Trabajo de Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Diseño de una red neuronal en VHDL

Autora: Sandra Muñoz Lara
Tutor: Alejandro del Real Torres

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo de Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Diseño de una red neuronal en VHDL

Autora:
Sandra Muñoz Lara

Tutor:
Alejandro del Real Torres
Profesor Contratado Doctor

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo de Fin de Grado: Diseño de una red neuronal en VHDL

Autor: Sandra Muñoz Lara

Tutor: Alejandro del Real Torres

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia

A mis amigos

Agradecimientos

Me gustaría aprovechar estas líneas para agredecer a cada una de las personas que han aparecido en mi vida y me han ayudado a ser la persona que hoy escribe estas lineas.

En primer lugar, gracias a mi familia. Por el apoyo constante, la motivación y la paciencia necesarias en cada momento y sobretodo por confiar en mi mucho antes de que yo lo hiciera.

En segundo lugar, gracias a mis amigos, tanto a los de dentro como los de fuera de la ETSI. Por ser compañeros de batalla, confidentes y un hombro en el que apoyarse cuando las fuerzas flaquean, en definitiva por hacer de esta etapa universitaria sin duda una de las mejores de mi vida.

En último lugar, gracias a todos los profesores que me han acompañado durante estos años de carrera por el conocimiento compartido y las lecciones aprendidas, pero sobretodo por despertar en mí el interés en el mundo de la tecnología y a dia de hoy poder decir con orgullo que soy ingeniera.

Sandra Muñoz Lara

Escuela Técnica Superior de Ingeniería

Sevilla, 2020

Resumen

En el presente documento se detalla el proceso llevado a cabo para el diseño en VHDL de una red neuronal sencilla, además del estudio de los conceptos principales asociados tanto a las redes neuronales como a su relación con las FPGA.

En primer lugar se hace una introducción sobre el concepto de inteligencia artificial, la presencia de la misma en la actualidad y como la naturaleza de la misma ha motivado este proyecto.

A continuación, se tratan los dos principales temas de este proyecto: las redes neuronales y el diseño en VHDL con vista a su posterior implementación en una FPGA.

Por un lado, se realiza un estudio sobre el concepto de red neuronal; incidiendo en su definición, sus principales características y clasificaciones, el concepto de neurona y los pasos a seguir para realizar el diseño de una red neuronal.

Por otro lado, se realiza un estudio sobre el uso de las FPGA en el mercado actual y como estas suponen una vía de avance en el mismo.

Por último, se muestra el proceso realizado para la implementación de la red neuronal en VHD. En este apartado se detallan tanto el diseño de la red realizado en MATLAB como la implementación EN VHDL y la simulación asociada.

Abstract

The present document details the process carried out for the VHDL design of a simple red neuronal, in addition to the study of the main concepts associated with neural networks and their relationship with FPGAs.

Firstly, an introduction is made about the concept of artificial intelligence, its presence nowadays and how its nature has motivated this project.

Next, the two main topics of this project are covered: neural networks and VHDL design with a view to its subsequent implementation in an FPGA.

On the one hand, a study is carried out on the concept of neuronal network; focusing on its definition, its main characteristics and classifications, the concept of neuron and the design process of a neural network.

On the other hand, a study is carried out on the use of FPGAs in the current market and how they could be one possible future development path.

Finally, the process carried out for the implementation of the red neuronal in VHD is shown. This section details both the network design carried out in MATLAB as well as the VHDL implementation and its associated simulation.

Índice

Índice de Tablas

Índice de Figuras

Notación

RNA (ANN)	Red Neuronal Artificial (Atificial Neuronal Network)
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
VHDL	Versatile Hardware Description Language
LUT	Look-Up Table
NNT	Neural Network Toolbox
VLSI	Very Large Scale Integration

1 INTRODUCCIÓN

En este primer capítulo se muestra una breve introducción del proyecto, exponiendo contexto en el que se encuentra y el objetivo de este.

1.1 Contexto y objetivo

La presencia de la inteligencia artificial (AI) en nuestro día a día y el auge de esta durante los últimos años nos lleva a concluir que esta es considerada una de las revoluciones tecnológicas más importantes de nuestra era.

El estudio del ser humano y de su capacidad para razonar ha derivado en el desarrollo de un campo conocido como la inteligencia artificial, en el que se estudia la capacidad de las máquinas para aprender de la experiencia y actuar imitando el comportamiento humano y/o racionalmente.

[1] Rusell y Norvig en su libro *"Inteligencia Artificial: Un enfoque moderno"*, describieron 4 posibles aproximaciones en el desarrollo de la inteligencia artificial basándose en si la IA se basaba en como piensan los humanos o si se considera una racionalidad ideal y de si esta IA se basaba en el razonamiento o en el comportamiento

Tabla 1: Tipos de Sistemas basados en el modelo de 'Inteligencia Artificial: Un enfoque Moderno'

	Basado en humanos	**Basado en racionalidad ideal**
Basado en razonamiento	Sistemas que piensan como humanos	Sistemas que piensan racionalmente
Basado en comportamiento	Sistemas que actúan como humanos	Sistemas que actúan racionalmente

1. ***Sistemas que piensan como humanos***: Buscan modelar el pensamiento humano generando sistemas que reconstruyan el mismo.

2. ***Sistemas que actúan como humanos***: Buscan modelar el comportamiento humano de manera que los sistemas puedan replicar actividades como lo hacen los humanos. Esta aproximación incluye las tareas que pueden ser modeladas por redes neuronales las cuales imitan el comportamiento del cerebro humano y en concreto de las neuronas biológicas

3. ***Sistemas que piensan racionalmente***: Buscan modelar un pensamiento basado en leyes racionales y lógica formal.

4. ***Sistemas que actúan racionalmente***: Buscan modelar un comportamiento basado también las leyes racionales y la lógica formal.

El aprendizaje automático basado en redes neuronales ha adquirido mucha popularidad en múltiples ámbitos debido a la gran variedad de utilidades que nos presenta esta herramienta. Las tareas más comunes para su uso contemplan la predicción, clasificación y reconocimiento de patrones entre otros siendo utilizados en infinidad de campos como el control de robots, la visión artificial y el procesamiento de lenguaje entre otros.

Debido a que la naturaleza de las redes neuronales es un procesamiento en paralelo, se decidió hacer uso de un tipo de dispositivo que fuera capaz de simular esta de la manera más realista posible. En este punto es donde aparece el uso de las FPGA debido a su naturaleza de procesamiento también en paralelo.

De forma similar a las redes neuronales, el uso de las FPGA ha alcanzado cada vez más popularidad en la industria debido a las diferentes ventajas que estas presentan en términos de capacidad, funcionalidad y sobre todo de flexibilidad.

El principal objetivo de este proyecto es el diseño de una implementación en VHDL para una FPGA de una red neuronal previamente diseñada haciendo uso del NNT (Neural Network Toolbox) de Matlab.

2 REDES NEURONALES

En este capítulo se tratan algunos conceptos fundamentales de las redes neuronales, centrándonos en su definición , sus características principales y sus tipos.

2.1. Definición de una red neuronal

Definimos una red neuronal artificial como un procesador distribuido en paralelo que basado en unidades más simples denominadas neuronas, es capaz de predecir la salida de un sistema con cierto nivel de fiabilidad según unas entradas determinadas.

La creación de las redes neuronales artificiales nace en base a la idea de emular la resolución de problemas tal y como lo haría un cerebro humano.

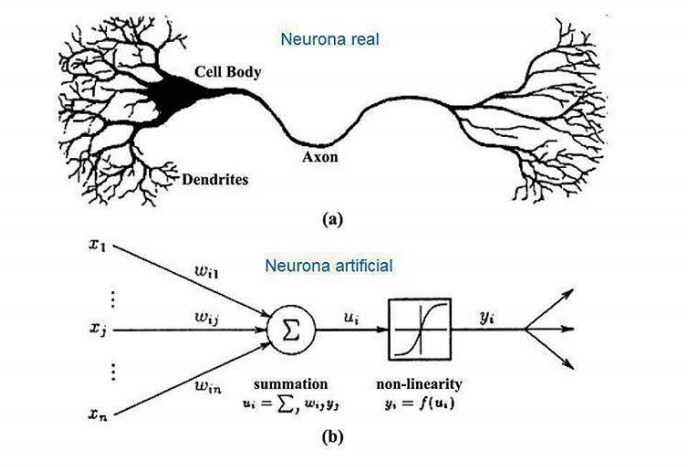

Ilustración 1: Comparación de neurona biológica y neurona artificial

De esta manera se define una estructura matemática que modela el comportamiento biológico de las neuronas y la estructura de organización de estas en el cerebro

El cerebro puede considerarse un sistema altamente complejo, donde se calcula que hay aproximadamente 100 mil millones (10111011) neuronas en la corteza cerebral (humana) y que forman un entramado de más de 500 billones de conexiones neuronales (una neurona puede llegar a tener 100 mil conexiones, aunque la media se sitúa entre 5000 y 10000 conexiones) [2]

Esto nos lleva a la conclusión de que el comportamiento de una red neuronal humana siempre resultara mucho más abstracta de lo que una red neurona artificial pueda suponer.

2.2 La neurona

Las redes neuronales artificiales pretenden simular el comportamiento biológico de las neuronas y en cómo se organizan formando la estructura del cerebro, determinando las siguientes analogías con este tras observar el proceso biológico que se da en la neurona y adoptando un modelo simplificado de este.

De esta manera el sistema completo presenta la siguiente estructura

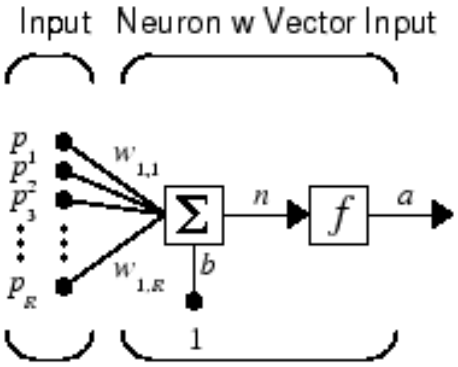

Ilustración 2: Esquema matemático de una neurona artificial

Cada señal de entrada, la cual puede ser tanto un vector como un valor simple, esta modulada según una ganancia o peso, el cual puede ser positivo o negativo. Dependiendo del valor y del signo de este peso la neurona puede ser excitada o inhibida.

Las entradas de nuestra red artificial se corresponden con su modelo biológico con las señales que entran a las neuronas y que pueden provenir de otras neuronas o de señales externas, las cuales son capturadas por las dendritas. Y la modulación que sufren estas con la intensidad que conecta la sinapsis entre dos neuronas

Una vez moduladas dichas señales de entrada con los pesos, estas pasaran a través de una función de transferencia la cual proporciona el estado de activación a la neurona y limita el rango de valores que puede tomar la variable de salida de la neurona.

La función de activación corresponderá directamente con la función umbral que necesitan las neuronas biológicamente necesitan para activarse y transmitir el impulso nervioso a la siguiente neurona.

A continuación, se hace un estudio de las diferentes funciones de activación debido a que la implementación realizada estas juegan un papel muy importante a la hora de la definición de la red neuronal.

2.2.1 Funciones de activación

En este apartado se van a detallar las funciones de activación más utilizadas a la hora del diseño de una RNA: la función escalón, lineal y sigmoide.

2.2.1.1 Función escalón

Esta función se asemeja biológicamente con el comportamiento de una neurona, ya que, dependiendo del nivel

de excitación, en nuestro caso del valor introducido, esta se activa o no.

Se encuentra definida por la siguiente ecuación la cual nos muestra una separación discreta de los valores y representado por la Ilustración 3.

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

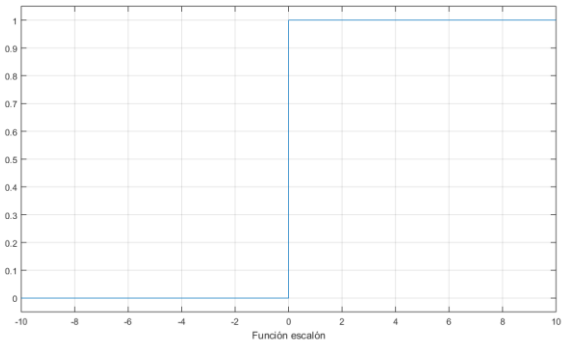

Ilustración 3:Representación de una función escalón

El problema de esta función es que para la separación lineal en categorías se tienen que usar un gran numero de neuronas ya que el rango de salida esta limitado entre dos únicos valores.

2.2.1.2 Función rampa

Esta función se encuentra definida por una ecuación de tipo lineal como la que se presenta en la siguiente ecuación y representada por la Ilustración 4.

$$f(x) = x$$

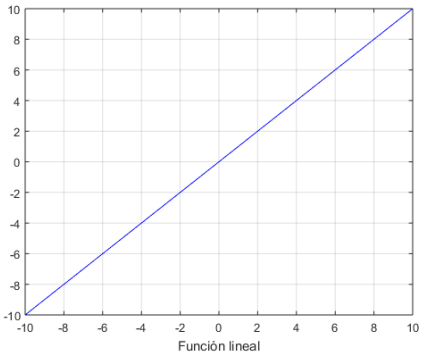

Ilustración 4:Representación de una función lineal

Esta función refleja el incremento de potencial de activación cuando la entrada se incrementa, aunque presenta los mismos problemas que la función escalón. El problema de esta función es que tiene una convergencia inestable, es decir, que a la larga tiende a incrementarse sin límite.

Esta función de transferencia suele ser utilizada en la capa de salida de las redes multicapa debido a que su carácter discreto nos proporciona la capacidad de aproximación de los resultados de salida.

