UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN EN FPGA DE UNA NEURONA ARTIFICIAL DE LATENCIA REDUCIDA

CARMEN MENDIZÁBAL ROCHE

2019

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Título: Diseño e implementación en FPGA de una neurona artificial de latencia reducida

Autor: Dª. Carmen Mendizábal Roche

Tutor: D. Pablo Ituero Herrero

Ponente: D. ……

Departamento: Departamento de Ingeniería Electrónica

MIEMBROS DEL TRIBUNAL

Presidente: D. ……

Vocal: D. ……..

Secretario: D. ……..

Suplente: D. ……..

Los miembros del tribunal arriba nombrados acuerdan otorgar la calificación de: ……

Madrid, a de de 20…

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN EN FPGA DE UNA NEURONA ARTIFICIAL DE LATENCIA REDUCIDA

CARMEN MENDIZÁBAL ROCHE

2019

RESUMEN

En los últimos años se están produciendo grandes avances en el desarrollo de las tecnologías relacionadas con la inteligencia artificial, especialmente en el campo de las redes neuronales artificiales. Este auge se debe a la cantidad de datos disponibles que permite el entrenamiento de las redes y al desarrollo de las capacidades de computación. Estos sistemas se utilizan para una gran variedad de aplicaciones en las que destacan el tratamiento de imágenes y videos, el reconocimiento del lenguaje hablado y escrito y el diagnóstico de distintas enfermedades.

La implementación de las redes neuronales requiere un equilibrio en el gasto de energía y la tasa de transferencia sin que estos requerimientos eleven el coste del hardware o reduzcan la precisión del sistema. Tampoco hay que olvidarse de la importancia que tiene realizar una implementación flexible capaz de adaptarse a las distintas aplicaciones existentes. Para conseguir una optimización de estos factores hay que aprovecharse de los distintos niveles de paralelismo inherentes a las redes neuronales.

Las placas FPGA ofrecen, a priori, un balance adecuado de todos estos factores. Por tanto, en este trabajo se ha realizado una implementación de una neurona sobre la placa Nexys 4 DDR de Digilent basada en la FPGA Artrix 7 de Xilinx. El diseño detallado en este documento está basado en el realizado por Santiago Romero Pomar en su Trabajo Fin de Grado. El objetivo principal de la implementación es reducir la latencia lo máximo posible respecto a los resultados obtenidos en el trabajo previo.

Para obtener esta reducción de latencia inicialmente se buscaron los módulos ya implantados pero capaces de trabajar con un menor retardo y se desarrolló el diseño de un multiplicador. Sin embargo, con esta nueva implantación no se obtuvieron los resultados esperados, ya que no se mejoró la latencia. Para continuar se realizó otro controlador del sistema que introducía un nuevo nivel de paralelismo.

Con todo esto, se ha obtenido una neurona artificial capaz de trabajar a una frecuencia máxima de 94,9 MHz y tiene una aceleración mínima de 1,24x respecto a la neurona implementada previamente. En el caso de la red neuronal se consiguió una frecuencia de trabajo de 94 MHz y con ello se mejoró la aceleración un 1,3x. Sin embargo, este diseño se ve limitado por la tasa de transferencia de datos disponible en la placa utilizada, esto significa que, a pesar de tener una frecuencia de trabajo elevada, no es posible implementar el sistema para rentabilizar el área al máximo.

SUMMARY

In recent years, there have been important developments regarding artificial intelligence technologies, especially in the field of artificial neural networks. This progress is related to the vast amount of data available necessary for the training of the network, and the improvements of computing capabilities. These systems are used for a wide variety of applications, the most relevant of which are image and video classification, natural language processing, and medical diagnosis.

The implementation of neural networks requires a balanced energetic efficiency and throughput, without raising the cost of hardware or reducing the systems accuracy. It is also important to have a flexible design capable of adapting to the different applications that could be utilized. To achieve an optimization of these factors, it is critical to take advantage of the different levels of parallelism inherent in neural networks.

FPGA boards offer an adequate balance of all previously mentioned factors. Subsequently, this project features an implementation of a neuron that has been carried out over the Nexys 4 DDR board, based on an Artix 7 FPGA from Xilinx. The design detailed in this document is based on a previous artificial neuron implementation made by Santiago Romero Pomar during his bachelor's Final Project. The main objective of the new implementation is to reduce the latency as much as possible in comparison to the results obtained in his previous work.

In order to further the latency reduction, there was an investigatory process in which a multiplier design was found. Theoretically, this latency delay was lower than the one implemented in Romero Pomar's previous work, but was proven to be false in actuality. Thereafter, a new controller for the system was designed, introducing a new level of parallelism.

As a result, an artificial neuron with a maximum working frequency of 94.9 MHz and a minimum speedup of 1.24x was achieved. In the case of the neural network, a working frequency of 94 MHz was obtained with a speedup of 1.3x. However, this design is limited by the throughput rate available on the FPGA board. This means that despite having a high working frequency, it is not possible to implement the system to maximize the area available.

PALABRAS CLAVE

Acelerador, aprendizaje automático, baja latencia, FPGA, inteligencia artificial, neurona, red neuronal, VHDL, Vivado

KEYWORDS

Accelerator, artificial intelligence, FPGA, low latency, machine learning, neural network, neuron, VHDL, Vivado

ÍNDICE DEL CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS DE VIVADO

1. INTRODUCCIÓN Y OBJETIVOS

Este documento es el trabajo de fin de titulación del Grado en Ingeniería de Tecnologías y Servicios de la Telecomunicación. Este primer capítulo es introductorio para poner el proyecto desarrollado en contexto. Inicialmente se definen los conceptos más destacados de la inteligencia artificial. En el apartado 1.2. se explica con mayor detalle el funcionamiento, las aplicaciones y la implementación de las redes neuronales. En el apartado 1.3. se describe el estado del arte relacionado con la implementación de redes neuronales en placas FPGA. A continuación, se plantean los objetivos concretos y el desarrollo seguido en la realización de este proyecto. Finalmente se explica la estructura de este documento.

1.1. INTRODUCCIÓN

El concepto de la inteligencia artificial surge en los años 1950 cuando John McCarthy acuña el término como "la ciencia e ingenio de hacer máquinas inteligentes, especialmente programas del cómputo inteligente".

A raíz de esta idea surgen diferentes campos, entre ellos el aprendizaje automático. Arthur Samuel lo define como "el campo de estudio que proporciona a los ordenadores la capacidad de aprender sin estar explícitamente programados". Esto implica que los programas, no necesitan estar definidos para realizar una tarea concreta, sino que a través de un entrenamiento son capaces de resolver distintos problemas.

1.1.1. TIPOS DE ALGORITMOS DE APRENDIZAJE AUTOMÁTICO

Los algoritmos de aprendizaje automático se pueden dividir en dos grandes tipos: algoritmos de aprendizaje supervisados y algoritmos de aprendizaje no supervisados [1].

Los algoritmos de aprendizaje supervisados son aquellos que parten de un conjunto de datos de los cuales ya conocemos su resultado. En función del tipo de resultado que se obtiene hay dos categorías distintas: regresión y clasificación. Los problemas de regresión son aquellos cuyos algoritmos obtienen mediante las variables de entrada una salida continua. Sin embargo, los problemas de clasificación deben ser capaces de formar categorías a partir de las variables de entrada.

Por otro lado, los algoritmos de aprendizaje no supervisados parten de un conjunto de datos de los cuales no conocemos el resultado a obtener. En estos casos los problemas se solucionan agrupando los datos en clústeres.

1.2. REDES NEURONALES

Las redes neuronales, son un subcampo del aprendizaje automático, que se caracteriza por ser un algoritmo inspirado en el funcionamiento del cerebro humano y pueden ser tanto algoritmos supervisados como no supervisados. Este trabajo se va a centrar en la implementación de dicho sistema.

1.2.1. FUNCIONAMIENTO

Las redes neuronales artificiales tratan de imitar el funcionamiento de las neuronas que trabajan en el cerebro humano. Para ello se utilizan como elementos básicos las dendritas, conjunto de ramificaciones por donde reciben los impulsos nerviosos, y el axón, elemento por el cual la neurona puede emitir la señal. En el modelo artificial, las dendritas son análogas a las señales de entrada $x = (x_1, x_2, \dots, x_I)$, que se computan dentro de la neurona antes de convertirse en señales de salida por el axón (y).

El axón de una neurona transmite la señal nerviosa a las dendritas de otras a través de la sinapsis. Es importante tener en cuenta que el proceso de sinapsis no solo conecta las entradas con las salidas, sino que también escala la señal transmitida por el axón. Este escalado, en la versión artificial, lo definiremos mediante los pesos $w = (w_1, w_2, \dots, w_I)$ [2].

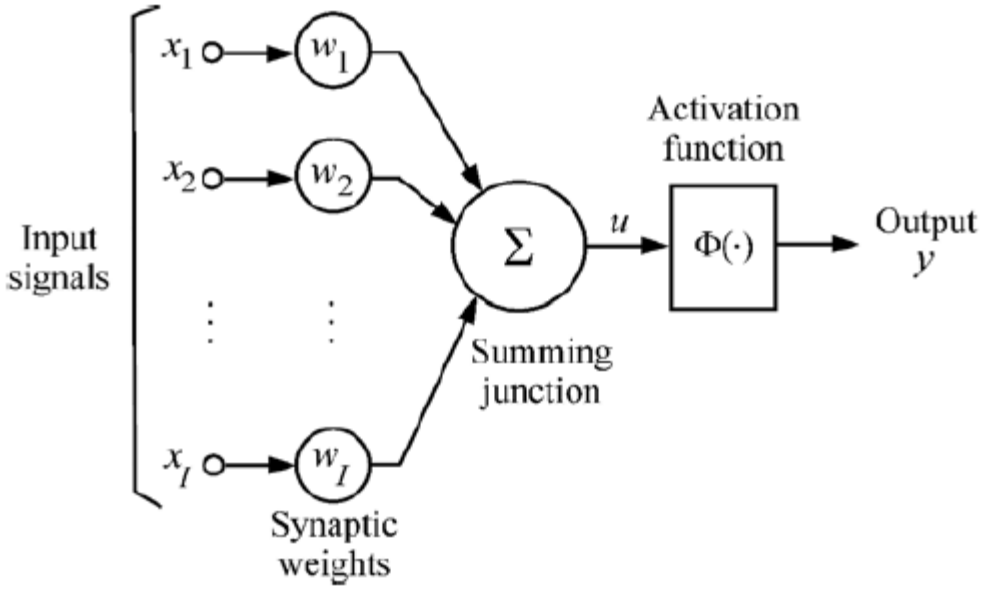


Ilustración 1. Componentes de una neurona artificial

El proceso de computación dentro de la neurona comienza con la suma de las entradas escaladas a sus pesos: $u = \sum_{i=0}^{i=I} w_i x_i$ y como este proceso es un conjunto de sumas y multiplicaciones en inglés se denomina MACC (multiply-acumulate). Posteriormente se obtiene la salida como $y = \phi(u)$, siendo ϕ la función de activación no lineal. Esta función puede tomar distintas formas, por ejemplo:

- la función escalón,

$$\phi(u) = \begin{cases} 1, & si\ u > 0 \\ 0, & si\ u \leq 0 \end{cases}$$

- la función rampa,

$$\phi(u) = max\{0, min\{1, u + 0{,}5\}\}$$

- la función sigmoide que puede ser unipolar

$$\phi = \frac{a}{1 + e^{-bu}}$$

o bipolar.

$$\phi = a\left(\frac{1 - e^{-bu}}{1 + e^{-bu}}\right)$$

En este trabajo se va a utilizar la función sigmoide unipolar.

Este modelo se puede considerar el clásico a la hora de representar el funcionamiento de las neuronas, pero existen otros, como las neuronas por impulsos, que se ajustan más a la realidad. En este caso se incorpora el tiempo como parámetro, es decir, las señales que forman el sistema no solo llevan la información en su amplitud, sino también en el tiempo de llegada del pulso o en su anchura [2].

Una vez que se ha definido el funcionamiento de la neurona hay que definir la topología de su red, para ello también existen una variedad de opciones entre las que elegir. En este trabajo nos vamos a centrar en una topología de red neuronal prealimentada, que al contrario que en las topologías recurrentes, no aparecen bucles realimentados. Estas redes están formadas por una capa de nodos de entrada, una o más capas ocultas y finalmente la capa de salida. En la capa de entrada debe haber el mismo número de neuronas que de entradas más una neurona bias, y en la capa de salida el número de neuronas debe coincidir con el número de categorías de la clasificación.

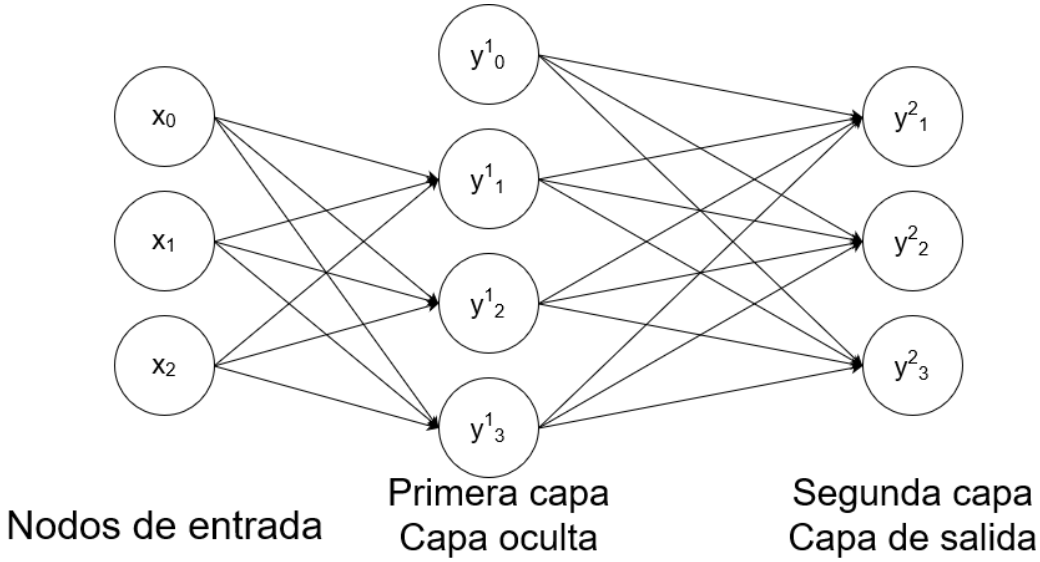


Ilustración 2. Ejemplo de conexiones en una red neuronal prealimentada

En la imagen anterior se puede ver un ejemplo de la topología de una red neuronal prealimentada. Aquellas neuronas que llevan el subíndice 0 son las llamadas unidades bias, tienen valor uno e influyen en las neuronas de la siguiente capa en función de su peso asignado. El cálculo que realiza cada una de las neuronas es el siguiente:

$$y_j^{l+1} = \phi\left(u_j^{l+1}\right) = \phi\left(w_{j0}^l y_0^l + w_{j1}^l y_1^l + \cdots + w_{jn}^l y_n^l\right)$$

Donde y_j^{l+1} representa el resultado de la función de activación de la neurona j de la capa $l+1$; w_{ji}^l, el peso que escala la salida de y_i^l a la neurona j de la capa $l+1$ y ϕ representa la función de activación.

Este algoritmo es el que realiza la función de inferencia y se conoce como propagación hacia delante, pero para poder utilizarlo previamente hace falta realizar un proceso de entrenamiento o de aprendizaje de la máquina, mediante el cual se establecen los valores adecuados a cada uno de los pesos. El método por el cual se establecen estos pesos es conocido como la propagación hacia atrás. En este trabajo vamos a suponer que el escalado de los pesos ya está definido previamente, por tanto, no será necesario implementarlo.

Existente otras muchas topologías, por ejemplo, redes neuronales convolucionales, redes neuronales de impulsos o mapas autoorganizados.

1.2.2. APLICACIONES

Gracias a la gran diversidad de topologías existentes dentro del campo de las redes neuronales, existen una gran variedad de aplicaciones en los que las distintas redes neuronales tienen o pueden llegar a tener un gran impacto [2]:

- Imagen y video: actualmente se puede considerar que los datos de imagen y video forman el conjunto más grande en el área del Big Data y para poder recopilar información relevante de ellos hace falta el desarrollo de la visión artificial. En este campo las tareas más relevantes se centran en el procesamiento y clasificación de imágenes, localización de objetos y reconocimiento de acciones.
- Lenguaje hablado y escrito: gracias a las redes neuronales se ha mejorado la precisión del reconocimiento del leguaje hablado, la traducción y la generación de grabaciones.
- Médico: las redes neuronales han tenido un gran papel en la genómica y en el análisis de imágenes médicas, especialmente en la detección de distintos canceres.

Las redes neuronales también son relevantes en distintos campos como los videojuegos, la robótica, las finanzas, la infraestructura y la predicción del tiempo.

