UNIVERSIDAD DE SEVILLA

Departamento de Electrónica y Electromagnetismo

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.

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

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

Memoria presentada por

LUIS ALEJANDRO CAMUÑAS MESA

Director

Dr. Bernabé Linares Barranco

Codirectores

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

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

Introducción

1.1. Antecedentes

Durante los últimos tiempos, la capacidad de procesamiento de los sistemas informáticos se ha incrementado hasta alcanzar cotas que hasta hace poco resultaban inimaginables. Esto ha permitido diseñar sistemas artificiales capaces de llevar a cabo tareas cada vez más complejas. Sin embargo, en el ámbito de la percepción sensorial, y en particular en algunas aplicaciones como las relativas al reconocimiento de objetos a partir de la información visual por parte del cerebro humano ponen de relieve una importante limitación: a pesar de trabajar con unidades básicas de procesamiento mucho más rápidas que las neuronas (los sistemas elecrónicos actuales tienen tiempos característicos del orden de los nanosegundos, mientras que las neuronas biológicas trabajan con milisegundos, lo que supone una diferencia de 6 órdenes de magnitud), los sistemas artificiales no consiguen llevar a cabo este tipo de tareas a la misma velocidad que los sistemas biológicos.

El principal motivo de la ventaja de los sistemas biológicos frente a los artificiales está en su arquitectura, masivamente paralela, los esquemas y métodos de codificar la información sensorial, así como las técnicas de procesamiento. Mientras que los sistemas informáticos convencionales están

programados para seguir algoritmos que implican la ejecución de una secuencia de instrucciones una detrás de otra, el cerebro sigue una estrategia de procesamiento completamente diferente. Se estima que el cerebro está formado por una cantidad de neuronas del orden de 10^{11} [82], con una cantidad de interconexiones entre ellas de alrededor de 10^{15} [83]. Esta estructura hace que el cerebro sea especialmente eficiente para llevar a cabo tareas basadas en computación paralela.

Además de realizar un procesamiento secuencial, los sistemas de visión artificiales convencionales operan tradicionalmente de un modo diferente al cerebro, ya que están basados en la captura y procesado de secuencias de fotogramas. Así, con una determinada frecuencia de muestreo, los sistemas tradicionales de visión obtienen una secuencia de imágenes, cada una de las cuales es procesada para extraer algún tipo de característica que, combinándola con otras diferentes, proporcione un resultado final de la operación de reconocimiento. Sin embargo, los sistemas biológicos no se basan en el procesamiento de fotogramas, sino de eventos. Se denomina eventos a los impulsos eléctricos generados por la retina y enviados al córtex cerebral cada vez que su nivel de actividad alcanza un determinado umbral. Este nivel de actividad se corresponde con diferentes propiedades de la imagen, como intensidad, contraste, movimiento, etc. Así, los píxeles más activos de la imagen generarán un mayor número de eventos, de modo que cuando estos eventos son procesados por el córtex cerebral, éste es capaz de detectar inmediatamente patrones gracias a la correlación espacial que presentan los eventos generados a partir de un objeto moviéndose frente a la retina. Todo este procesamiento se produce conforme los eventos se van generando, sin esperar a que transcurra ningún tiempo de muestreo artificial como hacían los sistemas basados en fotogramas, lo que permite una gran rapidez para obtener resultados.

En la Fig. 1.1 podemos ver un ejemplo de cómo funciona un sistema de procesamiento de visión multicapa basado en eventos. En él se muestra una retina de movimiento que genera una serie de eventos en los píxeles correspondientes al contorno de un objeto. Estos eventos son enviados a una primera capa de procesamiento. Cada capa de procesamiento está formada por una serie de unidades en paralelo que en general se encargan de extraer una determinada característica. Para extraer estas características, se aprovecha la correlación espacial entre los eventos y se implementa una operación de

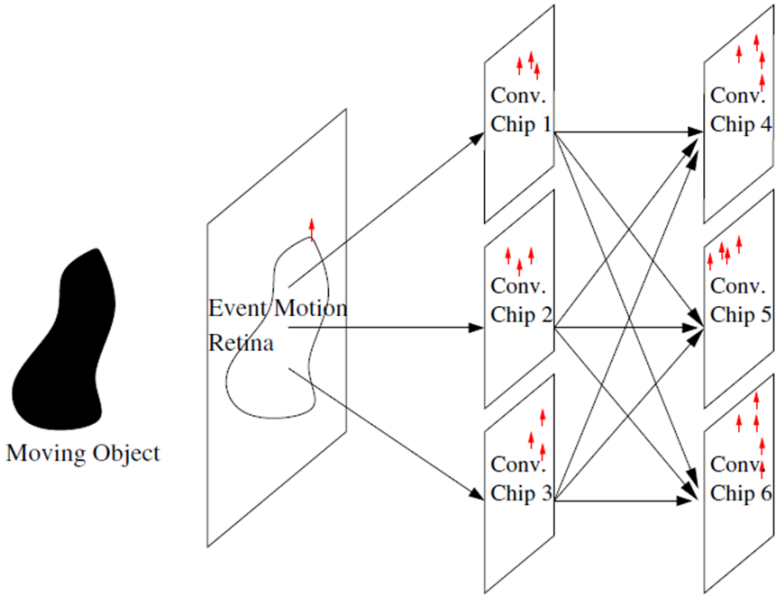


FIGURA 1.1. Ejemplo de sistema de procesamiento de visión multicapa basado en eventos.

convolución bidimensional. De este modo, en la primera capa de procesamiento de la Fig. 1.1 se realizan una serie de convoluciones en paralelo, cada una de ellas buscando una determinada característica espacial, de forma que como resultado de esta operación producen eventos de salida. Cada uno de los eventos generados por los distintos convolucionadores de la primera capa se envía a otra serie de convolucionadores en paralelo que forman la segunda capa de procesamiento. En esta segunda capa, se combinan los resultados producidos por la capa anterior para generar más eventos de salida. Siguiendo esta estructura de procesamiento, el córtex cerebral realiza operaciones de reconocimiento de formas en unas 8 ó 10 capas.

Una característica muy interesante de estos sistemas es la simultaneidad, como se puede ver representada en la Fig. 1.1. Conforme un píxel de la retina genera un evento, éste se propaga a través de las diferentes capas de procesamiento, produciendo resultados de forma prácticamente instantánea. Esta característica hace que los sistemas de procesamiento basados en eventos tengan la potencialidad de procesar imágenes a una gran velocidad.

Al emular el comportamiento del cerebro mediante una estructura de sistemas convolucionadores como se muestra en la Fig. 1.1, nos encontramos una importante limitación. Cada uno de los convolucionadores incluye una gran cantidad de unidades de procesamiento (píxeles), cada uno de los cuales necesita comunicarse con todos los píxeles de los convolucionadores de la siguiente capa, para así poder imitar la estructura del cerebro. Sin embargo, los sistemas electrónicos imponen ciertas limitaciones físicas a la hora de interconectar poblaciones de píxeles del orden de 10^3 integradas en distintos chips. Para eso utilizamos el protocolo AER (Address Event Representation). Gracias a este protocolo, una gran cantidad de neuronas integradas en un chip pueden comunicarse con las neuronas de otro chip multiplexando las conexiones en un único bus digital asíncrono. Así, los eventos generados por cada neurona se propagan a través de buses AER entre las distintas capas de los sistemas de procesamiento.

1.2. Objetivos

En este trabajo, se presentan dos versiones diferentes de microchips convolucionadores completamente digitales basados en el protocolo AER para sistemas de procesamiento visual basados en eventos. Estos chips constituyen la unidad básica para construir sistemas complejos multicapa a partir de la interconexión en serie y en paralelo de diferentes muestras de los mismos. Cada uno de ellos permite realizar convoluciones con kernel programable de un tamaño máximo de 32×32.

La primera versión de chip Conv1 opera sobre un array de píxeles de tamaño 32×32, aunque está diseñado para poder construir sistemas equivalentes de mayor resolución conectando varias muestras en paralelo. La segunda versión Conv2 cuenta con un tamaño 4 veces mayor, 64×64 píxeles, y además implementa la funcionalidad multikernel. Esta funcionalidad permite programar varios kernels diferentes dentro de un mismo chip (hasta 32) para que éste pueda recibir eventos de varios chips diferentes de una capa anterior, y aplicarle a cada uno de ellos un kernel diferente en función de su origen. Esto está especialmente indicado para facilitar la implementación de sistemas multicapa, a imitación de la corteza cerebral, y siguiendo

las estructuras típicas del paradigma conocido como "*Convolutional Neural Networks*".

En el presente documento se describen detalladamente las arquitecturas de cada una de las dos versiones propuestas de chips de convolución, así como algunos resultados experimentales obtenidos.

1.3. Estructura del documento

El documento está estructurado de la siguiente forma. En primer lugar, en el Capítulo 2 se presentan las ventajas de los sistemas de procesamiento visual basados en eventos, frente a los tradicionales sistemas basados en fotogramas. Este capítulo incluye también una descripción del protocolo AER, imprescindible para construir sistemas de procesamiento por eventos. El Capítulo 3 hace un repaso sobre las estructuras multicapa bioinspiradas de procesamiento de imagen, justificando el uso de la operación de convolución como unidad básica de este tipo de sistemas, describiendo también la arquitectura propuesta para los chips de convolución basados en AER. En el Capítulo 4 se describe en detalle el píxel de convolución como base de los chips propuestos, analizando las dos versiones diferentes, mientras que en el Capítulo 5 se detallan el resto de circuitos periféricos incluidos en los chips de convolución. El Capítulo 6 muestra exhaustivos resultados experimentales obtenidos a partir de las dos versiones Conv1 y Conv2. Por último, el Capítulo 7 presenta las conclusiones.

CAPÍTULO 2

Sistemas de procesamiento basados en eventos

2.1. Introducción

A la hora de hacer procesamiento de visión en tiempo real, los sistemas tradicionales basados en fotogramas tienen importantes limitaciones. El principal motivo de estas limitaciones es la propia naturaleza secuencial de sensado y procesado fotograma a fotograma. Si comparamos los sistemas artificiales de procesamiento de imagen con la forma de llevar a cabo las mismas tareas por parte del cerebro humano, podemos sacar una conclusión interesante: la unidad de procesamiento básica del cerebro, la neurona, es mucho más lenta que cualquier ordenador a la hora de hacer una operación sencilla, pero aun así, el cerebro es mucho más eficiente gracias a su modo de funcionamiento basado en el paralelismo.

Mientras que las representaciones clásicas de imágenes por fotogramas tienen un carácter secuencial (es decir, hasta que no se termina de ejecutar una tarea sobre un fotograma completo no se puede pasar a la siguiente tarea), los sistemas biológicos se basan en la ejecución en paralelo de muchas actividades diferentes. Para implementar este paralelismo, cada neurona envía pulsos de información a otras muchas neuronas simultánea-

mente. De este modo, la información ya no se procesa en forma de fotogramas, sino en forma de pulsos, también llamados eventos.

Por otra parte, para que el sistema basado en eventos sea eficiente es necesario que haya una masiva interconexión entre neuronas que permita que la información se procese en paralelo. Para permitir esa gran capacidad de interconexión entre neuronas se utiliza el protocolo AER *(Address Event Representation)*, que permite que dos grandes poblaciones de neuronas se comuniquen entre sí multiplexando las conexiones a través de un único bus común.

En este capítulo se justifica la utilización de sistemas de procesamiento de imagen basados en eventos. Para ello, en la Sección 2.2 se describen los sistemas tradicionales basados en fotogramas, mientras que en la Sección 2.3 se detallan los sistemas de procesamiento por eventos, poniendo especial énfasis en las principales ventajas que presenta sobre el anterior. Por último, en la Sección 2.4 se desarrolla el protocolo AER.

2.2. Representación visual basada en fotogramas

Al tratar sobre el procesamiento de imágenes, una de las primeras cuestiones que tenemos que considerar es de qué forma se representan las imágenes para poder trabajar con ellas. En el caso más genérico de imágenes en movimiento, el concepto de fotograma se encuentra tan arraigado a los sistemas de visión que con frecuencia se da por supuesto, ya que tradicionalmente ha sido así.

La imagen en el mundo real es continua tanto en el espacio como en el tiempo, así que el primer paso es muestrear en el tiempo, es decir, capturar imágenes estáticas a intervalos regulares [1]. Cada una de estas imágenes es un fotograma. De este modo, se capturan fotogramas a intervalos regulares y al reproducirlos todos seguidos producen para el ojo humano la sensación de movimiento, como muestra la Fig. 2.1. Para ello es fundamental una buena elección del tiempo de muestreo T_s. De forma habitual se usa la frecuencia de fotograma $f_s = 1/T_s$, expresada en fotogramas por segundo (fps)

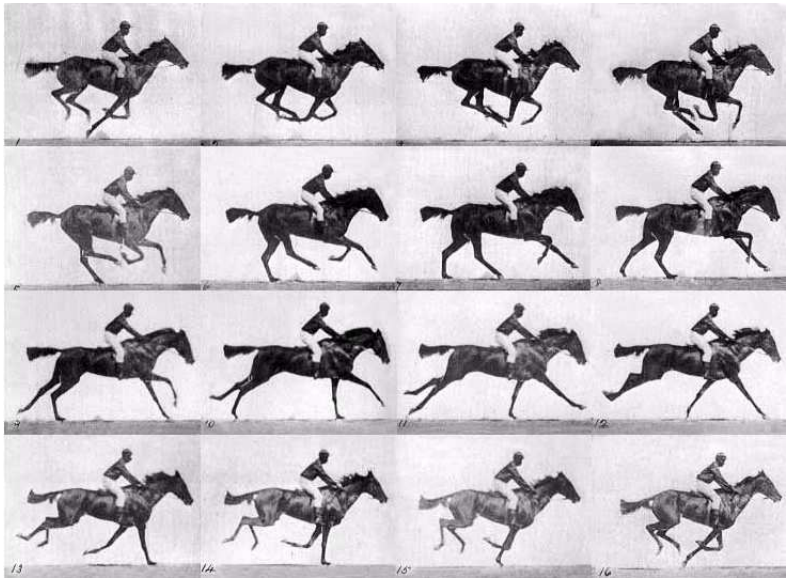


FIGURA 2.1. Secuencia de un caballo de carreras galopando publicada por Eadweard Muybridge en 1887 en Philadelphia [2]. Con 16 fotogramas recrea la sensación de movimiento al reproducirlas todas seguidas.

o Hertzios. Las frecuencias de fotograma de algunos de los sistemas más conocidos van desde los 16-18 Hz que se empezaron a utilizar durante los tiempos del cine mudo hasta los 24 Hz usados en el cine actual. En cuanto a la televisión, existen varios estándares de codificación. Tanto el sistema PAL *(Phase Alterning Line)* [3] como el SECAM *(Séquentiel Couleur à Mémoire)*, utilizados en Europa, Asia, Africa, Oceanía y parte de Sudamérica, muestrean a 25 Hz, que es la mitad de la frecuencia de la corriente eléctrica usada en estos lugares (50 Hz). Sin embargo, el sistema NTSC *(National Television System Committee)* [4] extendido en la mayor parte de América y Japón tiene una frecuencia de muestreo de 29.97 Hz, que es prácticamente la mitad de la frecuencia de la corriente eléctrica que en estos países es de 60 Hz.

De este modo, los sistemas tradicionales de procesamiento de imagen basados en fotogramas se comportan conceptualmente según se indica en la Fig. 2.2. La cámara o sensor captura una imagen estática cada $T_{fotograma}$, almacenando la información de cada uno de los píxeles. Cuando se trata de

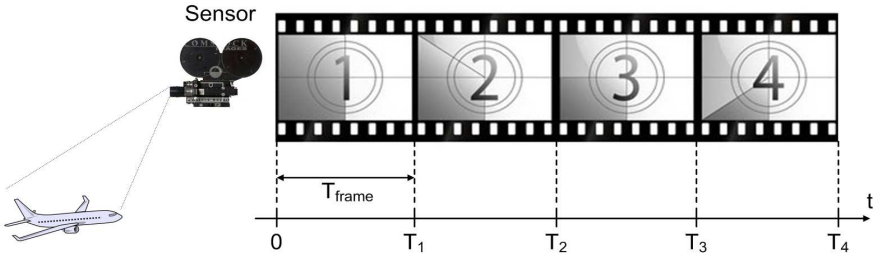


FIGURA 2.2. Descripción conceptual del procesamiento de imágenes basado en fotogramas.

utilizar estos fotogramas como entradas de un sistema de procesamiento, nos encontramos una serie de limitaciones. En primer lugar, se pierde la información de cualquier cambio que ocurra entre dos instantes de muestreo T_i, haciendo imposible la aplicación a un sistema que se mueva a una velocidad mayor que el propio tiempo de muestreo. Por otra parte, en el caso de que la mayor parte de la imagen no cambie entre dos instantes de muestreo (o incluso la imagen entera permanezca invariable), el sistema va a capturar y procesar un fotograma completo, aunque de ello no obtenga ninguna información. Así pues, los sistemas de procesamiento basados en fotogramas no sirven para aplicaciones de alta velocidad, pero además son enormemente ineficientes para aplicaciones de baja velocidad. Una consecuencia de esta ineficiencia es que en cada instante de muestreo hay que procesar una gran cantidad de información (la imagen completa), lo que puede acarrear una considerable carga computacional. Sin embargo, para una aplicación en tiempo real todo el procesamiento tiene que haber terminado antes de que se capture el siguiente fotograma, limitando también las posibilidades de llevar a cabo una computación más compleja sobre la imagen de entrada.

2.3. Representación visual basada en eventos

Los cerebros biológicos no procesan la visión fotograma a fotograma, sino que están basados en eventos [5]. En una retina, cada píxel envía un pulso (también llamado evento) al córtex cerebral cuando su nivel de actividad alcanza un umbral, de modo que la información se transmite conforme

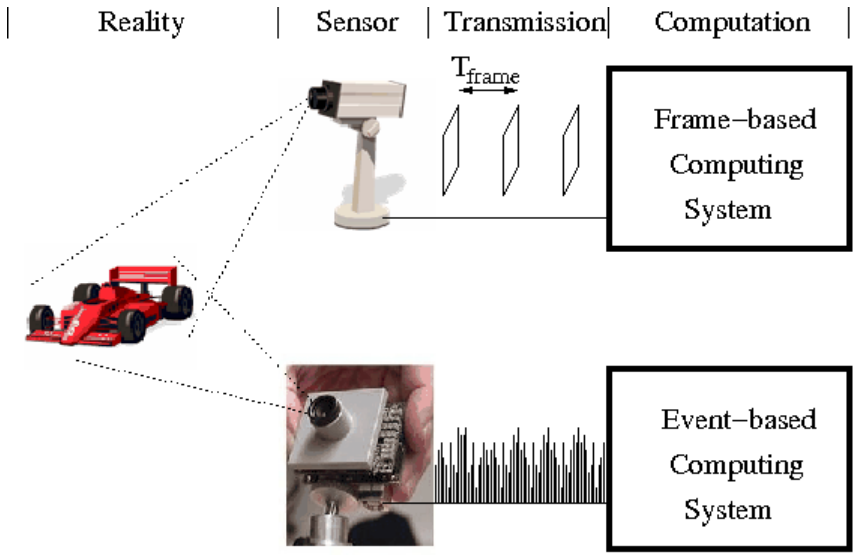


FIGURA 2.3. Comparación a nivel conceptual entre el sensado y procesamiento de imagen basado en fotogramas y basado en eventos.

se produce, sin esperar que llegue un instante de muestreo artificial que no guarda ninguna relación con la realidad. De este modo, una retina se encarga de sensar una determinada propiedad (que puede ser por ejemplo un cambio en la intensidad [6] o el contraste espacial [7]), y cuando detecta un nivel determinado de esa propiedad en un píxel cualquiera envía un evento (que habitualmente incluye la información de qué píxel lo ha generado). Así, cada vez que se produce un evento se actualiza el estado de todo el sistema, aunque esto sólo afecta a las partes de la imagen que aportan alguna información, evitando la carga computacional innecesaria.

Observando la Fig. 2.3 se puede entender con facilidad cómo se lleva a cabo el proceso de sensado en un sistema basado en eventos. Mientras que la cámara de la parte superior de la imagen captura un fotograma a intervalos regulares de T_{frame}, el sensor de la parte inferior genera eventos de salida de forma continua, codificando de este modo la información de lo que está ocurriendo en cada momento en una zona cualquiera de la imagen. Así, el sistema de computación actualiza su estado después de cada evento, efectuando operaciones de menor carga computacional que en el caso del

sistema basado en fotogramas, ya que los eventos sólo afectan en general a una parte reducida de la imagen.

Para entender las ventajas de los sistemas de procesamiento de imágenes basados en eventos, lo mejor es comparar su comportamiento temporal con los sistemas basados en fotogramas, tal como vemos a continuación.

2.3.1. Ventajas del sistema basado en eventos

La gran ventaja inherente a los sistemas de procesamiento basados en eventos se encuentra en el hecho de que la información más relevante se envía (y por lo tanto se procesa) en primer lugar. Esto ocurre para las distintas formas de codificar la información en eventos [8], [9]. Una posibilidad es que la información se codifique en el orden en el que se producen los eventos, es decir, la neurona que envía un evento en primer lugar es la más activa, lo que significa que su entrada es la más intensa. Otras opciones serían codificar la información en la tasa de eventos producidos por una neurona, o en el retraso de los eventos respecto a un tiempo periódico de referencia. En cualquier caso, los primeros eventos generados siempre serán aquellos pertenecientes a las neuronas cuyas entradas sean más intensas. De este modo, los primeros pulsos enviados codifican el fragmento más importante de la información, lo que implica que se puede hacer un procesamiento aproximado en un intervalo de tiempo realmente pequeño, tomando sólo unos pocos eventos iniciales. Esto es algo que el procesamiento por fotogramas no permite, ya que siempre procesa imágenes completas.

Esta ventaja se puede apreciar mejor observando la comparación entre el comportamiento temporal de un sistema basado en fotogramas y uno basado en eventos, tal como muestra la Fig. 2.4. En la parte superior de la imagen, observamos cómo responde un sistema de sensado y procesamiento basado en fotogramas ante un suceso producido en el intervalo temporal entre 0 y T_1. Debido a la propia naturaleza del procesamiento basado en fotogramas, independientemente del instante en el que se produzca el suceso, la información producida no llega al sistema de computación hasta que el fotograma completo está totalmente disponible en el instante T_1, con un retraso añadido Δ debido al tiempo de transmisión. Sólo a partir de ese momento el sistema de computación comienza a procesar la información

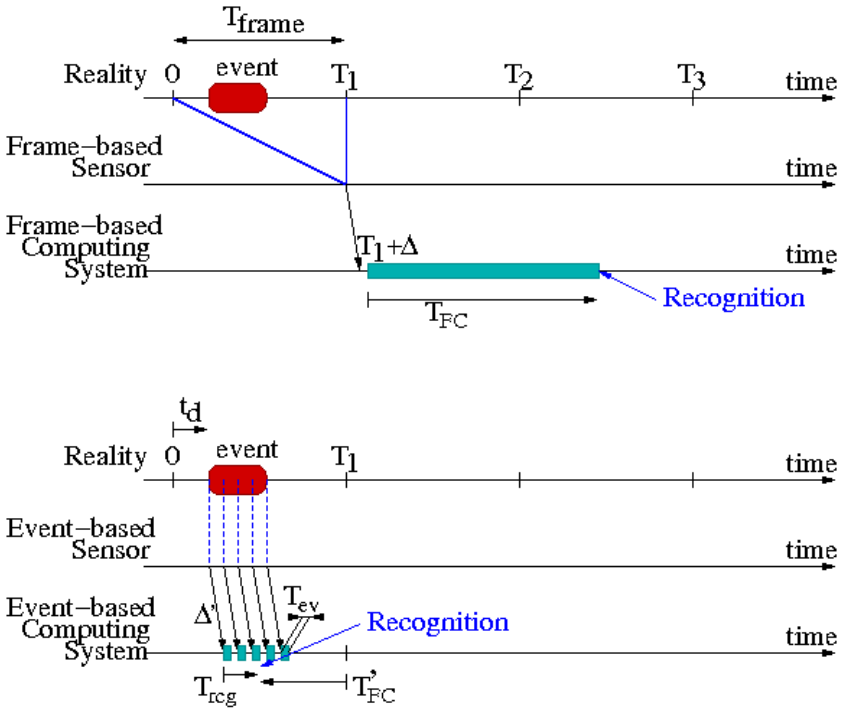


FIGURA 2.4. Comparación de la respuesta temporal entre un sistema basado en fotogramas y uno basado en eventos.

recibida, lo cual llevará un tiempo relativamente largo T_{FC} debido a la gran cantidad de información innecesaria incluida en el fotograma. Sólo entonces se obtiene la información deseada de reconocimiento. Sin embargo, observando en la parte inferior el sistema basado en eventos, vemos cómo cada píxel detecta inmediatamente un cambio en la realidad y lo envía al sistema de computación con un retraso Δ'. Cada evento tarda un tiempo T_{ev} muy pequeño (del orden de los nanosegundos) en ser procesado. Además, dado que los primeros eventos son los que portan la principal información, ni siquiera es necesario procesar todos los eventos para que el sistema de computación pueda realizar el reconocimiento deseado, obteniendo un resultado mucho más rápido que el sistema anterior, incluso antes del instante de muestreo T_1.

2.3.2. Tipos de codificación

Una vez descrito en qué consisten los sistemas de procesamiento basados en eventos, así como las grandes ventajas que presentan, habría que desarrollar la siguiente cuestión: ¿cómo se codifica la información visual en dichos eventos? Sobre este tema se podrían plantear gran cantidad de alternativas, y aquí vamos a describir algunas de las más importantes:

1. Codificación por frecuencia de eventos. Considerando que estamos tratando con un sensor que detecta la intensidad luminosa, este tipo de codificación consiste en generar un tren de eventos para cada píxel de frecuencia proporcional a la intensidad de dicho píxel. Esto implica una desventaja, y es que a la hora de decodificar es necesario integrar una cierta cantidad de eventos para obtener la información relativa a la intensidad.
2. Codificación por tiempo hasta el primer evento. Consiste en que, después de recibir un reset global, cada píxel genera un evento tras un intervalo de tiempo que depende de la intensidad de dicho píxel, de modo que la información se codifica en el tiempo transcurrido entre el reset y el primer evento. Los píxeles más activos generan su evento primero y los menos activos más tarde. Aprovechando la característica del procesamiento basado en eventos que hace que los píxeles más activos (los que tienen mayor información) disparen primero, se plantea este tipo de codificación, en el cual se evita tener que integrar gran cantidad de eventos para extraer la información relativa a cada píxel. Por el contrario, es suficiente con recibir un solo evento de cada píxel, ya que la información se encuentra simplemente en el tiempo transcurrido hasta el primer evento. El inconveniente que presenta este sistema es que al necesitar un reset global se introduce un “frame”, igual que en los sistemas basados en fotogramas.
3. Codificación por orden de eventos. Este sistema consiste (igual que el anterior) en que, después de recibir un reset global, cada píxel genera un evento tras un tiempo marcado por su nivel de intensidad. Sin embargo, la información no se codifica en la temporización de ese evento, sino que los eventos de todos los píxeles se ordenan en función de su instante de generación, de modo que la información va en el orden de dichos eventos. Así, este sistema de codificación elimina la información temporal de los eventos, quedándose sólo con el orden en que son generados. De este

modo, los píxeles más activos dispararán antes, obteniéndose en definitiva un orden de actividad para todos los píxeles de la imagen. Esto simplifica la codificación, pero como contrapartida pierde información, además de introducir un cierto "frame" asociado al reset global.

4. Codificación por variación de la intensidad. Este sistema de codificación genera para cada píxel un pulso si su intensidad ha cambiado un cierto umbral o factor desde el pulso anterior.

2.4. El protocolo Address Event Representation (AER)

Una de las principales causas de la gran eficiencia del procesamiento visual de los sistemas neurológicos es la masiva interconexión existente entre las neuronas, que hace que una neurona pueda estar conectada a miles de neuronas, consiguiendo así que un evento generado por una de ellas sea el estímulo de muchas otras. Sin embargo, para los sistemas electrónicos ésta podría ser una gran limitación.

En general, tratando de neuronas artificiales, podemos considerar dos casos diferentes: que queramos interconectar una población de neuronas dentro de un mismo circuito integrado, o bien conectar entre sí poblaciones de neuronas pertenecientes a diferentes chips. Con las tecnologías actuales, teniendo en cuenta que la complejidad de una neurona puede variar bastante, dentro de un chip se pueden integrar del orden de miles de neuronas o incluso millones. Sin embargo, las capas de metales disponibles para implementar interconexiones no suelen ser más de 7 u 8, lo cual haría imposible una comunicación directa entre todas las neuronas. Por otra parte, si queremos conectar poblaciones de neuronas incluidas en chips diferentes, el número de pines disponibles en cada encapsulado está limitado a algunos cientos como máximo, luego sería imposible que todas las neuronas integradas en su interior pudieran comunicarse de forma directa. No obstante, los circuitos integrados tienen una importante ventaja sobre los sistemas biológicos que merece la pena explotar, y se trata de su velocidad. Mientras que los tiempos característicos de los pulsos generados por las neuronas biológicas están en el orden de los milisegundos, los circuitos electrónicos nos per-

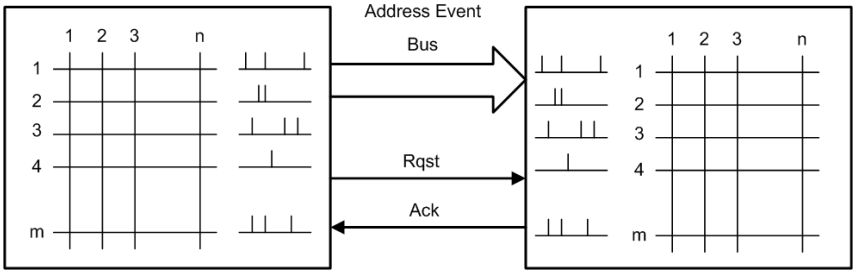


FIGURA 2.5. Esquema de una comunicación AER entre dos poblaciones de neuronas.

miten tratar con tiempos del orden de los nanosegundos. Esta cualidad nos permite multiplexar las salidas de gran cantidad de neuronas por una única conexión física, utilizando el protocolo AER (*Address Event Representation*).

El protocolo AER fue propuesto por primera vez en Caltech en 1991 por Sivilotti [10] y en 1992 por Mahowald [11]. Con él se permite la interconexión masiva punto a punto entre dos poblaciones de neuronas tal y como se muestra en la Fig. 2.5. En ella podemos ver dos arrays genéricos de $m \times n$ neuronas interconectados mediante un único bus digital de $\log_2(m \times n)$ bits más un par de líneas de *handshaking* para implementar la comunicación asíncrona. De este modo, cada una de las neuronas pertenecientes al sistema emisor producirá eventos a una tasa reducida del mismo orden que las neuronas biológicas (milisegundos), y al compartir el bus digital con las $(m \times n)$ neuronas se obtiene una tasa de generación de eventos en el bus mucho más alta. Así, cada evento incluye la identificación de la neurona de origen (su dirección), y al llegar al sistema receptor éste puede reproducir el estado de cada una de las neuronas emisoras en tiempo real.

La comunicación asíncrona AER sigue un protocolo de handshaking de 4 fases como el que se representa en la Fig. 2.6. Una vez que el chip emisor tiene el dato válido puesto en el bus, activa la señal *Request*, de modo que el chip receptor sepa que el dato está disponible, y una vez que captura dicho dato activa la señal *Acknowledge*. Para terminar con la comunicación,

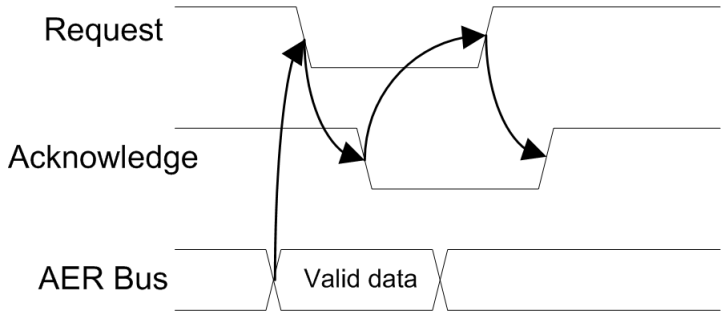


FIGURA 2.6. Diagrama temporal del protocolo AER

el chip emisor desactiva el *Request* cuando recibe el *Acknowledge*, que a su vez es desactivado a continuación por el receptor.

Partiendo de este protocolo genérico, según la forma de codificar la información en los eventos se pueden desarrollar gran cantidad de aplicaciones, siendo ésta una de las grandes virtudes de AER. Dentro de sistemas de sensado y procesamiento de imagen, se puede codificar el nivel de intensidad luminosa en la frecuencia de los eventos [6] o en el tiempo transcurrido hasta el primer evento [12]-[15], se pueden diseñar detectores de características visuales más elaboradas [16]-[18], o sistemas de sensado y computación de movimiento [19]-[23]. Por otra parte, además de en sistemas de visión, AER también se ha usado para sistemas de audición [24]-[27], para redes *winner-takes-all* [28]-[30], o incluso para sistemas distribuidos sobre redes inalámbricas [31]. Algunos sistemas genéricos como [32] procesan los eventos AER de forma independiente a la codificación utilizada sobre la información de entrada, siendo posible su aplicación a diferentes tipos de arquitecturas bioinspiradas.

Según la forma en la que está definido, el protocolo AER tiene que enviar por el bus digital compartido eventos generados por distintas neuronas, lo cual implica la posibilidad de colisiones. Ante una colisión entre eventos generados por sus respectivas neuronas simultáneamente, un sistema AER puede reaccionar de dos formas diferentes: descartando los eventos en colisión, como los descritos en [33]-[35], o bien arbitrando entre eventos simultáneos, como los descritos en [36]-[43]. Los primeros tienen la ventaja de ser más rápidos (al no añadir ningún tipo de procesamiento

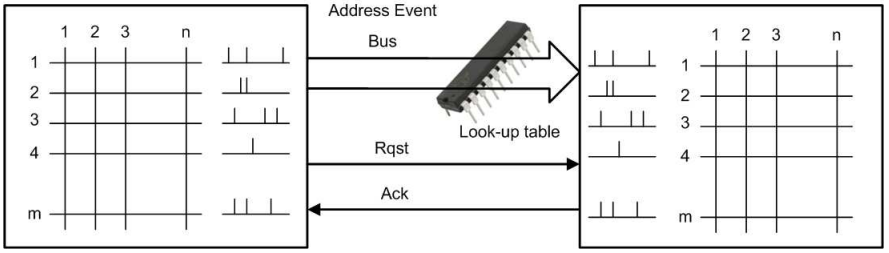


FIGURA 2.7. Transformación de imágenes en tiempo real a través del protocolo AER

extra a la transmisión de eventos) y de no alterar la temporización (ya que los eventos se transmiten conforme se generan), aunque tienen la desventaja de que algunos de los eventos se pierden. Sin embargo, los segundos aseguran que ningún evento se va a perder, aunque ralentizan la comunicación introduciendo una etapa de arbitración y alteran la temporización de los eventos. Las ventajas y desventajas de ambas estrategias serán más o menos críticas en función de la aplicación. Por ejemplo, si la tasa de generación de eventos es muy alta, la probabilidad de que se produzcan colisiones crece, lo cual aumentaría la cantidad de eventos perdidos en un sistema sin arbitración. Sin embargo, también habría que valorar la importancia de esa pérdida de eventos puntuales para el funcionamiento global del sistema.

2.4.1. Ventajas de AER

El protocolo AER presenta una serie de ventajas adicionales para su aplicación a los sistemas multi-chip de procesamiento de imagen. En primer lugar, permite aplicar transformaciones sencillas sobre las imágenes conforme se transmiten entre dos chips, como se muestra en la Fig. 2.7. Mediante la inserción de una simple memoria PROM en el bus AER se puede programar una *look-up table* que transforme la dirección de cada evento implementando traslaciones y rotaciones de la imagen en tiempo real sin ningún coste computacional.

Las otras grandes ventajas del protocolo AER están relacionadas con la capacidad de implementar sistemas complejos multi-emisor y multi-receptor con muchos chips compartiendo el bus digital, lo cual resulta de gran

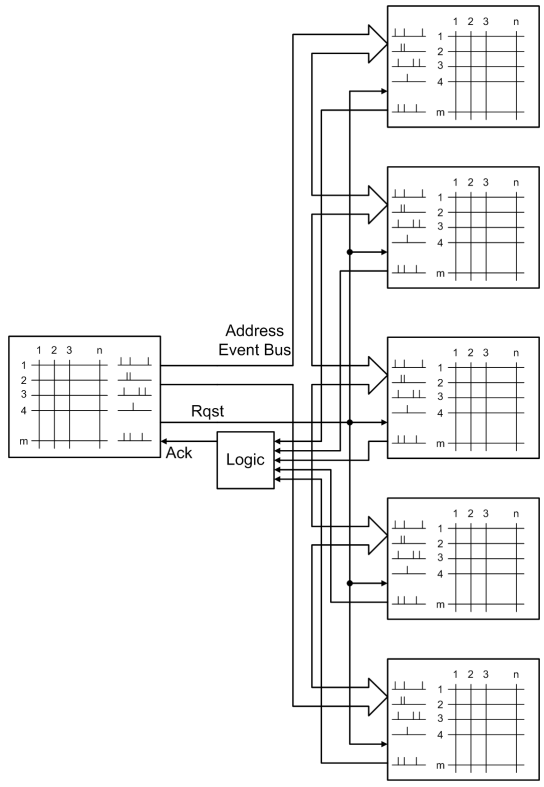


FIGURA 2.8. Arquitectura de un sistema AER multi-receptor

utilidad para construir sistemas multicapa. Comencemos por describir el sistema multi-receptor, que conlleva menos complicaciones.

En la Fig. 2.8 podemos ver la arquitectura de un sistema multi-receptor, en el cual varios receptores comparten un mismo bus AER. Si extendiéramos directamente el protocolo ocurriría que el emisor quitaría el dato del bus común en cuanto recibiera la respuesta de uno de los receptores (el más rápido de ellos), provocando que los demás capturaran un dato erróneo. Para evitar esta situación, tenemos que forzar al emisor a esperar hasta que el emisor más lento haya respondido, de modo que sólo en ese caso podrá retirar el *Request*. Además, tenemos que asegurarnos de que el emisor no enviará un nuevo *Request* hasta que todos los receptores no hayan retirado su *Acknowledge*, para no confundir ambas respuestas. Para ello hay que

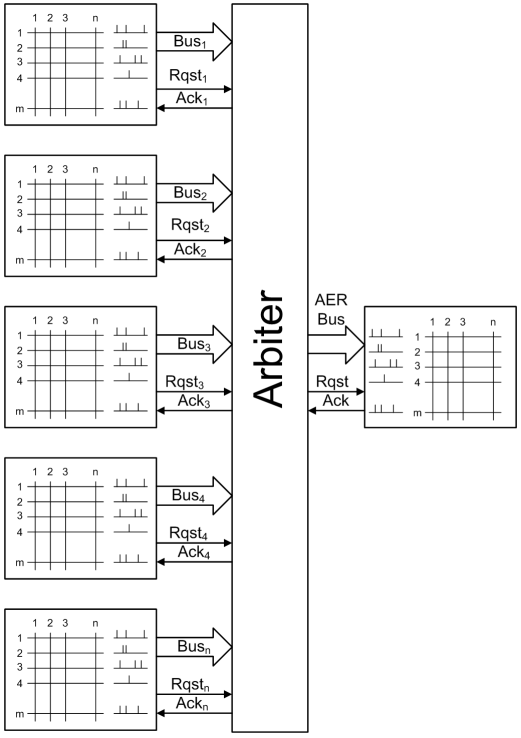


FIGURA 2.9. Arquitectura de un sistema AER multi-emisor.

añadir una operación lógica entre todas las líneas de *Acknowledge* antes de llegar al emisor, permitiendo con ello compartir el bus AER, como se muestra en la Fig. 2.8. Dicha operación lógica está basada en elementos-C, los cuales aseguran que no se producirá ningún cambio de estado a la salida mientras las entradas tengan distintos valores. Así, solamente en el caso de que todas las señales de entrada se pongan de acuerdo la salida cambiará.

En el caso contrario, si queremos que varios emisores compartan el bus de salida, la configuración no es tan inmediata. Si directamente permitiéramos a varios emisores escribir sobre el bus digital, sería imposible gestionar los conflictos, así que es necesario añadir algún tipo de arbitración. En [44] se proponen dos posibles alternativas. La primera de ellas, representada en la Fig. 2.9, consiste en añadir un circuito de arbitración externo a los emisores AER, el cual se encargaría de limitar el acceso de cada uno de dichos

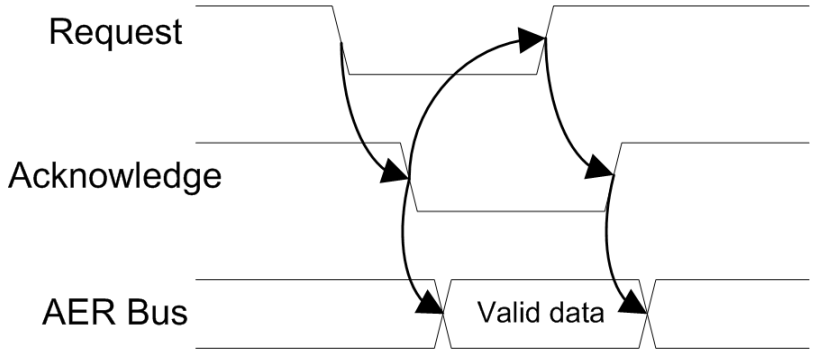


FIGURA 2.10. Diagrama temporal del protocolo AER modificado para sistema multi-emisor.

emisores al bus común. De este modo, cuando un chip quiere trasmitir un evento activa su señal de *Request* $Rqst_i$ $(i = 1, \ldots n)$, y cuando el arbitrador esté libre entonces copiará en el bus común el dato en cuestión, activando la señal *Rqst* que llega al receptor. Una vez que el receptor responde, se libera el arbitrador y se puede transmitir otro evento generado por cualquiera de los emisores.

La otra propuesta para construir sistemas AER multi-emisor consiste en modificar ligeramente el protocolo de los chips emisores para que se comuniquen entre ellos conectándose en forma de árbol. De este modo, sería posible que los buses AER de todos ellos se conectaran entre sí directamente, ya que cada emisor solamente escribiría en dicho bus cuando recibe permiso del sistema global. Es básicamente la misma idea que la propuesta anterior, con la diferencia de que la arbitración se lleva a cabo en los propios emisores. La ventaja es que nos permite compartir el bus directamente, pero con el inconveniente de modificar ligeramente el protocolo y añadir una cierta complicación interna a los emisores.

Con estas ideas básicas, se pueden plantear pequeñas variaciones dando lugar a diferentes alternativas. Una de ellas consiste en eliminar cualquier componente externo a los propios chips AER modificando el protocolo como muestra la Fig. 2.10. De este modo, cuando uno de los emisores activa la señal de *Rqst*, aún no ha puesto el dato en el bus común, sino que espera a que el receptor active la señal *Ack*, indicándole que tiene permiso para ponerlo. Así, en este esquema multi-emisor el *Ack* vuelve sólo a uno de

los emisores, y por tanto el receptor tiene que generar tantas señales de *Ack* como emisores haya. En [45], [46] se propone SCX (*Silicon Cortex*), una infraestructura configurable de comunicación AER que se puede usar para probar la comunicación entre chips en sistemas neuromórficos con diferentes conectividades.