La función de activación en la capa de salida dependerá de la tarea de dicha capa. La función de activación lineal suele ser utilizada en tareas de predicción y aproximación mientras que la función sigmoidal suele ser utilizada en tareas de clasificación.[3]

2.2.1.3 Función sigmoidal

La función sigmoide es la que presenta un modelo más realista ya que en el momento que llega la entrada esta va incrementando su frecuencia de activación gradualmente hasta que llega a ser una asíntota cuando la frecuencia es del 100%. Esta conducta es la que representa un comportamiento más realista ya que las neuronas no pueden activarse más rápido que una cierta tasa.

Esta función se encuentra definida por la ecuación que se presenta en la siguiente ecuación y representada por la ilustración 5

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

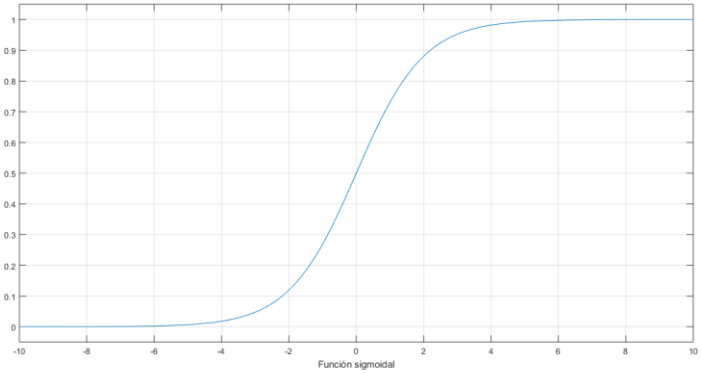

Ilustración 5: Representación de una función sigmoidal

Este tipo de función de activación suele ser utilizada en las capas ocultas de una red neuronal debido a que es diferenciable y a que su carácter no lineal permite que la red neuronal aprenda relaciones no lineales entre la entrada y la salida proporcionada [4].

2.3 Características principales de una RNA

Las características principales y más relevantes de una red neuronal artificial acerca de la integración de estas con el diseño para una FPGA son las siguientes [5]:

Aprendizaje adaptativo:

Las *RNA* pueden cambiar su comportamiento como respuesta al entorno. Dado un conjunto de entradas estas se ajustan para generar una salida valida respecto a esas entradas

Inmunidad al ruido*:*

Las RNA entrenadas son capaces de generalizar la esencia de un conjunto de entradas. De esta manera puede dar respuestas válidas frente a un conjunto de datos de entrada que presente variaciones debido al ruido o distorsión de las mismas.

Operación en tiempo real:

El procesamiento de las RNA es realizado en paralelo, esto es potencialmente aprovechable para realizar funciones mucho más complejas que si el procesado fuera lineal.

Facilidad de uso e implementación:

Debido a la propia naturaleza de autoaprendizaje de las RNA es posible la utilización por parte del usuario requiriendo únicamente un conocimiento a la hora de seleccionar correctamente los datos de entrada, lo cual facilita en gran porcentaje la utilización de las mismas en multitud de aplicaciones.

Además, debido a la naturaleza de procesamiento paralelo de las RNA resultan en estructuras que son capaces de realizar tareas con cierta tasa de rapidez. Esto permite que sean adaptadas a una integración a grande escala o VLSI (Very Large Scale Integration).[11]

Esta tecnología nace de una posibilidad de diseño que nos permite integrar decenas de miles de puertas en un único circuito integrado permitiendo así un alto grado de eficiencia en el diseño de nuestras aplicaciones.[6]

2.4 Clasificaciones principales de una RNA

Aunque existen diferentes tipos de clasificaciones dentro de las que agrupar las redes neuronales se van a tratar dos clasificaciones principales en función de dos de sus características más importantes: la estructura de la red y el algoritmo de aprendizaje utilizado.

Clasificación según la estructura de la red:

Podemos clasificar las diferentes redes neuronales según su número de capas, el tipo y la distribución de las mismas.

Redes monocapa:

Las redes mono capa o de capa simple son aquellas que están formadas por un conjunto de perceptrones simples, es decir aquellas que poseen una capa de neuronas y que nos ofrece únicamente un conjunto de salidas para un conjunto de entradas dado.

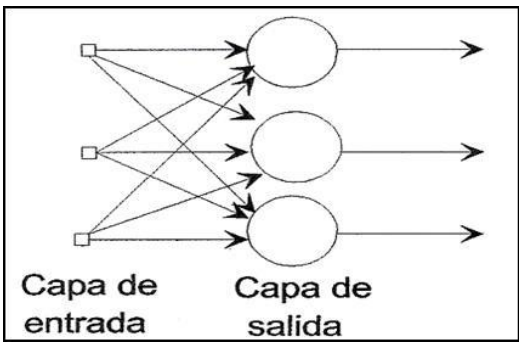

Ilustración 6:Estructura de una red monocapa

Este es el tipo de red neuronal más sencilla que existe y por ello presenta sus limitaciones a nivel de computación. Se corresponde con una lógica binaria de una puerta NAND por lo que, aunque a partir de esta se puedan crear cualquier otro tipo de función booleana, por ella misma solo nos permite una separación lineal de los resultados.

Redes multicapa**:**

Las redes multicapa cuentan con un número n de neuronas unidas en cascada entre sí por pesos. De esta manera además de contar con las capas de entrada y salida con la que cuenta la red de capa simple, la red contiene una o más capas de neuronas ocultas.

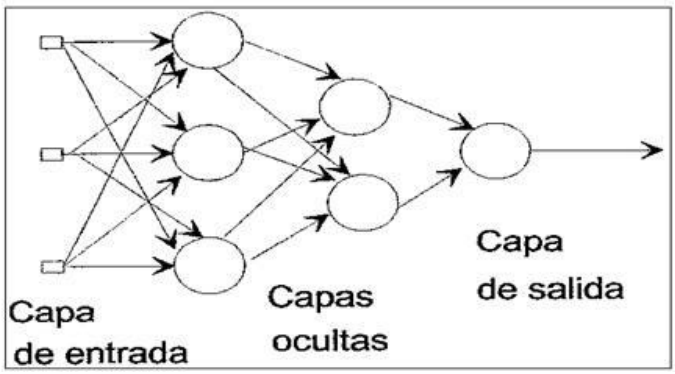

Ilustración 7: Estructura de una red multicapa

Se caracterizan principalmente por solucionar la limitación que ofrecen las redes de capa simple, son capaces de aportar una separación no lineal de los resultados aproximando funciones de tipo continuo de un grupo de variables de entrada y salida.

Los diferentes tipos de conexiones existentes entre las redes nos permiten realizar una nueva clasificación en función de las mismas: redes feedforward y redes feedforwar/feedback.

Las redes feedforward son aquellas cuyas conexiones son únicamente hacia delante es decir ninguna salida de una neurona tiene como origen una entrada de una neurona de una capa anterior o del mismo nivel

Las redes feedforward/feedback son aquellas cuyas conexiones son tanto hacia delante como hacia atrás, es decir las salidas de las neuronas pueden tener como origen las entradas de neuronas tanto de capas anteriores como del mismo nivel.

Clasificación según su algoritmo de aprendizaje

Podemos clasificar las diferentes redes dependiendo del método empleado para su entrenamiento en redes de entrenamiento supervisado y redes de entrenamiento no supervisado.

El aprendizaje es el proceso por el cual una red neuronal modifica sus pesos en respuesta a una información de entrada. Durante este proceso los pesos de las conexiones de la red se modifican y se mantienen estables cuando la red ya ha aprendido.[7]

Aprendizaje supervisado

Este tipo de aprendizaje se realiza en función a unos patrones de entrenamiento que le son proporcionado a la red y se caracteriza porque el entrenamiento está controlado de manera externa a la red.

A la red se le proporciona la salida que debería de generar en función de la entrada proporcionada y si la salida de la red no coincide con esta se ajustan los pesos con el fin de aproximar la salida obtenida a la real.

La tres principales formas de realizar este aprendizaje son: por corrección de error, por refuerzo y estocásticamente.

En el aprendizaje por corrección de error se ajustan los pesos de la red en función del error entre los valores deseados y los obtenidos.

En el aprendizaje por refuerzo se ajustan los pesos de una manera más lenta que en el tipo de aprendizaje anterior. El aprendizaje se basa en que tras la salida proporcionada por la red el supervisor externo indique si el valor obtenido por la red es válido o no.

En el aprendizaje estocástico se ajustan los pesos realizando cambios aleatorios en los valores de estos y observando los efectos de este cambio en el valor de salida. Si el cambio en el valor de salida se ajusta más al deseado que el valor anterior a realizar este cambio se acepta dicho cambio si no se descarta.

Aprendizaje no supervisado

Este tipo de aprendizaje se caracteriza por no necesitar de unos patrones de aprendizaje que le indique si la salida es correcta o no en función de los cuales modificar los valores de los pesos de la red.

Este aprendizaje se basa en las similitudes y correlaciones que presentan los datos de entrada con los que se entrena la red y en base a esto clasifica las salidas en base a diferentes patrones establecidos según características similares.

Las dos principales formas de realizar este aprendizaje son: el aprendizaje hebbiano y el aprendizaje competitivo y comparativo.

El aprendizaje hebbiano permite extraer las características principales del conjunto de entradas que se le proporciona. Para ello varias neuronas participan conjuntamente en la identificación del patrón de entrada de la

siguiente forma: si dos neuronas se activan a la vez (activo o inactivo) el valor del peso de conexión entre ellas aumenta.

El aprendizaje competitivo y comparativo por su parte se centra en clasificar los diferentes patrones de entrada. Para ello varias neuronas compiten entre ellas con el fin de identificar el patrón de entrada de la siguiente forma: tras presentar una entrada determinada se busca que se active aquella o aquel conjunto de ellas cuyos pesos proporcionen una salida que se asemeje más al patrón de entrada.

2.5 Pasos de diseño de una RNA

Se puede definir el proceso de diseño de la red neuronal en siete pasos principales:

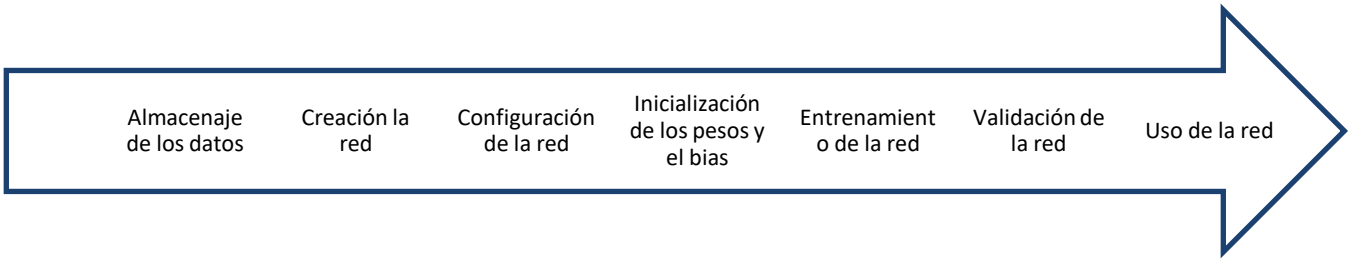

Ilustración 8: Pasos de diseño de una red neuronal

Almacenaje de los datos

El primer paso a la hora del diseño de una red neuronal es la recolección de los datos con los que se va a entrenar la red.

Es importante la correcta recolección de estos datos de manera que la información que se incluya sea relevante y de utilidad, pues la red será tan precisa como lo sean los datos con la cual la entrenemos.

Para ello es útil realizar previamente un estudio sobre las señales de entrada.

En este estudio se obtendrían las principales características que resultan más significativas de manera que podamos proveer a la red un conjunto de características relevantes para su clasificación y se reduzca la redundancia de los datos [8].

Creación de la red

En este punto se procede a la creación de la red neuronal, este proceso dependerá así del tipo de lenguaje y/o software utilizado.

Configuración de la red

Tras la creación de la red se procede a la configuración de la misma de manera que podamos especificar las características de la red que se quiere diseñar

Inicialización de los pesos y el bias

De manera previa a realizar el entrenamiento de la red es necesario realizar la inicialización de los pesos de la red neuronal los cuales posteriormente se van a reajustar durante el entrenamiento.

Entrenamiento de la red

Como previamente se ha visto este punto permite ajustar el valor de los pesos de manera que la red pueda predecir el valor de salida en función de los valores de entrada.

Este proceso requiere de un conjunto de ejemplos de la red con las entradas y sus salidas correspondientes y de la definición de un tipo de entrenamiento. Dependiendo del tipo de entrenamiento y el algoritmo seleccionado este proceso dependerá la eficiencia de la red y el tiempo de entrenamiento.

Validación de la red

Mientras se esta produciendo el entrenamiento de la red se produce una validación de la misma de manera que el entrenamiento se da por concluido cuando los resultados entre la comparación de la entrada y la salida de la red se consideran aceptables

En la implementación realizada este proceso se da mientras se ejecuta la función train que dirige el entrenamiento como anteriormente hemos comentado.

Uso de la red

Como ultimo paso en el proceso de diseño de la red se encuentra el uso de la misma para el uso para el cual esta ha sido diseñada.

En la implementación realizada en el presente documento se hará uso del NNT de Matlab para el diseño de la red neuronal, una vez esta red sea diseñada se procederá a la implementación de dicha red sobre lenguaje VHDL

3 FPGA'S Y REDES NEURONALES

Las FPGAs (Field Programmable Gate Arrays) son dispositivos electrónicos reprogramables que mediante la interconexión de diferentes elementos son capaces de describir un circuito digital. A diferencia de la mayoría de los procesadores que nos encontramos en el mercado, estos trabajan de forma paralela, de manera que se ajustan perfectamente con el modo de funcionamiento de una red neuronal.

