(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 697 693**

(21) Número de solicitud: 201730963

(51) Int. CI.:

G06N 3/00 (2006.01)
G06F 17/16 (2006.01)

(12) SOLICITUD DE PATENTE A1

(22) Fecha de presentación:

24.07.2017

(43) Fecha de publicación de la solicitud:

25.01.2019

(71) Solicitantes:

UNIVERSIDAD DEL PAÍS VASCO / EUSKAL HERRIKO UNIBERTSITATEA (100.0%)
Barrio Sarriena, S/N
48940 Leioa (Bizkaia) ES

(72) Inventor/es:

MARTÍNEZ CORRAL, Unai y
BASTERRECHEA OYARZABAL, Koldobika

(74) Agente/Representante:

VALLEJO LÓPEZ, Juan Pedro

(54) Título: **NÚCLEO IP, ARQUITECTURA QUE COMPRENDE UN NÚCLEO IP Y PROCEDIMIENTO DE DISEÑO DE UN NÚCLEO IP**

(57) Resumen:

Núcleo IP, arquitectura que comprende un núcleo IP y procedimiento de diseño de un núcleo IP.

Un núcleo IP configurable y programable de procesamiento para la computación de una pluralidad de productos matriciales, en el que tanto los datos a procesar como los resultados obtenidos se transfieren en serie, que comprende: El núcleo IP comprende: un bloque de entrada de datos para proporcionar un conjunto de vectores que representan una primera y una segunda matriz cuyo producto se quiere computar, donde dicho bloque de entrada de datos comprende: un primer sub-bloque y un segundo sub-bloque; un bloque de memoria que comprende N elementos de memoria asociados a una salida respectiva de dicho segundo sub-bloque del bloque de entrada de datos; un bloque multiplicador matriz-vector en coma fija para implementar una operación de multiplicación-acumulación; un bloque que comprende al menos una función de activación configurada para ser aplicada a la salida de dicho bloque multiplicador matriz- vector en coma fija; un bloque para almacenar las salidas de la al menos una función de activación y para leer las salidas de dichos componentes de almacenamiento; un bloque FIFO y un bloque de salida de datos que comprende un contador de fila y un contador de columna. Sistema en chip que comprende al menos un núcleo IP.FPGA que comprende al menos un núcleo IP. Procedimiento de diseño de un núcleo IP.

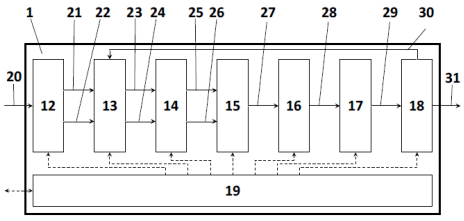

FIG. 2

ES 2 697 693 A1

DESCRIPCIÓN

NÚCLEO IP, ARQUITECTURA QUE COMPRENDE UN NÚCLEO IP Y PROCEDIMIENTO DE DISEÑO DE UN NÚCLEO IP

CAMPO DE LA INVENCIÓN

La presente invención pertenece al campo delos sistemas embebidos de procesamiento de datos. Más concretamente, la invención pertenece al campo de los sistemas de procesamiento de datos embebidos, tales como los sistemas de procesamiento para computación de redes neuronales, especialmente para redes neuronales de prealimenación (*feedforward*). La invención tiene especial aplicación en el campo del diseño de sistemas de aceleración del procesamiento de datos, tales como en *machine learning*, sobre plataformas embebidas, tales como Sistemas en el Chip (*SoC, System on Chip*) y FPGAs (*Field Programmable Gate Array*).

ANTECEDENTES DE LA INVENCIÓN

El aprendizaje en máquinas (del inglés *machine learning*) es un área tecnológica que está experimentando un enorme desarrollo en los últimos años, debido principalmente a tres factores: (1) Disponibilidad de enormes cantidades de datos (debido al desarrollo del Internet of Things o IoT, a avances en la tecnología de sensores y al uso generalizado de vídeo y audio digital, entre otros); (2) Gran desarrollo de hardware (tecnología microelectrónica) y consiguiente aumento de la capacidad de computación; y (3) Avances en los propios algoritmos de inteligencia computacional (en este sentido, la aparición del concepto de *Deep Learning* y su éxito de aplicación, sobre todo en el campo de la visión artificial, ha generado un gran interés por el uso de redes neuronales en aplicaciones industriales).

Entre las diferentes opciones de procesamiento para el aprendizaje de máquinas, las redes neuronales artificiales (RNAs) constituyen uno de los modelos de predictor (clasificador, regresor) más populares y que está recibiendo mayor atención por su amplia aplicabilidad. Sin embargo, las aplicaciones del procesamiento neuronal (procesamiento basado en RNAs) para *machine learning* requieren de una gran capacidad de computación que, debido a su arquitectura fundamentalmente paralela y masivamente interconectada, sólo puede ser satisfecha utilizando procesadores específicos de alto rendimiento y alta eficiencia. En el caso de los sistemas de

procesamiento embebidos (en contraposición con los sistemas de procesamiento masivos basados en la computación en la nube o *cloud computing* y grandes computadores con un elevado volumen y consumo), cuyas áreas de aplicabilidad están en constante crecimiento (sistemas autónomos, plataformas móviles, IoT, automóvil inteligente/autónomo, etc.), estas exigencias computacionales suponen un reto mayor debido a la necesidad de ser implantadas en hardware de pequeño tamaño, bajo consumo y bajo coste. Las FPGAs son, en este sentido, una de las plataformas con más potencial en este campo, ya que permiten aplicar las más avanzadas técnicas de diseño digital (paralelismo, segmentación, diseño específico con granularidad fina, tanto en lógica como en memoria) en la implementación de sistemas de procesamiento complejos en un solo chip, de forma que pueden obtenerse los más altos rendimientos en términos de capacidad de procesamiento por unidad de potencia consumida. Sin embargo, el diseño de este tipo de sistemas es complejo y laborioso, con ciclos de diseño relativamente largos, lo que hace necesaria la implicación de diseñadores expertos y alarga la llegada de los productos al mercado (*time to market*). En consecuencia, la tendencia actual es ofrecer a los diseñadores librerías de unidades prediseñadas, preferiblemente configurables a sus necesidades, en forma de bloques de propiedad intelectual (bloques IP o núcleos IP), a menudo configurables y escalables, de forma que puedan integrarse en sus diseños ajustándose a las necesidades de sus aplicaciones finales. El concepto de núcleo IP está íntimamente ligado al concepto de reusabilidad y a la utilización de las herramientas de CAD (EDA) en el diseño y síntesis de sistemas digitales. Idealmente, los núcleos IP son completamente portables, es decir, que se ha utilizado algún lenguaje estándar de descripción para su diseño (o un formato de netlist lógico) y que no lleva asociada ninguna información referente a la tecnología final de implementación. De esta manera, los diseñadores de sistemas digitales pueden hacer uso de estos núcleos IP, que se organizan en librerías, integrándolos directamente en sus diseños como simples cajas negras, a menudo configurables definiendo ciertos parámetros, que únicamente exponen los puertos de entrada/salida para su interconexión con el resto del sistema, frecuentemente mediante buses.

A modo de ejemplo, dentro de las redes neuronales, las redes neuronales de convolución (CNN, del inglés *Convolutional Neuron Network*) y las redes neuronales profundas (DNN, del inglés *Deep Neural Network*) representan un modelo computacional que está ganando popularidad debido a su potencialidad para resolver problemas de la interfaz humano-computadora, tales como la interpretación de imágenes. Esto es debido a que estas redes pueden alcanzar gran precisión mediante la emulación del comportamiento

del nervio óptico. El núcleo del modelo es un algoritmo que recibe como entrada un amplio conjunto de datos (por ejemplo, píxeles de una imagen) y aplica a esos datos un conjunto de transformaciones (convoluciones en el caso de las CNN) de acuerdo con unas funciones predefinidas. Los datos transformados pueden llevarse a continuación a una red neuronal para detectar patrones. Como en el caso general de las RNAs, debido al patrón computacional específico de las CNN y DNN, los procesadores de propósito general no son eficientes en implementaciones basadas en CNN o en DNN.

La solicitud de patente US 2015/0170021 A1 describe un dispositivo procesador que incluye un núcleo de procesador y un número de módulos de cálculo, siendo cada uno de estos configurable para realizar operaciones de un sistema CNN. Un primer conjunto de módulos de cálculo del dispositivo procesador está configurado para realizar operaciones de convolución, un segundo conjunto de módulos de cálculo se reconfigura para realizar operaciones de promediado y un tercer conjunto de módulos de cálculo se reconfigura para realizar operaciones de producto escalar. Sin embargo, se trata de un procesador de precisión fija, seleccionable entre 8 bits y 16 bits, y no se puede indicar un tamaño de palabra diferente para cada etapa del circuito. En consecuencia, no es posible optimizar la precisión de computación del procesador, con el consiguiente impacto negativo en recursos consumidos (área ocupada).

Tianshi Chen et. Al. han propuesto un acelerador para *machine learning* (redes neuronales) formado por un búfer de entrada para neuronas de entrada, un búfer de salida para neuronas de salida y un tercer búfer para pesos sinápticos, conectados a un bloque computacional configurado para realizar computaciones sinápticas y neuronales. El acelerador tiene también una lógica de control (“DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning”, SIGPLAN Not., vol. 49, nº 4, pp. 269-284, Feb. 2014).En esta propuesta, los elementos de memoria dispuestos a la entrada/salida del núcleo son buffers con DMA (un buffer para los datos de entrada, otro buffer para los pesos y un tercer buffer para almacenar los resultados) conectados a una interfaz de memoria común, por lo que se necesitan direcciones absolutas para acceder a la memoria externa. Es decir, no existe jerarquía de memoria. En lo que respecta a los recursos aritméticos, están dispuestos para una máxima paralelización, utilizando uno o varios árboles de sumadores (en inglés, *adder-tree*). Asimismo, el formato numérico utilizado es fijo e invariable (16 bits en coma fija). Además, atendiendo a la lógica de control, se entiende que los tamaños de las diferentes capas que componen el modelo deben cumplir ciertas restricciones para que la elección de los parámetros de subdivisión arroje resultados enteros.

A su vez, en la propuesta de J. Qiu et. al (“Going deeper with embedded fpga platform for convolutional neural network”, Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, ser. FPGA’16, Monterrey, California, USA: ACM, 2016, pp 26 {35, isbn: 978-1-4503-3856-1. doi: 10.1145/2847263.2847265.), el diseño de la unidad aritmética es muy similar al de la propuesta de Tianshi Chen et. al. en “DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning”, con la diferencia de que el buffer para cargar los pesos es de tipo FIFO.

A su vez, Chen Zhang et al. han propuesto un diseño de acelerador basado en FPGA para Redes Neuronales de Convolución profundas, en el que tratan de optimizar tanto los recursos lógicos como el ancho de banda de memoria ("Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," FPGA’15, February 22-24, 2015, Monterey, California, USA, ACM 978-1-4503-3315-3/15/02). El acelerador propuesto es capaz de ejecutar trabajos de aceleración a lo largo de diferentes capas sin necesidad de reprogramar la FPGA. Esta divulgación se centra en la reordenación algorítmica y en la elección óptima de parámetros, teniendo en consideración las características de la plataforma objetivo. Para la descripción del sistema, esta propuesta utiliza la herramienta de síntesis automática de alto nivel HLS (Vivado). El uso de una herramienta de este tipo, comparado con un diseño realizado a más bajo nivel (RTL), conlleva varias limitaciones en lo referente a la optimización de la comunicación de los datos (alternativas al uso de memorias compartidas, lecturas no secuenciales en las FIFOs etc.) y, en particular, a la organización y los accesos a memoria (gestión de espacios de direccionamiento independientes, asignación dinámica de memoria etc.). Asimismo, el formato numérico utilizado es de 32 bits (en coma fija o flotante), sin opción a su configuración.

Por otra parte, M. Motamedi et al. profundizan(“Design space exploration of fpga-based deep convolutional neural networks”, in 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC), Jan. 2016, pp. 575 {580.Doi: 10.1109/ASPDAC.2016.7428073)en las posibilidades de máxima paralelización y aprovechamiento de la localidad, utilizando módulos denominados Parallel Convolution Engines (PCD), compuestos por múltiples multiplicadores y un adder-tree. Este diseño solo es provechoso en la computación de capas de convolución.

Por último, H. Li et. al., proponen (“A high performance fpga-based accelerator for large-scale convolutional neural networks”, 2016, 26th International Conference on Field Programmable Logic and Applications (FLP), Aug. 2016, pp. 1 {9. doi: 10.1109/FLP.

2016.7577308) la utilización de una instancia del módulo aritmético para cada capa del modelo de la red neuronal, introduciendo buffers dobles entre cada etapa. En lo que respecta al diseño de cada módulo aritmético, se propone un array sistólico de MACCs, de longitud igual al filtro a aplicar. Los pesos se cargan mediante multiplexores y los datos a través de un registro de desplazamiento cuyo tamaño se limita a la longitud del filtro de convolución. Los datos no se utilizan hasta que el registro no está completamente lleno. También utilizan adder-trees para acumular resultados parciales computados en paralelo.

DESCRIPCIÓN DE LA INVENCIÓN

La presente invención proporciona un módulo de procesamiento que resuelve los inconvenientes de propuestas anteriores.

En el contexto de la presente divulgación, los términos “procesador”, “procesador neuronal”, “núcleo de procesamiento”, “núcleo IP” (del inglés IP core), “bloque IP” (del inglés IP block) y módulo IP (del inglés IP module) son intercambiables. La presente divulgación describe un núcleo IP de un procesador neuronal totalmente configurable por el usuario final, es decir, por ejemplo por el diseñador de un Sistema en Chip (*System on Chip*, SoC). Este núcleo IP permite la aceleración de computación y es especialmente aplicable a la aceleración de computación en algoritmos de *machine learning*. Entre las características del procesador de la invención, se pueden destacar que:

-El procesador neuronal puede adecuarse automáticamente, durante el proceso de síntesis de la red neuronal que se vaya a implementar, a los recursos disponibles en el dispositivo objetivo seleccionado (por ejemplo, FPGA) mediante una técnica de plegado (en inglés, *folding*) y reutilización de las capas neuronales. A partir de las fuentes de descripción, por ejemplo en VHDL, un sintetizador (CAD) genera un netlist adecuado para la plataforma o tecnología objetivo (dependiendo de si es una FPGA o un ASIC). Es decir, se ha realizado una descripción totalmente parametrizada del procesador neuronal (núcleo IP) en la que el usuario final puede indicar, entre otras cosas, cuántos recursos quiere/puede utilizar para la implementación final del procesador de forma que el sistema realiza los ajustes necesarios para plegar la arquitectura de la red de forma que el procesador sintetizado finalmente sea más serie (lo que implica mayor reutilización de menor cantidad de recursos hardware, y por tanto, un poco más lento), o más paralelo (lo que implica menor reutilización, o ninguna, en el caso más extremo, de una mayor cantidad de recursos hardware, y por tanto, más rápido). Esta característica permite

empotrar el procesador (o núcleo IP) tanto en FPGAs de bajo coste (pequeño tamaño y consumo y reducidos recursos lógicos) como en FPGAs de alto rendimiento. Más aún, una vez sintetizado cierto tamaño de red, la utilización del número de neuronas en cada capa de red puede seleccionarse en tiempo de ejecución, haciendo innecesario el uso de técnicas de reconfiguración dinámica. Esto se consigue mediante la activación/desactivación de una señal de habilitación presente en cada elemento de procesamiento (DSP + función de activación). Es decir, en contraposición con la adecuación automática del número de recursos hardware (DSPs, etc.) que se van a sintetizar, es posible activar/desactivar neuronas que ya han sido implementadas ("programación" vs. "configuración" en fase de síntesis). Para realizar esta programación, el usuario no necesita gestionar todas las señales de habilitación de forma individual, sino que se genera un registro en el módulo de control que permite indicar cuántas neuronas de las sintetizadas se van a utilizar o activar en el último fold o plegado (ya que en todos los anteriores, si los hay, se utilizan todas las neuronas físicas). El sistema detecta cuándo se ha modificado el valor de ese registro, y se utiliza un contador junto con un registro de desplazamiento para generar secuencialmente todas las señales de (des)habilitación. En una posible implementación, en la que en la mayoría de ejecuciones el número de neuronas físicas utilizadas es mayor que el número de neuronas deshabilitadas, inicialmente todas están habilitadas. Así, la latencia de configuración es igual al número de neuronas deshabilitadas. La latencia máxima, por tanto, es uno menos que el número de neuronas físicas (cuando sólo se utiliza una). En otra posible implementación, se puede utilizar otro patrón predefinido: que por ejemplo la mitad estén habilitadas y la otra mitad no. De esta forma, la latencia máxima es la mitad, pero se incrementa la latencia media en elecciones de parámetros óptimas (que el número de neuronas del modelo de la capa sea múltiplo del número de neuronas físicas).

-El procesador (o núcleo IP) utiliza direcciones virtuales, de forma que sus componentes internos trabajan en términos de matrices, columnas y filas, lo que facilita el desarrollo de aplicaciones basadas en algoritmos que integren operaciones de álgebra matricial. El procesador neuronal (núcleo IP) incluye módulos "puente" para gestionar el propio núcleo IP directamente desde puertos externos, tales como puertos AXI, que son el estándar en algunos productos (chips, ASICs), como los de Xilinx y ARM. Además, puesto que la gestión de grandes volúmenes de datos (es decir, grandes matrices) es crítica en los sistemas actuales, el procesador se ha provisto de un bloque de interconexión configurable y programable (MAI, que se describe más adelante) específicamente diseñado que permite gestionar bloques de memoria tanto internos

como externos.

Como resultado, el diseñador de una arquitectura SoC puede integrar múltiples bloques de computación y almacenamiento, y realizar de forma transparente pruebas de rendimiento asignando dinámicamente la memoria, para obtener la solución más eficiente. Esto se consigue por medio de un conjunto de tablas y microbloques en la MAI que ofrecen una amplia variedad y granularidad a la hora de asignar no sólo las matrices, sino cada fila/columna.

Además, el módulo de procesamiento de la invención se basa en la reutilización de los bloques de multiplicación-acumulación (bloques MACC), que en ocasiones en este texto se denomina como plegado de capas. En este sentido, los parámetros del modelo de la capa (a veces llamados 'pesos' o 'ganancias' de las interconexiones neuronales), que se ajustan en una fase previa de entrenamiento o aprendizaje de la red, se cargan en los scratchpads a través del mismo puerto que los datos de entrada (vectores de entrada de la red). En implementaciones de la invención, el módulo de control puede mantener el scratchpad del vector de entrada en standby hasta que todos los parámetros hayan sido cargados, de forma que una vez iniciada la computación, no sea interrumpida por la carga de parámetros. Los resultados finales pueden simplemente guardarse en memoria a través del puerto de salida, o realimentarse al scratchpad de entrada. La elección depende del modelo concreto de capa que se esté implementando. Por ejemplo, en las soluciones más compactas donde sea aceptable un tiempo de ejecución más lento, un vector puede realimentarse varias veces para procesar el efecto de múltiples capas utilizando únicamente memoria local.

Aunque el diseño del módulo de procesamiento es independiente de la tecnología objetivo en la que se vaya a integrar, en realizaciones de la invención, el alto grado de parametrización de la descripción del diseño del núcleo IP y su enorme escalabilidad, hacen que el núcleo IP pueda ser integrado de varias maneras en distintas tecnologías. Por un lado, las ya mencionadas FPGAs y PSoCs, que son dispositivos “prefabricados” con una alta configurabilidad, de forma que el núcleo IP está diseñado para adecuarse de forma óptima a las características de la arquitectura de los sistemas-en-un-chip (SoC) sobre FPGA, y más concretamente a las características del bus AXI. Así, el núcleo IP puede ser incrustado tanto en FPGAs de pequeño tamaño y bajo coste como en FPGAs de mayor tamaño y rendimiento. Por otro lado, el código de descripción del procesador, descrito en el lenguaje estándar de descripción hardware VHDL (acrónimo que resulta de combinar VHSIC (*Very High Speed Integrated Circuit*) y HDL (*Hardware Description*

Language)), proporciona gran portabilidad, de forma que el módulo de procesamiento es no solo integrable en arquitecturas PSoCs o FPGAs (ya sea integrándolo con procesadores blandos (soft processors) o en los denominados PSoCs (Programmable SoC), que contienen un procesador duro (hard processor), tal como el Cortex-A9 de ARM integrado en un dispositivo Zynq de Xilinx), sino que es también integrable en un ASIC con arquitectura SoC junto con otros procesadores y módulos de aceleración. Es decir, aunque la tecnología objetivo preferente para la que ha sido diseñado el módulo de procesamiento (o procesador) de la invención es una FPGA, ya que el código de descripción del procesador se ha desarrollado con el objetivo de realizar un uso optimizado de los recursos propios de las FPGAs (tales como los bloques de memoria (BRAM), unidades aritméticas (bloques DSPs) y gestores/sintetizadores/divisores de reloj (Mixed-Mode Clock Manager, MCMM)),el módulo de procesamiento de la invención puede también destinarse a ASIC para su integración en SoC. Nótese que recientemente se están comercializando productos en el mercado que integran bloques IP de FPGAs en ASICs (es decir, bloques IP configurables y adaptables a distintas tecnologías de fabricación), denominados comúnmente como eFPGAs, para que los fabricantes de SoCs en ASIC integren dichos bloques IP en sus chips de forma que éstos dispongan de una zona de lógica reconfigurable al estilo de las FPGAs (es una FPGA a medida integrada como parte de un ASIC).

Una diferencia significativa del procesador de la invención, con respecto a los divulgados en el estado de la técnica, es que el procesador se ha diseñado para optimizar su funcionamiento en aplicaciones con entrada de datos de tipo streaming. Por ejemplo, el procesador es especialmente adecuado para el procesamiento de imágenes hiperespectrales. En consecuencia, no se ha buscado la paralelización máxima del procesamiento, cuestión que condiciona al uso de adder-trees y el broadcasting de parámetros a múltiples instancias al mismo tiempo, sino que el bucle (*loop*) más interno es secuencial. Este hecho tiene impacto tanto en el tamaño del sistema, como en el tiempo de ejecución y frecuencia máxima de operación. En comparación con los diseños conocidos, el presente procesador da como resultado soluciones que ocupan menos área (menos recursos en una FPGA) y requiere más ciclos de reloj de computación, pero a su vez permite trabajar a frecuencias mayores, con lo que se compensa en cierta medida la mayor latencia en ciclos. Esto se debe a que el objetivo buscado es adaptarse a las limitaciones impuestas por el streaming y aprovecharlo para sacar el máximo partido a las FPGAs de tamaño reducido. Además, la presente propuesta evita la imposición de relaciones inviolables entre parámetros, lo cual redunda en una mayor flexibilidad y

escalabilidad.

