Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Electrónica y Automática Industrial

Trabajo Fin de Grado

Diseño de una arquitectura basada en FPGAs para la implementación eficiente de redes neuronales LSTM

ESCUELA POLITECNICA SUPERIOR

Autor: Miguel Cubero Vacas

Tutor: Álvaro Hernández Alonso

2022

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería en Electrónica y Automática Industrial

Trabajo Fin de Grado

Diseño de una arquitectura basada en FPGAs para la implementación eficiente de redes neuronales LSTM

Autor: Miguel Cubero Vacas

Tutor: Álvaro Hernández Alonso

Tribunal:

Presidente: Raúl Mateos Gil

Vocal 1º: Ángel Llamazares Llamazares

Vocal 2º: Álvaro Hernández Alonso

Fecha de depósito: 29 de junio de 2022

Dedicado a mis padres y a mi hermana, por ser el impulso que me ha permitido lograr mis metas y ser la persona que soy hoy...

"Lo que cuenta no es de dónde vienes, sino hacia dónde vas."
Ella Fitzgerald

Agradecimientos

Al expresar nuestra gratitud, nunca debemos olvidar que el mayor aprecio no es pronunciar palabras, sino vivir de acuerdo con ellas.

John F. Kennedy

Este trabajo es el resultado de un largo, duro, pero enriquecedor camino educativo, en el que no solo he podido adquirir innumerables conocimientos técnicos, sino que me ha permitido superarme continuamente y, lo que es más importante, crecer como persona. Sin embargo, no habría sido posible superar todas las adversidades del mismo sin el apoyo de varias personas. Es por ello, que los aportes científicos que realiza este trabajo no son solo míos, sino que pertenecen también a todas las personas que me han apoyado durante mi paso por la universidad y, especialmente, a las que me han apoyado durante el desarrollo del trabajo.

En primer lugar, me gustaría expresar mi gran agradecimiento a Álvaro, mi tutor, por aportarme sus conocimientos y por ayudar a mi orientación hacia un futuro profesional en el ámbito de la electrónica. La existencia de docentes como él, con gran vocación por la enseñanza y por la formación de profesionales cualificados que, además, realizan importantes aportes a la ciencia es algo altamente enriquecedor y valorable.

Por otro lado, tengo que darles las gracias a mis amigos, porque gracias a los innumerables buenos momentos que hemos pasado juntos durante estos cuatro años, los cuales me han permitido desconectar y eliminar el estrés, he podido seguir adelante en cada instante. Me gustaría hacer mención especial a Jesús Ángel, quien, desde antes de entrar a la univesidad, hasta el final de la carrera, se ha ofrecido para orientarme y ayudarme en todo lo que he necesitado.

Por último, el mayor de los agradecimientos debo dedicárselo a mis padres y a mi hermana. Gracias a su apoyo incondicional, a su confianza en mí, y a su cariño, he conseguido llegar hasta aquí con una formación de gran calidad y, por supuesto, he logrado crecer como persona. Por todo ello, finalizo reconociendo con mucha emoción, que mis logros siempre serán sus logros.

Resumen

El objetivo de este Trabajo Fin de Grado es el diseño de una arquitectura hardware basada en FPGA para la implementación de redes neuronales LSTM. Para conseguir este propósito, el proyecto comienza con el modelado y dimensionamiento de una celda LSTM en coma fija en Matlab. Posteriormente, se lleva a cabo el diseño de la arquitectura hardware equivalente, combinando la codificación de circuitos digitales en lenguaje VHDL con el uso de circuitos disponibles en la herramienta Vivado. Esta arquitectura es validada con ayuda de la misma herramienta. Finalmente, se realiza el diseño de una capa completa, cuya funcionalidad es validada con una reducida cantidad de unidades neuronales.

Palabras clave: FPGA, inteligencia artificial, redes neuronales, LSTM.

Abstract

This Final Degree Project aims to design a FPGA-based hardware architecture for the implementation of LSTM neural networks. To overcome this purpose, this project begins with the modeling and dimensioning of a fixed-point LSTM cell in Matlab. Subsequently, the design of the equivalent hardware architecture is carried out, combining digital circuits coding in VHDL language with the use of available cores in Vivado tool. This architecture is validated with the help of the same tool. Finally, the design of a complete layer is performed, whose functionality is validated with a reduced number of neural units.

Keywords: FPGA, artificial intelligence, neural networks, LSTM.

Resumen extendido

Los descubrimientos en el campo de la inteligencia artificial han provocado desde hace varias décadas una revolución tecnológica que continúa teniendo una gran importancia en la actualidad, debido a que una inmensa mayoría de las tecnologías utilizadas implementan múltiples funcionalidades gracias a estos. Estos avances mejoran la funcionalidad de todo tipo de tecnologías, desde dispositivos utilizados en el ámbito personal como los teléfonos móviles actuales hasta instrumentos destinados a sectores profesionales como el industrial o el militar, entre muchos otros. Uno de los grandes campos que actualmente continua en auge dentro de la inteligencia artificial es el de las redes neuronales artificiales.

La implementación de las redes neuronales se realiza de forma típica en computadores y, en general, en dispositivos que ejecuten aplicaciones software, puesto que la mayoría de modelos neuronales realizan las operaciones de forma secuencial y ordenada. Sin embargo, en los últimos años ha aumentado el número de investigaciones relacionadas con la implementación de redes neuronales en circuitos electrónicos debido a que, aunque las unidades neuronales operan secuencialmente, el conjunto de estas en una misma capa realiza las operaciones de forma paralela, por lo que se pueden conseguir mejoras de rendimiento muy notables. En este contexto, teniendo en cuenta la evolución de los circuitos digitales hacia la lógica reconfigurable, surge el planteamiento de realizar implementaciones de redes neuronales en dispositivos FPGA, con el objetivo de utilizar todos los beneficios que aportan las arquitecturas basadas en esta tecnología.

Entonces, este proyecto, enmarcado en el campo de la integración de redes neuronales en sistemas digitales configurables, presenta como principal pretensión el diseño de una arquitectura digital basada en FPGA para redes neuronales LSTM. Para ello, en primer lugar se desarrolla un modelo software de una celda LSTM en notación de punto fijo análogo al que se desea implementar en la FPGA. Durante este proceso, se dimensiona la celda para minimizar el impacto de los errores en los redondeos. Posteriormente, se realiza el diseño hardware aplicando distintas técnicas de diseño y se valida teniendo en cuenta las características de la tecnología destino. El diseño de los circuitos digitales finaliza con la integración de las celdas en un modelo de una capa completa y su posterior validación de funcionalidad con un pequeño número de neuronas.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Contexto e interés

Los avances tecnológicos en la actualidad tienen múltiples campos de aplicación, aunque uno de los objetivos más perseguidos en las últimas décadas es el de la automatización de tareas cotidianas. En este marco se desarrollan continuamente proyectos que varían desde un sencillo sistema de riego automático para jardines hasta trabajos en el ambicioso sector de la conducción autónoma.

Como podría esperarse, el objetivo final de estos trabajos es el de conseguir desarrollar sistemas que puedan tomar decisiones de la misma forma en que lo haría un humano. Los sistemas de computación tradicionales se crearon con la premisa de realizar las tareas que los humanos no pueden llevar a cabo con rapidez, como el cálculo numérico. Sin embargo, en las aplicaciones que el cerebro humano realiza de forma eficiente, como el reconocimiento de patrones o la percepción los computadores no ofrecen un rendimiento ideal, puesto que no se idearon especialmente para ello.

En este contexto surge la idea de obtener un modelo computacional que imite la estructura del cerebro humano, especialmente de las partes encargadas del aprendizaje y de la toma de decisiones. Estos modelos computacionales se denominan Redes Neuronales Artificiales (ANN).

Los avances en el campo de las ANN en los últimos años han sido muy notables, habiéndose creado múltiples tipos de ellas, los cuales han ofrecido resultados óptimos en una gran variedad de aplicaciones. Las Redes Neuronales Recurrentes (RNN) son un tipo de ANN, el cual, gracias a la realimentación entre neuronas de una misma capa, posee una capacidad de memoria a corto plazo. Como solución a los problemas que presentan dependencias de memoria a largo plazo, como son el reconocimiento de voz o la predicción de texto, surgió un tipo especial de RNN : la red Long Short-Term Memory (LSTM).

Un aspecto fundamental de las ANN es que la eficiencia de estas no radica exclusivamente en lo bien diseñado que esté el modelo, si no que la tecnología en la que se implementen posee una especial importancia. Los primeros elementos en los que se implementaban las ANN eran computadores tradicionales.

Los procesadores son elementos que procesan instrucciones de forma secuencial, pero las neuronas biológicas operan de forma paralela, por lo que en esta situación aparece un cuello de botella que no permite un aprovechamiento óptimo de las ANN. Para solucionar este problema, se comenzaron a optimizar ANN para su funcionamiento en Unidades de Procesamiento Gráfico (GPUs). Las GPUs son elementos que poseen múltiples unidades de procesamiento que operan en paralelo, por lo que, en el ámbito de las ANN, obtienen rendimientos mucho mayores que los microprocesadores tradicionales, ya que se adecuan mucho mejor al funcionamiento paralelo de las redes neuronales.

Sin embargo, las GPUs son dispositivos con una gran eficiencia computacional, pero con un consumo de potencia bastante elevado, lo que limita su utilización en aplicaciones que requieren un bajo consumo energético. Para solucionar este problema, se debía encontrar un dispositivo que pudiera aprovechar el paralelismo de las ANN para ofrecer un buen rendimiento en la ejecución de las mismas, al mismo tiempo que presentase un bajo consumo de energía, imprescindible en muchas aplicaciones. Una de las soluciones que actualmente están a la vanguardia son los dispositivos lógicos reconfigurables, en especial las Field-Programmable Gate Arrays (FPGAs). Estos dispositivos presentan un consumo de potencia significativamente menor que las GPUs y una gran eficiencia en la implementación de ANN, puesto que al basarse en circuitos lógicos, es decir, en hardware digital, operan de manera paralela. Por ello, las FPGAs resultan ser en muchas aplicaciones una solución de compromiso entre los procesadores tradicionales y las GPUs, presentando una mejor capacidad que los procesadores para ejecutar redes neuronales consumo de potencia y un menor consumo energético que las GPUs.

Este trabajo utiliza como base los conocimientos adquiridos en las diferentes asignaturas de diseño de circuitos electrónicos digitales del Grado en Ingeniería en Electrónica y Automática Industrial, en especial las asignaturas de Electrónica Digital y Diseño Electrónico. La primera de ellas muestra al alumno los fundamentos del análisis y diseño de circuitos electrónicos digitales, mientras que la segunda permite la introducción en el diseño de estos circuitos aplicado a FPGAs, profundizando en técnicas de diseño y en las consideraciones que deben tenerse en cuenta.

1.2 Objetivos del trabajo

El objetivo principal del proyecto es la definición de una arquitectura hardware para la implementación en tiempo real de redes neuronales LSTM. Para llevarlo a cabo, se diseñará una arquitectura basada en FPGA mediante la codificación de los circuitos lógicos correspondientes en Very High Speed Integrated Circuit Hardware Description Language (VHDL) y su posterior integración en la herramienta de desarrollo Vivado junto a elementos de Propiedad Intelectual (IP) optimizados para su implementación en FPGAs. Se partirá de la definición de una celda LSTM y su posterior validación, para abordar luego el diseño de una capa completa. Para ello, primero se realizará un modelo en software mediante la utilización de el lenguaje de programación de alto nivel Matlab y se simulará con las herramientas disponibles en el mismo. Posteriormente, se desarrollará el código necesario para implementarlo en la FPGA a utilizar y se simulará mediante las simulaciones de Vivado para validarlo.

La motivación fundamental para el desarrollo de este trabajo es la creación una arquitectura que pueda ser utilizada y mejorada en trabajos futuros, permitiendo la implementación de redes LSTM en FPGAs en diversas aplicaciones.

1.3 Estructura del documento

En las lineas posteriores se describe la organización presente en la memoria:

- **Capítulo 1. Introducción.** Este capítulo ha permitido al lector conocer el contexto en el que se enmarca este trabajo, así como los objetivos que tiene el mismo, finalizando con una breve descripción de la estructura que sigue el documento.
- **Capítulo 2. Estado del arte.** Esta parte del documento expone el contexto histórico de las ANN y las FPGAs, destacando hitos y aplicaciones importantes. Cada una de las secciones del capítulo

finaliza con la explicación de los conceptos más importantes relacionados con ambos campos que han sido aplicados en el trabajo.

- **Capítulo 3. Arquitectura de la red.** El objetivo de este capítulo es la explicación detallada del desarrollo seguido para el dimensionamiento, diseño y validación funcional de los modelos de neurona y capa.
- **Capítulo 4. Resultados Experimentales.** El objetivo de esta parte del documento es presentar al lector los resultados obtenidos tras la implementación del modelo de la neurona, realizando un análisis de los resultados temporales y del consumo de recursos presentado por la arquitectura.
- **Capítulo 5. Conclusiones y trabajos futuros.** Finalmente, en esta última parte se presentan al lector las conclusiones que se han obtenido en el proyecto, así como las posibles líneas futuras en las que este puede ser de aplicación.
- **Capítulo 6. Presupuesto.** En este capítulo se expone el análisis desglosado de los costes que se deberían asumir por parte de la entidad interesada en la realización de este trabajo.

Capítulo 2

Estado del arte

2.1 Introducción

La principal pretensión de este capítulo es exponer al lector el contexto científico en el que se enmarca este proyecto en los dos grandes campos con los que guarda relación: las redes neuronales artificiales y las FPGAs. Además, se incluyen las definiciones de algunos conceptos fundamentales presentes en ambas disciplinas, presentando con especial detalle aquellos que poseen más relevancia en este trabajo, puesto que están presentes en gran medida en el desarrollo del mismo.

2.2 Redes neuronales artificiales

2.2.1 Contexto histórico

El inicio de los intentos de construcción de máquinas inteligentes se remonta a casi un siglo atrás, durante la Segunda Guerra Mundial [1]. Uno de los hitos más conocidos hace referencia al desarrollo de un ordenador analógico para la predicción de movimiento de objetivos en cañones antiaéreos y navales desarrollado por la Marina de Estados Unidos. Se trata del ordenador Mark 1, que actuaba como el cerebro del sistema automático de control de disparos Mark 37 [2]. La Figura 2.1 muestra una imagen del Mark 1, extraída de la documentación del mismo.

Durante la posguerra, comenzaron los desarrollos de sistemas encaminados a convertirse en máquinas inteligentes. Se inició la creación de dispositivos con la novedosa intención de basar su comportamiento en el razonamiento humano. Uno de los primeros desarrollos que presentaban un avance tecnológico basado en este principio fue la *Machina Speculatrix*. Este es el nombre con el que se dotó a las tortugas robóticas, Elmer y Elsie, desarrolladas por el neurocientífico W. Grey Walter y posteriormente construidas por el ingeniero W. J. Warren en el año 1951 [3]. Estas tortugas robóticas interactuaban con el entorno gracias a la información que recibían de dos sensores: una fotocélula rotativa para la vista y un interruptor sensible al tacto que permitía reaccionar a la máquina si encontraba obstáculos.

Este hito de la historia posee especial importancia porque representa una de los primeros intentos de emulación del cerebro humano, ya que cada uno de los robots utilizaba un tubo de vacío para imitar el comportamiento de dos neuronas conectadas entre sí. La Figura 2.2 muestra la imagen de uno de estos robots. De forma paralela, en 1951, Marvin Lee Minsky desarrolló SNARC [4], un ordenador analógico que aplicaba el concepto de redes neuronales. Este implementaba las interconexiones entre neuronas al

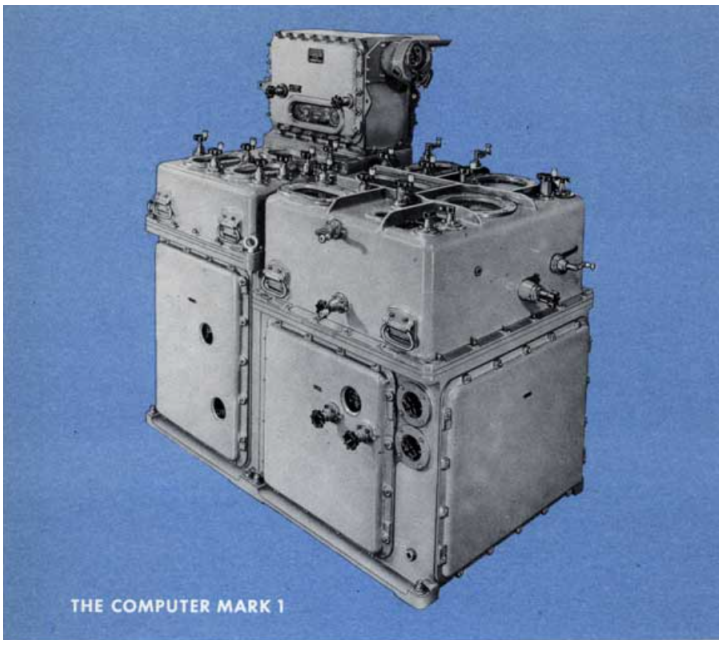


Figura 2.1: El ordenador Mark 1 [2].

igual que en el prototipo de Grey Walter, mediante válvulas de vacío. Este prototipo estaba formado por 40 unidades neuronales y un potenciómetro que controlaba la probabilidad de propagación de las señales.

