



Tesis doctoral

Evaluación y análisis de una aproximación a la fusión sensorial neuronal mediante el uso de sensores pulsantes de visión / audio y redes neuronales de convolución.

José Antonio Ríos Navarro
Sevilla, Mayo de 2017

Departamento de Arquitectura y Tecnología de Computadores
Escuela Técnica Superior de Ingeniería Informática
Universidad de Sevilla

Evaluación y análisis de una aproximación a la fusión sensorial neuronal mediante el uso de sensores pulsantes de visión / audio y redes neuronales de convolución.

por
José Antonio Ríos Navarro

PROPUESTA DE TESIS DOCTORAL
PARA LA OBTENCIÓN DEL GRADO DE
DOCTOR INGENIERO EN INFORMÁTICA
SEVILLA, MAYO DE 2017

Directores:
Dr. Alejandro Linares Barranco
Dr. Gabriel Jiménez Moreno
Dr. Ángel Jiménez Fernández

UNIVERSIDAD DE SEVILLA

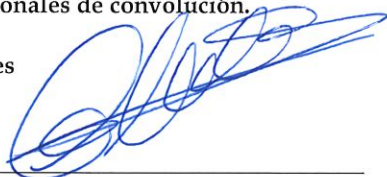
Memoria presentada para optar al grado de Doctor Ingeniero en Informática por la Universidad de Sevilla.

Autor: José Antonio Ríos Navarro

Título: Evaluación y análisis de una aproximación a la fusión sensorial neuronal mediante el uso de sensores pulsantes de visión / audio y redes neuronales de convolución.

Departamento: Arquitectura y Tecnología de Computadores

Vº Bº Directores



Dr. Alejandro Linares Barranco

Profesor Titular de Universidad



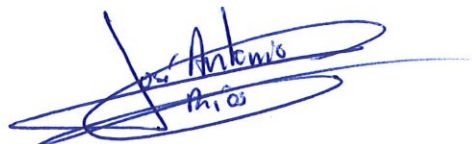
Dr. Gabriel Jiménez Moreno

Profesor Titular de Universidad



Dr. Ángel Jiménez Fernández

Profesor Interino Contratado Doctor



El autor

José Antonio Ríos Navarro

Ingeniero Técnico en Informática de Sistema

Agradecimientos

Este trabajo no hubiese sido posible sin la implicación, colaboración y apoyo de muchas personas, por lo que me gustaría que estas líneas sirvan para agradecerles su motivación y orientación en el desarrollo del trabajo presentado en esta tesis. En especial me gustaría hacer llegar mi más sincero y profundo agradecimiento a:

- Mi familia, que ha tenido que lidiar con todos los malos momentos, agradezco su continuo apoyo y constancia, sin los cuales no hubiera tenido la fuerza para terminar este reto. Gracias por la educación que me habéis prestado, con la cual he podido llegar donde ahora mismo me encuentro. En especial a mis padres, que desde muy pequeño me han inculcado ese espíritu de ser constante y no conformista, necesarios en el trabajo, para conseguir los objetivos. A ellos va dedicado este trabajo.
- A mis directores de tesis, Alejandro, Gabriel y Ángel, pues son los que me han guiado en todo momento y han dedicado incontables horas en aconsejarme y guiarme en los problemas encontrados en la elaboración de esta tesis, además de haber sido un pilar fundamental en mi formación científica e investigadora.
- A todos los miembros del departamento de Arquitectura y Tecnología de computadores, de los que he aprendido mucho y con lo que he disfrutado tanto trabajando como en momentos inolvidables de ocio. En especial a ese núcleo joven, Juan Pedro, Ricardo, Daniel, Ángel, Manu y Elena, por su contagiosa ilusión y ganar de trabajar.
- De manera particular, se merece un párrafo uno de mis directores de tesis, Alejandro, que me abrió las puertas de esta gran familia y siempre ha confiado y creído en mí.
- A mis amigos más próximos, cuyo apoyo y ánimo me ha ayudado a seguir adelante en la realización de este trabajo.

A todos aquellos que no han sido nombrados expresamente en los párrafos anteriores y que siempre han estado ahí, gracias.

Índice de Contenido

Agradecimientos	i
Índice de Contenido	iii
Índice de Figuras	vii
Índice de Ecuaciones.....	xi
Índice de Tablas.....	xiii
Resumen	1
Abstract.....	3
I. Introducción.....	5
I.1 Motivación	6
I.2 Ingeniería neuromórfica.....	8
I.2.1 Inicios y evolución	9
I.2.2 Sistemas neuro-inspirados	10
I.2.3 Address-Event-Representation	14
I.3 Visión artificial	17
I.3.1 Adquisición de una imagen digital.....	19
I.3.2 La retina de silicio	21
I.4 Procesamiento de audio. El habla	24
I.4.1 Conversión analógico-digital.....	26
I.4.2 Sistemas de audio neuromórficos	28
I.5 Redes neuronales de aprendizaje profundo	32
I.5.1 Breve historia sobre DNN y sus aplicaciones.....	32
I.6 Estructura de la tesis.....	34
II. Objetivos	35
III. Visión con computación paralela	37
III.1 Capas de procesado de una CNN	38

III.2.1 Capa de convolución.....	38
III.2.2 Capa de pooling.....	39
III.2.3 Capa dropout	40
III.2.4 Capa fully-conected.....	41
III.1 OpenCL.....	41
III.1.1 Modelo de plataforma.....	41
III.1.2 Modelo de memoria	42
III.1.2 Modelos de ejecución.....	44
III.2 Implementación de CNN utilizando OpenCL.....	46
III.2.1 Optimización de los kernels y ejecución la aplicación.....	48
III.2.3 Resultados	49
III.3 Aceleradores de CNN (NullHop).....	50
III.3.1 NullHop.....	51
III.3.2 Implementación del NullHop en SoC.....	57
III.3.3 Resultados	61
IV. Integración sensorial.....	73
IV.1 Tipos de integración sensorial	74
IV.2 Clasificación auditiva.....	75
IV.3 Aplicación práctica y resultados experimentales.....	81
V. Fusión sensorial	87
V.1 Generación de los datos de entrada	88
V.1.1 Generación de datos del DVS128.....	91
V.1.2 Generación de datos del NAS	92
V.2 Modelo CNN para la fusión sensorial audiovisual.....	95
V.3 Resultados	98
VI. Aportaciones más importantes y conclusiones	103
VII. Trabajo futuro.....	105

VIII. Bibliografía	107
V. Anexo	115
OKAERTool. Herramienta de depuración y testeo de sistemas neuromórficos	115

Índice de Figuras

Figura 1. Reproducción de un dibujo de Ramón y Cajal que muestra algunas neuronas del córtex de un mamífero.	11
Figura 2. Esquema básico de una neurona.	11
Figura 3. Diagrama de un pulso eléctrico, o spike, generado por una neurona.....	12
Figura 4. Vista transversal de las células del córtex en capas con sus dendritas en la materia gris y los axones proyectando hacia la materia blanca.	15
Figura 5. Multiplexado de diferentes neuronas en un único canal de comunicación	16
Figura 6. Demultiplexación de una entrada AER en spikes individuales para cada dendrita	17
Figura 7. Izquierda, primera fotografía que se conoce. Derecha, primera fotografía donde aparece una persona.....	18
Figura 8. Izquierda, imagen captada por un telescopio. Derecha, primera radiografía que se conoce.....	18
Figura 9. Sistema de conducción autónomo de Tesla	19
Figura 10. Imagen con diversos niveles de gris	21
Figura 11. Diagrama de una sección transversal de la retina.....	22
Figura 12. Representación en jAER de la salida de un sensor DVS frente a una mano en movimiento	24
Figura 13. Primera patente de un teléfono por Alexander Graham Bell.....	25
Figura 14. Muestreo en el dominio del tiempo	27
Figura 15. Ejemplo de las características de entrada-salida de una cuantificación de 8 niveles	28
Figura 16. El oído humano	29
Figura 17. Representación de la salida del NAS.....	30
Figura 18. (a) Arquitectura general del NAS. (b) Banco de filtros dispuestos en cascada, CFB. (c) Etapa básica del CFB.....	31
Figura 19. Ejemplo de operación max-pooling.....	40
Figura 20. Ejemplo de operación avg-pooling	40
Figura 21. Esquema general del modelo de plataforma OpenCL (Xilinx 2016b)	42
Figura 22. Esquema del modelo de memoria OpenCL (Xilinx 2016b).....	43
Figura 23. Modelo de ejecución unidimensional (Xilinx 2016a)	45
Figura 24. Module de ejecución bidimensional (Xilinx 2016a)	45
Figura 25. Modelo de ejecución tridimensional (Xilinx 2016a).....	46
Figura 26. Arquitectura CNN LeNet5.....	47

Figura 27. Diagrama de ejecución de los kernels en el dispositivo hardware (Tapiador et al. 2017)	49
Figura 28. Porcentaje de zeros en cada capa de la CNN GoogleNet (arriba) y la VGG19 (abajo) (Aimar et al. 2017).....	52
Figura 29. Esquema de global del acelerador de CNN NullHop (Aimar et al. 2017)	53
Figura 30. Esquema de compresión utilizando mapa de dispersión (Aimar et al. 2017)	53
Figura 31. Formato de las palabras enviadas al acelerador (Aimar et al. 2017).....	54
Figura 32. Secuenciación de los datos de entrada/salida por filas en un acelerador de CNN	56
Figura 33. Diagrama de bloques del escenario de test implementado en el SoC	59
Figura 34. Resultado de la estimación de consumo del NullHop utilizando la herramienta XPower de Vivado.....	60
Figura 35. Esquema temporal del procesamiento de un frame en la plataforma Zynq para la red RoShambo.....	63
Figura 36. Esquema del módulo hardware de medición de latencias	64
Figura 37. Captura de la señal nh_idle y mm2s_latency en el osciloscopio para la computación de un frame.....	65
Figura 38. Captura de la señal nh_idle y s2mm_latency en el osciloscopio para la computación de un frame.....	65
Figura 39. Transferencias de datos PS → PL y viceversa	66
Figura 40. Transferencias de escritura de la capa 1	67
Figura 41. Transferencias de escritura de la capa 2	67
Figura 42. Transferencias de escritura de la capa 3	68
Figura 43. Transferencias de escritura de la capa 4	68
Figura 44. Transferencias de escritura de la capa 5	69
Figura 45. Transferencias de lectura d fijadas a 1KB/transfer.....	69
Figura 46. Transferencias de datos PS → PL y viceversa. Implementación de doble buffer ..	71
Figura 47. Arquitectura multi-core HullHop.....	71
Figura 48. Plataforma Zynq_Dock. Interfaz entre la plataforma Zynq 7100 MMP y AERNode	72
Figura 49. Diagrama de bloques de un filtro paso de banda del NAS (Cerezuela Escudero 2015)	76
Figura 50. Valores para normalizar los filtros de spikes de cada una las bandas, arriba para la cóclea izquierda, abajo para la cóclea derecha (Cerezuela Escudero 2015)	77
Figura 51. Diagramas de bode del banco de filtros. Arriba NAS no normalizado, abajo NAS normalizado	78
Figura 52. Arquitectura de una neurona (Cerezuela Escudero 2015)	80

Figura 53. Estructura de una red de neuronas de convolución mono-dimensional.....	81
Figura 54. Elementos intervinientes en la demostración del sistema de integración sensorial	82
Figura 55. Estimaciones de las rpm del motor haciendo uso de la información de la DVS128	83
Figura 56. Salida de los diferentes canales de salida del NAS para diferentes velocidades...84	
Figura 57. Salida del sistema para las diferentes velocidades	85
Figura 58. Salida AER del DVS128	89
Figura 59. Salida AER del NAS.....	90
Figura 60. Muestras del resultado de los diferentes métodos de integración para la visión. (a) full-integration, (b)unsigned-integration, (c)half-rectified positive, (d)half-rectified negative	92
Figura 61. Muestras del resultado de los diferentes métodos de integración del audio. (a) full-integration, (b)unsigned-integration, (c)half-rectified positive, (d)half-rectified negative	95
Figura 62. Modelo CNN de fusión sensorial de audio y visión.....	96
Figura 63. Accuracy y loss siguiendo la estrategia de entrenamiento número 1	99
Figura 64. Accuracy y loss siguiendo la estrategia de entrenamiento número 2	100
Figura 65. Accuracy y loss siguiendo la estrategia de entrenamiento número 3	100
Figura 66. Dato de visión alterado y dato de audio sin alterar para el dígito 5.....	101
Figura 67. Dato de visión sin alterar y dato de audio alterado para el dígito 5	102
Figura 68. Ambos datos, visión y audio, alterados para el dígito 5.....	102
Figura 69. Plataforma AERNode de procesamiento basado en spikes con conexión LVDS.117	
Figura 70. Diagrama de bloques de la plataforma Opal Kelly XEM6010-LX150-2FGG (Opal Kelly 2014)	118
Figura 71. Plataforma OKAERTool. Placa de interconexión AER-Node-OKAERTool junto con la placa Opal Kelly XEM6010-LX150-2FGG.....	118
Figura 72. Diagrama lógico de bloques de la plataforma OKAERTool.....	119
Figura 73. Máquina de estados finita del módulo merger	120
Figura 74. Máquina finita de estados del módulo monitor.....	121
Figura 75. Primera máquina de estados del módulo player	122
Figura 76. Segunda máquina de estados del módulo player	123
Figura 77. Máquina de estados del módulo logger.....	124
Figura 78. Interfaz de control de la herramienta OKAERTool en jAER.....	125

Índice de Ecuaciones

Ecuación 1. Función asociada a una imagen de intensidad	20
Ecuación 2. Cálculo del número de bits necesarios para almacenar una imagen	21
Ecuación 3. Selección de los parámetros Δ y B para la cuantificación	27
Ecuación 4. Error de cuantificación	27
Ecuación 5. Función completa de la capa de convolución	38
Ecuación 6. Función de convolución por cada feature-map	39
Ecuación 7. Función no lineal de activación ReLU	47
Ecuación 8. Cálculo del rendimiento máximo del NullHop en Gops	63
Ecuación 9. Divisor de spikes	76
Ecuación 10. Operación mono-dimensional de convolución de una neurona	79
Ecuación 11. Condición de generación de la salida de una neurona	79
Ecuación 12. Cálculo del error cometido por el sistema al realizar una estimación	85
Ecuación 13. Número de muestras por sujeto para la BBDD de dígitos hablados	93
Ecuación 14. Función "inv" para modificar la tasa de entrenamiento	99
Ecuación 15. Función "sigmoid" para modificar la tasa de entrenamiento	99

Índice de Tablas

Tabla 1. Comparativa cualitativa entre un computador y un sistema neuronal	14
Tabla 2. Asignación de capas de la CNN LeNet5 a los kernels OpenCL	48
Tabla 3. Resultados de la ejecución y consumo hardware de la CNN LeNet5 OpenCL.....	50
Tabla 4. Comparativa de NullHop con otros aceleradores de CNN.....	56
Tabla 5. Consumo de recursos de la Zynq 7100 necesarios para la implementación del NullHop.....	59
Tabla 6. Mediciones de consumo real de la plataforma Zynq 7100 MMP.....	61
Tabla 7. Configuración de la red RoShambo.....	62
Tabla 8. Parámetros de entrenamiento para las diferentes estrategias utilizadas.....	98
Tabla 9. Máximos y mínimos de accuracy / loss respectivamente según estrategia de entrenamiento y método de integración de frame	101
Tabla 10. Accuracy de la robustez del modelo de fusión para cada método de integración de frames.....	102
Tabla 11. Comparación entre varias herramientas para la depuración de sistemas AER.....	116

Resumen

En este trabajo se pretende avanzar en el conocimiento y posibles implementaciones hardware de los mecanismos de *Deep Learning*, así como el uso de la fusión sensorial de forma eficiente utilizando dichos mecanismos. Para empezar, se realiza un análisis y estudio de los lenguajes de programación paralela actuales, así como de los mecanismos de *Deep Learning* para la fusión sensorial de visión y audio utilizando sensores neuromórficos para el uso en plataformas de FPGA.

A partir de estos estudios, se proponen en primer lugar soluciones implementadas en OpenCL así como en hardware dedicado, descrito en systemverilog, para la aceleración de algoritmos de *Deep Learning* comenzando con el uso de un sensor de visión como entrada. Se analizan los resultados y se realiza una comparativa entre ellos.

A continuación se añade un sensor de audio y se proponen mecanismos estadísticos clásicos, que sin ofrecer capacidad de aprendizaje, permiten integrar la información de ambos sensores, analizando los resultados obtenidos junto con sus limitaciones. Como colofón de este trabajo, para dotar al sistema de la capacidad de aprendizaje, se utilizan mecanismos de *Deep Learning*, en particular las CNN¹, para fusionar la información audiovisual y entrenar el modelo para desarrollar una tarea específica.

Al final se evalúa el rendimiento y eficiencia de dichos mecanismos obteniendo conclusiones y unas proposiciones de mejora que se dejarán indicadas para ser implementadas como trabajos futuros.

¹ Convolutional Neural Networks

Abstract

In this work it is intended to advance on the knowledge and possible hardware implementations of the Deep Learning mechanisms, as well as on the use of sensory fusion efficiently using such mechanisms. At the beginning, it is performed an analysis and study of the current parallel programming, furthermore of the Deep Learning mechanisms for audiovisual sensory fusion using neuromorphic sensor on FPGA platforms.

Based on these studies, first of all it is proposed solution implemented on OpenCL as well as dedicated hardware, described on systemverilog, for the acceleration of Deep Learning algorithms, starting with the use of a vision sensor as input. The results are analysed and a comparison between them has been made.

Next, an audio sensor is added and classic statistical mechanisms are proposed, which, without providing learning capacity, allow the integration of information from both sensors, analysing the results obtained along with their limitations.

Finally, in order to provide the system with learning capacity, Deep Learning mechanisms, in particular CNN, are used to merge audiovisual information and train the model to develop a specific task.

In the end, the performance and efficiency of these mechanisms have been evaluated, obtaining conclusions and proposing improvements that will be indicated to be implemented as future works.

I. Introducción

Desde los principios del mundo que conocemos, el ser humano ha tratado de resolver de manera artificial los diferentes problemas que se ha ido encontrado en el camino de su progreso. Para ello, se ha fijado en cómo esos problemas que ha experimentado están resueltos en la naturaleza que nos rodea. La inquietud que el ser humano ha mostrado siempre por intentar conocer lo desconocido le ha llevado a ingeniar soluciones para resolver el problema de sobrevivir en medios en los cuales no está naturalmente adaptado para ello. Desde la navegación, que se conoce desde la más remota antigüedad, pasando por el primer submarino navegable en 1620 de Cornelius Drebbel (BBC), hasta el primer vuelo a motor en 1903 de los hermanos Wright (Bede & Collum 2002), el ser humano no ha dejado de evolucionar controlando el medio que le rodea.

Paralelamente a este progreso, se ha desarrollado la capacidad de crear nuevas máquinas que permitan resolver problemas cada vez más complejos, con mayor precisión y cada vez de manera más eficiente. Parte de aquí la idea de dotar de cierta autonomía e inteligencia a estos nuevos sistemas para que sean capaces de automatizar tareas sin necesidad de la intervención del ser humano. Esta nueva investigación abre nuevos campos de estudio sobre la forma de tratar la información que captamos sobre el entorno que nos rodea mediante diferentes sensores.

A lo largo del progreso tecnológico hemos podido ir viendo la adaptación de cada uno de estos avances en el campo de la industria, con el fin de crear robots que realicen diversas tareas peligrosas, repetitivas o de alta precisión que no serían posibles o rentables si un ser humano las realizara. Pero hoy día, y cada vez con más frecuencia, podemos ver que los dispositivos tecnológicos que adquirimos y utilizamos en nuestro día a día poseen cierta inteligencia que nos ayudan o facilitan muchas de las tareas o acciones que antes no hacían, por ejemplo, cómo una cámara fotográfica convencional es capaz de enfocar automáticamente la imagen al sujeto que se fotografía o cómo nuestro teléfono móvil es capaz de cancelar parte del ruido ambiente para que nuestra voz se oiga mejor al otro lado de la conversación.

Es en la historia más reciente de la inteligencia artificial cuando nace un nuevo paradigma de aprendizaje y procesamiento automático, inspirado en el funcionamiento del sistema nervioso. En inteligencia artificial se les conoce como redes neuronales, y se trata de un sistema de interconexión de neuronas que procesan la información de manera colaborativa entre ellas para producir unos estímulos en forma de resultado en las neuronas de salida. Actualmente este tipo de algoritmos son usados para la búsqueda de patrones, o características, en los datos de entrada con el fin de obtener una salida concreta. Un claro ejemplo son las redes neuronales de convolución, conocidas en la literatura como CNN, que se ha comprobado, a lo largo de la literatura, que son un buen mecanismo para obtener información semántica de los datos de entradas para realizar tareas de clasificación, detección y segmentación.

En relación a esta serie de cuestiones y problemas nace una nueva línea de investigación a finales de los años 80, la cual se centra en estudiar los sistemas biológicos e internos del procesamiento neuronal humano (Maher et al. 1989; Mead & Mahowald 1988), con el fin de conseguir sistemas analógicos y digitales que sigan los mismos patrones de funcionamiento utilizados por las neuronas del cerebro humano. Esta nueva línea de investigación es conocida como ingeniería Neuromórfica, la cual plantea nuevos sensores y sistemas bio-inspirados cuyos objetivos más destacables son los de ofrecer soluciones de muy alta velocidad de transferencia de información y procesamiento, como el de entender el funcionamiento del cerebro intentando de imitarlo. En la literatura más reciente podemos encontrar sensores de visión (Lichtsteiner et al. 2008; Serrano-Gotarredona & Linares-Barranco 2013), auditivos (Chan et al. 2007; Jimenez-Fernandez et al. 2016), sistemas de control motor (Jimenez-Fernandez et al. 2012; Perez-Peña et al. 2013), redes neuronales pulsantes (Stromatias et al. 2015), entre otros muchos.

1.1 Motivación

Muchas son las soluciones aportadas en el campo de las redes neuronales en las que los datos de entrada de dichos sistemas son provenientes de un solo sensor o tipo de dato. ¿Por qué no pensar que estos tipos de sistemas podrían ser mejores y resolutivos si tenemos más de un tipo de dato en su entrada o una combinación de ellos?

La principal motivación del presente trabajo se centra en el ámbito científico y en el estudio de las redes neuronales de convolución para una aproximación de la fusión sensorial, en concreto haciendo uso de sensores neuromórficos. Este tipo de sensores tienen la

particularidad de obtener información relevante del entorno evitando aquella que nos aportaría menos valor a la hora de procesarla.

El presente trabajo trata de realizar un análisis y estudio de la viabilidad de uso de las redes neuronales de convolución combinando la información de dos tipos de sensores neuromórficos, audio y vídeo, en el campo de la fusión sensorial. Para ello se plantean dos modelos de red neuronal de convolución, cuyos datos de entradas proceden de cada uno de los dos sensores anteriormente mencionados para efectuar una tarea en concreto.

En cuanto a lo personal, cabe destacar la inquietud del autor por dotar de múltiples sensores a los sistemas diseñados para realizar diferentes tareas como la navegación de robots móviles, la integración de sensores inerciales para orientación en el espacio o tareas como la verificación de la calidad de un sistema motor. Parte de aquí la motivación de utilizar y estudiar nuevos enfoques que alineados a las nuevas tendencias de la inteligencia artificial, sean capaces de aprender la tarea que se le encomienda haciendo uso de más de un tipo de sensor. Los sensores utilizados tanto para vídeo como para audio son sensores neuromórficos, los cuales presentan algunas ventajas frente a los convencionales para este trabajo. El sensor de visión consiste en una retina de silicio que solamente recoge la información de la escena que presenta movimiento, eliminando aquella menos relevante. Para su conversión a frames, se integra dicha información con el fin de crear un histograma, por lo que podemos reducir el *frame rate* del sensor a nuestra necesidad. El sensor de audio utilizado, nos genera a su salida la división en frecuencias del sonido original captado, evitándonos todo el post-procesamiento para obtener esa información. Además, como líneas futuras se plantean la evolución de los modelos de fusión sensorial que se presentan en este trabajo al dominio pulsante, por lo que ambos sensores son ideales para los futuros modelos.

Todo lo expuesto anteriormente se enmarca en la trayectoria investigadora del grupo *Robotic and Technology of Computes (RTC, TEP-108)* al que pertenece el autor. Este trabajo se encuentra focalizado y alineado con diversas tareas de los proyectos de investigación nacionales e internacionales, que han servido de guía, causa y financiación. Los proyectos referidos son los siguientes:

- Proyecto nacional BIOSENSE: Sistema bio-inspirado de fusión sensorial y procesamiento neuro-cortical basado en eventos. Aplicaciones de alta velocidad y bajo coste en robótica y automoción.
- Proyecto de excelencia MINERVA: Mota-Infraestructura de Sensado y Transmisión Inalámbrica para la Observación y Análisis de la Pauta de Animales Salvajes o en Semilibertad

- Proyecto internacional NPP²: Procesador neuromórfico. Acelerador de redes neuronales de convolución.

A continuación se hará una introducción detallada de los grandes campos de la ciencia donde se enmarca este trabajo, empezando por la ingeniería neuromórfica, continuando con la visión y audición artificial, y finalizando con la evolución de la inteligencia artificial, focalizando en el aprendizaje profundo.

1.2 Ingeniería neuromórfica

Como se comentaba al principio de este capítulo, los seres vivos se han extendido a lo largo y ancho de la tierra colonizando los diversos hábitats. Una de las claves de esta expansión de la vida ha sido la capacidad de adaptación de los seres vivos, estando dotados por la naturaleza de las cualidades necesarias para poder sobrevivir a un determinado entorno. Por ejemplo, una abeja demuestra una notable tarea en navegación e inteligencia social mientras busca alimento del néctar, y logra este rendimiento usando menos de un millón de neuronas. Es por ello que la capacidad de los cerebros de los animales para interactuar con el mundo que los rodea ha proporcionado un desafío continuo para la tecnología.

A lo largo de la historia y en infinidad de ocasiones, los ingenieros se han inspirado y observado las soluciones alcanzadas por la naturaleza para resolver problemas en diversos campos, siendo éste el origen de los sistemas bio-inspirados, encontrándolos a nuestro alrededor cada vez con más frecuencia. Ejemplos muy claros que hoy día tenemos bastante interiorizados pueden ser los aviones, submarinos y más recientemente los robots con capacidad de aprender.

Por otro lado, tanto la industria como nuestra vida cotidiana han sufrido una gran revolución gracias a la aparición de los sistemas computacionales y robóticos. En la industria se ha ido incorporando de manera progresiva el uso de robots destinados a tareas en las que se necesita elevada precisión, repetitividad o la realización de esfuerzos sobrehumanos. Sin embargo, este tipo de robots están programados para realizar un número muy limitado de tareas, sin capacidad de aprendizaje, en entornos muy controlados y por norma general tienen un alto consumo energético. Si los contrastamos con las habilidades de los animales, no sólo pueden navegar inteligentemente (dentro de sus posibilidades) por su entorno, sino que son

² Neuromorphic Processor Project

capaces de procurarse su propia energía, aprender, desarrollar actividades sociales, personalidades propias, organizarse para conseguir fines comunes, etc... Resumiendo en otras palabras, el desarrollo de habilidades sociales, cognitivas y culturales.

Por regla general, este tipo de robots en la actualidad están gobernados por comportamientos algorítmicos procesados por sistemas basados en computadores. En los últimos años los computadores han evolucionado a un ritmo muy elevado, alcanzando una capacidad de cálculo elevadísima, tal y como predecía la Ley de Moore (Moore 1965). A pesar de este avance en los computadores, los robots no han aumentado en la misma medida, progresando a un ritmo mucho más lento. En este punto podríamos plantearnos si sería posible modelar con suficiente fidelidad el comportamiento de un ser vivo o de alguna habilidad concreta para ser codificado algorítmicamente. En caso de que este planteamiento fuera posible, ¿se podría ejecutar ese algoritmo en un tiempo razonable y con suficiente fiabilidad en un computador? Las posibles respuestas a este planteamiento y pregunta podrían dar lugar a nuevos planteamientos y preguntas, pero entre ellas cabe plantearse si los sistemas basados en computador actuales son los sistemas más apropiados para proporcionar a los robots de habilidades cognitivas avanzadas (Penrose 1990).

La solución a todas estas cuestiones es actualmente desconocida, pero tal vez pase por imitar al "controlador" de los propios seres vivos, el sistema nervioso central, realizando tareas de "ingeniería inversa", surgiendo los sistemas neuro-inspirados.

1.2.1 Inicios y evolución

El inicio de la ingeniería neuromórfica data de los años 80, cuando el grupo de Caver Mead en Caltech³ (Mead 1989), plantea como objetivo inicial imitar el comportamiento de las neuronas en el sistema nervioso mediante circuitos analógicos VLSI o aVLSI (Liu 2002). Sin embargo, el campo de estudio de los ingenieros neuromórficos se ha expandido en los últimos años, usando tanto circuitos analógicos, como digitales para modelar el comportamiento de las neuronas. Para conseguir este objetivo es muy importante el entender como la naturaleza ha sido capaz de crear arquitecturas neuronales muy robustas al ruido, increíblemente eficientes y que además son capaces de aprender. En la ingeniería neuromórfica se dan cabida una serie de investigadores de los más diversos campos, como son físicos, matemáticos, psicólogos, médicos, biólogos e ingenieros.

³ California Institute of Technology

Desde el comienzo de la ingeniería neuromórfica hasta hoy día, esta comunidad se ha expandido considerablemente desde sus orígenes en Caltech y Johns Hopkins. Al principio sólo unos cuantos de prototipos de sistemas neuromórficos podían ser encontrados en algunos laboratorios, pero en los últimos 20 años, tenemos plataformas neuromórficas donde desplegar modelos con cientos y miles de neuronas (Furber et al. 2014; Indiveri et al. 2006). Además podemos encontrar sensores neuromórficos con un gran rango dinámico que pueden ser utilizados para aplicaciones de alto rendimiento (Chan et al. 2007; Lichtsteiner et al. 2008; Jimenez-Fernandez et al. 2016; Serrano-Gotarredona & Linares-Barranco 2013) además de herramienta para la depuración de estos sistemas (Serrano-Gotarredona et al. 2009) como métodos de conversiones *frame-spikes* sintéticos (Linares-Barranco et al. 2004; Linares-Barranco et al. 2006).

Hoy en día se celebran dos convenciones internacionales sobre ingeniería neuromórfica, una con sede en Telluride (Estados Unidos) y otra con sede en CapoCaccia (Italia), representando un punto de encuentro para investigadores de todo el mundo. En estos workshops los investigadores tienen la oportunidad de exponer a la comunidad científica sus últimos avances, además de trabajar con personas de otros centros de investigación y compartir diversos dispositivos hardware neuro-inspirados para el diseño de sistemas de procesamiento de información pulsante.

1.2.2 Sistemas neuro-inspirados

En 1906 el científico español Santiago Ramón y Cajal recibió el premio Nobel de medicina por sus pioneras investigaciones sobre la estructura microscópica del cerebro, descubriendo que el cerebro estaba formado por un conjunto de células independientes conectadas entre sí, las neuronas. Sus estudios fueron posibles gracias a los avances en los métodos de tinción y de la tecnología de los microscopios, que permitieron obtener en láminas ilustradas sus observaciones. En la Figura 1 se puede ver una reproducción de unos de los dibujos del científico español.

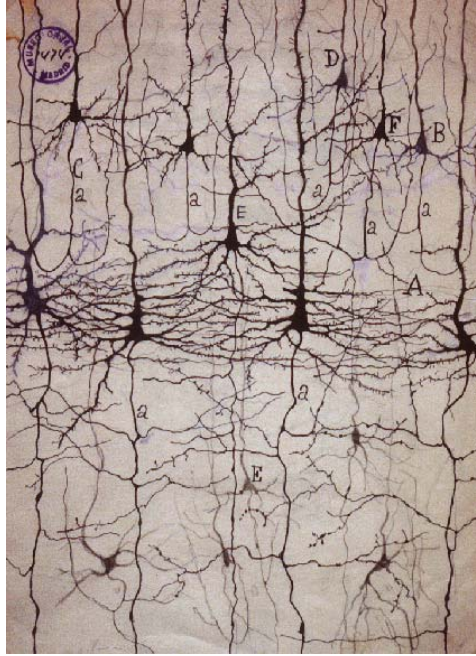


Figura 1. Reproducción de un dibujo de Ramón y Cajal que muestra algunas neuronas del córtex de un mamífero.

Una típica neurona puede ser dividida en tres partes funcionalmente distintas, llamadas dendritas, soma y axón. En la Figura 2 se puede ver un esquema básico de las partes de una típica neurona. Hablando a grandes rasgos, las dendritas juegan el rol de “dispositivos de entrada” que colectan información de otras neuronas y la transmiten al soma. El soma es “la unidad central de procesamiento” que realiza un tratamiento no lineal de la información recibida. La información recibida modifica el potencial de membrana que posee la neurona, y si éste alcanza un cierto umbral, una señal es generada y viajará por el “dispositivo de salida”, el axón, y será entregada a otras neuronas conectadas a ésta.

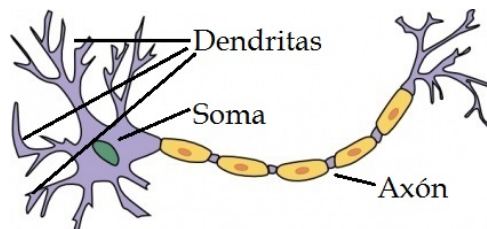


Figura 2. Esquema básico de una neurona.

La conexión entre dos neuronas se conoce como sinapsis. Si suponemos que una neurona envía una señal, o pulso eléctrico (también llamado spike), a través de una sinapsis, a la neurona que envía el spike se le conoce como neurona pre-sináptica y a la neurona que recibe el spike se le conoce como neurona post-sináptica. En el caso que la neurona pre-sináptica envíe pulsos excitatorios tendremos un cambio llamado PEPS⁴, mientras que si envía pulsos inhibitorios tendremos un cambio llamado PIPS⁵ (ver Figura 3). Una sola neurona del córtex vertebral suele estar conectada a 10^4 neuronas post-sinápticas. Muchas de las terminaciones de un axón terminan directamente en las neuronas más próximas, pero otros pueden llegar a medir varios centímetros, para alcanzar a neuronas que están en otras áreas del cerebro.

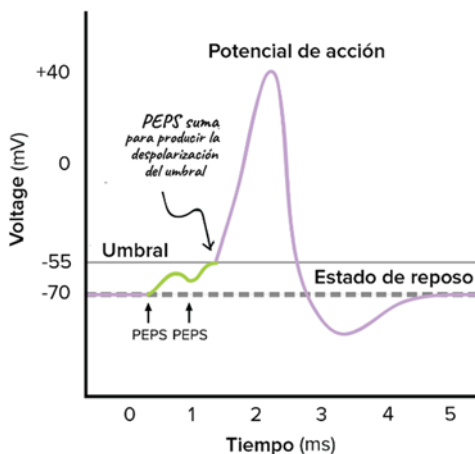


Figura 3. Diagrama de un pulso eléctrico, o spike, generado por una neurona

Uno de los puntos clave de los mecanismos de procesamiento neuronal es la hipótesis de la representación de la información por parte de las neuronas. Mucho se ha debatido sobre cómo codificar la información en los spikes, Horace Barlow en 1961 propuso varios modelos (Barlow 1961), siendo un modelo muy aceptado por parte en el campo de la ingeniería neuromórfica el que propone la codificación de la información en la frecuencia de los spikes, siguiendo una modulación en frecuencia de pulsos, spikes o PFM (Maass & Bishop 1999; Westerman et al. 1997). De esta manera la información puede ser codificada de manera continua, sin necesidad de realizar una discretización temporal de la información (Hynna & Boahen 2001; Fujii et al. 1996).

⁴ Potencial excitatorio post-sináptico

⁵ Potencial inhibitorio post-sináptico

La representación pulsante es muy eficiente desde varios puntos de vista, empezando por su simplicidad, reduciendo la cantidad de canales de comunicación necesarios para transmitir spikes, y finalmente y no menos importante, proporcionando una información continua en el tiempo y no discreta. Así pues, la representación pulsante proporciona una minimización de canales de comunicación, permitiendo una alta tasa de conectividad entre neuronas, y además, al no estar la información muestreada, evita la transmisión de información redundante, transmitiéndose así únicamente los spikes cuando son necesarios, no saturando los canales de comunicación de manera innecesaria.

En la mayoría de los animales, el cerebro es el sistema nervioso central localizándose en la cabeza, protegido por el cráneo y cerca de los principales órganos sensitivos. En el caso de los humanos, el cerebro es extremadamente complejo y se estima que tiene entre 15 y 33 billones de neuronas, pudiendo estar una de ellas conectadas a otras 10 mil. Las estructuras neuronales se agrupan por capas, las cuales están destinadas a procesar parte de la información que obtiene y tienen una funcionalidad definida (Shadlen & Newsome 1994; Rakic 1988). Para procesar la información cada neurona está conectada a un campo proyectivo de neuronas a lo largo de diferentes capas, fluyendo la información entre capas, y procesándose en este mismo flujo.