Por otra parte, puesto que el presente núcleo IP está especialmente diseñado para sistemas con recursos limitados (dispositivos pequeños), el sistema de control del núcleo IP no impone restricciones en las relaciones entre los parámetros de configuración. Es común, por ejemplo, que las redes CNN más extendidas (y complejas) utilicen potencias de dos para establecer los tamaños de las capas de la red y de los conjuntos de datos a procesar. Sin embargo, en aplicaciones con SLFN, se ha detectado la necesidad de poder escoger tamaños de capa de hasta 2k con una granularidad mayor. Así, las herramientas de autoconfiguración generan una solución que garantiza el menor tiempo de ejecución de la arquitectura para cualquier tamaño de red, sin introducir datos de padding (relleno) para su control. Gracias a que las señales de habilitación se activan secuencialmente, se minimiza la complejidad adicional en los elementos de control de la arquitectura. A esto contribuye el hecho de que el bucle más interno (es decir, el producto vectorial) se procese de forma secuencial, reduciendo con ello una dimensión a gestionar de forma no lineal. Al mismo tiempo, debe considerarse que la omisión de los DMA de la arquitectura y la utilización de formatos de dirección optimizados ofrece un balance para compensar el uso de recursos.

Con respecto a los registros de desplazamiento para la entrada de datos, una diferencia significativa del procesador de la invención, con respecto a los divulgados en el estado de la técnica, es que se utiliza un número elevado de multiplicadores y al computar en modo “ola” los datos empiezan a utilizarse desde que llega el primero al primer multiplicador, sin esperar a que los registros estén llenos.

Otra ventaja destacable con respecto a otras propuestas es que la precisión de computación en el procesador de la invención es ajustable, es decir, configurable para cada conjunto de datos y, opcionalmente, seleccionable en tiempo de ejecución. En las propuestas convencionales la precisión es fija debido a que se presupone que la FPGA se va a reconfigurar para cada modelo de red neuronal (no hay configurabilidad en tiempo de ejecución).

Por otra parte, varios parámetros del presente procesador son configurables en tiempo de ejecución, es decir, tras la síntesis. Algunos de estos parámetros son el tamaño del bloque (*chunk*) de datos a ser procesado, número de entradas, número de neuronas en las capas ocultas a procesar, número de salidas y uso (o no) de las funciones de activación en cada capa. Esto hace que el procesador, una vez sintetizado e implementado, sea más flexible y pueda adaptarse a distintos modelos de red sin

necesidad de reconfigurar el hardware.

Además, la presente solución utiliza un módulo de internconexión (MAI) entre la memoria externa y el núcleo IP, específicamente diseñado para conectar de forma eficiente y sencilla varios núcleos IP en una arquitectura heterogénea de tipo SoC. En realizaciones de la invención, este módulo (MAI) está basado en Wishbone B4, asegurando su total portabilidad e independencia de la tecnología objetivo.

Otro aspecto destacable de la presente solución es que el núcleo IP ha sido totalmente descrito utilizando el lenguaje estándar VHDL, lo que lo hace agnóstico a las herramientas de diseño/síntesis y totalmente portable desde el punto de vista tecnológico, además de permitir un control total sobre todos y cada uno de los aspectos del diseño. Más aún, se han escrito un conjunto de paquetes en este lenguaje que han permitido la práctica total parametrización del diseño, por lo que en realidad se ha desarrollado una completa herramienta de diseño y configuración del núcleo IP que permite al diseñador de sistemas su fácil uso e integración. El conjunto de scripts e interfaces de usuario que componen estas herramientas de parametrización automática son multiplataforma (por ejemplo, Windows, Linux, mac) y generan VHDL estándar, por lo que puede utilizarse tanto para síntesis en FPGA como para semi-custom ASIC.

Por último, el módulo de procesamiento descrito en la presente invención cuenta con gestión de memoria integrada.

En un primer aspecto de la invención, se proporciona un núcleo IP configurable y programable de procesamiento para la computación de una pluralidad de productos matriciales, en el que tanto los datos a procesar como los resultados obtenidos se transfieren en serie, que comprende: un bloque de entrada de datos configurado para proporcionar, a partir de unos datos de entrada, un conjunto de vectores que representan una primera y una segunda matriz cuyo producto se quiere computar, utilizando un formato de dirección virtual compuesto por punteros a matriz, fila y columna, donde dicho bloque de entrada de datos comprende: un primer sub-bloque configurado para obtener un puntero de fila (p_{ROW}) y un puntero de columna (p_{COL}); y un segundo sub-bloque que comprende N componentes, donde N es un número natural > 1, cada uno de los cuales comprende dos contadores encadenados correspondientes al número de vectores a transferir y a la longitud de dichos vectores, donde cada componente utiliza direcciones locales; un bloque de memoria que comprende N elementos de memoria, estando cada uno de dichos elementos de memoria asociado a una salida respectiva de dicho segundo sub-bloque del bloque de entrada de datos; un bloque multiplicador matriz-vector en

coma fija configurado para implementar una operación de multiplicación-acumulación para multiplicar una matriz por múltiples vectores recibidos en serie de forma continua, donde dicho bloque multiplicador matriz-vector en coma fija comprende un conjunto de sub-bloques, donde cada uno de dichos sub-bloques comprende una pluralidad de módulos aritméticos; un bloque que comprende al menos una función de activación configurada para ser aplicada a la salida de dicho bloque multiplicador matriz-vector en coma fija; un bloque para almacenar en componentes de almacenamiento las salidas de la al menos una función de activación y para leer las salidas de dichos componentes de almacenamiento; y un bloque de salida de datos que utiliza un formato de dirección virtual compuesto por punteros a matriz, fila y columna, que comprende un contador de fila y un contador de columna.

En realizaciones de la invención, el primer componente de dicho segundo sub-bloque está configurado para proporcionar un número de vectores igual al número de productos matriz-vector consecutivos que se desean computar.

En realizaciones de la invención, los componentes segundo a último de dicho segundo sub-bloque están configurados para proporcionar un número de vectores igual al número de pasadas que se deben realizan con el DSP correspondiente.

En realizaciones de la invención, el bloque multiplicador matriz-vector en coma fija está basado en un array sistólico lineal con carga de parámetros en paralelo y ejecución tipo ola.

En realizaciones de la invención, dichos N elementos de memoria comprendidos en dicho bloque de memoria son N bloques BRAM.

En realizaciones de la invención, cada sub-bloque o grupo de dicho bloque multiplicador matriz-vector en coma fija comprende un multiplexor a su salida.

En realizaciones de la invención, cada sub-bloque o grupo de dicho bloque multiplicador matriz-vector en coma fija comprende a su salida tantos registros de desplazamiento como módulos aritméticos tiene cada sub-bloque.

En realizaciones de la invención, dichos módulos aritméticos operando en paralelo generan, cada z ciclos, tantos datos como número de módulos aritméticos haya, donde z es la longitud del vector.

En realizaciones de la invención, la ejecución en paralelo de dichos módulos aritméticos se controla mediante una máquina de estados que toma como referencia sólo el primer módulo aritmético. En este caso, la máquina de estados puede utilizar tres contadores de

apoyo: longitud del vector, número de repeticiones y latencia del módulo aritmético.

En realizaciones de la invención, dicho bloque multiplicador matriz-vector en coma fija representa al menos una capa oculta de una red neuronal artificial.

En realizaciones de la invención, el núcleo IP comprende medios para realimentar la salida de dicho bloque FIFO para efectuar al menos dos operaciones matriz-vector consecutivas con un filtrado intermedio, de forma que con una sola instancia del bloque multiplicador matriz-vector en coma fija se procesa tanto la al menos una capa oculta como la capa de salida de la red neuronal.

En realizaciones de la invención, se utilizan M módulos aritméticos para efectuar h multiplicaciones vectoriales, donde h es el número de filas de la matriz en el producto matriz-vector, por lo que si h >M, se requiere más de una pasada, iteración o repetición para procesar cada vector de entrada.

En realizaciones de la invención, el núcleo IP comprende además un núcleo de interconexión configurado para facilitar la integración del núcleo IP en un sistema heterogéneo con uno o varios módulos coprocesadores, estando dicho núcleo de interconexión configurado para gestionar bloques de memoria internos y externos al núcleo IP.

En realizaciones de la invención, todas las operaciones se realizan en coma fija de precisión ajustable, estando configurado para definir el tamaño de palabra utilizado en cada una de las matrices que intervienen en cada ejecución.

En un segundo aspecto de la invención, se proporciona una arquitectura SoC (*System on a Chip*) que incorpora una módulo de procesamiento como el descrito anteriormente. En realizaciones de la invención, el módulo de procesamiento se integra en una FPGA. Es decir, se proporciona un sistema en chip (SoC) que comprende al menos un núcleo IP como el descrito anteriormente.

En otro aspecto de la invención, se proporciona una FPGA que comprende al menos un núcleo IP como el descrito anteriormente.

En otro aspecto de la invención, se proporciona un procedimiento de diseño de un núcleo IP como el descrito anteriormente, adecuado para una tecnología objetivo, que comprende: generar un netlist que comprende una descripción parametrizada del núcleo IP adecuado para dicha tecnología objetivo; sintetizar una red neuronal que se desee implementar, adecuando el núcleo IP a los recursos disponibles en dicha tecnología objetivo, donde dicha adecuación se realiza mediante una técnica de plegado y

reutilización de capas neuronales; una vez sintetizado cierto tamaño de red neuronal, seleccionar en tiempo de ejecución un número de neuronas que se desea utilizar en cada capa de red.

El procesador neuronal de la invención es aplicable en cualquier aplicación de *machine learning* basada en RNAs en la que convenga realizar un procesamiento acelerado de grandes volúmenes de datos y, más particularmente, en sistemas embebidos autónomos con requerimientos de reducido tamaño y peso y gran integrabilidad. En consecuencia, la aplicabilidad del procesador neuronal de la invención es amplísima: entre otras, clasificación de objetos en sistemas de visión embebida (detección de tumores, detección e identificación de objetos, detección de peatones, coches autónomos, guiado de drones, detección de objetivos, procesamiento de imágenes hiperespectrales, etc.), ya que las aplicaciones de *machine learning* atraviesan todos los mercados verticales, desde el militar/aeroespacial, pasando por la automoción, el industrial, el de instrumentación médica, hasta los grandes centros de procesamiento de datos (internet, computación en la nube, IoT). El procesador neuronal de la invención puede usarse también en aplicaciones de *deep learning*, por ejemplo para visión embebida, ya que el procesador puede configurarse fácilmente para trabajar como una red neuronal con múltiples capas de procesamiento, incluso con capas de convolución o capas recurrentes.

En suma, entre las ventajas del núcleo IP propuesto, puede destacarse que se ha concebido y realizado para que sea de fácil integración como un núcleo IP en el diseño de arquitecturas SoC junto con otros módulos de procesamiento (incluidos, claro está, los microprocesadores). Así, es destacable que un diseñador de SoC que quiera utilizarlo, únicamente debe ajustar una serie de parámetros de configuración a nivel alto (aspectos del modelo de red que quiere implementar así como restricciones en los recursos hardware disponibles para su implementación) y el código se autoconfigura para generar una estructura de procesamiento adecuada a estos requerimientos (ajustes de nivel bajo). Más aún, el diseño permite que, una vez implementado el procesador, algunas de sus funcionalidades sean programables en tiempo de ejecución.

Ventajas y características adicionales de la invención serán evidentes a partir de la descripción en detalle que sigue y se señalarán en particular en las reivindicaciones adjuntas.

BREVE DESCRIPCIÓN DE LAS FIGURAS

Para complementar la descripción y con objeto de ayudar a una mejor comprensión de las características de la invención, de acuerdo con un ejemplo de realización práctica de la misma, se acompaña como parte integrante de la descripción, un juego de figuras en el que con carácter ilustrativo y no limitativo, se ha representado lo siguiente:

La figura 1 representa una red neuronal artificial (RNA) convencional. Concretamente, se ha ilustrado una arquitectura típica de una red neuronal artificial de tipo Single Hidden Layer Feedforward Network (SLFN), así como la representación matricial de las fases de inferencia y entrenamiento del modelo directo Extreme Learning Machine (ELM).

La figura 2 ilustra un diagrama de bloques de un núcleo IP de acuerdo con una posible realización de la invención.

La figura 3 ilustra un diagrama de bloques de un posible sistema en el que puede integrarse el núcleo IP de la figura 2.

La figura 4 ilustra un diagrama de bloques de unos módulos puente para conectar módulos de interconexión basados en Wishbone B4 con módulos de interconexión basados en AMBA, de acuerdo con una posible realización de la invención.

La figura 5 ilustra una posible implementación de varios módulos del diagrama de bloques de la figura 2, de acuerdo con la presente invención.

La figura 6 representa en detalle las modificaciones de las interfaces FIFO de lectura correspondientes a la matriz de entrada y a la matriz de pesos, de acuerdo con una posible realización de la invención. Es decir, se muestran gráficamente los patrones de acceso para el procesamiento de tres productos matriz-vector consecutivos en un problema de cuatro pasadas (olas).

La figura 7 ilustra un ejemplo de implementación de un núcleo IP de acuerdo con una posible realización de la invención.

La figura 8 ilustra un diagrama de bloques de un núcleo IP de acuerdo con otraposible realización de la invención.

La figura 9 muestra un esquema del componente utilizado para obtener punteros de fila y columna en el sub-bloque intercon_si, para decodificar la dirección local en base a la posición y los índices dados, de acuerdo con una posible implementación de la invención.

La figura 10 muestra tres posibles dominios de reloj utilizados por el núcleo IP de acuerdo con realizaciones de la invención.

La figura 11 ilustra un ejemplo de combinación de múltiples instancias (*stack*) de un

coprocesador neuronal de acuerdo con realizaciones de la invención, para reducir la latencia en aplicaciones con recursos lógicos suficientes. Se observa cómo se pueden instanciar varias copias del núcleo IP para computar múltiples capas al mismo tiempo.

La figura 12 ilustra dos DSPs modificados, de acuerdo con posibles realizaciones de la invención.

La figura 13 muestra posibles implementaciones del módulo 15 de la figura 2, basado en DSPs con los correspondientes módulos de saturación, por tratarse de aritmética en coma fija. Se muestran dos variantes para gestionar las salidas: mediante registros de desplazamiento y mediante multiplexores.

La figura 14 representa de forma esquemática diferentes variantes de interconexión para adecuar el ancho de banda de entrada a los requerimientos de la aplicación.

La figura 15 representa la ejecución con un patrón de tipo ola y su posterior serialización a la salida, que es una característica relevante del funcionamiento del núcleo IP de la invención. Se trata de un cronograma, a modo de ejemplo, de la ejecución del núcleo IP, donde se observan dos características de diseño: i) los resultados se generan en un patrón tipo ola; y ii) la derivación automática de la mayoría de parámetros internos de la arquitectura minimiza el impacto de los cuellos de botella como el mostrado en este ejemplo.

La figura 16 ilustra un ejemplo de combinación de múltiples instancias del módulo 15 de la figura 2 para reducir la latencia.

DESCRIPCIÓN DE UN MODO DE REALIZACIÓN DE LA INVENCIÓN

La figura 1 representa una red neuronal artificial (RNA) convencional que representa un modelo computacional para cuyo procesamiento es necesaria una gran capacidad de computación que solo puede ser satisfecha usando procesadores específicos de alto rendimiento y alta eficiencia. El procesador o núcleo IP de la presente divulgación está especialmente diseñado para computar modelos computacionales complejos, tales como, pero de forma no limitativa, el representado en la figura 1. En la parte superior de la figura 1 se muestra la arquitectura típica de una red neuronal artificial de tipo Single Hidden Layer Feedforward Network (SLFN). Se trata de una red de tipo 'shallow', lo que quiere decir que no hay muchas capas ocultas (en este caso sólo hay una, referenciada en la arquitectura ilustrada como “capa oculta”), en contraposición con las redes neuronales profundas (DNN), que poseen muchas capas ocultas. En todo caso, con el diseño de la

presente divulgación pueden implementarse redes con cualquier número de capas. Nótese que cada una de las conexiones entre la capa de entrada y la capa oculta tiene asociado un valor, representado como $w_{h,f}$, mientras que cada una de las conexiones entre la capa oculta y la capa de salida tiene asociado un valor, representado como$g_{c,h}$. Estos valores, denominados pesos o ganancias, representan una multiplicación y son los parámetros principales de la red. La red está además definida a través del número de nodos en cada capa. En el ejemplo ilustrado, la capa de entrada tiene 4+1 nodos (4 nodos de entrada + valor de bias (el valor de bias es realmente un parámetro de la capa oculta), la capa oculta tiene 6 nodos y la capa de salida tiene 3 nodos. Un experto entenderá que el número de nodos por capa puede variar de una red a otra. Además de los pesos y el número de nodos en cada una de las capas, a una SLFN la definen el tipo de funciones de activación utilizadas en las neuronas artificiales que se describen a continuación. En el contexto de la presente divulgación, una SLFN está compuesta por dos capas, puesto que no se realiza ninguna operación aritmética en la indicada como capa de entrada. No obstante, se utiliza también el término capa de forma genérica.

En la capa oculta, cada nodo es una neurona artificial. Una neurona artificial está formada por la suma de todas sus entradas y la aplicación de una transformación no lineal al resultado, denominada 'función de activación'. Hay múltiples funciones matemáticas que pueden utilizarse como función de activación para la computación de la activación. A modo de ejemplo, pero de forma no limitativa, citamos sigmoide logística (sig), tangente hiperbólica (tanh), base radial (RBF), lineal-rectificada (ReLu), entre otras. Nótese que a lo largo del presente texto se hace referencia a la función de activación como 'sigmoide', pero ello no ha de entenderse como una pérdida de generalización, ya que puede utilizarse cualquier función matemática adecuada para computar la activación. De hecho, el diseño contempla la síntesis de las funciones mencionadas. En algunas funciones de activación es recomendable utilizar uno o varios parámetros, denominados 'bias', para heterogeneizar las respuestas de los nodos, de forma que la proyección espacial implícita sea más rica en detalles. Esta característica se ilustra en la figura como un nodo adicional en la capa anterior con un valor constante. En el ejemplo ilustrado, la capa de salida no utiliza bias, por lo que no se ilustra en la capa oculta. En el núcleo IP de la presente divulgación, la utilización del bias puede activarse para cada capa modificando un bit correspondiente a la constante indicada. En la capa de salida, cada nodo es una suma de todas sus entradas, sin que se aplique ninguna función de activación. Por lo tanto, cada nodo de la capa de salida puede interpretarse como una neurona artificial con transformación nula a la salida. Así, del mismo modo que en la capa anterior, la capa de

salida puede ser seleccionable de entre un conjunto de funciones. En aplicaciones de clasificación, es habitual añadir una capa adicional de un solo nodo a la salida de la capa de salida para identificar la salida con valor máximo. El núcleo IP de la presente divulgación contempla esta posibilidad. Sin embargo, no es parte del modelo SLFN de referencia (figura 1).

La parte inferior de la figura 1 representa desde el punto de vista algorítmico el modelo de red neuronal de la parte superior de la figura. Este modelo se representa como dos productos matriciales sucesivos, con una transformación no lineal (la definida por la función de activación) aplicada al resultado intermedio. Concretamente: 'I' es una matriz compuesta por filas correspondientes a los 'v' vectores de entrada. La longitud 'f' de estos vectores 'v'es igual al número de nodos en la capa de entrada, sin contar los bias; 'W' es la matriz de pesos de entrada, donde cada fila h corresponde a los parámetros de cada nodo en la capa oculta; 'H' es el producto matricial I*W; 'T' es el resultado de la aplicación de la función no lineal a cada elemento de 'H'; 'G' es la matriz de pesos de salida, donde cada fila c corresponde a los parámetros de cada nodo en la capa de salida; 'R' es el producto matricial $T * G^T$. Por lo tanto, en adelante denominamos 'capa' a un producto matricial y una transformación no lineal opcional. En una versión básica de núcleo IP de la presente divulgación, el núcleo se centra en una única capa cada vez, por lo que la computación de la fase de inferencia de una SLFN implica dos ejecuciones sucesivas del núcleo. No obstante, se pueden instanciar varias copias del núcleo IP para computar múltiples capas al mismo tiempo, tal como se ilustra como ejemplo en la figura 11. Ambas soluciones permiten extrapolar su uso a redes con cualquier número de capas, ya que el diseño hardware es agnóstico al número de éstas; es decir, independiente de las operaciones anteriores o posteriores. Adicionalmente, en la figura se muestra la operación $G^T=T^+ *B$. Esta operación representa una fase de entrenamiento en caso de utilizar un método de entrenamiento concreto denominado Extreme Learning Machine (ELM). En la descripción del núcleo IP dela presente divulgación no se describe en detalle esta etapa, ya que la implementación de núcleo IP aquí descrita se centra en la etapa de inferencia o feedforward (no en la fase previa de entrenamiento). No obstante, se ilustra para exponer que el primer producto matricial y la transformación no lineal son operaciones compartidas en ambas fases. Por lo tanto, el diseño propuesto puede utilizarse junto con un resolvedor lineal para acelerar la etapa de entrenamiento. Además, la programabilidad en tiempo de ejecución permite utilizar la misma arquitectura en ambas fases, entrenamiento e inferencia, no siendo necesaria la síntesis e implementación de dos versiones de diferente tamaño. Alternativamente, como método

de entrenamiento puede utilizarse el método Random Vector Functional-Link (RVFL), muy similar al ilustrado. La diferencia radica en que la matriz 'I' se añade a la derecha de 'T', de forma que cada nodo de entrada se conecta directamente a cada nodo de salida con un peso determinado, además de la propia capa oculta. Esta modificación se mantiene en las fases de entrenamiento e inferencia. La implementación del núcleo IP propuesta no diferencia las ELM de las RVFL desde el punto de vista de la arquitectura hardware, ya que se ha diseñado para soportar ambos modelos cambiando únicamente los espacios de memoria utilizados. Asimismo, el usuario puede realizar transformaciones adicionales entre etapas para soportar otros modelos de red.

En suma, cualquier red neuronal que pueda expresarse como una secuencia de productos matriciales con transformaciones no lineales intermedias opcionales(es decir, cualquier modelo de red neuronal de tipo feedforward), se puede 'mapear' al diseño de núcleo IP de la presente divulgación, que se describe a continuación. Una vez representado el modelo de red en capas, la selección de parámetros del núcleo IP propuesto se basa en la elección de los valores máximos para cada una de las tres dimensiones implicadas en los productos: el número de filas de dos matrices y el número de columnas de ambas (que debe ser igual). Por ejemplo, cuando se procesa una SLFN, estas parejas de matrices son 'I,W' y 'T,G', y los parámetros a definir son: v, max(f,h) y max(h,c). Adicionalmente, se debe(n) escoger las(s) función(es) de activación que se desea sintetizar.

A continuación se explica cómo se adapta el diseño e implementación de núcleo IP de la presente divulgación de forma automática para computar capas de una red neuronal de diferente tamaño con un número fijo (probablemente inferior al número de neuronas de las distintas capas del modelo) de bloques DSP (en adelante simplemente DSP). O lo que es lo mismo, cómo se mapea el esquema convencional de una red neuronal, probablemente multicapa, a la implementación propuesta de núcleo IP (hardware): El modelo de la red se divide en capas, siendo cada una de éstas un producto matricial con una función de activación no lineal opcionalmente aplicada al resultado. Las capas de la red neuronal pueden implementarse como operaciones de multiplicación-acumulación (*multiply-accumulate operations*), es decir, computando el producto de dos entradas y añadiendo ese producto a un acumulador. La unidad hardware que realiza esta operación es un MACC o unidad MACC (acumulación de multiplicación o *Multiplier-ACCumulator*).En una FPGA o ASIC, cada DSP realiza un producto vectorial. Por lo tanto, cada DSP equivale a la primera fase de cómputo de una 'neurona física' (operación Multiply-Accumulate o MACC). A esto hay que añadirle una transformación no lineal (por

ejemplo, normalmente de tipo sigmoidal, aunque esto también es configurable), que se realiza en un módulo situado a continuación en el datapath; a esta transformación no lineal se le denomina función de activación de la neurona. Sin embargo, el diseño está pensado de forma que los módulos de transformación no lineal no sean una limitación para el 'throughput' que puedan demandar los DSPs. Por un lado, el diseño está segmentado (pipeline) para aceptar un dato por cada ciclo de reloj. Por otro lado, hay tantas instancias como datos en paralelo puede generar el módulo anterior (15, maccs, que se describe en relación con la figura 2).