Tras esas importantes invenciones, los intentos de desarrollo de aplicaciones de inteligencia artificial aumentaron de forma notable. Entre las décadas 50 y 60, se desarrolló el modelo del perceptrón, el primer modelo matemático de neurona artificial. A finales de los años 60, Minsky y Seymour Papert exponían en el libro *Perceptrons: An Introduction to Computational Geometry* [5] las fortalezas y, especialmente, las debilidades del perceptrón, que surgían debido a que solo podía modelar funciones linealmente separables, por lo que las funciones que no lo fueran no podrían ser implementadas a través de un perceptrón. Un ejemplo de función que no es linealmente separable es la función lógica XOR. Entonces, en ese

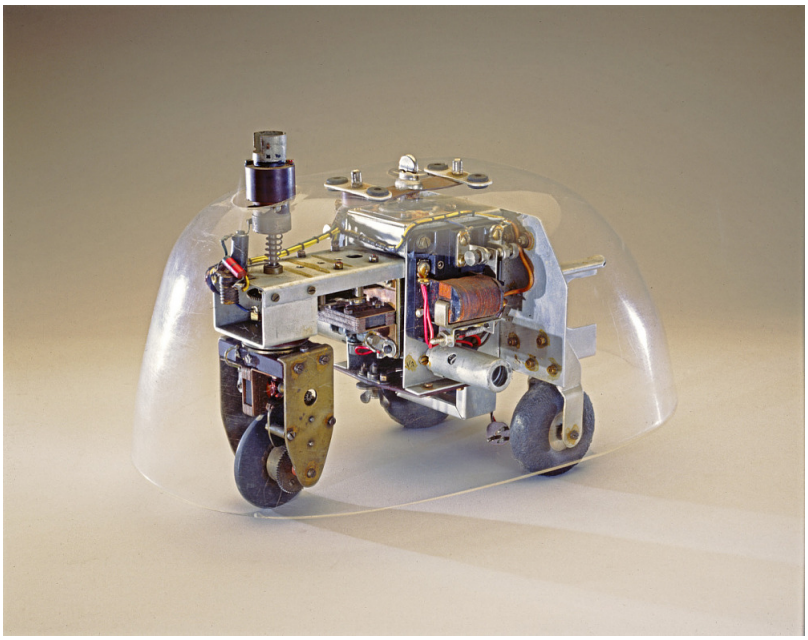


Figura 2.2: Una de las tortugas robóticas creadas por Grey Walter [3].

instante, surge la necesidad de crear nuevos modelos de redes neuronales que superasen las limitaciones del perceptrón simple y evolucionasen para la resolución eficiente de problemas mediante la aplicación de novedosas técnicas de machine learning. Estos son los desarrollos que se han llevado a cabo en el resto del siglo XX y en el siglo XXI, y que han dado lugar a los diversos conocimientos actuales sobre inteligencia artificial.

2.2.2 La neurona biológica

Para poder imitar las capacidades de razonamiento de los seres humanos era necesario estudiar las estructuras biológicas que otorgaban estas capacidades, en este caso, el cerebro y su composición neuronal. Resultaron especialmente importantes los descubrimientos del científico Santiago Ramón y Cajal, quien en el año 1888 pudo demostrar que el sistema nervioso humano estaba formado por células interconectadas entre ellas, las neuronas. Además, observó la forma en que la información viaja en las neuronas, desde las dendritas hacia el axón [1]. La Figura 2.3 muestra un diagrama de la estructura biológica de la neurona en el que puede apreciarse, además, el sentido del flujo de las señales nerviosas.

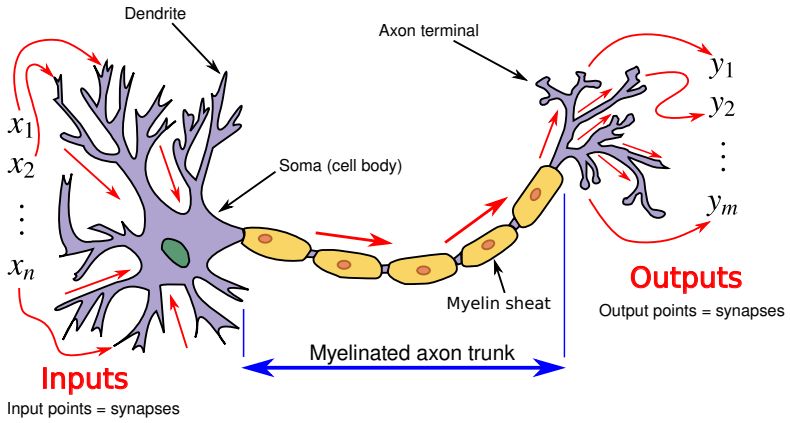


Figura 2.3: Estructura genérica de una neurona biológica. Las flechas representan el sentido y el camino de las señales que atraviesan las neuronas. Extraído de en.wikipedia.org. Biological neuron model.

Este modelo de neurona, que posee una serie de entradas (dendritas), un núcleo celular que genera un estímulo en base a las entradas (soma) y una serie de salidas (terminales del axón) es el que sirvió de base para construir el modelo genérico de una neurona artificial. Es importante también detallar el concepto de sinapsis, con el que se denomina a la unión entre dos neuronas. La intensidad de la sinapsis varía, de forma que interfiere en la manera en que la señal nerviosa llega a la neurona receptora.

2.2.3 El modelo genérico de la neurona artificial

Como se expuso en la sección 2.2.2, el modelo matemático de neurona artificial genérica se basa en el modelo biológico, por lo que puede notarse la analogía entre las partes de ambos modelos. El modelo matemático cuenta con un conjunto de entradas que representan las dendritas, con unos pesos sinápticos que representan la sinapsis entre neuronas, con un umbral que se resta al producto de las entradas por los pesos, siendo este último denominado regla de propagación; con una función de activación que proporciona una salida en función de la diferencia de la regla de propagación y del umbral, que funciona de forma análoga a como lo hace el soma en la neurona biológica y con una salida, análoga a los terminales del axón. Este se representa mediante la siguiente ecuación:

$$o_j(t) = \varphi_j(\sum_i w_{ij}x_i - \theta_j) \tag{2.1}$$

Donde φ_j representa la función de activación, x_i hace referencia a las entradas, w_{ij} son los respectivos pesos sinápticos, θ_j es el umbral y $o_j(t)$ es la salida de la neurona. Nótese que $\sum_i w_{ij}x_i$ hace referencia a la mencionada regla de propagación. Esta estructura se muestra de forma esquemática en la Figura 2.4.

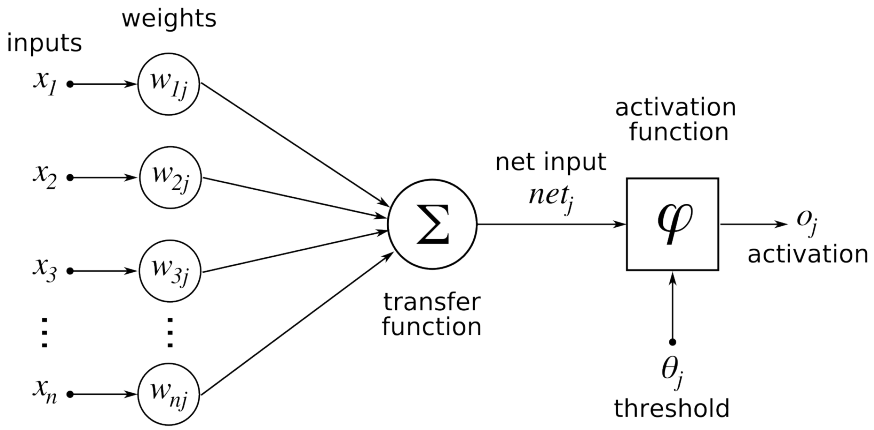


Figura 2.4: Modelo genérico de la neurona artificial. Extraído de commons.wikimedia.org. Artificial neural network

Las funciones de activación de una neurona pueden ser de distintos tipos. Los modelos lineales como el perceptrón suelen implementar la función escalón, que viene definida por la siguiente ecuación:

$$o_j = \begin{cases} 1 & \text{si } \sum w_{ij}x_i \geq \theta_j \\ 0 & \text{si } \sum w_{ij}x_i < \theta_j \end{cases} \tag{2.2}$$

Debido a las limitaciones de estos modelos, los cuales podían representar únicamente funciones linealmente separables, no siendo posible representar otras que no lo fueran como la función XOR, surgieron varias soluciones, entre ellas, el uso de funciones no lineales. Este es el caso de las funciones sigmoideas. Las dos funciones más utilizadas son la función sigmoide, cuya expresión viene dada por la ecuación (2.3) y la función tangente hiperbólica, cuya expresión se muestra en la ecuación (2.4). Nótese que los valores de salida de la función sigmoide se encuentran en el intervalo $[0, 1]$, mientras que los de la función tangente hiperbólica ocupan el intervalo $[-1, 1]$.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.4}$$

Nótese que, cuando se haga referencia a estas funciones en posteriores secciones del trabajo, $\sigma(x)$ hará referencia a la función sigmoide y $tanh(x)$ a la función tangente hiperbólica. Estas dos ecuaciones son de vital importancia en este trabajo puesto que, como se detallará en posteriores apartados, constituyen las funciones de activación del tipo de red a modelar. La Figura 2.5 muestra las representaciones gráficas de ambos tipos de funciones de activación.

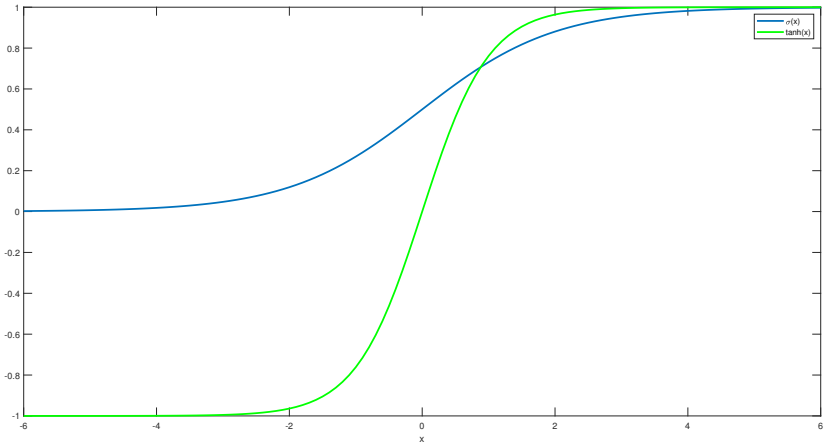


Figura 2.5: Funciones sigmoide y tangente hiperbólica.

2.2.4 Redes neuronales: Definiciones y tipos

Aunque como se ha comprobado, las unidades neuronales poseen la capacidad de representar funciones por sí solas, al igual que ocurre en los sistemas biológicos, el verdadero potencial surge cuando se realiza la interconexión entre ellas. Esta puede ser una definición adecuada para el concepto de red neuronal. Esto no representa otro concepto que un conjunto de nodos (neuronas) con una serie de conexiones establecidas entre ellos. Al igual que en la estructura biológica, a cada conexión se le asigna un peso sináptico que define la intensidad de la conexión entre dichas neuronas.

Las arquitecturas típicas de redes neuronales modelan un diseño basado en capas, en el que cada una de ellas cuenta con varias unidades neuronales. De forma genérica, en estos aparece una capa de entrada que es la encargada de recibir los estímulos del exterior, una o varias capas ocultas que simplemente procesan las señales que reciben y las envían a su salida, pero que no interactúan con su entorno y una capa de salida que proporciona las salidas de la red al entorno exterior. La Figura 2.6 muestra el modelo de una red neuronal unidireccional en la que pueden apreciarse los distintos niveles de las capas.

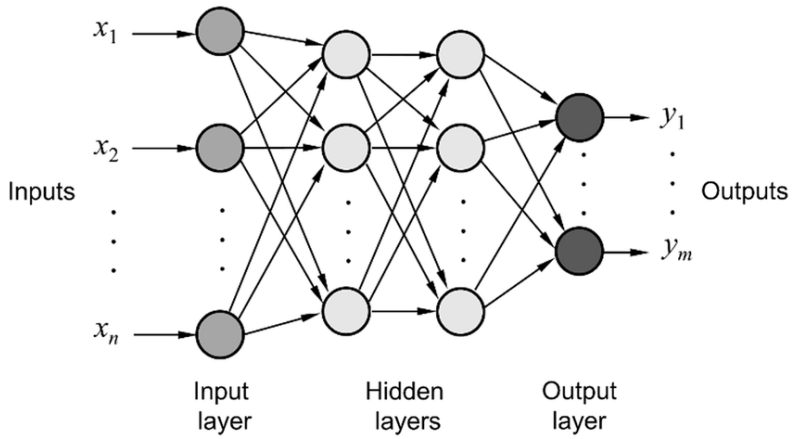


Figura 2.6: Estructura típica de una red neuronal unidireccional [6].

Las redes como la mostrada en la figura, que transportan la señal en un único sentido se denominan redes prealimentadas o *feedforward*. Sin embargo, las redes que tienen especial interés de estudio en el presente trabajo, puesto que son el tipo que pretende implementarse en la arquitectura, son las

redes neuronales que poseen realimentación entre unidades neuronales de una misma capa. Estas son las denominadas redes neuronales recurrentes o *feedback networks*.

2.2.5 Redes neuronales recurrentes

Las redes neuronales unidireccionales son útiles para una gran cantidad de aplicaciones. Este es el caso de aplicaciones típicas de clasificación, como el reconocimiento de los diferentes tipos de animales que aparecen en una imagen. Sin embargo, en los campos de aplicación en que se requiere conservar alguna información resultan muy poco eficientes, ya que deben realizarse redes muy extensas que tengan en cuenta todas las posibilidades. Este es el caso de la predicción automática de texto, como la que realiza el buscador de Google o la mayoría de smartphones actuales. En este caso, cuando se está escribiendo una palabra y se requiere la predicción de la última o de las últimas letras de la misma, es necesario tener en cuenta las letras que se han escrito anteriormente. Las redes neuronales recurrentes se crearon con el objetivo de dar solución a estos problemas. La Figura 2.7 muestra la estructura típica de una capa de una RNN.

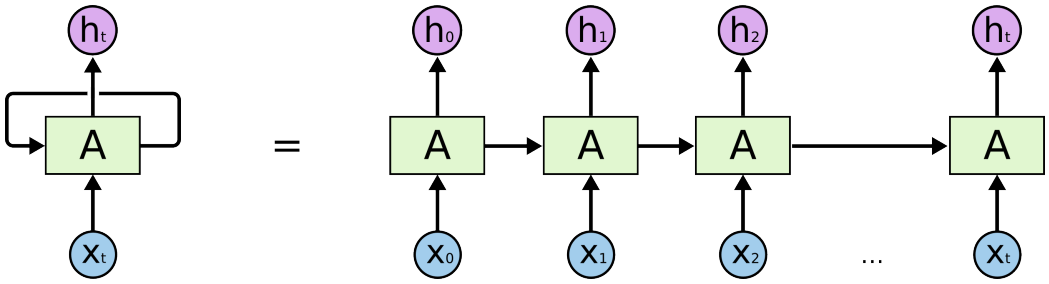


Figura 2.7: Capa genérica de una red RNN. Extraído de colah.github.io. Understanding LSTM Networks.

En la parte de la izquierda de la figura aparece la representación esquemática con el modelo de capa, el cual se realimenta con su salida. A la derecha se muestra la misma capa con sus neuronas internas, en la que puede apreciarse como cada neurona envía información a la siguiente unidad de la capa. Las neuronas RNN estándar poseen una función de activación, encargada de proporcionar la salida en función de la entrada que proviene de la capa anterior y de la salida de la neurona anterior de la misma capa. Las diferentes entradas representan una secuencia temporal de un número determinado de etapas. Por ejemplo, en una aplicación de predicción de texto como la mencionada anteriormente, si se desea realizar la predicción de la palabra "Hola" a partir de la secuencia "Hol", deberían introducirse en el orden siguiente y una por cada paso temporal la letra 'H' en la entrada x_0, la letra 'o' en la entrada x_1 y la letra 'l' en la entrada x_2.

2.2.6 Las dependencias a largo plazo. Redes LSTM

Como se menciona en la sección anterior, la estructura de una neurona RNN estándar está sencillamente compuesta por una función de activación que proporciona su resultado en función de la salida de la neurona anterior de la capa, además de su entrada, como en las redes *feedforward*. Aunque es cierto que este modelo resulta muy útil para solucionar problemas como el de la predicción de texto en palabras, se comprobó que su utilidad es limitada para aquellas aplicaciones que no poseen dependencias a largo plazo. Esta problemática aparece cuando los elementos a recordar pasan de ser un número muy reducido, como era el caso anterior, en el que solo se debían recordar unas pocas letras, a un número considerable. Este caso sería, partiendo del mismo ejemplo, la predicción de palabras completas, en lugar de letras

individuales. Un ejemplo que ilustra esto de forma clara, sería la predicción de la palabra "alemán" en la oración "Cualquier ciudadano de Alemania es reconocido como alemán.". En ella, para poder realizar correctamente la predicción, deben tenerse en cuenta la palabra "Alemania", que aparece detrás del conjunto de palabras "es reconocido como". Para poder abordar estas aplicaciones se diseñó una modificación de las RNN tradicionales: las redes LSTM. La Figura 2.8 muestra la estructura de una neurona LSTM.

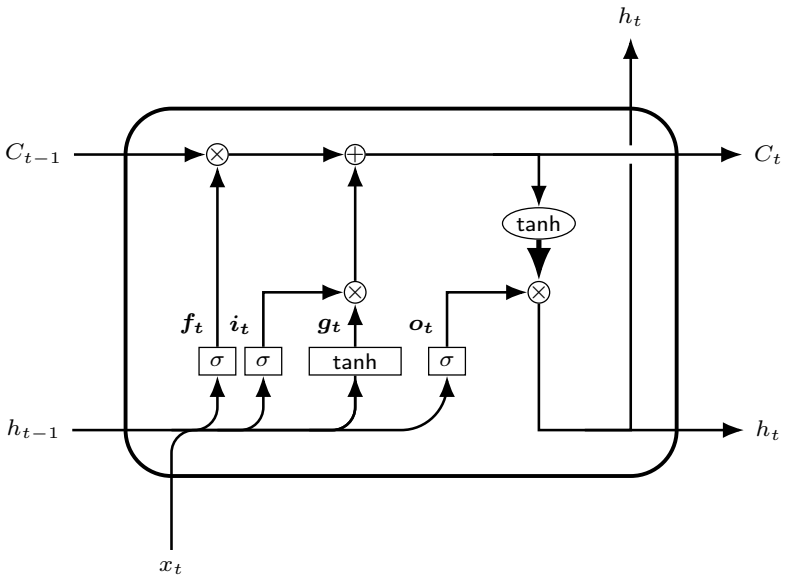


Figura 2.8: Estructura típica de una unidad neuronal LSTM.

