

UNIVERSIDAD POLITÉCNICA DE
MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
DE TELECOMUNICACIÓN



GRADO EN INGENIERÍA DE TECNOLOGÍAS Y
SERVICIOS DE TELECOMUNICACIÓN

TRABAJO DE FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN SOBRE
FPGA DE UNA NEURONA ARTIFICIAL
CONFIGURABLE

MARK JOHN REALISTA QUIOCCHO

ENERO 2021

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO DE FIN DE GRADO

Título: DISEÑO E IMPLEMENTACIÓN SOBRE FPGA DE UNA NEURONA ARTIFICIAL CONFIGURABLE

Autor: MARK JOHN REALISTA QUIOCHE

Tutor: PABLO ITUERO HERRERO

Ponente:

Departamento: DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

MIEMBROS DEL TRIBUNAL

Presidente:

Vocal:

Secretario:

Suplente:

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

Madrid, a de de 2021

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE
TELECOMUNICACIÓN



GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS
DE TELECOMUNICACIÓN

TRABAJO DE FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN SOBRE
FPGA DE UNA NEURONA ARTIFICIAL
CONFIGURABLE

AUTOR: MARK JOHN REALISTA QUIOCHE

TUTOR: PABLO ITUERO HERRERO

PONENTE:

ENERO 2021

Resumen

Este Trabajo de Fin de Grado se enmarca en el ámbito de la Inteligencia Artificial y el Aprendizaje Automático basado en algoritmos que se inspiran en el funcionamiento conocido del cerebro humano. En concreto, este trabajo plantea el diseño, el desarrollo y la implementación de una neurona artificial (elemento básico de una red neuronal artificial) configurable entre precisión y latencia. Para lograr este objetivo, se ha realizado, en primer lugar, un estudio de la literatura existente, poniendo un énfasis especial en la búsqueda de multiplicadores configurables y de métodos de implementación de funciones de activación.

A continuación, se ha diseñado e implementado en VHDL, utilizando la herramienta Vivado de Xilinx, un multiplicador serie-paralelo configurable capaz de realizar multiplicaciones en binario y en complemento a 2 entre dos operandos $A(m)$ y $B(n)$ introducidos en serie y en paralelo respectivamente, siendo m un número cualquiera y $n = 4, 8$ o 16 bits. Este multiplicador se basa en 4 multiplicadores de 4 bits que, o bien se pueden utilizar independientemente, o bien se pueden enlazar mediante la acción de multiplexores dando lugar a 2 multiplicadores de 8 bits o a un multiplicador de 16 bits. Esta propiedad de configurabilidad ofrece un compromiso entre precisión y latencia, dando lugar a una mayor latencia cuanto mayor es la precisión y viceversa. De este multiplicador configurable se ha visto que presenta una latencia de $m + n$ ciclos de reloj, un throughput de $1/(m + n)$ operaciones/ciclo, una frecuencia de trabajo máxima de 330 MHz y una utilización de LUT y FF inferior al 1 %. Estos resultados de frecuencia y utilización se han obtenido para la placa Nexys 4 DDR de Digilent basada en la FPGA XC7A100T-1CSG324C Artix-7 de Xilinx.

Posteriormente, se ha diseñado e implementado, también en VHDL, un módulo que implementa una función sigmoide configurable del cual se pueden obtener 1, 2 o 4 funciones sigmoide con diferentes precisiones y latencias a partir del multiplicador configurable desarrollado anteriormente. Para la implementación de este módulo se ha seguido un proceso de diseño basado en síntesis de alto nivel a partir de la expansión en serie de Taylor por intervalos de la función sigmoide. De este módulo se ha visto que presenta un latencia máxima igual a $(m + n) \times 6 + 11$ ciclos de reloj, un throughput igual a la inversa de la latencia, una frecuencia de trabajo máxima de 175 MHz y una utilización de LUT y FF inferior al 1 %. Los resultados de frecuencia y utilización de este módulo, al igual que en el multiplicador configurable, se han obtenido para la placa Nexys 4 DDR de Digilent basada en la FPGA XC7A100T-1CSG324C Artix-7 de Xilinx.

Finalmente, se ha realizado una primera aproximación de la estructura que tendría una neurona artificial configurable construida a partir del multiplicador configurable y de la función sigmoide desarrollados anteriormente.

Palabras clave: Red neuronal artificial, Neurona artificial configurable, Multiplicador configurable, Función sigmoide, Precisión, Latencia, VHDL, Vivado, FPGA.

Abstract

This Final Degree Project is framed within the field of Artificial Intelligence and Machine Learning based on algorithms that are inspired by the known functioning of the human brain. In particular, this paper proposes the design, development and implementation of a configurable artificial neuron (basic unit of an artificial neural network) between precision and latency. In order to achieve this objective, a study of the existing literature has been made, placing special emphasis on finding configurable multipliers and on activation functions implementation methods.

Next, a configurable serial-parallel multiplier capable of performing multiplication in binary and in 2's complement between one factor $A(m)$ fed serially with an arbitrary wordlength m and another factor $B(n)$ stored in parallel with a configurable number of bits $n = 4, 8$ or 16 bits has been designed and implemented in VHDL using the Vivado tool from Xilinx. This multiplier is based on 4 4-bit multipliers that can either be used independently, or they can be linked by multiplexing, yielding 2 8-bit multipliers or a 16-bit multiplier. This property of configurability offers a compromise between precision and latency, leading to higher latency at higher levels of precision, and vice versa. From this configurable multiplier, it has been seen that it presents a latency of $m + n$ clock cycles, a throughput of $1/(m + n)$ operations/cycle, a maximum working frequency of 330 MHz and a utilization of LUT and FF less than 1 %. These frequency and utilization results have been obtained for the Digilent Nexys 4 DDR board based on the Xilinx Artix-7 FPGA XC7A100T-1CSG324C.

Subsequently, a module that implements a configurable sigmoid function from which 1, 2 or 4 sigmoid functions with different precisions and latencies can be obtained has been designed and implemented in VHDL from the configurable multiplier developed previously. For the implementation of this module, a design process based on high-level synthesis (HLS) has been followed from the interval Taylor series expansion of the sigmoid function. It has been seen that this module has a maximum latency equal to $(m + n) \times 6 + 11$ clock cycles, a throughput equal to the inverse of the latency, a maximum working frequency of 175 MHz and a utilization of LUT and FF less than 1 %. The frequency and utilization results for this module, as well as in the configurable multiplier, have been obtained for the Digilent Nexys 4 DDR board based on the Xilinx Artix-7 FPGA XC7A100T-1CSG324C.

Finally, a first approximation has been made of the structure that a configurable artificial neuron would have, built from the configurable multiplier and the sigmoid function previously developed.

Key words: Artificial neural network, Configurable artificial neuron, Configurable multiplier, Sigmoid function, Precision, Latency, VHDL, Vivado, FPGA.

Índice general

Resumen	I
Abstract	II
1. Introducción	1
1.1. Redes neuronales artificiales	1
1.2. Implementación hardware de redes neuronales artificiales	2
1.3. Neurona artificial	2
1.4. Objetivos del Trabajo de Fin de Grado	3
1.5. Planificación del Trabajo de Fin de Grado	4
1.6. Código del Trabajo de Fin de Grado	5
1.7. Estructura de la memoria	5
2. Multiplicador serie-paralelo configurable de 4, 8 y 16 bits	6
2.1. Algoritmo de multiplicación	7
2.2. Multiplicador serie-paralelo de 4 bits	8
2.2.1. Registros de desplazamiento PISO de m bits	9
2.2.2. Registro de desplazamiento PISO de n bits	10
2.2.3. Registros de desplazamiento PIPO	10
2.2.4. Registro de desplazamiento SIPO	11
2.2.5. Controlador del multiplicador	12
2.2.6. Elementos de procesamiento	14
2.2.7. Sumadores de salida	14
2.2.8. Ejemplo de multiplicación	15
2.3. Multiplicador serie-paralelo configurable de 4, 8 y 16 bits	18
2.3.1. 4 multiplicadores de 4 bits	20
2.3.2. 2 multiplicadores de 8 bits	21
2.3.3. 1 multiplicador de 16 bits	21
2.4. Análisis de resultados del multiplicador configurable	22
2.4.1. Multiplicador configurable: Latencia y throughput	22
2.4.2. Multiplicador configurable: Frecuencia	22
2.4.3. Multiplicador configurable: Utilización de recursos	23

3. Función de activación	24
3.1. Función sigmoide	25
3.2. Implementación de la función sigmoide	28
3.2.1. Bloque Sigmoid Left	30
3.2.2. Bloque Sigmoid Right	34
3.2.3. Bloque Sigmoid Middle	35
3.2.4. Controlador de la función sigmoide	39
3.2.5. Cuantificación de los datos	41
3.2.6. Ejemplo de la función sigmoide	41
3.3. Análisis de resultados de la función sigmoide	43
3.3.1. Función sigmoide: Latencia y throughput	43
3.3.2. Función sigmoide: Frecuencia	43
3.3.3. Función sigmoide: Utilización de recursos	43
4. Primera aproximación hacia una neuronal artificial configurable	45
5. Conclusiones	48
Bibliografía	50
A. Aspectos éticos, económicos, sociales y ambientales	51
A.1. Introducción	51
A.2. Descripción de impactos relevantes relacionados con el proyecto	51
A.3. Conclusiones	52
B. Presupuesto económico	53

Índice de tablas

2.1. Tabla de multiplexores	20
3.1. Aproximación por tramos lineales en el intervalo [-5,5]	26
3.2. Bloque sigmoid left - Cronograma	32
3.3. Bloque sigmoid right - Cronograma	35
3.4. Bloque sigmoid middle - Cronograma	38

Índice de figuras

1.1. Arquitectura de una red neuronal artificial con dos capas ocultas	1
1.2. Arquitectura de una neurona artificial	3
1.3. Planificación temporal del TFG	4
2.1. Multiplicador de 4 bits	8
2.2. Registro de desplazamiento PISO de m bits	9
2.3. Registro de desplazamiento PIPO	10
2.4. Registro de desplazamiento SIPO	11
2.5. Controlador del multiplicador configurable - Diagrama ASMD	13
2.6. Testbench - Multiplicador de 4 bits	15
2.7. Testbench - Controlador del multiplicador	15
2.8. Testbench - Registro PISO_M para $A(m)$	16
2.9. Testbench - Registro PISO_M para $C_A(m)$	16
2.10. Testbench - Registro PIPO para $B(n)$	16
2.11. Testbench - Registro PIPO para $C_B(n)$	17
2.12. Testbench - PE y cadena de sumadores	17
2.13. Testbench - Registro SIPO	18
2.14. Multiplicador configurable de 4, 8 y 16 bits	19
2.15. Testbench - 4 multiplicadores de 4 bits	20
2.16. Testbench - 2 multiplicadores de 8 bits	21
2.17. Testbench - 1 multiplicador de 16 bits	22
2.18. Multiplicador configurable - Análisis de tiempo	23
2.19. Multiplicador configurable - Análisis de utilización de recursos	23
3.1. Curva de la función sigmoide	25
3.2. Aproximación polinómica de la función sigmoide	27
3.3. Gráficas de error de la expansión en serie de Taylor por intervalos	27
3.4. Arquitectura de la función sigmoide	28
3.5. Bloque sigmoid left - Diagrama de flujo	31
3.6. Bloque sigmoid left - Planificación temporal	31
3.7. Bloque sigmoid left - Ruta de datos	32

3.8. Controlador del bloque sigmoid left - Diagrama ASMD	33
3.9. Bloque sigmoid right - Ruta de datos	34
3.10. Controlador del bloque sigmoid right - Diagrama ASMD	35
3.11. Bloque sigmoid middle - Diagrama de flujo	36
3.12. Bloque sigmoid middle - Planificación temporal	37
3.13. Bloque sigmoid middle - Ruta de datos	38
3.14. Controlador del bloque sigmoid middle - Diagrama ASMD	38
3.15. Controlador de la función sigmoide - Diagrama ASMD	39
3.16. Testbench - Función sigmoide	41
3.17. Testbench - Función sigmoide para $x \in (-\infty, -1,5]$	42
3.18. Testbench - Función sigmoide para $x \in (-1,5, 1,5)$	42
3.19. Función sigmoide - Análisis de tiempo	43
3.20. Función sigmoide - Análisis de utilización de recursos	44
4.1. Neurona artificial configurable	45
A.1. Impactos provocados por el proyecto	51
B.1. Presupuesto económico del proyecto	53

Capítulo 1

Introducción

1.1. Redes neuronales artificiales

Las redes neuronales artificiales (RNAs) son modelos computacionales que se enmarcan en el ámbito de la Inteligencia Artificial y el Aprendizaje Automático y que se inspiran en el principio de funcionamiento de las redes neuronales biológicas del cerebro humano. Las RNAs se utilizan para resolver problemas que, utilizando modelos de computación clásicos, serían difíciles de abordar.

Del mismo modo que nuestro cerebro está compuesto por neuronas interconectadas, una RNA está formada por un conjunto de unidades básicas conectadas entre sí denominadas neuronas artificiales. En una RNA, las neuronas artificiales se agrupan en lo que se denominan capas. Una capa es un conjunto de neuronas cuyas entradas provienen de una capa anterior (o de los datos de entrada en el caso de la primera capa) y cuyas salidas son las entradas de una capa posterior (o la salida de la red neuronal en el caso de la última capa). La capa que recibe los datos externos que alimentan la red neuronal se denomina capa de entrada, la capa que computa la salida de la red neuronal se conoce como capa de salida, y las capas que se sitúan entre la capa de entrada y la capa de salida reciben el nombre de capas ocultas.

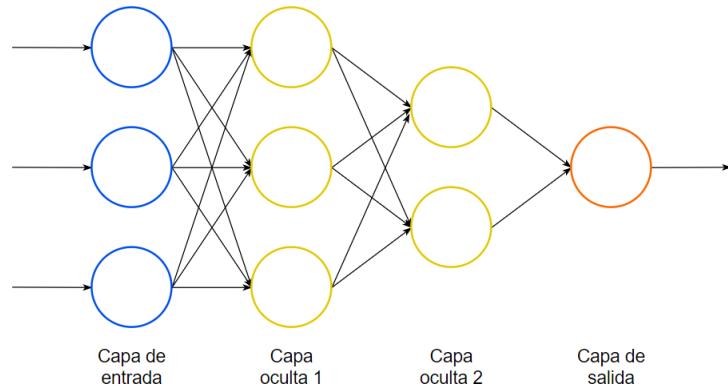


Figura 1.1: Arquitectura de una red neuronal artificial con dos capas ocultas

Las neuronas artificiales que forman una capa se conectan con las neuronas de la capa siguiente a través de unos enlaces que tienen un valor de peso asociado. De esta forma, el dato de salida de una neurona, antes de ser recibido por otras neuronas, se multiplica por el peso de los enlaces correspondientes. Estos pesos en los enlaces sirven para incrementar o inhibir el estado de activación de las neuronas posteriores. La capacidad de aprendizaje de las RNAs se relaciona precisamente con estas dos características mencionadas: la arquitectura de la red neuronal (número de neuronas, número de capas e interconexión entre ellas) y el ajuste de las interconexiones (asignación de pesos sinápticos a los enlaces neuronales).

A parte de esta capacidad de aprendizaje, las RNAs permiten reproducir otras características inherentes de las redes neuronales biológicas como (1) el paralelismo, procesado de información por parte de muchas neuronas simultáneamente, (2) la adaptabilidad, capacidad del cerebro de adaptarse a estímulos cambiantes, o (3) la tolerancia a fallos, la pérdida de algunas neuronas o conexiones no afecta de forma significativa al funcionamiento de la red neuronal. Hoy en día existen una gran variedad de aplicaciones en las que las RNAs tienen o pueden llegar a tener un gran impacto, como por ejemplo: procesado de señales, tratamiento de imágenes y vídeo, procesado del lenguaje, sistemas de diagnóstico médico y previsión financiera, entre otras.

1.2. Implementación hardware de redes neuronales artificiales

Una de las formas más versátiles de implementar redes neuronales artificiales es en software. Los programas software constituyen todo un entorno de desarrollo en los que se puede diseñar, construir, entrenar y probar redes neuronales artificiales. Sin embargo, la implementación en software, a diferencia de la implementación en hardware, no consigue aprovechar al completo el paralelismo inherente de las RNAs y se encuentra por debajo en cuanto a velocidad de computación.

Durante la década de los 80 y principios de los 90 se llevaron a cabo muchas investigaciones enfocadas en el diseño y en la implementación hardware de algoritmos basados en redes neuronales artificiales. Sin embargo, la mayoría de estas investigaciones no tuvieron éxito debido a que la tecnología por aquel entonces no estaba lo suficientemente desarrollada para adoptar redes neuronales a gran escala. Pero recientemente, gracias a los avances en la capacidad computacional de los procesadores, la implementación de RNAs en hardware está teniendo un importante resurgimiento.

Las estrategias más utilizadas para la implementación en hardware de redes neuronales artificiales son los ASIC (Application Specific Integrated Circuit) y las FPGAs (Field Programmable Gate Arrays). La principal ventaja que ofrecen los ASIC en la implementación de RNAs es su alto rendimiento (son muy eficientes en tiempo y en consumo). Sin embargo, este tipo de circuitos son caros a menos que se fabriquen en grandes cantidades y su tiempo de prototipado es elevado. Una alternativa a los ASIC son las FPGAs. En los últimos años, ha habido un interés creciente en implementar RNAs en FPGAs debido a que estas presentan un ratio rendimiento/coste mejor que los ASIC o incluso otro tipo de alternativas como los microprocesadores. Además, su capacidad de configurabilidad hace que se puedan utilizar en multitud de aplicaciones, como por ejemplo, en diferentes tipos de redes neuronales. Las FPGAs también permiten la implementación de diferentes tipos de paralelismo de RNAs tales como el paralelismo en el entrenamiento, el paralelismo por nodos, el paralelismo por pesos o el paralelismo a nivel de bit. Otra característica interesante de las FPGAs es que incluyen bloques DSP (Digital Signal Processor), los cuales están optimizados para realizar multiplicaciones, sumas u operaciones de multiplicación-suma (MAC). Por todo esto, a día de hoy las FPGAs son de las mejores opciones para la implementación de RNAs.

1.3. Neurona artificial

Como se comentó anteriormente, una neurona artificial es la unidad básica de una red neuronal artificial. Los elementos básicos de una neurona artificial son (1) una serie de entradas que reciben una serie de señales $x = (x_1, x_2, \dots, x_n)^T$, (2) un conjunto de co-

nexiones sinápticas representada por un conjunto de pesos $w = (w_1, w_2, \dots, w_n)^T$, y (3) una función de activación que relaciona la entrada sináptica total (u) con la salida de la neurona (y). La estructura principal de una neurona artificial se muestra en la siguiente figura [1]:

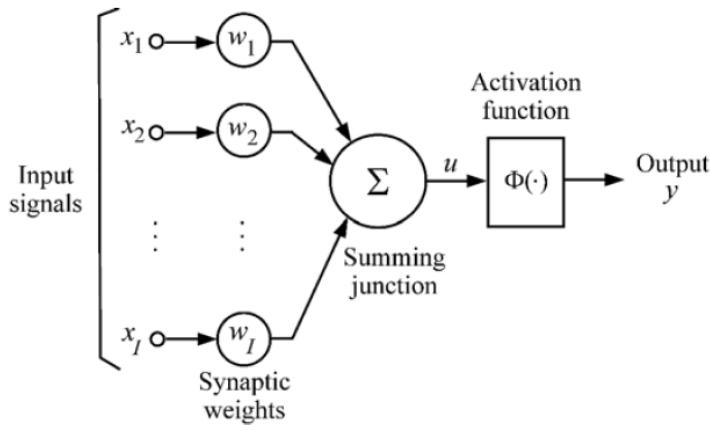


Figura 1.2: Arquitectura de una neurona artificial

La entrada sináptica total de la neurona viene dada por el producto interno entre los datos de entrada y los pesos $u = \sum_{i=1}^n w_i x_i = w^T x$. Posteriormente, a partir de esta entrada sináptica total, se obtiene la salida de la neurona a través de la función de activación $y = \phi(u)$.

1.4. Objetivos del Trabajo de Fin de Grado

Este Trabajo de Fin de Grado pretende realizar el diseño y la implementación sobre FPGA de una neurona artificial enfocada hacia la configurabilidad entre precisión y latencia. Para ello, los pasos a seguir son los siguientes:

1. En primer lugar, se realizará un estudio del estado del arte con el objetivo de profundizar en el conocimiento de las neuronas artificiales y en la configurabilidad de estas.
2. En segundo lugar, se buscará en la literatura existente información acerca de multiplicadores configurables y de funciones de activación.
3. A continuación, se realizará la implementación en VHDL de un multiplicador configurable. Para ello, se utilizará la herramienta Vivado de Xilinx.
4. Posteriormente, se realizará la implementación en VHDL de una función de activación buscando enfocarla hacia la configurabilidad.
5. Una vez hecho lo anterior, se planteará la estructura de una neurona artificial configurable a partir de los elementos desarrollados anteriormente.
6. Finalmente, se realizará la implementación de la neurona artificial configurable en VHDL para, posteriormente, implementarla sobre una FPGA, en concreto, sobre la placa Nexys 4 DDR basada en la FPGA XC7A100T-1CSG324C Artix-7 de Xilinx [2].

1.5. Planificación del Trabajo de Fin de Grado

A continuación, se muestra la planificación temporal que se ha seguido para la realización del proyecto:

Actividades	Horas	Tiempo de duración																Enero	
		Septiembre				Octubre				Noviembre				Diciembre				Enero	
		S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2
Curso de Machine Learning - Coursera	40																		
Estudio del estado del arte	20																		
Redacción de la introducción	12																		
Multiplicador configurable	Documentación	8																	
	Implementación	60																	
	Pruebas	20																	
	Redacción del multiplicador configurable	12																	
Función de activación	Documentación	8																	
	Implementación	60																	
	Pruebas	20																	
Neurona artificial configurable	Redacción de la función de activación	12																	
	Planteamiento de la estructura de la neurona configurable	18																	
	Redacción neurona artificial configurable	12																	
Redacción de conclusiones	8																		
Redacción de anexos	8																		
	318																		

Figura 1.3: Planificación temporal del TFG

Las actividades que se han llevado a cabo son las siguientes: en primer lugar, se ha realizado el curso de *Machine Learning* de la plataforma Coursera [3], el cual ofrece una extensa introducción al aprendizaje automático, a la minería de datos y al reconocimiento de patrones. Este curso sirve como primer contacto a las distintas técnicas de aprendizaje automático dentro de los algoritmos de aprendizaje supervisado y no supervisado. Para llevar a cabo esta actividad se han dedicado 40 horas repartidas en 2 semanas.

En segundo lugar, se ha realizado un estudio del estado del arte de las neuronas artificiales poniendo especial énfasis en la propiedad de configurabilidad de estas y en su implementación en FPGA. Esta actividad se llevó a cabo en 20 horas repartidas en 1 semana.

A continuación, se realizó el diseño y la implementación en VHDL de un multiplicador configurable. Esta actividad incluye la búsqueda de información acerca de este tipo de estructuras, su implementación, la realización de pruebas y su redacción en la memoria. Esta actividad se completó dedicando un total de 100 horas a lo largo de 6 semanas.

Posteriormente, se realizó el diseño y la implementación, también en VHDL, de una función de activación, en concreto, de una función sigmoide enfocada hacia la configurabilidad. Para esta actividad se han seguido pasos análogos a los de la implementación del multiplicador configurable. Esta actividad se completó en unas 100 horas a lo largo de 6 semanas.

Una vez realizada la implementación de la función de activación, se realizó un primer planteamiento de la arquitectura de una neurona artificial configurable, dedicando a esta actividad un total de 30 horas a lo largo de 3 semanas. La implementación tanto en VHDL como en FPGA de la neurona artificial configurable no se ha realizado por falta de tiempo y se ha dejado como continuación de este Trabajo de Fin de Grado. El número restante de horas se corresponden con la redacción de la introducción, las conclusiones y

los anexos, haciendo un total de 318 horas.

1.6. Código del Trabajo de Fin de Grado

Todo el código desarrollado en este Trabajo de Fin de Grado se encuentra en el siguiente repositorio: <https://github.com/markrealista/NeuronaArtificialConfigurable>

1.7. Estructura de la memoria

En lo que sigue, esta memoria se organiza de la siguiente manera: en el capítulo 2 se realizará el diseño y la implementación en VHDL de un multiplicador serie-paralelo configurable con precisión variable encontrado en la literatura. En el capítulo 3 se realizará el diseño y la implementación en VHDL de un módulo que implementa una función de activación, en concreto, una función sigmoide, con el objetivo de que sea configurable. En el capítulo 4 se realizará una primera aproximación hacia una neurona artificial configurable utilizando el multiplicador configurable y la función de activación desarrollados en los capítulos anteriores. Finalmente, en el capítulo 5 se expondrán las conclusiones del trabajo realizado.

Capítulo 2

Multiplicador serie-paralelo configurable de 4, 8 y 16 bits

Como se comentó anteriormente, el objetivo de este Trabajo de Fin de Grado es diseñar, desarrollar e implementar una neurona artificial enfocada hacia la configurabilidad entre precisión y latencia. Uno de los elementos fundamentales de esta neurona será el multiplicador, el cual será el encargado de realizar las operaciones de multiplicación necesarias dentro de la neurona, como por ejemplo, la multiplicación entre las señales de entrada y los pesos. Además, en este caso va a ser necesario un multiplicador que se pueda adaptar a los requerimientos de configurabilidad entre precisión y latencia de dicha neurona artificial. Después de buscar en la literatura, se ha encontrado un multiplicador serie-paralelo configurable de 4, 8 y 16 bits [4] que se ajusta a las necesidades requeridas.

Por un lado, este multiplicador es una combinación entre un multiplicador serie y un multiplicador paralelo. Un aspecto a tener en cuenta en los multiplicadores es el gran impacto que pueden tener en las prestaciones de los circuitos dependiendo del tipo y de la estrategia que se utilicen para implementarlos. Dos factores críticos en el diseño de multiplicadores son el consumo de recursos y la velocidad computacional. Los multiplicadores serie consiguen minimizar la utilización de recursos, pero requieren tiempo de computación. Por otra parte, los multiplicadores paralelos presentan altas velocidades de computación, pero son caras en términos de área. Utilizar un multiplicador serie-paralelo es, por tanto, un buen equilibrio entre estos dos factores.

Por otro lado, este multiplicador se basa en 4 multiplicadores de 4 bits que, o bien se pueden utilizar independientemente, o bien se pueden encadenar, a través de circuitos combinacionales, como por ejemplo, multiplexores, dando lugar a 2 multiplicadores independientes con precisiones de 8 bits o a un único multiplicador con una precisión de 16 bits. Esta variabilidad en la precisión aportará a la neurona artificial un grado de flexibilidad al poder modificar la precisión de los datos incluso en tiempo de ejecución. Además, el hecho de poder disponer de un número variable de multiplicadores y poder utilizarlos de forma independiente permiten mejorar la eficiencia de la utilización del área y optimizar el paralelismo de la neurona artificial. Otra característica interesante de este multiplicador es el compromiso que ofrece entre la precisión y la latencia. A mayores niveles de precisión, mayor es la latencia, y por tanto, menor es el throughput. Además de todo esto, este multiplicador se podrá configurar mediante dos comandos para que realice multiplicaciones en binario o en complemento a 2. Todas estas características se analizarán con mayor detalle a lo largo de este capítulo.

Este capítulo se organiza de la siguiente manera: en la sección 2.1 se realizará una breve descripción del algoritmo de multiplicación que emplea este multiplicador configurable. En la sección 2.2 se realizará el diseño y la implementación en VHDL de un multiplicador serie-paralelo de 4 bits, el cual será la base para realizar el multiplicador configurable. De este multiplicador de 4 bits se describirán los diferentes componentes que lo forman y se

explicará cómo realiza el proceso de multiplicación a través de una serie de simulaciones. En la sección 2.3 se realizará el diseño y la implementación en VHDL del multiplicador configurable, del cual se describirá su estructura, se analizarán cada una de sus configuraciones posibles y se realizarán una serie de simulaciones para explicar su principio de funcionamiento. Finalmente, en la sección 2.4 se analizarán los resultados de latencia, throughput, frecuencia y utilización de recursos que presenta este multiplicador configurable.

2.1. Algoritmo de multiplicación

El algoritmo que utiliza el multiplicador configurable de [4] se basa en los algoritmos de Braun y Baugh-Wooley. Considerando $A(m)$ y $B(n)$ dos números sin signo, siendo m una longitud de palabra cualquiera y $n = 4, 8$ o 16 bits, el producto de ambos viene dado por:

$$Q(m+n) = A(m)B(n) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} a_i b_j 2^{i+j} \quad (2.1a)$$

$$Q(m+n) = \sum_{k=0}^{m+n-1} q_k 2^k \quad \text{siendo} \quad q_k = \sum_{l=0}^k a_l b_{k-l} \quad (2.1b)$$

En la ecuación 2.1b, cada q_k consiste en un sumatorio de los productos de los bits simples a_l y b_{k-l} , los cuales se pueden realizar con puertas AND. La implementación en array de la ecuación anterior se conoce como multiplicador Braun.

Existen muchas formas de representar números con signo. Una de las más utilizadas es la representación en complemento a 2 (Ca2), debido a que permite representar el 0 de forma única. Los números anteriores $A(m)$ y $B(n)$ se pueden representar en Ca2 de la siguiente forma:

$$A(m) = -a_{m-1} 2^{m-1} + \sum_{i=0}^{m-2} a_i 2^i \quad (2.2)$$

$$B(n) = -b_{n-1} 2^{n-1} + \sum_{j=0}^{n-2} b_j 2^j \quad (2.3)$$

El producto de estos números en Ca2 viene dado por:

$$\begin{aligned} Q(m+n) &= \\ &= a_{m-1} b_{n-1} 2^{m+n-2} + \sum_{i=0}^{m-2} \sum_{j=0}^{n-2} a_i b_j 2^{i+j} \\ &+ 2^{n-1} \left(-2^m + 2^{m-1} + 1 + \sum_{i=0}^{m-2} \overline{a_i b_{n-1}} 2^i \right) \\ &+ 2^{m-1} \left(-2^n + 2^{n-1} + 1 + \sum_{j=0}^{n-2} \overline{a_{m-1} b_j} 2^j \right) \end{aligned} \quad (2.4)$$

donde \bar{x} representa el complemento de x . La ecuación 2.4 es la forma intermedia del algoritmo Baugh-Wooley. Se puede ver que la ecuación 2.1 es idéntica a la ecuación 2.4 excepto por los términos $2^{n-1}(-2^m + 2^{m-1} + 1)$ y $2^{m-1}(-2^n + 2^{n-1} + 1)$ y por el operador NAND que se introduce para realizar los productos $a_i b_{n-1}$ y $a_{m-1} b_j$. Para tener en cuenta estos términos extra, cuya función es realizar la multiplicación en Ca2, hay que sumar un '1' a los productos parciales q_{m+n-1} , q_{m-1} y q_{n-1} .

2.2. Multiplicador serie-paralelo de 4 bits

A lo largo de esta sección se va a describir el diseño y la implementación en VHDL de un multiplicador serie-paralelo de 4 bits que, como se verá más adelante, será la base para realizar el multiplicador configurable.

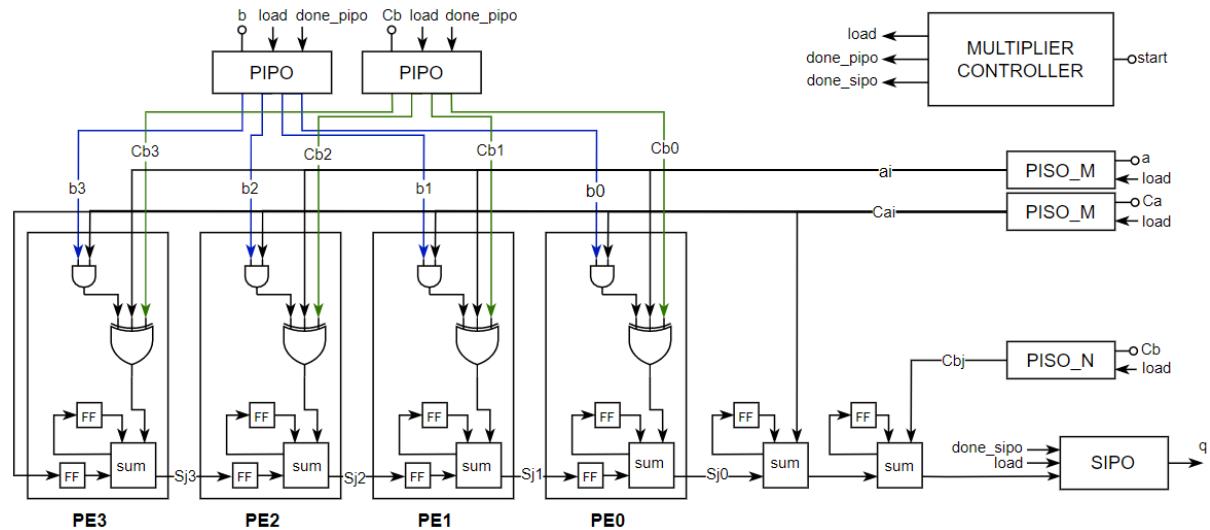


Figura 2.1: Multiplicador de 4 bits

Este multiplicador de 4 bits está formado por 4 elementos de procesamiento (PE), una cadena de sumadores a la salida de los PE y registros de desplazamiento PISO (Parallel In Serial Out), PIPO (Parallel In Parallel Out) y SIPO (Serial In Parallel Out). Además, para que el multiplicador funcione, es necesario un pequeño controlador que se encargue de proporcionar a los registros de desplazamiento las señales de control necesarias (*load*, *done_pipo* y *done_sipo*).

Los datos de entrada que recibirá el multiplicador serán dos operandos $A(m)$ y $B(n)$, que se introducirán en la entrada serie y paralelo respectivamente, dos comandos $C_A(m)$ y $C_B(n)$, que se introducirán de forma similar a $A(m)$ y $B(n)$, y una entrada de control *start* que controlará el inicio de la multiplicación. Dependiendo del valor de los comandos $C_A(m)$ y $C_B(n)$, la multiplicación se podrá realizar en binario o en Ca2. A partir de estos datos de entrada, el multiplicador proporcionará un dato de salida $Q(m + n)$ resultado del producto entre los operandos $A(m)$ y $B(n)$.

En cada instante i , cada PE recibirá 4 bits, a_i , b_j , C_{Ai} y C_{Bj} , correspondientes a los bits simples de los operandos $A(m)$ y $B(n)$ y los comandos $C_A(m)$ y $C_B(n)$ respectivamente y, a partir de estos bits de entrada, cada uno de estos PE computará la siguiente salida:

$$S_j(i) = (a_i b_j) \oplus C_{ij} = \overline{(a_i b_j)} C_{ij} + (a_i b_j) \overline{C_{ij}} \quad (2.5a)$$

siendo

$$C_{ij} = Ca_i \oplus Cb_j \quad (2.5b)$$

De esta forma, cuando $C_{ij} = 0$ ($Ca_i = Cb_j$), el PE_j computará el producto $a_i b_j$, y cuando $C_{ij} = 1$ ($Ca_i \neq Cb_j$), computará su complemento $\overline{a_i b_j}$.

Por un lado, de la ecuación 2.1 vemos que para realizar multiplicaciones binarias sólo es necesario realizar operaciones AND. Por tanto, para realizar multiplicaciones binarias, los bits Ca_i y Cb_j deberán ser '0' para todo i y j , es decir, $C_A(m) = C_B(n) = (0 \dots 0)$.

Por otro lado, a partir de la ecuación 2.4 se saca que durante los primeros $m - 1$ ciclos de reloj se realizan $n - 1$ operaciones AND y una operación NAND, mientras que durante el m -ésimo ciclo de reloj, se realiza una operación AND y $n - 1$ operaciones NAND. Por tanto, para realizar multiplicaciones en Ca2:

$$Ca_i = Cb_j = \begin{cases} 0 & \text{para } 0 \leq i \leq m - 2 \text{ y } 0 \leq j \leq n - 2 \\ 1 & \text{para } i = m - 1 \text{ y } j = n - 1 \end{cases} \quad (2.6)$$

Como resultado, $C_A(m) = (10 \dots 0)$ y $C_B(n) = (10 \dots 0)$. Además, para realizar la multiplicación en Ca2 hay que sumar un '1' a los productos parciales q_{m+n-1} , q_{m-1} y q_{n-1} . Para los dos primeros, se utilizarán los dos sumadores situados al final de la cadena de PE, mientras que para el último, se utilizará el sumador del PE_3 .

Para entender mejor el funcionamiento de este multiplicador de 4 bits, se van a describir cada uno de los elementos que lo componen.

2.2.1. Registros de desplazamiento PISO de m bits

Para poder proporcionar los bits a_i y Ca_i (bits simples del operando $A(m)$ y el comando $C_A(m)$ respectivamente) que se introducen en serie en los PE, se van a emplear registros de desplazamiento PISO (Parallel In Serial Out) de m bits (PISO_M).

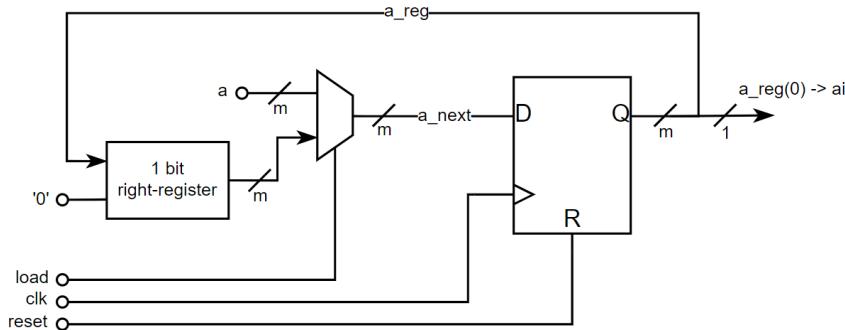


Figura 2.2: Registro de desplazamiento PISO de m bits

Estos registros de desplazamiento están compuestos por un flip-flop tipo D (FF D) con una señal de *reset* síncrona, un *right-register* de 1 bit, que desplaza una posición hacia la derecha los bits que le llegan introduciendo un '0' en la izquierda, y un multiplexor 2x1.

Descripción de puertos:

- a: dato de entrada de M bits. Este dato de entrada podrá ser o el operando $A(m)$ o el comando $C_A(m)$.

- ai: dato de salida de 1 bit. Este dato de salida proporcionará los bits simples del dato de entrada a desde el menos significativo hasta el más significativo.
- load: entrada de control proporcionada por el controlador del multiplicador (2.2.5). Si vale '1', se carga el dato de entrada, y si vale '0', se inicia el proceso de desplazamiento.

La función de estos registros PISO_M es proporcionar a la salida los bits simples de un determinado dato de entrada a de m bits desde el menos significativo hasta el más significativo. Cuando la señal *load* vale '1', el multiplexor selecciona el dato de entrada y este se carga, a través de la señal *a_next*, en el FF D en un flanco de subida de reloj. Cuando *load* pasa de valer '1' a valer '0', el multiplexor selecciona la salida del *right-register*, el cual desplaza los bits de la señal *a_reg* que almacena el valor del dato de entrada una posición hacia la derecha introduciendo un '0' en la izquierda. De esta forma, mientras *load* sea '0', se irán desplazando los bits de la señal *a_reg* y, seleccionando en la salida el bit menos significativo de esta señal *a_reg*, se irán obteniendo en cada ciclo de reloj los bits simples del dato de entrada desde el menos significativo hasta el más significativo.

2.2.2. Registro de desplazamiento PISO de n bits

Para poder sumar un '1' al producto parcial q_{n-1} con el fin de realizar la multiplicación en Ca2, va a ser necesario proporcionar en serie los bits Cb_j (bits simples del comando $C_B(n)$) a uno de los sumadores situados al final de la cadena de PE. Para ello, se va a emplear un registro de desplazamiento PISO de n bits (PISO_N).

Este registro de desplazamiento PISO_N tiene el mismo principio de funcionamiento que el registro de desplazamiento PISO_M descrito anteriormente, pero se diferenciarán en el número de bits del dato de entrada que cada uno puede desplazar.

2.2.3. Registros de desplazamiento PIPO

Para proporcionar los bits b_j y Cb_j (bits simples del operando $B(n)$ y el comando $C_B(n)$ respectivamente) que se introducen en paralelo en los PE, vamos a utilizar registros de desplazamiento PIPO (Parallel In Parallel Out).

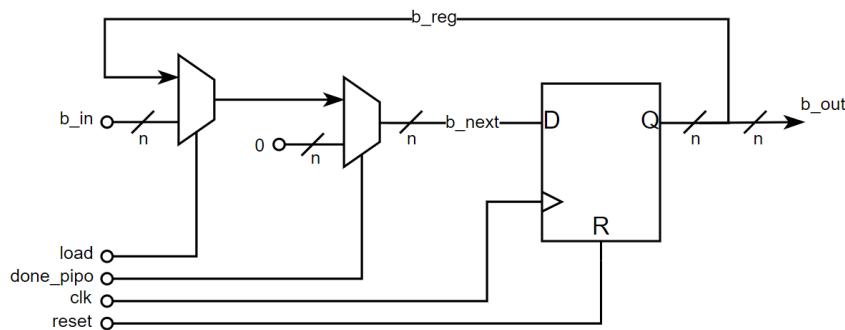


Figura 2.3: Registro de desplazamiento PIPO

Estos registros de desplazamiento están formados por un flip-flop tipo D (FF D) con una señal de *reset* síncrona y dos multiplexores 2x1.

Descripción de puertos:

- bin: dato de entrada de N bits que podrá ser o el operando $B(n)$ o el comando $C_B(n)$.
 - bout: dato de salida de N bits que tomará el valor, o bien del dato de entrada *bin*, o bien 0.
 - done_pipo: entrada de control proporcionada por el controlador del multiplicador (2.2.5). Cuando es '1', indica que los registros PISO_M han terminado de proporcionar todos los bits de los datos de entrada y hace que los registros PIPO dejen de proporcionar los datos de entrada pasando a sacar un 0.
 - load: entrada de control proporcionada por el controlador del multiplicador que controla la carga de los datos.

La función de estos registros PIPO es proporcionar a la salida el dato b_{in} que se introduce en la entrada, pero solo durante un determinado periodo de tiempo, proporcionando un 0 en el resto. La razón de que los registros PIPO proporcionen a la salida los datos de entrada solo durante un determinado periodo de tiempo se debe a que los bits b_j y Cb_j que se introducen en paralelo en los PE deben ir en consonancia con los bits a_i y Ca_i que se introducen en serie en los PE y que son proporcionados por registros de desplazamiento PISO_M. Cuando los registros PISO_M terminen de proporcionar todos los bits a_i y Ca_i a los PE, los registros PIPO también deberán dejar de proporcionar los bits b_j y Cb_j a los PE a fin de realizar correctamente la multiplicación en Ca2 (con la multiplicación en binario no habría ningún problema). Si no se dejaren de proporcionar los bits b_j y Cb_j a los PE una vez que los registros PISO_M terminasen de proporcionar todos los bits a_i y Ca_i , el bit Ca_{m-1} que se encarga de sumar un '1' al producto parcial q_{m+n-1} cuando se realiza la multiplicación en Ca2, se sumaría con el bit Cb_{n-1} y no se realizaría de manera correcta la multiplicación en Ca2.

El funcionamiento de estos registros de desplazamiento PIPO es el siguiente: por un lado, cuando $load$ vale '1' y $done_pipe$ vale '0', los multiplexores seleccionan el dato de entrada b_{in} y este se almacena en el FF D a través de la señal b_{next} en un flaco de subida de reloj, haciendo que el dato de salida b_{out} tome el valor de b_{in} . Por otro lado, cuando $done_pipe$ vale '0', se almacena en el FF D un 0 independientemente del valor de $load$, haciendo que $b_{out} = 0$. Y por último, cuando $load = done_pipe = '0'$, b_{out} tomará el dato que en ese momento esté almacenado en el FF D, que podrá ser o b_{in} o 0.

2.2.4. Registro de desplazamiento SIPO

Para poder obtener el resultado de la multiplicación a partir de los productos parciales q_i , se va a emplear un registro de desplazamiento SIPO (Serial In Parallel Out).

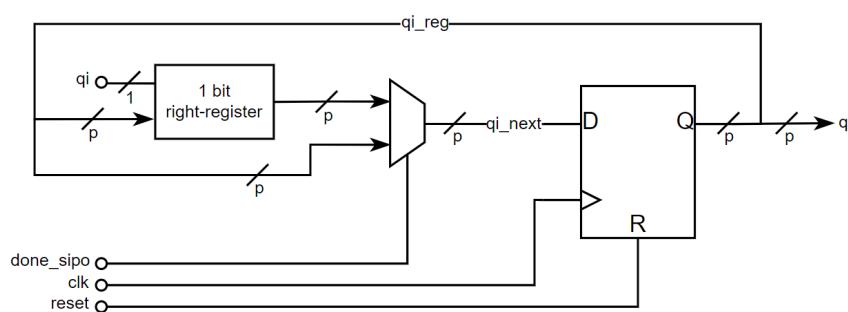


Figura 2.4: Registro de desplazamiento SIPO

Este registro de desplazamiento está formado por un flip-flop tipo D (FF D) con una señal de *reset* síncrona, un *right-register* de 1 bit, que desplaza una posición hacia la derecha los bits del dato que le llega introduciendo un nuevo bit q_i en la posición del bit más significativo, y un multiplexor 2x1.

Descripción de puertos:

- qi : dato de entrada de 1 bit que se corresponde con los productos parciales proporcionados por la cadena de PE y los sumadores de salida.
- q : dato de salida de P bits que proporciona el resultado de la multiplicación.
- $done_sipo$: entrada de control proporcionada por el controlador del multiplicador (2.2.5). Cuando vale '0', indica que la multiplicación todavía no ha terminado y se siguen almacenando los productos parciales que lleguen, y si vale '1', indica que la multiplicación ha terminado y se proporciona el resultado de la multiplicación.
- $done$: salida que indica si la multiplicación ha terminado ('1') o no ('0').

La función de este registro SIPO es almacenar los productos parciales procedentes de los PE y los sumadores de salida y proporcionar a la salida el resultado de la multiplicación. Para ello, cuando *done_sipo* vale '0', el multiplexor selecciona la salida del *right-register*, el cual desplaza hacia la derecha los bits de la señal *qi_reg* que almacena los productos parciales de la multiplicación introduciendo un nuevo producto parcial q_i en la izquierda del todo, y guarda este nuevo valor en el FF D. De esta forma, mientras *done_sipo* sea '0', se irán almacenando en la señal *qi_reg* los productos parciales de la multiplicación. Finalmente, cuando *done_sipo* vale '1', el multiplexor selecciona el valor almacenado en el FF D, que será el resultado de la multiplicación, haciendo que la salida *q* mantenga este valor hasta que se inicie una nueva multiplicación.

2.2.5. Controlador del multiplicador

Este controlador será el encargado de orquestar los diferentes registros de desplazamiento del multiplicador configurable suministrándoles las señales de control necesarias de forma que estos proporcionen los datos en los instantes de tiempo precisos a fin de realizar el proceso de multiplicación de forma correcta.

Para la implementación de este controlador se ha escogido una metodología de diseño basada en una máquina de estados finitos con ruta de datos asociada (FSMD). En la figura 2.5 se muestra su diagrama ASMD.

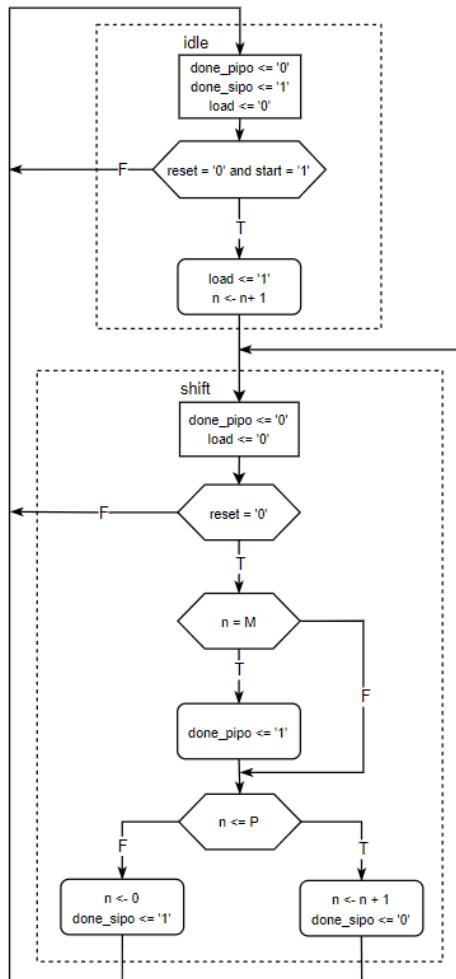


Figura 2.5: Controlador del multilpicador configurable - Diagrama ASMD

Descripción de puertos:

- done_pipo: salida de control para los registros de desplazamiento PIPO. Indica si los registros PISO_M han terminado ('1') o no ('0') de sacar todos los bits de los datos de entrada para que los registros PIPO dejen también de proporcionar los datos de entrada.
- done_sipo: señal de control para el registro de desplazamiento SIPO. Cuando es '1', hace que el registro SIPO deje de introducir nuevos productos parciales y mantenga a la salida el resultado de la multiplicación. Cuando es '0', se lleva a cabo el proceso de desplazamiento.
- start: entrada de control que cuando es '1' inicia el proceso de multiplicación poniendo la señal *load* a '1' durante un ciclo de reloj.
- load: salida de control generada a partir de la entrada *start* que carga los datos de entrada en los registros de desplazamiento cuando está a '1'.
- done: salida que indica, cuando está a '1', que el resultado de la multiplicación está disponible.

Este controlador tiene 2 estados:

Estado idle

Estado de reposo en el que se inicializan los datos de control que se proporcionan a los registros de desplazamiento. Si en este estado *start* vale '1' y *reset* es '0', se pasa al estado *shift* poniendo *load* a '1' e incrementando el valor de la señal interna *n* utilizada para contabilizar los ciclos de reloj.

Estado shift

En este estado los registros de desplazamiento inician los procesos de desplazamiento de los datos de entrada. En primer lugar, se comprueba si la señal interna *n* es igual a M, si lo es, quiere decir que los registros PISO han terminado de sacar todos los bits del dato de entrada y, por tanto, se pone la señal *done_pipo* a '1', y si no, no se hace nada y se pasa a la siguiente condición. En la siguiente condición se comprueba si *n* es menor o igual que P, si lo es (aún no ha terminado la multiplicación), se incrementa el valor de *n*, se pone *done_sipo* a '0' y se mantiene en el estado *shift*, y si no lo es (la multiplicación ha terminado), se resetea el valor de la señal interna *n*, se pone a '1' *done_sipo* y se vuelve al estado de reposo *idle*.

2.2.6. Elementos de procesamiento

Los elementos de procesamiento son los bloques encargados de realizar la multiplicación bit a bit de los operandos $A(m)$ y $B(n)$ y de proporcionar a la salida los productos parciales de la multiplicación.

Este multiplicador de 4 bits está formado por 4 PE, y a su vez, cada uno de estos PE está compuesto por una puerta AND de dos entradas, una puerta XOR de tres entradas, un *full-adder* y dos flip-flops tipo D (FF D) (ver figura 2.1).

Cada uno de estos PE realiza las siguientes operaciones: la puerta AND del PE_j recibe los bits a_i y b_j procedentes de registros PISO_M y PIPO respectivamente. Posteriormente, el resultado de la operación AND se introduce en una de las tres entradas de la puerta XOR, de forma que esta recibe la salida de la puerta AND y los bits Ca_i y Cb_j procedentes también de registros PISO_M y PIPO respectivamente. Por último, el sumador recibe en una de sus dos entradas el resultado de la operación XOR, y en la otra, la salida S_{j+1} del PE_{j+1} a través de un registro. Además, el acarreo de salida del sumador se conecta, a través de otro registro, al acarreo de entrada del mismo. De esta forma, los acarreos que se generan se van propagando entre las sucesivas sumas.

Como se explicó anteriormente, para poder realizar la multiplicación en Ca2 es necesario sumar un '1' a los productos parciales q_{m+n-1} , q_{m-1} y q_{n-1} . Para el producto parcial q_{m+n-1} se utiliza el sumador del PE_3 . Este sumador recibe en una de sus entradas, no la salida del PE previo como en el resto de sumadores, sino los bits Ca_i procedentes de un registro PISO_M. De esta forma, al realizar la multiplicación en Ca2, el sumador del PE_3 recibirá en el instante $i = m$ el bit $Ca_{m-1} = 1$, que se encargará de sumar un '1' a dicho producto parcial q_{m+n-1} .

2.2.7. Sumadores de salida

La cadena de *full-adders* situados a la salida de los PE (ver figura 2.1) sirven para sumar un '1' a los productos parciales q_{m-1} y q_{n-1} cuando se realiza la multiplicación en Ca2.

El primer sumador, situado a la salida del PE_0 , recibe en una de sus dos entradas los bits Ca_i procedentes de un registro PISO_M, y en la otra, la salida S_0 del PE_0 , el cual proporciona los productos parciales de la multiplicación. Este sumador se encarga de

sumar un '1' al producto parcial q_{m-1} . Posteriormente, el siguiente sumador recibe en una de sus dos entradas la salida del sumador previo, y en la otra, los bits Cb_j procedentes del registro PISO_N. De esta forma, este segundo sumador se encarga de sumar un '1' al producto parcial q_{n-1} . Además, el acarreo de salida de estos dos sumadores se realimentan, a través de un registro, al acarreo de entrada con el objetivo de propagar los acarreos.

Con esto, a la salida de esta cadena de sumadores se obtienen los productos parciales representados correctamente, tanto si se realiza la multiplicación en binario como en Ca2. Finalmente, estos productos parciales se envían al registro SIPO que se encargará de almacenarlos para proporcionar el resultado de la multiplicación.

2.2.8. Ejemplo de multiplicación

Una vez descritos todos los componentes del multiplicador de 4 bits, se van a realizar una serie de simulaciones para explicar cómo se realiza el proceso de multiplicación.

Como se explicó anteriormente, este multiplicador realiza la multiplicación entre dos operandos $A(m)$ y $B(n)$ y proporciona un resultado $Q(m + n)$. Además, en función del valor de los comandos $C_A(m)$ y $C_B(n)$, la multiplicación se puede realizar en binario o en Ca2. Para las siguientes simulaciones se han fijado los siguientes datos de entrada: $A(4) = 0101$, $B(4) = 1010$, $C_A(4) = 1000$ y $C_B(4) = 1000$. Con estos valores, se realiza la multiplicación en Ca2 entre los números 5 y -6.

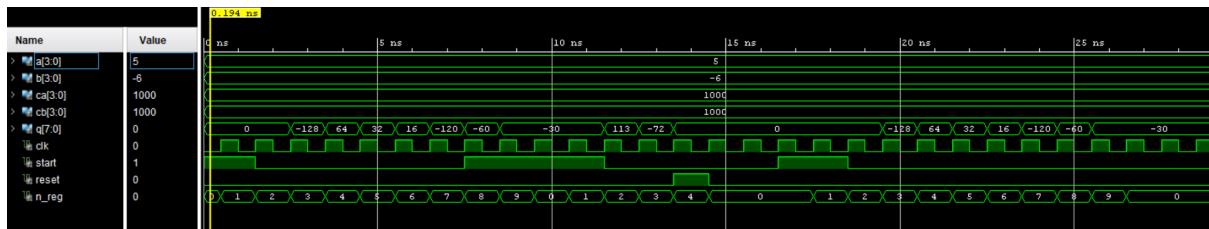


Figura 2.6: Testbench - Multiplicador de 4 bits

En la captura anterior se pueden ver los diferentes valores asignados a los operandos y a los comandos, así como a las entradas *start* y *reset*. Además, se puede ver el resultado de la multiplicación proporcionado en la salida *q*. Vemos que una vez que se inicia el proceso de multiplicación (*start* = '1'), el multiplicador tarda 8 ciclos de reloj en proporcionar el resultado (-30). Para entender mejor cómo se realiza este proceso de multiplicación se van a mostrar varias simulaciones correspondientes a los diferentes módulos que componen el multiplicador.

La siguiente figura muestra las señales correspondientes al controlador del multiplicador:

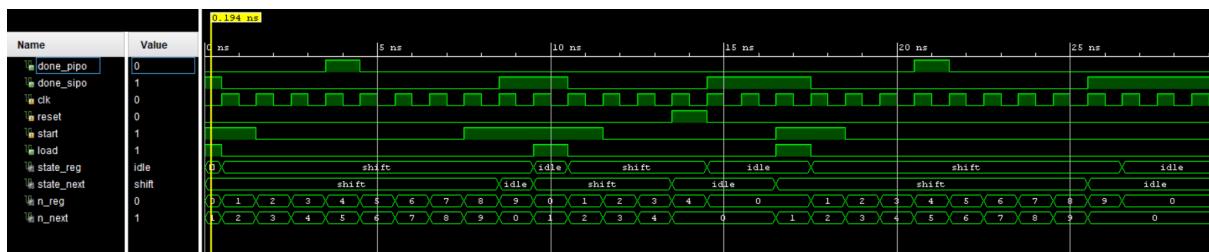


Figura 2.7: Testbench - Controlador del multiplicador

En la simulación de la figura 2.7 se puede ver que cuando *start* se pone a '1' estando

en el estado *idle*, *load* se pone a '1' durante un ciclo de reloj. Por otro lado, la salida de control *done_pipo*, que sirve para indicar a los registros de desplazamiento PIPO que dejen proporcionar los datos de entrada, se pone a '1' en el cuarto ciclo de reloj tras haberse puesto *load* a '1'. Como se verá en las siguientes simulaciones, el ciclo en el que *done_pipo* se pone a '1' coincide con el ciclo en el que los registros PISO_M terminan de proporcionar todos los bits simples de los datos de entrada. Por último, la salida de control *done_sipo*, que indica al registro SIPO si la multiplicación ha terminado, se pone a '1' en el noveno ciclo de reloj desde que se inicia la cuenta, y como se puede ver en la simulación de la figura 2.6, coincide con el ciclo en el que se proporciona el resultado de la multiplicación.

Las siguientes figuras corresponden a las simulaciones realizadas para los registros de desplazamiento PIPO_M:

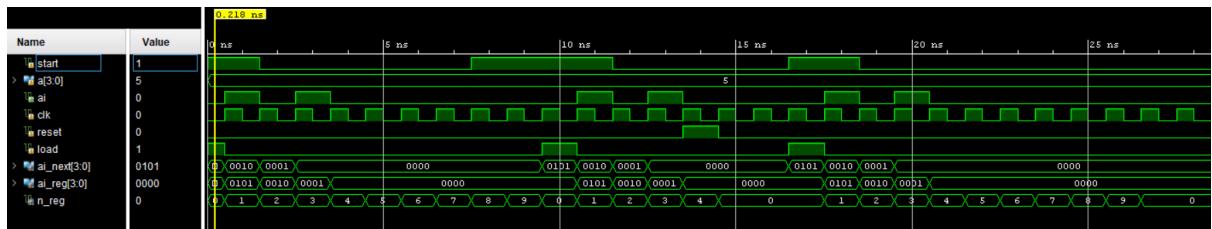


Figura 2.8: Testbench - Registro PISO_M para $A(m)$

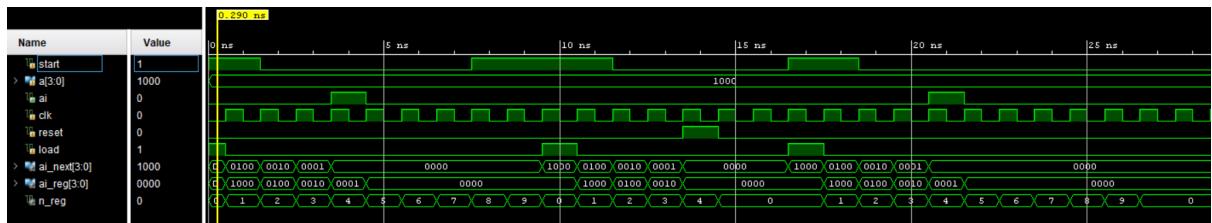


Figura 2.9: Testbench - Registro PISO_M para $C_A(m)$

En estas simulaciones se puede ver que, para proporcionar los 4 bits de los datos de entrada $A(m)$ y $C_A(m)$, los registros de desplazamiento tardan 4 ciclos de reloj. Llevándolo a un caso más general, para proporcionar todos los bits de un dato de entrada de m bits, tardarán m ciclos de reloj. Como se puede ver, la latencia de los registros PISO_M depende del número bits de los datos de entrada y es igual a tantos ciclos de reloj como bits de los datos de entrada. Esto también aplica para el registro de desplazamiento PISO_N.

Las siguientes capturas corresponden a las simulaciones realizadas de los registros de desplazamiento PIPO:

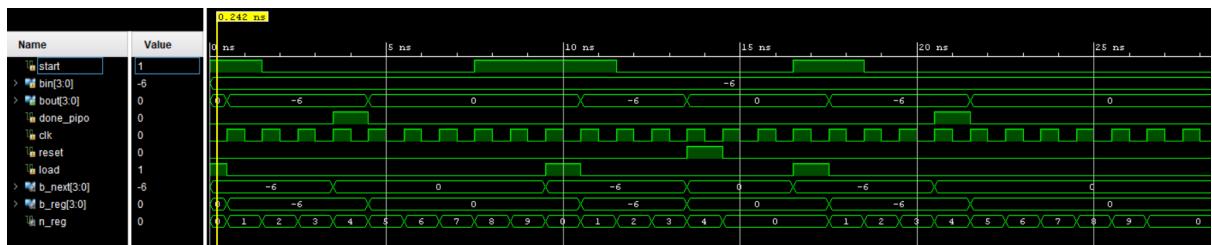
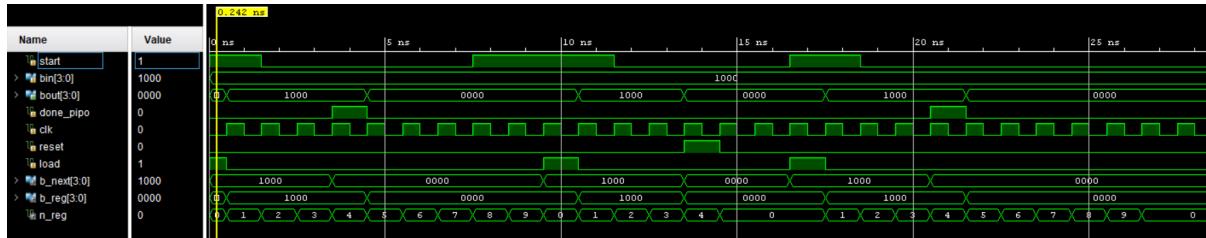


Figura 2.10: Testbench - Registro PIPO para $B(n)$

Figura 2.11: Testbench - Registro PIPO para $C_B(n)$

En estas simulaciones vemos que cuando *load* vale '1', los registros proporcionan a la salida los datos de entrada, y cuando *done_pipo* es '1', estos proporcionan un 0. Comparando estas simulaciones con las realizadas para los registros PISO_M vemos que, cuando los registros PISO_M terminan de proporcionar todos los bits de sus datos de entrada, los registros PIPO también dejan de proporcionar sus datos de entrada. Esto se consigue gracias a la entrada de control *done_pipo* proporcionado por el controlador del multiplicador. Como se mencionó anteriormente, hacer esto es necesario a fin de realizar la multiplicación en Ca2 correctamente. A diferencia de los registros PISO, los registros PIPO sólo tardan un ciclo de reloj en proporcionar los datos de entrada, es decir, su latencia es igual a un ciclo de reloj y no depende de la longitud del número de bits de los datos de entrada.

La siguiente simulación muestra los datos de salida *sout* que computa cada uno de los PE. Además, se muestran los datos de entrada *sin* que cada PE recibe del PE previo excepto para el PE_3 , que recibe en su entrada *sin3* los bits del comando $C_A(m)$. En esta simulación también se muestran los datos de entrada que reciben cada uno de los sumadores situados a la salida del bloque de PE. Vemos que la entrada *a_sum1* del primer sumador recibe la salida *sout0* del PE_0 , el cual contiene los productos parciales de la multiplicación, y la entrada *b_sum1* recibe los bits Ca_i . Posteriormente, la salida de este primer sumador se conecta a la entrada *a_sum2* del segundo sumador, y en la entrada *b_sum2* se introducen los bits Cb_j procedentes del registro PISO_N. Finalmente, la salida de este segundo sumador proporciona al registro SIPO los productos parciales listos para ser representados.

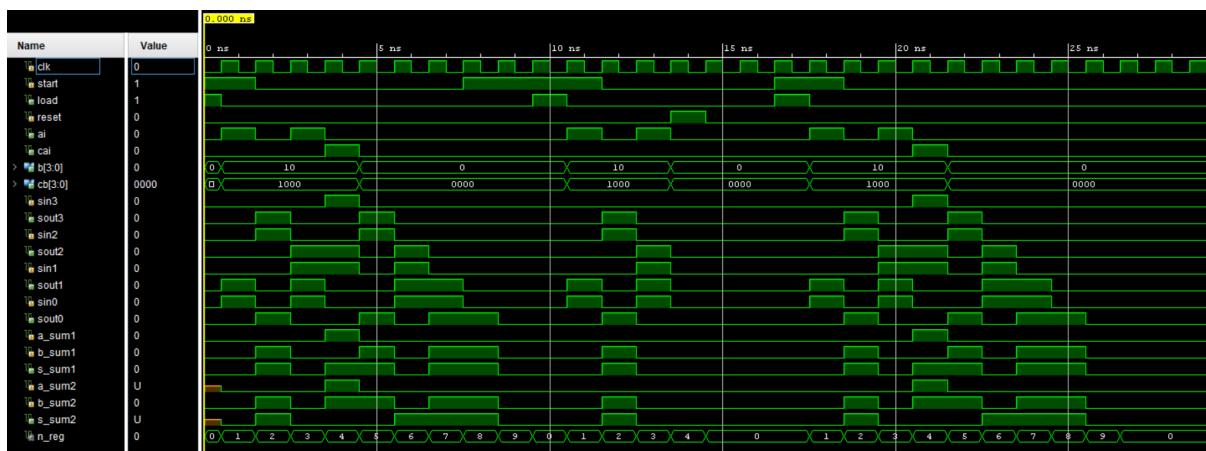


Figura 2.12: Testbench - PE y cadena de sumadores

La siguiente simulación muestra las señales correspondientes al registro de desplazamiento SIPO:

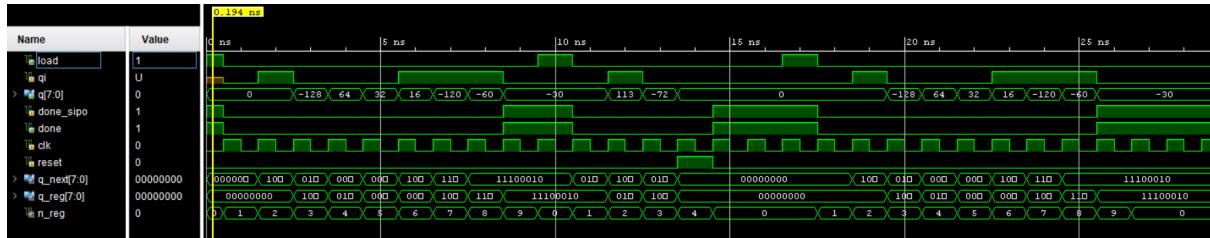


Figura 2.13: Testbench - Registro SIPO

En dicha simulación podemos ver que cuando *load* vale '1' comienza el proceso de desplazamiento y el registro empieza a acumular los bits que llegan en la entrada introduciéndolos en la posición del bit más significativo y desplazando el resto una posición hacia la derecha. Finalmente, cuando la entrada de control *done_sipo* se pone a '1', el registro SIPO proporciona el resultado de la multiplicación. Para proporcionar el resultado de multiplicar dos operandos de 4 bits, vemos que el registro de desplazamiento tarda 8 ciclos de reloj, siendo estos ciclos los mismos que el número de bits del resultado de la multiplicación. Por tanto, vemos que la latencia de este registro SIPO depende del número de bits del resultado de la multiplicación y es igual a tantos ciclos de reloj como bits del resultado de la multiplicación.

Latencia y throughput del multiplicador de 4 bits

Como se ha visto en las simulaciones anteriores, la latencia de este multiplicador de 4 bits depende, principalmente, de la latencia de los registros de desplazamiento. Los registros de desplazamiento PISO_M tienen una latencia de m ciclos de reloj, el registro PISO_N de n ciclos de reloj, los registros PIPO de un ciclo de reloj, y el registro SIPO tienen una latencia de $m + n$ ciclos de reloj. Como se puede ver, los registros que tienen una entrada o una salida en serie son los que mayor latencia tienen, y en este caso, el registro SIPO es el que mayor latencia presenta, ya que $m + n \geq m, n$. Por tanto, se puede concluir que este multiplicador de 4 bits tiene una latencia de $m + n$ ciclos de reloj. Además, como este multiplicador realiza las operaciones secuencialmente, su throughput es igual a la inversa de la latencia, esto es, $(1/m + n)$ operaciones/ciclo.

2.3. Multiplicador serie-paralelo configurable de 4, 8 y 16 bits

En esta sección se va a describir el diseño e implementación en VHDL del multiplicador serie-paralelo configurable de 4, 8 y 16 bits [4].

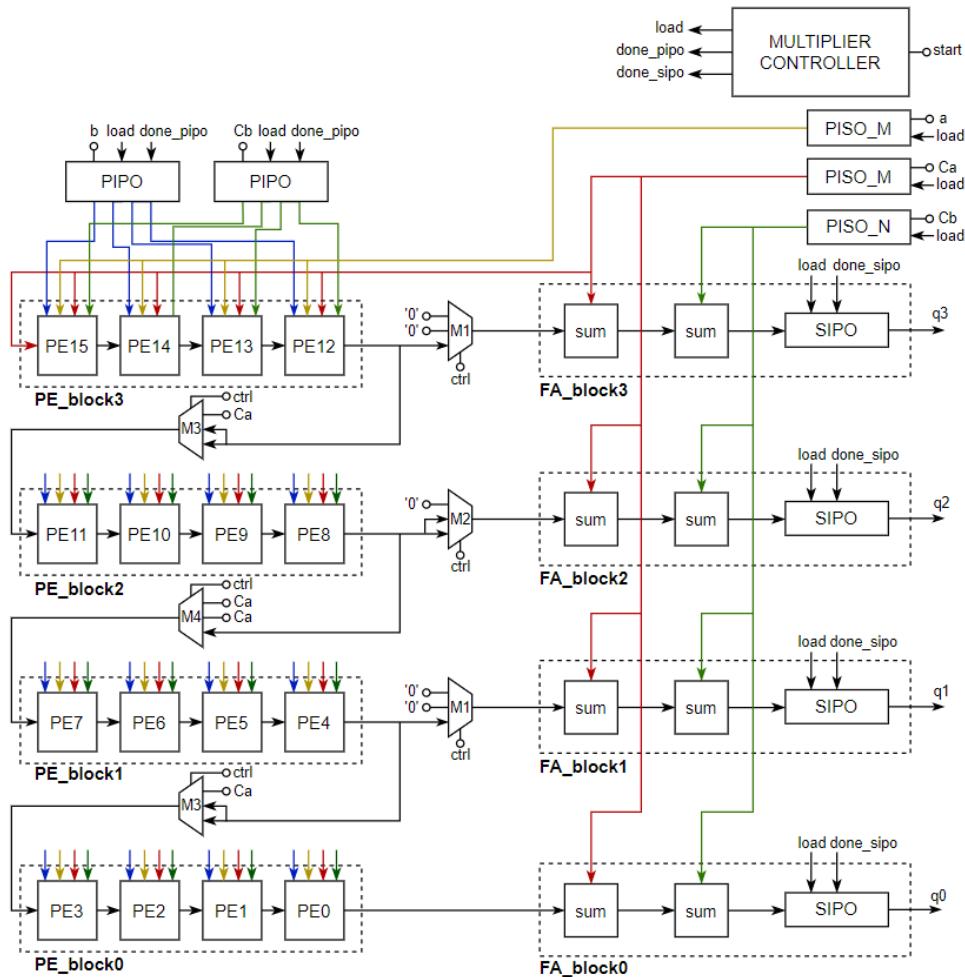


Figura 2.14: Multiplicador configurable de 4, 8 y 16 bits

En la figura 2.14 se muestra la arquitectura del multiplicador configurable de 4, 8 y 16 bits. Este multiplicador consta de 4 filas en las que cada una consiste en un multiplicador de 4 bits compuesto por un bloque de 4 elementos de procesamiento (PE_block) y un bloque formado por dos sumadores y un registro SIPO (FA_block). Además, las tres primeras filas tienen, entre estos dos bloques, una serie de multiplexores (M1, M2, M3 y M4) que sirven para conectar dos filas adyacentes de bloques de PE, de forma que, en función del valor de la señal de control, se pueden obtener:

- 1 multiplicador de 16 bits para $ctrl = "00"$.
- 2 multiplicadores de 8 bits para $ctrl = "01"$.
- 4 multiplicadores de 4 bits para $ctrl = "11"$.

De esta forma, se puede modificar la precisión de la longitud de la palabra paralelo n a 4, 8 o 16 bits manteniendo el paralelismo. Con esto vemos que, si la precisión n disminuye, el número de multiplicadores en el circuito aumenta y viceversa.

La siguiente tabla muestra los datos que seleccionan cada uno de los multiplexores en función del valor de la señal de control:

Multiplexores ctrl	M1	M2	M3	M4
"00"	'0'	'0'	sout12/sout4	sout8
"01"	'0'	sout8	sout12/sout4	C_A_i
"11"	sout12/sout4	sout8	C_A_i	C_A_i

Tabla 2.1: Tabla de multiplexores

Además de estos elementos mencionados, el multiplicador configurable utiliza registros de desplazamiento PISO, PIPO y un controlador como los descritos anteriormente en este mismo capítulo.

2.3.1. 4 multiplicadores de 4 bits

Como se ha visto, dependiendo del valor de la señal de control de los multiplexores se pueden conseguir precisiones de 4, 8 o 16 bits en la entrada paralelo obteniendo 4 multiplicadores de 4 bits, 2 de 8 bits o 1 de 16 bits respectivamente. El valor de la señal de control de los multiplexores con el que se obtiene una configuración de 4 multiplicadores de 4 bits es $ctrl = "11"$.

Para este caso, los multiplexores M1 y M2 conectan la entrada de los bloques FA_block3, FA_block2 y FA_block1 con la salida de los bloques PE_block3, PE_block2, PE_block1 respectivamente, y los multiplexores M3 y M4 introducen en la entrada de los bloques PE_block2, PE_block1 y PE_block0 los bits simples del comando $C_A(m)$ para poder realizar la multiplicación en Ca2. El funcionamiento de cada uno de estos multiplicadores de 4 bits resultantes es el mismo que el descrito en la sección 2.2.

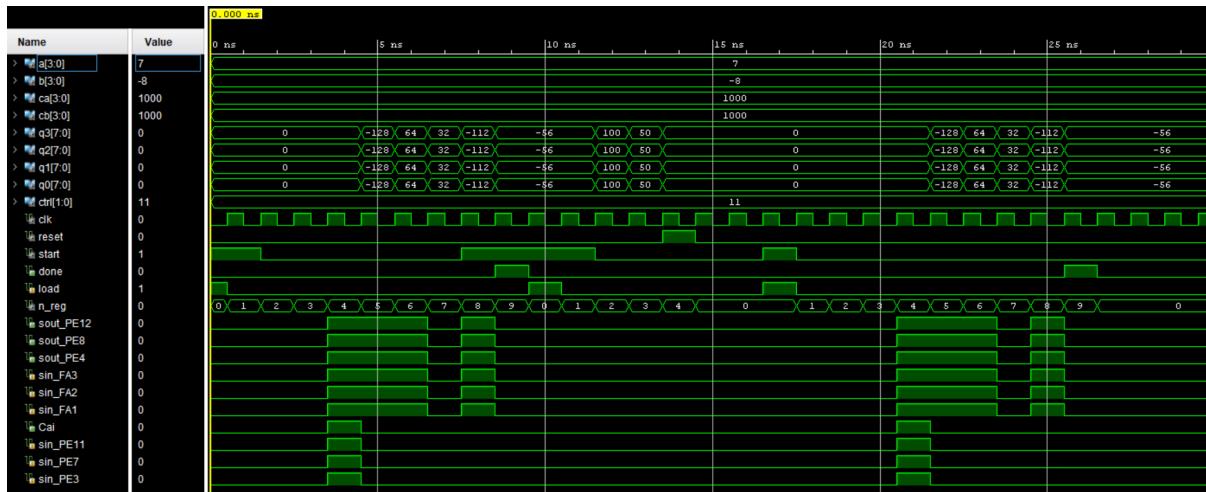


Figura 2.15: Testbench - 4 multiplicadores de 4 bits

En la simulación que se muestra en la figura 2.15 se ha realizado la multiplicación en Ca2 de los números $A(4) = "0111"$ (7) y $B(4) = "1000"$ (-8). En este caso, se han introducido los mismos datos de entrada en los 4 multiplicadores de 4 bits, es por eso que los 4 proporcionan los mismos datos de salida ($q3 = q2 = q1 = q0$). Además, en dicha simulación se puede comprobar que las salidas de los bloques PE_block se conectan a las entradas de los bloques FA_block correspondientes y las entradas de los bloques PE_block reciben los bits simples del comando $C_A(m)$.

Como se vio anteriormente, la latencia que presenta un multiplicador de 4 bit es de

$m + n$ ciclos de reloj, y para este caso en el que ambos operandos $A(m)$ y $B(n)$ tienen una longitud de 4 bits, los multiplicadores tardan 8 ciclos de reloj en proporcionar el resultado tal y como se puede observar en la simulación.

2.3.2. 2 multiplicadores de 8 bits

El valor de la señal de control de los multiplexores con el que se obtienen 2 multiplicadores de 8 bits es $ctrl = "01"$. En este caso, los multiplexores M1 introducen un '0' en la entrada de los bloques FA_block3 y FA_block1 inutilizándolos, y los multiplexores M3 conectan la salida de los bloques PE_block3 y PE_block1 con la entrada de los bloques PE_block2 y PE_block0 respectivamente, obteniendo dos bloques de 8 PE. Por otro lado, el multiplexor M2 conecta la salida del bloque PE_block2 con la entrada del bloque FA_block2, y el multiplexor M4 introduce en el bloque FA_block1 los bits simples del comando $C_A(m)$.

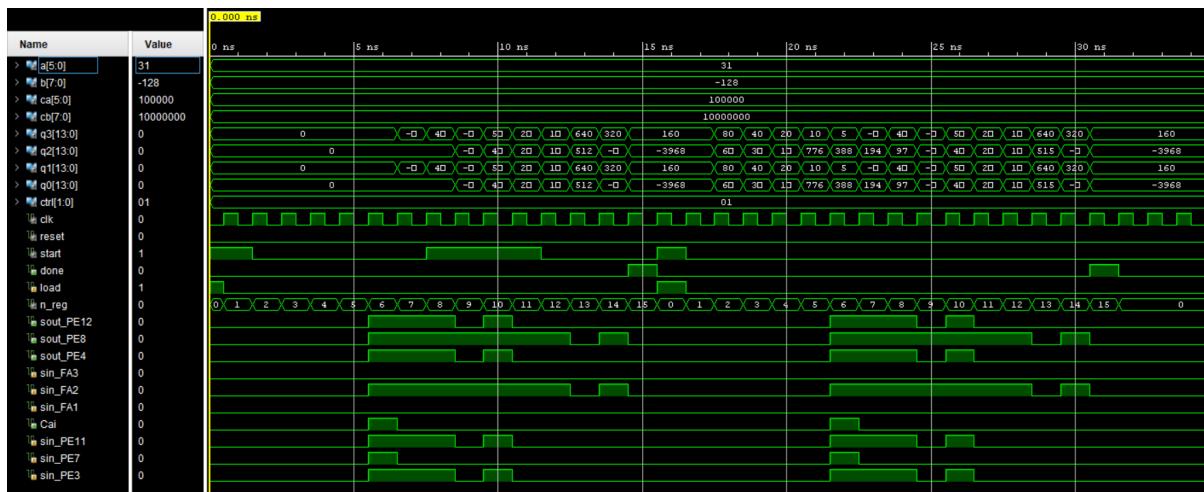


Figura 2.16: Testbench - 2 multiplicadores de 8 bits

En la simulación que se muestra en la figura 2.16 se ha realizado la multiplicación en Ca2 de los operandos $A(6) = "011111"$ (31) y $B(8) = "10000000"$ (-128). Para esta simulación se han introducido los mismos datos de entrada en los 2 multiplicadores de 8 bits, por lo que los resultados proporcionados por los bloques FA_block2 y FA_block0 son iguales ($q_2 = q_0$). En dicha simulación también se puede ver cómo se interconectan las salidas y entradas de cada uno de los bloques del multiplicador.

La latencia de estos multiplicadores de 8 bits, al igual que para los multiplicadores de 4 bits, viene determinada por la latencia de los registros SIPO, que es igual a $m + n$ ciclos de reloj. Para este ejemplo en concreto, donde $m = 6$ y $n = 8$, la latencia de los multiplicadores de 8 bits es de $6 + 8 = 14$ ciclos de reloj.

2.3.3. 1 multiplicador de 16 bits

La última configuración posible de este multiplicador configurable es la de un multiplicador de 16 bits, la cual proporciona una precisión de 16 bits en la entrada en paralelo. Esta configuración se consigue para $ctrl = "00"$. En este caso, los multiplexores M1 y M2 inhabilitan todos los bloques FA_block introduciéndoles un '0' en la entrada, excepto el bloque FA_block0, que es el encargado de proporcionar el resultado de la multiplicación, y los multiplexores M3 y M4 encadenan los todos los bloques PE_block formando un único bloque con 16 PE.

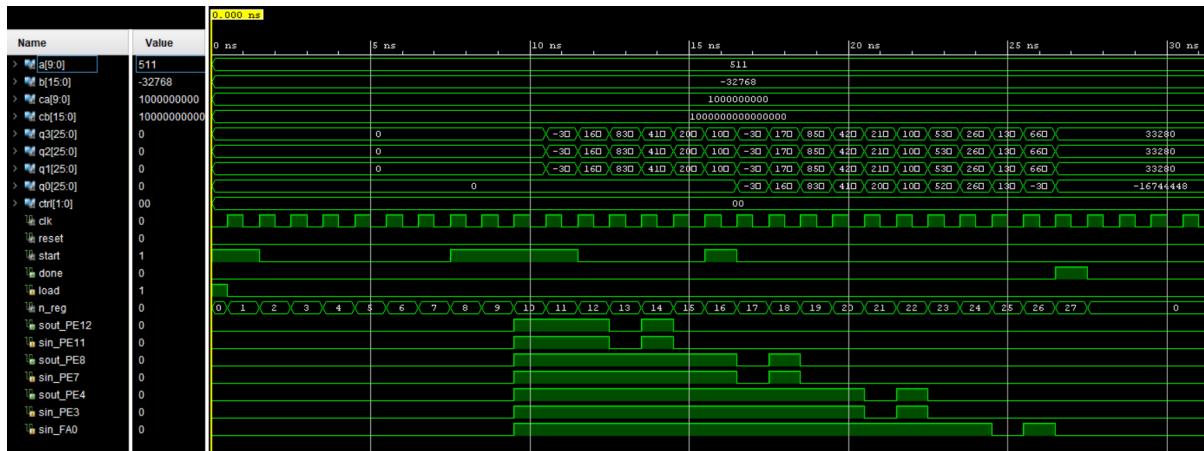


Figura 2.17: Testbench - 1 multiplicador de 16 bits

En la simulación de la figura 2.17 se ha realizado la multiplicación en Ca2 de los operandos $A(10) = "0111111111"$ (511) y $B(16) = "1000000000000000"$ (-32768). En dicha simulación se puede ver que el resultado de la multiplicación se proporciona en la salida q_0 , salida del bloque FA_block0. Además, se puede ver que los bloques PE_block se interconectan entre sí, ya que los datos de salida de un bloque son los datos de entrada del siguiente.

Al igual que para los multiplicadores anteriores, la latencia de este multiplicador de 16 bits viene determinada por la latencia del registro SIPO, que es igual a $m + n$ ciclos de reloj. Para esta simulación, donde $m = 10$ y $n = 16$, el multiplicador de 16 bits tarda $10 + 16 = 26$ ciclos de reloj en proporcionar el resultado.

2.4. Análisis de resultados del multiplicador configurable

En esta sección se van a analizar aspectos relativos a la latencia, al throughput, a la frecuencia y a la utilización de recursos del multiplicador configurable. Es preciso mencionar que los resultados de frecuencia y utilización se han obtenido para la placa Nexys 4 DDR de Digilent basada en la FPGA XC7A100T-1CSG324C Artix-7 de Xilinx [2].

2.4.1. Multiplicador configurable: Latencia y throughput

Como se ha ido comentando en las simulaciones realizadas para cada una de las configuraciones del multiplicador configurable, cuanta mayor precisión se quiera obtener, mayor es la latencia. Y también se ha visto que dicha latencia viene determinada principalmente por la latencia que introducen los registros de desplazamiento SIPO, que es igual a $m + n$ ciclos de reloj.

Este multiplicador configurable realiza las multiplicaciones de forma secuencial, por tanto, el throughput para cada una de las configuraciones es igual a la inversa de la latencia (operaciones/ciclo). De esta forma, a niveles de precisión más altos, menor es el throughput, al contrario de lo que ocurre con la latencia.

2.4.2. Multiplicador configurable: Frecuencia

Para conocer la frecuencia máxima a la que el multiplicador configurable es capaz de trabajar, se ha implementado una pequeña arquitectura de prueba formada por el propio multiplicador configurable y dos registros de desplazamiento que se encargan de introducir

y sacar en serie los datos del multiplicador configurable. Además, se ha utilizado el módulo *Cloking Wizard* que integra la herramienta Vivado para generar la señal de reloj con la frecuencia deseada a partir de una señal de reloj con una frecuencia de 100 MHz. Probando con diferentes frecuencias de reloj, se ha llegado a la conclusión de que la frecuencia más alta a la que puede trabajar este multiplicador configurable es a 330 MHz.

Design Timing Summary								
Setup	Hold			Pulse Width				
Worst Negative Slack (WNS): 0,085 ns				Worst Hold Slack (WHS): 0,049 ns				Worst Pulse Width Slack (WPWS): 0,875 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: 198				Total Number of Endpoints: 198				Total Number of Endpoints: 156

All user specified timing constraints are met.

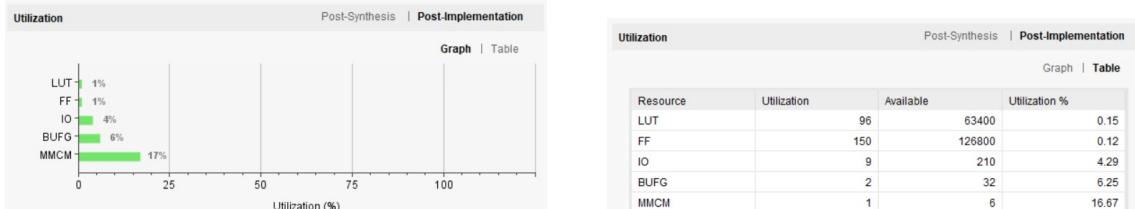
Figura 2.18: Multiplicador configurable - Análisis de tiempo

2.4.3. Multiplicador configurable: Utilización de recursos

Los resultados con respecto a la utilización de recursos del multiplicador configurable se muestran en las siguientes capturas. En ellas se puede ver que la utilización de LUT y FF es muy pequeña (< 1 %), por lo que en principio se podrían llegar a implementar varios multiplicadores configurables. También vemos que se utiliza aproximadamente un 17 % de MMCM. Los módulos MMCM se utilizan para generar múltiples señales de reloj. En este caso, solo se está utilizando un MMCM de los seis que hay, y como todos los elementos del circuito funcionan a la misma frecuencia, con uno es suficiente. Por tanto, aunque aparente ser limitante, el diseño no se vería limitado por este módulo.

Name	$\wedge 1$	Slice LUTs (63400)	Slice Registers (126800)	Slice (1585 0)	LUT as Logic (63400)	LUT Flip Flop Pairs (63400)	Bonded IOB (210)	BUFGCTRL (32)	MMCM2_ADV (6)
top	96	150	37	96	80	9	2	1	
CM (configurable_mult...)	75	102	30	75	60	0	0	0	0
> BIT4_MUL_PES0 (...)	9	8	6	9	7	0	0	0	0
> BIT4_MUL_PES4 (...)	10	8	5	10	7	0	0	0	0
> BIT4_MUL_PES8 (...)	10	8	5	10	7	0	0	0	0
> BIT4_MUL_PES12 (...)	10	8	4	10	7	0	0	0	0
> CONTROLADOR (...)	26	10	10	26	8	0	0	0	0
> FAs_BLOCK0 (fa_blo...)	2	10	6	2	2	0	0	0	0
> FAs_BLOCK4 (fa_blo...)	2	10	7	2	2	0	0	0	0
> FAs_BLOCK8 (fa_blo...)	3	10	6	3	2	0	0	0	0
> FAs_BLOCK12 (fa_b...)	2	10	7	2	2	0	0	0	0
> PIP0_B (PIP0)	1	4	2	1	0	0	0	0	0
> PIP0_CB (PIP0_6)	0	4	2	0	0	0	0	0	0
> PISO_A (PISO_M_M)	0	4	2	0	0	0	0	0	0
> PISO_CA (PISO_M_M...)	0	4	1	0	0	0	0	0	0
> PISO_CB (PISO_N)	0	4	1	0	0	0	0	0	0
> SIN (shifter_in)	5	16	9	5	0	0	0	0	0
> SOUT (shifter_out)	16	32	5	16	16	0	0	0	0
> Udk_mod (clk_mod)	0	0	0	0	0	0	2	1	

(a) Análisis detallado de utilización de recursos



(b) Análisis de utilización de recursos (Gráfico)

(c) Análisis de utilización de recursos (Tabla)

Figura 2.19: Multiplicador configurable - Análisis de utilización de recursos

Capítulo 3

Función de activación

Otra de las partes fundamentales de la neurona artificial configurable entre precisión y latencia que se está llevando a cabo en este Trabajo de Fin de Grado es la función de activación. Al igual que el multiplicador desarrollado en el capítulo anterior, esta función de activación también deberá ser configurable con el objetivo de lograr esta propiedad de configurabilidad deseada en la neurona artificial.

La función de activación es la unidad que proporciona a las redes neuronales artificiales la capacidad de aprender y reconocer patrones entre los datos. Cada una de las neuronas que forman una red neuronal tiene asociada una función de activación. La tarea de dicha función de activación es tomar la información computada en una neurona y transformarla de tal manera que pueda ser interpretada por otras neuronas. Realizando la comparativa con una neurona biológica, la función de activación se correspondería con el axón, parte de la neurona que se encarga de enviar los impulsos eléctricos generados hacia otras neuronas.

Como se vio en la introducción, la salida computada por una neurona viene dada por el producto interno entre los datos de entrada y los pesos $u = \sum_{i=1}^n w_i x_i = w^T x$. El problema de esta salida u es que no está limitada, por lo que en ausencia de una función de activación, puede tomar valores entre $[-\infty, +\infty]$, pudiendo dar lugar a problemas computacionales. Además, esta salida u no deja de ser una combinación lineal de los datos de entrada, lo cual limita las capacidades de la neurona para abordar, por ejemplo, problemas de clasificación no lineales. Las funciones de activación son por tanto fundamentales, ya que permiten limitar los valores de salida, típicamente entre los rangos $[0,1]$ o $[-1,1]$, y dotan a las neuronas de no linealidad.

Las funciones de activación se dividen generalmente en dos tipos: lineales y no lineales. Las más utilizadas son las funciones no lineales, ya que proporcionan a las neuronas de esta propiedad de no linealidad, permitiendo así abordar un abanico más amplio de problemas. Dentro de las funciones de activación no lineales existen diferentes tipos: sigmoide, tahn, ReLu, softplus, etc. De todas estas funciones no lineales se va a utilizar la función sigmoide como función de activación de la neurona artificial que se está desarrollando en este trabajo.

Este capítulo se organiza de la siguiente forma: en la sección 3.1 se realizará una breve introducción de la función sigmoide y se escogerá la metodología que se va a seguir para la implementación de la misma. En la sección 3.2 se realizará el diseño e implementación en VHDL del módulo que implementará la función sigmoide buscando enfocarla hacia la configurabilidad. De este módulo se describirán las partes básicas que lo forman y se realizarán una serie de simulaciones para explicar su principio de funcionamiento. Por último, en la sección 3.3 se realizará un análisis de los resultados de latencia, throughput, frecuencia y utilización de recursos obtenidos de la función sigmoide.

3.1. Función sigmoide

La función sigmoide se trata de una función no lineal popularmente utilizada como función de activación de redes neuronales artificiales que proporciona un rango de valores comprendidos entre [0,1]. Aunque hoy en día existen funciones de activación que proporcionan mejores características que la función sigmoide, esta sigue siendo una de las más utilizadas. Alguno de los problemas en los que se suele utilizar este tipo de función de activación es en problemas de clasificación binarios. La función sigmoide viene dada por la siguiente expresión:

$$f(x) = \frac{1}{1 + e^{-x}} \quad (3.1)$$

Y tiene la siguiente curva:

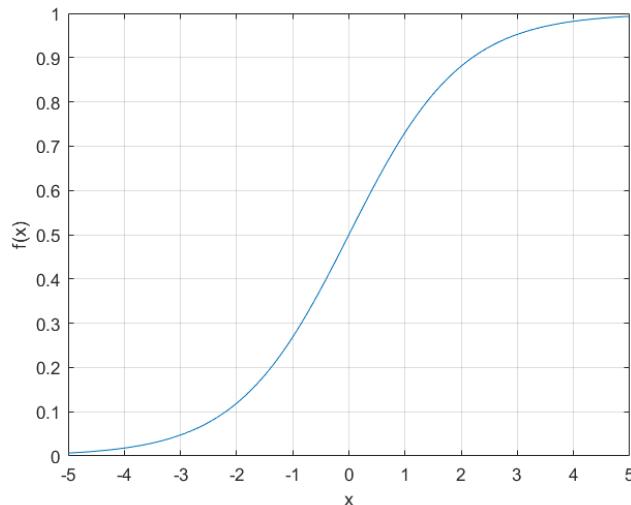


Figura 3.1: Curva de la función sigmoide

La implementación digital directa de la expresión matemática de la función sigmoide (3.1) es poco práctica, ya que consiste en una exponencial y en una operación de división que requieren tiempo de computación y excesivos recursos lógicos. Sin embargo, existen diferentes aproximaciones matemáticas que facilitan su implementación. A continuación, se describen algunas de estas:

Implementación por flujo de datos

Esta implementación consiste en formular la expresión matemática de la función sigmoide (3.1) de forma que no contenga ninguna operación trascendental como la exponencial. Una manera de formular dicha expresión puede ser como la que se muestra en [5]:

$$f(x) = \frac{1}{2} \left(\frac{x}{1 + |x|} + 1 \right) \quad (3.2)$$

Aproximación por tramos lineales

Esta aproximación se basa en dividir la función sigmoide en tramos de primer orden como los que se presentan en la siguiente tabla [6]:

$f(x)$	Condiciones
1	$ x \geq 5$
$ x /2^5 + 0,84375$	$2,375 \leq x < 5$
$ x /2^3 + 0,625$	$1 \leq x < 2,375$
$ x /2^2 + 0,5$	$0 \leq x < 1$

Tabla 3.1: Aproximación por tramos lineales en el intervalo [-5,5]

Aproximación por tramos de segundo orden

La función sigmoide también puede ser representada como una aproximación por tramos de segundo orden como la que se muestra a continuación [6]:

$$f(x) = \begin{cases} \frac{1}{2} \left(\frac{1}{4}x + 1 \right)^2 & \text{para } -4 < x < 0 \\ 1 - \frac{1}{2} \left(\frac{1}{4}x - 1 \right)^2 & \text{para } 0 \leq x < 4 \end{cases} \quad (3.3)$$

Tabla de búsqueda

Otra forma de implementar la función sigmoide es empleando una tabla de búsqueda. Este método consiste en una tabla de verdad almacenada en una memoria ROM que contiene el resultado esperado para cada valor de la entrada. Los niveles de exactitud que pueden alcanzarse con este método dependen de la cantidad de puntos que sean almacenados en la memoria. Si se aumenta la cantidad de puntos la función será más precisa, pero aumentará el tamaño de la memoria, y por consiguiente, el consumo de recursos será mayor.

Expansión en serie de Taylor

Otra alternativa para implementar la función sigmoide es a partir de su expansión en serie de Taylor. La expansión en serie de Taylor de la función sigmoide se puede realizar a partir de un único polinomio o dividida en intervalos. La principal ventaja de utilizar un desarrollo en serie de Taylor por intervalos con respecto a un único polinomio es que se obtiene una mejor aproximación de la función sigmoide para un rango más amplio de valores, proporcionando así, un mayor grado de precisión a la neurona. En [5] se realiza una aproximación de la función sigmoide a partir de su desarrollo en serie de Taylor dividido en 3 intervalos:

$$f(x) = \begin{cases} 0,571859 + (0,392773)x + (0,108706)x^2 + \\ +(0,014222)x^3 + (0,000734)x^4 & \text{para } -\infty < x \leq -1,5 \\ \frac{1}{2} + \frac{1}{4}x - \frac{1}{48}x^3 + \frac{1}{480}x^5 & \text{para } -1,5 < x < 1,5 \\ 0,428141 + (0,392773)x - (0,108706)x^2 + \\ +(0,014222)x^3 - (0,000734)x^4 & \text{para } 1,5 \leq x < \infty \end{cases} \quad (3.4)$$

En [5] se muestra que la implementación de la función sigmoide a partir de su expansión en serie de Taylor por intervalos ofrece una precisión de un 93,1 %, situándola por encima

de la implementación por flujo de datos (91,8 %) y de la aproximación por tramos lineales (92,5 %). El método de aproximación por tabla de búsqueda no será una opción para esta implementación debido a la gran cantidad de recursos que puede llegar a consumir. Por tanto, de todas las aproximaciones mencionadas, la expansión en serie de Taylor por intervalos es el método de aproximación que se va a seguir para la implementación de la función sigmoide.

La siguiente figura muestra la comparativa entre la función sigmoide y su expansión en serie de Taylor por intervalos:

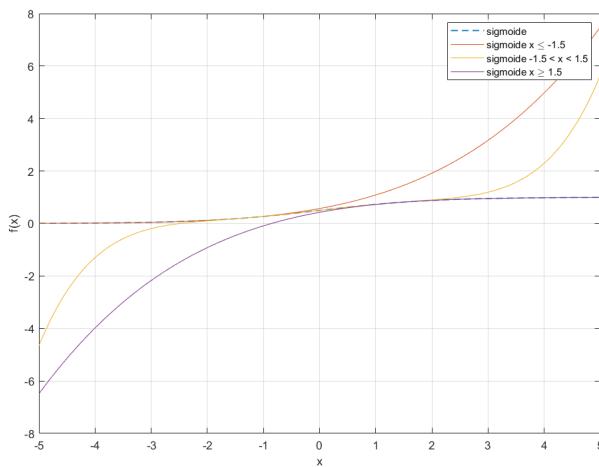
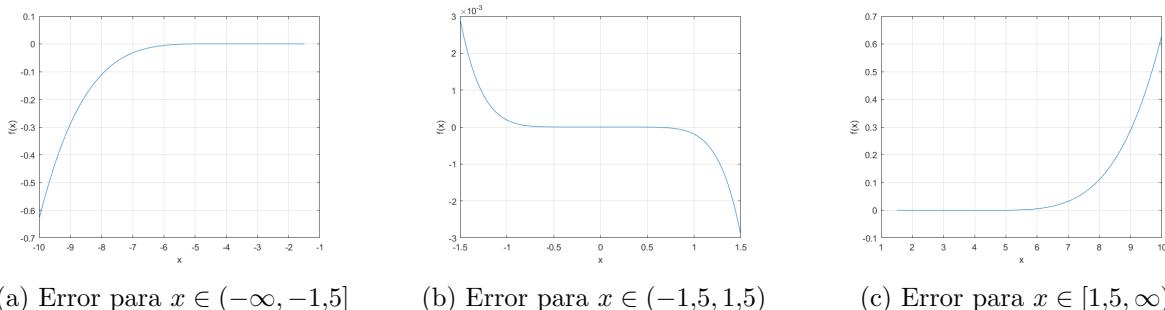


Figura 3.2: Aproximación polinómica de la función sigmoide

Las siguientes gráficas muestran el error cometido de aproximar la función sigmoide a partir de expansión en serie de Taylor por intervalos:



(a) Error para $x \in (-\infty, -1,5]$

(b) Error para $x \in (-1,5, 1,5)$

(c) Error para $x \in [1,5, \infty)$

Figura 3.3: Gráficas de error de la expansión en serie de Taylor por intervalos

En la gráfica 3.3a vemos que la expansión en serie de Taylor en el intervalo $(-\infty, -1,5]$ ofrece una buena aproximación de la función sigmoide para valores comprendidos entre $[-6, -1,5]$. Sin embargo, a medida que nos desplazamos hacia valores más negativos que estos, el error empieza a aumentar considerablemente. Por otro lado, en la gráfica 3.3b vemos que, para el intervalo $(-1,5, 1,5)$, se obtiene una muy buena aproximación de la función sigmoide, con un error del orden de 10^{-3} , el cual se puede considerar despreciable. Y por último, la gráfica 3.3c muestra el error de la aproximación para el intervalo $[1,5, \infty)$. Vemos que esta última gráfica es prácticamente igual que la gráfica 3.3a, salvo que esta es para valores positivos.

Con esto vemos que, para el número de términos escogidos de la expansión serie de Taylor en cada uno de los intervalos (3.4), se obtiene una buena aproximación de la función sigmoide para el rango de valores [-6,6], con un error que no supera el orden de las centésimas.

3.2. Implementación de la función sigmoide

En esta sección se va a describir el diseño y la implementación en VHDL de un módulo que implementa una función sigmoide configurable a partir de su expansión en serie de Taylor por intervalos.

En la figura 3.4 se muestran los diferentes bloques que implementan la función sigmoide a partir de su expansión en serie de Taylor por intervalos. Esta arquitectura que se muestra se trata de una arquitectura simplificada en la que realmente hay 4 bloques sigmoid left (SL), 4 bloques sigmoid middle (SM), 4 bloques sigmoid right (SR) y 4 sumadores y 4 registros, cada uno con un multiplexor asociado. El resto de bloques son los que se muestran en la figura, es decir, todos hacen uso de un único multiplicador y un único controlador.

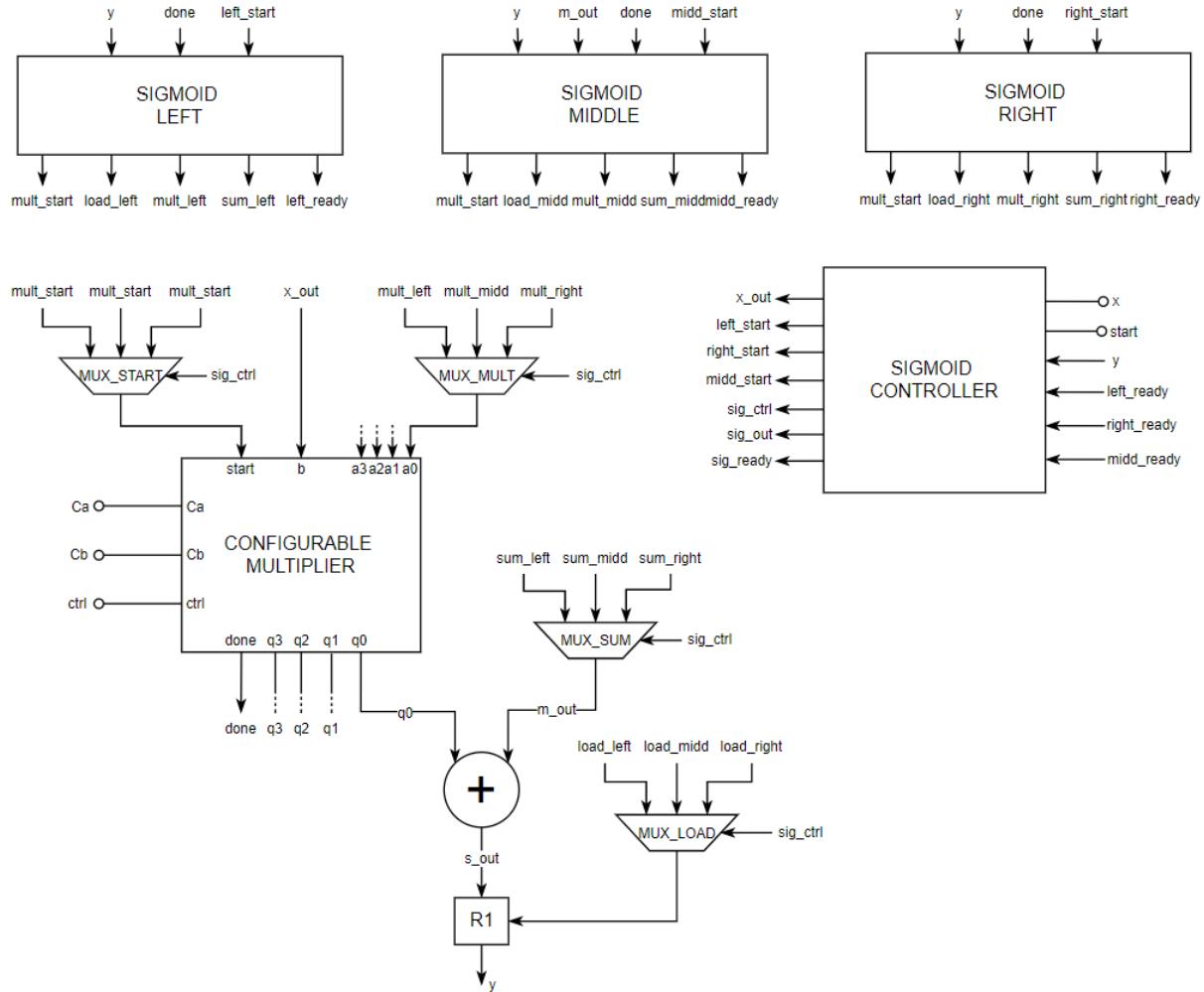


Figura 3.4: Arquitectura de la función sigmoide

Los bloques sigmoid left (SL), sigmoid middle (SM) y sigmoid right (SR) se encargan de proporcionar los coeficientes de los polinomios de la expansión en serie de Taylor en cada

intervalo (3.4). El bloque SL proporciona los coeficientes del polinomio correspondiente al intervalo $(-\infty, -1,5]$, el bloque SM los coeficientes del polinomio correspondiente al intervalo $(-1,5, 1,5)$, y el bloque SR los coeficientes del polinomio correspondiente al intervalo $[1,5, \infty)$. Además de estos coeficientes, estos bloques también proporcionan diferentes señales de control que van al multiplicador, a los registros y al controlador a través de multiplexores.

El multiplicador se utiliza para multiplicar los datos de entrada de la función sigmoide con los coeficientes de los polinomios. Cabe destacar que se está utilizando como multiplicador el multiplicador configurable descrito en 2, con el objetivo de enfocar esta función de sigmoide hacia la configurabilidad entre precisión y latencia. Por un lado, en la entrada paralela del multiplicador (b) se introduce el dato de entrada de la función sigmoide proporcionado por el controlador (x_out). Este dato de entrada se trata de un dato de 16 bits que, en función de la configuración del multiplicador, se divide en 4 datos de 4 bits, en 2 datos de 8 bits, o se considera como un único dato de 16 bits. Por otro lado, en las entradas serie del multiplicador ($a3 - a0$) se introducen, a través de multiplexores (MUX_MULT), los coeficientes de los polinomios procedentes de los bloques de coeficientes. Es necesario remarcar que el multiplicador configurable descrito en 2 se realizó con una única entrada serie y en este caso se están utilizando 4. Para poder disponer de 3 entradas serie más, basta con añadir otros 3 registros de desplazamiento PISO_M y conectar cada uno a un bloque PE_block (ver figura 2.14).

Por tanto, dependiendo de la configurabilidad del multiplicador se pueden realizar 4 multiplicaciones simultáneas cuyos resultados son proporcionados en las salidas $q3, q2, q1$ y $q0$, 2 multiplicaciones cuyos resultados se obtienen en $q2$ y $q0$, o una única multiplicación cuyo resultado es mostrado en $q0$. Posteriormente, estos datos de salida se introducen en unos sumadores que se encargan de realizar la suma entre los resultados de las multiplicaciones y los coeficientes de los polinomios proporcionados también por los bloques de coeficientes a través multiplexores (MUX_SUM). Finalmente, los resultados de estas operaciones de multiplicación y suma se almacenan en unos registros para, posteriormente, ser realimentados a los bloques de coeficientes o para proporcionar los resultados de la función sigmoide al controlador.

El controlador de la sigmoide es el encargado de enviar las señales de activación a los bloques de coeficientes en función de un dato de entrada x de 16 bits. Dependiendo de la configuración que se quiera utilizar, este controlador divide el dato de entrada x en 4 datos de 4 bits, en 2 datos de 8 bits, o toma los 16 bits del dato de entrada como un único dato. A continuación, comprueba a qué intervalo pertenecen cada uno estos datos resultantes y, en función de los intervalos en los que se encuentren los diferentes datos, el controlador activa los bloques de coeficientes correspondientes. Este controlador es también el encargado de controlar los diferentes multiplexores que controlan las entradas del multiplicador, los sumadores y los registros además de proporcionar los resultados de la función sigmoide a partir de la información de estado recibida de los bloques SL, SM y SR.

De esta forma, los elementos fundamentales que dotan a esta función sigmoide de la capacidad de configurabilidad son el multiplicador configurable, que gracias a sus opciones de configurabilidad se pueden conseguir 1, 2 o 4 funciones sigmoide con diferentes precisiones y latencias, y el controlador, que se encarga de orquestar todos los bloques que implementan la función sigmoide.

A continuación, se explican cada uno de los bloques que implementan esta función sigmoide. Para simplificar la explicación del funcionamiento de cada uno de estos bloques,

en lo que sigue se va a considerar una única entrada de coeficientes y una única salida de resultados en el multiplicador configurable, siendo la estructura resultante de esta consideración parecida a la que se muestra en la figura 3.4.

3.2.1. Bloque Sigmoid Left

Para la implementación del bloque sigmoid left se ha escogido un método de diseño basado en una ruta de datos y un controlador.

Bloque Sigmoid Left: Ruta de datos

La primera etapa en el diseño de un sistema basado en una ruta de datos es la descripción algorítmica de la especificación funcional del sistema en cuestión. A partir de esta descripción algorítmica se construye la ruta de datos.

Para implementar el algoritmo de este bloque se ha seguido un proceso de diseño basado en síntesis de alto nivel (High Level Synthesis en inglés). La síntesis en alto nivel se trata de un proceso de diseño automatizado que toma como entrada una descripción algorítmica para crear el hardware digital que implementa la función deseada. Este proceso se suele dividir en las siguientes etapas: (1) asignación, se realiza la selección del número y tipo de unidades hardware que van a implementar el algoritmo, (2) planificación temporal, se distribuyen en el tiempo las diferentes operaciones, (3) vinculación, se asocia cada una de las operaciones a una unidad concreta y se especifica a qué puerto concreto se conecta cada operando, e (4) implementación, se analiza el tiempo de vida de las variables, se determina el número de registros y multiplexores necesarios y, por último, se crea la estructura hardware.

Como se ha comentado anteriormente, la función principal de este bloque es proporcionar los coeficientes del polinomio de la expansión en serie de Taylor de la función sigmoide en el intervalo $(-\infty, -1,5]$:

$$y = 0,571859 + (0,392773)x + (0,108706)x^2 + (0,014222)x^3 + (0,000734)x^4 \quad (3.5)$$

Para reducir el número de operaciones y recursos necesarios, el polinomio anterior se puede reordenar de la siguiente manera:

$$y = 0,571859 + x(0,392773 + x(0,108706 + x(0,014222 + 0,000734x))) \quad (3.6)$$

Utilizando únicamente multiplicadores y sumadores, se obtiene el siguiente diagrama de flujo:

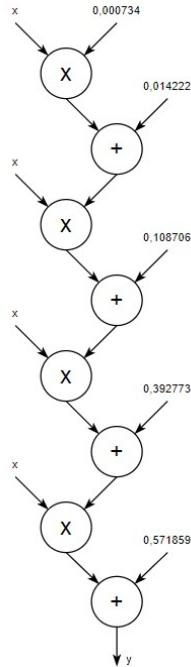


Figura 3.5: Bloque sigmoid left - Diagrama de flujo

A partir de este diagrama de flujo y empleando una asignación de un multiplicador y un sumador, se ha realizado la planificación temporal, el análisis de los tiempos de vida de las variables y se han determinado el número de registros necesarios:

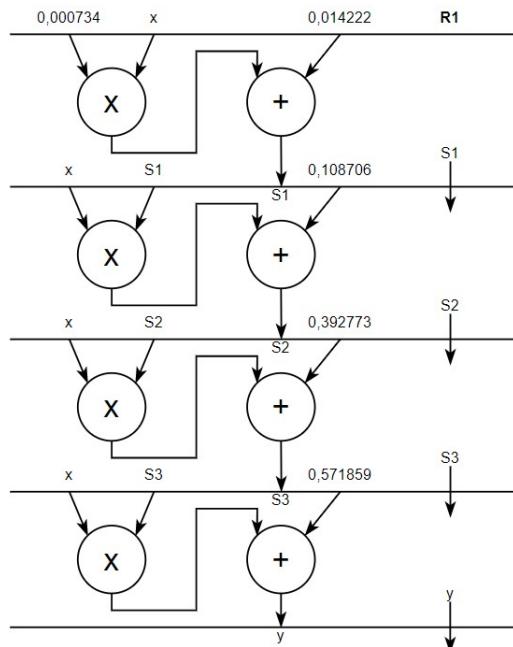


Figura 3.6: Bloque sigmoid left - Planificación temporal

En la figura 3.6 cada línea horizontal separa las operaciones que se realizan durante un ciclo de control del siguiente. Vemos que en cada ciclo de control se realiza una operación de multiplicación y una suma. La duración de cada uno de estos ciclos de control viene

determinada principalmente por el multiplicador. El multiplicador que se utiliza es el multiplicador configurable descrito en 2, que como ya se vio, su latencia varía en función del número de bits de los operandos de entrada. En la figura también se ha representado, mediante flechas verticales, el tiempo de vida de las distintas variables. El máximo número de flechas paralelas establece el número de registros necesarios, por lo que en este caso solo es necesario un registro. Los operadores y el registro reciben datos distintos a lo largo del tiempo, es por ello que son necesarios una serie de multiplexores que controlen qué datos se introducen en cada unidad en cada momento.

Finalmente, teniendo en cuenta toda la información anterior, se construye la estructura hardware que implementa el algoritmo de la función sigmoide en el intervalo $(-\infty, -1,5]$:

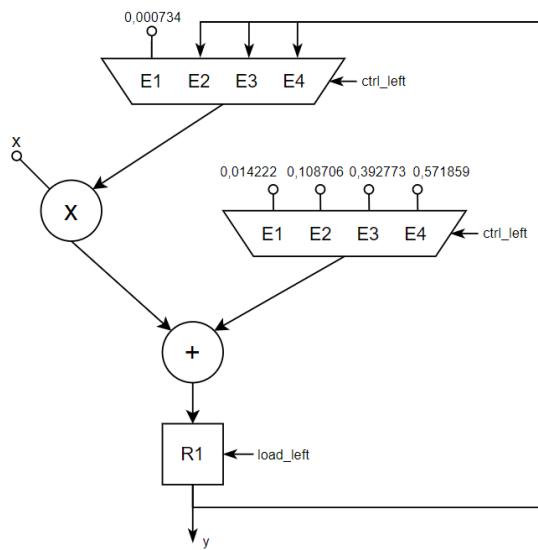


Figura 3.7: Bloque sigmoid left - Ruta de datos

Bloque Sigmoid Left: Controlador

Una vez diseñada la ruta de datos del bloque, es necesario un controlador que se encargue de proporcionar las señales de control necesarias a los diferentes elementos que forman la ruta de datos.

La siguiente tabla muestra un cronograma que especifica qué señal deja pasar cada multiplexor y entre paréntesis la señal de control asociada, así como los datos que se almacenan en el registro en cada instante de tiempo.

	t1	t2	t3	t4	t5
M1	0,000734 (00)	R1 (01)	R1 (10)	R1 (11)	
M2	0,014222 (00)	0,108706 (01)	0,392773 (10)	0,571859 (11)	
R1		S1	S2	S3	y

Tabla 3.2: Bloque sigmoid left - Cronograma

Para la implementación de este controlador se ha empleado una metodología basada en una máquina de estados finitos (FSMD). En la siguiente figura se muestra su diagrama ASMD:

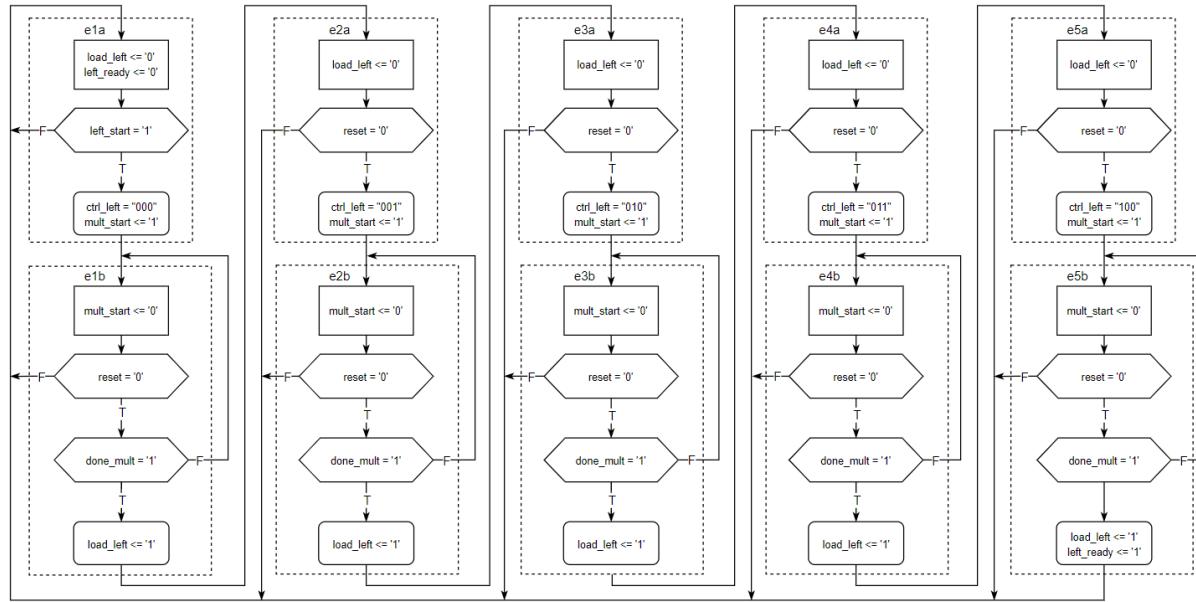


Figura 3.8: Controlador del bloque sigmoid left - Diagrama ASMD

Descripción de puertos:

- **left_start**: entrada de control procedente del controlador de la sigmoide (3.2.4) que indica, cuando está a '1', que $x \in (-\infty, -1,5]$ y, por tanto, activa este bloque.
- **done_mult**: entrada de control procedente del multiplicador configurable que indica, cuando está a '1', que el resultado de la multiplicación está disponible.
- **ctrl_left**: señal de control de los multiplexores.
- **mult_start**: salida de control que se envía al multiplicador configurable para iniciar el proceso de multiplicación.
- **load_left**: señal de control que habilita la carga de datos del registro cuando está a '1'.
- **left_ready**: salida de control que indica al controlador de la sigmoide, cuando está a '1', que se han proporcionado todos los coeficientes.

Este controlador está compuesto por 10 estados que se pueden agrupar en dos tipos:

Estados eXa

Los estados eXa son los estados en los que se cambia de ciclo, es decir, se cambia el valor de la señal de control de los multiplexores y se pone a '1' la salida de control $mult_start$ para iniciar la multiplicación con un nuevo coeficiente. En estos estados también se pone a '0' la salida de control $load_left$ del registro, de forma que mantenga los datos mientras se realiza el proceso de multiplicación. Cabe destacar que el estado $e1a$, al ser el primer estado que se ejecuta, es un poco diferente al resto. Lo que cambia en este estado con respecto a los demás es que se pone a '0' la salida de control $left_ready$.

Estados eXb

Los estados eXb son los estados de espera de la multiplicación. Se entra en estos estados cada vez que se inicia el proceso de multiplicación y no se sale hasta que la multiplicación no termina ($done_mult = '1'$). Una vez que el proceso de multiplicación termina, se pone a ' 1 ' la salida de control $load_left$ del registro para almacenar el resultado de las operaciones realizadas.

3.2.2. Bloque Sigmoid Right

La implementación del bloque sigmoid right es muy parecida a la del bloque sigmoid left. En este caso también se ha realizado una implementación basada en una ruta de datos y un controlador.

Bloque Sigmoid Right: Ruta de datos

Al igual que en el bloque sigmoid left, para implementar la ruta de datos de este bloque se ha seguido un proceso de diseño basado en síntesis de alto nivel (HLS).

La función principal de este bloque es proporcionar los coeficientes del polinomio de la expansión en serie de Taylor de la función sigmoide en el intervalo $[1,5, \infty)$:

$$y = 0,428141 + (0,392773)x - (0,108706)x^2 + (0,014222)x^3 - (0,000734)x^4 \quad (3.7)$$

Reordenando el polinomio anterior nos queda:

$$y = 0,428141 + x(0,392773 + x(-0,108706 + x(0,014222 - 0,000734x))) \quad (3.8)$$

Realizando las etapas de asignación, planificación temporal, vinculación e implementación, se llega a la siguiente estructura hardware que implementa el algoritmo de la función sigmoide en el intervalo $[1,5, \infty)$, la cual vemos que es prácticamente igual que la del bloque sigmoid left:

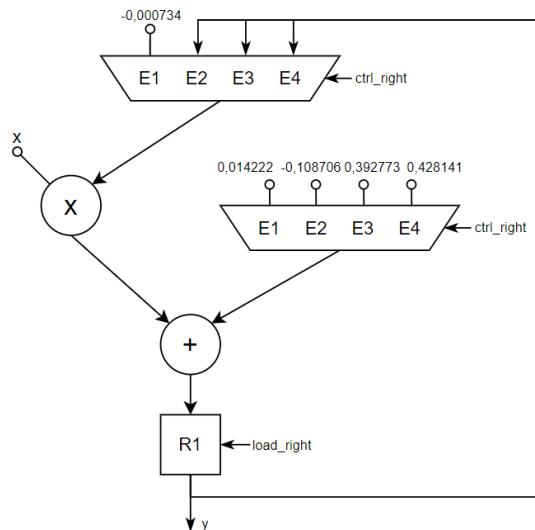


Figura 3.9: Bloque sigmoid right - Ruta de datos

El número de registros y de multiplexores que se utilizan es el mismo, y las conexiones son también iguales. Lo único que cambia son los coeficientes que se introducen en los

operadores.

Bloque Sigmoid Right: Controlador

Una vez diseñada la ruta de datos del bloque, es necesario un controlador que se encargue de proporcionar las señales de control necesarias a los diferentes elementos que forman la ruta de datos.

La siguiente tabla muestra el cronograma de señales de este bloque:

	t1	t2	t3	t4	t5
M1	-0,000734 (00)	R1 (01)	R1 (10)	R1 (11)	
M2	0,014222 (00)	-0,108706 (01)	0,392773 (10)	0,428141 (11)	
R1		S1	S2	S3	y

Tabla 3.3: Bloque sigmoid right - Cronograma

Para la implementación de este controlador también se ha empleado una metodología basada en una máquina de estados finitos (FSMD). En la siguiente figura se muestra su diagrama ASMD:

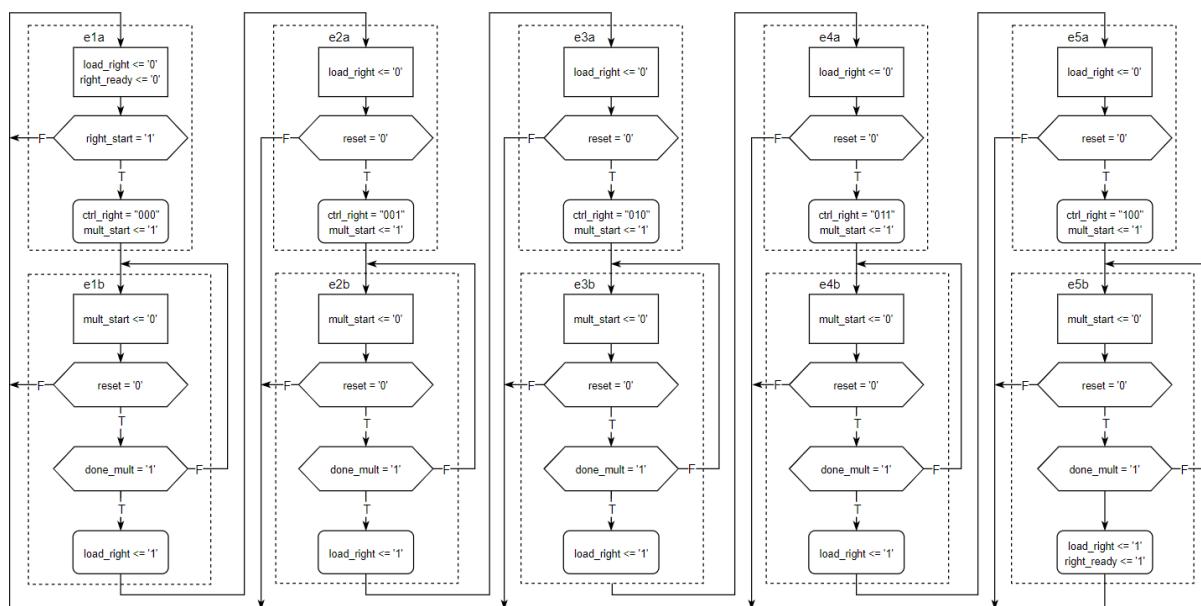


Figura 3.10: Controlador del bloque sigmoid right - Diagrama ASMD

Las entradas y salidas de este controlador realizan funciones parecidas a las del controlador del bloque sigmoid left.

Este controlador está compuesto por 10 estados que, al igual que en el controlador del bloque sigmoid left, se pueden agrupar en 2 estados, eXa y eXb , que realizan funciones similares.

3.2.3. Bloque Sigmoid Middle

Para la implementación de este bloque también se ha escogido un método de diseño basado en una ruta de datos y un controlador. Sin embargo, su estructura difiere ligeramente de los bloques anteriores debido al algoritmo que implementa.

Bloque Sigmoid Middle: Ruta de datos

Para implementar la ruta de datos de este bloque se ha seguido un proceso de diseño basado en síntesis de alto nivel (HLS).

La función principal de este bloque es proporcionar los coeficientes del polinomio de la expansión en serie de Taylor de la función sigmoide en el intervalo $(-1,5, 1,5)$:

$$y = \frac{1}{2} + \frac{1}{4}x - \frac{1}{48}x^3 + \frac{1}{480}x^5 \quad (3.9)$$

Para poder utilizar los mismos recursos que las rutas de datos de los dos bloques anteriores (multiplicador, sumador y registro), el polinomio anterior se ha reordenado de la siguiente manera:

$$y = \frac{1}{2} + x \left(\frac{1}{4} + x \left(x \left(-\frac{1}{48} + x \left(\frac{1}{480}x \right) \right) \right) \right) \quad (3.10)$$

Utilizando únicamente multiplicadores y sumadores, se obtiene el siguiente diagrama de flujo:

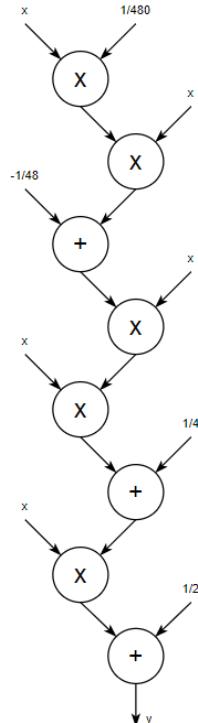


Figura 3.11: Bloque sigmoid middle - Diagrama de flujo

A partir de este diagrama de flujo y empleando una asignación de un multiplicador y un sumador, se ha realizado la planificación temporal, el análisis de los tiempos de vida de las variables y se han determinado el número de registros necesarios:

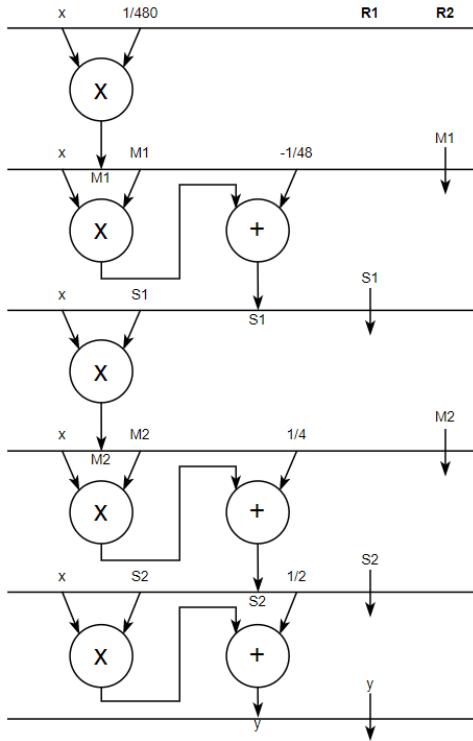


Figura 3.12: Bloque sigmoid middle - Planificación temporal

En la figura 3.12 vemos que, a diferencia de los bloques anteriores, se obtiene una planificación temporal de 6 ciclos y, además, el número de registros que se utilizan en este caso es dos. Aunque se podría minimizar el número de registros a uno, ya que las variables del circuito no se solapan en ningún momento, el número de registros del sistema completo no cambiaría en ningún caso. En caso de utilizar un único registro, el sistema global estaría formado por el registro compartido por las rutas de datos de los bloques anteriores y este. Y en caso de utilizar dos registros, uno de ellos sería compartido junto con la ruta de datos de los bloques anteriores, y el otro sería utilizado exclusivamente en este bloque. Por lo tanto, se utilicen uno o dos registros en esta ruta de datos, el número de registros del sistema global es siempre dos. La ventaja de realizar esta ruta de datos con dos registros es que se reduce en uno el número de multiplexores necesarios.

Una vez realizadas estas fases, se pasa a construir la estructura hardware que implementa el algoritmo de la función sigmoide en el intervalo $(-1,5, 1,5)$:

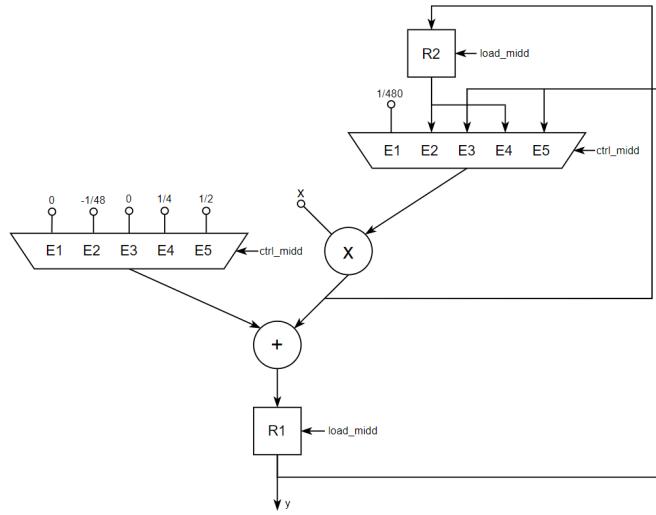


Figura 3.13: Bloque sigmoid middle - Ruta de datos

Bloque Sigmoid Middle: Controlador

Una vez diseñada la ruta de datos del bloque, es necesario un controlador que se encargue de proporcionar las señales de control necesarias a los diferentes componentes que forman la ruta de datos.

La siguiente tabla muestra el cronograma de las señales de este bloque:

	t1	t2	t3	t4	t5	t6
M1	$\frac{1}{480} (000)$	R1 (001)	R2 (010)	R1 (011)	R2 (100)	
M2	0 (000)	$\frac{-1}{48} (001)$	0 (010)	$\frac{1}{4} (011)$	$\frac{1}{2} (100)$	
R1		$\frac{x}{480}$	$\frac{x^2}{480}$	$\frac{-x}{48} + \frac{x^3}{480}$	$\frac{-x^2}{48} + \frac{x^4}{480}$	$\frac{x}{4} + \frac{-x^3}{48} + \frac{x^5}{480}$
R2		$\frac{x}{480}$	$\frac{-1}{48} + \frac{x^2}{480}$	$\frac{-x}{48} + \frac{x^3}{480}$	$\frac{1}{4} + \frac{-x^2}{48} + \frac{x^4}{480}$	$\frac{1}{2} + \frac{x}{4} + \frac{-x^3}{48} + \frac{x^5}{480}$

Tabla 3.4: Bloque sigmoid middle - Cronograma

Para la implementación de este controlador se ha empleado una metodología de FSMD. En la siguiente figura se muestra el diagrama ASMD asociado:

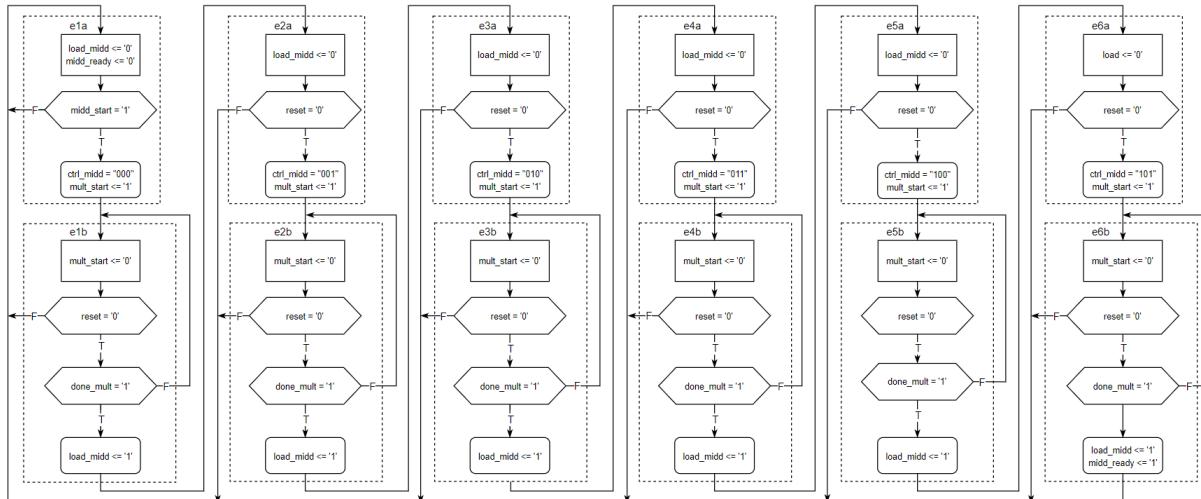


Figura 3.14: Controlador del bloque sigmoid middle - Diagrama ASMD

Las entradas y salidas de este controlador realizan funciones parecidas a las de los controladores de los bloques anteriores.

Este controlador está formado por 12 estados, 2 más que los controladores de los bloques anteriores debido a que el algoritmo de este bloque tarda un ciclo de control más en ejecutarse. Sin embargo, los estados se pueden dividir, al igual que en los controladores anteriores, en dos estados que realizan las mismas funciones: eXa para los cambios de ciclo y eXb como estados de espera de la multiplicación.

3.2.4. Controlador de la función sigmoide

Como se comentó anteriormente, el controlador de la sigmoide es el encargado de proporcionar información de control a los bloques sigmoid left, sigmoid right, sigmoid middle y a los multiplexores que controlan las entradas del multiplicador, los sumadores y los registros. Además, este controlador es el encargado de proporcionar el dato de entrada al multiplicador y de proporcionar los resultados de la función sigmoide. Para implementar este controlador se ha escogido, al igual que en los controladores anteriores, una metodología FSMD cuyo diagrama ASMD asociado se muestra a continuación:

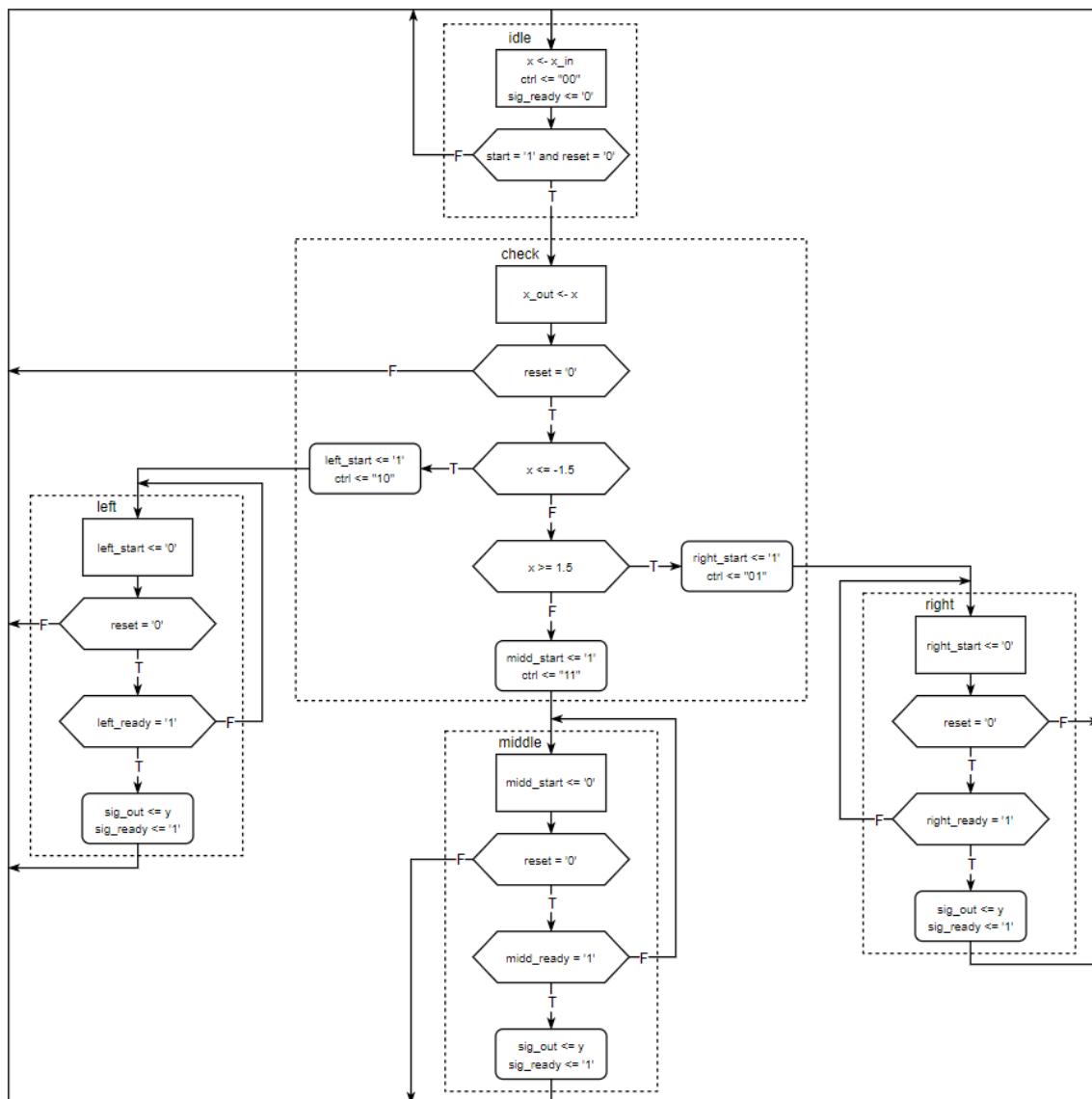


Figura 3.15: Controlador de la función sigmoide - Diagrama ASMD

Descripción de puertos:

- start: dato de entrada que controla el inicio del algoritmo.
- midd_ready, left_ready, right_ready: entradas de control procedentes de los bloques sigmoid left, sigmoid right y sigmoid middle que indican, cuando están a '1', que se han proporcionado todos los coeficientes del polinomio correspondiente.
- midd_start, left_start, right_start: salidas de control que activan los bloques sigmoid left, sigmoid right y sigmoid middle cuando están a '1'.
- sig_ctrl: salida que controla los multiplexores MUX_START, MUX_MULT, MUX_SUM, y MUX_LOAD.
- x_in: dato de entrada de N bits.
- x_out: dato de salida de N bits que registra el dato de entrada x_in y que se envía a la entrada paralelo del multiplicador configurable.
- y: dato de entrada correspondiente al resultado de la función sigmoide.
- sig_out: dato de salida que proporciona el resultado de la función sigmoide.
- sig_ready: salida que indica, cuando está a '1', que el resultado de la función sigmoide está disponible.

Este controlador dispone de 5 estados:

Estado idle

Estado de reposo. En este estado se registra el valor del dato de entrada, se pone a 0 el valor de la señal de control de los multiplexores y se pone a '0' la salida *sig_ready*. Si en este estado *start* se pone a '1' y *reset* es '0' se pasa al estado *check*.

Estado check

En este estado se comprueba el valor del dato de entrada *x*, y dependiendo de su valor, se pasa al estado *left*, *right* o *middle*. Si por ejemplo, $x \in (-\infty, -1,5]$, se pasa al estado *left* y se activa el bloque sigmoid left poniendo a '1' la salida de control *left_start*. Además, se cambia el valor de la señal de control de los multiplexores de forma que estos pasen a seleccionar las salidas del bloque sigmoid left.

Estados left, right y middle

El controlador entra en el estado *left* si $x \in (-\infty, -1,5]$ y se mantiene en él hasta que el bloque sigmoid left no termine de proporcionar todos los coeficientes del polinomio correspondiente. Cuando dicho bloque termina de proporcionar todos los coeficientes del polinomio (*left_ready* = '1'), el controlador proporciona el resultado en *sig_out* a partir del dato de entrada *y*, pone a '1' la salida de control *sig_ready* que informa de que el resultado ya está disponible y finalmente regresa al estado *idle*. Los otros dos estados, *right* y *middle*, son muy parecidos a este. Cuando se entra en ellos, se espera hasta que sus respectivos bloques terminen de proporcionar los coeficientes de los polinomios correspondientes, y una vez terminan, el controlador proporciona el resultado de la función y vuelve al estado *idle*.

3.2.5. Cuantificación de los datos

Una codificación posible para los coeficientes de los polinomios de la expansión en serie de Taylor es emplear palabras de 13 bits en Ca2 y en notación punto fijo $<1, 12>$. El rango de valores que se pueden representar con esta cuantificación es de:

$$-2^{13-1-12} \leq x \leq 2^{13-1-12} - \frac{1}{2^{12}} \rightarrow -1 \leq x \leq 0,9997$$

con una precisión de:

$$\frac{1}{2^{12}} = 2,4414 \times 10^{-4}$$

lo cual sería suficiente para poder representar estos coeficientes con un error de aproximación aceptable. Por otro lado, el dato de entrada que se introduce en la entrada paralela del multiplicador configurable tiene una longitud de 16 bits que, en función de la configuración del multiplicador configurable, podrá ser dividido en 4 datos de 4 bits, 2 datos de 8 bits o ser considerado como un único dato de 16 bits.

Un aspecto importante a destacar es que, al realizar tanto las operaciones de multiplicación como las operaciones de suma entre los diferentes coeficientes y los datos de entrada, se obtienen datos con un tamaño variable, por tanto, en cada caso habrá realizar el truncado y el redondeo de los datos necesario.

3.2.6. Ejemplo de la función sigmoide

A continuación, se muestra una simulación de la función sigmoide en la cual se ha realizado un barrido de valores de entrada dentro del rango [-4,4] con incrementos de 0,0625. Para esta simulación, los coeficientes de los polinomios de la expansión en serie de Taylor se han codificado con una longitud de 13 bits en Ca2 utilizando la notación punto fijo $<1, 12>$, y los datos de entrada se han codificado con una longitud de 8 bits representados en Ca2 en notación en punto fijo $<3, 5>$.

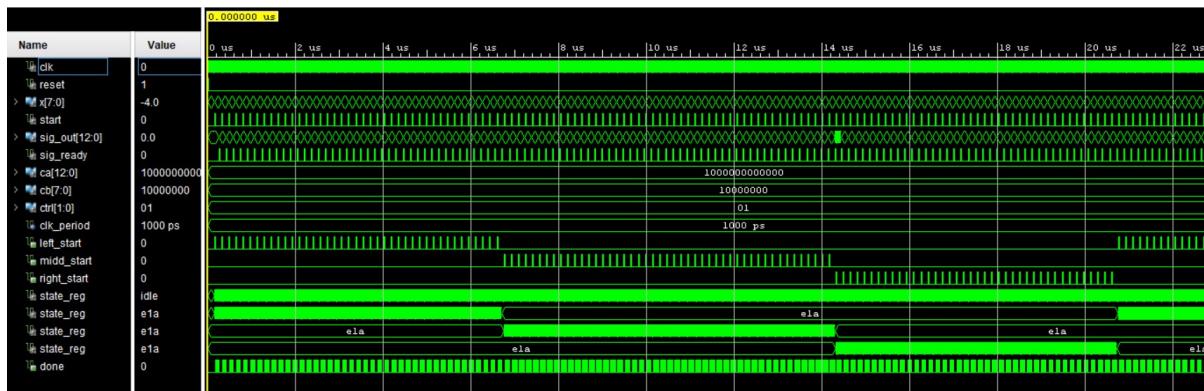
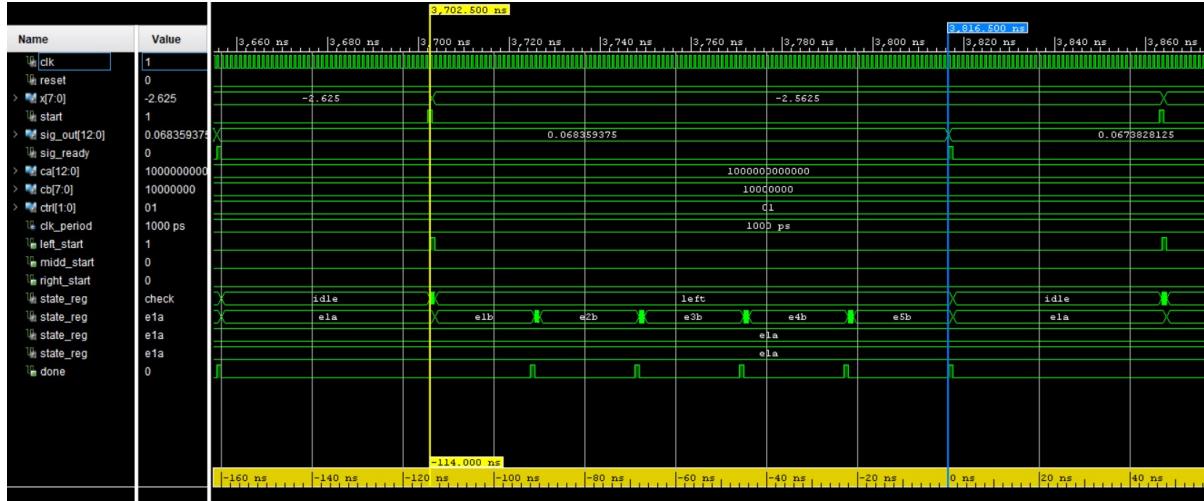
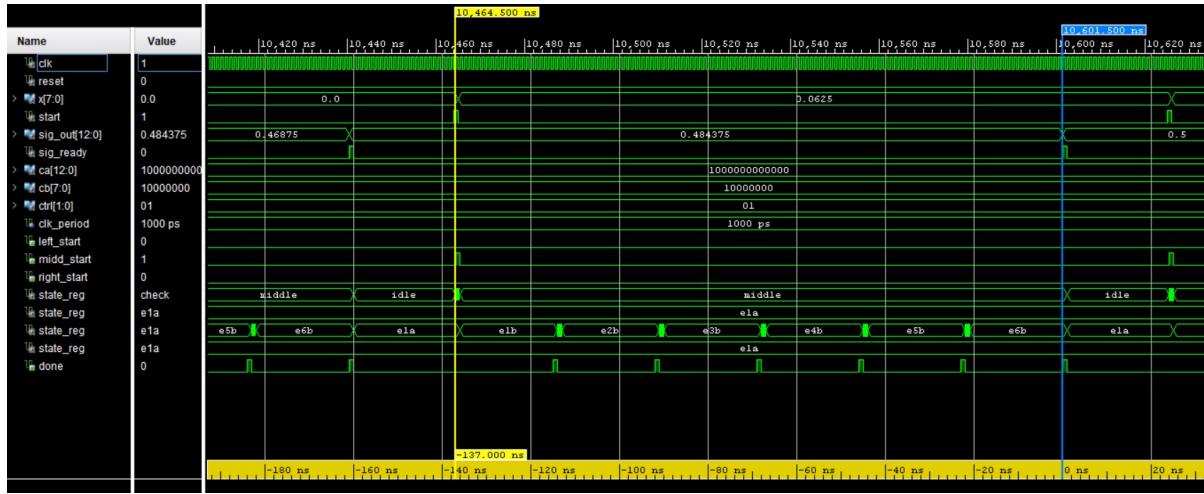


Figura 3.16: Testbench - Función sigmoide

En la simulación de la figura 3.16 se puede ver que, para valores de entrada comprendidos entre [-4,-1,5], se pone a '1' la señal *left_start* en diferentes instantes, activando el bloque sigmoid *left* que se encarga de proporcionar los coeficientes correspondientes al polinomio que implementa la función sigmoide en el intervalo $(-\infty, -1,5]$. Y lo mismo pasa con las señales *midd_start* y *right_start* para valores de entrada comprendidos entre (-1,5,1,5) y [1,5,4] respectivamente.

Figura 3.17: Testbench - Función sigmoide para $x \in (-\infty, -1,5]$

Ampliando la simulación anterior, en la captura de la figura 3.17 vemos que para un dato de entrada situado en el intervalo $(-\infty, -1,5]$ se tarda en proporcionar el resultado 114 ns, y como en este caso se ha definido un periodo de reloj de 1 ns, se tardan 114 ciclos de reloj en calcular el resultado. Esta latencia se debe principalmente al multiplicador configurable, que para este ejemplo, tarda $8 + 13 = 21$ ciclos de reloj en realizar una multiplicación. Y como para calcular el resultado de la función sigmoide para datos de entrada situados tanto en el intervalo $(-\infty, -1,5]$ como en el intervalo $[1,5, \infty)$ se requieren 5 ciclos de control, se tardan $21 \times 5 = 105$ ciclos de reloj en proporcionar el resultado, a los cuales hay que sumar 9 ciclos de reloj de transición entre estados, dando lugar a los 114 ciclos de reloj mencionados.

Figura 3.18: Testbench - Función sigmoide para $x \in (-1,5, 1,5)$

Como se vio en la subsección 3.2.3, para datos de entrada situados en el intervalo $(-1,5, 1,5)$ se tarda un ciclo de control más en obtener el resultado de la función sigmoide que en los otros dos intervalos, ya que se realiza una multiplicación más. Por tanto, para datos de entrada situados en dicho intervalo, la función sigmoide tarda $21 \times 6 = 126$ ciclos de reloj en calcular el resultado, a los que hay que sumar $6 + 5 = 11$ ciclos de reloj de transición entre estados, dando lugar a 137 ciclos de reloj para calcular el resultado como

puede observarse en la figura 3.18

3.3. Análisis de resultados de la función sigmoide

En esta sección se van a analizar aspectos relacionados con la latencia, el throughput, la utilización de recursos y la frecuencia de operación del módulo que implementa la función sigmoide. Al igual que en el multiplicador configurable, los resultados de frecuencia y utilización se han obtenido para la placa Nexys 4 DDR de Digilent basada en la FPGA XC7A100T-1CSG324C Artix-7 de Xilinx [2].