Se divide el número de neuronas del modelo de cada capa neuronal entre el número de neuronas físicas (o DSPs que se quieran utilizar en la implementación final), para obtener el número de 'folds' o plegados necesarios; esto supone una serialización parcial del procesamiento. Se carga un vector de entrada y se opera con todas las neuronas físicas en forma de ola. Si fuera necesario, el vector vuelve a leerse 'fold' veces, utilizando en cada caso todas las neuronas físicas con diferentes parámetros. Nótese que es necesario cuando el número de DSPs es inferior al número de neuronas de la capa que se está procesando. A eso se le llama plegado de la capa. Además, si hay más capas que procesar, posteriormente estos DSPs también se vuelven a utilizar para procesar la siguiente capa (que puede necesitar un plegado mayor o menor, o ninguno). No obstante, puede instanciarse varias copias del núcleo IP, en cuyo caso los DSP pueden no utilizarse para diferentes capas (sí para el plegado). Si el número de neuronas del modelo pendientes de computar en una ola es menor que el número de neuronas físicas, sólo se utilizan las necesarias. Esto es así incluso cuando ‘fold’ es cero. Es decir, cuando el modelo de la capa tiene un número de neuronas menor que el número de DSPs implementado/sintetizado (paralelismo máximo).

Por ejemplo, en una SLFN con k entradas, capa oculta con 3 neuronas ocultas y 1 salida, que se quiera implementar utilizando sólo 2 DSPs, el modelo se procesa de la siguiente manera: Para la capa oculta, se computan las dos primeras neuronas de la capa oculta [fold 0], se computa la neurona restante de la capa oculta [fold 1] y se aplica la función de activación no lineal a los tres resultados. Para la capa de salida, se computa la neurona de salida [fold 0] y se aplica la función de activación no lineal al resultado.

Nótese que la mayoría de las operaciones que afectan a una capa se realizan en paralelo (varios folds y la función de activación), a pesar de que la exposición en forma de lista exprese secuencia. Nótese también que el plegado (fold) es independiente del número de

nodos de la capa de entrada a la red, es decir, del número de elementos en cada producto vectorial.

La adecuación automática del procesador (núcleo IP) al modelo de red neuronal se realiza en dos fases: en síntesis, por el “plegado” de las capas para reutilizar los recursos hardware de forma optimizada (y la consiguiente adecuación del control y flujo de datos), y en tiempo de ejecución (una vez implementado el procesador) mediante el uso de unaserie de registros de configuración programables, lo que permite ajustar el tamaño de la capa a procesar y el tipo de función de activación, escribiendo en dichos registros.

A continuación se describe un núcleo IP de acuerdo con una implementación de la invención, que optimiza las operaciones de la red neuronal en términos del tiempo de acceso a memoria al aprovechar la localidad de los datos (memoria y registros internos), de forma que se evita el acceso continuo a los módulos de memoria de mayor capacidad pero menor ancho de banda (memoria externa). Al utilizar múltiples scratchpads, también se mejora el ancho de banda local. Se consigue también optimización energética, que es un compromiso entre área ocupada y el tiempo de computación requerido. La configurabilidad de este diseño permite buscar una relación deseada entre área y velocidad. Nótese que el consumo energético está asociado, principalmente, a la frecuencia de operación (consumo dinámico), pero también al área ocupada (consumo estático). Por lo tanto, el consumo energético depende del número de DSPs utilizado en la síntesis. A mayor número de DSPs utilizados, más operaciones en paralelo se realizan, por lo que no será necesario trabajar tan rápido como con un número reducido de DSPs.

La figura 2 ilustra un diagrama de bloques de un procesador neuronal (núcleo IP) 1 de acuerdo con una posible realización de la invención. El núcleo IP 1 es un acelerador computacional adecuado, entre otros, para el procesamiento de redes neuronales artificiales (RNA) de tipo “feedforward”, en el que las operaciones se realizan con aritmética en coma fija, especialmente optimizado para aplicaciones donde los datos a procesar se transfieren en serie. El hecho de que todas las operaciones se realicen en coma fija significa que está optimizado para una utilización eficiente de los recursos lógicos y aritméticos (en el caso de una FPGA) o del área de silicio ocupada (en el caso de ASIC), así como de la latencia de los cálculos y, consiguientemente, del consumo de energía. El diagrama de bloques representa además un diseño altamente parametrizado, que se auto-configura en tiempo de síntesis y que es programable en tiempo de ejecución, como se explica a continuación. Esto significa que un diseñador que vaya a utilizar este núcleo IP 1 simplemente debe especificar las características del modelo de la

red y de la tecnología objetivo (recursos disponibles) en el momento de sintetizar, y el núcleo IP 1 se configurará convenientemente para ajustar aquellas (características del modelo de la red) a ésta (tecnología objetivo). El código fuente desarrollado en VHDL está completamente parametrizado por medio de sentencias 'generic', 'generate' y 'package'. Así, modificando un grupo reducido de parámetros, el código calcula automáticamente el tamaño de palabra (número de bits) necesario en cada señal y registro del circuito. Asimismo, se añaden o eliminan algunos módulos automáticamente, en función de la elección. Por ejemplo, si sólo se sintetiza una función de activación, no es necesaria la lógica de selección de la misma, y se elimina; o, en caso de utilizar doble juego de precisión en la representación numérica, los DSP y módulos de saturación se adecuan para gestionar ambos de forma adecuada. En resumen, el usuario tiene a su disposición un conjunto de parámetros, a nivel alto, que definen el modelo de la red que quiere implementar y ciertos aspectos del hardware que quiere utilizar, y el código asociado al núcleo IP y procedimiento de diseño de la presente invención, ajusta automáticamente todos los aspectos del diseño del procesador 1 (internos o a nivel bajo) para que este se sintetice de forma optimizada cumpliendo los requerimientos que el usuario le impone. Una vez implementado el procesador (núcleo IP) 1, la programación en tiempo de ejecución se consigue sustituyendo constantes 'hard-wired' por registros y, al mismo tiempo, facilitando un mecanismo de comunicación para su lectura/escritura. Concretamente, uno de los puertos Wishbone de entrada al procesador neuronal 1 accede a los registros de configuración en el módulo control. El protocolo utilizado es el mismo que en los puertos de datos, por lo que se pueden consultar/modificar los registros individualmente o en bloque. A mayores, los registros de control están multiplexados para optimizar el número de conexiones a los subcomponentes del procesador 1. Así, es parte de la máquina de estados principal la actualización de los registros en cada subcomponente, tan pronto como se recibe el 'trigger' y antes de iniciar efectivamente la computación.

Antes de entrar en los detalles de cada módulo o elemento del núcleo IP 1 de la figura 2, con objeto de considerar un posible contexto de utilización para el que se ha diseñado el núcleo IP 1, la figura 3 representa un diagrama de bloques de una posible realización de un diseño completo basado en So(P)C (Systemon (Programmable) Chip). Prácticamente todos los módulos/componentes ilustrados en la figura 3 pueden ser integrados en un solo chip (ya sea FPGA, ASIC u otro). Es probable que la memoria DDR (DDR3 en la figura 3) se diseñe para que sea externa al chip, aunque técnicamente es posible incluirla en el mismo. Nótese que los módulos referenciados como “AMBA Interconnect IP”, “PCIe

IP”, “UART IP”, “Timer IP”, “DDR IP”, “CPU IP”, “DDR3” y “PC” no forman parte de la presente invención, y por tanto no deben considerarse limitativos, y se indican a modo de ejemplo del contexto en el que se espera pueda ser utilizado el núcleo IP de la presente divulgación, pero ninguno de estos módulos es necesario para el uso de dicho núcleo IP. No obstante, el núcleo IP 1 está pensado para trabajar con una CPU principal (referenciada como “CPU IP” en la figura 3) y con una memoria externa de alta capacidad (a modo de ejemplo, pero de forma no limitativa, una DDR). Volviendo a la figura 3, los módulos/componentes 3-9 representan diferentes tipos posibles de bloques de memoria que pueden emplearse: ROM de puerto sencillo 3, RAM de puerto sencillo 4, ROM de puerto dual 5, RAM de puerto dual simple 6, RAM de puerto dual lectura/escritura y lectura 7; RAM de puerto dual lectura/escritura y escritura 8; y RAM de puerto dual verdadero 9. Cada uno de estos módulos incluye un submódulo para interpretar el protocolo Wishbone utilizado en la MAI (Matrix-Aware Interconnect). En la figura 3 se muestran también unos módulos/componentes puente 10, 11 que se detallan en la figura 4. Como las BRAM, cada módulo puente 10, 11 incluye submódulos maestros/esclavos para interpretar los protocolos Wishbone y AMBA. En concreto, el primer módulo puente 10 comprende un sub-módulo maestro para interpretar el protocolo Wishbone (WB master en la figura 4), dos FIFO y un sub-módulo esclavo para interpretar el protocolo AMBA (AXI slave en la figura 4). A su vez, el segundo módulo puente 11 comprende un sub-módulo esclavo para interpretar el protocolo Wishbone (WB slave en la figura 4), dos FIFO y un sub-módulo maestro para interpretar el protocolo AMBA (AXI master en la figura 4). Las FIFO se utilizan para maximizar el *throughput*. Estos módulos puente 10, 11 pueden ser parte opcional de la MAI. Lo que en las figuras 3 y 4 se ilustra como “Matrix-Aware-Interconnect” es el núcleo de la misma. Desde el punto de vista del código fuente, un módulo jerárquicamente superior puede incluir la MAI y los módulos puentes, además de algún MCMM. Los protocolos Wishbone y AMBA son la forma de comunicar el procesador neuronal con la CPU. Nótese que un procesador que entienda el protocolo Wishbone no requiere de ningún puente 10, 11 y podría conectarse directamente al núcleo de la MAI. Aunque en la figura 3, de forma esquemática solo se muestra una instancia de cada módulo/componente 3-11, es posible instanciar múltiples copias de cualquiera de ellos, o instanciar solo uno de ellos. Nótese que los módulos e interfaces enumeradas permiten mapear prácticamente cualquier periférico o coprocesador adicional a la MAI. Así, en la figura 3 el módulo/componente 2 representa otro procesador o núcleo IP, que podría incluirse para complementar al núcleo IP 1, como por ejemplo, pero no limitativamente, un resolvedor lineal.

Volviendo a la figura 2, el núcleo IP 1 incluye un módulo o elemento 15 en el que se realizan las operaciones aritméticas para computar productos vectoriales que conformen un producto matricial. El módulo 15 comprende una serie de módulos DSP, módulos de saturación a la salida y registros. Opcionalmente puede comprender multiplexores y el contador asociado a cada uno de ellos. Cada uno de estos DSP se puede implementar con un multiplicador y un sumador, además de recursos auxiliares como puertas lógicas, registros y multiplexores. Esto módulos son bloques de procesamiento habitualmente integrados en las arquitecturas de las FPGAs. En el presente diseño se les han añadido unos módulos de saturación compuesto por un comparador y un multiplexor. La figura 12 muestra dos posibles realizaciones de DSPs modificados 33a, 33b. A estos módulos DSP modificados se les ha llamado también DSP por asociación. El módulo 15 en sí, además de los DSPs, incluye un registro de desplazamiento a la entrada. Por último, a la salida pueden incluirse registros de desplazamiento o multiplexores. Estas dos variantes se muestran en la figura 13: a la izquierda (referencia 15a), mediante registros de desplazamiento y a la derecha (referencia 15b), mediante multiplexores 35. La figura 16 ilustra un ejemplo de combinación de múltiples instancias del módulo 15 de la figura 2 para reducir la latencia.

El módulo 16 es el módulo en el que se realiza la transformación no lineal, que es opcional. Los módulos 12-14 implementan la gestión de la información para leer los datos de entrada. La función de estos módulos es recibir los datos de las dos matrices con las que se debe calcular el producto. El módulo 12 es la interconexión (bus compartido o crossbar) y el controlador asociado a cada FIFO. Se detalla en la figura 5. El módulo 13 representa un conjunto de multiplexores. Su uso principal es la realimentación. El módulo 14 son las FIFOs/scratchpads.

La figura 5 ilustra una posible implementación de los módulos de entrada 12-14 de acuerdo con la presente invención. El módulo 12 está formado por un primer bloque o submódulo 32, que es una interconexión de tipo bus compartido o crossbarswitch, a elección del integrador o diseñador, y por un conjunto de módulos controladores 331-335. Estos controladores son controladores de transferencia, y lo que hacen es adaptar el protocolo Wishbone a las interfaces FIFO. Por lo tanto, son ‘controladores de memoria’. Todos 331-335 son arquitecturalmente iguales. En la figura 5, el número de bloques ctrl 331-335 y de sus FIFOs asociadas no es necesariamente 5, ya que esto depende de la implementación. De forma general, el número de bloques es #DSP+1 (número de DSPs utilizadas más uno). Esta misma consideración es aplicable a los módulos marcados como 142-145 (nótese que 141 sí es diferente del resto). Con respecto al módulo 12, la

figura 14 representa de forma esquemática diferentes variantes 12a, 12b, 12c, 12d de interconexión para adecuar el ancho de banda de entrada a los requerimientos de la aplicación. El bloque 14 está formado por un conjunto de módulos 141-145 que representan un conjunto de memorias FIFO circulares ligeramente modificadas: Hay un registro adicional en comparación con implementaciones convencionales. Este registro sustituye al puntero de lectura en la generación de la señal 'empty'. Estas memorias, junto con el controlador f_ctrl 331-335 del módulo 12, controlador 331-335 asociado a cada una de ellas, actúan como almacenamiento temporal (scratchpads) de los datos que componen las matrices a procesar, de forma que se aprovecha la localidad tanto espacial como temporal de dichos datos. Un scratchpad es un tipo de memoria temporal, similar a una caché, pero con una lógica de control mucho más simple. Aún así, implica cierta lógica de gestión de direcciones. Típicamente las direcciones gestionadas pertenecen al espacio de memoria del microprocesador. En el caso del presente procesador, al interponer la MAI, el espacio de memoria no es el de ningún microprocesador, sino que tiene el tamaño necesario para diferenciar el volumen de datos concreto a gestionar. En suma, puede considerarse un conjunto de registros compartidos, sin mayor información sobre su procedencia o destino. El módulo 13 es, desde un punto de vista funcional, un multiplexor. Su función es dar paso a la realimentación 30 (figura 2). Es el módulo que permite transferir datos directamente del bloque 18 al 14 (figura 2). El patrón de acceso a las memorias FIFO 141-145 se ilustra más adelante en relación con la figura 6.

Los módulos 17-18 (figura 2) implementan la gestión de la información para guardar los resultados. El módulo 17 unifica en uno o varios streams los canales de salida del módulo 16. Lo hace de forma que el resultado esté ordenado como si el procesamiento se hubiera realizado en serie con un único DSP. El módulo 18 es una interconexión análoga al 12 (32). Es decir, un bus compartido o crossbar switch, con uno o varios controladores de transferencia. Como ya se ha dicho, los datos de realimentación salen del módulo 18, entran al módulo 13 y de ahí a su destino (módulo 14). Nótese que más adelante, en la figura 7, el elemento 180 está incluido en el módulo 17 (figura 2). A su vez, el elemento 190 en la figura 7 corresponde al bloque 18 en la figura 2. El módulo 19 es el módulo de control/orquestación. En la figura 7, más adelante, se representa como módulo de control 185. Éste contiene registros de configuración para programar el comportamiento del núcleo IP en tiempo de ejecución. A continuación se explican las conexiones entre módulos, qué información se transmite y qué protocolo puede usarse en cada conexión. La conexión 20 representa los puertos a través de los que se lee el contenido de las dos matrices utilizadas para computar el producto, de los periféricos (preferentemente

memorias) conectados a la MAI y/o a al AMBA Interconnect (ver figura 3). En una posible realización, estos puertos 20 son Maestros Wishbone B4 de solo lectura. Preferiblemente, los Maestros Wishbone B4 de solo lectura implementan el formato de direccionamiento virtual de la MAI, a través de los que leen el contenido de dichas matrices. El número de puertos 20 es definido por el usuario.

Entre los módulos 12 y 13 (el módulo 12 equivale al módulo swi 120 de la figura 7 y el módulo 13 corresponde a una generalización del módulo 130 de la figura 7) se establecen dos interfaces 21, 22 FIFO (First In First Out) de escritura. La interfaz 21 corresponde a la matriz de entrada y la interfaz 22 corresponde a la matriz de pesos. El número de puertos se deriva automáticamente de los parámetros definidos por el usuario. Entre los módulos 13 y 14 se establecen dos interfaces 23, 24 FIFO de escritura. Como en el caso de las interfaces 21, 22, la interfaz 23 corresponde a la matriz de entrada y la interfaz 24 corresponde a la matriz de pesos. En cuanto a las interfaces 25, 26 entre los módulos 14, 15, se trata de interfaces FIFO de lectura ligeramente modificadas que, como en los casos anteriores, corresponden respectivamente a la matriz de entrada y a la matriz de pesos. Las referidas modificaciones se detallan en la figura 6. El número de puertos se deriva automáticamente de los parámetros definidos por el usuario. Entre el módulo 15, en el que se realizan las operaciones aritméticas para computar productos vectoriales que conformen un producto matricial, y el módulo 16, en el que opcionalmente se realiza la transformación no lineal, se establecen unas interfaces 27 FIFO de escritura, a través de las que se transfiere el resultado del producto matricial de los DSP del módulo 15 a las funciones de activación, opcionales, del módulo 16. El número de puertos se deriva automáticamente de los parámetros definidos por el usuario. Entre el módulo 16, en el que opcionalmente se realiza la transformación no lineal, y el módulo 17, se establecen unas interfaces 28 FIFO de escritura. En implementaciones de la invención, el módulo de las funciones de activación 16 puede diseñarse de forma que sea transparente en lo que respecta a la interfaz 28. Los datos transmitidos en esta interfaz 28 son el resultado final de la capa (ya sea la capa oculta, que se computa primero; o la capa de salida, que se computa después). El número de puertos se deriva automáticamente de los parámetros definidos por el usuario. Entre el módulo 17 y el módulo 18 se establecen unas interfaces 29 FIFO de escritura. En implementaciones de la invención, el módulo 17 puede diseñarse de forma que sea transparente en lo que respecta a la interfaz 29. Los datos son transmitidos al módulo 17 a través de la interfaz 28. Estos datos pueden opcionalmente serializarse y/o reordenarse. El número de puertos es definido por el usuario. Entre el módulo 18 y el módulo 13 se establecen unas interfaces 30 FIFO de

lectura y escritura que proporcionan realimentación al núcleo IP 1. Los datos son transmitidos al módulo 18 a través de la interfaz 29. El número de puertos es definido por el usuario.

La conexión 31 representa los puertos a través de los que se escribe, por ejemplo en módulos periféricos, el resultado de la computación ejecutada en el núcleo IP 1. Los periféricos en los que se escribe el resultado pueden ser, por ejemplo, memorias. Estos periféricos suelen estar conectados a la MAI y/o a al AMBA Interconnect (ver figura 3). En una posible realización, estos puertos 31 son Maestros Wishbone B4 de solo escritura. Preferiblemente, los Maestros Wishbone B4 de solo escritura implementan el formato de direccionamiento virtual de la MAI. El número de puertos es definido por el usuario. Por otro lado, las líneas discontinuas entre el módulo de control 19 y los módulos 12-18 representan conexiones ad-hoc para la distribución de parámetros de ejecución desde dicho módulo 19. El tipo de interfaz que implementa estas conexiones es preferentemente memoria/registros direccionables. Puesto que preferentemente su profundidad es de entre 2 y 6 direcciones, el impacto del direccionamiento es despreciable. Por último, la flecha bidireccional a la izquierda del módulo 19 es un puerto es escritura/lectura utilizado para programar el coprocesador (núcleo IP 1) en tiempo de ejecución. A través del mismo se modifican los registros del módulo/componente 19, que después son interpretados y distribuidos automáticamente al resto de componentes. En una posible realización, este puerto es un esclavo Wishbone B4. En otras palabras, la capacidad de programación en tiempo de ejecución se consigue proveyendo un puerto de escritura/lectura que permite interactuar con el módulo 19. Éste incluye una serie de registros y varias máquinas de estado básicas, tales como máquinas de estado basadas en dos o tres bits de estado, que permiten automatizar la modificación del comportamiento de la arquitectura.

La figura 6 ilustra el patrón de acceso a las memorias FIFO 141-145 mostradas en la figura 5, de acuerdo con una posible implementación de la invención. Concretamente, se ha representado el patrón de acceso a las memorias FIFO 141-145 en un ejemplo que corresponde a la multiplicación de una matriz de tres filas por otra de cuatro filas (siendo independiente del número de columnas, que debe ser igual) teniendo en cuenta la posibilidad de aplicar el plegado de una capa de un modelo de red neuronal en un número determinado de DSPs. La referencia 14(i) se refiere a la primera memoria FIFO 141 de la figura 5, mientras que la referencia 14($b_?$) se refiere al resto de memorias FIFO 142-145 de la figura 5. Como puede observarse, la memoria 141 (14(i) en la figura 6) muestra mayor localidad temporal de los datos, leyéndose un mismo vector varias veces

de forma consecutiva. En el caso de la memoria 142-145 (14($b_?$) en la figura 6), hay localidad espacial equivalente (en ambos casos la lectura es vectorial), pero se observa un comportamiento menos favorable al aprovechamiento temporal. Nótese que, en caso de utilizar FIFOs circulares sin modificar, cada vector sólo puede ser leído una sola vez antes de que pueda ser sobreescrito. Por ello, las modificaciones realizadas se basan en la duplicación del registro de lectura: En la memoria 141 (también llamada 14(i)), existen dos señales de un solo bit adicionales a una memoria FIFO corriente. Una primera de ellas guarda el valor del puntero de lectura en registro de respaldo, que es el utilizado para general la señal full. La segunda de esas señales permite retornar el puntero de lectura efectivo al valor registrado. En la memoria 142-145 (también llamada 14($b_?$)), existe una sola señal adicional. Ya que la lectura de los vectores correspondientes a múltiples pliegues (folds) consecutivos es secuencial, únicamente se dispone de la señal que permite volver al inicio. Adicionalmente, en aplicaciones que requieran procesar productos con diferentes matrices (tal es el caso, por ejemplo, cuando se usan kernels privados en vez de kernels públicos en redes profundas), las memorias 142-145 (también llamadas 14($b_?$)) se pueden implementar con las mismas modificaciones que la memoria 141 (también llamada 14(i)), de forma que se aproveche todo el espacio disponible, cargando kernels tan pronto como los anteriores dejan de utilizarse.