Otra alternativa es añadir a cada uno de los chips AER una etapa de arbitración interna para conectarlos en cascada. De esta forma, cuando el primero de los chips quiere emitir un evento, la petición tiene que pasar por todos los demás chips de la cadena, y sólo cuando todos ellos le den paso se enviará. El problema de este esquema es que no trata de la misma forma a todos los emisores, ya que influye el orden en el que se conecten en la cascada. No obstante, esta asimetría se podría compensar incluyendo algún mecanismo que penalice a cada chip en función de la posición que ocupe en la cadena, haciendo que en caso de colisión tengan que esperar más los que tengan una posición "preferente".

En la presente tesis se ha optado por utilizar placas auxiliares para implementar sistemas multi-emisor o multi-receptor. Estas placas, en definitiva, siguen los esquemas descritos en Fig. 2.8 y Fig. 2.9, y desde el punto de vista de nuestros chips hacen que cada enlace se gestione siempre punto a punto [47], [48].

CAPÍTULO 3

Sistemas de convolución multicapa

3.1. Introducción

Los sistemas biológicos de visión no solamente están basados en el procesamiento por eventos como se describe en el capítulo anterior, sino que quizás una de sus principales características que lo hacen posible es la estructura multicapa del cerebro.

La información óptica (información sensorial, en general) capturada por la retina y transformada en pulsos eléctricos es procesada por una serie de capas que forman el córtex visual. Cada una de estas capas está formada por grandes cantidades de neuronas interconectadas de forma masiva. De forma general, una neurona de una capa está conectada con muchas neuronas de la siguiente capa, de modo que los eventos emitidos por ella son recibidos por lo que llamamos un campo proyectivo de la capa que se encuentra a continuación.

Así, la información capturada por una zona concreta de la retina se propaga con gran velocidad por cada una de las capas del córtex, que se encargan de detectar la forma de los objetos observados por la retina. Cada interconexión entre una neurona y su campo proyectivo correspondiente

cuenta con unos pesos específicos, en principio diferentes para cada neurona. No obstante, en las primeras capas del córtex los pesos son bastante aproximados para las distintas neuronas de una misma capa, de forma que esta interconexión entre capas se puede describir como una convolución. Así pues, el sistema completo se puede aproximar como un sistema de convoluciones multicapa. Por eso, el objetivo de esta tesis es el desarrollo de bloques convolucionadores que se puedan conectar formando un sistema multicapa, y emular el comportamiento de los sistemas de procesamiento de visión.

Las *Convolutional Neural Networks* (Redes Neuronales de Convoluciones) utilizan esta idea de capas de neuronas interconectadas con pesos compartidos para llevar a cabo diversas aplicaciones [49], como reconocimiento de caracteres [50], [51] (sistemas de visión), o de fonemas [52], [53] o palabras habladas [54] (en sistemas acústicos). Todos estos sistemas se basan en capas de convoluciones interconectadas entre sí con pesos configurables.

A continuación, en la Sección 3.2 se describen los conceptos básicos de la operación de convolución, así como la forma de implementarla mediante un sistema AER. Después, en la Sección 3.3 se profundiza sobre los sistemas bioinspirados de procesamiento multicapa de visión, destacando una herramienta software de simulación de estas estructuras. Por último, en la Sección 3.4 se describe en un primer nivel la arquitectura de chip de convolución propuesta, así como la capacidad de interconexión en sistemas multicapa que presenta.

3.2. Convolución 2-D

3.2.1. Operación matemática

Para un sistema lineal e invariante en el tiempo como el de la Fig. 3.1 (LTI, en sus siglas inglesas) con respuesta al impulso $h(t)$, siendo la señal de entrada $x(t)$ dependiente del tiempo, se puede calcular la señal de salida $y(t)$ como:

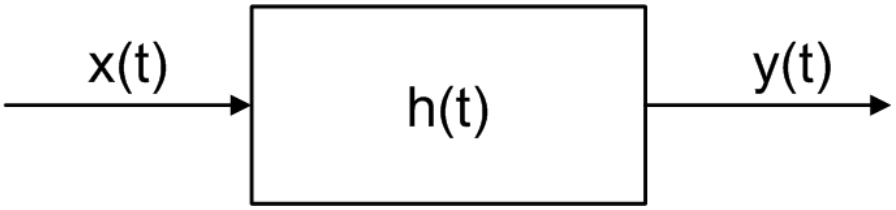


FIGURA 3.1. Representación de un sistema lineal e invariante en el tiempo (LTI).

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

Dicha operación se denomina convolución entre $x(t)$ y $h(t)$, y se expresa mediante la notación $y(t) = x(t) \otimes h(t)$.

Si en lugar de tratar con señales continuas en el tiempo, nuestro sistema funciona en tiempo discreto, la integral se convierte en un sumatorio y la expresión de la convolución pasa a ser [55]:

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

Considerando que no sólo estemos trabajando con señales discretas en el tiempo, sino también finitas, el resultado de la convolución será también una señal finita, expresada mediante la ecuación (3.3):

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

De este modo, si tenemos una señal de entrada $x(n)$ con N_x valores (es decir, que es distinta de 0 para $0 \le n \le N_x - 1$), y una respuesta al impulso $h(n)$ con N_h valores (es distinta de 0 para $0 \le n \le N_h - 1$), la con-

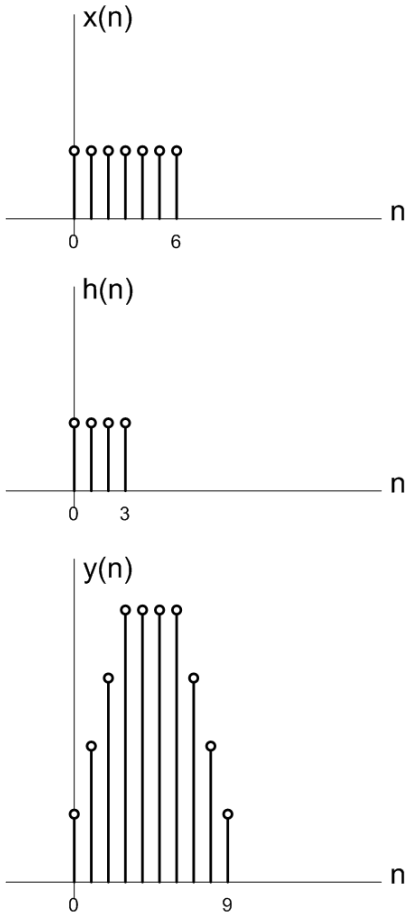


FIGURA 3.2. Ejemplo de convolución unidimensional entre una secuencia $x(n)$ con $N_x = 7$ y una respuesta al impulso $h(n)$ con $N_h = 4$, obteniéndose una señal $y(n)$ con $N_y = N_x + N_h - 1 = 10$

volución $y(n)$ tendrá $N_y = N_x + N_h - 1$ valores no nulos. Esto se puede observar en el ejemplo de la Fig. 3.2.

Una vez descrita la operación de la convolución unidimensional, es sencillo extenderla para el caso de dos dimensiones, simplemente considerando que las señales dependen de dos variables (x,y) [56]. El caso de con-

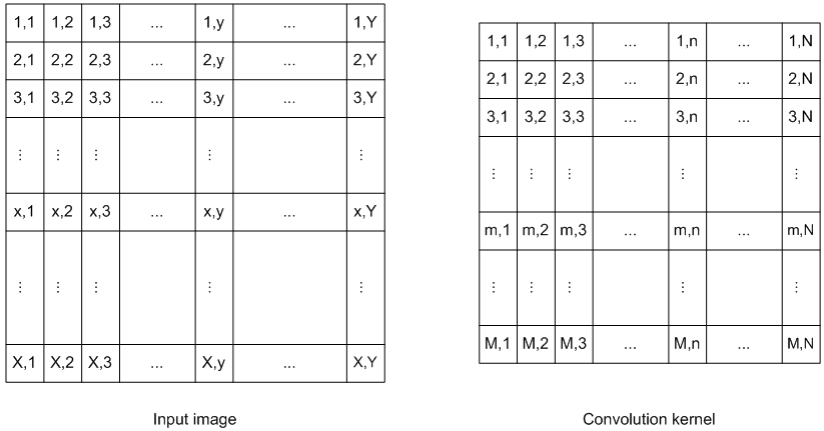


FIGURA 3.3. Descripción de una convolución bidimensional entre una imagen de entrada de tamaño (X,Y) y un kernel de tamaño (M,N)

voluciones 2-D es el que nos interesa, ya que se aplica a las imágenes. De este modo, consideramos que las variables x e y se corresponden con las coordenadas horizontal y vertical, y aplicamos la operación de convolución a una imagen de entrada $I(x,y)$ para obtener una imagen de salida $O(x,y)$. A la respuesta al impulso la denominamos kernel de la convolución $K(x,y)$, y obtenemos la expresión:

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

Esta expresión muestra una inversión horizontal en las posiciones del kernel frente a la imagen de entrada al calcular la convolución. Sin embargo, a la hora de implementar este cálculo habitualmente prescindimos de dicha inversión, ya que se puede eliminar definiendo el kernel adecuadamente. Además, en gran parte de los casos, las simetrías que presentan los kernels hacen que la expresión (3.4) sea equivalente.

El significado de esta expresión se puede comprender mejor con ayuda de la Fig. 3.3. En ella vemos una imagen de entrada de $X \times Y$ píxeles, y un kernel de tamaño $M \times N$. La imagen de salida, resultado de calcular la convolución entre ambos, se obtiene aplicando el kernel centrado sobre cada

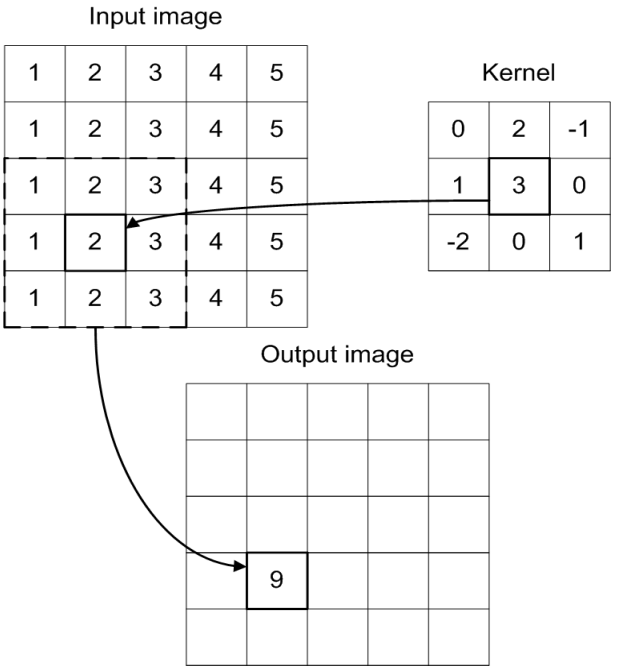


FIGURA 3.4. Ejemplo de cómo se calcula el resultado de una convolución para un píxel. El resto de los píxeles se calcularían del mismo modo.

uno de los píxeles de la imagen de entrada, y efectuando la suma del valor de todos los píxeles del vecindario pesados por los valores correspondientes del kernel. En el ejemplo de la Fig. 3.4 tenemos una imagen de entrada de 5×5 píxeles y un kernel de tamaño 3×3. En la imagen de salida se muestra el resultado de la convolución para un solo píxel, concretamente el $(4, 2)$. Vemos cómo el kernel se aplica sobre el vecindario de dicho píxel, y se calcula la operación:

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

De este modo, el valor de cada píxel de salida se calcula desplazando el kernel sobre la imagen de entrada y efectuando la misma operación.

3.2.2. Convolución basada en AER

Una vez descrita la operación matemática de la convolución bidimensional, ahora se trata de implementar dicha operación mediante un sistema basado en AER. Para ello, en primer lugar es necesario volver a la idea de procesamiento de imágenes basado en eventos del capítulo anterior.

Si estuviéramos tratando con sistemas tradicionales de procesamiento por fotogramas, se podría tratar cada imagen como una matriz con sus correspondientes valores numéricos y se podría efectuar la operación matemática tal cual ha sido definida. Sin embargo, en los sistemas de procesamiento por eventos no contamos con una imagen completa, sino que se van produciendo eventos que de algún modo codifican el estado de los píxeles de la imagen en tiempo real. De este modo, la convolución se lleva a cabo de la forma indicada en la Fig. 3.5. En ella vemos cómo cada vez que se recibe un evento con una dirección cualquiera (x, y), la operación no se limita al píxel con dicha dirección, sino que le afecta a un cierto vecindario alrededor de dicho píxel. Así, el evento le llega a cada píxel del vecindario con un cierto peso en función de su posición a través de la aplicación del kernel. Cada píxel del array almacena el valor, y conforme van llegando más eventos el kernel se sigue sumando sobre el vecindario correspondiente a la dirección de dichos eventos. De esta manera se consigue que en cada momento el array de píxeles contenga el resultado de calcular la convolución sobre la imagen de entrada.

Sin embargo, para que se trate verdaderamente de una convolución basada en AER el resultado de la operación debe estar también codificado en eventos. Por ello, cada píxel de convolución consistirá en una neurona *Integrate&Fire*, que cuando alcance un valor umbral establecido generará un evento de salida, obteniéndose un flujo de eventos que se corresponde con el resultado de la operación de convolución. Esto nos permitirá encadenar convolucionadores AER en cascada, haciendo que la salida de uno sea la entrada del siguiente, o bien en paralelo, creando sistemas complejos multicapa.

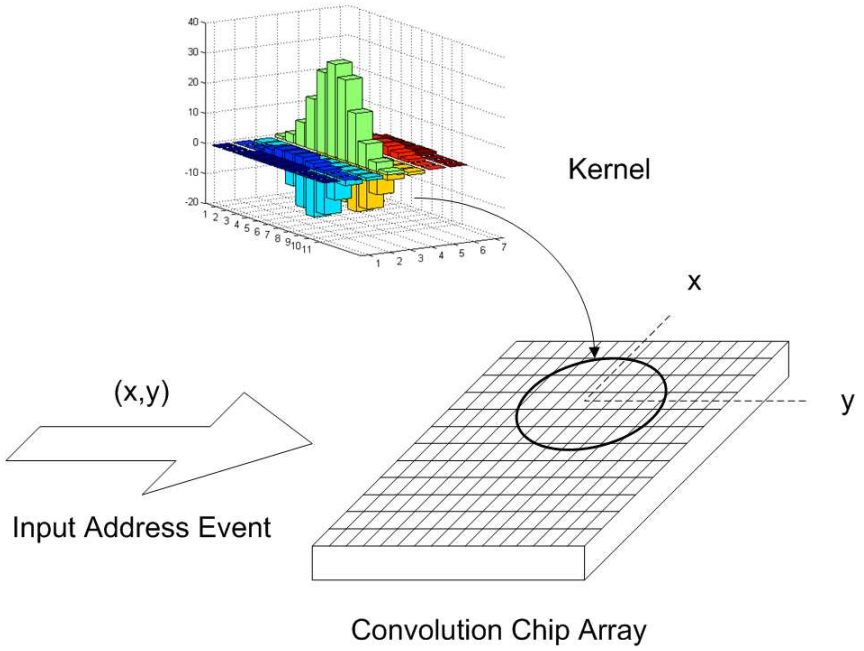


FIGURA 3.5. Implementación de una convolución bidimensional mediante un sistema AER de procesamiento por eventos.

Esta idea de las convoluciones en cascada y paralelo nos lleva al tema de los sistemas de procesamiento multicapa.

3.3. Sistemas multicapa bioinspirados

3.3.1. Inspiración biológica

Desde un punto de vista biológico, la visión se lleva a cabo en una zona del cerebro llamado córtex visual. Este córtex visual está estructurado en una serie de capas (entre 8 y 10 en el caso humano [57], [5]), partiendo desde la retina, la cual ya hace su propio preprocesamiento de la información que captura, y está especializado para procesar información de objetos

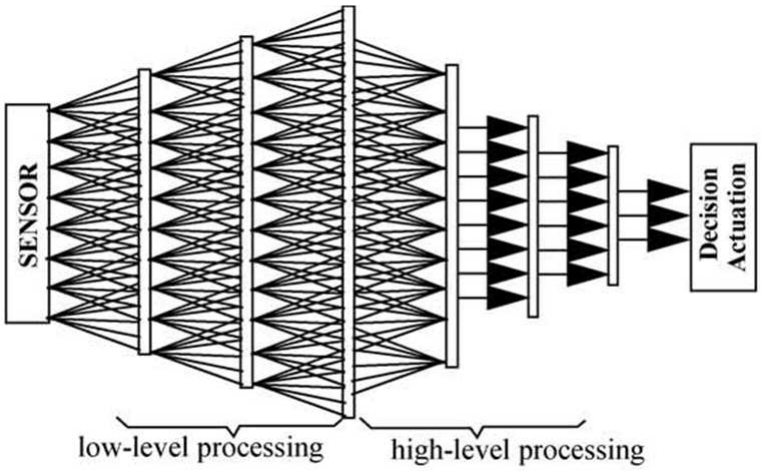


FIGURA 3.6. Estructura genérica de un sistema multicapa bioinspirado.

tanto estáticos como en movimiento, y para reconocimiento de patrones. A partir de la retina, existe una masiva interconexión entre las neuronas de las distintas capas del córtex visual, con una cantidad estimada de 10^{11} neuronas en el cerebro humano [58]. Sin embargo, el patrón de conectividad que sigue el córtex está estructurado de una forma muy concreta, ya que cada neurona perteneciente a una capa se conecta a un conjunto de neuronas perteneciente a la siguiente capa. A ese conjunto de neuronas de la siguiente capa se le conoce como campo proyectivo [59].

El concepto de campo proyectivo implica que cuando una neurona produce un pulso, este pulso es recibido por todo un conjunto de neuronas de la siguiente capa, lo cual nos permite establecer una relación entre este concepto y la convolución bidimensional AER. En el cerebro, la conexión de una neurona con un campo proyectivo en la capa siguiente se realiza de forma que cada una de las conexiones individuales tiene un peso determinado, consiguiendo así que un pulso no contribuya de la misma forma en todas las neuronas. Este mismo efecto es el que se consigue en una convolución 2-D a través de los valores del kernel.

En definitiva, la estructura del córtex visual se puede expresar conceptualmente según la Fig. 3.6. El sensor que aparece a la izquierda se corres-

pondería con la retina, y vemos cómo se representa la conexión de cada neurona de una capa con un campo proyectivo en la siguiente capa. Si consideramos que cada capa está realizando una convolución (o muchas en paralelo), podemos concluir que con una red artificial multicapa de convolucionadores es posible llevar a cabo tareas complejas de reconocimiento de objetos que emulen el funcionamiento del córtex cerebral.

Un ejemplo de modelo de visión artificial propuesto basado en la estructura del córtex cerebral es el algoritmo BCS-FCS (Boundary Contour System - Feature Contour System) para segmentación de imágenes [60]-[64]. Este modelo consiste en 9 capas que, después de una normalización de la iluminación local y una mejora del contraste de la imagen de entrada, llevan a cabo una extracción local de bordes para diferentes orientaciones y escalas. De este modo, se consigue identificar los contornos de los objetos en la imagen de entrada.

El esquema de este algoritmo podemos verlo en la Fig. 3.7. El sistema BCS consiste en varios subsistemas idénticos, cada uno de los cuales estaría ajustado para una escala espacial diferente. En el ejemplo de esta figura tendríamos 3 subsistemas que se muestran superpuestos. Cada uno de estos subsistemas consta de 8 capas, y las conexiones entre capas consecutivas se representan con anchas flechas oscuras. Estas conexiones indican una operación de convolución (o filtrado) que se aplica al estado de la capa anterior para producir el estado de la capa siguiente. Por ejemplo, la imagen bidimensional de entrada sufre 3 filtrados diferentes, cada uno de los cuales es el punto de partida de un subsistema BCS, el cual a partir de ahí funciona de modo autónomo frente a los otros dos.

Las capas 1, 2 y 3 solamente incluyen operaciones "*feed forward*" (sin realimentación), mientras que las capas entre la 4 y la 8 están conectadas en una configuración con un bucle de realimentación, lo cual implica que el sistema alcanzará un estado estacionario después de un cierto número de iteraciones (si el sistema está implementado de forma secuencial en un ordenador) o tras una cierta constante temporal (si el sistema funciona de forma asíncrona y en paralelo, como en los cerebros biológicos). Por otra parte, las salidas de las capas 1 y 5 de los 3 subsistemas BCS son las entradas del sistema FCS. Veamos brevemente la funcionalidad de cada una de las capas.

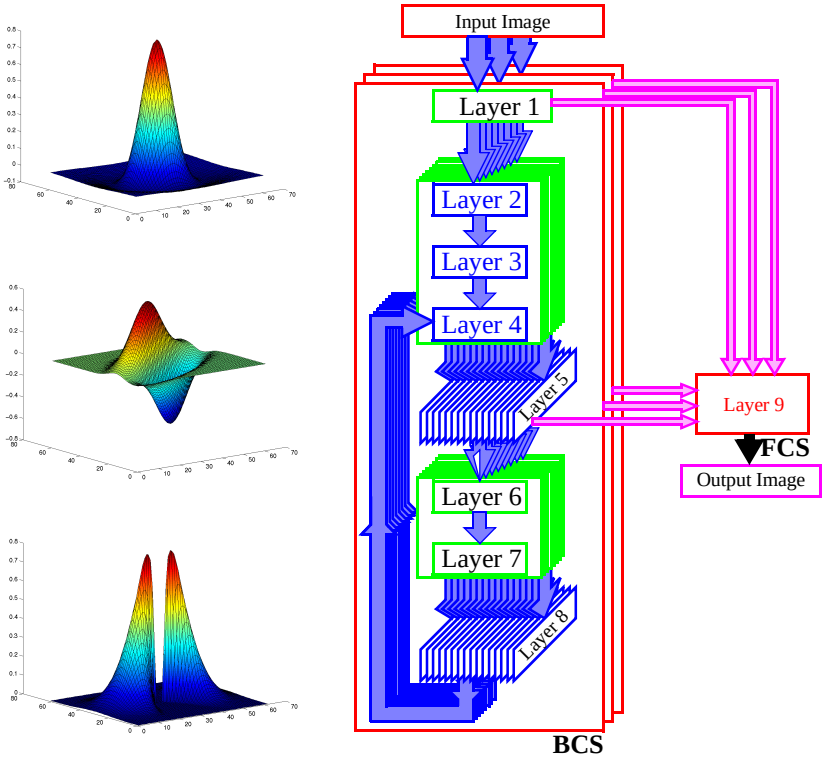


FIGURA 3.7. Diagrama de bloques del sistema BCS-FCS.

La capa 1 aplica sobre la imagen de entrada un kernel de tipo "*on-center off-surround*" como el que aparece en la parte superior de la Fig. 3.7, con pesos positivos para los píxeles próximos a la región central y pesos negativos para los demás. El resultado de esta convolución es una normalización de la iluminación local y una mejora del contraste. Este filtrado se aplica en las 3 capas 1 en paralelo sobre la misma imagen de entrada, con diferentes anchuras de la zona positiva del kernel.

La capa 2 aplica una serie de convoluciones en paralelo, tantas como orientaciones diferentes queramos detectar. Para ello, utiliza kernels para detección de bordes como el que aparece en la imagen central de la Fig. 3.7, cada uno de ellos rotado según el ángulo correspondiente. De este modo,

cuando la imagen de entrada de esta capa presente un cambio de contraste positivo en la dirección indicada producirá un valor positivo de salida, mientras que un cambio de contraste negativo producirá un valor negativo. Así pues, la capa 3 se encarga rectificar estos valores, quedándose solamente con la información referente a la presencia de bordes en cada zona de la imagen para cada orientación.

La capa 4 vuelve a aplicar un filtrado como el de la capa 1, con la finalidad de mejorar el contraste de la imagen en este punto. Una vez obtenidas todas las imágenes de todas las orientaciones que salen de la capa 4, todas ellas son filtradas en la capa 5 con kernels del tipo de la parte superior de la Fig. 3.7 para mejorar el contraste de las orientaciones que tienen un mayor valor.

En la capa 6 se aplican una serie de filtros con kernels bipolares como el de la parte inferior de la Fig. 3.7 con el objetivo de identificar contornos, es decir, bordes que se mantienen a lo largo de rangos espaciales mayores. Después de esto, la capa 7 y 8 repiten las mismas operaciones de las capas 4 y 5, respectivamente. La salida de la capa 8 sirve a su vez de realimentación combinándola en la capa 4 con la salida de la capa 3.

Por último, la capa 9 recibe la información de contornos generada por la capa 5 (una vez que el bucle de realimentación ha estabilizado los resultados) para todas las orientaciones computadas, y junto con la imagen inicial producida por la capa 1, realiza una operación de difusión entre píxeles para conseguir una imagen limpia de ruido y con los contornos claramente marcados.

Este tipo de algoritmos multicapa para procesamiento de imágenes tradicionalmente ha estado limitado por el concepto de fotograma, que dificultaba su implementación en tiempo real. Cada una de las capas de la Fig. 3.7 implicaba una serie de operaciones de convolución en paralelo, por ejemplo con un mismo kernel a distintas escalas y rotado para diferentes ángulos, produciendo una importante carga computacional. Por este motivo, para conseguir un procesamiento en tiempo real de este modelo es fundamental su adaptación a la visión por eventos AER [65].

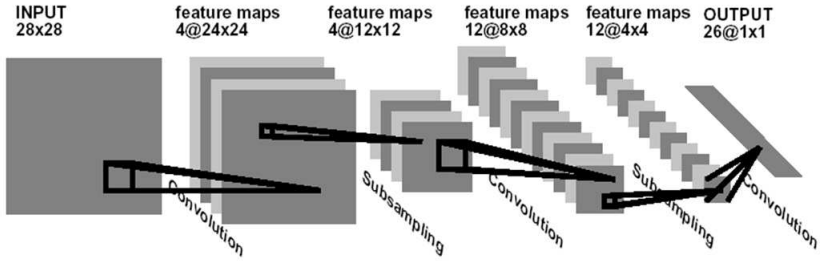


FIGURA 3.8. Ejemplo de Convolutional Neural Network propuesto por Y. LeCun.

Como se vio en el capítulo anterior, en los sistemas de procesamiento por eventos los píxeles más relevantes envían pulsos que se propagan por todas las capas en un tiempo muy pequeño (del orden de los microsegundos), permitiendo que se lleven a cabo detecciones con mucha rapidez en sistemas complejos multicapa [66], [8].

No obstante, la mayor parte de los sistemas de convolución multicapa desarrollados han estado basados en el procesamiento por fotogramas, dando lugar a las *"Convolutional Neural Networks"* (Redes Neuronales de Convoluciones). Este tipo de redes han permitido desarrollar aplicaciones para reconocimiento de caracteres [67]-[69], [71], detección de objetos [69], o reconocimiento de caras [69]-[71]. En la Fig. 3.8 se muestra como ejemplo la arquitectura multicapa propuesta por LeCun [71], donde aparece un gran número de convoluciones.

En este ejemplo se puede explicar una de las grandes funcionalidades presentes en uno de los chips de convolución diseñados en esta tesis, concretamente el llamado Conv2. Vemos en la Fig. 3.8 cómo cada uno de los *feature maps* (mapas de características) de una determinada capa aplica un kernel diferente para cada uno de los campos proyectivos de la capa anterior. Esto implica en nuestros sistemas que en función del lugar de procedencia de una determinada información hay que procesarla con un kernel diferente. Es lo que llamamos multikernel, y la forma de implementarlo viene detallada en los siguientes capítulos.

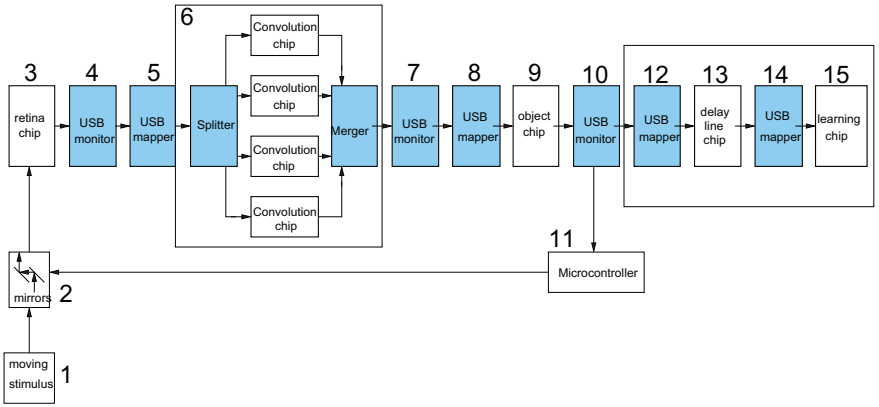


FIGURA 3.9. Diagrama de bloques del montaje experimental desarrollado por el proyecto europeo CAVIAR para un sistema de visión AER de seguimiento de objetos.

Con el objetivo de trasladar toda esta experiencia en sistemas de procesamiento multicapa basados en fotogramas al más eficiente procesamiento por eventos, se llevó a cabo el proyecto europeo CAVIAR, donde se desarrolló una arquitectura AER compuesta por una serie de bloques (entre ellos, convolucionadores [32]) que permite con una gran versatilidad la implementación de aplicaciones para procesamiento de imagen y reconocimiento y seguimiento de objetos [72], como la mostrada en la Fig. 3.9 que describimos a continuación.

Un sistema mecánico rotatorio (1 en la Fig. 3.9) hace girar un trozo de papel blanco con dos círculos de diferente radio y algunas otras formas geométricas usadas como distracción. El sistema de visión se encarga de seguir a ambos círculos y discriminar entre ellos. Un par de espejos controlados por servomotores (2) cambia el punto de vista de la retina AER (3), la cual envía sus eventos de salida a una placa de monitorización (4) y a un *mapper* (5) antes de alcanzar la placa de convolucionadores (6) con 4 chips en su interior. Las salidas de estos chips de convolución se envían, a través de otra placa de monitorización (7) y otra de mapeo (8), al chip de objeto WTA (*Winner Takes All*) (9), que a su vez envía su salida a un monitor (10) que por una parte la envía a un microcontrolador (11) que se encarga de

ajustar los espejos (2) para centrar el círculo detectado, y por otra parte la envía al sistema de aprendizaje, que consiste en un *mapper* (12), un chip de línea de retraso (13), otro *mapper* (14) y un chip clasificador de aprendizaje (15), que se encarga de clasificar las trayectorias del círculo entre distintos tipos.

La retina de contraste temporal genera eventos AER con un espacio de direcciones de 128×128 píxeles, mientras que cada chip de convolución tiene una resolución de 32×32. Sin embargo, la placa de 4 chips de convoluciones conectados en modo mosaico sería capaz de procesar el espacio de direcciones completo, aunque sólo puede generar salidas para los 64×64 píxeles centrales. Para resolver este problema se añade la etapa de mapeo (5) que se encarga de submuestrear la imagen desde los 128×128 píxeles hasta 64×64. De este modo, la placa de convolución generar salidas para todo el espacio visual de la retina. Los chips de convolución utilizan kernels circulares de un determinado diámetro, los cuales detectan la posición del centro de formas circulares de entrada de dicho diámetro.

El mapper (8) vuelve a submuestrear el espacio de direcciones hasta los 32×32 píxeles de entrada del chip de objeto WTA, el cual se encarga de proporcionar las coordenadas del centro del círculo detectado de forma limpia. Estas coordenadas, al ser enviadas al subsistema de control, hacen que el microcontrolador actúe sobre dos servomotores que sostienen dos espejos. Un espejo está en posición vertical y es controlado por la coordenada-y de la salida del chip de objeto, mientras que el otro está en posición horizontal y es controlado por la coordenada-x. Así, la coordenada proporcionada por el chip de objeto indica la desviación de la posición del círculo detectado sobre el centro del espacio visual, de modo que el microcontrolador está programado para hacer cero dicha desviación, manteniendo centrado el círculo correspondiente.

Por otra parte, el chip de línea de retraso convierte la información temporal de los eventos generados por el chip de objeto WTA en información espacial, obteniendo un cierto patrón espacial. Este patrón espacial es clasificado por el chip de aprendizaje según patrones de eventos, formados por eventos coincidentes en distintas localizaciones espaciales, o según patrones de actividad, formados por actividades medias de eventos coincidentes en diferentes localizaciones.

3.3.2. Implementación software basada en AER: aplicaciones

Siendo el objetivo de esta tesis el desarrollo de chips de convolución que nos permitan la implementación de sistemas modulares a gran escala, es fundamental contar también con la capacidad de estudiar desde un punto de vista teórico la forma de ensamblar, configurar, programar y entrenar estos sistemas. Es decir, a partir de los chips de convolución AER desarrollados en el presente trabajo, para llevar a cabo una aplicación concreta será necesario en primer lugar encontrar la estructura jerárquica óptima, así como los kernels de convolución más indicados, o qué otros parámetros deberían ser ajustados. Para ello, es de gran ayuda el simulador de comportamiento AER desarrollado por J. A. Pérez-Carrasco en Visual C++ [73], el cual nos permite hacer una descripción de comportamiento de cualquier módulo AER real (en nuestro caso, los convolucionadores), y ensamblar sistemas complejos con gran cantidad de módulos. Así podemos obtener una estimación realista del resultado de este tipo de sistemas multicapa antes de llevar a cabo la implementación hardware.

Con este simulador de comportamiento se han podido comprobar las enormes posibilidades que tienen este tipo de arquitecturas para emular el comportamiento del cerebro. Por ejemplo, en [74] se presenta la simulación de comportamiento de un sistema de reconocimiento de caracteres, que conseguiría discriminar entre distintas letras en tiempos inferiores a $10\mu s$, a pesar de incluir en dicho procesamiento hasta 52 convoluciones. Esto nos da una idea de la rapidez de estos sistemas multicapa AER. En la Fig. 3.10 se puede ver el esquema multicapa implementado por el simulador. Como se indica en la propia figura, el sistema está diseñado para discriminar entre 7 caracteres diferentes escritos a mano, produciendo eventos solamente por una de las 7 salidas de la cuarta y última capa, en función de la letra que se corresponda con los eventos de entrada del sistema.

También se han obtenido resultados satisfactorios utilizando este simulador para entrenar sistemas de reconocimiento de texturas, como se muestra en [75], [76]. En la Fig. 3.11 se puede ver la arquitectura propuesta en dicho ejemplo, donde la primera capa incluye 24 módulos de convolución

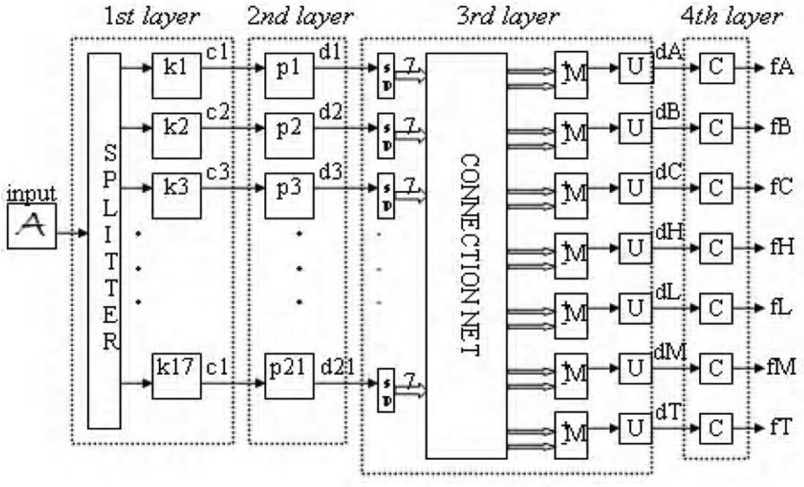


FIGURA 3.10. Esquema del sistema AER implementado con el simulador para reconocimiento de letras.

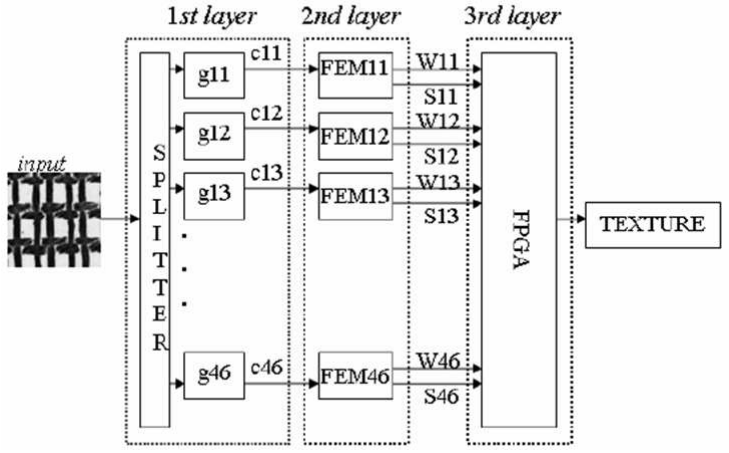


FIGURA 3.11. Esquema del sistema AER implementado con el simulador para reconocimiento de texturas.

en paralelo, implementando un banco de filtros de Gabor con 4 escalas y 6 orientaciones diferentes.

3.4. El chip de convolución AER

Una vez descrita la motivación de este trabajo, así como el marco donde se encuadra, es el momento de pasar a describir el objetivo básico de esta tesis: el diseño del chip de convolución AER.

Dentro del objetivo de diseñar una arquitectura multichip compleja, configurable y versátil que nos permita implementar sistemas neuronales, el elemento fundamental para poder construir este tipo de sistemas es el chip de convolución. En [77] se propone una arquitectura para la realización de convoluciones bidimensionales en tiempo real basada en sistemas de visión AER, con algunas restricciones sobre los kernels permitidos, debiendo ser éstos descomponibles en sus coordenadas (x, y) (un kernel de la forma $F(x, y) = H(x)V(y)$).

En [78] se describe un chip de convolución basado en integradores analógicos. Las principales limitaciones que presenta este chip son:

1. la necesidad de calibración para compensar el *mismatch* entre transistores,
2. una reducida resolución de sólo 3 bits (incluso después de la calibración), debido a la operación en baja corriente de los transistores,
3. un tiempo de latencia de eventos alto (de alrededor de $1ms$) debido a los retrasos de los componentes analógicos en los píxeles polarizados para bajo consumo.

Estas limitaciones se han resuelto en los chips desarrollados en la presente tesis. Al usar píxeles completamente digitales, desaparece la necesidad de calibración (con su correspondiente coste en términos de área y de consumo) y la precisión viene dada por el tamaño de los registros implementados en dichos píxeles. Además, al prescindir de componentes analógicos de bajo consumo la operación es mucho más rápida, alcanzando latencias de eventos tan bajas como $150ns$ [79].

En esta tesis se proponen dos chips de convolución diferentes, aunque ambos están basados en píxeles completamente digitales, y permiten el uso

de kernels de tamaño y forma arbitrarios. El tamaño está limitado por la memoria del kernel, que en nuestro caso has sido de 32×32.

3.4.1. Arquitectura

Desde el punto de vista de la descripción de diagrama de bloques, los dos chips de convolución diseñados para la presente tesis (que llamaremos Conv1 y Conv2) comparten la arquitectura de la Fig. 3.12. En ambos casos, el chip recibe eventos AER de entrada (el bus *Address_in*, más las señales del protocolo asíncrono *Rqst_in* y *Ack_in*), que representan información visual procedente de la etapa anterior, y genera eventos AER de salida (el bus *Address_out*, más las señales del protocolo asíncrono *Rqst_out* y *Ack_out*) que representan el resultado de la operación de convolución. Los bloques que se muestran en la Fig. 3.12, que están descritos en detalle en los siguientes capítulos, son los siguientes:

1. Array de píxeles, de tamaño 32×32 en el caso de Conv1, siendo ampliado hasta 64×64 en Conv2, cuadruplicando su resolución espacial.
2. Memoria RAM estática integrada en el propio chip, donde se almacena el kernel codificado en complemento a 2. En ambos chips el tamaño de la RAM es de 32×32 datos, aunque Conv1 sólo permite programar un único kernel mientras que Conv2 permite programar en un propio chip hasta 32 kernels diferentes, implementando así el sistema multikernel, descrito más adelante.
3. Controlador síncrono, que se encarga de secuenciar todas las operaciones necesarias para cada evento de entrada, así como del mecanismo de olvido, que es independiente de la llegada de eventos.
4. Generador de reloj de alta velocidad, de frecuencia controlable, que se utiliza para el controlador síncrono.
5. Registros de configuración, que almacenan una serie de parámetros cargados a través de un puerto serie.
6. Inversor de complemento a 2, es un bloque que cambia el signo del kernel antes de aplicarlo sobre los píxeles si el evento de entrada es negativo.

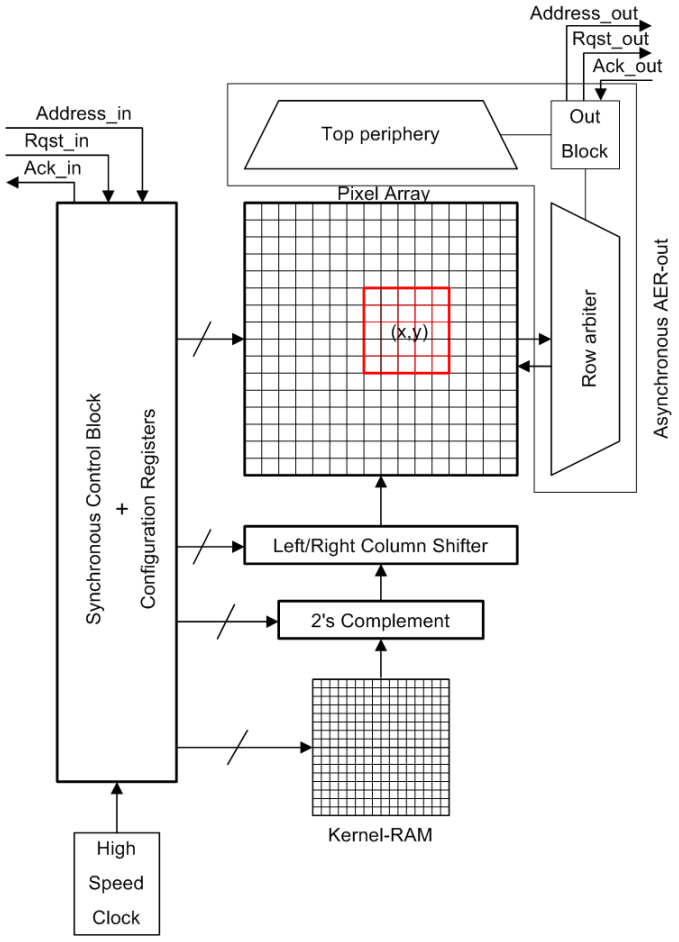


FIGURA 3.12. Arquitectura común de los chips Conv1 y Conv2

7. Bloque de desplazamiento horizontal, para alinear adecuadamente el kernel almacenado en la RAM con las coordenadas del evento de entrada.
8. Generador AER, bloque asíncrono encargado de arbitrar entre los eventos generados por los píxeles y enviarlos a la siguiente etapa del sistema multichip.

El funcionamiento del chip es el siguiente: cuando el controlador síncrono detecta un flanco de bajada en la señal de entrada *Rqst_in*, las coordenadas del evento (x, y) que aparecen en el bus *Address_in* se capturan y se completa el *handshaking* asíncrono. Entonces, el controlador utiliza la información relativa al tamaño del kernel (que ha sido almacenada previamente en los registros de configuración) para calcular los límites del campo proyectivo asociados a la dirección del evento. Este cálculo puede implicar tres situaciones distintas:

1. que el campo proyectivo caiga completamente dentro del array de píxeles,
2. que caiga parcialmente dentro del array,
3. o que esté completamente fuera del array.

Si estamos en el caso 3, el controlador directamente descarta el evento y se queda esperando que llegue el siguiente. Sin embargo, en cualquiera de las otras dos posibles situaciones, el controlador calcula el desplazamiento horizontal izquierda/derecha entre las columnas de la RAM donde se almacena el kernel y las columnas del campo proyectivo en el array de píxeles. A continuación, habilita la suma fila por fila de los valores del kernel sobre los píxeles correspondientes. De este modo, después de recibir un evento de entrada, los píxeles que se encuentran dentro del campo proyectivo actualizan su estado. En el caso de que alguno de ellos alcance el umbral programado, éste resetea su propio estado y genera un evento que, tras ser arbitrado por el bloque asíncrono generador AER, es enviado al exterior con sus correspondientes señales de *handshaking*. Paralelamente a este procesamiento por evento, hay un mecanismo de olvido global que es común para todos los píxeles.