En la Tabla 1 se realiza una comparativa cualitativa generalizada acerca de la manera en la que funcionan un computador y el sistema neuronal de los animales. La primera diferencia que encontramos es que un computador se encuentra dominado bajo una señal global de reloj, que lo hace reaccionar continuamente en cada ciclo de reloj, sin embargo, las neuronas se comportan de manera totalmente asíncrona, no existiendo ningún mecanismo de sincronización entre ellas. Además, el computador es un elemento totalmente determinista, que dictamina en todo momento tras una sucesión de operaciones aritméticas y lógicas en qué estado debe encontrarse, mientras que las neuronas responden a un modelo estocástico, dependiendo su reacción de modelos probabilísticos dinámicos. Actualmente los sistemas basados en computador tienen una alta resolución de la información que manejan, muestreada a un ritmo constante, una vez más, los sistemas neuronales son completamente opuestos, la resolución no es tan elevada, pero son capaces de adaptarse a las características de la información para mejorar su representación. En los sistemas computacionales actuales el procesamiento en sí está muy centralizado, por ejemplo, un ordenador personal, comparado con la manera en que las neuronas procesan la información, ya que cada neurona procesa una pequeña parte de la información, no dependiendo de otras neuronas, implementando así una arquitectura de procesamiento masivamente paralela. El uso de computadores tiene como imperativo el uso de memoria tanto para almacenar los algoritmos

a utilizar, así como los datos iniciales, intermedios y finales, en contraposición, los sistemas neuronales no necesitan memoria para ninguno de estos fines, ya que el “algoritmo neuronal” que ejecutan está codificado en la conexión de las distintas neuronas que lo componen así como las características fisiológicas de cada una de ellas.

Computador	Sistema neuronal
Reloj centralizado de alta velocidad.	Asíncrono, sin ninguna señal global de reloj.
Dictamina perfectamente y de manera determinista el estado lógico en que se debe encontrar.	Las neuronas se comportan de forma estocástica, respondiendo a modelos probabilísticos dinámicos.
Alta resolución de la información con una tasa de muestreo constante.	De baja resolución, pero adaptativo. Sin período de muestreo, estando la información contenida en los spikes.
La computación está centralizada o levemente distribuida.	Cada neurona procesa una pequeña parte de la información, implicando una computación completamente distribuida y masivamente paralela.
La memoria está muy “lejos” del computador, necesitando memoria tanto para el algoritmo como para almacenar los datos.	Las características morfológicas y las interconexiones de cada neurona son el algoritmo en sí, la información está contenida en el flujo de los spikes.

Tabla 1. Comparativa cualitativa entre un computador y un sistema neuronal

1.2.3 Address-Event-Representation

El córtex cerebral de los animales está estructurado en diferentes capas divididas en lo que se conoce como materia gris y materia blanca. En la Figura 4 se muestra una sección transversal de un pequeño pedazo del córtex visual del cerebro de un gato. Los muchos tipos de neuronas, de los cuales hay unas $10^5/\text{mm}^3$, están conectadas a otras muchas neuronas a través de sus axones (con una densidad de “cableado” de 9m de axón por mm^3) en la zona de materia blanca, mientras la materia gris está compuesta mayormente por las dendritas, con una densidad de “cableado” enorme de $4\text{km}/\text{mm}^3$ (Braitenberg & Schüz 1991).

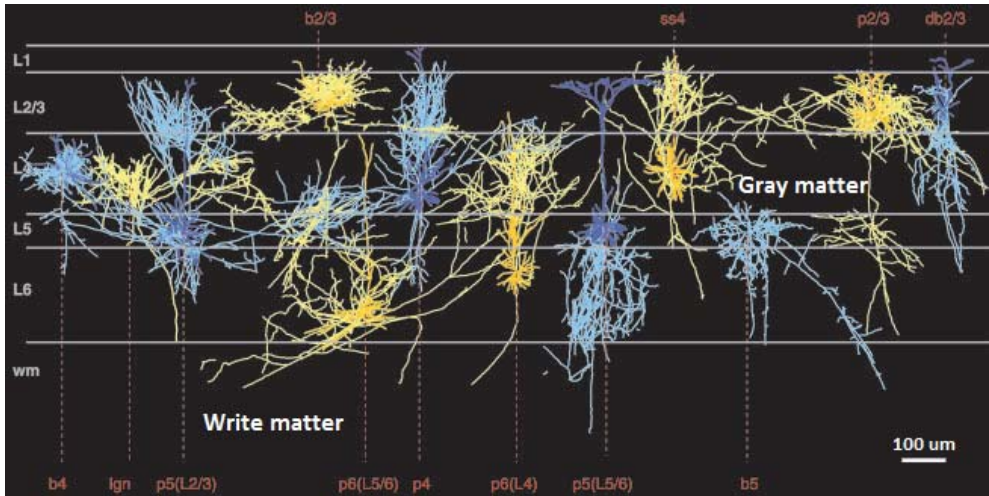


Figura 4. Vista transversal de las células del córtex en capas con sus dendritas en la materia gris y los axones proyectando hacia la materia blanca.

Una típica neurona tiene una frecuencia de disparo en el rango 1-10 Hz, por lo tanto, cientos de neuronas combinadas tendrán una frecuencia de disparo en el rango de los KHz o MHz. Los sistemas digitales actuales pueden soportar esta frecuencia de datos. Para modelar la comunicación en los sistemas neuromórficos electrónicos, no se crean redes o conexiones entre neuronas de manera individual, se crean conjuntos de neuronas que se conectan unos con otros, correspondiéndose cada conjunto de neuronas con una capa de procesamiento. Los circuitos utilizados multiplexan la comunicación para un conjunto de neuronas en un canal de comunicación individual donde cada neurona es identificada con una dirección única dentro del conjunto. Este tipo de representación es conocida como representación por dirección de evento (AER⁶). AER fue propuesto por primera vez en 1991(Sivilotti 1991; Lazzaro et al. 1993; Lazzaro & Wawrzynek 1995; Mahowald 1992).

La función de los circuitos AER es la de proporcionar funcionalidades de *multiplexión / demultiplexión* para los spikes que son *generados por / entregados a* un conjunto individual de neuronas. La Figura 5 muestra un ejemplo de codificación de los spikes de cuatro neuronas en un único canal de salida. Los spikes son generados asincrónicamente y recogidos por el circuito AER que los codificará y multiplexará en el canal de salida en el orden que fueron generados. La secuencia de valores producidos en el canal de salida indica que neurona (su dirección en el array de neuronas) disparó. El tiempo en el que la dirección de la neurona es

⁶ Address Event Representation

generado se corresponde con el tiempo en el que esa neurona generó un spike, más un pequeño retraso debido al proceso de codificación. Mientras que los spikes estén suficientemente separados en el tiempo, el proceso de codificación asegura que las direcciones que identifican a las neuronas están correctamente ordenadas en la secuencia.

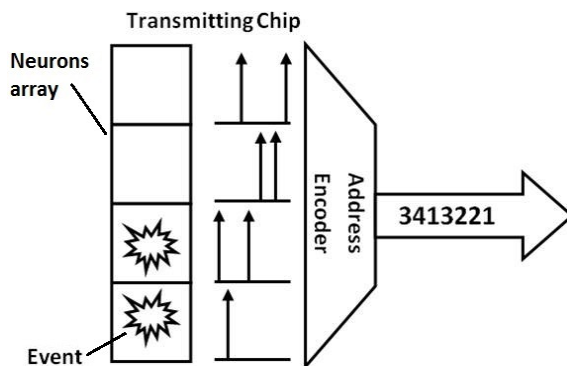


Figura 5. Multiplexado de diferentes neuronas en un único canal de comunicación

Si en el array de neuronas se garantizara que sólo una neurona puede generar un spike mientras que las restantes no pueden en el mismo instante de tiempo, el proceso de multiplexión se correspondería con un circuito de codificación asíncrono estándar. Sin embargo, esta no es una restricción válida para los conjuntos de neuronas, se puede dar que varias neuronas disparen al mismo tiempo produciendo solapamiento en los spikes, denominado colisión. Así pues, los circuitos de codificación utilizan una lógica para manejar la arbitración de la potencial llegada simultánea de spikes de varias neuronas.

Los codificadores AER tienen varios axones de neuronas como entradas y tienen que implementar principalmente dos funcionalidades. La primera es la de determinar qué spike tiene que ser el siguiente en ser comunicado en el canal de salida. Esto se corresponde a un problema clásico de arbitración asíncrona en el que una vez seleccionado el axón de la neurona que ha disparado, el circuito debe de codificar dicho axón en una dirección. Este funcionamiento es el de un codificador tradicional, donde una de las N direcciones diferentes es codificada utilizando $\log N$ bits. La diferencia entre los distintos codificadores AER que hay en la literatura es el mecanismo utilizado para resolver el conflicto entre múltiples spikes simultáneos y cómo las direcciones de las neuronas son codificadas. Cada opción utilizada tiene sus ventajas e inconvenientes, y son apropiados para diferentes tipos de sistemas neuromórficos. En (Liu 2015) podemos encontrar una amplia explicación de cada uno de los mecanismos y sus diferentes usos.

El circuito de decodificación o demultiplexación es menos complejo que el anterior. En este proceso se reciben unos valores de entrada proveniente de un canal AER que especifican las direcciones de las dendritas/axones (dependiendo el punto de vista) en el que los spikes deben de ser entregados. La dirección de la dendrita es decodificada utilizando un decodificador asíncrono, y el spike es entregado a la neurona correspondiente. La Figura 6 muestra cómo una secuencia de direcciones de spikes es decodificada en spikes que son entregados a cada dendrita individualmente y que están conectadas a su neurona correspondiente. Si el retraso entre los spikes adyacentes en la secuencia AER de entrada es lo suficientemente grande, la salida del decodificador AER entregará los spikes a las dendritas en el tiempo que corresponde al tiempo de llegada en la entrada más un pequeño retraso debido al circuito de decodificado.

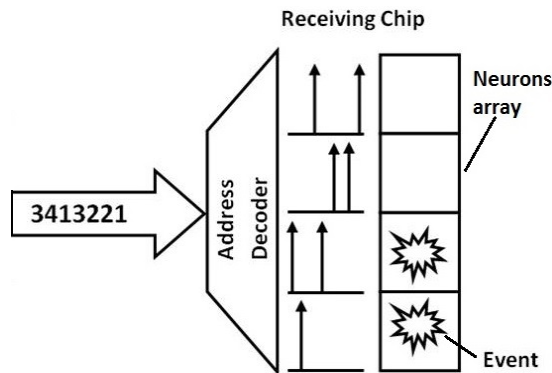


Figura 6. Demultiplexación de una entrada AER en spikes individuales para cada dendrita

En el esquema AER más simple posible, las direcciones de las neuronas generadas por el circuito de codificación se corresponden con las direcciones de las dendritas de las neuronas en el destino. Si conectamos, por ejemplo, directamente la salida de la Figura 5 con la entrada de la Figura 6, tendríamos una conexión directa entre los axones de las neuronas del array fuente con las dendritas de las neuronas del array de destino.

1.3 Visión artificial

Uno de los sentidos más importantes de los seres humanos es la visión. A través de este sentido podemos captar información visual, de una forma muy directa, del entorno físico que nos rodea. Es uno de los sentidos más complejos que el ser humano posee y se calcula que más del 70% de las tareas del cerebro están dedicadas al análisis de información visual.

El hecho de conectar una cámara a un computador, captar información visual del entorno y ser capaz de procesarla, supuso un gran avance en la era tecnológica. Es a partir de este momento donde nace la visión por computador, o visión artificial, (Harmon & Knowlton 1969) como disciplina que intenta emular la capacidad que poseen algunos seres vivos para captar información de su alrededor y procesarla. A lo largo de su historia se han ido diseñando e implementando algoritmos para realizar tareas como la segmentación, clasificación de patrones, localización de objetos, etc.

Si echamos la vista atrás, en el año 1826 el químico francés Niepce (1765-1833) llevó a cabo la primera fotografía colocando una superficie fotosensible dentro de una cámara oscura para fijar la imagen (Meteyard 1871). Posteriormente, en 1838 el químico francés Daguerre (1787-1851) hizo el primer proceso fotográfico práctico. Daguerre utilizó una placa fotográfica que era revelada con vapor de mercurio y fijada con trisulfato de sodio.

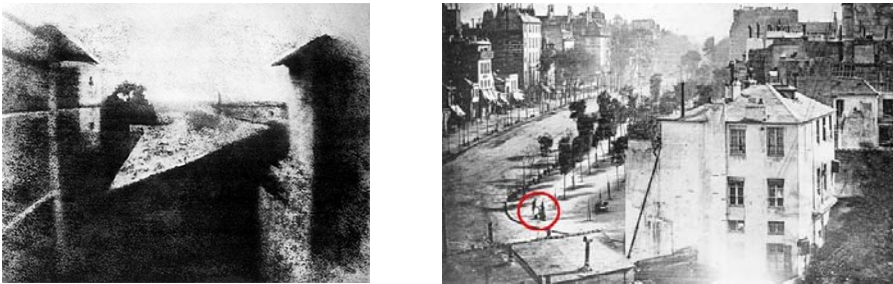


Figura 7. Izquierda, primera fotografía que se conoce. Derecha, primera fotografía donde aparece una persona



Figura 8. Izquierda, imagen captada por un telescopio. Derecha, primera radiografía que se conoce.

Desde que se inventó la fotografía se ha venido intentando extraer características o patrones de las imágenes. Por ejemplo, las radiografías transformaron el campo de la medicina, llegando a tener hoy día imágenes en 3D de nuestro cerebro completo o también la astronomía avanzó enormemente debido a la capacidad de adquirir imágenes de sistemas muy lejanos al nuestro a través de los telescopios.

Hoy en día existen sistemas autónomos, que gracias a la visión artificial junto con otros tipos de sensores, son capaces de interactuar con su entorno y navegar a través de él. Este tipo de sistemas, son capaces de obtener gran cantidad de información visual y procesarla en busca de ciertas características o patrones, que en función de esa información procesada son capaces de realizar tareas que previamente han aprendido. Un claro ejemplo de ello es el sistema de conducción autónomo de Tesla.



Figura 9. Sistema de conducción autónomo de Tesla

1.3.1 Adquisición de una imagen digital

La primera etapa en la visión artificial pasa por la adquisición y digitalización de la imagen que se muestra frente a nuestro sensor visual. En particular para este trabajo, existen tres tipos interesantes de imágenes que somos capaces de adquirir; las imágenes de intensidad, térmicas y las de profundidad.

- Las imágenes de intensidad se basan en la cantidad de luz recogida por el dispositivo de captura en cada uno de los puntos de la escena. Son las más comúnmente utilizadas debido principalmente a la facilidad de obtener este tipo de sensores.
- Imágenes térmicas: Para la adquisición de este tipo de imágenes se hace uso de dispositivos que son capaces de recoger la señal de calor que emiten los cuerpos creando un mapa de color atendiendo al calor recogido de cada uno de los objetos de la escena infrarroja.
- Las imágenes de profundidad, también conocidas como mapas de profundidad, atiendes a imágenes 3D donde el valor de cada punto x,y se corresponde a la dimensión z que denomina la profundidad de ese punto en la escena.

Si nos centramos en las imágenes de intensidad, debido a su amplio uso, tras su captura obtenemos una matriz de dos dimensiones donde cada posición indica el valor de la intensidad de ese punto de la escena. El concepto de imagen está asociado a una función bidimensional $f(x,y)$, cuya amplitud o valor será el grado de iluminación en el espacio de coordenadas (x,y) de la imagen para cada punto. El valor de esta función depende de la cantidad de luz que incide sobre la escena vista, así como de la parte que sea reflejada por los objetos que aparecen en dicha escena. Estos componentes llamados iluminación y reflexión son descritos por $i(x,y)$ y $r(x,y)$ respectivamente. El producto de ambas funciones da como resultado la función $f(x,y)$.

$$f(x,y) = i(x,y) \cdot r(x,y)$$

Ecuación 1. Función asociada a una imagen de intensidad

La función de imagen $f(x,y)$ es digitalizada espacialmente y en amplitud. La digitalización de las coordenadas espaciales (x,y) está asociada al concepto de muestreo, mientras que la digitalización de la amplitud al de cuantificación de los niveles de gris. Así pues, una imagen está almacenada en una matriz de $N \times M$ elementos teniendo en cuenta que el origen de coordenadas es la esquina superior izquierda, donde el eje x es el horizontal y el eje y el vertical.

El muestreo es la conversión que sufren las dos coordenadas espaciales de la señal analógica, y que genera la noción de píxel. La imagen se comporta como una matriz donde el valor de los elementos es su nivel de gris, la fila y la columna su posición en el espacio. La cuantificación es la conversión que sufre la amplitud de la señal analógica; así se genera el concepto de nivel de gris o intensidad. Para el caso de tener por ejemplo 256 niveles de gris en cada píxel (0-255), el 0 corresponde a un píxel no iluminado o a un punto que absorbe todos

los rayos luminosos que inciden sobre él (negro), mientras que el nivel 255 corresponde a un píxel muy iluminado o a un punto que refleja todos los rayos que inciden en él (blanco).

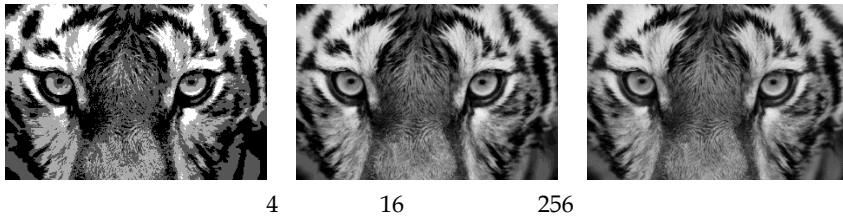


Figura 10. Imagen con diversos niveles de gris

En el proceso de digitalización es preciso determinar el valor de N y M así como el número de niveles de gris asignados para cada píxel. Es una práctica común en el proceso de digitalización de imágenes que estas cantidades sean números enteros potencias de dos. Para calcular el número de bits nb requeridos para almacenar la imagen digitalizada se hace uso de la siguiente ecuación, siendo 2^g el número de niveles de gris:

$$nb = MNg$$

Ecuación 2. Cálculo del número de bits necesarios para almacenar una imagen

Por último cabe destacar el concepto de resolución, que está asociado al número de muestras y niveles de gris necesarios para tener una buena aproximación de la imagen. La resolución de una imagen se mide en ppp (píxel por pulgada) y es tenido en cuenta para medir la calidad de una imagen.

1.3.2 La retina de silicio

Emular la retina electrónicamente es uno de los mayores objetivos de los ingenieros electrónicos neuromórficos desde el inicio de los tiempos. Cuando Fukushima y su grupo (Fukushima et al. 1970) demostraron su modelo discreto de la retina, se generó una gran expectación aunque el impacto en la industria fue minúsculo. En la ingeniería neuromórfica se ha realizado un gran esfuerzo en los últimos 20 años para convertir esos primeros prototipos de los laboratorios en dispositivos comerciales que pueden ser incluidos en aplicaciones reales.

1.3.2.1 Retinas biológicas

La retina es una estructura compleja con tres capas primarias: la capa foto-receptora, la capa plexiforme externa y la capa plexiforme interna (Werblin & Dowling 1969; Masland 2001; Kuffler 1953). La capa foto-receptora contiene dos tipos de células: los conos y los bastones, los cuales transforman la luz que entra en una señal eléctrica que afecta a la liberación de los neurotransmisores en las sinapsis de salida del foto-receptor. Las células foto-receptoras a su vez conducen las células horizontales y bipolares en la capa plexiforme externa. Las dos principales clases de células bipolares, las células ON y OFF, codifican por separado los cambios de contraste espacio-temporal del brillo y oscuridad. En la capa plexiforme externa, las células bipolares ON y OFF realizan la sinapsis con diversos tipos de células amacrinas y con varios tipos de células ganglionares en la capa plexiforme interna.

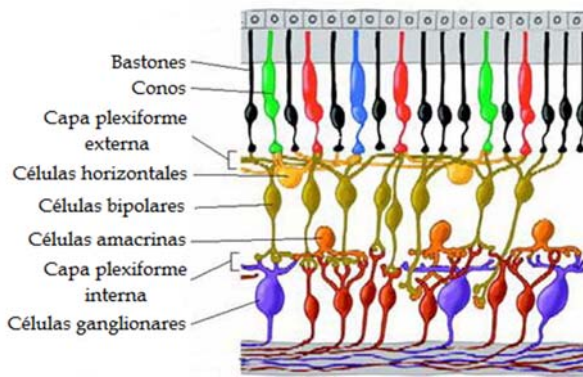


Figura 11. Diagrama de una sección transversal de la retina

Las retinas biológicas tienen muchas características deseables de las que carecen las cámaras convencionales, de las cuales vamos a mencionar dos que han sido incorporadas en las retinas de silicio. En primer lugar, los ojos operan sobre un amplio rango dinámico (RD) de intensidad de luz, permitiendo la visión en condiciones de iluminación muy variada. Por otro lado, las células de las capas plexiformes interna y externa codifican el contraste espacio-temporal, eliminando así la información redundante y permitiendo a las células codificar las señales dentro de su limitado RD (Barlow 1961).

1.3.2.2 Retinas AER

La naturaleza asíncrona de la salida AER está inspirada en la forma en que las neuronas se comunican. Mejor que muestrear los valores de los píxeles, los píxeles de la retina AER escriben asíncronamente en la salida sus direcciones (dirección del evento) cuando ellos detectan una señal significativa. La definición de “significante” depende del tipo de píxel, pero

en general, el número de eventos a la salida de estos sensores deberían de ser menor que los valores analógicos obtenidos a la salida de un sensor visual típico. En otras palabras, los píxeles de una retina determinan asincrónicamente su propia región de interés (ROI), una tarea que por ejemplo en los sistemas de visión convencionales se deja para un procesamiento algorítmico y comúnmente pesado en términos computacionales.

Las retinas AER pueden ser divididas generalmente en las siguientes clases:

- **Contraste espacial:** Estos sensores reducen la redundancia espacial basada en las relaciones de intensidad versus a los sensores de diferencia espacial los cuales utilizan las diferencias de intensidad. Este tipo de sensor es especialmente útil para analizar el contenido de escenas estáticas con el fin de obtener características y reconocer objetos.
- **Contraste temporal:** Estos sensores reducen la redundancia temporal basada en los cambios relativos de intensidad a diferencia de los sensores de diferencia temporal que utilizan los cambios absolutos de intensidad. Este tipo de sensor se utiliza en escenas dinámicas con iluminación uniforme para aplicaciones de seguimiento de objetos y navegación.

Actualmente en la literatura científica se pueden encontrar cuatro tipos más significativos de retinas de silicio: el sensor de visión dinámica (DVS⁷) (Lichtsteiner et al. 2008; Serrano-Gotarredona & Linares-Barranco 2013; Guo & Chen 2016), el sensor de imagen asíncrona basada en el tiempo (ATIS⁸) (Posch et al. 2008), la retina Magno-Parvo (Zaghloul & Boahen 2004a; Zaghloul & Boahen 2004b), el sensor de imagen biométrica Octopus (Culurciello et al. 2003), y el sensor de contraste espacial y orientación (VISE) (Rüedi et al. 2003). Particularmente en el presente trabajo se hará uso de un sensor DVS (Serrano-Gotarredona & Linares-Barranco 2013), por lo que se procederá a explicar con un poco más de detalle este tipo de sensor.

Sensor de Visión Dinámica

El sensor DVS responde asincrónicamente a los cambios relativos temporales de intensidad lumínica. El sensor tiene como salida un flujo asíncrono de direcciones de eventos (AEs⁹) que codifican los cambios de reflectancia en la escena, reduciendo así la redundancia de datos mientras se preserva una precisa información temporal. Estas propiedades se consiguen modelando las siguientes propiedades claves de la visión biológica: dispersión, salida basada

⁷ Dynamic Vision Sensor

⁸ Asynchronous Time-based Image Sensor

⁹ Address Events

en eventos, representación de los cambios de luminancia relativa y la rectificación de señales positivas y negativas en diferentes canales de salida. Este tipo de sensor ha sido utilizado en varias aplicaciones: Seguimiento de una bola a alta velocidad en un robot portero (Delbruck & Lichtsteiner 2007), detección y seguimiento de las manos de un conductor al volante (Rios-Navarro et al. 2012), cálculo de la velocidad de un motor en tiempo real (Rios-Navarro et al. 2015), robot que mantiene en equilibrio un lápiz (Conradt et al. 2009) entre otras. Muchos de los proyectos desarrollados con este sensor pueden encontrarse implementados en un proyecto open-source llamado jAER (jAER 2007), que está enfocado al procesado por eventos. En la Figura 12 se puede ver una salida de un sensor DVS visualizado en esta aplicación.

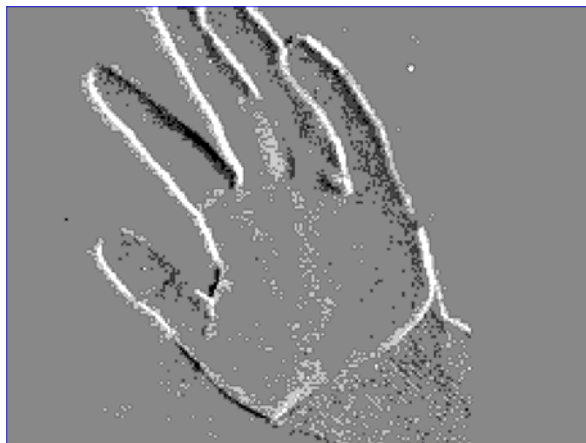


Figura 12. Representación en jAER de la salida de un sensor DVS frente a una mano en movimiento

1.4 Procesamiento de audio. El habla

Otro de los principales sentidos que posee el ser humano, como otras muchas especies animales, es la audición o el oído. A través de este sentido muchos seres vivos son capaces de obtener ondas acústicas del entorno en el que se encuentran, además es fundamental para la comunicación mediante sonido. En el caso del ser humano esta comunicación es conocida como el habla.

La comunicación mediante habla es uno de las capacidades más básicas y esenciales procesadas por el ser humano. El habla puede ser considerada como el mecanismo más importante con el que las personas pueden realmente transmitir información sin la necesidad de utilizar ninguna herramienta. La onda del habla no es una onda simple, ya que lleva

consigo información lingüística, características vocales y la emoción del emisor. El intercambio de información mediante el habla juega un rol claramente esencial en nuestras vidas.

Atendiendo a la historia, en 1875, Alexander Graham Bell le comenta a su asistente (Laboratories & Schindler 1975); *“Watson, tengo otra idea que aún no te he comentado que creo que te sorprenderá. Si puedo obtener un mecanismo que sea capaz de convertir la variación de la corriente eléctrica en su intensidad como el aire varía en densidad cuando un sonido está pasando a través de él, yo puedo telegrafiar cualquier sonido, incluso el sonido del habla”*. Esto, comúnmente se conoce como el concepto base del teléfono, pero hay que apuntar que en 1854, Antonio Meucci, construyó el teletrófono (Schiavo 1958), que posteriormente fue bautizado como teléfono. Debido a su carencia económica fue incapaz de patentar su invento por lo que no pudo defenderlo contra futuras invenciones.

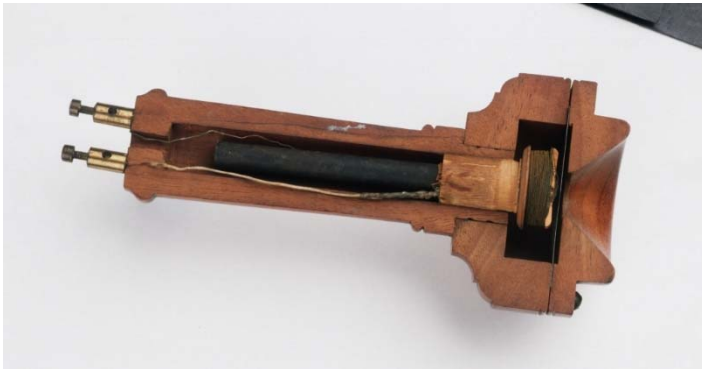


Figura 13. Primera patente de un teléfono por Alexander Graham Bell

La invención del teléfono no constituye sólo la época más importante de la historia de las comunicaciones, sino que también representa el punto de partida en el que el habla se convierte en un objetivo de la ingeniería. Para satisfacer este objetivo, la evolución en el procesamiento de audio a lo largo de la historia ha jugado un papel fundamental, teniendo como punto de inflexión el paso del procesamiento analógico al digital. El audio digital comprende nuevos conceptos y técnicas distintas a las utilizadas en la tecnología analógica, y con los computadores y sistemas de procesamiento actuales es mucho más simple trabajar con la representación digital de una señal que con la señal misma.

Actualmente podemos encontrar sistemas que realizan procesamiento del habla en ámbitos muy diversos. Muchas entidades poseen sistemas automáticos para la atención básica de sus usuarios a través del teléfono. Existen aplicaciones de dictado en las que se traduce la

voz del usuario en las frases que está dictando. También son comúnmente conocidos los asistentes inteligentes Siri¹⁰ y Cortana¹¹, integrados en algunos teléfonos móviles y en el sistema operativo Windows 10, que nos permiten interactuar con el sistema a través de la voz.

1.4.1 Conversión analógico-digital

Lo expuesto en este apartado se entiende que son conocimientos básicos para el lector, pero se repasa de manera global por el hecho de que se trata de un mecanismo utilizado por uno de los sensores que se usan en este trabajo.

La conversión analógico-digital, comúnmente conocida como digitalización, consiste en los procesos de muestreo, cuantificación y codificación. El muestreo es el proceso para representar una señal continua y variable como una secuencia periódica de valores. La cuantificación implica la representación aproximada de los valores de la forma de onda por uno de los valores de un conjunto infinito. Por último, la codificación consiste en asignar un número a cada valor. Para esta última tarea, la codificación binaria es comúnmente utilizada. A continuación se explicarán cada uno de estos procesos aunque no se entrará a un gran nivel de detalle debido a que no es objetivo de este trabajo tal efecto.

1.4.1.1 Muestreo

En el proceso de muestreo, una señal analógica $x(t)$ es convertida en una secuencia (secuencia muestreada) de valores $\{x_i\} = \{x(iT)\}$ en un tiempo periódico $t_i = iT$ (siendo i un número entero). T [s] es el periodo de muestreo, y su homóloga, $S = 1/T$ [Hz] es denominada frecuencia de muestreo. Si T es demasiado grande, se puede perder información y por tanto la señal original no puede ser reproducida a partir de la secuencia muestreada, y a la inversa, si T es muy pequeña, podemos llegar a tener sobre-información por lo que muestras inservibles de la conversión de la señal original son introducidas en la secuencia de muestras resultantes.

¹⁰ Asistente inteligente de Apple

¹¹ Asistente inteligente de Microsoft

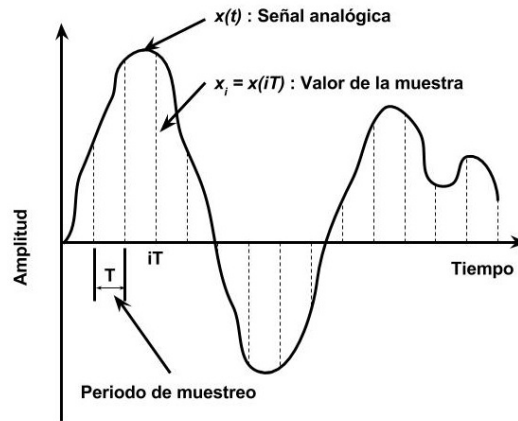


Figura 14. Muestreo en el dominio del tiempo

1.4.1.2 Cuantificación y codificación

Durante la cuantificación, todo el rango de amplitud continua es dividido en sub-rangos finitos, así pues a las amplitudes que están en el mismo sub-rango se les asignará el mismo valor de amplitud. La Figura 15 muestra un ejemplo de características de entrada/salida de una cuantificación de 8 niveles (3 bits), donde Δ es el tamaño del paso de cuantificado. En este ejemplo, cada codificación es asignada de la forma que representa el valor de la amplitud. Las características de la cuantificación depende tanto del número de niveles como del tamaño del paso de cuantificado. Cuando se va a cuantificar la señal utilizando B bits, el número máximo de niveles será 2^B . Ambos valores Δ y B son seleccionados simultáneamente con el fin de cubrir adecuadamente el rango de la señal. Si asumimos que $|x_i| \leq x_{max}$, deberíamos de establecer:

$$2x_{max} = \Delta 2^B$$

Ecuación 3. Selección de los parámetros Δ y B para la cuantificación

La diferencia entre el valor muestreado después de la cuantificación x'_i y el valor de la señal original x_i , es llamado error de cuantificación, distorsión de la cuantificación o ruido de cuantificación.

$$e_i = x'_i - x_i$$

Ecuación 4. Error de cuantificación

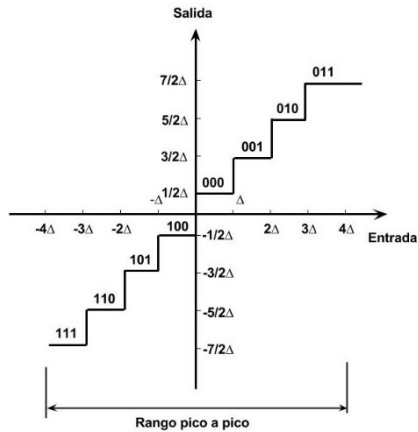


Figura 15. Ejemplo de las características de entrada-salida de una cuantificación de 8 niveles

1.4.2 Sistemas de audio neuromórficos

La cóclea es el sensor de audio de la biología, capaz de convertir variaciones de presión del aire en una señal neuronal pulsante. En el campo de la ingeniería neuromórfica podemos encontrar varios tipos de cócleas artificiales, entre las que se pueden destacar las de silicio y las digitales implementadas en FPGA¹². Ambas soluciones tienen a la salida el mismo tipo de codificación (AER) para la descomposición de la señal original en cada canal de frecuencia.

1.4.2.1 La cóclea biológica

La cóclea biológica es una estructura huesuda, en forma de espiral y llena de fluido que compone la mayoría del oído interno. Realiza la traducción entre la señal de presión, que representa la entrada acústica y las señales neuronales pulsantes que llevan la información al cerebro. En la Figura 16 se muestra la localización relativa de la cóclea con otros elementos importantes del oído humano.

La cóclea se enrolla en espiral desde la base (vuelta más baja) hasta el ápice (vuelta más alta) y contiene aproximadamente 2.5 vueltas. Dentro, la cóclea se divide en tres cámaras: la rampa vestibular, el conducto medio y el conducto timpánico; la membrana Reissner separa la primera de la segunda cámara y la membrana basilar (BM¹³) separa la segunda de la tercera cámara. Encima de la BM se sitúa el órgano de Corti, que contiene las células ciliadas del oído

¹² Field Programmable Gate Array

¹³ Basilar Membrane

tanto internas como externas (IHC¹⁴ y OHC¹⁵). Las extremidades de estas células son estereocilios parecidos a pequeños vellos. Los cambios de direcciones de las estereocilias de IHC generan señales neuronales pulsantes que viajan hacia el cerebro (Meddis 1986; Meddis 1988). Las señales neuronales pulsantes provenientes del cerebro pueden alterar la longitud y anchura de los cuerpos de las células OHC.

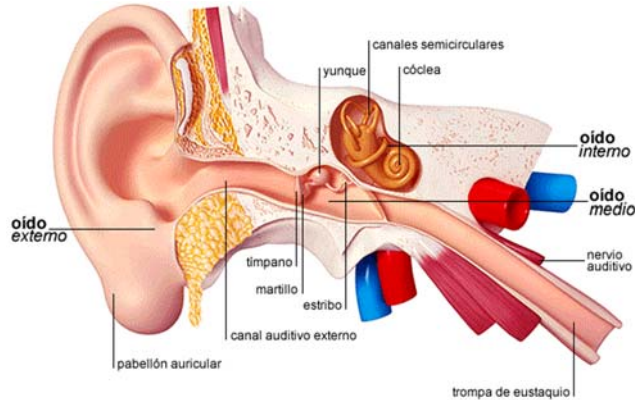


Figura 16. El oído humano

La BM cambia tanto en anchura como en rigidez, estrecha y rígida en la base, y ancha y flexible en el ápice. Estos cambios en las características físicas de la BM asisten a como la cóclea divide una señal de entrada en sus componentes frecuenciales. En la base de la cóclea las características de la BM son tales que responden mejor a estímulos de altas frecuencias, mientras que en el ápice responde mejor a los estímulos de bajas frecuencias.

1.4.2.2 El sistema de audición neuromórfico AER (NAS)

En el presente trabajo se utiliza el sensor de audio neuromórfico (NAS¹⁶) que es implementado digitalmente en una FPGA (Jimenez-Fernandez et al. 2016) en vez de la cóclea de silicio, por lo que se procede a explicarla arquitectura y funcionamiento del NAS. La Figura 18 (a) muestra la arquitectura global del NAS correspondiente a un sistema biaural o estéreo. El propósito de este sensor es descomponer dos señales digitalizadas (señales correspondientes al oído izquierdo y derecho) en un conjunto de bandas (correspondientes a frecuencias determinadas) que previamente son convertidas en secuencia de pulsos o spikes. La descomposición de las señales en cada una de las bandas correspondientes se lleva a cabo

¹⁴ Inner Hair Cells

¹⁵ Outer Hair Cells

¹⁶ Neuromorphic Auditory Sensor

por dos bancos de filtros en el dominio pulsante SLPF¹⁷ (Jimenez-Fernandez et al. 2010; Dominguez-Morales et al. 2011) conectados en cascada. La salida del sensor es codificada utilizando la representación por dirección del evento AER, tal y como la retina de silicio explicada anteriormente.

Para la descomposición de cada señal de entrada (izquierda y derecha) se realiza el mismo procesamiento. Este procesamiento es modelado utilizando un banco de filtros basados en el dominio pulsante dispuestos en cascada (CFB¹⁸). Cada CFB posee varias etapas donde cada una de ellas se corresponde con un SLPF y un SH&F¹⁹ (Jimenez-Fernandez et al. 2010) capaces de extraer dos secuencias de pulsos, una que pasa como entrada a la siguiente etapa y otra que es la correspondiente a la señal filtrada por la etapa actual. Como en otras implementaciones de Cócleas (Liu et al. 2010; Leong et al. 2003; Clive & Lyon 1992; Lyon & Mead 1988), varias etapas de filtros paso de baja conectados en cascada sustraen la información relevante eliminando aquella correspondiente a las frecuencias que están fuera de banda, obteniendo una respuesta equivalente a filtros paso de banda.