La figura 7 ilustra un posible ejemplo de implementación de un núcleo IP 100 de acuerdo con la invención. Este ejemplo responde a una implementación según cierta parametrización. Nótese que existen muchas posibilidades alternativas de síntesis. Los datos de entrada, que se transfieren a través del puerto 101 (20 en la figura 2) son los vectores que componen las dos matrices cuyo producto se va a computar. Los datos correspondientes a una de las matrices se transfieren únicamente a la FIFO 'i' (140_i en la figura 7, 141 en la figura 5) del bloque 140 (bloque 14 en las figuras 2 y 5) mientras que los datos correspondientes a la otra matriz se distribuyen en las FIFOS 'b_0…b_9' (140_{b0}…140_{b9} en la figura 7) del bloque 140. El sub-módulo 320 (32 en la figura 5) del bloque 120 (bloque 12 en las figuras 2 y 5) puede ser un bus compartido o un crossbar switch, según se configure. En el caso de que sea un crossbar switch, se dispone de más de un puerto de entrada, por lo que se implementan varias instancias de los puertos para la entrada de datos 101. El árbitro de la interconexión, que se implementa en el bloque de intercon_si (320 en la figura 7, 32 en la figura 5), actúa de doble maestro y convierte los índices facilitados por los módulos 3301-3311 del bloque 120 (convierte la interfaz Wishbone en interfaz FIFO). Nótese que actuar como doble maestro quiere decir que tanto la MAI 200 como los módulos 3301-3311 son esclavos, por lo que es el árbitro

quien inicia y finaliza para transferencia. Los módulos 3301-3311 son conversores de protocolo, en este ejemplo, de esclavo Wishbone a interfaz FIFO. La dirección está compuesta por dos índices, que son contadores encadenados. El valor de límite de cuenta de cada uno de ellos depende del valor de los registros de control en tiempo de ejecución.

El núcleo IP 100 tiene un bloque multiplicador matriz-vector en coma fija (bloque maccs)150, que implementa la operación MACC (Multiply ACCumulate) basado en un array sistólico lineal con carga de parámetros en paralelo y ejecución tipo ola. El bloque maccs150 está optimizado para la multiplicación de una matriz por múltiples vectores recibidos en serie de forma continua (stream), efectuando en la práctica un producto matricial. La elección de las etapas de entrada y salida al mismo, así como la variedad en la lógica de control, permite configurar este núcleo IP 100 para realizar el procesamiento de distintos modelos de RNAs de tipo feedforward como por ejemplo, pero de forma no limitativa, las “Extreme Learning Machines”, las “Random Vector Functional-Link (RVFL)”, y,en general, cualquier RNA de tipo multicapa, incluidas las “Deep Neural Network (DNN)” o las “Convolutional Neural Network (CNN)”.

Puesto que hoy día la mayoría de dispositivos de almacenamiento masivo (memorias), con capacidad suficiente para almacenar el volumen de datos requerido en las aplicaciones objetivo, y con tiempo de acceso que permitan la ejecución en tiempo real, tienen interfaces serie, en el diseño del núcleo IP 100 ilustrado se ha impuesto esta característica como condición de diseño. Ello implica que el circuito debe considerar la necesidad de distribuir los datos recibidos en serie a través de la interfaz 101 (o interfaces, si el módulo 320 es un crossbar) a múltiples módulos aritméticos trabajando en paralelo. Por el requerimiento de ir paralelizando los datos recibidos en serie, no se espera tener disponible al mismo tiempo más de un elemento de un vector. Así, hablamos de recibir en serie cada dato (que constará de múltiples bits), no cada bit. Asimismo, deben recogerse los resultados y serializarse para volver a enviarlos a la memoria principal a través de la interfaz de salida 102 (o interfaces, si el sub-módulo 171del bloque 170 es un crossbar).

En realizaciones de la invención, para aprovechar la localidad de los datos, en el bloque 140 se utilizan bloques BRAM 140_i 140_{b1}-140_{b9} en forma de FIFO, tal como indica el nombre del bloque fifos140.La estrategia habitual es esperar a la recepción de un número de datos igual al número de módulos trabajando en paralelo e iniciar la computación (bloque 150) al mismo tiempo en todos ellos. El número de módulos trabajando en

paralelo viene dado por el número de DSPs que se utilicen en el bloque 150. Además, el número de salidas del módulo 320 siempre es uno más que el número de DSPs, ya que la salida adicional del módulo 320 es la entrada al registro de desplazamientoR1-R9 (que, a su vez, va a cada DSP). Por lo tanto, si la latencia del circuito es menor que el tiempo requerido para cargar cada grupo de datos, hay periodos de ineficiencia en que los módulos aritméticos no se utilizan. En cualquier caso, siempre hay una latencia inicial. Por otro lado, al finalizar, todos los datos están disponibles al mismo tiempo, por lo que, en una posible realización, serializarlos requiere recorrer cada una de las salidas. En una realización alternativa, se pueden encadenar los registros de salida a modo de registro de desplazamiento, de forma que no se requiera multiplexor. Estas dos alternativas se ilustran en la figura 13, que se han referenciado como 15a y 15b. En el ejemplo concreto de la figura 7, el bloque maccs150 se ha dividido en tres grupos 150a 150b 150c. Los dos primeros 150a 150b tienen cuatro DSP cada uno (DSP_0-DSP_3 el sub-bloque 150a y DSP_4-DSP_7 el sub-bloque 150b), mientras que el tercero 150c tiene dos DSP (DSP_8-DSP_9). El número de grupos se calcula automáticamente en tiempo de síntesis a partir de los parámetros facilitados por el usuario. Cada sub-bloque tiene un multiplexor MUX_a, MUX_b, MUX_c para serializar los datos del respectivo sub-bloque 150a 150b 150c. El número máximo de DSPs en cada grupo es igual a la longitud mínima esperada en el vector de entrada, que es un parámetro indicado previo a la síntesis. En el ejemplo de la figura 7, el valor de dicho parámetro es 4. Siempre que el remanente de la división entre el número de DSPs y este parámetro sea diferente de cero, el último grupo o sub-bloque tendrá un tamaño menor. En este caso 10 rem 4 = 2. La figura 8 ilustra un diagrama de bloques de otro ejemplo de núcleo IP 100’ de acuerdo con otra posible realización de la invención, en la que los registros de salida se encadenan a modo de registro de desplazamiento, no necesitándose por tanto los multiplexores.

Una novedad que puede destacarse en este núcleo IP 1, 100, 100’ y procedimiento de diseño del mismo, es que se realiza un análisis inverso de la secuenciación, es decir, el ajuste de los requerimientos de throughput y latencia en cada uno de los módulos se realiza a partir de la capacidad máxima de escritura. Es decir, la arquitectura se ajusta automáticamente a los condicionantes impuestos en su parametrización. El efecto principal es la derivación automática del número de módulos 'sigmoid_wp' del bloquede funciones de activación 160 (bloque 16 en la figura 2) y, a partir del número de salidas del módulo 150 (bloque 15 en la figura 2), que depende de un parámetro de síntesis, como se ha explicado. Concretamente, la operación MACC 150realizada en los módulos DSP_0-DSP_9 comprime z elementos en uno solo, por lo que un solo puerto que escriba datos en

serie a un dato por ciclo puede gestionar las salidas de los módulos aritméticos (cada DSP y su bloque de saturación). Nótese que z es un parámetro programable en tiempo de ejecución, y corresponde a la longitud de los vectores con los que se computa el producto. Aludiendo a la figura 1, concretamente los valores que puede tomar z son: z= f; z= f+1; z= h; o z= h+1, dependiendo de si se está computando la red de entrada o de salida y de si se está utilizando bias o no. Tal como se ha explicado, el producto vectorial se computa de forma secuencial (de ahí que este parámetro z no se refleje en la figura 7). En la arquitectura, z se implementa mediante un registro ubicado en el módulo de control 185 (módulo 19 en la figura 2). Puesto que debe leerse la salida de uno de ellos en cada ciclo de reloj, es razonable pensar que resulta una estrategia ineficiente generar resultados en varios de ellos al mismo tiempo. En una realización preferente, por eficiencia, cada módulo (cada DSP y su bloque de saturación) genera el resultado válido en el ciclo inmediatamente posterior al módulo anterior, a diferencia de lo que ocurre, por ejemplo, en el diseño de DianNao. Como todos ellos tienen la misma latencia, para que los resultados se generen con un ciclo de diferencia, la computación debe iniciarse en el mismo orden. La evolución temporal de las señales de control cumpliendo los requisitos anteriores permite observar un patrón de tipo ola, tal como el ilustrado a modo de ejemplo en la figura 15, característico de los registros de desplazamiento. En el diseño propuesto se aprovecha dicho patrón para conectar un único puerto de entrada de datos 140out (salida del elemento 140_i en el bloque fifos140) sólo al primer DSP DSP_0 del bloque maccs140 de forma directa, y al resto de DSPs DSP_1-DSP_9 a través de una serie de registros encadenados R_1-R_9. Así, se reduce la complejidad del rutado y el fanout requerido. Asimismo, reducir la longitud de las pistas permite la operación con frecuencias de reloj mayores. Además, la computación en el primer DSP DSP_0 se inicia tan pronto como esté disponible el primer dato, por lo que la latencia inicial es independiente del número de módulos aritméticos utilizados en paralelo; se reduce a la longitud del vector.

Los módulos aritméticos operando en paralelo generan hasta #DSP datos (tantos datos como número de DSP haya) cada z ciclos, donde z es la longitud del vector. Por lo tanto, si #DSP>z (si el número de DSP es mayor que z), una sola salida no puede recoger todos los resultados generados. Por ello, un parámetro de diseño es la longitud mínima de los vectores que se desean procesar. En base al mismo, se generan en tiempo de síntesis ceil(#DSP/g_z) puertos de salida, y g_z es un parámetro que indica el valor mínimo que puede tomar z, donde ceil es una operación de redondeo hacia arriba, y se añade un módulo de multiplexación MUX_a MUX_b MUX_c en la figura 7 entre cada grupo de

g_z DSPs y el correspondiente puerto. El tamaño máximo de los grupos 150a, 150b, 150c está condicionado por g_z. En el ejemplo mostrado, cada módulo de multiplexación se implementa mediante un multiplexor y una pequeña máquina de estados basada en un contador. Las señales de sincronización interna se ajustan automáticamente, considerando también los casos en que el remanente de la división no es cero, por lo que se deben gestionar instancias de diferente tamaño, como representa el sub-boque 150c en la figura 7. Hay dos posibles implementaciones de estos bloques de multiplexación: Como se ha comentado, en posibles implementaciones de la invención, tal y como en la ilustrada en la figura 7, el bloque de multiplexación MUX_a MUX_b MUX_c se implementa mediante un conjunto de multiplexores y un contador. Esta implementación no implica ninguna latencia pero, en función del tamaño de grupo elegido, los requerimientos lógicos pueden ser considerables. En otras implementaciones de la invención, tal y como la ilustrada en la figura 8, el bloque de multiplexación se implementa mediante un registro de desplazamiento R'_0-R'_9 con carga independiente por registro y el contador del operador máximo en el siguiente bloque sigmoid_wp del bloque de funciones de activación 160. Una característica de este diseño es que el sistema de control está muy simplificado (se puede decir que es una máquina de estados con solo cuatro estados) y prácticamente no tiene programación. El diseño se ha realizado de tal manera que cuando se genera la arquitectura a implementar se crean automáticamente los elementos necesarios (contadores) para la generación de las señales de control que gestionan el flujo de datos. En este caso, un simple contador actúa de máquina de estados: tan pronto como los datos de todo el grupo están disponibles, se lee secuencialmente la salida del registro de desplazamiento, tantas veces como salidas tenga el grupo. Al llegar el último, se genera un pulso que 'pasa el relevo' al contador del siguiente grupo. En esta implementación hay que esperar a que se escriba el grupo para iniciar la lectura. El tamaño máximo del grupo es g_z/2, en lugar de g_z. Las líneas son mucho más cortas y los requerimientos de rutado se simplifican.

Dada esta división (es decir, la agrupación de los módulos aritméticos), se pueden conectar varios grupos de salidas (de bloques de multiplexación o registros de desplazamiento equivalentes) en paralelo, de forma que el puerto de entrada se conecte en paralelo a hasta tantos DSP como grupos haya. La figura 14 representa de forma esquemática esta posibilidad de instanciación múltiple del módulo 15. Esto permite establecer un compromiso entre los requerimientos de rutado y la latencia hasta alcanzar la máxima eficiencia del circuito. Se trata de una característica especialmente interesante para implementaciones 3D (apilamiento de sucesivas capas de procesamiento 2D), es

decir, implementaciones en las que se distribuye el cómputo de un número indeterminado de productos vectoriales entre un número menor de unidades aritméticas. Puede interpretarse como un 'folding' o plegado de segundo nivel, ya que se trata del mismo 'loop' en software. La lógica de control que orquesta la ejecución en los diferentes módulos en paralelo se ha optimizado diseñando una máquina de estados que toma como referencia sólo el primer DSP DSP_0. Esta máquina utiliza tres contadores de apoyo: longitud del vector, número de repeticiones y latencia del módulo aritmético. Considerando que la secuencia en el resto de módulos es idéntica aunque retardada, se utiliza preferentemente un registro de desplazamiento de tres bits para transferir las señales de control. Esto supone un requerimiento adicional con respecto a las soluciones que ejecutan todos los módulos aritméticos con la misma secuencia. Sin embargo, cuando el número de estos es alto, por las razones anteriormente descritas, puede ser aconsejable la introducción de registros para reducir la longitud de las pistas y simplificar el rutado. Por lo tanto, esta es una situación de compromiso en la que se ha optado por utilizar unos pocos registros más para el diseño, en previsión de configuraciones que requieran un número alto de módulos aritméticos operando en paralelo.

La ejecución del módulo 150 es solidaria, es decir, todos los DSP funcionan al mismo ritmo siempre y cuando i) ninguna de las memorias de las que se requiera leer en un ciclo dado esté vacía y ii) ninguna de las salidas a las que se requiera escribir en un ciclo dado esté llena. Para ello, las señales “vacías” de las memorias del módulo fifos 140 están encadenadas con puertas “or”, de forma que se reduce la complejidad del rutado. Las señales “llenas” de las memorias del módulo smerger_wp 170 se evalúan al mismo tiempo, ya que escogiendo adecuadamente los parámetros de síntesis es poco probable que se llenen, y su número es muy reducido en comparación con el número de DSPs. El módulo smerger_wp 170 está formado por un grupo de FIFOs simples 170a, 170b, 170c y por un serializador (smerger) 171. Esto quiere decir que el datapath es un multiplexor, y hay dos contadores encadenados para producir un orden de salida equivalente al que produciría un procesador secuencial. Dada la solidaridad en la ejecución, la lógica de control es la misma independientemente de que se utilice una disposición en 2D o 3D.

La figura 7 muestra también un núcleo de interconexión 200, referido también como bloque MAI (*Matrix Aware Interconnect*), que ha sido diseñado en paralelo al núcleo IP 100 para facilitar su integración en un sistema heterogéneo con uno o varios módulos coprocesadores. El núcleo de interconexión 200 permite utilizar el acelerador (núcleo IP 100) en solidaridad con otros módulos para acelerar también la fase de entrenamiento de algunas de las metodologías, y no sólo en el procesamiento. El núcleo de interconexión

200 (bloque MAI 200) se encarga de la gestión de bloques de memoria internos y externos. El núcleo de interconexión 200 se detalla más adelante.

El diseño del núcleo IP1, 100, 100'está completamente parametrizado, lo que significa que es configurable antes de la síntesis. En una posible implementación, en la que el módulo de procesamiento 10 se diseña para sintetizar una red de una sola capa oculta (SLFN, *Single Layer Feedforward Network*) con una única instancia del bloque maccs15, 150, en el diseño se utilizan un conjunto de parámetros que representan el tamaño de la red sintetizada y un conjunto de parámetros genéricos del módulo.

Los parámetros que representan el tamaño de la red sintetizada son:

x_m número de maccs. Se trata del número de DSPs que hay en cada bloque 15, 150. Nótese que el número de sub-bloques 150a 150b 150c viene definido por ceil(g_z/x_m);

x_snúmero de sigmoids. Se trata del número de sub-bloques160a 160b 160c dentro del bloque de funciones de activación 160;

x_vmaxvectors. Se trata del número máximo de vectores que se espera procesar de una tirada, que son los que se cargarían de un elemento externo a la memoria interna para su procesamiento como un 'stream'. Concretamente, define el valor de fin de cuenta de uno de los contadores en el módulo f_ctrl120 correspondiente a la FIFO 'i' 140_i del módulo fifos 140. El módulo 150 es agnóstico sobre el número total de vectores que se van a procesar.

x_f max features (inputs).Se trata del número máximo de elementos que se espera tenga cada vector de un producto vectorial, es decir, la dimensión máxima de los vectores de entrada que tiene que procesar la red neuronal (que es el número de nodos que tiene la capa de entrada de la red).

x_h max hidden neurons. Se trata del número máximo de productos vectoriales que van a computar con un mismo vector de entrada. En el caso de una SLFN, es el número de neuronas en la capa oculta.

x_c max classes|regressions (outputs). Se trata de, en caso de utilizar el fichero de parametrización para SLFNs, el número de nodos o neuronas en la capa de salida. Nótese que normalmente x_c<<x_h, por lo que este parámetro es una salvaguarda.

g_z min features (inputs). Se trata del número mínimo de elementos que se espera tenga cada vector de un producto vectorial. La relación entre este parámetro y

x_m define en cuántos grupos se dividirámaccs 150.

El conjunto de parámetros genéricos del módulo son:

g_mltlatencia de MACC. Se refiere a latencia de un módulo aritmético, es decir, un DSP y el módulo de saturación inmediatamente posterior al mismo. Debe ser modificado si se utiliza otra descripción para los mismos. En principio, no se espera que se modifique en la mayoría de aplicaciones y es fijo.

g_mp MACC accumulator width 48|64.La operación MACC implica la acumulación de sucesivos productos, normalmente en un registro. De la correcta elección de su tamaño depende: i) la no incursión en overflow al operar con coma fija, ii) la inferencia de módulos DSP 'duros' en las plataformas que disponen de los mismos (por ejemplo en las FPGAs). Este parámetro define el tamaño de dicho registro.

g_slt Sigmoid latency. Se refiere a la latencia de cada módulo sigmoid_wp (160a, 160b, 160c) del bloque de funciones de activación 160.

g_swi_b switch i max block length. Las transferencias a través de la interfaz de entrada 101 se realizan en bloques o ‘bursts’. Cada uno de los esclavos intervinientes en la comunicación pueden pausarla. Adicionalmente, para evitar que una sola comunicación absorba todo el ancho de banda, el árbitro puede decidir cuándo finalizarla para atender otras peticiones. Este parámetro establece el número máximo de elementos que deben transmitirse en cada bloque. Nótese que a menor valor de g_swi_b habrá mayor 'overhead' debido al protocolo, por lo que el throughput se verá reducido. Mayores valores de g_swi_b mejorarán el throughput a costa de introducir latencias ligeramente mayores. No obstante, el procedimiento recomendado, cuando los recursos y requerimientos lo permitan, es cargar todas las fifos antes de comenzar la computación, de forma que todo el ancho de banda esté disponible para FIFO 'i'.

g_swi_a switch i max stb to ack offset (power). El protocolo Wishbone utilizado envía 'STB' y espera recibir 'ACK' como respuesta. Debido a los elementos que intervienen en la comunicación, la respuesta no es inmediata y es posible que se envíen varios STB antes de recibir el primer ACK. Este parámetro define el tamaño del contador que hace un seguimiento de la diferencia entre el número de STB enviados y el número de ACKs recibidos.

g_swo_b` switch o max block length. Lo mismo que g_swi_b, pero aplicado a la interfaz de salida 102.

g_swo_a` switch o max stb to ack offset (power). Lo mismo que g_swi_a, pero aplicado a la interfaz de salida 102.

A partir de los parámetros anteriores y de la elección hecha entre la variedad de arquitecturas ofrecida en algunos de los módulos (tales como el módulo 320 sea un bus compartido o un crossbar; o utilizar multiplexores o registros de desplazamiento en los módulos MUX_a, MUX_b, MUX_c; o utilizar una sola o varias funciones de activación, además de opcionalmente el operador de máximo y el bypass; o la opcionalidad de la realimentación de la salida del elemento 180 al módulo 130), el diseño optimiza en tiempo de síntesis el tamaño de todos los registros, señales y elementos de memoria del circuito, utilizando los recursos mínimos necesarios para posibilitar la ejecución de problemas con los tamaños dados. A continuación se especifica cuál es la variedad de arquitecturas/funcionalidades contemplada en el diseño:

-Bias de las funciones de activación de cada neurona. Las RNAs contienen un conjunto de parámetros ajustables cuyos valores se optimizan aplicando algún algoritmo de entrenamiento/aprendiaje (learning). Estos parámetros son los denominados “bias” de los nodos o neuronas y los denominados “pesos” de las interconexiones entre neuronas. El “bias” de las funciones de activación de cada neurona se introduce como valor de precarga en el DSP. Cada neurona requiere una DSP (como se ha dicho, no necesariamente exclusiva a ella) para realizar la operación de suma de productos (MACC) necesaria para el cómputo de la función neuronal. Después de esta operación se pasa, opcionalmente, la salida por la función de activación. Asimismo, se recibe a través del mismo puerto que los pesos. Como resultado, para su implementación sólo se requiere una señal de habilitación de un solo bit cuidadamente sincronizada. Esta señal es parte del registro de tres bits anteriormente mencionado. Esta solución es más eficiente que otras soluciones que requieren un canal de carga específico para los bias. La utilización de los bias se define (se programa) opcionalmente en tiempo de ejecución para cada operación matricial.

-Funciones de activación. El producto vectorial (o productos), es decir, el resultado de cada operación MACC, puede ser filtrado por una función (en el sentido matemático). Esto completa el procesamiento neuronal de la(s) capa(s) oculta(s). La implementación de la figura 7 incluye un bloque 160 de generación de una o más funciones de activación. Los sub-módulos 160a 160b 160c se refieren a distintas instancias (copias) del mismo bloque, no a funciones de activación diferentes. Cada instancia 160a 160b 160c puede incluir como submódulos diferentes funciones. Ejemplos no limitativos de estas funciones

son la sigmoide logística, la función rectificadora, la función máximo, la función tangente hiperbólica, funciones de base radial, etc. Estas funciones se implementan por medio de un circuito basado en la Interpolación Recursiva Centrada (CRI, *Centered Recursive Interpolation*), que permite la selección de distintos tipos de funciones, y que optimiza notablemente los requerimientos de recursos lógicos. Concretamente, se evita cualquier multiplicación adicional, limitando la complejidad del circuito a un comparador, un restador y desplazamientos cableados (*hard wired shifts*). Adicionalmente, cuando se computa una función simétrica, se opera sobre el módulo, que es transformado en el último paso en caso de que el signo de entrada fuera negativo. El componente se encuentra completamente segmentado (*pipelined*), por lo que permite un flujo o throughput de un dato por ciclo de reloj, y la latencia total del circuito es de 4 ciclos como máximo. En la figura 7, se ha referido cada circuito CRI configurable y programable 160a 160b 160c como sigmoid_wp. En realidad, un circuito CRI puede generar distintas funciones de activación, como las mencionadas anteriormente a modo de ejemplo. Antes de la síntesis del circuito se puede configurar el número de funciones diferentes a implementar, y una vez implementado el núcleo IP, se puede escoger entre las opciones implementadas cambiando sólo un selector (una señal de longitud log2(#funciones), es decir, logaritmo en base 2 del número de funciones).