La composición de estos dispositivos cuenta con diferentes elementos como: puertas lógicas, biestables, cables y puertos de entrada y salida , los cuales son conectados y organizados entre si tras la carga de un bitstream , un archivo generado a partir de la descripción realizada en un lenguaje especifico HDL como son VHDL o Verilog el cual agrupa los bits de configuración de la FPGA y a través del cual se permite la reconfiguración de la misma

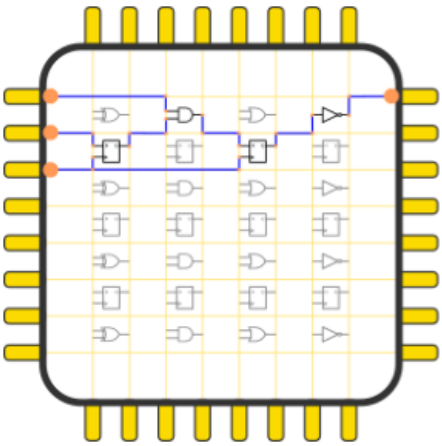

Ilustración 9: Representación de la interconexión dentro de de una FPGA

Todo esto nos permite la implementación de un circuito electrónico en una FPGA, qué aunque si de manera más ineficiente que los ASIC, los cuales están desarrollados para una aplicación específica, cuenta con la ventaja de que son reprogramables, lo que se traduce en un coste y tiempo de desarrollo inferior a estos.

En la actualidad el mercado de las FPGA se encuentra en auge ya que, debido al estancamiento de la ley de Moore, se buscan nuevas soluciones que permitan seguir mejorando los sistemas de procesamiento actuales y dando solución a las nuevos retos que se nos plantean.

A diferencia de los procesadores, los cuales constan con un conjunto fijo de instrucciones, esta nos ofrece una mayor libertad a la hora de la aplicación a implementar, ya que esta únicamente estará limitada por la cantidad de bloques lógicos conectados entre sí que se encuentren en ella.

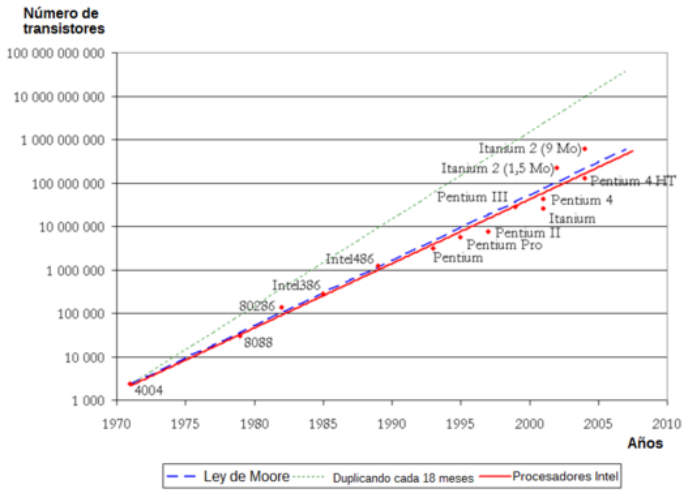

Ilustración 10: Ley de Moore

Estas ventajas de flexibilidad y potencial que nos ofrecen las FPGA frente a otros dispositivos están haciendo que suscite interés en diferentes sectores como son la electrónica, la industria aeroespacial, automotriz y médica, el audio y en otros sectores que están tomando mucha fuerza como son el big data y la inteligencia artificial [9].

Otra de las principales diferencias que encontramos en una FPGA con respecto al resto de microprocesadores que trabajan de forma secuencial, es que la actividad de estas se desarrolla de forma paralela, de manera que múltiples acciones son llevadas a cabo al mismo tiempo. Dicha característica es lo que la hace más interesante a nivel de su integración con el uso de redes neuronales debido a la naturaleza paralela también de las mismas.

Por otra parte, muchas de las placas de desarrollo en las que se encuentran integradas las FPGA, cuentan con una zona de memoria donde es posible la implementación de un soft-processor como es el caso de Xilinx's MicroBlaze. Esto nos permite contar con un procesador que si trabaja de forma secuencial y con una serie de instrucciones fijas que para procesos complejos puede llegar a resultar muy útil.

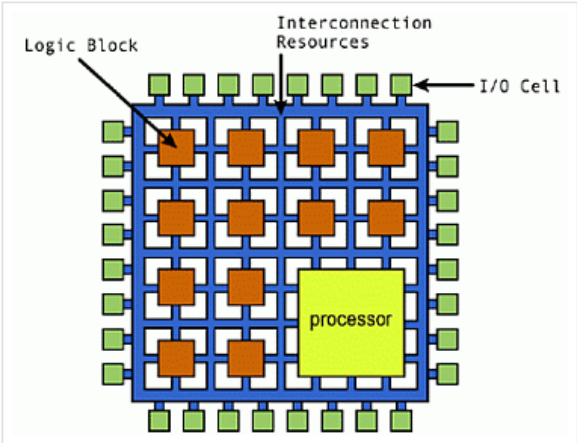

Ilustración 11:Representación de la estructura de una FPGA con soft-processor

4 IMPLEMENTACIÓN DE LA RED EN VHDL: PROTOTIPO DE UN SISTEMA REAL

Debido a la gran aplicación de las redes neuronales como solución a múltiples retos que se plantean hoy día y al mercado en auge que supone la utilización de las FPGA debido al potencial que estas nos ofrecen, se ha decidido realizar la implementación de una red neuronal en una FPGA.

En esta aplicación particular, el escenario en el que se ha implementada la red neuronal es el siguiente:

Disponemos de un sistema con 15 sensores del que obtenemos ciertas medidas características, en concreto, EPSILON_DMS. El objetivo de la aplicación de la red neuronal es predecir el valor de uno de ellos en el caso de que este fallara.

De esta manera se previene el posible error en aplicaciones posteriores que sean dependientes de este valor, proporcionando un valor aproximado del sensor que ha fallado.

En el presente documento únicamente se recoge la implementación de la red neuronal para uno de los sensores, aunque esta seria idéntica para el resto de ellos cambiando los parámetros que definen la red neuronal.

4.1 Diseño de la red neuronal con MATLAB

El diseño de la red neuronal se ha llevado a cabo con el uso de la herramienta *Neural Network Toolbox de Matlab*, la cual nos permite la creación de redes neuronales proporcionados una serie de especificaciones y parámetros.[10]

La red neuronal creada se ha realizado para proporcionar a un sistema que cuenta con una serie de sensores, los valores que estos nos proporcionarían si no estuvieran. Por ello se ha generado una red neuronal que introduciendo los valores de entrada de los sensores que si están siempre presentes nos da el valor de salida del sensor que no se encuentra y con el que con dicha salida hemos entrenado la red.

Para ello los datos son almacenados en una matriz de 50x32 donde 32 corresponde con el número de sensores que tenemos y 50 con el número de valores que estos nos han proporcionado.

A continuación, se detallará como se han procedido a realizar los diferentes pasos de diseño para la implementación realizada

En primer lugar, para la ***creación de la red*** se ha hecho uso de la función de Matlab `feedforwardnet` la cual nos permite la creación de una red neuronal multicapa de tipo feedforward

```
%%CREACIÓN DE LA RED
%Creación la red vacia
red_1=feedforwardnet;
```

Una vez creada la red se procede a la **configuración de la red** indicándoles así el valor de los diferentes parámetros de esta: número de entradas, tamaño de las entradas, número de capas, número de neuronas de las capas, funciones de activación de las neuronas de las diferentes capas, función de entrenamiento de la red y bias de las neuronas.

En este caso se han utilizado 2 capas , una capa con 3 neuronas con función de activación sigmoidal y otra capa con una neurona con función de activación lineal.

El algoritmo de entrenamiento utilizado se define con la función de entrenamiento 'trainbr' la cual corresponde con la función bayesiana de regularización, la cual actualiza los valores de los pesos y los bias según la optimización de Levenberg-Marquardt, el cual busca la minimización de los errores cuadrados mínimos.

```
%%CONFIGURACIÓN DE LA RED
%Numero de entradas
red_1.numInputs=1

%Numero de tamaño de las entradas
red_1.inputs{1}.size=14

%Numero de capas
red_1.numLayers=2

%Numero de neuronas de la capa 1 Y 2
red_1.layers{1}.size=3
red_1.layers{2}.size=1

%Funcion de transferencia de las neuronas de la capa 1 y 2
red_1.layers{1}.transferFcn='tansig'
red_1.layers{2}.transferFcn='purelin'

%Función de entrenamiento
red_1.trainFcn='trainbr';

%Añado bias a las neuronas de la capa 1 y 2
red_1.biasConnect=[1;1];
```

La **inicialización de los pesos y bias** se ha realizado con un valor inicial de 1 que posteriormente será ajustado al realizar el entrenamiento de la red.

```
%%INICIALIZACIÓN DE LOS PESOS Y EL BIAS DE LAS NEURONAS
%Peso de las neuronas de la capa 1
red_1.iw{1}=[1 1    1 1 1  1 1 1   1 1 1   1 1 1;
   1 1   1 1 1   1 1 1   1 1 1   1 1 1;
   1 1   1 1 1   1 1 1   1 1 1   1 1 1;];

%Bias de las neuronas de la capa 1
red_1.b{1}=[1 1 1]';

%Peso inicial de las neuronas de la capa 2
red_1.lw{2}=[1 1 1];

%Bias inicial de las neuronas de la capa 2
red_1.b{2}=1;
```

Posteriormente con la función `train` y la función `'trainbr'` definida como algoritmo de aprendizaje se produce el **entrenamiento de la red**. Para realizar el entrenamiento de la red son necesarios como parámetros de entrada: la red que va a ser entrenada y un conjunto de entradas con sus salidas deseadas para el aprendizaje de la red. Mientras se produce el entrenamiento de la red también se realiza la **validación** de la misma

```
%%ENTRENAMIENTO DE LA RED
%Entrenamiento de la red
[red_1,tr_1]=train(red_1,x_1,t_1);
```

Una vez creada, configurada y entrenada la red el siguiente paso será conocer los valores de dicha configuración final para poder realizar la implementación en VHDL

A través de la línea de comandos podemos determinar por cuantas capas y de cuantas neuronas en cada capa, está constituida nuestra red. Para ello hacemos uso de los comandos:

net.iw{a,b} donde: a se corresponde con el número de capa de entrada y b con el número de entrada del cual queremos observar el peso.

net.lw{a,b} donde: a se corresponde con el número de capa y b con el número de neurona del cual queremos observar el peso.

net.b{a,b} donde: a se corresponde con el número de capa del cual queremos observar los bias.

4.2 Implementación de la Red Neuronal en VHDL

Debido a la estructura que presenta una FPGA, a la hora de la implementación de cualquier sistema se deben tener en cuenta ciertos aspectos fundamentales.

4.2.1 Naturaleza de las señales en VHDL

Una FPGA trabaja fundamentalmente con señales que representan un voltaje a nivel alto o bajo. Esto nos lleva a que no podemos definir como tales señales que no representen números enteros, suponiendo esta restricción uno de los principales obstáculos a la hora de la realización del proyecto. Como solución a esto, se ha propuesto la utilización únicamente de números enteros, a pesar de que ello supone la pérdida de precisión del valor estimado. La cuestión viene cuando nos preguntemos acerca del uso de la red neuronal, la cual trabaja fundamentalmente con valores decimales en un rango muy limitado, lo cual nos lleva a que si redondeamos estos valores las soluciones que proporciona no tienen sentido.

Teniendo en cuenta que las redes neuronales estudiadas en el nivel del modelo matemático que estas representan suponen un conjunto de sumas, multiplicaciones y otras operaciones matemáticas sencillas. Se ha procedido al escalado de este conjunto de transformaciones matemáticas para que los datos con los que se opera y los que se dan como resultado correspondan con un múltiplo en base 10 del dato decimal requerido.

Para la obtención de una precisión de dos decimales, son necesarias transformaciones tanto en algunas de las funciones matemáticas usadas, como en los datos que se utilizan en estas. Las transformaciones realizadas en cada uno de los pasos se ira explicando detalladamente conforme se va haciendo referencia a cada uno de ellos y las realizadas en los datos son las siguientes:

- Valores escalados de los pesos de las neuronas = 100*Valores originales de los pesos de las neuronas.
- Valores escalados de los bias de las neuronas = 1000*Valores originales de los bias de las neuronas.
- Redondeo de los valores de entrada a la red.
- Redondeo de los valores de salida de la red.

4.2.2 Combinación de proceso síncrono y combinacional

Una de las principales características del lenguaje VHDL es el procesado en paralelo, esto conlleva la existencia de un proceso síncrono y de un proceso combinacional.

4.2.2.1 Proceso Síncrono

El proceso síncrono se encarga de actualizar el estado en el que se encuentran las variables cada vez que se produce un pulso de reloj, esto nos permite que los valores de las señales que entran a las neuronas cambien en el mismo momento, de manera que el procesamiento de dichos valores se ejecute de forma paralela en cada una de las fases de la red neuronal.