1.2.3. IMPLEMENTACIONES EN FPGA

Para la implementación de las redes neuronales es importante tener en cuenta los distintos tipos de paralelismos que aparecen [3]:

- Paralelismo en el entrenamiento: Busca ejecutar simultáneamente diferentes entrenamientos. El nivel de paralelismo de este método es medio.
- Paralelismo por capas: En el perceptrón multicapa se pueden procesar diferentes capas de forma simultánea. La capacidad de paralelizar con este método es baja, por tanto, no se consiguen mejoras muy sustanciales.
- Paralelismo por nodos: En este caso se deberían procesar todos los nodos al mismo tiempo. Este nivel de paralelismo es el que aporta mejores resultados, ya que implica que también se aprovecha el paralelismo a nivel de capa. Sin embargo, es complicado implementar un sistema que explote el paralelismo por nodos al completo por que se pueden llegar a necesitar millones de neuronas en ejecución concurrente.
- Paralelismo por pesos: En este nivel el objetivo es explotar el paralelismo que ofrece la MACC: $u = \sum_{i=0}^{i=I} w_i x_i$,. Tanto la multiplicación, de los pesos con las entradas, como las sumas de los productos. Este nivel también es de un paralelismo alto.
- Paralelismo a nivel de bit: Este nivel de paralelismo se implementa en el diseño individual de cada unidad funcional.

Los procesadores de uso general no tienen la capacidad de aprovechar todos los niveles de paralelismo que ofrecen las redes neuronales, lo que motiva el desarrollo de procesadores diseñados exclusivamente para esta tarea, como los ASIC. Sin embargo, estos circuitos tienen un coste muy elevado para los casos en que no se fabrica el producto en masa, como en este caso. Por tanto, las placas ASIC, a pesar de tener la capacidad de optimizar la implementación paralela de las redes neuronales, no tienen un alto rendimiento teniendo en cuenta el coste.

En este contexto las FPGAs pueden suponer la solución alternativa. A diferencia de los circuitos ASIC las FPGAs son reconfigurables, esta flexibilidad favorece que se usen para distintos tipos de aplicaciones o de redes neuronales. También permiten la implementación de distintos tipos de paralelismo, en especial, el paralelismo en el entrenamiento, ya que FPGAs grandes son capaces de implementar este nivel de paralelismo al completo. Por tanto, pueden llegar a alcanzar un mayor rendimiento relativo al precio de circuitos ASIC y de procesadores de uso general.

Las FPGAs incluyen bloques DSP (por sus siglas en inglés, Digital Signal Processor), estos están optimizados para realizar multiplicaciones, sumas y funciones de multiplicación-suma (MACC). Gracias a ellos se puede lograr un nivel de paralelismo muy alto con los dispositivos FPGA, especialmente en el nivel de nodo, pero, aun así, estos dispositivos no siempre son capaces de explotar este nivel de paralelismo al completo, ya que algunas redes tienen del orden de millones de neuronas.

1.3. ESTADO DEL ARTE

En los últimos años se han desarrollado distintas implementaciones de redes neuronales en FPGA. A continuación, voy a contar algunos de los estudios más interesantes para este trabajo:

En el capítulo 10 de [3] se explica una posible implementación de un perceptrón multicapa y un eXtended Multi-Layer Perceptron (XMLP), ambos enfocados en el reconocimiento del habla. El XMLP es un sistema modificado de una red neuronal modular, esta topología propone dividir el perceptrón multicapa completo en diferentes subtareas que se posteriormente se combinan par formando el sistema completo.

Para el estudio del perceptrón multicapa se realizan distintas implementaciones, combinando sistemas de ejecución secuencial o paralela y sistemas con distintos tipos de almacenamiento de datos. Finalmente se obtienen los siguientes resultados, donde se puede ver que claramente los sistemas paralelizados tienen un rendimiento mucho mayor, independientemente del sistema de memoria escogido.

MLP Design		♯ Slices	% Slices	♯EMBs RAM	%EMBs RAM	Clock (ns)	♯ Cycles	EvaluationTime (ms)
(a)	Serial	2582	13	0	0	50.620	5588	282.864
	Parallel	6321	32	0	0	58.162	282	16.402
(b)	Serial	710	3	24	15	62.148	5588	347.283
	Parallel	4411	22	24	15	59.774	282	16.856
(c)	Serial	547	2	36	22	65.456	5588	365.768
	Parallel	4270	22	36	22	64.838	282	18.284

Tabla 1. Resultados de la implementación del MLP. (a) solo distributed RAM. (b) block y distributed RAM. (c) solo block RAM

En el caso de XMLP se hace un estudio similar donde se vuelve a comparar las implementaciones secuenciales y paralelas y los mismos sistemas que antes en el almacenaje de memoria. Y al igual que en el caso anterior, el sistema paralelizado obtiene muchos mejores resultados.

MLP Design		♯ Slices	% Slices	♯EMBs RAM	%EMBs RAM	Clock (ns)	♯ Cycles	EvaluationTime (ms)
(a)	Serial	2389	12	0	0	44.851	2566	115.087
	Parallel	5754	29	0	0	47.964	143	6.858
(b)	Serial	1700	8	96	60	71.568	2566	183.643
	Parallel	5032	26	96	60	64.270	143	9.190
(c)	Serial	1608	8	140	91	77.220	2566	198.146
	Parallel	4923	25	147	91	64.830	143	9.271

Tabla 2. Resultados de la implementación del XMLP. (a) solo distributed RAM. (b) block y distributed RAM. (c) solo block RAM

Por último, es importante destacar que XMLP obtiene mejores resultados que el perceptrón multicapa, esto se debe a que el sistema reduce el número de multiplicaciones y las necesidades de almacenamiento de datos.

En el capítulo 11 de [3] se explica un estudio también realizado sobre perceptrones multicapa. En este caso además de realizar la implementación de la propagación hacia adelante realiza la implementación del proceso de aprendizaje. Finalmente se realiza un diseño capaz de implementar una red neuronal completa, tanto la fase de aprendizaje como la fase de cálculo en un solo dispositivo FPGA con una tasa de transferencia de 50 MHz.

Otro trabajo previo, del que este es continuación, es el Trabajo Fin de Grado de Santiago Romero Pomar con Pablo Ituero Herrero como tutor [4]. En él también se ha realizado una implementación de un perceptrón con el objetivo de obtener una baja latencia. En este trabajo se incluyen explicaciones detalladas del funcionamiento del perceptrón que podría formar parte de una red neuronal. En el caso del perceptrón simple (neurona) se obtienen una alta frecuencia de trabajo de 71,17 MHZ y en la implementación de una red neuronal la máxima frecuencia de reloj disminuye a 50 MHz.

Los mapas autoorganizados (self organiced featured map en inglés) son un tipo de redes neuronales basadas en algoritmos de aprendizaje no supervisado. Estos sistemas obtienen muy buenos resultados, especialmente en el tratamiento de datos muy ruidosos o incompletos. Sin embargo, la resolución de mapas de tamaño medio puede tardar horas en ejecutarse en procesadores de uso general y cuando se trata de mapas de tamaño grande puede tardar del orden de cientos de horas.

En un estado previo (capítulo 9 de [3] y en [5]) se implementa un mapa autoorganizado basado en el acelerador RAPTOR2000. El objetivo de este trabajo es incorporar un sistema capaz de realizar ejecuciones de estas redes desde ordenadores personales. Por tanto, para las simulaciones de este estudio se ha conectado el acelerado al host mediante un bus PCI. Como resultado de este estudio se ha obtenido una aceleración de entre 87 a 190 en la etapa de inferencia, dependiendo de la placa utilizada, y entre 39 a 86 durante la etapa de aprendizaje.

En el diseño expuesto en *DLAU: A Scalable Deep Learning Accelerator Unit on FPGA* [6] también se quiere implementar un acelerador, en este caso recibe las señales de entrada a través de JTAG-UART y devuelve los resultados tras la ejecución. Tras las simulaciones realizadas, se hace un análisis de tiempos y calcula la aceleración en base al procesador Intel Core2 y se muestra en comparación con los estudios de otros aceleradores.

Work	Network	Clock	Speedup	Baseline
Ly&Chow [5]	256×256	100MHz	32×	2.8GHz P4
Kim et.al [7]	256×256	200MHz	25×	2.4GHz Core2
DianNao [6]	General	0.98GHz	117.87×	2GHz SIMD
Zhang et.al [3]	256×256	100MHz	17.42×	2.2GHz Xeon
DLAU	256×256	200MHz	36.1×	2.3GHz Core2

Tabla 3. Comparación entre distintos aceleradores

1.4. OBJETIVOS

En este trabajo se va a desarrollar e implementar una neurona artificial en una FPGA, con el objetivo de con seguir una baja latencia. La neurona que se va a desarrollar debe cumplir con los siguientes requisitos:

- La neurona se implementará con el lenguaje VHDL, con la herramienta de Xilinx Vivado.
- La implementación de la neurona se hará sobre la placa Nexys 4 DDR basada en la Artix 7 FPGA de Xilinx.
- Se realizarán pruebas funcionales con puertas lógicas, tanto una neurona sola como una red.

Para completar estos requisitos partiré de la implementación realizada en [4]. Punto a partir del cual comenzará este trabajo. Por tanto, otro objetivo importante es que la latencia de la neurona a desarrollar mejore los resultados obtenidos en la implementación anterior.

Además de cumplir los propósitos impuestos para el sistema a lo largo de este proyecto ha habido distintos objetivos académicos que también se han cumplido:

- Adquirir conocimientos sobre el aprendizaje automático y sobre distintos algoritmos relacionado a este campo.
- Desarrollar habilidades ingenieriles en el ámbito del diseño electrónico a nivel RTL y algorítmico.
- Mejorar habilidades de programación en VHDL y en el uso de la herramienta de Vivado especialmente en el proceso de eliminación de bugs, simulación y realización de pruebas.
- Aprender a trabajar sobre proyecto realizados por otras personas y a modificar programas ya implementados.

1.5. DESCRIPCIÓN DEL DESARROLLO DEL PROYECTO

A continuación, voy se explica el proceso que seguido para la realización del proyecto. Para ello se describirán las distintas tareas que completadas, junto con las horas que dedicadas a cada una.

El curso de *Machine Learning* de la plataforma de Coursera [1] ofrece una extensa introducción al aprendizaje automático, a la minería de datos y al reconocimiento de patrones. Sirve de primer contacto con distintas técnicas del aprendizaje automático dentro de los algoritmos de aprendizaje supervisado y de los no supervisado, además en él se realizan implementaciones de sistemas de regresión y de redes neuronales con la herramienta Matlab. Esta tarea se completó dedicando 40 horas.

Tarea de investigación sobre distintos artículos científicos [3], [2], [6] y [5]. El objetivo de estas lecturas es continuar la formación en el campo de las redes neuronales enfocándolo a su implementación en placas FPGA, con especial énfasis a los métodos de reducción de su latencia. Esta tarea se completó dedicando 30 horas.

Lectura del Trabajo Fin de Grado realizado por Santiago Romero Pomar [4]. Este proceso se realizó con mucho detenimiento, debido a la importancia que tenía comprender cada uno de los módulos que forman la neurona. En este documento aparece el desarrollo completo seguido por el autor y todo el código utilizado, lo que será de gran utilidad para la nueva implementación. Esta tarea se completó dedicando 25 horas.

Con los objetivos del trabajo claros se buscó información específica sobre la implementación en hardware de los distintos componentes que forman el perceptrón, especialmente implementaciones que tratan de reducir latencia. Distintos artículos [7] y [8] hacen referencia a un multiplicador con el cual se obtenían mejores resultados en cuanto a latencia que el multiplicador implementado en el trabajo previo [4]. Por tanto, se realizó el desarrollo de este diseño.

La implementación se ha realizado con el lenguaje VHDL sobre el Vivado 2018.3, y en este proceso se han seguido los pasos análogos a los de programación de un sistema software. Inicialmente se hace un análisis de la situación existente y con ello se diseña el algoritmo a completar. A continuación, se realiza la implementación del sistema, codificando el diseño planteado y realizando simulaciones para comprobar que cumple con los objetivos impuestos. Finalmente se sintetiza para hacer el estudio de los tiempos de ejecución.

En la implementación del multiplicador aparecieron ciertos problemas de funcionamiento derivados de errores existentes en los artículos que explican el diseño. Por lo que finalmente los resultados obtenidos no fueron útiles para la mejora de la latencia. Esta tarea se completó dedicando 50 horas.

Tras este fracaso se modificó del controlador MACC de la neurona. Esta modificación está basada en el hecho de que la implementación previa no se explotaban todos los niveles de paralelismo existente en las neuronas. Por tanto, el nuevo diseño aprovecharse del paralelismo a nivel de pesos. Para poder realizar este diseño también se tuvo que modificar partes del módulo del sumador.

Una vez que realizado este controlador, se integraron los cambios al control de la neurona completa obteniendo así una neurona que incorpora el paralelismo en los nodos manteniendo la configurabilidad existente del sistema. Todas las tareas relacionadas con el control de la neurona se completaron dedicando aproximadamente 40 horas.

A continuación, se procede a realizar un estudio de los resultados obtenidos del sistema, tanto para una neurona simple como para una red neuronal. El estudio incluye un análisis del tiempo y del área ocupado, así como una prueba sobre la placa FPGA. Fueron pruebas similares a las realizadas en trabajo previo realizado por Santiago Romero Pomar, para así poder realizar una buena comparación.

Finalmente, se plantea la posibilidad de utilizar la placa FPGA como acelerador conectado a un host a través de los periféricos I/O construidos en la placa. Sin embargo, se pudo comprobar que no es un uso lógico para el diseño, ya que las velocidades de transferencias disponibles no rentabilizarían el área de la placa. Esta última parte del trabajo se completó dedicando alrededor de 20 horas.

En resumen, el desarrollo de este trabajo me ha llevado en torno a 205 horas de trabajo. En la siguiente tabla Gantt se muestra un desglose más detallado de las horas dedicadas a cada tarea.

Tareas		Inicio (h)	Duración (h)
Coursera: Machine Learning		0	40
Investigación		41	30
Lectura TFG		71	25
Multiplicador	Busqueda de información	96	15
	Implementación y pruebas	111	35
Controlador	Implementación MAC	146	15
	Pruebas MAC	161	5
	Implmentación Neurona	166	15
	Pruebas funcionales NA	181	5
Pruebas	Neurona Artificial	186	10
	Red Neuronal	196	5
Implementación como Acelerador		201	5
		Total	205

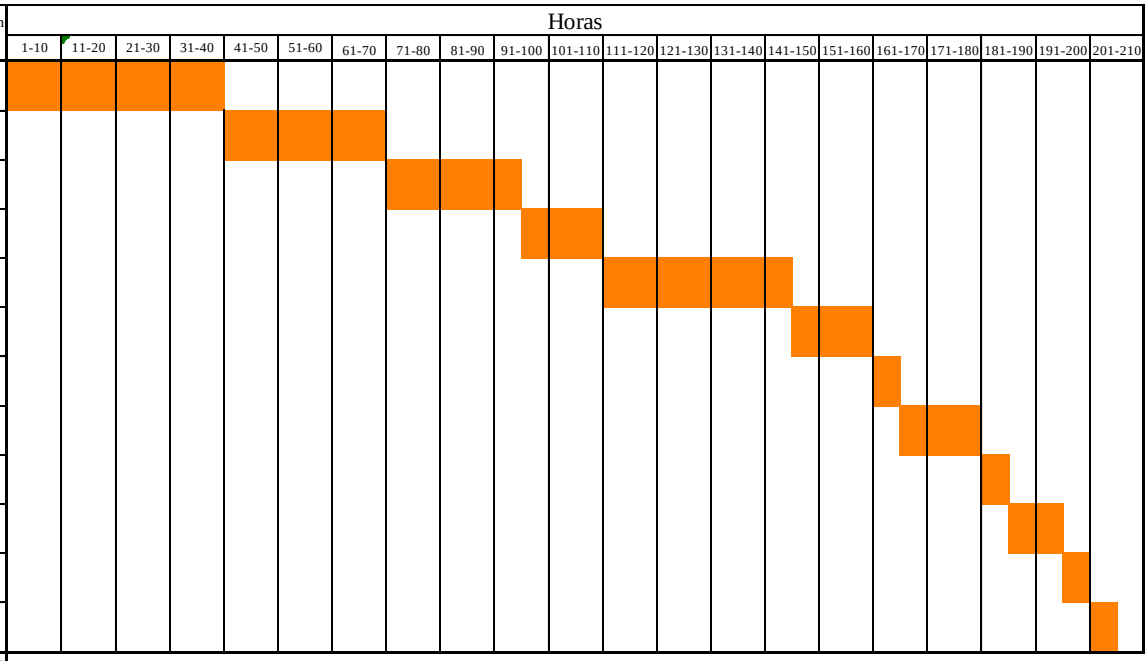


Ilustración 3. Diagrama de Gantt

Este documento se organiza de la siguiente forma. En el segundo capítulo aparece el desarrollo del hardware, donde se explica la metodología de trabajo seguida y las mejoras realizadas respecto a la implementación diseñada por Santiago Romero Pomar. En el tercer capítulo se muestran los resultados de tiempo y área obtenidos para distintas pruebas, también se analiza la viabilidad de la implementación propuesta como acelerador. En el último capítulo se exponen las conclusiones del trabajo realizado y se presentan líneas futuras de trabajo.