Estas redes poseen un carril que actúa como memoria. La información que proviene de las entradas puede añadirse al contenido de esta memoria o puede olvidarse, si no resulta útil. Además, estas redes también pueden decidir qué información de la memoria va a entregarse a la salida de la neurona. En la Figura 2.8, el carril horizontal que aparece en la parte superior es el camino que representa la memoria. Este es denominado estado celular o, en inglés, *Cell State*. Como puede observarse, posee un multiplicador y un sumador, así como una salida a otro carril. El carril cuya salida es multiplicada por el contenido de la memoria que proporciona la neurona anterior es denominado *Forget Gate*, debido a que es el encargado de decidir qué cantidad de información de la memoria no es útil y, por tanto, debe borrarse. Por otro lado, los dos carriles centrales que convergen en un multiplicador cuya salida se suma al contenido de la memoria son, de izquierda a derecha, denominados *Input Gate* y *Cell Candidate*. Estos, en conjunto, son los encargados de decidir qué parte de la información que se introduce en las entradas debe añadirse al contenido de la memoria. Finalmente, el carril de la derecha que contiene el elemento o_t, junto al carril que posee como entrada la salida del estado celular y que contiene una función tangente hiperbólica y un multiplicador, forman en conjunto la denominada *Output Gate*. Esta puerta es la encargada de decidir qué información va a proporcionarse a la salida de la celda, basándose en la entrada actual y en el contenido de la memoria.

Debido a que se presentan como soluciones eficientes a los problemas de las dependencias a largo plazo, proporcionando una memoria de mucho mayor capacidad que las redes RNN tradicionales, las redes LSTM son utilizadas actualmente en un amplio rango de aplicaciones de inteligencia artificial, como el reconocimiento de texto escrito a mano y generación del mismo, reconocimiento de voz, subtitulado automático o la predicción del consumo eléctrico en edificios.

2.3 Dispositivos FPGA

2.3.1 Contexto histórico

En la actualidad, los circuitos electrónicos digitales son utilizados en una inmensa mayoría de aplicaciones que requieren el uso de dispositivos electrónicos. Esto es debido fundamentalmente a que con el paso de los años tras su invención, el tamaño de estos circuitos se ha ido reduciendo sucesivamente a escalas que hace no tantos años resultaban inimaginables, habiéndose conseguido hasta ahora la fabricación de transistores del orden de nanómetros.

El hito que permitió el gran acercamiento de la electrónica hacia los circuitos digitales fue la invención del transistor de unión bipolar (BJT) por el físico norteamericano William Shockley, trabajador entonces de la compañía Bell Telephone Laboratories, en 1948. Esta creación permitió durante las décadas posteriores, la fabricación de circuitos electrónicos de un tamaño mucho más reducido que el que se había conseguido hasta entonces mediante el uso de las válvulas de vacío. También presentaban un consumo energético mucho menor que estas. Especialmente fue notable el uso de los transistores BJT en dispositivos que requerían la amplificación de señales eléctricas, como la radio o la televisión. Más adelante, en el año 1959, se inventó en la misma compañía en que una década atrás se había desarrollado el transistor BJT, el transistor MOSFET (Metal–Oxide–Semiconductor Field-Effect Transistor). Este tipo de transistores, basados en el aprovechamiento de las propiedades de los materiales semiconductores y en efectos producidos por el campo eléctrico, supusieron, al igual que lo habían hecho en su momento los transistores BJT, una importante revolución en la electrónica, puesto que se trataba de unos dispositivos de menor tamaño, menor coste de fabricación y mucho menor consumo de potencia que estos. La creación de estos transistores permitió la fabricación de circuitos integrados (ICs) de una gran densidad. Uno de los dispositivos basados en MOSFET más popular, y que es utilizado en la actualidad por una gran cantidad de personas, es el microprocesador. Mientras que los ordenadores cuya tecnología estaba basada en válvulas de vacío ocupaban salas completas, con los transistores MOSFET se ha logrado la fabricación de ordenadores de un tamaño tan reducido como los utilizados actualmente. Uno de los primeros microprocesadores construidos en un único chip, considerado por muchas fuentes el primero de ellos, fue el Intel 4004. Este microprocesador, desarrollado en 1971, contenía aproximadamente 2300 transistores MOSFET canal P.

Los dispositivos de almacenamiento también experimentaron grandes avances tras la invención del transistor MOSFET a partir de 1960. De esta forma, se consiguieron sucesivas mejoras en las memorias de solo lectura (ROM). El primer avance significativo fueron las memorias programables (PROM), las cuales, aunque solo podían ser configuradas una única vez quemando unos determinados fusibles, ya permitían hacerlo a través del usuario, minimizando así de forma considerable los tiempos en que un producto que utilizara esa tecnología tardaría en llegar al mercado. La Figura 2.9 ilustra el funcionamiento de una celda de memoria PROM. Posteriormente, se inventaron las memorias PROM borrables (EPROM) que permitían su borrado para su posterior programación a través de la exposición a rayos ultravioleta. Más adelante se crearon las memorias PROM eléctricamente borrables (EEPROM), que permitían ser borradas con excitación eléctrica, sin necesidad de utilizar rayos ultravioleta.

Por otro lado, también se desarrollaron avances importantes mediante la creación de las memorias de acceso aleatorio (RAM), memorias volátiles basadas en tecnología MOSFET que permiten, a diferencia de las memorias ROM, los accesos de escritura. Estas se dividen en las memorias estáticas (SRAM), cuyos valores almacenados permanecen inalterados hasta que se retira la alimentación o se hace un acceso de escritura y memorias dinámicas (DRAM), cuyos valores deben ser refrescados con cierta periodicidad para compensar los efectos de descarga de los condensadores que contienen las celdas.

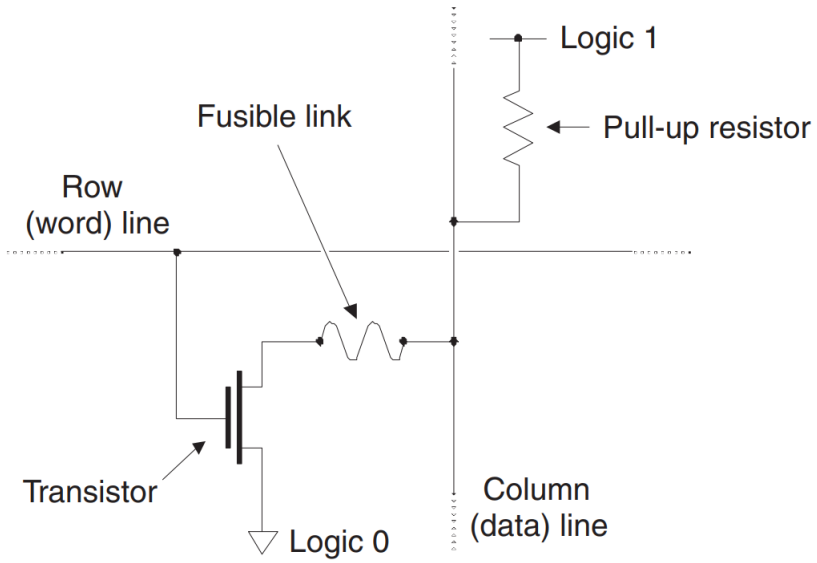


Figura 2.9: Celda de memoria PROM. El enlace fusible puede ser quemado mediante la aplicación de una determinada corriente eléctrica para conseguir un circuito abierto entre sus dos terminales de forma permanente [7].

Estos importantes desarrollos en el campo de las memorias dieron lugar a la creación de los dispositivos lógicos programables (PLDs) desde los años 70, ya que se descubrió que las memorias PROM podían ser utilizadas para la implementación de funciones lógicas. Estos dispositivos se clasifican en dos grandes grupos: PLDs simples (SPLDs), denominación que adquirieron los primeros dispositivos basados en memorias PROM y PLDs complejos (CPLDs), basados en bloques de menor tamaño con conexiones programables entre sí. Los CPLDs son una tecnología muy utilizada en la actualidad, debido a que ofrecen grandes posibilidades de configuración y los tiempos requeridos para modificar la lógica son muy reducidos. Debe destacarse que estos circuitos no permiten la implementación de funciones lógicas de gran complejidad.

Por otro lado, para conseguir soluciones que utilizaran gran cantidad de elementos lógicos podría plantearse el uso de circuitos integrados de propósito general, esto es, el montaje de circuitos a partir de la interconexión manual de distintos chips que contienen circuitos lógicos. Sin embargo, esta solución no resultaba muy eficiente debido al gran tamaño que ocupaban los circuitos. Es por ello que, tras la invención de los transistores MOSFET, comenzaron a fabricarse circuitos integrados de aplicación específica (ASICs). Estos dispositivos se basan en la integración de una gran cantidad de elementos lógicos en un área reducida de Silicio. La fabricación de un ASIC puede realizarse mediante la utilización de diferentes técnicas. Las diferencias que se deben tener en cuenta para elegir una de ellas residen en la relación entre el coste de diseño y fabricación y la densidad que se pretende conseguir, esto es, la cantidad de lógica que puede introducirse en una superficie determinada. De esta manera, aunque el coste de producción de estos dispositivos es significativamente alto, estos continúan diseñándose y fabricándose en la actualidad, puesto que pueden conseguirse diseños de gran densidad y elevadas prestaciones.

Puede entonces observarse que, mientras que los PLDs son altamente configurables pero no ofrecen la posibilidad de implementar funciones lógicas muy complejas, los ASICs sí permiten la integración de grandes cantidades de lógica en superficies de tamaño muy reducido, aunque no son configurables y su precio resulta elevado. En este contexto, en los años 80 se busca una solución de compromiso entre ambos tipos de circuitos que da lugar a un nuevo dispositivo, el cual revolucionaría el mercado de los sistemas electrónicos digitales en los próximos años: la matriz de puertas lógicas programable en campo o *Field-Programmable Gate Array*. (FPGA).

2.3.2 FPGAs: concepto, evolución y aplicaciones

Las FPGAs presentan, de manera simplificada, una arquitectura compuesta de un conjunto de bloques lógicos configurables (CLBs), una serie de interconexiones programables que permiten conectar los CLBs entre sí, y puertos de entrada y salida. Además, poseen bloques de interruptores que permiten la conmutación entre las distintas ramas de interconexión. En la Figura 2.10 se ilustra este concepto.

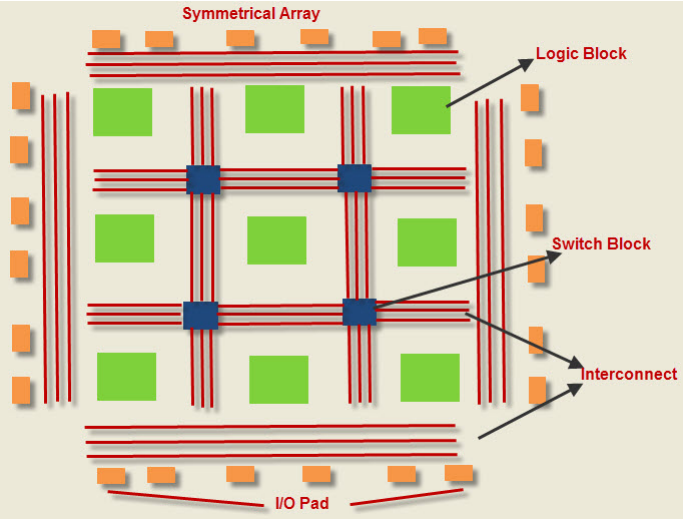


Figura 2.10: Arquitectura genérica de una FPGA. Extraído de www.elprocus.com. Basics of FPGA Architecture and Applications.

Los CLBs contienen en su interior varias celdas lógicas formadas por un biestable, un multiplexor y una *Lookup Table* (LUT). En la Figura 2.11 se muestra un diagrama simplificado de una celda lógica utilizada en algunas FPGAs de Xilinx, uno de los mayores fabricantes de FPGAs en la actualidad. Las LUTs son elementos que se comportan como un multiplexor de un número de entradas de selección determinado, variando este en la actualidad entre 3 y 6 entradas. Con estos elementos puede implementarse cualquier función lógica que contenga un número de entradas igual o inferior al de la LUT, independientemente de su complejidad. Para ello, suele conectarse a las entradas de datos del multiplexor una memoria SRAM en la que se almacenan los valores de salida de la función lógica, de forma que se consigue implementar la función deseada. Además, como puede apreciarse en la Figura 2.11, actualmente las LUTs poseen funcionalidades adicionales, como la de comportarse como memorias RAM o registros de desplazamiento.

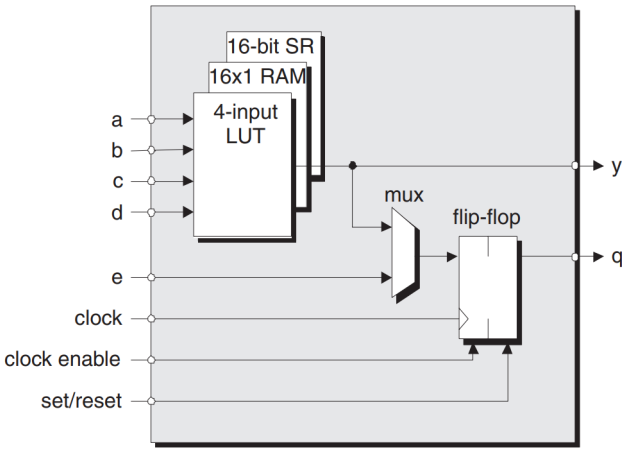


Figura 2.11: Diagrama simplificado de una celda lógica utilizada en algunas FPGAs de Xilinx [7].

Desde su creación, el uso de las FPGAs ha ido aumentando considerablemente debido a su rentabilidad, ya que, con el paso de los años, la capacidad (cantidad de celdas lógicas en un mismo área de Silicio) y la velocidad de estas han aumentado notablemente a la vez que se ha producido una reducción notable de sus precios y del consumo de energía de las mismas. El gráfico de la Figura 2.12 muestra un gráfico en el que se ilustra la evolución de estos parámetros en las FPGAs producidas por Xilinx desde 1988 hasta aproximadamente 2010.

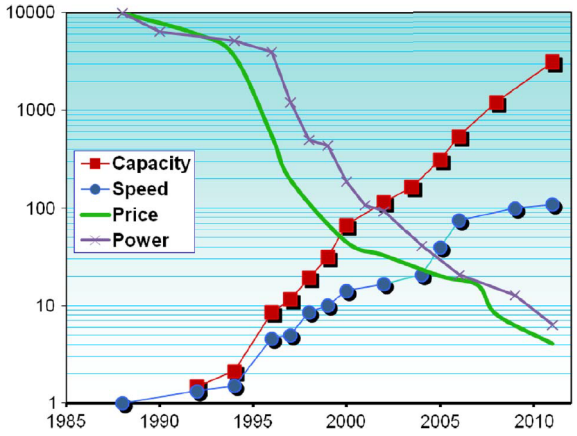


Figura 2.12: Evolución de los atributos de las FPGAs Xilinx desde 1988. La variable del eje de ordenadas es el factor en el que aumenta o disminuye cada uno de los parámetros en referencia a los existentes en 1988. Gráfica publicada por Xilinx [8].

La evolución de los factores mencionados anteriormente junto a la inclusión de elementos como multiplicadores y sumadores lógicos en bloques optimizados para procesamiento digital de señales o memorias RAM, que hacen a las FPGAs más útiles en muchos campos de aplicación, han permitido que las FPGAs se utilicen actualmente en múltiples ramas de la Ingeniería. Estas aplicaciones varían desde sencillos circuitos de *glue logic* para conectar dispositivos de mayor complejidad hasta el procesamiento de grandes cantidades de datos provenientes de sensores, procesamiento digital de señales digitales o prototipado de ASICs. Además, hace algunas generaciones de FPGAs se inició la tendencia de incorporar, además de la lógica reconfigurable, sistemas basados en microprocesador o *Processing Systems* (PS), lo que ha potenciado el uso de las FPGAs en aplicaciones de inteligencia artificial y en la implementación eficiente de algoritmos de visión artificial, entre muchos otros campos.

Las FPGAs son dispositivos especialmente interesantes para la implementación de redes neuronales. En las redes neuronales prealimentadas, el paralelismo de los circuitos digitales es muy aprovechable, ya que tanto las neuronas de una misma capa como las propias capas operan de forma paralela. Por otro lado, en las redes neuronales recurrentes, como es el caso de esta aplicación, también se aprovecha el paralelismo, puesto que la operación de las capas también es simultánea. Por supuesto, las FPGAs resultan muy útiles para estos propósitos debido a que, en ocasiones, cuando una red lleva un tiempo funcionando, aparece la necesidad de cambiar esta por otra versión mejorada o por otra red que permita satisfacer nuevas especificaciones. En estos casos, resulta muy útil la posibilidad de reconfigurar la FPGA en cualquier instante.

2.3.3 La tarjeta de desarrollo Nexys 4 DDR

Aunque la mayor parte de los elementos del diseño que se propone en este trabajo pueden ser implementados en cualquier FPGA debido al uso de código VHDL plano, se aprovechan algunos recursos de

la tecnología destino utilizada para validar el modelo: la FPGA de Xilinx XC7A100T-1CSG324C perteneciente a la familia Artix-7. Estos recursos son los bloques DSP48E1 y las memorias Block RAM (BRAM), cuyo funcionamiento se explica en detalle en el capítulo 3. Esta FPGA es incorporada en la placa de desarrollo Nexys 4 DDR fabricada por Digilent. En la Figura 2.13 se muestra una imagen de esta tarjeta de desarrollo.