Dentro de este proceso síncrono encontramos dos situaciones:

Activación de la función reset de nuestro programa

En este caso los valores de la red neuronal vuelven a su valor por defecto de manera que el procesamiento llevado a cabo por la red neuronal finaliza y se vuelve a ejecutar desde cero con los valores presentes en la entrada en ese instante.

```
if(reset='1') then
--Entradas a la red neuronal
             ent_x_1<=E1;
             ent_x_2<=E2;
             ent_x_3<=E3;
             ent_x_4<=E4;
             ent_x_5<=E5;
             ent_x_6<=E6;
             ent_x_7<=E7;
             ent_x_8<=E8;
             ent_x_9<=E9;
             ent_x_10<=E10;
             ent_x_11<=E11;
             ent_x_12<=E12;
             ent_x_13<=E13;
      ent_x_14<=E14;

--Entradas a las neruonas
             ent_n1<="0000000000000000000";
             ent_n2<="0000000000000000000";
             ent_n3<="0000000000000000000";
             ent_n4<="0000000000000000000";
--Salidas de las neruonas
             sal_n1<="0000000000000000000";
             sal_n2<="0000000000000000000";
             sal_n3<="0000000000000000000";
             sal_n4<="0000000000000000000";
--Salida de la red neuronal
      Y<="0000000000000000000";      salidared<="0000000000000000000"
```

Actualizacion de los valores cada ciclo de reloj

En caso de que no se haya hecho uso de la función reset de nuestro programa, los valores de las señales de entrada y salida de las neuronas se actualizarán cada ciclo de reloj

```
elsif (rising_edge(clk)) then--en el caso de no estar el reset activado y que haya un pulso de reloj
             -----Actualizamos todas las señales a sus valores actuales-----
```

```
            Y <=p_salidared;
            ent_x_1<=E1;
            ent_x_2<=E2;
            ent_x_3<=E3;
            ent_x_4<=E4;
            ent_x_5<=E5;
            ent_x_6<=E6;
            ent_x_7<=E7;
            ent_x_8<=E8;
            ent_x_9<=E9;
            ent_x_10<=E10;
            ent_x_11<=E11;
            ent_x_12<=E12;
            ent_x_13<=E13;
            ent_x_14<=E14;
            ent_n1<=p_ent_n1;
            ent_n2<=p_ent_n2;
            ent_n3<=p_ent_n3;
            ent_n4<=p_ent_n4;
sal_n1<=p_sal_n1;         sal_n2<=p_sal_n2;
            sal_n3<=p_sal_n3;
            sal_n4<=p_sal_n4;
            salidared<=p_Sali
```

Proceso combinacional

El proceso combinacional se encarga de determinar el valor de las variables que se les va a definir a estas en el proceso síncrono que se active en el siguiente ciclo de reloj. En este proceso se ejecutará la lógica del procesamiento de nuestra red neuronal.

En este proyecto se ha llevado a cabo la implementación de una red con dos capas, una de entrada y de salida.

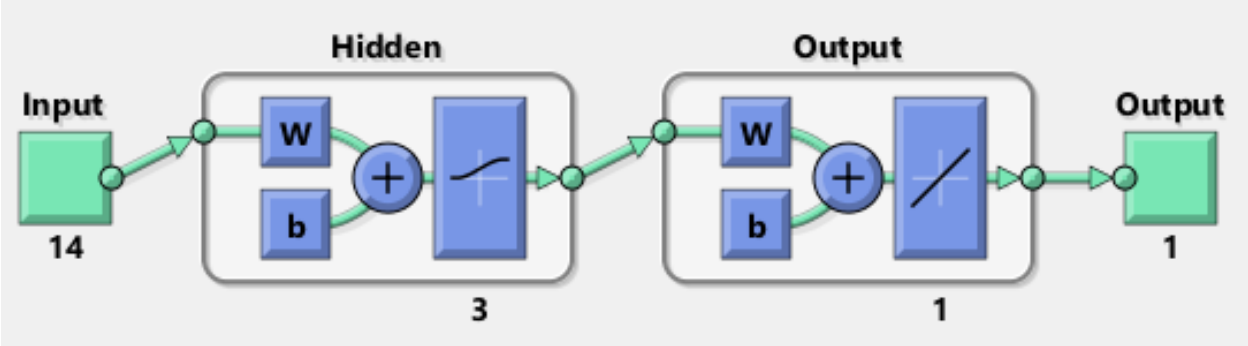

Ilustración 12: Esquema del diseño realizado con el NNT de Matlab para la implementación de la red neuronal

La capa de entrada consta de un vector de entradas de 14 componentes, correspondiente con el número de sensores presentes, y de tres neuronas con una función de activación sigmoide.

La capa de salida consta de una única neurona cuyas entradas son las salidas de las tres neuronas de la capa de entrada, y una función de activación lineal.

Dentro de la red neuronal diseñada existen tres etapas bien diferenciadas.

- **Pre-Procesamiento**: En esta etapa se busca que los datos que se introducen a la red sean los adecuados para que nuestra red funcione de la manera más eficiente posible.

- **Procesamiento de los datos**: En esta etapa se produce el cálculo de la salida de la red neuronal en función a los parámetros de esta y a los valores de entrada que se le pasan.

- **Post-Procesamiento**: En esta etapa, al igual que en la de preprocesamiento, se busca la eficiencia de nuestra red neuronal. Los datos de salida se convierten de manera que estos sean mas eficientes para el objetivo con el que se ha desarrollado nuestra red.

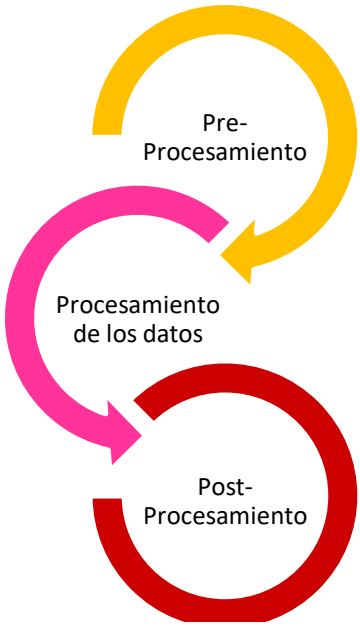

Ilustración 13: Gráfico de las etapas implementadas en la red neuronal

```
-----------------------------------PROCESO COMBINACIONAL-------------------------
----------------

comb:process(ent_n1,ent_n2,ent_n3,ent_n4,sal_n1,sal_n2,sal_n3,sal_n4,x_1,x_2,x_3,x_4,
x_5,x_6,x_7,x_8,x_9,x_10,x_11,x_12,x_13,x_14,ent_x_1,ent_x_2,ent_x_3,ent_x_4,ent_x_5,
ent_x_6,ent_x_7,ent_x_8,ent_x_9,ent_x_10,ent_x_11,ent_x_12,ent_x_13,ent_x_14,salidare
d)

 VARIABLE num_entrada: integer;

 VARIABLE num_neurona: integer;

      begin

---------------Preprocesamiento:NORMALIZACION-----------------------------------
--------------------

            x_1<=f_norm_directa(ent_x_1,1);

            x_2<=f_norm_directa(ent_x_2,2);

            x_3<=f_norm_directa(ent_x_3,3);

            x_4<=f_norm_directa(ent_x_4,4);

            x_5<=f_norm_directa(ent_x_5,5);

            x_6<=f_norm_directa(ent_x_6,6);

            x_7<=f_norm_directa(ent_x_7,7);

            x_8<=f_norm_directa(ent_x_8,8);

            x_9<=f_norm_directa(ent_x_9,9);

            x_10<=f_norm_directa(ent_x_10,10);

            x_11<=f_norm_directa(ent_x_11,11);

            x_12<=f_norm_directa(ent_x_12,12);

            x_13<=f_norm_directa(ent_x_13,13);

            x_14<=f_norm_directa(ent_x_14,14);


-----------------------------------------------------------------------------------
--------------------

-------------Procesamiento de los datos-------------------------------------------
--------------------

------Neuronas de la capa de entrada-----------------------

------NEURONA 1--------------------------------------------

            --num_entrada:=2;

            --num_neurona:=1;

            --p_salida_n2<=neurona(entrada_b,2,1);

            p_ent_n1<=std_logic_vector( to_signed
(to_integer(signed(lookuppesos(x_1,1,1)))+ to_integer(signed(lookuppesos(x_2,2,1)))+
to_integer(signed(lookuppesos(x_3,3,1)))+ to_integer(signed(lookuppesos(x_4,4,1)))+
```

```
to_integer(signed(lookuppesos(x_5,5,1)))+ to_integer(signed(lookuppesos(x_6,6,1)))+
to_integer(signed(lookuppesos(x_7,7,1)))+ to_integer(signed(lookuppesos(x_8,8,1)))+
to_integer(signed(lookuppesos(x_9,9,1)))+ to_integer(signed(lookuppesos(x_10,10,1)))+
to_integer(signed(lookuppesos(x_11,11,1)))+
to_integer(signed(lookuppesos(x_12,12,1)))+
to_integer(signed(lookuppesos(x_13,13,1)))+
to_integer(signed(lookuppesos(x_14,14,1)))+  lookupbias(1) , 20 ));

             p_sal_n1<=f_sigmoidal(ent_n1);

-----------------------------------------------------------

------NEURONA 2-----------------------------------------

             --num_entrada:=2;

             --num_neurona:=1;

             --p_salida_n2<=neurona(entrada_b,2,1);

             p_ent_n2<=std_logic_vector( to_signed
(to_integer(signed(lookuppesos(x_1,1,2)))+ to_integer(signed(lookuppesos(x_2,2,2)))+
to_integer(signed(lookuppesos(x_3,3,2)))+ to_integer(signed(lookuppesos(x_4,4,2)))+
to_integer(signed(lookuppesos(x_5,5,2)))+ to_integer(signed(lookuppesos(x_6,6,2)))+
to_integer(signed(lookuppesos(x_7,7,2)))+ to_integer(signed(lookuppesos(x_8,8,2)))+
to_integer(signed(lookuppesos(x_9,9,2)))+ to_integer(signed(lookuppesos(x_10,10,2)))+
to_integer(signed(lookuppesos(x_11,11,2)))+
to_integer(signed(lookuppesos(x_12,12,2)))+
to_integer(signed(lookuppesos(x_13,13,2)))+
to_integer(signed(lookuppesos(x_14,14,2)))+  lookupbias(2) , 20 ));

             p_sal_n2<=f_sigmoidal(ent_n2);

-----------------------------------------------------------

------NEURONA 3-----------------------------------------

             --num_entrada:=2;

             --num_neurona:=1;

             --p_salida_n2<=neurona(entrada_b,2,1);

             p_ent_n3<=std_logic_vector( to_signed
(to_integer(signed(lookuppesos(x_1,1,3)))+ to_integer(signed(lookuppesos(x_2,2,3)))+
to_integer(signed(lookuppesos(x_3,3,3)))+ to_integer(signed(lookuppesos(x_4,4,3)))+
to_integer(signed(lookuppesos(x_5,5,3)))+ to_integer(signed(lookuppesos(x_6,6,3)))+
to_integer(signed(lookuppesos(x_7,7,3)))+ to_integer(signed(lookuppesos(x_8,8,3)))+
to_integer(signed(lookuppesos(x_9,9,3)))+ to_integer(signed(lookuppesos(x_10,10,3)))+
to_integer(signed(lookuppesos(x_11,11,3)))+
to_integer(signed(lookuppesos(x_12,12,3)))+
to_integer(signed(lookuppesos(x_13,13,3)))+
to_integer(signed(lookuppesos(x_14,14,3)))+  lookupbias(3) , 20 ));

             p_sal_n3<=f_sigmoidal(ent_n3);

-----------------------------------------------------------

-----------------------------------------------------------

------Neuronas de la capa de salida-----------------------

------NEURONA 4-----------------------------------------

             --num_entrada:=2;
```

```
            --num_neurona:=1;

            --p_salida_n2<=neurona(entrada_b,2,1);

            p_ent_n4<=std_logic_vector( to_signed
(to_integer(signed(lookuppesos(sal_n1,1,4)))+
to_integer(signed(lookuppesos(sal_n2,2,4)))+
to_integer(signed(lookuppesos(sal_n3,3,4)))+ lookupbias(4) , 20 ));

            --p_ent_n5<="0000011100";

            p_sal_n4<=f_lineal(ent_n4);

------------------------------------------------------------

------------------------------------------------------------------------------------------
--------------------

----------------------Postprocesamiento:NORMALIZACION INVERSA-------------------------
--------------------



            p_salidared<=f_norm_inversa(sal_n4);

------------------------------------------------------------------------------------------
--------------------

end process;

end Behavioral;
```

4.2.2.1.1 Implementación funciones preprocesamiento

Normalización directa

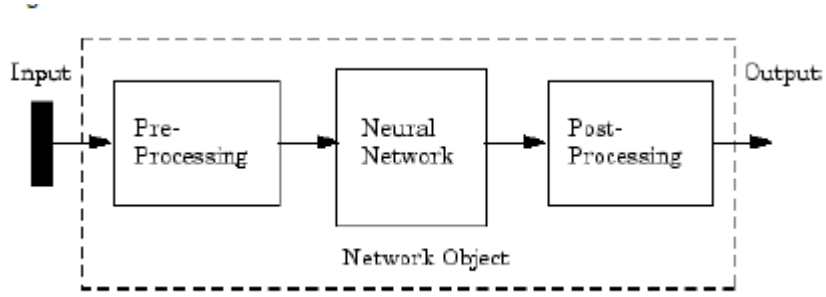

Ilustración 14: Esquema de las etapas de pre-procesamiento, procesamiento y post-procesamiento de una red neuronal

Con el objetivo de que la red neuronal sea más eficiente se lleva a cabo un preprocesamiento de los datos antes de que estos entren en la red neuronal. Este preprocesamiento de los datos se denomina normalización directa y consiste en el escalado de los valores de entrada a un rango [-1,1].