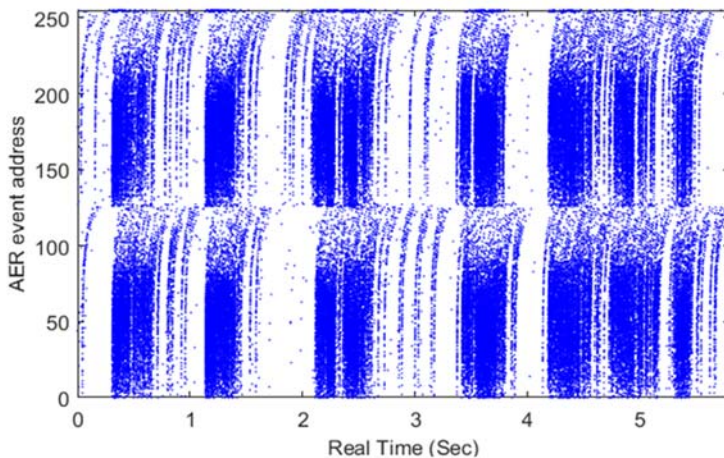


Figura 17. Representación de la salida del NAS

Atendiendo a la arquitectura global del NAS mostrada en la Figura 18 (a), el primer elemento que aparece en la cadena de conversión es el códec de audio AC97, éste tiene como entrada las dos señales analógicas (izquierda y derecha), las cuales que son digitalizadas y

¹⁷ Spike Low Pass Filter

¹⁸ Cascade Filter Bank

¹⁹ Spike Hold and Fire

multiplexadas en un canal de salida. Esta señal de salida es dividida con el fin de obtener los valores muestreados y digitalizados correspondientes a cada señal de entrada. En este punto, el procesamiento para cada una de esas muestras es común para ambas señales (izquierda y derecha). Es aquí donde se realiza el cambio del contexto digital al pulsante utilizando un generador sintético de pulsos RBSSG (Jimenez-Fernandez et al. 2010). A la salida del generador de pulsos se obtienen los pulsos positivos y negativos que codifican el valor digital anteriormente muestreado.

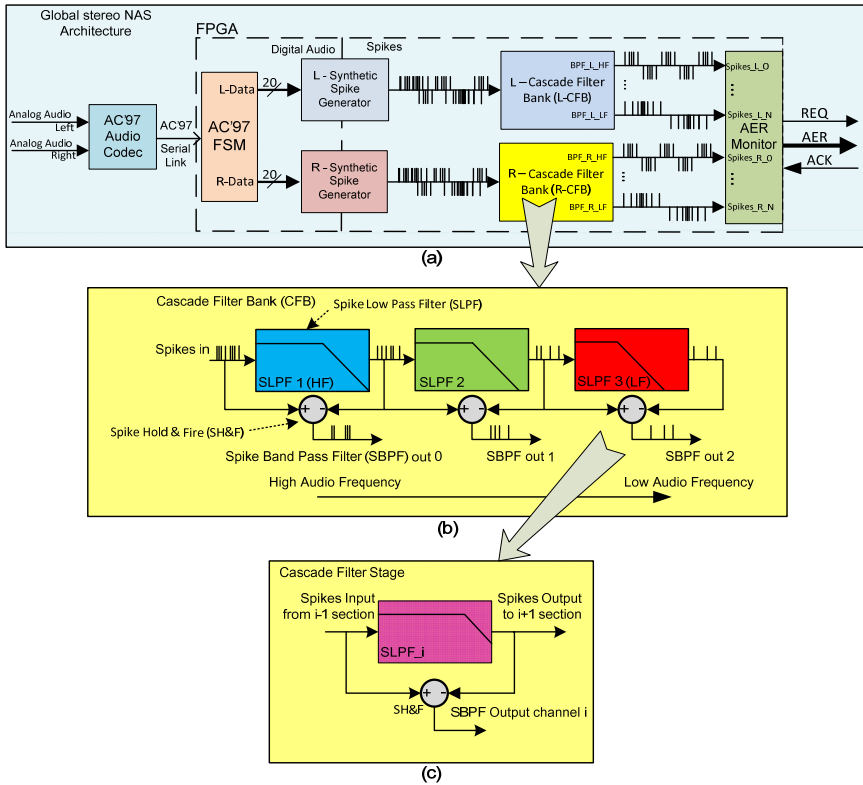


Figura 18. (a) Arquitectura general del NAS. (b) Banco de filtros dispuestos en cascada, CFB. (c) Etapa básica del CFB

Ese tren de pulsos es la entrada al banco de filtros pulsantes dispuestos en cascada CFB. La Figura 18 **¡Error! No se encuentra el origen de la referencia.**(b) muestra cómo están distribuidos esos filtros formando dicho banco, donde los primeros filtros se corresponden a los de alta frecuencia mientras que los últimos se corresponden a los de baja frecuencia. Como se comentaba anteriormente, el banco de filtros pulsantes están divididos en etapas, donde cada etapa se corresponde con un filtro pulsante paso de baja junto con un SH&F (Figura 18

¡Error! No se encuentra el origen de la referencia.(c)). La salida de este último elemento de cada una de las etapas es conectada al monitor de eventos AER que codifica la actividad de cada una de las etapas (conocidas como canales) poniendo en el bus de salida AER la dirección correspondiente a cada etapa cada vez que se obtiene un evento de ella. En la Figura 17 se puede ver salida del sensor cuando es excitado diciendo a la entrada del micrófono la frase “En un lugar de la Mancha”.

1.5 Redes neuronales de aprendizaje profundo

Las redes neuronales de aprendizaje profundo DLNNs²⁰ pueden ser definidas como arquitecturas de redes neuronales que pueden facilitar la recuperación y análisis de datos que están profundamente ocultos en la información de entrada y que no son fácilmente recuperables. Su habilidad de indagar profundamente en los datos de entrada es a veces superior y/o más rápida que otros métodos computacionales (no basados en redes neuronales) para ciertas tareas. Otra definición de DLNNs (Deng & Yu 2014) es que, “DLNNs son una clase de técnicas de aprendizaje automático que aprovecha varias capas de procesamiento de información no lineal para la extracción y transformación supervisada y no supervisada de características y para el análisis y clasificación de patrones.”

El aprendizaje profundo (en adelante *Deep Learning*) se utiliza cuando métodos más simples son insuficientes para realizar la tarea deseada. Estos mecanismos necesitan un gran y variado conjunto de datos etiquetados que representan el conocimiento y son utilizados para dotar de “inteligencia” y por ello ser capaces de desarrollar las tareas para las que son entrenados. Las DLNN deben aprender y clasificar de forma adaptativa todo el conjunto de datos mediante aprendizaje. Ese es su propósito.

1.5.1 Breve historia sobre DNN y sus aplicaciones

El desarrollo del *Deep Learning* fue uno de los principales objetivos de la inteligencia artificial desde sus comienzos, así como uno de los principales propósitos de las redes neuronales artificiales (en adelante ANN²¹). La idea era que las ANN pudieran utilizar la potencia de cálculo de las computadoras para ser capaces de indagar más profundamente en la información que lo que puede una persona y que pudieran integrar distintos métodos

²⁰ Deep Learning Neural Networks

²¹ Artificial Neural Networks

matemáticos y directamente aplicarlos a dicha información. Con el avance en la tecnología de los computadores se ha llegado a conseguir una potencia de cálculo adecuada para ejecutar este tipo de algoritmos a una velocidad suficiente para muchas de sus aplicaciones. Además, actualmente se ha llegado al punto de utilizar las ANNs con el fin de modelar ciertas zonas del cerebro humano de las que se conoce su funcionamiento (Graupe 2016).

La primera ANN que fue diseñada para ser lo suficientemente general para *Deep Learning* fue la red neuronal Back-Propagation, propuesta en 1986 por David Rumelhart (Rumelhart 1986). BP²² se basa en la teoría de programación dinámica de Richard Bellman (Bellman 1966), y aún se sigue utilizando en la mayoría de las arquitecturas de *Deep Learning*. En 1975, Kunihiko Fukushima (Fukushima 1975) propuso la red neuronal Cognitrón, que imita el funcionamiento de la retina para el reconocimiento de patrones visuales. Fukushima extendió el Cognitrón en 1980 proponiendo el Neocognitrón (Fukushima 1980), que aún era lento y estaba limitado como su predecesor al reconocimiento de patrones visuales. Estas arquitecturas de redes neuronales no son consideradas dentro del campo de *Deep Learning*, pero sirvieron como base para las redes neuronales de convolución que son las que se han usado como arquitectura de *Deep Learning* en este trabajo.

Las redes neuronales de convolución (CNN) se han convertido en las más reconocidas y populares en el campo de *Deep Learning*. Históricamente, las CNN fueron inspiradas por el modelo del córtex visual (Fukushima 1980). Esto junto con el trabajo de Yann LeCun y sus asociados (LeCun et al. 1989) originó el uso de las CNNs para problemas relacionados con la visión artificial. En este trabajo de 1989, LeCun incorporó convoluciones en su modelo de 5 capas basado en BP, consiguiendo así un procesamiento más rápido y siendo capaz de obtener más características sobre los datos de entrada que si solo usaba la técnica BP. Aunque el entrenamiento de estos primeros modelos tardaban alrededor de tres días, hoy por hoy los diseños de CNN basados en modelo de LeCun, Le-Net 5 (LeCun, Bottou, et al. 1998), tardan solo unos minutos en el entrenamiento, especialmente si se hace uso del procesamiento en paralelo.

Geoffrey E. Hinton y su grupo extendieron el rango de aplicaciones de las CNN hacia el reconocimiento de voz y los problemas de procesamiento natural del lenguaje (Hinton et al. 2012). Así, las CNNs pronto se convirtieron en el enfoque líder en el campo del procesamiento de imagen y el procesamiento de voz. Actualmente el rango de aplicaciones de las CNNs se extiende a muchas otras aplicaciones, siempre que la información de entrada pueda ser representada o reformulada en un espacio bidimensional o mayor, comúnmente una matriz.

²² Back-Propagation

Por lo tanto, las CNNs se han convertido en las redes neuronales más utilizadas para resolver problemas complejos de *Deep Learning*.

1.6 Estructura de la tesis

La estructuración de esta tesis se ha llevado a cabo en 7 capítulos que abordan todo el contenido de años de trabajo:

- El presente, introducción (Capítulo I), dedicado a presentar las motivaciones, y poner en contexto el estado del arte de las cuatro disciplinas científicas que son utilizadas a lo largo de este trabajo (Ingeniería Neuromórfica, Visión Artificial, Procesamiento de Audio y *Deep Learning*).
- Objetivos (Capítulo II), donde se plantean los objetivos a cubrir con el presente trabajo.
- Visión con computación paralela (Capítulo III): se estudia la adaptación de computación paralela en plataformas con FPGA para CNN así como la evolución a sistemas de aceleración para este tipo de NN. Tras ello, se aporta una solución para cada uno de estos aspectos.
- Integración sensorial (Capítulo IV): se plantean mecanismos que combinan información proveniente de un sensor visual y otro auditivo, con el fin de desarrollar tareas de manera más robusta. Los mecanismos propuestos en este punto no están basado en *Deep Learning*, por lo que no poseen la propiedad de aprender.
- Fusión sensorial (Capítulo V): se estudia el uso de CNN para fusionar la información visual y auditiva con el fin de realizar tareas determinadas. Se plantean dos modelos de CNN para la realización de una tarea en concreto con el fin de estudiar sus resultados.
- Conclusiones (Capítulo VI), donde se exponen los resultados obtenidos tras este trabajo.
- Trabajos futuros (Capítulo VII), que suponen una serie de nuevas líneas de investigación.

II. *Objetivos*

En este trabajo se propone abarcar dos tipos de objetivos que fueron planteados al comienzo del mismo: uno general, que pretenden analizar y estudiar la viabilidad del uso de las redes neuronales de convolución para combinar la información de dos tipos de sensores, audio y vídeo, en el contexto de la fusión sensorial; y otros específicos, centrados en la resolución de problemas particulares relacionados con la temática introducida anteriormente y el diseño de sus correspondientes sistemas reales.

- a) **Objetivo General:** Estudio y análisis de algoritmos de *Deep Learning*, análogos a los sistemas neuronales, para la combinación de información de sensores de visión y audio para fusión sensorial.
 - i) En el cerebro humano, la fusión sensorial se realiza en diferentes zonas. Estudiar algoritmos neuroinspirados con la capacidad de ser entrenados para realizar una tarea concreta.
 - ii) Desarrollar un modelo de CNN capaz de realizar la fusión de la información de dos sensores diferentes.

- b) **Objetivos Específicos:** Implementar mecanismos para la evaluación de algoritmos de *Deep Learning*, en particular las CNN, en plataformas FPGA.
 - i) **Visión con computación paralela:**
 - (1) Estudio de la tecnología para computación paralela, OpenCL, y su aplicación a *Deep Learning*.
 - (2) Estudio del framework para *Deep Learning*, Caffe, y la generación de código OpenCL a partir de la red diseñada.
 - (3) Análisis del funcionamiento y rendimiento en plataformas FPGA.
 - (4) Estudio e implementación de hardware dedicado para la aceleración de algoritmos de *Deep Learning*, en particular las CNN.
 - (5) Análisis del funcionamiento y rendimiento en plataformas SoC.
 - ii) **Integración sensorial:**
 - (1) Estudio de diferentes técnicas para combinar la información de varios sensores aplicables a visión y audio.

- (2) Aplicar algoritmos neuronales para el reconocimiento de audio.
 - (3) Combinar la información de dos tipos de sensores utilizando métodos estadísticos.
 - (4) Análisis del funcionamiento del sistema diseñado mediante diversas pruebas.
- iii) Fusión sensorial:
- (1) Estudio de la adaptación de las CNN para el uso en el campo de la fusión sensorial.
 - (2) Implementación de diferentes métodos de conversión de spikes a frames para su uso en las CNN.
 - (3) Diseño e implementación de un modelo de CNN para la fusión de información de sensores de visión y audio.
 - (4) Análisis del funcionamiento del modelo diseñado mediante diversas pruebas.

III. *Visión con computación paralela*

La visión por computador es una de las disciplinas tecnológicas que más aplicación tiene tanto en ámbitos industriales como domésticos. Desde sus comienzos hasta hoy día ha sufrido un gran progreso, pero es en los últimos años cuando ha conseguido sus mayores avances debido a la gran mejora en la potencia de cálculo de los sistemas computacionales. Una de las disciplinas que más se ha visto favorecida por este avance es la visión artificial mediante *Deep Learning*.

Como se planteó en el capítulo de introducción, el *Deep Learning*, y más concretamente las CNN, son mecanismos, como se ha podido ver en la literatura, capaces de ofrecer excelentes resultados para las tareas de reconocimiento visual tales como clasificación, detección y localización. Estos tipos de algoritmos son costosos computacionalmente por lo que necesitan un hardware adecuado y potente para ejecutar sus instrucciones. Este requerimiento fue una limitación que no permitía el uso de estas soluciones para tareas que necesitaran una latencia de procesamiento pequeña, por lo que se vio mermado su uso en el pasado. Una vez superada esta limitación computacional, la aplicación de las CNN se ha visto incrementada en numerosas tareas de reconocimiento visual.

En cuanto a los avances que han permitido el mayor uso de las CNN, cabe destacar la computación paralela. Las CNN tienen una fase enormemente costosa desde el punto de vista computacional, en la que mediante numerosas iteraciones el modelo se va adaptando usando un algoritmo de reajuste para cumplir con la tarea que se persigue. Esta fase es conocida como etapa de entrenamiento y la mayoría de los frameworks, entre los que podemos destacar Caffe (Jia et al. 2014), debido a su amplio uso, dedicados a trabajar con *Deep Learning*, dan soporte para utilizar coprocesadores como las GPU²³. Con esta particularidad se reduce considerablemente el tiempo de computación de la fase de entrenamiento, gracias al masivo paralelismo que ofrece este tipo de hardware.

²³ Graphic Process Unit

Si bien es útil el uso de la computación paralela basada en GPU para la fase de entrenamiento, cabe reconsiderar su utilidad para la fase de ejecución una vez entrenado el modelo. Los modelos algorítmicos de las CNN pueden ser planteados de tal forma que gran parte de sus operaciones pueden ser ejecutadas simultáneamente. Así pues, la agrupación de los datos de entrada, junto con las operaciones necesarias para su procesado, nos abre la posibilidad de dividir la complejidad algorítmica en numerosos subprocesos para que puedan ser ejecutados en paralelo. En el apartado III.2 de este capítulo se presenta un modelo de implementación paralela de una CNN utilizando OpenCL. En ese apartado se hace uso de un ejemplo de CNN (comúnmente utilizado en la literatura de *Deep Learning*) en el que se explica cada una de sus partes junto con el planteamiento de paralelismo y sus resultados.

III.1 Capas de procesado de una CNN

Las CNN son un caso particular de las NN para aplicaciones de *Deep Learning*, cuya principal diferencia radica en que poseen un tipo de capa de procesado en la que se realiza la operación de convolución. Además de este tipo de capa, posee otras que ayudan a acelerar la fase de entrenamiento así como la de ejecución gracias a la reducción de datos en sus salidas. Por otro lado, en este tipo de redes también se hacen uso de otras capas de procesamiento dedicadas a la clasificación y agrupación de características. A continuación se exponen los diferentes tipos de capas más comunes que podemos encontrar en una CNN.

III.2.1 Capa de convolución

Esta es la capa que da nombre a las CNN, y por definición requiere un filtro (función) con el que convolucionar los datos de entrada. En el procesamiento de imagen estos filtros son matrices en las que sus elementos son conocidos como pesos, y que han sido calculados en la fase de entrenamiento. En una capa de convolución habrá varios de estos filtros que convolucionan los datos de entrada aplicando esas matrices. Los datos de entrada de una capa están agrupados en diferentes características que son conocidas como *feature-maps*. Así pues, cada uno de los filtros de una capa de convolución, recorren todos los *feature-maps* (Ecuación 5) de entrada, convolucionando sus datos (Ecuación 6), para generar los datos de salida.

$$w_{rq} * x_q = z$$

Ecuación 5. Función completa de la capa de convolución

Donde w_{rq} y X_q denotan la matriz de convolución y los datos de entrada respectivamente.

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

Ecuación 6. Función de convolución por cada feature-map

con:

$$\begin{aligned} q &= 1, 2, \dots, Q \\ r &= 1, 2, \dots, R \end{aligned}$$

Donde Q, R son los *feature-maps* de entrada y de salida respectivamente. Los valores de cada uno de los filtros han sido calculados en la fase de entrenamiento utilizando el algoritmo conocido como Backpropagation (LeCun et al. 1989), el cual consiste en la propagación del error calculado en la salida hacia todas las capas anteriores, recalculando y ajustando el valor de esos filtros con el fin de ir minimizando el error.

III.2.2 Capa de pooling

Esta capa es utilizada para acelerar la computación de las CNN para arquitecturas muy complejas, especialmente para aquellas que tienen una gran cantidad de datos en la entrada de cada etapa. Este tipo de capa agrupa los datos de entrada localmente reduciendo el tamaño y la resolución de los *features-maps*, por lo que se reducen también la carga de computación en las siguientes capas de convolución acelerando las operaciones y controlando la sobrealimentación. La capa de *pooling* no se ve afectada en la fase de entrenamiento, ya que siempre aplica la misma función que no posee ningún parámetro variable que ajustar. Las funciones más utilizadas en esta capa son:

- *Max-pooling*

Es la función utilizada por defecto en este tipo de capas y obtiene el valor más representativo de una región de interés reduciendo así su tamaño. Como se puede ver en la Figura 19, cada región, de tamaño 2x2 en este ejemplo, es remplazada por una más pequeña que consiste en el máximo valor de dicha región.

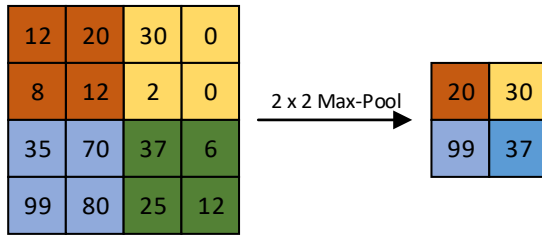


Figura 19. Ejemplo de operación max-pooling

- *Avg-pooling*

Es una alternativa a la anterior, y difiere en la función que aplica para calcular el valor que reemplaza a la región seleccionada. En este caso se utiliza la media aritmética en vez de utilizar el máximo. En la Figura 20 se puede ver un ejemplo de la operación *avg-pooling*.

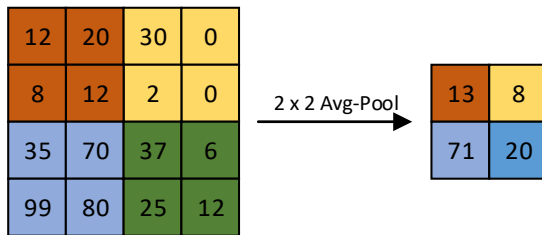


Figura 20. Ejemplo de operación avg-pooling

III.2.3 Capa dropout

Dropout es una técnica introducida en *Deep Learning* por (Srivastava et al. 2014) con el fin de reducir la sobrealimentación de las NN. Esta capa elimina o suprime el número de neuronas en función de un vector de probabilidades. Al ejecutar esta etapa, el vector recorre todos los elementos de entrada reduciendo el número de elementos de salida con el fin de acelerar la computación en las sucesivas capas. Este tipo de capa es utilizada en redes realmente grandes, que poseen un gran número de *feature-maps* tanto de entrada como de salida. En las redes utilizadas en este trabajo no se ha hecho uso de ella.

III.2.4 Capa fully-connected

Este tipo de capa es conocida como la etapa de decisión, y se utiliza para realizar la clasificación final. La función de esta capa es relacionar todos los datos de entrada entre sí, haciendo uso de unos pesos que son calculados y ajustados en la fase de entrenamiento. El número de salidas de esta etapa se corresponde con el número de clases a ser clasificadas.

III.1 OpenCL

El estándar OpenCL para programación paralela ha sido desarrollado por el grupo industrial Khronos con el fin de albergar los retos de la programación “multi-core” y la heterogeneidad de las plataforma. Las especificaciones de este estándar definen un modelo simple de programación y un conjunto de abstracciones a nivel de sistema, las cuales son soportadas por todas las plataformas hardware conformes al estándar. Así pues todo software desarrollado utilizando OpenCL puede ser utilizado entre “*devices*” de distintos fabricantes.

OpenCL proporciona una serie de abstracciones hardware de bajo nivel que permiten desarrollar software sin la necesidad de un completo conocimiento del hardware donde se va a ejecutar ese software. Estas abstracciones son los modelos de plataforma, memoria y ejecución, que son necesarios conocerlos para poder realizar aplicaciones con alto grado de optimización. Es por ello que pasaremos a describir cada uno de ellos sin entrar en profundidad.

III.1.1 Modelo de plataforma

El modelo de plataforma OpenCL define la representación lógica de todo el hardware capaz de ejecutar una aplicación OpenCL. Las plataformas OpenCL son definidas por el conjunto de un “*host*” y uno o más “*device*” de computación OpenCL conectados a través de un bus de alta velocidad como es el PCIe. El “*host*”, donde se ejecuta el sistema operativo, es el responsable de las siguientes tareas:

- Manejar el sistema operativo y habilitar los drivers para todos los “*devices*” OpenCL.
- Ejecutar la aplicación principal.
- Preparar todos los buffers globales en memoria y controlar las transferencias de datos entre el “*host*” y los “*devices*”.

- Monitorizar el estado de todas las unidades de cómputo del sistema.

Por otro lado el *“device”* es el elemento hardware en el que los kernels de la aplicación son ejecutados. Cada *“device”* se divide en un conjunto de *“compute units”*, que a su vez estas son divididas en *“processing elements”*. Un *“processing element”* es la unidad fundamental de computación en una *“compute unit”*, y es responsable de la ejecución de las operaciones de un *work-item*. Un *work-item* es una colección de instrucciones paralelas que se ejecutan en uno o más *“compute unit”*. Más adelante, en el apartado del modelo de ejecución se explicará con un ejemplo que clarificará su definición. La Figura 21 muestra un esquema general del modelo de plataforma OpenCL.

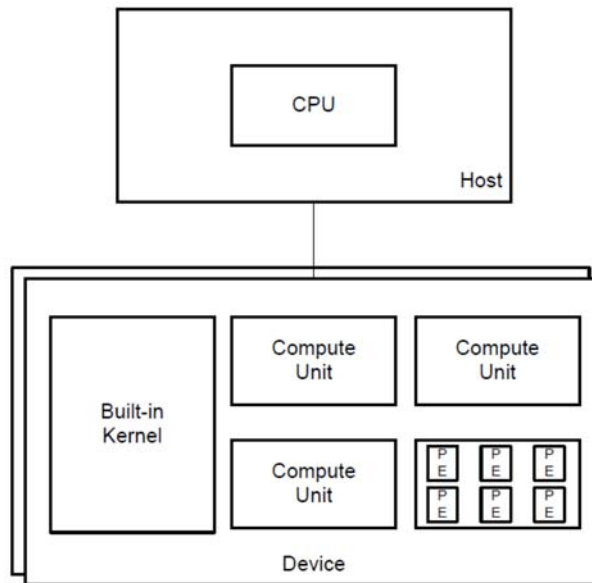


Figura 21. Esquema general del modelo de plataforma OpenCL (Xilinx 2016b)

III.1.2 Modelo de memoria

El modelo de memoria OpenCL define el comportamiento y la jerarquía de memoria que puede ser utilizado por aplicaciones OpenCL. Esta representación jerárquica de la memoria es común para todas las implementaciones OpenCL, pero depende del fabricante del dispositivo definir como el modelo de memoria se mapea en el hardware. Para este trabajo se ha elegido una plataforma hardware de Xilinx, cuya herramienta para trabajar con OpenCL se llama SDAccel, por lo que a continuación se define el mapeo utilizado por esta herramienta.

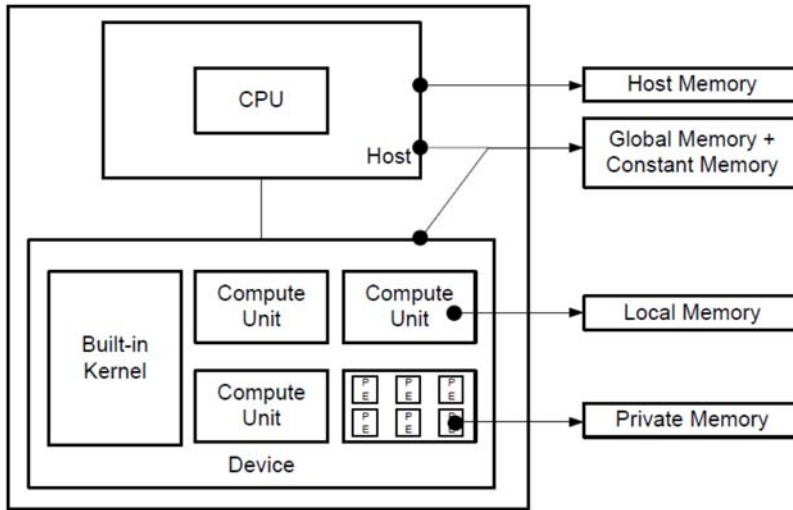


Figura 22. Esquema del modelo de memoria OpenCL (Xilinx 2016b)

III.1.2.1 Memoria del host

La memoria del "host" es definida como la región del sistema de memoria que es directamente (y únicamente) accesible desde el "host". Cualquier dato que los kernels necesiten deben de ser transferidos hacia y desde la memoria global del "device" utilizando la API de OpenCL.

III.1.2.2 Memoria global

La memoria global se define como la región que es accesible en modo lectura y escritura tanto por el "host" como por el "device". En el caso de Xilinx, esta memoria está dividida en dos partes, una externa a la FPGA y otra implementada en el interior de la FPGA. El "host" es responsable de la asignación y des-asignación de los buffers en esta memoria además de transferir los datos de la memoria del "host" a esta. Una vez que el kernel inicia su ejecución, el "host" pierde el acceso a la memoria global tomando el "device" el control sobre ella y permitiendo la lectura/escritura de datos hasta que el kernel finaliza. Una vez completadas las operaciones asociadas al kernel, el "device" devuelve el control de la memoria global al "host" para transferir los datos desde esta memoria a la memoria del "host".

III.1.2.3 Memoria global de constantes

Como su propio nombre indica, esta memoria se utiliza para transferir datos que son constantes desde el "host" hacia el "device". Esta memoria es accesible por el "host" en modo lectura y escritura, y por el "device" sólo en modo lectura.

III.1.2.4 Memoria local

La memoria local es una región de memoria que es local a una unidad de cómputo. No es visible para el "host", y es accesible en modo lectura/escritura por todos los elementos de procesamiento que componen la unidad de cómputo a la que pertenece esa memoria. Esta memoria es utilizada para guardar los datos que son compartidos entre múltiples *work-items*.

III.1.2.5 Memoria privada

La memoria privada es relativa a un *work-item* individual en ejecución. Al igual que la memoria local, ésta no es visible para el "host", y sólo puede ser leída y escrita por el *work-item* actual que se encuentre en el elemento de procesamiento correspondiente a esa memoria.

III.1.2 Modelos de ejecución

El modelo OpenCL de ejecución define como se ejecutan los kernels dentro de un espacio de índices. Este espacio de índices es definido como NDRange, y es el concepto más importante de entender. El espacio de índices NDRange puede estar definido por 1, 2 ó 3 dimensiones. Las funciones u operaciones que componen un kernel son ejecutadas una vez por cada elemento del espacio de índices NDRange. Ese elemento de trabajo de cada coordenada del NDRange se llama *work-item*. Los *work-items* no son programados individualmente para su ejecución, si no que se agrupan formando *work-groups* que sí son programados para su ejecución en las unidades de cómputo.

Cuando un usuario envía un kernel para su ejecución, define también tanto el NDRange (tamaño global, *global_size*) como el tamaño de los *work-groups* (tamaño local, *local_size*). Este proceso se conoce como definición del problema, y una vez realizado el NDRange es dividido automáticamente en *work-groups* y pasan a ser programados para su ejecución en el "device".

La Figura 23 ilustra un ejemplo de un NDRange unidimensional con *global_size* (4096, 1, 1) y *local_size* (512, 1, 1). Esta configuración permite la división de la computación en 8 *work-groups* con 512 *work-items* cada uno. La Figura 24 muestra una definición del NDRange bidimensional con *global_size* (64, 64, 1) y *local_size* (2, 2, 1). En este caso tendremos 1024 *work-groups* con 4 *work-items* cada uno. Por último en la Figura 25 se muestra un ejemplo de un NDRange tridimensional con *global_size* (16, 16, 16) y *local_size* (4, 4, 4). Así pues la división de la computación es en 64 *work-groups* con 64 *work-items* cada uno.

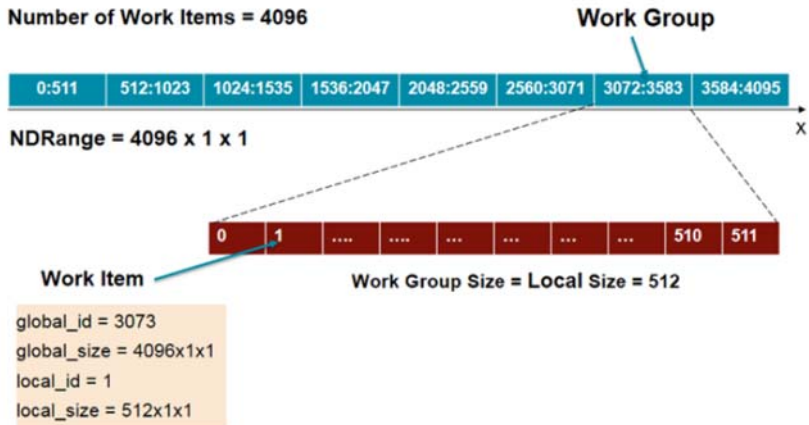


Figura 23. Modelo de ejecución unidimensional (Xilinx 2016a)

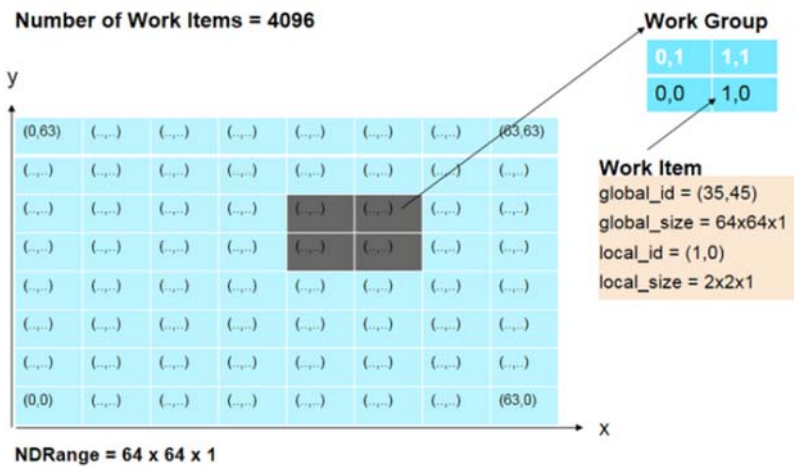


Figura 24. Module de ejecución bidimensional (Xilinx 2016a)

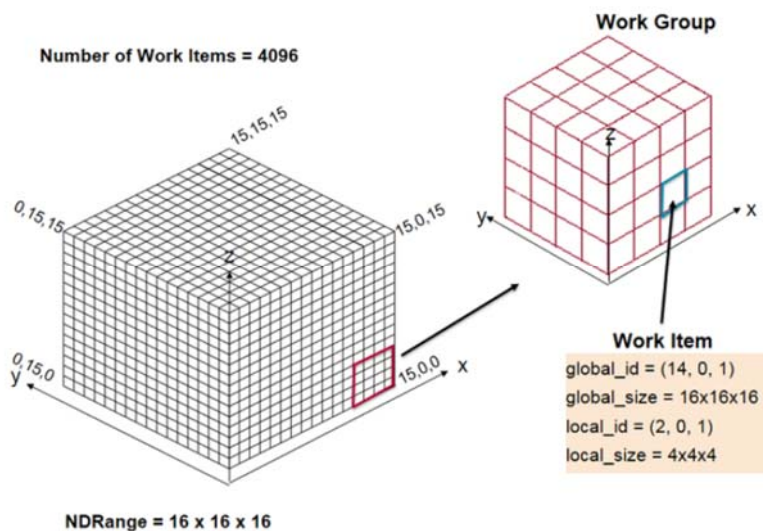


Figura 25. Modelo de ejecución tridimensional (Xilinx 2016a)

III.2 Implementación de CNN utilizando OpenCL

Existen numerosos frameworks para trabajar en el campo de *deep-learning*. Para este trabajo se ha hecho uso de Caffe (Jia et al. 2014), por ser uno de los más utilizados en la literatura científica. Caffe ofrece la posibilidad de describir una CNN de manera sencilla, configurar cada una de las capas que la componen y entrenar dicha red. Además, una vez entrenada la red, se pueden exportar los valores de todos los filtros de convolución que componen la arquitectura descrita y también ofrece la posibilidad de generar código OpenCL en el que se describen todos los kernels que serán implementados para una FPGA.

Para este trabajo se ha utilizado la arquitectura LeNet5, que es una evolución de la arquitectura propuesta en (LeCun, Bottou, et al. 1998) para la clasificación de dígitos manuscritos. Como se puede ver en la Figura 26, esta arquitectura consta de varias etapas de procesamiento (conocidas como capas), siendo dos de convolución, dos de *pooling* y dos *fully-connected*. Esta arquitectura consta de 10 salidas que se corresponden con cada uno de los dígitos a reconocer.

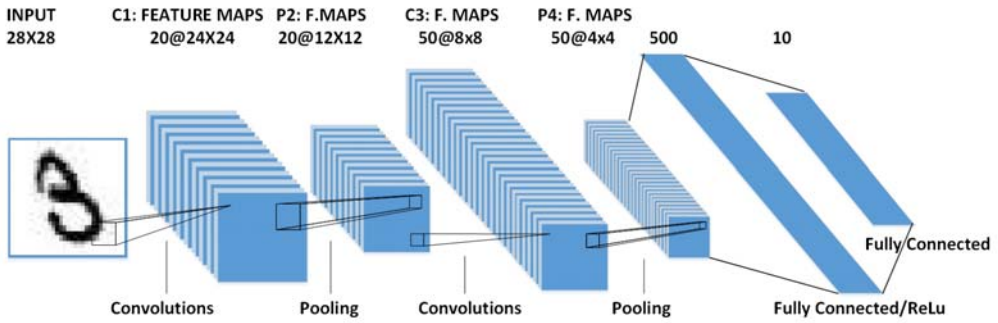


Figura 26. Arquitectura CNN LeNet5

La primera capa de convolución cuenta con un *feature-map* de entrada de tamaño 28x28 píxeles, al que se le aplican 20 filtros de tamaño 5x5 píxeles, obteniendo 20 *feature-maps* de salida de tamaño 24x24 píxeles. Como se puede observar, en las capas de convolución, el tamaño de los *feature-maps* de entrada y de salida son diferentes, siendo los de salidas más pequeños que los de entrada. Esto se debe a que se pierde un borde de tamaño $\lfloor k/2 \rfloor$ siendo k el tamaño en píxeles del filtro de convolución. La siguiente capa es la primera capa de *pooling*, que tras aplicar un filtro de 2x2 píxeles correspondiente a una función de *max-pooling* a los *feature-maps* de entrada de tamaño 24x24 píxeles, se obtienen 20 *feature-maps* de salida de tamaño 12x12 píxeles. Tal y como pasa en las capas de convolución, en las capas de *pooling* también se ve reducido el tamaño de su salida, pero en este caso, cada *feature-map* de salida se reduce $k/2$, siendo k el tamaño en píxeles del filtro aplicado. En este tipo de capa, los filtros aplicados han de ser de tamaño múltiplos de dos. A continuación se aplica otra capa de convolución, con una configuración de 20 *feature-maps* de entrada de tamaño 12x12 píxeles y filtros de 5x5 píxeles obteniendo 50 *feature-maps* de salida de tamaño 8x8 píxeles. Le sigue otra capa de *max-pooling* con filtros de tamaño 2x2 píxeles resultando 50 *features-maps* de salida de tamaño 4x4 píxeles.