2. DESARROLLO HARDWARE

Este capítulo incluye una explicación de la metodología de trabajo seguida en el desarrollo del hardware, donde se detallan las estrategias elegidas y los pasos tomados para la realización del proyecto. Después, se describe el desarrollo detalladamente de cada uno de estos pasos, tanto las decisiones de diseño tomadas como los problemas encontrados.

2.1. METODOLOGÍA DE TRABAJO PARA DISEÑO DE BAJA LATENCIA

El trabajo desarrollado en este documento está basado en el Trabajo Fin de Titulado de Santiago Romero Pomar [4], por tanto, inicialmente se realizó un estudio de este trabajo, tanto de la metodología, como del diseño y de los resultados obtenidos.

Para continuar se realiza una búsqueda de métodos para desarrollar los módulos ya implantados con una menor latencia. Una vez encontrado el adecuado se puede comenzar a diseñar el algoritmo nuevo y codificarlo en VHDL. Este paso se realiza sobre la herramienta de Vivado, ya que permite simular el funcionamiento de los diseños según unos estímulos de entrada, examinar los diagramas a nivel RLT, sintetizar e implementar diseños, con lo que obtengo análisis de la situación (tiempos, área, esquemático, etc.), y configurar un dispositivo FPGA mediante un programador.

A continuación, se hacen las simulaciones de la funcionalidad de los módulos, para ello se necesita elaborar unas pruebas de *testbench*. Estas pruebas permiten comprobar que para distintas señales de entrada el módulo trabaja como se había diseñado, también es interesante ver las señales intermedias para ver el flujo de trabajo.

Después, se sintetiza e implementa el diseño para distintas frecuencias de trabajo. Este paso en Vivado se puede optimizar en función de distintos parámetros y como el objetivo es reducir la latencia lo máximo posible se utiliza la estrategia *Performance_NetDelay_low*. Con esta configuración se consigue reducir el retardo neto y por tanto se consigue una menor latencia.

Mediante los análisis de tiempo se puede comprobar que el diseño cumple los requerimientos de tiempo establecidos. Aquí se ven los resultados tanto para los tiempos de *setup*, de *hold* y de anchura del pulso y para cada uno de ellos se muestra el *Worse Slack*, el *Total Slack* y el *Total number of failing endpoints*. Además, se puede ver el camino crítico del sistema. Este análisis es muy importante para nuestro desarrollo, ya que buscamos optimizar la frecuencia de trabajo lo más posible.

Asimismo, esta herramienta permite realizar un estudio de área para comprobar los recursos utilizados, como los LUT, los FF, los bloques DSP o los puertos I/O. Este análisis es interesante, aunque este trabajo no se va a centrar en la lectura de estos resultados, ya que el objetivo es la reducción de latencia. Aun así, hay que destacar que los bloques DSP permiten reducir los recursos utilizados y la latencia del sistema. Por tanto, se incluirán en el módulo del multiplicador y en el módulo que engloba a todo el sistema (TOP) para mejorar los resultados obtenidos.

Para cada módulo desarrollado se realiza el análisis de tiempo y así poder comparar el resultado obtenido con la implementación que se busca mejorar. De esta forma se puede elegir el sistema con la menor latencia. Los resultados de tiempo y de área se pueden comparar directamente, ya que en ambas implementaciones el hardware objetivo del diseño es el mismo.

Tras no conseguir el resultado deseado al intentar mejorar los módulos que conforman la neurona se planteó un diseño para el controlador global y así poder aprovechase del paralelismo inherente de las neuronas. El flujo de trabajo seguido para este diseño es análogo al planteado anteriormente. Para concluir la implementación se realizan una serie de pruebas sobre el hardware, explicadas en más detalle en el siguiente capítulo.

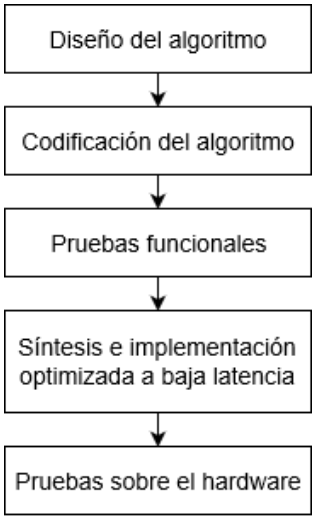


Ilustración 4. Flujo de trabajo

El hardware utilizado en este trabajo es la placa Nexys 4 DDR que está basada en la FPGA Artrix 7 de Xilinx. Incluye un elevado número de periféricos, entre los que quiero destacar el puerto USB, ya que será relevante más adelante.

2.2. UNIDADES FUNCIONALES QUE FORMAN LA NEURONA

En el siguiente diagrama se puede ver el funcionamiento de la implementación realizada en el proyecto de Santiago Romero Pomar [4]:

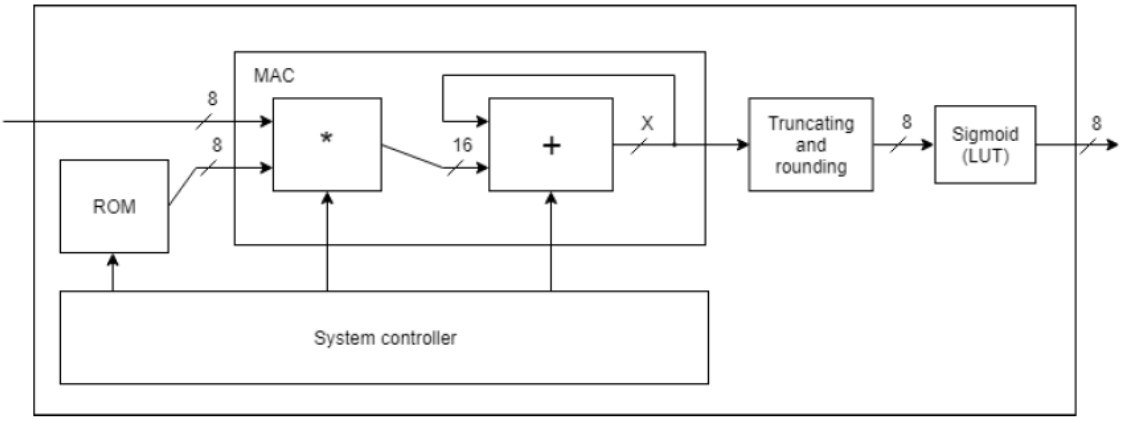


Ilustración 5. Diagrama de una neurona artificial

La señal de entrada tiene una longitud de ocho bits y los pesos se obtienen a partir de una memoria ROM en la que están almacenados. Estas dos señales entran a un módulo MACC, que está formado por un multiplicador y un sumador dirigidos por el controlador del sistema. De este módulo sale una señal de longitud variable, *X*, que sufre un proceso de truncado y redondeo para finalmente salir tras la función de activación, que en este caso se trata de la función sigmoide.

El multiplicador implementado es el DADDA, el sumador el CSA (por sus siglas en inglés Carry Select Adder) y la función sigmoide es una tabla de consulta (LUT por sus siglas en inglés LookUp Table).

2.2.1. MULTIPLICADOR

Durante el periodo de búsqueda de información se encontraron varios estudios sobre el multiplicador Vedic. Según estos artículos [7], [8] se consiguen latencias menores respecto al multiplicador DADDA, tal y como se puede apreciar en esta tabla [7]:

MULTIPLIER	Maximum Combinational path Delay(ns)			
	8 BIT	16 BIT	32 BIT	64 BIT
VEDIC	18.353	24.541	31.835	33.882
DADDA	20.871	28.037	35.372	37.57
BRAUN	21.049	28.155	35.444	37.642
WALLACE	26.248	33.354	40.643	42.842

Tabla 4. Retardo de distintos multiplicadores

La implementación de este análisis está realizada sobre Verilog-HDL usando la herramienta Xilinx ISE 10.1i.

2.2.1.1. DISEÑO DEL ALGORITMO Y PRUEBAS FUNCIONALES

Según los artículos mencionados antes, para obtener los multiplicadores de N bits x N bits hay que seguir este diagrama[8]:

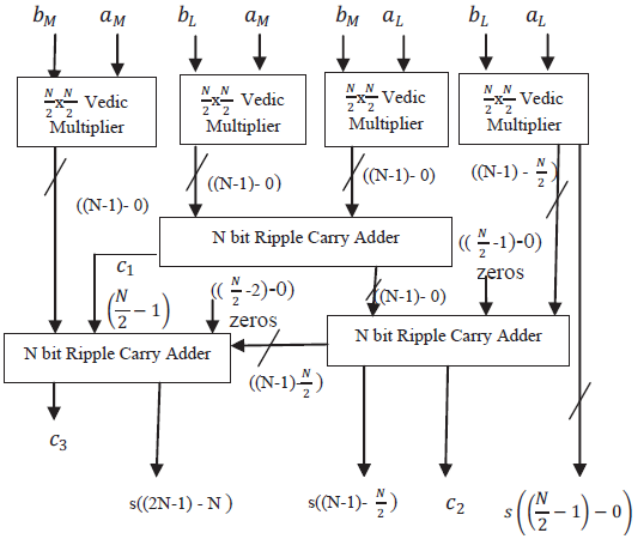


Ilustración 6. Diagrama de bloques de multiplicador Vedic N bits x N bits

En este diagrama *a* y *b* son las señales de entrada y cada una de ellas tiene N bits. Las señales a_M y b_M contienen los ((N) a (N/2)) bits más significativos de sus respectivas señales y a_L y b_L contienen los ((N/2 -1) a 0) bits menos significativos. Los *Ripple Carry Adder* (RCA) son sumadores formados por sumadores completos de un bit (FA por sus siglas en inglés, *Full Adder*) concatenados. También se puede observar que para implementar un multiplicador Vedic de ocho bits es necesario implementar el multiplicador de cuatro y de dos bits.

A partir de este diagrama general se obtienen los siguientes diagramas de multiplicadores de 8 bits x 8 bits, 4 bits x 4 bits y 2 bits x 2 bits:

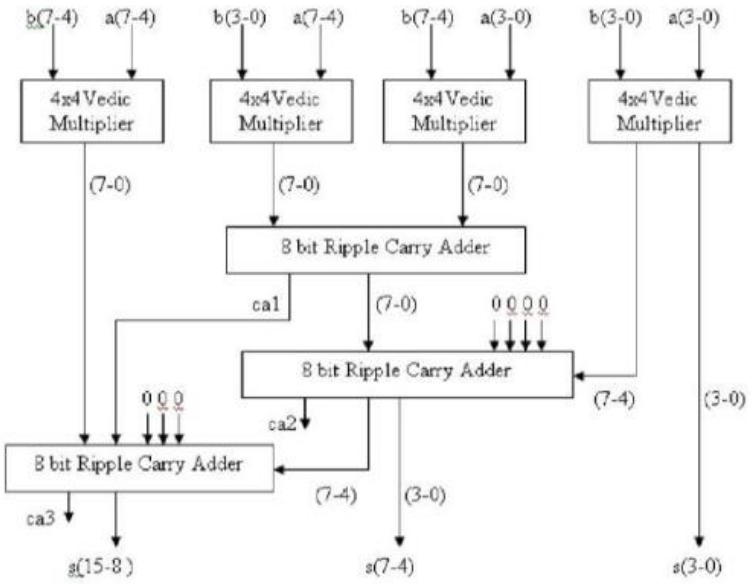


Ilustración 7. Diagrama de bloques del multiplicador Vedic 8 bits x 8 bits

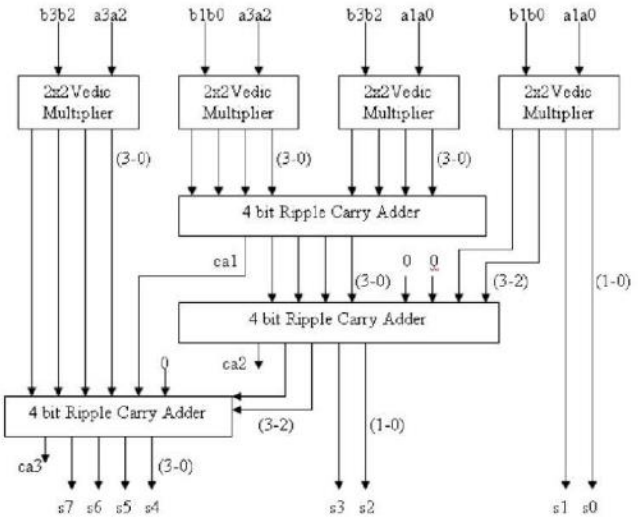


Ilustración 8. Diagrama de bloques del multiplicador Vedic 4 bits x 4 bits

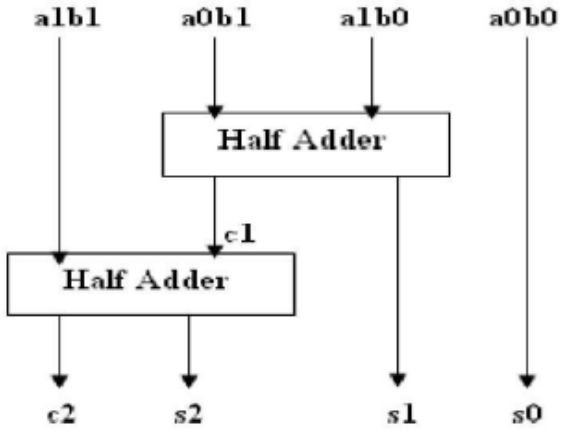


Ilustración 9. Diagrama de bloques del multiplicador Vedic 2 bits x 2 bits

Después de la codificación del sistema planteado se realizaron las pruebas funcionales. En este paso se pudo comprobar que el diseño no realizaba correctamente la función de multiplicación. Para encontrar el fallo se elaboraron pruebas de los multiplicadores más pequeños a los más grandes.

El multiplicador de señales de dos bits realiza la función deseada, por tanto, el siguiente paso fue realizar las pruebas del multiplicador de cuatro bits. Se pudo ver que tenía fallos en algunos casos, concretamente cuando el segundo RCA tenía acarreo (*ca2=1*). Para solucionar este problema se debe introducir la señal *ca2* en el siguiente sumador y así tenerla en cuenta en la salida. Sin embargo, este bit es del mismo orden de magnitud que el bit de acarreo del primer RCA (*ca1*), por tanto, se deben sumar ambas señales mediante un semisumador (HA por sus siglas en inglés, *Half Adder*).

Una vez que el multiplicador de 4 bits x 4 bits funciona correctamente se vuelven a realizar las pruebas funcionales del sistema completo, pero siguen existiendo problemas de funcionamiento. En este caso, los errores aparecen cuando el primer o el segundo RCA tiene acarreo (*ca1=1* o *ca2=1*). Para solucionarlo se añade un HA para sumar las señales *ca1* y *ca2* y el resultado se introduce en el tercer RCA en la posición apropiada para su orden de magnitud.

Los siguientes diagramas representan el diseño obtenido tras las modificaciones:

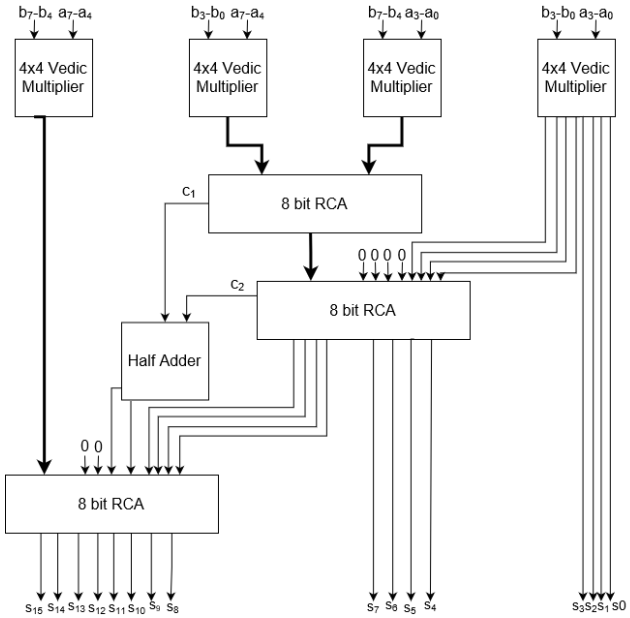


Ilustración 11. Diagrama de bloques de multiplicador Vedic 8 bits x 8 bits modificado

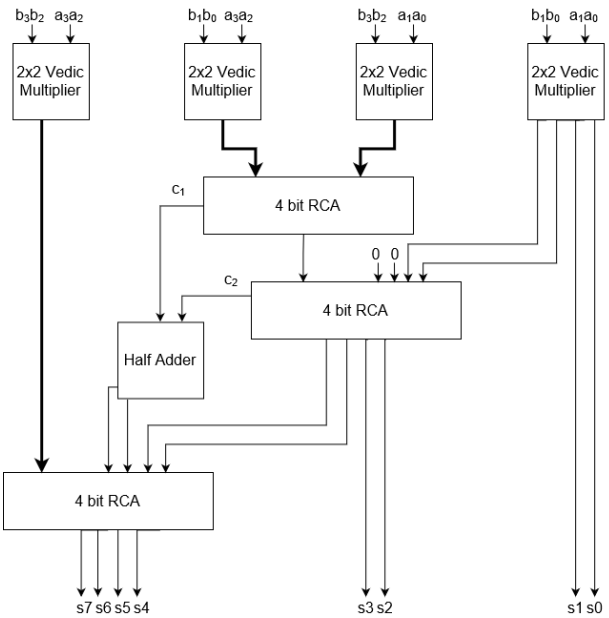


Ilustración 10. Diagrama de bloques de multiplicador Vedic 4 bits x 4 bits modificado

2.2.1.2. IMPLEMENTACIÓN DEL MULTIPLICADOR

Este diseño se implementa en Vivado y se obtienen los siguientes esquemáticos en los que se puede observar que la estructura coincide con la explicada anteriormente.