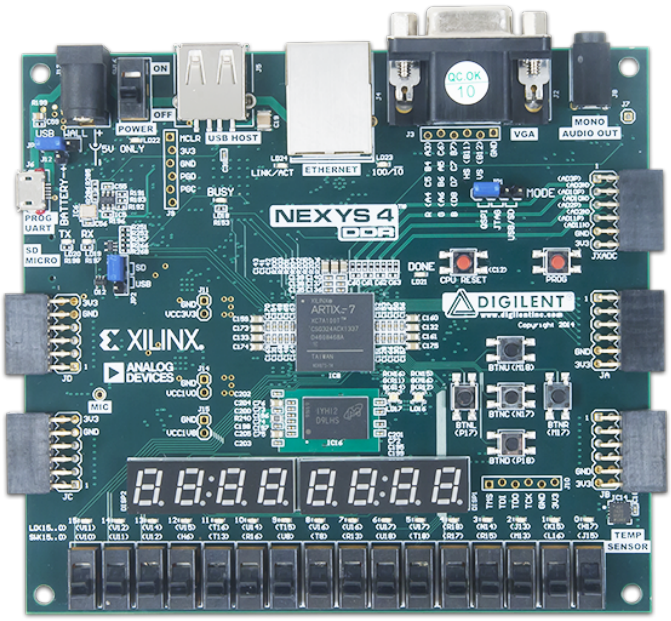


Figura 2.13: Tarjeta de desarrollo Nexys 4 DDR. Extraído de digilent.com. Nexys 4 DDR Reference Manual.

En la Figura pueden apreciarse varias interfaces de entrada y salida, de las cuales gran parte son interfaces Pmod, estándar definido por Digilent [9] para la conexión de módulos de periféricos de baja frecuencia con un número reducido de pines. Debe destacarse que uno de estos puertos posee un conversor analógico-digital integrado. Además, cuenta con puertos USB y Ethernet, entre otros. Esta placa, además, contiene interruptores, pulsadores genéricos para permitir al usuario introducir estímulos a través de ellos. Por otro lado, también ofrece la posibilidad de utilizar como salidas un conjunto de LEDs integrados y ocho displays de siete segmentos de ánodo común. Los LEDs resultan especialmente interesantes para validar las salidas de los diseños en la fase de pruebas. La elección de esta FPGA como la tecnología destino propuesta para este proyecto se debe, además de a los bloques DSP y BRAM, a que las FPGAs de la familia Artix-7 son recomendadas para aplicaciones de bajo coste y reducido consumo de potencia [10].

Capítulo 3

Arquitectura de la red

3.1 Introducción

En este capítulo se estudiará el diseño del circuito digital planteado para la implementación de redes LSTM. En el mismo, se expondrá la codificación escogida para las señales del sistema. Posteriormente, se realizará una explicación detallada del modelo creado para la implementación de una celda LSTM única, detallando su estructura y la de cada uno de los módulos que la componen, justificando las técnicas utilizadas para optimizar su funcionamiento. En esta misma sección se explica la metodología utilizada para cuantificar cada una de las señales de la celda y se aportan los resultados de la simulación funcional que permiten validar la operación del diseño en términos de especificaciones funcionales. Para concluir el capítulo, se presenta la construcción del modelo hardware para la implementación de una capa LSTM. Finalmente, se exponen los resultados de la validación de la operación funcional de dicho modelo, obtenidos mediante simulación.

3.2 Representación binaria de las señales

En un sistema digital resulta fundamental definir la forma en la que se representan los números para permitir su interpretación. El propósito de esta sección es exponer la codificación utilizada de manera general para la representación numérica de las señales de la celda.

3.2.1 Representación en complemento a dos

En primer lugar, se debe escoger un sistema de codificación para poder representar números positivos y negativos. Este sistema será el complemento a dos, debido a que con esta notación puede realizarse correctamente la resta de números sumando al minuendo el complemento a dos del sustraendo, que representaría ese mismo número con signo negativo.

El complemento a dos de un número se calcula como la suma de una unidad al complemento a uno de ese mismo número, que consiste en la negación de todas sus cifras. Esto se puede representar mediante la siguiente igualdad:

$$C_2(N) = \bar{N} + 1 \tag{3.1}$$

De esta forma, cuando se pretende obtener el valor de un número negativo en complemento a dos, se debe realizar la operación descrita en la ecuación 3.1 sobre este. Se debe operar de la misma forma en la situación opuesta, esto es, si se tiene un número negativo en complemento a dos y se desea obtener el mismo con signo positivo. Nótese que, en esta representación, el bit de mayor peso actúa como bit de signo, de forma que en números positivos este tomará el valor '0' y en números negativos el valor '1'. Por último, debe considerarse que, para ampliar el tamaño en bits de un número en complemento a dos, debe copiarse el bit de signo a la izquierda de dicho número tantas posiciones como requiera la ampliación.

3.2.2 Aritmética de punto fijo

Por otro lado, es necesario utilizar un sistema para la representación de números reales con parte decimal. En múltiples sistemas, como los microprocesadores, es muy común utilizar la representación en coma flotante, ya que permite expresar números de gran resolución de manera muy compacta, es decir, con un número reducido de bits. Esto resulta muy útil para ahorrar recursos en dispositivos que trabajan con grandes cantidades de datos expresados con gran precisión.

Sin embargo, para poder realizar operaciones aritméticas con estos números deben diseñarse circuitos digitales que realicen específicamente operaciones con esta representación. Normalmente los circuitos que realizan estas operaciones son las Unidades de Punto Flotante (FPU). Debido a que la implementación de estos circuitos utilizaría una cantidad notable de recursos de la FPGA y con el objetivo de disminuir la latencia de reloj y mejorar la eficiencia, se opta por utilizar la representación en coma fija. Esto permite utilizar los circuitos sumadores y multiplicadores ya implementados por el fabricante de la FPGA, lo que se traduce en un ahorro de recursos significativo.

Esta representación resulta sencilla: cada palabra contiene una parte fija y una parte fraccionaria. Para interpretar los números debe seleccionarse el número de bits que contiene cada parte. La Figura 3.1 ilustra esta representación, teniendo en cuenta, además, el bit de signo que aparece en la codificación en complemento a dos.

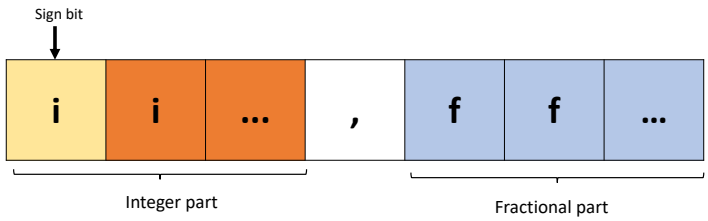


Figura 3.1: Representación de números con signo en coma fija.

3.3 Modelo de la celda

En esta sección se pretende explicar el funcionamiento interno de la celda. La Figura 3.2 muestra la interfaz externa de la neurona, en la que se pueden apreciar sus entradas y salidas. Nótese que los puntos suspensivos hacen referencia a las entradas correspondientes de los pesos sinápticos y sesgos del resto de puertas, esto es, input gate y cell candidate.

El modelo propuesto para la celda cuenta con una señal de reset `rst` que fija a 0 todas las señales internas y los resultados. También posee una señal de habilitación (`enable`), la cual permite el funcionamiento normal de la celda. En caso de que la señal de habilitación se desactive durante el funcionamiento,

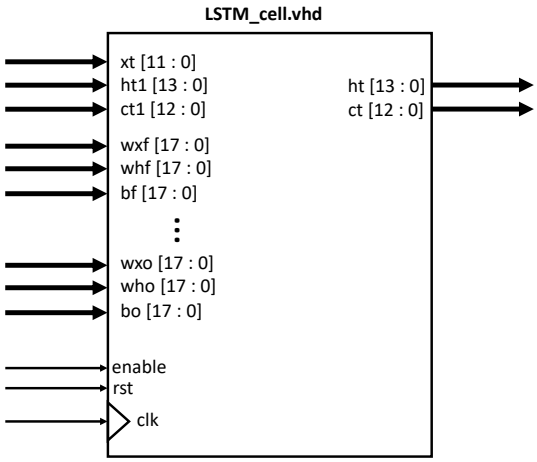


Figura 3.2: Interfaz externa de la celda.

todas las señales de la celda, incluidas las salidas, permanecerán con su último valor inalterado hasta que vuelva a habilitarse el funcionamiento o se produzca un `reset`.

El diseño es completamente síncrono, por lo que se introduce una señal de reloj común para todos los elementos secuenciales de la neurona. Se opta, además, por un diseño segmentado en seis etapas para optimizar la frecuencia de operación del circuito. También debe destacarse que los pesos sinápticos y sesgos se introducen como entradas.

El diseño hardware de la neurona se crea utilizando una estructura jerárquica compuesta de distintos módulos para facilitar el orden del código y la edición de las diferentes partes del mismo, ya que cada una de ellas puede operar de forma independiente al resto. En la Figura 3.3 se muestra la interconexión de los módulos que componen la celda.

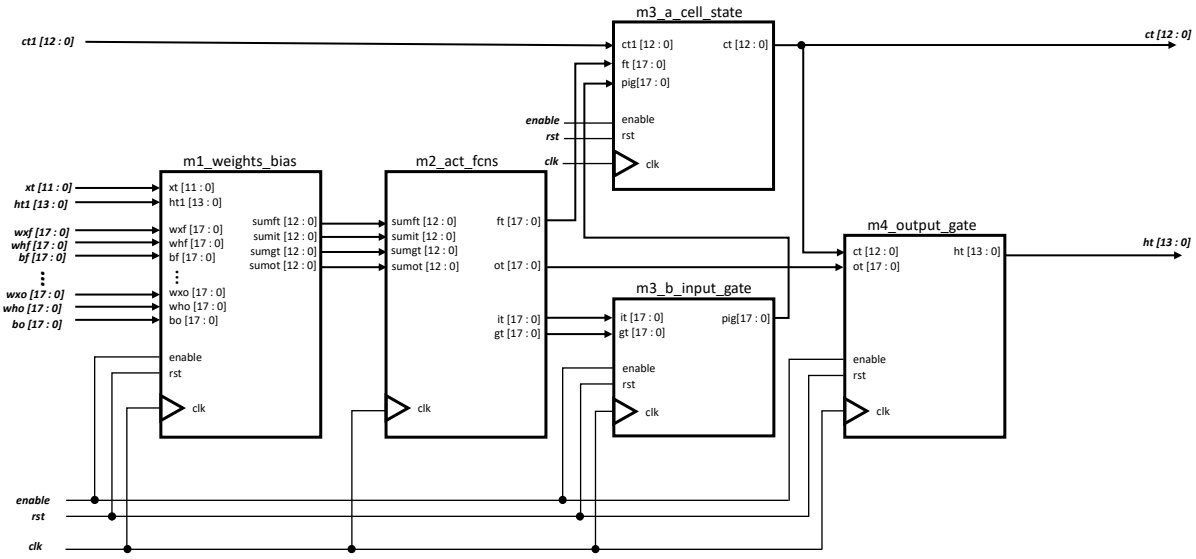


Figura 3.3: Esquemático interno de la neurona.

En las posteriores subsecciones se define el comportamiento de cada uno de los módulos de la celda. Por último, el apartado 3.3.5 explica con detalle la temporización que se pretende conseguir en el diseño y cómo afecta a esto la estructura segmentada.

3.3.1 Módulo 1: Pesos sinápticos y sesgos

El propósito de este módulo es realizar los productos correspondientes de los pesos sinápticos (w) con las entradas x_{t-1} y h_{t-1} y sumar a estos las señales de polarización o sesgo (b), obteniendo así las señales de entrada de las funciones de activación. Debe destacarse que estos multiplicadores se implementan en un diseño secuencial y constituyen la primera etapa de la segmentación. De igual forma, se registran las señales de sesgo para que puedan sincronizarse con estos. La parte de la celda que implementa este primer módulo se destaca en la Figura 3.4. Las siguientes ecuaciones ilustran estas operaciones:

$$sumft = W_{xf} \cdot x_t + W_{hf} \cdot h_{t-1} + b_f \tag{3.2}$$

$$sumit = W_{xi} \cdot x_t + W_{hi} \cdot h_{t-1} + b_i \tag{3.3}$$

$$sumgt = W_{xg} \cdot x_t + W_{hg} \cdot h_{t-1} + b_g \tag{3.4}$$

$$sumot = W_{xo} \cdot x_t + W_{ho} \cdot h_{t-1} + b_o \tag{3.5}$$

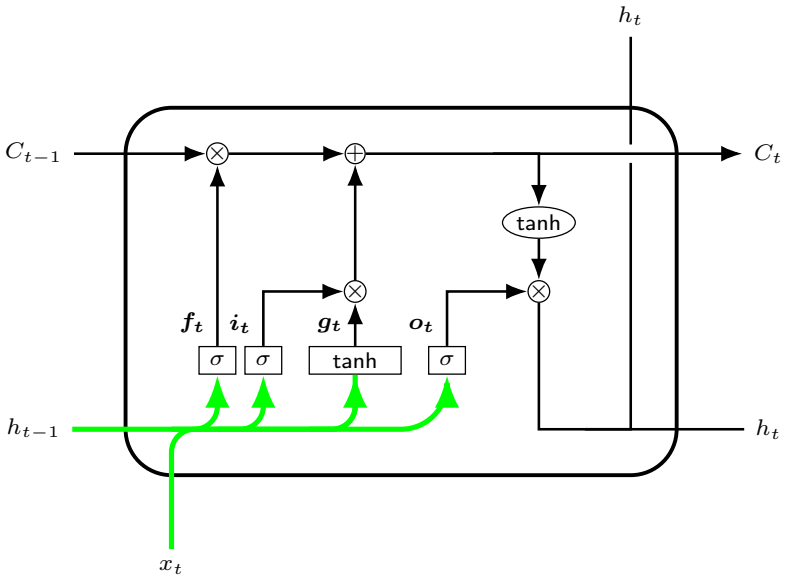


Figura 3.4: Diagrama de los elementos del módulo 1.

La entrada a la celda x_t se codifica en 12 bits: 1 bit para la parte entera y 11 bits para la parte fraccionaria. Esto se debe a que en los sistemas digitales las señales de entrada suelen introducirse desde un convertidor ADC, cuyo tamaño más típico es de 12 bits. Por otro lado, la entrada h_{t-1}, que proviene de la salida de la neurona anterior de la capa es codificada en 14 bits, con un único bit para la parte entera y el resto para la parte fraccional. La justificación para seleccionar este tamaño es que en ciertas ocasiones puede resultar conveniente en el proceso de adquisición de datos obtener la salida de la red en formato analógico, por lo que esta se conectaría a un DAC para realizar la conversión. De manera análoga a lo que ocurre con los ADC de 12 bits, los DAC de 14 bits son muy comunes en los sistemas de adquisición de datos. Los pesos sinápticos y el sesgo poseen un ancho de palabra de 18 bits, estando formada la parte decimal por 17 bits. Con esto se pretende buscar una solución de compromiso con la que se obtenga una exactitud válida para los datos sin que ello implique un consumo excesivo de recursos.

Los multiplicadores que utiliza esta FPGA, debido a que pertenece a la familia *Artix-7*, son los incluidos en los slices DSP48E1, cuyo tamaño máximo de operación es de 25×18 bits. La Figura 3.5 muestra el circuito de estos slices. Todos los resultados de los productos se reducen a 18 bits para evitar sobrecargar posteriormente los sumadores, ya que sus tamaños son muy elevados.

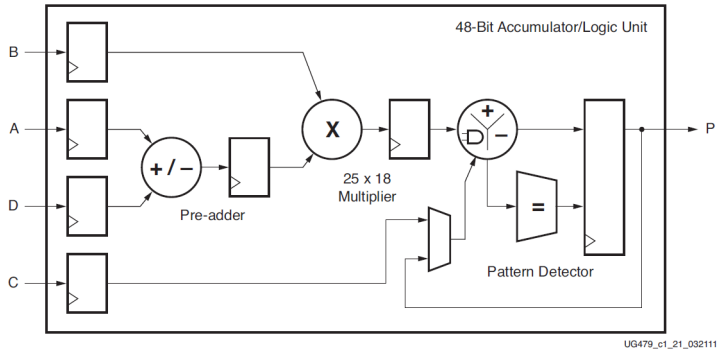


Figura 3.5: Diagrama de bloques del slice DSP48E1 [11]

Tras realizarse estas operaciones, el ancho de palabra de los resultados obtenidos es de 20 bits, ya que se han sumado tres operandos de 18 bits cada uno. Por último, el tamaño de estos se reduce a 13 bits. Esta decisión se explica en el apartado 3.3.2.

3.3.2 Módulo 2: Funciones de activación

Este módulo incorpora las funciones de activación de las cuatro compuertas. Se crea un modelo que implemente las siguientes ecuaciones:

$$\begin{aligned} f_t &= \sigma(W_{xf} \cdot x_t + W_{hf} \cdot h_{t-1} + b_f) \\ i_t &= \sigma(W_{xi} \cdot x_t + W_{hi} \cdot h_{t-1} + b_i) \\ g_t &= tanh(W_{xg} \cdot x_t + W_{hg} \cdot h_{t-1} + b_g) \\ o_t &= \sigma(W_{xo} \cdot x_t + W_{ho} \cdot h_{t-1} + b_o) \end{aligned} \tag{3.6}$$

A continuación se detalla el método utilizado para implementar las funciones matemáticas en la FPGA, así como los recursos empleados para ello. La Figura 3.6 muestra la parte de la celda que se implementa en este módulo.

3.3.2.1 Tablas de consulta

La incorporación de circuitos que realicen funciones matemáticas en un sistema digital puede abordarse con diversas metodologías. La diferencia entre estas reside en el consumo de recursos y el error de los datos de cada una de ellas.