A continuación comienza la etapa de clasificación, con una primera capa *fully-connected* de 500 elementos, que a su vez aplican la función no lineal ReLU (ver Ecuación 7). Por último, encontramos otra capa *fully-connected* de 10 elementos que se corresponden con cada una de las clases a clasificar. Estas clases se identifican con los dígitos del 0 al 9.

$$f(x) = \max(0, x)$$

Ecuación 7. Función no lineal de activación ReLU²⁴

²⁴ Rectified linear unit

Tras el diseño, configuración y entrenamiento de la CNN en Caffe, se ha generado el código OpenCL para su implementación en la plataforma Alpha Data ADM-PCIE-7V3 (Alpha Data 2016). El código generado consta de cinco kernels tal y como muestra la Tabla 2.

Capa de la CNN	Kernel OpenCL
Convolución 1 y Pooling 1	conv_pool1
Convolución 2	conv2
Pooling 2	pool2
Fully_connected 1 y ReLU	ip1_relu
Fully_Connected 2	ip2

Tabla 2. Asignación de capas de la CNN LeNet5 a los kernels OpenCL

III.2.1 Optimización de los kernels y ejecución la aplicación

Una vez exportado el código OpenCL del modelo descrito en Caffe, se observa que cada kernel consta de una serie de bucles que recorren los datos de entrada (*feature-maps*), aplicando cada uno de los filtros de la correspondiente capa. Para optimizar y acelerar la computación de cada uno de esos bucles, se incluye una directiva de desenrollado (`__attribute__((opencl_unroll_hint))`) para paralelizar las operaciones internas de cada bucle. Al utilizar directivas de optimización, normalmente el kernel OpenCL se vuelve más complejo y necesita más recursos de la FPGA en términos de LUTs²⁵, DSP²⁶ y BRAM²⁷. En el siguiente apartado de resultados se exponen estos datos de manera cuantitativa.

Otra opción de optimización consiste en replicar el hardware de cada kernel en más de una unidad de computación, tal y como se comentó en el apartado modelos de ejecución. En este caso nos encontramos con una limitación clave de esta implementación de la CNN en OpenCL, que se trata del cuello de botella en el acceso a la memoria global. Al replicar el hardware de cada kernel, el intento de un número masivo de accesos simultáneos a la

²⁵ Lookup Table

²⁶ Digital Signal Processor

²⁷ Block RAM

memoria hace que el rendimiento decaiga considerablemente. En la Figura 27 se puede ver un ejemplo de ejecución de los distintos kernels y sus accesos a memoria.

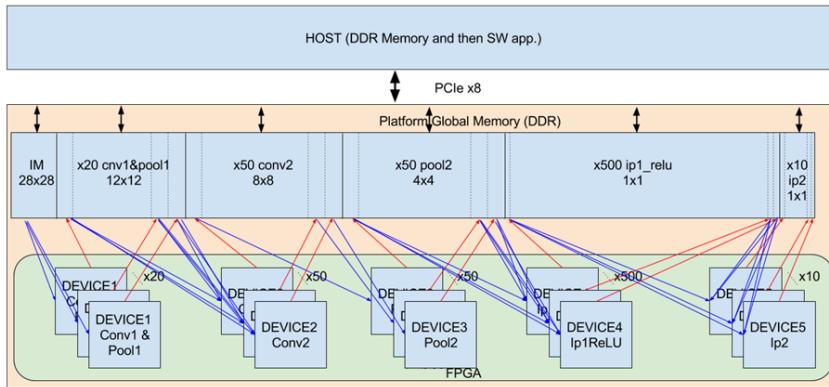


Figura 27. Diagrama de ejecución de los kernels en el dispositivo hardware (Tapiador et al. 2017)

El hilo global de ejecución de la aplicación OpenCL que se ejecuta en el “host” es secuencial. Como se explicó anteriormente, una aplicación OpenCL consta principalmente de dos partes, la aplicación principal y los kernels que se ejecutan en el “device”. La aplicación principal es la responsable de llevar el hilo de ejecución global, colocar en memoria los buffers de entrada y salida de cada kernel y de lanzar la ejecución de cada uno de ellos. Debido a que la aplicación principal ha de esperar a que un kernel termine su ejecución para poder acceder al buffer de sus datos de salida, el siguiente kernel (correspondiente a la siguiente capa de la CNN) no puede ser lanzado hasta que la aplicación principal vuelva a escribir en su buffer de entrada los datos que se encuentran en el buffer de salida del kernel anterior. Por este motivo en esta implementación no sería posible realizar pipeline entre las diferentes capas de la CNN.

III.2.3 Resultados

Para la implementación de esta arquitectura se ha hecho uso de la herramienta SDAccel de Xilinx, que da soporte a la plataforma hardware Alpha Data ADM-PCIE-7V3. Se ha hecho uso de esta plataforma gracias al programa XUP²⁸ por su donación.

La Tabla 3 muestra los recursos hardware necesarios y tipos de ejecución para tres experimentos diferentes, uno sin ningún tipo de paralelismo, otro utilizando la directiva de

²⁸ Xilinx University Program

desenrollado de bucle y finalmente utilizando múltiples unidades de computación por kernels. Los resultados son mostrados en el siguiente orden; sin paralelismo / desenrollado / múltiples UC²⁹. Dichos resultados se corresponden a los tiempos de ejecución, los recursos lógicos (LUT), el número de unidades DSP y los bloques de RAM utilizados por cada kernel. Como se puede observar, en general el tiempo de ejecución es mejorado al emplear más paralelismo hasta el límite. Dicho límite se produce por el cuello de botella que los accesos a la memoria global imponen. En el caso de aumentar el número de unidades de computación no se mejora el tiempo de ejecución alcanzado por la optimización mediante desenrollado.

Nombre del kernel	Tiempo de ejecución (ms)	LUTs (K)	DSP	BRAM (Kb)
conv_pool1	3.63/1.96/1.96	4.9/6.2/5.1	11/11/11	180/216/216
conv2	7.62/4.92/4.92	4.8/4.8/4.9	11/11/11	108/144/144
pool2	0.06/0.03/0.03	3.0/4.0/3.0	4/4/4	72/144/144
ip1_relu	0.55/0.55/0.55	4.2/4.2/4.2	11/11/11	72/72/72
ip2	0.35/0.35/0.35	4.0/4.0/4.0	9/9/9	72/72/72

Tabla 3. Resultados de la ejecución y consumo hardware de la CNN LeNet5 OpenCL

III.3 Aceleradores de CNN (NullHop)

Como se ha comentado en el capítulo de introducción de este trabajo, cada vez más las CNN son usadas para tareas de reconocimiento visual. También se ha explicado a lo largo de estos capítulos que este tipo de algoritmos son computacionalmente costosos, por lo que se requiere de un hardware potente para su procesamiento. Es por ello, que para sistemas empujados, estos algoritmos presentan un difícil reto, ya que normalmente la potencia de cómputo de estos sistemas es relativamente pequeña. Es ahí donde cobra especial interés la creación de un hardware dedicado para resolver las operaciones más complejas de las CNN. Recientemente han ido surgiendo diversos modelos de aceleradores dedicados a procesar las diferentes capas que componen una CNN. Estos aceleradores se diseñan con el fin de

²⁹ Unidades de Computación

conseguir realizar el número máximo de operaciones en el menor tiempo posible, pero siempre teniendo muy presente que el consumo de potencia debe de ser reducido.

III.3.1 NullHop

En el marco del proyecto NPP, se ha desarrollado un acelerador de CNN llamado NullHop, cuya principal característica se basa en procesar sólo aquellos píxeles que no son nulos. La razón de hacer el esfuerzo de diseñar un acelerador que solo procese los píxeles que no son nulos, es debido a que en las capas de procesamiento no lineal de las CNN la operación más utilizada en la ReLU, por lo que a su salida convierte todos los valores negativos a cero. Por tanto el *feature-map* resultante es normalmente disperso. En la Figura 28 se pueden ver dos ejemplos de la dispersión en cada capa de dos CNN muy conocidas en la literatura de *deep-learning*.

El acelerador de CNN NullHop consta principalmente de tres módulos, IDP³⁰, MKM (donde se realiza la computación) y PRE³¹. En la Figura 29 se puede ver el esquema global del acelerador de CNN NullHop. El prototipo, que actualmente se encuentra en funcionamiento, posee un tamaño de palabra de 16 bits (utilizando la codificación Q8.8) y es capaz de acelerar una etapa de convolución. Una etapa de convolución consta de la propia operación de convolución, y opcionalmente las de *max-pooling* y ReLU. Antes de enviar los píxeles de los *feature-maps* de entrada, el acelerador ha de ser configurado con los valores de los filtros para la operación de convolución de la capa actual. Tras realizar la configuración, se envían los píxeles de los *feature-maps* que no son nulos, por lo que es necesario un mecanismo de codificación de dichos píxeles.

³⁰ Input Decoder Processor

³¹ Pooling ReLU Encoding

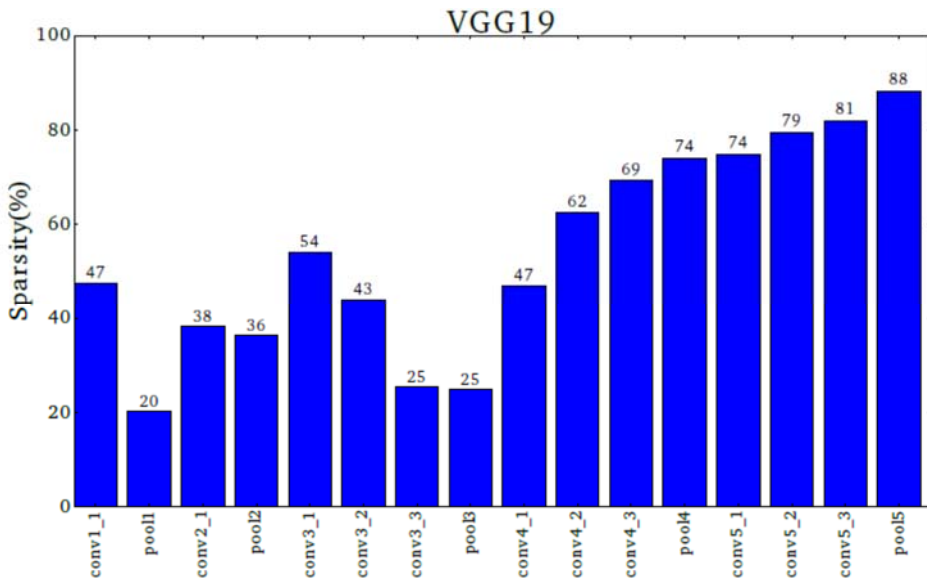
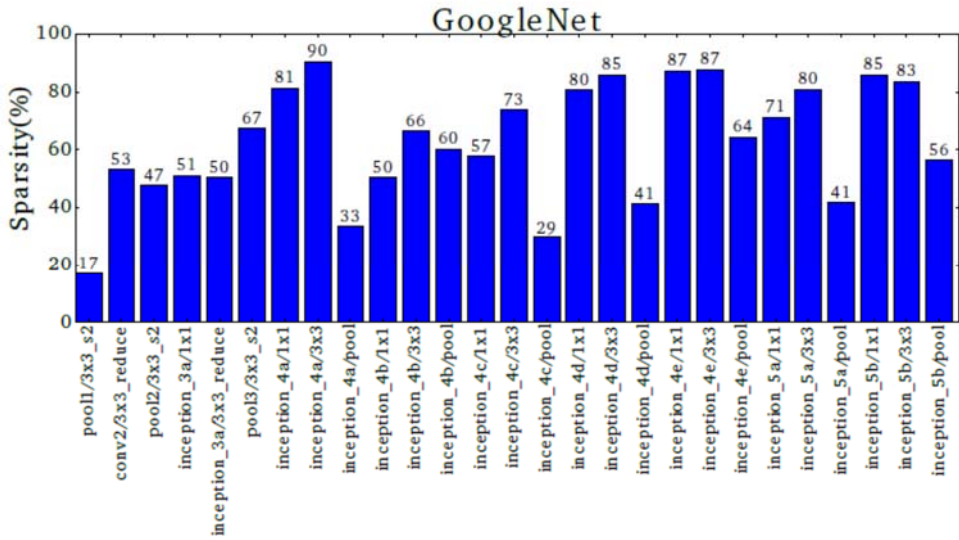


Figura 28. Porcentaje de zeros en cada capa de la CNN GoogleNet (arriba) y la VGG19 (abajo) (Aimar et al. 2017)

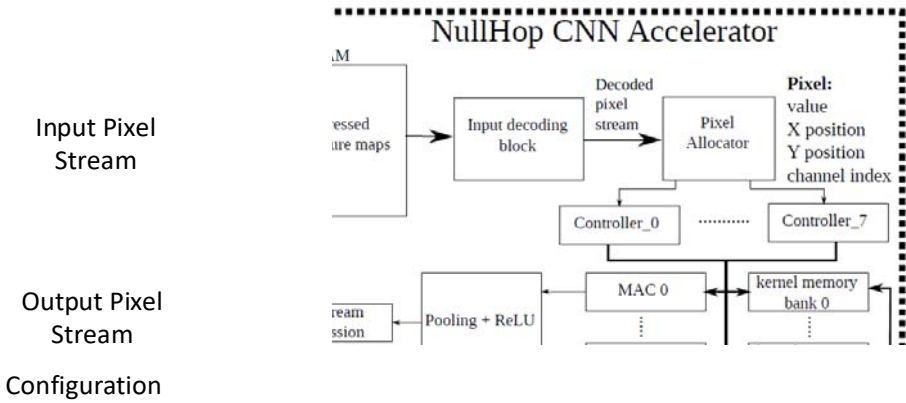


Figura 29. Esquema de global del acelerador de CNN NullHop (Aimar et al. 2017)

Para realizar dicha codificación, se recorren los *feature-maps* de entrada en busca de aquellos valores distintos de cero, creando una lista conocida como NZVL³². Junto a esta lista, se genera un mapa de dispersión, en el que se indica qué píxeles son nulos y cuáles no con un bit por píxel. En la Figura 30 se puede ver una ilustración del mecanismo de compresión.

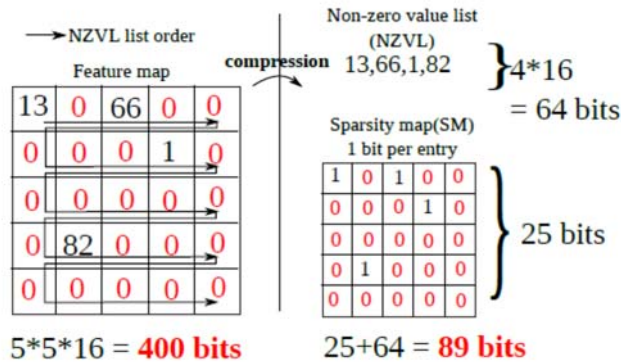


Figura 30. Esquema de compresión utilizando mapa de dispersión (Aimar et al. 2017)

El mapa de dispersión es una palabra de 16 bits en la que se indican que píxeles, de los que vienen a continuación, no son nulos mediante la aparición de un 1 las posiciones correspondientes. Como se puede deducir, cada mapa de dispersión codifica la información de los siguientes 16 píxeles. En el caso que hubiera 16 píxeles consecutivos nulos, aparecería un mapa de dispersión completamente a cero, siendo la siguiente palabra otro mapa de

³² Non Zero Value List

dispersión que codificaría los siguientes 16 píxeles. En la Figura 31 se puede ver dos ejemplos de dos palabras consecutivas que son enviadas al acelerador.

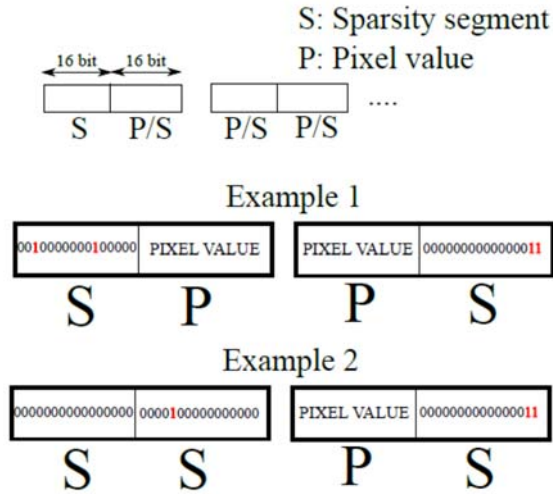


Figura 31. Formato de las palabras enviadas al acelerador (Aimar et al. 2017)

Una vez que los datos son codificados en el formato adecuado, son enviados al acelerador y decodificados por el módulo IDP. Este módulo es el encargado de interpretar los datos codificados a la entrada del procesador y colocarlos correctamente en la memoria de píxeles. Una vez que haya datos suficientes para comenzar la computación, cada uno de los 128 MACs que componen la arquitectura actual, es alimentado con los píxeles y porciones de los filtros correspondientes para ir generando los *feature-maps* de salida. Cada bloque MAC es manejado por un controlador que se encarga de alimentar dicho bloque con los píxeles necesarios así como de cargar la porción de kernel necesaria para cada operación parcial. Una vez se encuentran los datos en el bloque MAC, se realiza la operación de multiplicación y acumulación generando paso a paso el *feature-map* de salida correspondiente. Por último el módulo PRE es el encargado de realizar las operaciones de *pooling* y ReLU (si previamente fueron activadas en la etapa de configuración). El flujo de píxeles de salida está codificado siguiendo el mismo patrón que los datos de entrada. En (Aimar et al. 2017) se describe de forma detallada el funcionamiento interno de este acelerador.

Una de las principales ventajas de la utilización de aceleradores implementados en hardware dedicado frente a la propuesta explicada anteriormente mediante OpenCL es la capacidad de realizar operaciones en pipeline si contamos con más de un chip acelerador. Como se comentó, en la implementación de una CNN con OpenCL exportada desde Caffe

(mapeando capas de la CNN en kernels), es necesario esperar a que un kernel termine su ejecución para poder acceder a sus datos de salida. Estos datos de salida se corresponden con los datos de entrada del siguiente kernel, por lo que este no podría comenzar su ejecución hasta la finalización del anterior. En el caso de los aceleradores, si por ejemplo contamos con tres chips, cada uno es configurado para procesar una capa de la CNN, y son conectados en cascada, donde la entrada del chip $n+1$ es la salida del chip n . Cada chip acelerador es capaz de generar datos de salidas antes de que todos los datos de entrada sean procesados, por lo que se podrá realizar *pipeline* entre los diversos chips reduciendo la latencia entre el procesado por capa.

De lo anteriormente expuesto, se puede ver que la gran limitación de OpenCL, siguiente la estructura de kernels exportada desde Caffe, es el bloqueo de la memoria principal entre el "host" y el "device". Esta limitación hace que los datos no puedan ser enviados del "host" al "device" de manera constante, por lo que han de ser enviados todos al principio y esperar que el correspondiente kernel termine su computación para poder acceder a todos los datos de salida generado por este. Para realizar la operación de convolución, no es necesario tener todos los datos del frame a computar, y además los datos generados como salida son obtenidos de manera progresiva desde que la primera operación es realizada.

Los aceleradores de CNN, tanto NullHop como otros que podemos encontrar en la literatura tales como Origami (Cavigelli, Lukas and Benini 2016), NNX (Gokhale et al. 2014), Eyeriss (Chen et al. 2016) and Caffeine (Zhang et al. 2016) utilizan esas propiedades para aumentar el número de Op/s (operaciones por segundo) así como la capacidad de crear etapas de pipeline entre etapas de convolución. Una de las claves para conseguir esto se basa en la ordenación y secuenciado de los datos de entrada. Como se ha comentado anteriormente, no es necesario tener todos los datos del frame para empezar a realizar operaciones en una etapa de convolución, por lo que si se secuencian los datos de entrada en un cierto orden, es posible ir generando datos de salidas mucho antes que toda el frame sea secuenciado. En la Figura 32 se puede ver como los datos de un frame son secuenciados por filas generando así datos de salida al mismo tiempo que se tienen los datos mínimos para computar la primera operación.

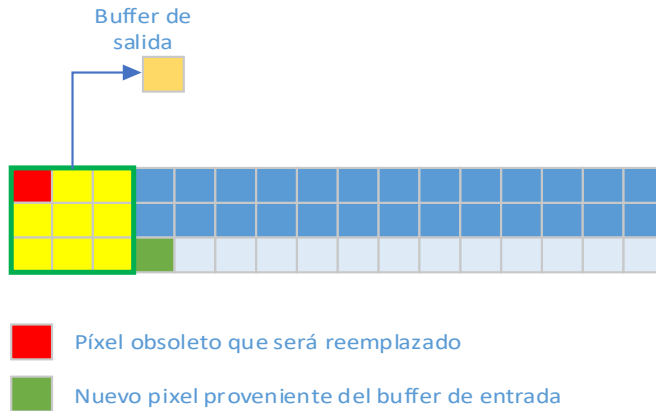


Figura 32. Secuenciación de los datos de entrada/salida por filas en un acelerador de CNN

En la Tabla 4 se muestra una comparativa en cuanto a rendimiento, precisión utilizada, consumo y eficiencia entre NullHop y otros aceleradores que existen en la literatura.

	Eyeriss (Chen et al. 2016)	Origami (Cavigelli, Lukas Benini 2016)	Caffeine (Zhang et al. 2016)	NNx (Gokhale et al. 2014)	(Zhang et al. 2015)	NullHop (Aimar et al. 2017)
Plataforma	ASIC 65nm CMOS	ASIC 65nm CMOS	Ultrascale KU060	Zynq XC7Z045	Virtex-7 VX485T	Zynq XC7Z100
Frecuencia (MHz)	200	250	200	142	100	500
Precisión	16-bits fijo	12-bits fijo	16-bits fijo	16-bits fijo	32-bits flotante	16-bits fijo
Unidades MAC	168	196	1058	800	448	128
Rendimiento (Gops/s)	46.1	145	310	200	61.6	450
Máx. teórico (Gops/s)	67.2	196	423.2	227	89.6	128
Consumo (W)	0.28	0.45	25	12	18.61	0.155
Eficiencia	68.6%	74%	73.3%	88.1%	68.8%	351.6%

Tabla 4. Comparativa de NullHop con otros aceleradores de CNN

III.3.2 Implementación del NullHop en SoC

Para validar el diseño del NullHop se ha implementado en un SoC³³. El chip elegido es la Zynq 7100 de Xilinx, que posee un procesador ARM dual-core Cortex-A9 (PS³⁴ en adelante) y una lógica reprogramable Kintex-7 en el mismo chip (PL³⁵ en adelante). Para hacer uso de este chip se ha elegido la plataforma comercial Zynq 7100 MMP³⁶, que posee 1GByte de memoria DDR2, conexión Ethernet, soporte para una tarjeta SD, interfaz USB OTG entre otros dispositivos.

Para llevar a cabo esta implementación se han identificado las siguientes tareas que han sido repartidas entre los socios INI³⁷ y RTC³⁸ (grupo al que pertenece el autor de este trabajo) del proyecto NPP donde parte de este trabajo se encuentra en marcado:

1. Desarrollo hardware de la arquitectura del NH en RTL³⁹
2. Implementación del controlador software NH
3. Interfaz hardware entre NullHop y el bus AXI-DMA
4. Implementación del controlador software del bus AXI-DMA orientado al uso del NH
5. Módulo hardware de cálculo de latencias entre transferencias AXI-DMA como de latencias de computación del NH

Las tareas 1 y 2 han sido desarrolladas mayoritariamente por el INI, participando el RTC en tareas de despliegue y depuración en la plataforma hardware elegida. El resto de tareas, identificadas como 3,4 y 5 han sido desarrolladas en su totalidad por el RTC, asumiendo el autor del presente trabajo la mayor carga de tareas.

El NullHop es implementado en la PL del SoC junto con la lógica necesaria para realizar la comunicación con el PS. Una serie de máquinas de estados finitas y FIFOs asíncronas conectan el NullHop al bus AXI, que es el usado por el PS para comunicarse con la PL. El bus AXI es un protocolo *open-source* de ARM y se utiliza para conseguir el máximo ancho de banda posible en la comunicación del PS y PL. Este protocolo consiste en un mecanismo de dos pasos

³³ System On Chip

³⁴ Processing System

³⁵ Programmable Logic

³⁶ Mini Module Plus

³⁷ Institute of Neuroinformatics

³⁸ Robotics and Technology of Computers Lab

³⁹ Register Transfer Level

maestro-esclavo. En el primer paso, el maestro envía la dirección del esclavo junto con la señal de control de lectura/escritura. Luego, el dispositivo maestro (operación de escritura) o esclavo (operación de lectura) contesta con el dato correspondiente. Debido a la gran cantidad de flujo requerido por el NullHop, con tamaños de transferencia variables por cada capa, y para reducir la latencia del protocolo, se ha seleccionado la modalidad AXI-Stream con DMA⁴⁰. Además se ha habilitado el controlador USB para poder utilizar sensores de visión neuromórficos (Berner et al. 2013) en demostraciones de tiempo real.

La Figura 33 muestra el diagrama de bloque de la arquitectura alojada en la PL. El NullHop recibe los parámetros de configuración y los píxeles de entrada desde el espacio de memoria donde el PS los coloca a través del canal MM2S⁴¹ del AXI-DMA, y los píxeles de salida del NullHop resultantes son enviados a través del canal S2MM⁴² hacia otro espacio de memoria del PS. Los datos de configuración representan la parametrización del NullHop y los datos de los filtros de convolución para la capa actual. La lógica que conecta el AXI-DMA con el NullHop está compuesta por dos módulos, el MM2S2NH, que decodifica cada palabra de 64bits procedente del AXI-DMA en cada entrada del NullHop, y el NH2S2MM, que codifica la salida del NullHop en la palabra de 64bits del AXI-DMA.

Debido a que la lógica del AXI-DMA puede trabajar con un reloj más rápido que el NullHop, cada uno de estos módulos consta de una FIFO asimétrica, la cual posee dos dominios de reloj diferentes. El tamaño de estas FIFOs es de 512 palabras de 64bits, está fijado de esta forma para aprovechar el consumo de recursos pero evitando bloqueos. En la API implementada para manejar las comunicaciones entre el PS y la PL que corre en una distribución especial de Linux llamada Petalinux, se hace uso de un espacio de memoria de hasta 8MBytes para las transferencias de escritura (desde PS a PL), mientras que para las lecturas (desde PL a PS) el espacio de memoria es de 4KBytes, ambos configurables en tamaño.

⁴⁰ Direct Memory Access

⁴¹ Memory Mapped to Stream

⁴² Stream to Memory Mapped

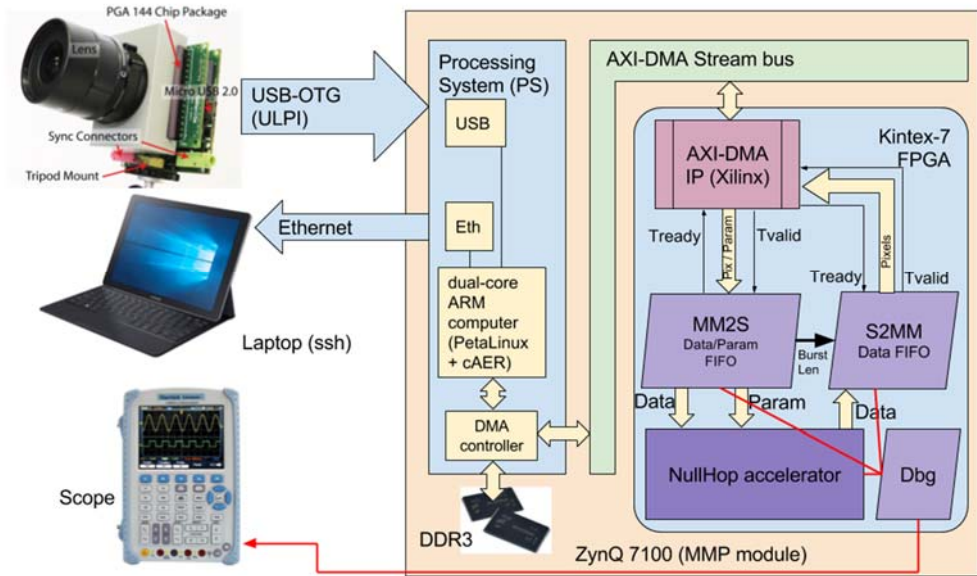


Figura 33. Diagrama de bloques del escenario de test implementado en el SoC

En caso de saturación, tanto en los datos de salida como en los de entrada del NullHop, dos señales de control, *Tready* y *Tvalid*, se activan para controlar el flujo de datos sin pérdidas. *Tready* es manejada por el módulo que recibe la transacción de datos, indicando si está o no preparado para ello, mientras que *Tvalid* es manejada por el módulo que realiza la transacción, activando esta señal cada vez que haya un dato válido en el bus.

Recursos	Lógica	FF ⁴³	BRAM	DSP
Depuración	0.14%	0.1%	4.30%	0%
AxiStream	1.25%	0.73%	1.46%	0%
IDP	7.55%	0.41%	17.22%	0%
MAC	50.19%	16.99%	33.9%	6.30%
PRE	16.41%	1.88%	0%	0%

Tabla 5. Consumo de recursos de la Zynq 7100 necesarios para la implementación del NullHop

Después de la síntesis e implementación de toda la arquitectura para nuestra Zynq 7100, NullHop junto con las interfaces entre la PL y el PS, se ha conseguido una frecuencia de reloj

⁴³ Flip Flops

máxima de 60MHz, identificando como rutas más críticas las relacionadas con los accesos a los bloques de memoria y el rutado de los datos a los bloques MACs. Esta implementación del NullHop consta de 128 bloques MACs, capacidad para 512 *feature-maps* de entrada/salida de un tamaño máximo de 512 píxeles de alto y ancho. La Tabla 5 muestra la utilización de recursos de la FPGA necesaria para esta implementación.

En cuanto a la potencia consumida estimada, el sistema (SoC) posee un consumo estático de 317mW del SoC, mientras que el consumo dinámico es de 2006mW. El consumo dinámico de la Zynq es compartido por el PS y los recursos consumidos de la PL. El PS consume el 74% del consumo dinámico total, mientras que el 26% es empleado en la PL, distribuido entre sus componentes como la lógica (2%), el reloj (13%), las señales (7%), los bloques BRAM (2%), DSP y la entrada/salida (ambos < 1%).

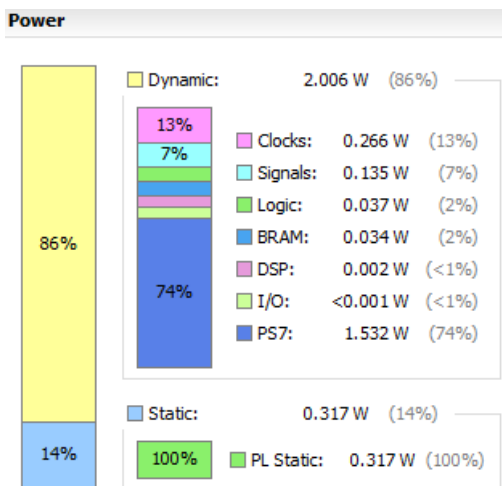


Figura 34. Resultado de la estimación de consumo del NullHop utilizando la herramienta XPower de Vivado

Como se comentó anteriormente, la plataforma comercial utilizada para realizar esta implementación es la Zynq 7100 MMP (Avnet 2016) que cuenta recursos adicionales (DDR3, USB Host, Ethernet, SD, entre otros) a parte del SoC. Es por ello que se ha realizado también un estudio de consumo real, teniendo en cuenta diferentes estados del sistema. En la Tabla 6 se pueden ver las diferentes medidas para los diferentes estados que presentan interés para hacer una comparativa con el consumo estimado anteriormente.

#	Medición real	Potencia (mW)
1	Placa base sin MMP	5146
2	#1 + ventilador del SoC	5773
3	#2 + MMP sin programar en reset	6956
4	#2 + MMP sin programar	7126
5	#2 + MMP programada con NH en reposo	8274
6	#5 + funcionando la red RoShambo	9032

Tabla 6. Mediciones de consumo real de la plataforma Zynq 7100 MMP

El estado #3 implica que la FPGA no está programada, los cores ARM se encuentran en IDLE y la DDR, la interfaz Ethernet y USB están en funcionamiento. El consumo dinámico se corresponde con la diferencia de potencia del estado #6 y #3, siendo de 2076mW, que está muy próximo al obtenido en la estimación con XPower (2006mW). En cuanto al consumo del sistema en funcionamiento con respecto a reposo se obtienen 758mW (#6 - #5).

III.3.3 Resultados

Para probar la implementación del NullHop desarrollada para el SoC, se ha ideado una demostración en la que se ha diseñado y entrenado una CNN para reconocer los gestos de una mano mostrando piedra, papel y tijeras (Aimar et al. 2016). La red es acelerada con el NullHop, por lo que una vez detectado el gesto mostrado por el usuario, el sistema encenderá un led correspondiente al gesto que gana al usuario, siguiendo las reglas del tradicional juego piedra-papel-tijeras, conocido como RoShambo.

Para entrenar la red se han realizado grabaciones a diferentes personas mostrando frente a un sensor DAVIS240 los tres gestos a reconocer. Este tipo de sensor se corresponde a un sensor neuromórfico de visión dinámica, explicado en el apartado 1.3.2. Para la generación de los frames, se han realizado acumulaciones de 2000 eventos generando así un histograma que se corresponde con el frame. Una vez capturados los eventos se genera un fichero de etiqueta donde se realiza la correspondencia del frame con la clase que representa. Se ha añadido una clase extra, identificada como "fondo", que se corresponde con la no identificación de ninguna

de las tres clases anteriores. El diseño y entrenamiento de la red se ha realizado en Caffe, consiguiendo un 96% de aciertos. La configuración de la red se muestra en la Tabla 7 donde después de cada capa de convolución está la función ReLU activada y consta de otra capa de *max-pooling* donde se aplica un filtro de tamaño 2x2 píxeles.

Etapa de convolución	1	2	3	4	5
Tamaño canal de entrada (píxeles)	64 x 64	30 x 30	14 x 14	6 x 6	2 x 2
Número de canales de entrada	1	16	32	64	128
Número de canales de salida	16	32	64	128	128
Tamaño filtro de convolución (píxeles)	5 x 5	3 x 3	3 x 3	3 x 3	1 x 1
2x2 max-pooling	Sí	Sí	Sí	Sí	Sí
ReLU	Sí	Sí	Sí	Sí	Sí

Tabla 7. Configuración de la red RoShambo

Tras la última etapa de convolución, la red cuenta con una capa *fully-connected* con 128 canales de entrada de tamaño de 1 píxel, y 4 canales de salida correspondiente a las 4 clases a identificar, piedra, papel, tijeras y fondo.

Para estudiar el rendimiento del acelerador, con una frecuencia de reloj de 60MHz para la lógica y de 666 MHz para el ARM, se ha medido el tiempo de un frame procesado por cada una de las capas de la CNN. En la Figura 35 se pueden ver los tiempos de procesado de cada una de las capas junto con el tiempo de las transferencias y el necesario por el software para preparar los datos de una capa para otra. En la primera capa se puede ver un tiempo considerablemente mayor que para el resto, pero ese tiempo es debido al usado por el software que integra los eventos procedentes del sensor DAVIS para la generación del nuevo

frame. Como se puede observar en dicha figura, el tiempo total para procesar un frame son 8ms, con lo que se conseguiría un frame rate de 125 fps⁴⁴. La red utilizada en esta prueba, RoShambo, tiene un total de 18MOp, por lo que el rendimiento efectivo del sistema sería de 2.25Gop/s. Si se tiene en cuenta sólo el tiempo que los bloques MACs están en funcionamiento, conocido como computación activa, se obtendrían 5.6GOp/s. La computación máxima del acelerador NullHop sería 15.4GOp/s, resultado correspondiente a la Ecuación 8.

$$\frac{128MACs * 2 \frac{op}{MAC}}{60MHz} = 15.36Gops$$

Ecuación 8. Cálculo del rendimiento máximo del NullHop en Gops

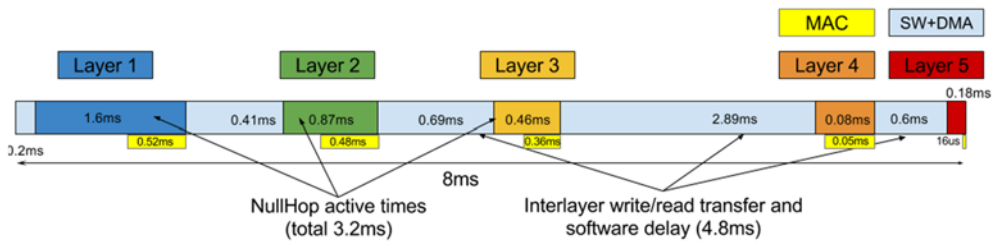


Figura 35. Esquema temporal del procesado de un frame en la plataforma Zynq para la red RoShambo

Para la obtención de los resultados temporales anteriores, se ha hecho uso de la implementación del módulo hardware de latencias correspondiente a la tarea número 5 citada en el apartado anterior. Este módulo se corresponde con el esquema mostrado en la Figura 36 donde cada una de sus salidas indica la latencia de distintas operaciones:

- `nh_idle`: Esta señal indica el tiempo de procesamiento del NH
- `comp_time`: Tiempo desde el primer dato recibido en el bus MM2S hasta el último dato escrito en el bus S2MM
- `frame_comp_time`: Tiempo desde que el NH recibió el primer píxel hasta que termina el procesamiento del último píxel del frame.
- `nh_out_time`: Tiempo desde que se recibe el primer píxel hasta que aparece en el bus de salida del NH el primer píxel como resultado.
- `nh_comp_time`: Tiempo desde que el NH recibió el primer parámetro de configuración hasta que termina el procesamiento del último píxel del frame.

