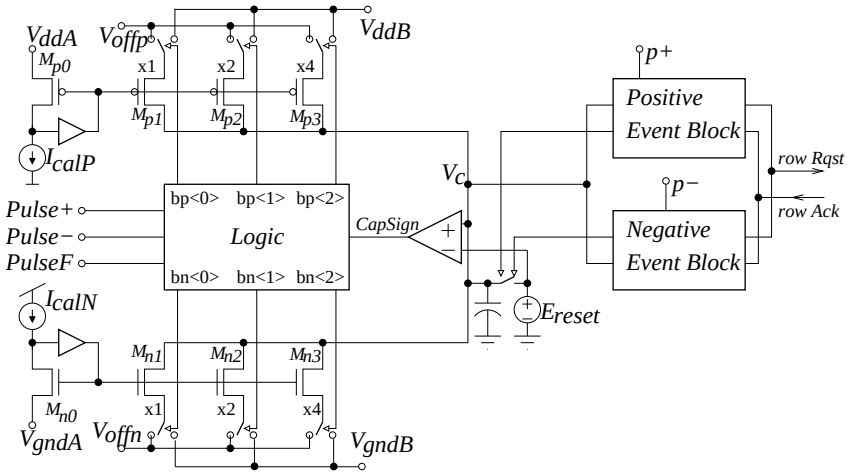


FIGURA 4.6. Complete block diagram of the convolution analog pixel.

In general, input spikes can be either positive or negative. Then, depending on the sign of the spike, the pixel must produce either an injection or a subtraction of charge into the capacitor, in both cases with a current proportional to the weight indicated by the kernel. This is shown in Fig. 4.6, where the complete diagram of the pixel is illustrated. When the pixel receives an input pulse in Pulse+ or Pulse-, the Logic block activates or deactivates transistors $M_{n1} - M_{n3}$ and $M_{p1} - M_{p3}$, depending on the sign of the event and the previously stored value of the kernel. Therefore, the structures formed by these transistors generate a current pulse from the corresponding calibration current I_{calN} or I_{calP}. Instead of a single comparator, two of them are necessary to be able to detect a positive or negative threshold in the capacitor voltage, and to produce signed output events. This comparator are called in Fig. 4.6 Positive Event Block and Negative Event Block. The weight given by the kernel must be stored in the pixel, so a dynamic memory is included in the Logic block.

To obtain a correct behavior of the pixel, it is also necessary to include calibration circuitery inside the pixel, due to the high mismatch between transistors biased in subthreshold region. To compensate these variation, some memory cells are included inside each pixel to store a 5-bit digital

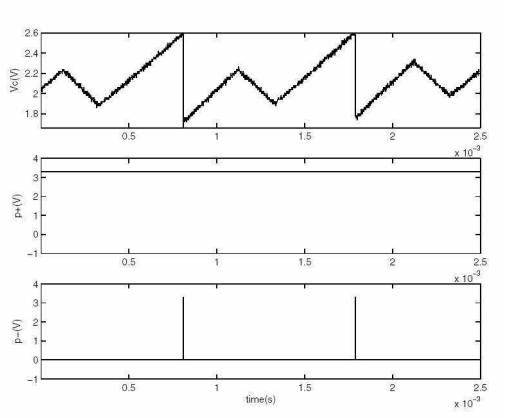


FIGURA 4.7. Representation of the capacitor voltage, and output spikes produced by the pixel.

word which controls a current mirror used to calibrate a common reference current, producing for each pixel its calibrated currents I_{calN} and I_{calP}. This increases the pixel area and the power consumption, and also makes it necessary to perform a calibration process once fabricated, calculating the optimized digital words for each pixel.

Fig. 4.7 shows the variations of the capacitor voltage as input spikes are received, both positive and negative. This voltage is increased or decreased depending on the sign of the input spikes, until it reaches a threshold and it is reset after generating a signed output spike.

In short, the main limitations presented by this analog pixel are:

1. The necessity of introducing calibration circuitery inside the pixel to compensate mismatching between transistors.
2. Reduced precision achieved by the pixel, due to subthreshold operation in analog transistors.
3. High event latency (around $1ms$ delay between an input spike and its corresponding output), produced by the slow comparators inside the pixel, as they are biased for low current.

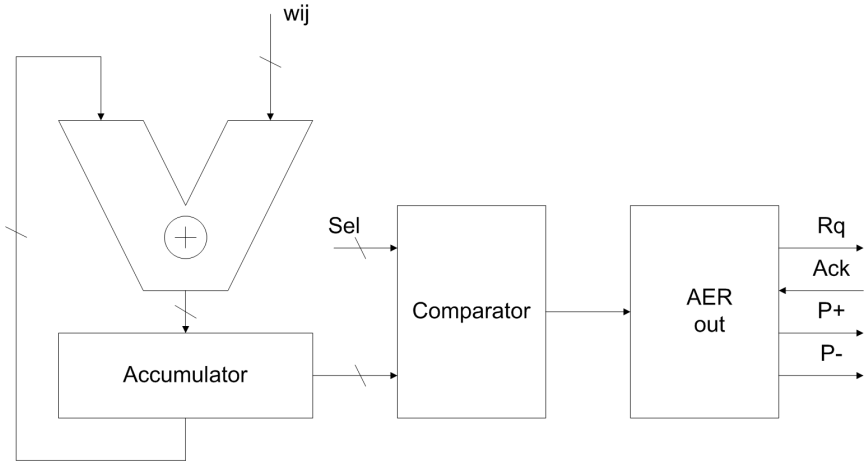


FIGURA 4.8. Basic structure of the proposed digital convolution pixel.

To overcome these limitations, a new digital pixel is proposed in this thesis, and it is described in the following section.

4.4. The digital pixel

In both convolution chips proposed (Conv1 and Conv2), the convolution operation is performed at the pixel level by integrating input events weighted by the kernel values. The generic structure of this digital pixel is shown in Fig. 4.8. The pixel consists on an accumulator and a full adder, which receives as inputs the kernel value and the state stored in the accumulator. Also, a comparator generates a pulse when the accumulator reaches a programmed threshold, and an interface block handles the communication between the pixel and the AER periphery to produce output events. A different pixel has been designed for each version of the convolution chip, although both of them are based in the same structure. This section describes in detail both digital pixels, starting with the initial version to finish with the improved one.

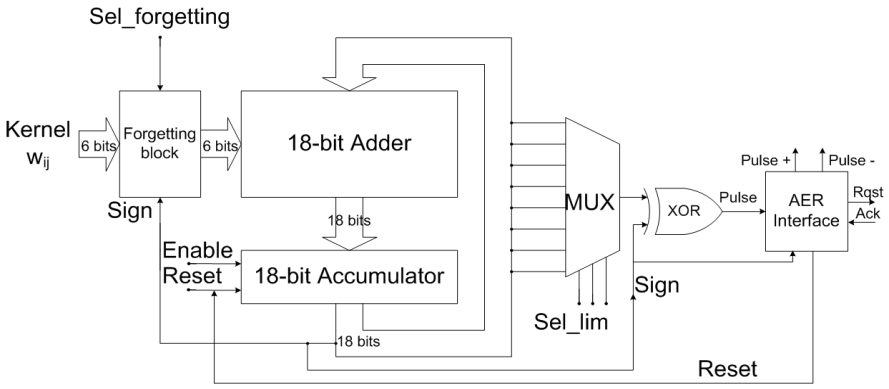


FIGURA 4.9. Block diagram of the initial version of the digital convolution pixel.

4.4.1. Initial version Conv1

The digital pixel is formed by an 18-bit adder and accumulator, whose block diagram is shown in Fig. 4.9. In the initial version, an over-dimensioned adder was implemented intentionally, so that the maximum possible precision could be obtained. The criteria was to allow accumulation of a kernel sized 32×32 with all weights at maximum 6-bit value, while allowing the least significant bit of to contribute as well to the accumulation. Thus, a dynamic range of 18 bits was chosen.

In previously described analog models, the state of the neuron was stored in a voltage capacitor. However, in this digital pixel, the state is stored in the accumulator, represented as a signed digital word coded in 2's complement with 18 bits (17 bits + sign). Therefore, the state of the pixel can change from $-2^{17} = -131072$ to $2^{17} - 1 = 131071$. The kernel values are also coded in 2's complement with 6 bits (5 bits + sign), so the minimum and maximum values are $-2^5 = -32$ and $2^5 - 1 = 31$.

The operation of the pixel is as follows: each time an input event is received by the convolution chip, the enable signal (which is common to all the pixels in the same row) is activated, and the accumulator is updated with

TABLA 4.1. Programmable accumulator limits.

Sel_lim	Maximum	Minimum
000	$2^{16} = 65536$	$-2^{16} - 1 = -65537$
001	$2^{0} = 1$	$-2^{0} - 1 = -2$
010	$2^{4} = 16$	$-2^{4} - 1 = -17$
011	$2^{5} = 32$	$-2^{5} - 1 = -33$
100	$2^{1} = 2$	$-2^{1} - 1 = -3$
101	$2^{7} = 128$	$-2^{7} - 1 = -129$
110	$2^{3} = 8$	$-2^{3} - 1 = -9$
111	$2^{2} = 4$	$-2^{2} - 1 = -5$

the corresponding kernel weight. If it reaches the programmed threshold, an output event is generated and the accumulator is reset to its resting value 0.

This initial version of the pixel was designed to obtain as much programmability as possible, so an 8-input multiplexer was included to be able to select between 8 possible accumulator limits, in order to adapt the size of the accumulator for different applications. Hence, a 3-bit control parameter is used (Sel_lim) to select one of the accumulator bits. This selected bit is compared continuously with the sign bit (the most significant one). Therefore, for positive values ($msb = 0$) the comparator will fire when the selected bit becomes '1', while for negative values ($msb = 1$) it will fire when the selected bit becomes '0'. The 8 possible thresholds are shown in Tabla 4.1. Notice that a XOR gate is used to implement the comparator (as shown in Fig. 4.9), so if the kernel value is greater than the programmed threshold a wrong behavior could be produced. For that reason, we must make sure that the maximum kernel value w_{ij} used will be smaller than the selected threshold. For instance, if bit2 was selected as threshold (so the comparator would fire when the accumulator reaches 4) a mistake would be produced if the added kernel value was 8 ($001000|_2$), as it would not affect

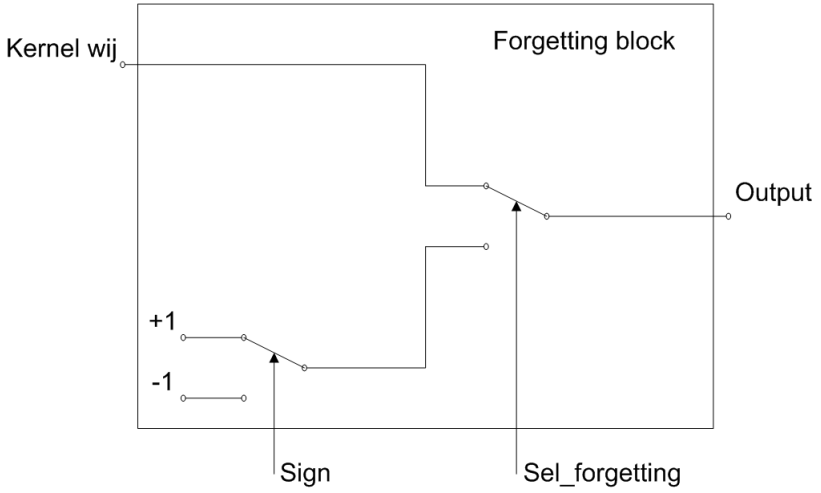


FIGURA 4.10. Scheme of the forgetting block.

the selected bit. This problem is solved by limiting the kernel values to be smaller than the programmed accumulator threshold.

For implementing the forgetting mechanism, the synchronous controller generates periodic forgetting pulses that must be processed by the pixels. This forgetting pulse produces a pulse in the enable signal for all the pixels in the array, updating the accumulators. Therefore, when the pixel receives an enable pulse, its response will depend on the value of the Sel_forgetting signal. If this signal indicates the enable pulse corresponds to a forgetting action, the Forgetting_block in Fig. 4.9 will select a different input for the adder (instead of the kernel value w_{ij} stored in the RAM), as shown in Fig. 4.10. This block includes a set of switches controlled by the Sel_forgetting signal, so that they can select '1' as adder input if the accumulator value is negative or '-1' if the accumulator value is positive. Fig. 4.11 shows the detailed schematic of this block. If Sel_forgetting=0, then Output=w_{ij}, while if Sel_forgetting=1, Output will become $000001|_2 = 1|_{10}$ if the accumulator value is positive (Sign=0), and Output will become $111110|_2 = -1|_{10}$ if the accumulator value is negative (Sign=1). This way, when a pixel receives no input events, after a controlled time it will reach a resting value, which will oscillate between '0' and '-1', values produced by the forgetting mechanism.

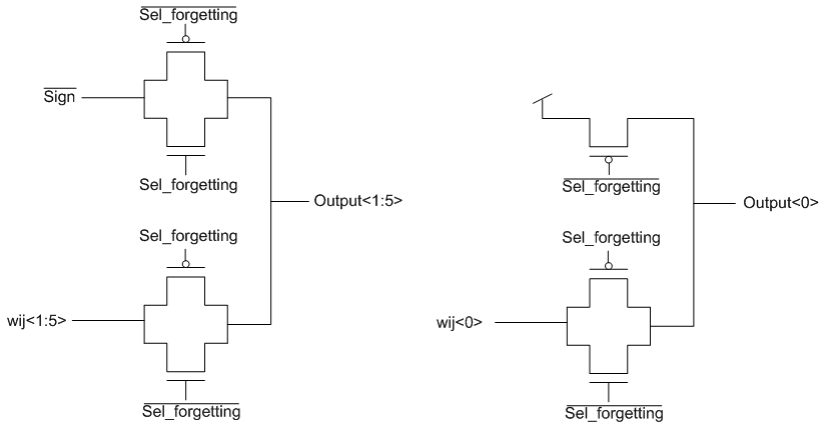


FIGURA 4.11. Detailed schematic of the forgetting block.

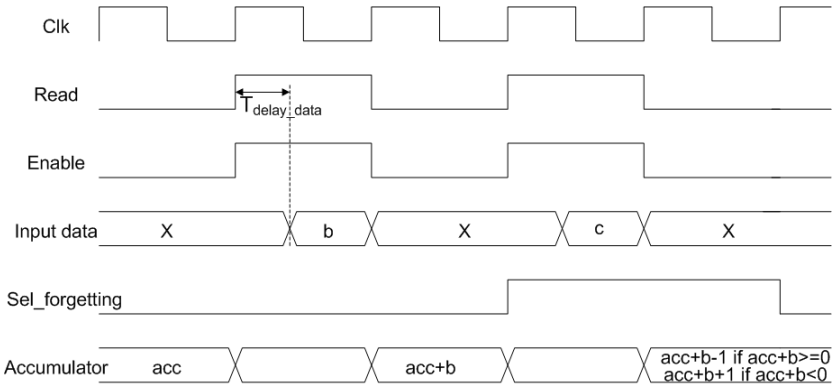


FIGURA 4.12. Timing diagram of the pixel behavior.

Fig. 4.12 shows the timing diagram of the pixel behavior. Signal Clk represents the synchronous controller clock, while signals Read, Enable and Sel_forgetting are generated by the controller. When Read is activated, the reading operation of the data stored in the RAM is enabled, so this data is sent to all the pixels in the same column. At the same time, an Enable pulse is generated, so that the output of the adder will be stored in the accumulator with its falling edge. T_{delay_data} represents the time between the instant

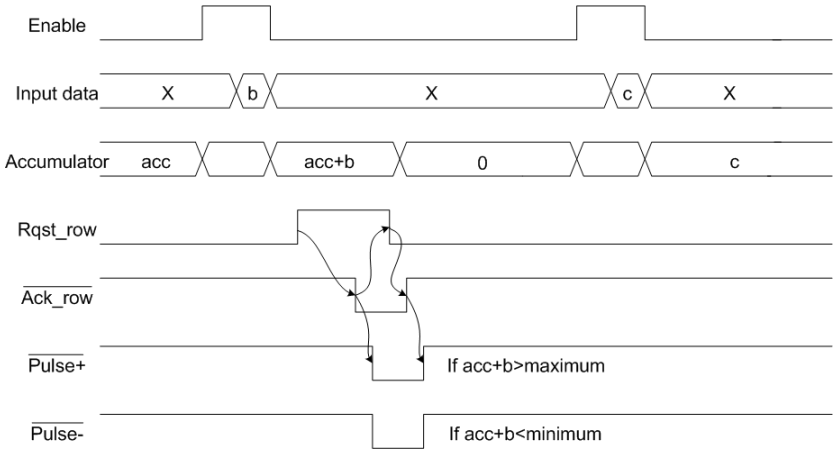


FIGURA 4.13. Timing diagram of the handshaking between the píxel and the periphery.

when the data is read from the RAM and it has been added by the pixel. To obtain a correct behavior of the pixels, the clock period T_{clk} must fulfill the condition $T_{clk} \geq T_{delay\_data}$. This delay depends on the adder structure included in the pixels, and also on the time spent by the kernel data to be processed by the 2's complement inverter and the horizontal shift block. The timing diagram in Fig. 4.12 shows how the pixel is increased or decreased when it receives a forgetting pulse, depending on its previous sign.

The timing diagram in Fig. 4.13 shows what happens when the accumulator reaches the programmed threshold, either the positive or the negative one. The four-phase handshaking can be better explained with the schematic in Fig. 4.14, which describes the AER-Interface block in Fig. 4.9, which handles the communication between the pixel and the periphery. When the pixel has not reached the threshold value yet, signal Rqst_row is at low level, while $\overline{\text{Ack\_row}}$, $\overline{\text{Pulse+}}$ and $\overline{\text{Pulse-}}$ are high. They are set to these levels by pull-ups and pull-downs in the AER-out block, in the periphery. At the same time, signals Pulse and Reset_pulse are at low level. When the accumulator reaches the programmed threshold (either the positive or the negative one), signal Pulse goes high (set by the XOR comparator) and it activates transistor M_{n4i}. This produces that the inverter formed by transistors M_{p3i} and M_{ri} activates signal Rqst_row. This signal is common to

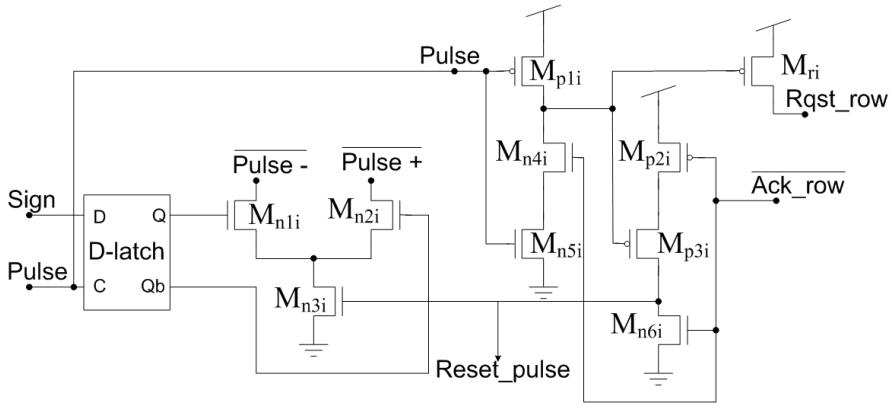


FIGURA 4.14. Schematic of the interface block inside the píxel which handles the communication with the periphery.

all the pixels in the same row, implementing a wired-OR. Then, each time a pixel reaches the limit, the row arbiter receives a request from the corresponding file. Therefore, when the row arbiter acknowledges, signal $\overline{\text{Ack\_row}}$ is activated at low level, producing that the inverter formed by transistors M_{p2i} and M_{n6i} activates signal Reset_pulse (at high level). This signal resets the accumulator, allowing signal Pulse to go back low, turning off transistor M_{ri} so that Rqst_row will be pulled down by the periphery. While signal $\overline{\text{Ack\_row}}$ is active, transistor M_{n4i} is off in order to prevent a new event to be produced before the handshaking process is finished. Signal Reset_pulse turns on transistor M_{n3i}, activating either $\overline{\text{Pulse+}}$ or $\overline{\text{Pulse-}}$, depending on the sign stored at D-latch (it was stored when signal Pulse was activated). Once the row arbiter detects that Rqst_row is low, $\overline{\text{Ack\_row}}$ is set back to high level, turning off transistor M_{n3i}. Finally, either $\overline{\text{Pulse+}}$ or $\overline{\text{Pulse-}}$ will be pulled up by the periphery.

Several alternatives were considered to implement the full adder [86], [87]. Although some alternatives presented a smaller number of transistors and lower consumption, for this initial version the conventional CMOS adder was chosen, as lower raise and fall times were achieved. These characteristics were critical for implementing an 18-bit adder, with 18 cascaded cells. This adder is shown in Fig. 4.15, with 28 transistors for each bit. Each

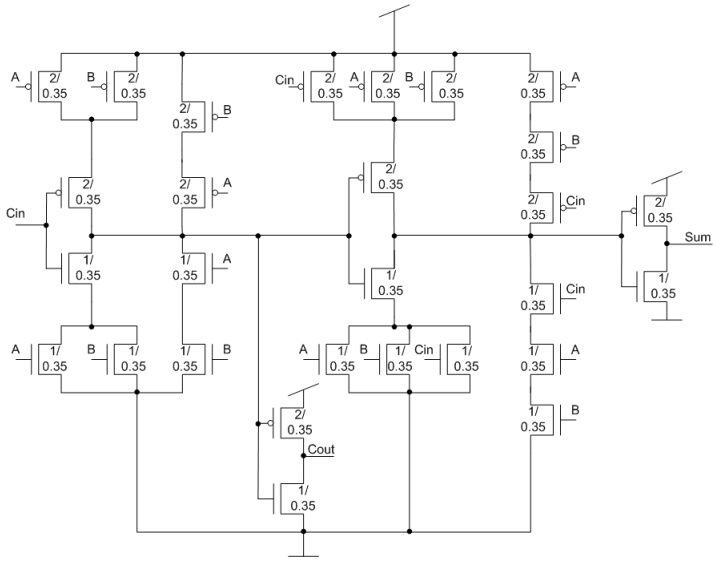


FIGURA 4.15. Conventional CMOS adder used for the initial version of the digital pixel.

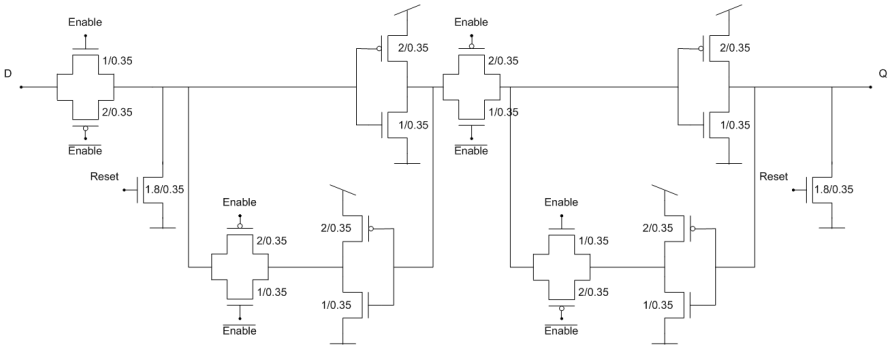


FIGURA 4.16. Schematic of the flip-flop used to implement the accumulator.

cell has three inputs: the binary numbers A and B and the previous bit carry (Cin); and two outputs: the Sum and the carry for next stage (Cout).

The accumulator includes 18 flip-flops (one per bit) where the adder output is stored with the falling edge of signal Enable. Its schematic is

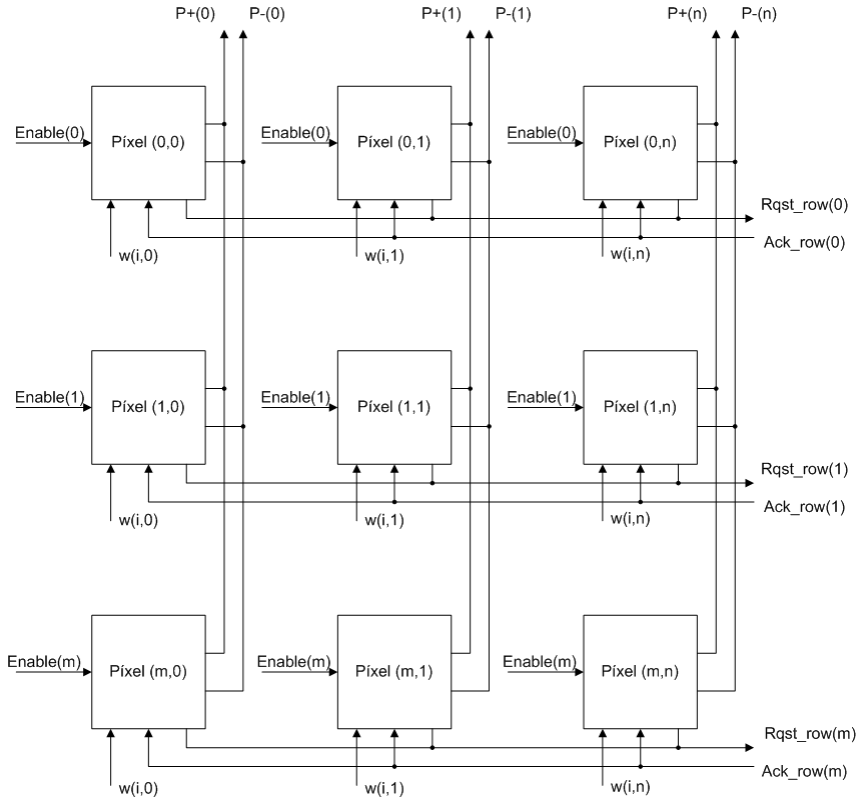


FIGURA 4.17. Array configuration of the convolution pixel.

shown in Fig. 4.16. Signal Reset writes a 0 value in the accumulator when the threshold has been reached, or when an external reset is produced.

The pixel is designed to build a 32×32 array. The kernel data are shared by pixels in the same column, as the information written in the RAM is selected row by row, while Enable signal is common to all the pixels in the same row, although it will only produce an effect for the pixels that are receiving a non-zero value (pixels included in the projection field). The handshaking signals Rqst and Ack are common for pixels in the same row, while Pulse+ and Pulse- are common for pixels in the same column. The array configuration is illustrated in Fig. 4.17.

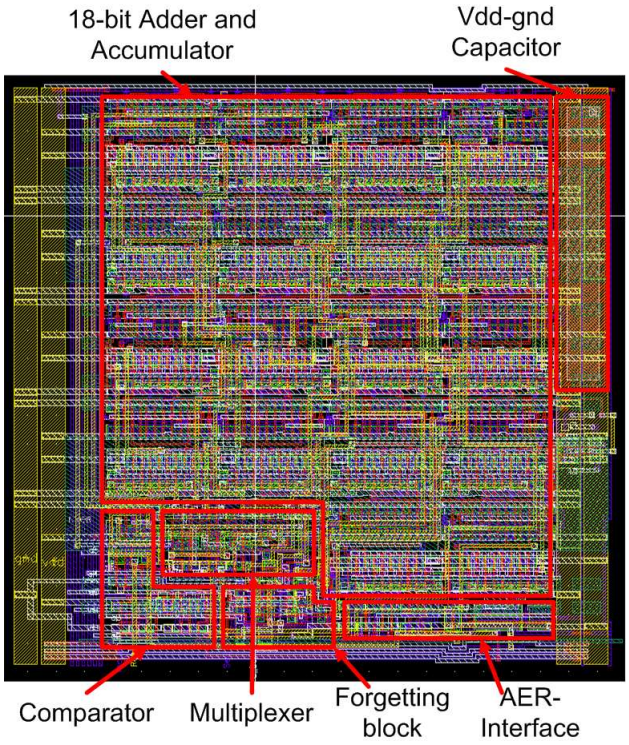


FIGURA 4.18. Layout of the initial version of the digital pixel.