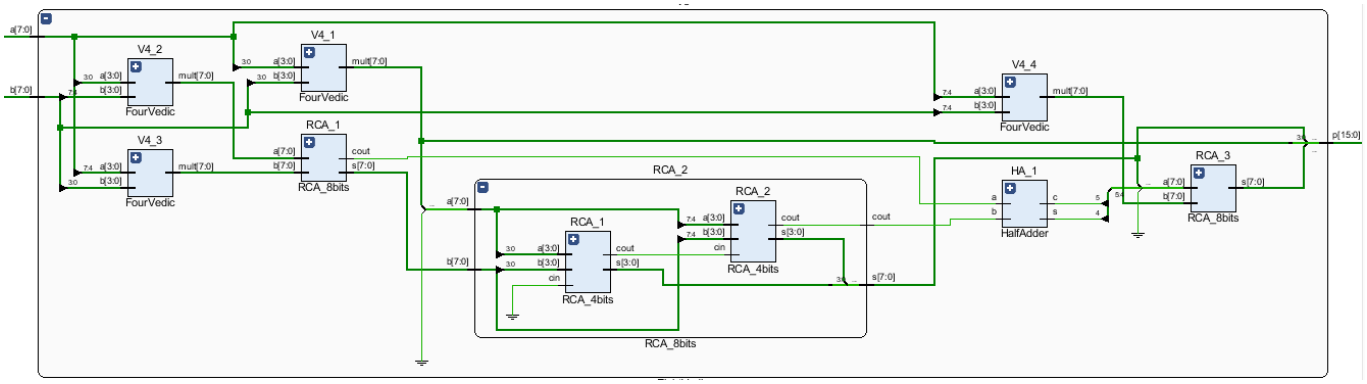


Figura de Vivado 1. Multiplicador Vedic 8 bits x 8 bits: Esquemático RTL

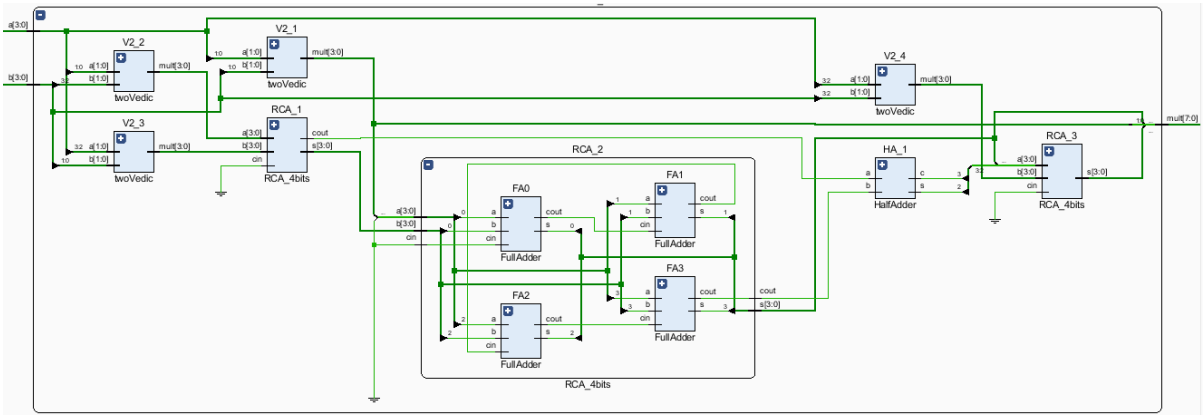


Figura de Vivado 2. Multiplicador Vedic 4 bits x 4 bits: Esquemático RTL

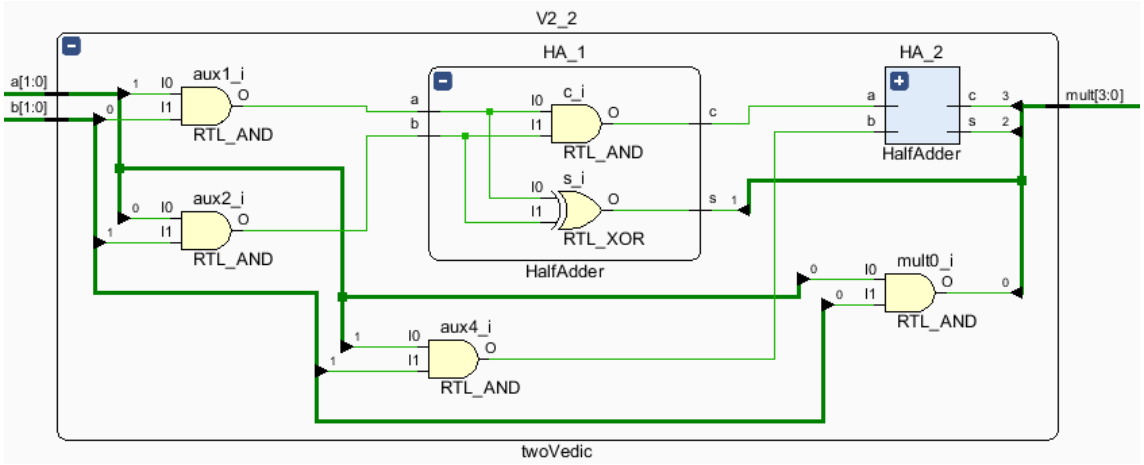


Figura de Vivado 3. Multiplicador Vedic 2 bits x 2 bits: Esquemático RTL

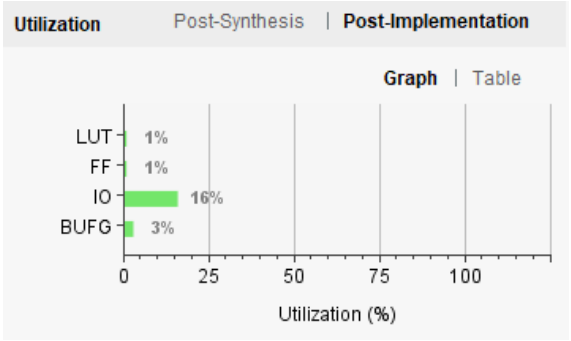
Para poder conocer la frecuencia máxima de trabajo del multiplicador se implementa el sistema con un programa TOP que incluye una señal de reloj. De esta forma se comprueba que el sistema puede trabajar como máximo a 85,47MHz (11,7ns).

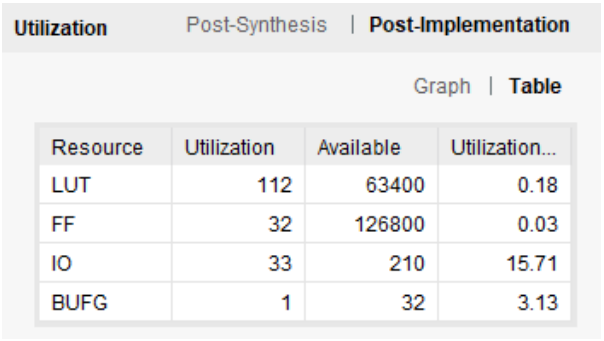
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0,013 ns	Worst Hold Slack (WHS):	0,058 ns	Worst Pulse Width Slack (WPWS):	5,250 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	48	Total Number of Endpoints:	48	Total Number of Endpoints:	33

All user specified timing constraints are met.

Figura de Vivado 5. Multiplicador Vedic: Análisis de tiempo





Resource	Utilization	Available	Utilization...
LUT	112	63400	0.18
FF	32	126800	0.03
IO	33	210	15.71
BUFG	1	32	3.13

Figura de Vivado 4. Multiplicador Vedic: Análisis de área

2.2.1.3. ANÁLISIS DE LOS RESULTADOS

En el trabajo de Santiago Romero Pomar [4] se desarrolló el multiplicador DADDA y se obtuvo una frecuencia máxima de trabajo de 86,96MHz (11,5ns). Por lo tanto, el nuevo diseño implementado (85,47MHz, 11,7 ns) no mejora los tiempos ya que la nueva frecuencia de trabajo es menor. Además, este nuevo diseño también sufre un aumento de área respecto al DADDA.

Debido a que este nuevo multiplicador no mejora la latencia del sistema, se mantendrá el uso del multiplicador DADDA para la implementación de la neurona. A continuación, subiremos al nivel del controlador para diseñar un sistema que introduzca un mayor nivel de paralelización en el sistema.

2.2.2. CONTROLADOR

En este apartado se plantea un controlador que realiza solo la función del MACC, sin embargo, para la neurona completa se implantará con un solo controlador que también se encargue del truncado y redondeo y de la función de activación.

Tal y como explicamos en la introducción existen distintos niveles de paralelización posible para la implementación de una neurona, entre ellos está la paralelización a nivel de peso. El sistema que se trata de mejorar no se aprovechaba de este nivel. Con lo cual el nuevo diseño realizará varias multiplicaciones simultáneamente.

Para poder conseguir esto he diseñado un sistema con ocho multiplicadores en paralelo que continúa con una estructura en árbol de sumadores hasta acabar con un acumulador. En la figura se muestra el hardware que requiere el sistema y las conexiones entre los distintos módulos:

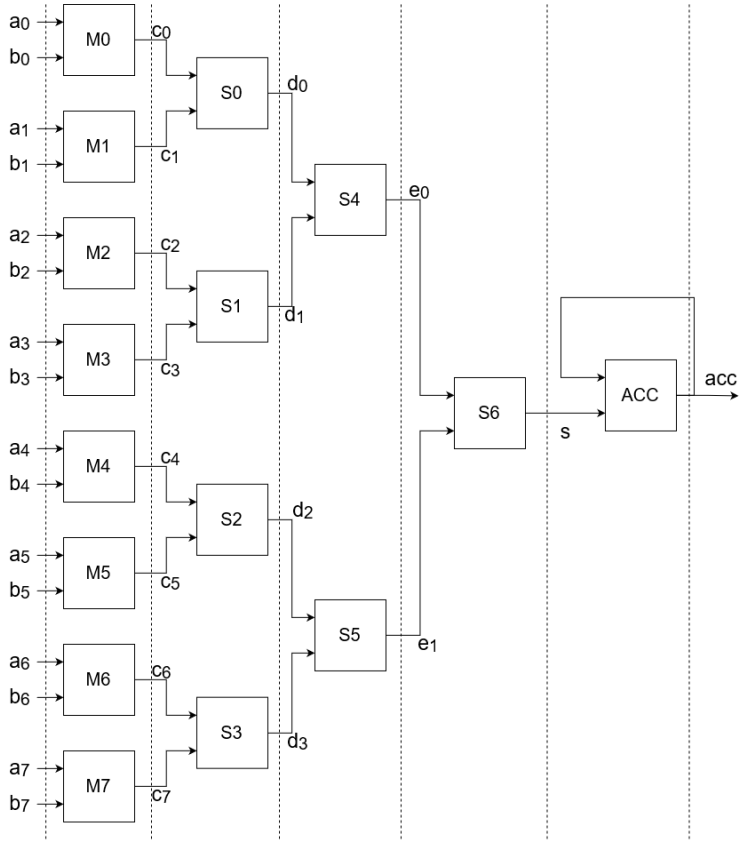


Ilustración 12. Diagrama de bloques del MACC

En este diagrama las líneas de puntos horizontales representan los registros de las señales con las que se cruzan. Los bloques con una M representan los multiplicadores DADDA, los bloques con una S representan los sumadores y el bloque con ACC representa al acumulador. Para este diseño, los sumadores y el acumulador que se han utilizado están tomados del acumulador implementado en el trabajo realizado por Santiago Romero Pomar. Solo se ha tenido que modificar el tamaño de las entradas para que realizasen la función requerida.

El sistema está basado en una señal contador que cuenta el número de ciclos que han transcurrido desde el comienzo de la ejecución. La ejecución comienza una vez que la señal *start* se encuentra activa simultáneamente con un flanco de subida del reloj. En el primer pulso de reloj se registran las entradas de los multiplicadores, siendo estas las ocho primeras entradas de la neurona (x_i) y sus respectivos pesos (w_i). En cada uno de los siguientes ciclos de reloj se registran las señales entradas de los multiplicadores, las entradas de la neurona y sus respectivos pesos. Si el número de entradas de la neurona no es múltiplo de ocho se completan con ceros las señales de los multiplicadores. El número de ciclos de reloj extras necesarios para introducir todas las entradas en los multiplicadores se guarda en la variable *ncycles*.

Simultáneamente, por cada ciclo de reloj el flujo de la señal va avanzando, es decir, la salida de los multiplicadores (M0 a M7) se convierten en las entradas de los sumadores (S0-S3), que a su vez sus salidas pasan a ser las entradas de los siguientes sumadores (S4 y S5), cuyas salidas se convierten en las entradas del último sumador (S6). En el quinto ciclo de reloj ya ha llegado un valor a la entrada del acumulador, este tiene una salida de longitud variable dependiendo del número de entradas de la neurona.

El cálculo se completa una vez que han pasado cinco ciclos de reloj desde que se introdujo la última señal de entrada al multiplicador, es decir, cuando la señal *count = ncycles+5*. Una vez que finaliza el cálculo se activa la señal de *finish* y la señal de salida toma el valor de la señal *acc*.

Por último, hay que tener en cuenta que el diseño realizado también cuenta con una señal de *reset*, es una señal asíncrona y en el momento que se activa la neurona vuelve al estado de *Idle* a espera de la siguiente activación del sistema

A continuación, se muestra el diagrama ASMD:

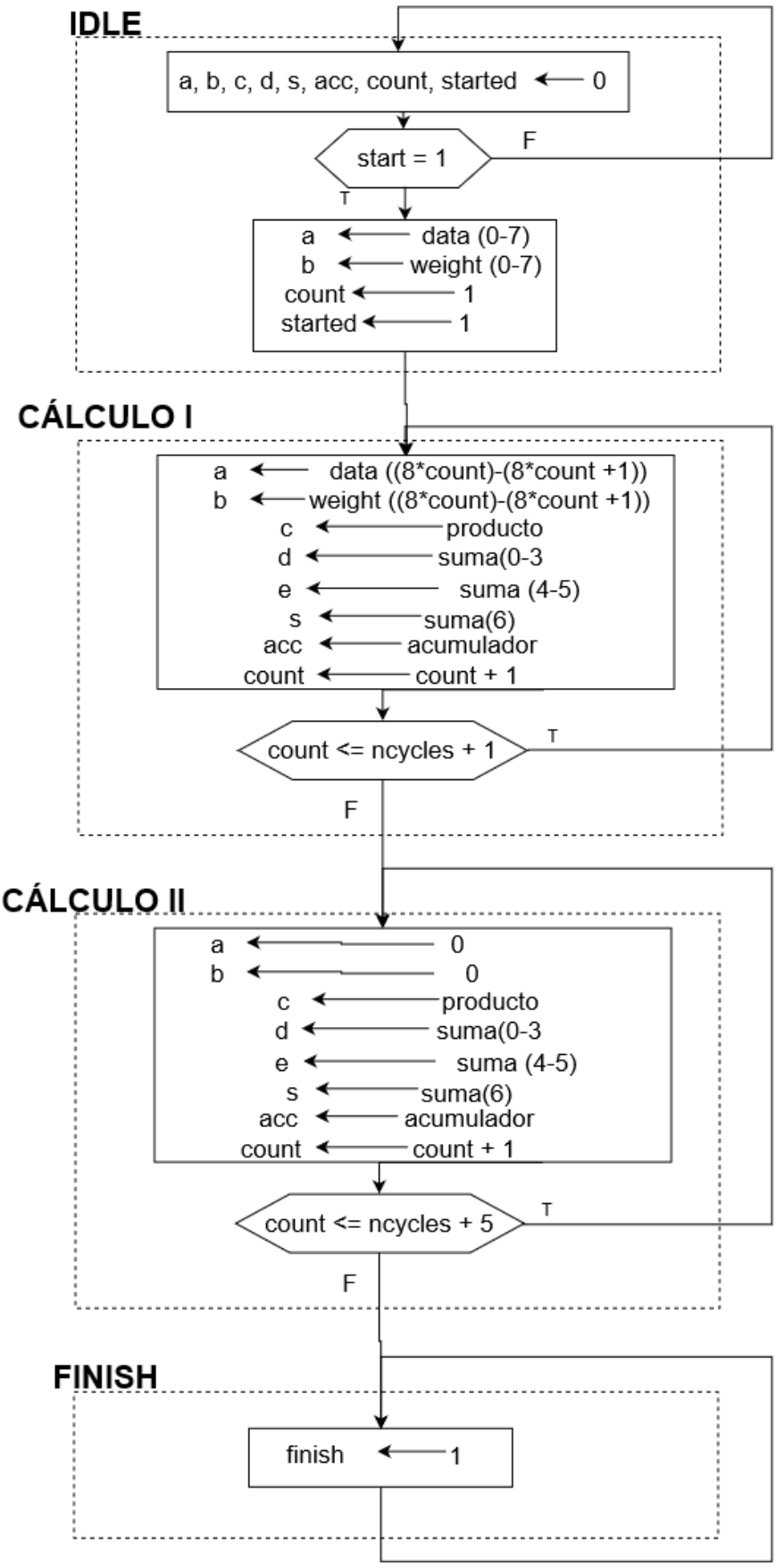


Ilustración 13. Diagrama ASMD del controlador

2.3.NEURONA ARTIFICIAL

Para poder implementar la neurona hacen falta otros módulos que aún no he explicado, el de redondeo y truncado y la función de activación [4]. Estos dos procesos están muy relacionados, ya que dependiendo del redondeo y del truncado de la señal obtenemos una precisión en la salida de la función de activación distinta.