En este diseño se busca una implementación que trabaje con números en coma fija y que proporcione unos datos cuyo error de cuantificación no resulte excesivamente elevado para calcular el resultado de dos funciones matemáticas: la función logística y la función tangente hiperbólica. Por ello, se opta por la utilización de tablas de consulta (traducción del término inglés *"lookup table"*). Las tablas de consulta permiten la aproximación de funciones matemáticas mediante el almacenamiento de los resultados pre-calculados. Esto podría hacerse sencillamente almacenando los datos y posteriormente realizando accesos

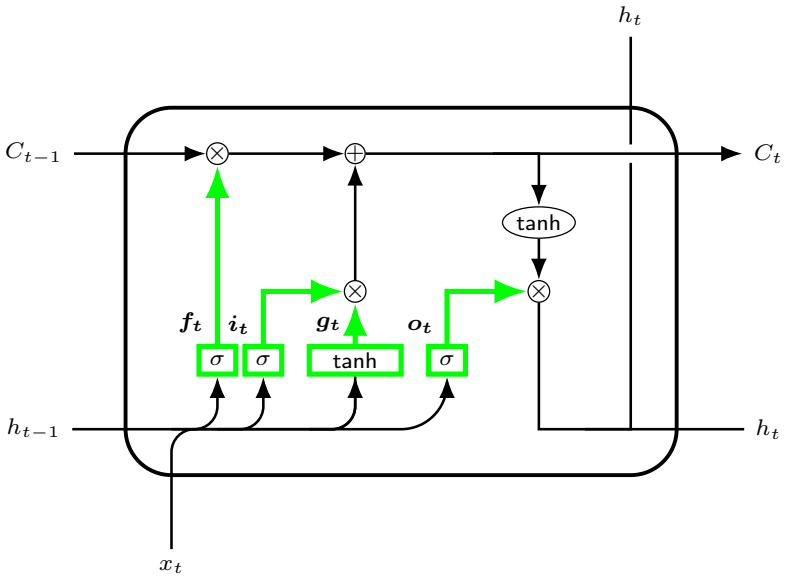


Figura 3.6: Diagrama de los elementos del módulo 2.

de lectura en una memoria RAM. La Figura 3.7 muestra un ejemplo de esto. En ella, la columna *Address* contiene las direcciones de memoria en las que se almacena cada resultado y la columna *Data* contiene los resultados de las entradas x_n almacenados en las distintas direcciones de la memoria.

Address	Data
0x0100	$f(x_1)$
0x0104	$f(x_2)$
0x0108	$f(x_3)$
0x010C	$f(x_4)$
0x0110	$f(x_5)$

Figura 3.7: Ejemplo de tabla de consulta en memoria RAM.

3.3.2.2 Memorias Block RAM

Para crear las tablas de consulta se utilizan las memorias BRAM disponibles para la familia de FPGAs *Artix-7*. Estas memorias integradas en el área de silicio de la placa son memorias RAM de 36 Kb. Debido a la posición que tienen las funciones de activación en la celda, el funcionamiento síncrono de las memorias permite que estas constituyan la segunda etapa de la segmentación del modelo.

En este diseño se plantea calcular los resultados de la función para un número determinado de entradas con Matlab y generar el archivo de inicialización que contenga estos resultados para proporcionárselo a la memoria BRAM, permitiendo que se puedan realizar accesos de lectura en esta durante el funcionamiento normal de la celda. Debido a este mecanismo de trabajo, en la memoria se realizan únicamente accesos de lectura, sin ser necesarios los accesos de escritura. Por ello, esta se configura como una memoria

ROM. El funcionamiento consiste en introducir los datos de entrada, que corresponderían a la variable independiente de la función, por la entrada de direcciones de la memoria, obteniendo a la salida el resultado correspondiente al dato de entrada introducido. Por ello, resulta especialmente importante el número de bits que componen los datos de entrada (salidas del módulo 1), ya que de este factor depende el número de memorias BRAM consumidas y la cantidad de resultados que pueden almacenarse en estas. Tras realizar las pruebas y comprobar que se superaba satisfactoriamente el error de cuantificación en Matlab (véase apartado 3.3.6), se decidió que las entradas se codificarían en 13 bits: 3 de parte entera y 10 de parte fraccional. El hecho de que haya 3 bits de parte entera se justifica debido a la pretensión de evitar situaciones de desbordamiento en la suma realizada en el módulo 1 de tres elementos con un bit de parte entera cada uno. Las salidas se cuantifican con 18 bits: uno para la parte entera y el resto para la parte fraccionaria. No son necesarios más bits para la parte entera debido a que la función logística devuelve números en el intervalo $[0, 1]$ y la función tangente hiperbólica en el intervalo $[-1, 1]$.

Teniendo en cuenta que, como se ha mencionado en el párrafo anterior, se tiene un bus de direcciones de 13 bits y un bus de datos de salida de 18 bits, para la implementación de una única función de activación es necesaria una memoria de 8Kx18. Las BRAM de 36Kb se configuran para esta aplicación como memorias de 2Kx18, y realizando una expansión en profundidad utilizando cuatro unidades se obtiene el número de palabras deseado. Entonces, teniendo en cuenta que cada celda contiene 4 funciones de activación, se tienen 20 memorias BRAM por celda. El consumo de memorias BRAM se analiza con más detalle teniendo en cuenta la tecnología destino en la sección de consumo de recursos del capítulo 5.

Debe destacarse que los puertos de la memoria son simplemente la entrada de direcciones y la entrada de reloj debido a que opera de forma síncrona y la salida. Esto se ilustra en la Figura 3.8, que muestra los bloques que componen el módulo 2 del diseño.

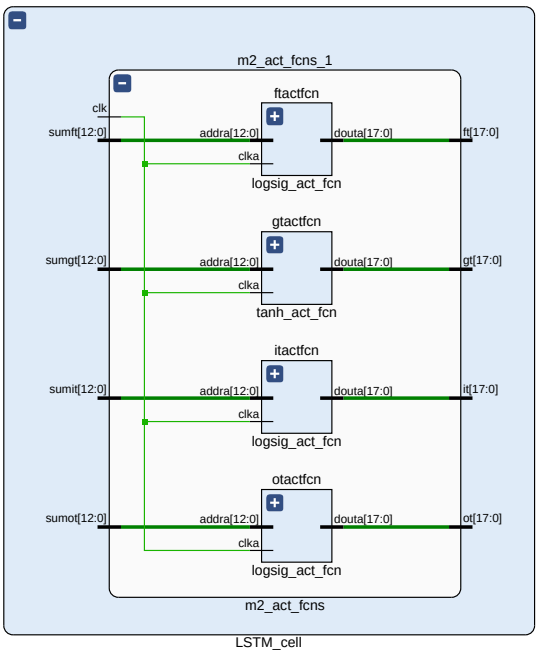


Figura 3.8: Diagrama de bloques del módulo 2.

3.3.2.3 Inicialización de datos en Matlab

Para crear el fichero de inicialización de las memorias BRAM se utilizan varias funciones de Matlab. Estas crearán los ficheros con los datos ordenados correctamente y en el formato necesario para introducirlos directamente en Vivado (formato de fichero `.coe`). Un ejemplo de este fichero de inicialización seria el siguiente:

```
memory_initialization_radix = 10;
memory_initialization_vector =
65536,65567,65599,65631,65663,65695;
```

Donde `memory_initialization_radix` indica la base en la que se introducen los números, en este caso 10 porque se introducen en formato decimal y `memory_initialization_vector` contiene los números que se introducen como inicialización en el orden deseado.

La primera función necesaria es `input_test_gen`, que se encarga de crear el vector de entradas para la tabla de consulta en función del ancho de palabra y del tamaño de la parte fraccional deseados para las entradas. La realización de este script debe contemplar el orden en que deben generarse los datos, ya que se pretende que al introducir por la entrada de direcciones de la BRAM un dato x_n, la salida proporcione el resultado de la función $f(x_n)$. Por ello, el valor del dato representado en punto fijo con el número de bits deseado para cada parte y codificado en complemento a dos debe coincidir con el valor de la dirección en la que está almacenado el resultado correspondiente a ese dato. Para conseguir ordenar los datos de esta forma, se obtiene el valor de la entrada de direcciones en función del valor del dato de entrada en coma fija como la siguiente función definida a tramos:

$$\begin{cases} addr(x_n) = x_n \cdot 2^{fraclength} & \text{si } x_n \geq 0, \\ addr(x_n) = 2^{wordlength} + x_n \cdot 2^{fraclength} & \text{si } x_n < 0 \end{cases}$$

Donde *wordlength* es el ancho de palabra y *fraclength* es el tamaño en bits de la parte decimal del número x_n. Nótese que la causa de que la función esté definida a tramos es la codificación en complemento a dos, ya que el bit de signo de los números es el MSB. Por ello, cuando se pretenden obtener las direcciones de los números negativos en orden, debe hacerse con el orden inverso.

Tras obtener el vector de entradas, los valores de la salida que se almacenarán en la Lookup Table se generan mediante la función `lutgen`, la cual obtiene los valores de salida tras introducirle los argumentos necesarios: tipo de función de activación (función logística o tangente hiperbólica), ancho de palabra de la entrada y de la salida, y ancho de la parte fraccionaria de la entrada y de la salida. Los tamaños de palabra de entrada se introducen para conocer el número de elementos que debe generar, además, para poder calcular el resultado de la función, internamente hace una llamada a la función `input_test_gen`. Finalmente, proporciona la salida cuantificada conforme a las especificaciones de tamaño deseadas.

Por último, se necesita una función que genere el fichero .coe con los datos obtenidos y en el formato correcto. Para ello se crea la función `luttocoe`, que recibe como argumentos de entrada el vector de elementos generado con la función `lutgen` y el ancho de palabra y el tamaño de la parte fraccional de la salida. Esta función genera el vector con los valores obtenidos como entrada codificados en formato decimal para poder introducirlos correctamente en el .coe. Tras ello, imprime en un fichero de extensión `.coe` con el formato adecuado. Puesto que los números deben estar codificados en coma fija y Matlab trabaja por defecto con números en coma flotante de precisión doble (64 bits), se utiliza el toolbox Fixed-Point Designer para facilitar la conversión de los números a formato decimal en notación de punto fijo.

Para automatizar la tarea de creación de ficheros `.coe` para las distintas funciones de activación de cada puerta se crea el script `luts_obtain.m`, en el que se definen los distintos tamaños de cada parte de las señales de entrada y salida de cada puerta y, mediante el uso de las funciones descritas anteriormente genera el fichero `.coe` deseado listo para cargarlo desde el asistente de personalización de IPs de Vivado.

3.3.3 Módulo 3: Cell State e Input Gate

Esta etapa del diseño se divide en dos partes: el módulo 3.a que modela el Cell State de la neurona y el módulo 3.b que modela el producto que aparece en la Input Gate. Se utiliza esta denominación para los módulos debido a que los multiplicadores de ambos módulos operan de forma paralela. En conjunto, ambas partes llevan a cabo la implementación de la ecuación de la salida c_t:

$$c_t = f_t \cdot c_{t-1} + i_t \cdot g_t \tag{3.7}$$

3.3.3.1 Módulo 3.a: Cell State

En este módulo se implementa el producto de la entrada del *Cell State* C_{t-1} por la salida de la compuerta *Forget Gate* f_t junto a la posterior suma de este resultado y la salida de la puerta *Input Gate* i_t. La Figura 3.9 destaca en color verde las partes implementadas en este módulo.

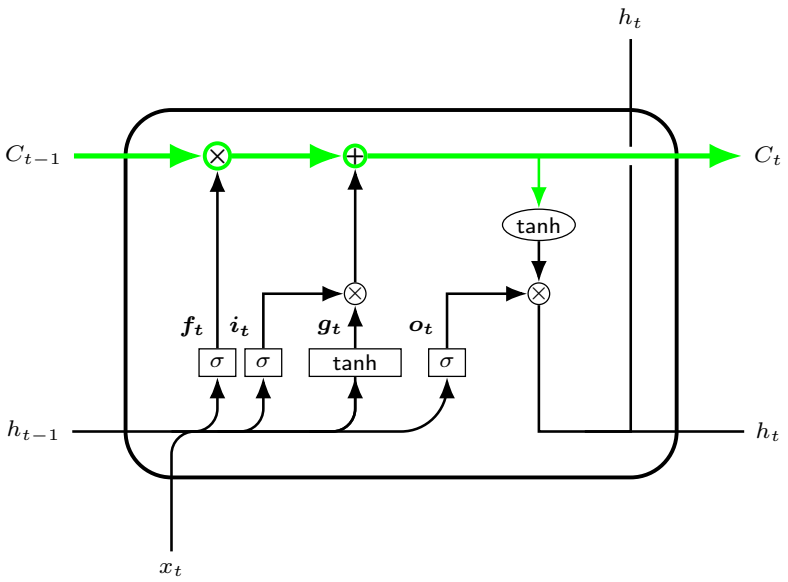


Figura 3.9: Diagrama de los elementos del módulo 3.a.

En primer lugar, debe notarse que, para que la entrada C_{t-1} esté sincronizada correctamente con f_t, la entrada del *Cell State* debe registrarse dos ciclos de reloj antes de introducirla al multiplicador, ya que desde que entra a la celda una entrada hasta que finalizan las operaciones pertinentes y aparece la correspondiente f_t, las señales atraviesan dos etapas de segmentación, es decir, deben ocurrir dos flancos ascendentes de reloj. Tras tener sincronizadas ambas señales se realiza el producto y se ajusta el número de bits del resultado a 18. Debe tenerse en cuenta que la entrada C_{t-1} de 13 bits contiene 2 bits de parte entera. Esto se debe a que C_t es la suma de dos señales con un único bit de parte entera cada una. Entonces, el producto que, inicialmente aparece codificado en 31 bits (3.28) se reduce a 1.17. La salida de

este producto se obtiene de forma registrada, por lo que constituye la tercera etapa de la segmentación del diseño junto al producto de la *Input Gate*.

La última operación que realiza es la suma del *Cell State*, que realiza la suma de dos señales de 1.17 bits cada una. Para evitar desbordamiento la suma se realiza en 19 bits. Esta suma se obtiene también de forma registrada con el propósito de que actúe como la cuarta etapa de la segmentación. Finalmente, se obtiene la salida C_t como los 13 bits MSB de este resultado.

3.3.3.2 Módulo 3.b: Input Gate

El propósito de este módulo es modelar el multiplicador de la compuerta *Input Gate* que, como se ha mencionado anteriormente, constituye de forma paralela al multiplicador del *Cell State* la tercera etapa de segmentación del diseño, por lo que tras obtener el producto se registra antes de enviarlo a la salida. En la Figura 3.10 se ilustra esta parte de la celda.

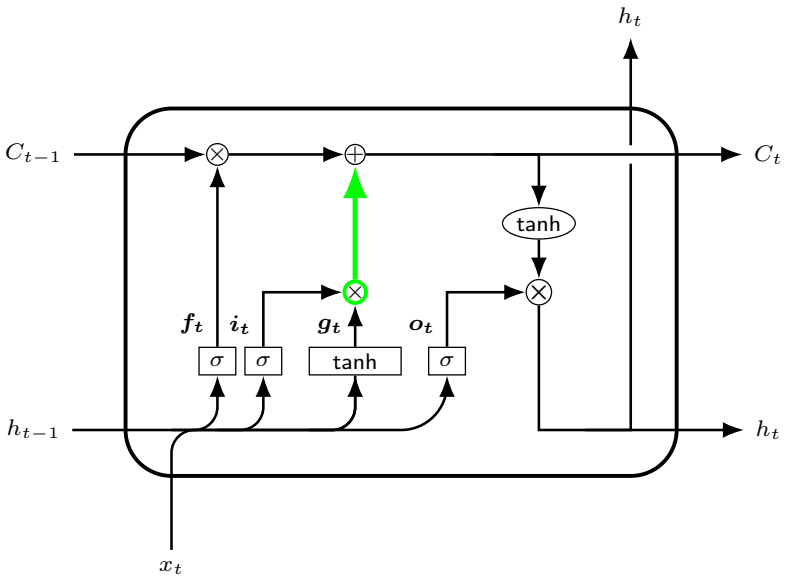


Figura 3.10: Diagrama de los elementos del módulo 3.b.

El producto se realiza en 36 bits, ya que multiplica dos señales de 18 bits. Tras realizarlo, se reduce su ancho a 1.17, eliminando el bit de parte entera que repite el bit de signo (MSB) y los bits de la parte fraccional de mayor precisión: los 17 bits LSB.

3.3.4 Módulo 4: Output Gate

Por último, en este módulo se realizan las operaciones de la *Output Gate* para proporcionar la salida h_t. Esta se obtiene mediante la siguiente ecuación:

$$h_t = o_t \cdot \tanh(c_t) \tag{3.8}$$

La Figura 3.11 destaca en color verde la etapa de la celda que se implementa en este módulo.

La primera operación interna que realiza es la aplicación de la función tangente hiperbólica a la salida del Estado de la celda c_t. Esta se implementa, al igual que en las funciones de activación del módulo 2

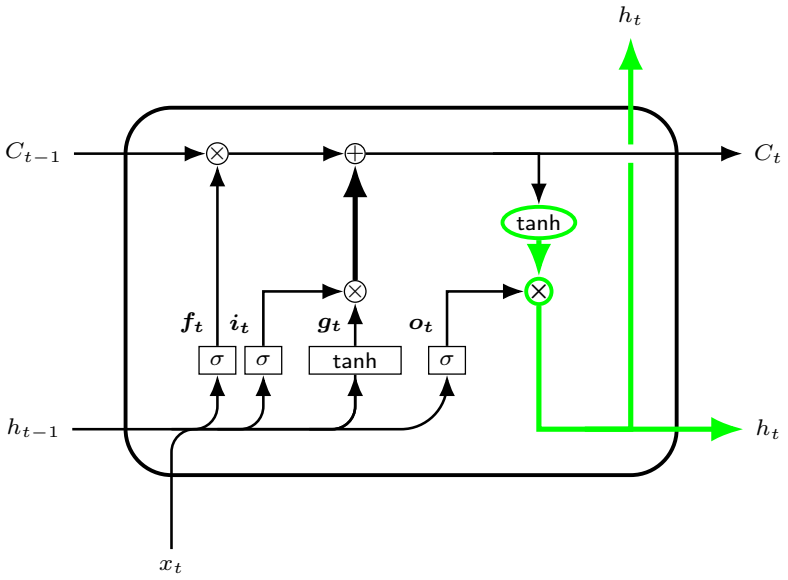


Figura 3.11: Diagrama de los elementos del módulo 4.

mediante una tabla de consulta y se sigue el mismo procedimiento para la inicialización. Esta función proporciona el resultado en 18 bits (1.17). Debido al comportamiento síncrono de la memoria, este proceso representa la quinta etapa de segmentación del diseño.