Fig. 4.18 shows the pixel layout, with an area of $95.6 \times 101.3 \mu m^2$. Most of the area is consumed by the 18-bit adder and accumulator. A capacitor has been included between supply and ground to filter power glitches. This $240fF$ capacitance is placed under the supply and ground stripes to avoid extra area consumption.

The routing of the lines inside the pixel is a highly critical issue, with critical parasitic capacitance couplings, as some of them are shared by all the pixels in the same row or column, and can be as long as $3mm$. Some of these lines are used for configuration parameters, which are loaded at start-up and remain silent throughout normal operation. These “static” lines were layed out between fast dynamic lines, avoiding couplings among dynamic lines.

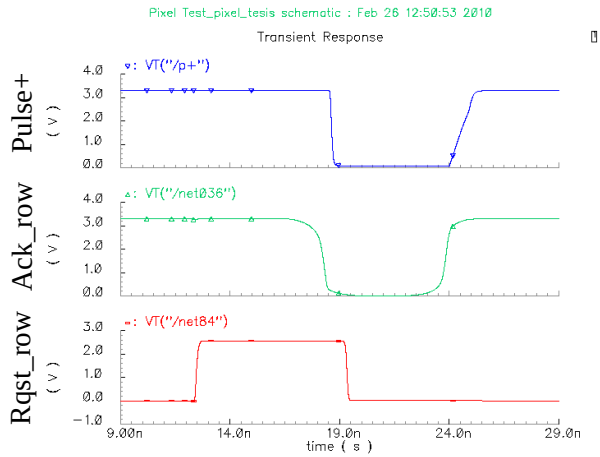


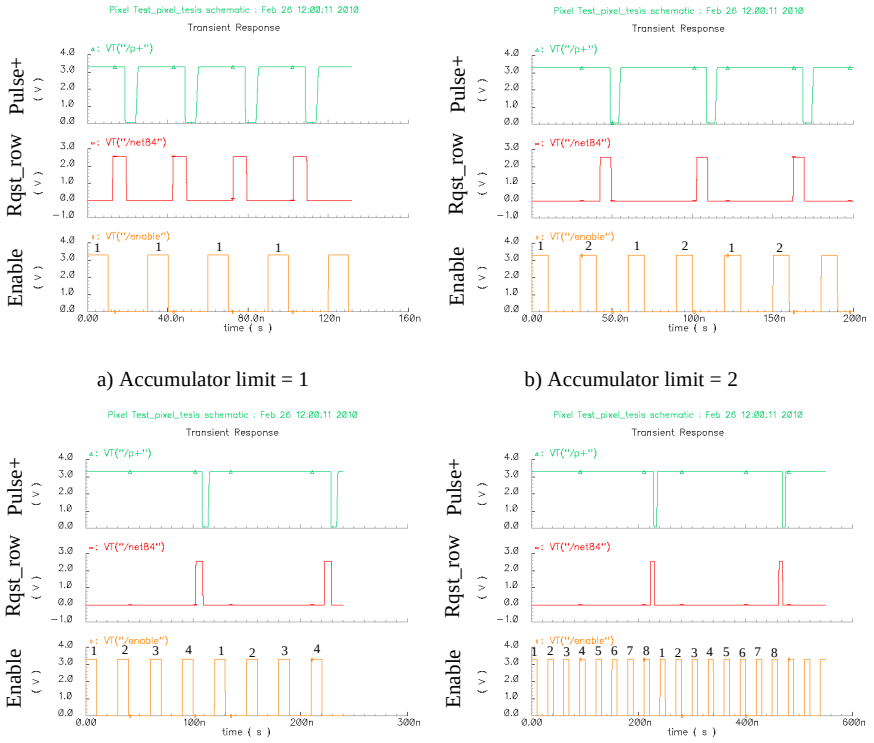
FIGURA 4.19. Handshaking signals obtained from simulation.

4.4.1.1. Simulation results

Once described the initial version of the convolution digital pixel, this subsection includes some results obtained from electrical simulation with Spectre (Cadence) which illustrate the behavior of the pixel.

The results in Fig. 4.19 show the behavior of the AER Interface block, when the pixel has reached a positive threshold and activated signal Rqst_row. Once the arbiter acknowledges through Ack_row, the pixel deactivates Rqst_row and activates Pulse+, until the arbiter deactivates signal Ack_row. The reason for the different slopes between the rising and falling edge in Pulse+ is related to the way how this signal is set. The pixel sets it low when it is indicated, producing a fast transition. However, it is set up by a pull-up in the periphery, which is much slower. Now, some examples are shown to check the correct pixel behavior.

1. On example 1, the kernel data is set to 1, in order to check with precision the adder behavior. Each subfigure in Fig. 4.20 shows what happens when the pixel receives input events with 4 different selected thresholds (1, 2, 4 and 8, respectively). The figures show signals Enable (which rep-



a) Accumulator limit = 1

b) Accumulator limit = 2

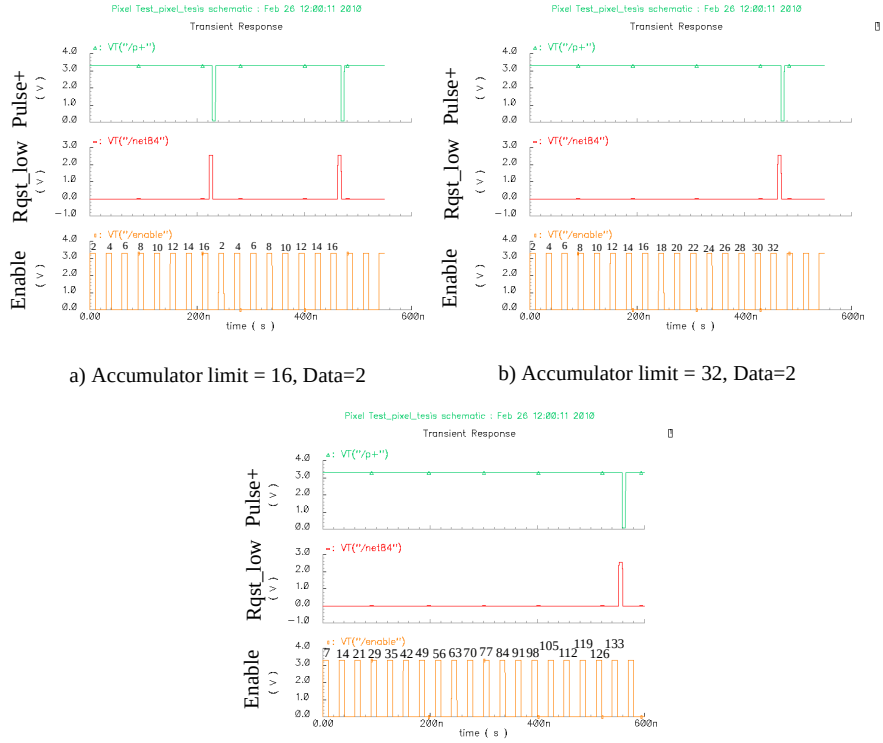
c) Accumulator limit = 4

d) Accumulator limit = 8

FIGURA 4.20. Simulation results from example 1, with kernel data = 1, and the accumulator limit = 1, 2, 4 and 8.

resent the input events), Rqst_row and Pulse+ (which are generated by the pixel). Above each Enable signal the state of the accumulator after processing each event is indicated. Therefore, it can be checked how the Rqst_row pulses are produced with the correct accumulator value.

2. On example 2, some simulations were performed changing the kernel data. The results are shown in Fig. 4.21. First, the kernel data is 2, while the accumulator limit is 16. The corresponding figure a) shows how the pixel receives 8 input events before reaching the limit and activating signal Rqst_row. In figure b) the threshold is increased to 32, so 16 input events are received before reaching the threshold. Finally, as shown in figure c), the kernel data is set to 7 while the accumulator limit is 128.



a) Accumulator limit = 16, Data=2

b) Accumulator limit = 32, Data=2

c) Accumulator limit = 128, Data=7

FIGURA 4.21. Simulation results for example 2, with kernel data = 2 in a) and b), and kernel data = 7 in c). Accumulator limit = 16, 32 and 128, respectively.

Once more, the numbers above the enable pulses indicate the state of the accumulator after adding each event. After adding 18 input events, the state of the accumulator is $7 \times 18 = 126$, so the limit has not been reached yet. However, when event number 19 is received, we obtain $126 + 7 = 133 > 128$, so an output event is generated. In this case, it can be seen how the pixel detects that the limit is exceeded, with no need of obtaining the exact threshold value.

3. On example 3, the effect of negative data and forgetting pulses is checked, as shown in Fig. 4.22. The kernel data used is -1, while the accumulator limit is -2. When the pixel receives the first input event, the

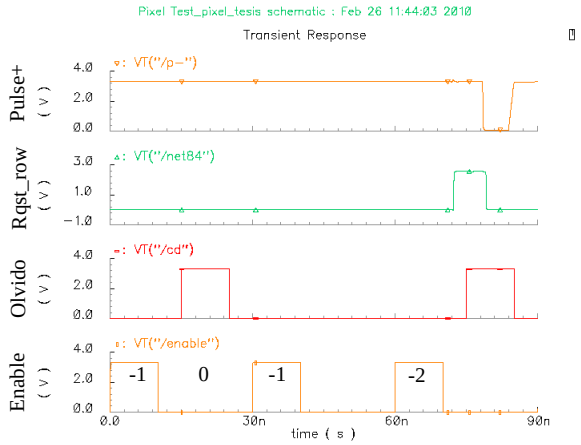


FIGURA 4.22. Simulation results for example 3, with kernel data = -1, and accumulator limit = -2, with the forgetting effect enabled.

state of the accumulator is -1, so when a forgetting pulse is generated, it adds +1 instead of -1, as the previous state is negative, reaching 0 again. After that, two consecutive input events produce an output event, although it is Pulse- the signal activated, indicating it has reached the negative threshold.

4.4.2. Advanced version Conv2

Once fabricated and tested convolution chip Conv1 (based on the initial version of the digital pixel described in previous subsection), we checked that for some applications such a large accumulator was not necessary, specially when fast output events were expected from the convolutional operation. However, a larger number of pixels integrated in a single chip was considered an interesting improvement, so an advanced version of the pixel was designed to achieve a larger resolution with no cost in terms of area.

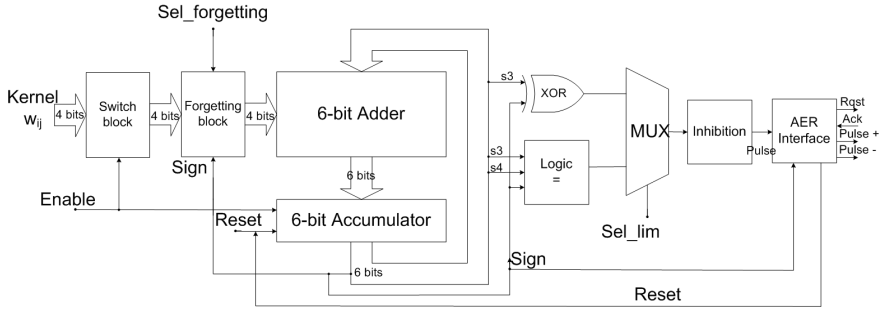


FIGURA 4.23. Block diagram of the advanced version of the convolution digital pixel.

This pixel uses a 6-bit accumulator (5 bits + sign) and a 4 bit kernel data (3 bits + sign). This accumulator could represent numbers in 2's complement from $-2^5 = -32$ to $2^5 - 1 = 31$, while the minimum and maximum values for the kernel are $-2^3 = -8$ and $2^3 - 1 = 7$.

Fig. 4.23 shows the block diagram of the new convolution pixel, with some changes with respect the previous one: the Switch block, the "Logic = "block to select the accumulator limit, and the Inhibition block.

1. Switch block. This block was added to the pixel to reduce the unnecessary poser consumption from previous version. The kernel data written in the RAM are selected row by row, so when a single row is selected it reaches the input of all the pixels in the same row. In the previous version of the pixel, all these pixels in the same column would calculate the addition of these data, although only the pixel in the row indicated by the Enable pulse would store the result. This operation mode produces correct results, but there are some extra power consumption. To avoid this, the switch block was introduced in the advanced version of the pixel. Fig. 4.24 illustrates the functionality of this block, with a timing diagram of its behavior. Unless the pixel receives an Enable pulse, no operation will be performed. The rising edge of Enable allows the data to be processed by the adder, while the falling edge stores the obtained result. This sets a limit on the maximum clock frequency, as an Enable pulse lasts for one clock cycle. A very high clock frequency produces a very

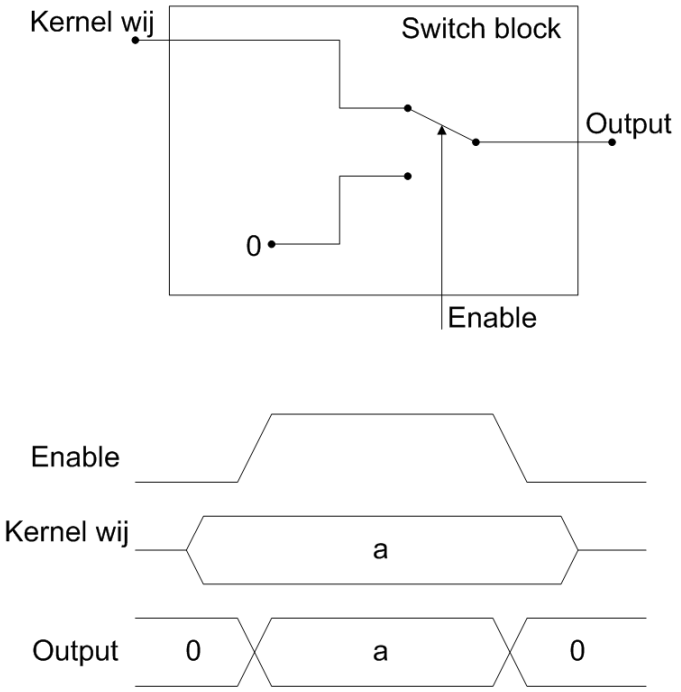


FIGURA 4.24. Scheme of the switch block and timing diagram of its behavior.

short Enable pulse, and the result is stored before the adder has finished. Fig. 4.25 shows the detailed schematic of the switch block.

2. Accumulator limit selection. Previous version included the possibility to select between 8 different thresholds, as the accumulator was over-dimensioned and several tests need to be done. However, with the present 6-bit accumulator, such a high programmability makes no sense. Following the same scheme from the initial pixel, the maximum and minimum thresholds would be $2^4 = 16$ and $-2^4 - 1 = -17$, although the accumulator can represent values between -32 and 31. This happens because we are using a single bit for the comparator. The whole resolution could be used by using all the bits in a more complex comparator, although it would produce an area increment. The proposed alternative compares only two bits, obtaining thresholds $011000|_2 = 24|_{10}$ and

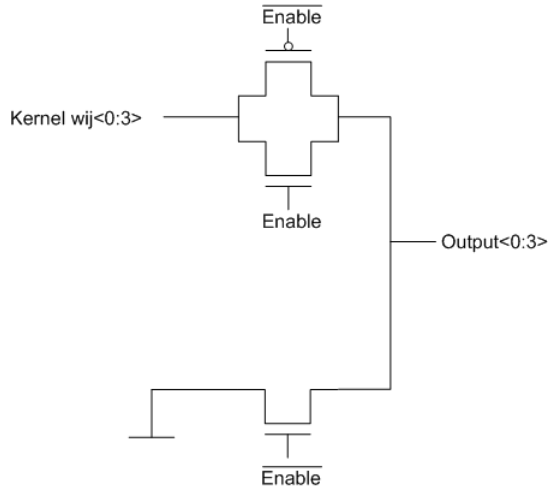


FIGURA 4.25. Schematic of the switch block.

TABLA 4.2. Programmable accumulator limits.

Sel_lim	Maximum	Minimum
0	8	-9
1	24	-25

$100111|_2 = -25|_{10}$. This comparison is implemented by the logic gate shown in Fig. 4.26. The inputs A, B and C are connected to bit3, bit4 and the inverted sign. However, a 2-input multiplexer is included to introduce the possibility of selecting either -25 and 24 as thresholds, or $-2^3 - 1 = -9$ and $2^3 = 8$, just by comparing bit3 with the sign. The different thresholds which can be selected are shown in Tabla 4.2.

3. The inhibition block is included to implement a new functionality at the pixel level. This functionality is designed to select the sign of the desired output events. An example could be an application where the convolution chip is programmed to detect a certain shape, using a kernel that produces positive events in the center of this shape and negative events in the rest of the visual space. In this case, negative events carry no information for next processing layers. With the initial version of the convo-

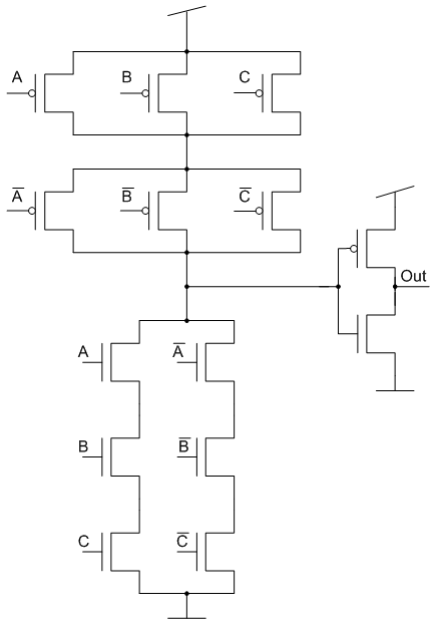


FIGURA 4.26. Logic gate which indicates if 3 input signals have the same value.

lution chip, an external mapper [48] had to be inserted to eliminate the negative events. However, the advanced version can discard these events at the pixel level, avoiding an external element (the mapper) and reducing the AER communication bandwidth consumed, with a minimum cost in terms of area. An alternative would be to implement this functionality in the periphery, but it would not reduce communication bandwidth. Therefore, a small block is introduced in the pixel which allows events generation only if their sign is not selected to be inhibited. Two global signals are added to indicate which sign has to be inhibited (positive or negative, although both of them can be selected, or none of them). Fig. 4.27 shows the schematic of the logic gate introduced to implement this functionality. Signals Inh_p, Inh_n and Sign control the path between Pulse_in and Pulse_out nodes, or set the output node to ground. If a positive event is generated (Sign=0), there will be a path between Pulse_in and Pulse_out only if Inh_p=0 (no inhibition for positive events). For negative events (Sign=1), they only can be processed if

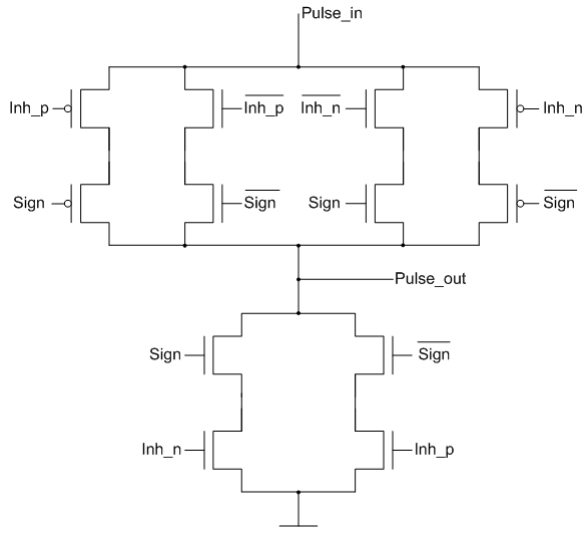


FIGURA 4.27. Logic gate used to inhibit spikes of the selected sign.

Inh_n=0. Otherwise, the pulse is not transmitted to the AER interface and signal Pulse_in will reset the accumulator.

4.4.2.1. Adder structures proposed

To minimize the area consumed by the pixel, two different approaches were analyzed to implement the full adder:

1. 10-transistor adder [88], based on an optimized design of the XOR function [89] and pass-through logic [90]. This structure implements logic functions with a smaller number of transistors, but produces signal degradation, so it is not a good alternative for our pixel. This cell is shown in Fig. 4.28.
2. 16-transistor adder [91] based on transmission gates. This model, whose schematic is shown in Fig. 4.29, includes transistors for signal regeneration. Therefore, we use this model for the advanced version of the pixel, as it produces a correct behavior with the minimum area consumption. Exhaustive corner and mismatching simulations were done to check the correct behavior of this model.

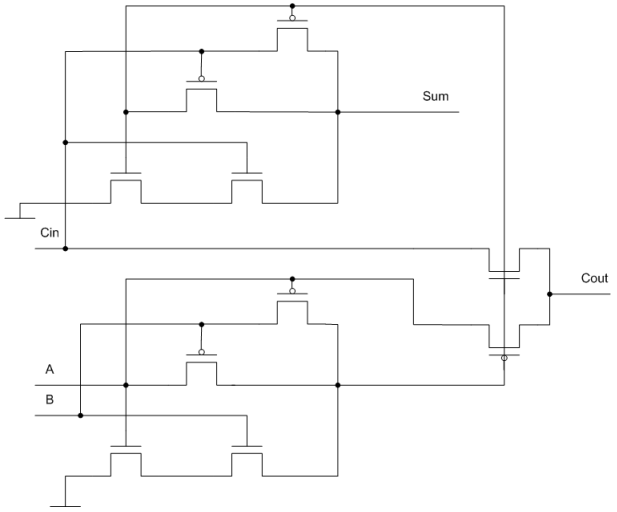


FIGURA 4.28. Schematic of the 10-transistors adder cell based on pass-transistor logic.

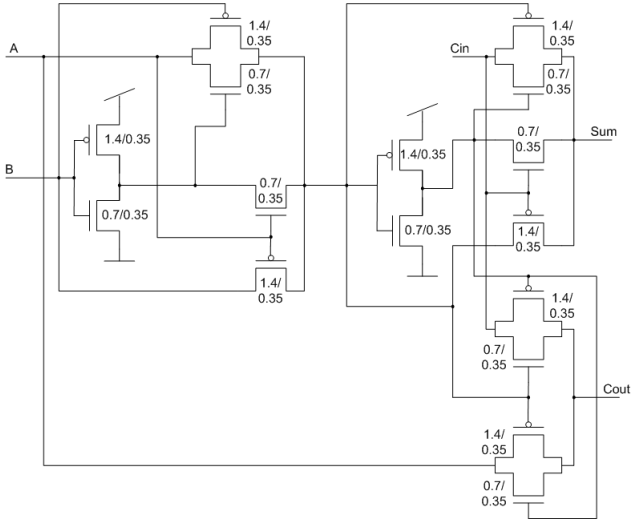


FIGURA 4.29. Schematic of the 16-transistors adder cell based on transmission gates, used for the advanced digital pixel.

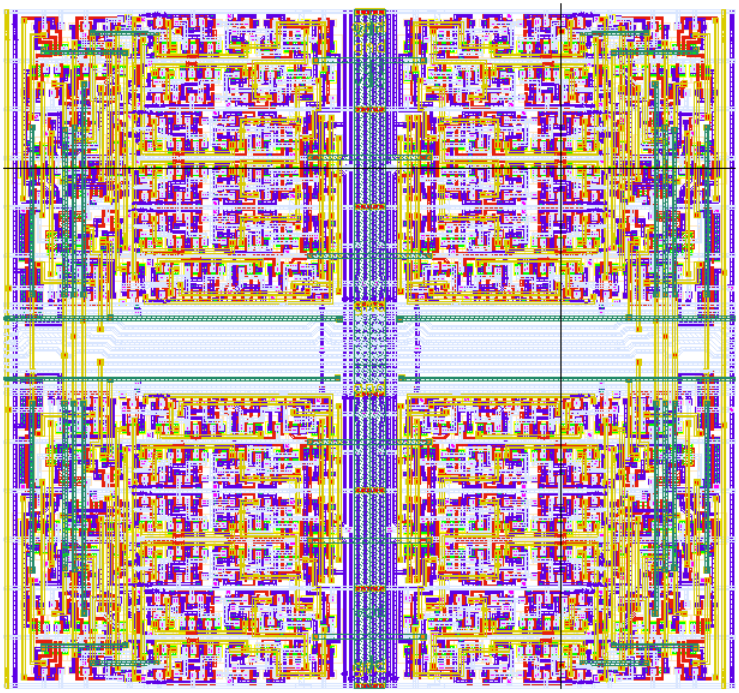


FIGURA 4.30. Layout of a cluster of 2x2 convolution pixels.

The pixel obtained has a total area of $58 \times 53.8 \mu m^2$. Sharing the supply and ground lines, and the common configuration signals, a 2×2 cluster was built with an area of $116 \times 107.6 \mu m^2$. A single pixel of the initial version consumed $95.6 \times 101.3 \mu m^2$, so the spatial resolution has been increased by a factor of 4 with a very reduced increment of area. Therefore, a 64×64 pixels array can be integrated in a prototype convolution chip with $5 \times 4 mm^2$. Fig. 4.30 shows the layout of a 2×2 pixels cluster.

4.4.2.2. Simulation results

Some simulations are presented to illustrate the behavior of the advanced convolution pixel.

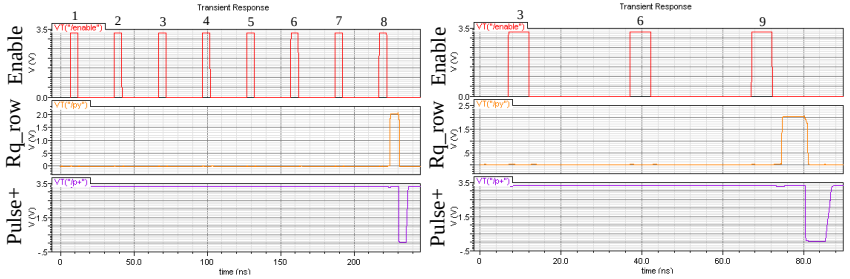


FIGURA 4.31. Simulation results for example 1, with accumulator limit = 8, and kernel data = 1 in the left figure and =3 in the right figure.

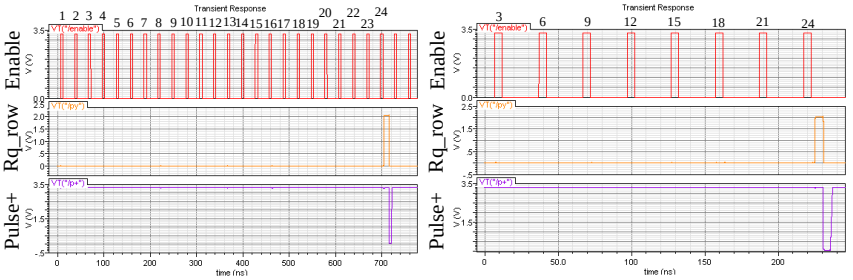


FIGURA 4.32. Simulation results for example 2, with accumulator limit = 24, and kernel data = 1 in the left figure and =3 in the right figure.