A la salida del acumulador obtenemos una señal de tamaño variable, esta señal la modificamos para obtener una señal de ocho bits donde el bit más significativo corresponde al signo, los siguientes cuatro bits corresponden a la parte entera y los últimos tres a la parte decimal. Una vez que tenemos esta señal se aplica la función de activación, función sigmoide en este caso, que está formada por una LUT.

Con todo esto puedo formar la neurona, que se encuentra en Accelerator.vhd junto con accelerator_utils.vhd.

A continuación, muestro unas capturas de las pruebas de los *testbench* donde se puede ver el flujo de la ejecución para distinto número de entradas en la neurona. En las explicaciones que vienen a continuación el nombre de las señales corresponden a la nomenclatura de la ilustración 12.

2.3.1. NEURONA CON TRES ENTRADAS

La ejecución comienza cuando se activa la señal de *start* de forma que coincida con un flanco de subida del reloj. En el momento que esto ocurre se registran las señales de entrada y los pesos en *a* y *b* respectivamente. *a* y *b* son arrays de ocho señales de ocho bits, como en este caso solo hay tres entradas las cinco restantes se completan con ceros. Estas señales se introducen en los multiplicadores y en el segundo flanco de reloj el producto se ajusta a un tamaño de 19 bits y se registra en c. Estos valores entran a los primeros cuatro sumadores (S0 -S3) y el resultado posteriormente con el tercer flanco de reloj se registra en *d*. Después se obtiene el resultado de los siguientes dos sumadores (S4-S5) que se registran en *e*. En el quinto flanco de reloj el resultado del último sumador (S6) se registra en la señal *s*. Con el sexto y último flanco de reloj la salida del acumulador, *acc*, sufre el proceso de truncado y redondeo, *LUT_input*, y se introduce en la LUT para obtener el resultado final, *data_out*.

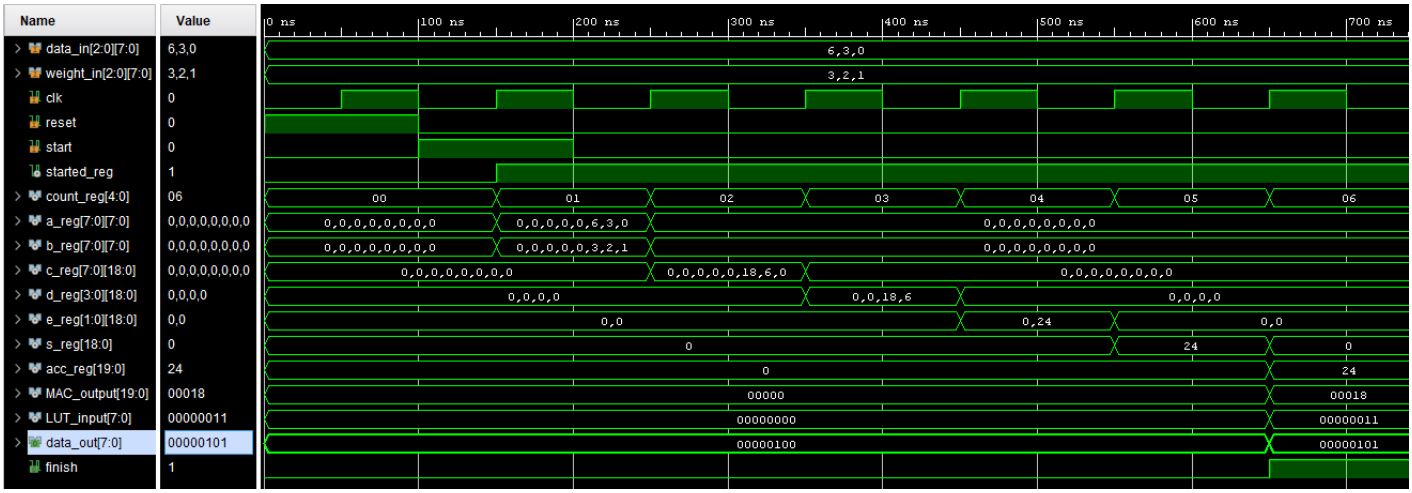


Figura de Vivado 6. Neurona con tres entradas: Testbench

2.3.2. NEURONA CON OCHO ENTRADAS

Cuando la neurona tiene ocho señales de entradas el flujo de ejecución es análogo a cuando hay tres señales de entrada, aunque en este caso no es necesario completar las señales *a* y *b* con ceros.

	t1	t2	t3	t4	t5	t6
a	Inputs (0-7)					
b	Pesos (0-7)					
c		Productos (0-7)				
d			Sum0-3 (0-7)			
e				Sum4,5 (0-7)		
s					Sum6 (0-7)	
acc						Acc (0-7)
LUT_IN						Valor asociado al acc (0-7)
Data_out						Valor de salida

Tabla 5. Neurona con ocho entradas: Flujo de datos

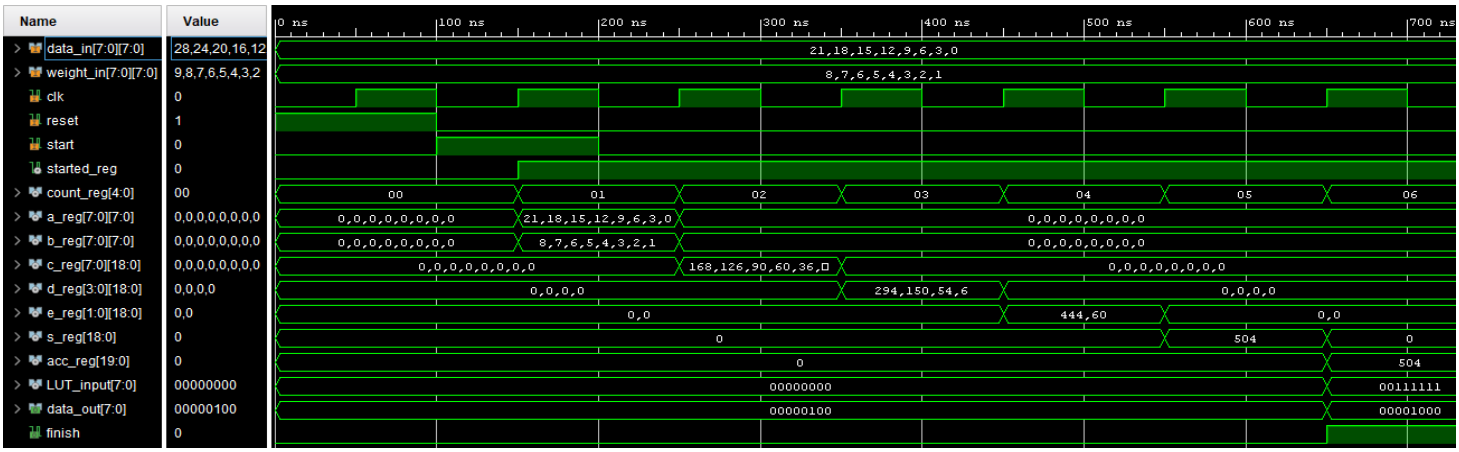


Figura de Vivado 7. Neurona con ocho entradas: Testbench

2.3.3. NEURONA CON 16 ENTRADAS

La neurona con 16 entradas necesita un ciclo de reloj más para la ejecución, respecto a la neurona con ocho entradas, en la siguiente table se puede ver el flujo de datos.

	t1	t2	t3	t4	t5	t6	t7
a	Inputs (0-7)	Inputs (8-15)					
b	Pesos (0-7)	Pesos (8-15)					
c		Producto(0-7)	Producto(8-15)				
d			Sum0-3 (0-7)	Sum0-3(8-15)			
e				Sum4,5 (0-7)	Sum4,5(8-15)		
s					Sum6 (0-7)	Sum6 (8-15)	
acc						Acc (0-7)	Acc (0-15)
LUT_IN						Valor asociado al acc (0-7)	Valor asociado al acc (0-15)
Data_out							Valor de salida

Tabla 6. Neurona con 16 entradas: Flujo de datos

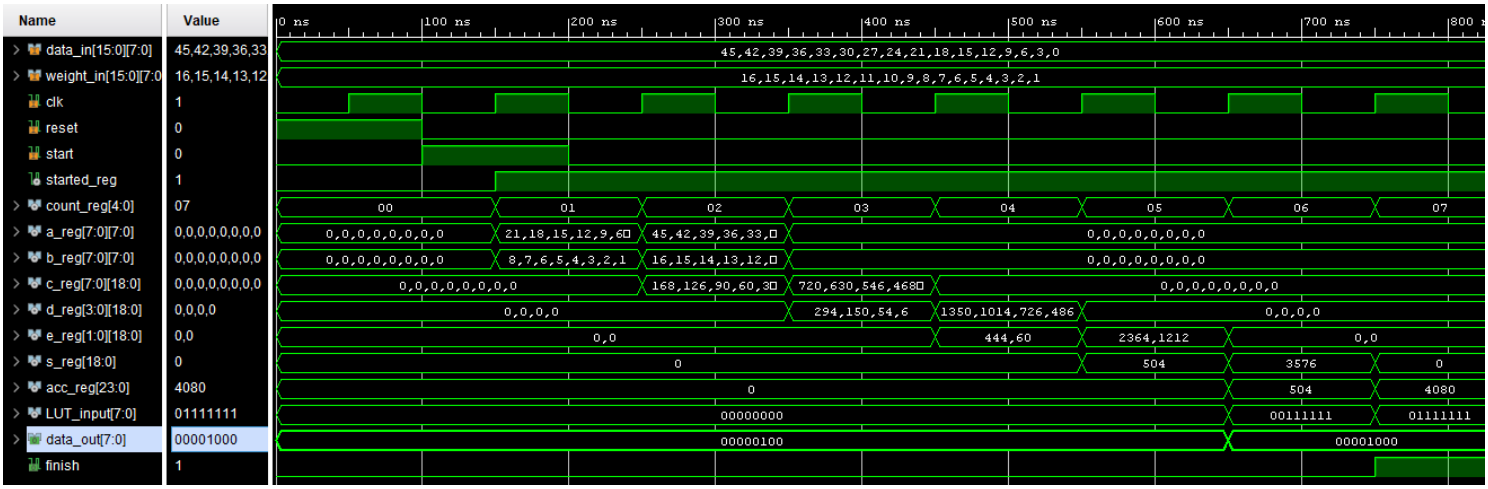


Figura de Vivado 8. Neurona con 16 entradas: Testbench

2.3.4. NEURONA CON 24 ENTRADAS

La neurona con 24 entradas requiere un ciclo de reloj más que la neurona de 16 entradas y en la siguiente tabla se puede ver el flujo de datos:

	t1	t2	t3	t4	t5	t6	t7	t8
a	Inputs (0-7)	Inputs (8-15)	Inputs (16-23)					
b	Pesos (0-7)	Pesos (8-15)	Pesos (16-23)					
c		Producto (0-7)	Producto (8-15)	Producto (16-23)				
d			Sum0-3 (0-7)	Sum0-3 (8-15)	Sum0-3 (16-23)			
e				Sum4,5(0-7)	Sum4,5 (8-15)	Sum4,5 (16-23)		
s					Sum6(0-7)	Sum6 (8-15)	Sum6 (16-23)	
acc						Acc (0-7)	Acc (0-15)	Acc (0-23)
LUT_IN						Valor asociado al acc (0-7)	Valor asociado al acc (0-15)	Valor asociado al acc (0-23)
Data_out								Valor de salida

Tabla 7. Neurona con 24 entradas: Flujo de datos

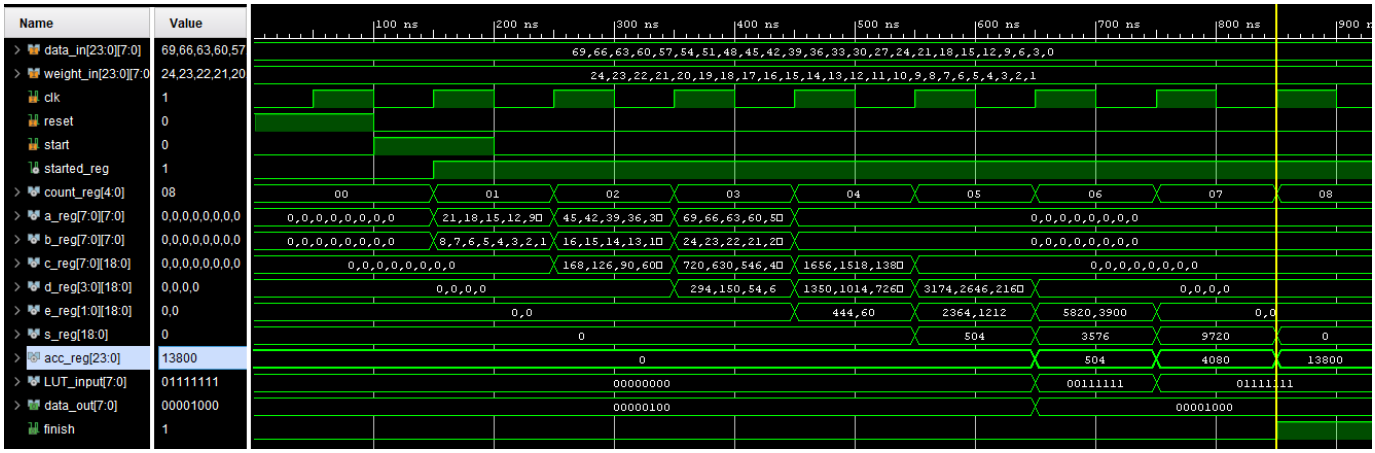


Figura de Vivado 9. Neurona con 24 entradas: Testbench

2.4. DISTANCIA ENTRE EL DISEÑO REALIZADO Y TRABAJOS PREVIOS

La implementación que se ha descrito en este apartado está basada en una realización previa de Santiago Romero Pomar [4]. Este sistema está formado por distintos módulos: multiplicador, acumulador, memoria ROM que almacena los pesos, fase de truncado y redondeo y función de activación. La interconexión entre estos módulos se realiza mediante un controlador del sistema de la forma descrita en la ilustración 5.

En el sistema final implementado en este proyecto se utilizan los mismos módulos que conforman la neurona anterior, excepto la memoria ROM que se utilizaba para almacenar los valores de los pesos y ahora no es necesaria, ya que en este nuevo diseño los pesos se deben introducir como señales de entrada en la neurona. Al módulo del acumulador original se le modifican los tamaños de las entradas para que pueda cumplir la función de sumador. El controlador del sistema también es distinto, porque incluye en la implementación el paralelismo a nivel de peso. El funcionamiento de dicho controlador está basado en el MACC descrito en la ilustración 12, a este diseño se le añaden el módulo de truncado y redondeo y el módulo de la función de activación para así obtener la neurona completa.

3. RESULTADOS

En este capítulo se exponen los resultados obtenidos a partir de las implementaciones realizadas. Primero se muestran los resultados obtenidos con distintas implementaciones de la neurona y estas implementaciones también se prueban sobre el hardware. Una vez que se ha analizado el funcionamiento de la neurona completa se procede a hacer las pruebas de las redes neuronales implementando un ejemplo sencillo. Los resultados obtenidos en ambos casos se comparan con trabajos anteriores. Finalmente se hace un análisis de la viabilidad de la utilización de este sistema como acelerador hardware.

3.1. NEURONA

La neurona desarrollada en el apartado anterior se encuentra en el programa Accelerator.vhd, que junto con su paquete accelerator_utils.vhd, ejercen la función de unir todos los módulos para formar el sistema completo.

La estructura implementada requiere que se introduzcan tanto las señales de entrada como los pesos de ocho bits cada una y los bits de control de *reset* y *start*. Una vez que se realizan los cálculos se obtiene la señal de salida de ocho bits junto con un bit de control *finish*.

Para comenzar con las pruebas se va a implementar la prueba más sencilla que se puede realizar, una neurona que emula la puerta lógica AND de dos entradas [1], que requiere una neurona de tres entradas.