-Realimentación. La salida del bloque 180puede ser realimentada para efectuar dos operaciones matriz-vector consecutivas (en el bloque MACCs 150), con un filtrado intermedio opcional (en el bloque de función de activación 160); es decir, con una sola instancia de maccs se procesa tanto la capa oculta como la capa de salida de la red (procesamiento completo de un vector en una “Single Layer Feedforward Network” o SLFN). También puede guardarse el vector de entrada que se está procesando, que se recibe a través de la interfaz de entrada 101, hasta la recepción del producto de las operaciones MACC filtrado por la función de activación de la primera operación, computando la segunda operación matriz-vector como en una RVFL. Nótese que una RVFL es un tipo de red similar a una SLFN, pero en aquélla algunas conexiones de los nodos de entrada se realizan directamente con nodos de salida, sin pasar por neuronas de la capa oculta. Esta funcionalidad de realimentación se gestiona desde el módulo de control 180, ilustrado en la figura 7, desde el que se define el modelo de diseño del núcleo IP 100. La síntesis de la línea de realimentación, así como la lógica asociada a su gestión, tiene un coste en recursos lógicos que puede ahorrarse si sólo se van a procesar redes regulares, en las que no existan saltos de interconexión entre capas (como ocurre, por ejemplo, en las RVFL). Por ello, se puede implementar el núcleo IP con

realimentación o sin ella. En el caso de que se incluya esta realimentación en la síntesis, su uso es opcional en tiempo de ejecución (se puede utilizar o no).Es decir, en tiempo de síntesis se definen qué modelos de red se van a poder ejecutar y en tiempo de ejecución se escoge un modelo de red en particular escribiendo en un solo registro. Es decir, se aprovecha la localidad de los datos, evitando el paso por la memoria principal (que no forma parte de la presente divulgación).

-Redes multicapa. Para implementar modelos de red multicapa puede reutilizarse (instanciar una sola copia y realizar múltiples ejecuciones) el componente principal (núcleo IP 1, 100, 100'), o instanciarlo múltiples veces (tantas como capas quieran implementarse). A lo largo del presente texto se utiliza la expresión "capas físicas" para aludir al número de instancias implementadas en hardware, y la expresión "capas del modelo" al número de capas que tenga el modelo de RNA a computar.

-Coma fija. Todas las operaciones se realizan en coma fija de precisión ajustable, lo cual quiere decir que se puede definir el tamaño de palabra utilizado en cada una de las matrices que intervienen en cada ejecución. Se ha automatizado la declaración de constantes en tiempo de síntesis, de forma que a partir de la definición del problema se calculan los tamaños óptimos en cada componente del circuito. Esta optimización permite reducir, por un lado, el número de elementos de memoria necesario y, por otro, la complejidad de los operadores aritméticos utilizados. En el caso particular de implementación en FPGA, esto último sólo tiene reflejo en las funciones de activación, ya que los módulos DSP son recursos duros (hard), es decir, no pueden modificarse. No obstante, esto sí es relevante para el diseño de ASICs, ya que en este caso solo se realizan operadores aritméticos de la precisión requerida. En realizaciones de la invención, se utilizan tamaños de palabra fijos en todo el circuito, por lo que siempre se computa con la misma precisión. Es la solución más eficiente en área. En otras realizaciones de la invención, se utilizan dos conjuntos de tamaños, por lo que se pueden computar dos operaciones con diferente precisión. Esta opción está especialmente pensada para computar SLFN/RVFL en dos pasos con una sola instancia del componente principal. El incremento en área es mínimo. En otras realizaciones de la invención, se incluye un "barrel shifter" a la salida de cada DSP, para que la precisión sea ajustable en tiempo de ejecución. En este caso, el módulo de control 185 permite cambiar la precisión escribiendo un único registro. Sin embargo, el "barrel shifter" puede ser un componente costoso en función del tamaño de palabra utilizado y las posibilidades de desplazamiento deseadas. En suma, en comparación con las soluciones en coma flotante, la coma fija ofrece una mayor precisión utilizando el mismo tamaño de palabra,

siempre y cuando el rango de las señales pueda mantenerse representado. Por ello, para minimizar el riesgo de incorrección, todas las operaciones se realizan preferentemente en “full precisión” y preferentemente los resultados se saturan antes de truncarse o aplicar “trounding”.

El funcionamiento de todos los componentes del circuito (núcleo IP 100) se deriva del bloque maccs 150. Puesto que el bloque maccs150tiene un patrón de ejecución definido, la mayoría de módulos (todos excepto el módulo de control 180) funcionan como un “stream” (nótese que en un stream la mayoría de componentes no tienen noción de la función que cumplen en el sistema y no tienen sincronización explícita más que con los módulos inmediatamente anteriores y posteriores. Además, esta sincronización es básica en complejidad, por lo que se prioriza el uso de 'flags' de uno o dos bits),y no se gestiona ninguna información relativa a la posición de memoria de los datos, optimizando así los recursos lógicos necesarios para su gestión. La orquestación de las operaciones garantiza que los resultados a la salida se obtienen en un orden determinado. A continuación se detalla cómo se gestiona la información en cada módulo y qué rangos se utilizan. En concreto, a continuación se explica: las conexiones a la MAI 200, la entrada de datos, la salida de datos, los dominios de reloj y la programabilidad y control:

-Conexiones al núcleo de interconexión 200 (MAI). Los puertos de conexión a la MAI 200 reflejados en las figuras 2-3 son de sólo lectura (puerto 101 de conexión al bloque swi 11) o sólo escritura (puerto 102 de conexión al bloque swo 17). Dependiendo de la versión de módulo MAI 200, esto puede suponer una optimización en la implementación. Por otro lado, existe un tercer puerto (no ilustrado en las figuras 7-8) de lectura y escritura entre el módulo MAI 200 y el bloque de control 185 del núcleo IP 100, que se describe más adelante. En lo que respecta a las transferencias entre el bloque swi120 y el bloque swo190, existen principalmente dos posibilidades de implementación: implementación basada en mapeo en memoria (*memory-mapped*) e implementación en modo “stream”, ya que los datos se utilizan por filas o columnas de una matriz y el módulo MAI 200 permite transferencias en bloque utilizando cualquiera de las ordenaciones. Por otra parte, en las interfaces con el módulo MAI 200 se utilizan direcciones virtuales que representan punteros de matriz, columna y fila. El módulo MAI 200 se encarga de traducir esa representación a la dirección física de los bloques de memoria externos al núcleo IP 100.

-Entrada de datos. Uno de los aspectos principales a explotar para acelerar la computación de algoritmos con requerimientos intensivos de acceso a memoria es el

aprovechamiento de la localidad de los datos. Es sabido que un estudio minucioso de la localidad tanto temporal como espacial permite evitar el paso por memoria principal (externa al núcleo IP 1, 100, 100') utilizando bloques de almacenamiento más pequeños y cercanos a los módulos aritméticos. En realizaciones de la invención, se utilizan N módulos aritméticos DSP (#DSP) DSP_0-DSP_9para efectuar h multiplicaciones vectoriales, donde h es el número de filas de la matriz en el producto matriz-vector. Nótese que h depende del modelo de capa que se desee computar. Concretamente, h se utiliza para calcular el número de folds (pliegues) necesarios. Por lo tanto, si h>#DSP (si h > N), se requiere más de una pasada, iteración o repetición para procesar cada vector de entrada. Esta capacidad de repetición se denomina plegado de capas (en inglés, *folding* o *tiling*), y se refiere al número de repeticiones que requiere un vector. Nótese que el plegado se refiere al número de veces que se reutiliza el bloque maccs 150, pero debe tenerse en cuenta que las pasadas (olas) por el bloque maccs 150 son pasadas (olas) continuas (puede verse como un solapamiento) hasta que deja de haber información que procesar. Es decir, si por ejemplo h=12, #DSP=3 y se procesan 5 vectores consecutivamente, habrá ceil(12/3)*5=20 olas de datos circulando por el bloque maccs 150.En el presente texto, el término "pasada" u "ola" se aplica a ceil(#DSP/h), por lo que plegado = pasada -1 (*folding=pass* – 1) o, lo que es lo mismo, el número de pasadas u olas = plegado + 1. Así, si h<=#DSP, un vector se procesa en una sola pasada/ola, por lo que no hay repetición (folding=0). Como se ha dicho, para aprovechar la localidad de los datos se utilizan bloques BRAM 120_i 120_{b1}-120_{b9} en forma de FIFO (bloque fifo 140 en las figuras 7 y 8). El primer bloque BRAM 140_i se utiliza para como buffer para los vectores de entrada. El resto de bloques BRAM 140_{b0}-140_{b9} almacenan el contenido de la matriz: cada uno de ellos almacena una fila de cada grupo de N filas de la matriz, donde N es el número de DSP (#DSP). La figura 6 muestra gráficamente los patrones de acceso a los bloques BRAM que se utilizan como fifos para el procesamiento de tres productos matriz-vector consecutivos en un problema con cuatro pasadas (i a la derecha, y el resto (b?) a la izquierda). En relación con la entrada de datos, a continuación se explica la entrada de datos al primer bloque fifos120_i, la entrada de datos al resto de bloques fifos120_{b0}-120_{b9} y la entrada de datos al bloque swi 11 (sub-bloque intercon_si 110 y sub-bloque f_ctrl111_0-111_{10}).

Con respecto al primer bloque fifos 140_i, este bloque tiene una alta localidad espacial y temporal. Tanto es así, que la profundidad mínima recomendada es de unos pocos vectores, ofreciendo unas propiedades de escalabilidad muy adecuadas para problemas con gran cantidad de datos a procesar. A la hora de reflejar este comportamiento en la

descripción del módulo1, 100, 100', se ha modificado la referencia de inferencia de FIFOs para proteger cada vector hasta que su utilidad haya expirado. Nótese que la referencia de inferencia son plantillas que facilitan el fabricante de la plataforma objetivo (léase FPGA) y/o el desarrollador de la herramienta de síntesis, para que los recursos lógicos descritos en HDL produzcan el resultado deseado. Al modificarla, no se pueden utilizar FIFOs 'duras'. Sin embargo, la mayoría de FPGAs implementan éstas mediante BRAMs con lógica adicional optimizada para que se comporten como tales. Por ello, las modificaciones introducidas son fácilmente asimiladas. Concretamente, una FIFO circular tiene dos índices, de lectura y de escritura, que se incrementan cada vez que se realiza la operación correspondiente. Por lo tanto, en caso de que se escribieran cuatro vectores desde la memoria principal antes de que se terminara el procesamiento del primero, habría riesgo de que el primer vector fuera sobreescrito. Para impedirlo, este diseño utiliza un registro de salvaguarda: el flag f toma como referencia la primera posición de un vector y no avanza hasta que no expira su utilidad. El flag e se genera con el puntero de lectura, que se incrementa en el modo habitual. La modificación realizada reside en reiniciar el puntero de lectura al valor registrado al finalizar cada "pase" o "pasada". En el último "pase" de cada vector, sin embargo, el registro toma el valor del puntero. El hecho de intervenir el puntero impide a las herramientas de síntesis en FPGA aprovechar las plantillas de inferencia de FIFOs. Por ello, en realizaciones de la invención, en lugar de buffer se utiliza un scratchpad con la interfaz simplificada para evitar la gestión directa de ninguna dirección. El consumidor del scratchpad (es decir, cualquier módulo que lea la información del scratchpad) sólo tiene tres señales de control: RST o señal para reiniciar todos los punteros a cero; FB o señal para reiniciar el puntero de lectura al valor registrado; FW o señal para avanzar el registro al valor del puntero de lectura. En el núcleo IP de la presente invención, el módulo consumidor es el módulo maccs.

En cuanto al resto de bloques fifos 140_{b0}-140_{b9}, estos bloques muestran una alta localidad espacial, pero un comportamiento menos deseable en cuanto a localidad temporal. Esta característica no es deseable para grandes matrices, ya que cada bloque debe almacenar tantas filas completas como pasadas requiera el problema. Sin embargo, este inconveniente se ve compensado por el hecho de que la implementación es preferentemente una FIFO pura, es decir, un buffer clásico sin información sobre direcciones. Al tener un patrón de lectura lineal, en caso de no poder introducir todo el contenido, el controlador de la comunicación (f_ctrl (swi)) puede y debe repetir la secuencia de lectura para cada vector. El diseño detecta automáticamente esta situación a partir de la elección de los parámetros de configuración.

Con respecto al bloque swi 120 (sub-bloque intercon_si 320 y sub-bloque f_ctrl 3301-3311), la instancia f_ctrl de cada sub-bloque 3301-3311 tiene dos contadores encadenados correspondientes al número de vectores que se deben transferir y la longitud de éstos. En el caso del sub-bloque 3301 correspondiente al primer bloque fifos 140_i, el número de vectores equivale al número de productos matriz-vector consecutivos que se desean computar. En el resto de sub-bloques 3302-3311, el número de vectores es el número de pasadas que se deben realizan con el DSP correspondiente. Esto es posible porque el presente diseño utiliza la posición relativa de los bloques fifos 140_{b1}-140_{b9} (y por tanto, de los DSPs, y por extensión de los módulos de saturación) como codificación parcial de las direcciones. El componente encargado de decodificar la dirección local en base a la posición y los índices dados es el árbitro en el sub-bloque intercon_si 320 del bloque 120. La figura 9 muestra un esquema del componente principal (módulo si_adr) utilizado para obtener punteros de fila p_{ROW} y columna p_{COL} en el sub-bloque intercon_si 320, donde it indica si la transferencia corresponde a la memoria i o a cualquier otra, id es la posición en el array de la memoria activa (aquélla para la que se está decodificando la dirección) y x e y son los valores de los contadores dados por f_ctrl. En la figura 9, los bloques negros son registros; el bloque blanco es un multiplexor y el bloque “AND” es una puerta lógica. Las líneas continuas representan el datapath y la línea de puntos representa una señal de selección.

Volviendo a la figura 7, la operación de un sub-bloque f_ctrl 3301-3311 es la siguiente: En primer lugar, el bloque de control 185 escribe los límites de los contadores en sendos registros de f_ctrl. Cuando el sub-bloque f_ctrl 3311_j detecta el cambio, pone STALL a 0 para indicar que quiere recibir información. Nótese que STALL es una señal definida en el estándar Wishbone. CYC, GNT y STB, aludidas a continuación, son también señales definidas en dicho estándar. Asimismo, INC es una señal de usuario de acuerdo con dicho estándar. Cuando el árbitro selecciona uno de entre los sub-bloques 3301-3311 que esté requiriendo transferencia preferentemente aplicando el algoritmo de planificación round-robin, es decir, en orden desde el primero hasta el último y vuelta a empezar, pone CYC a 1. El árbitro mantiene INC a 1 durante dos ciclos, lo que incrementa los contadores del sub-bloque f_ctrl 3301-3311 y permite llenar si_adr (figura 9).El puntero de matriz se obtiene del módulo control 185, ya que es el mismo para todas las instancias del sub-bloque f_ctrl, excepto la del primer bloque fifo 140_i. El árbitro utiliza los punteros para requerir inicio de transferencia a la MAI 200.Cuando se recibe el GNT, con cada STB: Se guarda en una pequeña FIFO el tag (x e y) de la petición (propia del estándar Wishbone) realizada y se activa INC, para ir avanzando en el pipe al mismo

tiempo que se hacen peticiones. Lo que se pide concretamente es el contenido de las matrices a multiplicar, para guardarlo en los submódulos de fifos. Concretamente, una matriz se guarda repartida en 'b?' (módulos $140_{b0\text{-}b9}$), y la otra se carga sólo en 'i' (módulo 140_i). Cada ACK recibido de la MAI 200es transmitido directamente al sub-bloque f_ctrl junto con el tag correspondiente leído de la FIFO correspondiente. La transferencia finaliza si: el sub-bloque f_ctrl llega al final de su tarea y pone STALL a 1; o si el STALL proveniente de la MAI 200, a su vez de la memoria principal, se pone a 1; o si un contador de longitud de transferencia interno del árbitro llega al límite. Cuando finaliza la transferencia el árbitro pone CYC a 0, y el sub-bloque f_ctrl actualiza los contadores con los valores inmediatamente posteriores al último tag recibido.

Este mecanismo, en el que el árbitro del bloque intercon_si 320 hace de puente entre las instancias de los sub-bloques f_ctrl3301-3311y el árbitro dentro de la MAI 200, permite utilizar direcciones locales en cada sub-bloque f_ctrl 3301-3311, reduciendo así el tamaño de contadores y registros. Asimismo, se pueden establecer longitudes de transferencia independientes de la longitud de los vectores, y programables en tiempo de ejecución (incluso de forma dinámica). Esto último es especialmente interesante porque permite que la memorias en el bloque fifos 140 vayan llenándose más o menos al mismo ritmo, y evita que algunas hayan recibido todo su contenido mientras otras estén vacías. El ritmo al que estas memorias van llenándose es un parámetro configurable en tiempo de ejecución: puede ser desde 1 (descompensación nula prácticamente, típicamente cuando se limitan las transferencias a bloques pequeños) hasta decenas o cientos de elementos de diferencia (típicamente cuando se utilizan bloques grandes). Nótese que la descompensación no es necesariamente algo negativo, ya que puede resultar más eficiente leer las memorias en bloques mayores. Además, habitualmente se esperará a haber cargado todos los datos antes de iniciar la computación.

A alto nivel, el funcionamiento de los bloques swi 120 y control 185 con la MAI 200 se puede considerar un DMA (acceso directo a memoria, del inglés Direct Memory Access) que gestiona #DSP+1 tareas de forma concurrente (es decir, gestiona un número de tareas igual al número de DSP + 1). Lo que lo diferencia es la programación, ya que gracias a un análisis exhaustivo de las dependencias dimensionales, en la implementación de la presente invención el usuario define los movimientos en términos de matrices completas, y se ajustan automáticamente los registros y contadores intervinientes. En una DMA común se requiere la generación de una secuencia de instrucciones por tener que indicar la transferencia de cada vector. Por ejemplo, T. Chen et. Al (“DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-

Learning", SIGPLAN Not., vol. 49, nº 4, pp. 269-284, Feb. 2014) utilizan una FIFO anexa a cada DMA para precargar las instrucciones de forma que no se requiera intervención continua. En la presente invención, gracias al uso de contadores y la codificación posicional, no se necesitan memorias, ya que la secuencia es implícita.

Además de lo anterior, aunque en las figuras 7-8 se representa un solo puerto 101 entre el bloque intercon_si 320 y la MAI 200, el diseño permite utilizar varios: O bien dos, uno para el sub-bloque 3301 y otro para el resto 3302-3311; o bien uno para el sub-bloque 3301 y k puertos para el resto, donde k ≤ #DSP. En ese caso el puerto se convierte en un crossbar.

-Salida de datos (de los sub-bloques mmg (también llamados MUXa MUXb MUXc) 13a_out, 13b_out y 13c_out; de las funciones de activación; del bloque smerger_wp 170; y del bloque swo 190):

En los sub-bloques mmg (o MUXa, MUXb, MUXc), como los datos salen en escalera (salidas 13a_out, 13b_out y 13c_out), únicamente se utiliza un contador cuyo módulo máximo es igual al número de líneas que gestiona. A partir de un pulso indicando la disponibilidad del primer dato, los datos se leen secuencialmente y se transmiten en serie al módulo de activación 160. El módulo de activación 160 (sigmoids) está compuesto por múltiples instancias, que pueden incluir una o varias funciones de activación (circuitos CRI). Cuando se escribe el último dato de cada grupo, se emite un pulso para indicar al próximo sub-bloque mmg (MUXb) la disponibilidad de su primer elemento. De esta forma se optimiza la lógica de control, al no ser necesario que la máquina de estados principal supervise la secuenciación entre instancias de mmg.

En cuanto a las funciones de activación (bloque 160), por un lado, el número de funciones de activación se calcula automáticamente en tiempo de ejecución en función del número de puertos de salida 13a_out 13b_out 13c_out resultante de evaluar los parámetros globales. En realizaciones de la invención, se implementa una función de activación a la salida de cada acumulador 13a_out, 13b_out, 13c_out. En otras realizaciones de la invención, se implementa una sola función de activación para todos los acumuladores. Por otro lado, las funciones de activación se encuentran segmentadas y son pasivas, es decir, los datos entran y salen con una misma frecuencia y latencia constantes. Se puede escoger la función en tiempo de ejecución, pero en caso de utilizar la misma no se requiere intervención. Estas características hacen que el filtrado requiera pocos recursos lógicos y un tiempo de cálculo mínimo. Sin embargo, en el caso particular de que la RNA se utilice como clasificador o predictor con varios nodos de salida, a la

salida de las capas de clasificación se requiere un contador con un módulo máximo igual al del contador en el sub-bloque mmg correspondiente (MUX_a, MUX_b, MUX_c en la figura 7). Esto es debido a que, a diferencia del resto, el operador máximo aplicado a un vector de elementos es un compresor, por lo que requiere conocer cuántos elementos tiene el grupo a comprimir. No obstante, se trata de un parámetro que se calcula automáticamente: parcialmente en tiempo de síntesis y parcialmente en ejecución, a partir de la definición del problema dado por el usuario.

En caso de utilizar la variante que implementa los sub-bloques mmg como un registro de desplazamiento (ver figura 8), el contador del operador máximo es necesario, independientemente de que se use el operador o no. Asimismo, el Daisy chain de las señales de sincronización se transfiere a los sub-bloques sigmoid_wp 160a 160b 160c.

En cuanto al bloque smerger_wp 170, para poder escribir los resultados en la memoria principal en el orden requerido, el sub-bloque smerger171 debe leer las salidas de las FIFOs (que son bufferes) 170a 170b 170c en grupos del mismo tamaño que las agrupaciones a la salida 13a_out 13b_out 13c_out del bloque maccs 150. Para ello, utiliza dos contadores, uno con módulo máximo igual al tamaño del grupo, y otro con módulo máximo igual al número de filas de la matriz en el producto matriz-vector. Esto es debido a que el número de elementos del último grupo leído depende de la programación en tiempo de ejecución, por lo que una aproximación que optimice el tamaño del segundo contador requiere procesamiento adicional de algún parámetro. En el caso particular de que la RNA se utilice como clasificador, cuando se procesa la capa de salida de un clasificador, la utilización del operador máximo es opcional, ya que puede ser útil obtener el peso de todas las etiquetas de clasificación. En caso de que se utilice, sólo es necesario leer un dato de cada FIFO170a 170b 170c, porque han sido previamente comprimidos en el sub-bloque sigmoid_wp 160a 160b 160c. Al mismo tiempo, se utiliza un operador máximo para comprimir los resultados parciales a un único valor, que es escrito enla memoria o 180.Al igual que en el bloque intercon_si320, el diseño contempla la implementación del sub-bloque smerger 171 como un crossbar, de forma que se disponga de varias instancias de la memoria o 180.