⁴⁴ Frames por segundos

- mm2_latency: Tiempo entre dos transferencias PS → PL
- s2mm_latency: Tiempo entre dos transferencias PL → PS

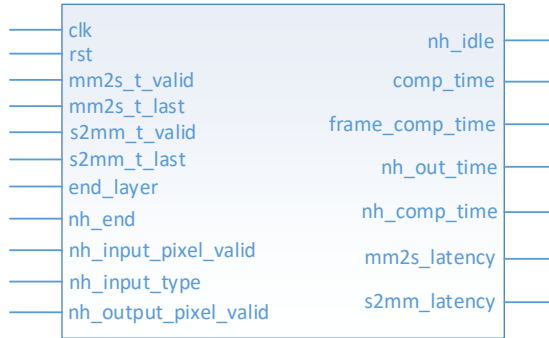


Figura 36. Esquema del módulo hardware de medición de latencias

En la Figura 37 se muestra la señal de `nh_idle` y la señal `mm2s_latency` durante la computación de un frame en la red RoShambo que cuenta con 5 capas de convolución. Se observa como la señal de `nh_idle` se activa (`nd_idle = 0`) al recibir los primeros datos, que se corresponden con los datos de configuración para la capa actual que será computada. Cuando la señal de `nh_idle` se desactiva (`nd_idle = 0`) se debe a que el NullHop ha terminado de computar la capa actual. Se puede ver como por cada capa se realizan dos transferencias de escrituras, correspondientes a los parámetros de configuración de la capa y a los píxeles. Por ejemplo, para la capa primera, se ve en la señal `mm2s_latency` dos activaciones identificadas como "L1 config" y "L1 Data" correspondientes a los datos de configuración y píxeles respectivamente.

Por otro lado, en la Figura 38, se observan las activaciones de la señal `s2mm_latency` que se corresponden con las transferencias de lectura, donde los datos de salida del NullHop son enviados al ARM. Estos datos pueden ser utilizados para realimentar la siguiente capa de convolución, una vez que el acelerador haya sido reconfigurado para la ejecución de tal capa, o para alimentar a las capas *fully_connected* que están implementadas en software. En ambas figuras se puede observar que el tiempo de computación de un frame es de 8ms.

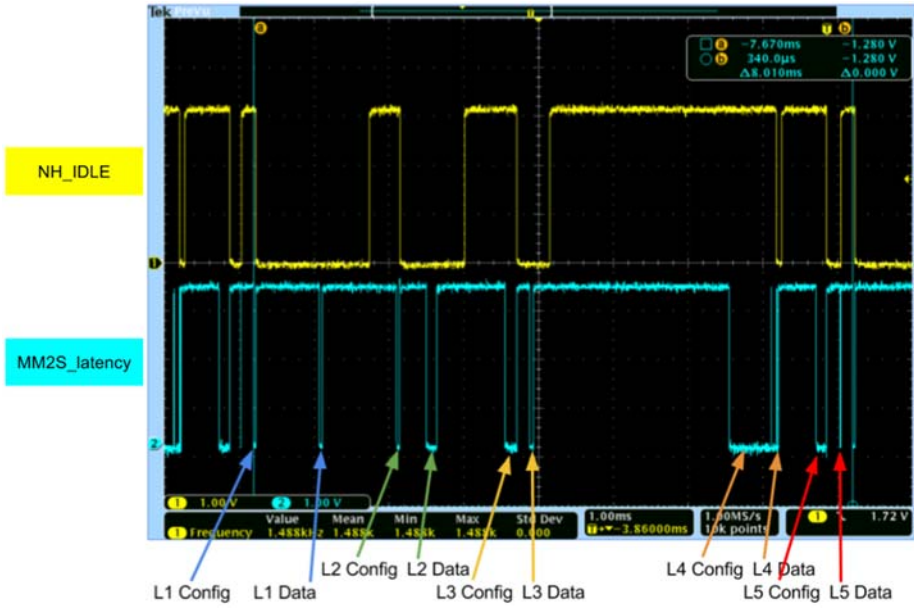


Figura 37. Captura de la señal *nh_idle* y *mm2s_latency* en el osciloscopio para la computación de un frame

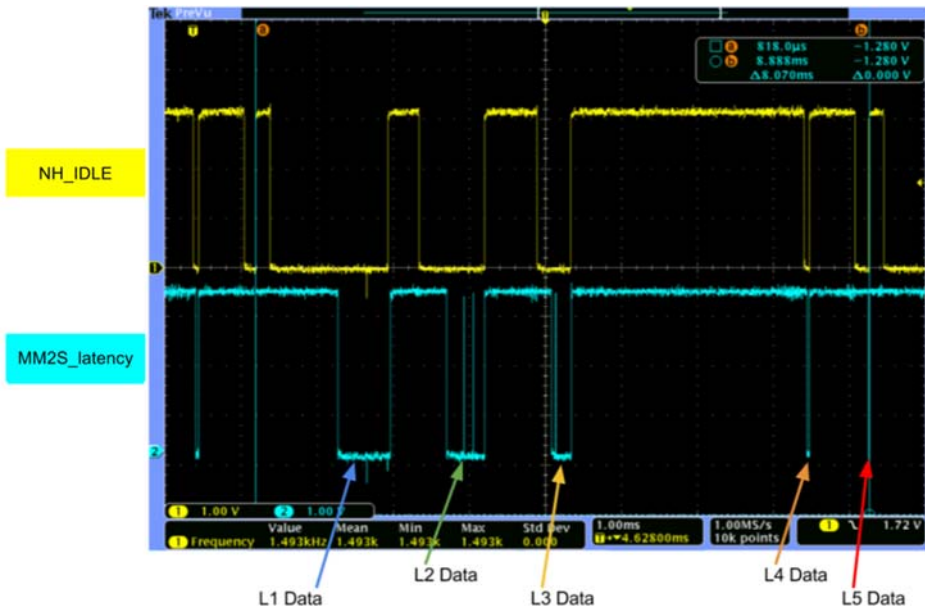


Figura 38. Captura de la señal *nh_idle* y *s2mm_latency* en el osciloscopio para la computación de un frame

Además de los tiempos medidos y representados en la Figura 35, se ha querido estudiar los tiempos necesarios para preparar los datos en los buffers de memoria utilizados para las transferencias de datos entre el PS y la PL. En la Figura 39 se muestra un esquema de cómo los datos son transferidos desde el PS a la PL y viceversa.

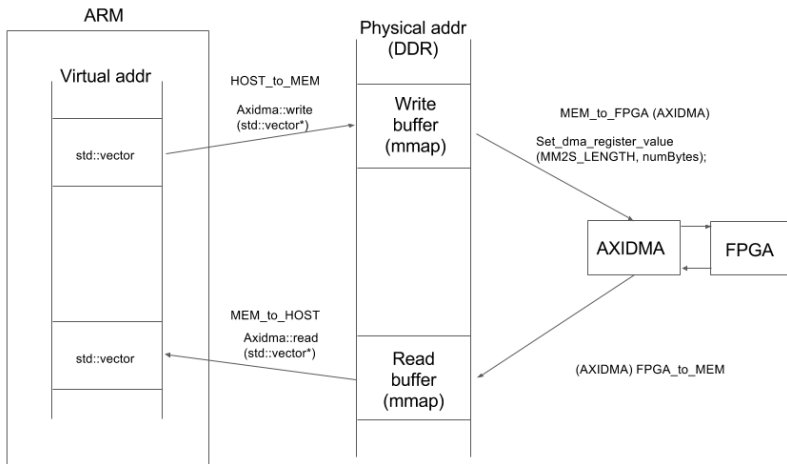


Figura 39. Transferencias de datos PS → PL y viceversa

Como se puede ver en la figura anterior, existen dos buffers (zonas de memoria) reservados para realizar las transferencias de datos. La zona identificada como *write_buffer* se utiliza para colocar los datos que quieren ser enviados desde el PS a la PL. Así pues, el proceso que se está ejecutando en el *host* (ARM) coloca los datos en este buffer y lanza la operación de escritura. Realmente, las operaciones de escritura son encoladas en una lista y se van realizando de manera secuencial sin realizar bloqueos en el hilo principal del proceso. Por otro lado, la zona identificada como *read_buffer* se utiliza para colocar los datos provenientes de la PL, que son leídos por el proceso principal una vez la transferencia ha sido realizada.

En la Figura 39 se pueden ver dos zonas de transferencias, una identificada como *HOST_to_MEM* y otra como *MEM_to_HOST*. Estas transferencias son las que se realizan entre la memoria virtual del proceso que se encuentra en ejecución y los buffer que están alojados en la memoria principal. Estos tiempos están contemplados en las zonas marcadas como *SW+DMA* de la Figura 35, y se han medidos para ser comparados a nivel de software con los tiempos de transferencias entre la memoria principal y la FPGA. El fin de este estudio es aportar mejoras para disminuir estas latencias favoreciendo el rendimiento del procesado de un frame en el acelerador.

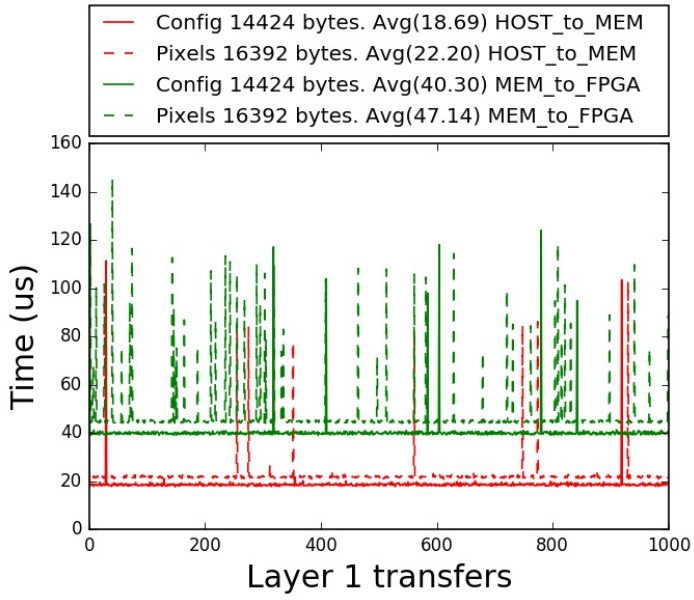


Figura 40. Transferencias de escritura de la capa 1

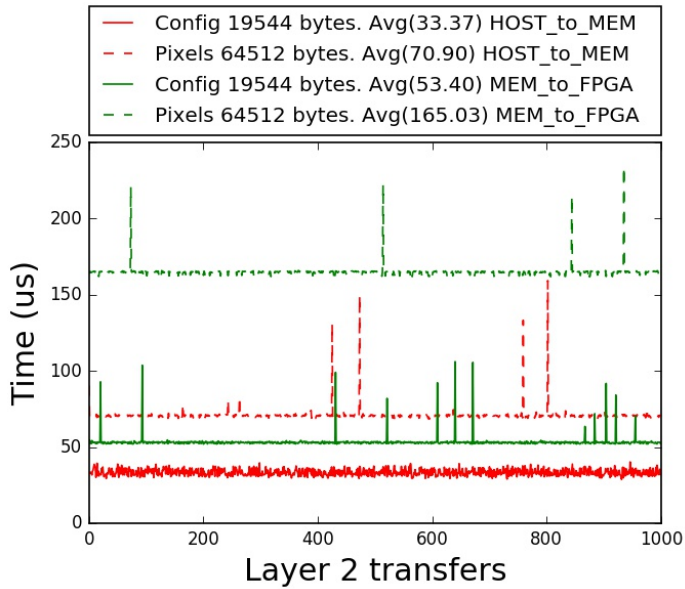


Figura 41. Transferencias de escritura de la capa 2

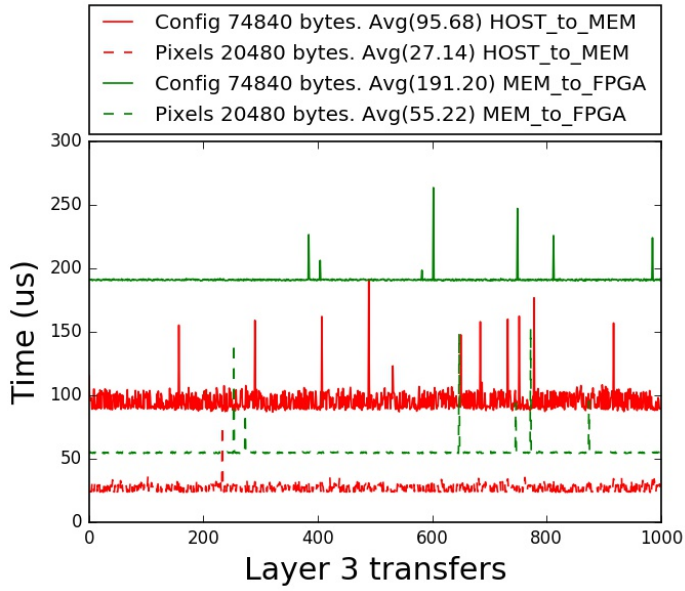


Figura 42. Transferencias de escritura de la capa 3

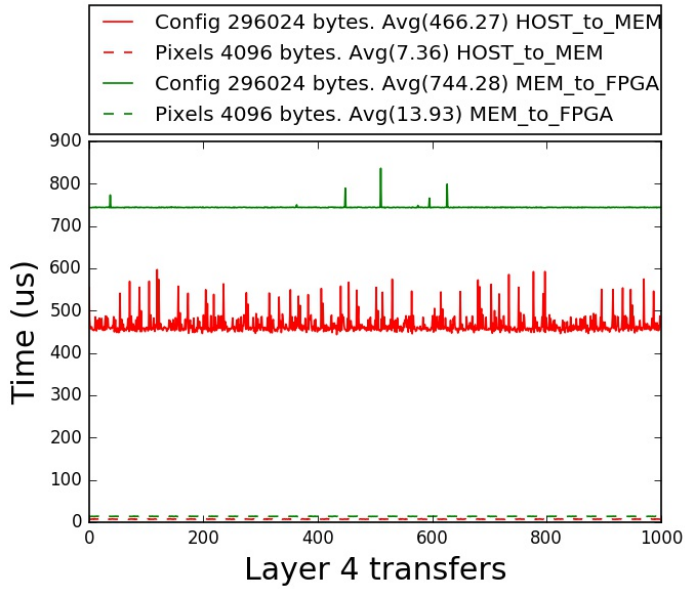


Figura 43. Transferencias de escritura de la capa 4

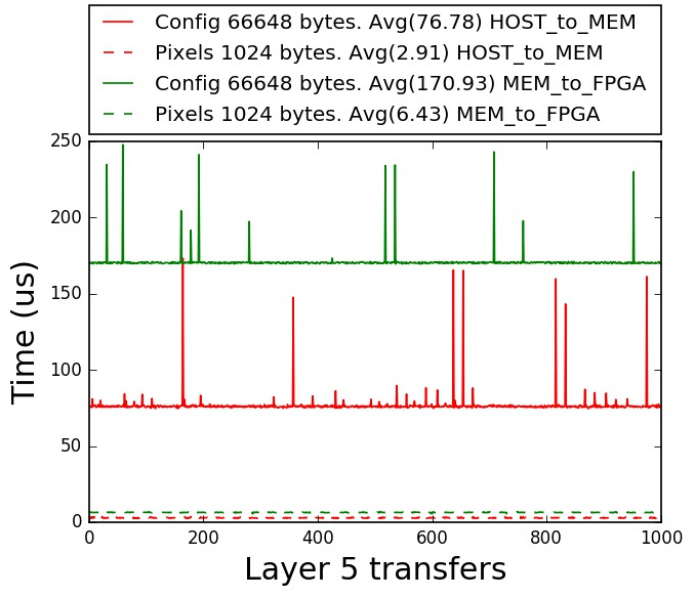


Figura 44. Transferencias de escritura de la capa 5

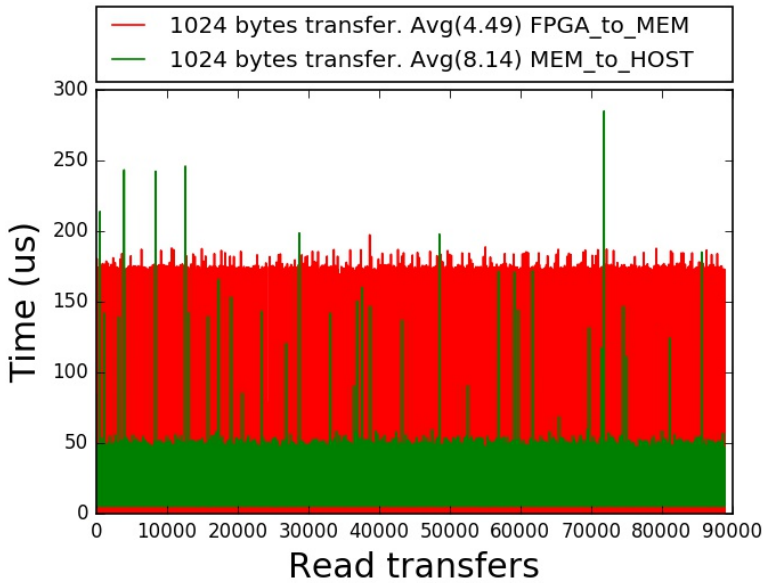


Figura 45. Transferencias de lectura d fijadas a 1KB/transfer

Las figuras (40-45) muestran la tasa de tiempo necesaria para cada tramo de transferencias mostrados en la Figura 39. Estas figuras se corresponden con el procesamiento de cada una de las capas de la red RoShambo que fue descrita anteriormente.

Tras ver los resultados obtenidos tras el estudio de latencias entre las diferentes capas que son procesadas en el acelerador, aunque se consigue un buen número de Ops/s, existen algunas modificaciones que reducirían algunas latencias beneficiando así el rendimiento del acelerador. En la Figura 35 se puede ver cómo la primera capa necesita mucho tiempo de computación, pero la mayor parte de ese tiempo es tomado por un proceso llamado cAER (Longinotti & Brändli 2014) que es el encargado de recoger la información del sensor visual, integrar el histograma de eventos y normalizarlo. Este proceso puede ser sustituido por un módulo hardware que tome como entrada la salida paralela del sensor de visión y realice todo ese procedimiento en hardware, disminuyendo considerablemente su latencia.

Por otro lado, en el prototipo actual sólo existe un dominio de reloj, fijado a 60MHz que es la frecuencia máxima conseguida para la implementación en FPGA del NullHop. La idea es utilizar dos dominios de reloj diferentes, el de 60MHz para la lógica del NullHop y otro más rápido para el resto de la lógica, donde se incluyen los módulos de adaptación que funcionan de interfaz entre el NullHop y el bus AXIDMA. Junto con esta solución se está diseñando un nuevo controlador software de la librería *axidma* con el que establecer un nuevo mapa de memoria donde habrá un buffer por cada conjunto de parámetros de configuración perteneciente a una misma capa, además de un doble buffer de escritura/lectura para los datos correspondiente a los píxeles de entrada/salida de cada capa a procesar. En la Figura 46 se puede ver el nuevo esquema de memoria a implementar. Con esto se reducirían considerablemente las transferencias entre la memoria virtual del proceso y la memoria principal.

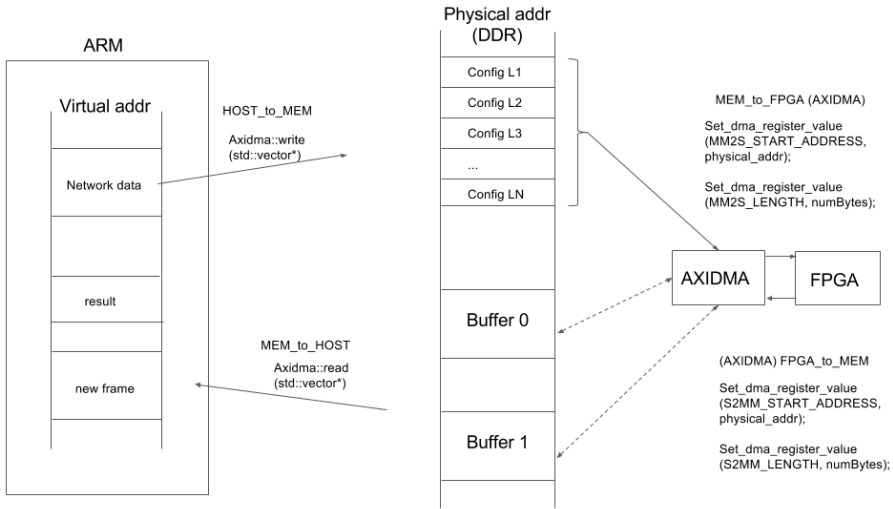


Figura 46. Transferencias de datos PS → PL y viceversa. Implementación de doble buffer

Además de estas mejoras también se propone la posibilidad de implementar un escenario multi-chip de aceleración, haciendo escalable el sistema como un sistema multi-procesador. Con ello se explotará la capacidad de pipeline comentada al principio de este apartado que estos sistemas aceleradores nos brindan. Para ello se ha diseñado y fabricado la plataforma Zynq_Dock (ver Figura 48) que funciona como interfaz entre las plataformas MMP y la AERNode, utilizada para implementar sistemas basados en eventos como (Zamarreno-Ramos et al. 2013). La plataforma AERNode cuenta con una FPGA Spartan 6, dos conectores de 40 pines y 4 conectores LVDS y se utilizarán como elementos de interconexión entre los diferentes NullHops.



Figura 47. Arquitectura multi-core HullHop

Como se puede ver en la Figura 47, al conectar por ejemplo 3 NullHops, mediante las plataformas AERNode que actúan como routers, de tal forma que la salida de uno se conecte directamente a la entrada del siguiente. Así pues el NH0 estaría implementado en la plataforma Zynq 7100 MMP (SoC), descrita anteriormente, mientras que el NH1 y el NH2 se

implementarán en la plataforma Kintex 7 MMP (FPGA), mientras que en las diferentes plataformas AERNodes conectadas a cada NullHop se implementaría la interfaz con el LVDS y la lógica de rutado de datos.

En la Figura 47 hay una serie de puntos clave identificados del 1-7 donde ocurren los siguientes actos:

1. Se configuran los chips de NH 0,1 y 2 para las capas 1,2 y 3 de la red respectivamente.
2. Se escriben los datos correspondientes a los píxeles de la capa 1 al NH0.
3. El NH0 comienza a generar píxeles en su salida que son enviados directamente al NH1.
4. El NH1 comienza a generar píxeles en su salida que son enviados directamente al NH2. Además se escriben los datos de configuración de la capa 4 en el NH0.
5. El NH2 comienza a generar píxeles en su salida que son enviados directamente al NH0. Además se escriben los datos de configuración de la capa 5 en el NH1.
6. El NH0 comienza a generar datos que son enviados al NH1.
7. El NH1 comienza a generar datos en su salida que se corresponden a la última capa de la red. En este caso los datos son enviados de vuelta al PS.

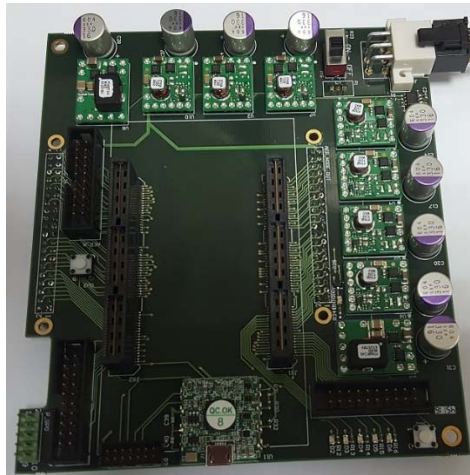


Figura 48. Plataforma Zynq_Dock. Interfaz entre la plataforma Zynq 7100 MMP y AERNode

IV. Integración sensorial

El ser humano posee cinco sentidos que le permite obtener información del entorno que le rodea para ser capaz de moverse e interactuar con él. De manera análoga, en el campo de la robótica, los sistemas que se diseñan normalmente poseen más de un tipo de sensor, como pueden ser cámaras de visión RGB⁴⁵, cámaras de profundidad, sensores de ultrasonido, sensores térmicos,... que permiten al robot recolectar una gran cantidad de información con el fin de procesarla para realizar tareas más complejas y precisas.

En el campo de la visión, existen ejemplos en los que se combinan información de dos tipos de cámaras diferentes con el fin de aportar un resultado con más valor que aquel resultante de una sola cámara. Por ejemplo, la combinación de una imagen RGB junto con una imagen procedente de una cámara térmica, nos aporta una imagen en la que se podría identificar de manera sencilla la diferencia de temperaturas de los objetos captados en la escena. Otro ejemplo bastante común es la combinación de una cámara RGB con una cámara de profundidad, donde la salida resultante de su combinación sería un mapa tridimensional de toda la escena, tal como ocurre con el dispositivo sensorial Kinect de Microsoft.

Combinando varias modalidades sensoriales, un robot puede operar en un amplio rango de entornos tomando ventaja del uso de diferentes sensores. En situaciones donde uno de los sensores esté ausente, el sistema puede continuar operando haciendo uso del resto de sensores. Por otro lado, en situaciones donde múltiples sensores están disponibles, la precisión y robustez del sistema es mejorada combinando la información de los diferentes sensores. Además, es posible obtener información que no sería posible de obtener por un solo sensor, como por ejemplo una IMU.

Tradicionalmente, en la literatura se encuentran dos grandes metodologías para abordar el problema de combinar la información de varios sensores para la identificación de patrones, las basadas en métodos estadísticos y las que se basan en redes neuronales (Hall & Llinas 1997; Luo et al. 2002). Con respecto a las primeras, las diferentes versiones de filtros de

⁴⁵ Red Green Blue

Kalman son las más usadas y extendidas, pero no se realizan a nivel cognitivo por lo que son consideradas que integran la información de los sensores. Respecto a las segundas, aunque son menos usadas en sistemas industriales debido a la necesidad de entrenamiento, han sido estudiadas desde el origen de las redes neuronales (Haykin 1994) y realizan el procesamiento a nivel cognitivo considerando que fusionan la información de los sensores. Existen reportados sistemas de fusión sensorial audio-visual a nivel cognitivo (Chan et al. 2012; O'Connor et al. 2013) y de integración de sensores por eventos (Rios-Navarro et al. 2015), que será expuesta en este capítulo, donde la combinación de sensores es integrada para compensar posibles errores de interpretación de alguno de ellos frente a ruido, pero no se realiza a nivel cognitivo.

IV.1 Tipos de integración sensorial

Atendiendo a la combinación de los datos de los sensores a combinar, la integración sensorial se puede dividir en dos grandes categorías.

IV.1.1 Complementaria

En esta categoría encontramos aquellos sistemas donde la información de los sensores es combinada aportando un resultado con mayor información que el ofrecido por un sensor aislado. Además de los ejemplos planteados anteriormente, otro caso de integración complementaria sería cuando dos cámaras son utilizadas para ampliar el campo de visión, resolviendo la zona común mediante algoritmos de solape utilizando puntos clave.

IV.1.3 Compensatoria

A esta categoría pertenecen los sistemas en los que predomina la información de un sensor, utilizando la información del resto de sensores para corregir o compensar las derivas producidas por el sensor principal. Un ejemplo de este tipo de integración es el uso del filtro de Kalman para monitorizar el comportamiento de la vida salvaje utilizando un acelerómetro y giróscopo (Tapiador-Morales et al. 2015).

En este trabajo no se persigue el uso de cualquier tipo de sensores con el fin de combinar sus salidas para realizar una tarea específica, si no que se trata de combinar información de sistemas audio-visuales que utilizando sensores que imitan aquellos que podemos encontrar en el ser humano e implementan modelos inspirados en la biología. Hasta este momento en el presente trabajo sólo se ha hecho uso del sensor visual, y a continuación se plantea el uso de un sensor auditivo con el fin de plantear soluciones que integren la información de ambos sensores.

El sensor de audio a utilizar en este trabajo, NAS, fue introducido en el primer capítulo. Para el procesamiento de la información procedente de este sensor, se hará uso de un tipo de neuronas de clasificación propuestas en (Cerezuela-Escudero et al. 2015) con el fin de utilizar esa clasificación para compensar y limitar la información procedente del sensor DVS, proponiendo una integración sensorial de tipo compensatoria.

IV.2 Clasificación auditiva

El NAS presentado en el capítulo de introducción se corresponde con una implementación estéreo de 64 canales, donde las bandas de mayor frecuencia tienen una mayor tasa de spikes que los canales de baja frecuencia. Para algunos tipos de procesamiento de audio en el que no existe un entrenamiento previo, nos podemos plantear que una salida normalizada para todas sus bandas puede llegar a dar mejores resultados. Esta normalización está detallada en (Cerezuela Escudero 2015), pero a continuación la explicaremos de forma cualitativa para entender cómo es la entrada que alimenta a las neuronas de clasificación auditiva.

IV.2.1 Normalización del NAS

Como se explicó en el capítulo de introducción, en la arquitectura del NAS hay un banco de filtros, en el dominio de los spikes, el cual descompone la señal de entrada en cada una de las bandas de frecuencia en las que el NAS ha sido sintonizado. Cada uno de esos filtros ha sido implementado como la diferencia de dos filtros paso de baja pulsante, consiguiendo así un filtro paso de banda, cuya salida es atenuada por un divisor de spikes. En la Figura 49 se puede ver un esquema de un filtro paso de banda, donde SLPF son dos filtros de paso de baja conectados en cascada y configurados a diferentes frecuencias. La salida de la resta de ambos filtros se corresponde con la salida del filtro paso de banda. Esta resta se realizada en el dominio pulsante con el bloque SH&F.

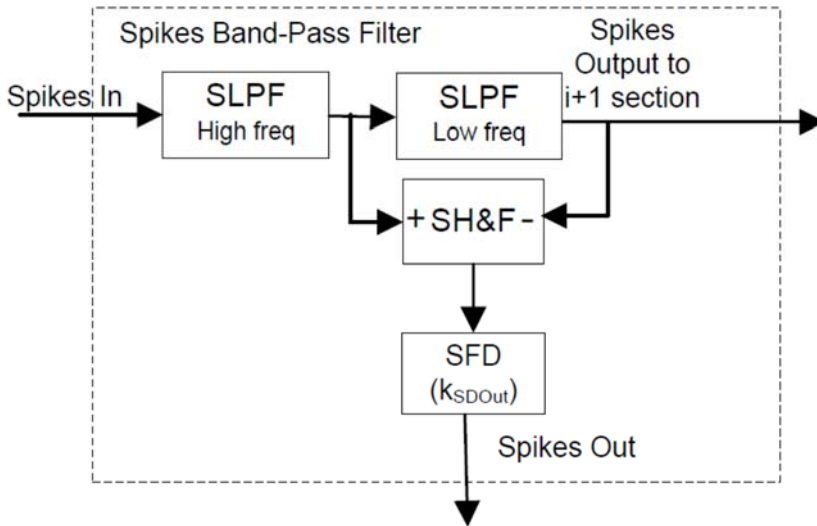


Figura 49. Diagrama de bloques de un filtro paso de banda del NAS (Cerezuela Escudero 2015)

La función de transferencia del atenuador de spikes *SFD* (Jimenez-Fernandez et al. 2010) se corresponde con la Ecuación 9, donde N representa el número de bits que tiene el divisor y *spikesDiv* es el parámetro con el cual se establece el valor del divisor. Por lo tanto, hay que variar el valor *spikesDiv* de cada uno de los *SFD* de manera individual para conseguir una salida normalizada para todas las bandas del NAS. El proceso de sintonización de los divisores de spikes está explicado detalladamente en (Cerezuela Escudero 2015), consiguiendo como resultado los distintos valores *spikesDiv* para cada banda (ver Figura 50).

$$F_{SFD} = \frac{F_{outSpikes}}{F_{inputSpikes}} = \frac{spikesDiv}{2^{N-1}}$$

Ecuación 9. Divisor de spikes

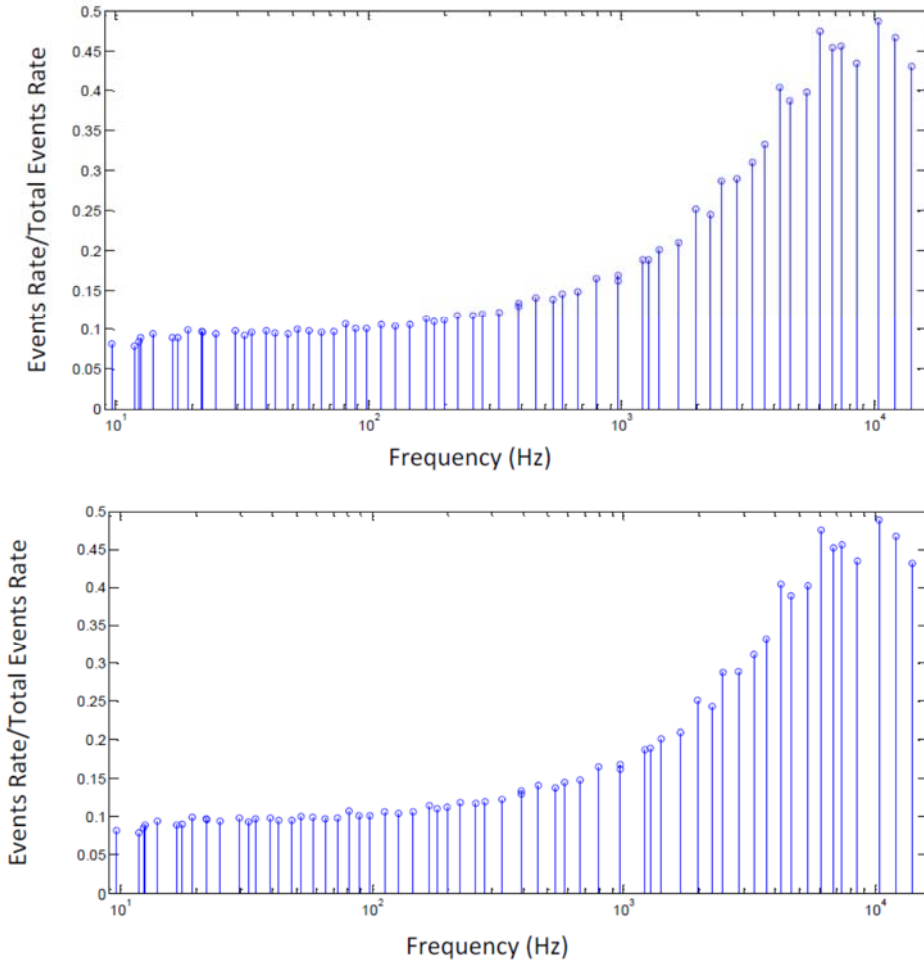


Figura 50. Valores para normalizar los filtros de spikes de cada una las bandas, arriba para la cóclea izquierda, abajo para la cóclea derecha (Cerezuela Escudero 2015)

Como comparativa entre ambas versiones del NAS, se muestra la Figura 51 la diferencia entre ambas salidas de los canales.

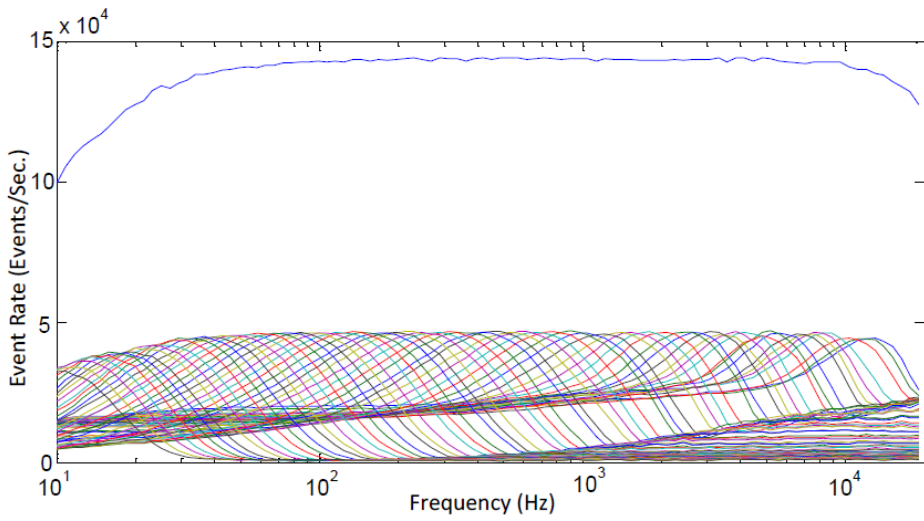
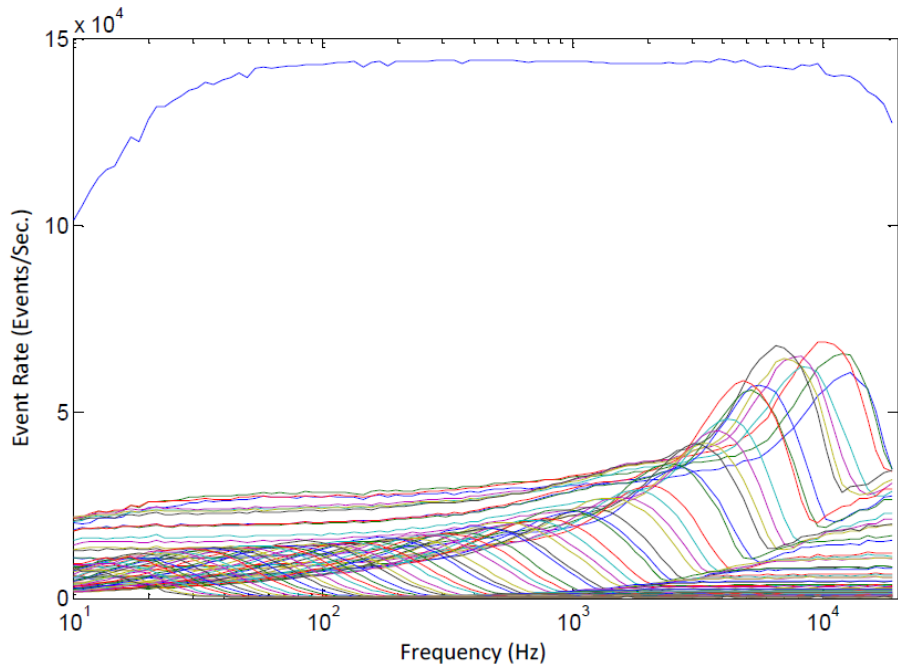


Figura 51. Diagramas de bode del banco de filtros. Arriba NAS no normalizado, abajo NAS normalizado

IV.2.2 Neuronas de clasificación auditiva

Las neuronas propuestas en (Cerezuola-Escudero et al. 2015), utilizadas en este capítulo, se basan en la operación de convolución mono-dimensional. En el capítulo anterior se habló de las CNN aplicadas a la visión, en las cuales la operación de convolución utilizada es bidimensional, ya que se aplican a datos representados por frames de dos dimensiones.

En este caso la arquitectura de la neurona consta de un número de entradas igual al número de canales de salida del NAS, el valor actual de la neurona y el umbral de disparo. En la Figura 52 se puede ver un esquema de una neurona. El funcionamiento de esta neurona es similar a la LIF⁴⁶, en la que el valor actual de la neurona, conocido como potencial de membrana, es modificado en función de las entradas, produciendo una salida (spike) si el valor de la membrana supera un cierto umbral establecido. La operación de convolución mono-dimensional de una neurona queda definida por la Ecuación 10, donde x es el instante actual, W es el núcleo de convolución, S es la salida de cada canal del NAS, M el número de entradas e Y es la salida de la convolución, cuyo valor depende del valor del instante anterior. La Ecuación 11 representa la salida de la neurona, que solo se produce si el valor de Y es mayor o igual que al del umbral θ establecido. Este sistema utiliza la información pulsante sobre la intensidad de cada una de las componentes frecuenciales del sonido de entrada, así pues es un sistema basado en frecuencias características y la amplitud del sonido.

$$Y(x + 1) = Y(x) + \sum_{m=0}^M (W(m) * S(x))$$

Ecuación 10. Operación mono-dimensional de convolución de una neurona

$$Out = Y(x) \geq \theta \rightarrow Y(x) = 0$$

Ecuación 11. Condición de generación de la salida de una neurona

⁴⁶ Leaky Integrate & Fire

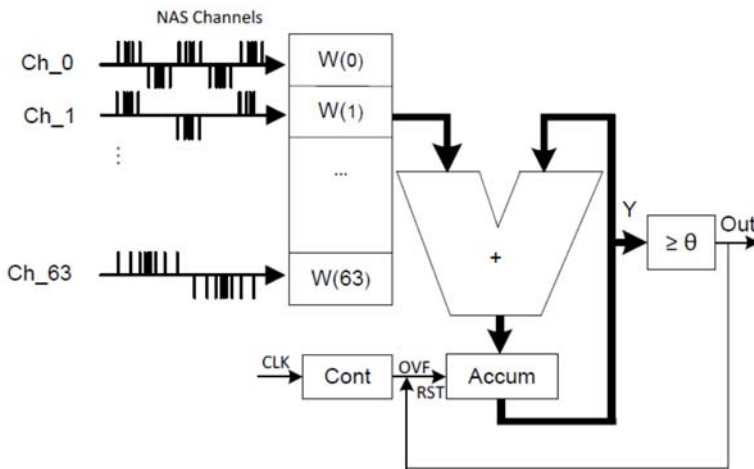


Figura 52. Arquitectura de una neurona (Cerezuela Escudero 2015)

Los valores de los núcleos de convolución de cada neurona se obtienen a partir de la tasa de eventos de cada canal del NAS ante sonidos ejemplares de cada clase que se desea reconocer. Con el objetivo de que los núcleos de convolución sean valores independientes del volumen de la señal, son normalizados en el rango $[0, 1]$.

Para configurar un sistema de reconocimiento utilizando estas neuronas se establece una capa con un número de neuronas igual al número de clases a reconocer. Luego, para establecer los valores de cada uno de los núcleos de convolución de las distintas neuronas, se calcula la tasa de eventos de cada canal de salida del NAS, para cada uno de los sonidos ejemplares normalizando el resultado. Por último, a la salida de cada neurona se conecta una capa WTA⁴⁷ para dar como clase ganadora aquella que más actividad tenga a su salida. En la Figura 53 se puede ver un esquema global de la configuración de una red de este tipo.

⁴⁷ Winner Take All

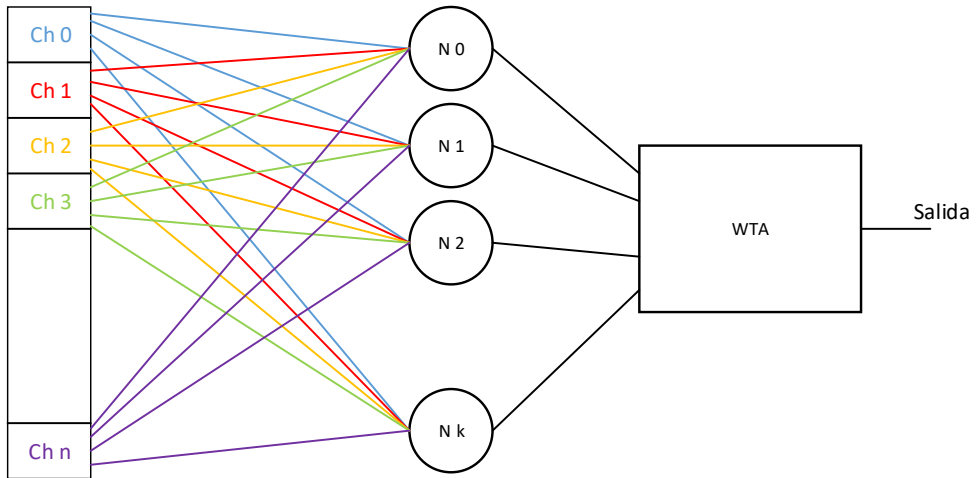


Figura 53. Estructura de una red de neuronas de convolución mono-dimensional

IV.3 Aplicación práctica y resultados experimentales

Como aplicación práctica se ha optado por diseñar un sistema capaz de detectar las revoluciones de un motor DC, que es aplicable en sistemas de control de calidad. El escenario se compone de un motor, al que se le ha acoplado un disco circular en su eje con una pequeña figura pintada en él. Un sensor DVS128 (Lichtsteiner et al. 2008) enfoca directamente al disco que rotará haciendo girar la figura que hay impresa en él, estimulando los píxeles de la retina que captarán su movimiento. En cuanto al audio, se ha colocado un micrófono tras el motor que capta el sonido producido por éste cuando está en funcionamiento. El micrófono está conectado a una mesa de mezclas (XENYX QX1002USB) para eliminar el ruido de fondo y amplificar la señal de audio, que se utiliza como interfaz de entrada al NAS, el cual está desplegado en una placa de evaluación de Xilinx con una Virtex 5 (ML507 Virtex 5 evaluation board). En la Figura 54 se puede ver los elementos que componen la distribución del experimento.

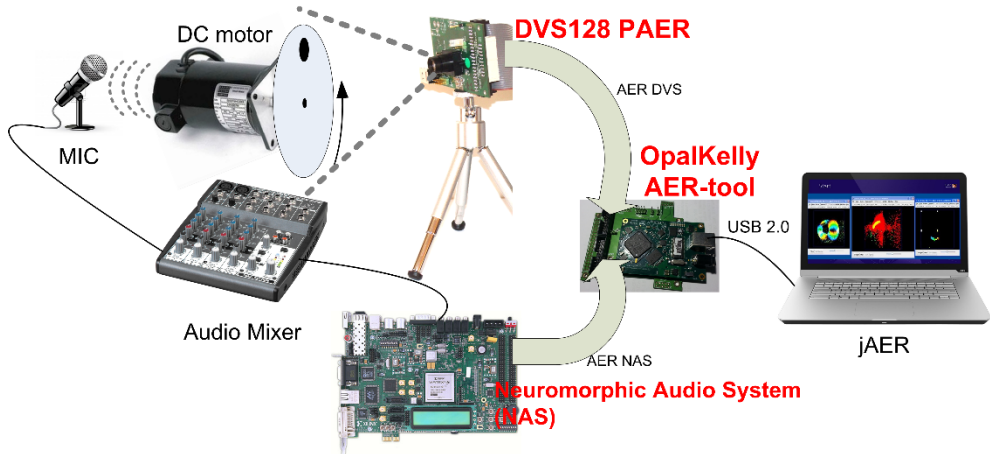


Figura 54. Elementos intervinientes en la demostración del sistema de integración sensorial

La salida de ambos sensores está conectada a la plataforma OKAERTool (Rios-Navarro et al. 2016), donde se mezclan y son enviadas al PC a través del puerto USB. En el apartado anexos, están definidas todas las funcionalidades y la arquitectura de la plataforma OKAERTool, que ha sido diseñada para trabajar con sistemas basados en spikes y cubre las funcionalidades de *merger*, *monitor*, *sequencer*, *load* y combinaciones entre ellas, y representa una aportación en este trabajo de investigación.

La idea global de esta prueba consiste en que con la información proveniente de la DVS128 se estima una aproximación de la velocidad del motor, con la información del NAS se establece un máximo y un mínimo que limitan las estimaciones calculadas con la información de la DVS128. Este procesamiento es realizado mediante un filtro implementado en la aplicación jAER (jAER 2007), que está orientada al procesamiento por spikes. En la Figura 55 se puede ver las estimaciones calculadas con la DVS128 frente a la información obtenida por un encoder óptico que se utiliza como referencia para calcular la fiabilidad del sistema.

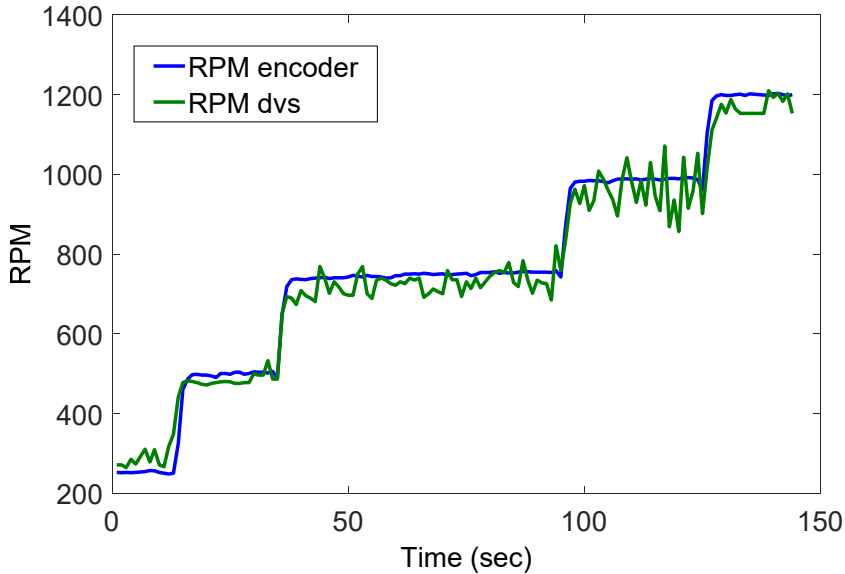


Figura 55. Estimaciones de las rpm del motor haciendo uso de la información de la DVS128

El cálculo de las estimaciones de la velocidad del motor haciendo uso de la información de la DVS128 se realiza calculando el tiempo que tarda la figura impresa en el disco en pasar dos veces consecutivas por el mismo punto, que es el tiempo en dar una vuelta. Dicho tiempo es la diferencia de *timestamp* entre los dos grupos de spikes (de la misma polaridad) correspondientes a diferentes vueltas. Para el cálculo de los máximos y mínimos de los rangos que limitan las estimaciones anteriores, se ha hecho uso de las neuronas comentadas en el apartado anterior. En este caso se requiere clasificar 9 rangos diferentes de velocidades, por lo que se necesitan 9 neuronas. Para la configuración de cada núcleo de convolución de las neuronas se establece la velocidad del motor en aquellas que quieren ser clasificadas. Por cada una de ellas se detectan cuáles son los canales de salidas del NAS que más se excitan, identificando de esta manera cuáles son las entradas de la neurona que más deben excitarse para identificar a un rango de velocidad concreto. Los valores de los núcleos de convolución son normalizados siguiendo la gráfica de la Figura 56, con el fin de obtener un sistema de reconocimiento independiente del volumen.

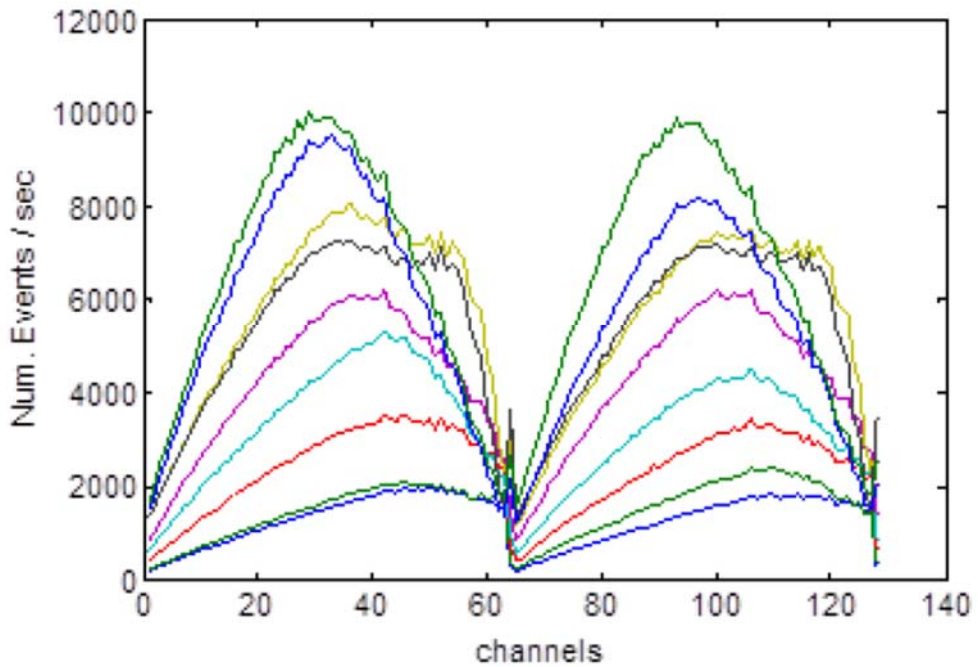


Figura 56. Salida de los diferentes canales de salida del NAS para diferentes velocidades

La prueba realizada consiste en aumentar la velocidad del motor con valores diferentes, los cuales son estimados por el sistema propuesto. La Figura 57 muestra como la información obtenida de la DVS128 está limitada por los rangos obtenidos de la información del NAS. La línea verde es la salida final del sistema, que coincide con la estimación obtenida mediante la DVS128, pero teniendo en cuenta los rangos que han sido obtenidos mediante la clasificación del sonido. Las líneas roja y marrón identifican el máximo y el mínimo respectivamente de cada rango. Como se puede ver, la salida del sistema está muy próxima a la línea azul, que es la estimación de la velocidad que ha sido obtenida mediante un encoder óptico, considerada como la velocidad real del motor.

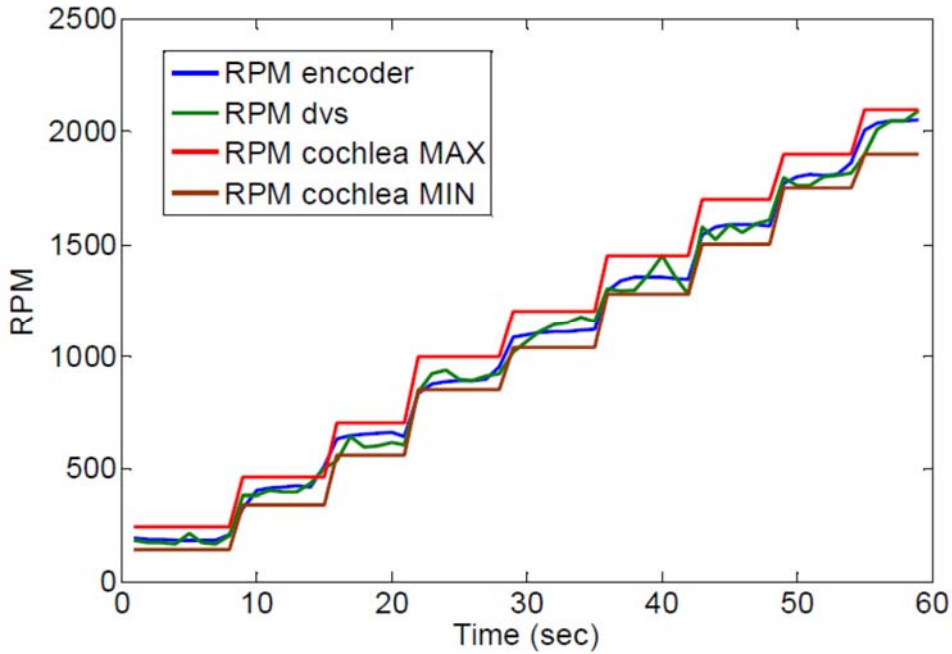


Figura 57. Salida del sistema para las diferentes velocidades

Para calcular el error del sistema se ha hecho uso de la Ecuación 12, donde se decide si la estimación del sistema es considerada como fallo o éxito, siendo t el instante de tiempo, V_r la velocidad calculada por el encoder, V_e la velocidad calculada por el sistema propuesto, y ΔV_r la tolerancia soportada. Considerando que los rangos obtenidos por la parte del audio tienen una diferencia del 40% entre el centro y sus límites, la tolerancia se ha establecido al 10%. Por lo tanto, la precisión del sistema ha sido calculada a 94.33%.

$$E(t) = Vr(t) - \Delta Vr \leq Ve(t) \leq Vr(t) + \Delta Vr$$

Ecuación 12. Cálculo del error cometido por el sistema al realizar una estimación

Atendiendo a los resultados obtenidos para el tipo de integración sensorial aportada, vemos que aunque es válido para esta tarea concreta, es difícil de aplicar para otros tipos de tareas, teniendo que modificar el comportamiento de cómo combinar la información de ambos sensores. Es por ello que cabe pensar en el uso del *Deep Learning* como un mecanismo para fusionar información proveniente de más de un sensor, describiendo modelos de redes neuronales que pueden ser entrenados para diferentes tareas. Un ejemplo de esta aplicación la podemos ver en (Gutierrez-Galan et al. 2017) en el marco del proyecto de Excelencia de la

Junta de Andalucía MINERVA, donde una NN diseñada y entrenada para clasificar el comportamiento de un animal en libertad, ha sido empotrada en un microcontrolador, fusionando la información de un giróscopo y un acelerómetro de 3 grados de libertad cada uno. En este caso en particular, se ha aplicado para un caballo, clasificando 3 comportamientos diferentes, pero el algoritmo de procesado de la NN es capaz de procesar otro modelo de NN que haya sido diseñado y entrenado para otra especie diferente.

Uno de los objetivos de este trabajo es el estudio de combinar sensores neuromórficos de audio/video con el fin de conseguir modelos que sean capaces de adaptarse, mediante una fase de entrenamiento, a diferentes tareas o retos. Es por ello que en el siguiente capítulo se expondrá un modelo de CNN, que cumple con la capacidad de ser entrenado para diferentes tareas que fusione la información de ambos sensores con el fin de hacer un sistema más robusto que si solo se empleara uno de ellos.

V. *Fusión sensorial*

Un problema fundamental en la neurociencia cognitiva es cómo percibimos y representamos el mundo exterior empezando por la información producida por nuestros sentidos. La investigación tradicional sobre este tema generalmente adoptaba un enfoque de “sentido a sentido”, la atención estaba focalizada en una simple modalidad sensorial donde se estudiaba cómo la información es codificada en cada canal sensorial y posteriormente procesada y transmitida. Durante algunas décadas, se ha asumido que la información proveniente de varios sentidos individuales es inicialmente codificada y procesada en áreas separadas del cerebro (tales como el córtex visual en los lóbulos occipitales, el córtex auditivo en los lóbulos temporales y el córtex somato sensorial en los lóbulos parietales) y sólo después de su procesamiento, esta información distribuida puede fusionarse. Este punto de vista tradicional postula que una gran parte del cerebro puede ser reducida en pequeñas colecciones de módulos “unisensoriales” que pueden ser estudiados de manera aislada.

Sin embargo, investigaciones recientes desafían esta visión tradicional (Dalton et al. 2000; Bermant & Welch 1976; Bolognini et al. 2005; Calvert et al. 2004; Cuppini et al. 2010), demostrando claramente que nuestros sentidos están diseñados para trabajar en conjunto, y no como módulos separados. De hecho, casi cualquier experiencia en nuestra vida diaria consiste en un flujo multisensorial de datos, que proporcionan información concurrente o complementaria sobre el mismo evento u objeto externo. Esta información multisensorial debe ser considerada simultáneamente por el cerebro y globalmente procesada para alcanzar una percepción única y coherente. Además existen pruebas evidentes de que la información representada por cada sistema sensorial principal (visual, auditivo, somatosensorial, olfato y gusto) es altamente susceptible a la influencia de otros sentidos, como por ejemplo el gusto y el olfato para degustar los alimentos. Esto significa que la teoría de la integración sensorial debe de ir más allá que la noción de una suma o cualquier otra operación básica de las contribuciones individuales de cada sentido, debiendo considerar sistemas más complejos que fusionen la información de los sentidos.

La sofisticada integración de múltiples señales sensoriales proporciona a los animales y a los humanos una gran flexibilidad en cuanto a su comportamiento y respuestas. De hecho, el

objetivo fundamental del cerebro con respecto al mundo exterior es alcanzar un reconocimiento de objetos y eventos lo más fiable posible (especialmente aquellos esenciales para la supervivencia) así como dirigir su comportamiento y reacciones físicas de la manera correcta. La decisión sobre los objetos y eventos del entorno requiere que la información de todos los sentidos sea explotada simultáneamente para maximizar la probabilidad de una correcta detección y minimizar la probabilidad de errores. Para alcanzar este objetivo, las señales sensoriales originadas en una ventana espacio-temporal coincidente, de ahí que probablemente deriven del mismo evento, deben ser tratadas juntas, mientras que las señales sensoriales que no exhiben coherencia espacial y/o temporal deben ser separadas.

El procesamiento multisensorial se conoce que ocurre en muchas regiones cortical y subcortical del cerebro, e involucra muchos de los aspectos de la vida cotidiana y su comportamiento: el rápido movimiento de los ojos y cabeza dirigido a apuntar señales audiovisuales, la unión del gusto y el olfato durante la alimentación y la fusión audiovisual para el reconocimiento de objetos, entre otros.

Desafortunadamente, a pesar del gran rol que la fusión sensorial juega en nuestras vidas, los mecanismos neuronales que favorecen esta propiedad no se conocen aun suficientemente. Cabe señalar que los modelos neuronales para la fusión sensorial no deben sumar (o realizar otra operación simple) la información proveniente de diferentes sentidos, pero deben implementar un procesamiento no lineal más complejo para enfatizar los estímulos con características espacio-temporales coherentes, rechazando aquellos estímulos incongruentes.

En este capítulo nos enfocaremos en una fusión sensorial audiovisual para la clasificación de objetos, particularmente orientado a dígitos manuscritos que también son leídos. Para ello se hará uso de un modelo de CNN, diseñado expresamente para este fin, debido a su enfoque biológico sobre el córtex visual que en este caso es extendido para extraer características de información auditiva. La CNN aportada en este trabajo trabaja con frames como datos de entrada, mientras que los sensores utilizados generan spikes en su salida, por lo que se requiere una acumulación de spikes para la preparación de un frame. En el siguiente apartado se exponen diferentes métodos de acumulación de spikes para la generación de un frame.

V.1 Generación de los datos de entrada

Como se comentó en el capítulo de introducción, para el presente trabajo se ha hecho uso de un par de sensores neuromórficos, uno de visión y otro de audio. Concretamente para la implementación del modelo de fusión sensorial aportado en este capítulo, se ha hecho uso de

la DVS128 (Serrano-Gotarredona & Linares-Barranco 2013) cuya resolución es de 128x128 píxeles (o neuronas) y el NAS (Jimenez-Fernandez et al. 2016) con 64 canales (o neuronas) mono. El modelo de fusión sensorial consiste en una CNN que toma como entrada las informaciones de ambos sensores, las cuales son fusionadas a lo largo de las capas de procesamiento de la CNN. Como se explicó en el apartado *III.1*, este tipo de algoritmo de *Deep Learning* toma como entrada en cada una de sus capas, tipos de datos multidimensionales, concretamente de 3 dimensiones, donde las dos primeras se corresponden al tamaño del frame (o *feature-map*) y la tercera al número de frames (o *feature-maps*).

Como también vimos en el apartado de introducción, estos sensores no ofrecen este tipo de datos en su salida, por lo que la primera tarea consiste en crear mecanismos de conversión entre la salida proporcionada por los sensores (spikes) y la entrada demandada por la CNN. Antes de comenzar a explicar los métodos utilizados para la generación de la BD de visión que alimentará al modelo CNN de fusión sensorial, se va a proceder a explicar brevemente el tipo de dato que generan estos sensores a su salida.

El sensor DVS128 genera una palabra AER en su salida que identifica aquel píxel que ha sido alterado para generar un spike. En esa palabra AER, está codificada la coordenada X e Y del píxel que generó el spike junto con un bit de polaridad para indicar si el spike es positivo y negativo. La polaridad positiva del spike identifica un cambio a más luminosidad partiendo del estado anterior del píxel, mientras que la negativa identifica un cambio a menos luminosidad partiendo del estado anterior del píxel. En la Figura 58 se representa una palabra AER a la salida del sensor DVS128.

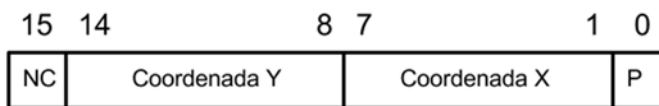


Figura 58. Salida AER del DVS128

Por otro lado, el NAS de 64 canales mono utilizado, genera una palabra AER que identifica aquel canal que ha sido alterado para generar un spike. En esa palabra AER están codificados el canal y la polaridad del spike que ha sido generado, siendo los spikes positivos aquellos que codifican las componentes frecuenciales mayores que cero de la señal resultante a la salida del canal, mientras que los spikes negativos codifican las componentes frecuenciales menores que cero de dicha señal. La transición de positivos a negativos del flujo de spikes a la salida de cada canal representan los cortes por cero de la señal codificada. En la Figura 59 se representa una palabra AER a la salida del sensor NAS.

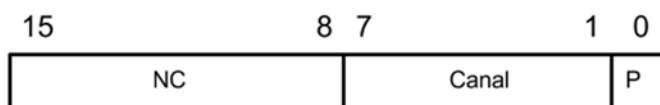


Figura 59. Salida AER del NAS

Como se puede ver, ambas salidas tienen polaridad, y haciendo uso de esta propiedad se plantea 4 métodos de generación de frames a partir de los spikes generados por los sensores:

- I. *Full-integration*: Se tienen en cuenta ambos tipos de spikes, positivos y negativos. Para la elaboración del frame, se inicializan todos los píxeles del frame de entrada a la CNN al valor 128, que se corresponde con el valor medio del píxel (se utilizan píxeles de 8 bits de profundidad). Por cada spike generado, se actualizará el valor del píxel indicado por las coordenadas X e Y del spike, aumentando en una unidad si el spike es positivo y disminuyendo en una unidad si es negativo.
- II. *Unsigned-integration*: En este método también se tienen en cuenta ambos tipos de spikes, pero la elaboración del frame difiere con respecto a la anterior. En este método la inicialización del frame se realiza estableciendo el valor de todos los píxeles a cero. Para cada spike generado, ya sea positivo o negativo, se incrementa en una unidad el valor del píxel indicado por las coordenadas X e Y del spike.
- III. *Half-rectified positive*: En este método solo se tienen en cuenta los spikes positivos, inicializando el valor de todos los píxeles del frame a cero, y aumentando en una unidad aquel valor del píxel indicado por las coordenadas X e Y de cada spikes positivo.
- IV. *Half-rectified negative*: Este método funciona de forma opuesta al anterior. En este caso solo se tienen en cuenta los spikes negativos, inicializando el valor de todos los píxeles del frame a cero, y aumentando en una unidad aquel valor del píxel indicado por las coordenadas X e Y de cada spikes negativo.

Por último, el valor de los píxeles son normalizados entre [0-255], siendo 0 el valor que codifica el negro y 255 el valor que codifica el blanco. Así pues, para cada uno de los dos sensores se generará una base de datos (BD) utilizando cada uno de los métodos anteriormente descritos, con muestras para la etapa de entrenamiento y muestras para la etapa de test, que serán utilizadas para entrenar y probar el modelo de fusión sensorial para el reconocimiento de dígitos del 0 al 9.

V.1.1 Generación de datos del DVS128

Para el sensor de visión se parte de la colección de imágenes MNIST (LeCun, Cortes, et al. 1998), que consta de 60000 imágenes para la fase de entrenamiento y 10000 para la fase de test. Esta colección de imágenes es utilizada para la generación del conjunto de datos MNIST pero basados en spikes. Para la captura de este conjunto de datos, cada imagen en escala de grises de la colección MNIST original es mostrada 5 veces de manera intermitente en un monitor LCD frente a una retina DVS128. Cada flash tarda 16ms (tiempo de refresco del monitor) y hay una espera de 200ms entre cada flash consecutivo. Las grabaciones de cada nuevo dato referente a un dígito, pero codificado en spikes, se ha llevado a cabo utilizando la aplicación jAER (jAER 2007), y puede ser reproducido utilizando la misma aplicación. Esta colección de datos está disponible en (Yousefzadeh et al. 2012), junto con una serie de estructuras de datos en MATLAB donde se encuentran las etiquetas correspondientes de cada grabación, identificando el dígito al que corresponde cada una de ellas.

Para la generación de las BBDD utilizando los diferentes métodos de integración, se ha desarrollado un script que recorre todos los ficheros (*.aedat) que contienen los spikes que codifican los diferentes dígitos y convierte uno a uno a frames teniendo en cuenta el método de integración utilizado. Este script básicamente realiza las siguientes acciones:

- Leer el fichero .aedat correspondiente, creando una matriz donde cada fila se corresponde con un spike y las columnas identifican cada uno de los datos relativos a dicho spike:
(Timestamp (us) | Coordenada Y | Coordenada X | Polaridad)
- Una vez obtenida la matriz anterior que contiene todos los spikes del fichero .aedat, se llama a la función *DVSaer2frame(spikes, max_x_axe_value, method_integration)*. Esta es la función que va creando el frame según el método de integración aplicado. Además realiza la normalización una vez que se han integrado todos los spikes. Para este caso en particular, que se requiere un frame de 64x64 píxeles, al frame resultante de la integración (128x128 píxeles) se le aplica una función de *max_pooling* aplicando un filtro de 2x2 píxeles consiguiendo el frame necesario de 64x64 píxeles.
- Por último se utiliza la función nativa de MATLAB *imwrite()* para generar la imagen del frame en formato 'png'. Junto con la creación de esta imagen se va generando el fichero de etiquetas para identificar la clase (en este caso el dígito) a la que corresponde el frame generado. Este fichero de etiquetas será utilizado posteriormente para entrenar el modelo CNN.

Este procedimiento es aplicado a todo el conjunto de datos MNIST en spikes, creando una BBDD por cada método de integración donde cada una de ella cuenta con 60000 frames para la etapa de entrenamiento y 10000 frames para la etapa de prueba. En la Figura 60 se puede ver el resultado de las diferentes integraciones en frames para el dígito 7 codificado en spikes.

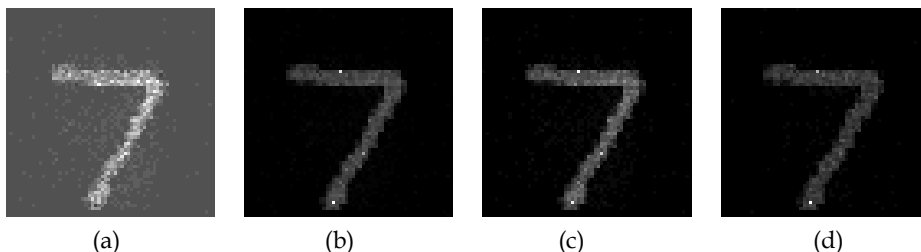


Figura 60. Muestras del resultado de los diferentes métodos de integración para la visión. (a) full-integration, (b) unsigned-integration, (c) half-rectified positive, (d) half-rectified negative

V.1.2 Generación de datos del NAS

Para el caso del audio, no contamos con una BD donde tengamos los dígitos del 0 al 9 hablados por diferentes sujetos. Es por ello que la primera tarea a realizar es la de conseguir muestras separadas en las que se encuentre el audio de cada uno de los dígitos hablados por varios sujetos. Para ello se ha desarrollado un script en PYTHON que a tipo de guía le va indicando al usuario qué dígito tiene que decir frente a un micrófono, recogiendo la muestra hablada. Esta muestra es guardada identificando a qué dígito corresponde para su futuro etiquetado.

Para el presente trabajo se han recogido muestras para los dígitos del 0 al 9 hablados en inglés, de 36 sujetos diferentes (hombres y mujeres), totalmente anónimos de edades comprendidas entre 19 y 59 años. Para generar una BD suficiente y que se asemeje a la generada para el sensor de visión, es necesario ampliar las muestras sintéticamente. Para ello se realiza un proceso que cuenta con los siguientes pasos:

1. **Variación del volumen**

Se han aplicado 10 modificaciones de volumen diferentes, 5 de ellas aumentando el volumen en un 0.5%, y otras 5 disminuyendo el volumen en un 0.5%. Aplicando estas 10 modificaciones conseguimos tener 11 muestras de cada uno de los dígitos hablador, la original más las 10 modificaciones.

2. **Aplicación de distintos niveles de ruido**

Con el fin de ampliar la BD y hacer el sistema más robusto, se aplican 9 niveles de ruido, utilizando la función de MATLAB *awgn*, que aplica ruido blanco Gaussiano comúnmente presente en sistemas de audio. El valor inicial SNR⁴⁸ a aplicar es inicializado a 40db, reduciendo en 2.22db los siguientes SNR consecutivamente aplicados. Con esta modificación conseguiremos 10 muestras más por cada una de las ya existentes.

3. Desplazar la actividad de la muestra

Una vez que la muestra es adquirida, es posible que ésta haya sido recogida principalmente en tres ventanas temporales diferentes, al comienzo, en mitad o al final de la muestra. Es por ello que por cada una de las muestras se detectará dónde está la principal actividad, y se desplazará a aquellas ventanas temporales restantes. Por ejemplo, si una muestra tiene su principal actividad en la primera ventana temporal (al principio de la muestra), se generarán dos nuevas muestras con la actividad desplazada lateralmente a las ventanas temporales 2 y 3 (centro y final respectivamente). Con esto conseguimos aumentar la discriminación de dónde se encuentra la actividad de la muestra, además de conseguir 3 muestras por cada una existente.

Tras ver los pasos seguidos para aumentar el número de muestras de la BD de audio de forma sintética, podemos calcular el número de muestras que se generan por cada uno de los sujetos de las grabaciones. La Ecuación 13 indica el número de muestra por sujeto, y particularmente en este caso contamos con 36 sujetos, con lo que obtenemos una BD con un total de 118800 muestras.

$$10\text{muestras} \cdot 11n.\text{volumen} \cdot 10\text{SNR} \cdot 3\text{desplazamientos} = 3300 \text{muestras/sujeto}$$

Ecuación 13. Número de muestras por sujeto para la BBDD de dígitos hablados

Solo el primer paso de los anteriormente descritos es aplicado cuando se tiene la muestra en formato *.wav*, aún sin convertir a spikes. El resto de modificaciones son aplicadas cuando se está realizando la integración de los spikes al frame. Para obtener la colección de datos en spikes, cada uno de los ficheros *.wav* grabados con el script descrito en PYTHON comentado anteriormente, son reproducidos utilizando un script descrito en MATLAB hacia el NAS y captura a su vez todos los eventos generados por éste, guardando esa información en un fichero *.aedit*. Una vez han sido convertidas todas las muestras en spikes, se lanza el script

⁴⁸ Signal Noise Ratio

para convertir cada una de las muestras .aedat a frames utilizando los diferentes métodos de integración. Este script básicamente realiza las siguientes acciones:

- Leer el fichero .aedat correspondiente, creando una matriz donde cada fila se corresponde con un spike y las columnas identifican cada uno de los datos relativos a dicho spike:
(Timestamp (us) | Canal | Polaridad)
- Una vez obtenida la matriz anterior que contiene todos los spikes del fichero .aedat, se llama a la función *NASaer2frame(spikes, max_x_axe_value, method_integration)*. Esta es la función que va creando el frame según el método de integración aplicado. Además realiza la normalización una vez que se han integrado todos los spikes. Como se indicó anteriormente, para este caso en particular se requiere un frame de 64x64 píxeles, así pues el eje X, que en este caso se corresponde con el tiempo, ha de ser comprimido de tal forma que todos los eventos generados en la muestra sean codificados en 64bits de ancho. Para ello se calcula qué ventanas temporales (indicadas por el timestamp del spike) están codificadas para cada valor de X. Por otro lado, el eje Y representa cada uno de los canales del NAS, que en este caso son 64.
- Se aplica el paso 2 y 3 del procedimiento de generación sintético de muestras.
- Por último se utiliza la función nativa de MATLAB *imwrite()* para generar la imagen del frame en formato 'png'. Junto con la creación de esta imagen se va generando el fichero de etiquetas para identificar la clase (en este caso el dígito) a la que corresponde el frame generado. Este fichero de etiquetas será utilizado posteriormente para entrenar el modelo CNN.

Este procedimiento se aplica a todo el conjunto muestras en spikes, creando una BD por cada método de integración donde cada una de ella cuenta con 118800 frames, de los cuales serán tomados 60000 frames para la etapa de entrenamiento y 10000 frames para la etapa de prueba. Esto es necesario para poder realizar un entrenamiento simultáneo del modelo, tomando el mismo número de muestras de entrenamiento y prueba tanto para el audio como para la visión, no afectando para el entrenamiento el descarte de las muestras de audio sobrantes. En la Figura 61 se puede ver el resultado de las diferentes integraciones en frames para el dígito 7 codificado en spikes.

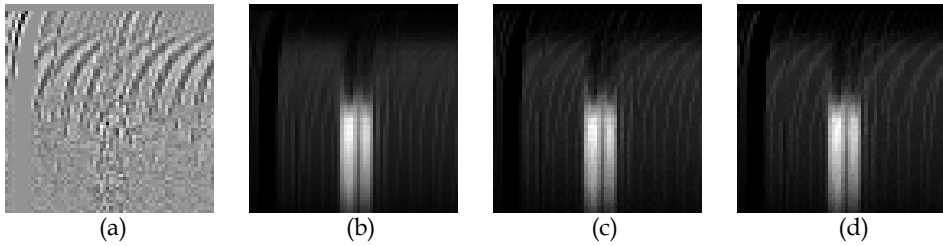


Figura 61. Muestras del resultado de los diferentes métodos de integración del audio. (a) *full-integration*, (b) *unsigned-integration*, (c) *half-rectified positive*, (d) *half-rectified negative*

V.2 Modelo CNN para la fusión sensorial audiovisual

Como se ha comentado a lo largo de este trabajo, las CNN son algoritmos de *Deep Learning* capaces de extraer características de datos representados en dos dimensiones, tipo frames, con el fin de clasificar y detectar patrones. Es por ello que han sido ampliamente utilizados en el campo de la visión, por lo que en este trabajo vamos a probar su funcionalidad en el campo de la fusión sensorial codificando la información de dos sensores neuromórficos, uno de audio y otro de visión (como se ha explicado en el apartado anterior), con el fin de hacer un sistema más robusto y con una alta tasa de acierto.

El modelo de CNN que se ha diseñado para la fusión de la información de ambos sensores cuenta con dos entradas de datos diferenciadas, una para el sensor de visión y otra para el sensor de audio. Cada entrada recorrerá una serie de capas de convolución con el fin de extraer la información necesaria (representada en *features maps*) que la caracteriza. A continuación, ambas ramas se fusionan en una sola capa con el fin de utilizar los *features maps* de ambas para realizar la clasificación necesaria. En la Figura 62 se puede ver una representación del modelo de CNN.

Como se puede ver en la siguiente figura, cada entrada de datos atraviesa una serie de capas de convolución y *pooling* para la extracción de características. Tras estas etapas, ambos caminos son fusionados en una capa concatenando los dos conjuntos resultantes en uno solo, para a continuación alimentar una capa *fully connected* con 500 unidades que a su salida aplica la función de activación ReLU. Por último, el resultado es conectado a la última capa *fully connected* con 10 unidades correspondientes a las clases de salida de la red.

Cada uno de los caminos diferenciados para la visión y para el audio, cuenta con una primera capa de convolución la cual toma el dato de entrada de tamaño 64x64 píxeles y aplica 20 filtros de tamaño 5x5 píxeles obteniendo 20 *feature maps* de tamaño 60x60 píxeles. Luego,

una capa de *pooling* es aplicada, utilizando la función *max pooling*, con filtros de tamaño 2x2 píxeles consiguiendo reducir los *feature maps* a un tamaño 30x30 píxeles. A continuación, se aplica una segunda capa de convolución con 50 filtros de tamaño 5x5 píxeles resultando 50 *feature maps* de tamaño 26x26 píxeles. Al final de cada camino, se vuelve a aplicar otra capa de *pooling* como la anterior, filtros de tamaño 2x2 píxeles y función *max pooling*, reduciendo los *features maps* a un tamaño de 13x13 píxeles.

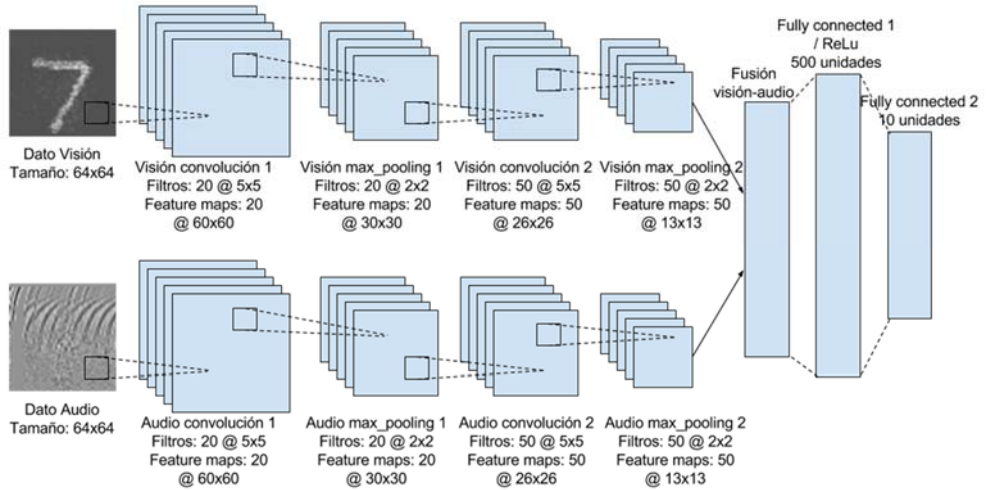


Figura 62. Modelo CNN de fusión sensorial de audio y visión

Para el diseño y entrenamiento del modelo, se ha utilizado el framework de *Deep Learning* llamado Caffe (Jia et al. 2014). En Caffe los modelos de redes son descritos en ficheros de tipo *prototxt*, añadiendo cada una de las capas identificando cuál es su entrada y su salida, conectado así cada una de las capas. Una vez descrito el modelo, hay que especificar en otro fichero tipo *prototxt* los parámetros de entrenamiento, que son los siguientes:

- net: especifica la ruta al fichero *prototxt* que contiene la descripción de la red.
- test_interval: Indica la frecuencia (número de iteraciones) con la que se debe de ejecutar la fase de prueba de la red.
- test_iter: Este parámetro indica cuántas iteraciones de prueba deben producirse por *test_interval*. Es un valor entero positivo.
- base_lr: Indica la tasa de aprendizaje base (la de partida) de la red. Es un valor real.
- momentum: Este parámetro indica cuánto del peso anterior será retenido en el nuevo cálculo. Es un valor real.

- weight_decay: Indica el factor de penalización (regularización) de pesos grandes.
- lr_policy: Este parámetro indica cómo la tasa de aprendizaje debería de cambiar en el tiempo. Existen los las siguientes políticas:
 - step: Disminuye la tasa de aprendizaje en pasos de tamaño indicado por el parámetro *gamma*.
 - multistep: Disminuye la tasa de aprendizaje en pasos indicado por el parámetro *gamma* por cada *stepvalue* especificado.
 - fixed: La tasa de aprendizaje no varía su valor.
 - exp: La tasa de aprendizaje cambia en función de: $base_lr * gamma^{iteración}$
 - inv: La tasa de aprendizaje cambia en función de: $base_lr * (1 + gamma * iter)^{-power}$
 - sigmoid: La tasa de aprendizaje cambia en función de: $base_lr (1 / (1 + exp(-gamma * (iter - stepsize))))$
- gamma: Este parámetro indica cuanto debería cambiar la tasa de aprendizaje cada vez que se alcanza el siguiente paso. Es un valor real.
- power: Es un parámetro utilizado para definir la potencia de la *lr_policy* de tipo *inv*.
- stepsize: Indica la frecuencia (número de iteraciones) en la que debemos movernos al siguiente paso de entrenamiento. Es un valor entero positivo.
- display: Indica la frecuencia (número de iteraciones) con la que se muestran los resultados en pantalla.
- max_iter: Indica cuándo la red debe parar el entrenamiento.
- snapshot: Indica la frecuencia (número de iteraciones) con la que Caffé debe hacer una copia del estado de la red.
- solver_mode: Indica si el entrenamiento es ejecutado en CPU o se utiliza GPU

Para llevar a cabo el entrenamiento del modelo aportado, se han hecho uso de tres estrategias diferentes en las que se han variado algunos parámetros de entrenamiento para mejorar la tasa de acierto resultante del modelo. En la Tabla 8 se muestran los valores de cada parámetro de entrenamiento según la estrategia.

	Estrategia 1	Estrategia 2	Estrategia 3
test_iter	100	100	100
test_interval	500	500	500
base_lr	0.001	0.001	0.001
momentum	0.9	0.9	0.9
weight_decay	0.0005	0.0005	0.005
lr_policy	inv	sigmoid	fixed
gamma	0.0001	0.0001	-
power	0.75	0.75	-
stepsize	-	100	-
max_iter	10000	10000	10000
solver_mode	GPU	GPU	GPU

Tabla 8. Parámetros de entrenamiento para las diferentes estrategias utilizadas

V.3 Resultados

Para realizar el entrenamiento y prueba del modelo de fusión sensorial audio-visual, se han convertido los datos de visión y audio a BBDD tipo *lmdb*⁴⁹, que es la utilizada por Caffe para alimentar la fase de entrenamiento y test de las redes neuronales descritas. Se han realizado entrenamientos aplicando las 3 estrategias descritas en el apartado anterior a todas las BBDD generadas a partir de los métodos de integración de frames.

Para cada una de las estrategias de entrenamiento utilizadas, se han obtenido los resultados de *accuracy* (tasa de acierto) y de *loss* (error) para cada conjunto de datos correspondientes a cada método de integración de frame. El valor de *accuracy* nos presenta la tasa de acierto en tanto por ciento del modelo de red que ha sido entrenado. Por ejemplo, si se han utilizado 100 muestras para probar el modelo y clasifica 968 correctamente, la tasa de acierto del modelo que se devuelve es del 96.8%. En cuanto al resultado representado por *loss*, no es el tanto por ciento opuesto a la *accuracy*, es la suma de los errores cometidos por cada muestra en los conjuntos de entrenamiento y validación. El *loss* es calculado sobre el entrenamiento y la validación, e indica cómo de bien está funcionando el modelo de red sobre esos dos conjuntos de datos. Naturalmente, el objetivo en la fase de aprendizaje es minimizar la función que calcula el *loss* con respecto a los parámetros del modelo, cambiando el valor de

⁴⁹ Lightning Memory-Mapped Database

los pesos de los filtros haciendo uso de métodos de optimización, como por ejemplo *backpropagation* (Benvenuto & Piazza 1992).

Estrategia 1

En esta estrategia de entrenamiento, la política para cambiar la tasa de entrenamiento “*lr*” aplicada en la “*inv*”, atiende a la siguiente ecuación:

$$lr = base_lr * (1 + gamma * iter)^{-power}$$

Ecuación 14. Función “inv” para modificar la tasa de entrenamiento

En la Figura 63 se pueden ver dos gráficas que atienden a la comparativa del resultado de la *accuracy* (izquierda) y del *loss* (derecha) para cada método de integración de frames.

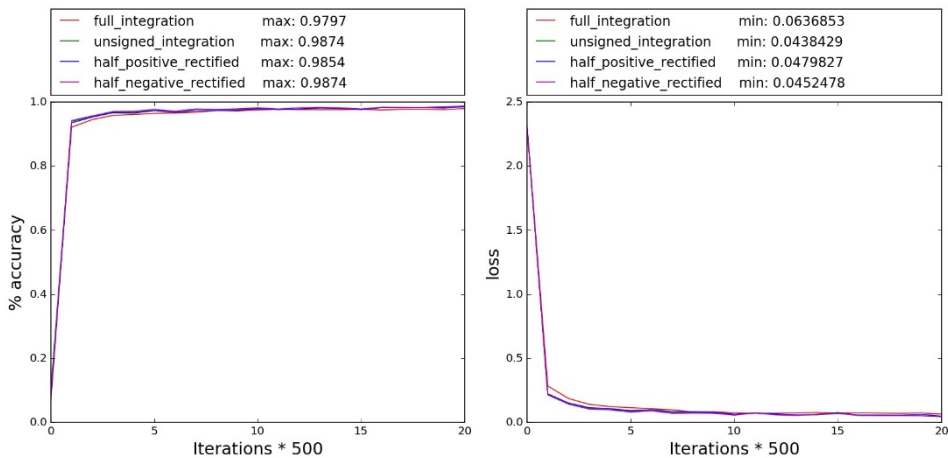


Figura 63. Accuracy y loss siguiendo la estrategia de entrenamiento número 1

Estrategia 2

Para esta estrategia, la política aplicada es la “*sigmoid*” que utiliza la siguiente ecuación para modificar la tasa de entrenamiento “*lr*”:

$$lr = base_lr * \left(\frac{1}{1 + e^{-gamma * (iter - stepsize)}} \right)$$

Ecuación 15. Función “sigmoid” para modificar la tasa de entrenamiento

Como para la estrategia anterior, en la Figura 64 se puede ver la comparativa para la *accuracy* y el *loss* de cada método de integración de frames.

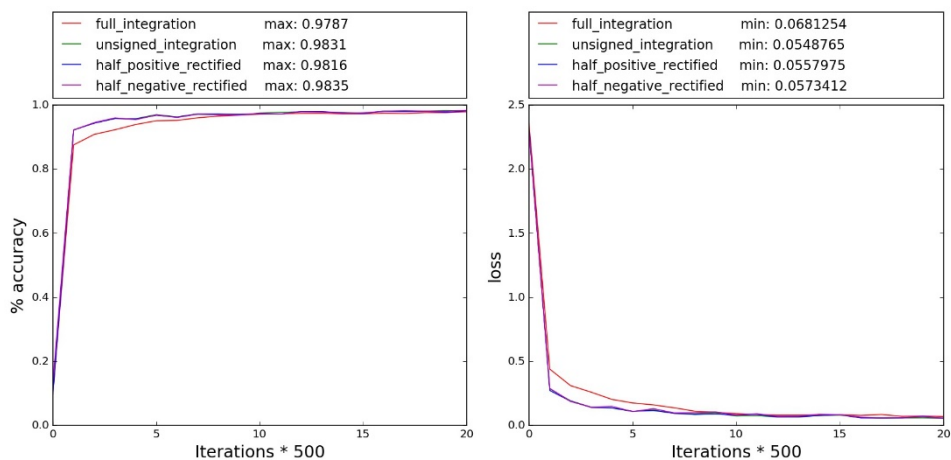


Figura 64. Accuracy y loss siguiendo la estrategia de entrenamiento número 2

Estrategia 3

En esta estrategia la política aplicada es “fixed”, que indica que la tasa de aprendizaje no varía su valor a lo largo de las iteraciones. Los resultados de *accuracy* y *loss* están representados en la Figura 65 para cada método de integración de frames.

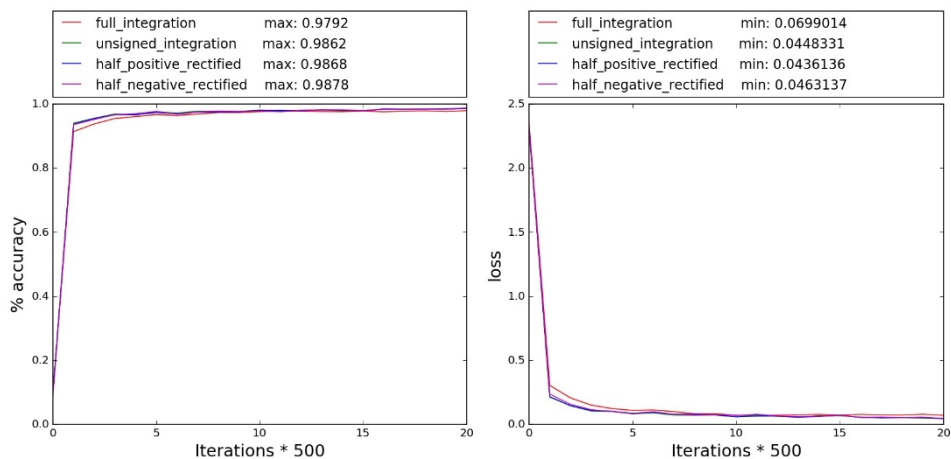


Figura 65. Accuracy y loss siguiendo la estrategia de entrenamiento número 3

En la Tabla 9 se muestran los máximos y mínimos de *accuracy* y *loss* recogidos de las gráficas de cada estrategia de entrenamiento para cada método de integración de frames. Se puede observar que los resultados son muy parecidas para el caso estudiado, teniendo la

mayor tasa de acierto el método *half_negative_rectified* utilizando la estrategia de entrenamiento número 3.

	Estrategia 1	Estrategia 2	Estrategia 3
full_integration	97.97% / 0.0636853	97.87% / 0.0681254	97.62% / 0.0699014
unsigned_integration	98.74% / 0.0438429	98.31% / 0.0548765	98.62% / 0.0448331
half_positive_rectified	98.54% / 0.0479827	98.16% / 0.0557975	98.68% / 0.0436136
half_negative_rectified	98.74% / 0.0452478	98.35% / 0.0573412	98.78% / 0.0463137

Tabla 9. Máximos y mínimos de accuracy / loss respectivamente según estrategia de entrenamiento y método de integración de frame

Para probar la robustez del modelo se han realizado tres tipos de pruebas diferentes, que tratan de perturbar uno o ambos datos de entrada y ver si el modelo es capaz de realizar la clasificación de manera satisfactoria. La perturbación llevada a cabo consiste en aplicar ruido blanco gaussiano con un SNR de +20db a aquel dato de entrada que se quiera alterar. Para la realización de las siguientes pruebas se ha utilizado el modelo entrenado siguiendo la estrategia que mejor resultado, en cuanto a *accuracy* ha obtenido por cada método de generación de frames.

Prueba 1

En esta prueba se altera el frame de la visión mientras que para el audio se utiliza una muestra del conjunto de test de la BD de audio. En la Figura 66 se puede ver un ejemplo de los datos de entrada utilizados para esta prueba.

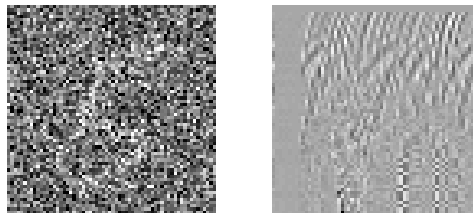


Figura 66. Dato de visión alterado y dato de audio sin alterar para el dígito 5

Prueba 2

Para esta prueba se altera el dato de audio y se utiliza para la visión una muestra del conjunto de test de la BD de visión. En la Figura 67 se ve un ejemplo de los datos de entrada de esta prueba.

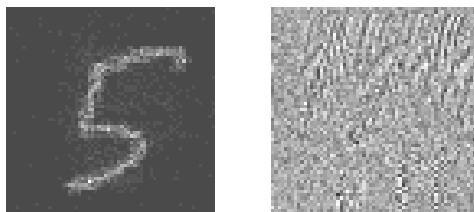


Figura 67. Dato de visión sin alterar y dato de audio alterado para el dígito 5

Prueba 3

En esta prueba se alteran ambos datos de entrada, audio y visión. En la Figura 68 se ve un ejemplo de estos datos de entrada utilizados para esta prueba.

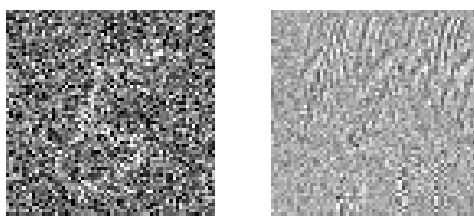


Figura 68. Ambos datos, visión y audio, alterados para el dígito 5

	Prueba 1	Prueba 2	Prueba 3
full_integration	96.10%	97.20%	81.35%
unsigned_integration	97.85%	98.51%	83.62%
half_positive_rectified	97.11%	97.95%	82.36%
half_negative_rectified	97.06%	97.87%	82.49%

Tabla 10. Accuracy de la robustez del modelo de fusión para cada método de integración de frames

En la Tabla 10 se presenta la comparativa en cuanto a *accuracy* de los diferentes métodos de integración de frames para cada una de las pruebas realizadas. Se observa que el modelo de fusión sensorial audio-visual aportado en este trabajo, presenta buena robustez cuando uno de los dos tipos de datos de entrada se ve alterado, consiguiendo el 97.85% cuando sólo se altera la visión y el 98.51% cuando sólo se altera el audio. Aunque si ambos sufren alteración, la tasa de acierto se ve disminuida considerablemente cayendo hasta el 81.35%.

VI. *Aportaciones más importantes y conclusiones*

- Se ha diseñado, implementado y probado una nueva solución OpenCL para plataformas de codiseño Host + FPGA capaz de ejecutar una CNN acelerando los tiempos de ejecución en CPU.
- Se ha implementado un *testbed* para pruebas de funcionamiento, rendimiento y consumo de hardware específico para la aceleración de CNN sobre plataformas SoC.
- Se ha diseñado, implementado y probado un sistema para integrar la información de un sensor de visión y otro de audio utilizando modelos estadísticos.
- Se ha construido una nueva plataforma para combinar información de dos sensores, monitorizar y logar flujo de datos AER a su entrada, siendo capaz de secuenciarlos a su salida.
- Se han propuesto cuatro nuevos métodos diferentes para la creación de frames a partir de información pulsante con el fin de poder utilizarlos como entrada en las CNN.
- Se ha diseñado, entrenado y probado, creemos que por primera vez, un modelo de CNN capaz de fusionar la información de un sensor de audio y otro sensor de vídeo para la realización de una tarea concreta referente a la clasificación de dígitos manuscritos y hablados.

La tendencia en los últimos años en los sistemas inteligentes ha sido el desarrollo de sistemas de *Deep Learning*, y más concretamente la implementación de CNN. Dada la complejidad de las CNNs necesarias para realizar procesados de cierto nivel, se han de realizar una ingente cantidad de operaciones aritméticas que limitan en gran medida el despliegue de esta tecnología en diversos ámbitos, como por ejemplo los sistemas en tiempo real, sistemas móviles o aquellos en los que el consumo de potencia es crítico. Es por ello que el diseño y desarrollo de hardware específico para implementar este tipo de redes es crucial.

Por otro lado, cuando se diseñan sistemas en los que la tolerancia a fallos y la robustez es crítica, se suele hacer uso de múltiples sensores con el fin de mejorar los resultados y ser capaces de continuar operando a pesar de que un sensor sufra perturbaciones o directamente

se averíe. En consecuencia, pensamos que el uso de sistemas multisensoriales fusionando su resultado mediante técnicas cognitivas, tal y como se hace en la biología, son una gran oportunidad para el desarrollo de sistemas críticos. Mediante la combinación de sensores neuromórficos basados en eventos con CNNs basadas en frames, se ha conseguido dar un paso hacia delante para el diseño y desarrollo de SCNNs que permitan la implementación de sistemas cognitivos neuroinspirados.

VII. Trabajo futuro

Tras las pruebas y resultados obtenidos en la aplicación OpenCL para la aceleración de CNN, se observa que la distribución en los kernels realizada por el framework Caffe no es óptima, obteniendo un decremento del rendimiento debido al cuello de botella sufrido en la memoria principal por la demanda de múltiples accesos de los kernels. Es por ello que se debe plantear una nueva estructura de kernels OpenCL para cada una de las capas que constituyen una CNN de tal forma que cada kernel copie a su memoria local aquellos datos con los que va a trabajar, disminuyendo de esta manera los accesos a memoria principal.

Con respecto al acelerador de CNN, presentado en el apartado III.3, existen varias mejoras en el controlador AXI-DMA, como es la implementación de buffers bidireccionales reduciendo así los movimientos de datos en memoria principal entre buffers de lectura y de escritura. También se pretende modificar el diseño para albergar dos dominios de reloj diferentes, uno propio para el acelerador (de menor frecuencia) y otro para el resto del sistema (de mayor frecuencia), con el fin de mantener siempre alimentado al acelerador con datos de entrada y enviar lo más rápido posible los datos de salida del acelerador al host. Otro aspecto a tener en cuenta es explotar la capacidad de pipeline que ofrece el uso de este tipo de hardware, proponiendo una plataforma escalable en la que se pueden utilizar más de un acelerador conectados entre sí.

Actualmente existen soluciones de aceleración tanto en CPU como en GPU para el proceso de entrenamiento de las ANN, por lo que sería interesante aplicar el paralelismo alcanzable en sistemas FPGA (hardware dedicado) para tal tarea tan costosa, computacionalmente hablando, que presenta el proceso de entrenamiento de las ANN.

Migrar el modelo de CNN para la fusión sensorial a un modelo SCNN⁵⁰, donde la entrada sería directamente la salida de los sensores utilizados en este trabajo con el fin de utilizarlo en aplicaciones de alta velocidad, aprovechando la capacidad de procesar cada spike sin la necesidad de esperar a tener un conjunto de ellos.

⁵⁰ Spiking Convolutional Neural Network

VIII. Bibliografía

- Aimar, A. et al., 2017. NullHop: A Flexible Convolutional Neural Network Accelerator Based on Sparse Representations of Feature Maps. *IEEE Transactions on Very Large Scale Integration Systems*, p.(En revisión).
- Aimar, A. et al., 2016. Nullhop: Flexibly efficient FPGA CNN accelerator driven by DAVIS neuromorphic vision sensor. In *Neural Information Processing Systems*. Barcelona.
- Alpha Data, 2016. ADM-PCIE-7V3 Product Page: Functionality and Applications. Available at: <http://www.alpha-data.com/dcp/products.php?product=adm-pcie-7v3> [Accessed March 4, 2017].
- Avnet, 2016. Zynq MMP | Zedboard. Available at: <http://zedboard.org/product/zynq-mmp> [Accessed May 26, 2017].
- Barlow, H., 1961. Possible principles underlying the transformations of sensory messages. *Sensory communication*, 6(2), pp.57-58.
- BBC, BBC - History - Cornelius Drebbel. Available at: http://www.bbc.co.uk/history/historic_figures/drebbel_cornelis.shtml [Accessed December 20, 2016].
- Bede, D. & Collum, M., 2002. Conserving the Wright Brothers' 1905 Flyer III. In *Textile Specialty Group Postprints*. American Institute for Conservation, pp. 58-66.
- Bellman, R., 1966. Dynamic programming. *Science*, 153, pp.34-37.
- Benvenuto, N. & Piazza, F., 1992. The backpropagation algorithm. *IEEE Transactions on Signal Processing*, 40(4), pp.967-969.
- Bermant, R.I. & Welch, R.B., 1976. Effect of degree of separation of visual-auditory stimulus and eye position upon spatial interaction of vision and audition. *Perceptual and Motor Skills*, 43(2), pp.487-493.
- Berner, R. et al., 2013. A 240× 180 10mW 12us latency sparse-output vision sensor for mobile applications. *VLSI Circuits (VLSIC), 2013 Symposium on*, pp.186-187. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6578656%5Cnpapers2://publication/uuid/3AF9D27E-5313-4407-9CA4-354081F5C082.
- Berner, R. et al., 2007. A 5 Meps \$100 USB2.0 Address-Event Monitor-Sequencer Interface. In *2007 IEEE International Symposium on Circuits and Systems*.

- Bolognini, N. et al., 2005. Visual search improvement in hemianopic patients after audio-visual stimulation. *Brain*, 128(12), pp.2830-2842.
- Braitenberg, V. & Schüz, P.D.D.A., 1991. Cortical architectonics. In *Anatomy of the Cortex*. Springer, pp. 147-149.
- Calvert, G., Spence, C. & Stein, B.E., 2004. *The handbook of multisensory processes*, MIT press.
- Cavigelli, Lukas and Benini, L., 2016. Origami: A 803 GOP/s/W Convolutional Network Accelerator. *IEEE Transactions on Circuits and Systems for Video Technology*, 8215(c), pp.1-14.
- Cerezuela-Escudero, E. et al., 2015. Musical notes classification with neuromorphic auditory system using FPGA and a convolutional spiking network. In *Proceedings of the International Joint Conference on Neural Networks*.
- Cerezuela Escudero, E., 2015. *Una aportación a los sistemas de procesamiento de la información basados en modelos neuronales pulsantes*. Universidad de Sevilla. Available at: <http://hdl.handle.net/11441/28886> [Accessed April 1, 2017].
- Chan, V., Liu, S.C. & van Schaik, A., 2007. AER EAR: A matched silicon cochlea pair with address event representation interface. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 54(1), pp.48-59.
- Chan, V.Y.S., Jin, C.T. & van Schaik, A., 2012. Neuromorphic audio-visual sensor fusion on a sound-localizing robot. *Frontiers in Neuroscience*, (FEB).
- Chen, Y.H. et al., 2016. 14.5 Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. In *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*. pp. 262-263.
- Clive, D.S. & Lyon, R.F., 1992. Asic implementation of the lyon cochlea model. *Signal Processing*, pp.673-676.
- Conradt, J. et al., 2009. A pencil balancing robot using a pair of AER dynamic vision sensors. In *Proceedings - IEEE International Symposium on Circuits and Systems*. pp. 781-784.
- Culurciello, E., Etienne-Cummings, R. & Boahen, K.A., 2003. A biomorphic digital image sensor. *IEEE Journal of Solid-State Circuits*, 38(2), pp.281-294.
- Cuppini, C. et al., 2010. An emergent model of multisensory integration in superior colliculus neurons. *Frontiers in integrative neuroscience*, 4, p.6.
- Dalton, P. et al., 2000. The merging of the senses: integration of subthreshold taste and smell. *Nature neuroscience*, 3(5), pp.431-432.
- Delbruck, T. & Lichtsteiner, P., 2007. Fast sensory motor control based on event-based hybrid neuromorphic-procedural system. 2007 *IEEE International Symposium on Circuits and Systems*, (ISCAS), pp.845-848. Available at: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4252767>.
- Deng, L. & Yu, D., 2014. *DEEP LEARNING: Methods and Applications*,

- Dominguez-Morales, M. et al., 2011. On the designing of spikes band-pass filters for FPGA. In *International Conference on Artificial Neural Networks*. pp. 389–396.
- Domínguez-Morales, M. et al., 2011. An approach to distance estimation with stereo vision using address-event-representation. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. pp. 190–198.
- Fujii, H. et al., 1996. Dynamical cell assembly hypothesis - Theoretical possibility of spatio-temporal coding in the cortex. *Neural Networks*, 9(8), pp.1303–1350.
- Fukushima, K. et al., 1970. An electronic model of the retina. *Proceedings of the IEEE*, 58(12), pp.1950–1951.
- Fukushima, K., 1975. Cognitron: A self-organizing multilayered neural network. *Biological Cybernetics*, 20(3–4), pp.121–136.
- Fukushima, K., 1980. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4), pp.193–202. Available at: <http://link.springer.com/10.1007/BF00344251> [Accessed January 8, 2017].
- Furber, S.B. et al., 2014. The SpiNNaker project. *Proceedings of the IEEE*, 102(5), pp.652–665.
- Gokhale, V. et al., 2014. A 240 g-ops/s mobile coprocessor for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. pp. 682–687.
- Gómez- Rodríguez, F. et al., 2010. Live demonstration: Real time objects tracking using a bio-inspired processing cascade architecture. In *ISCAS 2010 - 2010 IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems*. p. 1398.
- Gomez-Rodriguez, F. et al., 2006. AER tools for communications and debugging. *2006 IEEE International Symposium on Circuits and Systems*, pp.3253–3256.
- Gomez-Rodriguez, F. et al., 2016. ED-Scorbot: A robotic test-bed framework for FPGA-based neuromorphic systems. In *Proceedings of the IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics*.
- Graupe, D., 2016. *Deep Learning Neural Networks: Design and Case Studies*, World Scientific Publishing Co Inc.
- Guo, M. & Chen, S., 2016. *Dynamic Vision Sensor with Direct Logarithmic Output and Full-Frame Picture-on-Demand*. School of Electrical and Electronic Engineering, University of Singapore. Available at: <https://repository.ntu.edu.sg/handle/10356/69258?show=full>.
- Gutierrez-Galan, D. et al., 2017. Embedded neural network for real-time animal behavior classification. *Neurocomputing*, Accepted.
- Hall, D.L. & Llinas, J., 1997. An introduction to multisensor data fusion. *Proceedings of the IEEE*, 85(1), pp.6–23. Available at:

http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=554205&isnumber=12063%5Cnhttp://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=554205.

- Harmon, L.D. & Knowlton, K.C., 1969. Picture Processing by Computer. *Science*, 164(3875), pp.19–29. Available at: <http://science.sciencemag.org/content/164/3875/19>.
- Haykin, S., 1994. Neural networks-A comprehensive foundation. *New York: IEEE Press*. Herrmann, M., Bauer, H.-U., & Der, R, psychology, p.pp107-116.
- Hinton, G. et al., 2012. Deep Neural Networks for Acoustic Modeling in Speech Recognition. *IEEE Signal Processing Magazine*, 29(6), pp.82–97.
- Hynna, K. & Boahen, K., 2001. Space-rate coding in an adaptive silicon neuron. *Neural Networks*, 14(6), pp.645–656.
- Iakymchuk, T. et al., 2014. An AER handshake-less modular infrastructure PCB with x8 2.5Gbps LVDS serial links. In *Proceedings - IEEE International Symposium on Circuits and Systems*. pp. 1556–1559.
- Indiveri, G., Chicca, E. & Douglas, R., 2006. A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity. *IEEE Transactions on Neural Networks*, 17(1), pp.211–221.
- jAER, 2007. jAER Open Source Project. Available at: <http://jaerproject.org> [Accessed July 28, 2014].
- Jia, Y. et al., 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093*.
- Jimenez-Fernandez, A. et al., 2016. A Binaural Neuromorphic Auditory Sensor for FPGA: A Spike Signal Processing Approach. *IEEE Transactions on Neural Networks and Learning Systems*, PP, pp.1–15. Available at: <http://ieeexplore.ieee.org/document/7523402/> [Accessed December 20, 2016].
- Jimenez-Fernandez, A. et al., 2012. A Neuro-Inspired Spike-Based PID Motor Controller for Multi-Motor Robots with Low Cost FPGAs. *Sensors*, 12(4), p.3831. Available at: <http://www.mdpi.com/1424-8220/12/4/3831>.
- Jimenez-Fernandez, A. et al., 2010. Building blocks for spikes signals processing. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*. pp. 1–8.
- Kuffler, S.W., 1953. Discharge patterns and functional organization of mammalian retina. *Journal of neurophysiology*, 16(1), pp.37–68.
- Laboratories, A.T.& T.B. & Schindler, G.E., 1975. *A History of Engineering and Science in the Bell System*, Bell Telephone Laboratories.
- Lazzaro, J. et al., 1993. Silicon Auditory Processors as Computer Peripherals. *IEEE Transactions on Neural Networks*, 4(3), pp.523–528.
- Lazzaro, J. & Wawrzynek, J., 1995. A multi-sender asynchronous extension to the AER protocol. In *arolsi*. pp. 158–171.

- LeCun, Y. et al., 1989. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4), pp.541–551.
- LeCun, Y., Bottou, L., et al., 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), pp.2278–2323.
- LeCun, Y., Cortes, C. & Burges, C.J.C., 1998. The MNIST database of handwritten digits.
- Leong, M.P., Jin, C.T. & Leong, P.H.W., 2003. An FPGA-based electronic cochlea. *Eurasip Journal on Applied Signal Processing*, 2003(7), pp.629–638.
- Li, C.H., Delbruck, T. & Liu, S.C., 2012. Real-time speaker identification using the AEREAR2 event-based silicon cochlea. In *ISCAS 2012 - 2012 IEEE International Symposium on Circuits and Systems*. pp. 1159–1162.
- Lichtsteiner, P., Posch, C. & Delbruck, T., 2008. A 128 x 128 120 dB 15 us latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, 43(2), pp.566–576.
- Linares-Barranco, A. et al., 2006. On algorithmic rate-coded AER generation. *IEEE Transactions on Neural Networks*, 17(3), pp.771–788.
- Linares-Barranco, A. et al., 2004. On synthetic AER generation. *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512)*, 5.
- Liu, S.-C., 2002. *Analog VLSI: circuits and principles*, MIT press.
- Liu, S.-C., 2015. *Event-based neuromorphic systems*, John Wiley & Sons.
- Liu, S.C. et al., 2010. Event-based 64-channel binaural silicon cochlea with Q enhancement mechanisms. In *ISCAS 2010 - 2010 IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems*. pp. 2027–2030.
- Longinotti, L. & Brändli, C., 2014. *CAER: a Framework for Event-based Processing on Embedded Systems: Bachelor Thesis*, Institut für Neuroinformatik. Available at: <https://books.google.es/books?id=KLCooAEACAAJ>.
- Luo, R.C., Yih, C.-C. & Su, K.L., 2002. Multisensor fusion and integration: approaches, applications, and future research directions. *IEEE Sensors Journal*, 2(2), pp.107–119.
- Lyon, R.F. & Mead, C., 1988. An Analog Electronic Cochlea. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(7), pp.1119–1134.
- Maass, W. & Bishop, C.M., 1999. *Pulsed Neural Networks*,
- Maher, M.A.C. et al., 1989. Implementing Neural Architectures Using Analog VLSI Circuits. *IEEE Transactions on Circuits and Systems*, 36(5), pp.643–652.
- Mahowald, M., 1992. *VLSI analogs of neuronal visual processing: a synthesis of form and function*. California Institute of Technology.
- Masland, R.H., 2001. The fundamental plan of the retina. *Nature neuroscience*, 4(9), pp.877–886.
- Mead, C., 1989. *Analog VLSI and Neural Systems*, Boston, MA, USA: Addison-Wesley Longman

Publishing Co., Inc.

- Mead, C.A. & Mahowald, M.A., 1988. A silicon model of early visual processing. *Neural Networks*, 1(1), pp.91–97.
- Meddis, R., 1988. Simulation of auditory-neural transduction: further studies. *The Journal of the Acoustical Society of America*, 83(3), pp.1056–1063.
- Meddis, R., 1986. Simulation of mechanical to neural transduction in the auditory receptor. *The Journal of the Acoustical Society of America*, 79(3), pp.702–711. Available at: <http://scitation.aip.org/content/asa/journal/jasa/79/3/10.1121/1.393460>.
- Meteyard, E., 1871. *A Group of Englishmen (1795 to 1815) Being Records of the Younger Wedgwoods and Their Friends: Embracing the History of the Discovery of Photography and a Facsimile of the First Photograph*, Longmans, Green, and Company.
- Moore, G.E., 1965. *Cramming more components onto integrated circuits (Reprinted from Electronics, pg 114-117, April 19, 1965)*, Available at: <papers3://publication/uuid/8E5EB7C8-681C-447D-9361-E68D1932997D>.
- O'Connor, P. et al., 2013. Real-time classification and sensor fusion with a spiking deep belief network. *Frontiers in Neuroscience*, (7 OCT).
- Opal Kelly, 2014. Opal Kelly XEM6010-LX150 board. Available at: <https://www.opalkelly.com/products/xem6010>.
- Penrose, R., 1990. *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics*, Available at: http://link.springer.com/chapter/10.1007/978-94-010-1248-5_2.
- Perez-Peña, F. et al., 2013. Neuro-inspired spike-based motion: from dynamic vision sensor to robot motor open-loop control through spike-VITE. *Sensors (Basel, Switzerland)*, 13(11).
- Posch, C., Matolin, D. & Wohlgenannt, R., 2008. An asynchronous time-based image sensor. In *Proceedings - IEEE International Symposium on Circuits and Systems*. pp. 2130–2133.
- Rakic, P., 1988. Specification of Cerebral Cortical Areas. *Science*, 241, pp.170–176.
- Rios-Navarro, A. et al., 2016. A 20Mevps/32Mev event-based USB framework for neuromorphic systems debugging. In *2016 2nd International Conference on Event-Based Control, Communication, and Signal Processing, EBCCSP 2016 - Proceedings*.
- Rios-Navarro, A. et al., 2012. Detection system from the driver's hands based on Address Event Representation (AER). In *Computational Modelling of Objects Represented in Images: Fundamentals, Methods and Applications III - Proceedings of the International Symposium, CompIMAGE 2012*. pp. 7–11.
- Rios-Navarro, A. et al., 2015. Real-Time motor rotation frequency detection with event-based visual and spike-based auditory AER sensory integration for FPGA. In *Proceedings of 1st International Conference on Event-Based Control, Communication and Signal Processing, EBCCSP 2015*.
- Rüedi, P.F. et al., 2003. A 128 × 128 Pixel 120-dB Dynamic-Range Vision-Sensor Chip for Image

- Contrast and Orientation Extraction. In *IEEE Journal of Solid-State Circuits*. pp. 2325–2333.
- Rumelhart, D.E., 1986. Learning intenal representations by error propagation. In *Parallel Distributed Processing: Foundations*. pp. 318–362. Available at: http://books.google.si/books/about/Parallel_Distributed_Processing_Foundati.html?id=eFPqQMBK-p8C&pgis=1.
- Schiavo, G.E., 1958. *Antonio Meucci: Inventor of the Telephone*, Vigo Press.
- Serrano-Gotarredona, R. et al., 2009. CAVIAR: A 45k neuron, 5M synapse, 12G connects/s AER hardware sensory--processing--learning--actuating system for high-speed visual object recognition and tracking. *IEEE Transactions on Neural Networks*, 20(9), pp.1417–1438.
- Serrano-Gotarredona, T. & Linares-Barranco, B., 2013. A 128 × 128 1.5% contrast sensitivity 0.9% FPN 3 μs latency 4 mW asynchronous frame-free dynamic vision sensor using transimpedance preamplifiers. *IEEE Journal of Solid-State Circuits*, 48(3), pp.827–838.
- Shadlen, M.N. & Newsome, W.T., 1994. Noise, neural codes and cortical organization. *Current Opinion in Neurobiology*, 4(4), pp.569–579.
- Sivilotti, M.A., 1991. *Wiring considerations in analog VLSI systems, with application to field-programmable networks*. Citeseer.
- Srivastava, N. et al., 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1), pp.1929–1958.
- Stromatias, E. et al., 2015. Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms. *Frontiers in Neuroscience*, 9(JUN).
- Tapiador-Morales, R. et al., 2015. System based on inertial sensors for behavioral monitoring of wildlife. In *IEEE CITS 2015 - 2015 International Conference on Computer, Information and Telecommunication Systems*. Institute of Electrical and Electronics Engineers Inc.
- Tapiador, R. et al., 2017. Comprehensive Evaluation of OpenCL-based Convolutional Neural Network Accelerators in Xilinx and Altera FPGAs. *International Work-conference on Artificial Neural Networks*, p.Accepted.
- Werblin, F.S. & Dowling, J.E., 1969. Organization oh the retina of the mdpuppy necturus maculosus: II Intracellular recording. *J Neurophysiol*, 32, pp.339–355.
- Westerman, W.C., Northmore, D.P.M. & Elias, J.G., 1997. Neuromorphic Synapses for Artificial Dendrites. *Analog Integrated Circuits and Signal Processing*, 12(1--2), pp.167–184.
- Xilinx, 2016a. *SDAccel Environment Optimization Guide*, Available at: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_4/ug1207-sdaccel-optimization-guide.pdf.
- Xilinx, 2016b. *SDAccel Environment User Guide*, Available at: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_4/ug1023-sdaccel-user-guide.pdf.

- Yousefzadeh, A., Serrano-Gotarredona, T. & Linares-Barranco, B., 2012. MNIST-DVS and FLASH-MNIST-DVS Databases. *Instituto de Microelectrónica de Sevilla, IMSE-CNM (CSIC & Univ. of Sevilla)*.
- Zaghloul, K.A. & Boahen, K., 2004a. Optic Nerve Signals in a Neuromorphic Chip I: Outer and Inner Retina Models. *IEEE Transactions on Biomedical Engineering*, 51(4), pp.657–666.
- Zaghloul, K.A. & Boahen, K., 2004b. Optic Nerve Signals in a Neuromorphic Chip II: Testing and Results. *IEEE Transactions on Biomedical Engineering*, 51(4), pp.667–675.
- Zamarreno-Ramos, C. et al., 2013. Multicasting mesh AER: A scalable assembly approach for reconfigurable neuromorphic structured AER systems. Application to ConvNets. *IEEE Transactions on Biomedical Circuits and Systems*, 7(1), pp.82–102.
- Zhang, C. et al., 2016. Caffeine: Towards Uniformed Representation and Acceleration for Deep Convolutional Neural Networks. In *ICCAD*. pp. 1–8. Available at: <http://dl.acm.org/citation.cfm?doid=2966986.2967011>.
- Zhang, C. et al., 2015. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '15*, pp.161–170. Available at: <http://dl.acm.org/citation.cfm?id=2689060>.

V. Anexo

OKAERTool. Herramienta de depuración y testeo de sistemas neuromórficos

En este anexo se explica detalladamente la herramienta diseñada para la depuración y testeo de sistemas neuromórficos desarrollados en este trabajo. Actualmente la comunidad neuromórfica está creciendo, y con ello el desarrollo en áreas de sensores (Lichtsteiner et al. 2008; Serrano-Gotarredona & Linares-Barranco 2013; Jimenez-Fernandez et al. 2016; Liu et al. 2010), aprendizaje y robótica (Perez-Peña et al. 2013; Gomez-Rodriguez et al. 2016). Algunos de estos sistemas han sido desarrollados para identificar a una persona hablando (Li et al. 2012), para estimar la distancia de objetos en movimiento utilizando un sistema de estéreo-visión AER (Domínguez-Morales et al. 2011) o para seguir múltiples objetos en paralelo utilizando una arquitectura en cascada de procesamiento (Gómez- Rodríguez et al. 2010). Además, existen algunos trabajos que combinan dos tipos de sensores (visión y audio) para desarrollar sistemas inspirados en la fusión sensorial (O'Connor et al. 2013).

Con el fin de hacer más fácil el desarrollo de estos sistemas basados en AER, es necesario contar con un conjunto de herramientas capaces de realizar las siguientes tareas:

- *Monitor*: Observar la salida del sistema AER en tiempo real para ser visualizada en un ordenador con la máxima precisión temporal posible.
- *Sequencer*: Adquiere la información AER cargada previamente en un fichero y genera un stream con dichos datos a la salida del sistema con el fin de tener una entrada controlada para un sistema AER.
- *Logger*: Guarda la información de la salida de un sensor o sistema AER en una memoria local a la mayor velocidad posible. Esta información puede ser descargada al ordenador para su posterior visualizado/almacenamiento o puede ser reproducida por la salida del sistema

- *Player*: Reproduce la información almacenada en la memoria para ser utilizada como entrada en otro sistema AER.
- *Merger*: Multiplexa la información de salida de varios sensores o sistemas AER en un solo stream de datos.

En la literatura existe un conjunto de herramientas que realizan algunas de estas funcionalidades, pero no unificadas en la misma plataforma y careciendo de capacidad de almacenamiento (Berner et al. 2007), o completa conectividad con el ordenador (Gomez-Rodriguez et al. 2006). La Tabla 11 muestra una comparativa entre las distintas herramientas.

Comparación	USBAERmini2	USBAER	OKAERTool
Capacidad	N/A	2MBytes	128MBytes
Tamaño del EVENTO AER	16 bits	16 bits	16 bits
Ancho de banda del <i>monitor</i>	5Mevps (6Mevps de pico)	N/A	6.5Mevps (8Mevps de pico)
Ancho de banda del <i>player/logger</i>	N/A	3.75Mevps 12Mevps	6.5Mevps 20Mevps
Consumo (mA)	60mA	30mA	32mA

Tabla 11. Comparación entre varias herramientas para la depuración de sistemas AER

Para cubrir el conjunto de funcionalidades AER anteriormente expuestas, se requiere una plataforma hardware capaz de ser reconfigurada, con una alta capacidad de memoria y con baja latencia. Además, debe de ser fácilmente adaptable a otras plataformas de procesamiento AER como la placa AER-Node, por lo que se ha diseñado una PCB para la interconexión entre ambas plataformas. Esta plataforma (Iakymchuk et al. 2014), la AER-Node, cuenta con una FPGA Spartan6 (XC6S150FXT) y es utilizada para implementar sistemas de procesamiento AER. Cuenta con dos conexiones paralelas de 40 pines así como con 4 conectores SATA para comunicaciones bidireccionales LVDS de alta velocidad.

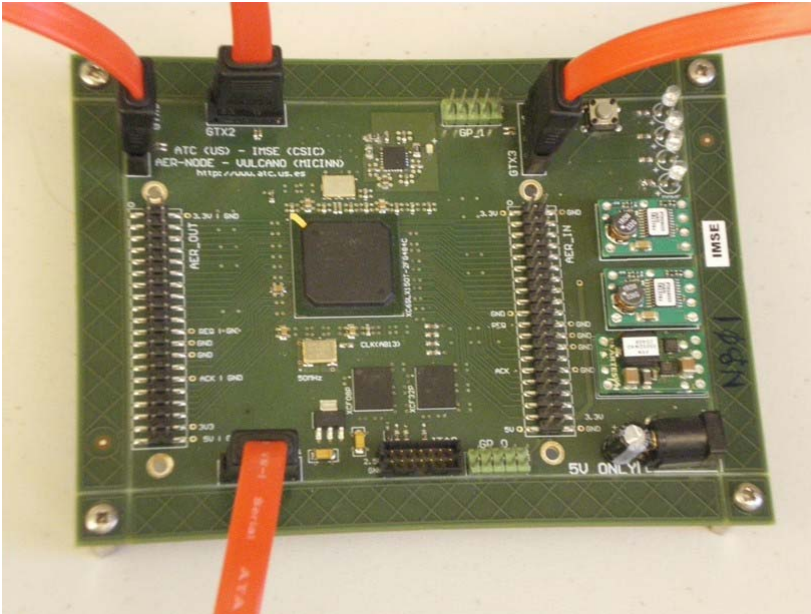


Figura 69. Plataforma AERNode de procesamiento basado en spikes con conexión LVDS

Para la implementación de la herramienta OKAERTool se ha utilizado una plataforma comercial llamada Opal Kelly XEM6010-LX150 (Opal Kelly 2014), que cuenta con una FPGA Spartan 6 XC6SLX150-2FGG, 128MBytes DDR2 y un puerto USB 2.0 (Cypress FX2LP-CY6801A). En la Figura 70 se puede ver el diagrama de bloques de la placa. Además, cuenta con dos puertos de expansión de 80 pines para realizar una conexión placa a placa, la cual ha sido utilizada para conectarla a la PCB diseñada para la interconexión entre la AER-Node y la OKAERTool. Esta placa de interconexión cuenta con 4 interfaces AER, 3 funcionan como entrada (1 conector tipo CAVIAR y 2 tipos ROME (Serrano-Gotarredona et al. 2009)) y una como salida (tipo CAVIAR). Además tiene dos conectores de propósito general de 8 bits que pueden ser usados como expansión para las interfaces AER o para configuración, por ejemplo, una interfaz SPI⁵¹. La Figura 71 muestra la plataforma OKAERTool al completo.

⁵¹ Serial Peripheral Interface

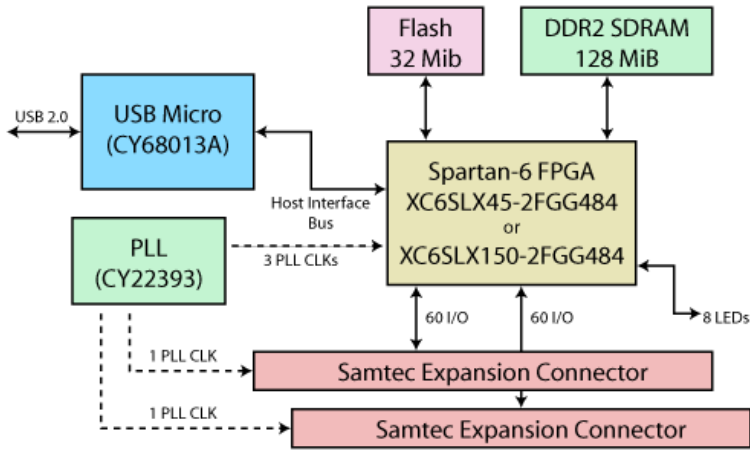


Figura 70. Diagrama de bloques de la plataforma Opal Kelly XEM6010-LX150-2FGG (Opal Kelly 2014)

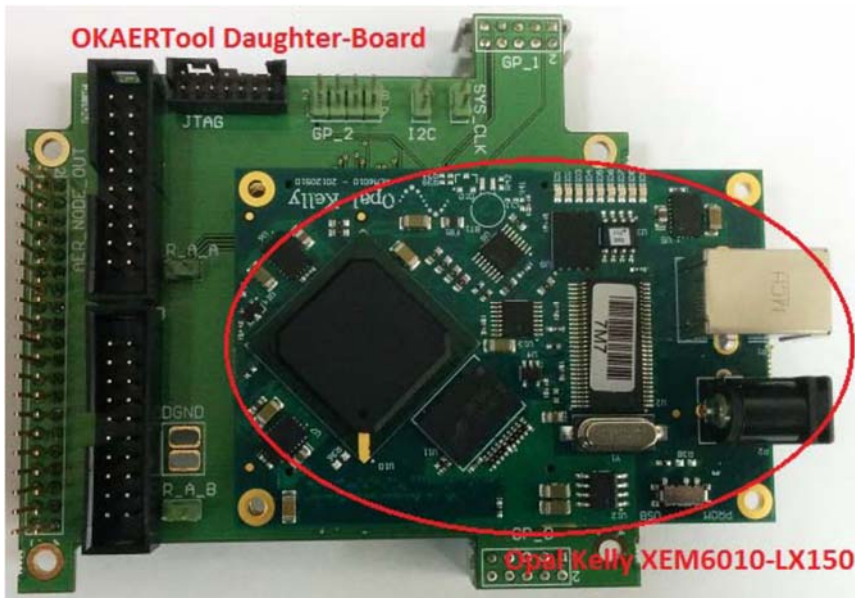


Figura 71. Plataforma OKAERTool. Placa de interconexión AER-Node-OKAERTool junto con la placa Opal Kelly XEM6010-LX150-2FGG

Como se ha comentado anteriormente, la OKAERTool alberga una serie de herramientas AER que facilitan las tareas de depuración y testeo de los sistemas basados en eventos. La Figura 72 muestra un diagrama de bloques correspondiente a la lógica implementada de forma modular en la FPGA y descrita en VHDL. El bloque *merger* es capaz de unificar las dos

entradas AER con conectores tipo ROME en un solo stream de datos AER. Luego, este stream de datos puede ser seleccionado o no por un multiplexor (bloque *MUX*) para ser almacenado o monitorizado por los módulos *logger* o *monitor* respectivamente. Este multiplexor puede seleccionar la salida del *merger* o la entrada AER tipo CAVIAR (Serrano-Gotarredona et al. 2009) que puede venir de la salida de la plataforma AER-Node, con el fin de logar y monitorizar esa información. Para las operaciones de logado, todo el flujo AER de datos recibido se guarda en una DDR2 (accesible a través de un IPCore de controlador DDR2), mientras que para la monitorización, el stream de datos AER son enviados al ordenador a través del microcontrolador Cypress FX2 (conectado a la lógica de un IPCore de controlador USB). Por otro lado, la OKAERTool es capaz de generar en su salida un stream de datos AER mediante el bloque *player*. Esos eventos que reproduce en su salida pueden venir de dos fuentes diferentes: un conjunto de datos que han sido guardados en la memoria DDR2 utilizando el módulo *logger* o un conjunto de datos guardados también en la memoria pero que han sido descargados desde el ordenador a la OKAERTool utilizando la interfaz USB de la plataforma. Una máquina de estados finita controla el protocolo de *handshaking* para enviar los datos AER por el puerto de salida. Esta salida puede ser conectada a la entrada de otro sistema AER o se puede conectar a la entrada de la plataforma AER-Node.

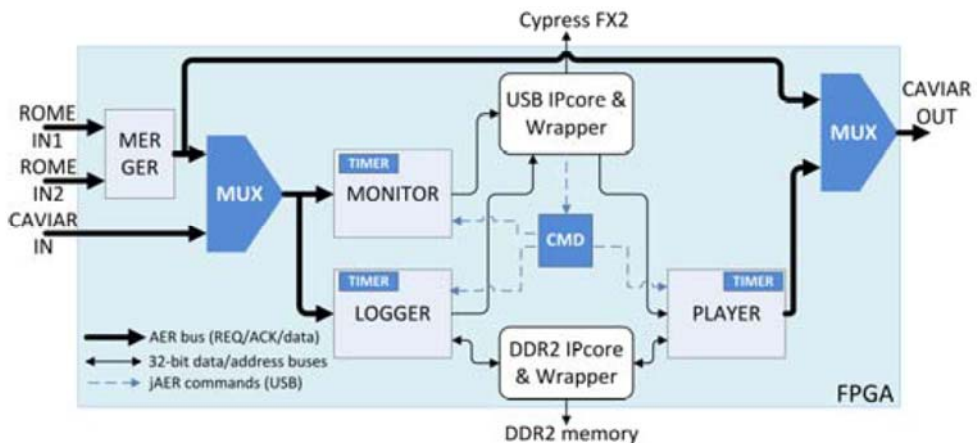


Figura 72. Diagrama lógico de bloques de la plataforma OKAERTool

Además, gracias a la conexión de la salida del módulo *merger* directa al multiplexor de salida general, la OKAERTool puede trabajar en paralelo como una entrada pasarela al sistema AER que se encuentre en ejecución en la AERNode y a su vez recolectar los datos de salida de dicho sistema (bien sea mediante el módulo *logger* o el *monitor*) para depurar sus operaciones. Otra combinación posible de operaciones sería secuenciar eventos guardados en

la DDR2 por la salida de la OKAERTool hacia la AER-Node donde serían procesados, y recoger los datos resultantes para que mediante el módulo *monitor* sean enviados al ordenador. En las siguientes subsecciones se detallará el funcionamiento de cada uno de los módulos que componen la plataforma.

A. Merger

Este módulo es capaz de combinar spikes provenientes de múltiples fuentes en un solo bus AER. Particularmente en esta implementación sólo se permiten dos entradas. Cada entrada tiene una lógica de control de *handshaking* para el protocolo AER y una pequeña FIFO para guardar los eventos que se van recolectando con el fin de evitar posibles bloqueos. A continuación, un arbitrador lee ambas FIFOs y multiplexa ambas salidas en el tiempo. La Figura 73 muestra la máquina de estados finita de este módulo para el proceso de arbitraje. En caso de producirse colisiones, se alternan la salida de ambas FIFOs.

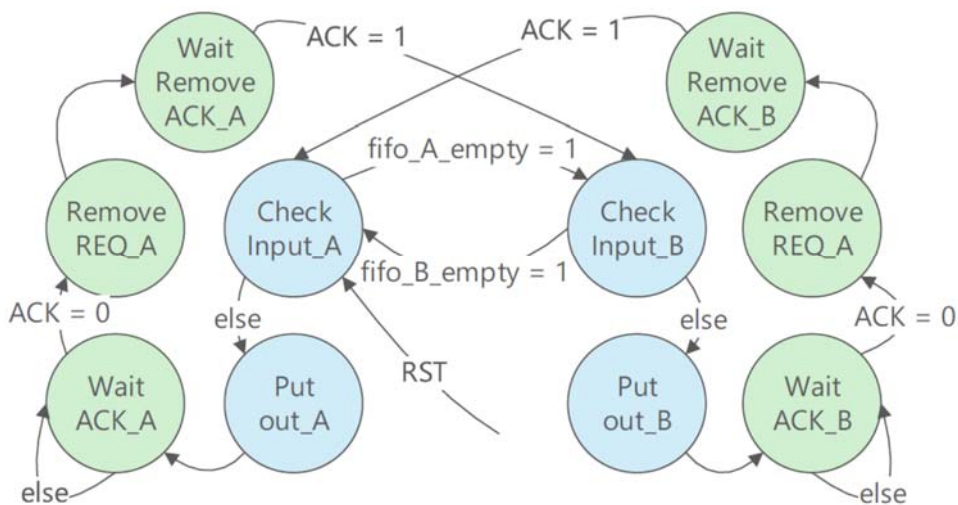


Figura 73. Máquina de estados finita del módulo merger

B. Monitor

Mediante el uso de este bloque se puede observar el stream de salida de un sistema AER. Esta observación se lleva a cabo asignando una marca de tiempo cuando un spike (evento AER) llega, y transmitiendo luego la pareja spike AER/marca de tiempo hacia el ordenador mediante la interfaz USB. Este módulo se utiliza para analizar el rendimiento o el comportamiento real de un sistema neuromórfico. La funcionalidad de este monitor utiliza

varios bloques del diagrama de la Figura 72: el módulo *monitor*, la interfaz USB mediante el Cypress FX2 y el decodificador de comandos (*CMD*). La entrada del monitor puede venir de la salida del *merger* o de la entrada AER tipo CAVIAR. El ancho de banda máximo alcanzado es de 5Mevps, debido al cuello de botella de la interfaz USB 2.0. La Figura 74 muestra la máquina de estados finita del *monitor*. Cuando un comando se recibe a través de la interfaz USB para comenzar la monitorización, la máquina de estados sale del estado “*idle*” y pasa al estado “*wait_event*” con el fin de esperar la activación de la señal *REQ* (activa a nivel bajo). Una vez que la señal *REQ* es recibida, en el estado “*capture*” se lee tanto el spike que está en el bus de entrada como la marca de tiempo generada por un contador, activando posteriormente la señal *ACK*. El estado “*capture*” se realiza en un solo ciclo de reloj, avanzando al siguiente estado llamado “*wait_req*”. Este estado asegura el correcto comportamiento del protocolo handshake. Este módulo posee un mecanismo de desborde del contador de marcas de tiempo, generando un evento especial cada vez que este caso sucede, con el fin de reconstruir de manera temporalmente correcta todos los eventos que han sido monitorizados.

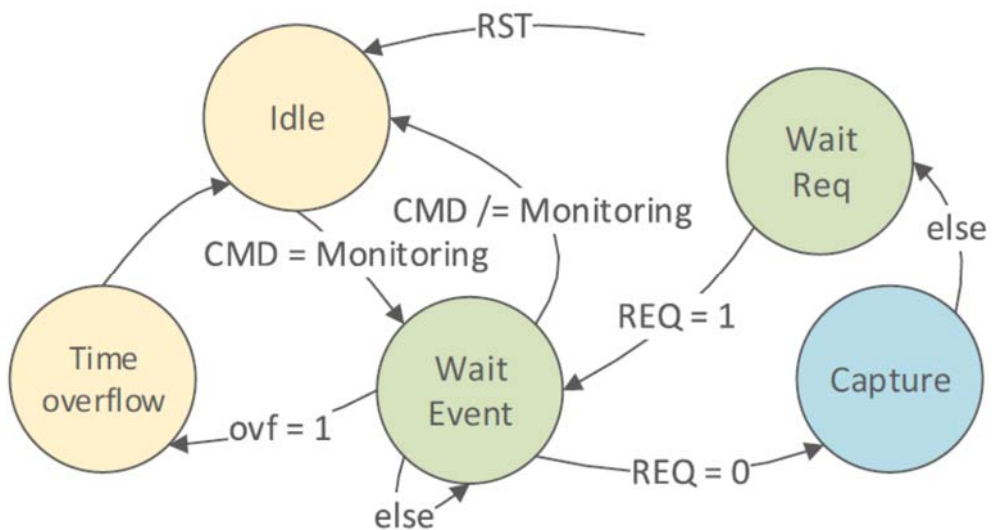


Figura 74. Máquina finita de estados del módulo *monitor*

C. Player

El uso de sensores conectados directamente a las entradas de sistemas AER que se encuentran bajo depuración, no es una buena práctica para asegurar un comportamiento repetitivo de los sistemas aún en evaluación. Por este motivo es necesario ser capaz de secuenciar un stream sintéticamente generado o un stream capturado previamente mediante

el *logger* de manera repetitiva. Este módulo nos permite descargar un conjunto de paquetes de datos AER mediante el USB2.0 desde el ordenador hacia la memoria DDR2, para más tarde secuenciar esos datos a través del bus AER de salida. Gracias a que los datos, que son guardados en la memoria DDR2, poseen su marca de tiempo en la que fueron capturados, este módulo es capaz de reproducir los spikes respetando la representación temporal.

La funcionalidad del *player* es dividida en dos máquinas de estados finitas. La primera, mostrada en la Figura 75 funciona como interfaz entre la DDR2 y las FIFOS de entrada y de salida. Estas FIFOS son utilizadas posteriormente por la segunda máquina de estados, representada en la Figura 76, que es la encargada de secuenciar los eventos por el puerto de salida de la herramienta. La primera máquina de estados tiene dos modos de funcionamiento: (1) *loading mode*, que se corresponde con la parte derecha de la Figura 75 y se encarga de guardar en la DDR2 los datos recibidos a través de la interfaz USB; (2) *playing mode*, que se encuentra en la parte izquierda de la Figura 75, y es la que obtiene los datos que previamente han sido cargados en la DDR2 para irlos preparando en la FIFO de salida a medida que la segunda máquina de estados va demandando nuevos eventos para ser secuenciados por la salida del sistema. Diferentes comandos recibidos por el USB se utilizan para cambiar de un modo a otro.

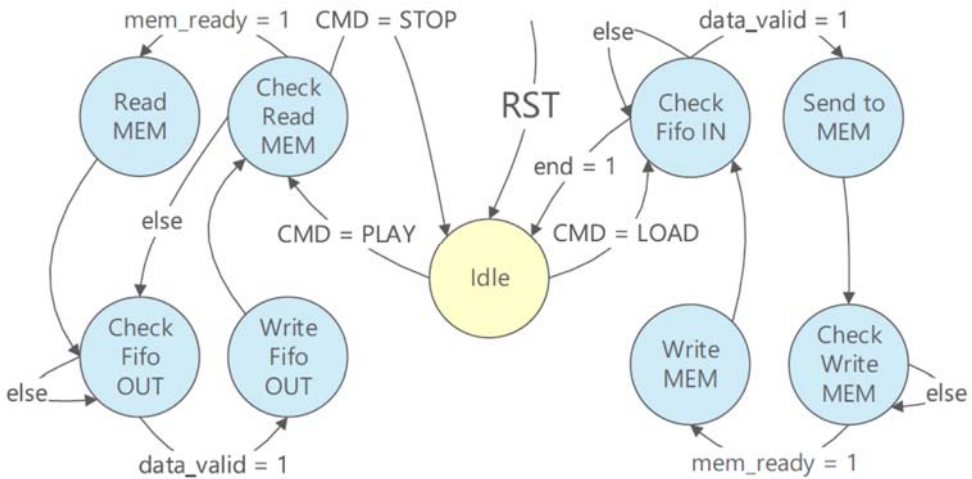


Figura 75. Primera máquina de estados del módulo *player*

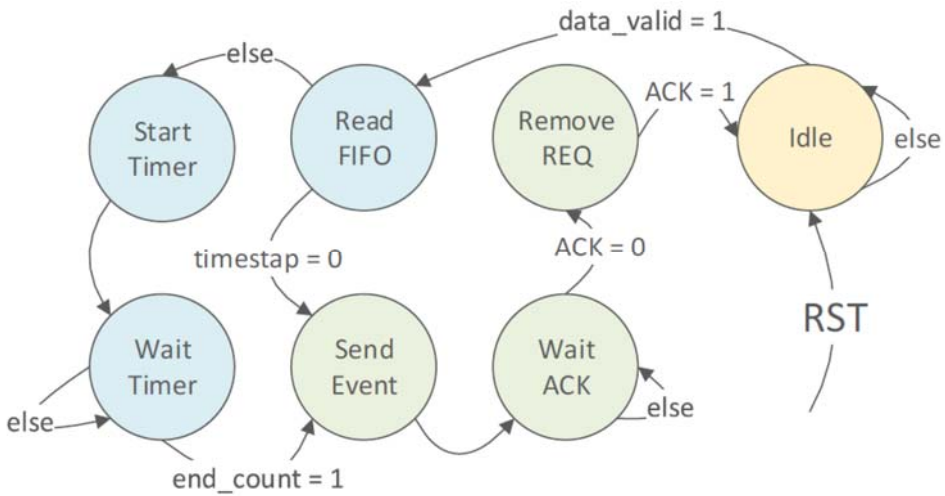


Figura 76. Segunda máquina de estados del módulo *player*

La parte que se encarga de secuenciar los spikes del módulo *player*, envía por la salida del sistema los eventos que se encuentran en la FIFO de salida (que anteriormente se encontraban en la DDR2) utilizando la marca de tiempo asociada a cada spike. Esta marca temporal representa el tiempo entre dos spikes consecutivos. Así pues, la marca de tiempo es utilizada para esperar un tiempo determinado desde que se envió un spike por la salida hasta que se debe de enviar el siguiente. La Figura 76 muestra la máquina de estados que implementa el secuenciador del módulo *player*. Como se puede ver en dicha figura, se espera hasta que hay un dato válido en la FIFO de salida, y una vez recibido, comienza una cuenta atrás utilizando un contador interno hasta que dicha cuenta llega a cero, momento para poner el spike en el bus AER de salida.

D. Logger

Esta funcionalidad es similar al *monitor*, pero la diferencia es que los spikes de entrada son marcados temporalmente y almacenados en la DDR2 en vez de ser enviados al ordenador a través del USB. Los eventos almacenados en la DDR2 pueden ser posteriormente descargados en el ordenador o secuenciados por el *player*. El ancho de banda máximo de captura de los stream de datos AER entre el *monitor* y el *logger* es muy relevante, 5Mevps en el *monitor* y 20Mevps en el *logger*. La capacidad de la DDR2 que posee la Opal Kelly es de 128MBytes, y la resolución en bits tanto de los spikes como la marca temporal es de 16 bits para cada uno, por lo que se pueden guardar en esta memoria hasta 32 Meventos, lo que hace posible grabar varios segundos de una típica actividad AER.

La Figura 77 muestra la máquina de estados del módulo *logger*. Al igual que el módulo *player* también posee dos modos diferentes de funcionamiento: (1) *logging mode*, se corresponde con la parte derecha de la Figura 77 y captura los spikes de entrada con su marca temporal y los guarda en la DDR2; (2) *download mode*, se corresponde con la parte izquierda de la imagen y obtiene los spikes guardados en la memoria para enviarlos al ordenador a través de la interfaz USB.

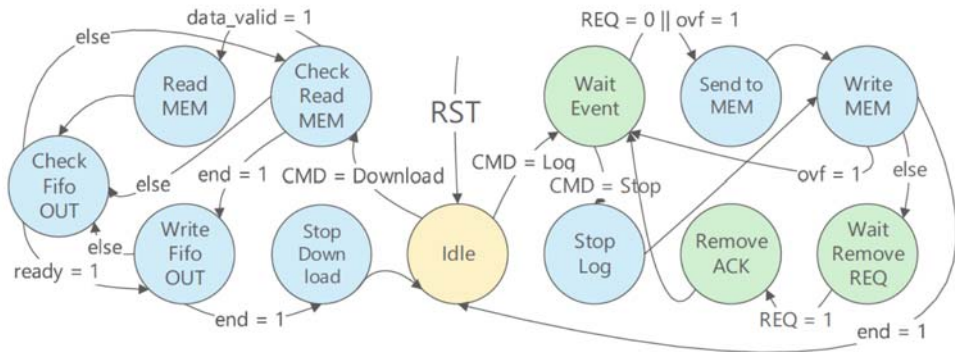


Figura 77. Máquina de estados del módulo *logger*

Todo el sistema es controlado a través de comandos que son recibidos a través de la interfaz USB. Estos comandos son interpretados por el módulo *CMD*, que es responsable de habilitar y deshabilitar cada una de las máquinas de estados de cada uno de los módulos. Los bloques restantes que aparecen en la Figura 72, *USB IPCore Wrapper* y *DDR2 IPCore Wrapper*, son proporcionados por el fabricante en el caso del *USB IPCore Wrapper* para enviar y recibir datos a través de la interfaz USB, y por parte de Xilinx en el caso del *DDR2 IPCore Wrapper* para enviar y recibir datos a/de la memoria DDR2. El sistema posee dos dominios de reloj, una de 48MHz para la lógica relacionada con el USB y de 100MHz para el resto del sistema.

La herramienta *OKAERTool* está integrada en la herramienta *JAER* (*JAER* 2007) y posee una interfaz de control para manejar todas sus funcionalidades. En la Figura 78 se pueden ver los controles de la herramienta para seleccionar cada uno de sus modos de funcionamiento así como la entrada o salida deseada del sistema. En dicha figura se observa, por ejemplo, que en ese caso se encuentra seleccionada como entrada la interfaz tipo *CAVIAR* (que se corresponde con la salida de la plataforma *AERNode*) y como salida del sistema, la salida del módulo *merger*. Con esta configuración, se podría por ejemplo monitorizar la salida del filtro

o proceso que se encuentre en ejecución en la plataforma AERNode, que toma como entrada lo que se encuentre conectado en las interfaces tipo ROME de la OKAERTool.

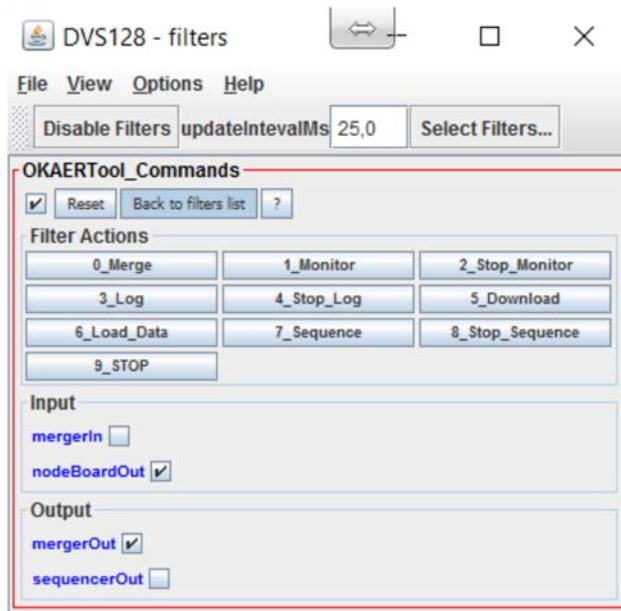


Figura 78. Interfaz de control de la herramienta OKAERTool en jAER

Además, la herramienta propuesta en este anexo, cuenta con una interfaz de comandos integrada en MATLAB. Esta interfaz permite obtener los datos en este software que es bastante útil para el tratamiento y visualizado de los eventos en distintos tipos de gráficas. Esta interfaz cuenta con los siguientes comandos:

- $[ok] = initializeOpalKellyFPGA()$
Inicializa la interfaz, configura el USB del dispositivo y carga las librerías necesarias para su manejo. Devuelve el puntero al objeto que maneja el dispositivo USB.
- $[allTs_Out, allAddr_Out] = sendCommand(ok, CMD, selInput, allTs_In, allAddr_In)$
Esta función es la que se utiliza para enviar el comando deseado a la herramienta. El significado de cada uno de los parámetros es el siguiente:
 - ok : Puntero al objeto que maneja el dispositivo USB
 - CMD : Comando a enviar a la herramienta. Sus valores pueden ser: *Merger, Monitor, Stop_Monitor, Log, Stop_Log, Download_Log, Load_Data, Play, Stop_Play, Stop_General*

- *selInput*: Selecciona la entrada deseada. Los valores permitidos son: *Merger_Out*, para seleccionar la salida del módulo *merger*, *NodeBoard_Out*, para seleccionar la interfaz CAVIAR.
- *allTs_In*, *allAddr_In*: (Opcionales) Pareja de vectores que indican los eventos AER en pareja de marca de tiempo y dirección. Son utilizados en el comando *Load_Data* para cargar datos en la memoria DDR2 de la herramienta.
- *allTs_Out*, *allAddr_Out*: (Opcionales) Similar a los anteriores pero es este caso son utilizados en el comando *Download_Log* para descargar datos de la memoria DDR2 o en el comando *Monitor* para recuperar los eventos monitorizados por la herramienta.

Un ejemplo para la monitorización de la salida de dos sensores conectados a las interfaces de tipo ROME interfaz de la OKAERTool sería la siguiente lista de comandos:

```
ok = initializeOpalKellyFPGA();
[allTs_Out, allAddr_Out] = sendCommand(ok, 'Monitor', 'Merger_Out');
```