Para esta etapa de preprocesamiento de los datos Matlab nos ofrece una función 'mapminmax' la cual realiza este escalado introduciendo como parámetros los valores de los experimentos realizados y los límites entre los que realizar este escalado y la cual presenta el siguiente algoritmo:

y = (ymax-ymin)*(x-xmin)/(xmax-xmin) + ymin;

Siendo los parámetros de entrada:

Xmax= Valor máximo registrado de cada entrada de todos los experimentos utilizados para entrenar la red

Xmin= Valor mínimo registrado de cada entrada de todos los experimentos utilizados para entrenar la red

Ymax= Límite superior del escalado

Ymin= Límite inferior del escalado

A continuación, se presenta la implementación realizada para esta función de normalización contemplando únicamente el cado de un sensor, se puede consultar la función completa para el resto de sensores en el anexo de este documento:

```
----------------------NORMALIZACION DIRECTA-------------------------


FUNCTION f_norm_directa(entrada: STD_LOGIC_VECTOR ( 19 DOWNTO 0) ;
n_señal: integer )RETURN STD_LOGIC_VECTOR  IS


VARIABLE salida: STD_LOGIC_VECTOR ( 19 DOWNTO 0);
```

```
  VARIABLE X_Min : integer ;
  VARIABLE X_Max : integer ;
  VARIABLE Y_Min : integer ;
  VARIABLE Y_Max : integer;


 BEGIN
 Y_Min := -1;
 Y_Max := 1;
case n_señal is
-----Neurona capa de entrada--------
--Sensor 1--
when 1 => X_Min := 10;---151;-- -1.5
          X_Max := 147;
end case;


salida:=std_logic_vector(to_signed(((to_integer(signed(entrada))-
X_Min)*(Y_Max-Y_Min)*10/     (X_Max-X_Min)) -10 ,20));


RETURN salida;
END f_norm_directa;
```

Aunque no aplique a la función de normalización directa también cabe destacar la utilización de otra función de preprocesamiento de los datos utilizada por el NNT de Matlab: RemoveConstantRows. Dicha función elimina los elementos de entrada que tienen el mismo valor ya que la información repetida no aporta información útil a la hora de entrenar la red.

Para ello a la hora de realizar el entrenamiento de la red se ha procedido a seleccionar los experimentos que resultaban relevantes es decir aquellos que tenían valores diferentes para cada uno de los sensores.

Uno de los puntos más importantes a la hora del entrenamiento y creación de una red neuronal eficiente es la correcta determinación de los parámetros utilizados para identificar el patrón de salida. Un conjunto de entradas incompleto o que no se ajuste a dicho patrón puede derivar en que la red no realice su tarea correctamente.

Así mismo, también hay que tener presente que un conjunto de parámetros sobredimensionados afecta también a la eficiencia de la red, pues incrementa el número de neuronas innecesariamente y con este el tiempo de procesamiento.

Debido a la limitación de trabajar con números decimales explicada previamente, el rango de escalado implementado ha sido realizado entre -10 y 10 veces el valor escalado.

En la siguiente tabla se muestra el escalado ideal al que se vería sometido el vector de entrada y el aproximado en la implementación realizada. Como se puede observar en la gráfica el redondeo ha sido realizado hacia arriba ya que experimentalmente se ha observado que proporciona mejores resultados

Tabla 2: Comparativa entre los valores reales de entrada, la aproximación ideal y la realizada

Número de entrada	Valor entrada	Aproximación Ideal	Aproximación Realizada
1	61	-0.2686	-3
2	38	-0.2058	-3
3	19	-0,9295	-10
4	38	0,0252	0
5	57	-0,2803	-3
6	8	-0,3280	-4
7	13	-0,9393	-10
8	10	-0,1114	-1
9	93	-0,1060	-2
10	44	-0,2768	-3
11	54	-0,1926	-2
12	26	-0,8867	-9
13	-16	0,7905	9
14	-13	0,9900	11

4.2.2.1.2 Procesamiento de los datos : Implementación de las neuronas

La implementación de las neuronas se lleva a cabo en dos fases:

En primer lugar, una etapa que consiste en la entrada de las diferentes señales a la propia neurona y el sumatorio de la multiplicación de los diferentes valores de entrada por su peso correspondiente y el bias asociado a cada una de ellas.

En segundo lugar, una etapa que consiste en la activación de la neurona, dependiendo de si nos encontramos en la capa de entrada o en la de salida las funciones de activación serán la función sigmoidal o lineal respectivamente.

Entrada a las neuronas

En el siguiente gráfico se muestran los diferentes procesos con los que se realiza la implementación de la etapa de entrada a las neuronas.

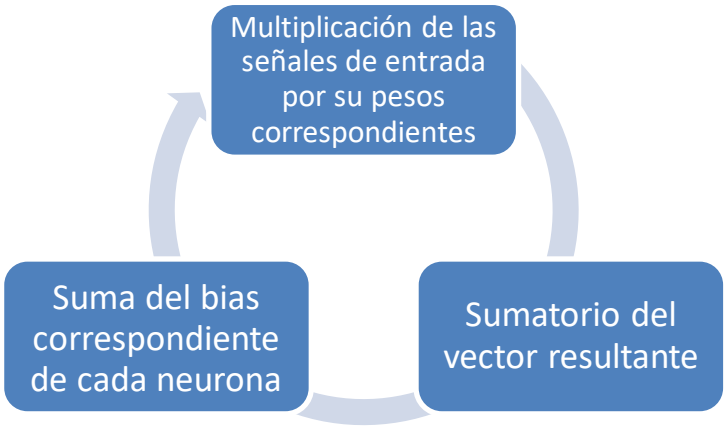

Ilustración 15: Gráfico de las etapas para la implementacion de una neurona

Multiplicación de las señales de entrada por sus pesos correspondientes

La salida de este proceso nos da como respuesta la multiplicación de cada una de las neuronas por su peso correspondiente a la neurona que está calculando y la suma del bias de esta.

Debido a que la FPGA trabaja en paralelo la suma multiplicación de los pesos por las entradas se realiza en paralelo lo que supone un tiempo de procesado mucho menor que si realizara de forma secuencial.

Este proceso se realiza a través del uso de la función *lookuppesos(señal, n_señal, n_neurona)*, la cual contiene almacenados diferentes LUTs (Look up Tables) con los diferentes valores de los pesos y la cual recibe por parámetros:

- Señal: Valor de la señal de entrada a la neurona
- N_señal: Identificador de cada una de las señales que entran a las diferentes neuronas de la red
- N_neurona :Identificador de cada una de las neuronas de la red

```
------Look up pesos-----------------------------------------------------------

FUNCTION lookuppesos(señal : in std_logic_vector(19 downto 0); n_señal : integer ;
n_neurona : integer)RETURN std_logic_vector IS

        VARIABLE peso: integer;
```

```
        VARIABLE bias: integer;
        VARIABLE valor: std_logic_vector(19 downto 0);
        BEGIN
            case n_neurona is
            --Neurona 1--
            when 1 => case n_señal is
                                    when 1 => peso := 0;
                                    when 2 => peso := 20;
                                    when 3 => peso := 27;
                                    when 4 => peso := -23;
                                    when 5 => peso := 28;
                                    when 6 => peso := 35;
                                    when 7 => peso := 52;
                                    when 8 => peso := -74;
                                    when 9 => peso := 59;
                                    when 10 => peso := -4;
                                    when 11 => peso := 16;
                                    when 12 => peso := -61;
                                    when 13 => peso := 11;
                                    when 14 => peso := -13;
                                    when others => peso := 0;
                        end case;
            when others => peso := 0;-- 0
            end case;
            valor:=std_logic_vector(to_signed(to_integer(signed(señal))*peso,20));
            RETURN valor;
            END lookuppesos;
-------------------------------------------------------------
```

Sumatorio del vector resultante y suma del bias correspondiente de cada neurona

El siguiente paso en la etapa de entrada a la neurona es la suma del vector resultante de la operación anterior y la adición del valor del bias asociado a cada neurona.

Para el cálculo del valor del bias de cada neurona se hace uso de la función *lookupbias(n_neurona)* a la cual se le pasa el identificador de la neurona y devuelve el valor del bias asociado a esta

```
------NEURONA 1-----------------------------------------

p_ent_n1<=std_logic_vector( to_signed (to_integer(signed(lookuppesos(x_1,1,1)))+
to_integer(signed(lookuppesos(x_2,2,1)))+ to_integer(signed(lookuppesos(x_3,3,1)))+
to_integer(signed(lookuppesos(x_4,4,1)))+ to_integer(signed(lookuppesos(x_5,5,1)))+
to_integer(signed(lookuppesos(x_6,6,1)))+ to_integer(signed(lookuppesos(x_7,7,1)))+
to_integer(signed(lookuppesos(x_8,8,1)))+ to_integer(signed(lookuppesos(x_9,9,1)))+
to_integer(signed(lookuppesos(x_10,10,1)))+
to_integer(signed(lookuppesos(x_11,11,1)))+
to_integer(signed(lookuppesos(x_12,12,1)))+
to_integer(signed(lookuppesos(x_13,13,1)))+
to_integer(signed(lookuppesos(x_14,14,1)))+  lookupbias(1) , 20 ));

-----------------------------------------------------
```

Activación de las neuronas de la primera capa.

Se han implementado diferentes funciones de activación de las neuronas en vista a líneas de trabajo futuras, ya que algunas, como es el caso de la función de activación de tipo de escalón no se hace uso de ellas en esta implementación concreto, aun no siendo útiles en este escenario concreto si puede serlo en otro diferente.

Las funciones de activación implementadas han sido: función sigmoide, función escalón y función lineal.

Debido a la estructura de estas, la implementación de las funciones de activación de tipo escalón y lineal no han presentado complejidad, en comparativa con el caso de la función de activación sigmoide, la cual es el tipo de función utilizada en las neuronas de la primera capa.

Implementación de la función escalón

La implementación de la función escalón se traduce en una simple discretización de las variables de entrada según el signo del valor de entrada a la neurona tal y como se muestra en el código adjunto.

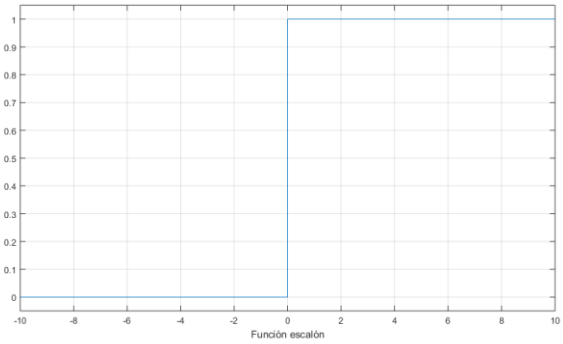

Ilustración 16:Función escalón implementada

```
----F.Activ Escalon-----------------------------------------------


FUNCTION f_escalon(entrada: STD_LOGIC_VECTOR ( 10 DOWNTO 0) )RETURN STD_LOGIC_VECTOR
IS


        VARIABLE salida: STD_LOGIC_VECTOR ( 10 DOWNTO 0);
```

```
        BEGIN
            if ( to_integer(signed(entrada)) <= 0 ) then
            salida:="00000000000";
            else
            salida:="00000001010";
            end if;
            RETURN salida;
            END f_escalon;
-----------------------------------------------------------------
```

Implementación de la función lineal

Para la neurona de la capa de salida de la red neuronal diseñada se ha implementado una función lineal con punto de corte con el eje x e y el origen de coordenadas, tal y como se muestra en el código adjunto.

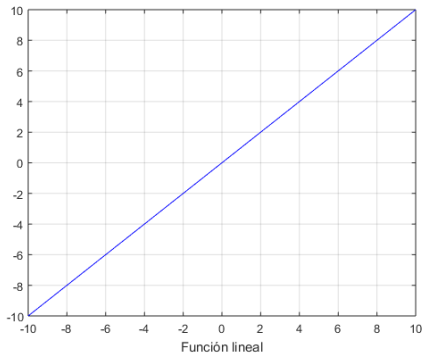

Ilustración 17: Función lineal implementada

```
----F.Activ Lineal--------

FUNCTION f_lineal(entrada: STD_LOGIC_VECTOR ( 10 DOWNTO 0) )RETURN STD_LOGIC_VECTOR
IS

        VARIABLE salida: STD_LOGIC_VECTOR ( 10 DOWNTO 0);
        BEGIN
            salida:=entrada;

            RETURN salida;
            END f_lineal;
--------------------------------------------------------------------------------
```

Implementación de la función sigmoidal

Como se ha comentado previamente la función sigmoidal es la que más se asemeja al comportamiento biológico de una neurona y la cual suele ser utilizada en las redes neuronales, así mismo esta conlleva una carga computacional mucho más elevada que las otras funciones de activaciones descritas: escalón y rampa.

En la implementación digital sobre FPGA de redes neuronales, es importante la eficiencia a la hora del cálculo. La síntesis hardware directa de la expresión matemática de la función de activación sigmoide logarítmica no es práctica ya que tanto la exponencial como las divisiones que conforman esta función derivan en una lógica excesiva, la cual converge lentamente.

En consecuencia, se ha desarrollado una aproximación matemática que mejora la eficiencia de este cálculo. Para ello se ha realizado en Matlab la comparación de la función aproximada con la función sigmoide para comprobar que la aproximación es adecuada.

$$\begin{aligned} x < -3 \quad & \quad y = 0 \\ -3 < x < 3 \quad & \quad y = (x+3)/6 \\ x > 3 \quad & \quad y = 1 \end{aligned}$$

A continuación, se adjunta el código de Matlab realizado para comprobar la aproximación de la función sigmoidal y la gráfica comparativa.

```
%%%%%%%%%%%%% Aproximacion de la funcion sigmoidal%%%%%%%%%%%%

%%Funcion sigmoidal
x = -10:0.1:10;
y = sigmf(x,[1 0]);
plot(x,y)
xlabel('sigmf, P = [1 0]')
ylim([-0.05 1.05])
grid
hold on

 %%Aproximacion funcion lineal y saturacion

%Saturacion por debajo
plot([-10,-3],[0,0],'r')

%Recta-Valores intermedios
x = -3:0.1:3;
y_aprox=(x+3)/6;
plot(x,y_aprox,'r')
```

```
%Saturacion por encima
plot([3,10],[1,1],'r')
```

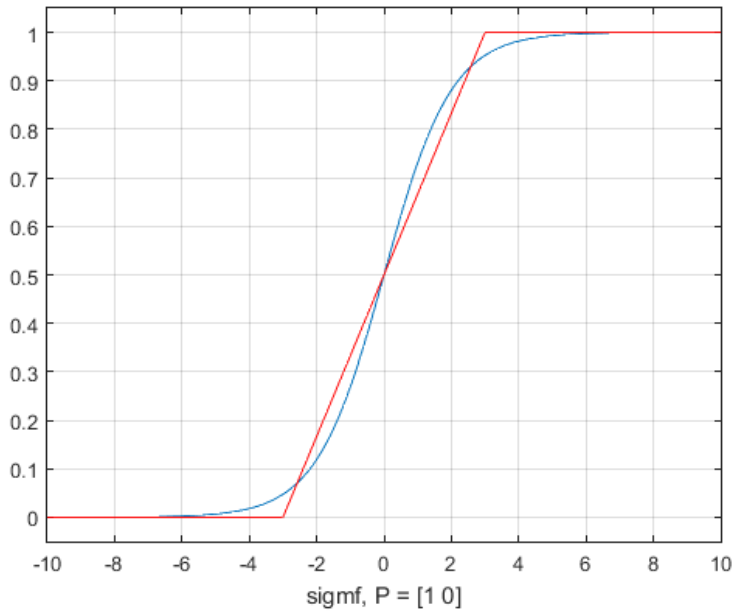

Ilustración 18:Aproximación de la función sigmoidal implementada

A la hora de realizar la implementación de la función sigmoidal nos encontramos con el problema de la falta de precisión en los resultados debido a la discretización que aparecen en los valores de salida debido a la naturaleza discreta del lenguaje.