Por último, en cuanto al bloque swo 190, al igual que el bloque swi 120, swo 190 utiliza un formato de dirección virtual compuesto por punteros a matriz, fila y columna. Sin embargo, a diferencia de a la entrada 120, durante la ejecución de un problema la matriz en swo 190es fija, ya que todos los vectores resultantes pertenecen al mismo conjunto. Otra diferencia notable es que no hay pasadas que decodificar, ya que llegan

serializadas. Por lo tanto, la implementación se resuelve con dos contadores, de fila y columna. La longitud de la columna se establece en tiempo de ejecución mediante la definición del problema en el bloque control 185. En el caso particular de una red de clasificación, cuando se utiliza el operador máximo en esta red, como ya se ha mencionado, el contador de filas columnas resulta innecesario y se mantiene en un valor fijo. En caso de implementar el sub-bloque smerger 171 como un crossbar, y disponer por tanto de varias memorias o 180, el bloque swo 190 se puede implementar como un multiplexor, o como otro crossbar con varias conexiones a la MAI 200. En caso de que el número de bloques swo190 sea igual al de memorias o 180, automáticamente se sustituye el crossbar por conexiones directas, ya que resulta ineficiente e innecesario el exceso de conexiones del crossbar.

Otro aspecto relevante en el bloque IP 1, 100, 100' de la invención, es el relativo a los dominios de reloj. En realizaciones de la invención, la arquitectura puede usar hasta tres dominios de reloj diferentes. Las realizaciones mostradas en las figuras 2, 7 y 8 utilizan dos dominios de reloj: uno para la lógica y la aritmética, y otro para las comunicaciones y transferencias de/a la memoria principal. En realizaciones de la invención, en las que el módulo de procesamiento 10 10' se utilice como subsistema en un sistema mayor, el bloque IP 1, 100, 100' usa preferentemente al menos un tercer dominio (además de los dos dominios de reloj citados). La figura 10 muestra cada uno de estos tres dominios de reloj. Un primer dominio 71, para los bloques maccs 150 y sigmoids 160. Un segundo dominio 72, para los bloques swi 120, smerger_wp 170 y swo 190. Y un tercer dominio 73 es el la matriz de interconexión común (AXI Interconect) con otros periféricos. En algunos dispositivos, como el Zynq de Xilinx, esta frecuencia (del tercer dominio de reloj) está fijada. La figura 10 representa un ejemplo de arquitectura de sistema (SoC), en la que puede usarse el bloque IP de la invención. Este ejemplo de SoC incluye, además del bloque IP 1, 100, 100' de la invención, otros bloques (AXI-Interconnect, CPU, DDR Cont., DDR3, UART, PC). Nótese que en la figura 10, a diferencia de la figura 7, el sub-bloque smerger170es un crossbar, por lo que hay múltiples instancias de FIFO o 190. Aunque un número relativamente tan elevado con respecto al número de DSP (#DSP), no es óptimo en una implementación práctica, en la figura 10se emplea para ilustrar que todos los cambios de dominio se realizan a través de FIFOs de doble reloj, aprovechando así los mismos buffer o scratchpad. De esta forma se simplifica la lógica requerida para sincronizar dominios. El único componente en el diseño del bloque IP 1, 100, 100' que utiliza ambos dominios de reloj es el bloque control 185, ya que debe escribir en los registros de configuración internos de todos los módulos en tiempo de ejecución.

Como se ha mencionado anteriormente, el bloque IP 1, 100, 100’ descrito con referencia a las figuras 1-10 está especialmente diseñado para integrarse en la arquitectura de un sistema-en-un-chip (SoC) o en la arquitectura de una FPGA. El bloque IP 1, 100, 100’ es especialmente compatible con las características del bus AXI. A modo de ejemplo, sin carácter limitativo, cuando el bloque IP 1, 100, 100’ se ha diseñado para integrarse en una FPGA, una vez que el bloque IP 1, 100, 100’ se ha sintetizado con una cierta configuración, se ha hecho el “place and route” (que es parte del proceso de implementación de un diseño descrito en HDL; las herramientas EDA primero sintetizan el código extrayendo un netlist genérico, pero luego hay que colocar espacialmente cada elemento o componente (place) y las interconexiones entre ellos (*route*) en la tecnología final de implementación (chip)) y se ha configurado la FPGA, pueden procesarse en tiempo de ejecución problemas de cualquier tamaño, siempre que no se superen los valores de síntesis (número de entradas y número de salidas). Para ello, a través de una subred de control dentro de la MAI 200, se escriben en el módulo control 185una serie de registros (v, f, h, c, véase figura 1). Adicionalmente se requiere la escritura de varios registros más (en realizaciones de la invención, entre siete y diez registros más), en función del modelo de red a computar. Una vez iniciada la ejecución, es completamente desatendida (autónoma) y cualquier procesador externo puede realizar otras operaciones hasta que hayan terminado todos los productos consecutivos programados. Asimismo, a fin de evitar la inclusión de módulos con un solo uso, se espera que se pre-computen cuatro parámetros derivados de las relaciones entre h y #DSP (número de DSP), por un lado, y entre c y #DSP, por otro. Por último, se deben escribir los tres punteros de matriz que se utilizan en cada producto matriz-vector: los dos de entrada y la de resultado. Estos punteros son identificadores virtuales. Son direcciones no absolutas. Son unos pocos bits (típicamente de entre 2 y 7, dependiendo del número de matrices a gestionar) que se utilizan en la MAI para obtener la dirección absoluta completa (típicamente 32 bits). Alternativamente, si se desea prescindir de la MAI, estos identificadores pueden representar el offset de cada matriz en una memoria compartida, considerándolos los bits más significativos (MSBs) de la dirección absoluta. En caso de ejecutar un modelo SLFN, que realiza dos operaciones consecutivas de forma automática, se requiere la escritura de otros tres punteros. Finalmente, se inicia el procesamiento con la escritura de un valor que representa la secuencia deseada.

El módulo de procesamiento de la presente invención, así como la arquitectura basada en SoC que incluya dicho módulo de procesamiento, tienen especial aplicación en la computación de redes neuronales.

En suma, el núcleo IP de la presente divulgación se adecúa de forma automática al modelo de red seleccionado en dos fases: en síntesis, por el “plegado” de las capas para reutilizar los recursos hardware de forma optimizada (y la consiguiente adecuación del control y flujo de datos), y en tiempo de ejecución (una vez implementado el procesador) mediante el uso de unaserie de registros de configuración programables, lo que permite ajustar el tamaño de la capa a procesar y el tipo de función de activación, escribiendo en dichos registros.

Durante la síntesis, mientras propuestas como la de DianNao optimizan cada producto vectorial utilizando un árbol de sumadores y múltiples multiplicadores en paralelo, el diseño de la presente divulgación realiza cada producto vectorial de forma secuencial (operación MACC en un DSP), y explota las posibilidades de paralelización en el 'bucle exterior de un producto matricial. En otras palabras, DianNao optimiza la computación de cada neurona y el presente bloque IP optimiza la computación de varias neuronas en paralelo. Además, como DianNao no computa múltiples neuronas en paralelo, el concepto de plegado (folding) no se considera.

Durante la ejecución, frente a procesadores neuronales convencionales, que tienen una estructura fija, por lo que pueden ser configurables (en síntesis) pero no programables, en el diseño de la presente divulgación sí se procesa efectivamente una red menor en caso de desactivación de algunas neuronas, de manera que el tiempo de computación se reduce. Más aún, los recursos hardware necesarios son prácticamente independientes del tamaño de las capas, siendo el número de DSPs el indicador principal. En otras palabras, utilizando el mismo número de DSPs, el diseño de la invención tiene requisitos similares para capas con vectores de decenas, cientos o miles de elementos.

En suma, la presente invención proporciona un procesador (computador) configurable y programable de productos matriciales (en posibles realizaciones, grandes) en coma fija. El procesador posee unas características particulares (las funciones de activación, etc.) que lo hacen especialmente interesante para el procesamiento neuronal, aunque en su fase de configuración (pre-síntesis) estas características pueden eliminarse para usarse como multiplicador matricial simple. En este sentido, el procesador es un acelerador de transformaciones lineales, que puede tener amplia aplicabilidad, al que se le añaden transformaciones no-lineales (funciones de activación) para operar como red neuronal. Es relevante el hecho de que sea programable. Se ha primado la reutilización de un solo núcleo IP, de modo que es altamente configurable/escalable, portable y reusable para la computación de múltiples problemas (modelos de red) de diferente tamaño y

configuración, en lugar de buscar la configuración óptima para un abanico más reducido. Esto es también consecuencia de cómo se ha escrito el código VHDL. Además al ser un array sistólico lineal, requiere muchos menos recursos que una solución basada en un array sistólico rectangular. Tampoco se descarta la aplicabilidad a redes recurrentes (no feedforward), en las que se trata de realimentar algunas o todas las salidas de una capa a sus entradas.

En este texto, la palabra “comprende” y sus variantes (como “comprendiendo”, etc.) no deben interpretarse de forma excluyente, es decir, no excluyen la posibilidad de que lo descrito incluya otros elementos, pasos etc. Por otra parte, la invención no está limitada a las realizaciones concretas que se han descrito sino abarca también, por ejemplo, las variantes que pueden ser realizadas por el experto medio en la materia (por ejemplo, en cuanto a la elección de materiales, dimensiones, componentes, configuración, etc.), dentro de lo que se desprende de las reivindicaciones.

REIVINDICACIONES

1.- Un núcleo IP (1, 100, 100') configurable y programable de procesamiento para la computación de una pluralidad de productos matriciales, en el que tanto los datos a procesar como los resultados obtenidos se transfieren en serie, que comprende:

un bloque de entrada de datos (12, 120) configurado para proporcionar, a partir de unos datos de entrada, un conjunto de vectores que representan una primera y una segunda matriz cuyo producto se quiere computar, utilizando un formato de dirección virtual compuesto por punteros a matriz, fila y columna, donde dicho bloque de entrada de datos (120) comprende:

un primer sub-bloque (32, 320) configurado para obtener un puntero de fila (p_{ROW}) y un puntero de columna (p_{COL}); y

un segundo sub-bloque (332; 3301-3311) que comprende N componentes (3301-3311), donde N es un número natural > 1, cada uno de los cuales comprende dos contadores encadenados correspondientes al número de vectores a transferir y a la longitud de dichos vectores, donde cada componente (3301-3311) utiliza direcciones locales;

un bloque de memoria (14, 140) que comprende N elementos de memoria (141-145; 140_i, 140_{b0} …140_{b9}), estando cada uno de dichos elementos de memoria asociado a una salida respectiva de dicho segundo sub-bloque (32; 3301-3311) del bloque de entrada de datos (120);

un bloque multiplicador matriz-vector en coma fija (15; 150) configurado para implementar una operación de multiplicación-acumulación para multiplicar una matriz por múltiples vectores recibidos en serie de forma continua, donde dicho bloque multiplicador matriz-vector en coma fija (15; 150) comprende un conjunto de sub-bloques (150a, 150b, 150c), donde cada uno de dichos sub-bloques comprende una pluralidad de módulos aritméticos (DSP_0-DSP_3; DSP_4-DSP_7 ; DSP_8-DSP_9).

un bloque (16, 160) que comprende al menos una función de activación configurada para ser aplicada a la salida de dicho bloque multiplicador matriz-vector en coma fija (15, 150);

un bloque (17; 170, 180) para almacenar en componentes de almacenamiento (170a, 170b, 170c) las salidas de la al menos una función de activación y para leer (171) las salidas de dichos componentes de almacenamiento (170a, 170b, 170c); y

un bloque de salida de datos (18, 190) que utiliza un formato de dirección virtual

compuesto por punteros a matriz, fila y columna, que comprende un contador de fila y un contador de columna.

2.- El núcleo IP (1, 100, 100') de la reivindicación 1, donde el primer componente (3301)de dicho segundo sub-bloque está configurado para proporcionar un número de vectores igual al número de productos matriz-vector consecutivos que se desean computar.

3.- El núcleo IP (1, 100, 100') de cualquiera de las reivindicaciones anteriores, donde los componentes segundo a último (3302-3311) de dicho segundo sub-bloque están configurados para proporcionar un número de vectores igual al número de pasadas que se deben realizan con el DSP correspondiente.

4.- El núcleo IP (1, 100, 100') de cualquiera de las reivindicaciones anteriores, donde dicho bloque multiplicador matriz-vector en coma fija (150) está basado en un array sistólico lineal con carga de parámetros en paralelo y ejecución tipo ola.

5.- El núcleo IP (1, 100, 100') de cualquiera de las reivindicaciones anteriores, donde dichos N elementos de memoria (140_i, 140_{b0} …140_{b9}) comprendidos en dicho bloque de memoria (140) son N bloques BRAM.

6.- El núcleo IP (1, 100, 100') de cualquiera de las reivindicaciones anteriores, donde cada sub-bloque (150a, 150b, 150c) de dicho bloque multiplicador matriz-vector en coma fija (150) comprende un multiplexor (MUXa, MUXb, MUXc) a su salida.

7.- El núcleo IP (1, 100, 100') de cualquiera de las reivindicaciones 1 a 5, donde cada sub-bloque (150a, 150b, 150c) de dicho bloque multiplicador matriz-vector en coma fija (150) comprende a su salida tantos registros de desplazamiento (R'_0…R'_9) como módulos aritméticos tiene cada sub-bloque.

8.- El núcleo IP (1, 100, 100') de cualquiera de las reivindicaciones anteriores, donde dichos módulos aritméticos (DSP_0-DSP_3; DSP_4-DSP_7; DSP_8-DSP_9) operando en paralelo generan, cada z ciclos, tantos datos como número de módulos aritméticos haya, donde z es la longitud del vector.

9.- El núcleo IP (1, 100, 100') de cualquiera de las reivindicaciones anteriores, donde la ejecución en paralelo de dichos módulos aritméticos (DSP_0-DSP_3; DSP_4-DSP_7; DSP_8-DSP_9) se controla mediante una máquina de estados que toma como referencia sólo el primer módulo aritmético (DSP_0).

10.- El núcleo IP (1, 100, 100') de la reivindicación 9, donde dicha máquina de estados

utiliza tres contadores de apoyo: longitud del vector, número de repeticiones y latencia del módulo aritmético.

11.- El núcleo IP (1, 100, 100') de cualquiera de las reivindicaciones anteriores, donde dicho bloque multiplicador matriz-vector en coma fija (150) representa al menos una capa oculta de una red neuronal artificial.

12.- El núcleo IP (1, 100, 100') de cualquiera de las reivindicaciones anteriores, que comprende medios para realimentar la salida de dicho bloque FIFO (180) para efectuar al menos dos operaciones matriz-vector consecutivas con un filtrado intermedio, de forma que con una sola instancia del bloque multiplicador matriz-vector en coma fija (150) se procesa tanto la al menos una capa oculta como la capa de salida de la red neuronal.

13.- El núcleo IP (1, 100, 100') de cualquiera de las reivindicaciones anteriores, donde se utilizan M módulos aritméticos (DSP_0-DSP_9) para efectuar h multiplicaciones vectoriales, donde h es el número de filas de la matriz en el producto matriz-vector, por lo que si h >M, se requiere más de una pasada, iteración o repetición para procesar cada vector de entrada.

14.- El núcleo IP (1, 100, 100') de cualquiera de las reivindicaciones anteriores, que comprende además un núcleo de interconexión (200) configurado para facilitar la integración del núcleo IP (1, 100, 100') en un sistema heterogéneo con uno o varios módulos coprocesadores, estando dicho núcleo de interconexión (200) configurado para gestionar bloques de memoria internos y externos al núcleo IP (1, 100, 100').

15.- El núcleo IP (1, 100, 100') de cualquiera de las reivindicaciones anteriores, en el que todas las operaciones se realizan en coma fija de precisión ajustable, estando configurado para definir el tamaño de palabra utilizado en cada una de las matrices que intervienen en cada ejecución.

16.- Un sistema en chip (SoC) que comprende al menos un núcleo IP (1, 100, 100') de acuerdo con cualquiera de las reivindicaciones anteriores.

17.- Una FPGA que comprende al menos un núcleo IP (1, 100, 100') de acuerdo con cualquiera de las reivindicaciones 1-15.

18.- Un procedimiento de diseño de un núcleo IP (1, 100, 100') de acuerdo con cualquiera de las reivindicaciones 1-15, adecuado para una tecnología objetivo, que comprende:

generar un netlist que comprende una descripción parametrizada del núcleo IP (1, 100, 100') adecuado para dicha tecnología objetivo;

sintetizar una red neuronal que se desee implementar, adecuando el núcleo IP (1, 100, 100') a los recursos disponibles en dicha tecnología objetivo, donde dicha adecuación se realiza mediante una técnica de plegado y reutilización de capas neuronales;

una vez sintetizado cierto tamaño de red neuronal, seleccionar en tiempo de ejecución un número de neuronas que se desea utilizar en cada capa de red.

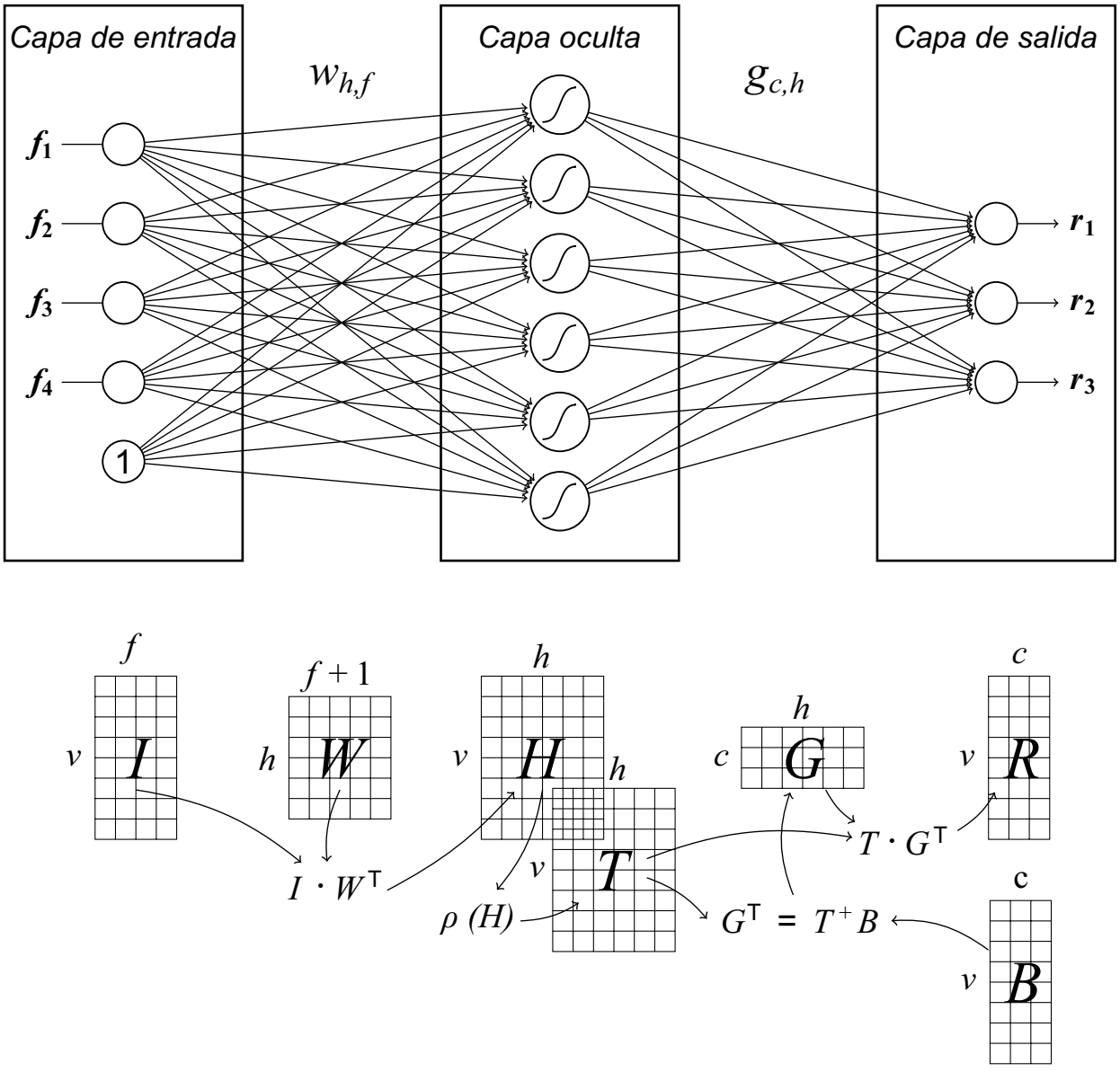
Capa de entrada
Capa oculta
Capa de salida
$w_{h,f}$
$g_{c,h}$
f_1
f_2
f_3
f_4
1
r_1
r_2
r_3
f
$f+1$
h
c
v
I
h
W
v
H
h
h
c
G
v
R
$I \cdot W^{\mathsf{T}}$
v
T
$T \cdot G^{\mathsf{T}}$
$\rho\ (H)$
$G^{\mathsf{T}} = T^{+} B$
c
v
B

FIG. 1

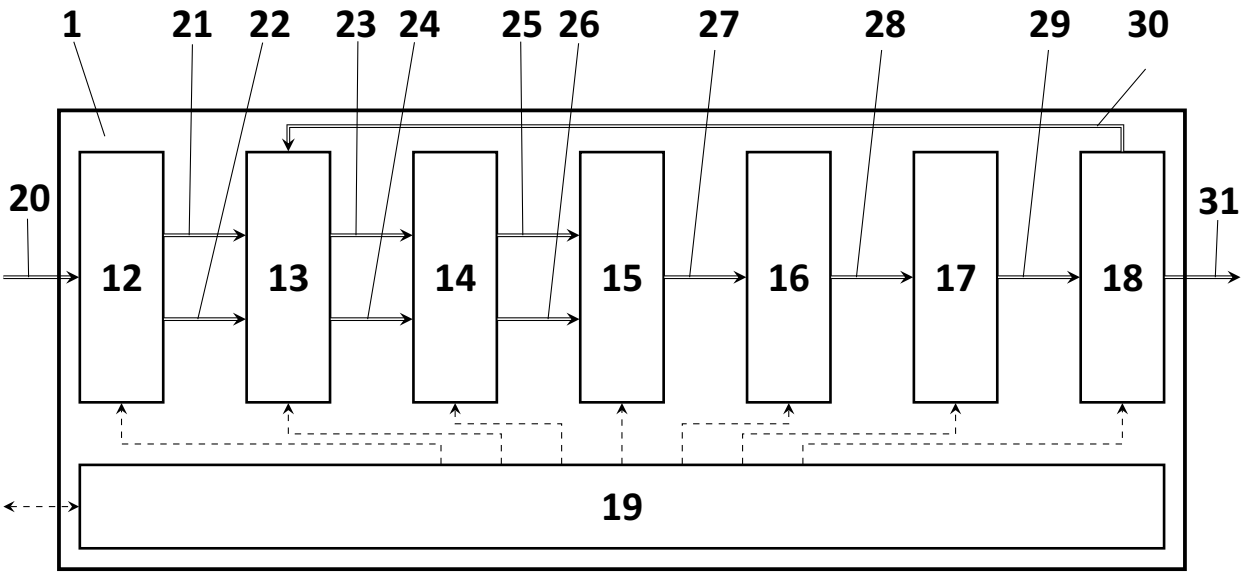
1
21
22
23
24
25
26
27
28
29
30
20
31
12
13
14
15
16
17
18
19