El bloque asíncrono generador de AER sigue la técnica de lectura de eventos en paralelo por filas [41]. De este modo, los eventos se arbitran por filas (para una misma fila, todas las señales de petición de evento implementan una OR cableada). Una vez que el arbitrador por filas responde, todos los eventos generados por esta fila se almacenan en la periferia superior, liberando el arbitrador por filas. Así, éste puede atender la petición de otra fila mientras todos los eventos de la fila anterior son emitidos al exterior en modo ráfaga.

En general, ya que los kernels de convolución pueden tener tanto valores positivos como negativos, los eventos de salida generados por un chip de convolución también tendrán signo. En un sistema multicapa, las operaciones de convolución se pueden ejecutar en cascada, lo que implica que un chip de convolución genérico debe ser capaz de manejar eventos de entrada con signo, y de producir eventos de salida con signo. Por este motivo, los chips de convolución desarrollados en el presente trabajo incluyen un bit de signo tanto en el bus AER de entrada como en el de salida, así como para los valores almacenados en la RAM (codificados en complemento a 2). Los píxeles deben ser capaces también de ejecutar sumas con signo y producir eventos positivos o negativos. Cuando está procesando un evento negativo, el controlador habilita el bloque inversor de complemento a 2 para cambiar el signo de los valores del kernel antes de ser sumado sobre los píxeles.

En cuanto al mecanismo de olvido, éste también es manejado por el controlador síncrono. El objetivo de dicho mecanismo es que los valores absolutos almacenados en los píxeles se decrementen a un ritmo programable, para que éstos puedan "olvidar" su estado tras un tiempo controlado.

3.4.2. Estructura multichip propuesta

Como se ha venido comentando a lo largo de todo el capítulo, el objetivo de este trabajo no se limita al diseño de las dos versiones de chip de convolución Conv1 y Conv2, sino que ambos circuitos están ideados para formar estructuras multichip complejas. Para ello, en un primer nivel describimos la configuración en mosaico y el sistema multikernel, para después comentar la idea general de estructura multichip propuesta.

1. Configuración en mosaico.

Ambos chips de convolución tienen 14 bits de direcciones en el bus de entrada AER, 7 para cada coordenada. Así pues, son capaces de ver un espacio de direcciones de 128×128. Sin embargo, las dimensiones del array de píxeles son de 32×32 en el caso de Conv1 y de 64×64 en el caso de Conv2. De este modo, ambos chips ofrecen la posibilidad de configurar la dirección base del array, permitiendo así conectar varios chips en paralelo para emular el comportamiento de un único chip de mayores dimensiones. En el caso de Conv1, podríamos crear un mosaico de 4×4

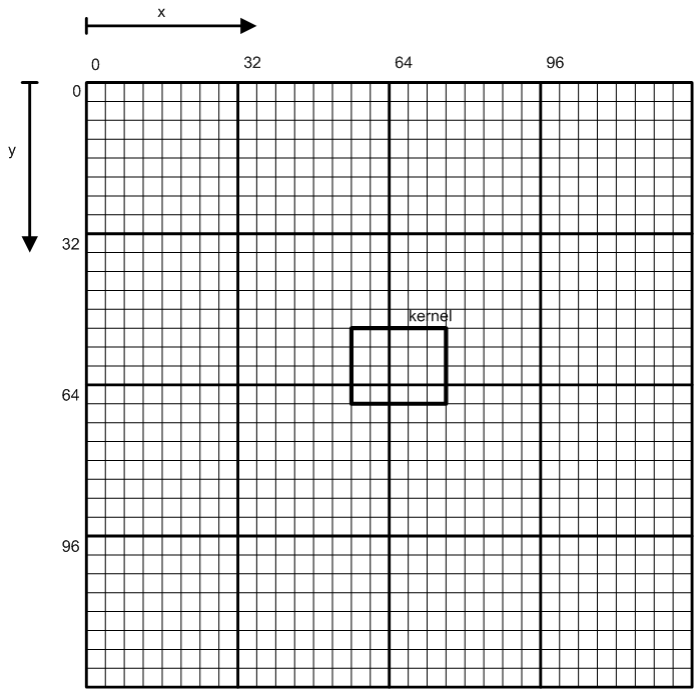


FIGURA 3.13. Configuración en mosaico con varios chips para procesar imágenes mayores.

chips para tener un array equivalente de 128×128, mientras que con Conv2 sería suficiente con un mosaico de 2×2 chips.

Un ejemplo de configuración en mosaico se muestra en la Fig. 3.13. En ella podemos ver un array de 4×4 chips de convolución, cada uno de ellos de 32×32, formando en total un único array de 128×128 píxeles. En el sistema de la figura puede ocurrir que llegue un evento de entrada tal que al aplicar el kernel de convolución el campo proyectivo caiga repartido entre 4 chips diferentes. Así pues, cada uno de los 16 chips del array recibiría dicho evento, y teniendo en cuenta sus propias coordenadas calcularía si alguna parte del campo proyectivo le corresponde a él. Si la respuesta es negativa, lo ignoraría, y si es afirmativa procedería a sumar el kernel sobre los píxeles correspondientes. De este modo, el sistema se comporta verdaderamente como un único array de tamaño total. Para que esto funcione correctamente

es necesario que cada uno de los chips disponga del kernel completo programado en su propia RAM, luego el tamaño máximo del kernel seguirá siendo el mismo que para un único chip individual, en nuestro caso 32×32.

2. Sistema multikernel.

La idea básica del sistema multikernel es que podamos programar varios kernels dentro de un mismo chip de convolución. Este sistema solamente ha sido incluido en el chip Conv2, permitiendo que en la memoria RAM interna se puedan escribir hasta un máximo de 32 kernels, cuyos tamaños vendrán limitados por el tamaño total de la RAM, que sólo cuenta con 32×32 posiciones.

Para implementar este sistema, es necesario añadir algunas modificaciones a distintos niveles, y todo ello está detallado en los capítulos correspondientes a la descripción de cada uno de los bloques. En primer lugar, el bus AER tiene que incluir una serie de bits extra (concretamente 5 bits para el caso de 32 posibles kernels) para que cada evento que llegue a nuestro chip incluya, además de las coordenadas (x, y), un identificador del kernel k_i $(i = 0,\ldots, 31)$ con el que tiene que ser procesado. El objetivo de este sistema es que con un único chip podamos implementar la funcionalidad de varios chips diferentes, reduciendo así el número de elementos necesarios en un sistema multichip. Esto está representado en la Fig. 3.14.

En la parte superior de la Fig. 3.14 se muestra un sistema multicapa convencional, el cual tiene n chips de convolución en paralelo en una capa cualquiera i. Las salidas de cada uno de los n chips de la capa i está conectada a la entrada de uno de los n chips de convolución de la capa $i + 1$. Este tipo de estructura es muy frecuente en las redes neuronales, por ejemplo en los sistemas de reconocimiento de caracteres [74] o de texturas [75], [76] descritos en el apartado anterior. Por último, las n salidas de los chips de convolución de la capa $i + 1$ se unen en un único chip en la capa $i + 2$ que se encarga de sumar las contribuciones de todos ellos. Pues bien, todo este sistema es completamente equivalente al de la parte baja de la Fig. 3.14. En este caso, un único chip de convolución Conv2 podría sustituir a los n chips que había en la capa $i + 1$, más el chip de la capa $i + 2$, simplemente haciendo que cada evento que provenga de la capa i incluya información

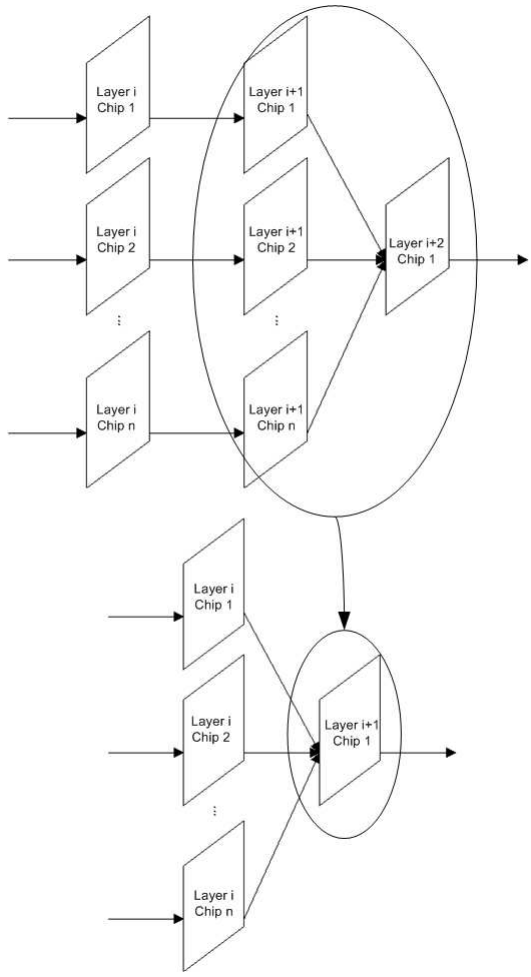


FIGURA 3.14. Descripción del sistema multikernel: en la parte superior, sistema convencional, y en la parte inferior, sistema multikernel equivalente.

sobre su chip de origen, para que nuestro chip Conv2 le aplique el kernel que le corresponda.

3. Estructura general multichip.

En general, independientemente de las dos configuraciones específicas descritas en los puntos anteriores, el objetivo de los chips de convolución

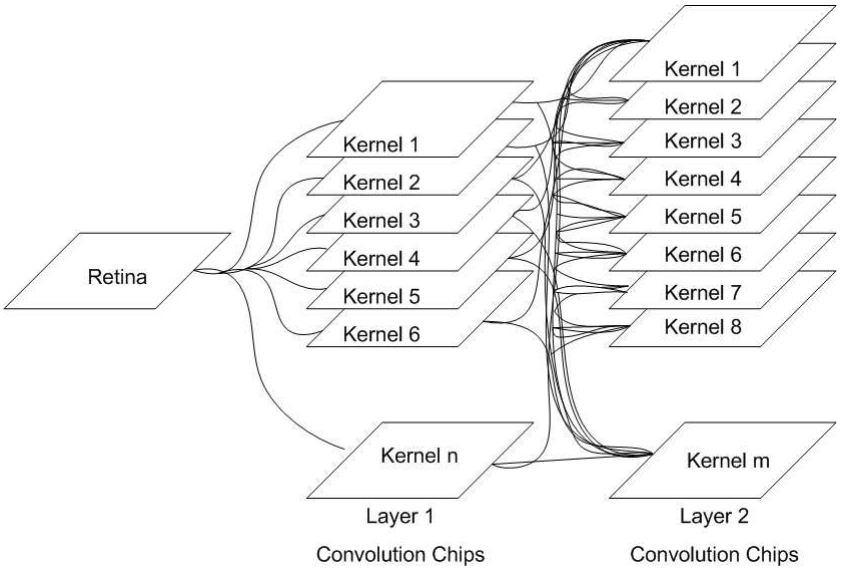


FIGURA 3.15. Sistema multicapa para procesamiento de visión.

Conv1 y Conv2 desarrollados en la presente tesis es formar redes multicapa del tipo de la que se muestra en la Fig. 3.15, aunque con más capas de procesamiento aún.

La idea es que estos chips están diseñados para comunicarse entre sí formando redes con tantos chips de convolución como deseemos, y tan complejas como las queramos implementar. Así, la utilidad de estos chips no terminará con las aplicaciones desarrolladas para esta tesis, sino que son una herramienta con la que seguir trabajando en adelante para construir sistemas cada vez más complejos.

CAPÍTULO 4

El píxel de convolución

4.1. Introducción

En los capítulos anteriores se ha analizado el funcionamiento del cerebro humano para procesamiento de visión, y hemos visto cómo el cerebro se estructura en sistemas multicapa procesadores de eventos. También hemos dicho que el objetivo de la presente tesis es el desarrollo de chips de convolución con los que emular algunas funciones del cerebro. Una vez descrita en el capítulo anterior la arquitectura de dichos chips, en el presente capítulo vamos a analizar la unidad básica de procesamiento en la cual se basan: el píxel de convolución.

En primer lugar, vamos a describir brevemente el comportamiento de la unidad básica de procesamiento del cerebro: la neurona biológica. A continuación, en la Sección 4.3 se hace referencia al píxel analógico propuesto en un trabajo previo, explicando cómo trata de emular el funcionamiento de la neurona biológica, aunque con una serie de desventajas. Por último, en la Sección 4.4 se describe el píxel digital objeto de esta tesis, diferenciando entre las dos versiones propuestas: la versión inicial integrada en el chip de convolución Conv1, y la versión avanzada integrada en el chip Conv2.

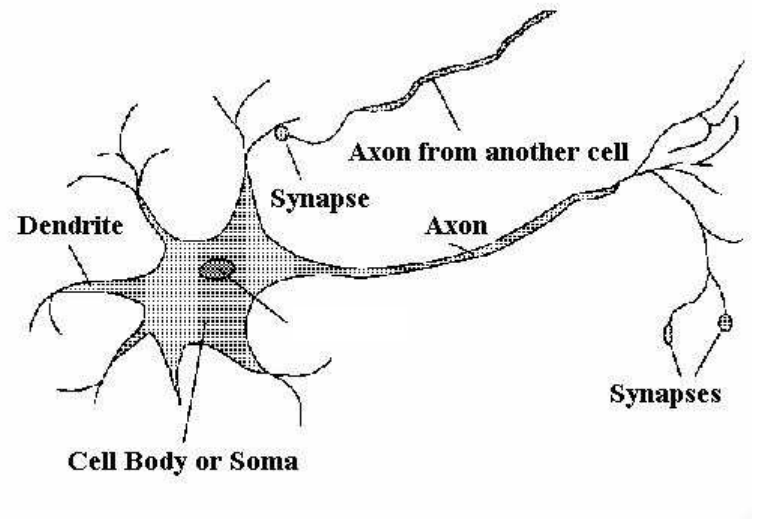


FIGURA 4.1. Estructura básica de una neurona.

4.2. La neurona biológica

El modelado y la implementación de cualquier red neuronal artificial está basado en la estructura y en el modo de operación del sistema nervioso biológico, y la estructura básica de dicho sistema es la neurona.

La neurona es la unidad elemental de procesamiento, y forma dentro del cerebro una densa red con más de 10^4 unidades y varios kilómetros de conexiones por milímetro cúbico [80], lo cual lleva a una estimación total del orden de 10^{11} neuronas en el cerebro humano [82], con una cantidad de conexiones del orden de 10^{15} [83].

Una neurona típica consta de tres partes fundamentales, que se pueden ver en la Fig. 4.1: soma o cuerpo neuronal, axón y dendritas. En general, las señales que provienen de otras neuronas llegan por las dendritas. Si el nivel de excitación causado por esta entrada es suficiente, una señal de salida es generada, la cual se propaga a través del axón y sus ramificaciones a otras neuronas. La unión entre una ramificación del axón y la dendrita de otra

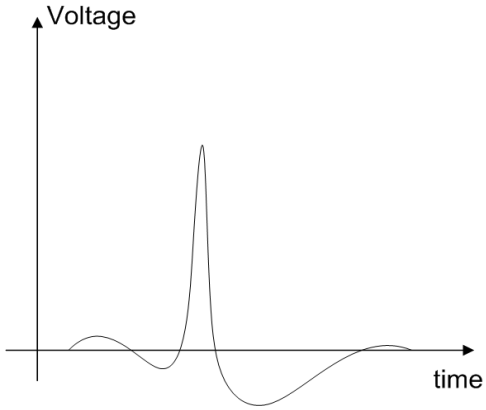


FIGURA 4.2. Aspecto de un impulso eléctrico transmitido entre neuronas.

neurona se llama sinapsis, y también se muestra en la Fig. 4.1. Es habitual referirse a una neurona emisora como presináptica, y a una neurona receptora como postsináptica.

Cuando hablamos de las señales que se transmiten entre neuronas, se trata tanto de un proceso químico como eléctrico, ya que la señal que se propaga a lo largo del axón es un impulso eléctrico, pero cuando éste llega a las sinapsis son unas sustancias especiales llamadas neurotransmisores las que se propagan.

El impulso eléctrico que se propaga entre las neuronas tiene la forma que podemos ver en la Fig. 4.2, con una duración de 1 ó $2ms$. La generación de este impulso está relacionada con la composición de la membrana de la neurona, la cual es semipermeable, de modo que en reposo alcanza unos $70mV$ de diferencia de potencial entre ambas partes, debido a una diferente concentración de iones. El líquido interno tiene una concentración de potasio 10 veces mayor que en el líquido exterior. Sin embargo, la concentración de sodio es 10 veces mayor fuera que dentro. Esto es lo que se llama potencial de reposo.

Cuando la neurona recibe neurotransmisores a través de las dendritas, sus efectos se acumulan causando que el potencial de la membrana caiga lentamente. Esto produce un cambio en la permeabilidad de la membrana,

permitiendo que los iones la atraviesen (el sodio entre y el potasio salga). De este modo, la polaridad de la membrana se invierte, y como consecuencia se produce un impulso eléctrico. El potencial de inversión causa que la permeabilidad de la membrana vuelva a cambiar, y la neurona vuelva a su estado de reposo.

El potencial de inversión se propaga a lo largo del axón, y produce la emisión de neurontransmisores en las dendritas. Como consecuencia, las neuronas producen trenes de pulsos cuya frecuencia es proporcional a la cantidad de neurotransmisores recibidos a su entrada. Una característica de este mecanismo es que las neuronas más activas emitirán pulsos más rápido que las demás. De este modo, la información más importante se propaga mucho más rápido por las diferentes capas de neuronas a lo largo del cerebro para producir respuestas muy rápidas con sólo una pequeña cantidad de pulsos.

En cuanto a las sinapsis, hay dos tipos de ellas: las inhibitorias, cuyos neurotransmisores tienden a estabilizar el potencial de la membrana, y las excitatorias, cuyos neurotransmisores tienden a decrementar dicho potencial, y como consecuencia a favorecer la producción de pulsos. Cada neurona tiene como entradas sinapsis de los dos tipos. Estas sinapsis tienen unos pesos que potencian más o menos la influencia de los neurotransmisores, y se regulan permitiendo el aprendizaje a través del cambio de dichos pesos.

Cuando una neurona no recibe ningún estímulo, la membrana permite el paso de una pequeña cantidad de iones de dentro hacia fuera, haciendo que el potencial de la membrana tienda hacia el potencial de reposo. De este modo, los eventos de entrada más antiguos tienen menos relevancia en el estado actual de la neurona. Es lo que podríamos llamar mecanismo de olvido. Este mecanismo es fundamental para detectar correlaciones espacio-temporales, que es una forma en la que se codifica la información en el cerebro [81]. Sin este mecanismo de olvido, sería imposible distinguir la información nueva de la antigua.

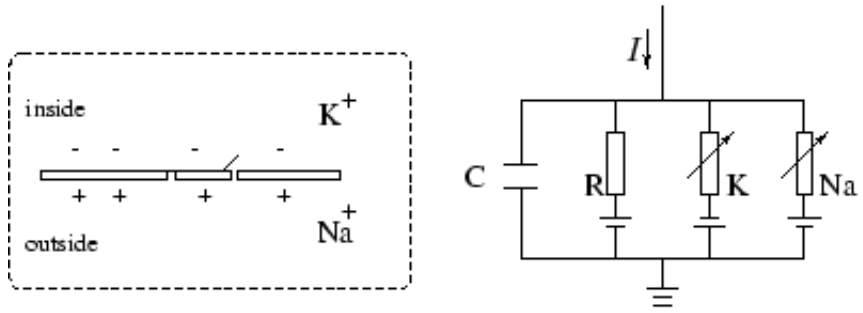


FIGURA 4.3. Esquema del modelo neuronal propuesto por Hodgkin y Huxley.

4.3. Primera propuesta: el píxel analógico

A la hora de diseñar modelos que reproduzcan el comportamiento de la neurona biológica, hay diferentes alternativas. Una de las más utilizadas es el modelo propuesto por Hodgkin y Huxley [84], que consiste en un modelo basado en conductancias extraído del estudio del comportamiento de las neuronas de los calamares. La idea básica de este modelo es reproducir el flujo de iones a través de la membrana mediante conductancias. El modelo aparece representado en la Fig. 4.3. Como se puede ver, está basado en tres conductancias: *Na* modela el tránsito de iones de sodio a través de la membrana, *K* representa el flujo de potasio desde el interior hasta el exterior de la célula, y *R* modela las pérdidas que implementan el mecanismo de olvido. Como se ve en la figura, los valores de las conductancias relativas al flujo de iones (tanto de sodio como de potasio) son variables, ya que dependen del potencial de la membrana; sin embargo, el efecto de las pérdidas es constante. Asimismo, la corriente que aparece en la figura modela las entradas recibidas por la neurona producidas por conexiones con otras neuronas.

Este modelo describe con gran precisión el comportamiento de la membrana de potencial, pero es bastante complicado computacionalmente. Por este motivo se propuso el modelo SRM (*Spike Response Model*) como una simplificación del modelo anterior [85]. Este modelo está representado en la Fig. 4.4. Cada vez que llega un pulso por un axón de entrada, se

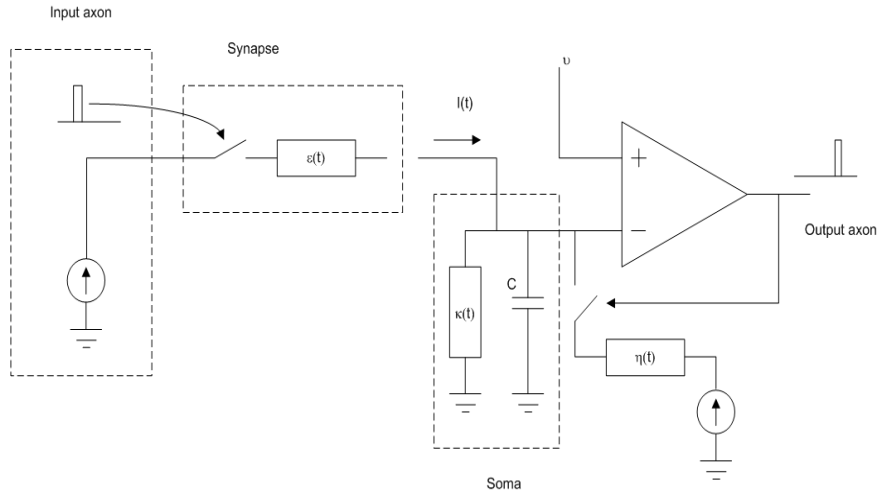


FIGURA 4.4. Esquema del modelo neuronal SRM.

inyecta una cierta cantidad de carga al soma de la neurona de acuerdo con el peso de la sinapsis correspondiente, que está modelado a través de la función $\varepsilon(t)$. Esta carga se integra en una capacidad C, de modo que cuando la tensión de esta capacidad alcanza un cierto umbral υ, la neurona produce un pulso por el axón de salida y resetea su estado a un cierto valor de reposo. El efecto de este pulso de reset sobre el potencial de la neurona está modelado a través de la función $\eta(t)$. El efecto de cualquier entrada externa sobre el potencial de la membrana se encuentra modelado por la función $\kappa(t)$. Todas estas interacciones entre distintos elementos del modelo siguen unas ciertas funciones temporales que en general dependen tanto del instante actual como de los instantes de llegada de los últimos eventos, para acercarse al comportamiento de la neurona biológica todo lo posible.

La neurona analógica *Integrate&Fire* utilizada por Serrano [78] está basada en este modelo SRM, aunque de forma simplificada, reduciendo la complejidad de las funciones internas de dicho modelo $\varepsilon(t)$, $\eta(t)$ y $\kappa(t)$ hasta convertirlas en pulsos cuadrados. En la Fig. 4.5 se muestra una versión reducida de dicha neurona. En ella, cada vez que llega un pulso de entrada se cierra el *switch* inyectando corriente en la capacidad, incremen-

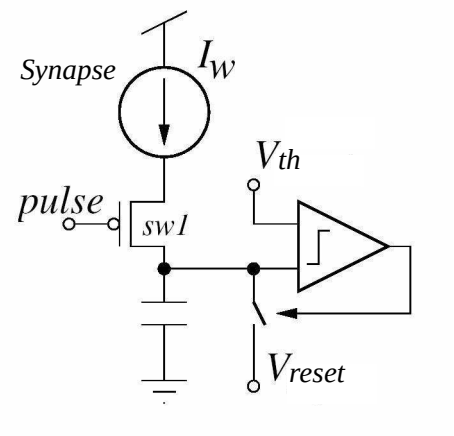


FIGURA 4.5. Esquema simplificado de la neurona analógica Integrate&Fire utilizada por Serrano [78].

tando el valor de tensión de dicha capacidad una cantidad $\Delta V = \frac{I_w \cdot \Delta T}{C}$, siendo I_w el valor de la corriente de entrada, ΔT la duración del pulso y C el valor de la capacidad. Cuando la tensión del condensador alcanza el valor umbral V_{th}, el comparador producirá un pulso de salida que reseteará el condensador descargándolo hasta un valor de tensión V_{reset}.

De forma general, los pulsos de entrada pueden ser tanto positivos como negativos, luego según el signo se producirá una inyección o una sustracción de carga en el condensador, en ambos casos a partir de una corriente proporcional al peso proporcionado por el kernel correspondiente. Esto se puede ver en la Fig. 4.6, donde se muestra el esquemático completo del píxel analógico. Cuando el píxel recibe un pulso de entrada por *Pulse+* o *Pulse-*, el bloque *Logic* se encarga de activar o desactivar los transistores $M_{n1} - M_{n3}$ y $M_{p1} - M_{p3}$ en función del valor del kernel previamente almacenado en el propio bloque y del signo del evento. De este modo, las estructuras formadas por estos transistores generan un pulso de corriente a partir de la corriente de calibración correspondiente I_{calN} o I_{calP}. Asimismo, en lugar de un único comparador, son necesarios dos de ellos para poder detectar un umbral negativo o positivo en la tensión de la capacidad, y poder generar pulsos de salida con signo. Estos comparadores son llamados en la Fig. 4.6 *Positive Event Block* y *Negative Event Block*. Todo esto incluye una

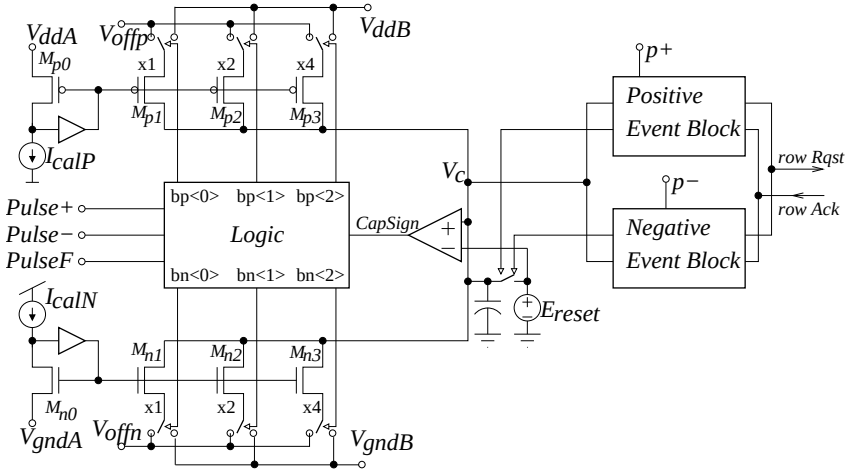


FIGURA 4.6. Esquemático completo del píxel de convolución analógico.

cierta complejidad en el píxel analógico, ya que además es necesario almacenar en el propio píxel el valor del peso que tiene que aplicar para cada evento. Por este motivo incluye una memoria dinámica dentro del bloque *Logic*.

Por otra parte, para obtener un correcto funcionamiento es necesario incluir circuitería de calibración en el interior del píxel. Esta necesidad de calibración es debida al *mismatch* entre transistores polarizados en región subumbral. Para compensar estas variaciones, se incluye en cada píxel unas celdas de memoria en las que almacenar una palabra digital de 5 bits que controla un espejo de corriente que se usa para calibrar una corriente de referencia común para todos los píxeles, produciendo para cada píxel sus propias corrientes ya calibradas I_{calN} e I_{calP}. Además de complicar el píxel (tanto en términos de área como de consumo), esto implica un proceso inicial de calibración que se tiene que llevar a cabo una vez fabricado, calculando la palabra digital de calibración óptima para cada píxel.

En la Fig. 4.7 se muestra una representación de las variaciones de la tensión del condensador conforme va recibiendo pulsos de entrada (tanto positivos como negativos). Dicha tensión va incrementándose y decrementándose en función del signo de los eventos de entrada, hasta que alcanza un

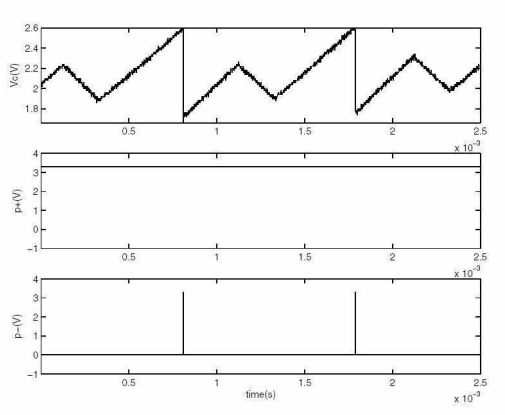


FIGURA 4.7. Representación de la tensión del condensador y pulsos de salida producidos por el píxel.

valor umbral y se resetea generando un pulso de salida del signo correspondiente.

Resumiendo, las principales limitaciones que presentaba este píxel analógico eran las siguientes:

1. Complejidad incluida a nivel de píxel a causa de la necesidad de calibración para compensar el *mismatch* entre transistores.
2. Reducida precisión conseguida por el píxel, debido a la operación en subumbral de los transistores analógicos.
3. Alto valor de latencia conseguido (alrededor de $1ms$ de retraso entre un evento de entrada y su correspondiente salida), producido por la lentitud en los comparadores internos del píxel polarizados en baja corriente.

Con la finalidad de resolver los principales problemas del píxel analógico, se propone el píxel digital objeto de la presente tesis, descrito en la siguiente sección.

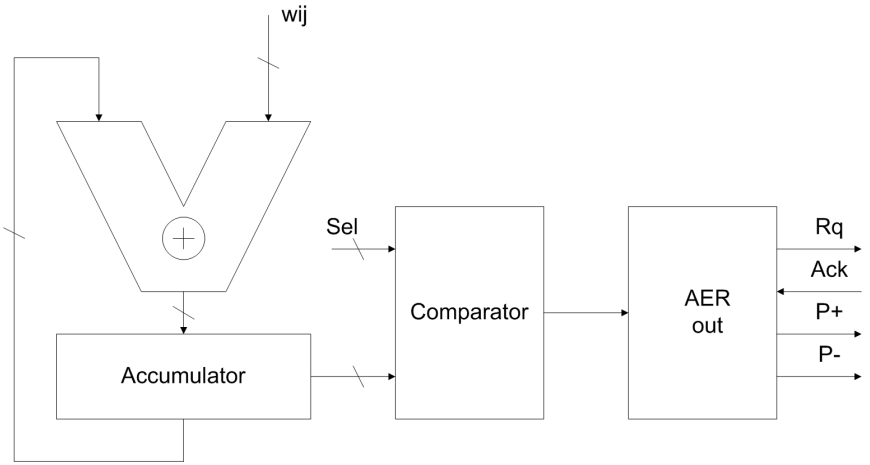


FIGURA 4.8. Esquema básico del píxel digital de convolución propuesto.

4.4. El píxel digital

En las dos versiones de chip de convolución propuestas (Conv1 y Conv2), la operación de convolución se lleva a cabo a nivel de píxel mediante la integración de eventos de entrada pesados convenientemente por los valores del kernel. El esquema genérico del píxel digital se muestra en la Fig. 4.8. Como se puede ver, el píxel consiste de forma genérica en un acumulador y un sumador, el cual recibe como entradas el valor del kernel y el propio acumulador, además de un comparador que genera un pulso cuando el valor del acumulador alcanza un límite seleccionado y un bloque que se encarga de gestionar la comunicación del píxel con la periferia AER para generar eventos. Para cada una de las dos versiones de chip de convolución, se ha diseñado un píxel diferente, aunque los dos se basan en este esquema genérico. A continuación vamos a describir de forma detallada cada una de las dos versiones del píxel digital, comenzando por la versión inicial para seguir después con la versión avanzada.

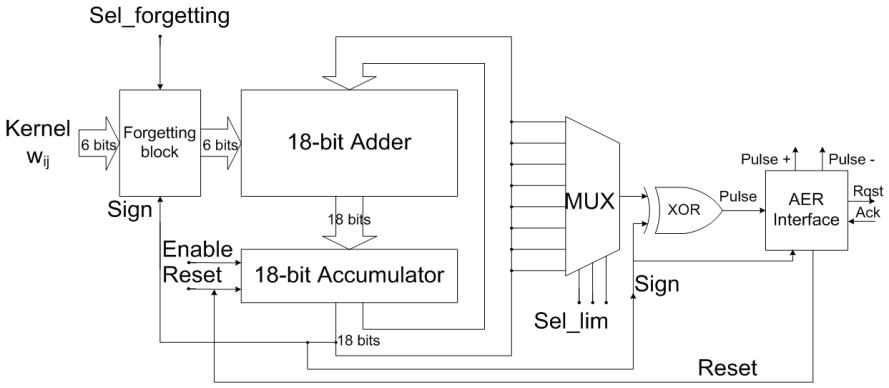


FIGURA 4.9. Diagrama de bloques del píxel digital de convolución, en su versión inicial.

4.4.1. Versión inicial Conv1

La parte principal del píxel es un sumador de 18 bits y un acumulador de la misma resolución. En la Fig. 4.9 se muestra el diagrama de bloques de dicho píxel. En esta primera versión se implementó un tamaño de sumador sobredimensionado de forma intencionada con el objetivo de permitir la máxima precisión posible. El criterio utilizado fue permitir la acumulación de un kernel de tamaño 32×32 con todos sus pesos al máximo valor posible para sus 6 bits (los valores del kernel se codifican en 6 bits), permitiendo a su vez que el bit menos significativo de dicho kernel también tuviera la posibilidad de contribuir al valor del acumulador. De este modo, se seleccionó un rango dinámico de 18 bits.

Si tenemos en cuenta los modelos analógicos descritos previamente donde el estado de la neurona se almacenaba en la tensión de un condensador, en nuestro caso el estado del píxel se almacena en el acumulador, representado en un número con signo codificado en complemento a 2 con 18 bits (17 más el signo). Esto implica que el estado del píxel puede variar entre $-2^{17} = -131072$ y $2^{17} - 1 = 131071$. A su vez, los valores del kernel se codifican también en complemento a 2, pero con 6 bits solamente (5 más el signo), luego pueden variar entre $-2^5 = -32$ y $2^5 - 1 = 31$.

El comportamiento del píxel es el siguiente. Cada vez que recibe un evento de entrada, la señal *enable* (que es común para todos los píxeles de una misma fila) se activa, produciendo que el acumulador se actualice sumando a su valor anterior el correspondiente peso del kernel w_{ij}. Si el nuevo valor del acumulador alcanza el umbral, se genera un evento de salida y se resetea el acumulador, volviendo a su valor en reposo, que es 0.

En esta primera versión del píxel se buscaba la mayor programabilidad posible, así que se incluyó un multiplexor de 8 entradas para poder seleccionar entre 8 distintos límites del acumulador, con la idea de poder adaptar el tamaño del mismo para diferentes aplicaciones. Para ello, usamos un parámetro de control de 3 bits (*Sel_lim*), con el cual se elige uno de los bits del acumulador. Este bit se compara de forma continua con el bit de signo (que es el más significativo). De este modo, para datos positivos (es decir, que tienen el $msb = 0$) el comparador disparará cuando el bit seleccionado se ponga a '1', mientras que para datos negativos ($msb = 1$), disparará cuando el bit seleccionado valga '0'. Los 8 posibles umbrales seleccionables se muestran en la Tabla 4.1. Es importante tener en cuenta que, como se muestra en la Fig. 4.9, utilizamos una puerta XOR como comparador para detectar el umbral. Por este motivo, hay que tener precaución a la hora de seleccionar el máximo valor posible del kernel w_{ij} en función del bit seleccionado para llevar a cabo la comparación. Así pues, debemos asegurarnos que los valores máximos utilizados en el kernel serán menores o iguales que el límite seleccionado para el acumulador, tanto en lo referente a los valores positivos como a los negativos. Si el valor del kernel fuera mayor que el umbral programado, cabría la posibilidad de que no se produjera el evento correspondiente. Por ejemplo, si seleccionamos como umbral el bit 2 (es decir, que dispare cuando el acumulador alcance el valor 4) se produciría un error si sumáramos un valor 8 ($001000|_2$), ya que no afectaría al bit elegido para el comparador. Este problema se resuelve limitando los valores del kernel a los propios umbrales del acumulador.

En cuanto a la implementación del mecanismo de olvido a nivel de píxel, éste recibe un pulso de olvido de forma periódica que es generado por el controlador síncrono. Dicho pulso de olvido produce un pulso en la señal *enable* que llega a todos los píxeles del array, provocando un cambio en el valor del acumulador. De este modo, cuando el píxel recibe un pulso en la

TABLA 4.1. Posibles límites programables para el acumulador.

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	$2^{2} = 4$	$-2^{2} - 1 = -5$

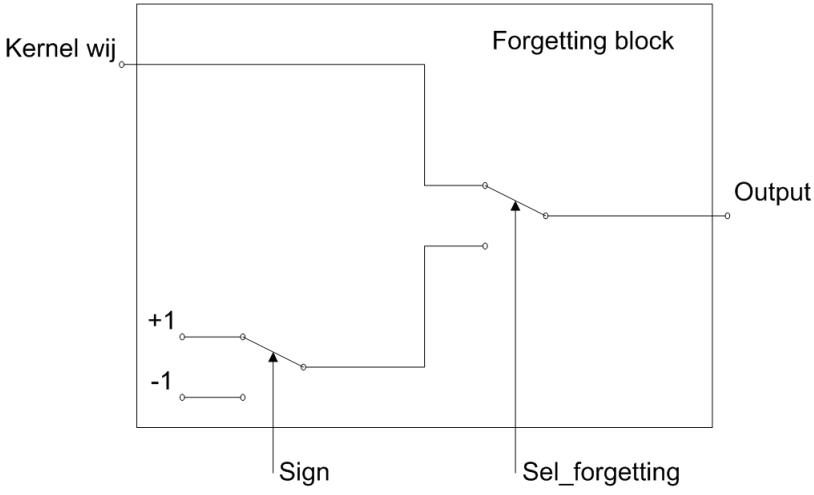


FIGURA 4.10. Esquema del bloque de olvido.

señal de *enable*, en función del valor de la señal *Sel_forgetting* actuará de un modo u otro. Si esta señal indica que se trata de un pulso de olvido, en lugar de tomar como entrada del sumador el valor del kernel almacenado en la

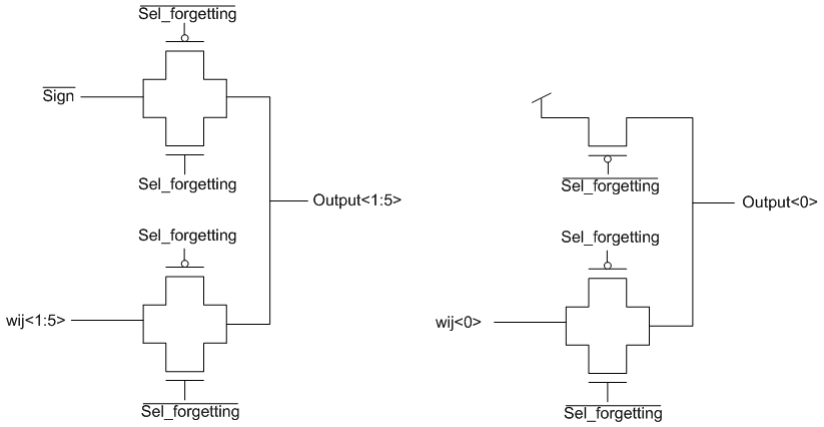


FIGURA 4.11. Esquemático detallado del bloque de olvido.

RAM w_{ij}, el bloque de olvido ('*Forgetting block*' en la Fig. 4.9) selecciona una entrada diferente para el sumador, como se muestra en el esquema de la Fig. 4.10. Este bloque incluye una serie de *switches* controlados por la señal *Sel_forgetting*, de forma que durante el procesamiento de un evento seleccionan el valor del kernel, mientras que durante un pulso global de olvido estos *switches* conmutan, seleccionando '1' como entrada si el valor del acumulador es negativo o '-1' si es positivo. El esquemático a nivel de transistores del bloque de olvido se muestra en la Fig. 4.11, donde se ve cómo la señal *Sel_forgetting* y su correspondiente valor negado seleccionan para cada bit de salida entre el bit de entrada y el valor del signo invertido. Así, para *Sel_forgetting*=0, tendremos *Output*=w_{ij}, mientras que para *Sel_forgetting*=1, tendremos *Output*=$000001|_2 = 1|_{10}$ si el signo del acumulador es positivo (*Sign*=0), o bien *Output*=$111110|_2 = -1|_{10}$ si el signo es negativo (*Sign*=1). De este modo, si un píxel deja de recibir eventos, en un tiempo controlable a través de la frecuencia de olvido alcanza un estado de reposo con el acumulador a '0'. Siendo precisos, un píxel que no recibe eventos de entrada tendrá su acumulador oscilando entre '0' y '-1', valores producidos por el mecanismo de olvido.

En la Fig. 4.12, el diagrama temporal muestra cómo se suma el valor del kernel al acumulador. La señal *Clk* que aparece en la figura es el reloj del controlador síncrono del sistema, y las señales *Read*, *Enable* y

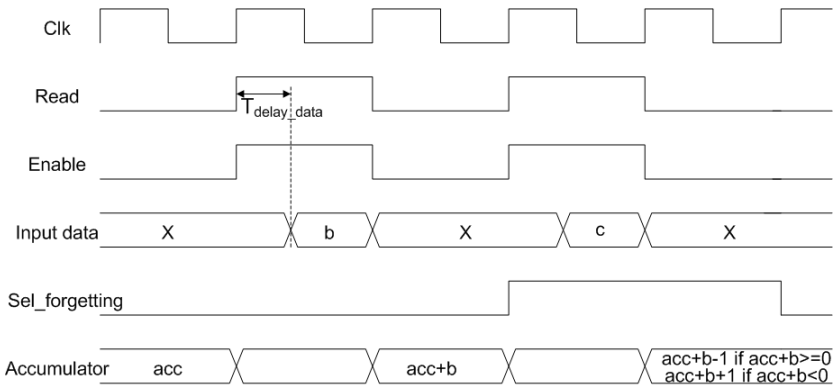


FIGURA 4.12. Diagrama temporal del funcionamiento del píxel.

Sel_forgetting son generadas por dicho controlador. Cuando se activa la señal *Read*, se habilita la lectura de un dato de la RAM, enviando el valor del kernel almacenado en la posición correspondiente a todos los píxeles del array que se encuentran en una misma columna. Al mismo tiempo, se genera la señal *Enable*, que hace que la salida del sumador se almacene en el acumulador con el flanco de bajada. El retraso de esta operación establece un límite para la duración de un pulso de *Enable* (que coincide con la duración de un ciclo de reloj). Como consecuencia, debemos asegurarnos de que $T_{clk} \geq T_{delay\_data}$, siendo T_{clk} el periodo de reloj, y T_{delay_data} el tiempo que transcurre desde que se habilita la lectura del dato en la memoria RAM hasta que dicho dato ha sido sumado al estado previo del píxel. Este retraso dependerá tanto de la estructura sumadora incluida en el píxel como del tiempo que tarda el dato en atravesar los demás bloques del chip (inversor de complemento a 2 y bloque de desplazamiento horizontal), y marcará la máxima frecuencia de reloj que podamos utilizar. Esta limitación se analizará con detenimiento más adelante. Por otra parte, el diagrama de la Fig. 4.12 también muestra cómo el estado del píxel se incrementa o decrementa cuando recibe un pulso de olvido, dependiendo del signo de su estado anterior.