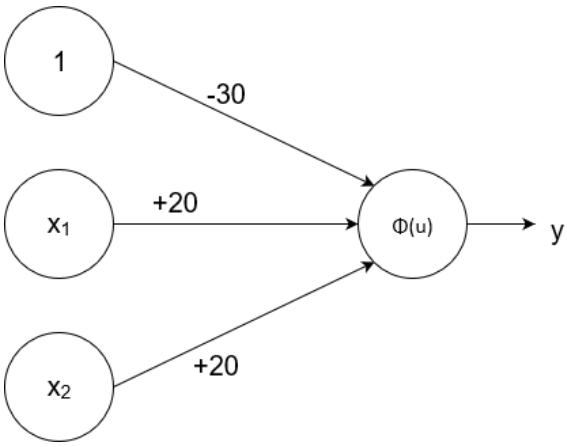


Ilustración 14. Neurona emulando una puerta AND

x_1	x_2	$y = \phi(u) = x_1 AND x_2$
0	0	$\phi(-30) < 0{,}5 \approx 0$
0	1	$\phi(-10) < 0{,}5 \approx 0$
1	0	$\phi(-10) < 0{,}5 \approx 0$
1	1	$\phi(+10) > 0{,}5 \approx 1$

Tabla 8. Neurona emulando una puerta AND

La señal de los pesos de la implementación cuenta con solo ocho bits, por tanto, no es posible representar estos valores. Entonces se va a realizar un procedimiento análogo al realizado en el proyecto de Santiago Romero Pomar para ajustar la precisión de los valores. Como el menor número de esta neurona es -30 y el menor número representable es -16, escalamos todos los pesos con el factor $\frac{30}{16} = \frac{15}{8} = 0{,}5\hat{3}$ y obtenemos que: $-30 \rightarrow -16$ y $20 \rightarrow 10{,}5$.

Además de implementar esta neurona de tres entradas se van a realizar pruebas de neuronas de ocho y dieciséis entradas que emulan el funcionamiento de unas puertas lógicas AND de siete y 15 entradas respectivamente. Para la selección de pesos de estas entradas hay que tener en cuenta que:

- Para que la salida se aproxime a 1 la función de activación tiene que obtener un valor mayor que 0,5 y para que la salida se aproxime a 0 la función de activación tiene que obtener un valor menor que 0,5.
- La función de activación $\phi(u) > 0{,}5\ si\ u > 0$ y $\phi(u) < 0{,}5\ si\ u < 0$

Por tanto, se puede concluir que:

- El peso de la señal bias es -16, ya que es el mínimo número representable.
- Los pesos de las señales de entrada deben cumplir las siguientes condiciones:

$$\left.\begin{matrix} Nentradas * w > 16 \\ (Nentradas - 1) * w < 16 \end{matrix}\right\}$$

  Siendo *Nentradas* el número de entradas de la puerta lógica, la neurona tiene una entrada más, la señal bias.

3.1.1. IMPLEMENTACIÓN

Para implementar estos ejemplos he realizado un programa "TOP" para cada uno de ello, a través del cual se introducen todas las señales ajustadas a Accelerator.vhd. Este programa recibe las señales de control *reset* y *start* a través de los botones de la placa, las señales de entrada las recibe a través de los switches, el valor final de la salida se muestra en los LEDs disponibles, al igual que la señal de *finish*. La señal de reloj la genera el reloj interno de la placa a 100Mhz, pero mediante un módulo IP se adapta a la frecuencia necesaria para cada caso.

3.1.1.1. PUERTA LÓGICA AND CON DOS ENTRADAS

Para poder emular esta función es necesario utilizar una neurona con tres entradas debido a la bias. La frecuencia más alta a la que funciona esta neurona es a 94,9 MHz.

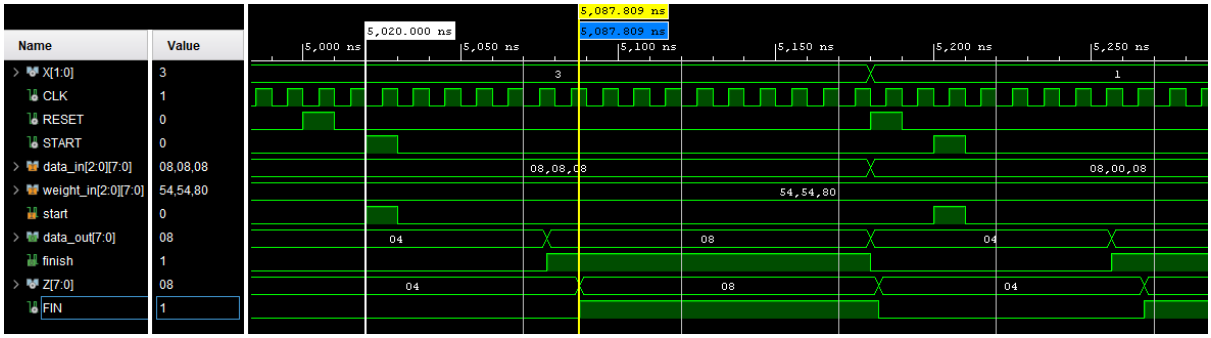


Figura de Vivado 10. AND con dos entradas: Testbench

En esta simulación se puede ver que la ejecución tarda seis ciclos de reloj desde que se lee la señal de *start* con el flanco de subida del reloj hasta que se obtiene el resultado. Sin embargo, para calcular el tiempo de ejecución completo se debe tener en cuenta desde el momento que se pulsa la señal de *start* hasta que se obtiene el resultado por la señal de salida registrada, y por tanto tarda 67,8ns.

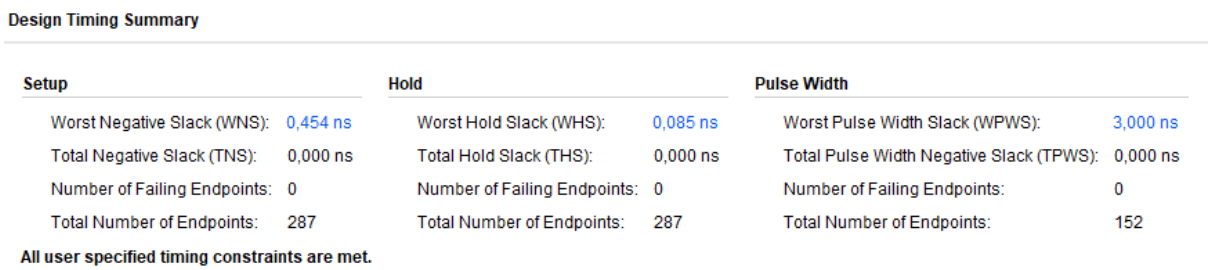
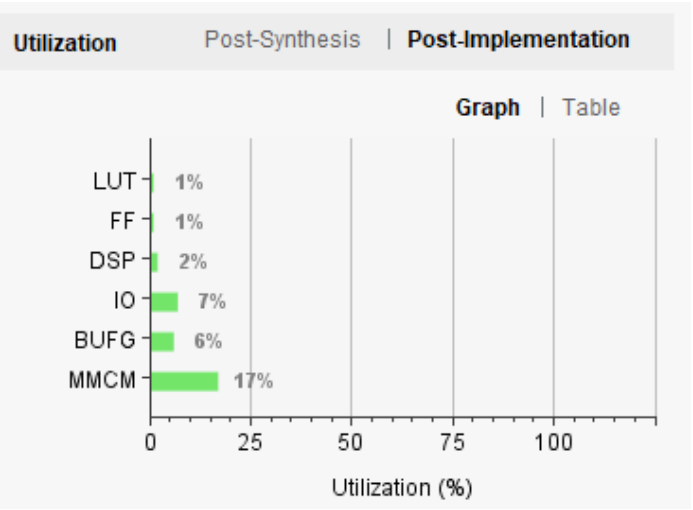
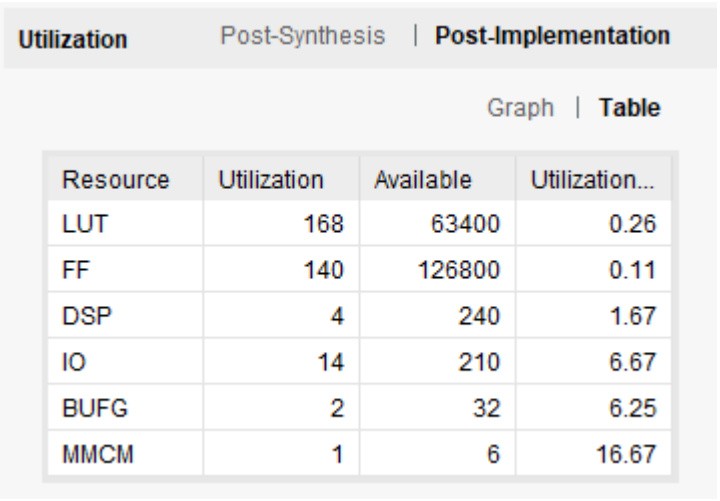


Figura de Vivado 11. AND con dos entradas: Análisis de tiempo

Name	Slice LUTs (63400)	DSPs (240)	Bonded IOB (210)	BUFGCTRL (32)	MMCME2_ADV (6)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)
AND_LG	179	4	14	2	1	145	64	179
A (Accelerator)	179	4	0	0	0		64	179
CSA_FINAL (CSA...	20	0	0	0	0		12	20
gen_stage_mux[0...	24	1	0	0	0		10	24
gen_stage_mux[1...	49	1	0	0	0		26	49
gen_stage_mux[2...	10	1	0	0	0		7	10
clk_94_9 (clk_aux)	0	0	0	2	1		0	0





Resource	Utilization	Available	Utilization...
LUT	168	63400	0.26
FF	140	126800	0.11
DSP	4	240	1.67
IO	14	210	6.67
BUFG	2	32	6.25
MMCM	1	6	16.67

Figura de Vivado 12. Multiplicador Vedic: Análisis de área

3.1.1.2. PUERTA LÓGICA AND CON SIETE ENTRADAS

Para poder emular esta función es necesario utilizar una neurona con ocho entradas debido a la bias. Los pesos los he seleccionado teniendo en cuenta que debía estar dentro de este rango:

$$\left.\begin{matrix} 7 * w_i > 16 \\ 6 * w_i < 16 \end{matrix}\right\} w_i \in [2{,}286\ 2, \hat{6}] \Longrightarrow w_i = 2{,}5$$

La frecuencia más alta a la que funciona esta neurona es a 88,5MHz.

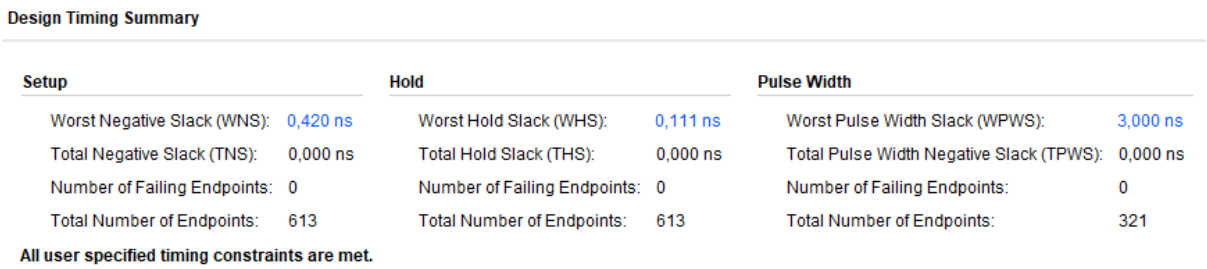
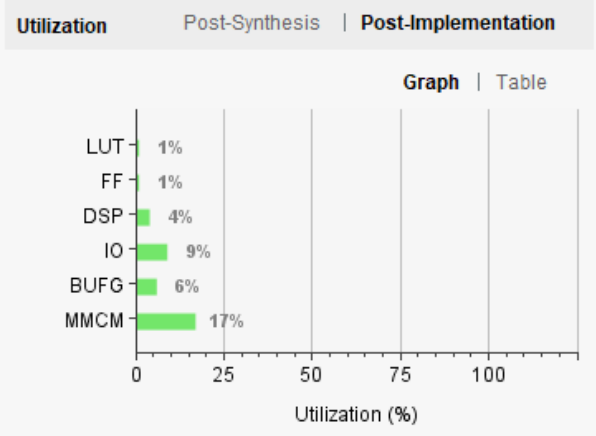
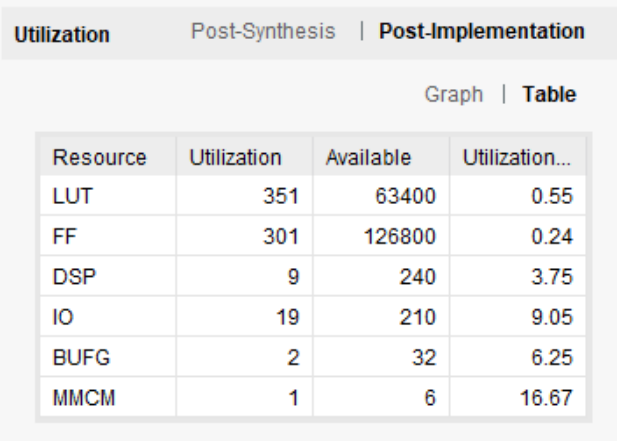


Figura de Vivado 13. AND con siete entradas: Análisis de tiempo

Name	Slice LUTs (63400)	DSPs (240)	Bonded IOB (210)	BUFGCTRL (32)	MMCME2_ADV (6)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)
AND8_LG	410	9	19	2	1	315	159	410
A (Accelerator)	410	9	0	0	0		159	410
CSA_FINAL (CSA_ACC)	20	0	0	0	0		12	20
gen_stage_mux[0].MUX_X (dadda_multi)	24	1	0	0	0		10	24
gen_stage_mux[1].MUX_X (dadda_multi_0)	49	1	0	0	0		29	49
gen_stage_mux[2].MUX_X (dadda_multi_1)	46	1	0	0	0		28	46
gen_stage_mux[3].MUX_X (dadda_multi_2)	12	1	0	0	0		14	12
gen_stage_mux[4].MUX_X (dadda_multi_3)	46	1	0	0	0		33	46
gen_stage_mux[5].MUX_X (dadda_multi_4)	13	1	0	0	0		13	13
gen_stage_mux[6].MUX_X (dadda_multi_5)	46	1	0	0	0		23	46
gen_stage_mux[7].MUX_X (dadda_multi_6)	12	1	0	0	0		14	12
clk_88_5 (clk_aux)	0	0	0	2	1		0	0





Resource	Utilization	Available	Utilization...
LUT	351	63400	0.55
FF	301	126800	0.24
DSP	9	240	3.75
IO	19	210	9.05
BUFG	2	32	6.25
MMCM	1	6	16.67

Figura de Vivado 14. AND con siete entradas: Análisis de área

3.1.1.3. PUERTA LÓGICA AND CON 15 ENTRADAS

Para poder emular esta función es necesario utilizar una neurona con 16 entradas debido a la bias. Los pesos los he seleccionado teniendo en cuenta que debía estar dentro de este rango:

$$\left.\begin{matrix} 15 * w_i > 16 \\ 14 * w_i < 16 \end{matrix}\right\} w_i \in [1{,}0\hat{6}\ 1{,}143] \Longrightarrow w_i = 1{,}125$$

La frecuencia más alta a la que funciona esta neurona es a 88,5MHz.