FIG. 2

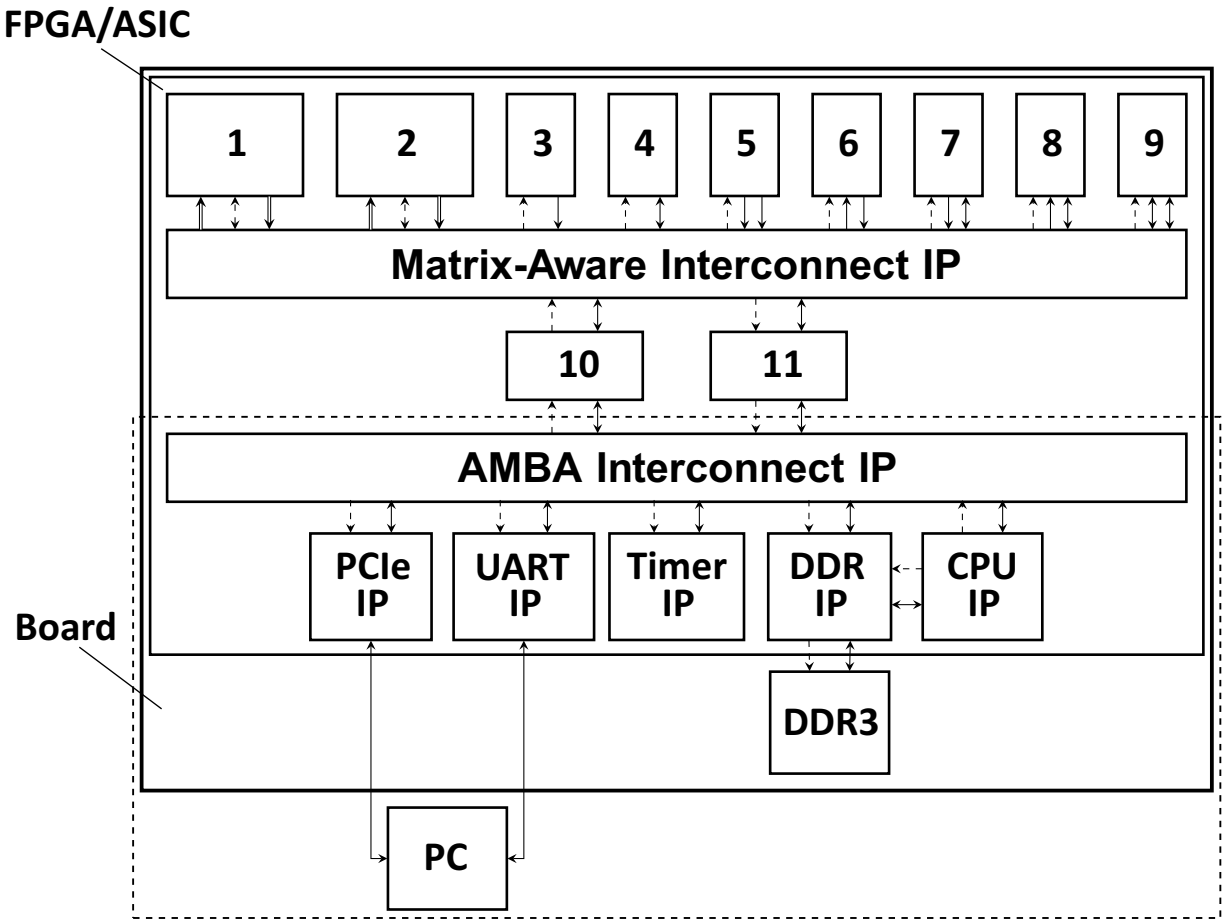
FPGA/ASIC
1
2
3
4
5
6
7
8
9
Matrix-Aware Interconnect IP
10
11
AMBA Interconnect IP
PCIe IP
UART IP
Timer IP
DDR IP
CPU IP
Board
DDR3
PC

FIG. 3

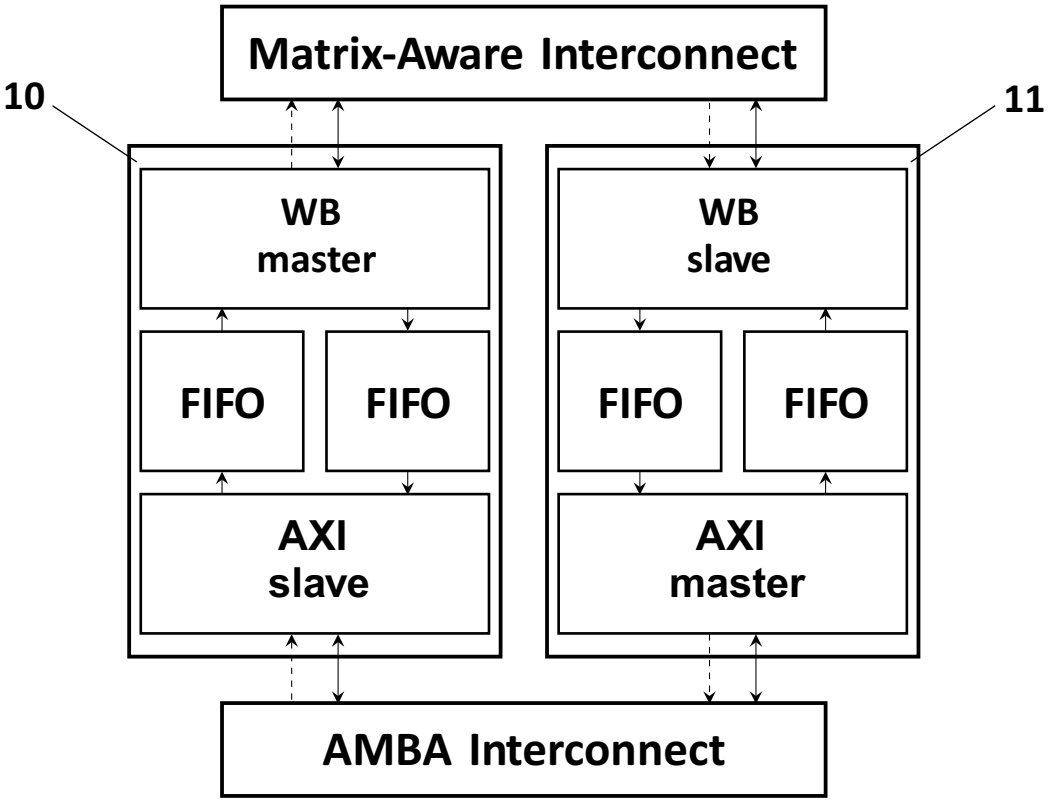
Matrix-Aware Interconnect
10
11
WB
master
WB
slave
FIFO
FIFO
FIFO
FIFO
AXI
slave
AXI
master
AMBA Interconnect

FIG. 4

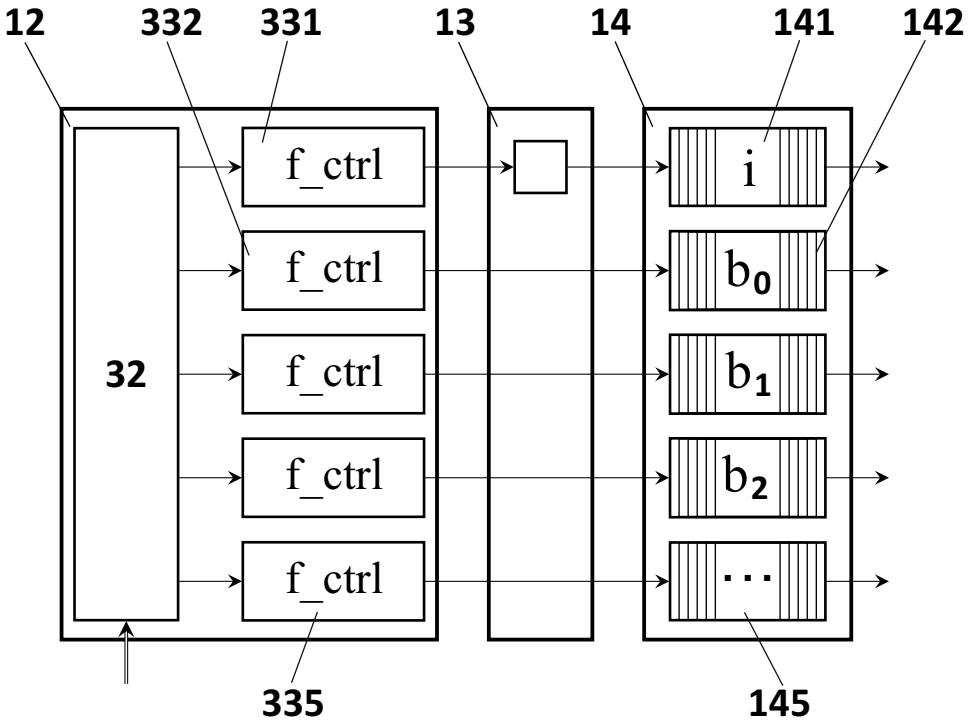
12
332
331
13
14
141
142
32
f_ctrl
f_ctrl
f_ctrl
f_ctrl
f_ctrl
i
b0
b1
b2
...
335
145

FIG. 5

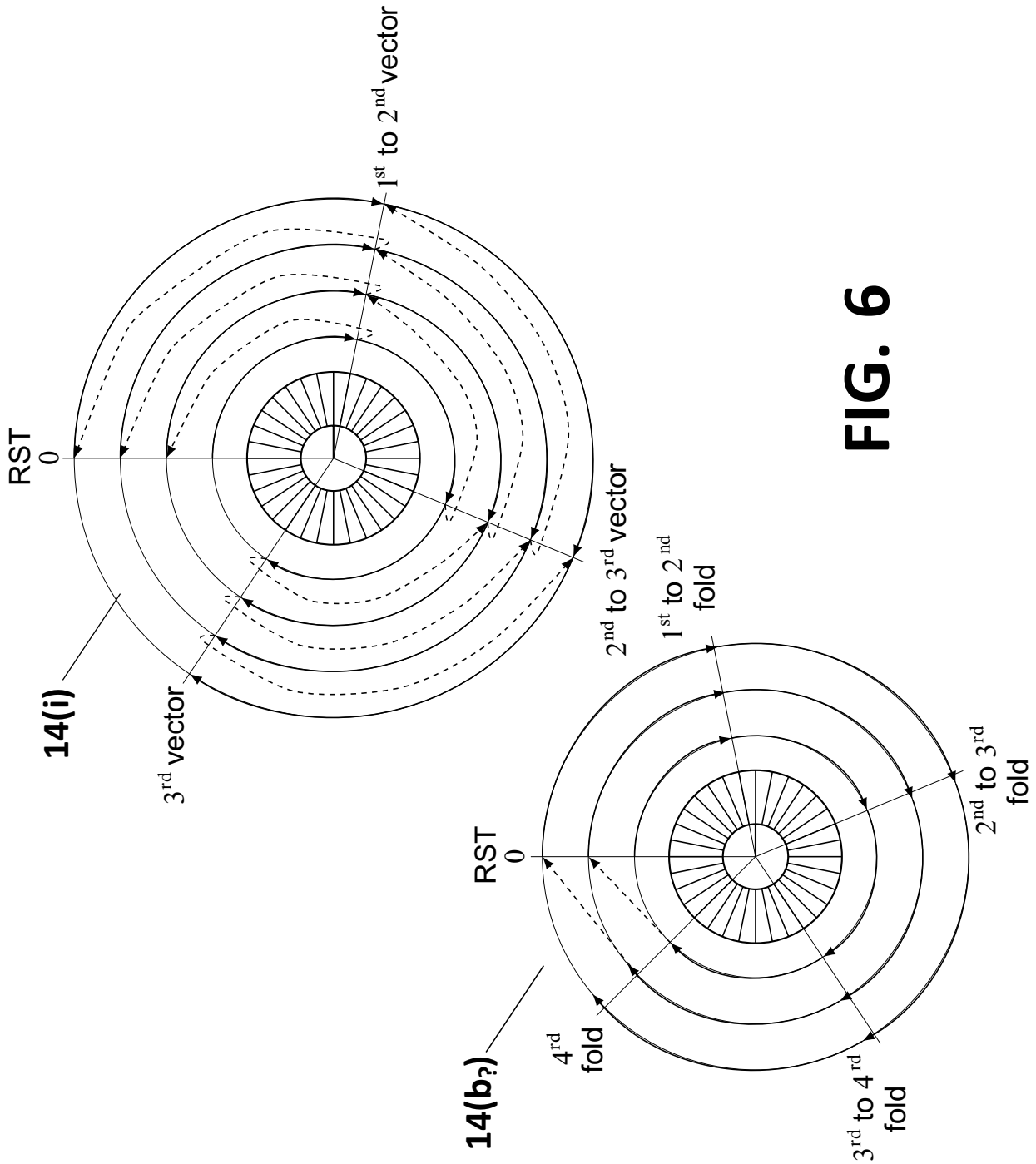
RST
0
1st to 2nd vector
14(i)
3rd vector
RST
0
2nd to 3rd vector
1st to 2nd fold
14(b2)
4rd fold
2nd to 3rd fold
3rd to 4rd fold

FIG. 6

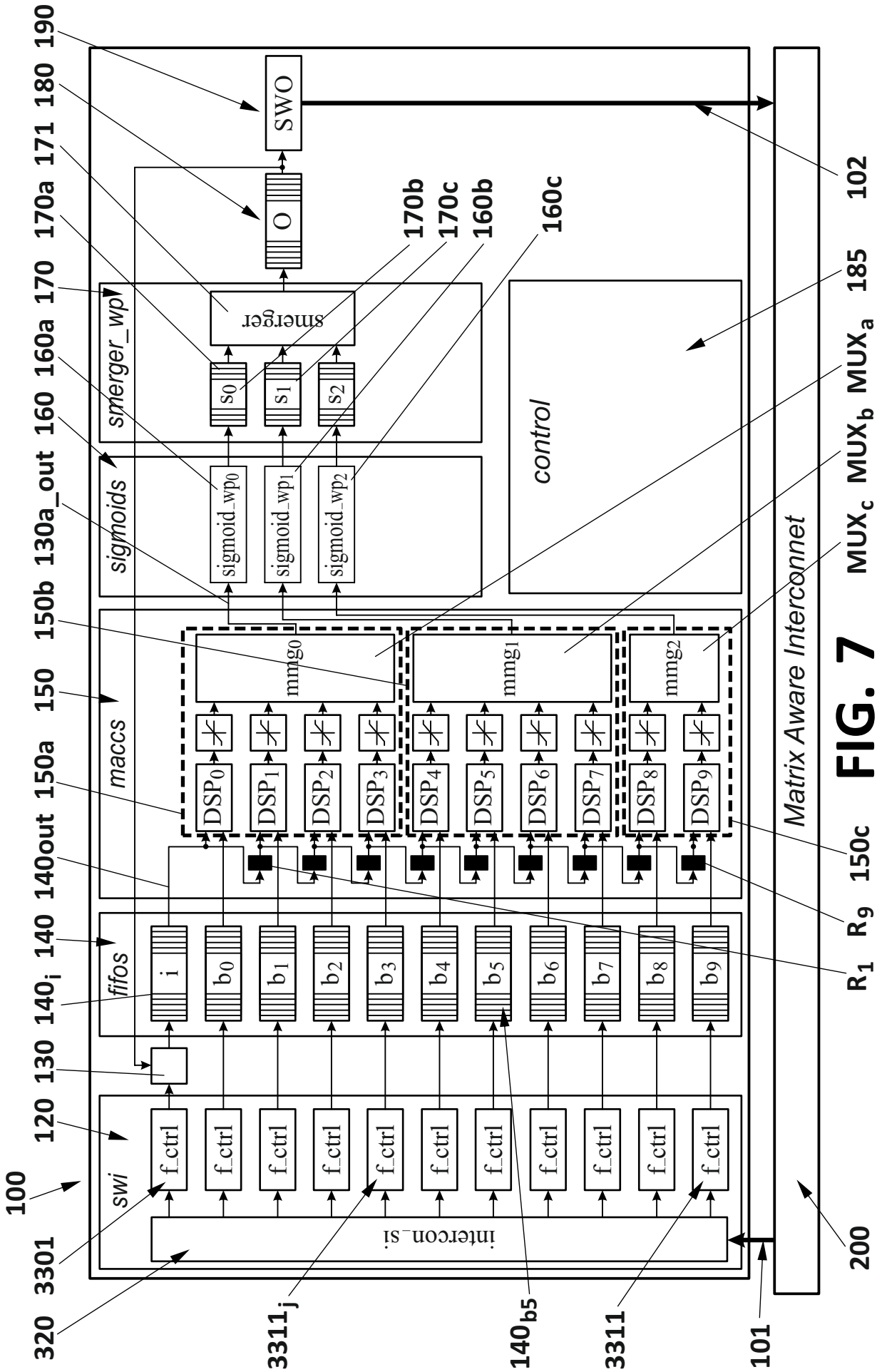
100
120
130
140
140i
140out
150
150a
150b
150c
160
160a
130a_out
170
170a
171
180
190
102
185
swi
fifos
maccs
sigmoids
smerger_wp
control
Matrix Aware Interconnet
f_ctrl
intercon_si
i
b0
b1
b2
b3
b4
b5
b6
b7
b8
b9
DSP0
DSP1
DSP2
DSP3
DSP4
DSP5
DSP6
DSP7
DSP8
DSP9
mmg0
mmg1
mmg2
sigmoid_wp0
sigmoid_wp1
sigmoid_wp2
s0
s1
s2
smerger
O
SWO
170b
170c
160b
160c
MUXa
MUXb
MUXc
R1
R9
320
3301
3311j
140b5
3311
101
200

FIG. 7

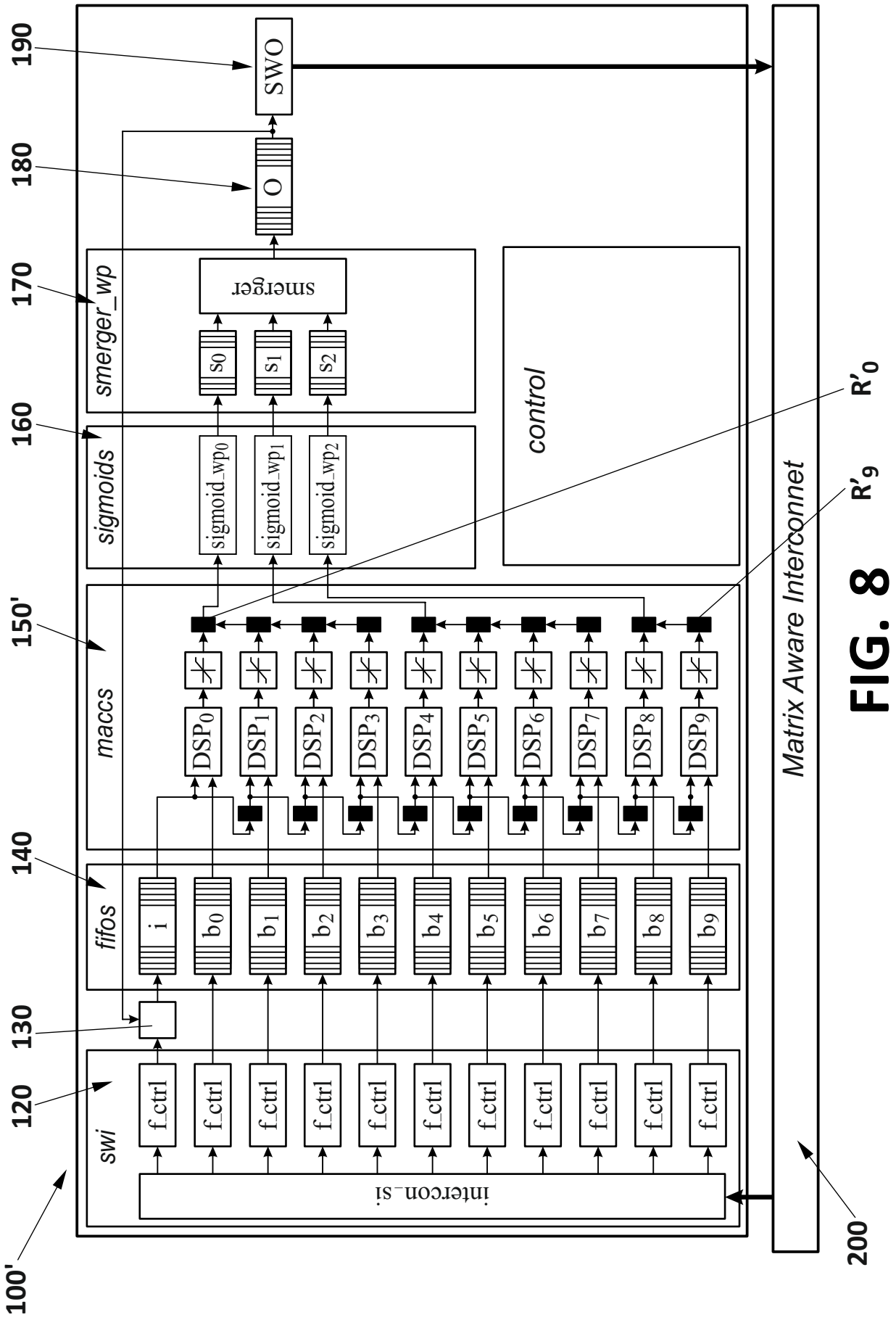
100'
120
130
140
150'
160
170
180
190
200
swi
f_ctrl
interconn_si
fifos
i
b0
b1
b2
b3
b4
b5
b6
b7
b8
b9
maccs
DSP0
DSP1
DSP2
DSP3
DSP4
DSP5
DSP6
DSP7
DSP8
DSP9
sigmoids
sigmoid_wp0
sigmoid_wp1
sigmoid_wp2
smerger_wp
s0
s1
s2
smerger
O
SWO
control
Matrix Aware Interconnet
R'0
R'9