El diagrama temporal de la Fig. 4.13 muestra lo que ocurre cuando el acumulador alcanza el umbral programado, ya sea el límite positivo o el negativo. El *handshaking* de 4 fases se entiende mejor con la ayuda del esquemático de la Fig. 4.14, que describe la interfaz AER de la Fig. 4.9,

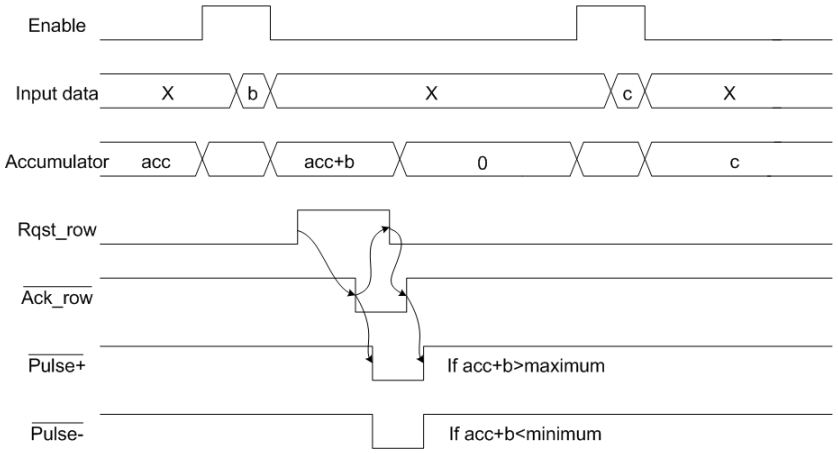


FIGURA 4.13. Diagrama temporal del handshaking entre el píxel y la periferia.

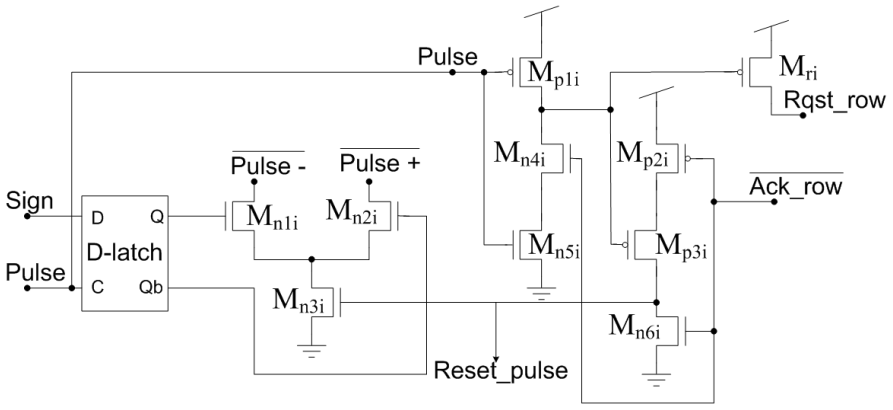


FIGURA 4.14. Esquemático del bloque de interfaz dentro del píxel que se encarga de la comunicación con la periferia.

encargado de llevar a cabo la comunicación entre el píxel y la periferia. Cuando el píxel aún no ha alcanzado su valor umbral, la señal *Rqst_row* está a nivel bajo, mientras que las señales $\overline{Ack\_row}$, $\overline{Pulse+}$ y $\overline{Pulse-}$ están a nivel alto (ya que son activas a nivel alto). Todas estas señales son fijadas a estos niveles de reposo por una serie de *pull-ups* y *pull-downs* que se encuentran en el bloque generador de AER, en la periferia. Al mismo tiempo, las señales *Pulse* y *Reset_pulse* se encuentran a nivel bajo. Cuando

el acumulador alcanza el umbral programado (ya sea el positivo o el negativo), la señal *Pulse* se pone a nivel alto (por el comparador XOR), y al activar el transistor M_{n4i}, el inversor formado por los transistores M_{p3i} y M_{ri} provocará la activación de la señal *Rqst_row*. Esta señal es común para todos los píxeles de una misma fila, implementando de este modo una función OR cableada entre todos ellos. Así cada vez que un píxel de una fila alcanza su umbral, el arbitrador por filas recibe una petición de la fila correspondiente. De este modo, cuando el arbitrador por filas asiente a esta petición, activa la señal $\overline{Ack\_row}$ a nivel bajo, provocando que el inversor formado por los transistores M_{p2i} y M_{n6i} active la señal *Reset_pulse* (a nivel alto). Esta señal resetea el valor del acumulador, haciendo que la señal *Pulse* vuelva al estado de reposo a nivel bajo, cortando el transistor M_{ri} y dejando la señal *Rqst_row* en alta impedancia para que pueda ser devuelta a nivel bajo por los *pull-downs* de la periferia. Mientras que la señal $\overline{Ack\_row}$ está activa, el transistor M_{n4i} se mantiene cortado para evitar que un nuevo evento sea enviado a la periferia antes de que termine el *handshaking*. La señal *Reset_pulse* se encarga de activar el transistor M_{n3i}, activando o bien $\overline{Pulse+}$ o bien $\overline{Pulse-}$, en función del valor del signo almacenado en *D-latch* (valor que se almacenó cuando la señal *Pulse* se activó). Una vez que el arbitrador por filas detecta que la señal *Rqst_row* ha vuelto a nivel bajo, desactiva la señal $\overline{Ack\_row}$, haciendo que vuelva a nivel alto y corte el transistor M_{n3i}. Por último, cualquiera de las señales $\overline{Pulse+}$ o $\overline{Pulse-}$ que había sido activada previamente vuelve a su nivel de reposo gracias a los *pull-ups* de la periferia.

En cuanto a la estructura utilizada para el sumador, se consideraron varias alternativas [86], [87]. A pesar de que había sumadores con menor número de transistores y con menor consumo, en esta primera versión del píxel digital se optó por utilizar el sumador convencional CMOS por presentar menores tiempos de subida y de bajada en las señales de salida de cada bit (suma y acarreo), para evitar posibles problemas al poner 18 de estos sumadores en cascada. Así, el sumador utilizado es el que se muestra en la Fig. 4.15, con un total de 28 transistores por cada bit. Cada celda del sumador recibe un bit de cada sumando (A y B) y el acarreo del bit anterior (Cin), y genera la suma (Sum) y el acarreo (Cout).

El acumulador está formado por 18 *flip-flops* (uno por cada bit) en los cuales se almacena la salida del sumador en el flanco de bajada de la señal

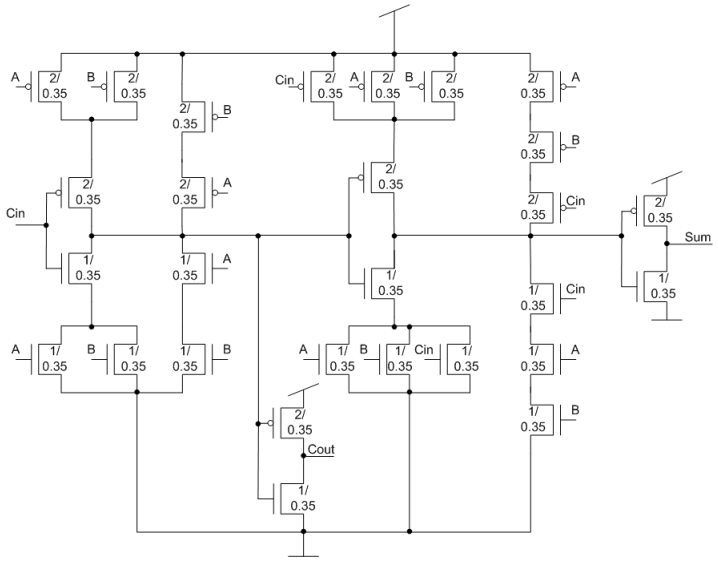


FIGURA 4.15. Sumador convencional CMOS utilizado en la versión inicial del píxel digital.

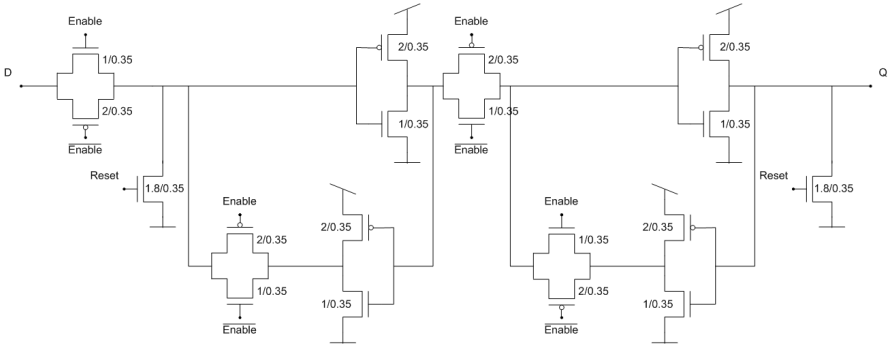


FIGURA 4.16. Esquemático del flip-flop utilizado para implementar el acumulador.

Enable. El esquemático de uno de estos bloques puede verse en la Fig. 4.16. La señal de *Reset* se encarga de poner a 0 el acumulador cada vez que se alcanza el umbral, o bien cuando se fuerza un reset exteriormente.

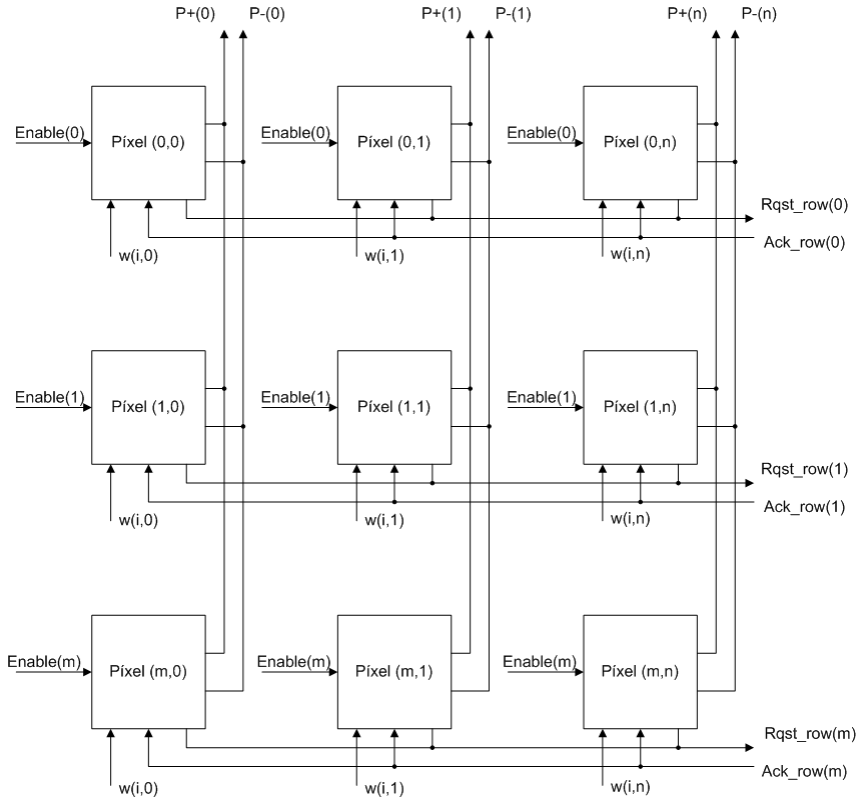


FIGURA 4.17. Estructura de la configuración en array del píxel de convolución.

El píxel ha sido diseñado para formar un array de tamaño 32×32. Como ya hemos visto al describir su funcionamiento, hay una serie de señales compartidas por los píxeles. Concretamente, los datos del kernel son compartidos por todos los píxeles en una misma columna, ya que la RAM se selecciona fila a fila, de modo que cuando se selecciona una de las filas, cada dato de esa fila es visto por todos los píxeles de una misma columna. Así, para que la operación sólo le afecte a la fila de píxeles correspondiente a la dirección del evento de entrada, cada señal *Enable* es común para todos los píxeles de una misma fila, aunque sólo tendrá algún efecto para los píxeles que estén recibiendo un dato válido de la RAM (es decir, un dato no nulo, que es lo que reciben los píxeles que quedan fuera del campo proyec-

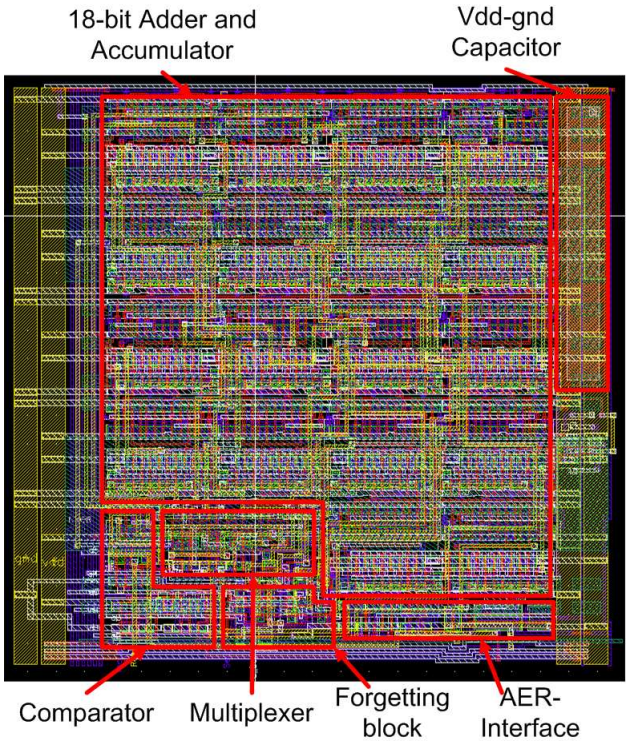


FIGURA 4.18. Layout de la versión inicial del píxel digital.

tivo). En cuanto a las señales de *handshaking*, tanto *Rqst* como *Ack* son comunes por filas, mientras que las señales que indican el signo del evento generado (*Pulse+* y *Pulse-*), son comunes por columnas. De este modo, la configuración en array sigue el esquema de la Fig. 4.17.

En cuanto al layout del píxel, lo podemos ver en la Fig. 4.18. El área total que ocupa es $95,6 \times 101,3 \mu m^2$. La mayor parte del área es ocupada por los 18 bits del sumador y el acumulador, mientras que el resto de los bloques ocupan un área menor. Hemos incluido en el píxel una capacidad entre alimentación y tierra para filtrar *glitches* de potencia. Esta capacidad, de $240fF$, está situada bajo las líneas de alimentación y tierra para evitar un consumo extra de área.

El rutado de las líneas dentro del píxel es un tema bastante delicado, con algunas capacidades parásitas de acoplo bastante críticas, ya que algunas de ellas son compartidas por todos los píxeles en una misma fila o columna, y pueden alcanzar longitudes de hasta $3mm$. Algunas de estas líneas se usan para parámetros de configuración, para los cuales fijamos un valor al conectar el chip y permanecen en reposo durante el funcionamiento normal. Así pues, hemos aprovechado estas líneas "estáticas" para situarlas entre las rápidas líneas dinámicas, evitando acoplos entre dichas líneas dinámicas.

4.4.1.1. Resultados de simulación

Una vez descrita la versión inicial del píxel digital de convolución, vamos a exponer algunos resultados de simulación eléctrica con Spectre (Cadence) que ilustren el comportamiento de dicho píxel. Para comprobar su funcionamiento, en los ejemplos que vamos a mostrar fijamos un valor del dato correspondiente del kernel y seleccionamos un límite para el acumulador. Con esos parámetros fijos, enviamos una serie de eventos de entrada y comprobamos que el píxel genera un evento de salida cuando ha alcanzado el límite establecido para el acumulador.

Un primer resultado de simulación que ilustra el comportamiento del bloque Interfaz AER podemos verlo en la Fig. 4.19. En ella se muestra un caso en el que el píxel ha alcanzado el umbral positivo del acumulador, y tras activar la señal *Rqst_row*, espera la respuesta del arbitrador. Una vez que recibe dicha respuesta a través de la señal *Ack_row*, desactiva la señal *Rqst_row* y activa la señal *Pulse+*, manteniéndola hasta que el arbitrador desactiva la señal *Ack_row*. El motivo por el cual la pendiente de subida de la señal *Pulse+* es más lenta que la de bajada es que no viene fijada por el píxel, que se limita a forzarla a nivel bajo cuando corresponde, sino que la pendiente de subida viene dada por un pull-up de la periferia. Veamos ahora algunos ejemplos del funcionamiento del píxel.

1. En un primer ejemplo, para comprobar con precisión el funcionamiento del sumador, fijamos el dato del kernel a 1. En cada una de las 4 gráficas mostradas en la Fig. 4.20 podemos ver lo que ocurre cuando enviamos eventos al píxel con un valor umbral del acumulador diferente, que vale

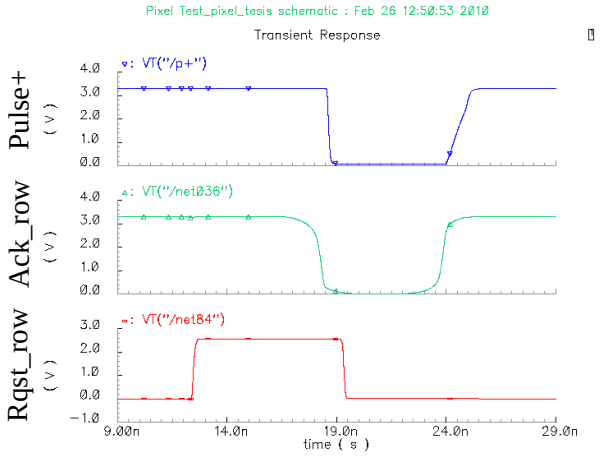
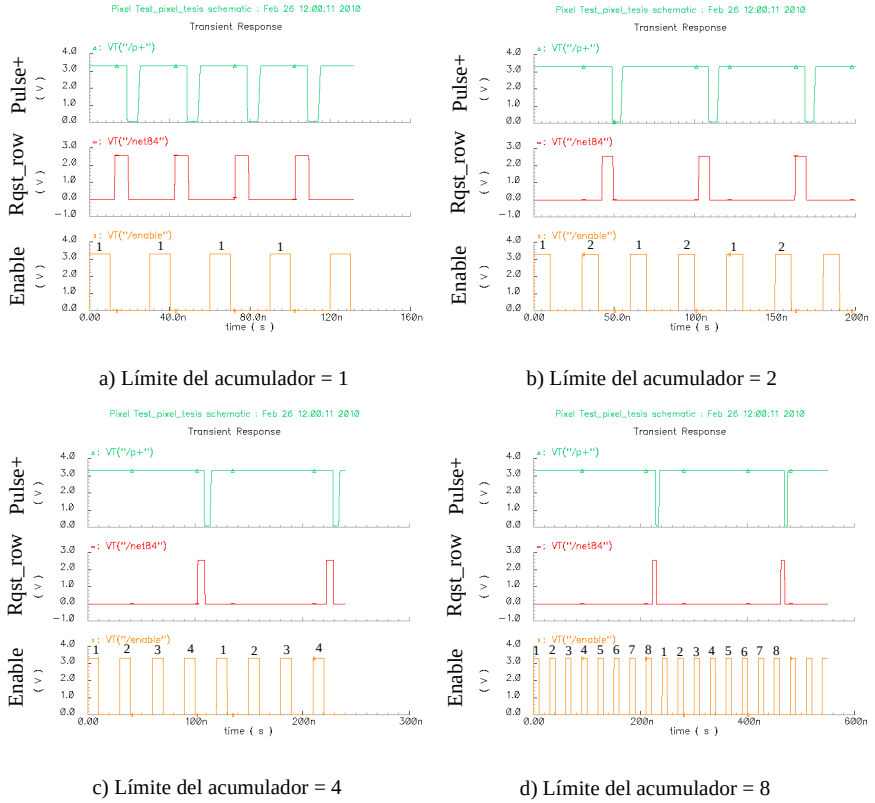


FIGURA 4.19. Señales de handshaking obtenidas en simulación.

en cada caso 1, 2, 4 y 8, respectivamente. En cada gráfica podemos ver la señal *Enable* (entrada) en la parte inferior, y las señales *Rqst_row* y *Pulse+* generadas por el píxel en la parte central y superior, respectivamente. Encima de cada señal *Enable* podemos ver el estado del acumulador después de procesar cada uno de los eventos, lo cual nos permite comprobar cómo todos los pulsos de *Rqst_row* se producen con el valor adecuado.

2. En un segundo ejemplo, vamos a jugar también con la posibilidad de variar el dato del kernel que el píxel está recibiendo. Las diferentes gráficas correspondientes a este ejemplo las podemos ver en la Fig. 4.21. En primer lugar, fijamos el dato a 2, y seleccionamos el límite del acumulador 16, resultando que tras acumular 8 eventos de entrada el píxel alcanza el límite y activa la señal *Rqst_row*, como vemos en la gráfica a). En la gráfica b) mantenemos el mismo valor del dato, pero modificamos el umbral a 32, observando que en este caso son 16 pulsos de entrada los que el sumador ha contado, ya que cada uno añade un kernel de valor 2. Por último, en la gráfica c) le damos al dato del kernel el valor 7, mientras que el umbral lo establecemos en 128. Una vez más, los números sobre los pulsos de *Enable* nos indican el valor del acumulador tras sumar cada uno de los pulsos. Como podemos ver, tras sumar 18 eventos



a) Límite del acumulador = 1

b) Límite del acumulador = 2

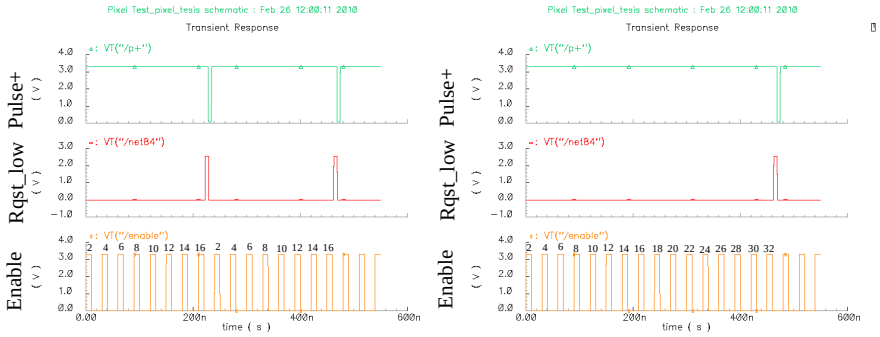
c) Límite del acumulador = 4

d) Límite del acumulador = 8

FIGURA 4.20. Resultados de simulación correspondientes al ejemplo 1, con el dato del kernel igual a 1, y el límite del acumulador variando entre 1, 2, 4 y 8.

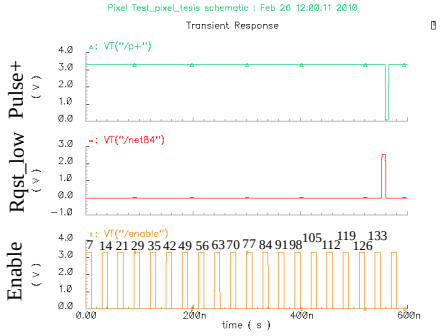
el valor del acumulador es de $7 \times 18 = 126$, lo cual implica que aún no se ha alcanzado el límite. Sin embargo, al sumar el evento número 19, obtenemos $126 + 7 = 133 > 128$, luego ya ha superado el límite, provocando el pulso de salida. En este caso, podemos comprobar cómo el píxel detecta que se ha sobrepasado el límite, aunque la cuenta no produzca el valor exacto, ya que solamente comprueba el cambio en el bit correspondiente (en el caso del límite 128, comprueba el valor del bit 7, ya que $2^7 = 128$).

3. En este otro ejemplo, vamos a observar cómo el píxel suma igualmente un dato negativo, y además tiene en cuenta los pulsos de olvido, como se



a) Límite del acumulador = 16, Dato=2

b) Límite del acumulador = 32, Dato=2



c) Límite del acumulador = 128, Dato=7

FIGURA 4.21. Resultados de simulación correspondientes al ejemplo 2, con el dato del kernel igual a 2 en a) y b) e igual a 7 en c), y el límite del acumulador variando entre 16, 32 y 128, respectivamente.

muestra en la Fig. 4.22. Concretamente, en esta prueba utilizamos como dato del kernel -1, mientras que el límite del acumulador es -2. De este modo, cuando el píxel recibe un primer pulso de *Enable*, el acumulador almacena el valor -1, pero cuando justo a continuación recibe un pulso de olvido, el píxel suma +1 (ya que el estado actual es negativo) y el acumulador se pone a 0 de nuevo. A continuación, cuando llegan dos pulsos de *Enable* consecutivos sin que haya ningún olvido, entonces sí alcanza el acumulador el valor -2 , y genera un evento de salida. En esta figura se puede observar también como la señal que se activa es *Pulse-*, y no

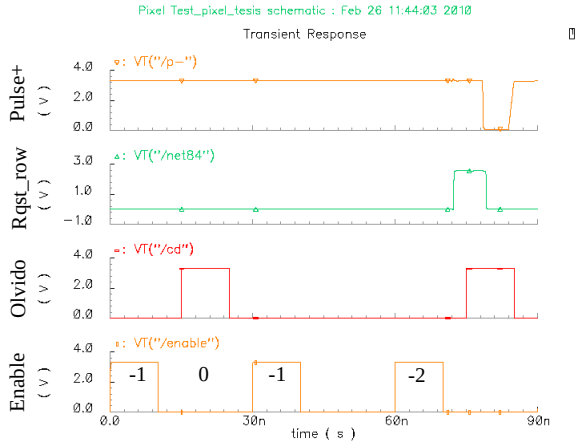


FIGURA 4.22. Resultados de simulación correspondiente al ejemplo 3, con el dato del kernel igual a -1, el límite del acumulador igual a -2, y con el efecto del olvido.

Pulse+ como en los ejemplos previos, indicando correctamente el signo del evento producido.

4.4.2. Versión avanzada Conv2

Una vez fabricado y testado el chip de convolución Conv1, basado en la versión inicial del píxel digital descrita en el apartado anterior, comprobamos que para ciertas aplicaciones en las que queremos obtener rápidamente eventos de salida como resultado de la operación de convolución no necesitamos acumuladores con gran cantidad de bits. Asimismo, observamos que sería interesante poder alcanzar en un único chip una mayor resolución espacial, es decir, mayor cantidad de píxeles integrados. Por este motivo, planteamos esta versión avanzada del píxel de convolución, con el objetivo de incrementar el número de píxeles integrados en un mismo chip sin que ello suponga un coste de área.

Las dimensiones utilizadas para esta nueva versión del píxel digital son 6 bits para el acumulador (5 más el signo), y 4 bits para los datos del kernel

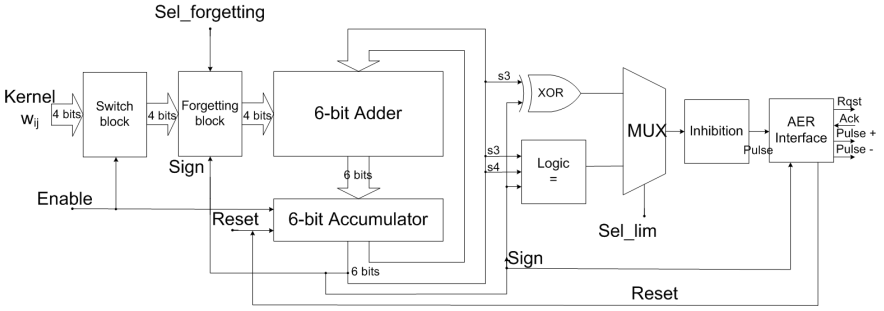


FIGURA 4.23. Diagrama de bloques de la versión avanzada del píxel digital de convolución.

(3 más el signo). De este modo, el acumulador puede representar valores codificados en complemento a 2 entre $-2^5 = -32$ y $2^5 - 1 = 31$, mientras que el kernel puede presentar valores entre $-2^3 = -8$ y $2^3 - 1 = 7$.

En la Fig. 4.23 se muestra el diagrama de bloques del nuevo píxel de convolución. Como se puede observar, además de los nuevos tamaños del sumador, del acumulador y del dato del kernel, hay una serie de diferencias sobre la versión anterior que hay que describir: el bloque de *switches* a la entrada del dato, la nueva selección del límite del acumulador con el circuito *"Logic ="* y el bloque de inhibición.

1. Bloque de *switches*. Este bloque se añadió para reducir el consumo de potencia innecesario en la versión inicial del píxel. Como se describió en el apartado anterior, los datos del kernel almacenados en la RAM se seleccionan por filas para sumarlos sobre una fila de píxeles, con el desplazamiento horizontal adecuado. Así, una vez que se activa la lectura de una fila de la RAM, cada dato aparece a la entrada del sumador de todos los píxeles de una misma columna. En la versión anterior del píxel, esto implicaba que todos los píxeles del array calculaban la suma correspondiente, aunque sólo los píxeles de la fila seleccionada recibían un pulso de *Enable*, por lo cual solamente éstos almacenaban el resultado de dicha suma. Este modo de operación producía un resultado correcto, aunque tenía el inconveniente de hacer que todos los píxeles efectuaran la suma de forma innecesaria, produciendo un consumo de potencia excesivo. Para solucionar este problema, en la versión avanzada del píxel se añadió

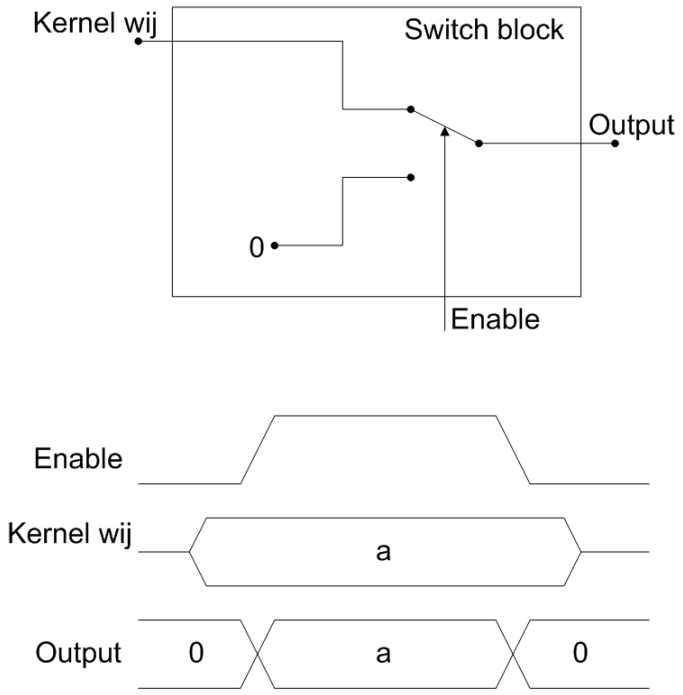


FIGURA 4.24. Esquema del bloque de switches y diagrama temporal de su funcionamiento.

el bloque de switches a la entrada. En la Fig. 4.24 se puede ver un esquema de la funcionalidad de este bloque, así como un diagrama temporal que muestra su comportamiento. De este modo conseguimos evitar que el píxel haga cualquier tipo de operación a menos que reciba un pulso de *Enable* del controlador, y además sin necesidad de haber añadido ninguna señal extra, ya que hemos utilizado la propia señal de *Enable*. Así pues, el flanco de subida de dicha señal permite el paso del dato, y el flanco de bajada hace que se almacene el resultado de la suma en el acumulador. Esto va a imponer una limitación sobre la máxima frecuencia de reloj posible, ya que la duración de la señal de *Enable* viene dada por la duración de un ciclo de reloj. De este modo, una frecuencia de reloj muy elevada provocará que el sumador no tenga tiempo de realizar la operación en la duración del pulso de *Enable*. En la Fig. 4.25 se puede ver el esquemático del bloque de switches a nivel de transistores.

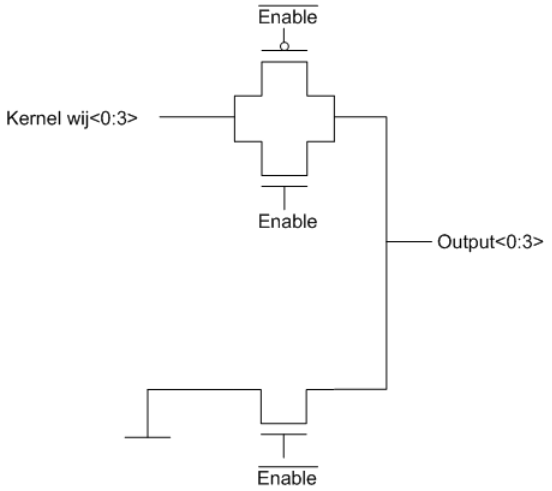


FIGURA 4.25. Esquemático del bloque de switches.

2. Selección del límite del acumulador. En la versión inicial existía la posibilidad de elegir entre 8 posibles umbrales, ya que teníamos un acumulador sobredimensionado de 18 bits, y así conseguíamos tener un mayor control sobre el píxel, y realizar pruebas con distintos tamaños. Sin embargo, con el acumulador actual de 6 bits no tiene sentido mantener tan alta programabilidad. Así pues, nuestro objetivo es mantener la posibilidad de escoger entre dos posibles umbrales, con lo cual además reducimos el tamaño del multiplexor, lo que se traduce en un ahorro de área (que es uno de nuestros grandes objetivos para este píxel). Sin embargo, siguiendo con la misma estrategia de comparar un bit del acumulador con su propio bit de signo no conseguíamos los valores que nos interesan. Es decir, en nuestro nuevo acumulador de 5 bits más el signo podríamos establecer un umbral máximo de $2^4 = 16$ y un umbral mínimo de $-2^4 - 1 = -17$, a pesar de poder representar valores entre -32 y 31. Esta limitación es debida a la forma de detectar el umbral a través de la observación de un único bit. Para poder aprovechar completamente la resolución que nos da nuestro acumulador sería necesario incluir un bloque comparador más complejo, en el que se observe el valor de todos los bits. Sin embargo, esto añadiría una complicación extra a nuestro píxel, lo que se traduce en un incremento de área, algo que queremos evitar.

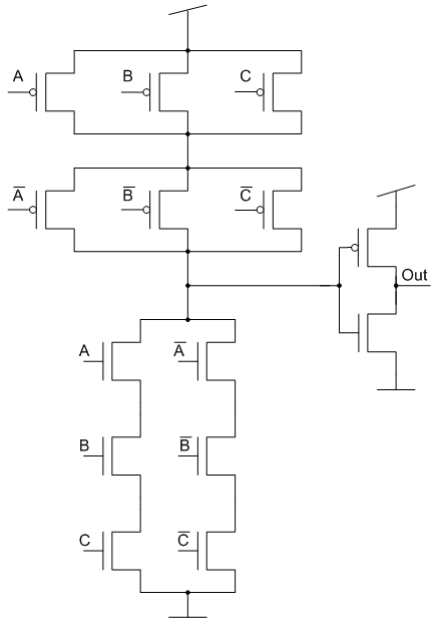


FIGURA 4.26. Puerta lógica que indica si 3 señales de entrada son iguales entre sí.

Por este motivo proponemos una solución intermedia, comparando solamente los dos bits más significativos de la magnitud con el bit de signo. De este modo, detectaremos el umbral para el valor $011000|_2 = 24|_{10}$ en los positivos, y $100111|_2 = -25|_{10}$. Esta operación la realiza la puerta lógica que podemos ver en la Fig. 4.26, ya que se trata de comparar el valor de estos dos bits con el bit de signo invertido. Por ese motivo, las entradas de este bloque A, B y C se conectan a los bits 3 y 4, y al bit de signo invertido, respectivamente. No obstante, incluimos en el píxel también un multiplexor de 2 entradas para poder seleccionar entre los umbrales de -25 y 24, o bien $-2^3 - 1 = -9$ y $2^3 = 8$, simplemente seleccionando el bit 3 del acumulador del mismo modo que lo hacíamos en la versión anterior del píxel. Así pues, los posibles umbrales quedan tal y como se muestra en la Tabla 4.2.

3. El bloque de inhibición es una nueva utilidad añadida a nivel de píxel para poder seleccionar el signo de los eventos que nos interesan en una

TABLA 4.2. Posibles límites programables para el acumulador.

Sel_lim	Maximum	Minimum
0	8	-9
1	24	-25

determinada aplicación. Un posible ejemplo sería una situación en la que el chip de convolución está programado para detectar una determinada forma geométrica en la imagen de entrada, para lo cual se utilizaría un kernel que produciría eventos positivos en el centro de donde se encuentre dicha forma y eventos negativos dispersos por el resto de la imagen. En este caso, los eventos negativos no aportan ninguna información a la siguiente capa de procesamiento. Con la versión anterior del chip de convolución, los eventos negativos se podían eliminar insertando un *mapper* externo [48] entre las capas que descartara los eventos de un determinado signo y dejara pasar los demás. Sin embargo, al añadir esta funcionalidad a nivel de píxel conseguimos en primer lugar eliminar un elemento externo, y sobre todo eliminar consumo de ancho de banda de comunicación AER innecesario, todo ello con un coste de área mínimo en el interior del píxel. Otra alternativa podría ser realizar este procesamiento en la periferia, pero así no evitaríamos la carga de ancho de banda introducida por la comunicación entre el píxel y la arbitración, provocando además retrasos en los eventos que sí nos interesan. Así pues, nuestra propuesta consiste en añadir un pequeño bloque interno al píxel de convolución que cuando se alcanza un límite del acumulador sólo deja pasar ese evento si el signo correspondiente no se encuentra seleccionado para su inhibición. Para ello, añadimos dos señales globales de configuración comunes para todos los píxeles que permiten seleccionar qué signo queremos inhibir (o bien inhibir los dos o ninguno). En la Fig. 4.27 se muestra el esquemático de la puerta lógica introducida para realizar dicho procesamiento. Como podemos ver, en función del estado de las señales *Inh_p*, *Inh_n* y *Sign* se permite el paso del pulso entre la entrada y la salida o bien se mantiene la salida fija a tierra. Si se produce un evento positivo (*Sign*=0), sólo habrá camino entre *Pulse_in* y *Pulse_out* si *Inh_p*=0 (es decir, si no hay inhibición de los eventos positivos). Para los eventos negativos, ocurre lo mismo, si tenemos *Sign*=1, sólo se permite el paso del pulso si *Inh_n*=0. En los demás casos, el

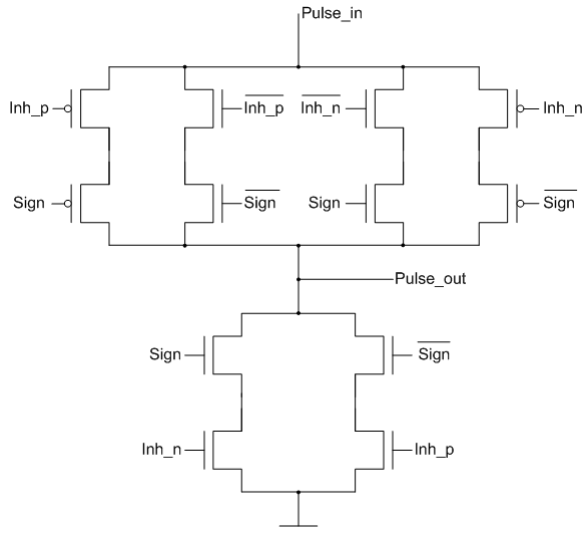


FIGURA 4.27. Puerta lógica utilizada para inhibir los pulsos según su signo.

pulso no se transmite a la interfaz AER para que ésta lo envíe a la periferia. Así pues, en el caso de que se produzca un evento cuyo signo está programado para su inhibición, será la señal *Pulse_in* la encargada de resetear el estado del acumulador para que el píxel pueda seguir funcionando con normalidad.

4.4.2.1. Estructuras sumadoras propuestas

Además de los cambios ya descritos en la nueva versión del píxel de convolución, a la hora de diseñarlo se volvió a analizar el modelo de sumador a utilizar, con el objetivo de minimizar el área del píxel. Para ello se estudiaron 2 propuestas:

1. Modelo de sumador de sólo 10 transistores [88], basado en un diseño optimizado de la función XOR [89] y en la lógica de transistores de paso [90]. Este tipo de lógica permite realizar funciones con un menor número de transistores, pero a cambio tiene el inconveniente de degradar las señales. El esquemático de esta celda podemos verlo en la Fig. 4.28. Debido a esta degradación de las señales, este modelo no sirve para nuestro píxel.

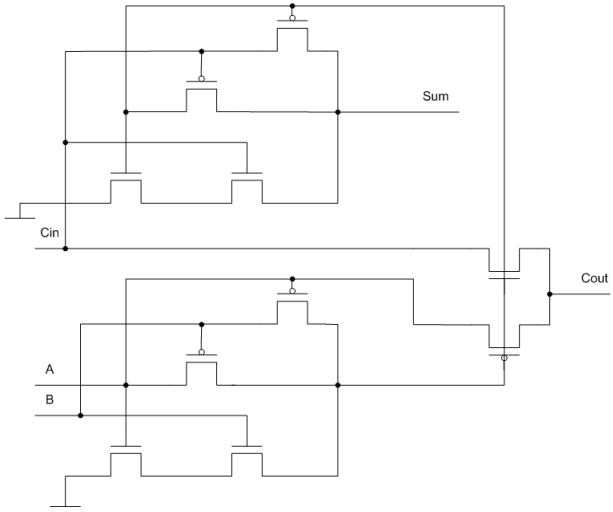


FIGURA 4.28. Esquemático de la celda sumadora de 10 transistores basada en lógica de transistores de paso.

2. Modelo de sumador de 16 transistores [91] basado en puertas de transmisión. En este caso, cuyo esquemático podemos ver en la Fig. 4.29, se incluyen un par de transistores dentro de la celda, los cuales se encargan de regenerar las señales que pasan por ellos, evitando la degradación. Así pues, utilizamos este modelo de sumador, por aportarnos la menor área posible a la vez que mantiene el correcto funcionamiento del píxel. Para escoger este modelo en lugar del anterior se hicieron exhaustivas simulaciones de esquinas y *mismatching*.

Con todo ello, el píxel obtenido tiene un área total de $58 \times 53.8 \mu m^2$. Con este píxel, compartiendo líneas de alimentación y configuraciones comunes, y conectándolo de forma especular con sus vecino, obtenemos un *cluster* de 2×2 píxeles de dimensiones $116 \times 107.6 \mu m^2$. Si comparamos el área de este *cluster* con el área de un sólo píxel de la versión inicial ($95.6 \times 101.3 \mu m^2$), vemos que hemos conseguido cuadruplicar la resolución espacial con un incremento de área muy reducido. Este resultado nos permite construir un array de 64×64 píxeles en un chip de prototipado pequeño de unos $5 \times 4 mm^2$. En la Fig. 4.30 podemos ver el layout de uno de estos *clusters* de 2×2 píxeles.