3.3.1. Función sigmoide: Latencia y throughput

Como se ha visto en la simulación realizada en la subsección 3.2.6, la latencia de la función sigmoide cambia en función del intervalo en el que se encuentre el dato de entrada y depende, principalmente, de la latencia del multiplicador configurable, que como se vio anteriormente, varía en función de la precisión. De esta forma, si el dato de entrada se encuentra en los intervalos $(-\infty, -1,5]$ y $[1,5, \infty)$, la latencia es igual a $(m + n) \times 5 + 9$ ciclos de reloj, y si el dato de entrada se encuentra en el intervalo $(-1,5, 1,5)$, la latencia es igual a $(m + n) \times 6 + 11$ ciclos de reloj, siendo m y n los bits de los operandos que se introducen en el multiplicador configurable.

Por otro lado, debido a que el cálculo de valores de la función sigmoide se realiza de forma secuencial, el throughput, medido en operaciones/ciclo, es en cada caso la inversa de la latencia.

3.3.2. Función sigmoide: Frecuencia

Para conocer la frecuencia máxima a la que el módulo que implementa la función sigmoide es capaz de trabajar, se ha implementado una pequeña arquitectura de prueba con dos registros de desplazamiento y un módulo MMCM para generar la señal de reloj con la frecuencia deseada a partir de una señal de reloj con una frecuencia de 100 MHz. Probando con diferentes frecuencias de reloj, se ha llegado a la conclusión de que la frecuencia más alta a la que puede trabajar dicho módulo es a 175 MHz.

Design Timing Summary		
Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 0,122 ns	Worst Hold Slack (WHS): 0,064 ns	Worst Pulse Width Slack (WPWS): 1,877 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: 278	Total Number of Endpoints: 278	Total Number of Endpoints: 209

All user specified timing constraints are met.

Figura 3.19: Función sigmoide - Análisis de tiempo

3.3.3. Función sigmoide: Utilización de recursos

Los resultados con respecto a la utilización de recursos del módulo que implementa la función sigmoide se muestran en las siguientes capturas. En ellas se puede ver que la utilización de LUT y FF es mayor, como es de esperar, que en el caso del multiplicador configurable, pero aún así sigue siendo muy pequeña (< 1 % en ambos casos).

Name	^ 1	Slice LUTs (63400)	Slice Registers (126800)	Slice (1585 0)	LUT as Logic (63400)	LUT as Memory (19000)	LUT Flip Flop Pairs (63400)	Bonded IOB (210)	BUFGCTRL (32)	MMCME2_ADV (6)
top	255	202	86	254	1	133	9	2	1	
SIG (sigmoid)	237	160	85	236	1	108	0	0	0	
CONFIG_MULTIPLI...	67	111	45	66	1	50	0	0	0	0
CONTROLLER_SIG...	80	11	45	80	0	3	0	0	0	0
REG_1 (registro)	1	13	5	1	0	0	0	0	0	0
SIGMOID_LEFT (sig...	25	4	20	25	0	4	0	0	0	0
SIGMOID_MIDDLE (...)	40	17	29	40	0	4	0	0	0	0
SIGMOID_RIGHT (s...	24	4	22	24	0	4	0	0	0	0
SIN (shifter_in)	5	29	16	5	0	0	0	0	0	0
SOUT (shifter_out)	13	13	9	13	0	13	0	0	0	0
Uclk_mod (clk_mod)	0	0	0	0	0	0	0	2	1	

(a) Análisis detallado de utilización de recursos

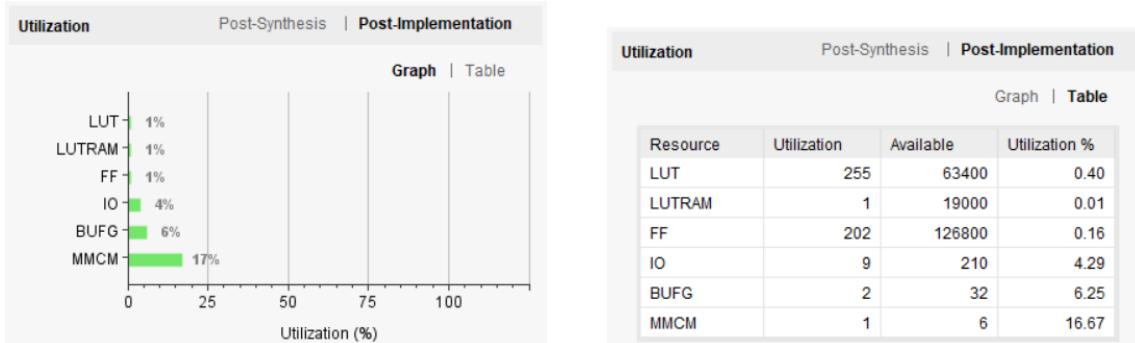


Figura 3.20: Función sigmoide - Análisis de utilización de recursos

Capítulo 4

Primera aproximación hacia una neurona artificial configurable

En este capítulo se va a describir la estructura de una neurona artificial configurable entre precisión y latencia utilizando los módulos desarrollados anteriormente. Debido a la falta de tiempo, se deja como continuación de este Trabajo de Fin de Grado tanto su implementación en VHDL como en hardware.

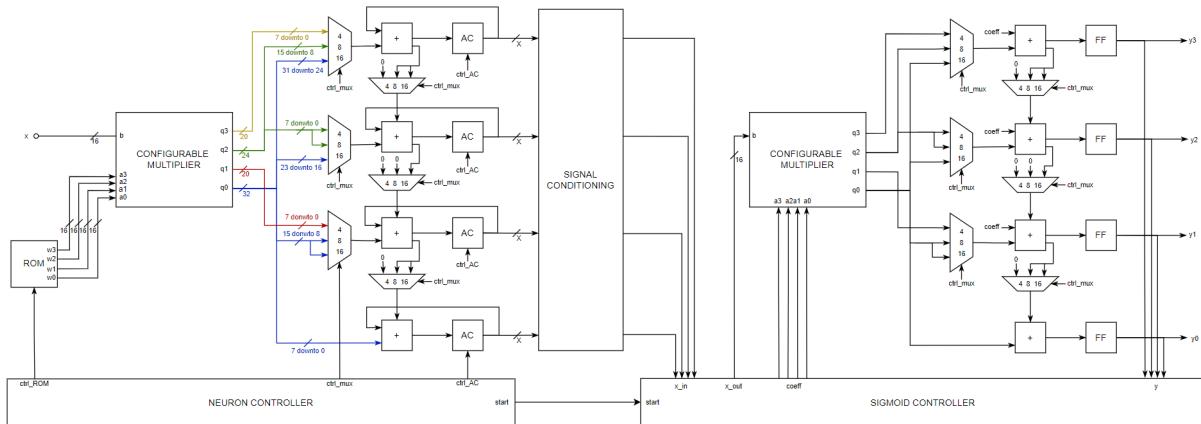


Figura 4.1: Neurona artificial configurable

El esquema que se muestra en la figura 4.1 representa la arquitectura de una neurona artificial configurable de la cual se pueden obtener 1, 2 o 4 neuronas independientes con diferentes precisiones y latencias. Los elementos que componen esta neurona artificial configurable son los siguientes:

Multiplicador configurable

En primer lugar, tenemos un multiplicador configurable como el descrito en 2 que recibe, en su entrada paralelo, un dato x de 16 bits, el cual puede estar compuesto por 4 palabras de 4 bits, 2 palabras de 8 bits, o una palabra de 16 bits, y en sus entradas serie ($a_3 - a_0$), recibe un conjunto de pesos ($w_3 - w_0$) proporcionados por una memoria ROM que tienen el mismo tamaño que los datos de entrada (4, 8 o 16 bits). A la hora de sintetizar la neurona hay que tener en cuenta la longitud de los datos que permite cubrir todos los casos, es por este motivo que los pesos se codifican con una longitud de 16 bits. Sin embargo, de estos 16 bits, solo habrá que tener en cuenta los bits relevantes en cada caso al realizar la multiplicación con los datos de entrada.

Como se vio anteriormente, este multiplicador configurable proporciona 3 tipos de configuraciones. Si se configura el multiplicador configurable como 4 multiplicadores de 4 bits, se obtienen 4 neuronas independientes. Para esta configuración, el dato de entrada x se divide en 4 palabras de 4 bits, y cada una de estas palabras es multiplicada por uno de

los cuatro pesos que se introducen en las entradas serie, dando lugar a 4 datos de salida de $4 + 16 = 20$ bits. Sin embargo, para esta configuración, de los 16 bits que componen los pesos, solo los 4 primeros bits son relevantes. Por tanto, de los 20 bits resultantes, sólo es necesario tener en cuenta los 8 bits menos significativos para el proceso de suma. Este proceso de selección es realizado por los multiplexores que hay a la salida del multiplicador configurable.

Si el multiplicador configurable se configura como 2 multiplicadores de 8 bits, se obtienen 2 neuronas independientes. En esta configuración, el dato de entrada x se divide en 2 datos de 8 bits, y cada uno de ellos es multiplicado por uno de los dos pesos proporcionados por las parejas de entradas $a3 - a2$ y $a1 - a0$, dando lugar a 2 datos de salida de $8 + 16 = 24$ bits que se proporcionan en las salidas $q2$ y $q0$. Para esta configuración, de los 16 bits que forman los pesos, solo los 8 primeros son los que contienen la información relevante. Por tanto, de los 24 bits resultantes, solo hay que tener en cuenta los 16 menos significativos.

Por último, si se configura el multiplicador configurable como un multiplicador de 16 bits, se obtiene 1 neurona, pero con una precisión mayor. En este caso, el dato x de 16 bits se multiplica por un único peso proporcionado por el conjunto de las 4 entradas $a3 - a0$, dando lugar a un único dato de salida de $16 + 16 = 32$ bits que se proporciona en la salida $q0$. En este caso se tienen en cuenta los 32 bits del resultado.

La latencia que presenta este multiplicador configurable para las tres configuraciones depende de la precisión que se quiera obtener y es igual a $m + n$ ciclos de reloj.

Sumadores, acumuladores y acondicionador de señal

A la salida del multiplicador y de los multiplexores tenemos 4 filas compuestas por un sumador de 8 bits y un acumulador, los cuales se encargan de realizar la suma de los productos. Además, entre cada uno de estos sumadores se colocan una serie de multiplexores que sirven para interconectar 2 sumadores adyacentes, de forma que, en función de la señal de control de los multiplexores, se pueden obtener 4 sumadores de 8 bits que realizan la suma de los 4 productos resultantes de 8 bits en la configuración de 4 multiplicadores, 2 sumadores de 16 bits que realizan la suma de los 2 productos resultantes de 16 bits en la configuración de 2 multiplicadores, o un sumador de 32 bits que realiza la suma del único producto resultante de 32 bits en la configuración de un multiplicador. Los resultados que se obtienen en cada una de estas filas de sumadores y acumuladores son resultados que tienen una longitud variable (X). Por tanto, es necesario añadir un bloque de acondicionamiento de señal que proporcione a las etapas posteriores, en este caso a la función de activación, los datos con las longitudes adecuadas.

Función sigmoide

El último elemento de esta neurona artificial configurable es un módulo que implementa una función sigmoide configurable como la descrita en [3.2](#). Este módulo se encarga de recibir los datos procedentes del bloque de acondicionamiento y proporcionar las salidas de las neuronas.

Estos datos procedentes del bloque de acondicionamiento, que podrán ser 1, 2 o 4 datos de 16, 8 o 4 bits respectivamente dependiendo de la configuración de la neurona, son enviados al controlador de la función sigmoide, el cual se encarga de comprobar a qué intervalo pertenece cada dato y, en función de dicha comprobación, asignar los coeficientes de los polinomios de la serie de Taylor correspondientes al realizar las operaciones de multiplicación y suma. El controlador de la sigmoide también es el encargado de proporcionar los datos en paralelo al multiplicador configurable a partir de los datos de entrada

procedentes del acondicionador de señal o a partir de los datos de salida de los registros, además de otras señales de control necesarias.

El multiplicador configurable de la función sigmoide realiza la multiplicación entre los datos proporcionados por el controlador y los coeficientes de una forma similar a la del multiplicador configurable del principio. El multiplicador recibe un dato de 16 bits resultado de la concatenación de los datos proporcionados por el bloque de acondicionamiento de señal, y según la configuración de este, toma estos 16 bits como un único dato de entrada o divide el dato de entrada en 4 datos de 4 bits o en 2 datos de 8 bits. De esta forma, dependiendo de la configuración del multiplicador configurable, se pueden obtener 1, 2 o 4 funciones sigmoide que actúan como funciones de activación de las neuronas resultantes.

Posteriormente, se realizan las operaciones de suma entre los diferentes coeficientes y los resultados de las multiplicaciones. Para las operaciones de suma, se puede utilizar una estructura de sumadores en cadena como la descrita anteriormente con el objetivo de reducir la utilización de recursos y el área. Una cuantificación posible para los coeficientes de los polinomios de la serie de Taylor puede ser la que se comenta en [3.2.5](#). Sin embargo, el detalle de la cuantificación de las señales internas de este sistema se deja como una posible línea de estudio futura.

La latencia de esta función sigmoide cambia en función del intervalo en el que se encuentren los datos de entrada y depende principalmente de la latencia que presenta el multiplicador configurable, el cual varía con la precisión.

Controlador de la neurona artificial

Además de todos estos bloques descritos, la neurona cuenta con un controlador que se encarga de coordinar los distintos elementos que componen la neurona artificial, proporcionándoles las señales de control necesarias en los momentos precisos. Una forma de implementar este controlador podría ser utilizando una metodología FSMD como la que se ha visto en los capítulos anteriores.

Con todo esto, podemos concluir que el elemento clave que proporciona la configurabilidad entre precisión y latencia a esta neurona artificial es el multiplicador configurable, el cual permite obtener 1, 2 o 4 neuronas independientes con precisiones de 4, 8 o 16 bits respectivamente y con unas latencias que vienen dadas por la suma de la latencia del multiplicador configurable, que realiza la multiplicación entre los datos de entrada y los pesos, y la latencia del módulo que implementa la función sigmoide.

Capítulo 5

Conclusiones

En este Trabajo de Fin de Grado se ha realizado el diseño y la implementación en VHDL de las partes principales que formarían una neurona artificial enfocada hacia la configurabilidad entre precisión y latencia, y se ha descrito, en líneas generales, la arquitectura que esta tendría.

En primer lugar, se ha desarrollado un multiplicador serie-paralelo configurable basado en los algoritmos de Braun y Baugh-Wooley. Se ha visto que este multiplicador se puede configurar mediante dos comandos de forma que puede realizar multiplicaciones tanto en binario como en Ca2. Otra característica interesante de este multiplicador es que se puede variar su precisión separando o uniendo los 4 multiplicadores de 4 bits que lo componen, pudiendo disponer así de 4 multiplicadores con una precisión de 4 bits, de 2 multiplicadores de 8 bits, o de un multiplicador de 16 bits. También se ha comprobado que esta variabilidad en la precisión afecta a la latencia y al throughput, dando lugar a una mayor latencia y a un menor throughput a mayores niveles de precisión. En concreto, se ha visto que para las tres configuraciones la latencia es igual a $m + n$ ciclos de reloj y que el throughput es la inversa de la latencia, es decir, $1/(m + n)$ operaciones/ciclo, siendo m la longitud del operando que se introduce en la entrada serie y que puede ser un número cualquiera, y n la longitud del operando que se introduce en la entrada paralelo y que puede ser igual a 4, 8 o 16 bits dependiendo de la configuración. Además, de este multiplicador configurable se han realizado pruebas con respecto a la frecuencia y a la utilización de recursos y se ha comprobado que la frecuencia máxima a la que puede trabajar es a 330 MHz y que su utilización de LUT y FF es inferior al 1 %. Estas pruebas de frecuencia y utilización se han realizado para la placa Nexys 4 DDR de Digilent basada en la FPGA XC7A100T-1CSG324C Artix-7 de Xilinx.

Por otro lado, se ha desarrollado un módulo que implementa el algoritmo de una función sigmoide. Para ello, se ha seguido un proceso de diseño basado en síntesis de alto nivel (HLS) a partir de la expansión en serie de Taylor por intervalos de la función sigmoide. Uno de los objetivos de esta parte era hacer que dicho módulo fuese configurable. Con este fin, se ha integrado el multiplicador configurable desarrollado anteriormente que, en conjunto con un controlador, una estructura de multiplexores y registros, dotan al sistema de dicha propiedad de configurabilidad, pudiendo llegar a obtener 1, 2 o 4 funciones sigmoide con diferentes precisiones y latencias. De este módulo que implementa la función sigmoide se ha visto que su latencia se ve principalmente afectada por la latencia del multiplicador configurable y que esta varía en función del intervalo en el que se encuentren los datos de entrada, dando lugar a una latencia de $(m+n) \times 5 + 9$ ciclos de reloj cuando los datos de entrada se encuentran en los intervalos $(-\infty, -1,5]$ y $[1,5, \infty)$, y a una latencia de $(m + n) \times 6 + 11$ ciclos de reloj cuando los datos de entrada pertenecen al intervalo $(-1,5, 1,5)$. El throughput en cualquiera de estos casos es igual a la inversa de la latencia. Adicionalmente, de este módulo se han realizado pruebas con respecto a la frecuencia y al consumo de recursos y se ha determinado que la frecuencia máxima de trabajo es de 175 MHz y que la utilización de LUT y FF es inferior al 1 %. Estas pruebas, al igual que

en el multiplicador configurable, se han realizado para la placa Nexys 4 DDR de Digilent basada en la FPGA XC7A100T-1CSG324C Artix-7 de Xilinx.

Finalmente, debido a la falta de tiempo, se ha realizado un primer planteamiento de la estructura que tendría una neurona configurable compuesta por los elementos desarrollados anteriormente y se ha dejado como una línea de trabajo futura su implementación tanto en VHDL como en hardware (FPGA).

Como conclusión personal, realizar este Trabajo de Fin de Grado me ha resultado muy satisfactorio, ya que me ha servido para profundizar en un tema muy interesante como es la inteligencia artificial, y más en concreto, en el aprendizaje automático y en los algoritmos que se desarrollan dentro de esta disciplina, y también me ha servido para desarrollar habilidades de diseño electrónico a nivel RTL y algorítmico y mejorar mis habilidades de programación en VHDL.

Bibliografía

- [1] Amos R Omondi and Jagath Chandana Rajapakse. *FPGA implementations of neural networks*, volume 365. Springer, 2006.
- [2] Digilent. Nexys 4 DDR Reference Manual. <https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/reference-manual>.
- [3] Andrew Ng's. Machine Learning Course. <https://www.coursera.org/learn/machine-learning>.
- [4] Amine Bermak, D Martinez, and J-L Noullet. High-density 16/8/4-bit configurable multiplier. *IEE Proceedings-Circuits, Devices and Systems*, 144(5):272–276, 1997.
- [5] Onursal Cetin, Feyzullah Temurtas, and Senol Gulgonul. An application of multi-layer neural network on hepatitis disease diagnosis using approximations of sigmoid activation function. *Dicle Medical Journal / Dicle Tip Dergisi*, 42, 06 2015.
- [6] José Abiel Caballero Hernández, Martha Díaz Salazar, Meyli Moradillos Paz-Lago, and Sonnia Pavoni Oliver. Implementación de la función sigmoidal logarítmica en un fpga. *Ingeniería Electrónica, Automática y Comunicaciones*, 35(2):35–44, 2014.
- [7] Jianli Feng and Shengnan Lu. Performance analysis of various activation functions in artificial neural networks. *Journal of Physics: Conference Series*, 1237:022030, 06 2019.

Apéndice A

Aspectos éticos, económicos, sociales y ambientales

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.

El diseño se realiza sobre una placa FPGA, que aporta 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.

	Aspectos éticos y profesionales	Aspectos económicos	Aspectos sociales	Aspectos ambientales
Materia prima y recursos naturales		No se aprecian impactos significativos		
Diseño, producción y pruebas	Pérdida de privacidad	No se aprecian impactos significativos		Gasto energético
Empaque y distribución		No se aprecian impactos significativos		
Uso y mantenimiento	Pérdida de privacidad	Possible destrucción de ciertos empleos	Salud	Gasto energético
Reutilización, reciclaje y desecho		No se aprecian impactos significativos		Material reconfigurable

Figura A.1: 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.

El sistema diseñado en este trabajo está pensado para 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.3. 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.

Apéndice B

Presupuesto económico

Este proyecto lo he realizado en aproximadamente 318 horas. 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 1.000€ y he contado con una 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 la siguiente:

COSTE DE MANO DE OBRA (coste directo)		Horas	Precio/hora	Total
		318	15 €	4.770 €
COSTE DE RECURSOS MATERIALES (coste directo)		Precio de compra	Uso en meses	Amortización (en años)
Ordenador personal (Software incluido)		1.000,00 €	6	5
Nexys 4 DDR		230,00 €	6	5
Licencia Vivado		-		
COSTE TOTAL DE RECURSOS MATERIALES				123,00 €
GASTOS GENERALES (costes indirectos)		15%	sobre CD	
BENEFICIO INDUSTRIAL		6%	sobre CD+CI	
MATERIAL FUNGIBLE				
Impresión				50,00 €
SUBTOTAL PRESUPUESTO				6.014,57 €
IVA APPLICABLE			21%	1.263,06 €
TOTAL PRESUPUESTO				7.277,63 €

Figura B.1: Presupuesto económico del proyecto