Por ello, aun teniendo en cuenta que supone un aumento de coste computacional, la aproximación lineal realizada se hace sobre una escala (-1000,1000) ya que el rango de valores de la entrada a la función sigmoidal se encuentra en esta escala debido a las multiplicaciones previas de los pesos que presentada la entrada original a la red.

Experimentalmente se ha comprobado que se podría haber realizado un escalado de las variables de entrada a la función de activación para cambiar el rango de valores de la aproximación, pero esta opción mostraba unos resultados muchos menos precisos que los obtenidos manteniendo una escala (0-1000).

Una vez realizada la aproximación previamente descrita para que este valor no interfiera con cálculos posteriores lo cual supondría aumentar el tamaño de las señales con las cuales se trabaja en la FPGA, se convierte el valor de un rango de -1000 a 1000 a un rango de -10 a 10.

En el anexo a este documento se adjunta la aproximación realizada a la función sigmoidal en lenguaje VHDL realizada.

4.2.2.1.3 Implementación funciones post-procesamiento

Normalización inversa

En último lugar se lleva a cabo una etapa de post-procesamiento de los datos, lo cual nos permite obtener datos útiles para el objetivo con el que se ha diseñado la red neuronal.

El proceso que se lleva acabo se denomina normalización inversa y como su nombre indica es el proceso inverso al realizado en la etapa de preprocesamiento con la normalización directa. De manera similar a como se ha realizado en la normalización directa, el resultado de la capa de salida, el cual se encuentra en un rango de -10 a 10 en nuestro caso, se convierte a un valor con sentido respecto a las entradas introducidas en la red neuronal.

En este caso el algoritmo que se procesa en la función es:

```
y = (Xmax-Xmin)*(y-ymin)/(ymax-ymin) + xmin;
```

Siendo los parámetros de entrada:

Xmax= Valor máximo registrado de cada salida de todos los experimentos utilizados para entrenar la red

Xmin= Valor mínimo registrado de cada salida de todos los experimentos utilizados para entrenar la red

Ymax= Límite superior del escalado

Ymin= Límite inferior del escalado

A continuación, se presenta la implementación realizada para esta función de normalización:

```
----------------------NORMALIZACION INVERSA---------------------------------

FUNCTION f_norm_inversa(entrada: STD_LOGIC_VECTOR ( 19 DOWNTO 0))RETURN
STD_LOGIC_VECTOR  IS


                          VARIABLE salida: STD_LOGIC_VECTOR ( 19 DOWNTO 0);

                          VARIABLE X_Min_Inv : integer ;

                          VARIABLE X_Max_Inv : integer ;

                          VARIABLE Y_Min_Inv : integer ;

                          VARIABLE Y_Max_Inv : integer    ;


                         BEGIN

                         Y_Min_Inv := -1000;

                         Y_Max_Inv := 1000;

                         X_Min_Inv := -42;

                         X_Max_Inv := -5;
```

```
salida:=std_logic_vector(to_signed((((to_integer(signed(entrada))) -
Y_Min_Inv)*(X_Max_Inv - X_Min_Inv)/(Y_Max_Inv - Y_Min_Inv)+(X_Min_Inv)) ,20)
);



                        RETURN salida;

                        END f_norm_inversa;

--------------------------------------------------------------------------
```

4.2.3 Diseño y Simulación de la implementación de la red neuronal en VHDL

Como punto previo a la implementación de la red neuronal en VHDL, se realizado un modelado del algoritmo a implementar en MATLAB el cual se ha utilizado para el ajuste de la simplificación de los pesos posteriormente introducidos en la red en VHDL, debido a las limitaciones con respecto a la escala que previamente se han comentado. Dicho diseñe se encuentra presente en el *ANEXO D– Diseño de la red implementada en VHDL haciendo uso del Neural Network Toolbox MatLab*

Como comprobación del correcto funcionamiento de la red implementada se ha realizado una simulación de la misma a través de la herramienta 'TestBench' que nos proporciona 'ISE Design Suite' , herramienta sobre la que se ha realizado la implementación. Dicha simulación consta de tres experimentos en los que se le proporciona a la red implementada 3 diferentes vectores de entrada con los que se busca observar si la salida tiene un valor aceptable.

Los valores presentes en dicho vector son proporcionados de forma paralela a las 13 entradas de las cuales consta nuestra red y con una separación de 10 ciclos de reloj entre ellos, así la red implementada cambia su valor de salida cada 10 ciclos de reloj dependiendo de la entrada proporcionada previamente.

Los resultados obtenidos se adjuntan en la siguiente tabla y como se puede observar para valores de la salida más pequeños se obtiene una peor aproximación

Tabla 3:Comparativa entre los resultados obtenidos con la red neuronal diseñada en Matlab y la implementada en la FPGA

Experimento	**Resultado Red Neuronal**	**Resultado FPGA**
Experimento 1	-8.5	-6
Experimento 2	-12.47	-11
Experimento 3	-13.65	-13