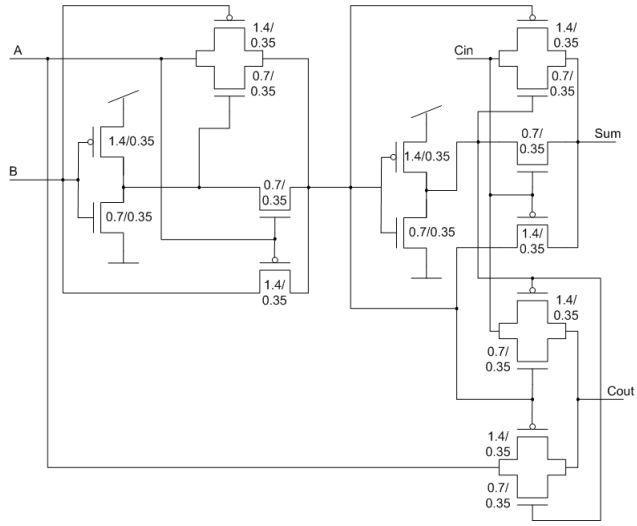


FIGURA 4.29. Esquemático de la celda sumadora de 16 transistores basada en puertas de transmisión utilizada en el diseño del píxel avanzado.

4.4.2.2. Resultados de simulación

Para comprobar el correcto funcionamiento de la nueva versión del píxel de convolución, realizamos algunas simulaciones similares a las presentadas para la versión anterior.

1. En un primer ejemplo, con la selección del umbral del acumulador establecida a 8, fijamos en primer lugar el dato del kernel a 1 y a continuación a 3, para observar que el sumador realiza la cuenta de forma correcta. En la Fig. 4.31 se muestran los resultados de estas simulaciones. En ambas gráficas se muestra en la parte superior la señal de entrada de *Enable*, indicando cada vez que el píxel recibe un pulso, y las señales de salida *Rqst_row* y *Pulse+* (en el centro y abajo, respectivamente). Además, encima de cada pulso de *Enable* aparece indicado con un número el valor almacenado en el acumulador después de procesar dicho pulso. Así vemos cómo en la gráfica de la izquierda el píxel genera un pulso de *Rqst_row* justo después de contar 8 eventos de valor 1, mientras

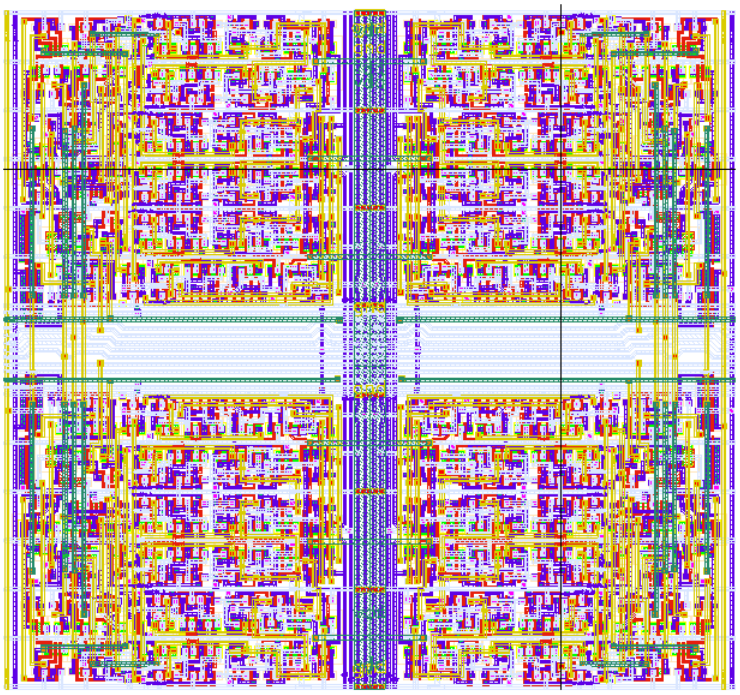


FIGURA 4.30. Layout de un cluster de 2x2 píxeles de convolución.

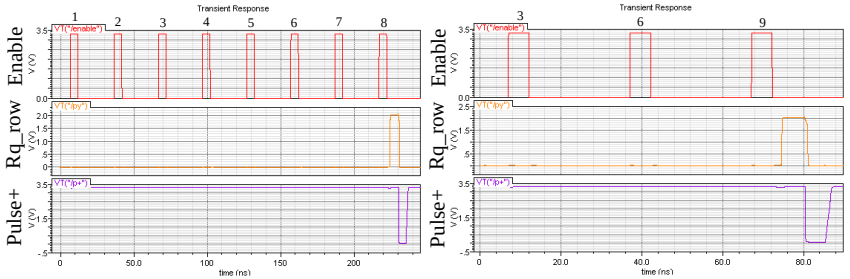


FIGURA 4.31. Resultados de simulación correspondientes al primer ejemplo, con el límite del acumulador establecido a 8, y el dato del kernel igual a 1 en la gráfica de la izquierda e igual a 3 en la gráfica de la derecha.

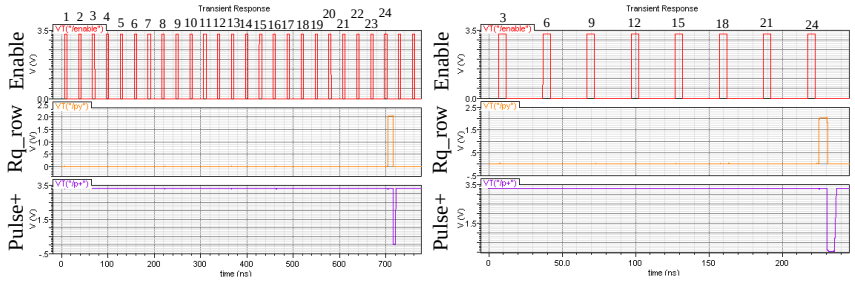


FIGURA 4.32. Resultados de simulación correspondientes al ejemplo 2, con el límite del acumulador en 24, y el dato del kernel igual a 1 en la gráfica de la izquierda e igual a 3 en la gráfica de la derecha.

que en la gráfica de la derecha genera el pulso de *Rqst_row* después de contar sólo 3 pulsos de valor 3.

2. En el ejemplo 2, seleccionamos el límite del acumulador 24, y aplicamos las mismas dos configuraciones del ejemplo anterior, es decir, en primer lugar le enviamos eventos con el dato del kernel con valor 1, y a continuación repetimos la operación con valor 3. En la Fig. 4.32 se muestran los resultados de esta prueba. En la gráfica de la izquierda vemos cómo el píxel necesita sumar el dato de valor 1 hasta 24 veces para generar un evento de salida, mientras que en la gráfica de la derecha vemos cómo es suficiente con sumar 8 veces el dato de valor 3 para alcanzar el umbral del acumulador.

3. En el ejemplo 3, planteamos dos pruebas diferentes. En la primera de ellas simplemente utilizamos un valor de kernel negativo, concretamente -1, para comprobar cómo el píxel alcanza el umbral negativo, en este caso -25. Este resultado se muestra en la gráfica de la izquierda de la Fig. 4.33, donde sobre los pulsos de la señal *Enable* se muestra el valor del acumulador tras cada evento, pero indicando sólo el valor absoluto para conseguir una mayor claridad en la figura (en realidad, todos los valores indicados son negativos, hasta alcanzar el límite de -25). En la otra prueba planteada, se vuelve a utilizar un valor de kernel igual a 1 con el límite del acumulador establecido a 8, pero en esta ocasión se alternan los pulsos de *Enable* con los pulsos de olvido (señal *Forgetting*). En la gráfica de la derecha de la Fig. 4.33 se puede ver el valor del acumulador tras cada pulso (tanto de *Enable* como de olvido), y se

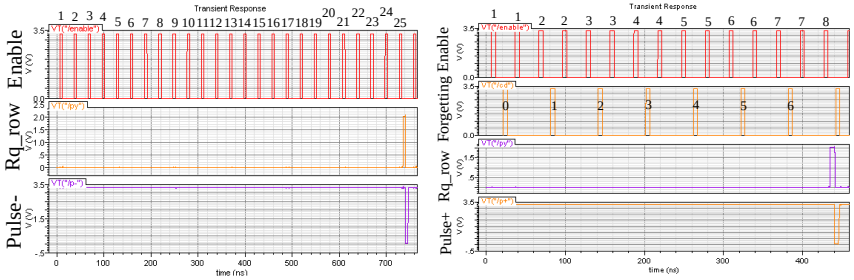


FIGURA 4.33. Resultados de simulación correspondientes al ejemplo 3. En la gráfica de la izquierda se utiliza un dato igual a -1 con un umbral del acumulador de -25, mientras que en la gráfica de la derecha se usa un dato igual a 1 con un umbral de 8, pero con el efecto del olvido activado.

observa cómo son necesarios hasta 15 eventos de entrada para alcanzar el valor umbral de 8, debido a los pulsos de olvido intercalados.

Con esto, damos por concluido el capítulo dedicado a la descripción del píxel de convolución. En el siguiente capítulo pasamos a describir el resto de bloques periféricos utilizados en los chips de convolución Conv1 y Conv2.

CAPÍTULO 5

Bloques periféricos en los chips de convolución

5.1. Introducción

La forma de llevar a cabo convoluciones bidimensionales mediante un sistema AER ha sido descrita en capítulos previos. Resumiendo, nuestro chip de convolución tiene que recibir eventos de entrada, y para cada uno de ellos ha de sumar el kernel (que previamente ha sido cargado en la RAM del kernel) sobre un vecindario de píxeles determinado por la dirección del evento de entrada. Además cuando cada uno de estos píxeles alcance un umbral, tiene que generar un evento de salida y transmitirlo al puerto AER de salida. La parte central de este procesamiento, el píxel de convolución, ya ha sido descrito en el capítulo anterior, y es el encargado de computar de forma local la operación de convolución, pero son necesarios una serie de bloques para gestionar el procesamiento completo.

En general, aparte del array de píxeles, podemos distinguir dos partes fundamentales en nuestros chips de convolución: la que se encarga de recibir los eventos de entrada y sumar el kernel sobre los píxeles, y la que se encarga de recibir los eventos generados por los píxeles y enviarlos al exterior. Todo ello se muestra en la Fig. 5.1.

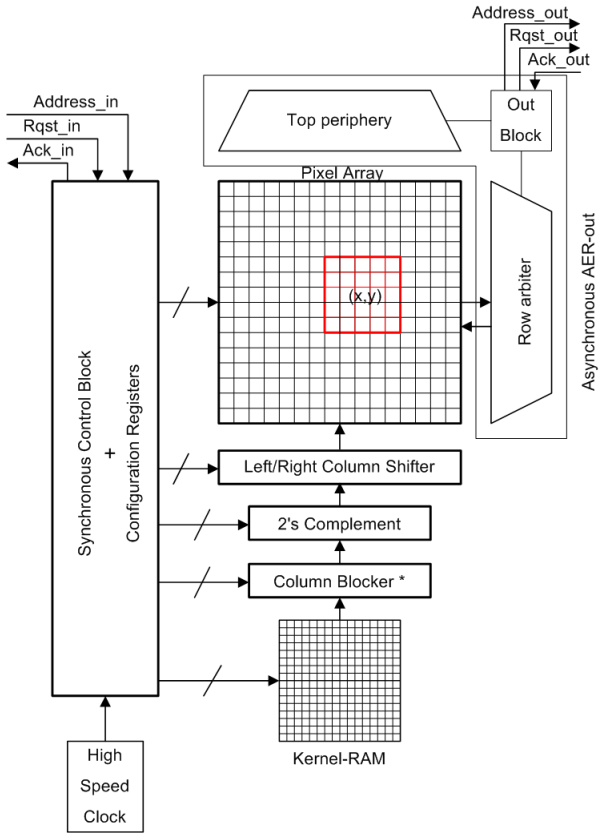


FIGURA 5.1. Arquitectura del chip de convolución (* Bloque incluido solamente en la versión Conv2)

La que podríamos llamar etapa de entrada consta de los siguientes bloques:

1. El controlador síncrono, que captura el evento de entrada y realiza los cálculos oportunos para decidir qué acciones tiene que llevar a cabo para procesar dicho evento. Además se encarga de enviar al resto de bloques las señales oportunas para llevar a cabo dicho procesamiento. Este bloque está detallado en la Sección 5.2.
2. La memoria RAM, donde se almacena el kernel de convolución antes de que el chip entre en modo de funcionamiento, y de donde se leen los

datos que tienen que ser sumados por los píxeles. Este bloque aparece descrito en la Sección 5.3.

3. El inversor de complemento a 2, que se encarga de invertir los datos del kernel antes de sumarlos en los píxeles en el caso de que el evento de entrada tenga signo negativo. En la Sección 5.4 se encuentra la descripción de este bloque.
4. El bloque de desplazamiento horizontal. Según la dirección del evento de entrada, el kernel almacenado en la RAM deberá sumarse sobre unos píxeles concretos dentro del array. Este bloque se encarga de desplazar horizontalmente el kernel de la RAM para que coincida justo con los píxeles deseados. Está descrito en la Sección 5.5.

Por otra parte, la etapa de salida es la que llamamos generador AER, y aparece detallada en la Sección 5.6.

En general, hay que recordar que en la presente tesis estamos describiendo dos versiones diferentes de chip de convolución, a las que nos referimos como Conv1 y Conv2, y ya en el Capítulo anterior describimos los píxeles de convolución correspondientes a las dos versiones. Del mismo modo, a lo largo del presente capítulo, para cada bloque iremos comentando las diferencias existentes entre ambas versiones de dicho bloque si las hubiera.

5.2. El controlador síncrono

El esquema del controlador síncrono podemos verlo en la Fig. 5.2. Aunque la parte fundamental del controlador es la máquina de estados, que es la encargada de realizar todo el procesamiento y habilitar las operaciones que tienen que realizar el resto de bloques, dicha máquina de estados necesita una serie de bloques para su funcionamiento. Si analizamos este diagrama de bloques desde el punto de vista de la entrada de eventos, en primer lugar nos encontramos un bloque de sincronizadores. Este bloque es necesario por la naturaleza asíncrona del protocolo AER, para procesar un evento asíncrono por una máquina de estados síncrona (esto se explicará más adelante, con resultados experimentales, en la Sección 5.2.4). A continuación, aparece la cola de entrada, que se utiliza para almacenar en ella los

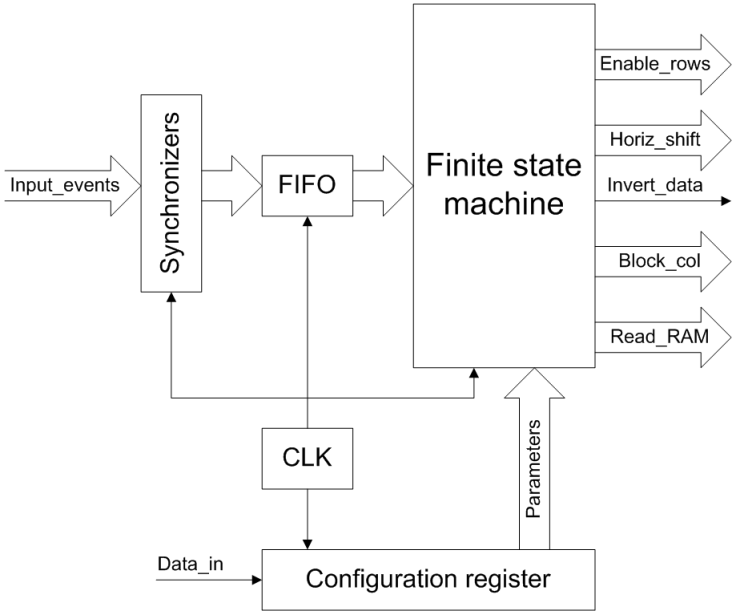


FIGURA 5.2. Diagrama de bloques del controlador síncrono.

eventos de entrada mientras esperan para ser procesados. Esto evita el riesgo de perder eventos que lleguen en un periodo de tiempo menor del necesario para procesarlos. A continuación, los eventos capturados por la cola son procesados por la máquina de estados, que se encarga de habilitar las señales oportunas para la lectura de las filas del kernel programadas en la RAM, invertir los datos si el evento que está procesando es negativo, indicar el desplazamiento horizontal necesario para que el kernel quede centrado sobre los píxeles adecuados, y por último generar los pulsos de *Enable* para que cada fila de píxeles sume los datos oportunos del kernel. También se encarga, en el caso de Conv2, de generar unas señales para bloquear columnas, como parte del sistema multikernel.

El sistema multikernel (utilidad incluida en Conv2) permite programar varios kernels diferentes (hasta un máximo de 32) en la memoria RAM, de forma que cada evento de entrada incluye, aparte de la dirección, información sobre qué kernel debe usarse para procesarlo. Así, el controlador tiene que activar las filas correspondientes de la RAM en función del kernel indi-

cado por cada evento, y también tiene que bloquear las columnas que no forman parte del kernel (que en general, formarán parte de otro kernel diferente) para que sólo se sumen las posiciones adecuadas sobre el array de píxeles. En las siguientes secciones se describirá cómo afecta esta utilidad a los diferentes bloques del chip.

Además de estos bloques, vemos en la Fig. 5.2 que también tenemos un generador de reloj, que permite que el controlador síncrono funcione sin necesidad de una señal de reloj externa (que también se le puede suministrar si lo deseamos). Y por último, aunque no necesariamente debe estar integrado dentro del controlador, representamos en la figura el registro de configuración. En este registro, antes de que nuestro chip entre en modo de operación, cargamos una serie de parámetros en serie con los cuales establecemos el modo de funcionamiento. Algunos de estos parámetros son utilizados por la máquina de estados para realizar su cometido (como la dirección base de los píxeles del array), y otros se usan para configurar los propios píxeles (como el valor del umbral del acumulador).

El controlador ha sido descrito mediante código VHDL [93] y sintetizado de forma automática, con la particularidad del registro de configuración, que en la versión de Conv1 se diseñó aparte del controlador mientras que en Conv2 se incluyó dentro del bloque global.

5.2.1. El generador de reloj de alta frecuencia

El generador de reloj utilizado es un oscilador en anillo de 5 inversores, tal y como se muestra en la Fig. 5.3. El realidad, el quinto inversor del anillo está integrado en la puerta NAND que se utiliza para habilitarlo o deshabilitarlo. Como podemos ver, hay un multiplexor que, mediante la señal *Sel_clk*, permite conmutar entre la señal de reloj generada internamente u otra introducida desde fuera del chip. La puerta NAND se encarga de que el anillo deje de oscilar cuando está seleccionado el reloj externo, para evitar consumo y ruido innecesario.

Además, como se indica en la figura, uno de los inversores del anillo tiene limitada la corriente máxima mediante una señal de control *Vbias_clk*, la cual es accesible desde el exterior del chip y nos permite programar la

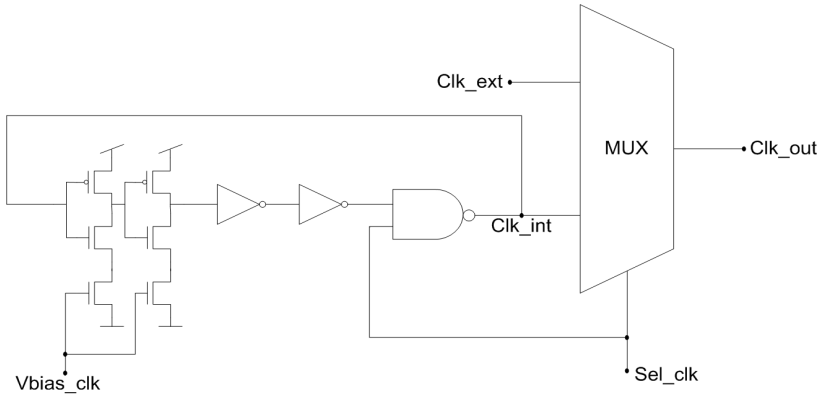


FIGURA 5.3. Esquema del generador de reloj.

frecuencia de reloj. La señal de reloj generada por este bloque (Clk_out) está conectada a un divisor de frecuencia que nos permite acceder a su salida desde fuera del chip. Gracias a eso, podemos medir externamente la frecuencia de reloj (dividida por 32, para facilitar su visualización en laboratorio) y ajustar la tensión de control para conseguir la frecuencia que queramos. En la Fig. 5.4 podemos ver la relación obtenida por simulación del extraído con Spectre entre esta tensión de control y la frecuencia de oscilación. El controlador puede trabajar con frecuencias de reloj de hasta 200MHz, aunque la frecuencia máxima de funcionamiento del chip viene limitada por otros bloques.

5.2.2. La cola de entrada

Al hablar de la cola de entrada nos referimos al bloque encargado de capturar los eventos (previamente sincronizados con el reloj del sistema) y pasárselos a la máquina de estados para que ésta realice las operaciones oportunas. Además, incluye una cola circular de 4 posiciones que actúa como un *buffer*, es decir, permite almacenar hasta 5 eventos mientras la máquina de estados está ocupada (los 4 de la cola más la posición de almacenamiento en el bloque *External Interface*). En realidad el objetivo de esta cola no es ser capaz de gestionar un alto promedio de tráfico de entrada, ya que la cola se saturaría al hacerlo la máquina de estados, sino que está diseñada para responder ante pequeños picos de tráfico puntuales. Si llegan

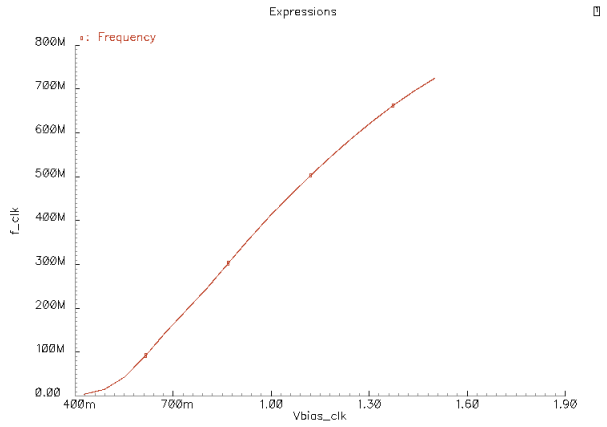


FIGURA 5.4. Caracterización del generador de reloj (frecuencia de oscilación frente a la tensión de control).

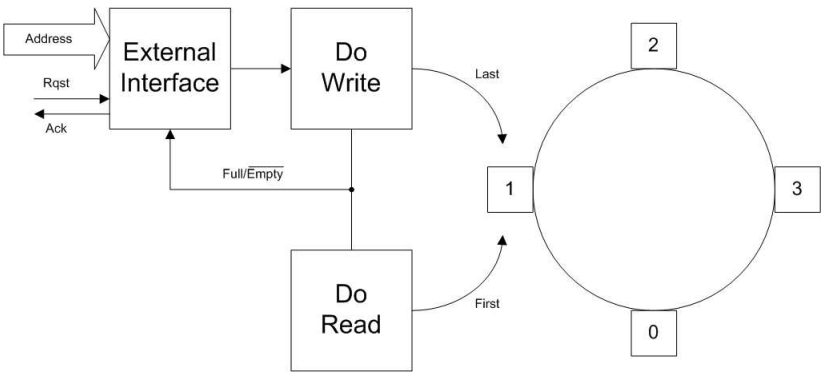


FIGURA 5.5. Diagrama que ilustra el funcionamiento de la cola de entrada.

eventos mientras todas las posiciones están ocupadas, el receptor detiene al emisor no respondiendo a la señal de *Rqst*.

El funcionamiento de la cola es el que se explica a continuación, y está ilustrado en la Fig. 5.5. En estado de reposo permanece mientras espera que se active la señal *Rqst*. Una vez que se activa dicha señal, comprueba si hay

alguna posición libre de la cola (mirando una señal interna *Full*/$\overline{Empty}$), y si la respuesta es afirmativa entonces almacena la dirección del evento y activa la señal *Ack*. Este proceso se lleva a cabo por el bloque llamado *External Interface*, que a su vez se encarga de enviar el evento almacenado al bloque llamado *Do Write*, que se encarga de la escritura en la cola. Así pues, a continuación este bloque se encarga de copiar la dirección del evento en la primera posición disponible de la cola, y de incrementar el valor del registro llamado *Last*, que apunta al último elemento de la cola. Entonces verifica si el valor de *(Last+1)MOD*4 coincide con el de *First MOD*4, lo cual significaría que la cola está completa, activando en ese caso la señal *Full*/$\overline{Empty}$.

En cuanto al proceso de lectura, cuando la máquina de estados esté en espera, comprobará si hay algún evento en la cola preguntándole al bloque *Do Read*, el cual se encarga de mirar los valores de *First*, *Last* y *Full*/$\overline{Empty}$. Si se da el caso de que *(Last+1)MOD*4 no coincide con *First MOD*4, o bien *Full*/$\overline{Empty}$ está activo, eso significa que hay al menos un evento en la cola esperando a ser procesado. En ese caso, envía a la máquina de estados el evento almacenado en la posición *First* de la cola, incrementando el valor de First y desactivando *Full*/$\overline{Empty}$ si estaba activo.

5.2.3. La máquina de estados

La máquina de estados es la responsable de llevar a cabo la operación de convolución paso por paso, habilitando la suma del kernel sobre los píxeles oportunos fila por fila. Para ello, necesita conocer las coordenadas del array de píxeles y del evento de entrada, así como el tamaño del kernel, para así poder calcular la posición exacta sobre la que aplicar dicho kernel. Puede ocurrir que el campo proyectivo correspondiente a un evento de entrada caiga completamente fuera del espacio de direcciones del chip, o bien que caiga dentro parcialmente. En función del caso concreto en el que se encuentre, el controlador efectuará unas operaciones u otras. El diagrama de estados se muestra en la Fig. 5.6, y lo describimos a continuación:

1. Estado de reposo. El controlador se encuentra esperando hasta la llegada de algún evento de entrada. Mientras la cola no active la señal de dato disponible, la máquina permanece indefinidamente en este estado. Una vez que detecta dicha señal, pasa al estado de cálculo. Este estado se

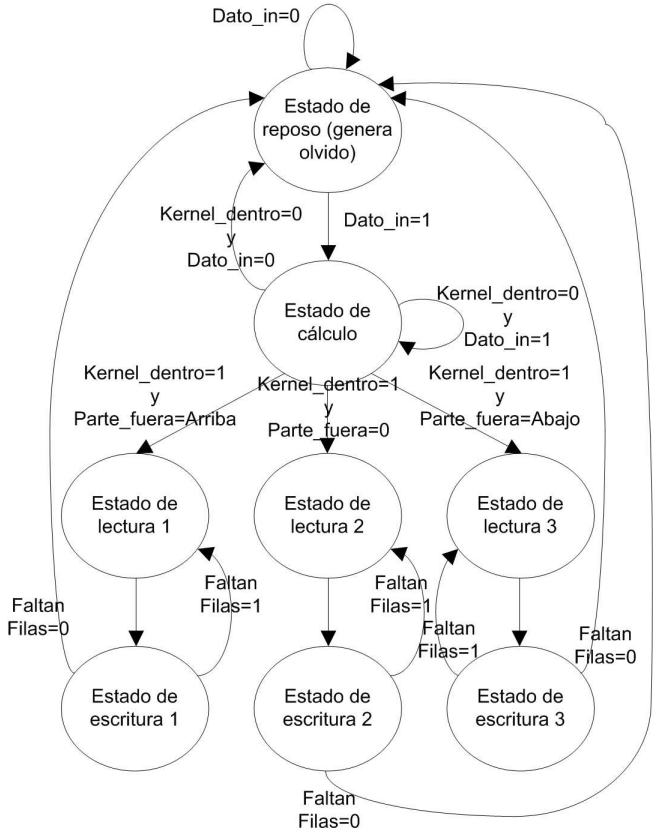


FIGURA 5.6. Diagrama de estados del controlador síncrono.

encarga también de la generación de la señal de olvido global de forma periódica. El controlador, aparte de esta máquina de estados, incluye también un contador programable de 20 bits que activa una señal de fin de cuenta cada vez que alcanza el límite, quedándose a la espera. Sin embargo, para evitar colisiones entre la señal de olvido y las señales *Enable*, es el estado de reposo de la máquina de estados el que comprueba el valor de la señal de fin de cuenta, generando entonces el pulso global de olvido y reiniciando el contador.

2. Estado de cálculo. Es en este estado donde la máquina tiene que realizar los cálculos y decidir las operaciones necesarias. En primer lugar, computa la posición horizontal del kernel dentro del array de píxeles, de

modo que si queda fuera del espacio de direcciones vuelve al estado de reposo. Si el campo proyectivo en sentido horizontal queda al menos parcialmente dentro del array, entonces calcula el desplazamiento horizontal que tiene que aplicar, activando las señales oportunas para ello. Entonces los límites verticales del campo proyectivo, y en función del resultado decide el próximo estado. Si el campo proyectivo queda parcialmente dentro del espacio de direcciones del chip, pero parte de él queda fuera por la parte superior, entonces pasa al estado de lectura 1. Si queda completamente dentro del array, pasa al estado de lectura 2, mientras que si parte del campo proyectivo queda fuera del chip por la parte inferior pasa al estado de lectura 3. Si todo el kernel queda fuera de los límites verticales del espacio de direcciones, entonces comprueba si hay algún nuevo dato disponible de entrada, en cuyo caso se mantiene en el estado de cálculo con el nuevo dato, mientras que en caso contrario vuelve al estado de reposo. En el caso del chip Conv2, el sistema multikernel hace que el controlador tenga que mirar el número del kernel con el cual tiene que procesar el evento recibido, y con ese número acceder a la información relativa a la posición del kernel en cuestión dentro de la RAM, y al vecindario donde debe ser aplicado. Las operaciones que realiza en este estado son completamente equivalentes al caso de Conv1, simplemente cambian los datos que utiliza para hacer los cálculos, que dejan de ser fijos para todos los eventos de entrada.

3. Estado de lectura 1. En este estado se habilita la lectura de la fila correspondiente de la RAM. Para cada evento de entrada, la máquina tendrá que pasar por este estado tantas veces como filas tenga el kernel, ya que se encarga de la lectura de una sola fila. Tras habilitar dicha lectura, pasa al estado de escritura 1.
4. Estado de escritura 1. Aquí se habilita la señal de *Enable* de la fila correspondiente del array de píxeles para que los datos que está leyendo de la RAM se sumen sobre dichos píxeles. Además, comprueba si aún quedan más filas del kernel por escribir, en cuyo caso vuelve al estado de lectura 1, mientras que si se trata de la última fila pasa de nuevo al estado de reposo, a esperar que llegue un nuevo dato.
5. Estado de lectura 2. Este estado es equivalente al estado de lectura 1, pero para el caso en el que todo el kernel queda dentro del campo de visión del chip en sentido vertical. Así pues, se encarga de habilitar la lectura para pasar a continuación al estado de escritura 2.

6. Estado de escritura 2. Igual que en el caso anterior, realiza las operaciones equivalentes al estado de escritura 1 cuando el kernel completo queda dentro del espacio de direcciones del chip. Según si quedan más filas por sumar o no, pasa al estado de lectura 2, o bien vuelve al estado de reposo.
7. Estado de lectura 3. Para la situación en que el kernel en sentido vertical está parcialmente dentro del array de píxeles, pero parte de él queda fuera por la parte inferior, este estado se encarga de habilitar la lectura de la fila correspondiente de la RAM y pasar al estado de escritura 3.
8. Estado de escritura 3. Del mismo modo que en los otros estados de escritura, habilita la suma de la fila del kernel en cuestión sobre los píxeles correspondientes. Vuelve al estado de lectura 3 si quedan más filas por escribir, o si no pasa al estado de reposo.

La máquina de estados es fundamentalmente igual en Conv1 y Conv2, salvo que en el primer caso los cálculos los realiza para un array de píxeles de tamaño 32×32 y en el segundo caso 64×64. Por otra parte, la principal diferencia entre ambos chips de convolución se debe al sistema multikernel incluido en Conv2. Ya que dicho sistema permite almacenar en la RAM varios kernels diferentes (hasta 32), hay que contemplar algunos pequeños cambios. En primer lugar, el evento de entrada tiene que incluir información acerca del kernel con el cual hay que procesarlo (además de los bits de dirección), de modo que en función de dicho dato, la máquina de estados tendrá que acceder a los datos de configuración donde se indica la posición que el kernel seleccionado ocupa dentro de la RAM. En el caso de Conv1, donde solamente se puede programar un kernel en la memoria RAM, el resto de posiciones de memoria que quedan fuera del kernel (suponiendo que no ocupe la RAM completa) se rellenan de ceros, de forma que no influyen en la operación de suma sobre los píxeles. Sin embargo, para Conv2 nos encontramos que las posiciones de la RAM fuera del kernel que estamos procesando para un evento determinado también pueden estar escritas con otros kernels diferentes. Así pues, para evitar que se produzcan operaciones indeseadas sobre los píxeles del array, la máquina de estados se encarga de habilitar solamente las columnas correspondientes al kernel deseado, sin que esto produzca grandes cambios sobre el controlador. Esto lo hace a través de la activación de las señales *Bloqueo_col* que aparecen en

la Fig. 5.2, que se encargan de bloquear todas las columnas de la RAM excepto aquellas donde se almacena el kernel.

En cuanto a la duración del procesamiento de un evento por el controlador síncrono, éste necesita 4 ciclos de reloj fijos para capturar el evento de entrada, ponerlo en la cola, sacarlo de la cola y comprobar si queda dentro del espacio de direcciones del chip. A continuación, necesita 2 ciclos de reloj por cada fila del kernel que tenga que procesar (1 ciclo para lectura y otro para escritura). De este modo, el tiempo que necesita para procesar un evento se puede expresar como $(4 + 2 \times n_k) \times T_{clk}$, donde n_k indica el número de filas del kernel y T_{clk} el periodo de la señal de reloj. Así, para una frecuencia de reloj máxima de 200MHz se puede conseguir un tiempo entre eventos de entrada desde los $30ns$ para un kernel de tamaño 1×1 hasta los $340ns$ para un kernel de tamaño 32×32.

Por otra parte, el controlador síncrono se encarga también de gestionar el mecanismo de olvido en paralelo al propio funcionamiento de la máquina de estados. Para ello, dispone de un contador de 20 bits configurable, que cuando alcanza un determinado valor genera una señal global de olvido que llega a todos los píxeles del array decrementando su estado en una unidad si su estado es positivo, o incrementándolo si es negativo. El periodo de olvido (tiempo que transcurre entre dos pulsos de olvido) se configura con una palabra de 20 bits que nos permite seleccionar el número de ciclos de reloj n_{olv}. De este modo, el periodo de olvido será $n_{olv} \times T_{clk}$, siendo el máximo valor posible $(2^{20} - 1) \times T_{clk} = 1048575 \times T_{clk}$.

5.2.4. Los sincronizadores

Al tratar con un protocolo asíncrono como AER, los chips de convolución tienen que manejar eventos asíncronos mediante un controlador síncrono. Esto puede provocar errores debidos a la metaestabilidad [92], produciendo resultados no deseados. Un error de sincronización se produce cuando la señal de reloj encargada de almacenar el dato cambia durante un transitorio de la señal de entrada. Esta metaestabilidad puede producir errores en la computación y eventos espurios a la salida. Llamemos Probabilidad de Error a la tasa entre el número de eventos espurios producidos a la salida y el número de total de eventos. Si no introducimos ningún método

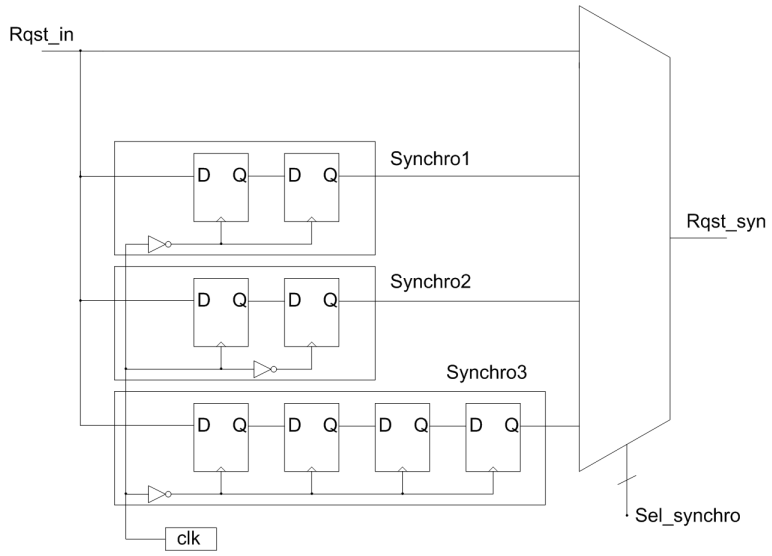


FIGURA 5.7. Bloque configurable de sincronizadores a la entrada del chip de convolución.

de sincronización, en nuestros chips de convolución medimos una probabilidad de error de 4.62%, la cual es extremadamente alta. Para conseguir una probabilidad de error mucho menor incluimos sincronizadores. Estos bloques se insertan en la línea *Rqst_in*, tal y como se muestra en la Fig. 5.7. Este bloque configurable nos permite caracterizar las diferentes implementaciones de sincronizadores y comparar su efecto respecto a la ausencia de sincronizador. En general, la probabilidad de error es inversa y exponencialmente proporcional al número de biestables incluidos. Sin embargo, cada biestable añade un retraso extra sobre el procesamiento del evento, aunque sin influir en la tasa de entrada de eventos.

Para caracterizar estos circuitos, se llevaron a cabo una serie de medidas cuyos resultados se muestran en la Tabla 5.1. Para medir la probabilidad de error, se envío un tren de eventos a la entrada mientras el chip de convolución estaba programado para generar un evento de salida por cada uno de entrada. De este modo, se detecta un error cuando aparece un evento de salida no esperado. Como podemos ver en la tabla, en ausencia de sincronizadores tenemos una probabilidad de error de 4.62%, mientras que el retraso medio medido entre las señales *Rqst_in* y *Ack_in* es de 0.89 ciclos de

TABLA 5.1. Caracterización de los sincronizadores

Sincronizador	Probabilidad de error (en %)	Retraso medio (en ciclos de reloj)
Ninguno	4.62	0.89
Synchro1	21×10^{-6}	2.23
Synchro2	44×10^{-6}	1.84
Synchro3	4×10^{-10}	4.23

reloj (este retraso indica el tiempo que tarda el sistema en capturar el dato del evento, y se debe a la estadística de la señal asíncrona de entrada respecto de la señal síncrona de reloj). La configuración llamada *Synchro1* inserta dos biestables en cascada en la línea *Rqst_in*, disparados ambos por el propio reloj del controlador. Esta configuración reduce la probabilidad de error hasta un 21×10^{-6} %, a la vez que añade un retraso medio de $2.23 - 0.89 = 1.34$ ciclos de reloj en el tiempo de respuesta. La configuración llamada *Synchro2* es una pequeña modificación de la anterior en la cual ambos biestables son disparados en distintas fases de reloj. De esta forma, la probabilidad de error se degrada ligeramente hasta un 44×10^{-6} %, aunque el retraso medio se reduce a $1.84 - 0.89 = 0.95$ ciclos de reloj. La última configuración propuesta, *Synchro3*, utiliza 4 biestables en cascada disparados por la misma señal de reloj. Esta configuración introduce un mayor retraso de $4.23 - 0.89 = 3.34$ ciclos de reloj. Para este caso, no ha sido posible detectar ningún error después de estar observando durante varios días. En consecuencia, hemos estimado la probabilidad de error usando la expresión del MTBF (tiempo medio entre fallos, en sus siglas en inglés) [92], que es la inversa de la probabilidad de error. De este cálculo resulta una probabilidad de error de 4×10^{-10} %, lo que significa que enviando eventos AER de entrada a una tasa de 1Meps (10^6 eventos por segundo) necesitaríamos del orden de 70 horas para detectar un único espurio a la salida.

En nuestro chip de convolución, se puede seleccionar cualquiera de los tres sincronizadores propuestos, o bien no usar ninguno, mediante la configuración de 2 bits de control.

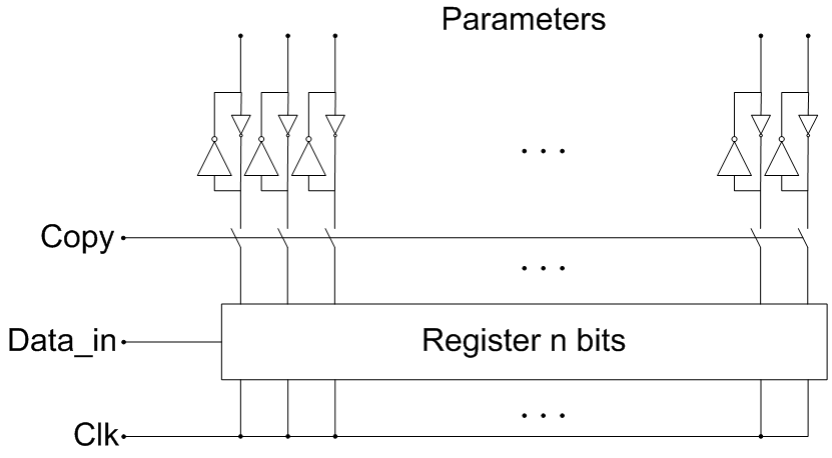


FIGURA 5.8. Estructura del registro de configuración utilizado en el chip de convolución Conv1.

5.2.5. Los registros de configuración

Como ya se ha comentado anteriormente, nuestros chips de convolución necesitan conocer el valor de una serie de parámetros de configuración con los cuales podemos controlar su funcionamiento. Para ello, utilizamos los registros de configuración. En primer lugar, vamos a describir el bloque utilizado en la primera versión Conv1.

En la Fig. 5.8 podemos ver la estructura del registro de configuración utilizada en esta primera versión del chip de convolución. Como se muestra en la figura, se trata de un registro de desplazamiento de 128 bits, los cuales se introducen en serie, y una vez que todos los bits están escritos se activa una señal de copia para almacenar sus valores y conectarlos a los bloques del circuito que los necesiten. La lista de los parámetros de configuración aparece en la Tabla 5.2. Tanto los parámetros que indican las coordenadas del chip de convolución como las dimensiones del kernel dentro de la RAM son utilizados por la máquina de estados para calcular las operaciones que tiene que llevar a cabo en función de la dirección del evento de entrada. El parámetro n_{olv} también lo utiliza el controlador, en este caso para generar

TABLA 5.2. Lista de parámetros de configuración

Parámetro	Número de bits	Significado
$(x, y)_{min}$	14	Coordenada del píxel superior izquierdo dentro del espacio de entrada de 128x128
$(x, y)_{max}$	14	Coordenada del píxel inferior derecho dentro del espacio de entrada de 128x128
(p, q)	10	Dimensiones del kernel dentro de la RAM
n_{olv}	20	Ciclos de reloj entre dos pulsos de olvido globales
sel_{clk}	1	Selección del reloj interno o externo
sel_{acc}	3	Selección del límite del acumulador
sel_{pd}	64	Configuración de los transistores *pull-down* en la arbitración por filas
sel_{syn}	2	Selección de sincronizador

los pulsos de olvido. Tanto los bits de selección de sincronizador como el de selección de reloj se usan para controlar sus bloques correspondientes. En cuanto a los 64 bits de configuración de los *pull-downs* de la arbitración por filas, su funcionamiento aparece descrito en la Sección 5.6, al describir el generador AER.

En cuanto a la otra versión de chip de convolución Conv2, la etapa de configuración presenta una serie de novedades, fundamentalmente debidas al sistema multikernel. Como ya se ha comentado anteriormente, este sistema se basa en la posibilidad de programar hasta 32 kernels diferentes en la RAM, así que para que la máquina de estados pueda acceder al kernel correspondiente en función del evento de entrada necesita información acerca de la posición de cada uno de ellos. Por ese motivo se ha diseñado una tabla de memoria formada por 34 filas de 32 bits cada una. En cada fila de las 32 primeras se almacena la información referente a cada kernel, mientras que en las dos últimas filas se almacenan el resto de parámetros del sistema.