1. On example 1, the accumulator limit is programmed to be 8, while the kernel data is 1 and 3, respectively. Fig. 4.31 shows the results obtained. Both figures include the input Enable pulses and the output signals Rqst_row and Pulse+ generated by the pixel. Also a small number above each Enable pulse indicates the accumulator state after processing that pulse. In the figure on the left, the pixel activates Rqst_row after receiving 8 input events with data=1, while on the right the pixel activates Rqst_row after receiving 3 input events with data=3. Both results are correct.
2. On example 2, the accumulator limit is 24, with the same kernel data as previous example. Fig. 4.32 shows the results. On the left, with kernel data=1, 24 input events are received before the Rqst_row signal is acti-

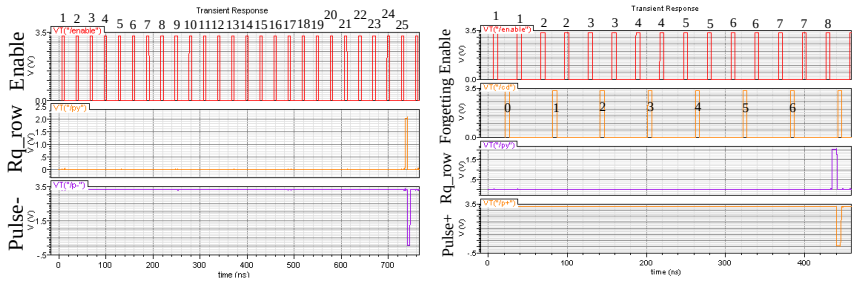


FIGURA 4.33. Simulation results for example 3. In the left figure, kernel data = -1, with accumulator limit = -25. In the right figure, kernel data = 1 and accumulator limit = 8, with forgetting mechanism enabled.

vated by the pixel. On the right, with kernel data=3, only 8 input events were necessary to reach the accumulator threshold.

3. On example 3, tow different simulations were performed. First, a negative value was programmed as the kernel value, -1, with an accumulator limit of -25. The left figure in Fig. 4.33 shows how 25 input events were added before signal Rqst_row was activated. The other simulation uses a kernel data=1 and accumulator threshold=8, but the forgetting mechanism is enabled (Forgetting signal). The right figure in Fig. 4.33 shows the accumulated value after each input pulse (either Enable or Forgetting), where 15 input events were needed to reach the threshold 8, due to the forgetting pulses.

Once described the convolution pixel, next chapter is dedicated to the rest of peripheral circuits included in the convolution chips Conv1 and Conv2.

CAPÍTULO 5

Peripheral circuitry in the convolution chips

5.1. Introduction

The convolution chips presented in this thesis must be able to receive input events and process them, adding the kernel stored in the RAM around a specific neighborhood of pixels given by the address of the input event. When one of these pixels reaches the threshold, an output event must be generated at the corresponding AER port. The main part for this processing is the convolution chip, which has been described in previous chapter, but some other blocks are necessary to perform the convolution operation.

Two different parts can be distinguished: the one which receives the input events and implements the convolution operation, and the one which arbitrates between the events generated by the pixels and sends them off-chip. The whole architecture is shown in Fig. 5.1.

The input part consists of the following blocks:

1. The synchronous controller, which latches the input event and calculates the precise operations that must be performed to process it, sending the

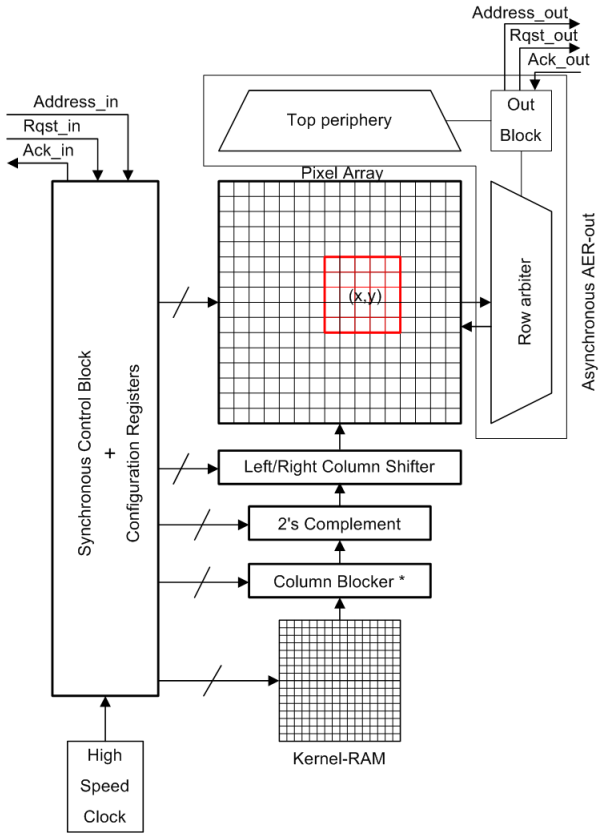


FIGURA 5.1. Architecture of the convolution chip (* Block included only in version Conv2)

corresponding signals the other blocks. It is described in detail in Section 5.2.

2. The RAM, where the convolution kernel is stored at start up. This block is described in Section 5.3.
3. 2's complement inverter. This block calculates the inverted values of the data in the RAM before being added to the pixels when the input event is negative. A complete description is given in Section 5.4,
4. Left/right column shifter. The kernel stored in the RAM must be centered around the pixel indicated by the input event address, so it must be shifted horizontally by this block, as is described in Section 5.5.

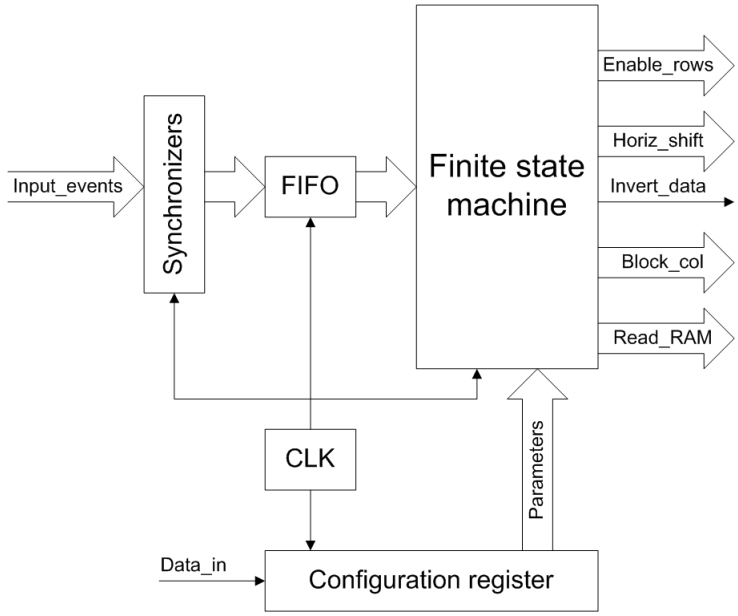


FIGURA 5.2. Block diagram of the synchronous controller.

The output part is what we call AER-out generator, and is described in Section 5.6.

This thesis includes the design of two different versions of the convolution chip Conv1 and Conv2, although both of them follow the same general architecture. However, during this chapter, each section will show how the corresponding block is adapted for each version.

5.2. The synchronous controller

The block diagram of the synchronous controller is shown in Fig. 5.2. The main part of this circuit is the finite state machine, which enables the operations that are performed by other circuits in the chip. However, the finite state machine needs some auxiliar blocks. The AER input events are

asynchronous, so a synchronizer block is implemented at the input of the controller (experimental results are given for this circuit in Section 5.2.4). The input queue (FIFO) stores the input events while they are waiting to be processed, reducing the probability of losing an event when the incoming rate is faster than the processing time. Once stored in the queue, the events are processed by the finite state machine, which enables the reading operation of the RAM rows where the kernel is written, inverts the data if the event is negative, indicates the precise horizontal shift to center the kernel around the correct pixels, and enables the addition of the kernel data into the pixel rows. In the advanced version of the convolution chip Conv2, it also controls some signals to block the columns of the RAM outside of the kernel to implement the multikernel capability.

The multikernel system (included only in Conv2) allows to program several different kernels (up to 32) in the RAM, so that each input event includes information about what kernel must be used to process it. Then, the controller must select the RAM rows where the indicated kernel is stored and block the columns that are not part of this kernel (in general, they can be part of another kernel). The following sections describe how this system affects the different circuits in the chip.

As is shown in Fig. 5.2, a clock generator is included in the chip, so that the synchronous controller can work without an external clock signal (although we also can use an external one). And finally, a configuration register is represented in the figure. This register is used to store several parameters at start up. Some of these parameters are used by the finite state machine (like the coordinates of the pixel array), while some others are used to configure the pixels directly (like the accumulator threshold).

The controller has been described with VHDL [93] and synthesized automatically, except for the configuration register, which was designed outside of the controller in version Conv1 (in Conv2 it was included in the whole block).

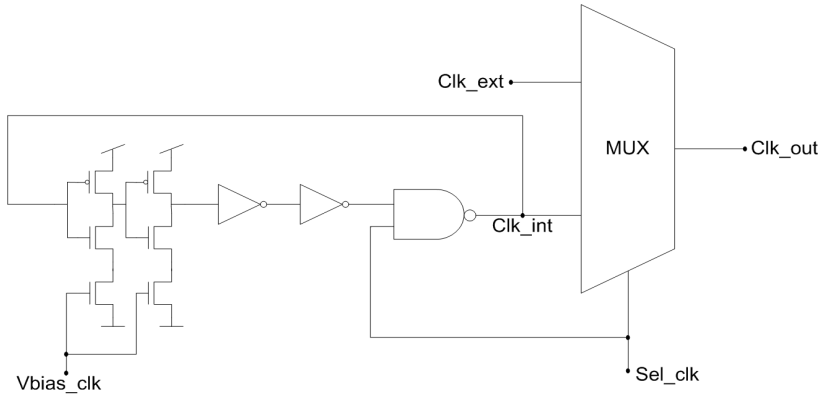


FIGURA 5.3. Schematic of the clock generator.

5.2.1. High-speed clock generator

The clock generator used is a ring oscillator with 5 inverters, like shown in Fig. 5.3. The fifth inverter is integrated in the NAND gate used to enable the circuit. There is a multiplexer which is used to commute through signal Sel_clk between the internally generated clock or an external one. The NAND gate stops oscillation when the external clock is selected, to avoid extra consumption and noise.

As indicated in the figure, two inverters in the ring have their maximum current limited through signal Vbias_clk, which is used to program the clock frequency off-chip. The clock signal generated by this block (Clk_out) is connected to a bonding pad through a frequency divider. Therefore, the clock frequency can be measured (divided by 32, to be visualized at the lab) and the control voltage can be adjusted to obtain the desired frequency. Fig. 5.4 represents the results obtained for the clock frequency simulating with Spectre the extracted circuit. The synchronous controller can work with a frequency up to 200MHz, although the maximum frequency used will be limited by other blocks.

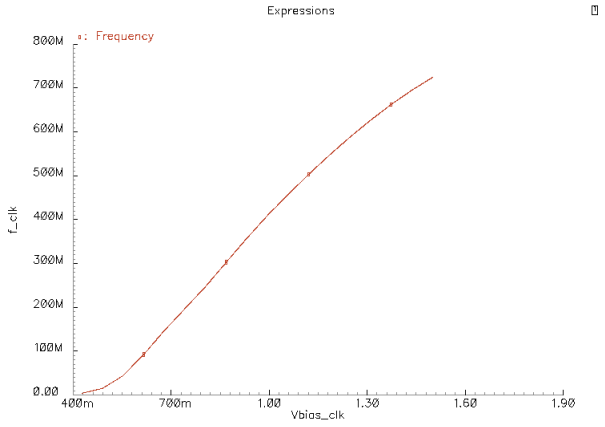


FIGURA 5.4. Clock generator characterization (oscillation frequency vs. control voltage).

5.2.2. The input FIFO

The input FIFO is the block which captures the input events (previously synchronized with the system clock) and sends them to the finite state machine to be processed. It includes a circular queue with 4 positions which acts as a buffer, so up to 5 events can be stored while the finite state machine is busy (the 4 positions of the queue and the extra position in the External Interface block). This queue is not designed to handle very fast average input traffic, as it would be saturated right after the state machine, but to handle short-time traffic peaks. If an event arrives when all the positions are busy, communication is stopped by not answering to the Rqst signal.

The operation of the FIFO is illustrated in Fig. 5.5, and is as follows. It waits until the input Rqst signal is active. Then, if there is any spare space in the queue (by checking the internal signal Full/$\overline{\text{Empty}}$), the event address is latched and the Ack signal activated. This process is implemented by the External Interface block, which also sends the latched event to the block Do

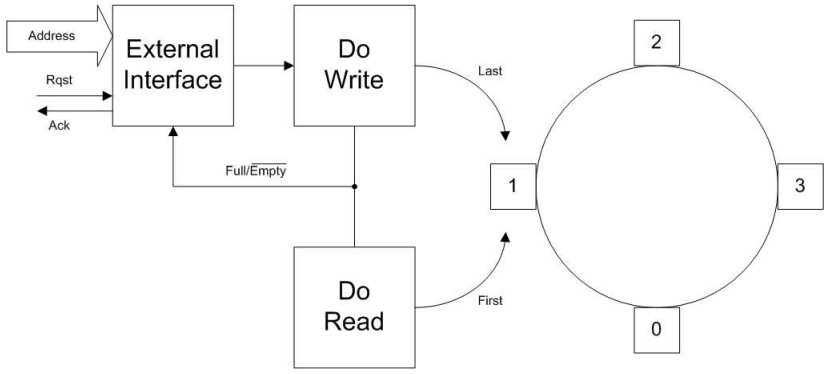


FIGURA 5.5. Diagram that illustrates the behavior of the input FIFO.

Write, which writes it in the first available position of the queue and increases the value of the register Last (which indicates the last element of the queue). Then, it verifies the value of (Last+1)MOD4, and compares it with FirstMOD4. If both of them are the same, it means the queue is full, so signal Full/$\overline{\text{Empty}}$ is activated.

When the finite state machine is idle, it checks for a new event by asking the Do Read block. This block looks at the values of First, Last and Full/$\overline{\text{Empty}}$. If (Last+1)MOD4 is different from FirstMOD4, or Full/$\overline{\text{Empty}}$ is active, it means that there is at least one event waiting to be processed. Then, the event stored in the position indicated by First is sent to the state machine, increasing the value of First, and deactivating Full/$\overline{\text{Empty}}$ if it was active.

5.2.3. The finite-state machine

The finite-state machine performs the convolution operation step by step, enabling the addition of the kernel onto the pixels row by row. It needs to know the coordinates of the pixel array and the input event, and also the kernel size, in order to calculate the exact position of the projection field. The projection field can be outside of the chip's address space, or partially inside, so the state machine must handle any possible situation. The state transition diagram is shown in Fig. 5.6, and is described next:

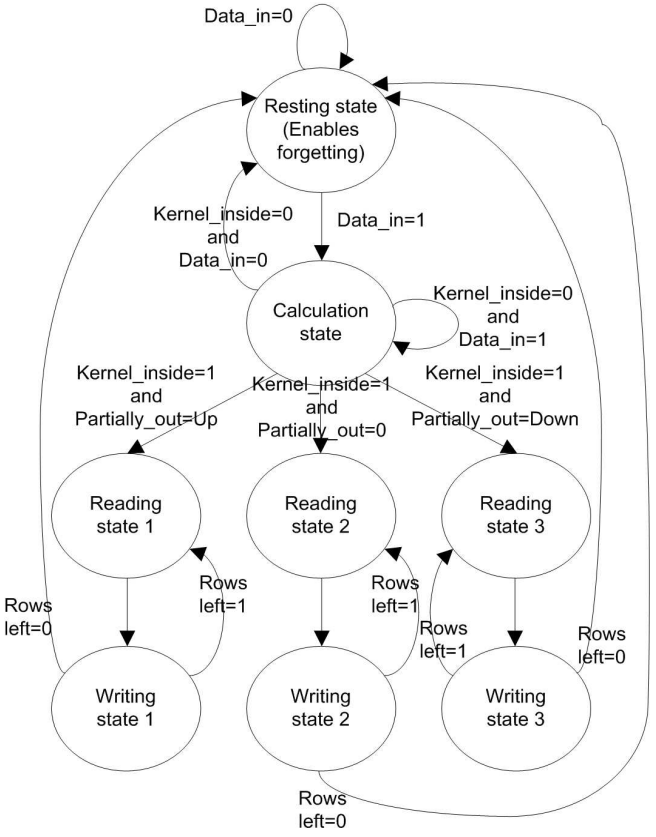


FIGURA 5.6. State transition diagram for the synchronous controller.

1. Resting state. The controller is idle until an input event arrives. The machine continues in this state while the input queue does not activate the signal Data_in to indicate that a new event is available. Once detected a change in this signal, it goes to the calculation state. The resting state also generates the global forgetting signal. The controller includes a 20-bit programmable counter which activates a signal to indicate when it has finished the programmed count. To avoid collisions between the forgetting signals and those for enabling the kernel addition, the finite-state machine generates the forgetting signal when it detects the end of count in the resting state, restarting the counter.

2. Calculation state. This is where the machine must calculate the position of the projection field inside the array and decide the corresponding operations. First, it calculates the horizontal position of the kernel, so if it is completely outside of the address space, it goes back to the resting state. If the horizontal coordinates of the projection field are partially inside the array, then it calculates the horizontal shift that must be applied to place the kernel in those coordinates, activating the correct signals. After that, it calculates the vertical limits of the projection field in order to decide the next state. If it is partially inside the address space, but the part which is outside is the upper one, the next state is reading state 1. If it is completely inside the array, it goes to reading state 2. If it is partially inside, but the part which is outside is the lower one, it goes to reading state 3. Finally, it the whole kernel is outside of the vertical limits of the address space, it checks for a new event. If a new event is available, it continues in the same state with it. Otherwise, it goes back to the resting state. For chip Conv2, the controller must look at the kernel number that has to be applied to implement the multikernel system. After looking at the kernel number, it accesses to the information about the position of that kernel inside the RAM and the pixel neighborhood where it has to be added. The operations performed to implement the addition are equivalent to version Conv1, although the information used for the calculations is relative to a different kernel for each event.
3. Reading state 1. The reading of a single RAM row is enabled. For each input event, the machine will go through this state as many times as kernel rows it has to add. After enabling a single row reading, it goes to writing state 1.
4. Writing state 1. The Enable signal of a single row in activated to store the results of the addition of the RAM row into the pixels. It also checks if more kernel rows are left to be written. If the answer is positive, it goes back to reading state 1. Otherwise, it goes to the resting state to wait for a new incoming event.
5. Reading state 2. It is equivalent to reading state 1, but with the whole kernel vertically inside the address space. It activates the reading of a single RAM row before going to writing state 2.
6. Writing state 2. As in previous case, it implements the equivalent operation as writing state 1 when the kernel is inside the vertical limits of the

pixel array. If more rows are left to be added, if goes back to reading state 2. Otherwise, it goes to the resting state.

7. Reading state 3. When the projection field is partially inside the array, but the lower limit is outside, this state enables the reading of a single RAM row before going to the writing state 3.
8. Writing state 3. As previous states, it enables the addition of a single row into the array. It goes back to reading state 3 if more rows must still be added, or it goes to the resting state if there are no more rows left.

The finite-state machine is almost identical for Conv1 and Conv2. The main difference is related to the size of the pixel array used for the calculations, 32×32 in the first case and 64×64 in the second. Also, the controller in Conv2 implements the multikernel operation, so it must includes some small changes. First, the input event must carry information relative to the kernel number that has to be used to process it, so that the finite-state machine will access to the configuration parameters that indicate the position of the corresponding kernel inside the RAM. In Conv1, as a single kernel can be programmed in the RAM, the positions outside of the kernel (when the kernel is smaller than the RAM) are filled with zeros, so that they have no influence when added to the pixels. However, in Conv2 the positions outside of the selected kernel can be written with non-zero values. To avoid undesired addition operations, the machine must enable only the correct columns of the RAM (those that include the selected kernel). To achieve this, it activates the Block_col signals shown in Fig. 5.2 for the columns outside of the kernel.

For processing a single input event, the finite-state machine first needs 4 clock cycles to capture the event, write it in the queue, read it from the queue and check if the projection field is inside the address space of the chip. After that, it needs 2 more clock cycles for each row it must add to the pixels (1 cycle for reading and 1 cycle for writing). Therefore, the time necessary to process an event can be expressed as $(4 + 2 \times n_k) \times T_{clk}$, where n_k indicates the number of kernel rows and T_{clk} the clock period. For a maximum clock frequency of 200MHz, the minimum time between input events can go from $30ns$ for a minimum kernel of size 1×1 to $340ns$ for a maximum kernel of size 32×32.

The synchronous controller also handles the forgetting mechanism in parallel to the finite-state machine. It includes a 20-bit configurable counter which generates a forgetting signal when it reaches the programmed value. The finite-state machine checks for this signal when it is in the resting state, and sends it to all the pixels in the array, making them decrement their state if it was positive, or increment if it was negative. The forgetting period (time between two global forgetting pulses) is programmed through a 20-bit digital word which indicates the number of clock cycles n_{olv}. Then, the forgetting period will be $n_{olv} \times T_{clk}$, being the maximum possible value $(2^{20} - 1) \times T_{clk} = 1048575 \times T_{clk}$.

5.2.4. Synchronizers

AER is an asynchronous protocol. Consequently, the convolution chip receives asynchronous events that are handled by the synchronous controller. As a consequence, synchronization errors may be produced due to metastability [92], producing undesirable results. A synchronization error is produced when the latch clock signal is triggered during a transient of the latch input. This produces metastability in the latch which can be propagated forward into the sequential hierarchy of latches, thus producing incorrect computations and spurious output events. Let us call the Probability of Error the rate between the number of spurious output events over the total number of output events. In our chip we observe a 4.62% Probability of Error, which is extremely high. To reduce this probability, synchronizers are required. We tested three configurations. The synchronizers were included in the Rqst_in line path, as shown in Fig. 5.7. This allows to characterize each implementation and compare their impact with respect to the absence of synchronizer. In general, the probability of error is inversely and exponentially proportional to the number of bistables. However, each bistable introduces extra delay, but without reducing event throughput.

To characterize these synchronizers, a set of measurements were performed whose results are shown in Tabla 5.1. To measure the probability of error, a train of input events was sent, while the convolution chip was programmed to generate one output event after each input event. This way, an error was detected when an unexpected output event was generated.

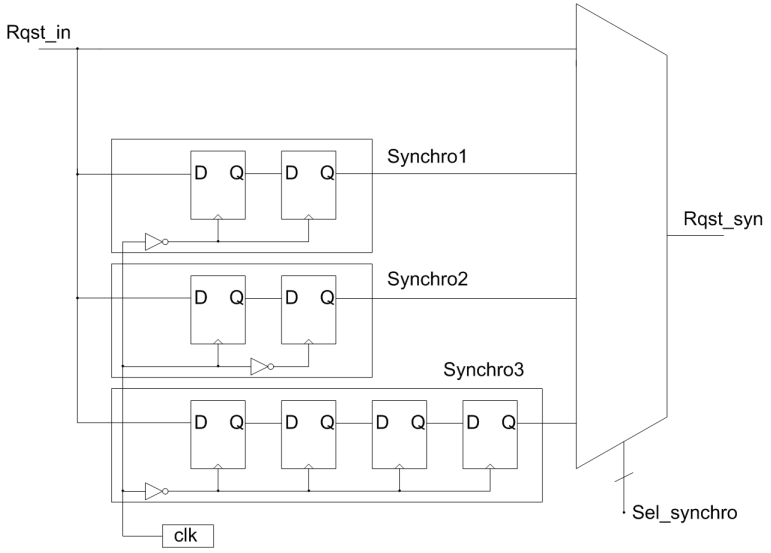


FIGURA 5.7. Configurable synchronization block at the input of the convolution chip.

TABLA 5.1. Characterization of synchronizers.

Synchronizer	Probability of error (in %)	Mean delay (in clock cycles)
None	4.62	0.89
Synchro1	21×10^{-6}	2.23
Synchro2	44×10^{-6}	1.84
Synchro3	4×10^{-10}	4.23

Tabla 5.1 shows that without synchronizers we measured a 4.62% Probability of Error. In this case, the mean delay measured between Rqst_in and Ack_in was 0.89 clock cycles. The configuration “Synchro1” inserts two cascaded bistables in the Rqst_in path, triggered both by the controller’s clock. This configuration reduces error probability from 4.62% to 21×10^{-6} %, while introducing an average extra delay of $2.23 - 0.89 = 1.34$ clock cycles in the response between Ack_in and

Rqst_in. Configuration “Synchro2” is a slight modification of “Synchro1” where now the two bistables are triggered at different clock phases. Error probability degrades slightly to 44×10^{-6}, although delay is improved to $1.84 - 0.89 = 0.95$. Configuration “Synchro3” uses 4 cascaded bistables triggered by the same inverted clock. This configuration introduces an average extra delay of $4.23 - 0.89 = 3.34$. For this configuration, we were not able to detect any single error after having the system running for several days. Consequently, we estimated this probability with the expression of MTBF (Mean Time Between Failure) proposed in [92], which is the inverse of the probability of error. This revealed an error probability of 4×10^{-10} %, which means that when sending AER input events at a rate of 1Meps (10^6 events per second), we would need about 70 hours to see one output spurious event.

In our convolution chip, any of the three proposed synchronizers can be selected (or none of them) through a 2-bit configuration control word.

5.2.5. Configuration registers

The convolution chips need several configuration parameters that can be used to control their behavior. These parameters are written in the configuration registers. First, we describe the block used for the Conv1 version.

Fig. 5.8 shows the structure of the configuration register used for the first version of the convolution chip. It is a 128-bit shift register which is loaded serially, and once all the bits are introduced a Copy signal is activated so that their values are stored and connected to the rest of the circuit. The configuration parameters list is described in Tabla 5.2. Both the parameters that indicate the coordinates of the chip and those that indicate the kernel dimensions inside the RAM, are used by the finite-state machine to calculate the operations which must be performed depending on the address of the input event. Parameter n_{olv} is also used by the controller to generate the forgetting pulses. Both the synchronizer and clock selection bits are used to control their corresponding blocks. The 64 bits that control the arbitration’s pull-downs are described in Section 5.6, when talking about the AER generator.

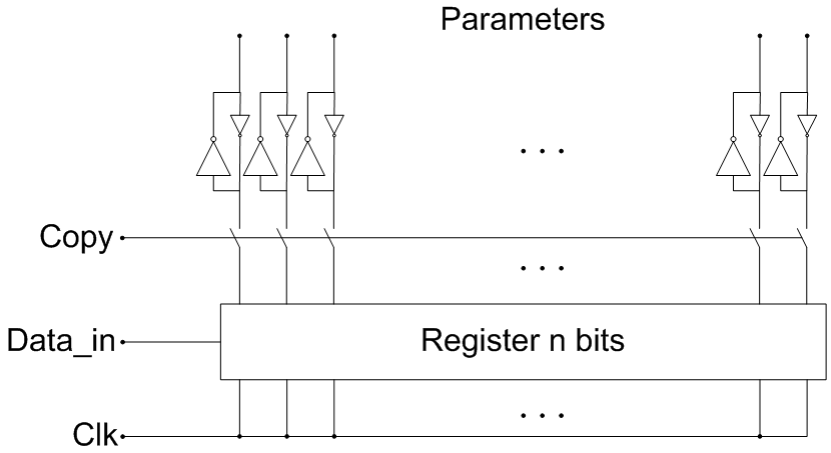


FIGURA 5.8. Configuration register structure used in convolution chip Conv1.

TABLA 5.2. Configuration parameters list

Parameter	Number of bits	Meaning
$(x, y)_{min}$	14	Coordinate of the upper left pixel within the 128x128 input address space
$(x, y)_{max}$	14	Coordinate of the lower right pixel within the 128x128 input address space
(p, q)	10	Kernel dimensions inside the RAM
n_{forg}	20	Clock cycles specified between two global forgetting pulses
sel_{clk}	1	Selection of either internal or external clock
sel_{acc}	3	Selection of the accumulator limit
sel_{pd}	64	Configuration of W/L for row communication pull-downs
sel_{syn}	2	Selection of synchronizer

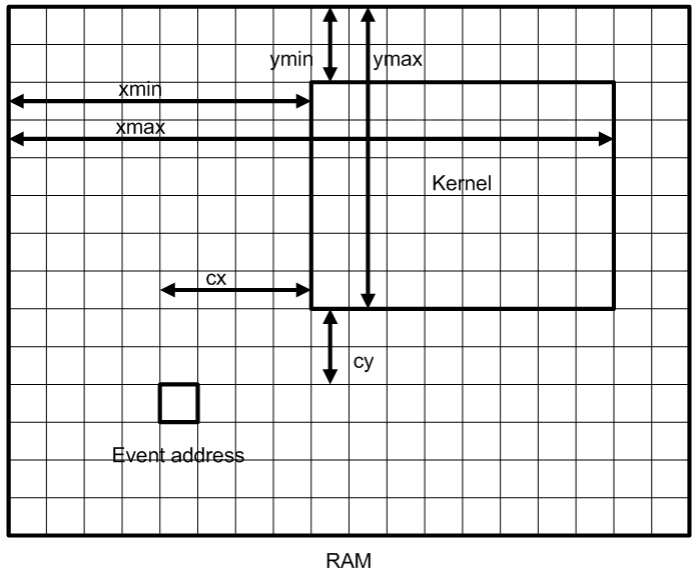


FIGURA 5.9. Definition of parameters for each kernel.

In the advanced version of the convolution chip Conv2, the configuration block is different, specially because of the multikernel system. As was described previously, the system is based on the possibility of programming up to 32 different kernels in the RAM, so the finite-state machine needs some information about their position in order to be able to access to them. For this reason, a memory array was designed with 34 rows of 32 bits each. Each of the first 32 rows stores information about the programmed kernel, while the last rows include the rest of parameters.

The 6 parameters necessary to describe each kernel are represented in Fig. 5.9. 4 of these parameters indicate the kernel position inside the RAM, through the coordinates of the upper left corner (x_{min}, y_{min}) and the lower right one (x_{max}, y_{max}). Each of these parameters are represented with 5 bits. The other parameters indicate the coordinates of the kernel's application center. In version Conv1 these parameters were not necessary, as the kernel was always applied around its own center. However, for some applications like character recognition the kernels are applied around a pixel different

from its center, in order to detect some features in the periphery of the character [74]. For this reason, two parameters are included (c_x, c_y) that indicate the position of the kernel relative to the input event address. These parameters can be either positive or negative, so they are coded in 2's complement with 6 bits (5 + sign), reaching 32 bits to define the whole characteristics of a single kernel, which is the length of each row in the memory array.

Therefore, each time an input event arrives, the kernel number indicated by the event is used to select a single row in the memory array, so that the finite-state machine gets the information from that row. Then, the horizontal shift is calculated, and the RAM positions that must be read are activated. These parameters are also used to activate the Block_col signals with values below x_{min} and above x_{max}.

The lower two rows in the memory array are used to store the information relative to the convolution chip configuration. Row 33 includes the coordinates of the address space (i_{min}, j_{min}), (i_{max}, j_{max}), 4 parameters with 8 bits each. Last row includes 1 bit for selecting the pixels accumulator threshold, 2 bits for selecting the inhibition of positive or negative events, 1 bit for enabling the forgetting mechanism and 20 bits to establish the frequency of the forgetting signal.

5.3. Static RAM

The RAM is the block where the convolution kernel is stored (a single kernel in Conv1 or up to 32 in Conv2), so it imposes a limit over the kernel size. In Conv1 the kernel can be as large as 32×32. In Conv2, the size of the kernels depends on the number of them. If a single kernel is programmed, the maximum size is also 32×32, while if several kernels are used they must be smaller so that the whole of them can be written in the 32×32 positions memory. Anyway, the RAM has 32×32 positions in both versions. The only difference between them is that Conv1 uses 6-bit data, and Conv2 uses 4 bits.

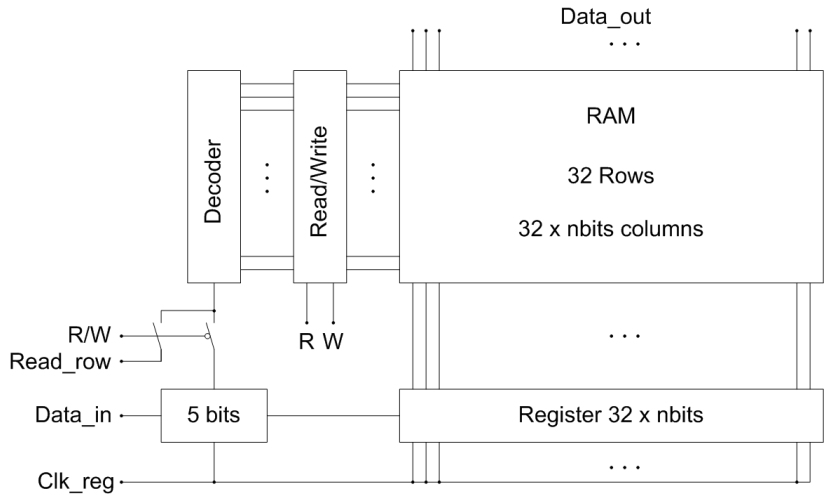


FIGURA 5.10. Scheme of the whole RAM block.

Fig. 5.10 shows the scheme of the whole RAM block, with the read/write circuitry included. Below the RAM, it is represented the shift register with $32 \times n$ bits, being n the number of bits per data (6 for Conv1 and 4 for Conv2). This register has 5 extra bits to code the row address. This way, the RAM is written by sending serially flows of $(32 \times n) + 5$ bits, while the decoder enables the corresponding row so that the data can be stored in the right place. The reading process is very similar, although the 5 bits used to select the row are not in the shift register, but they are provided by the controller, which enables the reading process and indicate the right row.

In Conv2, the RAM includes an extra block at the data output to implement the multikernel capability, as the memory positions outside of the selected kernel should not affect the pixels. Therefore, the finite-state machine generates a Block_col signal for each one of the 32 RAM columns, in order to block the digital words which do not belong to the kernel. These columns are block by the circuit illustrated in Fig. 5.11. The circuit shown in this figure is designed to block a single data (formed by 4 bits in the case of Conv2), so the whole circuit includes 32 cells like this one. The whole circuit receives 32 signals $\overline{\text{Block\_col\_i}}$, which will be at high value for

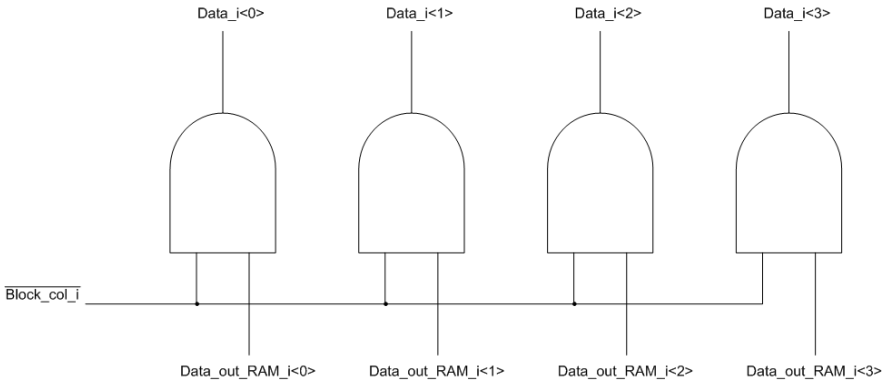


FIGURA 5.11. Circuit design to block the RAM columns to implement multikernel capability.

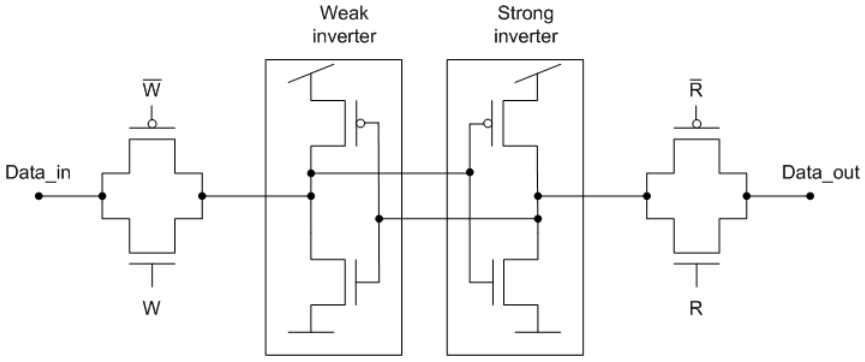


FIGURA 5.12. Basic RAM cell.

$x_{min} \leq i \leq x_{max}$ and at low value otherwise (x_{min} and x_{max} correspond to the kernel position inside the RAM as indicated in Fig. 5.9). Then, signals Data_i<0:3> will be 0 for all the columns outside of the kernel.

Fig. 5.12 shows the schematic of a single RAM cell, designed to achieve a reading time smaller than 2ns. Any extra delay would have an influence on the time spent to process each row, so it would limit the maximum clock frequency. Fig. 5.13 shows the layout of the RAM cell of Fig. 5.12.

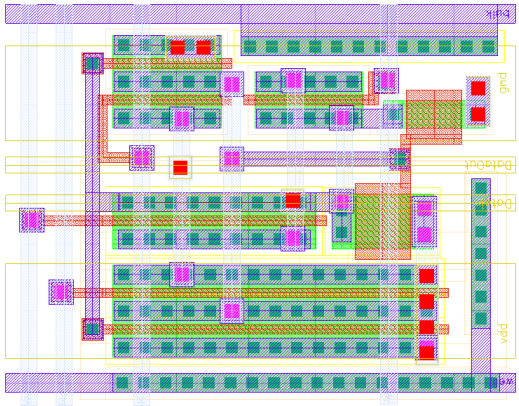


FIGURA 5.13. Layout of a single RAM cell.

5.4. 2's complement inverter

The convolution chip receives signed input AER events, and it generates signed output AER events, indicating if a single pixel has reached the positive or the negative threshold. Then, when a negative input events is received, the chip has to process it adding the inverted kernel to the selected neighborhood, so a block to invert the kernel is necessary.

Two different alternatives were proposed. The first one consists on writing in the RAM both the kernel and its inverted version. This would simplify the chip operation, as it only would have to access to a different part of memory to process a negative event. However, the main drawback is the limitation for the kernel size, as twice its size was needed. The second alternative consists on adding an extra block which implements the inversion operation when the data is read. This alternative adds no limitation for the kernel size, although it adds an extra delay to the reading operation. After designing the inverter block to minimize this delay, an estimation of $400ps$ was obtained from Spectre simulation, while the additional area consumed by this circuit is $3750 \times 40\mu m^2$ (the RAM area is much bigger, $3750 \times 480\mu m^2$). Then the second alternative was chosen.

Then, 32 inverter blocks were needed, one for each RAM position. For Conv1, the proposed block includes 6 combinational circuits (one for each output), while Conv2 needs only 4. For this reason, the one used in Conv1 will be described. The input for each block is a 6-bit word $(i_0, \ldots, i_5)$, being i_5 the most significant bit (the sign). They also receive an input Sel which enables the inversion operation when the input event is negative. The output of each block is a 6-bit word $(o_0, \ldots, o_5)$, where o_5 is the most significant bit. The truth tables that express all the possible combinations of inputs provide the logic expressions for the 6 output signals indicated in equations (5.1)-(5.6).

$$o_0 = i_0 \quad \textbf{(EQ 5.1)}$$

$$o_1 = \overline{(\bar{i}_1 \vee sel) \wedge (i_0 \vee \bar{i}_1) \wedge (i_1 \vee \bar{i}_0 \vee \overline{sel})} \quad \textbf{(EQ 5.2)}$$

$$o_2 = \overline{(\bar{i}_2 \vee sel) \wedge (i_2 \vee \bar{i}_0 \vee \overline{sel}) \wedge (i_2 \vee \bar{i}_1 \vee \overline{sel}) \wedge (\bar{i}_2 \vee i_1 \vee i_0)} \quad \textbf{(EQ 5.3)}$$

$$o_3 = \overline{(\bar{i}_3 \vee sel) \wedge (i_3 \vee \bar{i}_0 \vee \overline{sel}) \wedge (i_3 \vee \bar{i}_1 \vee \overline{sel}) \wedge} \quad \textbf{(EQ 5.4)}$$

$$\overline{\wedge (i_3 \vee \bar{i}_2 \vee \overline{sel}) \wedge (\bar{i}_3 \vee i_2 \vee i_1 \vee i_0)}$$

$$o_4 = \overline{(\bar{i}_4 \vee sel) \wedge (i_4 \vee \bar{i}_0 \vee \overline{sel}) \wedge (i_4 \vee \bar{i}_1 \vee \overline{sel}) \wedge (i_4 \vee \bar{i}_2 \vee \overline{sel}) \wedge} \quad \textbf{(EQ 5.5)}$$

$$\overline{\wedge (i_4 \vee \bar{i}_3 \vee \overline{sel}) \wedge (\bar{i}_4 \vee i_3 \vee i_2 \vee i_1 \vee i_0)}$$

$$o_5 = \overline{(\bar{i}_5 \vee sel) \wedge (i_5 \vee \bar{i}_0 \vee \overline{sel}) \wedge (i_5 \vee \bar{i}_1 \vee \overline{sel}) \wedge (i_5 \vee \bar{i}_2 \vee \overline{sel}) \wedge} \quad \textbf{(EQ 5.6)}$$

$$\overline{\wedge (i_5 \vee \bar{i}_3 \vee \overline{sel}) \wedge (i_5 \vee \bar{i}_4 \vee \overline{sel}) \wedge (\bar{i}_5 \vee i_4 \vee i_3 \vee i_2 \vee i_1 \vee i_0)}$$

These expressions can be translated into the corresponding logic gates, represented in Fig. 5.14-Fig. 5.18, where the schematics are shown on the left and the layout on the right. They correspond to the cells which calculate

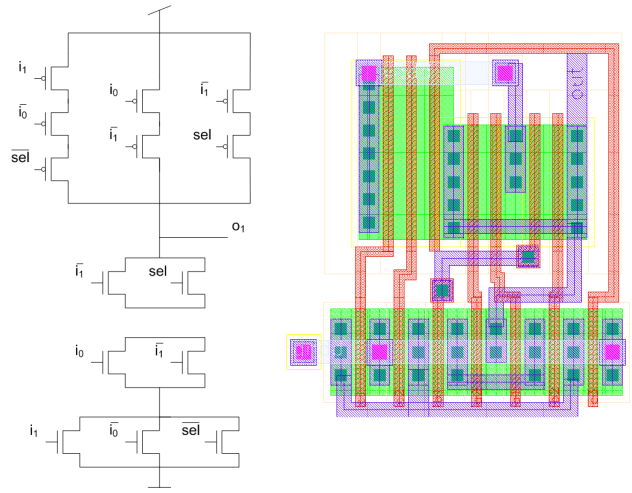


FIGURA 5.14. Schematic and layout of the combinational circuit to calculate bit o_1 in the 2's complement inversion block.

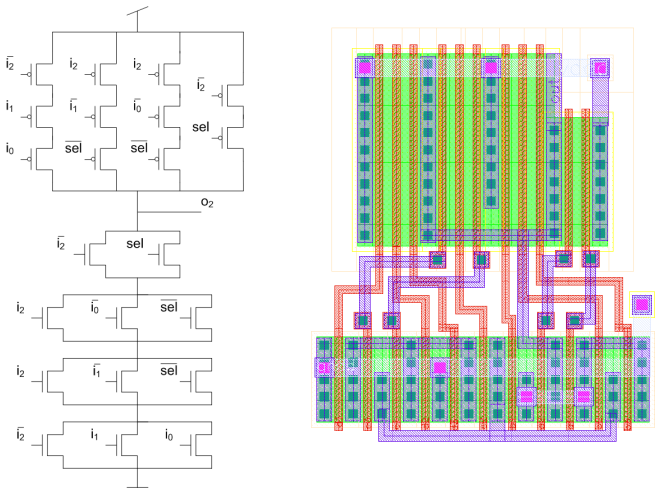


FIGURA 5.15. Schematic and layout of the combinational circuit to calculate bit o_2 in the 2's complement inversion block.

output bits o_1-o_5 (equation (5.1) indicates that no logic gate is necessary to calculate o_0).

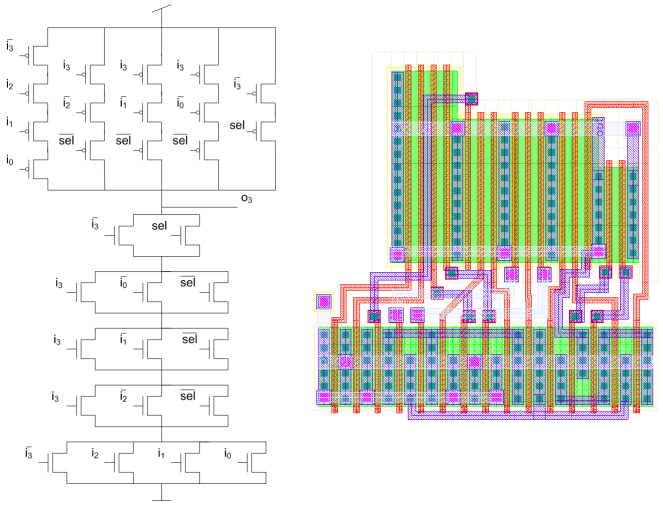


FIGURA 5.16. Schematic and layout of the combinational circuit to calculate bit o_3 in the 2's complement inversion block.

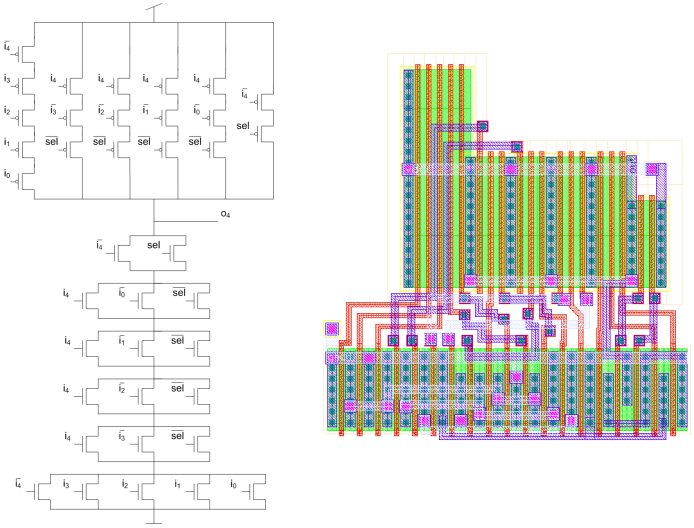


FIGURA 5.17. Schematic and layout of the combinational circuit to calculate bit o_4 in the 2's complement inversion block.

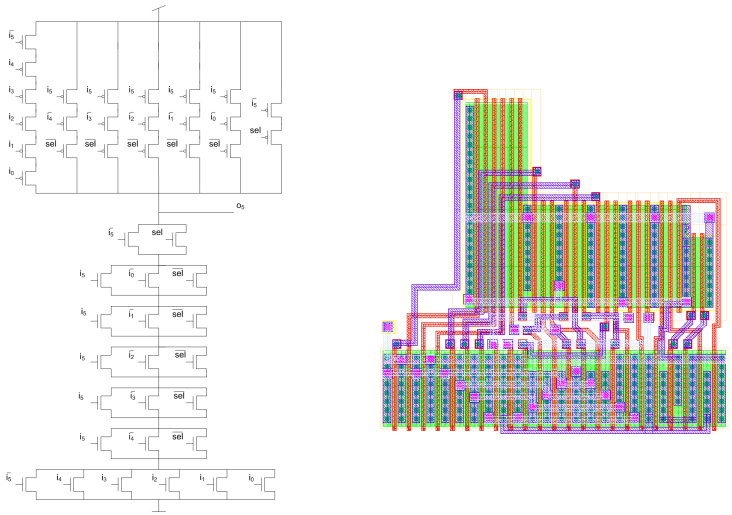


FIGURA 5.18. Schematic and layout of the combinational circuit to calculate bit o_5 in the 2's complement inversion block.

5.5. Horizontal shift block

Once the kernel data are read from the RAM, they must be shifted horizontally so that they can be centered around the right pixel in the array. This operation is performed by the horizontal shift block.

This block consists on a 2-D array of tri-state buffers which are controlled by shift signals Δx and right/left. This signals are generated by the finite-state machine, and two decoders included in the controller enable the right shift signal. Fig. 5.19 illustrates the structure of this block. When the state machine is processing an event, it calculates the difference between the input event address and the position of the kernel in the RAM, in order to obtain the number of positions the kernel has to be shifted (Δx) to perform the convolution operation. By activating the correct signals, the horizontal shift block enables a path between the RAM columns and the pixel array.

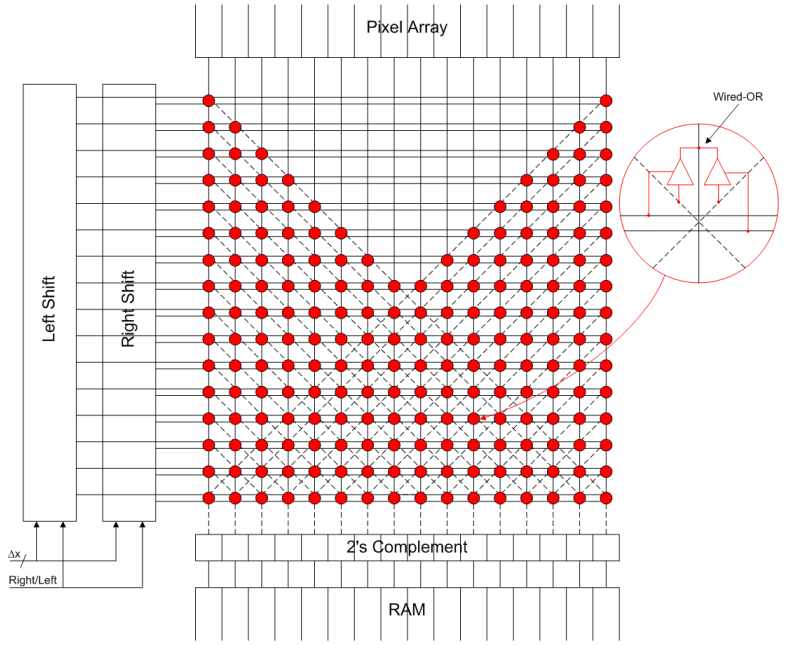


FIGURA 5.19. Scheme of the horizontal shift block.

Each element inside this 2-D array includes two tri-state buffers controlled by shift signals. The inputs of these buffers are the RAM data corresponding to the left and right positions related to their control shift signals. Therefore, the inputs of the whole block are the RAM outputs (after being processed by the 2's complement inverter) while the outputs of this block are these RAM outputs horizontally shifted to reach the inputs of the pixel array properly centered. The outputs have pull-down transistors to set them to 0 when the columns are not selected.

The only difference in this block for versions Conv1 and Conv2 is its size. Although in both cases the input is formed by 32 RAM positions, they are 6-bit positions for Conv1 and 4-bit positions for Conv2. Also, the output of the block has 32 6-bit positions for Conv1 (to be connected to a 32×32 pixels array) while it has 64 4-bit positions for Conv2, as the array has 64×64 pixels. This is a simple scaling of the circuit. However, this scaling increases the delay introduced by this block, which starts to be critical for

reading the data from the RAM. For future systems with larger arrays, a new architecture will be necessary.

In Conv1, each input of the horizontal shift block is the input of 32 tri-state buffers, so only one of them will be enabled for a shift signal. To reduce the fan-out of the input signals, a tree of inverters if introduced to reduce each node's impedance. However, in Conv2 each input bit reaches 64 tri-state buffers, so the trees of inverters must be re-designed and increased. Then, the RAM data must go through a larger number of gates before reaching the pixels, and also longer stripes and with larger resistance. As a consequence, the propagation delay for the data is increased, limiting the frequency of the system clock.

5.6. AER generator

In Section 5.1, a division was made between the input and output stages of the convolution chip. The input stage receives the input events and process them, so that the previously programmed kernel is added around the selected pixels of the array. This stage has been described in sections 5.2-5.5. On the other hand, the output stage must handle the events generated by the convolution pixels and transmit them off-chip. The block which performs this operation is called AER generator.

The structure of this block is described in the literature [94], and the block diagram is shown in Fig. 5.20. Its behavior is as follows: each time a single pixel reaches its saturation limit, it activates its Rqst through a common node for all the pixels in the same row, implementing a wired-OR (signal Rqst_row in Fig. 4.14). Once the row arbiter acknowledges, all the pixels in the same row which have previously reached their limits activate the corresponding column request, indicating the sign of the limit reached. These column requests are stored in the periphery, so that the row arbiter can acknowledge another Rqst_row while all the events generated by the previous row are transmitted in burst mode.

The main critical issue for this structure is the common row request line. This node is shared by all the pixels in the same row, so any of them

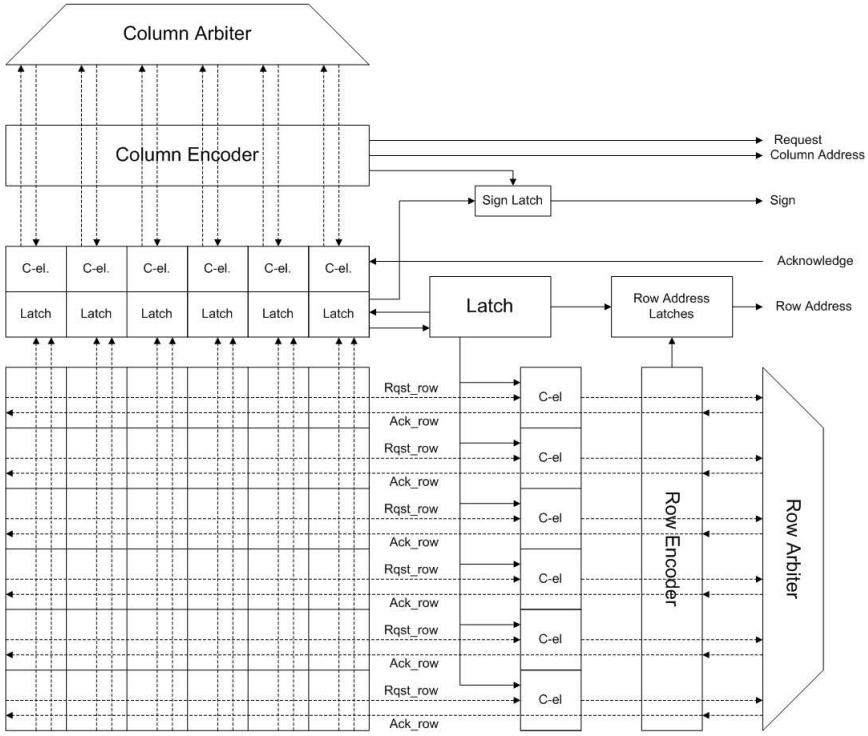


FIGURA 5.20. Block diagram of the AER-out generator.

must be able to activate it on its own, while a pull-down transistor in the periphery must be able to deactivate it. This way, the dimensions of these transistors are quite critical, affecting to the delays for activating and deactivating this line. For this reason, both convolution chips presented in this work include some improvements.

In first version Conv1, an alternative is proposed to increase the programmability of these transistors, following the schematic in Fig. 5.21. The timing in the communication between the pixels and the row arbiter is quite critical for optimum behavior, but unfortunately this timing relies heavily on parasitics and changes significantly from chip to chip, and even between different rows of the same chip. To overcome this, we included a row-wise coarse calibration for the row pull-down transistors. Fig. 5.21 shows that the basic communication circuitry for each row includes 32 pMOS transistors

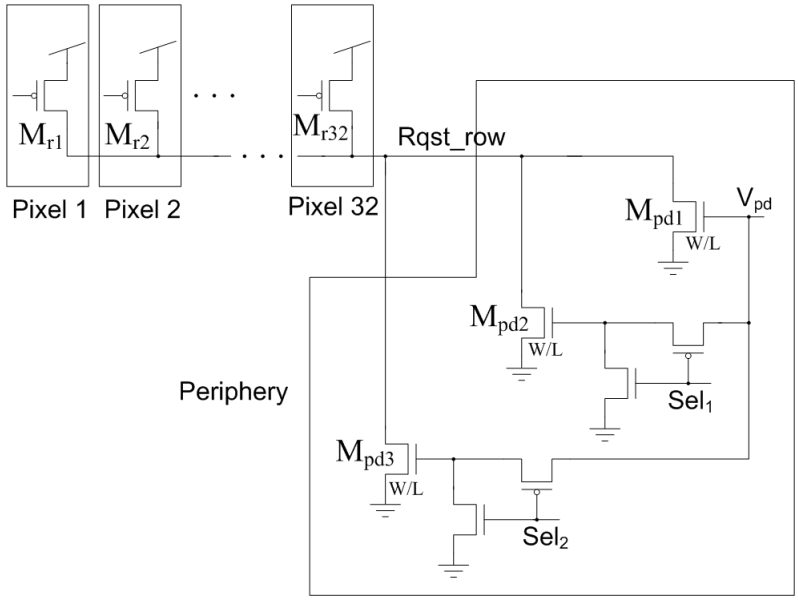


FIGURA 5.21. Schematic of the communication circuit between the pixels and the row arbiter in Conv1.

M_{r1}, …, M_{r32} (one per pixel) and one nMOS pull-down transistor M_{pd1} at the periphery. This way, the size W/L of M_{pd1} has to be designed to make sure that 1) any single pixel is able to activate the node Rqst_row, and 2) the pull-down is able to set Rqst_row low fast enough. If the pull-down is too strong, the gate voltage V_{pd} (which by default is connected to V_{dd}) can be decreased to reduce the current that flows through M_{pd1}. On the other hand, if the pull-down is too weak, transistors M_{pd2} and/or M_{pd3} can be activated through calibration switches Sel_1 and/or Sel_2. If both switches are off, the pull-down size is W/L; if one of them is on, the size becomes 2W/L; and if both switches are one, it becomes 3W/L. This row-wise calibration, together with the possibility of globally tuning V_{pd} allows to optimize the timing and compensate for process and in-chip variations.

On the other hand, the second version of the convolution chip Conv2 includes a scaled AER generator adapted to a 64×64 pixel array, and it also implements a different scheme for row communication to improve the previous one [43]. The new structure is shown in Fig. 5.22. The pull-down

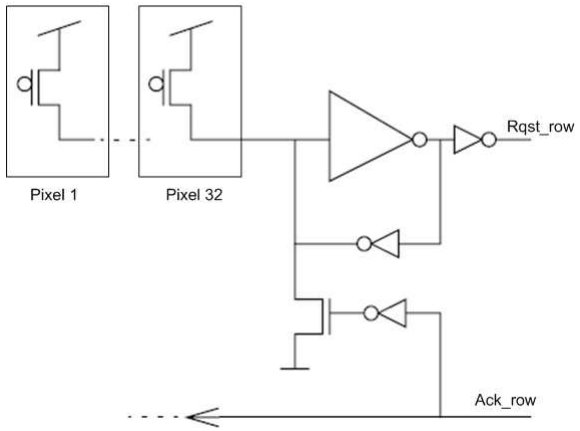


FIGURA 5.22. Schematic of the communication circuit between the pixels and the row arbiter in Conv2.

transistor which resets the common node to all the pixels in the same row is controlled by an Ack_row signal generated by the arbiter (while in previous version, it was controlled by a bias voltage). This way, this pull-down transistor is not active when the pixels are sending a request, so they do not have to fight against the pull-down to activate the line. The Rqst_row pulse is stored by the asymmetric latch, so that the pull-down transistor do not have to fight against the strong inverter, but the weak one. Therefore, the row communication is more robust and faster.

CAPÍTULO 6

Experimental results

6.1. Introduction

In this chapter, the experimental results obtained with both convolution chips are shown. First, Section 6.2 illustrates some tests performed at the pixel level with a first prototype based on a 2×2 pixels array. These tests were very useful to validate the initial version of the pixel before designing the whole convolution chip.

In Section 6.3, the basic tools used for testing the convolution chips are described. These tools are several PCBs which handle AER events and the software environment developed to control the chips and perform the desired tests.

Finally, Sections 6.4 and 6.5 describe in detail the tests performed with convolution chips Conv1 and Conv2, respectively, showing the obtained results. An error in the implementation of the synchronous controller of Conv2 made it impossible to perform all the tests. This error has been detected and a new version of the chip is in fabrication process at the moment of elaborating this thesis. Therefor, only the results related to the chip characterization are included, as they are not affected by this error.

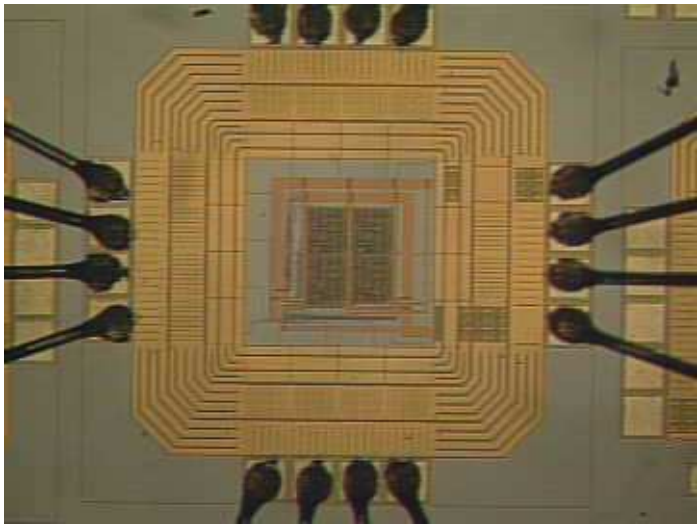


FIGURA 6.1. Photograph of the 2x2 pixels prototype.

6.2. 2x2 pixels prototype

As a first step before designing the complete initial version of the convolution chip, a small array with 2×2 pixels was fabricated with the AMS $0.35\mu m$ technology to validate the behavior of the digital pixel. Fig. 6.1 shows a photograph of the fabricated circuit.

This prototype used a single kernel word, which was shared by the four pixels. To avoid a large number of pads for the circuit, a shift register was used to program the kernel value and the accumulator threshold (both of them are common for the four pixels), and the signal Sel_row is used to select one of the rows, as Enable and Ack signals are common for both. This configuration is described in Fig. 6.2. As the tests were quite simple, Agilent 82000 was used for them. Later, for tests with asynchronous stimuli, a specific AER infrastructure was developed.

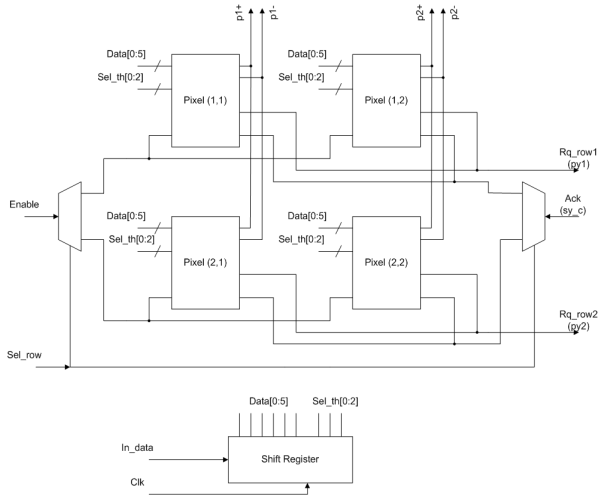
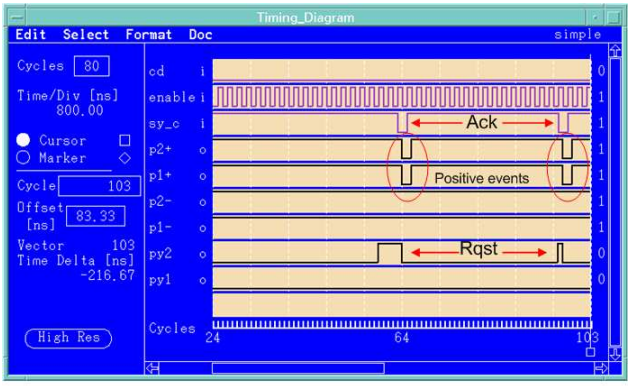


FIGURA 6.2. Test configuration used to control the 2x2 pixels prototype.

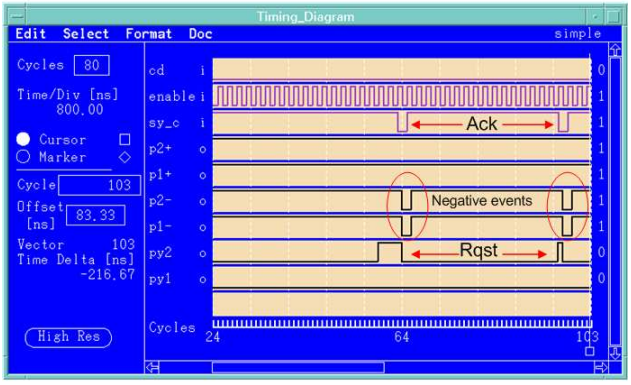
6.2.1. Pixel characterization

To make tests with this prototype, the following method is followed: first, the shift register is programmed selecting a limit for the accumulator and writing a kernel value; then, one of the rows is selected through signal Sel_row. Once configured, Enable signals can be sent to check that the pixel activates Rqst_row when it reaches the programmed limit. Signal Ack is programmed externally to acknowledge, so that we can check the sign of the produced events by monitoring signals p1+, p1-, p2+ and p2-.

Fig. 6.3 shows the results obtained for a first test. In this case, the accumulator limit was set as $2^9 = 512$ for positive values and $-2^9 - 1 = -513$ for negative values. In this prototype the possible configurable values were not those described in the pixel version which was integrated in Conv1. The programmed kernel value was 31 (the maximum value with 6 bits in 2‘s complement). With this configuration, second row was selected, and input Enable pulses were sent to the circuit, while monitoring signal Rq_row2. This signal generates an event after accumulating 17 input events



a) Positive input word

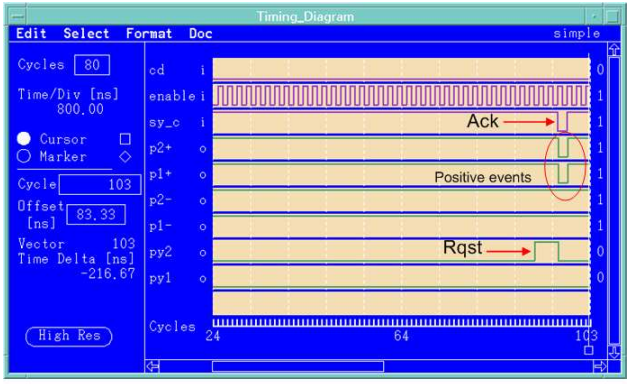


b) Negative input word

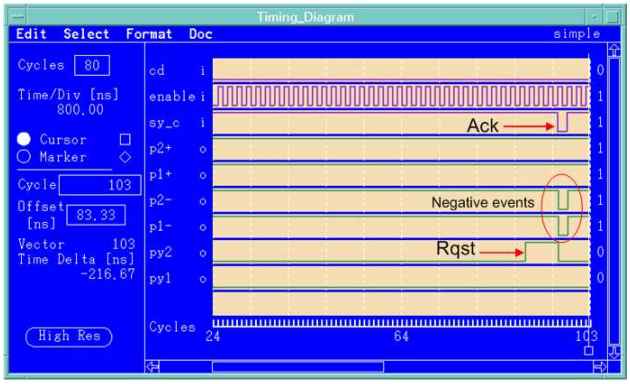
FIGURA 6.3. Test results for the 2x2 prototype with accumulator limit of 512 for positive values and -513 for negative values.

($16 \times 31 = 496 < 512$ and $17 \times 31 = 527 \geq 512$). When the pixels in row 2 receive the Ack signal, both of them activate signals p1+ and p2+, respectively, indicating the sign of the events. This results are shown in Fig. 6.3. In general, these results show Rqst pulses with different widths. Although the activation of Rqst signal depends only on the accumulator value, its deactivation is produced when signal Ack is activated externally. For this reason, the duration of the Rqst pulse is variable.

The lower picture in Fig. 6.3 shows the results after changing the kernel value, programming the negative value -31. The accumulator limit is



a) Positive input word



b) Negative input word

FIGURA 6.4. Test results for the 2x2 prototype with accumulator limit of 1024 for positive values and -1025 for negative values.

reached after receiving 17 input events ($16 \times (-31) = -493 > -513$ and $17 \times (-31) = -527 \leq -513$). When the pixels receive the Ack signal, both of them activate the signals p1- and p2-, as the negative limit has been reached.

In order to check the behavior of the pixels with a different accumulator threshold, it was programmed as $2^{10} = 1024$ for positive values and $-2^{10} - 1 = -1025$ for negative values. After repeating the same tests for this configuration, the results shown in Fig. 6.4 were obtained.

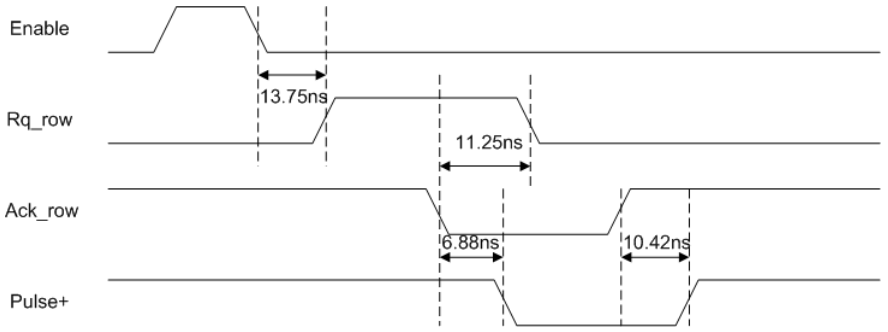


FIGURA 6.5. Timing information of the 2x2 prototype.

Upper picture in Fig. 6.4 represents what happens when the kernel has been programmed as 31. The pixels accumulate 34 input events before activating Rqst signal ($33 \times 31 = 1023 < 1024$). Then, both pixels activate p1+ and p2+ when they receive the Ack signal, indicating that the output event is positive. Lower picture in Fig. 6.4 shows the equivalent results obtained for the negative kernel value -31. The pixels activate Rqst after receiving 34 input events ($33 \times (-31) = -1023 > -1025$). It is also checked how both pixels activate p1- and p2- to indicate that they have reached the negative limit of the accumulator.

The timing information measured for this prototype is represented in Fig. 6.5.

6.3. AER infrastructure for asynchronous tests

Several configurable PCBs [48], [96], [97], [98] have been used to perform the tests presented in this thesis for the convolution chips. Fig. 6.6 shows an example of the AER infrastructure designed for these tests, with the following blocks: an AER board for the convolution chip (3) with a configuration board (4); two USB-AER boards, one of the configured as data-player (1) and the other one as data-logger (6); and finally, two Splitter-Merger boards, one of them configured as Splitter (2), and the other one as

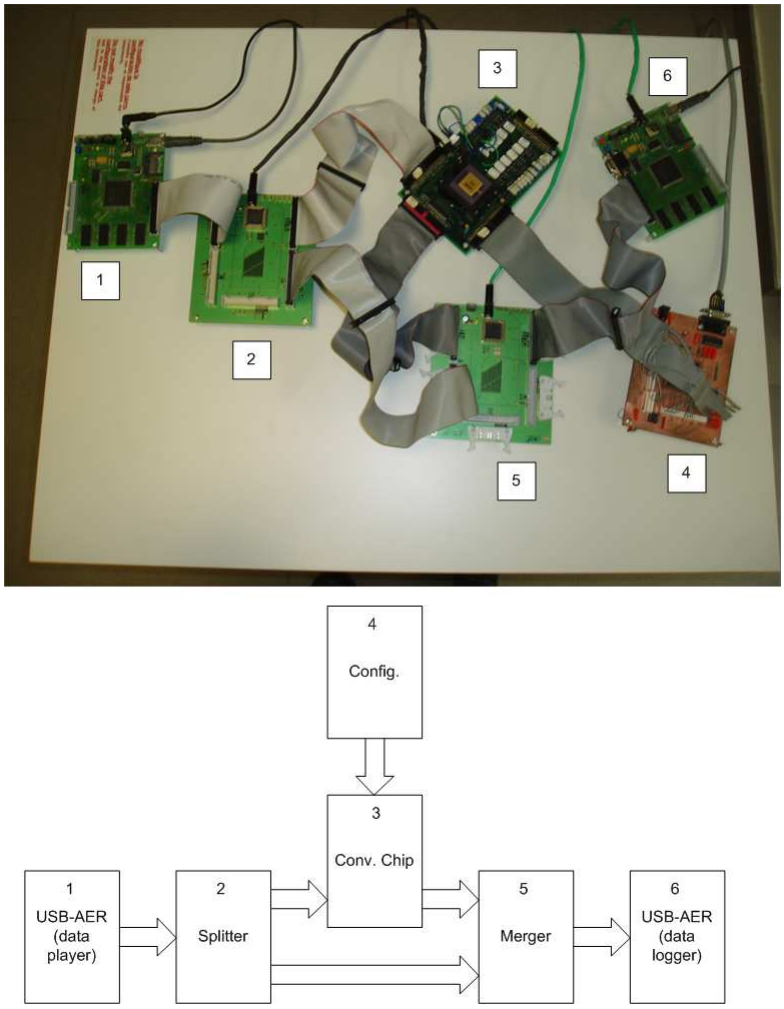


FIGURA 6.6. AER test infrastructure. Photograph of the different boards above, and equivalent scheme below.

Merger (5). This Section includes a brief description of these boards, and the software environment developed to control the convolution chip.

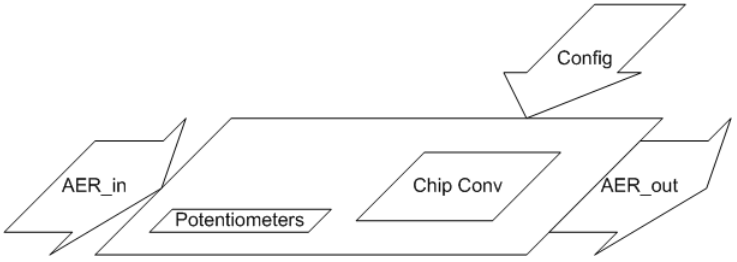


FIGURA 6.7. Scheme of the convolution board.

6.3.1. AER Board

This is a board where the convolution chip is introduced, with a PGA-100 package. As is shown in Fig. 6.7, this board includes some potentiometers that are used to control bias voltages inside the chip (to set the clock frequency, and to control pull-ups and pull-downs inside the AER generator). The board also receives a configuration bus as input, and through this bus the configuration registers and the kernel are programmed. It also includes two AER buses (input and output) that are connected to the convolution chip. It is board 3 in Fig. 6.6.

6.3.2. USB-AER board

This board is able to handle AER traffic, and can be configured in two different ways: as data-player or data-logger. This is selected by programming the correct firmware from the PC [48].

On the top of Fig. 6.8 it is shown the scheme of this board configured in data-player mode (board 1 in Fig. 6.6). The board receives a list of AER events from a PC through a USB port, and stores it in a memory included in the board. This list of events includes the address and the timestamp of each event, up to 500k-events. When the board receives a certain command through the USB port, it sends the events with the correct timing, performing the handshaking with the receiver chip. These lists of events reproduced by this board can be artificially generated, or they can be captured by other

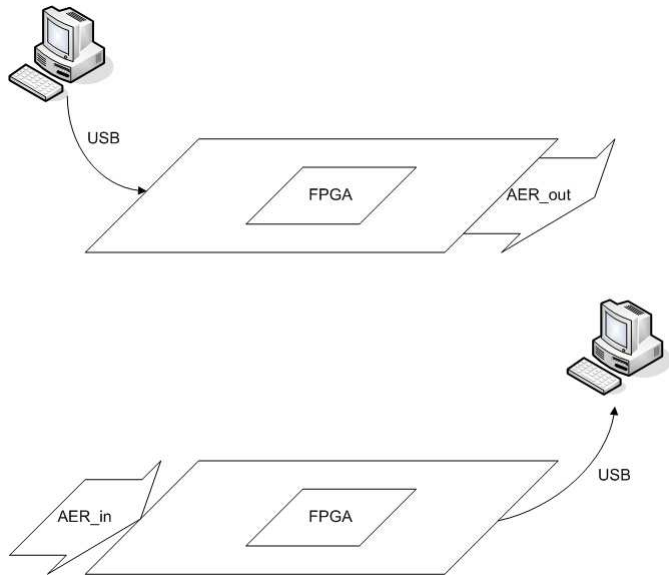


FIGURA 6.8. Scheme of the USB-AER board. Above is configured in data-player mode, while below it is configured as data-logger.

AER chip (either a retina or another convolution chip from a previous processing stage). Each event is stored as a 32-bit word, and the maximum rate allowed is 10×10^6 events per second (100ns between two consecutive events). By choosing the right firmware, the event list can be reproduced only once or continuously.

On the bottom of Fig. 6.8 it is shown the scheme for data-logger mode (board 6 in Fig. 6.6). When the board is configured this way, it receives input events through the AER bus and handles the handshaking with the sender chip. The events received are stored in its internal memory with the timestamp. Once finished the test, these events can be read from a PC (through the USB port), so that they can be processed and analyzed.

6.3.3. Configuration board

This auxiliary board is used to configure the convolution chip before performing any test. As is shown in Fig. 6.9, it has an FPGA which is con-

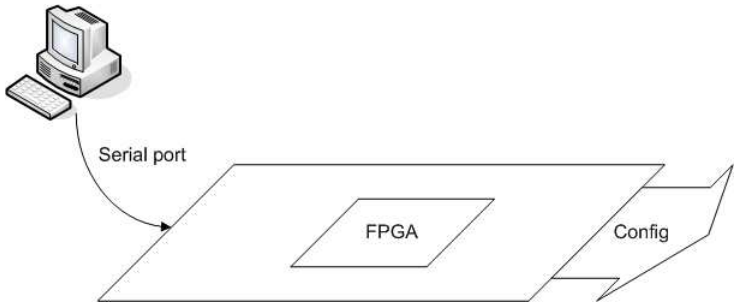


FIGURA 6.9. Scheme of the configuration board.

trolled by the PC though a serial port. Then, each time we want to change the value of a configuration data, or the kernel programmed in the RAM, we send this data to the FPGA so that it sends them serially to the chip through the configuration bus connected to the AER board in Fig. 6.7. The configuration board is number 4 in Fig. 6.6.

6.3.4. Splitter-Merger Board

Although in general we work with AER point-to-point links, it is necessary to be able to handle multi-sender and multi-receiver systems, and events re-mapping. Previously described boards, can perform simple tests with the convolution chip, but another element is necessary to implement multichip systems: the Splitter-Merger board [48]. This board can be configured for two different modes through a set of jumpers included in the board: splitter or merger mode. It has two fixed AER buses: one for input and one for output. The other three AER ports include bi-directional buffers, and can be configured for both possibilities. The board includes a CPLD which controls the handshaking signals and enables or disables the buffers.

On the top of Fig. 6.10, the board is configured as a splitter. The board receives AER events through an input bus, and sends them through the 4 output buses. It is possible to enable or disable any of these 4 output buses, so that we can use only those necessary for our application. With this configuration, it is easy to build a system which sends the events generated by a

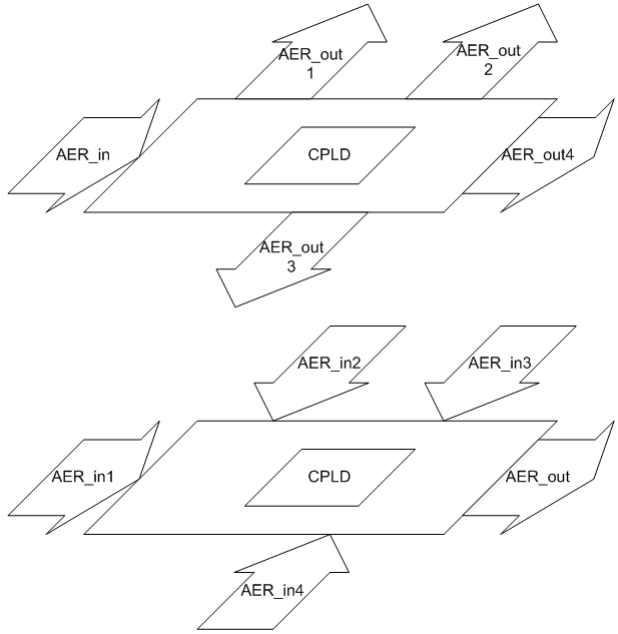


FIGURA 6.10. Scheme of the Splitter-Merger board. Above is configured as splitter, while below it is configured as merger.

single convolution chip in layer i to several convolution chips in parallel inlayer i+1. The splitter corresponds to board 2 in Fig. 6.6.

On the bottom of Fig. 6.10, it is configured as a merger. This way, the board receives input AER events though 4 different buses, and all of them are transmitted through the only output AER bus. To achieve this, the board must implement some arbitration between the inputs, so the timing can be slightly altered, although it does not affect for our applications. This board allows to implement multichip systems where several chips in parallel in layer i can send their events to a single convolution chip in layer i+1. In Fig. 6.6, the merger board corresponds to board 5.

6.3.5. Software environment

A software application based on MATLAB has been designed to control the configuration of the convolution chip. This application allows to

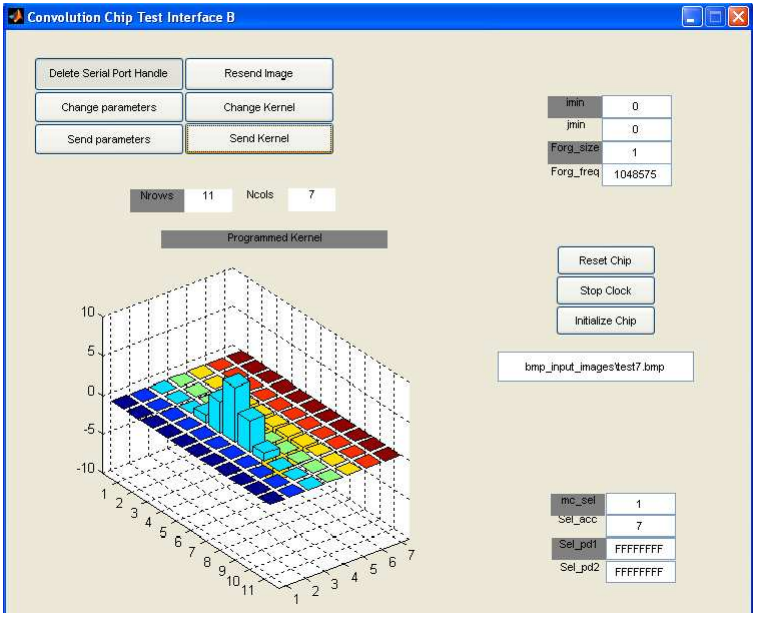


FIGURA 6.11. MATLAB-based interface developed to control the configuration of the convolution chip

program the kernel in the RAM and to write the configuration registers included in the chip. Fig. 6.11 shows a snapshot of the Graphic User Interface (GUI) developed for this work.

This application is designed to handle all the configuration of the convolution chip, but it does not control the AER traffic. Some MATLAB functions have been developed to send and receive flows of AER events, but they must be executed through the command line. This functions are designed to communicate with the boards described previously, selecting their configuration modes through the right firmware, and sending or receiving lists or AER events.

6.4. 32x32 convolution chip Conv1

In this Section, the experimental results obtained for Conv1 are described. This convolution chip has 32×32 pixels, and has been fabri-

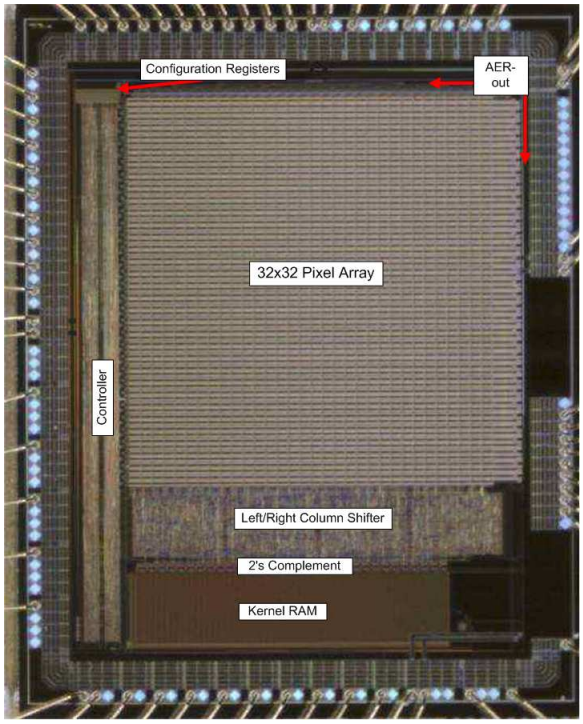


FIGURA 6.12. Photograph of the convolution chip Conv1.

cated in the AMS $0.35\mu m$ CMOS process with a total area of $4.3 \times 5.4 mm^2$ A die photograph is shown in Fig. 6.12. The largest block is the array of pixels, with an approximate area of $3.0 \times 3.2 mm^2$. The synchronous controller consumes around $4500 \times 300 \mu m^2$, the static kernel-RAM of 32×32 6-bit words $600 \times 2700 \mu m^2$, and the horizontal column shifter $600 \times 3100 \mu m^2$. The rest of the circuits, like the AER generator, 2's complement inverter or the clock generator, consume much less area.

Next, the chip characterization is described, while the tests results are shown later.

6.4.1. Chip characterization

6.4.1.1. Internal clock

The frequency of the internal clock can be adjusted through parameter V_{bias_clk}. It could be set up to 120MHz before observing some random spurious events appearing at the output. If these spurious events can be tolerated, clock frequency could be further increased until 200MHz, beyond which the convolution operation degrades completely. In our experiments we set the clock frequency at 120MHz, in order to obtain the maximum possible precision.

6.4.1.2. Power consumption

The power consumption of the chip depends both on the input throughput and the kernel size. For instance, for an input event rate of 5Meps (events per second) the power consumption varies between 66mW and 198mW, for the smallest possible kernel (1×1) and the largest (32×32), respectively.

6.4.1.3. Timing characterization

The convolution chip can generate output events at a maximum rate of 50×10^6 eps, measured shorting Rqst_out and Ack_out for events generated by pixels in the same row (in burst mode). This corresponds to an event cycle time of $T_{burst} = 20ns$. However, depending on the position of the pixels within the array, the event cycle changes significantly. Pixels closer to the arbiter will produce shorter propagation delays. However, this influence can be minimized by setting carefully the values of the bias voltages and calibration switches for the pull-ups and pull-downs on the periphery. This way, we could adjust this delay between 20 and 24ns for all the pixels in the array.

Fig. 6.13 shows the measured signals Rqst_in, Ack_in and Rqst_out (shorted with Ack_out) when the 5-line kernel in Fig. 6.14 is loaded. Pixel threshold was set for firing an output event when receiving one single input

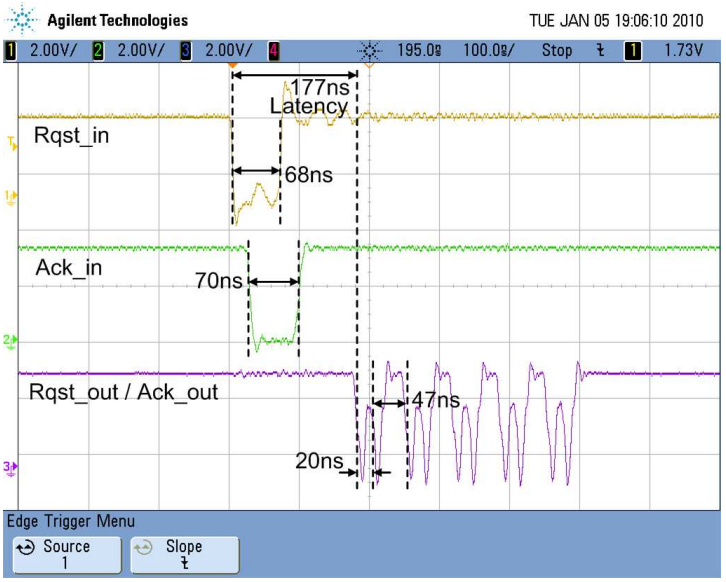


FIGURA 6.13. Measured input and output events Rqst y Ack signals when maximum clock frequency is set to 120MHz, with shorted Rqst_out and Ack_out. The oscilloscope used is Agilent DSO7054A, with a bandwidth of 500MHz and a sample rate of 4GSa/s.

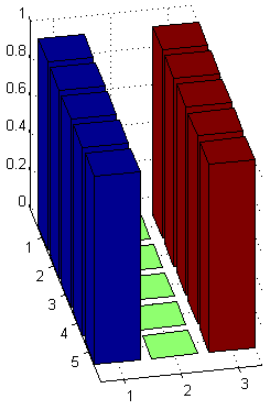


FIGURA 6.14. Kernel programmed to measure the maximum output event rate for pixels located at the four corners of the array.

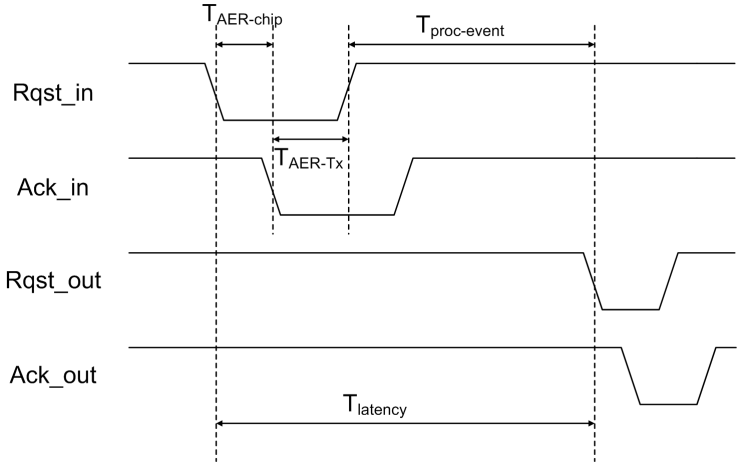


FIGURA 6.15. Time delay between input and output events.

event. Consequently, this kernel activates 10 different pixels (belonging to 5 different rows) for the same input event. The output events in Fig. 6.13 show a delay of $T_{burst} = 20ns$ when both events belong to the same row (once the row is acknowledged by the arbiter, all the events produced in the same row are generated in burst mode) and a delay of $T_{non-burst} = 47ns$ for events from different rows. Consequently, the difference $47ns - 22ns = 25ns$ corresponds to the row arbitration delay T_{arb}.

The input event throughput depends on the kernel size and the internal clock frequency. The synchronous controller needs $n_{clk} = 4 + (2 \times n_k)$ clock cycles to process a single input event, where n_k is the number of rows of the programmed kernel (up to 32). This way, for a clock frequency of 120MHz ($T_{clk} = 8.33ns$), the maximum possible input rate is 20×10^6 eps, which corresponds to an input event cycle time of 50ns when the kernel has only one row. For a full kernel of 32 rows, the input event maximum rate would be 1.77×10^6 (566ns input event cycle time).

In Fig. 6.15, the latency between input and output events is represented by $T_{latency}$, which can be expressed as $T_{latency} = T_{AER-chip} + T_{AER-Tx} + T_{proc-event}$. Delay $T_{AER-chip}$ is the one introduced by the convolution chip when Rqst_in is activated by the emitter.

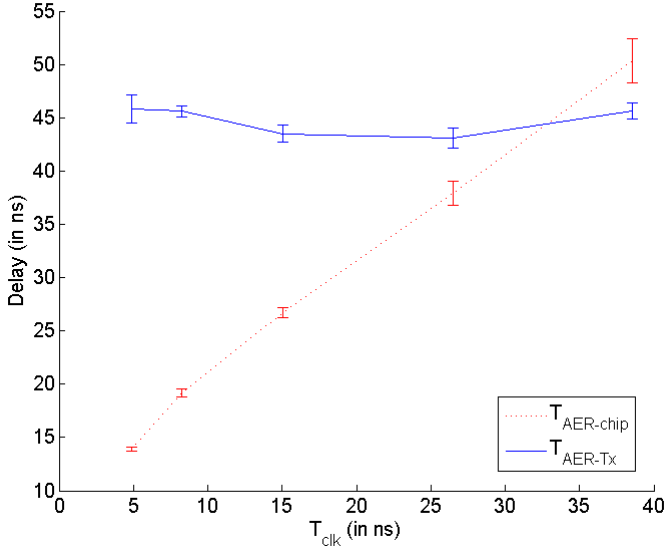


FIGURA 6.16. Measured values for $T_{AER-chip}$ and T_{AER-Tx} for different settings of the clock period.

Delay T_{AER-Tx} is the one introduced by the emitter when the convolution chip acknowledges. Delay $T_{proc-event}$ is the time used by the chip to process the input event and generate an output event (considering a situation where one input event produces an output event). Delay $T_{proc-event}$ can be expressed as $T_{proc-event} = T_{syn} + T_{asyn}$, where T_{syn} represents the time the synchronous controller needs for adding the kernel onto the array of pixels, and T_{asyn} represents the time for the asynchronous arbitration and generation of the output event.

Fig. 6.16 shows the measured $T_{AER-chip}$ and T_{AER-Tx} for different values of T_{clk}. Vertical bars indicate deviations over 5000 measurements. Delay T_{AER-Tx} is constant, as it depends only on the emitter. The delay measured for the emitter user board in our tests is about $T_{AER-Tx} = 44ns$. However, $T_{AER-chip}$ depends almost linearly on T_{clk}, as can be seen in the figure, reaching 14ns for $T_{clk} = 5ns$ and with an estimated residual value of 10ns for $T_{clk} = 0$ (probably due to internal lines and pads delays).

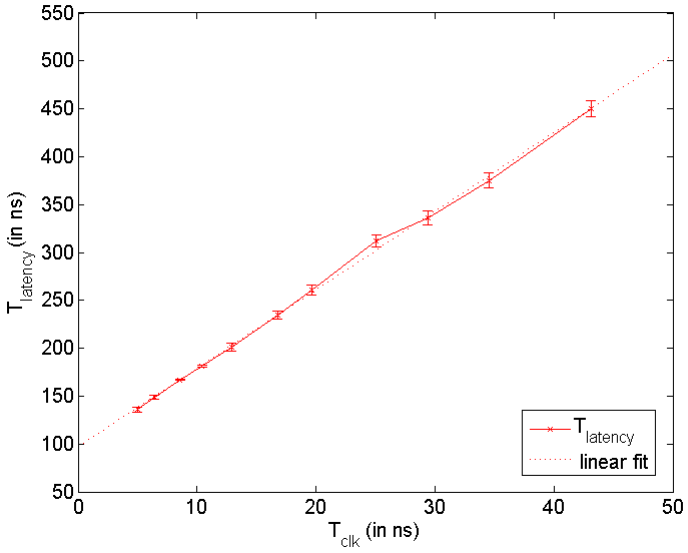


FIGURA 6.17. Measured values of $T_{latency}$ for different settings of the clock period. Each measurement was repeated 5000 times. Error bars indicate measured spread. Dotted line represents linear fit ($y = 8.2x + 97$).

Fig. 6.17 shows the measured $T_{latency}$ versus T_{clk}. The linear fit of this data reveals that at $T_{clk} = 0$ we would have a latency of 97ns, which corresponds to $T_{proc-event} = T_{asyn}$, as T_{syn} would be 0. Extrapolating also the values of $T_{AER-chip}$ and T_{AER-Tx}, we can estimate the value of $T_{asyn} = 97ns - 44ns - 10ns = 43ns$, which is coherent with the 47ns delay measured between two output events for different rows. For a clock frequency of 120MHz, the measured latency is 177ns.

When cascading convolution chips, the delay T_{AER-Tx} of the emitter board should be replaced by either $T_{burst} = 22ns$ or $T_{non-burst} = 47ns$. Consequently, the true minimum latency when cascading these chips would be given by $T_{latency} = 177ns - 44ns + 22ns = 155ns$. Note that this latency is independent of the number of lines of the kernel, because it is the

TABLA 6.1. Conv1 specifications.

Technology	4M 2P $0.35\mu m$ CMOS
Pixel size	$95.6 \times 101.3\mu m^2$
Chip size	$4.3 \times 5.4mm^2$
Pixel array	32×32
Pixel computing resolution	18 bits
Kernel resolution	6 bits
Signed computations	Yes
Input event throughput	1.77-20 Meps
Max. output event rate	50 Meps
Min. latency in-to-out	155ns
Power consumption	200mW max

delay between the input event Rqst_in and the first output event Rqst_out produced by the first kernel row.

The chip specifications for Conv1 are summarized in Tabla 6.1.

6.4.2. Convolution processing of static images

To illustrate convolution processing, a 32×32 pixel input image was selected from a real photograph (shown in Fig. 6.19.(a)) to perform an edge-orientation extraction with the 11×7 kernel shown in Fig. 6.18. This difference of gaussians kernel is described by equations (6.1), (6.2) and (6.3):

$$F_g(p_k, q_k) = \frac{1}{2\pi} H_g(p_k) V_g(q_k) \quad \textbf{(EQ 6.1)}$$

$$H_g(p_k) = \frac{1}{\sigma_{gh}} e^{-\frac{1}{2}\left(\frac{p_k}{\sigma_{gh}}\right)^2} \quad \textbf{(EQ 6.2)}$$

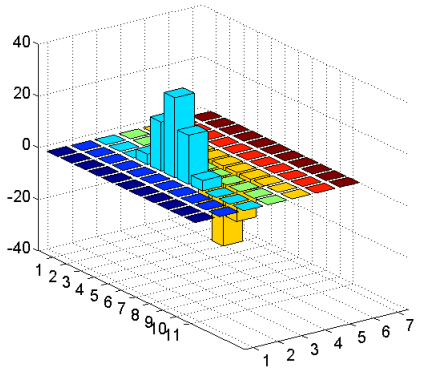


FIGURA 6.18. Difference of gaussians kernel for vertical edges extraction described by Eq. (6.1)-(6.3).

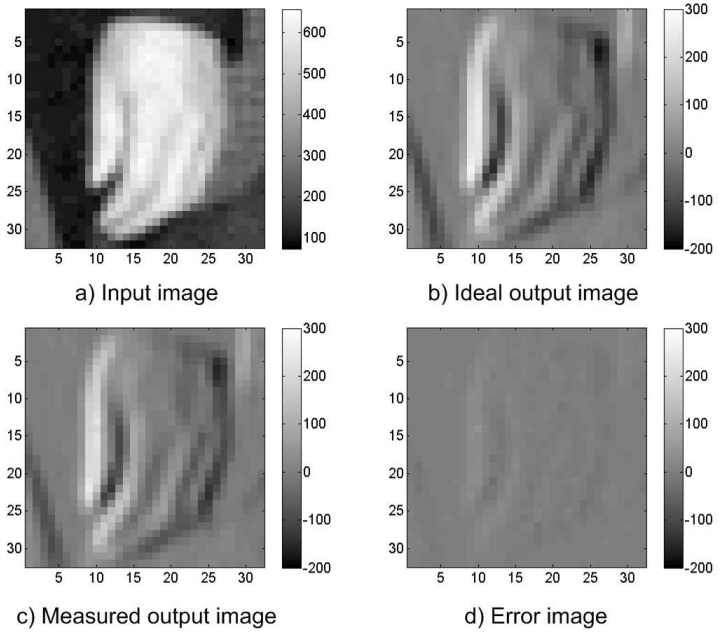


FIGURA 6.19. Experimentally obtained convolution processing results with a kernel for vertical edge extraction.

$$V_g(q_k) = \frac{1}{\sigma_{gv}}\left[e^{-\frac{1}{2}\left(\frac{q_k}{\sigma_{gv}}+\frac{1}{2}\right)^2} - e^{-\frac{1}{2}\left(\frac{q_k}{\sigma_{gv}}-\frac{1}{2}\right)^2} \right] \quad \text{(EQ 6.3)}$$

where σ_{gh} and σ_{gv} are the horizontal and vertical spatial width parameters of the gaussian lobes.

The image was rate coded into AER events with a maximum frequency of 660Hz. The frequencies associated to the intensity levels are indicated in the vertical bars on the right side of each image in Fig. 6.19. The mathematical computation of the convolution operation was performed with MATLAB, obtaining the image in Fig. 6.19.(b). Fig. 6.19.(c) shows the output from the convolution chip, by mapping the signed output event frequency of each pixel into a gray level. A negative pixel frequency means that the sign bit of the output events for this pixel was negative. Fig. 6.19.(d) represents the error image as the difference between the ideal and measured output.

Although the chip size is only 32×32, the input address space it can see is 128×128. This allows to assemble a 2D array of convolution chips for processing pixel arrays which are multiples of 32×32 [78], [95], [72]. Each convolution chip is programmed with parameters $(i_{min}, j_{min}), (i_{max}, j_{max})$ to indicate where the 32×32 pixel array is with respect to the total 128×128 input space it can see. Using AER splitter and merger boards [48], it is possible to build an array of 4×4 convolution chips to process arrays of 128×128 pixels. To process larger input spaces, we need to use also AER mapper blocks [48] to map conveniently the larger pixel space into the 128×128 pixels each convolution chip can see.

In Fig. 6.20, a large 256×256 static image was processed. The original 256×256 pixel array (Fig. 6.20.(a)) was split into 8×8 smaller subarrays. Then, each subarray was transformed into a sequence of AER events, processed by the convolution chip with the kernel shown in Fig. 6.18 and the output events recorded. The 64 recorded AER events sequences were then assembled off-line into 64 subimages and remapped to obtain the 256×256 output shown in Fig. 6.20.(c). If this mathematical convolution is computed directly with MATLAB, the result obtained is shown in

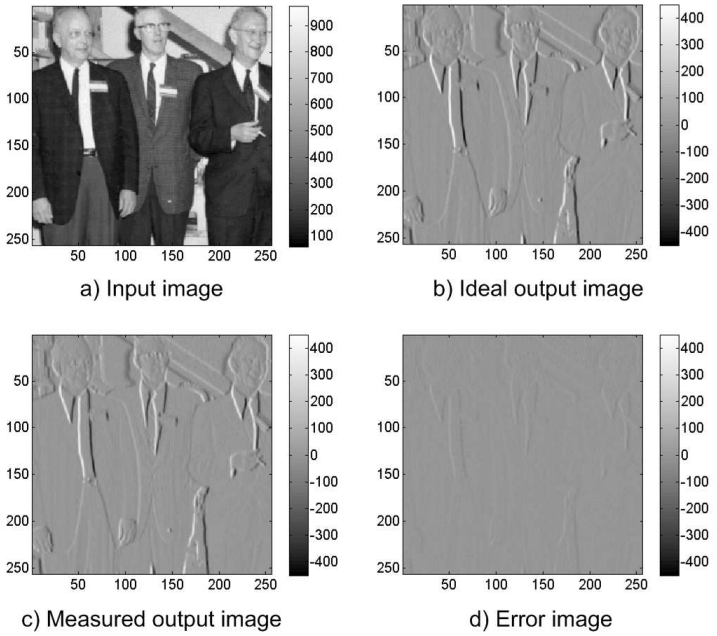


FIGURA 6.20. Experimentally obtained convolution processing results with a kernel for vertical edge extraction of a 256×256 input image.

Fig. 6.20.(b), while the error image that represents the difference between the ideal frequencies and the measured ones is shown in Fig. 6.20.(d).

6.4.3. Convolutions for moving stimuli

Although the experiments shown previously correspond to static images, the aim of the chip is to compute in real time convolutions of dynamic stimuli coming from an AER retina. To illustrate this, a sequence of events was captured with a 128×128 temporal contrast vision AER retina [18] showing the moving contours of two people walking. A 40ms snapshot of this sequence is shown in Fig. 6.21.(a), which uses only 1810 events from the retina. As the retina address space is 128×128, this requires an array of 4×4 convolution chips. Programming the 11×7 Gabor kernel in Fig. 6.21.(b) for vertical edge detection, we obtained the corresponding out-

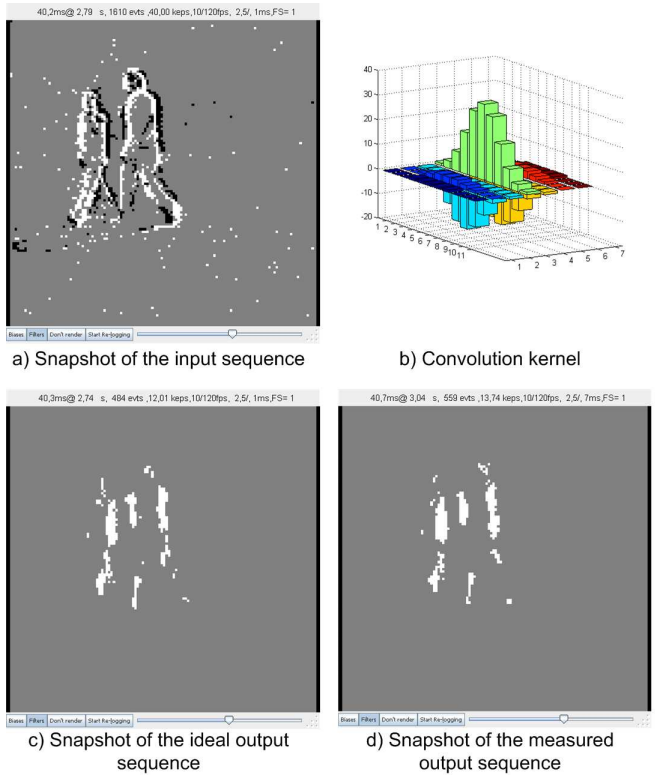


FIGURA 6.21. Experimentally obtained convolution processing results with a Gabor kernel for vertical edge extraction of a 128x128 input sequence.

put sequence. A snapshot of this output sequence for the same 40ms of the input sequence is shown in Fig. 6.21.(d). Since the latency between input and output events for this chip is about 150ns, input and output streams are in practice simultaneous.

To compare with the theoretical response, we fed the same retina stimulus to an AER behavioral simulator [74] performing the same convolution kernel. The AER output stream produced by this simulator was virtually identical to the one obtained experimentally by the chip. Fig. 6.21.(c) shows the behavioral simulator events for the same 40ms of the input in Fig. 6.21.(a).

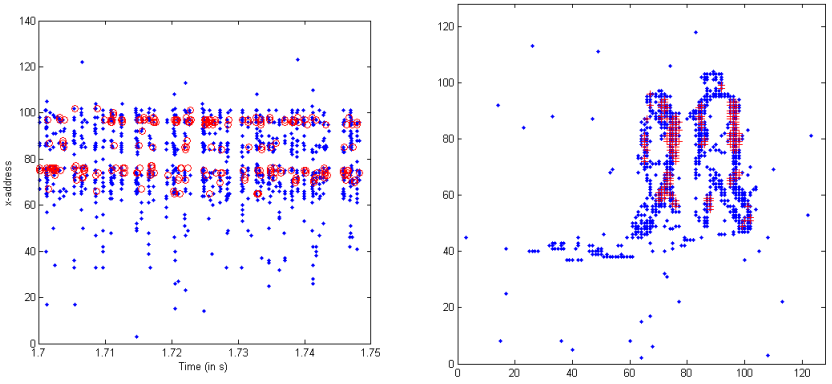


FIGURA 6.22. On the left, representation of the input and output events horizontal coordinate versus time for a 50ms interval. Input events are represented with blue dots and outputs with red circles. On the right, reconstruction of the input and output images corresponding to the same interval, with input events represented with blue dots, and outputs with red crosses.

Fig. 6.22 shows the events for an interval of 50ms from the complete sequence. In the left figure, the horizontal coordinate of the input and output events is represented versus time, while the right figure shows the reconstruction of the image in the same interval. There, it is checked how the output events (red crosses) are placed at the same pixels as the input ones (blue dots), with no noticeable delay. If we choose an even smaller time interval, the results are shown in Fig. 6.23 for a 5ms sequence. In the left figure, the timing of the input events captured by the retina is represented with blue dots, while the output events generated by the convolution chip are represented with red circles. In the right figure, the reconstruction of both the input and output images is shown for the 5ms interval. There is no noticeable delay between inputs and outputs.

6.4.4. Discrimination of fast rotating propellers

To demonstrate the high-speed processing capabilities of the convolution chip, a very interesting experiment is the discrimination between high-

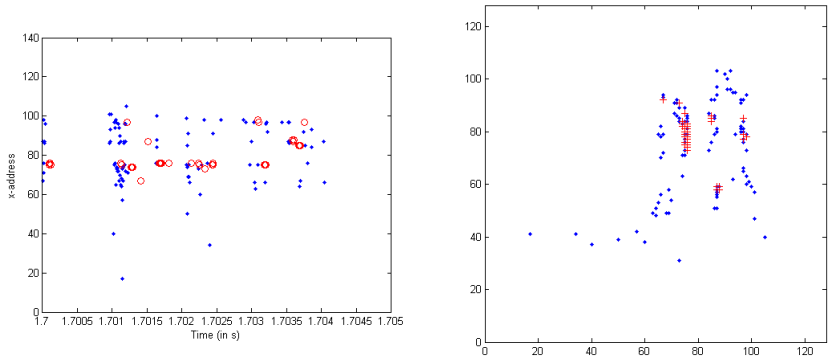


FIGURA 6.23. Representation corresponding to a time interval of 5ms. On the left, horizontal coordinate versus time. On the right, reconstruction of input and output images.

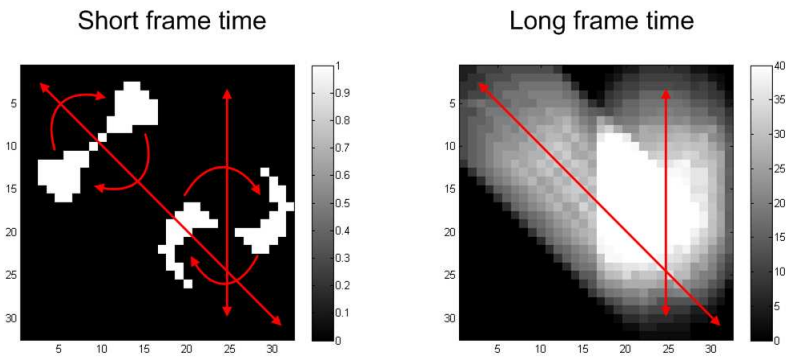


FIGURA 6.24. The left figure shows the two different propellers: the rectilinear and the S-shaped, indicating the trajectory followed by them. The right figure shows what the human eye would see when both propellers are rotating moving, not being able to discriminate between both shapes.

speed rotating propellers [78]. We create a sequence of events corresponding to two rotating propellers with different shapes. One of the propellers is rectilinear, and the other one has an S-like shape, as is illustrated on the left capture in Fig. 6.24. Both have a diameter of 16 pixels. The propellers will be rotating at a high speed and moving slowly across the screen. A human

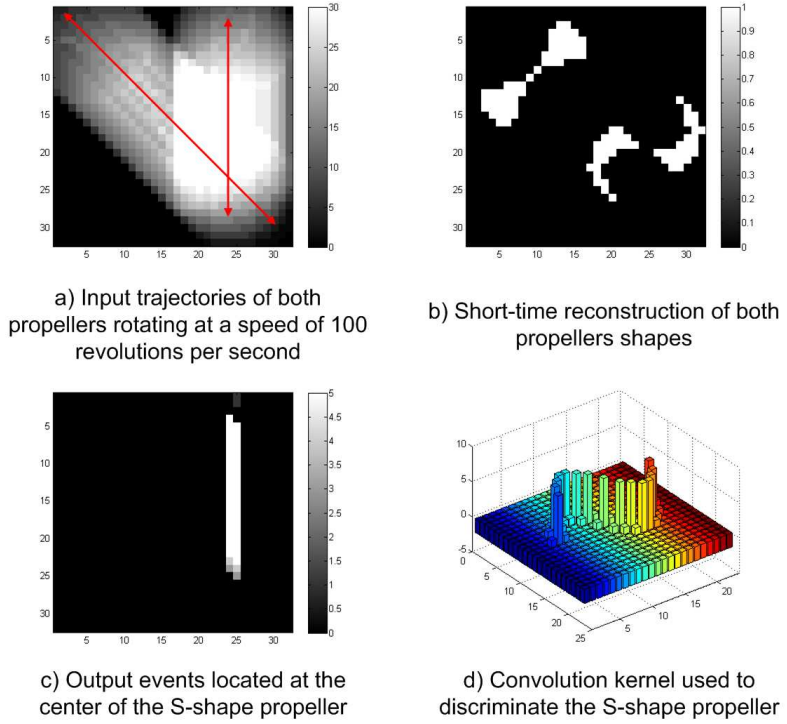


FIGURA 6.25. Real-time discrimination of simultaneous rotating propellers at 100 revolutions per second.

observer would only see two solid circles without being able to discriminate between them.

In this experiment, the sequence of input events is loaded in an AER data-player board [48], which sends the events to the convolution chip with the correct timing. After programming the specific kernel, the input events are sent to the chip and the chip output events are recorded by an AER data-logger board [48]. This way, both sequences can be carefully analyzed in a computer. The aim of the experiment is to track the center of the S-shaped propeller by programming the 23×23 kernel shown in Fig. 6.25.(d). This kernel is designed to produce positive output events in the center of the propeller when it is in horizontal position, and the large neighborhood with negative weights prevents from positive events being produced outside of

the center of the propeller. Fig. 6.25.(a), (b) and (c) show the results of the experiment, where the two propellers rotate at 100 revolutions per second. Fig. 6.25.(a) shows the complete trajectories of both propellers moving across the screen and intersecting at a given point. This corresponds to a 1-second capture, while the snapshot in Fig. 6.25.(b) corresponds to a 1-millisecond capture. Fig. 6.25.(c) shows how the output of the convolution chip follows the center of the S-shaped propeller as it moves, using the kernel represented in Fig. 6.25.(d). As expected, no output is produced for the center of the rectilinear propeller. In this experiment, as the rotation speed was very low, the average event rates were $85 \times 10^3 eps$ at the input port and only $500 eps$ at the output one. One revolution of the two propellers generates about 850 events.

Fig. 6.26 shows the results of a very similar experiment, but now the propellers are rotating at 2000 revolutions per second. This results in an average input event rate of 1.69Meps. The capture times for Fig. 6.26.(a) and (b) are $50 ms$ and $50 \mu s$, respectively. The output is shown in Fig. 6.26.(c), for which we measured an output event rate of 9.5keps. Since the kernel has 23 lines, processing one event requires 50 clock cycles, or 417ns at 120MHz clock frequency. Processing the 850 events of one revolution thus needs at least $355 \mu s$, which results in a theoretical maximum propellers rotating speed of 2821 rps (revolutions per second). To find out the real maximum rotating speed the chip is able to handle for the two propellers, we proceeded as follows. We set the rotation speed in the AER data player above this theoretical limit. This way, the chip will slow down the event throughput through the handshaking protocol, at the limit it can handle. Measuring the event throughput under these circumstances reveals the maximum rotating speed, which we measured as 2688rps.

In order to be able to work with higher rotating speeds, a pair of smaller propellers of 10 pixel diameter each (which generate about 325 events per revolution) were used, for which we need a smaller kernel of only 15 lines (shown in Fig. 6.27.(d)). Fig. 6.27 shows the result of the experiment with these propellers rotating at 5000 revolutions per second. The captures in Fig. 6.27.(a) and (b) correspond to $20 ms$ and $20 \mu s$, respectively. At this rotating speed the input event throughput is 1.62Meps. Fig. 6.27.(c) shows how the output of the chip follows the center of the S-shape propeller, producing an output event rate of 19.6keps. For a 15 line

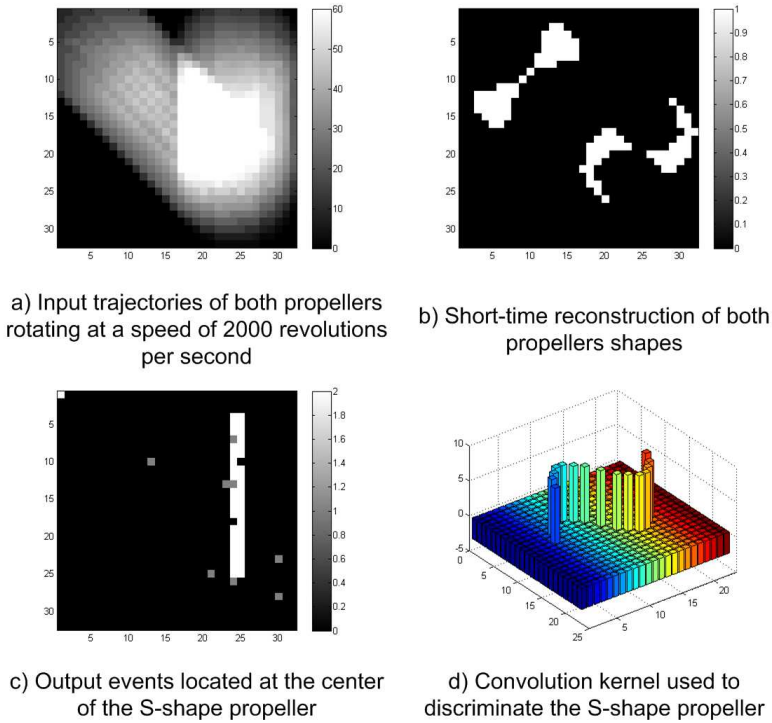
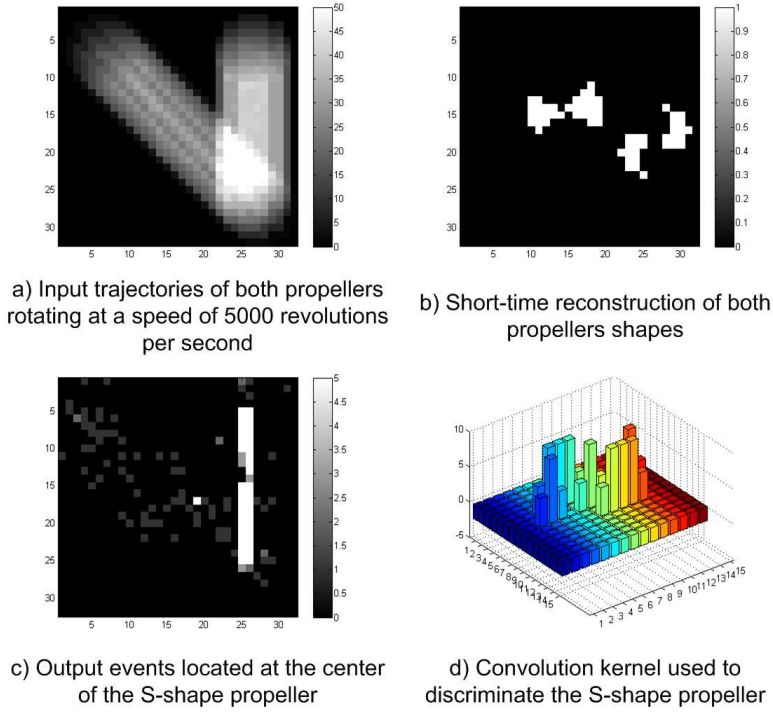


FIGURA 6.26. Real-time discrimination of simultaneous rotating propellers at 2000 revolutions per second.

kernel, each event needs 283ns, yielding a maximum possible event throughput of 3.53Meps, which corresponds to a theoretical maximum rotation speed of 10860 rps. By setting the AER data-player to provide rotations at a higher speed, the convolution chip slowed them down to 9433 rps.

These experiments reveal the potential of frame-free event based (vision) representation sensory and processing systems for very high speed object recognition. Note that for recognizing 10krps propellers in a frame-based representation system would require to sense and process images at least at 100k frames per second.



a) Input trajectories of both propellers rotating at a speed of 5000 revolutions per second

b) Short-time reconstruction of both propellers shapes

c) Output events located at the center of the S-shape propeller

d) Convolution kernel used to discriminate the S-shape propeller

FIGURA 6.27. Real-time discrimination of simultaneous rotating propellers at 5000 revolutions per second.

6.4.5. Recognition latency experiments

In order to show the short latency between input and output event flows, we did the following experiment. As input, we used a sequence of events recorded with a temporal contrast retina when a circle of flashing light-emitting diodes (LEDs) is turned on and off every 40ms. This experiment was done in [72] with the analog convolution chip, and now we can show an important improvement for the digital one.

The convolution chip was programmed with a circular kernel to detect the center of the input circle of LEDs. Both input and output events are shown in Fig. 6.28 in a 3D representation. Note that only the on or off tran-

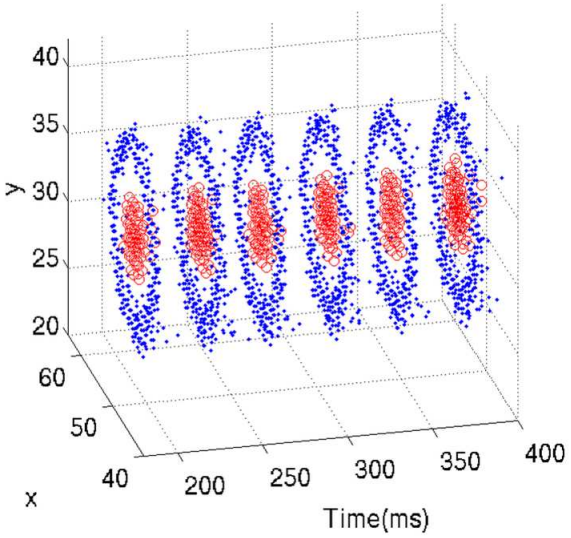


FIGURA 6.28. 3D representation in (x,y,time) space of the 40ms period ON and OFF retina transients (dots) and convolution chip outputs (circle) when exposed to a flashing circle of LEDs.

sients generate events. Input events are represented with a dot and the output ones with a circle. The recorded stimulus from the retina can be played back at different accelerated speeds with the AER data-player.

Each sub-figure in Fig. 6.29 shows a y-coordinate vs. time 2D projection of a single on or off transient. The left column graphs correspond to results obtained with the previous analog chip, while the right column graphs correspond to the present digital convolution chip. Each row in Fig. 6.29 corresponds to playing back the retina recorded data at different acceleration rates. The top row is for real time. Each on (or off) retina transient (dots) lasts for about 2ms and has about 400 events distributed in a circle in the x/y plane (see Fig. 6.28). The convolution chip output events, which appear in the center of the circle, are practically simultaneous to the input events, for both the analog and the digital convolution chips.

The second row in Fig. 6.29 corresponds to playing back the recorded retina events at a speed 10 times faster than real time. We can see that for the analog chip the output events appear slowed down, while for the digital

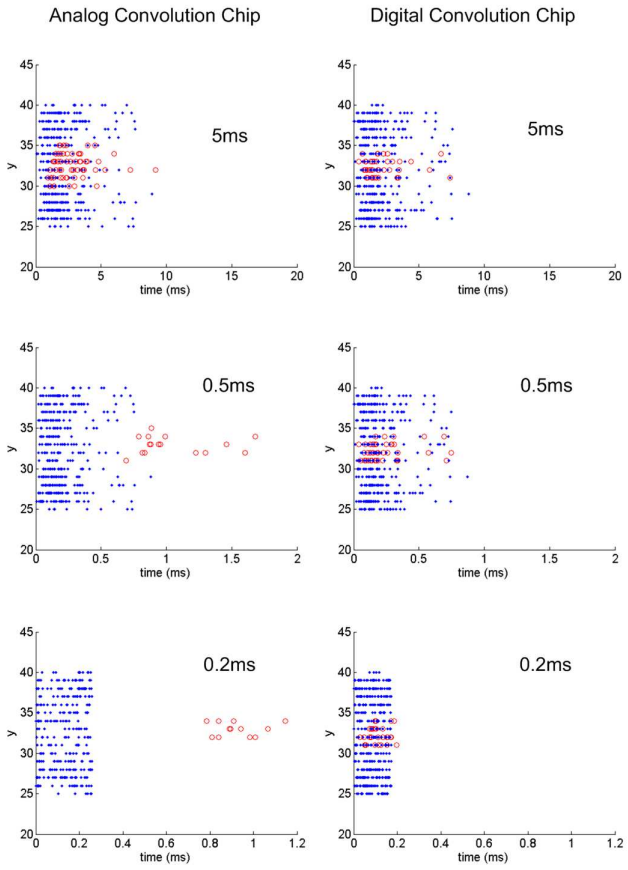


FIGURA 6.29. Results for processing the circle of LEDs and discriminate the center, both with the analog convolution chip and the digital one. Dots represent the events generated by the retina, and circles represent the events generated by the convolution chip, being the y-axis the y-address of the events. Each row shows the results for different duration of the input burst.

chip the graph is virtually identical to the real time one. In the third row the recordings have been accelerated a factor of 25 with respect to real time. As can be seen, in the analog chip there is a latency between the retina burst and the convolution chip output burst which is always in the order of 1ms. This is because the analog chip includes analog comparators in the pixel that decide when to generate output events. These comparators are biased for low power and have a bandwidth in the order of KHz. However, the

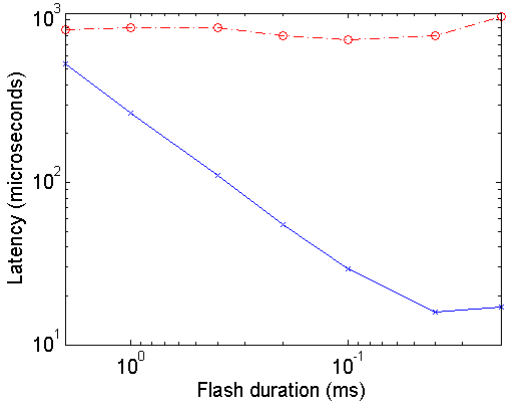


FIGURA 6.30. Measurement of the latency between input and output events for different values of the LEDs flash duration. The dotted line represents the results for the analog chip, and the solid line shows the results for the digital chip.

present digital chip will produce output events as soon as sufficient representative events for recognizing the circle are received. This happens after receiving about 30-40 events.

Fig. 6.30 shows the measured latencies between the retina first event and the convolution first event, as function of transient duration, for both chips. As can be seen, the latency for the analog chip stays approximately constant (it varies between $800\mu s$ and $1ms$), while for the digital convolution chip it decreases linearly with the transient duration, until it saturates at about $18\mu s$. Consequently, this chip is capable of performing recognition of this shape within $18\mu s$.

6.5. 64x64 convolution chip Conv2

In this section, the experimental results obtained with Conv2 are shown. This convolution chip has 64×64 pixels and has been fabricated in them AMS $0.35\mu m$ CMOS process. A die photograph is shown in Fig. 6.31, with a total area of $5.5 \times 5.8 mm^2$. The largest block is the 64×64 array of pixels, with an approximate area of $3.7 \times 3.4 mm^2$, while the synchronous controller with the configuration table which stores the

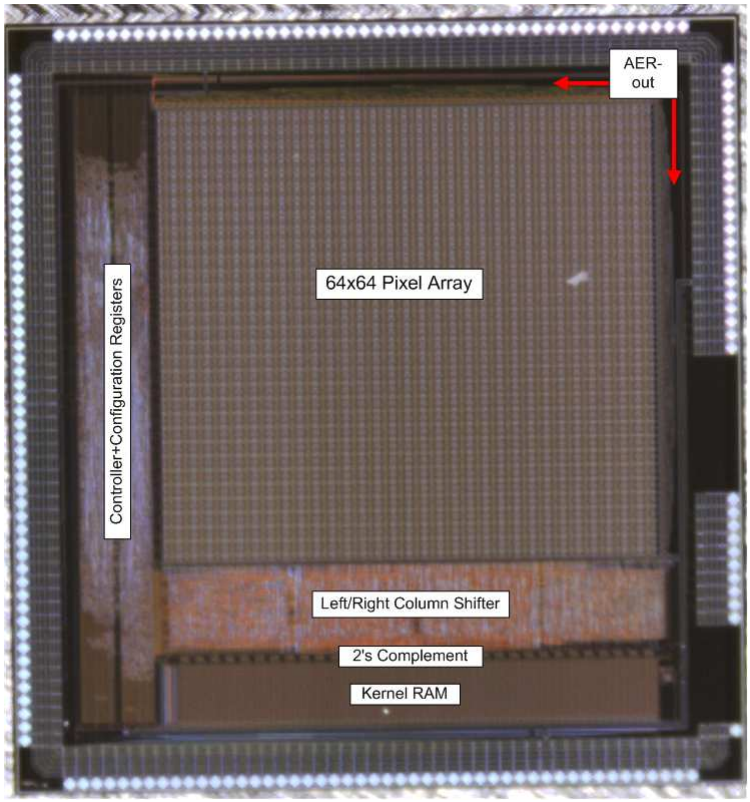


FIGURA 6.31. Photograph of the convolution chip Conv2.

multikernel parameters consumes around $4850 \times 650 \mu m^2$. The static kernel-RAM of 32×32 4-bit words consumes around $450 \times 3700 \mu m^2$, and the horizontal column shifter $650 \times 3800 \mu m^2$. The rest of the circuits, like the AER generator, 2's complement inverter or the clock generator, consume much less area.

6.5.1. Chip characterization

6.5.1.1. Internal clock

As in previous version, the frequency of the internal clock can be adjusted through the voltage bias V_{bias_clk}. It could be set up to 100MHz

without losing events. The main limitation for incrementing the frequency is produced by the delay introduced when reading the kernel from the RAM, as was described in Section 5.5. However, under certain circumstances when working with a small kernel located at the center of the RAM, it is possible to increase the clock frequency, reaching 200MHz with an acceptable behavior. Nevertheless, in our experiments we set the clock frequency at 100MHz.

6.5.1.2. Power consumption

The power consumption of the chip, as in previous version, depends both on the input throughput and the kernel size. For an input rate of 5Meps (which corresponds to a 200ns event time), the power consumption for the smallest possible kernel (1×1) is 132mW, while for the largest kernel (32×32) it is 198mW.

6.5.1.3. Timing characterization

Convolution chip Conv2 can generate output events at a maximum rate of 37×10^6 eps, measured shorting Rqst_out and Ack_out, and for events generated by pixels in the same row (in burst mode). This rate corresponds to an event cycle time of $T_{burst} = 27ns$. For pixels belonging to different rows, the measured delay between two events is $T_{non-burst} = 60ns$. These delays are slightly larger than those measured for Conv1, due to the size of the new version. In particular, delays produced by the AER generator are increased because both the row and column arbiters have 64 inputs (instead of 32 in first version), so one more level is introduced in the arbiter trees.

Fig. 6.32 shows the measured signals Rqst_in, Ack_in and Rqst_out (shorted with Ack_out) when the 5-line kernel in Fig. 6.14 is loaded, with the measured delays. With this information, the delay introduced by the row arbiter can be calculated as $T_{arb} = 60ns - 27ns = 33ns$.

The input event throughput depends on the kernel size and the internal clock frequency, as in version Conv1. Therefore, the synchronous controller needs $n_{clk} = 4 + (2 \times n_k)$ clock cycles to process a single input event, where n_k is the number of rows of the programmed kernel (up to 32). This

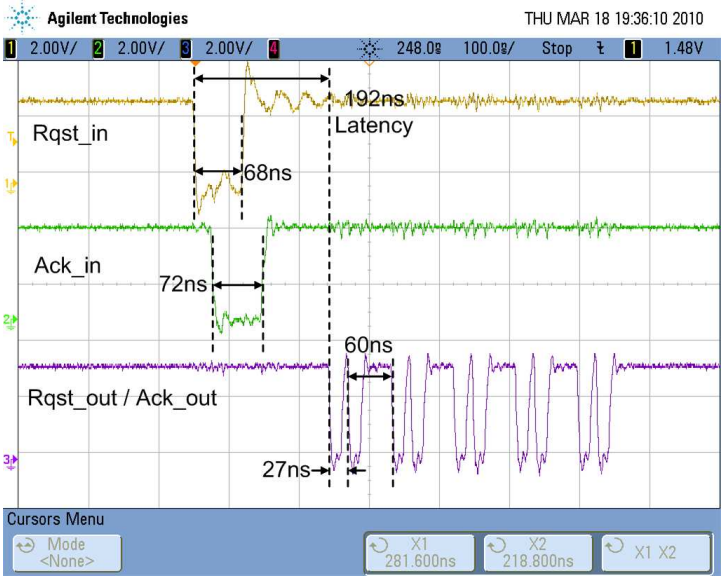


FIGURA 6.32. Measured input and output events Rqst and Ack signals when clock frequency is set to 100MHz, with shorted Rqst_out and Ack_out.

way, for a clock frequency of 100MHz (10ns period) the maximum possible input rate is 16.6×10^6 eps, which corresponds to an input event cycle time of 60ns when the kernel has only one row. For a full kernel of 32 rows, the input event maximum rate would be 1.47×10^6 eps (680ns input event cycle time).

To obtain the latency between input and output events, we express it as $T_{latency} = T_{AER-chip} + T_{AER-Tx} + T_{proc-event}$, as represented in Fig. 6.15. Fig. 6.33 shows the measured $T_{AER-chip}$ and T_{AER-Tx} for different values of the clock period T_{clk}. Delay T_{AER-Tx} is constant (it does not depend on the clock period), and has a value of 44ns (the same emitter board was used with Conv1 and Conv2). $T_{AER-chip}$ depends almost linearly on T_{clk}, with an estimated residual value of 4ns for $T_{clk} = 0$.

Fig. 6.34 shows the measured $T_{latency}$ versus the clock period. The linear fit of the experimental data indicates that at $T_{clk} = 0$ we would have a latency of 102ns. This value corresponds to $T_{proc-event} = T_{asyn}$, as ideally

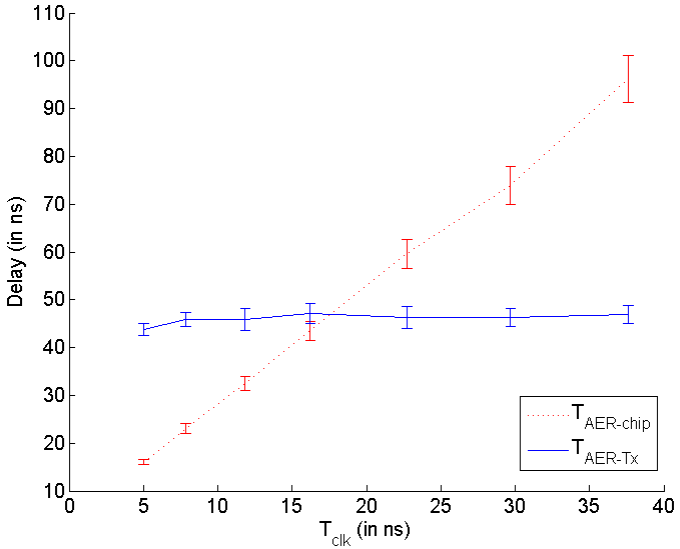


FIGURA 6.33. Measured values for $T_{AER-chip}$ and T_{AER-Tx} for different settings of the clock period.

the synchronous processing time would be 0. Extrapolating the values of $T_{AER-chip}$ and T_{AER-Tx}, we can estimate the value of $T_{asyn} = 102ns - 44ns - 4ns = 54ns$, which is coherent with the 60ns delay measured between two output events from different rows. For a clock frequency of 100MHz, the measured latency is 192ns.

As in previous version, when cascading convolution chips Conv2, the delay T_{AER-Tx} would be replaced either by $T_{burst} = 27ns$ or $T_{non-burst} = 60ns$. This way, the real minimum latency when cascading chips would be given by $T_{latency} = 192ns - 44ns + 27ns = 175ns$.

In Tabla 6.2, chip specifications for Conv2 are summarized.

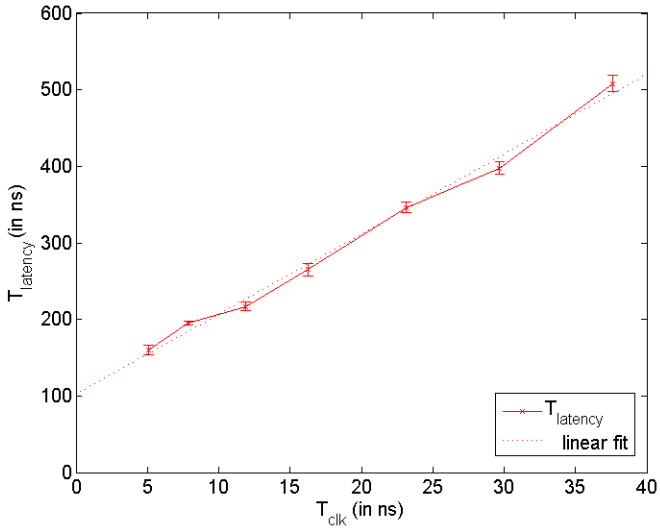


FIGURA 6.34. Measured values of $T_{latency}$ for different settings of the clock period. Dotted line represents linear fit ($y = 10x + 102$).

TABLA 6.2. Conv2 specifications.

Technology	4M 2P 0.35μm CMOS
Pixel size	$58.0 \times 53.8 \mu m^2$
Chip size	$5.5 \times 5.8 mm^2$
Pixel array	64×64
Pixel computing resolution	6 bits
Kernel resolution	4 bits
Signed computations	Yes
Multikernel system	Yes
Events inhibition	Yes
Input event throughput	1.47-16.6 Meps
Max. output event rate	37 Meps
Min. latency in-to-out	175ns
Power consumption	200mW max

CAPÍTULO 7

Conclusions and future outlook

In this dissertation, we have presented the design of convolution microchips for real-time vision asynchronous processing, with AER communication protocol. These convolution chips are not frame-based, but they work with event-coded sensory information. The event-based processing systems have important advantages over those frame-based, as has been described in this document, specially the fast results produced. When coding the information about the pixels state on events, most important information is processed first, so results are obtained in a very short time.

The chips described in this thesis implement the convolutional operation of the input image with programmable kernels of arbitrary shape and size in real time. With this programmability, these chips can be the basic element to build more complex multi-layer bio-inspired high-speed processing systems. Therefore, connecting many of these convolution chips (cascading and in parallel) properly programmed, plenty of applications can be implemented, like systems for discriminating and tracking geometric shapes, image segmentation, characters recognition,...

The main contributions of this thesis are:

1. Two different versions of a new Integrate&Fire fully digital pixel to implement convolutions. This digital pixel overcomes the main drawbacks presented by previous analog implementations [78], like the need for calibration to compensate mismatching, the reduced resolution (3 bits) and the high latency (1ms). The new digital pixel needs no calibration circuitry (which reduces the area consumed by a single pixel and allows to build larger arrays), reaches a high resolution up to 18 bits, and a reduced latency as low as 150ns. These pixels are designed to process both positive and negative input weights, and to generated signed output events.
2. Events inhibition at the pixel level. This capability allows to select the possibility to inhibit the events generated by the pixels depending on their sign. If for a certain application, the negative events carry no important information for the next processing layer, the pixel can discard these events without consuming communication bandwidth.
3. Computation of the inverted kernel in 2's complement, to process negative input events. The design of the inverter block allows to obtain the inverted kernel as it is being processed with a delay of 400ps, so that the clock frequency of the system is not affected.
4. Implementation of a programmable forgetting mechanism, which decrements at a controlled rate the state of the pixels, so that a temporal correlation between the input events can be detected.
5. Implementation of the multikernel system. We propose the capability to program up to 32 different kernels in a single convolution chip, so that each input event can be processed by a specific kernel, depending on the origin of this event. To implement this system, a configuration table is designed to store the position of each kernel inside the RAM of the chip, so that the controller can process the input events with the corresponding kernel. A circuit is included to block the columns of the RAM, in order to avoid the addition of the values not included in the selected kernel.

Therefore, two convolution chips have been fabricated and tested, with a first version of 32×32 pixels and a maximum resolution of 18 bits, and a second version of 64×64 pixels and 6-bit resolution. Both versions are designed to build larger arrays with several chips in parallel that behave like a single chip with larger size. Chapter 6 shows exhaustive experimental results obtained with both chips.

The future work is intended to implement more and more complex multi-chip systems, with the following steps:

1. Development of a modular and scalable infrastructure of AER serial-communication PCBs with reduced connectors, so that several tens of convolutional modules can be assembled.
2. Adaptation of objects visual recognition techniques based on the software systems known as “Convolutional Neural Networks” to the AER hardware infrastructure.
3. Development of learning techniques specific for “Spiking Neural Networks”, and their adaptation and implementation to the AER convolution hardware.
4. Application to real systems which need efficient and high-speed visual perception, like vehicles automatic driving, robotic applications in non-structured environments, or intelligent surveillance in security systems.

References

[1] P. Symes, “Video Compresion Demystified”, McGraw-Hill, 2001.

[2] Eadweard Muybridge, Human and Animal Locomotion, Philadelphia, 1887.

[3] Recommendation ITU-R BT. 470-6, Conventional Television Systems.

[4] Recommendation ITU-R BT. 470-7, Conventional Analog Television Systems.

[5] G. M. Shepherd, *The Synaptic Organization of the Brain*, Oxford University Press, 3rd Edition, 1990.

[6] E. Culurciello, R. Etiene-Cummings and K. A. Boahen, “A Biomorphic Digital Image Sensor”, *IEEE Journal of Solid-State Circuits*, vol. 38, pp. 281-294, 2003.

[7] J. Costas-Santos et al., “A contrast retina with on-chip calibration for neuromorphic spike based AER vision systems”, *IEEE Transactions on Ciruits and Systems I, Regular Papers*, vol. 54, no. 7, pp. 1444-1458, July 2007.

[8] S. Thorpe, A. Delorme and R. Van Rullen, “Spike based strategies for rapid processing”, *Neural Networks*, Vol. 14, pp. 715-725, (2001 Special issue).

[9] W. Maass and C.M.Bishop, “Pulsed Neural Networks”, MIT press 1999.

[10] M. Sivilotti, “Wiring considerations in analog VLSI systems with applications to field programmable networks”, Ph.D. dissertation, California Institute of Technology, Pasadena, CA, 1991.

[11] M. Mahowald, “VLSI analogs of neural visual processing: A synthesis of form and function”, Ph.D dissertation, California Institute of Technology, Pasadena, CA, 1992.

[12] P. F. Ruedi et al., “A 128 x 128 pixel 120-dB dynamic-range vision-sensor chip for image contrast and orientation extraction”, *IEEE J. Solid-State Circuits*, vol. 38, no. 12, pp. 2325-2333, Dec. 2003.

[13] M. Barbaro, P. Y. Burgi, A. Mortara, P. Nussbaum and F. Heitger, “A 100 x 100 pixel silicon retina for gradient extraction with steering filter capabilities and temporal output coding”, *IEEE J. Solid-State Circuits*, vol. 37, no. 2, pp. 160-172, Feb. 2002.

[14] C. Shoushun and A. Bermak, “A low power CMOS imager based on time-to-first-spike enconding and fair AER”, in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS’05)*, pp. 5306-5309. 2005.

[15] X. Qi, X. Guo and J. Harris, “A time-to-first-spike CMOS imager”, in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS’04)*, Vancouver, BC, Canada, 2004, pp. 824-827.

[16] J. Kramer, “An on/off transient imager with event-driven, asynchronous read-out”, in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS’02)*, Phoenix, AZ, 2002, pp. 165-168.

[17] P. Lichsteiner, T. Delbrück and J. Kramer, “Improved on/off temporally differentiating address-event imager”, in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS’04)*, Vancouver, BC, Canada, 2004, pp. 211-214.

[18] P. Lichsteiner, C. Posch and T. Delbrück, “A 128 x 128 120dB 30mW asynchronous vision sensor that responds to relative intenstity change”, in *IEEE ISSCC Dig. Tech. Papers*, San Francisco, CA, 2006, pp. 508-509.

[19] M. Arias-Estrada, D. Poussart and M. Tremblay, “Motion vision sensor architecture with asynchronous self-signaling pixels”, in *Proc. 7th Int. Workshop Comput. Architecture Machine Perception (CAMP’97)*, 1997, pp. 75-83.

[20] C. M. Higgins and S. A. Shams, “A biologically inspired modular VLSI system for visual measurement of self-motion”, *IEEE Sensors J.*, vol. 2, no. 6, pp. 508-528, Dec. 2002.

[21] E. Ozalevli and C. M. Higgins, “Reconfigurable biologically inspired visual motion system using modular neuromorphic VLSI chips”, *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 52, no. 1, pp. 79-92, Jan. 2005.

[22] G. Indiveri, A. M. Whatley and J. Kramer, “A reconfigurable neuromorphic VLSI multi-chip system applied to visual motion computation”, in *Proc. Int. Conf. Microelectron. Neural, Fuzzy Bio-Inspired Syst. (Microneuro’99)*, Granada, Spain, 1999, pp. 37-44.

[23] K. Boahen, "Retinomorphic chips that see quadruple images", in *Proc. Int. Conf. Microelectron. Neural, Fuzzy Bio-Inspired Syst. (Microneuro'99)*, Granada, Spain, 1999, pp. 12-20.

[24] J. Lazzaro, J. Wawrzynek, M. Mahowald, M. Sivilotti and D. Gillespie, "Silicon auditory processors as computer peripherals", *IEEE Trans. Neural Netw.*, vol. 4, no. 3, pp. 523-528, May 1993.

[25] R. Z. Shi and T. K. Horiuchi, "A VLSI model of the bat dorsal nucleus of the lateral lemniscus for azimuthal echolocation", in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS'05)*, Kobe, Japan, 2005, pp. 4217-4220.

[26] A. van Schaik and S.-C. Liu, "AER EAR: a matched silicon cochlea pair with address event representation interface", in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS'05)*, Kobe, Japan, 2005, pp. 4213-4216.

[27] G. Cauwenberghs, N. Kumar, W. Himmelbauer and A. G. Andreou, "An analog VLSI chip with asynchronous interface for auditory feature extraction", *IEEE Trans. Circ. Syst. II, Analog Digit. Signal Process.*, vol. 45, no. 5, pp. 600-606, May 1998.

[28] M. Oster and S.-C. Liu, "Spiking inputs to a spiking winner-take-all circuit", in *Advances in Neural Information Processing Systems*, Y.Weiss, B. Schölkopf, and J. Platt, Eds. Cambridge, MA: MIT Press, 2006, vol. 18, pp. 1051–1058 [Online]. Available: http://books.nips.cc/papers/files/nips18/NIPS2005_0521.pdf, (NIPS'06)

[29] J. Abrahamsen, P. Häfliger, and T. S. Lande, "A time domain winner-take-all network of integrate-and-fire neurons," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS'04)*, Vancouver, BC, Canada, May 2004, vol. V, pp. 361–364.

[30] E. Chicca, G. Indiveri, and R. J. Douglas, "An event-based VLSI network of integrate-and-fire neurons," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS'04)*, Vancouver, BC, Canada, 2004, vol. V, pp. 357–360.

[31] T. Teixeira, A. G. Andreou, and E. Culurciello, "Event-based imaging with active illumination in sensor networks," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS'05)*, Kobe, Japan, 2005, pp. 644–647.

[32] R. Serrano-Gotarredona, T. Serrano-Gotarredona, A. Acosta-Jiménez and B. Linares-Barranco, "A neuromorphic cortical-layer microchip for spike-based event processing vision systems", *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 53, no. 12, pp. 2548-2566, Dec. 2006.

[33] A. Mortara and E. A, Vittoz, "A communication tailored for analog VLSI artificial neural networks: intrinsic performance and limitations", *IEEE Trans. Neural Netw.*, vol. 5, no. 3, pp. 459-466, May 1994.

[34] A. Mortara and E. A. Vittoz, "A communication scheme for analog VLSI perceptive systems", *IEEE Journal of Solid-State Circuits*, vol. 30, no. 6, pp. 660-669, June 1995.

[35] V. Brajovic, "Lossless non-arbitrated address event coding", *Proc. of International Symposium on Circuits and Systems* (ISCAS'03), vol. 5, pp. 825-828, May 2003.

[36] Z. Kalayjian and A. G. Andreou, "Asynchronous communication of 2D motion information using winner-takes-all arbitration", *International Journal on Analog Integrated Circuits and Signal Processing*, vol. 13, no. 1-2, pp. 103-109, March/April 1997.

[37] K. Boahen, "Retinomorphic vision systems", *Proc. of International Conference on Microelectronics for Neural Networks*, pp. 2-14, 1996.

[38] K. A. Boahen, "Communication neuronal ensembles between neuromorphic chips", *Neuromorphic Systems engineering*, chapter 11, Kluwer Academic Publishers, edited by T. S. Lande, 1998.

[39] K. A. Boahen, "A throughput-on-demand address-event transmitter for neuromorphic chips", ARVLSI'99, Atlanta, GA. *Proceedings published by IEEE Computer Society Press*, Los Alamitos, CA, pp. 72-86, 1999.

[40] K. A. Boahen, "Point-to-point connectivity between neuromorphic chips using address events", *IEEE Trans. on Circuits and Systems Part-II*, vol. 47, no. 5, pp. 416-434, May 2000.

[41] K. A. Boahen, "A burst-mode word-serial address-event link-I: transmitter design", *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 51, pp. 1269-1280, 2004.

[42] K. A. Boahen, "A burst-mode word-serial address-event link-II: receiver design", *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 51, pp. 1281-1291, 2004.

[43] K. A. Boahen, "A burst-mode word-serial address-event link-III: analysis and test results", *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 51, no. 7, pp. 1292-1300, 2004.

[44] J. P. Lazzaro and J. Wawrzynek, "A multi-sender asynchronous extension to the Address-Event protocol", *16th Conference on Advanced Research in VLSI*, W. J. Dally, J. W. Poulton, and A. T. Ishii (Eds.), pp. 158-169, 1995.

[45] S. R. Deiss, R. J. Douglas and A. M. Whatley, "A pulse-coded communications intrastructure for neuromorphic systems", in *Pulsed Neural Networks*, W. Maass and C. M. Bishop Editors, MIT Press, pp. 157-178, 1999.

[46] B. J. Sheu and J. Choi, "Neural information processing and VLSI", *The Kluwer International Series in Engineering and Computer Science*, Kluwer Academic Publishers, chapter 15, pp. 486-488, 1995.

[47] A. Jiménez-Fernández, C. D. Luján, A. Linares-Barranco, F. G.-. Rodríguez, M. Rivas, G. Jiménez, and A. Civit, "Address-event based platform for bio-inspired spiking systems," *Proc. SPIE*, vol. 6592, 2007, DOI: 10.1117/12.724156, 659206.

[48] R. Paz, A. Linares-Barranco, M. Rivas, L. Miró, S.Vicente, G. Jiménez, and A. Civit, "AER tools for communications and debugging," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2006, pp. 3253–3256.

[49] Y. LeCun, "Generalization and network design strategies", *Technical Report CRG-TR-89-4*, Department of Computer Science, University of Toronto, 1989.

[50] L. Bottou et al., "Comparison of classifier methods: a case study in handwritten digit recognition", *Proceedings of the International Conference on Pattern Recogintion*, Los Alamitos, CA: IEEE Computer Society Press, 1994.

[51] I. Guyon et al., "Design of a neural network character recognizer for a touch terminal", *Pattern Recognition*, 24(2): pp. 105-119, 1991.

[52] K. Lang and G. Hinton, "The development of the time-delay neural network architecture for speech recognition", *Technical Report CMU-CS-88-152*, Carnegie-Mellon University, Pittsburgh, 1988.

[53] A. Waibel et al., "Phoneme recognition using time-delay neural networks", *IEEE Trans. Acoustics, Speech, Signal Processing*, 37: pp. 328-339, 1989.

[54] L. Bottou et al., "Speaker independent isolated digit recognition: multilayer perceptron vs dynamic time warping", *Neural Netw.*, 3: pp. 453-465, 1990.

[55] L. R. Rabiner and B. Gold, "Theory and application of digital signal processing", *Prentice-Hall*, Inc. 1975.

[56] A. Rosenfeld and A. C. Kak, "Digital Picture Processing", *Academic Press*, Inc. 1982.

[57] T. Serre, "Learning a dictionary of shape-components in visual cortex: Comparison with neurons, humans and machines," *MIT. Comput. Sci. & AI Lab*, Cambridge, MA, Tech. Rep. MIT-CSAIL-TR-2006-028 CBCL-260, 2006.

[58] G. Leuba and R. Kraftsik, "Changes in volume, surface estimate, three-dimensional shape and total number of neurons of the human primary visual cortex from midgestation until old age", *Anatomy and embryology*, 190(4): 351-66, 1994.

[59] H. Fujii, H. Ito, K. Aihara, N. Ichinose, and M. Tsukada, "Dynamical cell assembly hypothesis – Theoretical possibility of spatio-temporal coding in the cortex," *Neural Netw.*, vol. 9, pp. 1303–1350, 1996.

[60] S. Grossberg and E. Mingolla, "Neural dynamics of form perception: Boundary completion, illusory figures and neon colour spreading", *Psycological Review*, vol. 92, pp. 173-211, 1985.

[61] S. Grossberg and E. Mingolla, "Neural dynamics of perceptual grouping: Textures, boundaries and emergent segmentations", *Perception and Psychophysics*, vol. 38, pp. 141-171, 1985.

[62] S. Grossberg, E. Mingolla and J. Williamson, “Synthetic aperture radar processing by a multiple scale neural system for boundary and surface representation”, *Neural Networks*, vol. 8, No. 7/8, pp. 1005-1028, 1995.

[63] S. Grossberg, E. Mingolla and W. D. Ross, “Visual brain and visual perception: how does the cortex do perceptual grouping?”, *Trends in Neuroscience*, vol. 20, pp. 106-111, 1997.

[64] E. Mingolla, W. Ross and S. Grossberg, “A neural network for enhancing boundaries and surfaces in synthetic aperture radar images”, *Neural Networks*, vol. 12, No. 3, pp. 499-511, 1999.

[65] T. Serrano-Gotarredona, A. G. Andreou and B. Linares-Barranco, “A programmable VLSI filter architecture for application in real-time vision processing system”, *International Journal of Neural Systems*, vol. 10, pp. 179-190, 2000.

[66] S. Thorpe, D. Fize, and C. Marlot, “Speed of processing in the human visual system,” *Nature*, vol. 381, pp. 520–522, Jun. 1996.

[67] K. Fukushima, “Visual feature extraction by a multilayered network of analog threshold elements”, *IEEE Transactions on Systems Science and Cybernetics*, SSC-5, vol. 4, pp. 322-333, 1969.

[68] K. Fukushima and S. Miyake, “Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position”, *Pattern Recognition*, vol. 15, pp. 455-469, 1982.

[69] K. Fukushima, “Neocognitron: A hierarchical neural network capable of visual pattern recognition”, *Neural Networks*, vol. 1, pp. 119-130, 1988.

[70] K. Fukushima, “Analysis of the process of visual pattern recognition by the neocognitron”, *Neural Networks*, vol. 2, pp. 413-420, 1989.

[71] Y. LeCun and Y. Bengio, “Convolutional Neural Networks for images, speech and time series”, *Handbook of Brain Theory and Neural Networks*, M.A.Arbib (Ed.), pp. 255-258. MIT press, Cambridge, MA, 1995.

[72] R. Serrano-Gotarredona et al., "CAVIAR: A 45k-Neuron, 5M-Synapse, 12G-connects/sec AER Hardware Sensory-Processing-Learning-Actuating System for High Speed Visual Object Recognition and Tracking," *IEEE Trans. on Neural Networks*, vol. 20, No. 9, pp. 1417-1438, 2009.

[73] J. A. Pérez-Carrasco, T. Serrano-Gotarredona, C. Serrano-Gotarredona, B. Acha and B. Linares-Barranco, “On the computational power of Address-Event Representation (AER) vision processing hardware”, *Proc. XXII Int. Conference on Design of Circuits and Integrated Systems* (DCIS), Sevilla, Spain, 21-23 November, 2007.

[74] J. A. Pérez-Carrasco, T. Serrano-Gotarredona, C. Serrano-Gotarredona, B. Acha and B. Linares-Barranco, "High-speed character recognition system based on a complex hierarchical AER architecture", *IEEE International Symposium on Circuits and Systems* (ISCAS) 2008, pp. 2150-2153, 2008.

[75] J. A. Pérez-Carrasco, C. Serrano, B. Acha, T. Serrano-Gotarredona and B. Linares-Barranco, "Event based vision sensing and processing", *15th International Conference on Image Processing 2008*, ICIP 2008, pp. 1392-1395, 2008.

[76] J. A. Pérez-Carrasco, B. Acha, C. Serrano, L. Camuñas-Mesa, T. Serrano-Gotarredona, B. Linares-Barranco, "Fast vision through frame-less event-based sensing and convolutional processing. Application to texture recognition", *IEEE Transactions on Neural Networks*, accepted for publication.

[77] T. Serrano-Gotarredona, A. G. Andreou, and B. Linares-Barranco, "AER image filtering architecture for vision processing systems," *IEEE Trans. Circuits Syst. II, Anal. Digit. Signal Process.*, vol. 46, no. 9, pp. 1064–1071, Sep. 1999.

[78] R. Serrano-Gotarredona, T. Serrano-Gotarredona, A. Acosta-Jimenez, and B. Linares-Barranco, "A neuromorphic cortical-layer microchip for spike- based event processing vision systems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 53, no. 12, pp. 2548–2566, Dec. 2006.

[79] L. Camuñas-Mesa, A. Acosta-Jiménez, T. Serrano-Gotarredona and B. Linares-Barranco, "A convolution processor chip for address event vision sensors with 150ns event latency and 20Meps throughput", submitted to *Trans. Circuits Syst. I, Reg. Papers*, 2010.

[80] W. Gerstner, "Spiking neurons", in *Pulsed Neural Networks*, W. Maass and C. M. Bishop Editors, MIT Press, pp. 3-54, 1998.

[81] W. Maass, "Computing with spiking neurons", in *Pulsed Neural Networks*, W. Maass and C. M. Bishop Editors, MIT Press, pp. 55-85, 1999.

[82] R. W. Williams and K. Herrup, "The control of neuron number", *Annual Review on Neuroscience*, vol. 11, pp. 423-453, 1988.

[83] B. Pakkenberg and H. J. G. Gundersen, "Neocortical neuron number in humans: effect of sex and age", *Journal of Comparative Neurology*, vol. 384, pp. 312-320, 1997.

[84] A. L. Hodgkin and A. F. Huxley, "A quantitative description of ion currents and its applications to conduction and excitation nerve membranes", *Journal of Physiology*, vol. 117, pp. 500-544, London, 1952.

[85] W. Kistler, W. Gerstner and J. L. van Hemmen, "Reduction of Hodgkin-Huxley equations to a single-variable threshold model", *Neural Computation*, vol. 9, pp. 1015-1045, 1997.

[86] M. Alioto and G. Palumbo, "Analysis and comparison on full adder block in submicron technology", *IEEE Transactions of VLSI Systems*, vol. 10, no. 6, pp. 806-823, December 2002.

[87] J. Castro, P. Parra and A. J. Acosta, "Performance analysis of full adders in CMOS technologies", *Proceedings of SPIE'05*, VLSI Circuits and Systems II, Vol. 5837, pp. 339-348, Sevilla, May 2005.

[88] A. A. Fayed and M. A. Bayoumi, "A low power 10-transistor full adder cell for embeded architectures", *IEEE International Symposium on Circuits and Systems 2001*, pp. 226-229, 2001.

[89] J.-M. Wang, S.-C. Fang and W.-S. Feng, "New efficient designs for XOR and XNOR functions on the transistor level", *IEEE J. of Solid State Circuits*, vol. 29, no. 7, pp. 780-786, July 1994.

[90] R. Zimmermann and W. Fichtner, "Low-power logic styles: CMOS versus pass-transistor logic", *IEEE J. of Solid State Circuits*, vol. 32, no. 7, July 1997.

[91] N. Weste and K. Eshraghian, "Principles of CMOS VLSI design (A systems perspective)", 2nd Ed, Reading, MA, Addison Wesley, 1993.

[92] M. J. Bellido, A. J. Acosta, J. Luque, A. Barriga and M. Valencia, "Evaluation of metastability transfer models: an application to an N-bistable CMOS synchronizer", *Int. J. Electronics*, vol. 79, no. 5, 585-593, 1995.

[93] IEEE, "*IEEE Standard VHDL language reference manual*", IEEE standard 1076-1993 and standard 1076a-2000, 2000.

[94] R. Serrano-Gotarredona, "*AER-based bio-inspired architecture for real-time image convolutions*", Ph.D. Dissertation, Universidad de Sevilla, 2007.

[95] R. Serrano-Gotarredona et al., "On real-time AER 2-D convolution hardware for neuromorphic spike-based cortical processing", *IEEE Transactions on Neural Networks*, vol. 19, Issue 7, pp. 1196-1219, 2008.

[96] A. Linares-Barranco, B. Linares-Barranco, G. Jiménez-Moreno and A. Civit-Balcells, "AER synthetic generation in hardware for bio-inspired spiking systems", *Proceedings of SPIE: Bioengineered and Bioinspired Systems II*, vol. 5839, pp. 103-110, June 2005.

[97] A. Linares-Barranco, B. Linares-Barranco, G. Jiménez-Moreno and A. Civit-Balcells, "Synthetic generation of events for Address-Event-Representaion communications", *Lecture notes in computer science (Lecture notes in artificial intelligence)*, vol. 2451, no. 1, pp. 371-379, 2002.

[98] M. Rivas-Pérez et al., "AER tools for AER bio-inspired spikikng systems", *Proceeding of Symposium on Computational Intelligence, Sico 2005 (IEEE Computational Intelligence Society, SC)*, vol. 27, pp. 341-348, 2005.

List of publications

Articles in International Journals:

[1] R. Serrano-Gotarredona, L. A. Camuñas-Mesa, T. Serrano-Gotarredona, J. A. Leñero-Bardallo and B. Linares-Barranco, "The stocastic I-Pot: a circuit block for programming bias currents", *IEEE Transactions on Circuits and Systems Part 2: Analog and Digital Signal Processing*, vol. 54, nº9, pp. 760-764, 2007.

[2] R. Serrano-Gotarredona, M. Oster, P. Lichsteiner, A. Linares-Barranco, R. Paz-Vicente, F. Gómez-Rodríguez, L. Camuñas-Mesa, R. Berner, M. Rivas, T. Delbrück, S. C. Liu, R. Douglas, P. Häfliger, G. Jiménez-Moreno, A. Civit, T. Serrano-Gotarredona, A. Acosta-Jiménez and B. Linares-Barranco, "CAVIAR: a 45k-neuron, 5M-synapse, 12G-connects/sec AER hardware sensory-processing-learning-actuating systema for high speed visual object recognition and tracking", *IEEE Transactions on Neural Networks*, vol. 20, Issue 9, pp. 1417-1438, 2009.

[3] J. Pérez-Carrasco, B. Acha, C. Serrano, L. Camuñas-Mesa, T. Serrano-Gotarredona and B. Linares-Barranco, "Fast vision through frame-less event-based sensing and convolutional processing. Application to texture recognition", *IEEE Transactions on Neural Networks*, 2010.

[4] L. Camuñas-Mesa, A. Acosta-Jiménez, T. Serrano-Gotarredona and B. Linares-Barranco, "A convolution processor chip for address event vision sensors with 155ns event latency and 20Meps throughput", submitted to *IEEE Transactions on Circuits and Systems Part 1*, 2010.

Contributions to International Conferences:

[5] B. Linares-Barranco, T. Serrano-Gotarredona, R. Serrano-Gotarredona and L. A. Camuñas, "On leakage current temperature characterization using sub-pico-ampere circuit techniques", *Proceedings of IEEE International Symposium on Circuits and Systems ISCAS 2004*, vol. 1, pp. 361-364, 2004.

[6] L. Camuñas-Mesa, A. Acosta-Jiménez, T. Serrano-Gotarredona and B. Linares-Barranco, "A digital pixel cell for address event representation image convolution processing", *Proceedings of the International Symposium SPIE, Microtecnologies for the New Millenium 2005, Bioengineered and Bioinspired*, 5839-20, 2005.

[7] L. Camuñas-Mesa, A. Acosta-Jiménez, T. Serrano-Gotarredona and B. Linares-Barranco, "On fully digital AER convolution processing", *Proceedings of XX Conference on Design of Circuits and Integrated Systems*, DCIS 2005.

[8] R. Serrano-Gotarredona, T. Serrano-Gotarredona, A. J. Acosta-Jiménez, B. Linares-Barranco and L. A. Camuñas-Mesa, "A bio-inspired event-based real-time image processor", *Proceedings of International Conference on Biomedical Robotics and Biomechatronics, BioRob 2006*, pp. 1206-1211, 2006.

[9] L. A. Camuñas-Mesa, A. J. Acosta-Jiménez, T. Serrano-Gotarredona, B. Linares-Barranco and R. Serrano-Gotarredona, "Image processing architecture based on a fully digital AER convolution chip", *Proceedings of XXII Conference on Design of Circuits and Integrated Systems*, DCIS 2007, pp. 385-390, 2007.

[10] J. A. Leñero-Bardallo, R. Serrano-Gotarredona, L. A. Camuñas-Mesa, T. Serrano-Gotarredona and B. Linares-Barranco, "The stocastic I-Pot: a circuit block for programming bias currents", *Proceedings of XXII Conference on Design of Circuits and Integrated Systems*, DCIS 2007, pp. 430-435, 2007.

[11] L. A. Camuñas-Mesa, A. Acosta-Jiménez, T. Serrano-Gotarredona and B. Linares-Barranco, "Fully digital AER convolution chip for vision processing", *Proceedings of IEEE International Symposium on Circuits and Systems ISCAS 2008*, pp. 652-655, 2008.

[12] L. Camuñas-Mesa, J. A. Pérez-Carrasco, C. Zamarreño-Ramos, T. Serrano-Gotarredona and B. Linares-Barranco, "On Scalable Spiking ConvNet Hardware for Cortex-Like Visual Sensory Processing Systems", accepted for publication in *Proceedings of IEEE International Symposium on Circuits and Systems ISCAS* 2010.

[13] L. Camuñas-Mesa, J. A. Pérez-Carrasco, C. Zamarreño-Ramos, T. Serrano-Gotarredona and B. Linares-Barranco, "Neocortical frame-free vision sensing and processing through scalable

spiking ConvNet hardware", accepted for publication in *Proceedings of the International Joint Conference on Neural Networks* 2010.