ANEXO A – CÓDIGO VHDL

```
----------------------------------------------------------------------------------
-- Company: Universidad de Sevilla
-- Engineer: Sandra Muñoz Lara
--
-- Create Date:    13:24:03 05/21/2018
-- Design Name:
-- Module Name:    RED
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision: 20/6
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_unsigned.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity RED is
Port(    clk:in std_logic;
               reset:in std_logic;

               --Entradas procedentes del resto de sensores
               E1 : in  STD_LOGIC_VECTOR(19 DOWNTO 0);
               E2 : in  STD_LOGIC_VECTOR(19 DOWNTO 0);
```

```
               E3 : in  STD_LOGIC_VECTOR(19 DOWNTO 0);
               E4 : in  STD_LOGIC_VECTOR(19 DOWNTO 0);
               E5 : in  STD_LOGIC_VECTOR(19 DOWNTO 0);
               E6 : in  STD_LOGIC_VECTOR(19 DOWNTO 0);
               E7 : in  STD_LOGIC_VECTOR(19 DOWNTO 0);
               E8 : in  STD_LOGIC_VECTOR(19 DOWNTO 0);
               E9 : in  STD_LOGIC_VECTOR(19 DOWNTO 0);
               E10 : in  STD_LOGIC_VECTOR(19 DOWNTO 0);
               E11 : in  STD_LOGIC_VECTOR(19 DOWNTO 0);
               E12 : in  STD_LOGIC_VECTOR(19 DOWNTO 0);
               E13 : in  STD_LOGIC_VECTOR(19 DOWNTO 0);
               E14 : in  STD_LOGIC_VECTOR(19 DOWNTO 0);

              --Valor del sensor estimado
           Y : out STD_LOGIC_VECTOR(19 DOWNTO 0));

end RED;

architecture Behavioral of RED is


--------------------FUNCIONES-----------------------------------------------------------
------Look up pesos-------------------------------------------------------------------
FUNCTION lookuppesos(señal : in std_logic_vector(19 downto 0);
                                n_señal : integer ;
                                n_neurona : integer)RETURN std_logic_vector IS

        VARIABLE peso: integer;
        VARIABLE bias: integer;
        VARIABLE valor: std_logic_vector(19 downto 0);

        BEGIN
            -----LOOK UP-PESOS---
            case n_neurona is

            -------------Neuronas capa entrada------------------------
            --Neurona 1--
            when 1 =>
                        case n_señal is
                                    when 1 => peso := 0;
                                    when 2 => peso := 20;
                                    when 3 => peso := 27;
                                    when 4 => peso := -23;
                                    when 5 => peso := 28;
                                    when 6 => peso := 35;
                                    when 7 => peso := 52;
                                    when 8 => peso := -74;
                                    when 9 => peso := 59;
                                    when 10 => peso := -4;
                                    when 11 => peso := 16;
                                    when 12 => peso := -61;
                                    when 13 => peso := 11;
                                    when 14 => peso := -13;
                                    when others => peso := 0;
                        end case;

            --Neurona 2--
```

```
when 2 =>
            case n_señal is
                        when 1 => peso := 44;
                        when 2 => peso := -97;
                        when 3 => peso := 80;
                        when 4 => peso := 35;
                        when 5 => peso := 2;
                        when 6 => peso := 54;
                        when 7 => peso := 48;
                        when 8 => peso := 17;
                        when 9 => peso := 47;
                        when 10 => peso := -52;
                        when 11 => peso := 78;
                        when 12 => peso := 12;
                        when 13 => peso := 5;
                        when 14 => peso := -19;
                        when others => peso := 0;
            end case;

--Neurona 3--
when 3 =>
            case n_señal is
                        when 1 => peso := 87;
                        when 2 => peso := -46;
                        when 3 => peso := 27;
                        when 4 => peso := 38;
                        when 5 => peso := -7;
                        when 6 => peso := 11;
                        when 7 => peso := -8;
                        when 8 => peso :=-28;
                        when 9 => peso := -26;
                        when 10 => peso := -1;
                        when 11 => peso := -11;
                        when 12 => peso := 43;
                        when 13 => peso := 68;
                        when 14 => peso := -47;
                        when others => peso := 0;
            end case;

------Neurona capa de salida-----
--Neurona 1--
when 4 =>
            case n_señal is
                        when 1 => peso := 108;
                        when 2 => peso := -133;
                        when 3 => peso := 138;
                        when others => peso := 0;
            end case;


when others => peso := 0;
end case;

valor:=std_logic_vector(to_signed(to_integer(signed(señal))*peso,20));

RETURN valor;
```

```
            END lookuppesos;
-------------------------------------------------------------------------------



--Lookup-bias----------------------------------------------------------------------
FUNCTION lookupbias(n_neurona : integer)RETURN integer IS

        VARIABLE bias: integer;

        BEGIN

            -----LOOK UP-BIAS
            case n_neurona is

                        ------Neurona capa de entrada--------
                        --Neurona 1--
                        when 1 => bias := -147;

                        --Neurona 2--
                        when 2 => bias := 442;

                        --Neurona 3--
                        when 3 => bias := 853;

                        -----Neurona capa de salida---------
                        when 4 => bias := -210;

                        when others => bias := 0;

            end case;
            RETURN bias;
            END lookupbias;
-------------------------------------------------------------------------------



-----------------------FUNCIONES NORMALIZACION---------------------------------------
-----------------------NORMALIZACION DIRECTA-----------------------------------------

FUNCTION f_norm_directa(entrada: STD_LOGIC_VECTOR ( 19 DOWNTO 0) ; n_señal: integer
)RETURN STD_LOGIC_VECTOR  IS

        VARIABLE salida: STD_LOGIC_VECTOR ( 19 DOWNTO 0);
        VARIABLE X_Min : integer ;
        VARIABLE X_Max : integer ;
        VARIABLE Y_Min : integer ;
        VARIABLE Y_Max : integer      ;

       BEGIN

       Y_Min := -1;
       Y_Max := 1;

                   case n_señal is

                        ------Neurona capa de entrada--------
                        --Neurona 1--
```

```
                        when 1 => X_Min := 10;
                                  X_Max := 147;

                        --Neurona 2--
                        when 2 => X_Min := -19;
                                  X_Max := 84;

                        --Neurona 3--
                        when 3 => X_Min := 10;
                                  X_Max := 183;
                        -----Neurona capa de salida---------
                        when 4 => X_Min := -42
                                   X_Max := 85;

                        -----Neurona capa de salida---------
                        when 5 => X_Min := 12;
                                  X_Max := 144;

                        -----Neurona capa de salida---------
                        when 6 => X_Min := -15
                                  X_Max := 96;

                        -----Neurona capa de salida---------
                        when 7 => X_Min := 11;
                                 X_Max := 179;

                        -----Neurona capa de salida---------
                        when 8 => X_Min := -35;
                                 X_Max := 96;

                        -----Neurona capa de salida---------
                        when 9 => X_Min := 14;
                                 X_Max := 195;

                        -----Neurona capa de salida---------
                        when 10 => X_Min := 26;
                                  X_Max := 164;

                        -----Neurona capa de salida---------
                        when 11 => X_Min := 20;
                                   X_Max := 128;

                        -----Neurona capa de salida---------
                        when 12 => X_Min := 23;
                                  X_Max := 143;

                        -----Neurona capa de salida---------
                        when 13 => X_Min := -34;
                                  X_Max := -13;

                        ----Neurona capa de salida---------
                        when 14 => X_Min := -33;
                                   X_Max := -13;

                        when others => X_Min := 0;
                                      X_Max := 0;
```

```
            end case;

            salida:=std_logic_vector(to_signed(
((to_integer(signed(entrada)) - X_Min)*(Y_Max-Y_Min)*10/(X_Max-X_Min))-10 ,20
));

            RETURN salida;
            END f_norm_directa;
-------------------------------------------------------------------

----------------------NORMALIZACION INVERSA--------------------------------


FUNCTION f_norm_inversa(entrada: STD_LOGIC_VECTOR ( 19 DOWNTO 0))RETURN
STD_LOGIC_VECTOR  IS

        VARIABLE salida: STD_LOGIC_VECTOR ( 19 DOWNTO 0);
        VARIABLE X_Min_Inv : integer ;
        VARIABLE X_Max_Inv : integer ;
        VARIABLE Y_Min_Inv : integer ;
        VARIABLE Y_Max_Inv : integer  ;

       BEGIN

       Y_Min_Inv := -1000;
       Y_Max_Inv := 1000;
       X_Min_Inv := -42;
       X_Max_Inv := -5;



       salida:=std_logic_vector(to_signed(           (((to_integer(signed(entrada)))
- Y_Min_Inv)*(X_Max_Inv - X_Min_Inv)/(Y_Max_Inv - Y_Min_Inv)+(X_Min_Inv)) ,20)
);

            RETURN salida;
            END f_norm_inversa;
-----------------------------------------------------------------------------------


-------------------------FUNCIONES ACTIVACION-----------------------------------
----F.Activ Escalon-------------------------------------------------------------
FUNCTION f_escalon(entrada: STD_LOGIC_VECTOR ( 19 DOWNTO 0) )RETURN STD_LOGIC_VECTOR
IS

        VARIABLE salida: STD_LOGIC_VECTOR ( 19 DOWNTO 0);

        BEGIN
            if ( to_integer(signed(entrada)) <= 0 ) then
            salida:="00000000000000000000";
            else
            salida:="00000000000000001010";
            end if;

            RETURN salida;
            END f_escalon;
-----------------------------------------------------------------------------------
```

```
----F.Activ Lineal-------------------------------------------------------------------
FUNCTION f_lineal(entrada: STD_LOGIC_VECTOR ( 19 DOWNTO 0) )RETURN STD_LOGIC_VECTOR
IS

        VARIABLE salida: STD_LOGIC_VECTOR ( 19 DOWNTO 0);
        BEGIN
            if ( to_integer(signed(entrada)) <= 0 ) then
            salida:=entrada;
            else
            salida:=entrada;
            end if;

            RETURN salida;
            END f_lineal;
-------------------------------------------------------------------------------------

----F.Activ Sigmoidal----------------------------------------------------------------
FUNCTION f_sigmoidal(entrada: STD_LOGIC_VECTOR ( 19 DOWNTO 0) )RETURN
STD_LOGIC_VECTOR  IS

        VARIABLE salida: integer;
        VARIABLE salida_esc_10: STD_LOGIC_VECTOR ( 19 DOWNTO 0);
        BEGIN
            if ( to_integer(signed(entrada)) < -1700 ) then
            salida_esc_10:="00000000000000000000";
            elsif( to_integer(signed(entrada)) > 1700 )then
            salida_esc_10:="00000000000000001010";
            else

            --Calculo el valor en rango -1000 a 1000
            salida:=(20*(to_integer(signed(entrada)) +1700)/34 )-1000;

            --Convierto el valor en rango 0-10 : valores por redondeo
            case salida is

            when -1000 to -950 =>   salida_esc_10 := "11111111111111110101" ;
            when -949 to -850 =>   salida_esc_10 := "11111111111111110110" ;
            when -849 to -750 =>   salida_esc_10 := "11111111111111110111" ;
            when -749 to -650 =>   salida_esc_10 :="11111111111111111000" ;
            when -649 to -550 =>   salida_esc_10 :="11111111111111111001" ;
            when -549 to -450 =>   salida_esc_10 :="11111111111111111010" ;
            when -449 to -350 =>   salida_esc_10 :="11111111111111111011" ;
            when -349 to -250 =>   salida_esc_10 :=  "11111111111111111100" ;
            when -249 to -150 =>   salida_esc_10 := "11111111111111111101" ;
            when -149 to -50 =>   salida_esc_10 :=  "11111111111111111110" ;
            when -49 to 0 =>   salida_esc_10 :=  "00000000000000000000";
            when 1 to 50 =>   salida_esc_10 := "00000000000000000000" ;
            when 51 to 150 =>   salida_esc_10 := "00000000000000000001" ;
            when 151 to 250 =>   salida_esc_10 := "00000000000000000010" ;
            when 251 to 350 =>   salida_esc_10 := "00000000000000000011" ;
            when 351 to 450 =>   salida_esc_10 := "00000000000000000100" ;
            when 451 to 550 =>   salida_esc_10 := "00000000000000000101" ;
            when 551 to 650 =>   salida_esc_10 := "00000000000000000110" ;
            when 651 to 750 =>   salida_esc_10 := "00000000000000000111" ;
            when 751 to 850 =>   salida_esc_10 := "00000000000000001000" ;
            when 851 to 950 =>   salida_esc_10 := "00000000000000001001" ;
```

```
            when 951 to 1000 =>   salida_esc_10 := "0000000000000001010" ;

            when others => salida_esc_10 := "00000000000000000000";
            end case;
            end if;

            RETURN salida_esc_10;
            END f_sigmoidal;
-------------------------------------------------------------------------------------

-------------DECLARACIÓN DE SEÑALES-------------------------------------------------
--Entradas a la normalizacion
signal ent_x_1:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal ent_x_2:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal ent_x_3:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal ent_x_4:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal ent_x_5:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal ent_x_6:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal ent_x_7:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal ent_x_8:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal ent_x_9:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal ent_x_10:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal ent_x_11:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal ent_x_12:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal ent_x_13:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal ent_x_14:STD_LOGIC_VECTOR(19 DOWNTO 0);


--Entradas a las neuronas de la capa de entrada
signal x_1:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal x_2:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal x_3:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal x_4:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal x_5:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal x_6:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal x_7:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal x_8:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal x_9:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal x_10:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal x_11:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal x_12:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal x_13:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal x_14:STD_LOGIC_VECTOR(19 DOWNTO 0);

--Entrada a la funcion de activacion
signal ent_n1,p_ent_n1:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal ent_n2,p_ent_n2:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal ent_n3,p_ent_n3:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal ent_n4,p_ent_n4:STD_LOGIC_VECTOR(19 DOWNTO 0);

--Salidas de la funcion de activacion
signal sal_n1,p_sal_n1:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal sal_n2,p_sal_n2:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal sal_n3,p_sal_n3:STD_LOGIC_VECTOR(19 DOWNTO 0);
signal sal_n4,p_sal_n4:STD_LOGIC_VECTOR(19 DOWNTO 0);

--Salidas de la red
```

```
signal salidared,p_salidared:STD_LOGIC_VECTOR(19 DOWNTO 0);

---------------------------FIN DECLARACIÓN DE SEÑALES--------------------------------




begin

-------------------------COMIENZO DEL PROCESO SINCRONO-----------------------------
--Actualizaré el estado en el que me encuentro y el valor de cada una de las señales
mediante un proceso sincrono
sincrono:process(reset,clk,E1,E2,E3,E4,E5,E6,E7,E8,E9,E10,E11,E12,E13,E14)
	begin

	if(reset='1') then--si se activa el reset

		Y<="00000000000000000000";

		ent_x_1<=E1;
		ent_x_2<=E2;
		ent_x_3<=E3;
		ent_x_4<=E4;
		ent_x_5<=E5;
		ent_x_6<=E6;
		ent_x_7<=E7;
		ent_x_8<=E8;
		ent_x_9<=E9;
		ent_x_10<=E10;
		ent_x_11<=E11;
		ent_x_12<=E12;
		ent_x_13<=E13;
		ent_x_14<=E14;



		ent_n1<="00000000000000000000";
		ent_n2<="00000000000000000000";
		ent_n3<="00000000000000000000";
		ent_n4<="00000000000000000000";


		sal_n1<="00000000000000000000";
		sal_n2<="00000000000000000000";
		sal_n3<="00000000000000000000";
		sal_n4<="00000000000000000000";

		salidared<="00000000000000000000";


	elsif (rising_edge(clk)) then

		-----Actualizamos todas las señales a sus valores actuales-----
```

```
            Y <=p_salidared;

            ent_x_1<=E1;
            ent_x_2<=E2;
            ent_x_3<=E3;
            ent_x_4<=E4;
            ent_x_5<=E5;
            ent_x_6<=E6;
            ent_x_7<=E7;
            ent_x_8<=E8;
            ent_x_9<=E9;
            ent_x_10<=E10;
            ent_x_11<=E11;
            ent_x_12<=E12;
            ent_x_13<=E13;
            ent_x_14<=E14;

            ent_n1<=p_ent_n1;
            ent_n2<=p_ent_n2;
            ent_n3<=p_ent_n3;
            ent_n4<=p_ent_n4;

            sal_n1<=p_sal_n1;
            sal_n2<=p_sal_n2;
            sal_n3<=p_sal_n3;
            sal_n4<=p_sal_n4;

            salidared<=p_salidared;


      end if;
end process;
----------------------------FINAL DEL PROCESO SINCRONO-------------------------------




-----------------------------------PROCESO COMBINACIONAL---------------------------

comb:process(ent_n1,ent_n2,ent_n3,ent_n4,sal_n1,sal_n2,sal_n3,sal_n4,x_1,x_2,x_3,x_4,
x_5,x_6,x_7,x_8,x_9,x_10,x_11,x_12,x_13,x_14,ent_x_1,ent_x_2,ent_x_3,ent_x_4,ent_x_5,
ent_x_6,ent_x_7,ent_x_8,ent_x_9,ent_x_10,ent_x_11,ent_x_12,ent_x_13,ent_x_14,salidare
d)

 VARIABLE num_entrada: integer;
 VARIABLE num_neurona: integer;
      begin

--------------------------------FUNCIONES-----------------------------------------
---------------NORMALIZACION-------------------------------------
            x_1<=f_norm_directa(ent_x_1,1);
            x_2<=f_norm_directa(ent_x_2,2);
            x_3<=f_norm_directa(ent_x_3,3);
            x_4<=f_norm_directa(ent_x_4,4);
            x_5<=f_norm_directa(ent_x_5,5);
            x_6<=f_norm_directa(ent_x_6,6);
            x_7<=f_norm_directa(ent_x_7,7);
```

```
            x_8<=f_norm_directa(ent_x_8,8);
            x_9<=f_norm_directa(ent_x_9,9);
            x_10<=f_norm_directa(ent_x_10,10);
            x_11<=f_norm_directa(ent_x_11,11);
            x_12<=f_norm_directa(ent_x_12,12);
            x_13<=f_norm_directa(ent_x_13,13);
            x_14<=f_norm_directa(ent_x_14,14);


------Neuronas de la capa de entrada-----------------------------------------------------
------NEURONA 1-------------------------------------------
p_ent_n1<=std_logic_vector( to_signed (to_integer(signed(lookuppesos(x_1,1,1)))+
to_integer(signed(lookuppesos(x_2,2,1)))+ to_integer(signed(lookuppesos(x_3,3,1)))+
to_integer(signed(lookuppesos(x_4,4,1)))+ to_integer(signed(lookuppesos(x_5,5,1)))+
to_integer(signed(lookuppesos(x_6,6,1)))+ to_integer(signed(lookuppesos(x_7,7,1)))+
to_integer(signed(lookuppesos(x_8,8,1)))+ to_integer(signed(lookuppesos(x_9,9,1)))+
to_integer(signed(lookuppesos(x_10,10,1)))+
to_integer(signed(lookuppesos(x_11,11,1)))+
to_integer(signed(lookuppesos(x_12,12,1)))+
to_integer(signed(lookuppesos(x_13,13,1)))+
to_integer(signed(lookuppesos(x_14,14,1)))+  lookupbias(1) , 20 ));

p_sal_n1<=f_sigmoidal(ent_n1);
-----------------------------------------------------------



------NEURONA 2-------------------------------------------
p_ent_n2<=std_logic_vector( to_signed (to_integer(signed(lookuppesos(x_1,1,2)))+
to_integer(signed(lookuppesos(x_2,2,2)))+ to_integer(signed(lookuppesos(x_3,3,2)))+
to_integer(signed(lookuppesos(x_4,4,2)))+ to_integer(signed(lookuppesos(x_5,5,2)))+
to_integer(signed(lookuppesos(x_6,6,2)))+ to_integer(signed(lookuppesos(x_7,7,2)))+
to_integer(signed(lookuppesos(x_8,8,2)))+ to_integer(signed(lookuppesos(x_9,9,2)))+
to_integer(signed(lookuppesos(x_10,10,2)))+
to_integer(signed(lookuppesos(x_11,11,2)))+
to_integer(signed(lookuppesos(x_12,12,2)))+
to_integer(signed(lookuppesos(x_13,13,2)))+
to_integer(signed(lookuppesos(x_14,14,2)))+  lookupbias(2) , 20 ));

p_sal_n2<=f_sigmoidal(ent_n2);
-----------------------------------------------------------



------NEURONA 3-------------------------------------------
p_ent_n3<=std_logic_vector( to_signed (to_integer(signed(lookuppesos(x_1,1,3)))+
to_integer(signed(lookuppesos(x_2,2,3)))+ to_integer(signed(lookuppesos(x_3,3,3)))+
to_integer(signed(lookuppesos(x_4,4,3)))+ to_integer(signed(lookuppesos(x_5,5,3)))+
to_integer(signed(lookuppesos(x_6,6,3)))+ to_integer(signed(lookuppesos(x_7,7,3)))+
to_integer(signed(lookuppesos(x_8,8,3)))+ to_integer(signed(lookuppesos(x_9,9,3)))+
to_integer(signed(lookuppesos(x_10,10,3)))+
to_integer(signed(lookuppesos(x_11,11,3)))+
to_integer(signed(lookuppesos(x_12,12,3)))+
to_integer(signed(lookuppesos(x_13,13,3)))+
to_integer(signed(lookuppesos(x_14,14,3)))+  lookupbias(3) , 20 ));

p_sal_n3<=f_sigmoidal(ent_n3);
```

```
-------------------------------------------------------


-------------------------------------------------------------------------------------------




------Neuronas de la capa de salida-----------------------------------------------------
------NEURONA 4-------------------------------------------
p_ent_n4<=std_logic_vector( to_signed (to_integer(signed(lookuppesos(sal_n1,1,4)))+
to_integer(signed(lookuppesos(sal_n2,2,4)))+
to_integer(signed(lookuppesos(sal_n3,3,4)))+ lookupbias(4) , 20 ));

p_sal_n4<=f_lineal(ent_n4);
-------------------------------------------------------




--------------------NORMALIZACION INVERSA---------------------

p_salidared<=f_norm_inversa(sal_n4);

-------------------------------------------------------------------------------------------
end process;
end Behavioral;
```