FIG. 8

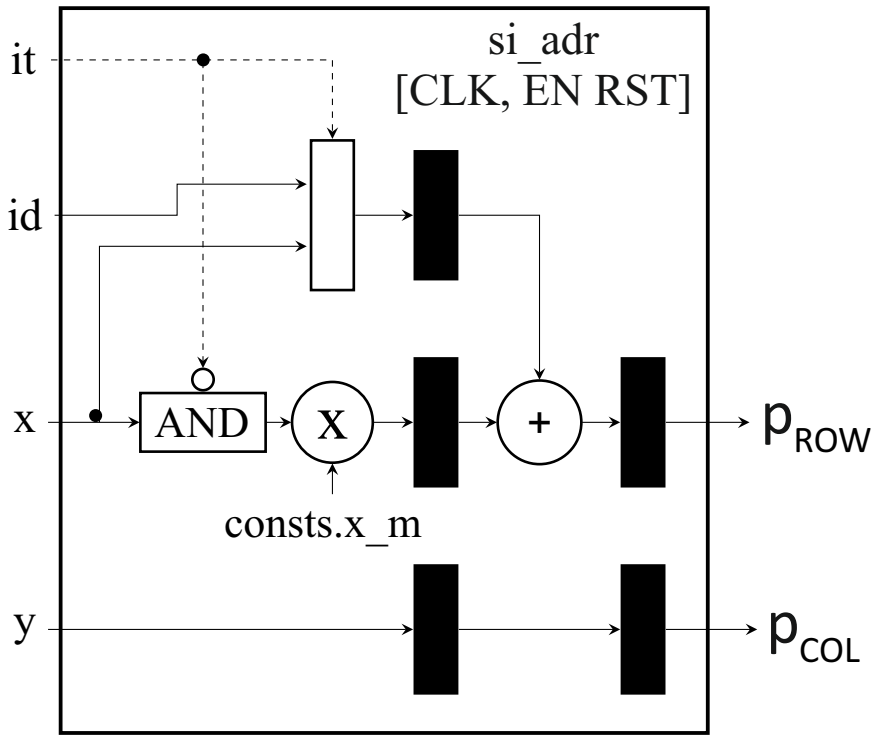

FIG. 9

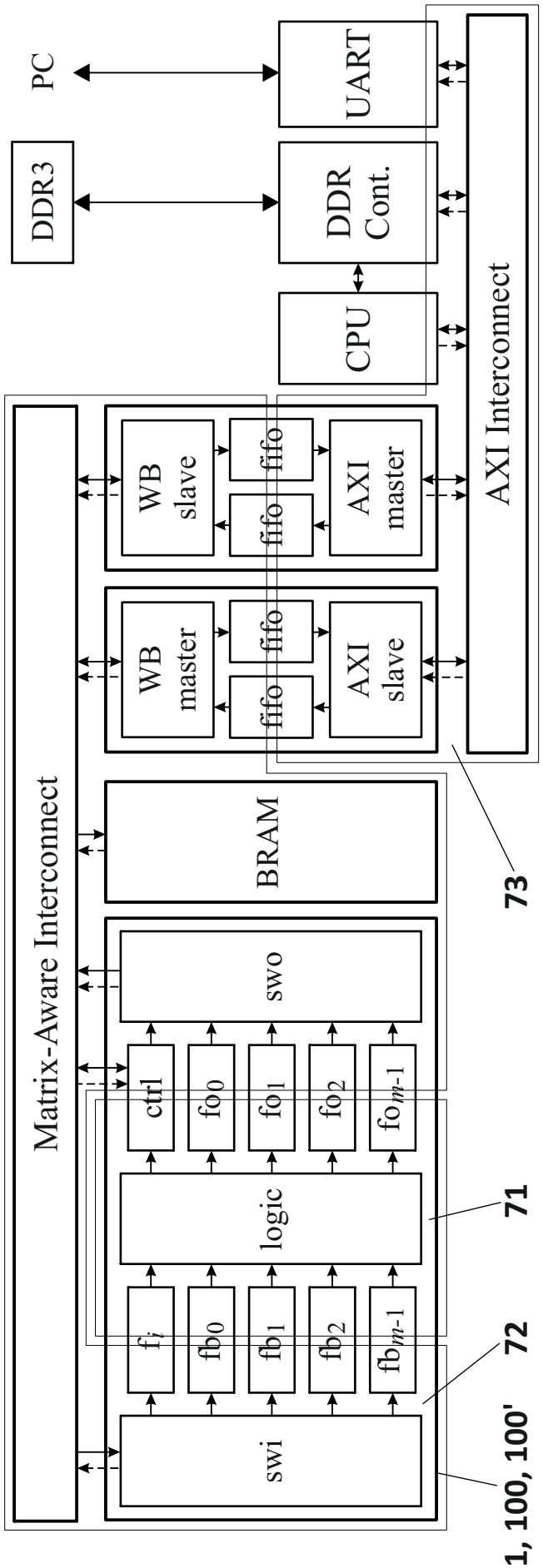
PC
DDR3
UART
DDR Cont.
CPU
AXI Interconnect
WB slave
fifo
fifo
AXI master
WB master
fifo
fifo
AXI slave
Matrix-Aware Interconnect
BRAM
swo
ctrl
fo0
fo1
fo2
fom-1
logic
fi
fb0
fb1
fb2
fbm-1
swi
73
71
72
1, 100, 100'

FIG. 10

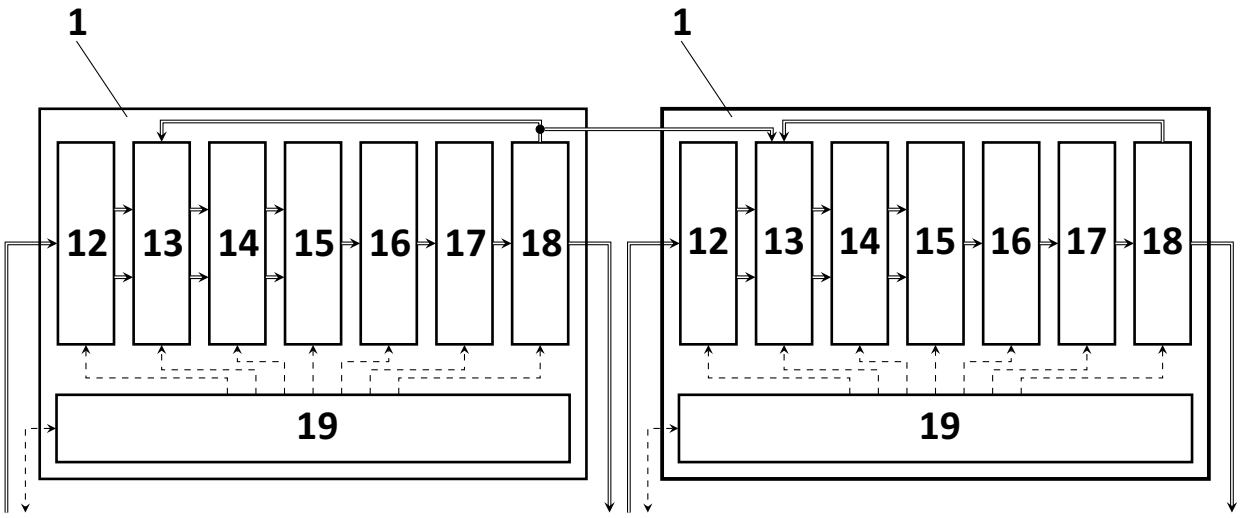
1
1
12
13
14
15
16
17
18
12
13
14
15
16
17
18
19
19

FIG. 11

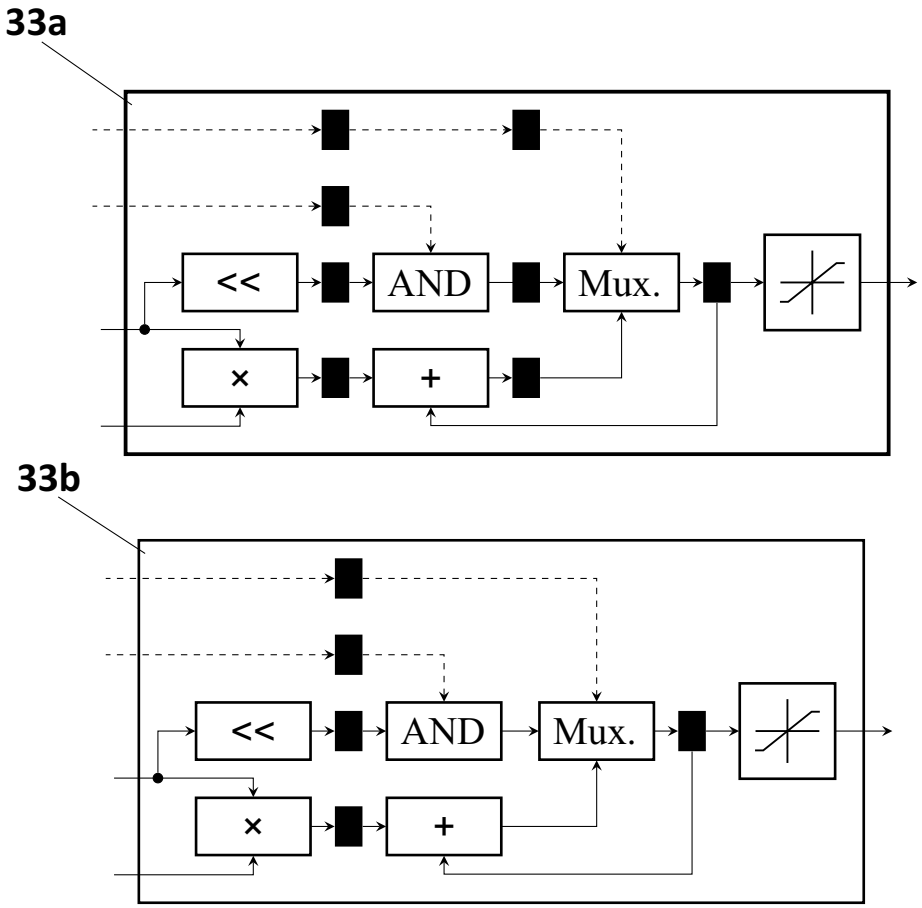
33a
<<
AND
Mux.
×
+
33b
<<
AND
Mux.
×
+

FIG. 12

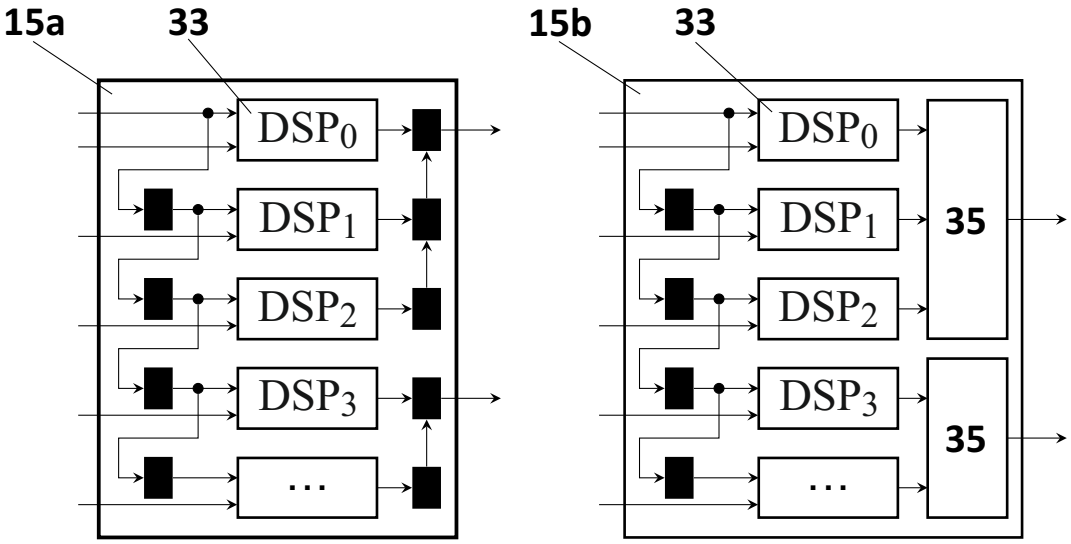
15a
33
DSP0
DSP1
DSP2
DSP3
...
15b
33
DSP0
DSP1
DSP2
DSP3
...
35
35

FIG. 13

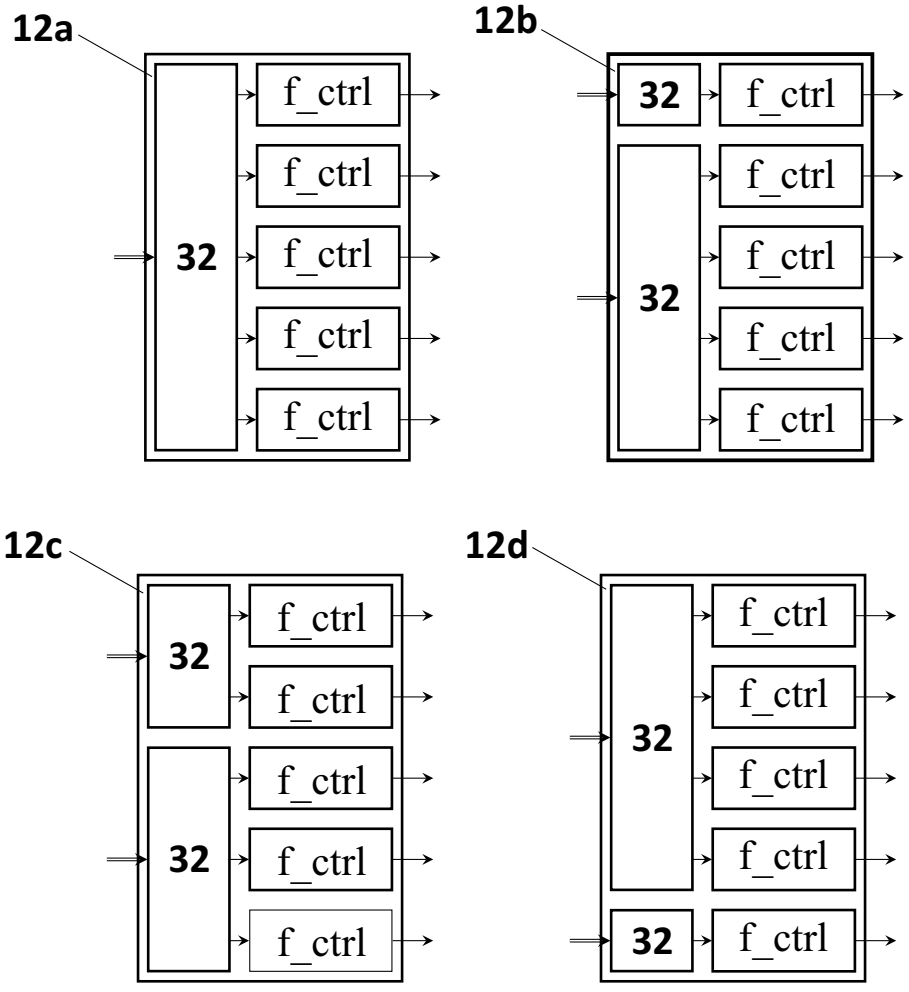
12a
32
f_ctrl
f_ctrl
f_ctrl
f_ctrl
f_ctrl
12b
32
f_ctrl
f_ctrl
f_ctrl
f_ctrl
f_ctrl
32
12c
32
f_ctrl
f_ctrl
32
f_ctrl
f_ctrl
f_ctrl
12d
32
f_ctrl
f_ctrl
f_ctrl
f_ctrl
f_ctrl
32

FIG. 14

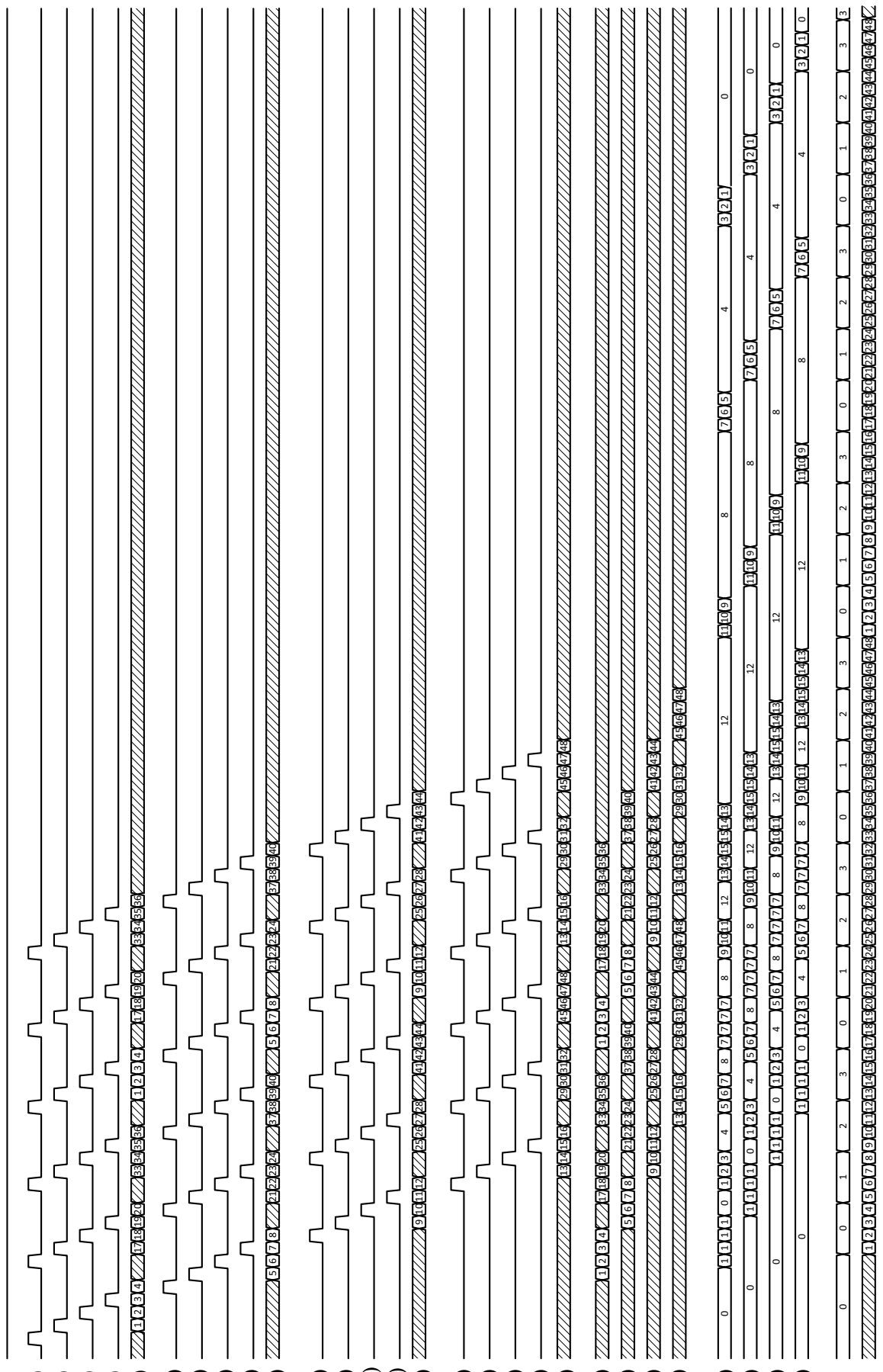
m_o (0)
m_o (1)
m_o (2)
m_o (3)
q (0)
m_o (4)
m_o (5)
m_o (6)
m_o (7)
q (1)
m_o (8)
m_o (9)
m_o (10)
m_o (11)
q (2)
m_o (12)
m_o (13)
m_o (14)
m_o (15)
q (3)
a (0)
a (1)
a (2)
a (3)
fa (0)
fa (1)
fa (2)
fa (3)
s
q

FIG. 15

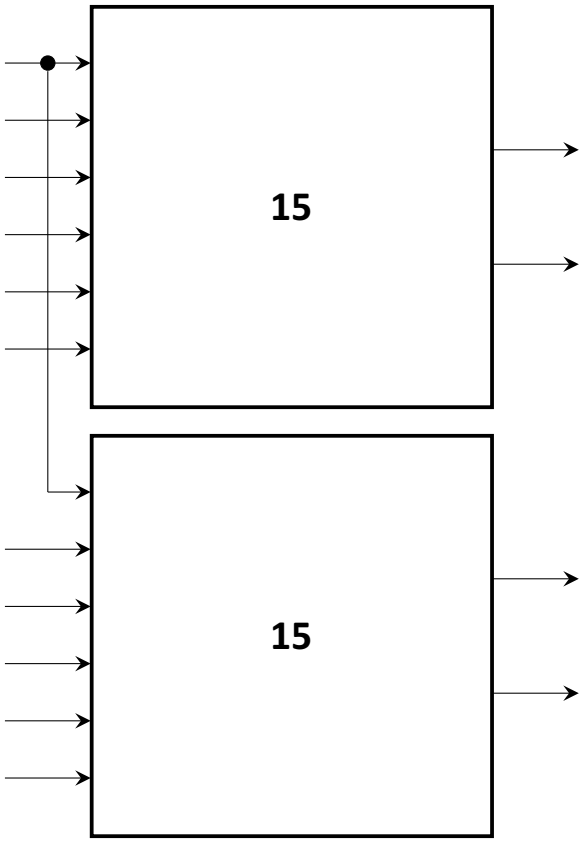

FIG. 16

OFICINA ESPAÑOLA
DE PATENTES Y MARCAS

ESPAÑA

(21) N.º solicitud: 201730963

(22) Fecha de presentación de la solicitud: 24.07.2017

(32) Fecha de prioridad:

INFORME SOBRE EL ESTADO DE LA TECNICA

(51) **Int. Cl. :** ***G06N3/00*** (2006.01)
G06F17/16 (2006.01)

DOCUMENTOS RELEVANTES

Categoría	**(56) Documentos citados**	**Reivindicaciones afectadas**
X	US 2014289445 A1 (SAVICH ANTONY) 25/09/2014, Resumen (Epodoc). Descripción: párrafo. 54, 93	1-18
A	THEOCHARIDES T et al. A generic reconfigurable neural network architecture implemented as a network on chip. SOC Conference, 2004. Proceedings. IEEE International Santa Clara, CA, USA Sept. 12-15, 2004, 20040912 - 20040915 Piscataway, NJ, USA, IEEE. , 12/09/2004, Páginas 191 - 194, ISSN ISBN 978-0-7803-8445-3; ISBN 0-7803-8445-8, <DOI: 10.1109/SOCC.2004.1362404>. Todo el documento.	1-18
A	STREY A et al. A configurable parallel neurocomputer. Artificial Neural Networks and Expert Systems, 1995. Proceedings., Sec ond New Zealand International Two-Stream Conference on Dunedin, New Zealand 20-23 Nov. 1, 19951120 - 19951123 Los Alamitos, CA, USA,IEEE Comput. Soc, US. Kasabov N K; Coghill G, 20/11/1995, Páginas 55 - 58, ISSN ISBN 978-0-8186-7174-6 , <DOI: 10.1109/ANNES.1995.499438>. Todo el documento.	1-18
A	MOREAU THIERRY et al. SNNAP: Approximate computing on programmable SoCs via neural acceleration. 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), 20150207 IEEE. , 07/02/2015, Páginas 603 - 614, <DOI: 10.1109/HPCA.2015.7056066>. Todo el documento.	1-18
A	HUDEC J et al. Design of the neural processor optimized for XILINX VIRTEX FPGA devices. Information Technology Interfaces, 2001. ITI 2001. Proceedings of the 23rd International Conference on June 19-22, 20010619 - 20010622 Piscataway, NJ, USA, IEEE. , 19/06/2001, Páginas A_7 - A_8, ISSN ISBN 978-953-96769-3-1; ISBN 953-96769-3-2. Todo el documento.	1-18
A	FARMER et al. A Rosetta stone for connectionism. PHYSICA D, 19900601 NORTH-HOLLAND, AMSTERDAM, NL. Kolokolnikov Theodore; Bertozzi Andrea; Fetecau Razvan; Lewis Mark, 01/06/1990, Vol. 42, Páginas 153 - 187, ISSN 0167-2789, <DOI: 10.1016/0167-2789(90)90072-W>. Todo el documento.	1-18
A	CA 2215598 A1 (LUCENT TECHNOLOGIES INC) 12/05/1998, Todo el documento.	1-18

Categoría de los documentos citados
X: de particular relevancia
Y: de particular relevancia combinado con otro/s de la misma categoría
A: refleja el estado de la técnica

O: referido a divulgación no escrita
P: publicado entre la fecha de prioridad y la de presentación de la solicitud
E: documento anterior, pero publicado después de la fecha de presentación de la solicitud

El presente informe ha sido realizado
☒ para todas las reivindicaciones
☐ para las reivindicaciones nº:

Fecha de realización del informe	**Examinador**	**Página**
28.11.2018	M. Muñoz Sanchez	1/2

Documentación mínima buscada (sistema de clasificación seguido de los símbolos de clasificación)

G06N, G06F

Bases de datos electrónicas consultadas durante la búsqueda (nombre de la base de datos y, si es posible, términos de búsqueda utilizados)

INVENES, EPODOC, WPI, NPL, XPI3E, XPIEE