Posteriormente, para poder realizar el producto que proporciona la salida debe sincronizarse la señal o_t, que sale de la segunda etapa del pipeline con la salida de la tangente hiperbólica de la propia compuerta, que sale tras la quinta etapa. Por ello, esta señal se debe introducir en un registro de tres etapas.

El último paso es efectuar el producto, cuyo resultado aparece en 36 bits (2.34). Este producto, al igual que el resto, proporciona su resultado de forma registrada, constituyendo así la última etapa de segmentación del diseño. Puesto que como se ha indicado en la sección 3.3.1, la salida h_t se representa con 14 bits, empleando 13 de ellos para la parte fraccionaria, esta toma un único bit de signo y los trece restantes bits MSB de la parte fraccional del producto.

3.3.5 Temporización prevista para la celda: segmentación

La frecuencia de operación de un circuito digital constituye una parte fundamental de su diseño, ya que esta debe tenerse en cuenta cuando se pretenden introducir señales para conocer cuanto tiempo debe esperarse hasta introducir un nuevo dato de entrada, evitando así que algunos datos no se procesen debido a que no se han mantenido en la entrada el tiempo suficiente. Por otro lado, también es necesario tener en cuenta la frecuencia para poder adquirir correctamente los datos que el sistema proporciona en sus salidas, ya que si la señal de reloj del sistema de adquisición de datos es diferente a la del circuito, deben sincronizarse ambos dominios de reloj para evitar la pérdida de datos.

Por ello, para abordar este diseño consiguiendo la máxima velocidad de operación se opta por utilizar la técnica de segmentación o pipelining. Esta técnica se basa en el principio de que los retardos de propagación de un circuito combinacional son mucho mayores que la suma del *clock-to-out delay* y el tiempo de setup del biestable al que atacan. De esta forma, si se tiene un bloque combinacional entre dos biestables y este puede separarse a su vez en dos circuitos combinacionales, puede introducirse entre

medias de ambos un registro con el objetivo de dividir en dos etapas la operación del sistema, pudiendo así entregar datos a su salida con mayor velocidad.

La Figura 3.12 muestra un ejemplo en el que puede comprobarse el funcionamiento del pipelining. En este caso, se asume que el tiempo de propagación de ambos bloques combinacionales es igual para facilitar la comprensión de las ecuaciones. Además, se asume en ambos casos que el tiempo de propagación de los bloques combinacionales es mucho mayor que los retardos del biestable, que es lo que ocurre con gran frecuencia en los circuitos reales. En el primer circuito, toda la operación se realiza en una etapa, y la frecuencia máxima es la mostrada en la Figura. Por otro lado, en el segundo circuito, se añade un registro entre los dos bloques combinacionales que permite que la operación se realice en dos etapas. Esto es, mientras un dato es procesado por el segundo bloque combinacional, el siguiente dato es procesado al mismo tiempo por el primero. Como se puede comprobar, la frecuencia de operación máxima se duplica gracias a la aplicación de esta segmentación.

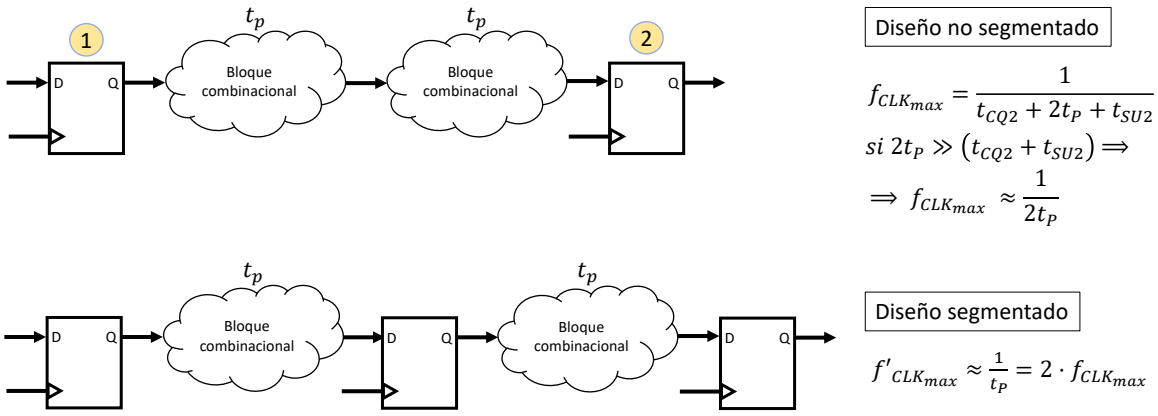


Figura 3.12: Ejemplo de comparación entre dos circuitos: con segmentación y sin segmentación. Idea basada en las transparencias de la asignatura de Diseño Electrónico

Como se ha ido explicando en el resto de apartados, la segmentación aplicada en este diseño divide el modelo en seis etapas. Con ella, se optimiza la frecuencia de operación de la celda una vez se llena el pipeline, consiguiendo así procesar hasta un dato por cada ciclo de reloj. Debe tenerse en cuenta que al introducir seis etapas de segmentación, la latencia del modelo, es decir, el tiempo de llenado del pipeline, es de 6 ciclos de reloj.

3.3.6 Dimensionamiento de la celda

Para que sea posible la implementación de una arquitectura hardware en una FPGA, debe adecuarse el tamaño de las señales internas de la misma para permitir que la celda realice las operaciones correctamente, evitando así desbordamientos o consumos excesivos de recurso. Para ello, es fundamental que el error producido por la disminución de tamaño de una señal permanezca dentro de un margen controlado, ya que errores excesivos pueden provocar funcionamientos no deseados en la celda, proporcionando resultados equívocos.

Sería posible hacer este proceso mediante la realización intensiva de pruebas en la FPGA. Sin embargo, esto requiere una gran cantidad de pruebas y de modificaciones de código. Es por ello que, en este trabajo, se decidió que este sería el primer paso a completar antes de comenzar con la descripción hardware de la arquitectura de la celda. El proceso consiste en modelar el comportamiento de la celda en Matlab con su

funcionamiento normal, es decir, con la representación en formato de coma flotante doble. Tras ello, se modela el comportamiento en coma fija. Para facilitar la realización de este proceso, este se llevó a cabo a través de Matlab, con ayuda de la toolbox Fixed-Point Designer, que proporciona algunas funciones que facilitan la cuantificación de variables en notación de coma fija.

Se fija como requisito que el error máximo producido por la cuantificación de cada una de las señales de la celda sea del 1,5 % del fondo de escala (o el 3 % de la mitad del fondo de escala). Por ejemplo, la mayoría de señales de la celda poseen un valor en el intervalo $[-1, +1]$. Entonces, el fondo de escala en este caso es de 2 unidades, por lo que el error máximo de cuantificación es un 3 % de 1, esto es, 0,03 unidades. Por ello, la expresión que determina el porcentaje de error es:

$$e(\,\%) = \frac{x - x_q}{\frac{1}{2}FS} \tag{3.9}$$

Donde x hace referencia a la señal original, x_q representa la versión cuantificada de la señal x y FS es el fondo de escala con respecto al que se determina el error.

El script `celda_lstm_fixed_point` aplica a cada una de las señales la cuantificación deseada y comprueba el requisito de error. Debe tenerse en cuenta, que al cuantificar señales y, a su vez, utilizar estas en las próximas etapas de la celda, el error se propaga a lo largo de la misma. Es por ello que las señales originales que se utilizan para calcular el error son las correspondientes al modelo de la neurona en coma flotante, mientras que las señales cuantificadas se obtienen a partir de las señales cuantificadas en etapas anteriores, al igual que ocurre en la FPGA.

Tras la realización de pruebas con diversos tamaños, estos se ajustan finalmente a los que se han ido detallando en las secciones desde 3.3.1 hasta 3.3.4. Para comprobar el requisito de error, se crea el script `test.m`, el cual realiza la inicialización pseudoaleatoria de todas las señales de entrada a la celda y los pesos sinápticos y ejecuta el modelo de la celda, que alerta al usuario si cualquier señal supera el error de cuantificación permitido. El script realiza esta prueba 50 veces. Tras ejecutarlo con los tamaños anteriormente mencionados, no se obtiene ningún error en ninguna de las ejecuciones, por lo que se puede concluir que las cuantificaciones son válidas.

3.3.7 Validación funcional del modelo

En el flujo de diseño típico de las FPGAs, una parte muy importante son las simulaciones, las cuales permiten comprobar si el diseño cumple las especificaciones planteadas antes de cargarlo en la FPGA. En un diseño hardware aparecen dos especificaciones bien diferenciadas: la funcionalidad, esto es, que el sistema relacione las entradas y salidas de una manera determinada y las especificaciones temporales, que fijan unos requisitos temporales que ya no dependen únicamente del diseño, si no que dependen de la tecnología destino en que se implemente el diseño.

Vivado ofrece herramientas de simulación para comprobar ambas especificaciones. Este apartado se centra en la simulación funcional, con el objetivo de comprobar que el comportamiento del diseño es correcto.

En primer lugar, se crea un modelo de simulación o testbench que contiene el diseño de la celda. Este incluye los estímulos que se aplicarán al modelo de la celda, es decir, señales de reloj, reset, entradas, etc. Los pesos sinápticos W_x se configuran a $0,5$, los pesos W_h se fijan a $0,125$ y a todos los sesgos b se les aplica valor nulo o 0.

Durante el tiempo de simulación se hace un reset inicial. Tras el mismo, se le da un valor determinado de forma aleatoria a las entradas x_t, h_{t-1} y c_{t-1}. Tras transcurrir un tiempo determinado, se cambia

el valor de las entradas y se desactiva la señal de habilitación. Transcurridos 10 ciclos de reloj, esta vuelve a activarse. Este proceso se hace para comprobar que esta señal funciona correctamente y para el funcionamiento de la celda. Los resultados de la simulación se muestran en la Figura 3.13.

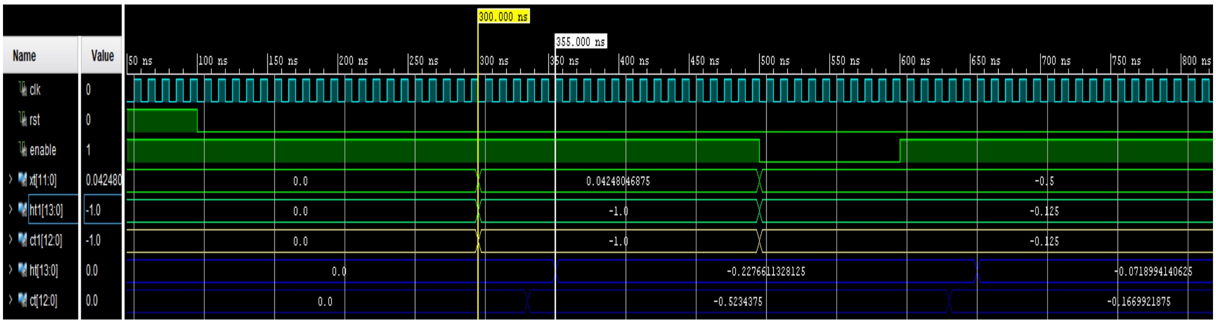


Figura 3.13: Simulación funcional de la celda.

En primer lugar, puede apreciarse que, desde que se introduce una entrada, hasta que se obtiene la salida correspondiente, hay una latencia de 6 ciclos de reloj. Esto indica que el funcionamiento es correcto, debido a la segmentación de 6 etapas. Después, puede comprobarse cómo el tiempo que la señal de habilitación permanece desactivada el valor de la salida no se altera. Una vez se activa y transcurren los ciclos de reloj correspondientes las salidas se actualizan. Para validar los resultados y que la precisión de estos sea la prevista con Matlab, se crea un script `test_script.m` que ejecuta la celda para sendos conjuntos de valores. Tras ejecutarlo se obtienen los mismos valores con la misma precisión decimal. Por ello, se comprueba que el modelo ofrece el comportamiento esperado.

3.4 Modelo de la capa

En esta sección se describirán la interfaz externa de la capa y su diseño interior. Además, se exponen los resultados de las simulaciones funcionales que permiten validar el comportamiento del modelo diseñado.

El modelo de la capa podrá contener una o varias celdas LSTM, por lo que su interfaz externa debe tener los puertos de entrada y salida necesarios para ello. La Figura 3.14 muestra la interfaz de forma esquematizada. En ella se aprecia que cuenta con una entrada x_t cuyo tamaño es directamente proporcional al número de neuronas, siendo N el número de celdas que hay en la capa. Este bus se desdobla internamente para dividirlo en las diferentes señales de entrada a cada una de las neuronas, como se explicará de forma detallada a continuación. En un primer momento, se pretendió poder crear un número variable de entradas x_t para que la introducción de las señales fuera más intuitiva para el usuario. Sin embargo, esta aproximación no se interpretaba de forma correcta en la síntesis, por lo que resultó preferible realizarlo en forma de bus desdoblado para que pudiera ser sintetizado correctamente por cualquier sintetizador que trabaje con el estándar IEEE 1164 [12]. Nótese que N es un genérico fácilmente modificable. Como puede observarse, ocurre lo mismo con la salida h_t. La salida c_t corresponde al estado celular de la última neurona de la capa. Esto se debe a que no es necesario obtener en la salida los estados celulares de las etapas intermedias, ya que estos, a diferencia de las salidas h_t, no son utilizados por otras capas de la red.

Debido a que la salida y el estado celular de una celda son utilizadas por la siguiente neurona de la capa, es necesario introducir unos valores iniciales que utilice la primera neurona de la capa. Estos son

los introducidos en las entradas `ht_init` y `ct_init`. Por otro lado, la interfaz incluye las entradas de `clk`, `rst` y `enable`, comunes a todas las celdas de la capa.

El valor de cada uno de los pesos sinápticos y los sesgos se asigna individualmente dentro del modelo de la capa, ya que estos difieren para cada neurona.

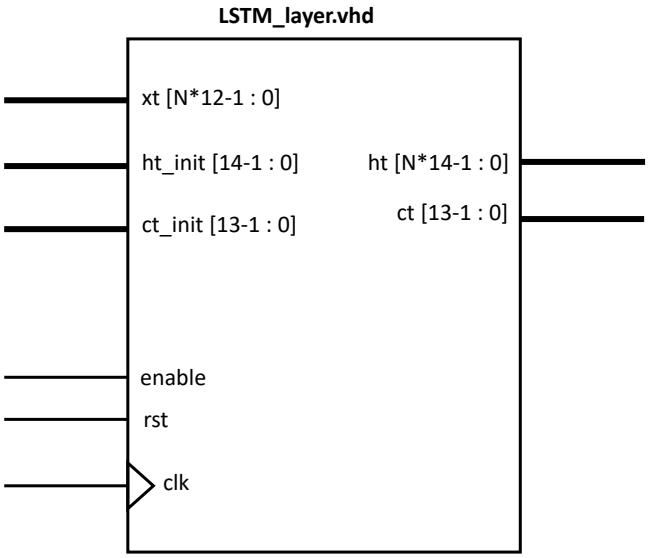


Figura 3.14: Interfaz externa de la capa.

Debido a la dependencia entre las celdas de una misma capa, se requiere sincronizar las salidas de cada celda con la entrada de la siguiente para que el funcionamiento no sea erróneo. En primer lugar, debe destacarse que la frecuencia máxima con la que pueden introducirse las entradas x_t es la frecuencia de reloj de la FPGA, puesto que el diseño es completamente síncrono. Sin embargo, si se conectan las celdas sin añadir más lógica, resulta sencillo comprobar que la frecuencia máxima con que pueden introducirse señales a la celda es de $\frac{1}{6}T_{clk}$, ya que si se introducen con frecuencias mayores, las diferentes unidades no se sincronizarán debido a que la latencia de operación de cada una de ellas es de 6 ciclos de reloj.

Para poder solucionar este inconveniente y aumentar la frecuencia con la que pueden introducirse las señales de entrada, se plantea la introducción de registros de desplazamiento para sincronizar la entrada de cada una de las neuronas con las salidas de la anterior. En la Figura 3.15 se muestra el diseño de la capa.

Los componentes `shift_reg` hacen referencia a los registros de desplazamiento, cuyo número de etapas para conseguir la sincronización es de $5 \times I$, donde I es el número de la celda, considerando que estas comienzan desde la neurona número 0. Esto se aprecia mejor en la Figura 3.15, ya que se puede observar que en la primera celda, lógicamente, no son necesarios registros puesto que es la neurona inicial, mientras que en el resto se necesitan registros cuyo número de etapas viene definido por el factor descrito anteriormente.

El componente `shift_reg` posee dos genéricos: S, que se corresponde con el número de etapas del registro y W, que hace referencia al ancho del bus de datos del mismo. La lógica de las instancias de este componente se genera en función de ambos parámetros.