ANEXO B– SIMULACIÓN EN VHDL

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   15:44:47 06/18/2018
-- Design Name:
-- Module Name:   C:/Users/Sandra/Desktop/SNAKE - copia/Simu_red_norm.vhd
-- Project Name:  SNAKE
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: RED_1
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY SIMU_RED IS
END SIMU_RED;
```

```
ARCHITECTURE behavior OF SIMU_RED IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT RED
    PORT(
         clk : IN  std_logic;
         reset : IN  std_logic;
         E1 : IN  std_logic_vector(19 downto 0);
         E2 : IN  std_logic_vector(19 downto 0);
         E3 : IN  std_logic_vector(19 downto 0);
         E4 : IN  std_logic_vector(19 downto 0);
         E5 : IN  std_logic_vector(19 downto 0);
         E6 : IN  std_logic_vector(19 downto 0);
         E7 : IN  std_logic_vector(19 downto 0);
         E8 : IN  std_logic_vector(19 downto 0);
         E9 : IN  std_logic_vector(19 downto 0);
         E10 : IN  std_logic_vector(19 downto 0);
         E11 : IN  std_logic_vector(19 downto 0);
         E12 : IN  std_logic_vector(19 downto 0);
         E13 : IN  std_logic_vector(19 downto 0);
         E14 : IN  std_logic_vector(19 downto 0);
         Y : OUT  std_logic_vector(19 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal clk : std_logic := '0';
   signal reset : std_logic := '0';
   signal E1 : std_logic_vector(19 downto 0) := (others => '0');
   signal E2 : std_logic_vector(19 downto 0) := (others => '0');
   signal E3 : std_logic_vector(19 downto 0) := (others => '0');
   signal E4 : std_logic_vector(19 downto 0) := (others => '0');
   signal E5 : std_logic_vector(19 downto 0) := (others => '0');
   signal E6 : std_logic_vector(19 downto 0) := (others => '0');
   signal E7 : std_logic_vector(19 downto 0) := (others => '0');
   signal E8 : std_logic_vector(19 downto 0) := (others => '0');
   signal E9 : std_logic_vector(19 downto 0) := (others => '0');
   signal E10 : std_logic_vector(19 downto 0) := (others => '0');
   signal E11 : std_logic_vector(19 downto 0) := (others => '0');
   signal E12 : std_logic_vector(19 downto 0) := (others => '0');
   signal E13 : std_logic_vector(19 downto 0) := (others => '0');
   signal E14 : std_logic_vector(19 downto 0) := (others => '0');

      --Outputs
   signal Y : std_logic_vector(19 downto 0);

   -- Clock period definitions
   constant clk_period : time := 10 ns;

BEGIN

      -- Instantiate the Unit Under Test (UUT)
   uut: RED PORT MAP (
          clk => clk,
          reset => reset,
```

```
          E1 => E1,
          E2 => E2,
          E3 => E3,
          E4 => E4,
          E5 => E5,
          E6 => E6,
          E7 => E7,
          E8 => E8,
          E9 => E9,
          E10 => E10,
          E11 => E11,
          E12 => E12,
          E13 => E13,
          E14 => E14,
          Y => Y
        );

   -- Clock process definitions
   clk_process :process
   begin
             clk <= '0';
             wait for clk_period/2;
             clk <= '1';
             wait for clk_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
        reset<='1';
             wait for 100 ns;
             reset<='0';
      wait for clk_period*10;

------EXPERIMENTO 1-------------

             E1<="0000000000000111101";
             E2<="0000000000000100110";
             E3<="0000000000000010011";
             E4<="0000000000000100110";
             E5<="0000000000000111010";
             E6<="0000000000000001000";
             E7<="0000000000000001101";
             E8<="0000000000000001010";
             E9<="0000000000001011101";
             E10<="0000000000000101101";
             E11<="0000000000000110110";
             E12<="0000000000000011010";
             E13<="1111111111111110001";
             E14<="1111111111111110011";


                 wait for clk_period*10;
```

```
------EXPERIMENTO 2-------------

            E1<="0000000000000111100";
            E2<="0000000000000011011";
            E3<="0000000000000010001";
            E4<="0000000000000011100";
            E5<="0000000000000111011";
            E6<="0000000000000010010";
            E7<="0000000000000001111";
            E8<="0000000000000010011";
            E9<="0000000000001100010";
            E10<="0000000000001000010";
            E11<="0000000000000111110";
            E12<="0000000000000011101";
            E13<="1111111111111110010";
            E14<="1111111111111110100";

            wait for clk_period*10;

------EXPERIMENTO 3-------------

            E1<="0000000000000111100";
            E2<="0000000000000010110";
            E3<="0000000000000010000";
            E4<="0000000000000010111";
            E5<="0000000000000111100";
            E6<="0000000000000010110";
            E7<="0000000000000010000";
            E8<="0000000000000011100";
            E9<="0000000000001011111";
            E10<="0000000000001001100";
            E11<="0000000000001000000";
            E12<="0000000000000011101";
            E13<="1111111111111110010";
            E14<="1111111111111110100";
      wait;
   end process;

END;
```

ANEXO C– CREACIÓN DE LA RED HACIENDO USO DEL NEURAL NETWORK TOOLBOX MATLAB

```
%%ALMACENAJE DE LOS DATOS
simVar='EPSILON_DMS';
disp(sprintf('Generando redes para %s',simVar));
load(sprintf('%s.mat',simVar));
data=data';

%Vector de entradas para el entrenamiento
x_1=data([1:8 11 13 17 18 24 26],:);

%Vector de salida deseada para el entrenamiento
t_1=data(28,:);

%%CREACIÓN DE LA RED
%Creación la red vacia
red_1=feedforwardnet;

%%CONFIGURACIÓN DE LA RED
%Numero de entradas
red_1.numInputs=1

%Numero de tamaño de las entradas
red_1.inputs{1}.size=14

%Numero de capas
```

```
red_1.numLayers=2

%Numero de neuronas de la capa 1 Y 2
red_1.layers{1}.size=3
red_1.layers{2}.size=1

%Funcion de transferencia de las neuronas de la capa 1 y 2
red_1.layers{1}.transferFcn='tansig'
red_1.layers{2}.transferFcn='purelin'

%Función de entrenamiento
red_1.trainFcn='trainbr';

%Añado bias a las neuronas de la capa 1 y 2
red_1.biasConnect=[1;1];

%%INICIALIZACIÓN DE LOS PESOS Y EL BIAS DE LAS NEURONAS
%Peso de las neuronas de la capa 1
red_1.iw{1}=[1 1    1 1 1  1 1 1   1 1 1   1 1 1;
   1 1   1 1 1   1 1 1   1 1 1   1 1 1;
   1 1   1 1 1   1 1 1   1 1 1   1 1 1;];

%Bias de las neuronas de la capa 1
red_1.b{1}=[1 1 1]';

%Peso inicial de las neuronas de la capa 2
red_1.lw{2}=[1 1 1];

%Bias inicial de las neuronas de la capa 2
red_1.b{2}=1;

%%ENTRENAMIENTO DE LA RED
%Entrenamiento de la red
[red_1,tr_1]=train(red_1,x_1,t_1);

%Simulación de la red
sim(red_1,x_1(:,2))
```

ANEXO D– DISEÑO DE LA RED IMPLEMENTADA EN VHDL HACIENDO USO DEL NEURAL NETWORK TOOLBOX MATLAB

```
%%DISEÑO DE LA IMPLEMENTACIÓN EN VHDL DE LA RED NEURONAL

load('x_1.mat');

%%%%%%%%%%%%%%%%%%%%%%%%%Normalizacion Directa%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Valores maximos y minimos de x e y
min_x_1=[10;-19;10;-42;12;-15;11;-35;14;26;20;23;-34;-33]
max_x_1=[147;84;183;85;144;96;179;96;195;164;128;143;-13;-13]
max_y=1;
min_y=-1;


for i=1:length(x_1(:,3))
a(i)=x_1(i,3)-min_x_1(i,1);
a_2(i)=(a(i)*  ( (max_y-min_y)  /  (max_x_1(i,1)-min_x_1(i,1)) ))+min_y;
end
a_2_col=a_2'


%%%%%%%%%%%%%%%%%%%%%%%%%%%CAPA DE ENTRADA%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Pesos de la capa de entrada
Pesos_N1=[0;0.20;0.27;-0.23;0.28;0.35;0.52;-0.74;0.59;-0.04;0.16;-0.61;0.11;-0.13];
Pesos_N2=[0.44;-0.97;0.80;0.35;0.02;0.54;0.48;0.17;0.47;-0.52;0.78;0.12;0.05;-0.19];
Pesos_N3=[0.87;-0.46;0.27;0.38;-0.07;0.11;-0.08;-0.28;-0.26;-0.01;-0.11;0.43;0.68;-
0.47];

%Bias de la capa de entrada
Bias_N1=-0.15;
Bias_N2=0.44;
Bias_N3=0.85;

%Multiplicacion de los pesos y suma de los bias
Ent_N1=(a_2*Pesos_N1)+ Bias_N1
Ent_N2=(a_2*Pesos_N2) + Bias_N2
Ent_N3=(a_2*Pesos_N3)+ Bias_N3
%Funcion de activacion sigmoidal logsig

if(Ent_N1>1.75) Sal_N1=1;
elseif(Ent_N1<-1.7) Sal_N1=-1;
else
Sal_N1=(20*(Ent_N1+1.7)/34)-1;
end;

if(Ent_N2>1.7) Sal_N2=1;
elseif(Ent_N2<-1.7) Sal_N2=-1;
else
Sal_N2=(20*(Ent_N2+1.7)/34)-1;
end;

if(Ent_N3>1.7) Sal_N3=1;
elseif(Ent_N3<-1.7) Sal_N3=-1;
else
Sal_N3=(20*(Ent_N3+1.7)/34)-1;
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%CAPA DE SALIDA%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Pesos de la neurona de la capa de salida
Peso_N4=[1.08;-1.33;1.38];

%Bias de la neurona de la capa de salida
```

```
Bias_N4=-0.21;

%Funcion de activacion de la capa de salida: purelin
Ent_N4=([Sal_N1 Sal_N2 Sal_N3]*Peso_N4 ) + Bias_N4;
Sal_N4=Ent_N4;

%%%%%%%%%%%%%%%%%%%%%%%%%Normalizacion Inversa%%%%%%%%%%%%%%%%%%%%%%%%

%Valores maximos y minimos de x e y
Min_x_RevNorm=-42.1;
Max_x_RevNorm=-5.5;
max_y=1;
min_y=-1;

a_inv=Sal_N4-min_y;
a_2_inv=(a_inv*((Max_x_RevNorm-Min_x_RevNorm)/(max_y-min_y))+Min_x_RevNorm);


%Salida Red Neuronal
Salida=a_2_inv
```

REFERENCIAS

[1] F. S. Caparrini. [En línea] *http://www.cs.us.es/~fsancho/?e=54*

[2] F. S. Caparrini. [En línea]. *http://www.cs.us.es/~fsancho/?e=72*

[3] Juan Miguel Marín Diazaraque -Dept de Estadística UC3M - Introducción a las redes neuronales aplicadas *http://halweb.uc3m.es/esp/Personal/personas/jmmarin/esp/DM/tema3dm.pdf*.

[4] Mark Hudson Beale, Martin T. Hagan, Howard B. Demuth, *Neural Network Toolbox User´s Guide.* University.

[5] Fernando Mateo Jiménez -Tesis Doctoral- Redes neuronales y preprocesado de variables para modelos y sensores en bioingenieria *https://riunet.upv.es/bitstream/handle/10251/16702/tesisUPV3874.pdf*

[6] K. Neshatpour, M. Malik, M. A. Ghodrat and H. Homayoun, "Accelerating Big Data Analytics Using FPGAs," 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, Vancouver, BC, 2015, pp. 164-164, doi: 10.1109/FCCM.2015.59 C.A. Mead. Analog VLSI and neural systems. Addison-Wesley, Reading, MA, 1989.

[7] OCHOA,Susana,Características de una Red Neuronal Artificial, fecha de recuperación : 30 de julio de 2012 http://fluidos.eia.edu.co/hidraulica/articuloses/flujoentuberias/neuronal/neuronal_archivos/page0002.html

[8] F. S. Caparrini. [En línea]. Available: http://www.cs.us.es/~fsancho/?e=165..

[9] https://www.intel.es/content/www/es/es/products/docs/storage/programmable/applications/data-analytics.html

[10] Mark Hudson Beale, Martin T. Hagan, Howard B. Demuth, Neural Network Toolbox User´s Guide. University.

[11] C.A. Mead. Analog VLSI and neural systems. Addison-Wesley, Reading, MA, 1989..

[12] F. S. Caparrini. [En línea]. Available: http://www.cs.us.es/~fsancho/?e=165.

[13] https://www2.ulpgc.es/hege/almacen/download/38/38584/practica_ia_2.pdf.