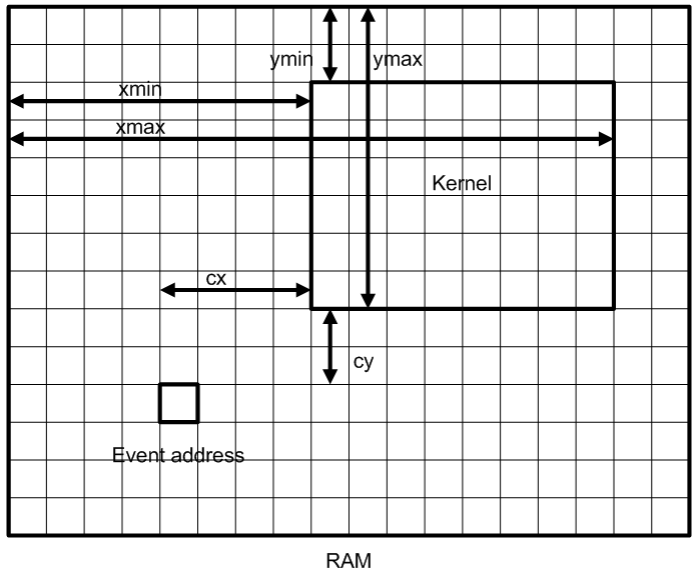


FIGURA 5.9. Definición de los parámetros relativos a cada kernel.

Para describir la información referente a cada kernel se necesitan 6 parámetros, los cuales aparecen representados en la Fig. 5.9. Como podemos ver, 4 de estos parámetros indican la posición del kernel dentro de la memoria RAM a través de las coordenadas de la esquina superior izquierda (x_{min}, y_{min}) y de la esquina inferior derecha (x_{max}, y_{max}). Cada uno de estos 4 parámetros tiene 5 bits. En cuanto a los dos parámetros restantes, indican las coordenadas del centro de aplicación del kernel. En la versión anterior Conv1 este parámetro no existía, ya que se consideraba que todos los kernels se aplicaban sobre su propio centro. Sin embargo, para algunas aplicaciones se ha comprobado que no siempre tiene que ser así. Por ejemplo, en las aplicaciones de reconocimiento de letras se usan convoluciones que tratan de detectar características de un carácter que se encuentran en un extremo del mismo, luego el kernel se tiene que aplicar desplazado sobre su propio centro [74]. Por este motivo se incluyen dos parámetros (c_x, c_y) que indican la posición relativa a la dirección del evento que debe ocupar el kernel. Estos parámetros pueden ser positivos o negativos, ya que esta posición puede suponer un desplazamiento hacia la derecha o hacia la izquierda, y

hacia arriba o hacia abajo. Por ello, se codifican en complemento a 2 con 6 bits (5 más el signo), conformando un total de 32 bits para definir las características de cada kernel, que es la longitud de una fila de la tabla de memoria.

Así, cada vez que llega un evento de entrada, en función del número de kernel que indique se selecciona la fila correspondiente de esta tabla de memoria, de forma que la máquina de estados solamente verá esa fila, a partir de la cual obtiene los datos necesarios para calcular el desplazamiento horizontal que debe aplicar para centrar el kernel sobre el array de píxeles, así como las posiciones de la RAM que tiene que leer. También utiliza estos parámetros para activar las señales *Block_col* para todos los valores menores que x_{min} y mayores que x_{max}.

En las dos filas restantes de la tabla de memoria se almacena la información relativa a la configuración del chip de convolución. En la fila 33 se incluyen las coordenadas del espacio de direcciones del chip (i_{min}, j_{min}), (i_{max}, j_{max}), 4 parámetros de 8 bits cada uno. La última fila, la 34, almacena 1 bit de selección para el límite del acumulador de los píxeles, 2 bits para la selección de inhibición de eventos positivos o negativos, 1 bit para la activación del mecanismo de olvido y 20 bits para establecer la frecuencia de dicho mecanismo.

5.3. La memoria RAM estática

La memoria RAM es el bloque que se encarga de almacenar el valor del kernel (ya sea un único kernel en Conv1 o múltiples de ellos en Conv2), así que también será el que limite el tamaño del kernel de convolución. En el caso de Conv1 podemos utilizar kernels de un tamaño máximo de 32×32, mientras que en el caso de Conv2 tendremos la misma limitación si programamos un único kernel, mientras que cuando programemos varios de ellos será obligatorio que estos sean más pequeños. En definitiva, en ambos casos la memoria RAM cuenta con 32×32 posiciones. La diferencia entre ambos chips es que, como se describió en el capítulo anterior, Conv1 utiliza datos de 6 bits mientras que Conv2 los usa de 4 bits. Por lo demás, desde un punto de vista de estructura, ambas memorias son iguales.

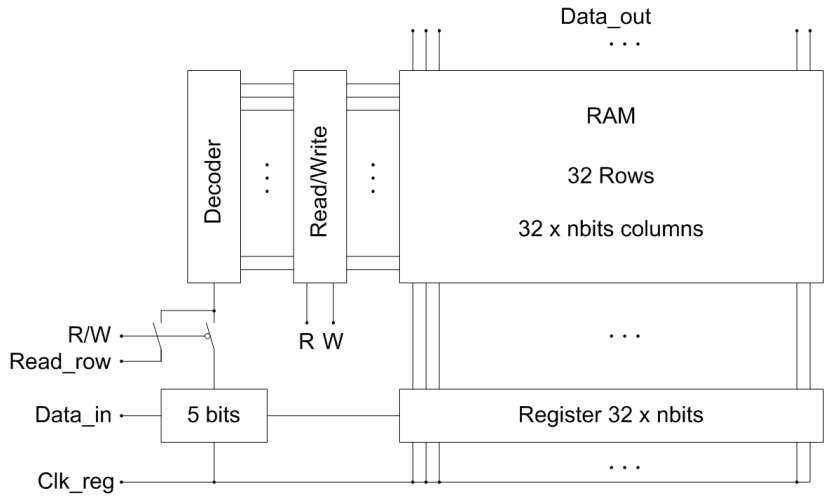


FIGURA 5.10. Esquema del bloque completo de memoria RAM.

En la Fig. 5.10 podemos ver el esquema del bloque completo de memoria RAM, incluyendo la circuitería de escritura y lectura. En la parte inferior vemos el registro de desplazamiento de $32 \times n$ bits, siendo n el número de bits de cada dato (6 para Conv1 y 4 para Conv2). Este registro de desplazamiento en realidad cuenta con 5 bits más que se usan para indicar la fila correspondiente. De este modo, cuando queremos escribir la RAM desde fuera del chip, enviamos cadenas de $(32 \times n) + 5$ bits, y el decodificador se encarga de activar la fila correspondiente y almacenar los datos en ella. El proceso de lectura es muy parecido, salvo que los 5 bits que seleccionan la fila no los toma del registro de desplazamiento, sino del propio controlador, que activa el modo de lectura y le indica la fila correspondiente. De este modo, el decodificador habilita la lectura de la fila correspondiente.

Para implementar el sistema multikernel, en el chip Conv2 se añade un bloque extra a la salida de los datos de la RAM, ya que tenemos que asegurarnos de que todas las posiciones de la memoria que no se corresponden con el kernel que estemos procesando en cada momento no afecten al array de píxeles. Para ello la máquina de estados genera una señal de *Block_col*

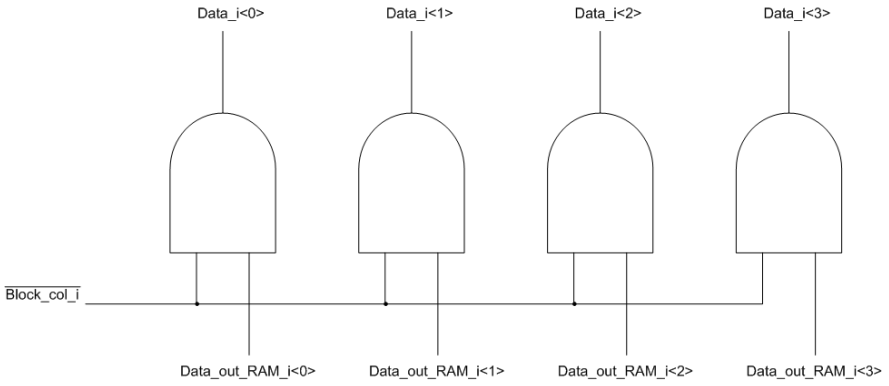


FIGURA 5.11. Circuito encargado de bloquear las columnas de la RAM para implementar el sistema multikernel.

para cada una de las 32 posiciones de la RAM (por columnas) que se encarga de bloquear todas las palabras que no pertenecen al kernel. El circuito que se encarga de implementar este bloqueo aparece representado en la Fig. 5.11. En esta figura se muestra el subcircuito correspondiente a cada uno de los 32 datos de la RAM (formados por 4 bits cada uno en el caso de Conv2). Así pues, el circuito completo incluirá 32 celdas como la de la figura. El circuito recibe del controlador 32 señales de $\overline{Block\_col\_i}$, las cuales valdrán 1 para $x_{min} \leq i \leq x_{max}$ y 0 para los valores restantes (x_{min} y x_{max} se corresponden con la posición del kernel en la RAM según se indica en la Fig. 5.9). Así las señales *Data_i<0:3>* valdrán 0 para todas las columnas que no pertenezcan al kernel.

En la Fig. 5.12 podemos ver la estructura de una celda básica de la memoria RAM, diseñada de forma que tiene un tiempo de lectura inferior a *2ns*, ya que necesitamos que el tiempo de escritura del kernel sea lo más rápido posible. Cualquier retraso extra que añadiera la RAM incidiría en el tiempo que se tarda en procesar cada fila, y como consecuencia limitaría la máxima frecuencia de reloj que puede ser programada. La Fig. 5.13 muestra el layout de una celda básica como la de la Fig. 5.12.

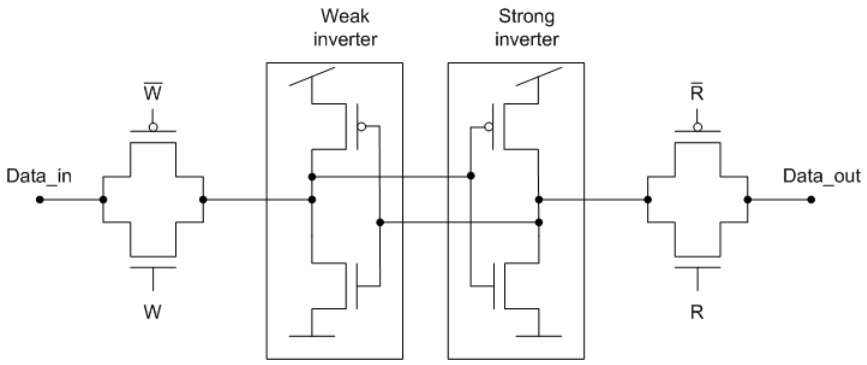


FIGURA 5.12. Celda básica de la memoria RAM.

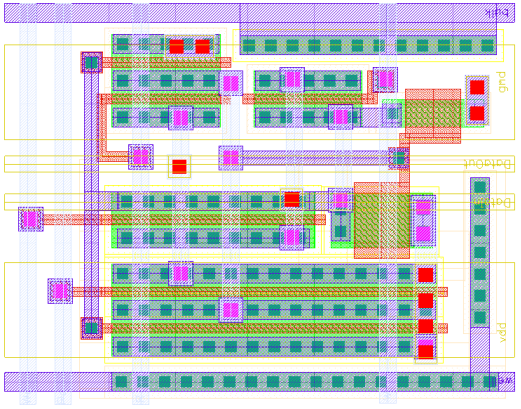


FIGURA 5.13. Layout de una celda de memoria RAM.

5.4. El inversor de complemento a 2

Como ya se explicó al describir el funcionamiento global del chip de convolución, éste recibe eventos AER con su correspondiente bit de signo, al igual que cada evento de salida debe ser generado también con su propio signo indicando si el píxel en cuestión ha saturado por el límite positivo o por el negativo. Así pues, cuando un chip recibe un evento de entrada nega-

tivo tiene que procesarlo como tal, es decir, sumando el kernel sobre el vecindario correspondiente pero cambiado de signo. Por este motivo necesitamos un bloque que invierta el signo del kernel.

Para desempeñar esta función se contemplaron 2 alternativas. La primera de ellas consistía en almacenar en nuestra memoria RAM el kernel natural y con el signo invertido. Esto simplificaría la operación del chip, ya que simplemente tendría que acceder a una zona diferente de la memoria en función del signo del evento. Sin embargo, la desventaja que tenía es que limitaba el tamaño del kernel, ya que éste necesitaba el doble de su tamaño para almacenarlo. La segunda alternativa consistía en añadir un bloque que calculara el inverso de cada dato de la RAM en el momento de ser leído. Esta alternativa nos permite mantener el tamaño máximo del kernel, y solamente tiene la pequeña desventaja de añadir un ligero retraso sobre el proceso de lectura, así que hemos diseñado el bloque inversor para minimizar dicho retraso, que en la práctica resulta despreciable dentro del proceso de lectura. El retraso añadido por el bloque inversor ha sido estimado en unos $400ps$ mediante simulación con Spectre, mientras que el coste de área adicional consumida por este circuito es de unas $3750 \times 40\mu m^2$ (siendo el área consumida por la RAM de $3750 \times 480\mu m^2$ mucho mayor).

De este modo, se necesitan 32 bloques, uno para cada posición de la RAM. Para Conv1, la estructura del bloque propuesto consta de 6 circuitos combinacionales, uno por cada salida. Para Conv2 sólo necesitamos 4 circuitos. Por este motivo vamos a describir el utilizado en Conv1. La entrada a cada bloque se trata de una palabra digital de 6 bits $(i_0, \ldots, i_5)$ siendo i_5 el bit más significativo, el que indica el signo. Además, recibirán una entrada *sel* que indica que se habilita la operación de inversión, y se obtiene otra palabra digital de 6 bits de salida $(o_0, \ldots, o_5)$ donde o_5 es el bit más significativo. A partir de las tablas de verdad que expresan los resultados para todas las posibles combinaciones, las expresiones lógicas obtenidas para las 6 señales de salida se pueden ver en las ecuaciones (5.1)-(5.6).

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

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

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

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

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

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

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

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

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

A partir de estas expresiones se obtienen las puertas lógicas correspondientes. En las figuras Fig. 5.14-Fig. 5.18 aparecen representados los esquemáticos resultantes (en la parte izquierda) y los *layouts* (en la parte derecha) de las celdas que calculan los bits o_1-o_5 (teniendo en cuenta la ecuación (5.1), no hace falta una puerta lógica para calcular o_0).

5.5. El bloque de desplazamiento horizontal

Una vez que los datos del kernel son leído de la RAM, antes de llegar a los píxeles correspondientes del array tienen que desplazarse hacia la izquierda o hacia la derecha para centrarse sobre las direcciones adecuadas. Esta operación la lleva a cabo el bloque de desplazamiento horizontal.

Este bloque consiste en un array bidimensional de *buffers* tri-estado controlados por las señales de desplazamiento Δx y *right/left*. Estas señales son generadas por la máquina de estados, que a través de dos decodificadores incluidos en el propio controlador habilita la señal de desplazamiento apropiada. En la Fig. 5.19 podemos ver la estructura de este bloque. Como

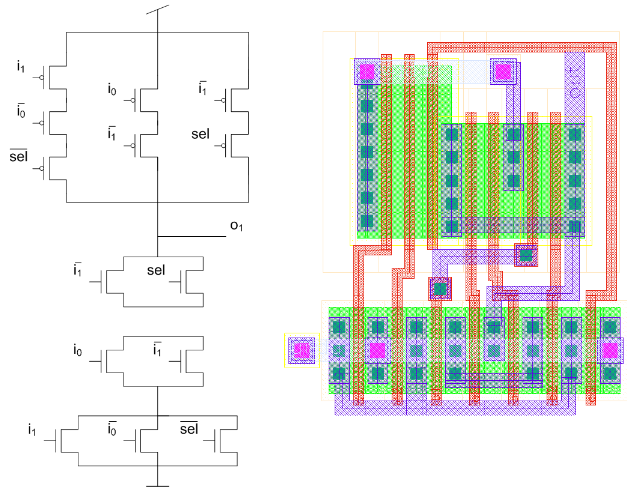


FIGURA 5.14. Esquemático y layout del circuito combinacional que calcula el bit o_1 en el bloque inversor de complemento a 2.

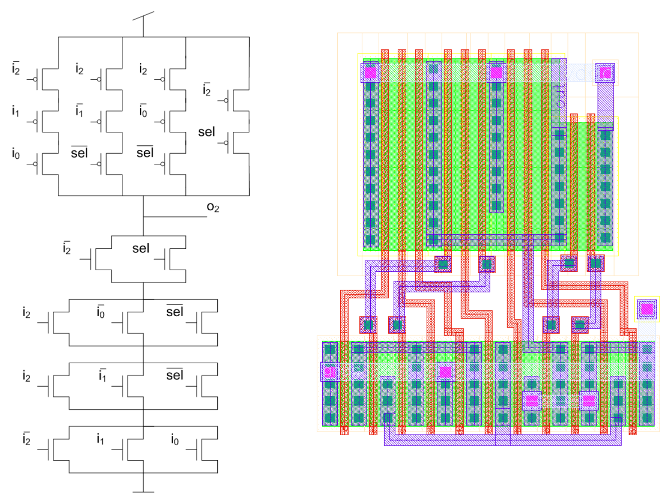


FIGURA 5.15. Esquemático y layout del circuito combinacional que calcula el bit o_2 en el bloque inversor de complemento a 2.

ya se ha descrito al hablar de la máquina de estados, cada vez que ésta está procesando un evento realiza los cálculos oportunos entre la dirección de

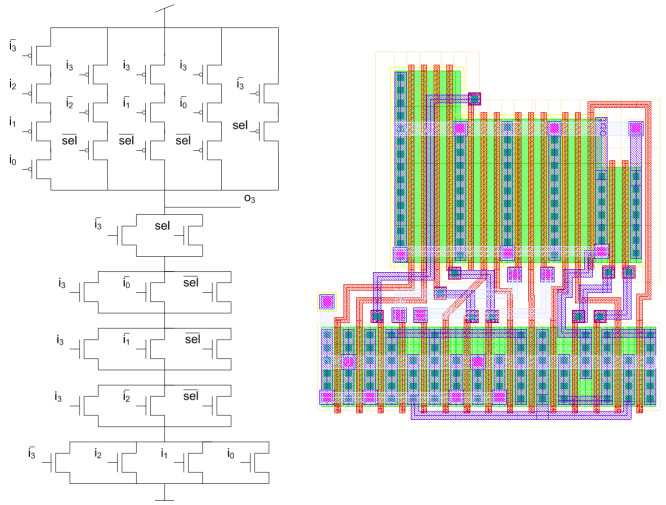


FIGURA 5.16. Esquemático y layout del circuito combinacional que calcula el bit o_3 en el bloque inversor de complemento a 2.

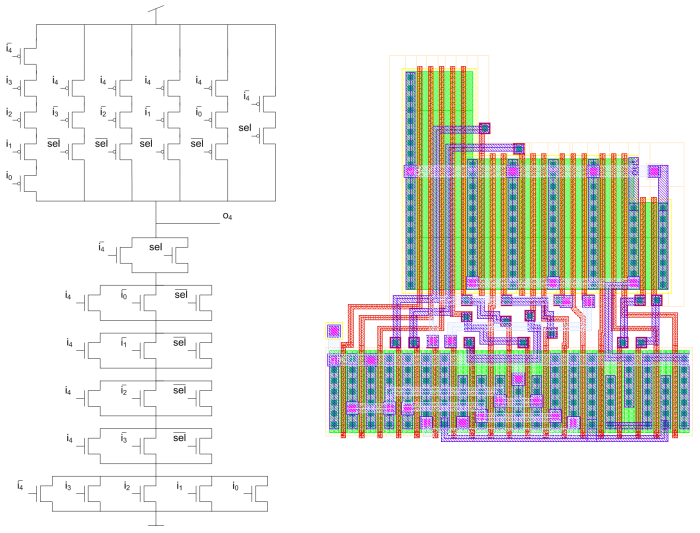


FIGURA 5.17. Esquemático y layout del circuito combinacional que calcula el bit o_4 en el bloque inversor de complemento a 2.

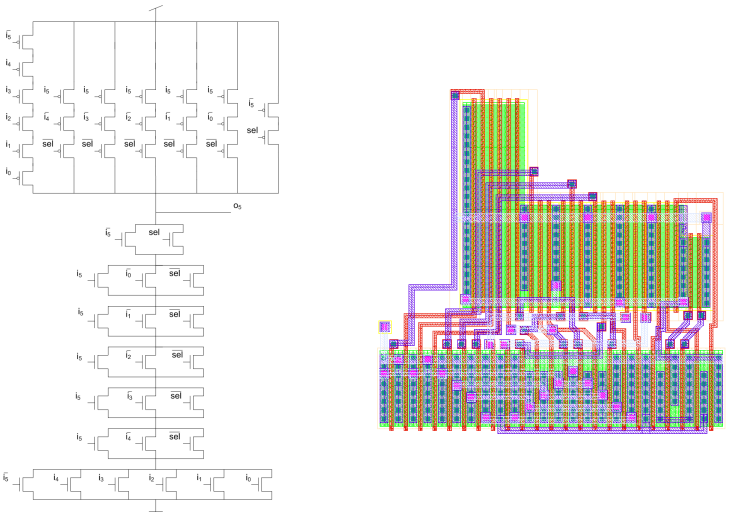


FIGURA 5.18. Esquemático y layout del circuito combinacional que calcula el bit o_5 en el bloque inversor de complemento a 2.

dicho evento de entrada y las coordenadas de aplicación del kernel, para así obtener la cantidad de posiciones que tiene que desplazar el kernel (Δx) para que la operación de convolución sea correcta. De este modo, a través de la activación de las señales correspondientes, el bloque de desplazamiento horizontal habilita un camino entre las columnas de la RAM y las columnas del array de píxeles.

Cada elemento de este array bidimensional consta de dos *buffers* triestado controlados cada uno por una señal de desplazamiento, y cuyas entradas son datos de la RAM que provienen de las posiciones a su izquierda y a su derecha correspondientes a la cantidad del desplazamiento. De este modo, las entradas de este bloque son las salidas de la RAM (después de atravesar el inversor de complemento a 2) y sus salidas son los mismos datos leídos de la RAM aunque trasladados horizontalmente para coincidir con los píxeles del array sobre los cuales tienen que sumarse. Las salidas de este bloque cuentan con transistores *pull-down* para forzar los datos a 0 en las columnas que no están seleccionadas.

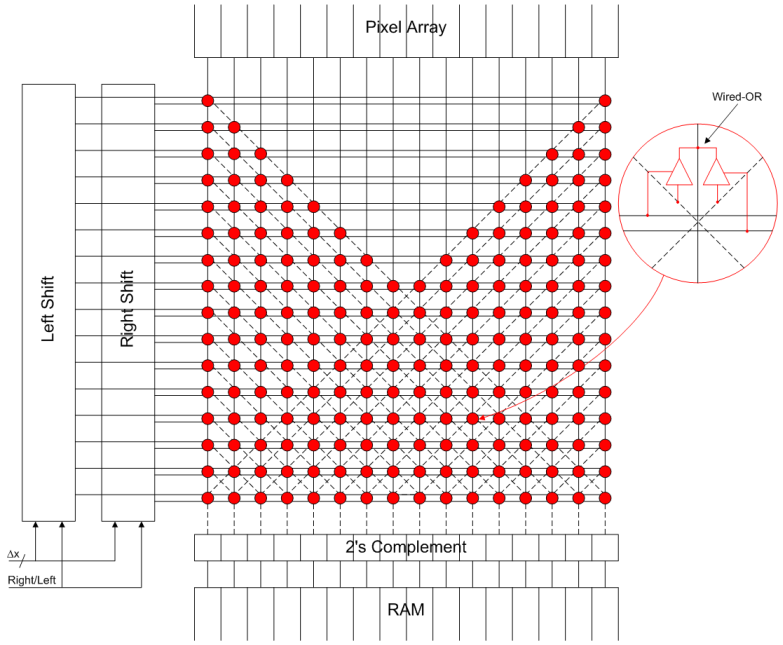


FIGURA 5.19. Esquema del bloque de desplazamiento horizontal.

La única diferencia que presenta este bloque entre las dos versiones de chip Conv1 y Conv2 es su tamaño. Aunque en ambos casos la entrada del circuito está formada por 32 posiciones de la memoria RAM, para Conv1 cada una de estas posiciones consta de 6 bits mientras que para Conv2 son solamente 4. Por otra parte, la salida del bloque para Conv1 también tiene 32 posiciones (de 6 bits) que se conectan al array de 32×32 píxeles, mientas que para Conv2 contamos con 64 posiciones de salida (de 4 bits cada una) para conectarse al array de 64×64. Esto no implica ningún cambio en el circuito a nivel estructural, sino que es simplemente un escalado. Sin embargo, al trasladar este bloque al nuevo tamaño de 64 salidas se observa la aparición de una limitación causada por el retraso introducido, que empieza a ser crítica en relación al retraso total del proceso de lectura del dato, y que hará necesario el desarrollo de nuevas arquitecturas para futuros sistemas con un mayor número de píxeles.

En la versión de Conv1, cada bit que llega al bloque de desplazamiento horizontal tiene que actuar como entrada de 32 *buffers* tri-estado, de modo que en función de la señal de desplazamiento que se active, en cada momento saldrá por uno de ellos. Para reducir el *fan-out* de las señales de entrada, se introduce un árbol de inversores que limita la carga de cada nodo. Sin embargo, en la versión de Conv2 cada bit de entrada llega a 64 *buffers* tri-estado, por lo que es necesario rediseñar los árboles de inversores, que se ven incrementados. Todo esto provoca que haya más puertas lógicas que tiene que atravesar cada dato desde que sale de la RAM hasta que llega a los píxeles, así como líneas más largas y con mayor resistencia. Como consecuencia, el retraso en la propagación del dato se incrementa, limitando la frecuencia del reloj del sistema.

5.6. El generador AER

En la sección 5.1 se hizo una división de los bloques del chip de convolución entre etapa de entrada y etapa de salida. La etapa de entrada es la encargada de recibir los eventos de entrada y realizar el procesamiento necesario para que el kernel previamente programado se sume sobre las posiciones oportunas del array de píxeles. Toda esta etapa ha sido detallada en las secciones 5.2-5.5. Por otra parte, la etapa de salida se encarga de gestionar los eventos producidos por los píxeles de convolución y emitirlos fuera del chip. El bloque encargado de esta tarea es el que llamamos generador AER.

La estructura del generador AER está descrita en la literatura [94], y el diagrama de bloques puede verse en la Fig. 5.20. El funcionamiento básico es el siguiente: cada vez que un píxel del array alcanza su límite de saturación, éste activa su *Rqst* por fila a través de una línea común para todos los píxeles de una misma fila, implementando una función OR-cableada de todos ellos (señal *Rqst_row* que aparece en el esquemático de la Fig. 4.14). Una vez que el arbitrador por filas responde a la petición, todos los píxeles de la fila asentida que tengan en ese momento una petición pendiente activan la correspondiente señal por columnas (indicando el signo del evento producido). Estas peticiones por columnas se almacenan en la periferia, per-

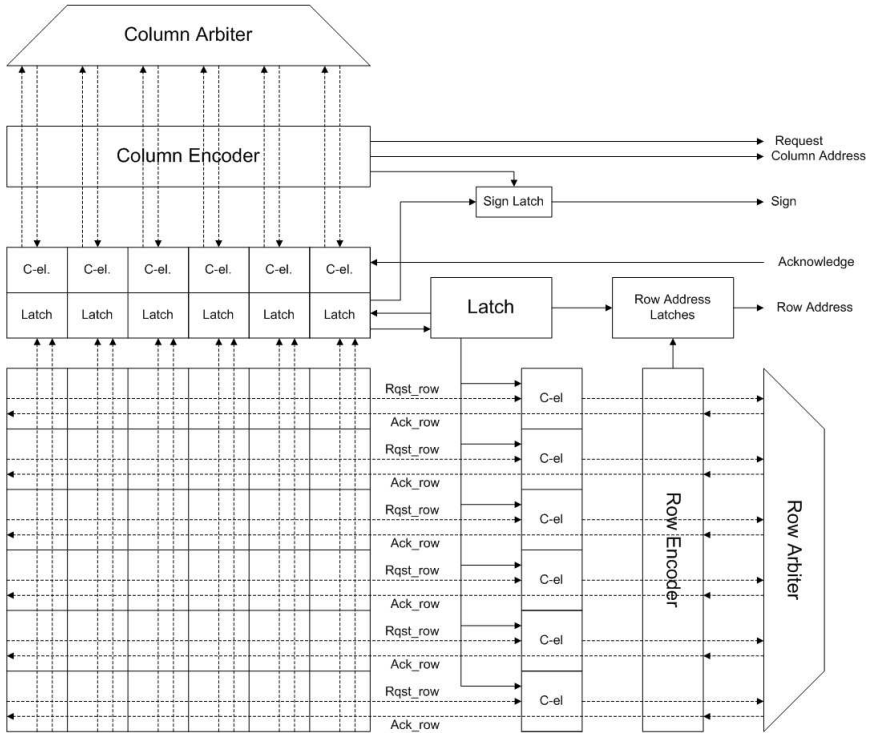


FIGURA 5.20. Diagrama de bloques del generador AER utilizado.

mitiendo que se libere el arbitrador por filas para que se atienda a una nueva petición mientras que todos los eventos de la fila anterior son emitidos en modo ráfaga.

El principal punto crítico que presenta este esquema está localizado en la línea común de petición por filas. Esta línea es compartida por todos los píxeles de una misma fila, de modo que cualquiera de ellos tiene que ser capaz de activarla en solitario, mientras que hay un transistor *pull-down* en la periferia encargado de desactivarla. De este modo, el dimensionamiento de estos transistores es bastante crítico para los retrasos de activación y desactivación de esta línea. Por ese motivo, en los chips de convolución presentados en este trabajo se han propuesto algunas mejoras.

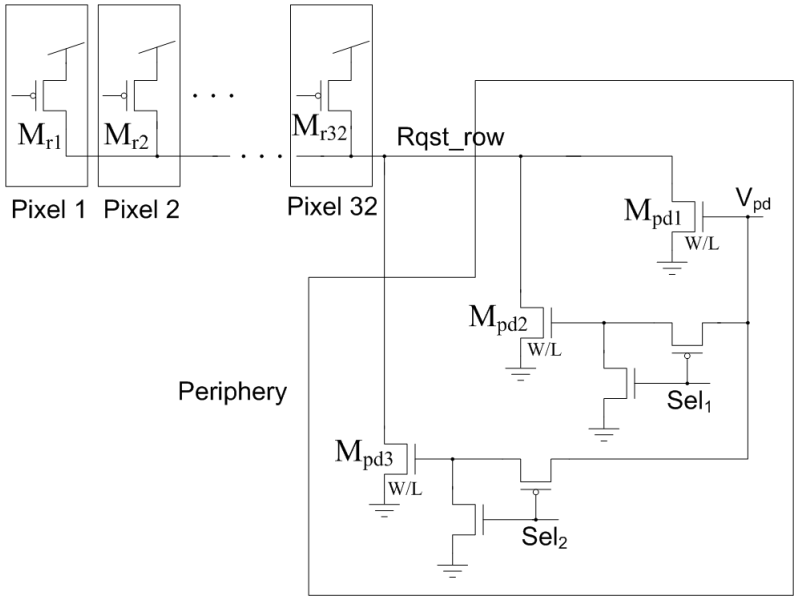


FIGURA 5.21. Esquema del circuito de comunicación entre los píxeles y el arbitrador por filas en Conv1.

En la primera versión (Conv1) se planteó la idea de aportar la mayor programabilidad posible a dichos transistores, según el esquema mostrado en la Fig. 5.21. La temporización en la comunicación entre los píxeles y la arbitración por filas es bastante crítica para un comportamiento óptimo, y por desgracia esta temporización depende mucho de los parásitos, y cambia de forma significativa entre distintos chips e incluso para distintas filas dentro de un mismo chip. Por este motivo se ha incluido un mecanismo de calibración a nivel de fila para los *pull-downs*. Como muestra la figura, el circuito de comunicación para cada fila incluye 32 transistores pMOS M_{r1}, …, M_{r32} (uno por cada píxel) y un *pull-down* nMOS M_{pd1} en la periferia. De este modo, el tamaño W/L de este transistor M_{pd1} tiene que diseñarse teniendo en cuenta que 1) cualquier píxel en solitario sea capaz de activar el nodo *Rqst_row*, y 2) el *pull-down* sea capaz de devolver *Rqst_row* a nivel bajo con suficiente rapidez. Si el *pull-down* resulta ser demasiado fuerte, su tensión de puerta V_{pd} (que por defecto está conectada a V_{dd}) se puede decrementar a través de un pad externo para reducir la corriente de

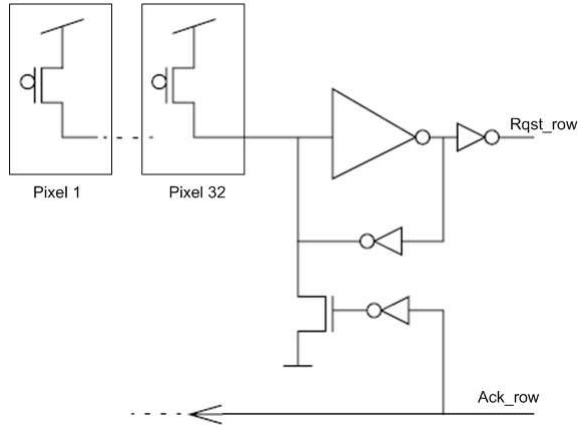


FIGURA 5.22. Esquema del circuito de comunicación entre los píxeles y el abitrador por filas en Conv2.

dicho transistor. Por otra parte, si el *pull-down* resulta ser demasiado débil, los transistores M_{pd2} y/o M_{pd3} pueden ser activado a través de los *switches* de calibración Sel_1 y Sel_2. Si ambos *switches* está apagados, el tamaño del *pull-down* es W/L; si sólo uno de ellos está conectado el tamaño pasa a ser 2W/L; y si los dos están conectados, entonces será 3W/L. Así, con esta calibración por filas junto con la posibilidad de ajustar la señal global V_{pd} podemos optimizar la temporización y compensar las variaciones del proceso de fabricación.

Por otra parte, en la segunda versión del chip de convolución (Conv2), aparte de que el bloque completo del generador AER está escalado al tamaño de 64×64, se ha empleado un nuevo esquema de comunicación por filas para eliminar los problemas del esquema previo [43]. La nueva estructura se muestra en la Fig. 5.22. En ella, el transistor *pull-down* encargado de resetear el nodo común a todos los píxeles de la fila es controlado por una señal *Ack_row* generada por el arbitrador, y no por una tensión de polarización como en el esquema anterior. De este modo, este transistor no está activo cuando los píxeles están haciendo una petición, así que no tienen que luchar contra él para activar la línea. El pulso de *Rqst* se almacena en el *latch* asimétrico de forma que el *pull-down* no tenga que luchar con el inver-

sor fuerte, sino con el débil. Así conseguimos que la comunicación por filas sea más robusta y más rápida.

CAPÍTULO 6

Resultados experimentales

6.1. Introducción

En este capítulo presentamos los resultados experimentales obtenidos con los chips de convolución fabricados, que han sido descritos en los capítulos anteriores.

En primer lugar, la Sección 6.2 muestra algunas pruebas a nivel de píxel efectuadas sobre un primer prototipo formado por un array de 2×2 píxeles de la versión inicial. Estas pruebas sirvieron fundamentalmente para validar dicho píxel como paso previo al diseño del chip de convolución completo con el resto de bloques.

A continuación, la Sección 6.3 describe las herramientas básicas utilizadas para los tests realizados sobre el chip de convolución, concretamente las diversas placas para gestionar los eventos AER y el entorno software que junto con dichas placas nos permite controlar los chips y realizar sobre ellos las pruebas oportunas.

Por último, en las Secciones 6.4 y 6.5 se detallan las pruebas realizadas sobre el chip de convolución Conv1 y Conv2, respectivamente, mostrando

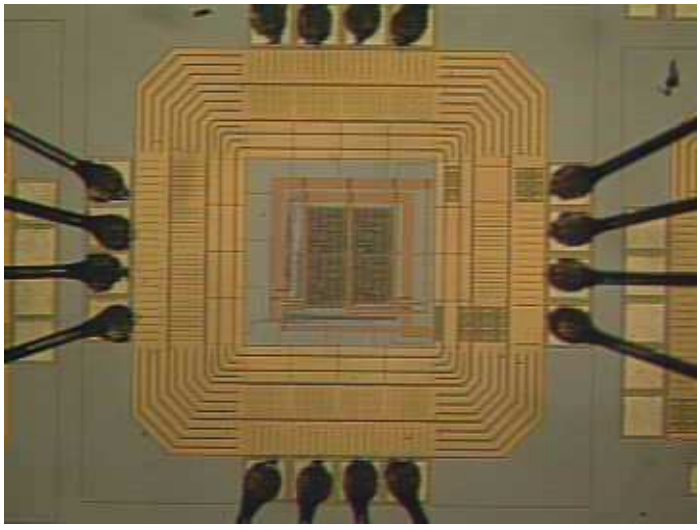


FIGURA 6.1. Fotografía del prototipo de 2x2 píxeles de convolución.

los principales resultados obtenidos. Para el chip Conv2 no se han podido completar todos los tests debido a un error en la implementación del controlador síncrono, que ha sido detectado y corregido en una nueva versión que en el momento de la elaboración de este documento se encuentra en fabricación. Así pues, se incluyen solamente los resultados relativos a la caracterización del chip, ya que estos no están afectados por el error del controlador.

6.2. Prototipo de 2x2 píxeles

Como un primer paso antes de diseñar todos los bloques del chip de convolución descritos en los capítulos previos, se planteó la necesidad de comprobar el correcto funcionamiento del píxel digital, como elemento base del sistema completo. Para ello, se fabricó en la tecnología de AMS $0.35\mu m$ un pequeño circuito con un array de 2×2 píxeles. En la Fig. 6.1 podemos ver una fotografía de dicho circuito.

El prototipo almacena un único valor de kernel que se comparte por los 4 píxeles. Para evitar tener un número elevado de pads en este circuito, se

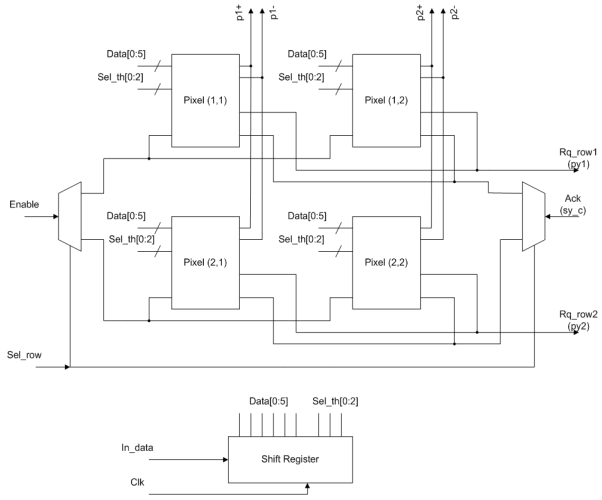
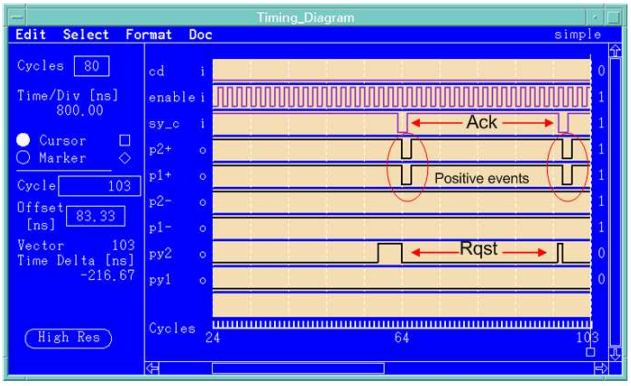


FIGURA 6.2. Esquema de la configuración de test utilizada para controlar el prototipo de 2x2 píxeles de convolución.

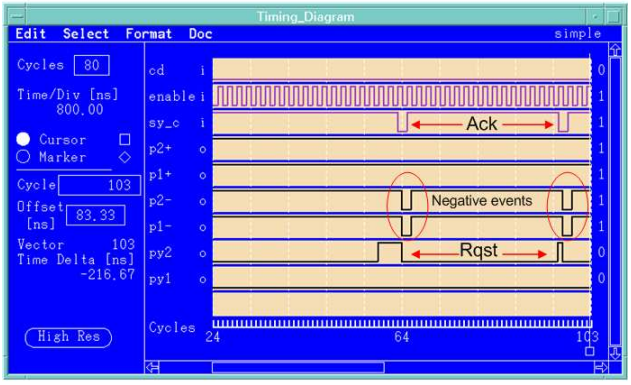
utilizó por una parte un registro de desplazamiento a través del cual podemos programar el valor de la palabra del kernel y del límite del acumulador (ambos datos comunes para los 4 píxeles), y por otra parte una señal *Sel_row* encargada de seleccionar cuál de las dos filas debe recibir las señales *Enable* y *Ack*, que externamente son comunes. Esta configuración se puede entender mejor con la ayuda del esquema mostrado en la Fig. 6.2. Se testó con el Agilent 82000, dada la simpleza de los tests. Más adelante, para test con estímulos asíncronos se empleará una infraestructura AER específica.

6.2.1. Caracterización del píxel

Para realizar pruebas sobre este prototipo, seguimos el siguiente método: primero se programa el registro de desplazamiento seleccionando un límite del acumulador y estableciendo el valor del dato del kernel, y a continuación se habilita una de las filas a través de la señal *Sel_row.* Una vez configurado de este modo, podemos enviarle señales de *Enable* y observar que los píxeles activan la señal *Rqst_row* cuando el acumulador alcanza el límite programado. Externamente programamos la señal *Ack* para que



a) Positive input word



b) Negative input word

FIGURA 6.3. Resultados del test del prototipo 2x2 con un límite del acumulador de 512 para valores positivos y -513 para los negativos.

responda y así podemos comprobar que el signo de los eventos es el correcto observando las señales *p1+*, *p1-*, *p2+* y *p2-*.

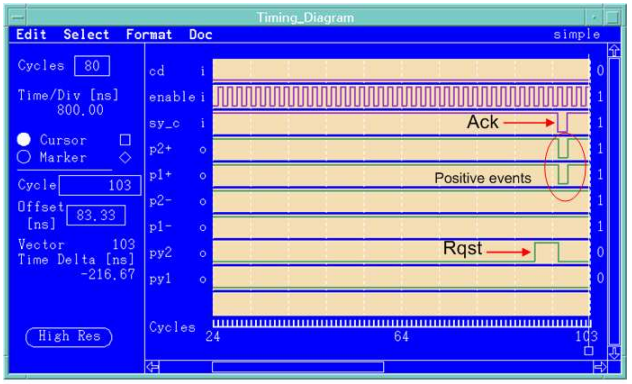
En la Fig. 6.3 podemos observar los resultados obtenidos de una primera prueba. En este caso se estableció el límite del acumulador en $2^9 = 512$ para valores positivos y $-2^9 - 1 = -513$ para valores negativos En este primer prototipo los valores seleccionables no eran los mismos que en la versión integrada en el chip de convolución Conv1 descrita en el capítulo correspondiente. Por otra parte, se programó un dato del kernel de 31 (el máximo valor positivo posible con 6 bits en complemento a 2) en primer

lugar. Con esta configuración, seleccionando la fila 2, se le enviaron pulsos de *Enable* al circuito, observando cómo la señal *Rq_row2* producía un evento después de acumular 17 pulsos de entrada ($16 \times 31 = 496 < 512$, y $17 \times 31 = 527 \geq 512$). También podemos comprobar cómo al recibir la señal de *Ack* ambos píxeles activan la señal *p1+* y *p2+*, indicando el signo del evento. Todo esto aparece reflejado en la gráfica superior de la Fig. 6.3. En general, en los resultados del test se aprecia que los pulsos de *Rqst* tienen diferentes duraciones. Esto es debido a que, si bien la activación de dicha señal depende del instante en el cual se alcanza el límite del acumulador, la desactivación se produce cuando externamente se activa la señal *Ack*. Así, en función del instante en el que programemos la activación del *Ack*, el pulso de *Rqst* durará más o menos.

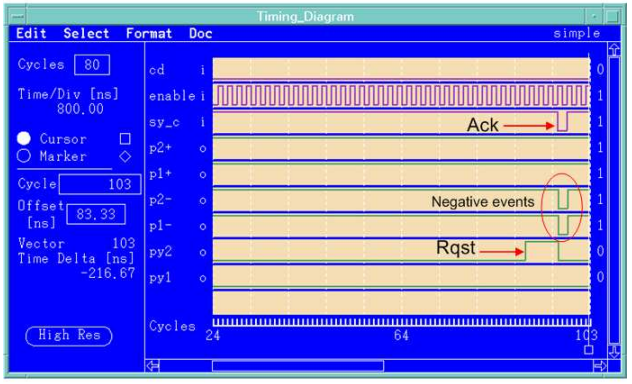
En cuanto a la gráfica inferior de la Fig. 6.3, en ella simplemente se modifica el valor del dato del kernel, siendo en este caso -31. De este modo, comprobamos que el límite del acumulador se alcanza igualmente tras recibir 17 pulsos de entrada ($16 \times (-31) = -493 > -513$, y $17 \times (-31) = -527 \leq -513$). La diferencia que podemos observar es que al recibir la respuesta de la señal *Ack*, los píxeles activan en esta ocasión la señal *p1-* y *p2-*, indicando que han alcanzado el límite negativo del acumulador.

Para comprobar qué ocurre cuando fijamos un límite diferente del acumulador, hacemos una segunda prueba, seleccionando en este caso como umbral $2^{10} = 1024$ para valores negativos y $-2^{10} - 1 = -1025$ para valores negativos. Al repetir con estos parámetros las mismas pruebas que hicimos con la configuración anterior, obtenemos los resultados mostrados en la Fig. 6.4.

Así, en la gráfica superior de la Fig. 6.4 vemos lo que ocurre con un dato programado de 31. Si contamos el número de eventos de entrada recibidos antes de generar un evento de salida, vemos que son 34, lo cual resulta coherente al ser $33 \times 31 = 1023 < 1024$. De este modo, una vez que recibe el evento número 34 los píxeles de la fila 2 (que son los que tenemos seleccionados en esta prueba) generan un pulso de *Rqst*, y al recibir la señal *Ack* activan *p1+* y *p2+* indicando que el evento es positivo. En la gráfica inferior de la Fig. 6.4 programamos como dato del kernel -31, de modo que los píxeles igualmente activan el *Rqst* después de recibir 34 pulsos de



a) Positive input word



b) Negative input word

FIGURA 6.4. Resultados del test del prototipo 2x2 con un límite del acumulador de 1024 para valores positivos y -1025 para los negativos.

Enable ($33 \times (-31) = -1023 > -1025$). También se comprueba cómo activan las señales *p1-* y *p2-* para indicar que ambos píxeles han alcanzado el límite negativo del acumulador.

En cuanto a los datos de temporización medidos dentro del bloque de interfaz del protocolo de los píxeles, podemos ver los resultados representados en la Fig. 6.5. Por una parte, podemos ver el retraso desde el flanco de bajada de la señal *Enable* que se encarga de que se almacene el resultado de la suma y el flanco de subida de la señal *Rqst_row* en el caso de que al alma-

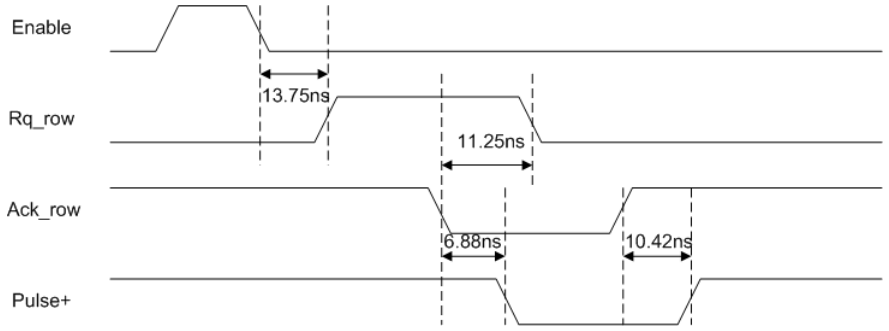
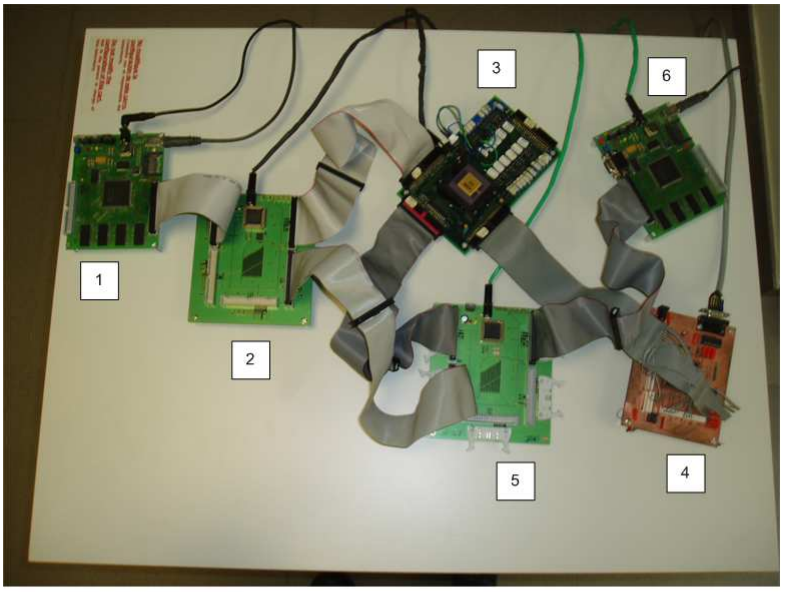


FIGURA 6.5. Datos de temporización obtenidos con el prototipo de 2x2.

cenar dicho resultado se haya superado el umbral programado. Del mismo modo, se muestra el tiempo que tarda el píxel en desactivar la señal *Rqst_row* después de recibir la respuesta de la periferia. Y por último, los retrasos que añade el píxel al activar y desactivar las señales verticales *Pulse+* cuando recibe el permiso para hacerlo a través de la señal *Ack*.

6.3. Infraestructura AER para tests asíncronos

Para la realización de los diversos tests que se han llevado a cabo con los chips de convolución diseñados para el presente trabajo, se han utilizado una serie de placas configurables [48], [96], [97], [98] que, junto con el entorno software adecuado, han facilitado mucho el desarrollo de las pruebas. En la Fig. 6.6 se muestra un ejemplo de infraestructura AER para test. En ella podemos ver una placa AER con un chip de convolución (3) con su correspondiente placa de configuración (4). Además, tenemos dos placas USB-AER, una de ellas configurada como *data-player* (1), y la otra como *data-logger* (6). Por último, se pueden ver dos placas *Splitter-Merger*, una configurada en modo *Splitter* (2) y la otra como *Merger* (5). A continuación incluimos una breve descripción de dichas placas, así como del entorno software.



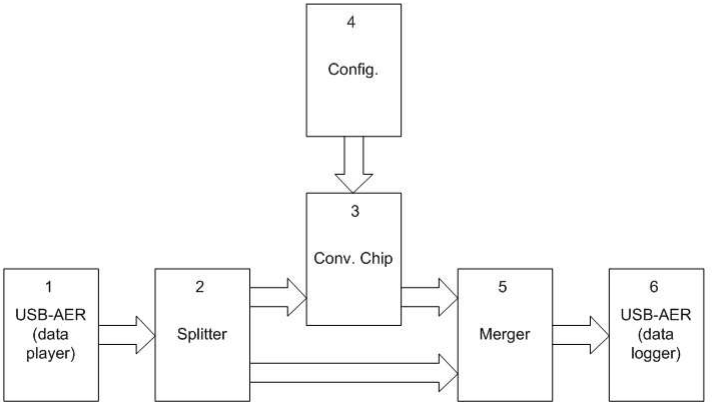


FIGURA 6.6. Ejemplo de infraestructura AER para test. Fotografía de las distintas placas en la parte superior, y esquema correspondiente en la parte inferior.

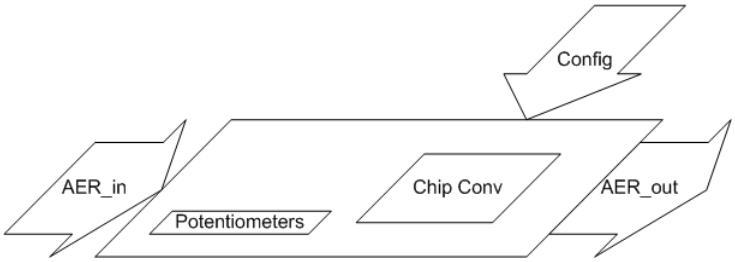


FIGURA 6.7. Esquema de la placa de convolución.

6.3.1. Placa AER

Ésta es una placa sobre la cual se inserta el propio chip de convolución (encapsulado sobre un PGA-100). Como se puede ver en la Fig. 6.7, esta placa incluye una serie de potenciómetros para controlar las tensiones de polarización del chip (para establecer la frecuencia de reloj y para controlar los *pull-ups* y *pull-downs* del generador AER). Además, la placa recibe como entrada un bus de configuración a través del cual se programan los registros de parámetros y el valor del kernel. Por último, cuenta con dos buses AER, uno de entrada y otro de salida, que se conectan al chip de convolución. En la Fig. 6.6, se trata de la placa 3.

6.3.2. Placa USB-AER

Esta placa se encarga de gestionar el tráfico AER, y se puede utilizar en dos modos de funcionamiento diferentes: como *data-player* y como *data-logger*. El modo de funcionamiento se selecciona programando el *firmware* correspondiente desde el PC [48].

En la parte superior de la Fig. 6.8 se puede ver el esquema del funcionamiento en modo *data-player* (placa 1 de la Fig. 6.6). En él, la placa recibe desde un PC una lista de eventos AER a través del puerto USB. Esta lista de eventos incluye la dirección y la marca temporal (*timestamp*) de cada uno de ellos, hasta un máximo de 500k-eventos. Una vez almacenados los eventos en una memoria incluida en la propia placa, al recibir la orden de repro-

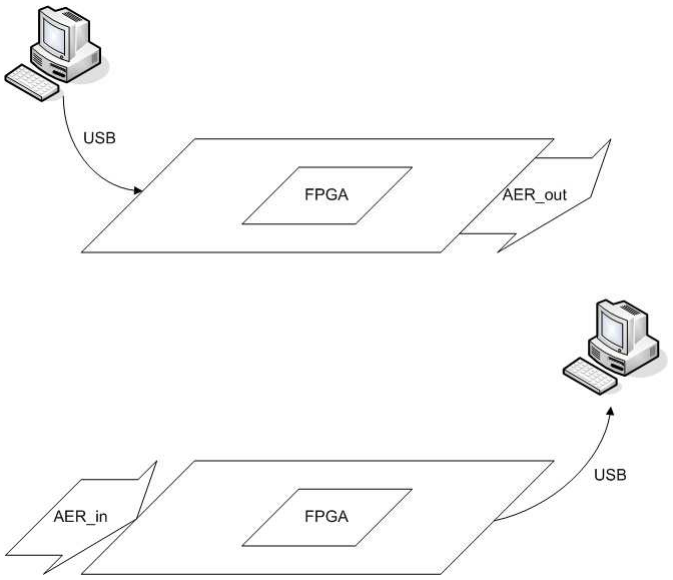


FIGURA 6.8. Esquema de la placa USB-AER, en la parte superior configurada en modo *data-player*, y en la parte inferior en modo *data-logger*.

ducción desde el USB ésta envía estos eventos a través del puerto AER de salida, respetando la temporización indicada por las marcas temporales y el *handshaking* con el chip receptor. Las listas de eventos reproducidas por esta placa pueden ser generadas artificialmente, o bien haber sido capturadas a partir de otro chip AER, ya sea una retina u otro chip de convolución que implemente una capa de procesamiento previa. Cada evento se almacena como una palabra de 32 bits, y la velocidad máxima que permite es de 10×10^6 eventos por segundo (un tiempo entre eventos de 100*ns*). Configurando el *firmware* de la forma adecuada, podemos hacer que la lista de eventos programada en la placa se emita una sola vez o bien que se emita en modo repetitivo indefinidamente.

En la parte inferior de la Fig. 6.8 se encuentra el esquema del funcionamiento en modo *data-logger* (placa 6 de la Fig. 6.6). Cuando la placa está configurada en este modo, recibe eventos de entrada a través de su bus AER. La placa se encarga de gestionar el *handshaking* con el chip emisor y de almacenar los eventos en su memoria interna añadiendo el *timestamp*. Una vez concluida la prueba, desde el PC se pueden leer los eventos alma-

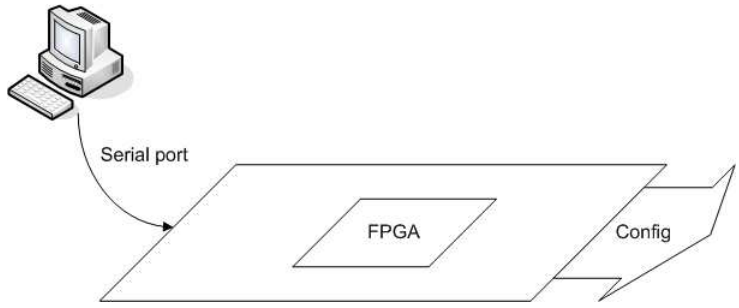


FIGURA 6.9. Esquema de la placa de configuración.

cenados a través del puerto USB, conservando dichos eventos las direcciones y marcas temporales correspondientes, lo que permite el análisis y procesamiento de los mismos.

6.3.3. Placa de configuración

Esta placa auxiliar la utilizamos para configurar el chip de convolución antes de realizar cualquier prueba. Como se puede ver en la Fig. 6.9, cuenta con una FPGA que controlamos desde un PC mediante el puerto serie. De este modo, cada vez que queremos modificar el valor de algún dato de configuración del chip, o bien del kernel programado en la RAM, enviamos dichos datos a la FPGA para que ésta los envíe a su vez en formato serie a través del bus de configuración que se conecta con la placa de convolución de la Fig. 6.7. En la Fig. 6.6 es la placa 4.

6.3.4. Placa Splitter-Merger

A pesar de que en principio trabajamos con links AER punto a punto, en general es necesario gestionar sistemas multi-emisor y multi-receptor, así como el remapeo de eventos. Las placas anteriores permiten realizar pruebas básicas sobre el chip de convolución, sin embargo, para montar sistemas multichip es necesario un elemento más, las placas *splitter-merger* [48]. Estas placas se pueden configurar en dos modos diferentes a través de unos *jumpers* incluidos en la propia placa: en modo *splitter* o en modo *merger*. La placa cuenta con 2 puertos AER fijos: uno de entrada y otro de salida. Los 3

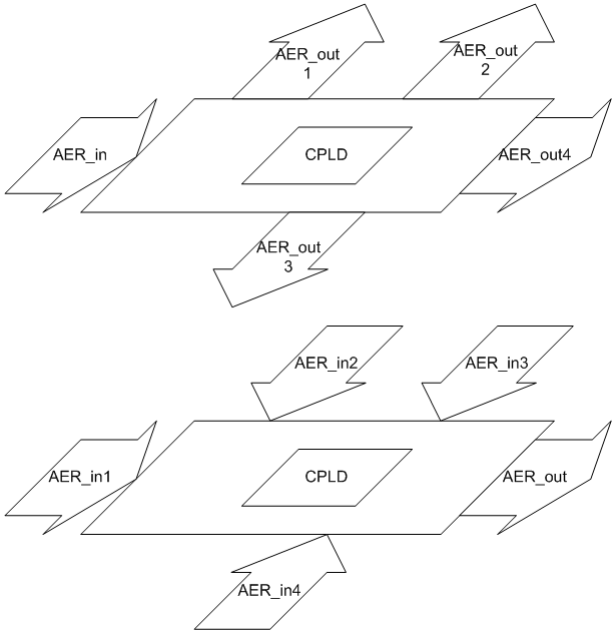


FIGURA 6.10. Esquema de la placa Splitter-Merger, en la parte superior configurada como *splitter* y en la parte inferior como *merger.*

puertos AER restantes incluyen *buffers* bidireccionales y se pueden configurar en cualquiera de los dos modos. La placa utiliza una CPLD que actúa sobre las señales de *handshaking* y se encarga de habilitar o deshabilitar los *buffers.*

En la parte superior de la Fig. 6.10 se puede ver el esquema del funcionamiento de la placa en modo *splitter.* En este modo, la placa recibe un bus de entrada AER y los eventos que llegan por dicho bus los envía por los 4 buses de salida AER que tiene. En realidad, se puede conectar o desconectar cada uno de los 4 buses de salida, dejando habilitados solamente los que nos interesen para nuestra aplicación. Con ello podemos montar con facilidad un sistema que envíe los eventos generados por un chip de convolución de la etapa *i* a varios chips de convolución en paralelo de la etapa *i+1*. En la Fig. 6.6 se corresponde con la placa 2.

En la parte inferior de la Fig. 6.10 se puede ver el esquema del funcionamiento en modo merger. En este modo, la placa recibe hasta 4 buses de entrada AER y los eventos que llegan por todos ellos los emite por el único bus de salida AER que tiene. Para ello es necesario que la propia placa introduzca una cierta arbitración entre las entradas, de modo que puede alterar ligeramente la temporización de los eventos, aunque generalmente no suele afectar a nuestras aplicaciones. Gracias a esta placa, podemos construir un sistema multichip en el cual una serie de chips en paralelo de la etapa *i* conecten todas sus salidas a la entrada de un único chip de convolución de la etapa *i+1*. En la Fig. 6.6 se corresponde con la placa 5.

6.3.5. Entorno software

Para controlar la configuración del chip de convolución, se ha utilizado una aplicación software basada en MATLAB. Con esta aplicación se facilita la escritura de los registros de configuración del chip, así como la escritura del kernel en la RAM. En la Fig. 6.11 se puede ver una imagen de la interfaz gráfica de usuario (GUI en sus siglas en inglés) desarrollada para este propósito.

Esta aplicación está diseñada con la única finalidad de gestionar la configuración del chip de convolución, pero no controla el envío y la recolección de eventos AER. Para estas tareas, se utilizan una serie de funciones de MATLAB que se ejecutan desde la línea de comandos. Estas funciones se encargan de comunicarse con las placas descritas en los apartados previos, tanto para establecer sus modos de funcionamiento a través de la escritura del firmware apropiado, como para enviar o recibir listas de eventos AER.

6.4. Chip de convolución 32x32 Conv1

En este apartado vamos a describir los resultados experimentales obtenidos sobre el chip de convolución Conv1 de 32×32 píxeles, fabricado con un área total de $4.3 \times 5.4 mm^2$ en la tecnología de AMS $0.35 \mu m$. Podemos ver una fotografía del chip en la Fig. 6.12. El bloque que ocupa una

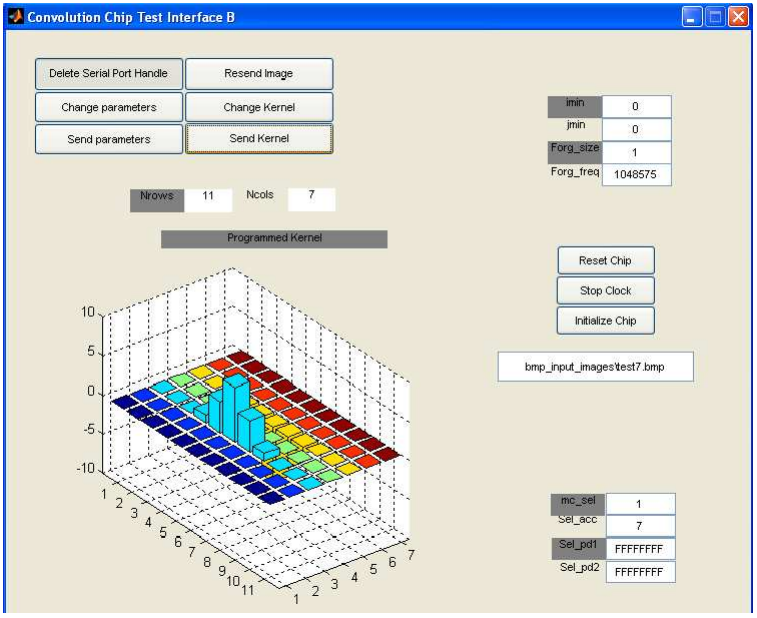


FIGURA 6.11. Interfaz basada en MATLAB para controlar la configuración del chip de convolución.

mayor área es el array de píxeles, con $3.0 \times 3.2 mm^2$ aproximadamente. El controlador síncrono ocupa unos $4500 \times 300 \mu m^2$, la RAM de 32×32 palabras de 6 bits unos $600 \times 2700 \mu m^2$ y el bloque de desplazamiento horizontal unos $600 \times 3100 \mu m^2$. Los bloque restantes como el generador AER, el inversor de complemento a 2 o el generador de reloj consumen mucha menos área.

A continuación vamos a detallar la caracterización del chip, para después mostrar los resultados obtenidos para algunas pruebas concretas.

6.4.1. Caracterización del chip

6.4.1.1. Reloj interno

En primer lugar, la frecuencia del reloj interno se puede ajustar a través de la tensión de polarización V_{bias_clk}. De las pruebas llevadas a cabo sobre el chip se comprueba que esta frecuencia se puede subir hasta los 120MHz

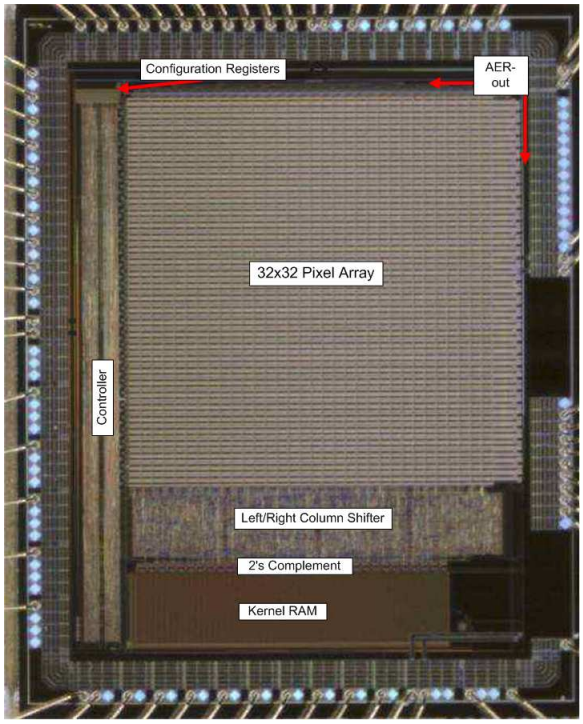


FIGURA 6.12. Fotografía del chip de convolución Conv1.

antes de observar la aparición de algunos eventos espúreos a la salida. Para los casos en los que estos espúreos puedan ser tolerables, la frecuencia de reloj se puede incrementar hasta los 200MHz. Con frecuencias mayores la operación de convolución se degrada completamente. En nuestros experimentos, con la finalidad de obtener unos resultados con la mayor precisión posible, fijamos la frecuencia de reloj a 120MHz.

6.4.1.2. Consumo de potencia

En cuanto al consumo de potencia del chip, éste depende tanto de la tasa de eventos a la entrada como del tamaño del kernel. Por ejemplo, para una tasa de entrada de 5*Meps* (eventos por segundo) el consumo de potencia varía entre los 66*mW* y los 198*mW*, siendo éstos para el kernel más pequeño posible (de 1×1) y para el mayor posible (32×32), respectivamente.

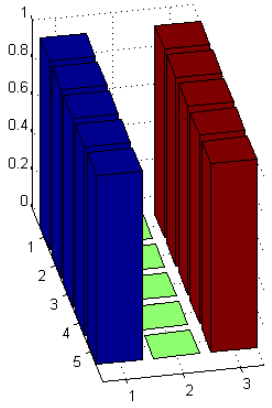


FIGURA 6.14. Kernel programado para medir la máxima tasa de salida en los píxeles situados en las cuatro esquinas del array.

6.4.1.3. Caracterización temporal

El chip de convolución puede generar eventos de salida con una tasa máxima de 50×10^6 eps, medidos cortocircuitando las señales *Rqst_out* y *Ack_out* y para eventos generados por píxeles de una misma fila (modo ráfaga). Esta tasa se corresponde con un tiempo entre eventos de $T_{rafaga} = 20ns$. Sin embargo, dependiendo de la posición de los píxeles dentro del array, el tiempo entre eventos cambia de forma significativa. Los píxeles cercanos a los arbitradores provocan menores retrasos de propagación. No obstante, esta influencia de la posición se puede minimizar ajustando cuidadosamente los valores de las tensiones de polarización y switches de calibración de los *pull-ups* y *pull-downs* del generador AER. De este modo, conseguimos compensar los retrasos a un rango que oscila entre 20 y 24*ns* para todos los píxeles del array.

En la Fig. 6.13 podemos ver las señales medidas *Rqst_in*, *Ack_in* *Rqst_out* (cortocircuitada con *Ack_out*) con el kernel de 5 filas de la Fig. 6.14. El umbral de acumulación de los píxeles estaba fijado para producir un evento de salida al recibir un único evento de entrada. De ese modo, este kernel activa 10 píxeles diferentes pertenecientes a 5 filas distintas para cada evento de entrada. Los eventos de salida de la Fig. 6.13 muestran un

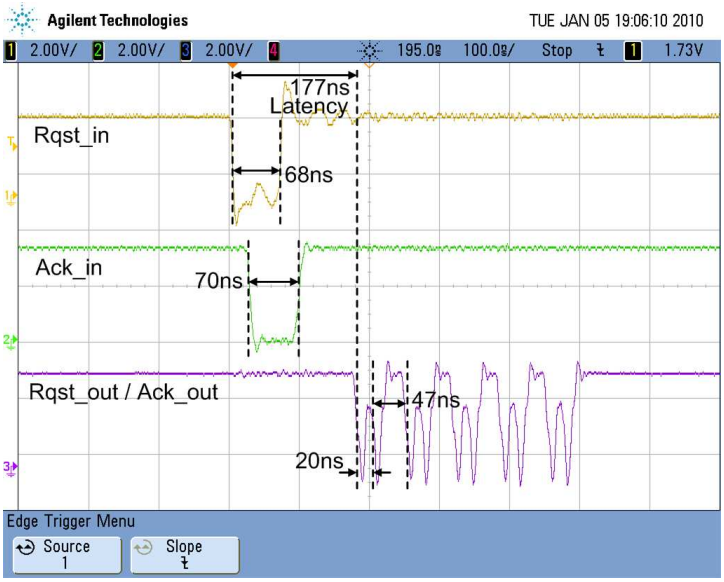


FIGURA 6.13. Señales Rqst y Ack medidas para los eventos de entrada y salida para una frecuencia de reloj de 120MHz, con Rqst_out y Ack_out cortocircuitadas. El osciloscopio usado es el Agilent DSO7054A, con un ancho de banda de 500MHz y una tasa de muestreo de 4GSa/s.

tiempo de $T_{rafaga} = 20ns$ entre cada par de eventos pertenecientes a una misma fila (una vez que dicha fila es asentida por el arbitrador, todos los eventos de esa misma fila se emiten en modo ráfaga), y un tiempo de $T_{no-rafaga} = 47ns$ para eventos generados por diferentes filas. Como consecuencia, podemos decir que la diferencia $47ns - 22ns = 25ns$ es el retraso introducido por el arbitrador por filas T_{arb}.

La tasa de eventos a la entrada depende tanto del tamaño del kernel como de la frecuencia de reloj. Como ya se ha comentado antes, el controlador síncrono necesita $n_{clk} = 4 + (2 \times n_k)$ ciclos de reloj para procesar cada evento, siendo n_k el número de filas del kernel (hasta un máximo de 32). De esta forma, para una frecuencia de reloj de 120MHz (que se corresponde con un periodo $T_{clk} = 8.33ns$), la máxima tasa de eventos posible a la entrada es de 20×10^6 eps, lo cual se corresponde con un tiempo entre eventos de $50ns$ cuando el kernel tiene una sola fila. Para un kernel com-

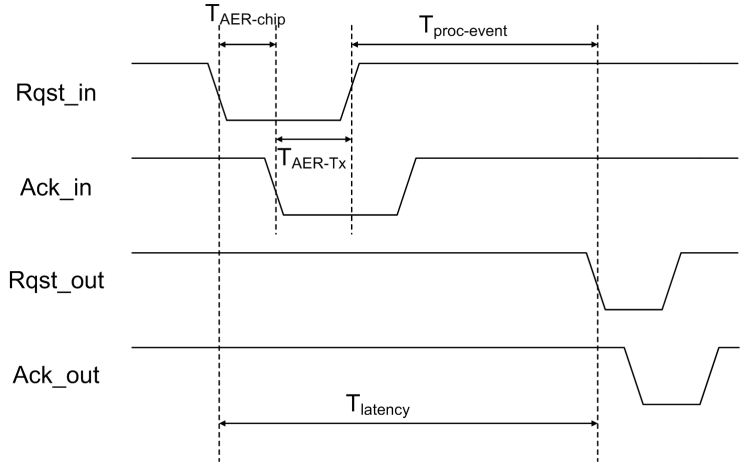


FIGURA 6.15. Retrasos entre eventos de entrada y salida.

pleto de 32 filas, la máxima tasa de entrada es de 1.77×10^6 eps, lo cual se corresponde con un tiempo entre eventos de 566*ns*.

En la Fig. 6.15, la latencia entre un evento de entrada y uno de salida se representa mediante $T_{latency}$, que puede ser expresada como $T_{latency} = T_{AER-chip} + T_{AER-Tx} + T_{proc-event}$. El retraso llamado $T_{AER-chip}$ es el que introduce el propio chip de convolución desde que recibe el *Rqst_in* hasta que responde con el *Ack_in*. El retraso T_{AER-Tx} es el que introduce el emisor desde que el chip de convolución asiente el evento hasta que retira la señal *Rqst_in*. Y el retraso $T_{proc-event}$ es el tiempo empleado por el chip para procesar el evento de entrada y generar el evento de salida (considerando una situación en la cual un evento de entrada produce uno de salida). De este modo, el retraso $T_{proc-event}$ se puede expresar a su vez como $T_{proc-event} = T_{syn} + T_{asyn}$, donde T_{syn} representa el tiempo que el controlador síncrono necesita para sumar el kernel sobre el array de píxeles, y T_{asyn} representa el tiempo que la arbitración asíncrona necesita para generar el evento de salida.

La Fig. 6.16 nos muestra los valores medidos de $T_{AER-chip}$ y T_{AER-Tx} para distintos valores del periodo de reloj T_{clk}. Las pequeñas barras verticales indican las desviaciones sobre 5000 medidas. Como podemos compro-

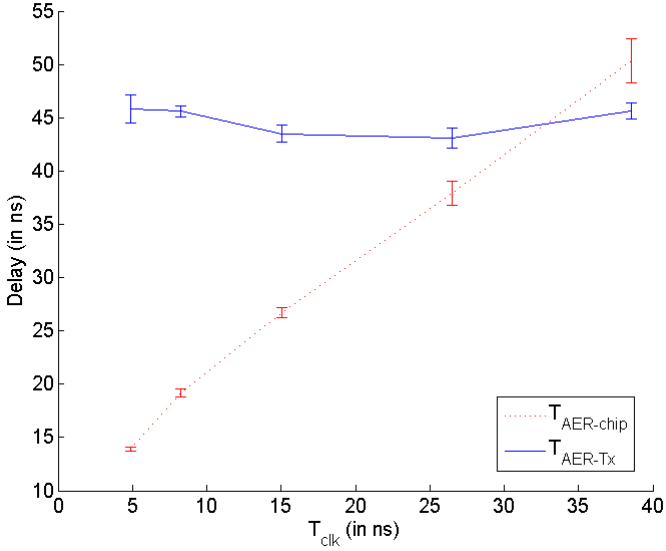


FIGURA 6.16. Valores medidos de $T_{AER-chip}$ y T_{AER-Tx} para distintos periodos de reloj.

bar fácilmente, el retraso T_{AER-Tx} es constante (no depende del periodo de reloj), ya que sólo depende del emisor. El retraso medido para la placa emisora en nuestros tests resulta en unos $T_{AER-Tx} = 44ns$. En cambio, $T_{AER-chip}$ depende linealmente de T_{clk}, como muestra la figura, alcanzando unos $14ns$ para $T_{clk} = 5ns$ y con un valor residual estimado extrapolando estos resultados de unos 10ns para $T_{clk} = 0$ (probablemente debido a retrasos internos de las líneas y pads).

La Fig. 6.17 muestra los valores medidos de $T_{latency}$ frente al periodo de reloj. El ajuste lineal de estos datos revela que para un periodo de reloj $T_{clk} = 0$ tendríamos una latencia de $97ns$, lo cual se correspondería con $T_{proc-event} = T_{asyn}$, ya que en este caso T_{syn} valdría 0. Extrapolando también los valores de $T_{AER-chip}$ y T_{AER-Tx}, podemos estimar el valor de $T_{asyn} = 97ns - 44ns - 10ns = 43ns$, lo cual es coherente con el valor de $47ns$ medido entre dos eventos de salida pertenecientes a filas diferentes.

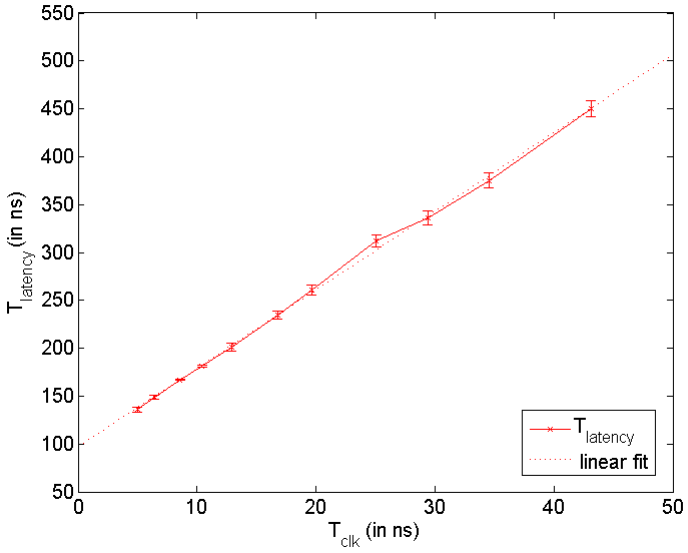


FIGURA 6.17. Valores medidos de $T_{latency}$ para distintos periodos de reloj. Cada medida se ha repetido 5000 veces. La barras de error indican la dispersión. La línea de puntos representa el ajuste lineal ($y = 8.2x + 97$).

Para una frecuencia de reloj de 120MHz el periodo de latencia medido es de $177ns$.

Al conectar chips de convolución en cascada, el retraso T_{AER-Tx} introducido por la placa emisora debería ser reemplazado o bien por $T_{rafaga} = 22ns$, o bien por $T_{no-rafaga} = 47ns$. Como consecuencia, la verdadera mínima latencia al conectar los chips en cascada vendría dada por $T_{latency} = 177ns - 44ns + 22ns = 155ns$. Esta latencia es independiente del número de filas del kernel programado, ya que se trata de el retraso entre un evento de entrada y el primer evento de salida producido por la primera fila del kernel.

En la tabla Tabla 6.1 se encuentran resumidas las especificaciones del chip de convolución Conv1.

TABLA 6.1. Especificaciones del chip Conv1.

Tecnnología	4M 2P $0.35\mu m$ CMOS
Tamaño del píxel	$95.6 \times 101.3\mu m^2$
Tamaño del chip	$4.3 \times 5.4mm^2$
Array de píxeles	32×32
Resolución del píxel	18 bits
Resolución del kernel	6 bits
Computación con signo	Sí
Tasa de eventos de entrada	1.77-20 Meps
Máxima tasa de salida	50 Meps
Mínima latencia entrada-salida	155ns
Consumo de potencia	200mW máximo

6.4.2. Convolución de imágenes estáticas

Para ilustrar la operación de convolución, se ha seleccionado en primer lugar una imagen de 32×32 píxeles de una fotografía real, que se muestra en la Fig. 6.19.(a), para llevar a cabo un procesamiento de extracción de bordes verticales con el kernel de tamaño 11×7 mostrado en la Fig. 6.18. Este kernel viene dado por una diferencia de gausianas, descrito mediante las ecuaciones (6.1), (6.2) y (6.3):

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

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

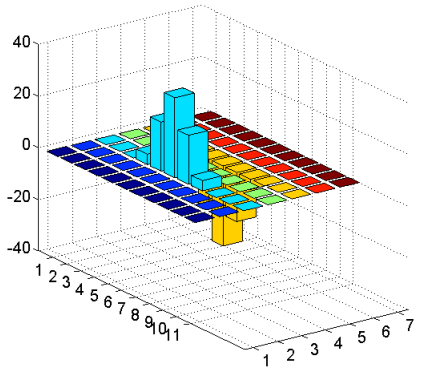


FIGURA 6.18. Kernel descrito como diferencia de gausianas para extracción de bordes verticales.

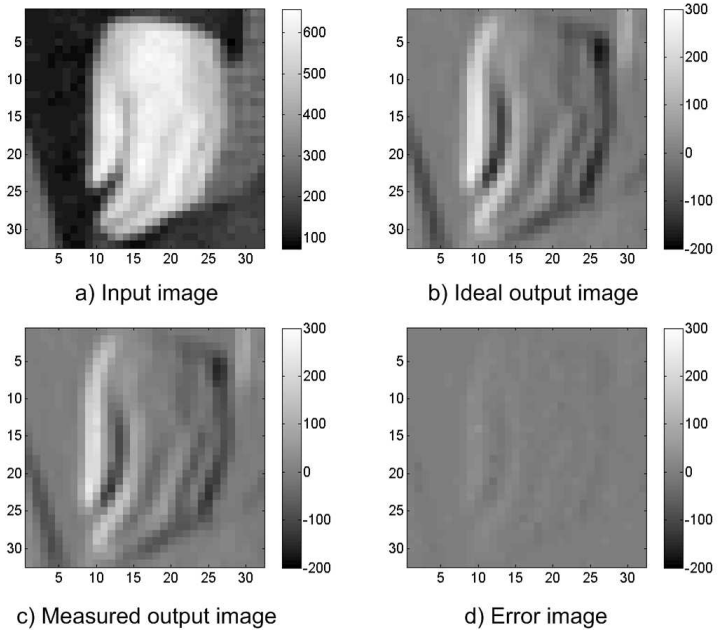


FIGURA 6.19. Resultado de calcular una convolución experimentalmente con un kernel para extracción de bordes verticales.

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

siendo σ_{gh} y σ_{gv} los parámetros de anchura horizontal y vertical de los lóbulos de la gaussiana.

La imagen se codificó a eventos AER asociando un nivel de frecuencia de eventos a cada nivel de gris, obteniendo una máxima frecuencia de 660Hz. Las frecuencias asociadas a cada nivel de intensidad se indican en las barras verticales a la derecha de cada imagen en la Fig. 6.19. La computación matemática de la operación de convolución se llevó a cabo con MATLAB, obteniendo la imagen de la Fig. 6.19.(b). En la Fig. 6.19.(c) se muestra la salida del chip de convolución, mapeando la frecuencia de salida de los eventos de cada píxel con su signo a un nivel de gris. Una frecuencia negativa indica que el bit de signo de los eventos de salida de ese píxel es negativo. En la Fig. 6.19.(d) se representa la imagen error calculada como la diferencia entre la salida ideal y la medida.

Aunque el tamaño de este chip es de sólo 32×32 píxeles, el espacio de direcciones de entrada que puede ver es de 128×128. Esto nos permite construir un array bidimensional de chips de convoluciones para procesar arrays de tamaños múltiplos de 32×32 [78], [95], [72]. Para cada chip de convolución se programan sus parámetros (i_{min}, j_{min}), (i_{max}, j_{max}) para indicar la posición de cada array de 32×32 dentro del total de 128×128. Utilizando placas *splitter* y *merger* [48], es posible construir un array de 4×4 chips de convolución para procesar imágenes de 128×128 píxeles. Para procesar imágenes aún mayores es necesario usar también bloques *mapper* AER [48], para mapear convenientemente el espacio de direcciones de entrada a los 128×128 píxeles que cada chip puede ver.

En la Fig. 6.20 se muestra el resultado de procesar una imagen de 256×256 píxeles. La imagen original (Fig. 6.20.(a)) se divide en 8×8 subimágenes más pequeñas, de 32×32 píxeles cada una. De este modo, cada subimagen se transforma en una secuencia de eventos AER, procesada por el chip de convolución con el kernel de la Fig. 6.18 y los eventos de

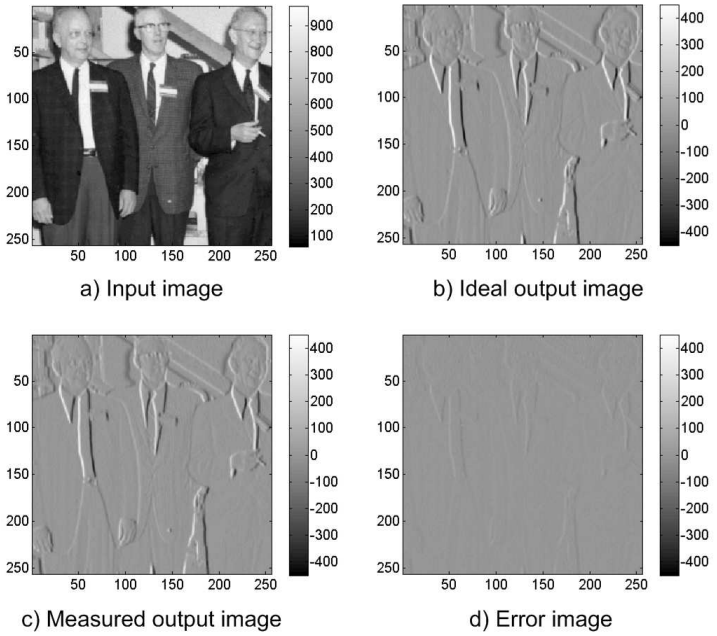


FIGURA 6.20. Resultado de calcular una convolución experimentalmente con un kernel para extracción de bordes verticales a una imagen de entrada de 256×256 píxeles.

salida se guardan. A continuación, a partir de las 64 secuencias de eventos AER grabadas se reconstruyen 64 subimágenes, y se remapean para construir la imagen de salida de 256×256 píxeles que se muestra en la Fig. 6.20.(c). Si efectuamos la convolución de forma matemática mediante MATLAB, el resultado obtenido se muestra en la Fig. 6.20.(b), mientras que la imagen error que representa la diferencia entre las frecuencias ideales y las medidas se muestra en la Fig. 6.20.(d).

6.4.3. Convolución de estímulos en movimiento

Aunque los experimentos mostrados anteriormente corresponden a imágenes estáticas, el objetivo de este chip es calcular convoluciones en tiempo real de estímulos dinámicos generados por una retina AER. Para ilustrar esto, se ha capturado una secuencia de eventos con una retina AER

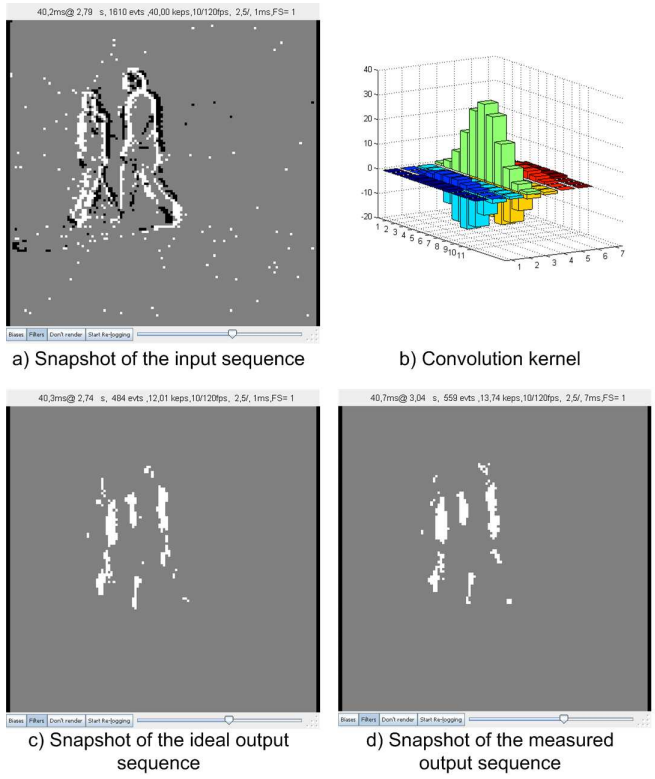


FIGURA 6.21. Resultados experimentales obtenidos procesando una secuencia de 128x128 con un kernel de Gabor para extracción de bordes verticales.

de contraste temporal de 128×128 [18], la cual muestra los contornos de dos personas caminando. Una captura de 40*ms* de esta secuencia se puede ver en la Fig. 6.21.(a), la cual representa 1810 eventos de la retina. Ya que el espacio de direcciones de la retina es de 128×128, esto requiere un array de 4×4 chips de convolución. Programando el kernel de Gabor de tamaño 11×7 de la Fig. 6.21.(b) para detección de bordes verticales, obtenemos la correspondiente secuencia de eventos de salida. Una captura de esta secuencia de salida para los mismos 40*ms* de la entrada se muestra en la Fig. 6.21.(d). Ya que la latencia entre los eventos de entrada y salida está en torno a los 150ns, se puede considerar que ambas secuencias de eventos son simultáneas.

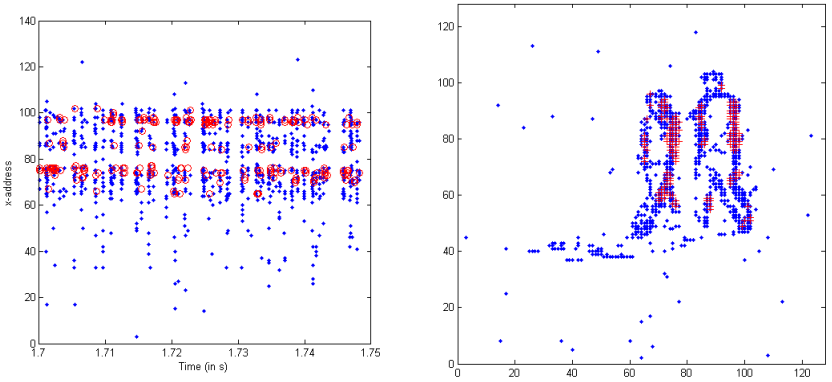


FIGURA 6.22. A la izquierda, representación de la coordenada horizontal de los eventos de entrada y salida frente al tiempo para un intervalo de 50*ms*. Los eventos de entrada se representan con puntos azules y los de salida con círculos rojos. A la derecha, reconstrucción de las imágenes de entrada y salida correspondientes al mismo intervalo, con los eventos de entrada como puntos azules y los de salida como cruces rojas.

Para compararlo con la respuesta teórica, utilizamos el mismo estímulo obtenido de la retina para procesarlo con un simulador de comportamiento AER [74], programando el mismo kernel. La secuencia AER de salida producida por este simulador es virtualmente idéntica a la obtenida experimentalmente por el chip. La Fig. 6.21.(c) muestra una captura de esta secuencia para los mismos 40*ms* de la entrada y de la salida experimental.

En la Fig. 6.22 se muestran los eventos correspondientes a un intervalo de 50*ms* de la secuencia total. En la figura de la izquierda se representa la coordenada horizontal de los eventos de entrada y salida frente al tiempo, mientras que en la figura de la derecha se muestra la reconstrucción de la imagen correspondiente a este intervalo. En ella se puede comprobar cómo los eventos de salida (cruces rojas) se superponen sobre los eventos de entrada (puntos azules), sin ningún retraso apreciable. Si seleccionamos un intervalo temporal aún más pequeño, podemos observar los resultados mostrados en la Fig. 6.23 para un tiempo de 5*ms*. En la parte izquierda podemos ver la distribución temporal de los eventos capturados por la retina en dicho intervalo (puntos azules), así como los eventos generados por el chip de

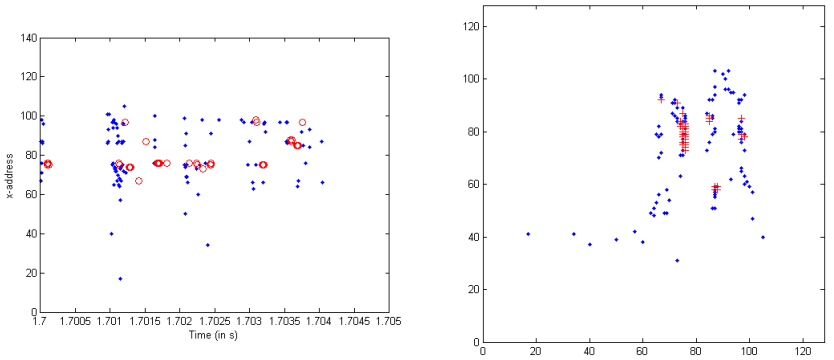


FIGURA 6.23. Representación correspondiente a un intervalo de tiempo de 5*ms*. A la izquierda, coordenada horizontal frente al tiempo, y a la derecha reconstrucción de las imágenes de entrada y salida.

convolución (círculos rojos), mientras que en la parte derecha podemos ver la reconstrucción de la imagen, con las entradas y salidas superpuestas (entradas como puntos azules, salidas como cruces rojas). Para este intervalo de tiempo tampoco se aprecia ningún retraso entre entrada y salida.

6.4.4. Discriminación de hélices rotando a alta velocidad

Para demostrar la capacidad de procesamiento a alta velocidad del chip de convolución, un experimento muy interesante es discriminar entre hélices rotando a alta velocidad [78]. Para ello, creamos una secuencia de eventos correspondientes a dos hélices con distinta forma rotando. Una de ellas tiene forma rectilínea, mientras que la otra tiene forma de *S*, tal como se ilustra en la Fig. 6.24. Ambas hélices tienen un diámetro de 16 píxeles, y rotan a una alta velocidad mientras se desplazan lentamente por la pantalla. Un observador humano sólo sería capaz de ver dos círculos completos, sin poder discriminar entre ellas.

En este experimento, la secuencia de eventos de entrada se carga en una placa USB-AER configurada como *data-player* [48], la cual envía dichos eventos al chip de convolución con la temporización correcta. Tras programar un kernel específico, estos eventos son procesados por el chip de

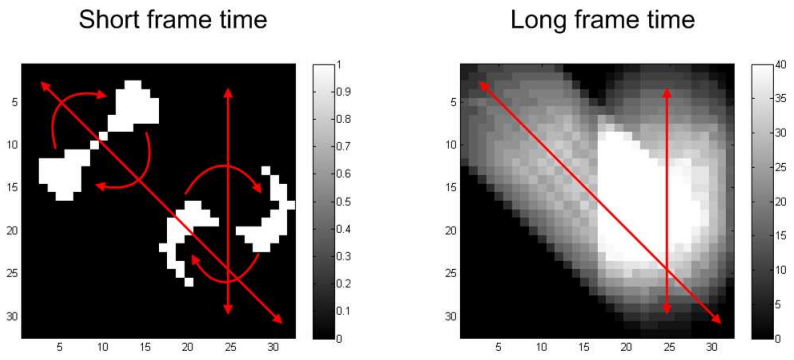
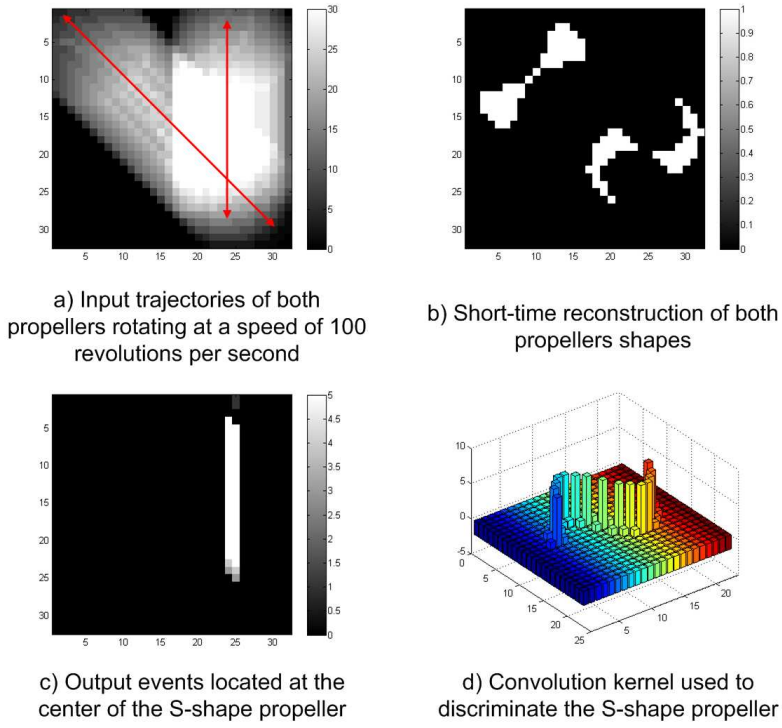


FIGURA 6.24. La imagen de la izquierda muestra las dos hélices diferentes: la rectilínea y la que tiene forma de *S*, indicando las trayectorias que siguen. La imagen de la derecha muestra lo que el ojo humano vería cuando ambas hélices rotan y se desplazan, sin ser capaz de discriminar entre ellas.

convolución y los eventos de salida se almacenan en otra placa USB-AER configurada como *data-logger* [48]. De este modo, ambas secuencias pueden ser analizada detalladamente en un ordenador. El objetivo del experimento es hacer un seguimiento del centro de la hélice en forma de S programando el kernel de 23×23 posiciones que se muestra en la Fig. 6.25.(d). Este kernel está diseñado para producir eventos positivos en el centro de la hélice cuando ésta está en posición horizontal, y todo el vecindario con pesos negativos evita que se produzcan eventos positivos en otras posiciones. Las imágenes de la Fig. 6.25.(a), (b) y (c) muestran el resultado del experimento en el cual ambas hélices dan vueltas a una velocidad de 100 rotaciones por segundo. La Fig. 6.25.(a) muestra las trayectorias completas de ambas hélices moviéndose a lo largo de la pantalla e intersectando en un punto concreto. Esto corresponde a un tiempo de captura de 1 segundo, mientras que la imagen de la Fig. 6.25.(b) corresponde a $1ms$. La Fig. 6.25.(c) muestra cómo la salida del chip de convolución sigue la trayectoria del centro de la hélice en forma de *S* mientras se mueve, utilizando el kernel de la Fig. 6.25.(d). Como se esperaba, no se producen salidas en la trayectoria de la hélice rectilínea. En este experimento, al ser la velocidad de rotación bastante baja, las tasas medias de eventos son de $85 \times 10^3 eps$ (eventos por segundo) en la entrada y solamente $500 eps$ a la salida. Una revolución de las dos hélices genera aproximadamente 850 eventos.



a) Input trajectories of both propellers rotating at a speed of 100 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 6.25. Discriminación en tiempo real de hélices rotando simultáneamente a 100 revoluciones por segundo.

La Fig. 6.26 muestra el resultado de un experimento equivalente, pero ahora con las hélices rotando a una velocidad de 2000 revoluciones por segundo. Esto produce una tasa de eventos a la entrada de 1.69*Meps*. Los tiempos de las capturas para las imágenes de la Fig. 6.26.(a) y (b) son de $50ms$ y $50\mu s$, respectivamente. La salida se muestra en la Fig. 6.26.(c), y se corresponde con una tasa de eventos de 9.5*Keps*. Ya que el kernel tiene 23 filas, procesar un único evento requiere 50 ciclos de reloj, lo que supone 417*ns* a la frecuencia de reloj de 120MHz a la que trabajamos. Procesar los 850 eventos de una sola rotación requiere al menos $355\mu s$, lo cual resulta en una velocidad de rotación máxima teórica de 2821 rps (revoluciones por segundo). Para comprobar el límite real, establecemos una velocidad de rotación en nuestra placa *data-player* superior al límite teórico. De este

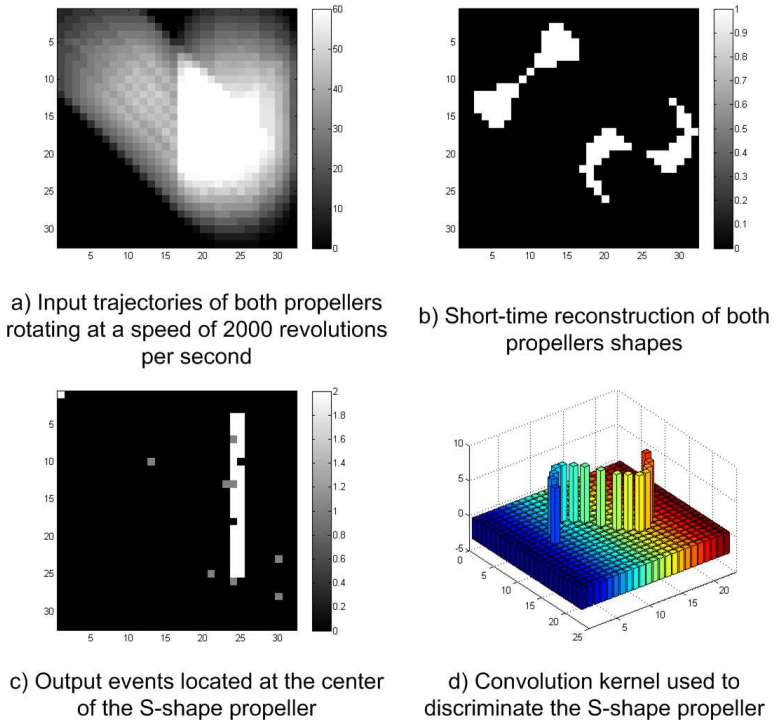
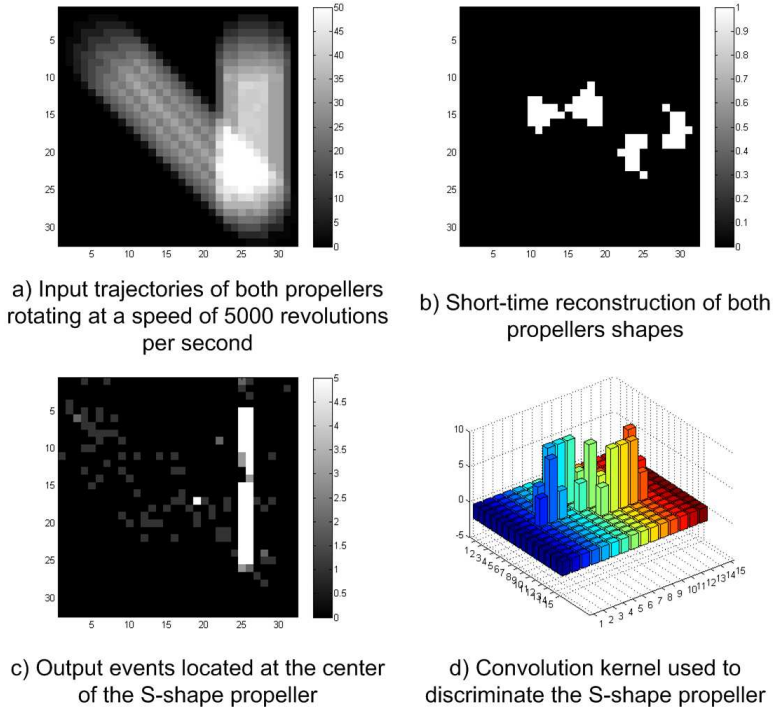


FIGURA 6.26. Disciminación en tiempo real de hélices rotando simultáneamente a 2000 revoluciones por segundo.

modo, el chip ralentizará la tasa de eventos de entrada mediante el protocolo *handshaking* hasta el límite que pueda mantener. Midiendo la tasa de eventos bajo estas circunstancias, encontramos la máxima velocidad de rotación que el chip de convolución es capaz de procesar en 2688 rps.

Con el objetivo de poder trabajar con mayores velocidades de rotación, utilizamos un nuevo par de hélices más pequeñas, con 10 píxeles de diámetro cada una, las cuales generan aproximadamente 325 eventos por cada rotación. Para estas hélices podemos usar un kernel más pequeño, de solamente 15 filas, tal como se muestra en la Fig. 6.27.(d). La Fig. 6.27 ilustra el resultado del experimento con estas hélices para una velocidad de rotación de 5000 revoluciones por segundo. Las capturas de la Fig. 6.27.(a) y (b) corresponden a unos tiempos de $20ms$ y $20\mu s$, respectivamente. A esta velocidad de rotación la tasa de eventos de entrada es de $1.62Meps$. La



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. Disciminación en tiempo real de hélices rotando simultáneamente a 5000 revoluciones por segundo.

Fig. 6.27.(c) muestra cómo la salida del chip de convolución sigue el centro de la hélice en forma de *S*, generando una tasa de eventos de salida de 19.6*Keps*. Para un kernel de 15 filas, cada evento necesita 283*ns* para ser procesado, lo que produciría una tasa de eventos de entrada máxima de 3.53*Meps*, que a su vez se corresponde con una velocidad de rotación máxima teórica de 10860 rps. Configurando la placa *data-player* para emitir rotaciones a esta velocidad, el chip de convolución lo ralentiza proporcionándonos un límite real de 9433 rps.

Estos experimentos demuestran el potencial de los sistemas de sensado y procesamiento de basados en eventos para reconocimiento de objetos a muy alta velocidad (frente a los sistemas basados en fotogramas). Es importante observar que para reconocer hélices rotando a 10000 rps en un sistema

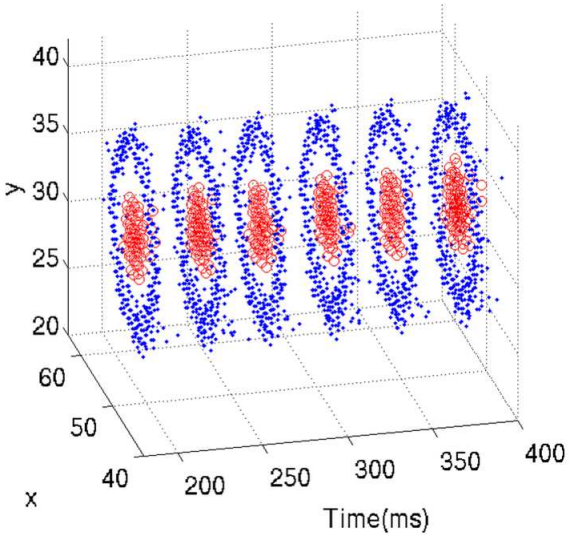


FIGURA 6.28. Representación 3D de las salidas de la retina (puntos) y las salidas del chip de convolución (círculos) expuestos al círculo de LEDs.

basado en fotogramas, sería necesario sensar y procesar las imágenes a una velocidad de al menos 100000 fotogramas por segundo.

6.4.5. Experimento para medición de latencia

Para comprobar otra de las grandes ventajas de este chip, la reducida latencia entre los eventos de entrada y los de salida, realizamos el siguiente experimento. Como entrada, usamos una secuencia de eventos capturados mediante una retina de contraste temporal cuando un círculo de diodos LEDs se enciende y apaga cada 40*ms*. Este experimento fue realizado previamente en [72] con la versión analógica del chip de convolución, y ahora podemos demostrar la importante mejora proporcionada por el chip digital propuesto en el presente trabajo.

El chip de convolución se programó con un kernel circular para detectar el centro del círculo de LEDs. Tanto los eventos de entrada como los de salida se pueden observar en la Fig. 6.28 en una representación tridimensional en la que se muestran las coordenadas (x, y) de los eventos frente al tiempo. En dicha figura se observa cómo solamente se generan eventos

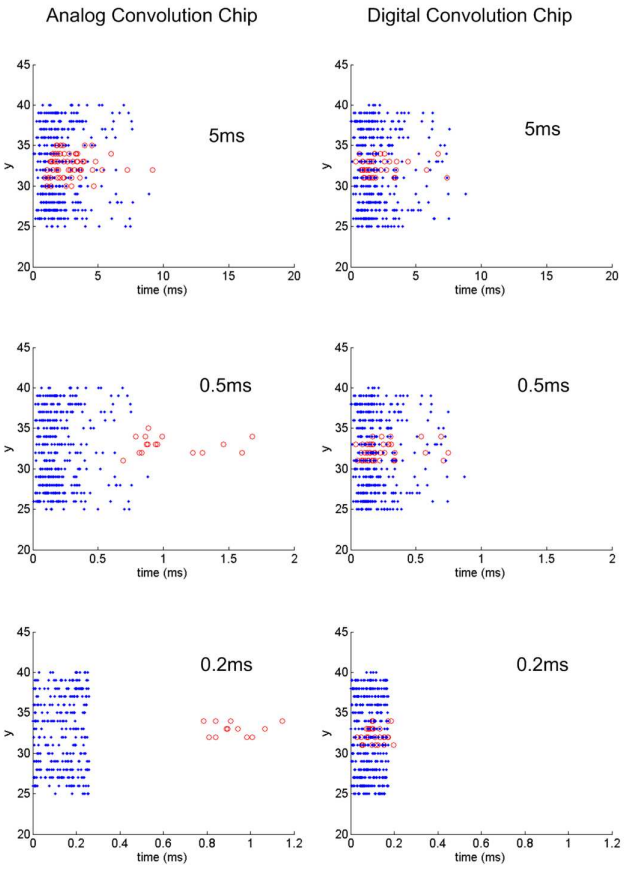


FIGURA 6.29. Resultados de procesar el círculo de LEDs y discriminar su centro, tanto con el chip analógico como con el digital. Los puntos representan los eventos generados por la retina y los círculos los generados por el chip de convolución. El eje y representa la dirección y de los eventos. Cada fila muestra los resultados para distintas duraciones de la ráfaga de entrada.

durante los transitorios de encender y apagar los LEDs. Los estímulos de la retina almacenados se pueden reproducir en una placa *data-player* acelerada a distintas velocidades.

Cada subfigura de la Fig. 6.29 muestra una proyección bidimensional de un único transitorio de encender o apagar los LEDs, representando la coordenada *y* de los eventos frente al tiempo. La columna de gráficas de la

izquierda se corresponde con los resultados obtenidos de la versión analógica del chip de convolución, mientras que la de la derecha muestra los resultados del actual chip de convolución digital. Cada fila se corresponde con una aceleración diferente aplicada a la reproducción de los eventos almacenados de la retina, siendo la fila superior "tiempo real". Cada transitorio de la retina (encenderse o apagarse los LEDs) dura unos 2*ms*, y produce unos 400 eventos distribuidos en un círculo en el plano x/y, como muestra la Fig. 6.28. Los eventos producidos por el chip de convolución, los cuales aparecen en la zona del centro del círculo, son prácticamente simultáneos a los eventos de entrada, tanto para el chip analógico como para el digital.

La segunda fila de la Fig. 6.29 corresponde a reproducir los eventos de entrada 10 veces más rápido que a tiempo real. Podemos ver cómo para el chip analógico esto provoca que los eventos de salida aparezcan ralentizados, mientras que para el chip digital el resultado es virtualmente idéntico al experimento inicial. En la tercera fila, los eventos se han acelerado un factor 25 sobre el tiempo real. Como se puede ver, en el chip analógico hay una importante latencia entre la ráfaga de eventos producida por la retina y la producida por el chip de convolución, que está en el orden de 1*ms*. Esto es debido a que el píxel analógico incluye comparadores que deciden antes de generar los eventos de salida, y estos comparadores están polarizados para bajo consumo y tienen un ancho de banda del orden de los KHz. Sin embargo, el chip de convolución digital objeto del presente trabajo produce eventos de salida tan pronto como recibe suficientes eventos de entrada para reconocer el círculo. Esto ocurre tras recibir unos 30 ó 40 eventos.

En la Fig. 6.30 se representa la comparación entre las latencias medidas entre el primer evento de la retina y el primer evento del chip de convolución para distintos valores de la duración de la ráfaga, todo ello para ambos chips. Como se puede ver, la latencia para el chip analógico se mantiene prácticamente constante (varía entre $800\mu s$ y $1ms$), mientras que para el chip digital decrece linealmente con la duración del transitorio, hasta que satura a unos $18\mu s$. Como consecuencia, podemos decir que el chip es capaz de reconocer esta forma en solo $18\mu s$.

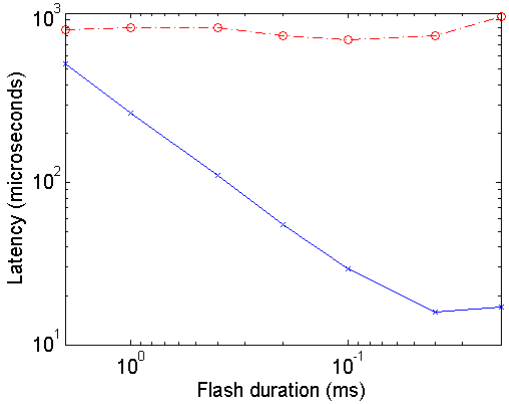


FIGURA 6.30. Comparación de las latencias medidas para los chips de convolución analógico y digital para distintas duraciones de la ráfaga de los LEDs. La línea de puntos representa el chip analógico, y la continua el digital.

6.5. Chip de convolución 64x64 Conv2

En este apartado vamos a describir los resultados experimentales obtenidos a partir del chip de convolución Conv2 de 64×64 píxeles, fabricado con área total de $5.5 \times 5.8 mm^2$ en la tecnología de AMS $0.35 \mu m$. La fotografía del chip se muestra en la Fig. 6.31. El bloque que ocupa un área mayor dentro del chip es el array de píxeles con unos $3.7 \times 3.4 mm^2$, mientras que el controlador síncrono junto con la tabla de configuración que almacena los parámetros del multikernel ocupa unos $4850 \times 650 \mu m^2$. Por otra parte, la memoria RAM de 32×32 datos de 4 bits ocupa unos $450 \times 3700 \mu m^2$ y el bloque de desplazamiento horizontal unos $650 \times 3800 \mu m^2$. Los demás bloques, como el generador AER, el inversor de complemento a 2 o el generador de reloj, consumen mucha menos área.

A continuación vamos a mostrar los resultados de la caracterización del chip.

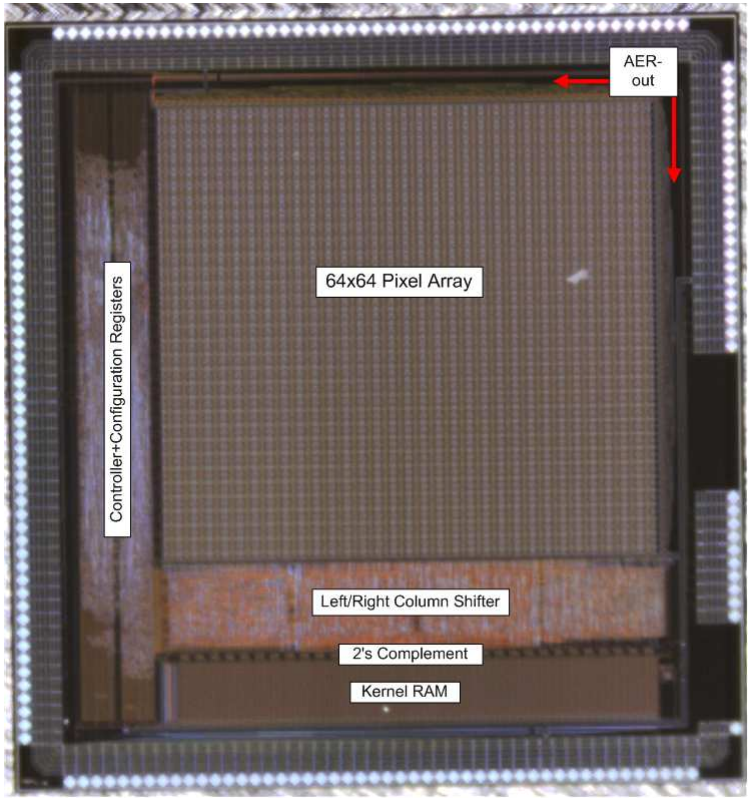


FIGURA 6.31. Fotografía del chip de convolución Conv2.

6.5.1. Caracterización del chip

6.5.1.1. Reloj interno

Al igual que en la versión anterior, la frecuencia del reloj interno se puede ajustar mediante la tensión de polarización V_{bias_clk}. Del proceso de caracterización se ha observado que para obtener un comportamiento preciso sin pérdida de eventos es recomendable utilizar una frecuencia máxima de 100MHz. La principal limitación que nos encontramos al incrementar dicha frecuencia se debe al retraso de los datos de la RAM desde su lectura hasta que se encuentran a disposición de los píxeles del array, como se

indicó en la Sección 5.5. Así pues, bajo ciertas circunstancias en las que trabajamos con un kernel pequeño situado en la zona central de la RAM es posible incrementar la frecuencia de reloj de forma importante, llegando a observarse un comportamiento bastante aceptable hasta los 200MHz. No obstante, para cualquier aplicación en general trabajaremos con un reloj de 100MHz.

6.5.1.2. Consumo de potencia

En lo referente al consumo de potencia del chip, igual que el caso anterior también depende de la tasa de eventos de entrada y del tamaño del kernel. Así pues, tomamos una tasa de entrada de 5*Meps* (lo que se corresponde con un tiempo entre eventos de 200*ns*), y observamos que para el kernel más pequeño posible (de tamaño 1×1) tenemos un consumo de 132*mW*, mientras que para el mayor kernel posible (de tamaño 32×32) el consumo es de 198*mW*.

6.5.1.3. Caracterización temporal

En cuanto al comportamiento temporal del chip, tiene la capacidad de generar eventos de salida con una tasa máxima de 37×10^6 *eps*, medidos con las señales *Rqst_out* y *Ack_out* cortocircuitadas, y para eventos generados por píxeles de una misma fila (modo ráfaga). Esta tasa se corresponde con un tiempo entre eventos de $T_{rafaga} = 27ns$. Para píxeles pertenecientes a filas diferentes, medimos un tiempo entre eventos de $T_{no-rafaga} = 60ns$. En general, estos tiempos son ligeramente más grandes que los observados en Conv1, lo cual se debe al mayor tamaño de la nueva versión. Concretamente los retrasos del bloque generador AER se ven incrementados a causa de haber pasado de 32 a 64 entradas en los arbitradores por filas y por columnas, lo que implica un nivel más en los árboles de arbitración.

En la Fig. 6.32 se pueden ver las señales medidas *Rqst_in*, *Ack_in* y *Rqst_out* (que para estas pruebas está cortocircuitada con *Ack_out*) en un experimento con el mismo kernel de 5 filas de la Fig. 6.14, con los tiempos indicados entre eventos de salida, que varían para píxeles de una misma fila y de filas distintas. A partir de ambos datos, podemos deducir que el retraso

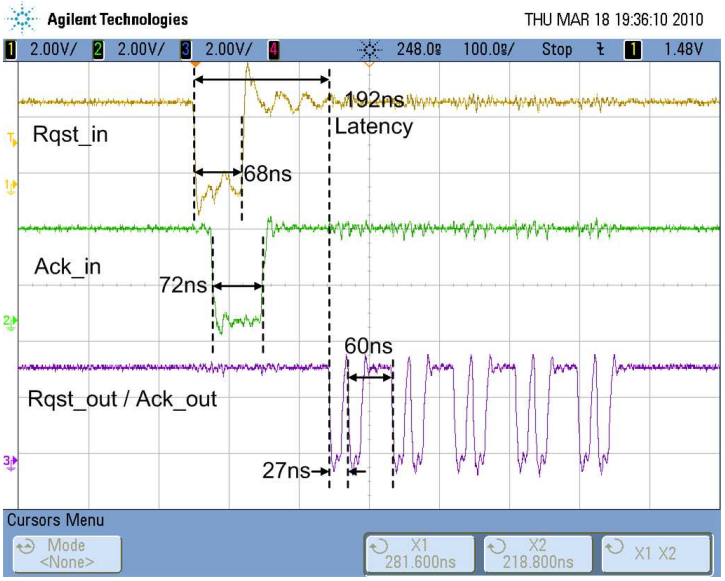


FIGURA 6.32. Señales Rqst y Ack medidas para los eventos de entrada y salida, para una frecuencia de reloj de 100MHz, con Rqst_out y Ack_out cortocircuitadas.

introducido por el arbitrador por filas vendrá dado por la diferencia entre ambos $T_{arb} = 60ns - 27ns = 33ns$.

La tasa de eventos a la entrada depende, al igual que para la versión anterior, tanto del tamaño del kernel como de la frecuencia de reloj, según la misma expresión que indica que el controlador necesita $n_{clk} = 4 + (2 \times n_k)$ ciclos de reloj para procesar cada evento, con n_k el número de filas del kernel (hasta un máximo de 32). Así pues, para la que hemos considerado frecuencia de trabajo de 100MHz (que se corresponde con un periodo de reloj de 10*ns*), la máxima tasa de eventos de entrada posible es de 16.6×10^6 *eps*, lo cual se corresponde con un tiempo entre eventos de 60*ns* para un kernel de una sola fila. En el caso de un kernel del máximo posible de filas (32), la máxima tasa de eventos a la entrada es de 1.47×10^6 *eps*, o lo que es lo mismo un tiempo entre eventos de 680*ns*.

Para medir el tiempo de latencia entre un evento de entrada y un evento de salida, recurrimos a la misma división mostrada en la Fig. 6.15, en la

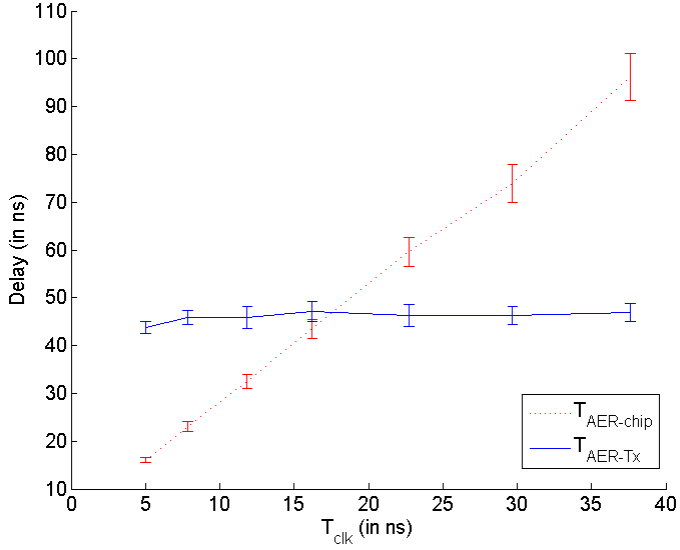


FIGURA 6.33. Valores medidos de $T_{AER-chip}$ y T_{AER-Tx} para distintos periodos de reloj

cual representamos el tiempo como $T_{latency} = T_{AER-chip} + T_{AER-Tx} + T_{proc-event}$. En la Fig. 6.33 representamos los valores medidos de $T_{AER-chip}$ y T_{AER-Tx} para distintos valores del periodo de reloj T_{clk}. Como se puede ver, el retraso T_{AER-Tx} es constante (no depende del periodo de reloj) igual que en el caso anterior, y resulta un mismo valor medio de unos 44*ns*, debido a que hemos usado la misma placa emisora que para caracterizar Conv1. Igualmente, $T_{AER-chip}$ depende de T_{clk} de forma lineal, con un valor residual calculado extrapolando estos resultados de unos 4*ns* para $T_{clk} = 0$.

Por otra parte, la Fig. 6.34 muestra los valores medidos de $T_{latency}$ frente al periodo de reloj. El ajuste lineal de los datos experimentales indica que para un periodo de reloj $T_{clk} = 0$ tendríamos una latencia de 102*ns*. Este valor se correspondería con $T_{proc-event} = T_{asyn}$, ya que suponemos una situación ideal en que la parte síncrona de procesamiento se lleve a cabo en un tiempo 0. Extrapolando también los valores de $T_{AER-chip}$ y T_{AER-Tx} para esta misma situación ideal, podemos estimar un valor del tiempo de

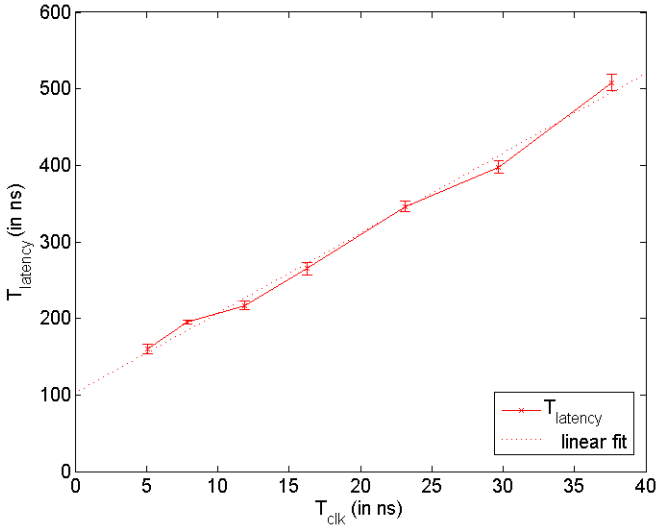


FIGURA 6.34. Valores medidos de $T_{latency}$ para distintos periodos de reloj. La línea de puntos representa el ajuste lineal ($y = 10x + 102$).

procesamiento asíncrono como $T_{asyn} = 102ns - 44ns - 4ns = 54ns$, lo cual es coherente con el valor experimental de 60*ns* medido entre dos eventos de salida pertenecientes a filas diferentes. Para una frecuencia de reloj de 100MHz, medimos un tiempo de latencia de 192*ns*.

Igual que con la versión anterior, al conectar dos chips de convolución Conv2 en cascada, el retraso que hemos llamado T_{AER-Tx} sería reemplazado o bien por $T_{rafaga} = 27ns$, o bien por $T_{no-rafaga} = 60ns$. De este modo, la auténtica latencia mínima al conectar los chips en cascada vendría dada por $T_{latency} = 192ns - 44ns + 27ns = 175ns$.

En la Tabla 6.2 se encuentran resumidas las especificaciones del chip de convolución Conv2.

TABLA 6.2. Especificaciones del chip Conv2.

Tecnología	4M 2P $0.35\mu m$ CMOS
Tamaño del píxel	$58.0 \times 53.8\mu m^2$
Tamaño del chip	$5.5 \times 5.8mm^2$
Array de píxeles	64×64
Resolución del píxel	6 bits
Resolución del kernel	4 bits
Computación con signo	Sí
Sistema multikernel	Sí
Inhibición de eventos	Sí
Tasa de eventos de entrada	1.47-16.6 Meps
Máxima tasa de salida	37 Meps
Mínima latencia entrada-salida	175ns
Consumo de potencia	200mW máximo

CAPÍTULO 7

Conclusiones y trabajos futuros

A lo largo de este trabajo hemos presentado el diseño de unos microchips convolucionadores para procesado asíncrono de visión en tiempo real, siguiendo el protocolo de comunicación AER (*Address Event Representation*). Estos convolucionadores no operan sobre fotogramas, sino que lo hacen sobre codificaciones por eventos de la información sensorial. Los sistemas de procesamiento por eventos tienen una serie de ventajas sobre los basados en fotogramas, como se ha descrito en este documento, siendo la principal la velocidad de procesamiento. Al codificar en eventos la información del estado de los píxeles, se consigue que la información más importante se procese en primer lugar, obteniendo resultados en un tiempo muy reducido.

Los microchips descritos en esta tesis implementan la convolución de la imagen de entrada con kernels programables de forma y tamaño arbitrario en tiempo real. Gracias a la programabilidad, estos chips están ideados para ser el elemento básico de sistemas complejos de procesamiento multicapa bio-inspirados de alta velocidad. De este modo, mediante la interconexión en cascada y en paralelo de estos convolucionadores programados convenientemente se pueden implementar sistemas de reconocimiento y seguimiento de formas geométricas, segmentación de imágenes, reconocimiento de caracteres, ...

Así pues, las principales contribuciones de esta tesis son las siguientes:

1. Dos versiones diferentes de un nuevo píxel I&F (*integrate and fire*) completamente digital para calcular convoluciones. Este píxel digital resuelve las principales limitaciones presentadas por otros píxeles analógicos [78], como eran la necesidad de calibración para compensar el *mismatching*, la reducida resolución (3 bits) y la alta latencia (1*ms*). El nuevo píxel, al ser completamente digital, no necesita circuitería de calibración (lo cual permite reducir el área ocupada y construir arrays con mayor cantidad de píxeles), permite hasta 18 bits de resolución y una latencia tan reducida como 150*ns*. Estos píxeles están diseñados para procesar pesos de entrada positivos y negativos, y generar eventos de salida de ambos signos.
2. Capacidad de inhibición de eventos a nivel de píxel. Esta propiedad permite seleccionar en cada aplicación la posibilidad de inhibir los eventos generados por el píxel en función del signo. De este modo, si por ejemplo los eventos negativos producidos por los píxeles no aportan información a la siguiente capa de procesamiento, no es necesario consumir ancho de banda de comunicación entre los chips, ya que el propio píxel descarta el evento directamente.
3. Computación del inverso de los pesos del kernel en complemento a 2, para procesar eventos de entrada de signo negativo. Con el diseño del bloque inversor se consigue invertir el signo del kernel en el instante de procesarlo con un retraso del orden de los 400*ps*, con el objetivo de no limitar la frecuencia de funcionamiento de nuestro sistema.
4. Implementación del mecanismo de olvido de frecuencia programable, que se encarga de decrementar el estado de los píxeles de forma periódica para permitir la detección de correlación temporal entre los eventos de entrada.
5. Implementación del sistema multikernel. Proponemos la capacidad de programar hasta un total de 32 kernels diferentes en un mismo chip de convolución para procesar cada evento de entrada con el kernel correspondiente en función de la procedencia de dicho evento. Para implementar este sistema se incluye una tabla de configuración donde se programa la posición de cada kernel dentro de la RAM interna, de forma que el controlador síncrono pueda procesar cada evento de entrada con el kernel que le corresponda. Por ese motivo se incluye también un circuito de

bloqueo de columnas a la salida de la RAM para que solamente las posiciones correspondientes al kernel en cuestión sean procesadas.

Con esto, se han fabricado y testado dos chips de convolución: una primera versión con 32×32 píxeles de una resolución máxima de 18 bits, y una segunda versión con 64×64 píxeles de 6 bits de resolución. Ambas versiones están diseñadas para construir arrays de varios chips en paralelo que se comporten como un único chip de mayor tamaño. En el Capítulo 6 se muestran exhaustivos resultados experimentales obtenidos de los dos chips.

El trabajo futuro estará orientado hacia la implementación de sistemas multi-chip cada vez más complejos, para lo cual se marcan las siguientes líneas:

1. Desarrollo de una infraestructura modular y escalable de placas con comunicación AER serie con conectores de tamaño reducido, que permita ensamblar varias decenas de módulos convolucionadores.
2. Adaptación de técnicas de reconocimiento visual de objetos basadas en los sistemas software actuales conocidos como "*Convolutional Neural Networks*" a la infraestructura AER hardware.
3. Desarrollo de técnicas de aprendizaje específicas para "*Spiking Neural Networks*", y su adaptación e implementación al hardware convolucionador AER.
4. Aplicación de todo ello a sistemas reales que requieran percepción visual eficiente y a alta velocidad, tales como conducción automática de vehículos, aplicaciones de robótica en ambientes no estructurados, o vigilancia inteligente en sistemas de seguridad, por ejemplo.

Referencias

[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.

Lista de publicaciones

Artículos en revistas:

[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.

Contribuciones a congresos:

[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.