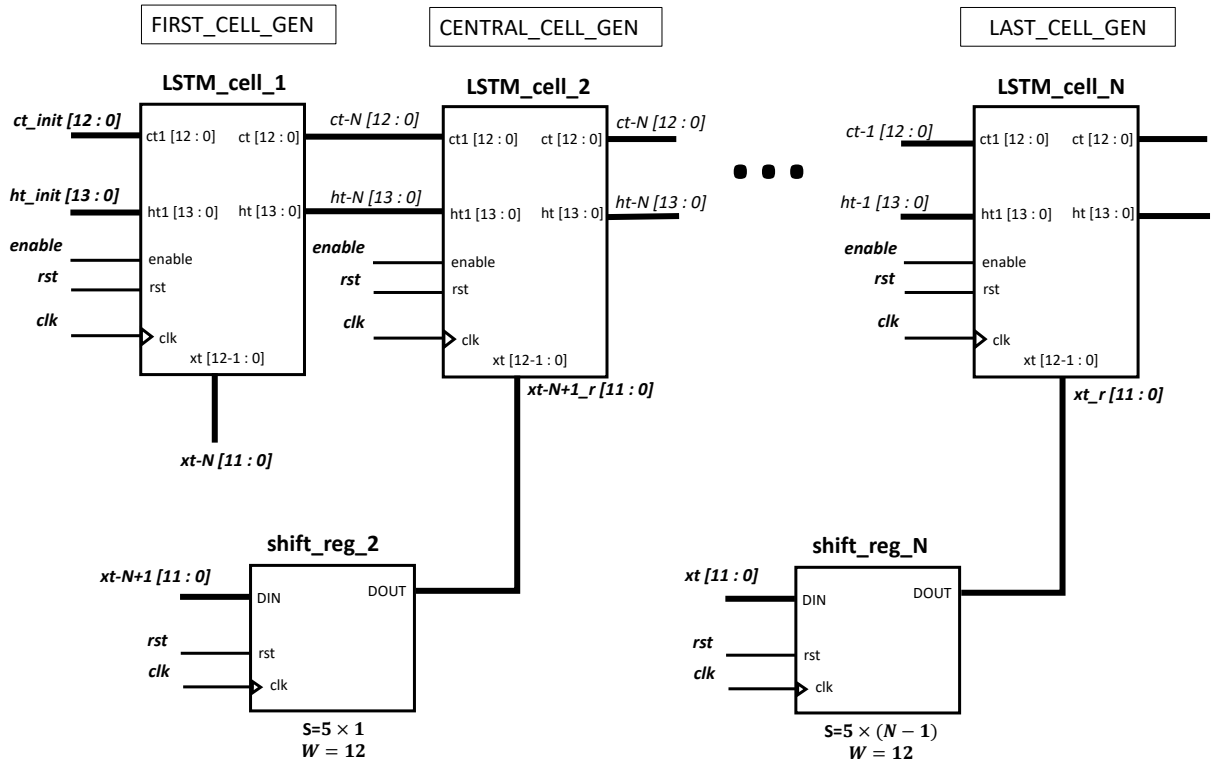


Figura 3.15: Arquitectura para la capa LSTM

3.4.1 Validación funcional del modelo

Para comprobar que el diseño ofrece la funcionalidad deseada, se codifica un testbench para introducir estímulos de prueba. Al igual que en la simulación de la sección 3.3.7, los pesos sinápticos W_x se configuran a $0,5$, los pesos W_h se fijan a $0,125$ y a todos los sesgos b se les aplica valor nulo o 0. En esta simulación se configuran con estos valores para todas las celdas. La capa se configura con 3 celdas LSTM. Durante el tiempo de simulación se hace un reset inicial. Tras el mismo, se le da un valor no nulo a la entrada $x_t - 2$, que corresponde a los 12 bits menos significativos de x_t, como se detalló anteriormente. También se asigna en el mismo instante un valor a las entradas iniciales h_{t_init} y c_{t_init}. En el siguiente ciclo de reloj se asigna un valor a la entrada $x_t - 1$ y en el próximo a $x_t - 0$. Con ello se completaría un ciclo de funcionamiento de la capa.

Los resultados de la simulación se muestran en la Figura 3.16. En ella, puede apreciarse que se obtienen los datos a la salida con la temporización deseada. Se observa en las marcas temporales de los cursores que la diferencia de tiempo desde que se introduce el primer dato a la celda hasta que aparece la salida correspondiente es de 55 ns, esto es, 5 periodos y medio de reloj de la FPGA. Por otro lado, debe notarse que los datos erróneos que aparecen antes de tiempo en las salidas de la segunda y la tercera celda se deben a que, una vez están disponibles los datos de salida de la celda anterior, estas comienzan a operar con ellos aunque no se haya introducido la nueva entrada. Sin embargo, el dato válido se obtiene en el tiempo previsto. Ocurre lo mismo con c_t.

Para comprobar que los resultados son correctos , se crea el script de Matlab `layer_test_script.m` que ejecuta el modelo de la capa de Matlab para estas entradas, teniendo en cuenta en la segunda y tercera celda la realimentación proveniente de las celdas anteriores. Tras su ejecución se comprueba que, tanto

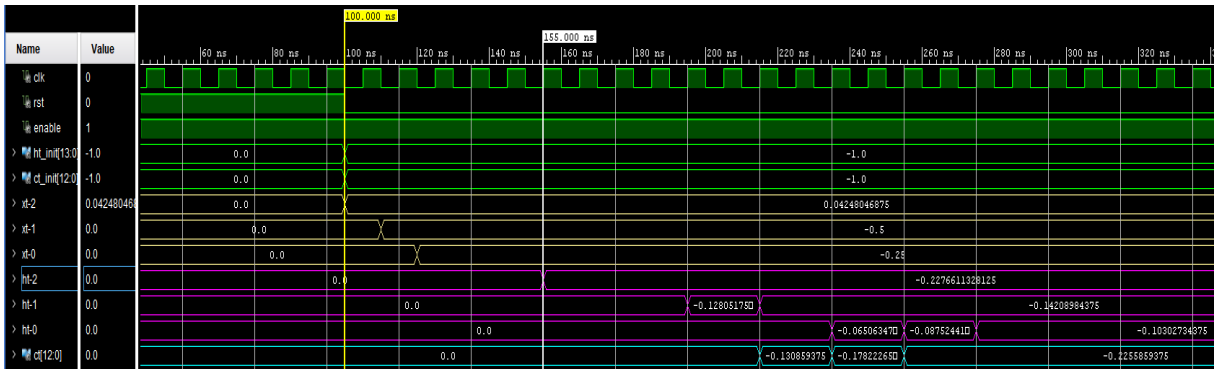


Figura 3.16: Simulación funcional de la capa con 3 neuronas.

los resultados obtenidos en la simulación funcional como la precisión decimal de los mismos son idénticos. Por ello, se comprueba que el modelo ofrece el comportamiento esperado.

Capítulo 4

Resultados experimentales

4.1 Introducción

En el presente capítulo se presentarán los principales resultados del diseño realizado. Para ello, se comprobará si el modelo de la celda cumple con las especificaciones temporales necesarias para asegurar su correcto funcionamiento. Este análisis se realiza exclusivamente con el modelo de la celda puesto que, con el modelo de la red no aportaría nuevas conclusiones, ya que los retardos que pueden producirse en el sistema vienen dados por los elementos internos de las celdas y no por la interconexión entre las mismas. Sin embargo, debe destacarse que el consumo de biestables se incrementa debido al uso de los Flip-Flops utilizados para sincronizar las señales de entrada (véase la Sección 3.4).

El análisis estará fundamentado por los resultados obtenidos en las simulaciones temporales y por los informes de temporización que ofrece la herramienta Vivado. Además, también se realizará el estudio de los recursos de la lógica configurable consumidos por cada uno de los modelos, estando este basado en el informe que proporciona la misma herramienta.

4.2 Análisis estático de tiempos

Cuando se realiza el diseño de un circuito digital, es fundamental comprobar el comportamiento del mismo mediante simulaciones funcionales, como se expone en este caso en la sección 3.3.7 del trabajo. Estas simulaciones validan el funcionamiento de una arquitectura cuyo comportamiento está definido por el código fuente VHDL. Sin embargo, en las simulaciones funcionales se comprueba la respuesta de un circuito completamente ideal, en el que ningún elemento ni camino posee retardos, ni los biestables tienen ningún requisito temporal que deba respetarse.

En el flujo de diseño típico en FPGAs, una vez validado el correcto funcionamiento del diseño, se procede a realizar la síntesis. Este proceso genera un circuito lógico el cual, además, tiene en cuenta las restricciones del diseño, esto es, a qué pin de la FPGA destino se conecta cada puerto del diseño. Este proceso es desarrollado por la herramienta de diseño, en este caso, por Vivado. Tras realizar la síntesis, se realiza mediante la herramienta el proceso de implementación. En este proceso se determina la posición de cada uno de los elementos que componen el circuito en el área de Silicio de la FPGA. Además, se realiza la interconexión de los mismos. La herramienta trata de realizar este proceso de la forma más optimizada posible, intentando evitar cualquier problema temporal que pueda hacer que el sistema falle una vez descargado en la placa. Para comprobar que la temporización se cumple tras la implementación,

Vivado realiza el denominado Análisis Estático de Tiempos (STA) y proporciona un informe que indica si el diseño cumple la temporización indicada en el fichero de restricciones con la frecuencia de reloj indicada en el mismo. En este análisis, se estudian en cada camino con biestables las holguras del diseño en cuanto a los tiempos de *setup*, *hold* y al ancho de pulso. El valor de estas holguras debe ser positivo, lo que implica que no se produce ninguna violación de estos tiempos, ya que, en caso de violar los tiempos de *setup* o de *hold* pueden producirse problemas de metaestabilidad en el circuito que lleven al mismo a comportarse de forma errónea.

La Tabla 4.1 muestra los resultados de este análisis para la frecuencia de reloj interna de la FPGA, es decir, 100 MHz. En este, se muestran los peores casos encontrados en el diseño, es decir, los caminos con mayores retardos.

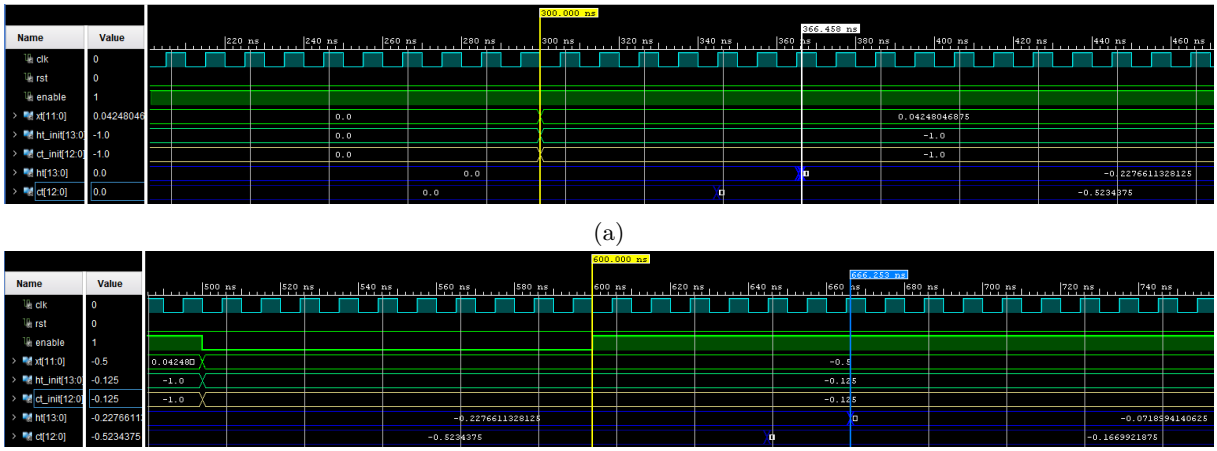
	Setup	**Hold**	**Ancho de pulso**
Peor holgura (ns)	2,824	0,182	4,500

Tabla 4.1: Resultados del STA realizado por Vivado para el modelo de la capa con una única celda.

Como puede apreciarse, en los tres casos se tienen holguras positivas, por lo que las restricciones temporales se cumplen. La peor holgura de *setup* es de 2,824 ns, por lo que es posible que la frecuencia de reloj máxima que pueda introducirse al diseño sea de aproximadamente 133 MHz o un periodo mínimo de 7,5 ns. Esto se deduce porque el margen que se tiene supera los 2,5 ns, aunque no puede asegurarse, ya que depende de como se realice la implementación y debería verificarse también que no se violase el tiempo de *hold*.

4.3 Validación mediante simulación temporal

Además de comprobar que las restricciones impuestas se cumplen mediante el STA, resulta útil realizar una medición de los retardos que se producen con estímulos reales, ya que así puede comprobarse si estos cumplen con los objetivos deseados antes de descargar el diseño a la FPGA. Esto permite que, en caso de que los retardos resulten altos, se planteen soluciones para mejorar la temporización, como la inclusión de más etapas en el pipeline. Estas mediciones pueden realizarse en las simulaciones temporales post-implementación, puesto que estas ya tienen en cuenta todos los factores que afectarán al circuito en condiciones reales. La Figura 4.1 muestra los resultados de esta simulación.



(a)

(b)

Figura 4.1: Simulación temporal de la capa con una única neurona.

En las formas de onda de la simulación, prestando especial atención a los dos cursores puede comprobarse que la operación de la celda es correcta, presentando una latencia de 6 ciclos de reloj, como ocurría en el caso de la simulación funcional del modelo de la celda mostrada en la Figura 3.13. Sin embargo, como puede observarse, aparece un retardo considerable en la salida h_t. Esto puede verse claramente comparando la diferencia temporal medida por los cursores con la que aparece en la simulación funcional mostrada en la Figura 3.13. Esto ocurre de forma análoga en la salida c_t. Este problema temporal es provocado por la suma de los retardos del camino de la señal y el *clock-to-out delay* de los biestables, puesto que ambas son salidas registradas.

Para disminuir el retardo del camino de la señal, pueden utilizarse los registros que integran los bloques de la FPGA que contienen los puertos de entrada y salida, denominados Input Output Blocks (IOB). Estos bloques contienen internamente biestables con retardos minimizados y predecibles. Por ello, se plantea el encapsulado del registro tras el multiplicador del módulo 4, esto es, el que proporciona la salida h_t, en un IOB. La Figura 4.2 muestra los resultados de la simulación con esta configuración. En la misma se puede apreciar la notable reducción del retardo con respecto a la simulación de la Figura 4.1, consiguiéndose que la salida h_t aparezca en el periodo de reloj que corresponde y no se propague a causa del retardo hasta el siguiente, como ocurría en el otro caso. Puesto que ocurre lo mismo en la salida c_t, para solucionarlo se procedería de forma análoga, encapsulando el registro que proporciona esta salida en un IOB. En este caso, sería necesario registrar la señal con dos registros diferentes: el registro encapsulado en los IOBs para proporcionar la salida en el puerto correspondiente y un registro fuera de estos para proporcionar la señal interna, que es utilizada como entrada en el módulo 4 del diseño.

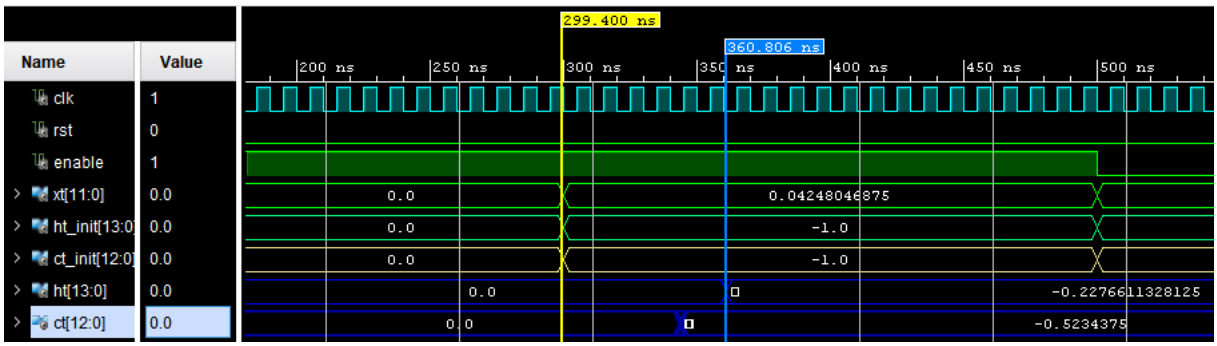


Figura 4.2: Simulación temporal de la capa con una única neurona. Para esta simulación, el biestable que proporciona la salida h_t está encapsulado en un IOB.

Con los resultados de esta simulación se comprueba que la mayor parte del retardo es provocada por los largos caminos hasta los puertos de salida. Sin embargo, también debe tenerse en cuenta que en el modelo existen retardos internos que resulta interesante minimizar. Para ello, se ha estudiado la aplicación de varias soluciones que se detallarán a combinación.

Para minimizar el retardo interno de la celda producido por la lógica combinacional del módulo 1, podría añadirse una etapa de segmentación adicional en el mismo (véase sección 3.3.1). La etapa de segmentación ya existente en el mismo se sitúa entre cada uno de los multiplicadores y sus correspondientes sumadores. Sin embargo, puesto que en este módulo se realizan dos operaciones de suma de forma consecutiva, podría introducirse una etapa de registros entre cada pareja de sumadores. Con ella, la segmentación total del diseño sería de siete etapas. En la Figura 4.3 se muestra el diagrama de la mejora propuesta.