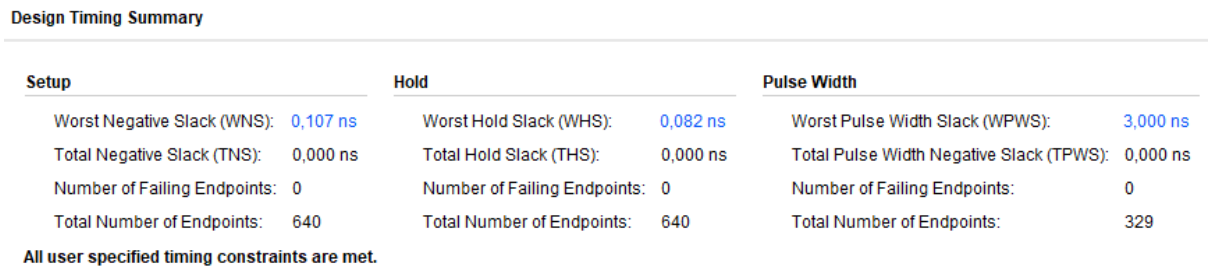
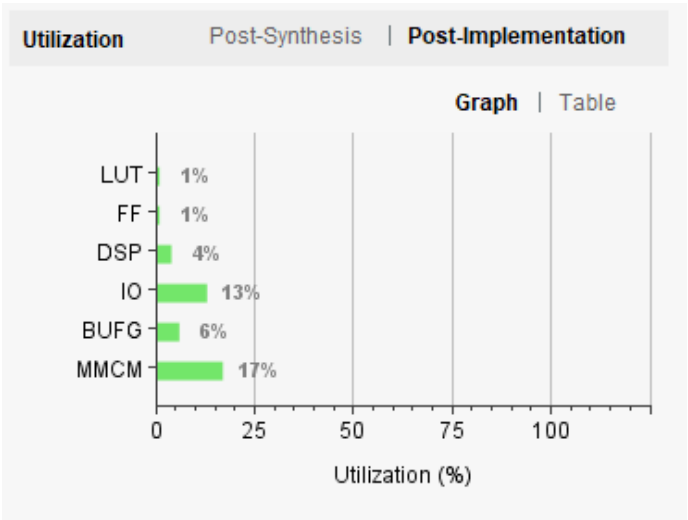
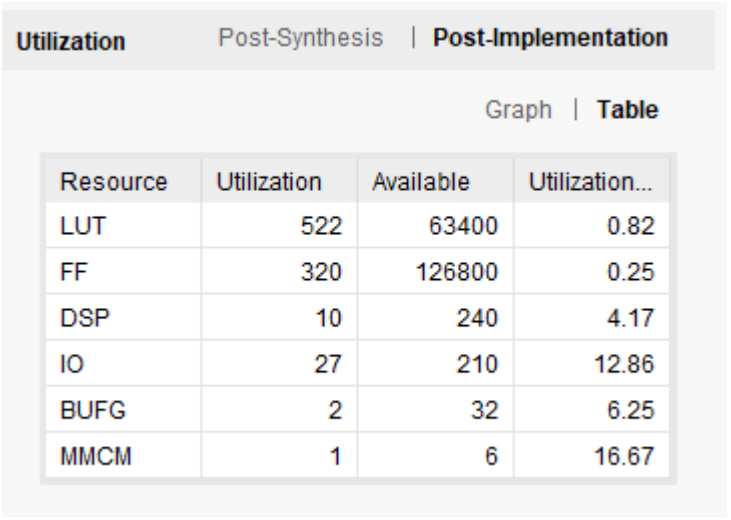


Figura de Vivado 15. AND con 15 entradas: Análisis de tiempo

Name	Slice LUTs (63400)	DSPs (240)	Bonded IOB (210)	BUFGCTRL (32)	MMCME2_ADV (6)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)
AND16_LG	561	10	27	2	1	323	176	561
A (Accelerator)	561	10	0	0	0		176	561
CSA_FINAL (CSA_ACC)	25	0	0	0	0		16	25
gen_stage_mux[0].MUX_X (dadda_multi)	26	1	0	0	0		9	26
gen_stage_mux[1].MUX_X (dadda_multi_0)	25	1	0	0	0		12	25
gen_stage_mux[2].MUX_X (dadda_multi_1)	25	1	0	0	0		9	25
gen_stage_mux[3].MUX_X (dadda_multi_2)	25	1	0	0	0		12	25
gen_stage_mux[4].MUX_X (dadda_multi_3)	25	1	0	0	0		11	25
gen_stage_mux[5].MUX_X (dadda_multi_4)	25	1	0	0	0		8	25
gen_stage_mux[6].MUX_X (dadda_multi_5)	26	1	0	0	0		13	26
gen_stage_mux[7].MUX_X (dadda_multi_6)	26	1	0	0	0		11	26
clk_88_5 (clk_aux)	0	0	0	2	1		0	0





Resource	Utilization	Available	Utilization...
LUT	522	63400	0.82
FF	320	126800	0.25
DSP	10	240	4.17
IO	27	210	12.86
BUFG	2	32	6.25
MMCM	1	6	16.67

Figura de Vivado 16. AND con 15 entradas: Análisis de área

3.1.2. ANÁLISIS DE RESULTADOS

Como se puede ver dependiendo del número de entradas a la neurona se obtiene una frecuencia de trabajo distinta, para la neurona con tres entradas la frecuencia máxima de trabajo es de 94,9 MHz (10,54 ns), pero para la neurona de ocho o 16 entradas la frecuencia máxima de funcionamiento es de 88,5 MHz. Respecto al área, se duplican los requisitos al pasar de la neurona de tres entradas a la de ocho, sin embargo, al pasar de una neurona de ocho entradas a una de 16 aumenta en un 50% el uso de LUT, pero la diferencia en los FF y los bloques DSP es mucho menor.

Para realizar la comparación entre este diseño y el realizado por Santiago Romero Pomar se utilizan los resultados obtenidos en la emulación de la puerta lógica de dos entradas, ya que, en ambos trabajos se ha realizado la misma prueba. En su trabajo se obtiene una frecuencia máxima de reloj de 71,71 MHz (14,05 ns) y requiere seis ciclos de reloj, por tanto, la ejecución tarda 84,3 ns. La implementación aquí explicada funciona a 94,9 MHz, requiere seis ciclos de reloj y la ejecución tarda 67,8 ns. Esto significa una aceleración de 1,24x.

Esto es una mejora significativa, sin embargo, hay que tener en cuenta que en la implementación original por cada entrada adicional hace falta un ciclo de reloj más en la ejecución y en el trabajo aquí desarrollado hace falta un ciclo de reloj más cada ocho entradas. Por lo que en el caso de una neurona de ocho entradas la implementación de Santiago Romero Pomar requiere once ciclos de reloj $14{,}05\,ns * (6+5) = 154{,}55\,ns$ y en la implementación de este trabajo la frecuencia de funcionamiento es de 88,5 Mhz (11,3 ns) y tarda $11{,}3\,ns * 6 = 67{,}8\,ns$. Lo que significa una aceleración de 2,28x.

En la gráfica que se muestra a continuación se puede ver como aumentaría la aceleración entre los trabajos en función del número de entradas de la neurona y como se puede observar el crecimiento sigue una función similar a la logarítmica.

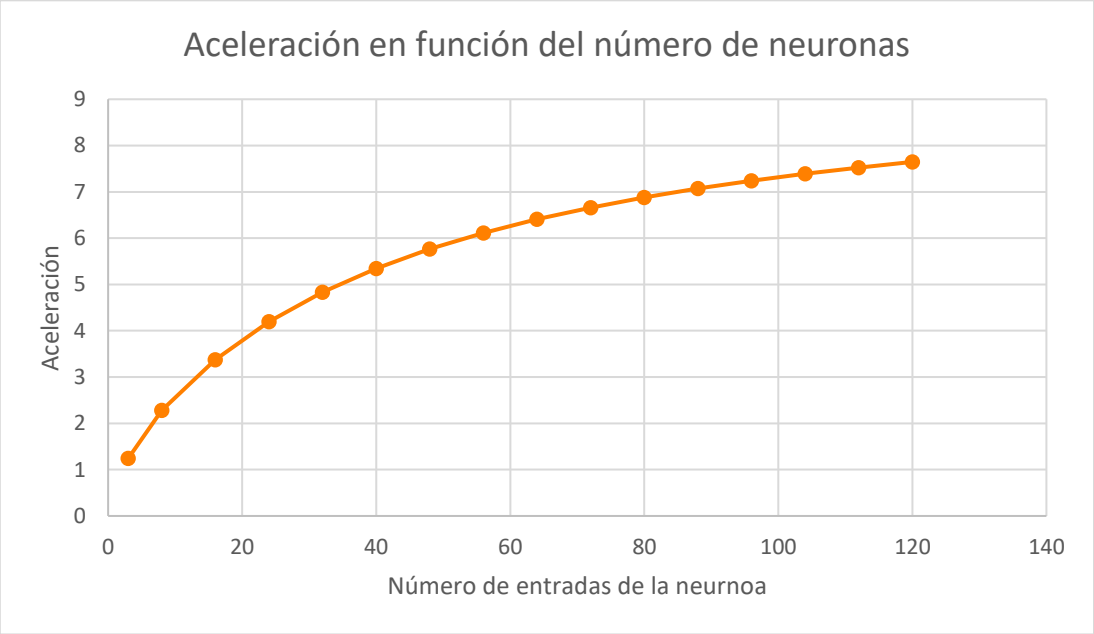


Ilustración 15. Resultados de aceleración de una neurona en función del número de entradas

Por otro lado, el diseño implementado en este trabajo tiene mayores requisitos de área, ya que el elemento limitante es el bloque DSP con un 1,67% de utilización, mientras que en el otro trabajo cada neurona utiliza un 1,25% del bloque DSP disponible.

3.2. RED NEURONAL

Una vez que la implementación de la neurona artificial está funcionando se realizan las pruebas de la red neuronal, simulando una puerta XNOR, ya que no puede ser implementado mediante una sola neurona, el ejemplo lo he tomado del curso de Coursera realizado [1]. La función XNOR se puede descomponer de esta forma:

$$x_1\ XNOR\ x_2 = [x_1\ AND\ x_2]\ OR\ [(NOT\ x_1) AND (NOT\ x_2)]$$

Esto implica que es necesaria una red con una capa intermedia con dos neuronas, una realizando la función $[x_1\ AND\ x_2]$ y la otra la función $[(NOT\ x_1) AND (NOT\ x_2)]$, además de la neurona bias. La neurona de la capa final debe realizar la función OR.

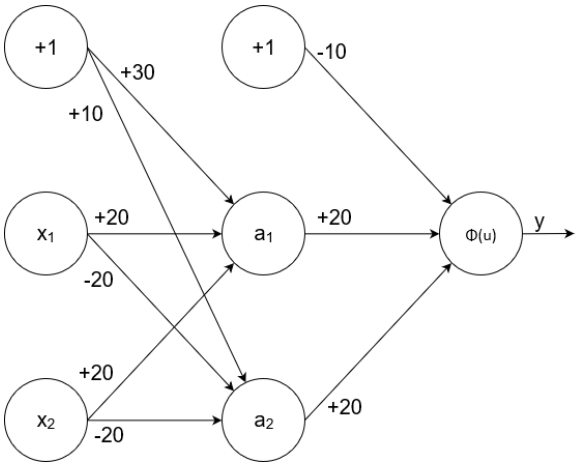


Ilustración 16. Red neuronal emulando una puerta XNOR

x_1	x_2	$a_1 = x_1\ AND\ x_1$	$a_2 = (NOT\ x_1) AND (NOT\ x_2)$	$y = a_1\ OR\ a_2 = \phi(u)$
0	0	$\phi(-30) < 0{,}5 \approx 0$	$\phi(+10) > 0{,}5 \approx 1$	$\phi(+10) > 0{,}5 \approx 1$
0	1	$\phi(-10) < 0{,}5 \approx 0$	$\phi(-10) < 0{,}5 \approx 0$	$\phi(-10) < 0{,}5 \approx 0$
1	0	$\phi(-10) < 0{,}5 \approx 0$	$\phi(-10) < 0{,}5 \approx 0$	$\phi(-10) < 0{,}5 \approx 0$
1	1	$\phi(+10) > 0{,}5 \approx 1$	$\phi(-30) < 0{,}5 \approx 0$	$\phi(+10) > 0{,}5 \approx 1$

Tabla 9. Red neuronal emulando una puerta XNOR

Al igual que antes, los pesos que se utilizan en este ejemplo no se pueden implementar en el sistema. Por tanto, se escalan de forma análoga al apartado anterior, resultando en: $|30| \rightarrow |16|$, $|20| \rightarrow |10{,}5|$ y $|10| \rightarrow |5{,}25|$.

3.2.1. IMPLEMENTACIÓN

Para implementar este ejemplo he realizado un programa cuyas señales entran y salen de la placa de forma análoga a la implementación de la puerta AND de dos entradas y también se genera la señal de reloj a través del reloj interno de la placa a 100 Mhz y mediante un módulo IP se adapta a la frecuencia necesaria para este caso que es 94 MHz.

Este programa introduce el bit de control *reset* a todas las neuronas del sistema. El bit de control *start* y las señales de entrada del sistema x_1 y x_2 se utilizan como las señales de entrada de las neuronas a_1 y a_2. Sin embargo, la neurona de la siguiente capa es un poco distinta, sus señales de entrada son las salidas de las neuronas a_1 y a_2 y su señal *start* se forma como una combinación de las señales de *finish* de las anteriores neuronas: *starts_a3 = finish _a1 AND finish _a2*.

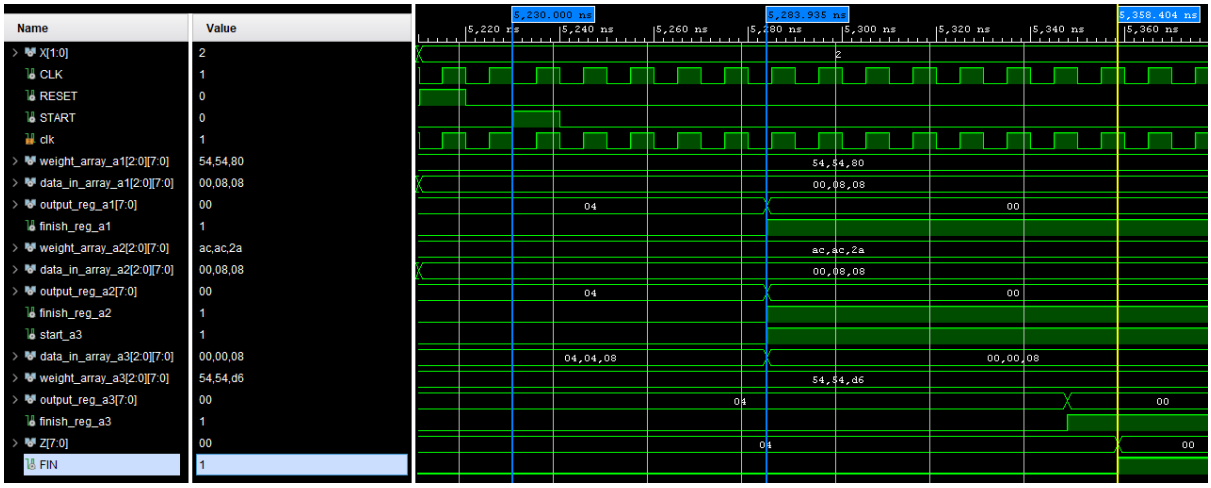


Figura de Vivado 17 . XNOR con dos entradas: Testbench

Como se puede apreciar en esta figura esta red neuronal tarda 12 ciclos de reloj para completarse, seis ciclos para cada capa de neuronas. Teniendo en cuenta el momento en el que se pulsa la señal de *start* hasta que se obtiene la señal de salida registrada transcurren 128,4 ns.

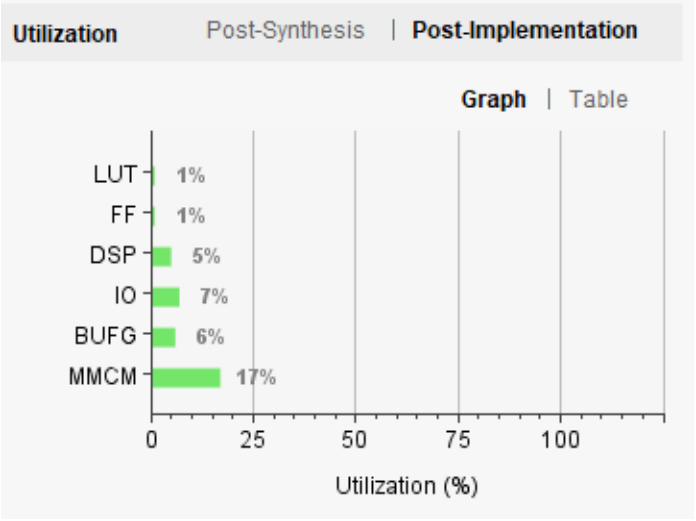
Design Timing Summary

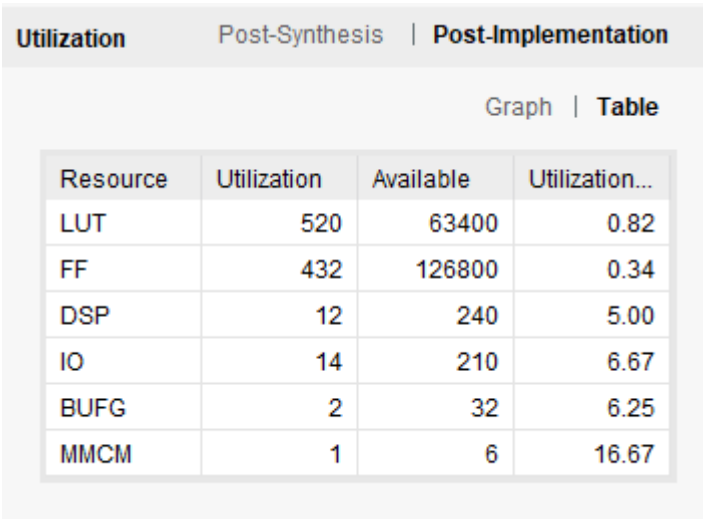
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0,034 ns	Worst Hold Slack (WHS):	0,028 ns	Worst Pulse Width Slack (WPWS):	3,000 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	890	Total Number of Endpoints:	890	Total Number of Endpoints:	451

All user specified timing constraints are met.

Figura de Vivado 18. XNOR con dos entradas: Análisis de tiempo

Name	Slice LUTs (63400)	DSPs (240)	Bonded IOB (210)	BUFGCTRL (32)	MMCME2_ADV (6)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)
XNOR_LG	596	12	14	2	1	443	202	596
A1 (Accelerator)	184	4	0	0	0		59	184
gen_stage_mux[0].MUX_X (dadda_multi_9)	24	1	0	0	0		11	24
gen_stage_mux[1].MUX_X (dadda_multi_10)	48	1	0	0	0		21	48
gen_stage_mux[2].MUX_X (dadda_multi_11)	11	1	0	0	0		9	11
LUT_8b (LUT_8)	6	0	0	0	0		4	6
A2 (Accelerator_0)	206	4	0	0	0		80	206
gen_stage_mux[0].MUX_X (dadda_multi_5)	49	1	0	0	0		24	49
gen_stage_mux[1].MUX_X (dadda_multi_6)	24	1	0	0	0		8	24
gen_stage_mux[2].MUX_X (dadda_multi_7)	25	1	0	0	0		8	25
LUT_8b (LUT_4)	6	0	0	0	0		3	6
A3 (Accelerator_1)	207	4	0	0	0		73	207
gen_stage_mux[0].MUX_X (dadda_multi)	24	1	0	0	0		11	24
gen_stage_mux[1].MUX_X (dadda_multi_2)	59	1	0	0	0		26	59
gen_stage_mux[2].MUX_X (dadda_multi_3)	22	1	0	0	0		10	22
LUT_8b (LUT)	19	0	0	0	0		9	19
clk_94 (clk_aux)	0	0	0	2	1		0	0





Resource	Utilization	Available	Utilization...
LUT	520	63400	0.82
FF	432	126800	0.34
DSP	12	240	5.00
IO	14	210	6.67
BUFG	2	32	6.25
MMCM	1	6	16.67

Figura de Vivado 19. XNOR con dos entradas: Análisis de área

3.2.2. ANÁLISIS DE RESULTADOS DE TIEMPO

En esta implementación he obtenido una frecuencia de trabajo máxima de 94 MHz (10,64 ns) con un tiempo de ejecución de 128,4 ns, frente a los 50 MHz (20 ns) y 168,35 ns de tiempo de ejecución obtenidos en el trabajo desarrollado por Santiago Romero Pomar. Lo que supone una aceleración de 1,31x. Al igual que antes, hay que tener en cuenta que la implementación realizada en este documento aumenta su el número de ciclos necesarios de forma más lenta.

Respecto al área utilizada esta implementación vuelve a tener mayores requisitos de área, un tercio más de DSP y de LUT, y más de dos tercios FF.

3.3. ANÁLISIS DE LA VIABILIDAD DE LA ACELERACIÓN HARDWARE DEL PROCESO DE INFERENCIA MEDIANTE FPGA

En el proceso de investigación de los trabajos existentes sobre las redes neuronales se trataba, en muchos de ellos, el tema de los aceleradores. En este apartado se va a analizar la posibilidad de utilizar la neurona implementada como un acelerador. A priori, esto parece un buen uso para el sistema, ya que con los switches y los botones disponibles en la placa no se puede alcanzar el máximo rendimiento del hardware disponible.

Para analizar la viabilidad de este uso he calculado la tasa de transferencia necesaria en una neurona con ocho señales de entrada. Por tanto, he utilizado como referencia la implementación de la AND con 7 entradas, que tenía una frecuencia de reloj de 88,5 MHz.

Señales necesarias:

- Señal de entrada que indica el número de entradas que tendría la neurona: $8\ bits$.
- Señal de entrada con los datos de entrada: $8 \times 8 = 64\ bits$, en este caso.
- Señal de entrada con el valor de los pesos: $8 \times 8 = 64\ bits$, en este caso.
- Señal de salida: $8\ bits$.

La implementación de una neurona de ocho entradas utiliza sumadores diferentes en cada ciclo de reloj. Esto permite que se puede comenzar la ejecución una neurona en cada ciclo de reloj utilizando el área propia de una sola neurona.

Para una neurona se requiere una velocidad de transferencia de:

$$144\ bits * 88{,}5\ MHz = 12.787{,}2 Mbits/s \approx 12{,}8\ Gbits/s$$

Analizando el área ocupada por la neurona de ocho entradas el elemento MMCM aparenta ser el limitante, sin embargo, este es el encargado de generar la señal de reloj y como todas las neuronas podrían funcionar a la misma velocidad el diseño no se vería limitado por este componente. Tampoco nos limitaría IO, ya que el problema de las señales de entrada y salida se resolvería a través del USB. Los BUFG tampoco nos limitarían el diseño, ya que es un componente que utiliza la entidad TOP y no la del acelerador. Por tanto, el componente limitante serán los recursos DSP y cada neurona utiliza un 3,75% de los disponibles, lo que nos permitiría ejecutar hasta 26 neuronas simultáneamente.

Para 26 neuronas se requiere una velocidad de transferencia de:

$$26 * 12{,}7872 Gbits/s = 332{,}5\ Gbits/s$$

El puerto USB del que dispone nuestra placa es el FTDI FT2232HQ USB-UART [9]. Este es un USB 2.0 High Speed que alcanza velocidades de 480 Mb/s. Por tanto, no se puede utilizar la neurona implementada como un acelerador con el hardware que he estado trabajando.

Esta limitación no se puede resolver fácilmente, ya que no es una limitación específica de esta placa, sino que los requisitos de transferencia de datos son muy exigentes. Por ejemplo, el bus PCI Express por un carril tiene una velocidad de 31,5 Gbits/s, lo que significa que serían necesarios 11 carriles en paralelo para poder hacer uso del área disponible al completo. Por tanto, con el diseño realizado la latencia de la neurona deja de ser un factor limitante en la implementación y el factor limitante pasa a ser la velocidad de transferencia de datos.

Todo el código desarrollado en este trabajo se encuentra en:
https://github.com/carmenmendizabal/NeuronaArtificial

4. CONCLUSIONES Y LÍNEAS FUTURAS

En este capítulo explico las conclusiones que se han obtenido de este proyecto y las líneas futuras que se pueden seguir para continuar realizando mejoras al trabajo presentado.

4.1. CONCLUSIONES

Actualmente las redes neuronales artificiales están sufriendo un gran crecimiento, gracias a su utilidad en el tratamiento de imágenes, el reconocimiento del lenguaje o el diagnóstico médico. Sin embargo, un problema de las redes neuronales es su implementación, ya que hay que equilibrar el gasto de energía, los requerimientos de hardware y la velocidad de computación, asimismo, es importante la flexibilidad de los diseños implementados. En este contexto, las FPGAs ofrecen la posibilidad de aprovecharse del paralelismo presente en las redes neuronales manteniendo un alto nivel de flexibilidad, por lo tanto, parecen ofrecer una buena solución para el problema planteado.

El objetivo principal de este trabajo era mejorar la latencia de una neurona ya implementada previamente. Para poder reducir la latencia se ha realizado una implementación que aprovecha el paralelismo a nivel de peso de las neuronas. Con esta implementación una neurona artificial de ocho o más entradas ha alcanzado una frecuencia máxima de funcionamiento de 88,5 MHz (11,3 ns), esta neurona solo requiere seis ciclos de reloj en el caso de ocho entradas y el número de ciclos aumenta con el número de entradas. Esto supone como mínimo una aceleración de 1,24x respecto al diseño sobre el que me basaba. Para el caso de la red neuronal, he conseguido una frecuencia máxima de funcionamiento de 94MHz (10,6ns) lo que supone una aceleración de 1,3 respecto a la ya implementada. La frecuencia máxima de funcionamiento puede variar ligeramente dependiendo de la complejidad del sistema y también varía el número de ciclos de reloj necesarios.

También se ha realizado un análisis de la viabilidad de la utilización de la placa FPGA como acelerador para una red. Pero siguiendo este camino se ha descubierto que el diseño realizado se ve limitado por la velocidad de transferencia de datos. Esto significa que la implementación descrita no tiene un uso práctico hoy en día, más allá del valor académico del trabajo.

En definitiva, he conseguido el objetivo principal de reducir la latencia del sistema respecto al diseño inicial. No obstante, tras los análisis de la implementación queda claro el sistema aquí desarrollado no tiene una utilidad clara.

4.2. LÍNEAS FUTURAS

El uso del diseño implementado no se ve limitado por la latencia del sistema, sino por la velocidad de la transferencia de datos. Por tanto, seguir reduciendo la latencia de la neurona artificial no mejoraría la velocidad de la función de inferencia. Para mejorar esta velocidad es necesario enfocar los futuros estudios en otros aspectos.

Un posible camino es la búsqueda de métodos para reducir la transferencia de datos requerida. Una forma efectiva de puede ser seguir métodos de compresión de datos o la implementación de

Otra forma de encarar el problema es explorar otras plataformas hardware en las que se pueda implementar una neurona, por ejemplo, las GPUs. Aunque estos dispositivos no sean capaces de implementar un sistema con una latencia tan baja como las FPGAs pueden estar más próximos a obtener un resultado más rápido gracias a no tener una velocidad de transferencia mucho mayor

La red neuronal planteada es capaz de realizar el proceso de inferencia, sin embargo, no es capaz de procesar el algoritmo de aprendizaje. Otra posible mejora del sistema sería incluir el algoritmo de propagación hacia adelante.

5. BIBLIOGRAFÍA

[1] A. Ng, «Coursera: Machine Learning,» [En línea]. Available: https://www.coursera.org/learn/machine-learning.

[2] V. Sze, Y.-H. Chen, T.-J. Yang y J. Emer, «Efficient Processing of Deep Neural Networks: A Tutorial and Survey,» *IEEE*, vol. 105, nº 12, pp. 2295-2329, 2017.

[3] A. R. Omondi y J. C. Rajapakse, FPGA Implementations of Neural Networks, Springer, 2006.

[4] S. R. Pomar, *Low-latency perceptron design and implementation in FPGA*, Madrid, 2018.

[5] M. Porrmann, U. Witkowski, H. Kalte y U. Ruckert, «Implementation of artificial neural networks on a reconfigurable hardware Accelerator,» de *Proceedings 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, 2002.

[6] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie y X. Zhou, «DLAU: A Scalable Deep Learning Accelerator Unit on FPGA,» *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* , vol. 36, nº 3, pp. 513-517, 2017.

[7] K. Rajaneesh y M. Bharathi, «A NovelHigh Performance Implementation of 64 Bit MAC Units and Their Delay Comparison,» *Int. Journal of Engineering Research and Applications*, vol. 4, nº 6, pp. 122-127, 2014.

[8] K. Deergha Rao, C. Gangadhar y P. K. Korrai, «FPGA implementation of complex multiplier using minimum delay Vedic real multiplier architecture,» de *2016 IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics Engineering (UPCON)*, Varanasi, India, 2017.

[9] Digilent, «Nexys 4 DDR Reference Manual,» [En línea]. Available: https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/reference-manual.

[10] Google, «Google Scholar,» [En línea]. Available: http://www.scholar.google.es/.

ANEXO A: ASPECTOS ÉTICOS, ECONÓMICOS, SOCIALES Y AMBIENTALES

En este anexo se explican los impactos éticos, económicos, sociales y ambientales que puede causar este proyecto.

A.1 INTRODUCCIÓN

El proyecto presentado ha sido realizado como trabajo académico dentro del marco universitario, por tanto, se trata de una actividad de investigación básica.

Actualmente el uso de la inteligencia artificial y más concretamente de las redes neuronales está extendido para una amplia variedad de aplicaciones, por tanto, el desarrollo en este campo influye en todos los aspectos de la vida. Entre los usos destacados se encuentra el diagnóstico médico.

Este trabajo en concreto se centra en un sistema de baja latencia, adecuado por tanto para su implantación en sistemas de tiempo real. Los usos específicos de estos sistemas son, por ejemplo, la implementación de vehículos autónomos basados en la clasificación de imágenes y de asistentes virtuales basados en el reconocimiento del lenguaje.

El diseño se realiza sobre una placa FPGA, que es capaz de optimizar el hardware disponible para una implementación de baja latencia. Adicionalmente, estas placas aportan un amplio nivel de flexibilidad al ser reconfigurables.

En la siguiente tabla se exponen algunos impactos que puede generar el proyecto realizado, en función del ámbito de influencia y el momento del ciclo de vida en el que se produce el efecto.

<table>
<tr><th></th><th>Aspectos éticos y profesionales</th><th>Aspectos económicos</th><th>Aspectos sociales</th><th>Aspectos ambientales</th></tr>
<tr><td>Materia prima y recursos naturales</td><td colspan="4">No se aprecian impactos significativos</td></tr>
<tr><td>Diseño, producción y pruebas</td><td>Pérdida de privacidad</td><td colspan="2">No se aprecian impactos significativos</td><td>Gasto energético</td></tr>
<tr><td>Empaquetado y distribución</td><td colspan="4">No se aprecian impactos significativos</td></tr>
<tr><td>Uso y mantenimiento</td><td>Pérdida de privacidad</td><td>Posible destrucción de ciertos empleos</td><td>Salud</td><td>Gasto energético</td></tr>
<tr><td>Reutilización, reciclaje y desecho</td><td colspan="3">No se aprecian impactos significativos</td><td>Material reconfigurable</td></tr>
</table>

Tabla 10. Impactos provocados por el proyecto

A.2 DESCRIPCIÓN DE IMPACTOS RELEVANTES RELACIONADOS CON EL PROYECTO

Para un correcto funcionamiento de las redes neuronales es necesario tener acceso a una gran cantidad de datos, especialmente en la fase de diseño del sistema. Por tanto, se deben establecer procedimientos que protejan los derechos de privacidad.

Entre los usos más importantes y beneficiosos de las redes neuronales se encuentra el diagnóstico médico. El uso de estos sistemas es especialmente efectivo diagnosticando enfermedades relacionadas con la genómica como el autismo o la atrofia muscular espinal. También es una buena herramienta para la detección de distintos cánceres como el de piel, de mama o tumores. Esto se encuentra dentro de los impactos sociales ya que su uso supone un beneficio directo en la salud de las personas.

Otra aplicación destacada dentro del campo de las redes neuronales es el desarrollo de los vehículos autónomos, reduciendo la influencia del error humano en accidentes. Sin embargo, hay que tener en cuenta que la conducción da empleo a un alto número de personas como los taxistas o los camioneros.

A.3 ANÁLISIS DETALLADO DE ALGUNO DE LOS PRINCIPALES IMPACTOS

En el trabajo realizado se buscaba una implementación de una red neuronal con una latencia lo más baja posible, esta característica es adecuada para los sistemas con requisitos estrictos de tiempo. Es decir, el sistema implementado puede ser usado para el procesamiento de datos en tiempo real. Los usos principales de estos sistemas son los vehículos autónomos y los asistentes virtuales.

Los asistentes virtuales recogen gran cantidad de información que en muchos casos puede contener datos sensibles por lo que es necesario establecer suficientes medidas de seguridad para no causar una pérdida de privacidad a sus usuarios. Y como ya se ha comentado anteriormente, la sustitución de vehículos convencionales por vehículos autónomos puede provocar la pérdida de empleo a un alto número de trabajadores.

El sistema diseñado en este trabajo necesita ser implementado sobre una placa FPGA, estas placas se caracterizan por ser reconfigurables, a diferencia de las placas ASIC. Esto permite reutilizar el mismo material para distintas aplicaciones e implica una reducción del impacto medioambiental.

A.4 CONCLUSIONES

Para concluir hay que destacar que los sistemas de inteligencia artificial tienen usos en una gran cantidad de aspectos de la vida cotidiana y generalmente los impactos creados son beneficiosos, sin embargo, es importante reducir en la mayor medida posible los impactos perjudiciales como lo son la pérdida de empleo o de privacidad.

ANEXO B: PRESUPUESTO ECONÓMICO

Este proyecto lo he realizado en aproximadamente 320h. Estas horas están divididas a lo largo de cuatro meses en los cuales he trabajado una media de 20 días y 4 horas por día.

Para poder realizar las tareas propuestas he necesitado el uso de un ordenador con un coste aproximado de 1.500€ y para realizar las implementaciones y las pruebas he usado la placa Nexys 4 DDR con un coste de 230€.

También he utilizado el software de Vivado, pero este no me ha supuesto ningún coste debido a que la Universidad Politécnica de Madrid me lo ha suministrado.

La tabla de presupuesto del proyecto es:

COSTE DE MANO DE OBRA (coste directo)	**Horas**	**Precio/hora**	**Total**
	320	15 €	**4.800 €**

COSTE DE RECURSOS MATERIALES (coste directo)	**Precio de compra**	**Uso en meses**	**Amortización (en años)**	**Total**
Ordenador personal (Software incluido)	1.500,00 €	6	5	150,00 €
Nexys 4 DDR	230,00 €	6	5	23,00 €
Licencia Vivado	-			

COSTE TOTAL DE RECURSOS MATERIALES	**173,00 €**

GASTOS GENERALES (costes indirectos)	15%	sobre CD	**745,95 €**
BENEFICIO INDUSTRIAL	6%	sobre CD+CI	**343,14 €**

MATERIAL FUNGIBLE

Impresión	**50,00 €**

SUBTOTAL PRESUPUESTO		**6.112,09 €**
IVA APLICABLE	21%	**1.283,54 €**

TOTAL PRESUPUESTO	**7.395,63 €**