Además, estos retardos también vienen provocados por los registros y multiplexores que aparecen en los cores de la BRAM. La Figura 4.4 muestra el esquemático del core. Como puede apreciarse en la misma, pueden incluirse en estos cores tanto un registro a la salida de las primitivas (o de los latches)

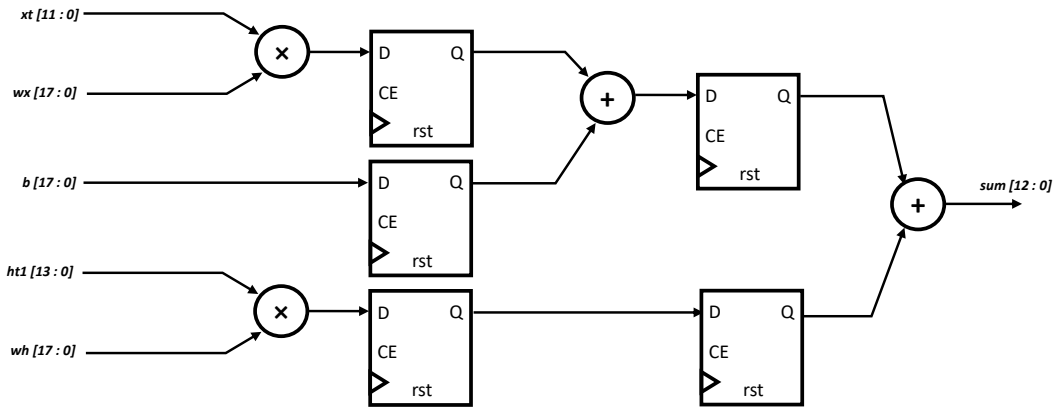


Figura 4.3: Mejora propuesta para el módulo 1. Por claridad, no se han incluido en el diagrama las señales de `clk`, `rst` y `enable`, aunque en el diseño existen y están conectadas a los puertos de reloj, de reset y de CE de los registros, respectivamente

como un registro a la salida del multiplexor. Estos registros permiten disminuir el *clock-to-out delay* del bloque. La inclusión de ambos registros en las funciones de activación, incrementaría en dos etapas la segmentación del diseño, mientras que, si además estos registros se añaden a la BRAM de la compuerta de salida, se añadirían otras dos etapas de segmentación adicionales.

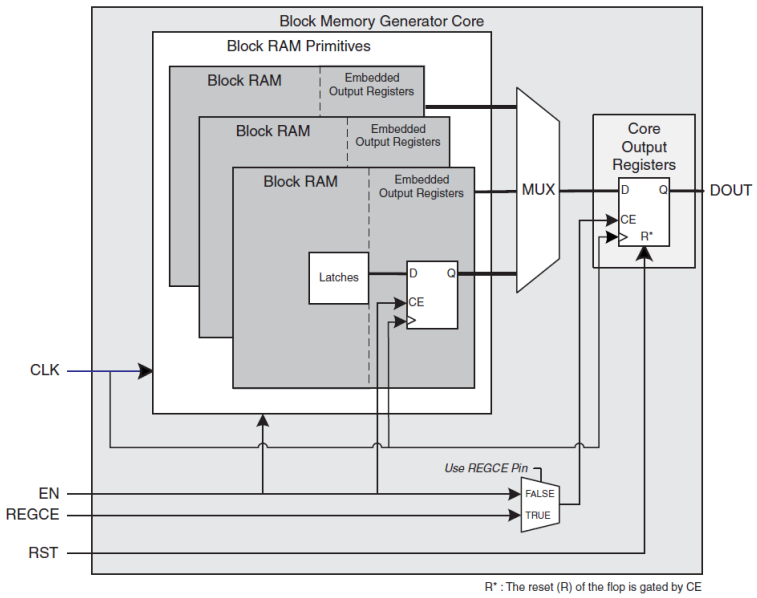


Figura 4.4: Esquemático del core BRAM [13].

Esta solución basada en registrar las salidas de las memorias BRAM, además, presenta una ventaja adicional, ya que todas estas salidas son utilizadas como entradas de los multiplicadores. Los multiplicadores, como se describe en la sección 3.3.1, se implementan mediante bloques DSP48E1. En la guía de uso de los bloques [11] se advierte que es recomendable utilizar un registro de pipeline a la salida de una block RAM antes de conectarla a la entrada de un slice DSP48E1, ya que esto asegurará el mejor rendimiento de las entradas que alimentan al slice. También debe destacarse que en el mismo manual se recomienda utilizar una segmentación de 3 etapas en los diseños basados en multiplicadores con el objetivo de conseguir la máxima eficiencia con estos bloques. De esta forma, con los dos registros a la

salida de las memorias BRAM y la salida ya registrada de los multiplicadores, se conseguirían las tres etapas de segmentación y, con ellas, una disminución de los retardos generados por los multiplicadores.

Con todas las soluciones propuestas se conseguiría una reducción significativa del retardo del sistema, aunque la latencia aumentaría a 11 ciclos de reloj, es decir, casi el doble de la prevista. Por ello, sería interesante probar con diferentes configuraciones de los registros de las funciones de activación y de la BRAM de la compuerta de salida para poder llegar a una solución de compromiso que garantice un retardo reducido y una latencia de operación no demasiado elevada.

4.4 Consumo de recursos

Otra etapa imprescindible en el diseño de arquitecturas hardware para FPGAs es el estudio del consumo de recursos del diseño. Esto es necesario ya que, en ocasiones, los elementos que ofrece una FPGA pueden resultar insuficientes para un diseño determinado. En esta situación debe estudiarse la posibilidad de utilizar otro chip con mejores caacterísticas o de optimizar el consumo de recursos del diseño empleando diferentes técnicas.

El asistente de Vivado genera un resumen de los recursos utilizados tras la implementación, mostrando además el número total de recursos del que dispone la Nexys 4 DDR. Estos se muestran en la Tabla 4.2.

Recurso	Utilización	Disponibles	Utilización (%)
LUT	81	63400	0,13
FF	75	126800	0,06
BRAM	20	135	14,81
DSP	3	240	1,25
BUFG	1	32	3,13

Tabla 4.2: Utilización de recursos de la capa con una única neurona.

En primer lugar, en la tabla se observa un consumo muy reducido de Look-Up Tables (LUTs), de Flip-Flops (FF), así como de bloques DSP, que son los que contienen los multiplicadores. El consumo de BRAM es de 20, que era lo que se esperaba conforme a lo estudiado en el apartado 3.3.2.2. Esto representa el 14,81 % de la totalidad de memorias BRAM que posee el chip. Por ello, se concluye que el número máximo de celdas que pueden implementarse mediante el diseño de capa realizado en esta FPGA es de 6, puesto que para capas con tamaños superiores no se dispondría del suficiente número de memorias BRAM. Este problema podría solucionarse recurriendo a arquitecturas cuyos recursos estén multiplexados en el tiempo. En la sección 5.2, en el párrafo que trata el aprovechamiento de las ventajas del modelo de celda basado en memorias BRAM se detallan algunas soluciones basadas en esta propuesta.

Por otro lado, puesto que el diseño es síncrono con un único dominio de reloj, solo se consume un buffer de reloj global (BUFG) de los 32 disponibles en el chip. Estos buffers son buffers de alto fanout, esto es, pensados para conectarlos a una gran cantidad de elementos. Además, presentan un skew muy bajo, con el objetivo de que el retardo con el que la señal llega a los distintos componentes del modelo sea mínimo.

Capítulo 5

Conclusiones y trabajos futuros

En este capítulo se expondrán las conclusiones que se han extraído durante la realización del trabajo y tras la finalización de este. Además, se discutirán posibles líneas futuras de investigación relacionadas con este trabajo, en cuanto a la mejora del mismo y a su utilización.

5.1 Conclusiones

Este proyecto presenta el diseño de una arquitectura hardware para la implementación de redes neuronales LSTM en FPGA y su validación en una Nexys 4 DDR.

En primer lugar, se lleva a cabo el diseño de la celda o neurona LSTM, que se compone de cinco circuitos digitales o módulos. Además, se optimizan los tamaños de cada una de las partes de las diferentes señales que componen la celda para asegurar que el error de cuantificación de ninguna de ellas no supera el 1,5 % del fondo de escala y para que las mismas puedan ser introducidas en los elementos digitales correspondientes, como por ejemplo, las señales que se introducen en los multiplicadores contenidos en los slices DSP48E1 cuyo tamaño máximo de operación es de 25 $\times$ 18 bits. Los bloques que implementan las funciones matemáticas, esto es, la función sigmoide y la función tangente hiperbólica, se integran en tablas de consulta basadas en memorias Block RAM con los valores de búsqueda precargados, con lo que se consigue mejorar la eficiencia en lo que respecta a ciclos de acceso, ya que únicamente deben realizarse accesos de lectura a las memorias durante la operación normal del sistema. El diseño realizado presenta una estructura segmentada de 6 etapas. El objetivo del pipelining en este modelo es el de conseguir una frecuencia de procesamiento de datos de la celda igual a la frecuencia de reloj. Además, con esta técnica se pretende reducir notablemente el retardo producido por la lógica combinacional presente en el diseño, permitiendo tener correctamente caracterizados los tiempos en que se procesarán las señales. Esto tiene como objetivo facilitar la interacción del circuito con elementos externos de control, es decir, elementos encargados de introducir las señales de entrada y de obtener las señales de salida. Sin embargo, como se explica en detalle en la sección 4.3, se comprueba en la simulación temporal que el retardo de la celda no es despreciable. Por esta causa, en la misma sección se expone una solución que permite reducir el retardo de las señales de salida provocado por los caminos de interconexión. Además, se proponen varias soluciones que tienen como objetivo reducir los retardos que genera la lógica combinacional en los diversos módulos del modelo. .En cuanto al consumo de recursos que se consigue, este es muy reducido, siendo el elemento más limitante las memorias BRAM, cuya utilización se aproxima a un 15 % por celda.

El segundo paso es el diseño de una arquitectura para la implementación de una capa. El modelo propuesto es sencillamente configurable mediante un parámetro, con el que se generan y conectan el

número de celdas deseado por el usuario. En el diseño de la capa se comprueba que, debido al carácter realimentado de las RNN, no es posible aprovechar la estructura segmentada de la celda, puesto que antes de introducir una nueva entrada a cada una de ellas deben esperarse los ciclos de reloj en los que se introducen las entradas al resto de celdas, siendo este número directamente proporcional al número de neuronas de la capa, por lo que no se producirá en ninguna ocasión el llenado del pipeline y el sistema no podrá funcionar en la frecuencia máxima que se consigue con la segmentación. Para que pueda introducirse en la red una entrada por cada ciclo de reloj, debe conseguirse que estas estén sincronizadas temporalmente con las salidas de la celda anterior, ya que con la estructura segmentada la celda tiene una latencia de 6 ciclos de reloj. Para solucionar este problema se realiza la introducción de registros en las entradas de las celdas cuyo número de etapas es directamente proporcional al número de celda. Puesto que este modelo de capa genera tantos modelos de celda como neuronas posea la capa, el número de celdas máximo que pueden introducirse en una capa es muy limitado, ya que el consumo de recursos se multiplica, siendo especialmente determinante el consumo de memorias BRAM. Una ventaja destacable de este modelo de capa es que puede accederse a las salidas de las distintas celdas de forma paralela.

Por todo ello, se concluye que se ha conseguido un diseño de celda con un funcionamiento y una velocidad de operación adecuados. Por otro lado, el modelo propuesto para la capa no es adecuado para aplicaciones de Deep Learning en las que las capas poseen un gran número de neuronas, puesto que no consigue un buen aprovechamiento del área de Silicio de la FPGA debido a que utiliza un número excesivo de recursos, específicamente de memorias BRAM.

5.2 Trabajos futuros

El objetivo de este trabajo es el de actuar como base para futuros proyectos de implementación de redes neuronales LSTM en FPGAs. Estos pueden ser tanto proyectos que se centren en intentar mejorar los modelos de celda y capa propuestos en este trabajo, como proyectos que utilicen el modelo de celda existente y propongan un modelo de capa optimizado. También pueden realizarse trabajos que comparen la eficiencia en diferentes aspectos de estos diseños con otros diseños existentes.

En cuanto a la mejora, se mencionan a continuación las líneas futuras que pueden llevarse a cabo:

- Trabajos de mejora del modelo de la celda, en los que se pueda mejorar la temporización, disminuyendo el retardo, e incluso si es posible la latencia, para lo que puede resultar interesante estudiar la efectividad de las mejoras propuestas en la sección 4.3. Puede mejorarse, además, el consumo de recursos, estudiando si alternativas interesantes a las tablas de búsqueda basadas en memorias BRAM para la implementación de funciones matemáticas. También pueden realizarse modificaciones que mejoren otros aspectos del funcionamiento. Por ejemplo, como se propone en el trabajo de He *et al.* [14], la utilización de entradas matriciales y la optimización de las operaciones con estas mediante matrices sistólicas.

- Trabajos de mejora del modelo de la capa, que consigan optimizar el consumo de recursos debido a los registros de la sincronización entre celdas, buscando soluciones alternativas basadas por ejemplo en memorias First In, First Out (FIFO).

Por otro lado, la realización de un modelo de capa optimizado que pueda aprovechar las ventajas del modelo de celda basado en memorias BRAM puede ser muy interesante. Para ello, podría realizarse un modelo que utilice una única celda en las sucesivas operaciones, puesto que como se menciona en diversas partes del trabajo, la operación de este tipo de redes es secuencial. Este modelo podría controlar el flujo

de datos de la celda mediante algunos bloques digitales, como por ejemplo, memorias FIFO. Otra opción interesante que podría estudiarse es utilizar una FPGA que permita integrar o que integre en su área de silicio un sistema de procesamiento basado en microprocesador para el control del flujo de datos. Por ejemplo, en el trabajo de Chang *et al.* [15] se propone un módulo que utiliza puertos de Acceso Directo a Memoria (DMA) para controlar las transferencias de datos y asegurar la sincronización de los mismos.

Finalmente, resulta muy útil disponer de métricas que midan la bondad de los diseños y que los comparen con respecto a otros, con el objetivo de estudiar cuáles son los mejores. Por ello, los trabajos que realicen estas comparaciones pueden ser de mucha utilidad. Sería interesante comparar el modelo de la celda propuesto en este trabajo con modelos generados a partir de síntesis de alto nivel (HLS), como por ejemplo el desarrollado en el trabajo de Rao [16].

Capítulo 6

Presupuesto

En las posteriores secciones se presenta un estudio del presupuesto derivado del desarrollo del trabajo. Este análisis incluye los costes correspondientes al material físico utilizado, los costes asociados a las herramientas software que han sido necesarias y el precio de la mano de obra. Puesto que el trabajo fue realizado desde noviembre del año 2021 hasta junio del año 2022, el tiempo de uso tenido en cuenta para los cálculos para todas las herramientas es de 8 meses. Los costes asociados al material y el software que aparecen en las tablas ya incluyen los impuestos aplicables de España. Los costes de mano de obra expuestos en las tablas también tienen en consideración las tasas impositivas aplicables a las empresas en España.

6.1 Coste de los medios físicos y de las herramientas software

	CONCEPTO	PRECIO	AMORTIZACIÓN	USO	COSTE
Materiales	Ordenador portátil	700 €	4 años	8 meses	117 €
Software	Office 365	69 €	1 año	8 meses	46 €
	Sublime Text	0 €	N/A	8 meses	0 €
	TexMaker	0 €	N/A	8 meses	0 €
	Vivado WebPack	0 €	1 año	8 meses	0 €
	Windows 10 Pro	259 €	N/A	8 meses	259 €
COSTE TOTAL					422 €

Tabla 6.1: Deglose de los costes materiales y software.

6.2 Coste de la mano de obra

CONCEPTO	PRECIO/ HORA	HORAS	COSTE
Desarrollo de modelo software	25 €	75	1875 €
Diseño hardware	30 €	170	5100 €
Redacción de la memoria	20 €	70	1400 €
COSTE TOTAL			8375 €

Tabla 6.2: Coste de mano de obra.

6.3 Coste total del proyecto

CONCEPTO	COSTE
Material y herramientas software	422 €
Mano de obra	8375 €
COSTE TOTAL	8797 €

Tabla 6.3: Coste total del proyecto.

El presupuesto del presente proyecto, presentado en el mes de julio de 2021, asciende a la cantidad de OCHO MIL SETECIENTOS NOVENTA Y SIETE EUROS, estando consideradas en el mismo las tasas impositivas correspondientes de España.

Bibliografía

[1] B. Martín-del Brío and A. Sanz, *Redes neuronales y sistemas borrosos*, 01 2006.

[2] N. D. B. of Ordnance, "Computer Mark 1 and Mods," June 29, 1945.

[3] N. M. of American History, ""Tortoise" Mobile Robot," accessed: June 12, 2022. [Online]. Available: https://americanhistory.si.edu/collections/search/object/nmah_879329

[4] D. Crevier, *AI: The Tumultuous History of the Search for Artificial Intelligence*, 01 1993.

[5] M. Minsky and S. A. Papert, *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, 09 2017. [Online]. Available: https://doi.org/10.7551/mitpress/11301.001.0001

[6] A. Pouliakis, E. Karakitsou, N. Margari, P. Bountris, M. Haritou, J. Panayiotides, D. Koutsouris, and P. Karakitsos, "Artificial Neural Networks as Decision Support Tools in Cytopathology: Past Present and Future," *Biomed Eng Comput Biol*, vol. 2016, pp. 1–18, 02 2016.

[7] C. Maxfield, *The Design Warrior's Guide to FPGAs: Devices, Tools and Flows*, 1st ed. USA: Newnes, 2004.

[8] S. M. Trimberger, "Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 318–331, 2015.

[9] "Digilent Pmod Interface Specification," October 28, 2020. [Online]. Available: https://digilent.com/reference/_media/reference/pmod/pmod-interface-specification-1_3_1.pdf

[10] "7 Series FPGAs Data Sheet: Overview," September 8, 2020. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documents/data_sheets/ds180_7Series_Overview.pdf

[11] "7 Series DSP48E1 Slice User Guide," March 27, 2018, accessed: June 11, 2022. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug479_7Series_DSP48E1

[12] "IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Std_logic_1164)," pp. 1–24, 1993, iD: 1.

[13] "Block Memory Generator v8.4.Â LogiCORE IP Product Guide," August 6, 2021, accessed: June 1, 2022. [Online]. Available: https://docs.xilinx.com/v/u/en-US/pg058-blk-mem-gen

[14] D. He, J. He, J. Liu, J. Yang, Q. Yan, and Y. Yang, "An FPGA-Based LSTM Acceleration Engine for Deep Learning Frameworks," *Electronics*, vol. 10, no. 6, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/6/681

[15] A. X. M. Chang, B. Martini, and E. Culurciello, "Recurrent Neural Networks Hardware Implementation on FPGA," 2015. [Online]. Available: https://arxiv.org/abs/1511.05552

[16] R. P. Rao, “Implementation of Long Short-Term Memory Neural Networks in High-Level Synthesis Targeting FPGAs,” 2020.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá