ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

PROYECTO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

TEMA: CONTROLADOR LÓGICO PROGRAMABLE BÁSICO CON FPGA Y REDES NEURONALES

AUTOR: JORGE LUIS RAMÍREZ TORRES

DIRECTOR: ING. VÍCTOR PROAÑO, MSc.

CODIRECTOR: ING. ABG. DARWIN ALULEMA, MSc.

SANGOLQUÍ

ABRIL, 2015

UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE

INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

CERTIFICADO

Ing. Víctor Proaño, MSc.

Ing. Darwin Alulema, MSc.

CERTIFICAN

Que el presente Proyecto de grado titulado: "CONTROLADOR LÓGICO PROGRAMABLE BÁSICO CON FPGA Y REDES NEURONALES.", desarrollado en su totalidad por el señor JORGE LUIS RAMÍREZ TORRES, con CI: 1104088255, ha sido guiado y revisado periódicamente bajo nuestra dirección, cumpliendo con las normas estatuarias establecidas en el Reglamento de Estudiantes de la ESPE.

Sangolquí, 20 de Abril de 2015.

Atentamente:

Ing. Víctor Proaño.
DIRECTOR

Ing. Darwin Alulema.
CODIRECTOR

UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE

INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

AUTORÍA DE RESPOSABILIDAD

JORGE LUIS RAMÍREZ TORRES

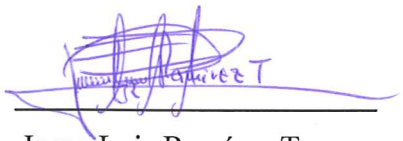
DECLARO QUE:

El proyecto de grado titulado: "CONTROLADOR LÓGICO PROGRAMABLE BÁSICO CON FPGA Y REDES NEURONALES.", ha sido desarrollado en base a una investigación exhaustiva, respetando derechos intelectuales de terceros, conforme a las citas que constan al pie de las correspondientes páginas, cuyas fuentes se encuentran en las referencias bibliográficas.

Consecuentemente el presente trabajo es de mi autoría.

En virtud de esta declaración, me responsabilizo del contenido, veracidad y alcance científico del presente proyecto de grado.

Sangolquí, 20 de Abril de 2015.



Jorge Luis Ramírez Torres

UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

INGENIERÍA EN ELECTRONICA, AUTOMATIZACIÓN Y CONTROL

AUTORIZACIÓN

Yo, Jorge Luis Ramírez Torres

Autorizo a la Universidad de las Fuerzas Armadas ESPE, la publicación en la biblioteca virtual de la institución el proyecto de grado titulado: "CONTROLADOR LÓGICO PROGRAMABLE BÁSICO CON FPGA Y REDES NEURONALES.", cuyo contenido, criterios e ideas son de mi exclusiva responsabilidad y autoría.

Sangolquí, 20 de Abril de 2015.

Jorge Luis Ramírez Torres

DEDICATORIA

La presente Tesis se la dedico a mi Dios. A mis padres por haber sido un pilar fundamental en mi formación profesional ya que sin ellos no habría llevado a cabo todos estos logros académicos que he podido alcanzar.

Con suficiente esfuerzo, entusiasmo y valentía se puede encontrar el camino adecuado. Ser valiente es cuando quieres descansar pero en su lugar solo sigues avanzando.

AGRADECIMIENTO

Agradezco a Dios, por haberme brindado la oportunidad de vivir y ser una persona con cualidades intelectuales y espirituales pues con ello pude llevar a cabo la presente.

A mis padres quienes supieron brindarme todo su apoyo de modo incondicional y entusiasta, en todo momento.

A mis maestros por haber compartido generosamente sus conocimientos, experiencia y haberse convertido en mi vida en un ejemplo a seguir.

A mis familiares, compañeros y amigos por haberme apoyado y haberme brindado su compañía y amistad incondicional.

TABLA DE CONTENIDOS

LISTA DE TABLAS

LISTA DE FIGURAS

RESUMEN

En el presente documento se estructura una RNA (Red Neuronal Artificial) en una tarjeta FPGA (Field Programmable Gate Array) XC3S500E haciendo uso del Lenguaje para Descripción y Modelado de Circuitos VHDL, para diseñar la funcionalidad de un Controlador Lógico Programable que consta de cuatro entradas digitales y una salida digital. La arquitectura de la RNA modelada sobre la tarjeta FPGA está conformada por cuatro neuronas en la capa de entrada, tres neuronas en la capa oculta y una neurona en la capa de salida. Se implementó el algoritmo de aprendizaje de retro-propagación en Matlab para el entrenamiento de la RNA mediante una HMI (Interfaz Humano Maquina) la cual requiere que el usuario ingrese una o más respuestas en relación a las entradas para hacer posible el entrenamiento. Mediante la misma HMI se transfieren los resultados del entrenamiento hacia la RNA implementada en la FPGA, brindando así la posibilidad de reprogramar el Controlador Lógico Programable según los requerimientos del caso. Para la transferencia de los datos en la tarjeta se implementó un módulo de recepción serial que se encarga de recibir los pesos, y direccionarlos a cada neurona, las mismas que tienen memoria propia, y la capacidad para determinar cuáles pesos les corresponden.

PALABRAS CLAVE:

FPGA
RNA
PLC
VLSI
CONROL
CHIP

ABSTRACT

In this paper it has been structure an Artificial Neural Network ANN into a FPGA(Field Programmable Gate Array) card XC3S500E using the Description and Modeling Language for VHDL circuits, with the purpose of designing the functionality of a Programmable Logic controller with four digital inputs and one digital output. The ANN architecture modeled on the FPGA card is structured by four neurons in the input layer, three neurons in the hidden layer and one neuron in the output layer. Besides the back-propagation learning algorithm was incorporated using Matlab, for the training of the ANN, using a HMI (Human Machine Interface) which requires the user to enter one or more outputs in relation to inputs for running the training. Using the same HMI training results are transferred to the RNA implemented in the FPGA, giving the chance to reprogram the Programmable Logic controller that has been incorporated, according to its requirements. For the data transfer, on the card was implemented a module of serial reception that is responsible to receive the weights, and direct to them to each neuron, thereof having own memory and the ability to determine which weights correspond to them.

KEY WORDS:

FPGA
ANN
PLC
VLSI
CONTROL
CHIP

CAPÍTULO I: INTRODUCCIÓN

1.1. ANTECEDENTES

En 1943, Warren McCullock y Walter Pitts, desarrollaron algoritmos matemáticos necesarios para posibilitar en sentido general el funcionamiento de una red neuronal artificial; posteriormente en 1949, Donal Hebb creó un algoritmo de aprendizaje para tales redes neuronales. Los trabajos de estos estudiosos se consideran actualmente como el origen de la inteligencia artificial.

La utilización que se ha venido dando a la IA (Inteligencia Artificial) está relacionada con una diversidad de ámbitos, algunos de los cuales son: Medicina en la ayuda al diagnóstico; Ingeniería en la optimización de procesos, diagnóstico de fallos, toma de decisiones; Informática en el procesado del lenguaje natural, criptografía, teoría de juegos, lingüística computacional; Robótica y automática en sistemas adaptativos de rehabilitación, interfaces cerebro computadora, sistemas de visión artificial, sistemas de navegación automática; así también para una serie de aplicaciones que requieren abstraer información para tomar decisiones complejas.

La inteligencia artificial es por definición una abstracción o software y por lo tanto es necesaria una plataforma o hardware para dar capacidad de generar efecto; entre los dispositivos electrónicos que brindan la posibilidad de servir como plataforma se encuentran la tarjeta FPGA (Field Programable Gate Array) que permite crear y emular diseños de hardware y el PLC (Controlador Lógico

Programable) que permite procesar señales eléctricas para realizar el control y automatización de procesos.

1.2. JUSTIFICACIÓN E IMPORTANCIA

Los Controladores Lógico Programables PLC tienen gran eficacia en su propósito, su funcionalidad y programación se rige a eventos predecibles con una lógica de funcionamiento exacta, y en esencia la CPU de un PLC procesa la información instrucción por instrucción.

Si en la funcionalidad de un PLC se pudiera implementar un sistema de auto programación frente a eventos abstractos y un procesamiento de varias instrucciones a la vez, entonces se ampliarían posibilidades de técnicas de control y automatización que involucren cierto nivel de inteligencia. Para ejecutar procesos de aprendizaje complejos y tareas con una elevada cantidad de variables, la inteligencia artificial necesita gran capacidad de procesamiento de información. Incluso en las tareas más básicas de automatización sería más fácil entrenar un sistema que programarlo, dado que un sistema capaz de aprender sería el equivalente a un sistema capaz de programarse de modo automático.

La implementación de RNA (Redes Neuronales Artificiales), permite que una aplicación pueda aprender de varios ejemplos para luego responder de forma adecuada a otros ejemplos similares, adaptándose a distintas situaciones. Resolver este tipo de aplicaciones resultaría difícil y extenso al implementarse caso por caso en la programación tradicional mediante código, debido a que en ella se realiza una programación estricta y específica para situaciones muy particulares, esta característica restringe la abstracción y generalidad que podría necesitar la solución de un problema.

Los sistemas implementados sobre FPGA, pueden presentar un incremento importante en la velocidad de procesamiento, debido a que permiten la ejecución de sentencias concurrentes, mientras que en una CPU convencional esto no es posible, pues las sentencias se ejecutan necesariamente de modo secuencial usando ciclos de reloj.

Al implementar RNA sobre una tarjeta FPGA, se espera un incremento en la velocidad de aprendizaje y capacidad para adaptarse de una aplicación, porque el dispositivo ejecuta tareas de manera simultánea, debido a su arquitectura; y a su principio de funcionamiento capaz de reconfigurar sus conexiones de hardware, con lo cual se tendrían dos parámetros importantes para la implementación de una RNA.

Implementar aplicaciones que sean mucho más rápidas y eficaces en un sistema embebido diseñando circuitos integrados, implicaría un incremento en la generación de riqueza utilizando conocimientos y capacidades intelectuales. El gobierno impulsa el cambio de la matriz productiva y esto requiere que las universidades promuevan el desarrollo de proyectos innovadores que involucren la capacidad intelectual de sus alumnos para crear, mejorando en el país la posibilidad de generar tecnología propia.

1.3. ALCANCE DEL PROYECTO

Se estructuró una Red Neuronal Artificial RNA en una tarjeta FPGA haciendo uso del Lenguaje para Descripción y Modelado de Circuitos VHDL, con el propósito de diseñar un Controlador Lógico Programable. Se implementó el algoritmo de aprendizaje de retro-propagación en Matlab, para el entrenamiento de la RNA, y un sistema de comunicación que permite transferir las condiciones de funcionamiento al sistema en otras palabras se transfieren los pesos, brindando la

posibilidad de reprogramar el Controlador Lógico Programable a implementar, según requiera el usuario.

En la filosofía de funcionamiento, el sistema es entrenado usando un algoritmo de aprendizaje. Se ingresa la tabla de verdad de un circuito combinacional, cuyos datos son utilizados por el algoritmo de aprendizaje como patrones de entrenamiento, para permitir que la RNA pueda aprender, es decir para generar los pesos. El sistema reacciona según la información que se le ha proporcionado, mostrando un comportamiento convergente a la tabla de verdad con la cual fue entrenado.

En el escenario de pruebas se puede evidenciar que la Red Neuronal Artificial RNA implementada se está comportando conforme lo requiera una aplicación para resolver un problema.

1.4. Objetivos

Objetivo general.

- Implementar un Controlador Lógico Programable básico con FPGA y Redes Neuronales Artificiales.

Objetivos específicos

- Diseñar una Red Neuronal Artificial sobre una tarjeta FPGA.

- Implementar un Algoritmo de Entrenamiento para la Red Neuronal Artificial.

- Implementar un Sistema de Transferencia de datos hacia la Red Neuronal Artificial.

- Evaluar la Red Neuronal Artificial, según su capacidad de aprendizaje y ejecución de una tarea.

- Definir un escenario de pruebas, para observar el comportamiento del sistema.

CAPÍTULO II: ESTADO DEL ARTE

2.1. REDES NEURONALES ARTIFICIALES

Las redes neuronales artificiales son modelos de comportamiento inteligente que pueden ser construidos como sistemas artificiales inspirados en el sistema nervioso de seres vivos. Las aplicaciones de sistemas basados en redes neuronales han permitido diseñar redes con propósitos específicos como el reconocimiento de patrones. Este esquema es novedoso en el área de la computación y es muy interesante dado que una computadora digital, aun la más sencilla, supera la velocidad y precisión del cerebro en la relación de operaciones numéricas; aunque las operaciones como reconocimiento de patrones, memoria asociativa y en general todas las relaciones con el comportamiento inteligente parecen imposibles de alcanzar mediante las concepciones computacionales tradicionales de la computación incluyendo el campo de la inteligencia artificial. (Maxines & Alcalá, 2002)

Las redes neuronales artificiales son mecanismos de procesamiento de datos, cuya idea fundamental está basada en emular las Redes Neuronales Biológicas, así las Redes Neuronales Artificiales necesitan aprender a reaccionar de formas variadas ante una gama de estímulos como sus homólogas biológicas. La unidad fundamental de una Red Neuronal Artificial, es la neurona y reacciona mediante una fórmula matemática que está formada por una suma ponderada y una función de activación que calcula un proceso básico dentro de todo el conjunto de procesos que se realizan en una Red Neuronal Artificial (RNA). El procesamiento en una RNA es el conjunto de los procesamiento individuales de todas y cada una de las neuronas que la

componen, intercambiando información de forma coordinada y organizada, de acuerdo a la arquitectura de la RNA en cuestión.

Existe una gran cantidad de problemas de ingeniería resueltos con diseños de redes neuronales artificiales capaces de emular en mayor o menor medida algunas funciones realizadas por el sistema nervioso de los seres vivos, como la discriminación de patrones. No obstante el éxito obtenido por las redes neuronales artificiales, no está comprobado que esta arquitectura computacional cumpla con el test de Turin y que, por consiguiente, sea la solución al problema de la generación de verdadera inteligencia artificial. Desde el punto de vista filosófico, las arquitecturas distribuidas y paralelas, como las redes neuronales, no son más que muchos procesadores tradicionales; desde este punto de vista, no poseen alguna propiedad nueva que evite las objeciones del test de Turin. (Maxines & Alcalá, 2002)

El tipo de procesamiento que realiza una RNA presenta una capacidad poco frecuente en las máquinas convencionales, ya que una RNA no siempre reacciona de manera exacta, sin embargo su respuesta se asemeja a la respuesta esperada. A una RNA no necesariamente se la debe entrenar para todos y cada uno de los casos de un evento a resolverse, aun así la RNA determina la respuesta adecuada a un nuevo caso, basándose en casos anteriores.

En la actualidad se han logrado avances importantes en la implementación de RNA’s que permiten realizar procesamiento de datos de manera inteligente utilizando nuevos sistemas en los cuales no solo se han implementado algoritmos materializados en software sino también ha sido posible materializar las RNA en hardware, donde la arquitectura del hardware es precisamente la red neuronal.

El 7 de agosto de 2014 la empresa IBM (International Business Machines) ha introducido un nuevo chip al cual le ha denominado neurón chip, o chip cognitivo, el cual posee una tecnología inspirada en el cerebro humano. El chip funciona con 1 millón de neuronas y 256 millones de sinapsis además resulta ser el más grande de los chip que IBM ha construido pues consta de 5.4 mil millones de transistores, la red implementada en el chip tiene 4096 núcleos neurosinápticos. Además el consumo de potencia es relativamente bajo pues consume la ínfima cantidad de 70mW durante su funcionamiento. Esta tecnología presenta grandes posibilidades para aplicaciones con necesidades sensoriales, así como también para expandirse más allá de los límites conocidos de la supercomputación distribuida (IBM Corporation, 2014).

"*The architecture can solve a wide class of problems from vision, audition, and multi-sensory fusion, and has the potential to revolutionize the computer industry by integrating brain-like capability into devices where computation is constrained by power and speed*" (Dharmendra S. Modha, 2014).

Los modelos de redes neuronales están ofreciendo nuevos enfoques para resolver problemas, mismos que pueden simular en máquinas de propósito específico, aceleradores de hardware con arquitectura neuronal incluso en computadoras convencionales. A fin de alcanzar velocidades de procesamiento máximo pueden utilizarse arreglos con base en tecnología de procesamiento óptico o en VLSI de silicio. El aspecto fundamental reside en utilizar el modelo neuronal de procesamiento paralelo en tiempo real con una enorme cantidad de procesadores sencillos. (Maxines & Alcalá, 2002)

Las aplicaciones de las redes neuronales y de la computación son muy abundantes. De entre ellos se destacan en el campo de procesamiento de señales o reconocimiento de patrones, la extracción de características, la inspección industrial, el pronóstico de negocios, la clasificación de crédito, la selección de seguridad, el diagnóstico

médico, el procesamiento de voz, la comprensión del lenguaje natural, el control de robots y la adaptación de procesos de control. (Maxines & Alcalá, 2002)

2.2. CLASIFICACIÓN DE LAS REDES NEURONALES ARTIFICIALES

Las Redes Neuronales Artificiales ó RNA pueden ser clasificadas según, la estructura de la red, su algoritmo de aprendizaje y su tipo de entrada, aquellas clasificaciones se detallan a continuación:

2.2.1. Según la estructura de la red

Se tiene como referencia el tipo de interconexión existente entre las neuronas que conforman la Red Neuronal Artificial, y se tiene en cuenta la dirección del flujo de datos, clasificándose de la siguiente manera:

2.2.1.1. Redes de propagación hacia adelante

La interconexión entre las neuronas que conforman la RNA, está dada de tal manera que no existe retroalimentación de datos, es decir la salida de una neurona está conectada a la entrada de otra que no ha participado en la generación de dicha salida, y bajo este principio de funcionamiento se pueden mencionar RNA Monocapa y Multicapa.

- Monocapa._ Se denomina así cuando el número de salidas es igual al número de neuronas que conforman la Red, y las salidas no se conectan a las entradas; algunos ejemplos de estas RNA son: Adaline y Perceptron.

- Multicapa._ Se denomina así cuando algunas neuronas reciben en sus entradas las salidas de otras, siempre y cuando no exista retroalimentación; el perceptron multicapa es un claro de ejemplo de este tipo de RNA.

2.2.1.2. Redes Recurrentes

La interconexión entre las neuronas que conforman la RNA, está dada de tal manera que existe retroalimentación de datos, es decir que por lo menos una salida de una neurona está conectada a la entrada de otra que ha participado en la generación de dicha salida; la red de HopField es un claro ejemplo de este tipo de interconexión de RNA.

2.2.2. Según su algoritmo de aprendizaje

Se hace referencia al método a través del cual se le proporciona la información necesaria a una RNA, para su adecuado funcionamiento, clasificándose de la siguiente manera:

2.2.2.1. Aprendizaje Supervisado

Los datos para el funcionamiento de la RNA son proporcionados a partir de un conjunto de datos (entradas-salidas), sin embargo no es necesario conocer todo el conjunto de datos posibles, pues la red tiene la capacidad de generalizar. Este tipo de RNA no puede adquirir un nuevo aprendizaje a partir de su propio funcionamiento para modificar el comportamiento de la red como tal; se puede mencionar como ejemplos a la red Adaline y el perceptrón multicapa.

2.2.2.2. Aprendizaje Auto-organizado

La RNA aprende del entorno en el cual esté en funcionamiento y no requiere de un conjunto de datos previos.

2.2.2.3. Aprendizaje Reforzado

En este tipo de RNA se combina el aprendizaje Supervisado y el Auto-organizado, es decir que a la RNA se le proporciona un conjunto de datos previos y también ella puede aprender paulatinamente del entorno en el cual se encuentra en funcionamiento.

2.2.3. Según su tipo de entrada

Se divide a las RNA según la naturaleza de la señal que va a ser conectada en su entrada y se clasifican de la siguiente manera:

2.2.3.1. Analógicas

Procesan datos analógicos, es decir señales de naturaleza continua.

2.2.3.2. Discretas

Procesan señales de tipo digital, es decir tienen valores puntuales, por lo general suelen ser 1, 0 y -1.

2.3. ALGORITMOS DE ENTRENAMIENTO

Una característica prominente de las RNA es su capacidad para aprender a través del entrenamiento de manera similar al sistema nervioso central de cualquier organismo vivo. Tiene capacidades de aprendizaje para reaccionar a estímulos externos ya sean conocidos o desconocidos, basándose en experiencias previas y en la arquitectura misma de red.

Un algoritmo de entrenamiento es un procedimiento para otorgar a la RNA de una capacidad de aprendizaje; uno de los algoritmos más utilizados es el de Retro-propagación. El aprendizaje le permite a la RNA proporcionar una respuesta adecuada frente a eventos iguales o similares al de su entrenamiento.

2.3.1. Algoritmo de Retro-propagación

Este algoritmo permite realizar el tipo de aprendizaje supervisado por lo tanto los requerimientos principales son: un banco de datos entradas vs salidas, una arquitectura de red bien definida y de propagación hacia adelante. Teniendo en cuenta estos parámetros la función del algoritmo es básicamente una actualización iterativa de los pesos de cada neurona basándose en el error medio cuadrático.

Se aplica un estímulo a las entradas de la RNA, las cuales son procesadas a través de todas las capas hasta generar una respuesta. La respuesta de la RNA es comparada con la respuesta deseada, se calcula el error y este se propaga hacia las capas previas de la RNA. Cada neurona recibirá una parte del error proporcional al porcentaje de contribución de dicha neurona a la respuesta total. Posteriormente

generan una variación de los pesos, para irlos sintonizando de tal forma que estos generen una respuesta adecuada.

Se le adjudica un conjunto de parámetros de funcionamiento a la RNA, los cuales se denominan pesos; para tener mejores resultados es recomendable colocar los valores de los pesos de manera aleatoria y con valores pequeños.

Para calcular el estado de la RNA al haberle asignado el conjunto de pesos aleatorios se le realiza una prueba de funcionamiento, obteniendo así una respuesta y un error con respecto a la respuesta esperada. El propósito del algoritmo es hacer que el error sea muy bajo. En la Figura 1 se ilustra la iteración fundamental que realiza el algoritmo para determinar el error y todos los parámetros que requiere para sus derivadas parciales, las mismas que son esenciales para determinar la variación de cada peso.

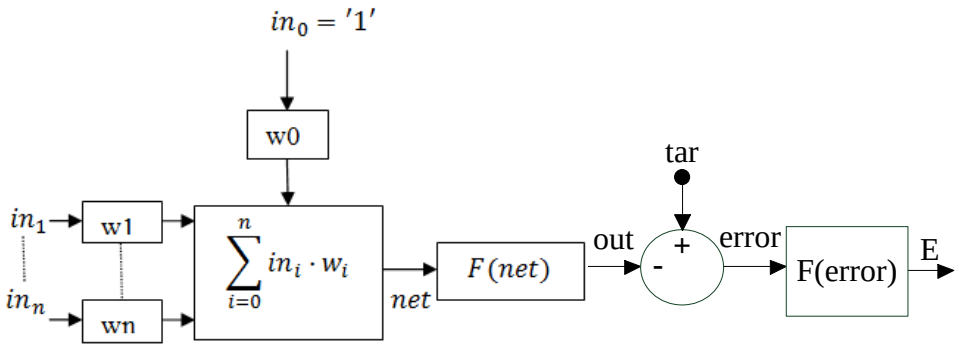


Figura 1. Diagrama fundamental de E para Retro-propagación

En la ecuación (1), se muestra la expresión que determina la función del error medio cuadrático.

$$E = F(error) = \frac{1}{2} \cdot \text{error}^2 \qquad (1)$$

En la ecuación (2) se muestra la función error.

$$error = tar - out \qquad (2)$$

Donde tar es la respuesta deseada y out es la respuesta actual de la RNA.

En la ecuación (3) se muestra la expresión que determina la función de net, la misma que es la función sigmoidea.

$$F(net) = out = \frac{1}{1 + e^{-net \cdot \alpha}} \qquad (3)$$

Donde α, determina la pendiente de esta función; se la considera una constante.

A la sumatoria $\sum_{i=0}^{n} in_i \cdot w_i$ que calcula el valor de net se la puede representar como se muestra en la ecuación (4):

$$net = in_1 \cdot w_1 + in_2 \cdot w_2 \ldots\ldots\ldots + in_n \cdot w_n + in_0 \cdot w_0 \qquad (4)$$

Donde el valor de in0 siempre será ‘1’ quedando así w0 representando el umbral de la neurona sin embargo se lo considera un peso más para facilidad de cálculos. in_1 hasta in_n, representan las entradas de la neurona, dichas entradas en ciertos casos pueden ser la salidas de otras neuronas. w_0 hasta w_n representan los pesos de las neuronas.

“La base del algoritmo backpropagation para la modificación de los pesos es la técnica conocida como gradiente decreciente.” (Introducción a las redes neuronales aplicadas, 2007)

“El gradiente toma la dirección que determina el incremento más rápido en el error, mientras que la dirección opuesta, es decir, la dirección negativa, determina el decremento más rápido en el error.” (Introducción a las redes neuronales aplicadas, 2007)

“A nivel práctico, la forma de modificar los pesos de forma iterativa consiste en aplicar la regla de la cadena a la expresión del gradiente y añadir una tasa de aprendizaje η.” (Introducción a las redes neuronales aplicadas, 2007)

La variación del peso w en la posición i, Δw_i se puede determinar al colocar un coeficiente de aprendizaje η multiplicado por el gradiente en decremento, es decir $\frac{\partial \mathrm{E}}{\partial \mathrm{w}_i}$ con signo negativo.

$$\Delta w_i = -\eta \cdot \frac{\partial \mathrm{E}}{\partial \mathrm{w}_i} \qquad (5)$$

Al aplicar la regla de la cadena, la expresión de Δw_i se muestra en la ecuación (6).

$$\Delta w_i = -\eta \cdot \left[\frac{\partial \mathrm{E}}{\partial \mathrm{error}}\right] \cdot \left[\frac{\partial \mathrm{error}}{\partial \mathrm{out}}\right] \cdot \left[\frac{\partial \mathrm{out}}{\partial \mathrm{net}}\right] \cdot \left[\frac{\partial \mathrm{out}}{\partial \mathrm{w}_i}\right] \qquad (6)$$

Al resolver cada una de las derivadas parciales de la ecuación (6) se obtiene la ecuación (7), que muestra la variación de un peso en particular, en una sola iteración.

$$\Delta w_i = -\eta \cdot [\mathrm{error}] \cdot [-1] \cdot [\mathrm{out} \cdot \alpha \cdot (1 - \mathrm{out})] \cdot [\mathrm{in}_i] \qquad (7)$$

Para una adecuada comprensión, los factores entre corchetes de la ecuación (7) están en el mismo orden que los factores entre corchetes de la ecuación (6).

Es importante aclarar que este mecanismo se realiza una cantidad de veces igual a la longitud del vector de salida tar.

2.4. DISPOSITIVOS FPGA

Un dispositivo FPGA (Field Programmable Gate Array) o en español (Arreglo de compuertas programables en el campo) es un dispositivo lógico de hardware reconfigurable in situ, es decir permite realizar una configuración de las compuertas lógicas inherentes a sí en el instante en el cual el programador lo requiere; permitiendo describir y modelar hardware sobre su arquitectura.

El tipo de hardware que se puede describir y modelar en un FPGA va desde compuertas lógicas hasta complejos circuitos embebidos en un solo chip. La limitación está dada por las capacidades del dispositivo FPGA utilizado. Algunas de las características técnicas de la tarjeta FPGA XC3S500E se ilustran en la *Tabla 1*.

Tabla 1.

Características técnicas de la FPGA XC3S500E.

System Gates	**500K**
Equivalent Logic Cells	10476
Distributed RAM bits	73K
Block RAM bits	360K
Dedicated Multipliers	20
DCMs	4
Maximum User I/O	232
Maximum Differential I/O Pairs	92
CLB Array (One CLB = Four Slices):	
Rows	46
Columns	34
Total CLBs	1164
Total Slices	4656

Buenos aportes sobre las FPGA han sido realizados por (Jorge A. Quiñones R., 2011) como se verá a continuación:

i. Los dispositivos FPGA (Arreglos de compuertas programables en campo) se basan principalmente de arreglos de compuertas y dentro de su arquitectura contienen tres elementos configurables(Figura 2):

ii. - Bloques lógicos configurables (CLB).
iii. - Bloques de entrada y salida (IOB).
iv. - Canales de comunicación.

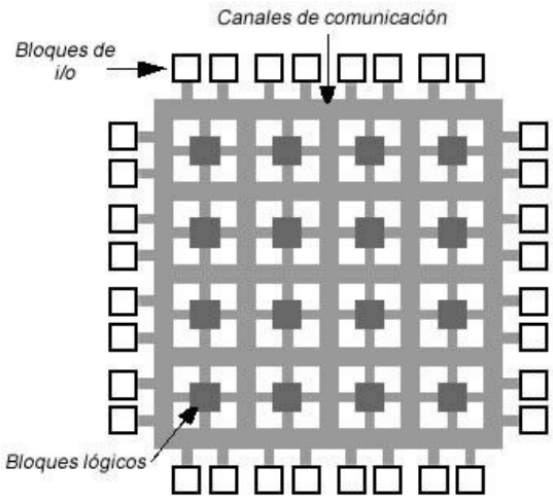


Figura 2. Arquitectura interna de una FPGA. (Jorge A. Quiñones R., 2011)

v. Internamente los bloques lógicos configurables están alambrados por los canales de comunicación de flujo y el datos son llevados por los bloques de entrada y salida de las terminales del FPGA. El CLB contiene generadores de funciones, biestables y multiplexores para rutear las señales dentro del CLB. Los generadores de funciones se implementan como tablas de consulta mejor conocidos como LUT(Lookup Tables). Una LUT básicamente es una memoria ROM reprogramable con 16 palabras de un bit.
vi. Para la programación de los FPGA se utilizan lenguajes de descripción de hardware (HDL) que permiten diseñar y depurar un sistema digital con un

alto grado de abstracción. Dentro de estos lenguajes los más populares en la industria son el VHDL y el Verilog.

vii. El Lenguaje VHDL permite describir un sistema digital en diferentes niveles de abstracción:

viii. 1. Por su comportamiento

ix. 2. Flujo de datos

x. 3. Estructural

xi. Una de las ventajas de VHDL es que la metodología de diseño es Arriba-Abajo (Top-Down), que permite describir un sistema a partir de su estructura general a una particular.

El dispositivo FPGA XC3S500E utilizado es una tarjeta modular, cuyos componentes más utilizados se ilustran en la Figura 3.

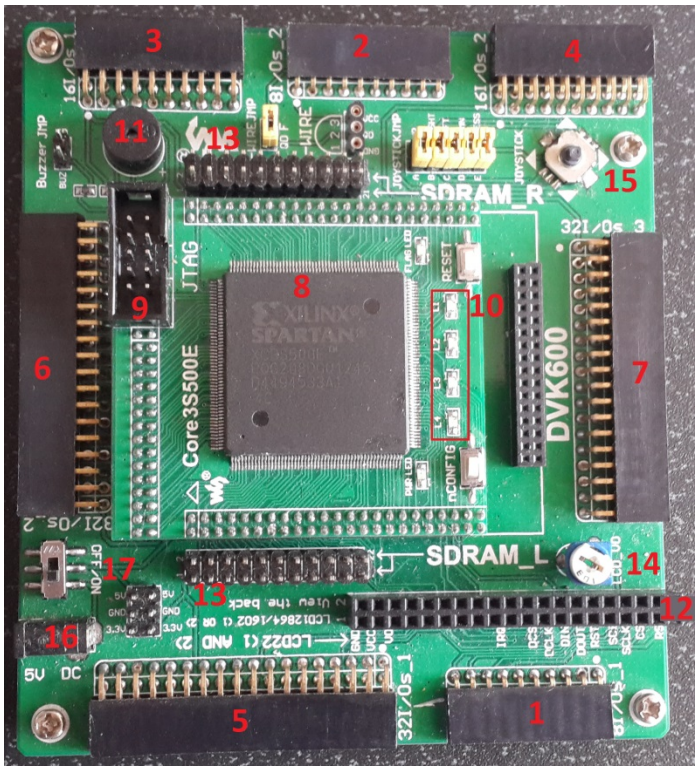


Figura 3. Tarjeta FPGA XC3S500E PQG208

La descripción de los componentes de acuerdo a la numeración de la Figura 3, se muestran en la *Tabla 2*.

Tabla 2.

Descripción de componentes de la FPGA.

Denominación		Descripción	
1	8I/Os_1	Puertos con 8 pines digitales que pueden ser configurados como entradas o salidas.	Se usan para conectar accesorios adicionales (módulos)
2	8I/Os_2		
3	16I/Os_1	Puertos con 16 pines digitales que pueden ser configurados como entradas o salidas.	
4	16I/Os_2		
5	32I/Os_1	Puertos con 32 pines digitales que pueden ser configurados como entradas o salidas.	
6	32I/Os_2		
7	32I/Os_3		
8	Procesador FPGA SPARTAN 3E XC3S500E	El dispositivo tiene las siguientes características: -Frecuencia de Reloj: 50MHz -Voltaje de operación: 1.15V-3.3V -Empaquetado: PQG208 -Entradas/Salidas: 116 -RAM: 360kb -DCMs: 4 -Puerto de programación compatible con JTAG	
9	JTAG	Da la interfaz para la programación de la tarjeta	
10	Leds 1,2,3,4	Indicadores luminosos para la interfaz entre el usuario y la tarjeta.	
11	Buzzer	Es un zumbador, sirve para emitir sonidos.	
12	Interfaz LCD	Para conectar LCD22, LCD12864, LCD1602	
13	SDRAM	Para conectar una tarjeta SDRAM. También sirve como conector de expansión para FPGA o CPLD.	
14	Potenciómetro	Para ajustar el brillo del LCD22 y LCD12864, o para ajustar el contraste del LCD1602.	
15	Joystick	Es una palanca de mando de 5 posiciones	
16	Fuente de alimentación	Conexión a 5V DC.	
17	Switch de encendido	Encendido y apagado de la tarjeta.	

CAPÍTULO III: DESARROLLO DEL SISTEMA

El sistema completo queda implementado como se muestra en la Figura 4, donde se ilustran dos bloques principales, Emisor y Receptor. En el Emisor se realiza el entrenamiento de una RNA que posee la misma arquitectura que la que está implementada en la FPGA. Una vez entrenada es decir ya obtenidos los pesos, éstos se almacenan en un vector y se los envía mediante comunicación serial hacia la FPGA que contiene dos módulos principales: una RNA y un sistema de recepción.

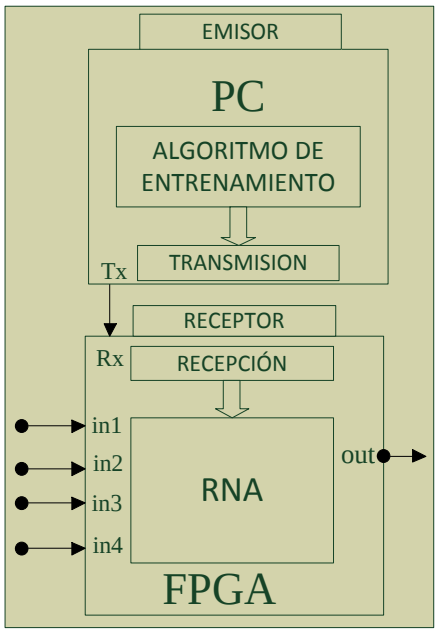


Figura 4. Diagrama de elementos que constituyen todo el sistema.

Para mostrar el desarrollo del sistema es importante indicar el funcionamiento, el mismo que se rige al proceso mostrado en el diagrama de bloques de la Figura 5.

Figura 5. Diagrama de bloques de Funcionamiento del Sistema

A continuación se muestra una descripción de cada uno de los procesos mostrados en la Figura 5:

- **Entrenamiento:** Es ejecutado en Matlab, mediante una interfaz gráfica que permite observar el progreso del entrenamiento de la RNA y el cálculo de los pesos correspondientes.

- **Transferencia:** La misma interfaz gráfica, permite realizar la Transferencia de los pesos hacia una tarjeta FPGA XC3s500E, mediante comunicación serial y el dispositivo USB to Serial PL2303.

- **Puesta en Marcha:** El dispositivo es capaz de recibir los pesos y redirigirlos a la RNA, para que sean almacenados en la neurona que corresponda; posterior e inmediatamente el dispositivo muestra el comportamiento de la RNA deseado.

En la Figura 6 se ilustran las entradas y salidas del sistema embebido en la FPGA y sus bloques de funcionamiento interno.

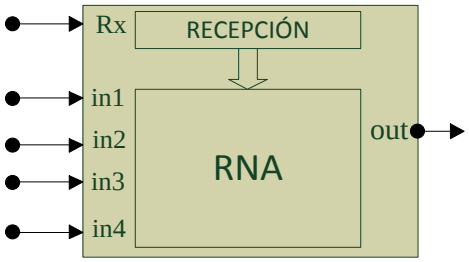


Figura 6. Sistema Embebido en la FPGA.

En general la RNA se puede describir como una caja negra cuyo funcionamiento depende del entrenamiento que haya recibido y va a presentar una respuesta discreta out según los valores de las entradas in1- in4 que son también discretos.

3.1. Arquitectura de la RNA

El tipo de Red Neuronal es totalmente conectada, es decir que todas sus entradas están conectadas a todas las neuronas de la capa de entrada y que todas las salidas de las neuronas de cualquier capa están conectadas a las entradas de las neuronas de la siguiente capa. La RNA es de tipo discreta, de propagación hacia adelante, multicapa, y de aprendizaje supervisado. La arquitectura de la Red Neuronal Artificial implementada en la tarjeta FPGA XC3s500E se ilustra en la Figura 7.

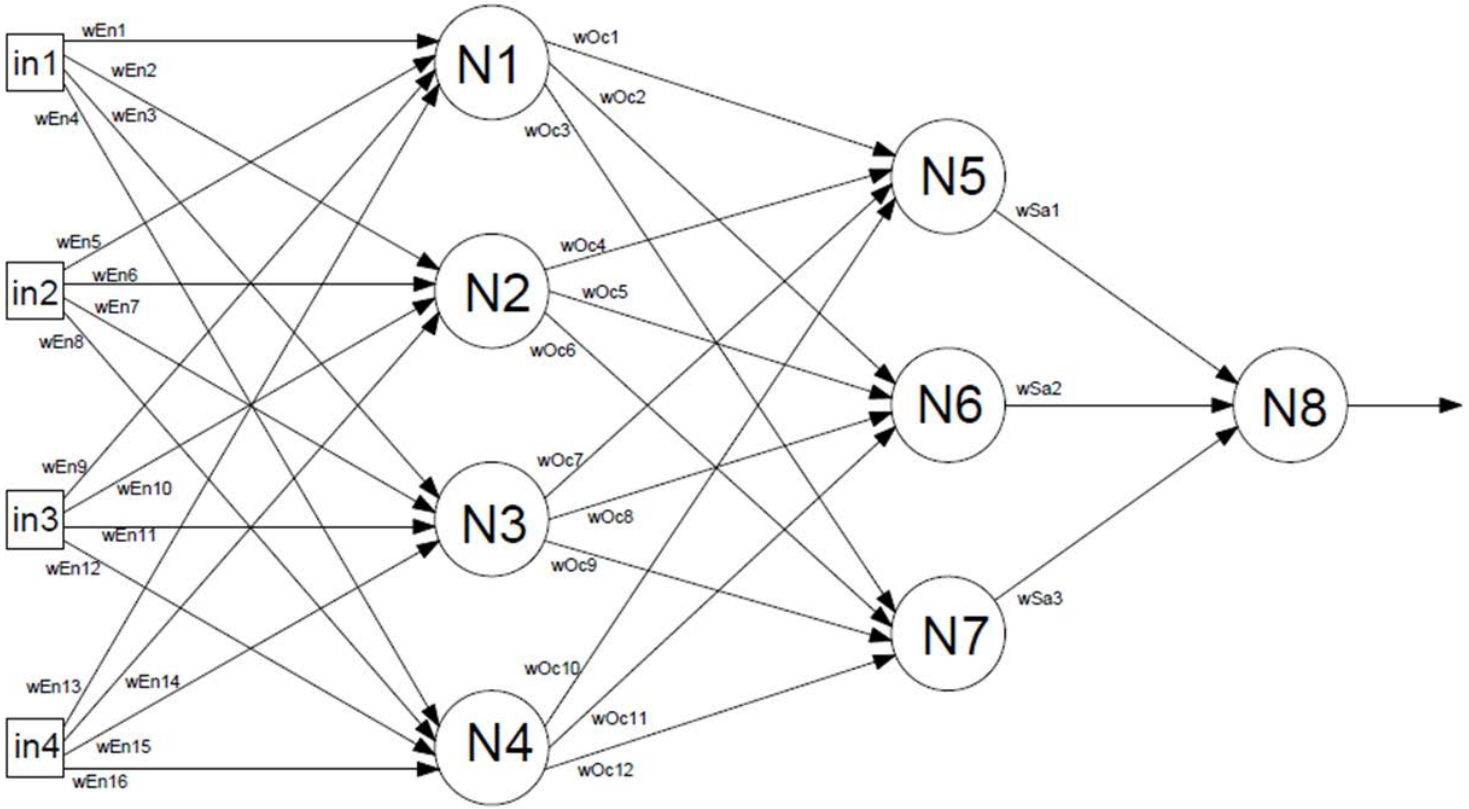


Figura 7. Arquitectura de la Red Neuronal Artificial

En la *Tabla 3*, *Tabla 4*, y *Tabla 5* y se muestra la denominación y distribución de los pesos conforme a las rutas que le corresponden a cada uno teniendo en cuenta la arquitectura de la RNA ilustrada en la Figura 7.

Tabla 3. Denominación de los pesos de las conexiones hacia la Capa de Entrada.

Pesos(w)	Desde la Entrada 1		Desde la Entrada 2		Desde la Entrada 3		Desde la Entrada 4		Equivalentes al umbral	
Hasta la Neurona 1	wEn1:	in1 --> N1	wEn5:	in2 --> N1	wEn9:	in3 --> N1	wEn13:	in4 --> N1	wEn17:	in0 --> N1
Hasta la Neurona 2	wEn2:	in1 --> N2	wEn6:	in2 --> N2	wEn10:	in3 --> N2	wEn14:	in4 --> N2	wEn18:	in0 --> N2
Hasta la Neurona 3	wEn3:	in1 --> N3	wEn7:	in2 --> N3	wEn11:	in3 --> N3	wEn15:	in4 --> N3	wEn19:	in0 --> N3
Hasta la Neurona 4	wEn4:	in1 --> N4	wEn8:	in2 --> N4	wEn12:	in3 --> N4	wEn16:	in4 --> N4	wEn20:	in0 --> N4
Total: 20 pesos(w)										

Tabla 4. Denominación de los pesos de las conexiones hacia la Capa Oculta.

Pesos(w)	Desde la Neurona 1		Desde la Neurona 2		Desde la Neurona 3		Desde la Neurona 4		Equivalentes al umbral	
Hasta la Neurona 5	wOc1:	N1 --> N5	wOc4:	N2 --> N5	wOc7:	N3 --> N5	wOc10:	N4 --> N5	wOc13:	in0 --> N5
Hasta la Neurona 6	wOc2:	N1 --> N6	wOc5:	N2 --> N6	wOc8:	N3 --> N6	wOc11:	N4 --> N6	wOc14:	in0 --> N6
Hasta la Neurona 7	wOc3:	N1 --> N7	wOc6:	N2 --> N7	wOc9:	N3 --> N7	wOc12:	N4 --> N7	wOc15:	in0 --> N7
Total: 15 pesos(w)										

Tabla 5. Denominación de los pesos de las conexiones hacia la Capa de Salida.

Pesos(w)	Desde la Neurona 5		Desde la Neurona 6		Desde la Neurona 7		Equivalentes al umbral	
Hasta la Neurona 8	wSa1:	N5 --> N8	wSa2:	N6 --> N8	wSa3:	N7 --> N8	wSa4:	in0 --> N8
Total: 4 pesos(w)								

Nomenclatura de Ejemplo: wEn1: in1 -- > N1, se lee que el peso de conexión desde la entrada “in1” hasta la “Neurona1” se denomina wEn1.

3.2. Algoritmo de aprendizaje

Se utiliza el algoritmo de aprendizaje Retro-propagación, y su funcionamiento consiste en calcular la salida de la RNA utilizando pesos aleatorios y determinar el error de dicha respuesta en relación a la respuesta deseada. Mediante derivadas parciales se calculan los errores cometidos por cada una de las neuronas que conforman la RNA, dicho error es proporcional al porcentaje de contribución de cada una de las neuronas a la respuesta total. Se calcula en qué valor se debe incrementar o disminuir cada uno de los pesos, estas variaciones se denominan deltas y son sumadas o restadas al peso que le corresponde. Se recalcula la salida con los nuevos pesos y se repite varias veces el proceso hasta que el error será muy pequeño. El procedimiento se implementó en Matlab y se ilustra mediante el diagrama de flujo de la Figura 8.

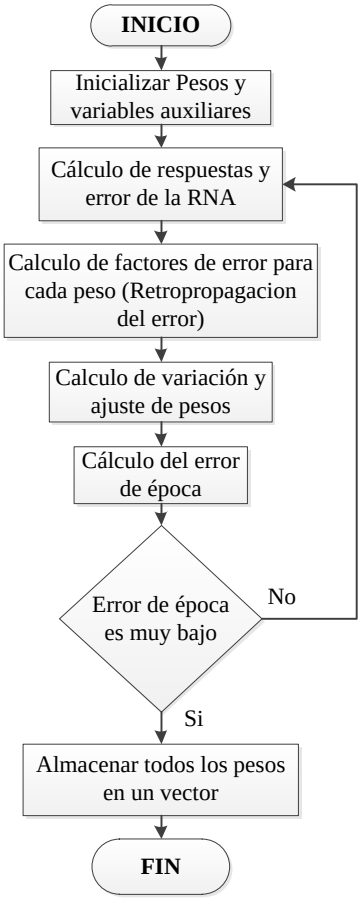


Figura 8. Diagrama de Flujo del Algoritmo de aprendizaje Retro-propagación

3.2.1. Inicializar Variables auxiliares.

Para ejecutar el Algoritmo de Retro-propagación se requiere variables con valores iniciales, que van a determinar los parámetros fundamentales de funcionamiento del algoritmo. Se requiere una matriz cuyos elementos son las entradas con su correspondiente salida esperada, y se ilustra en la Tabla 6. El vector salida corresponde a un ejemplo ya que puede variar según los requerimientos del usuario el mismo que tiene la posibilidad de cambiar dichos valores.

Tabla 6.

Inicialización de los vectores de entrada y salida

Vectores Entrada				**Vector Salida**
in1	in2	in3	in4	tar
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Se colocaron, un factor de aprendizaje de 0.1, una entrada in0 con un valor lógico “1”, y un factor de estiramiento con el valor 4. El factor de aprendizaje es un factor multiplicativo que permite controlar porcentualmente la variación de los pesos para evitar excesivas oscilaciones del error y para que la velocidad de aprendizaje no

sea excesivamente lenta. La entrada in0 permite implementar el umbral de activación de cada neurona, será constante y con el mismo valor para todas. El factor de estiramiento determina la pendiente de la función de activación sigmoidea, una pendiente muy elevada puede producir excesivas oscilaciones en el error de cada época, y una muy baja puede poner muy lento el proceso de aprendizaje.

3.2.2. Inicializar pesos.

El algoritmo de Retro-propagación sigue un procedimiento ordenado de etapas; una de sus primeras etapas requiere calcular la respuesta de la RNA, y para dar dicha respuesta se requiere tener inicializados los pesos correspondientes, por lo cual es importante proporcionar dichos datos de manera aleatoria al iniciar el entrenamiento. Los valores con los cuales se inicializaron los pesos de la capa de entrada, capa oculta y capa de salida se ilustran en la *Tabla 8*, *Tabla 7* y *Tabla 9*, respectivamente.

Tabla 7.

Inicialización de los pesos de la capa oculta.

Capa Oculta			
Nombre del Peso		**Valor del Peso**	**NEURONA**
wOc	1	0,2	Neurona 5
wOc	2	0,2	Neurona 6
wOc	3	0,5	Neurona 7
wOc	4	0,3	Neurona 5
wOc	5	0,4	Neurona 6
wOc	6	0,5	Neurona 7
wOc	7	0,5	Neurona 5
wOc	8	0,6	Neurona 6
wOc	9	0,2	Neurona 7
wOc	10	0,5	Neurona 5
wOc	11	0,6	Neurona 6
wOc	12	0,2	Neurona 7
wOc	13	0,4	Neurona 5
wOc	14	0,5	Neurona 6
wOc	15	0,7	Neurona 7

Tabla 8.

Inicialización de los pesos de la capa de entrada.

Capa de entrada			
Nombre del Peso		**Valor del Peso**	**NEURONA**
wEn	1	0,4	Neurona 1
wEn	2	0,5	Neurona 2
wEn	3	0,7	Neurona 3
wEn	4	0,7	Neurona 4
wEn	5	0,4	Neurona 1
wEn	6	0,5	Neurona 2
wEn	7	0,7	Neurona 3
wEn	8	0,7	Neurona 4
wEn	9	0,4	Neurona 1
wEn	10	0,5	Neurona 2
wEn	11	0,7	Neurona 3
wEn	12	0,7	Neurona 4
wEn	13	0,4	Neurona 1
wEn	14	0,5	Neurona 2
wEn	15	0,7	Neurona 3
wEn	16	0,7	Neurona 4
wEn	17	0,4	Neurona 1
wEn	18	0,5	Neurona 2
wEn	19	0,7	Neurona 3
wEn	20	0,7	Neurona 4

Tabla 9.

Inicialización de los pesos de la capa de salida.

Capa de Salida			
Nombre del Peso		**Valor del Peso**	**NEURONA**
wSa	1	0,5	Neurona 8
wSa	2	0,4	Neurona 8
wSa	3	0,5	Neurona 8
wSa	4	0,4	Neurona 8

3.2.3. Calculo de Respuestas y error de la RNA.

Antes de iniciar el entrenamiento se necesita saber en qué condiciones de funcionamiento se encuentra la RNA, entonces se calcula la respuesta que proporciona con los pesos aleatorios que en ella se han inicializado, posteriormente se calcula el error de su salida con respecto a la salida deseada.

Para calcular la repuesta de la Neurona 8, la misma que es también la respuesta de toda la RNA, se procede de la siguiente manera:

$$net = \mathrm{wSa1} \cdot \mathrm{out5} + \mathrm{wSa2} \cdot \mathrm{out6} + \mathrm{wSa3} \cdot \mathrm{out7} + \mathrm{wSa4} \cdot \mathrm{in0} \qquad (8)$$

$$out = \frac{1}{1 + e^{-net \cdot \alpha}} \qquad (9)$$

Para calcular la repuesta de la Neurona 7 se procede de la siguiente manera:

$$net7 = \mathrm{wOc3} \cdot \mathrm{out1} + \mathrm{wOc6} \cdot \mathrm{out2} + \mathrm{wOc9} \cdot \mathrm{out3} + \mathrm{wOc12} \cdot \mathrm{out4} + \mathrm{wOc15} \cdot \mathrm{in0} \qquad (10)$$

$$out7 = \frac{1}{1 + e^{-net7 \cdot \alpha}} \qquad (11)$$

Para calcular la repuesta de la Neurona 6 se procede de la siguiente manera:

$$net6 = \mathrm{wOc2} \cdot \mathrm{out1} + \mathrm{wOc5} \cdot \mathrm{out2} + \mathrm{wOc8} \cdot \mathrm{out3} + \mathrm{wOc11} \cdot \mathrm{out4} + \mathrm{wOc14} \cdot \mathrm{in0} \qquad (12)$$

$$out6 = \frac{1}{1 + e^{-net6 \cdot \alpha}} \quad (13)$$

Para calcular la repuesta de la Neurona 5 se procede de la siguiente manera:

$$net5 = \text{wOc1} \cdot \text{out1} + \text{wOc4} \cdot \text{out2} + \text{wOc7} \cdot \text{out3} + \text{wOc10} \cdot \text{out4} + \text{wOc13} \cdot \text{in0} \quad (14)$$

$$out5 = \frac{1}{1 + e^{-net5 \cdot \alpha}} \quad (15)$$

Para calcular la repuesta de la Neurona 4 se procede de la siguiente manera:

$$net4 = \text{wEn4} \cdot \text{in1} + \text{wEn8} \cdot \text{in2} + \text{wEn12} \cdot \text{in3} + \text{wEn16} \cdot \text{in4} + \text{wEn20} \cdot \text{in0} \quad (16)$$

$$out4 = \frac{1}{1 + e^{-net4 \cdot \alpha}} \quad (17)$$

Para calcular la repuesta de la Neurona 3 se procede de la siguiente manera:

$$net3 = \text{wEn3} \cdot \text{in1} + \text{wEn7} \cdot \text{in2} + \text{wEn11} \cdot \text{in3} + \text{wEn15} \cdot \text{in4} + \text{wEn19} \cdot \text{in0} \quad (18)$$

$$out3 = \frac{1}{1 + e^{-net3 \cdot \alpha}} \quad (19)$$

Para calcular la repuesta de la Neurona 2 se procede de la siguiente manera:

$$net2 = \text{wEn2} \cdot \text{in1} + \text{wEn6} \cdot \text{in2} + \text{wEn10} \cdot \text{in3} + \text{wEn14} \cdot \text{in4} + \text{wEn18} \cdot \text{in0} \quad (20)$$

$$out2 = \frac{1}{1 + e^{-net2 \cdot \alpha}} \quad (21)$$

Para calcular la repuesta de la Neurona 1 se procede de la siguiente manera:

$$net1 = \text{wEn1} \cdot \text{in1} + \text{wEn5} \cdot \text{in2} + \text{wEn9} \cdot \text{in3} + \text{wEn13} \cdot \text{in4} + \text{wEn17} \cdot \text{in0} \quad (22)$$

$$out1 = \frac{1}{1 + e^{-net1 \cdot \alpha}} \quad (23)$$

Al factor de estiramiento α, de las funciones de activación sigmoideas se lo considera como una constante en todos los cálculos.

El cálculo del error está dado por la ecuación(24).

$$error = tar - out \quad (24)$$

Donde tar corresponde a la respuesta ideal esperada de la RNA, mientras que out es la respuesta real que la RNA está proporcionando.

El error medio cuadrático está dado por la ecuación (25).

$$E = \frac{1}{2} \cdot error^2 \quad (25)$$

Para ingresar mediante código en Matlab a calcular las respuestas se realiza una suma ponderada que está regida por un bucle de repetición cuya variable incremental

es “i” y finaliza en la longitud del vector respuesta esperada tar que ingrese el usuario al momento de entrenar la RNA.

```
NEURONA 1
net4 = in0*wEn17 + in1(i)*wEn1 + in2(i)*wEn5 + in3(i)*wEn9  + in4(i)*wEn13;
out4 = sigmoide(net4,alpha);
NEURONA 2
net5 = in0*wEn18 + in1(i)*wEn2 + in2(i)*wEn6 + in3(i)*wEn10  + in4(i)*wEn14;
out5 = sigmoide(net5,alpha);
NEURONA 3
net6 = in0*wEn19 + in1(i)*wEn3 + in2(i)*wEn7 + in3(i)*wEn11  + in4(i)*wEn15;
out6 = sigmoide(net6,alpha);
NEURONA 4
net7 = in0*wEn20 + in1(i)*wEn4 + in2(i)*wEn8 + in3(i)*wEn12  + in4(i)*wEn16;
out7 = sigmoide(net7,alpha);
NEURONA 5
net1 = in0*wOc13 + out4*wOc1 + out5*wOc4 + out6*wOc7  + out7*wOc10;
out1 = sigmoide(net1,alpha);
NEURONA 6
net2 = in0*wOc14 + out4*wOc2 + out5*wOc5 + out6*wOc8 + out7*wOc11;
out2 = sigmoide(net2,alpha);
NEURONA 7
net3 = in0*wOc15 + out4*wOc3 + out5*wOc6 + out6*wOc9 + out7*wOc12;
out3 = sigmoide(net3,alpha);
NEURONA 8
net = in0*wSa4 + out1*wSa1 + out2*wSa2 +out3*wSa3;
out = sigmoide(net,alpha);
El error de la RNA se calcula de la siguiente manera:
outFinal(i) = out;  Para calcular el ultimo resultado del entrenamiento
error = tar(i) - out; Para calcular del error de cada iteración dentro de una época
```

3.2.4. Calculo de las variaciones de los pesos

Para sintonizar el funcionamiento de la RNA es indispensable determinar la variación que requiere cada uno de los pesos, ya que con dicha variación es posible calcular nuevos valores para los pesos que determinan la respuesta de la RNA y que además permiten disminuir paulatinamente el error. El algoritmo de retro-propagación utiliza el descenso de gradiente para determinar las variaciones de los pesos, y se logra mediante derivadas parciales sucesivas que permiten propagar el error hacia las capas previas a la capa de salida.

Para facilitar el cálculo matemático, se almacena un factor en una variable denominada $\Delta CAPA1$, como se muestra en la Ecuación(26).

$$\Delta CAPA1 = -\eta \cdot \frac{\partial \mathrm{E}}{\partial \mathrm{error}} \cdot \frac{\partial \mathrm{error}}{\partial \mathrm{out}} \cdot \frac{\partial \mathrm{out}}{\partial \mathrm{net}} \qquad (26)$$

Donde η es el factor de aprendizaje de la RNA.

En las siguientes ecuaciones se procede a calcular cada una de las derivadas parciales de la ecuación(26).

Para calcular $\frac{\partial \mathrm{E}}{\partial \mathrm{error}}$ se requiere el error medio cuadrático E, que está expresado en la ecuación(25).

$$\frac{\partial \mathrm{E}}{\partial \mathrm{error}} = \mathrm{error} \qquad (27)$$

Calcular $\frac{\partial \mathrm{error}}{\partial \mathrm{out}}$ requiere la función error, expresada en la ecuación(24).

$$\frac{\partial \mathrm{error}}{\partial \mathrm{out}} = -1 \qquad (28)$$

Calcular $\frac{\partial \mathrm{out}}{\partial \mathrm{net}}$ requiere la función out, que está expresada en la ecuación(9).

$$\frac{\partial \mathrm{out}}{\partial \mathrm{net}} = \frac{\alpha e^{-net \cdot \alpha}}{(1 + e^{-net \cdot \alpha})^2} \qquad (29)$$

$$\frac{\partial \mathrm{out}}{\partial \mathrm{net}} = \frac{1 \cdot \alpha}{1 + e^{-net \cdot \alpha}} \cdot \frac{(1 + e^{-net \cdot \alpha} - 1)}{1 + e^{-net \cdot \alpha}} \qquad (30)$$

$$\frac{\partial \mathrm{out}}{\partial \mathrm{net}} = \frac{1 \cdot \alpha}{1 + e^{-net \cdot \alpha}} \cdot \left(\frac{1}{1 + e^{-net \cdot \alpha}} + \frac{e^{-net \cdot \alpha}}{1 + e^{-net \cdot \alpha}} - \frac{1}{1 + e^{-net \cdot \alpha}}\right) \qquad (31)$$

$$\frac{\partial \text{out}}{\partial \text{net}} = \frac{1 \cdot \alpha}{1 + e^{-net \cdot \alpha}} \cdot \left(\frac{1}{1 + e^{-net \cdot \alpha}} + \frac{e^{-net \cdot \alpha}}{1 + e^{-net \cdot \alpha}} - \frac{1}{1 + e^{-net \cdot \alpha}}\right) \quad (32)$$

$$\frac{\partial \text{out}}{\partial \text{net}} = \frac{1 \cdot \alpha}{1 + e^{-net \cdot \alpha}} \cdot \left(\frac{1 + e^{-net \cdot \alpha}}{1 + e^{-net \cdot \alpha}} - \frac{1}{1 + e^{-net \cdot \alpha}}\right) \quad (33)$$

$$\frac{\partial \text{out}}{\partial \text{net}} = \frac{1}{1 + e^{-net \cdot \alpha}} \cdot \alpha \cdot \left(1 - \frac{1}{1 + e^{-net \cdot \alpha}}\right) \quad (34)$$

Debido a la ecuación (9), la expresión de la ecuación(34) puede expresarse como se muestra en la ecuación(35).

$$\frac{\partial \text{out}}{\partial \text{net}} = \text{out} \cdot \alpha \cdot (1 - \text{out}) \quad (35)$$

Al reemplazar las ecuaciones (27), (28) y (35) en la ecuación(26) se tiene el factor $\Delta CAPA1$ expresado en la ecuación(36).

$$\Delta CAPA1 = -\eta \cdot \text{error} \cdot (-1) \cdot \text{out} \cdot \alpha \cdot (1 - \text{out}) \quad (36)$$

$\Delta wSa1$ de la neurona 8 de la capa de salida se ilustra mediante la ecuación(37).

$$\Delta wSa1 = -\eta \cdot \frac{\partial \text{E}}{\partial \text{error}} \cdot \frac{\partial \text{error}}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{net}} \cdot \frac{\partial \text{net}}{\partial \text{Sa1}} \quad (37)$$

Debido a la ecuación(26), la ecuación(37) puede expresarse como se ilustra en la ecuación(38).

$$\Delta wSa1 = \Delta CAPA1 \cdot \frac{\partial \text{net}}{\partial \text{wSa1}} \quad (38)$$

Por lo tanto solo es necesario calcular $\frac{\partial \text{net}}{\partial \text{wSa1}}$, para lo cual se requiere la función net, que se encuentra expresada en la ecuación(8).

$$\frac{\partial \text{net}}{\partial \text{wSa1}} = out5 \qquad (39)$$

Entonces al reemplazar la ecuaciones (36) y (39) en (38) se tiene $\Delta wSa1$ que se ilustra en la ecuación (40). Para calcular $\Delta wSa2$, $\Delta wSa3$ y $\Delta wSa4$ se requiere el mismo procedimiento utilizado para $\Delta wSa1$.

$$\Delta wSa1 = -\eta \cdot \text{error} \cdot (-1) \cdot \text{out} \cdot \alpha \cdot (1 - \text{out}) \cdot out5 \qquad (40)$$

$\Delta wOc1$ de la neurona 5 de la capa oculta se calcula mediante la ecuación(41).

$$\Delta wOc1 = -\eta \cdot \frac{\partial \text{E}}{\partial \text{error}} \cdot \frac{\partial \text{error}}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{net}} \cdot \frac{\partial \text{net}}{\partial \text{out5}} \cdot \frac{\partial \text{out5}}{\partial \text{net5}} \cdot \frac{\partial \text{net5}}{\partial \text{wOc1}} \qquad (41)$$

Al asociar la ecuaciones (26) con la (41) se obtiene la ecuación(42)

$$\Delta wOc1 = \Delta CAPA1 \cdot \frac{\partial \text{net}}{\partial \text{out5}} \cdot \frac{\partial \text{out5}}{\partial \text{net5}} \cdot \frac{\partial \text{net5}}{\partial \text{wOc1}} \qquad (42)$$

Por lo tanto se requiere calcular las tres derivadas parciales de la ecuación (42). Para calcular $\frac{\partial \text{net}}{\partial \text{out5}}$ se requiere la función net que esta expresada en la ecuación (8).

$$\frac{\partial \text{net}}{\partial \text{out5}} = wSa1 \qquad (43)$$

Para calcular $\frac{\partial \text{out5}}{\partial \text{net5}}$ se requiere la función out5 expresada en la ecuación(15). Con el mismo procedimiento utilizado para calcular la ecuación(35) se puede determinar la ecuación (44).

$$\frac{\partial \text{out5}}{\partial \text{net5}} = \text{out5} \cdot \alpha \cdot (1 - \text{out5}) \qquad (44)$$

Calcular $\frac{\partial \text{net5}}{\partial \text{wOc1}}$ requiere la función net5 que está expresada en la ecuación(14).

$$\frac{\partial \text{net5}}{\partial \text{wOc1}} = out1 \qquad (45)$$

Entonces al reemplazar la ecuaciones(36), (43), (44) y (45) en la ecuación (42) se tiene $\Delta wOc1$ que se ilustra en la ecuación(46). Para calcular $\Delta wOc2$ a $\Delta wOc15$ se requiere el mismo procedimiento utilizado para $\Delta wOc1$.

$$\Delta wOc1 = -\eta \cdot \text{error} \cdot (-1) \cdot \text{out} \cdot \alpha \cdot (1 - \text{out}) \cdot wSa1 \cdot \text{out5} \cdot \alpha \cdot (1 - \text{out5}) \cdot out1 \qquad (46)$$

$\Delta wEn1$ de la neurona 1 de la capa de entrada recibe una retro-propagación de tres lazos. El primer lazo se forma entre las neuronas N1-N5-N8, el segundo lazo se forma entre las neuronas N1-N6-N8 y el tercer lazo se forma entre las neuronas N1-N7-N8. Al primero, segundo y tercer lazo se les denomina ΔN158, ΔN168 y ΔN178, respectivamente. Las expresiones para calcular $\Delta wEn1$ se muestran en las ecuaciones (47), (48), (49) y (50).

$$\Delta wEn1 = (\Delta N158 + \Delta N168 + \Delta N178) \cdot \frac{\partial \text{net1}}{\partial \text{wEn1}} \qquad (47)$$

$$\Delta N158 = -\eta \cdot \frac{\partial \mathrm{E}}{\partial \mathrm{error}} \cdot \frac{\partial \mathrm{error}}{\partial \mathrm{out}} \cdot \frac{\partial \mathrm{out}}{\partial \mathrm{net}} \cdot \frac{\partial \mathrm{net}}{\partial \mathrm{out5}} \cdot \frac{\partial \mathrm{out5}}{\partial \mathrm{net5}} \cdot \frac{\partial \mathrm{net5}}{\partial \mathrm{out1}} \cdot \frac{\partial \mathrm{out1}}{\partial \mathrm{net1}} \qquad (48)$$

$$\Delta N168 = -\eta \cdot \frac{\partial \mathrm{E}}{\partial \mathrm{error}} \cdot \frac{\partial \mathrm{error}}{\partial \mathrm{out}} \cdot \frac{\partial \mathrm{out}}{\partial \mathrm{net}} \cdot \frac{\partial \mathrm{net}}{\partial \mathrm{out6}} \cdot \frac{\partial \mathrm{out6}}{\partial \mathrm{net6}} \cdot \frac{\partial \mathrm{net6}}{\partial \mathrm{out1}} \cdot \frac{\partial \mathrm{out1}}{\partial \mathrm{net1}} \qquad (49)$$

$$\Delta N178 = -\eta \cdot \frac{\partial \mathrm{E}}{\partial \mathrm{error}} \cdot \frac{\partial \mathrm{error}}{\partial \mathrm{out}} \cdot \frac{\partial \mathrm{out}}{\partial \mathrm{net}} \cdot \frac{\partial \mathrm{net}}{\partial \mathrm{out7}} \cdot \frac{\partial \mathrm{out7}}{\partial \mathrm{net7}} \cdot \frac{\partial \mathrm{net7}}{\partial \mathrm{out1}} \cdot \frac{\partial \mathrm{out1}}{\partial \mathrm{net1}} \qquad (50)$$

Al asociar la ecuación (51) con la (26); la ecuación(56) con la (26) y la ecuación (52) con la (26), se tienen las expresiones de las ecuaciones (51), (52) y (53) respectivamente.

$$\Delta N158 = \Delta CAPA1 \cdot \frac{\partial \mathrm{net}}{\partial \mathrm{out5}} \cdot \frac{\partial \mathrm{out5}}{\partial \mathrm{net5}} \cdot \frac{\partial \mathrm{net5}}{\partial \mathrm{out1}} \cdot \frac{\partial \mathrm{out1}}{\partial \mathrm{net1}} \qquad (51)$$

$$\Delta N168 = \Delta CAPA1 \cdot \frac{\partial \mathrm{net}}{\partial \mathrm{out6}} \cdot \frac{\partial \mathrm{out6}}{\partial \mathrm{net6}} \cdot \frac{\partial \mathrm{net6}}{\partial \mathrm{out1}} \cdot \frac{\partial \mathrm{out1}}{\partial \mathrm{net1}} \qquad (52)$$

$$\Delta N178 = \Delta CAPA1 \cdot \frac{\partial \mathrm{net}}{\partial \mathrm{out7}} \cdot \frac{\partial \mathrm{out7}}{\partial \mathrm{net7}} \cdot \frac{\partial \mathrm{net7}}{\partial \mathrm{out1}} \cdot \frac{\partial \mathrm{out1}}{\partial \mathrm{net1}} \qquad (53)$$

Para calcular $\Delta N158$, solo es necesario calcular las derivadas parciales de la ecuación (51). Sin embargo $\frac{\partial \mathrm{net}}{\partial \mathrm{out5}} y \ \frac{\partial \mathrm{out5}}{\partial \mathrm{net5}}$ ya se encuentran expresadas en las ecuaciones (43) y (44) respectivamente.

Para calcular $\frac{\partial \text{net5}}{\partial \text{out1}}$ se requiere la función net5 expresada en la ecuación(14).

$$\frac{\partial \text{net5}}{\partial \text{out1}} = wOc1 \qquad (54)$$

Calcular $\frac{\partial \text{out1}}{\partial \text{net1}}$ requiere la función out1 expresada en la ecuación(23).

$$\frac{\partial \text{out1}}{\partial \text{net1}} = \text{out1} \cdot \alpha \cdot (1 - \text{out1}) \qquad (55)$$

Al reemplazar las ecuaciones (43), (44), (54) y (55) en la ecuación(51) se tiene $\Delta N158$.

$$\Delta N158 = \Delta CAPA1 \cdot \text{wSa1} \cdot \text{out5} \cdot \alpha \cdot (1 - \text{out5}) \cdot \text{wOc1} \cdot \text{out1} \cdot \alpha \cdot (1 - \text{out1}) \qquad (56)$$

Para $\Delta N168$, es necesario calcular las derivadas parciales de la ecuación (52), a excepción de $\frac{\partial \text{out1}}{\partial \text{net1}}$ que ya se encuentra expresada en la ecuación (55).

Para $\frac{\partial \text{net}}{\partial \text{out6}}$ se requiere la función net expresada en la ecuación (8).

$$\frac{\partial \text{net}}{\partial \text{out6}} = \text{wSa2} \qquad (57)$$

Para $\frac{\partial \text{out6}}{\partial \text{net6}}$ se requiere la función out6 expresada en la ecuación(13).

$$\frac{\partial \text{out6}}{\partial \text{net6}} = \text{out6} \cdot \alpha \cdot (1 - \text{out6}) \qquad (58)$$

Para $\frac{\partial \text{net6}}{\partial \text{out1}}$ se requiere la función net6 expresada en la ecuación(12).

$$\frac{\partial \text{net6}}{\partial \text{out1}} = \text{wOc2} \qquad (59)$$

Al reemplazar las ecuaciones (57), (58), (59) y (55) en la ecuación(52) se tiene $\Delta N168$.

$$\Delta N168 = \Delta CAPA1 \cdot \text{wSa2} \cdot \text{out6} \cdot \alpha \cdot (1 - \text{out6}) \cdot \text{wOc2} \cdot \text{out1} \cdot \alpha \cdot (1 - \text{out1}) \qquad (60)$$

Para $\Delta N178$, es necesario calcular las derivadas parciales de la ecuación(53), a excepción de $\frac{\partial \text{out1}}{\partial \text{net1}}$ que ya se encuentra expresada en la ecuación (55).

Para $\frac{\partial \text{net}}{\partial \text{out7}}$ se requiere la función net expresada en la ecuación (8).

$$\frac{\partial \text{net}}{\partial \text{out7}} = \text{wSa3} \qquad (61)$$

Para $\frac{\partial \text{out7}}{\partial \text{net7}}$ se requiere la función out6 expresada en la ecuación(11).

$$\frac{\partial \text{out7}}{\partial \text{net7}} = \text{out7} \cdot \alpha \cdot (1 - \text{out7}) \qquad (62)$$

Para $\frac{\partial \text{net7}}{\partial \text{out1}}$ se requiere la función net6 expresada en la ecuación(10).

$$\frac{\partial \text{net7}}{\partial \text{out1}} = \text{wOc3} \qquad (63)$$

Al reemplazar las ecuaciones (61), (62), (63) y (55) en la ecuación(53) se tiene $\Delta N178$.

$$\Delta N178 = \Delta CAPA1 \cdot \text{wSa3} \cdot \text{out7} \cdot \alpha \cdot (1 - \text{out7}) \cdot \text{wOc2} \cdot \text{out1} \cdot \alpha \cdot (1 - \text{out1}) \qquad (64)$$

Para calcular $\frac{\partial \text{net1}}{\partial \text{wEn1}}$ de la ecuación(47) se requiere la función net1 expresada en la ecuación (22) .

$$\frac{\partial \text{net1}}{\partial \text{wEn1}} = \text{in1} \qquad (65)$$

Al reemplazar la ecuación(65) en la (47) se tiene la ecuación (66).

$$\Delta wEn1 = (\Delta N158 + \Delta N168 + \Delta N178) \cdot \text{in1} \qquad (66)$$

Al reemplazar la ecuación (56), (60) y (64) en (66) se obtiene la expresión de la ecuación (67).

$$\begin{aligned}\Delta wEn1 = (&\Delta CAPA1 \cdot \text{wSa1} \cdot \text{out5} \cdot \alpha \cdot (1 - \text{out5}) \cdot \text{wOc1} \cdot \text{out1} \cdot \alpha \cdot (1 - \text{out1}) + \Delta CAPA1 \cdot \text{wSa2} \cdot \text{out6} \cdot \alpha \cdot (1 - \text{out6}) \cdot \text{wOc2} \cdot \text{out1} \cdot \alpha \cdot (1 - \text{out1}) + \Delta CAPA1 \cdot \text{wSa3} \cdot \text{out7} \cdot \alpha \cdot (1 - \text{out7}) \cdot \text{wOc2} \cdot \text{out1} \cdot \alpha \cdot (1 - \text{out1})) \cdot \text{in1}\end{aligned} \qquad (67)$$

Al sacar factor común de la ecuación(67) se tiene la ecuación(68).

$$\Delta wEn1 = [\Delta CAPA1 \cdot \text{out1} \cdot \alpha \cdot (1 - \text{out1}) \cdot \text{in1}] \cdot [\text{wSa1} \cdot \text{out5} \cdot \alpha \cdot (1 - \text{out5}) \cdot \text{wOc1} + \text{wSa2} \cdot \text{out6} \cdot \alpha \cdot (1 - \text{out6}) \cdot \text{wOc2} + \text{wSa3} \cdot \text{out7} \cdot \alpha \cdot (1 - \text{out7}) \cdot \text{wOc2}] \quad (68)$$

Al reemplazar la ecuación(36) en la ecuación (68) se tiene $\Delta wEn1$.

$$\Delta wEn1 = [-\eta \cdot \text{error} \cdot (-1) \cdot \text{out} \cdot \alpha \cdot (1 - \text{out}) \cdot \text{out1} \cdot \alpha \cdot (1 - \text{out1}) \cdot \text{in1}] \cdot [\text{wSa1} \cdot \text{out5} \cdot \alpha \cdot (1 - \text{out5}) \cdot \text{wOc1} + \text{wSa2} \cdot \text{out6} \cdot \alpha \cdot (1 - \text{out6}) \cdot \text{wOc2} + \text{wSa3} \cdot \text{out7} \cdot \alpha \cdot (1 - \text{out7}) \cdot \text{wOc2}] \quad (69)$$

Para calcular $\Delta wEn2$ a $\Delta wEn20$ se debe realizar para cada uno el mismo procedimiento realizado para $\Delta wEn1$.

Es importante mencionar que la constante α además de determinar la pendiente de la función sigmoidea también sirve como una constante de impulso para evitar que el algoritmo quede atrapado en un mínimo local, además se agrega un factor de 0,1 a las derivadas parciales de out con respecto a net, también como una constante para sacar al algoritmo de cualquier mínimo local. Entonces al incrementar el valor de 0,1 y simplificar las ecuaciones (40), (46) y (69) se obtienen las expresiones de las ecuaciones (70), (71) y (72) respectivamente.

$$\Delta wSa1 = \eta \cdot \text{error} \cdot [\text{out} \cdot \alpha \cdot (1 - \text{out}) + 0.1] \cdot out5 \quad (70)$$

$$\Delta wOc1 = \eta \cdot \text{error} \cdot [\text{out} \cdot \alpha \cdot (1 - \text{out}) + 0.1] \cdot wSa1 \cdot [\text{out5} \cdot \alpha \cdot (1 - \text{out5}) + 0.1] \cdot out1 \quad (71)$$

$$\Delta wEn1 = [\eta \cdot \text{error} \cdot [\text{out} \cdot \alpha \cdot (1 - \text{out}) + 0.1] \cdot [\text{out1} \cdot \alpha \cdot (1 - \text{out1}) + 0.1] \cdot \text{in1}] \cdot [\text{wSa1} \cdot [\text{out5} \cdot \alpha \cdot (1 - \text{out5}) + 0.1] \cdot \text{wOc1} + \text{wSa2} \cdot [\text{out6} \cdot \alpha \cdot (1 - \text{out6}) + 0.1] \cdot \text{wOc2} + \text{wSa3} \cdot [\text{out7} \cdot \alpha \cdot (1 - \text{out7}) + 0.1] \cdot \text{wOc2}] \quad (72)$$

Para ingresar en Matlab la formulación matemática explicada anteriormente se requiere el siguiente código:

```
CAPA DE SALIDA
Factor de error de la neurona 8 de la capa de salida:
deltaCAPA1 = fa*error*(alpha*out*(1-out)+0.1);

CAPA OCULTA
Factor de error de la neurona 5 de la capa oculta:
deltaCAPAout1 = wSa1*(alpha*out1*(1-out1)+0.1);
Factor de error de la neurona 6 de la capa oculta:
deltaCAPAout2 = wSa2*(alpha*out2*(1-out2)+0.1);
Factor de error de la neurona 7 de la capa oculta:
deltaCAPAout3 = wSa3*(alpha*out3*(1-out3)+0.1);


CAPA DE ENTRADA
FACTOR DE ERROR DE LAS NEURONAS 1 CON 5
deltaCAPAin1_5 = wOc1*(alpha*out4*(1-out4)+0.1);
FACTOR DE ERROR DE LAS NEURONAS 1 CON 6
deltaCAPAin1_6 = wOc2*(alpha*out4*(1-out4)+0.1);
FACTOR DE ERROR DE LAS NEURONAS 1 CON 7
deltaCAPAin1_7 = wOc3*(alpha*out4*(1-out4)+0.1);
FACTOR DE ERROR DE LAS NEURONAS 2 CON 5
deltaCAPAin2_5 = wOc4*(alpha*out5*(1-out5)+0.1);
FACTOR DE ERROR DE LAS NEURONAS 2 CON 6
deltaCAPAin2_6 = wOc5*(alpha*out5*(1-out5)+0.1);
FACTOR DE ERROR DE LAS NEURONAS 2 CON 7
deltaCAPAin2_7 = wOc6*(alpha*out5*(1-out5)+0.1);
FACTOR DE ERROR DE LAS NEURONAS 3 CON 5
deltaCAPAin3_5 = wOc7*(alpha*out6*(1-out6)+0.1);
FACTOR DE ERROR DE LAS NEURONAS 3 CON 6
deltaCAPAin3_6 = wOc8*(alpha*out6*(1-out6)+0.1);
FACTOR DE ERROR DE LAS NEURONAS 3 CON 7
deltaCAPAin3_7 = wOc9*(alpha*out6*(1-out6)+0.1);
FACTOR DE ERROR DE LAS NEURONAS 4 CON 5
deltaCAPAin4_5 = wOc10*(alpha*out7*(1-out7)+0.1);
FACTOR DE ERROR DE LAS NEURONAS 4 CON 6
deltaCAPAin4_6 = wOc11*(alpha*out7*(1-out7)+0.1);
```

```
FACTOR DE ERROR DE LAS NEURONAS 4 CON 7
deltaCAPAin4_7 = wOc12*(alpha*out7*(1-out7)+0.1);
FACTOR DE ERROR PROPAGADO ENTRE 8 Y 5
deltaCAPA21 = deltaCAPA1*deltaCAPAout1;
FACTOR DE ERROR PROPAGADO ENTRE 8 Y 6
deltaCAPA22 = deltaCAPA1*deltaCAPAout2;
FACTOR DE ERROR PROPAGADO ENTRE 8 Y 7
deltaCAPA23 = deltaCAPA1*deltaCAPAout3;
FACTOR DE ERROR PROPAGADO ENTRE 8,5 y 1
deltaCAPA_N158 = deltaCAPA21*deltaCAPAin1_5;
FACTOR DE ERROR PROPAGADO ENTRE 8,6 y 1
deltaCAPA_N168 = deltaCAPA22*deltaCAPAin1_6;
FACTOR DE ERROR PROPAGADO ENTRE 8,7 y 1
deltaCAPA_N178 = deltaCAPA23*deltaCAPAin1_7;
FACTOR DE ERROR PROPAGADO ENTRE 8,5 y 2
deltaCAPA_N258 = deltaCAPA21*deltaCAPAin2_5;
FACTOR DE ERROR PROPAGADO ENTRE 8,6 y 2
deltaCAPA_N268 = deltaCAPA22*deltaCAPAin2_6;
FACTOR DE ERROR PROPAGADO ENTRE 8,7 y 2
deltaCAPA_N278 = deltaCAPA23*deltaCAPAin2_7;
FACTOR DE ERROR PROPAGADO ENTRE 8,5 y 3
deltaCAPA_N358 = deltaCAPA21*deltaCAPAin3_5;
FACTOR DE ERROR PROPAGADO ENTRE 8,6 y 3
deltaCAPA_N368 = deltaCAPA22*deltaCAPAin3_6;
FACTOR DE ERROR PROPAGADO ENTRE 8,7 y 3
deltaCAPA_N378 = deltaCAPA23*deltaCAPAin3_7;
FACTOR DE ERROR PROPAGADO ENTRE 8,5 y 4
deltaCAPA_N458 = deltaCAPA21*deltaCAPAin4_5;
FACTOR DE ERROR PROPAGADO ENTRE 8,6 y 4
deltaCAPA_N468 = deltaCAPA22*deltaCAPAin4_6;
FACTOR DE ERROR PROPAGADO ENTRE 8,7 y 4
deltaCAPA_N478 = deltaCAPA23*deltaCAPAin4_7;
```

Para la capa de entrada, las variaciones de los pesos, también conocidos como deltas, se nombran como se ilustra en la *Tabla 10*.

Tabla 10.

Denominación de las variaciones de los pesos de la capa de entrada.

Nombre del Peso		Nombre de la variación	
wEn	1	deltaEn	1
wEn	2	deltaEn	2
wEn	3	deltaEn	3
wEn	4	deltaEn	4
wEn	5	deltaEn	5
wEn	6	deltaEn	6
wEn	7	deltaEn	7
wEn	8	deltaEn	8
wEn	9	deltaEn	9
wEn	10	deltaEn	10
wEn	11	deltaEn	11
wEn	12	deltaEn	12
wEn	13	deltaEn	13
wEn	14	deltaEn	14
wEn	15	deltaEn	15
wEn	16	deltaEn	16
wEn	17	deltaEn	17
wEn	18	deltaEn	18
wEn	19	deltaEn	19
wEn	20	deltaEn	20

Las variaciones de los pesos de la capa de entrada, se calculan mediante el siguiente código:

```
deltaEn17 = (deltaCAPA_N158 + deltaCAPA_N168 + deltaCAPA_N178)*in0;
deltaEn18 = (deltaCAPA_N258 + deltaCAPA_N268 + deltaCAPA_N278)*in0;
deltaEn19 = (deltaCAPA_N358 + deltaCAPA_N368 + deltaCAPA_N378)*in0;
deltaEn20 = (deltaCAPA_N458 + deltaCAPA_N468 + deltaCAPA_N478)*in0;

deltaEn1 = (deltaCAPA_N158 + deltaCAPA_N168 + deltaCAPA_N178)*in1(i);
```

```
deltaEn2 = (deltaCAPA_N258 + deltaCAPA_N268 + deltaCAPA_N278)*in1(i);
deltaEn3 = (deltaCAPA_N358 + deltaCAPA_N368 + deltaCAPA_N378)*in1(i);
deltaEn4 = (deltaCAPA_N458 + deltaCAPA_N468 + deltaCAPA_N478)*in1(i);

deltaEn5 = (deltaCAPA_N158 + deltaCAPA_N168 + deltaCAPA_N178)*in2(i);
deltaEn6 = (deltaCAPA_N258 + deltaCAPA_N268 + deltaCAPA_N278)*in2(i);
deltaEn7 = (deltaCAPA_N358 + deltaCAPA_N368 + deltaCAPA_N378)*in2(i);
deltaEn8 = (deltaCAPA_N458 + deltaCAPA_N468 + deltaCAPA_N478)*in2(i);

deltaEn9 = (deltaCAPA_N158 + deltaCAPA_N168 + deltaCAPA_N178)*in3(i);
deltaEn10 = (deltaCAPA_N258 + deltaCAPA_N268 + deltaCAPA_N278)*in3(i);
deltaEn11 = (deltaCAPA_N358 + deltaCAPA_N368 + deltaCAPA_N378)*in3(i);
deltaEn12 = (deltaCAPA_N458 + deltaCAPA_N468 + deltaCAPA_N478)*in3(i);

deltaEn13 = (deltaCAPA_N158 + deltaCAPA_N168 + deltaCAPA_N178)*in4(i);
deltaEn14 = (deltaCAPA_N258 + deltaCAPA_N268 + deltaCAPA_N278)*in4(i);
deltaEn15 = (deltaCAPA_N358 + deltaCAPA_N368 + deltaCAPA_N378)*in4(i);
deltaEn16 = (deltaCAPA_N458 + deltaCAPA_N468 + deltaCAPA_N478)*in4(i);
```

Para la capa oculta las variaciones de los pesos, se nombran como se ilustra en la *Tabla 11*.

Tabla 11.

Denominación de las variaciones de los pesos de la capa oculta.

Nombre del Peso		**Nombre de la variación**	
wOc	1	deltaOc	1
wOc	2	deltaOc	2
wOc	3	deltaOc	3
wOc	4	deltaOc	4
wOc	5	deltaOc	5
wOc	6	deltaOc	6
wOc	7	deltaOc	7
wOc	8	deltaOc	8
wOc	9	deltaOc	9
wOc	10	deltaOc	10
wOc	11	deltaOc	11
wOc	12	deltaOc	12
wOc	13	deltaOc	13
wOc	14	deltaOc	14
wOc	15	deltaOc	15

Las variaciones de los pesos de la capa oculta se calculan mediante el siguiente código:

```
deltaOc13 = deltaCAPA21*in0;
deltaOc14 = deltaCAPA22*in0;
deltaOc15 = deltaCAPA23*in0;
deltaOc1 = deltaCAPA21*in1(i);
deltaOc2 = deltaCAPA22*in1(i);
deltaOc3 = deltaCAPA23*in1(i);
deltaOc4 = deltaCAPA21*in2(i);
deltaOc5 = deltaCAPA22*in2(i);
deltaOc6 = deltaCAPA23*in2(i);
deltaOc7 = deltaCAPA21*in3(i);
deltaOc8 = deltaCAPA22*in3(i);
deltaOc9 = deltaCAPA23*in3(i);
deltaOc10 = deltaCAPA21*in4(i);
deltaOc11 = deltaCAPA22*in4(i);
deltaOc12 = deltaCAPA23*in4(i);
```

Para la capa de salida las variaciones de los pesos, se nombran como se ilustra en la *Tabla 12*.

Tabla 12.

Denominación de las variaciones de los pesos de la capa de salida.

Nombre del Peso		Nombre de la variación	
wSa	1	deltaSa	1
wSa	2	deltaSa	2
wSa	3	deltaSa	3
wSa	4	deltaSa	4

Las variaciones de los pesos de la capa de salida se calculan mediante el siguiente código:

```
deltaSa4 = deltaCAPA1*in0;
deltaSa1 = deltaCAPA1*out1;
deltaSa2 = deltaCAPA1*out2;
deltaSa3 = deltaCAPA1*out3;
```

3.2.5. Ajuste de pesos

La actualización o ajuste de los pesos es necesaria para tomar en cuenta las variaciones que se les deben realizar a los pesos según sugiera el algoritmo. Para realizar el ajuste es necesario asignarle a cada peso su valor anterior sumado a su correspondiente variación, ese proceso se ilustra mediante el siguiente código:

```
CAPA DE ENTRADA:
wEn17 = wEn17 + deltaEn17;
wEn18 = wEn18 + deltaEn18;
wEn19 = wEn19 + deltaEn19;
wEn20 = wEn20 + deltaEn20;
wEn1 = wEn1 + deltaEn1;
wEn2 = wEn2 + deltaEn2;
wEn3 = wEn3 + deltaEn3;
wEn4 = wEn4 + deltaEn4;
wEn5 = wEn5 + deltaEn5;
wEn6 = wEn6 + deltaEn6;
wEn7 = wEn7 + deltaEn7;
wEn8 = wEn8 + deltaEn8;
wEn9 = wEn9 + deltaEn9;
wEn10 = wEn10 + deltaEn10;
wEn11 = wEn11 + deltaEn11;
wEn12 = wEn12 + deltaEn12;
wEn13 = wEn13 + deltaEn13;
wEn14 = wEn14 + deltaEn14;
wEn15 = wEn15 + deltaEn15;
wEn16 = wEn16 + deltaEn16;

CAPA OCULTA:
wOc13 = wOc13 + deltaOc13;
wOc14 = wOc14 + deltaOc14;
wOc15 = wOc15 + deltaOc15;
wOc1 = wOc1 + deltaOc1;
wOc2 = wOc2 + deltaOc2;
wOc3 = wOc3 + deltaOc3;
wOc4 = wOc4 + deltaOc4;
wOc5 = wOc5 + deltaOc5;
wOc6 = wOc6 + deltaOc6;
wOc7 = wOc7 + deltaOc7;
wOc8 = wOc8 + deltaOc8;
wOc9 = wOc9 + deltaOc9;
wOc10 = wOc10 + deltaOc10;
wOc11 = wOc11 + deltaOc11;
wOc12 = wOc12 + deltaOc12;
```

```
CAPA DE SALIDA:
wSa4 = wSa4 + deltaSa4;
wSa1 = wSa1 + deltaSa1;
wSa2 = wSa2 + deltaSa2;
wSa3 = wSa3 + deltaSa3;
ERROR DE ÉPOCA:
error_epoca = error_epoca +(error*error);
```

Cuando el error_epoca alcanza una valor de 0.00000001, se termina el proceso de entrenamiento, y se almacenan todos los pesos en solo vector quedando así listos para ser enviados mediante comunicación serial hacia la RNA implementada en la FPGA.

3.3. Sistema de comunicación

Se tiene un sistema de comunicación serial de tipo Simplex o unidireccional, el mismo que será el encargado de realizar el proceso de transferencia, y se ilustra en la Figura 9.

Figura 9. Diagrama de bloques del sistema de comunicación

A continuación se explican algunos detalles importantes del Emisor y Receptor:

- **Emisor:** El Emisor es una computadora en la cual se tiene un programa realizado en Matlab, además para generar un puerto serial en la misma se

tiene un dispositivo USB to Serial PL2303 que viene incluido en los accesorios de la tarjeta FPGA.

- **Receptor:** El Receptor es la tarjeta FPGA, en la cual se ha descrito y modelado un hardware capaz de realizar la recepción y transferirla hacia la RNA implementada también en la tarjeta.

3.3.1 Emisor

El envío se lo materializa desde una computadora, mediante el protocolo de comunicación serial desde Matlab. En los parámetros para la comunicación se configuró un bit de parada, ningún bit de paridad, una velocidad de 2400 baudios, y ocho bits de datos. El envío de datos se lo realiza desde una interfaz gráfica del guide de Matlab y se lo ilustra mediante el diagrama de flujo mostrado en la Figura 10.

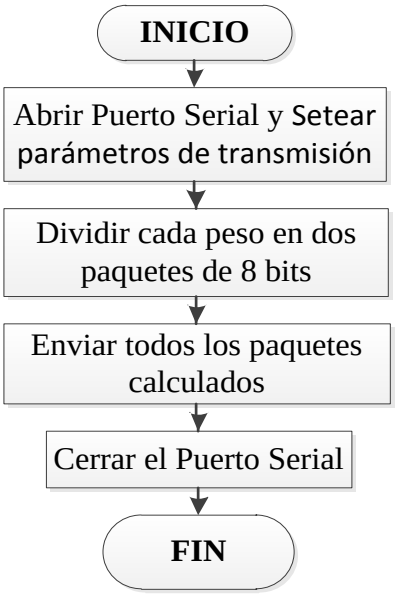


Figura 10. Diagrama de flujo del envío de datos.

3.3.2 Receptor

El receptor es la tarjeta FPGA , en la cual se ha programado un módulo de recepción serial, con los mismos parámetros de funcionamiento del emisor. La recepción consta de un conjunto de módulos interactuando entre sí, cuyo funcionamiento se detalla a continuación:

Módulo Top_Recepcion

Este módulo se encarga de realizar la recepción de la comunicación serial a una velocidad de 2400 baudios. Los datos ingresan bit a bit por la señal **Rx**, y da como resultado un número binario a través de la señal **Dato_11bits**, que representa a los pesos enviados desde la computadora, también muestra el signo de cada peso a través de la señal **signo**. Además se incrementa un conteo al finalizar la recepción cada peso, el mismo que representa la dirección en la cual debe ser asignado cada uno de los pesos, y se muestra a través de la señal **DireccionC**. Se incluye también una señal denominada **habilitante**, que cambia de estado cada vez que un peso está listo con su respectivo signo y dirección, y será utilizada por cada neurona para detectar un nuevo peso. La señal **clk** es utilizada para calcular la velocidad de recepción, y es conectada al reloj interno de la tarjeta que tiene una velocidad de 50MHz. La señal **rst** sirve para reiniciar el módulo de recepción en caso de ser necesario.

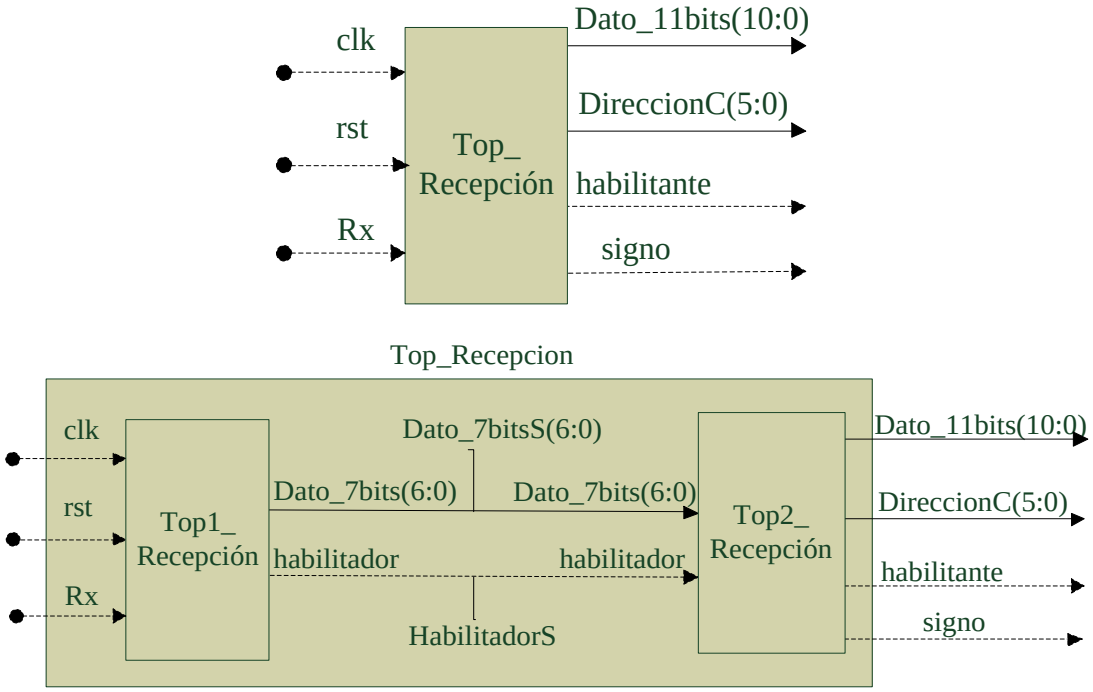


Figura 11. Diagrama RTL de la Recepción.

El código de programación requerido para este módulo es el siguiente:

```
--------DECLARACION DE LIBRERIAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
------FIN DECLARACION DE LIBRERIAS------

----SEÑALES I/O----
---Se realiza la declaración de señales
---de entrada y salida del módulo
entity Top_Recepcion is
    Port ( clk : in  STD_LOGIC;
           Rx : in  STD_LOGIC;
           rst : in  STD_LOGIC;
               signo : out  STD_LOGIC;
               habilitante : out std_logic;
               DireccionC : out  STD_LOGIC_VECTOR (5 downto 0);
               Dato_11bits : out  STD_LOGIC_VECTOR (10 downto 0));
end Top_Recepcion;
----FIN SEÑALES I/O----
-----------------------ARQUITECTURA-----------------------
---Se realiza la estructuración de la arquitectura del módulo
architecture Behavioral of Top_Recepcion is

-----SEÑALES AUXILIARES INTERNAS-----
---Se declaran señales internas al módulo,
---útiles para la interconexión entre módulos
---internos
signal Dato_7bitsS: std_logic_vector(6 downto 0);
signal HabilitadorS: std_logic;
---FIN SEÑALES AUXILIARES INTERNAS---

------MÓDULOS INTERNOS------
---Se Declaran los módulos internos con sus
---correspondientes entradas y salidas
```

```
---MÓDULO Top1_Recepcion---
---Dentro del módulo Top_Recepcion se declara un modulo
---interno denominado Top1_Recepcion, con sus
---correspondiente entradas y salidas
COMPONENT Top1_Recepcion
    PORT(
        clk : IN std_logic;
        Rx : IN std_logic;
        rst : IN std_logic;
        Dato_7bits : OUT std_logic_vector(6 downto 0);
        Habilitador : OUT std_logic
        );
    END COMPONENT;
---FIN MÓDULO Top1_Recepcion---

---MÓDULO Top2_Recepcion---
---Dentro del módulo Top_Recepcion se declara un modulo
---interno denominado Top2_Recepcion, con sus
---correspondiente entradas y salidas
COMPONENT Top2_Recepcion
    PORT(
        Dato_7bits : IN std_logic_vector(6 downto 0);
        habilitador : IN std_logic;
        signo : OUT std_logic;
        habilitante : out std_logic;
        DireccionC : out  STD_LOGIC_VECTOR (5 downto 0);
        Dato_11bits : OUT std_logic_vector(10 downto 0)
        );
    END COMPONENT;
---FIN MÓDULO Top2_Recepcion---
------FIN MÓDULOS INTERNOS-----

----INSTANCIACIÓN----
---Se realiza la interconexión interna
---entre módulos internos, mediante sus
---correspondientes señales de entrada y salida.
begin

---CONEXIÓN DEL MODULO Top1_Recepcion---
---se especifica con que señales del exterior
---van a ser conectadas las señales de entrada
---y salida del módulo Top1_Recepcion
Inst_Top1_Recepcion: Top1_Recepcion PORT MAP(
        clk => clk,
        Rx => Rx,
        rst => rst,
        Dato_7bits => Dato_7bitsS,
        Habilitador => HabilitadorS
    );
---FIN CONEXIÓN DEL MODULO Top1_Recepcion---

---CONEXIÓN DEL MODULO Top2_Recepcion---
---se especifica con que señales del exterior
---van a ser conectadas las señales de entrada
---y salida del módulo Top2_Recepcion
    Inst_Top2_Recepcion: Top2_Recepcion PORT MAP(
        Dato_7bits => Dato_7bitsS,
        habilitador => HabilitadorS,
        signo => signo,
        habilitante => habilitante,
        DireccionC => DireccionC,
        Dato_11bits => Dato_11bits
    );
---FIN CONEXIÓN DEL MODULO Top2_Recepcion---
end Behavioral;
-----FIN INSTANCIACIÓN----
-----FIN ARQUITECTURA-------
```

Módulo Top1_Recepcion

Este bloque se encarga de realizar la recepción de la mitad de la información de cada peso en una trama de 7 bits, cuenta con una señal Dato_7bits para mostrar dicha información. Además se incluye una señal denominada Habilitador que se activa cada que una trama de 7 bits de información ha sido completada; y se ilustra mediante el diagrama RTL mostrado en la Figura 12.

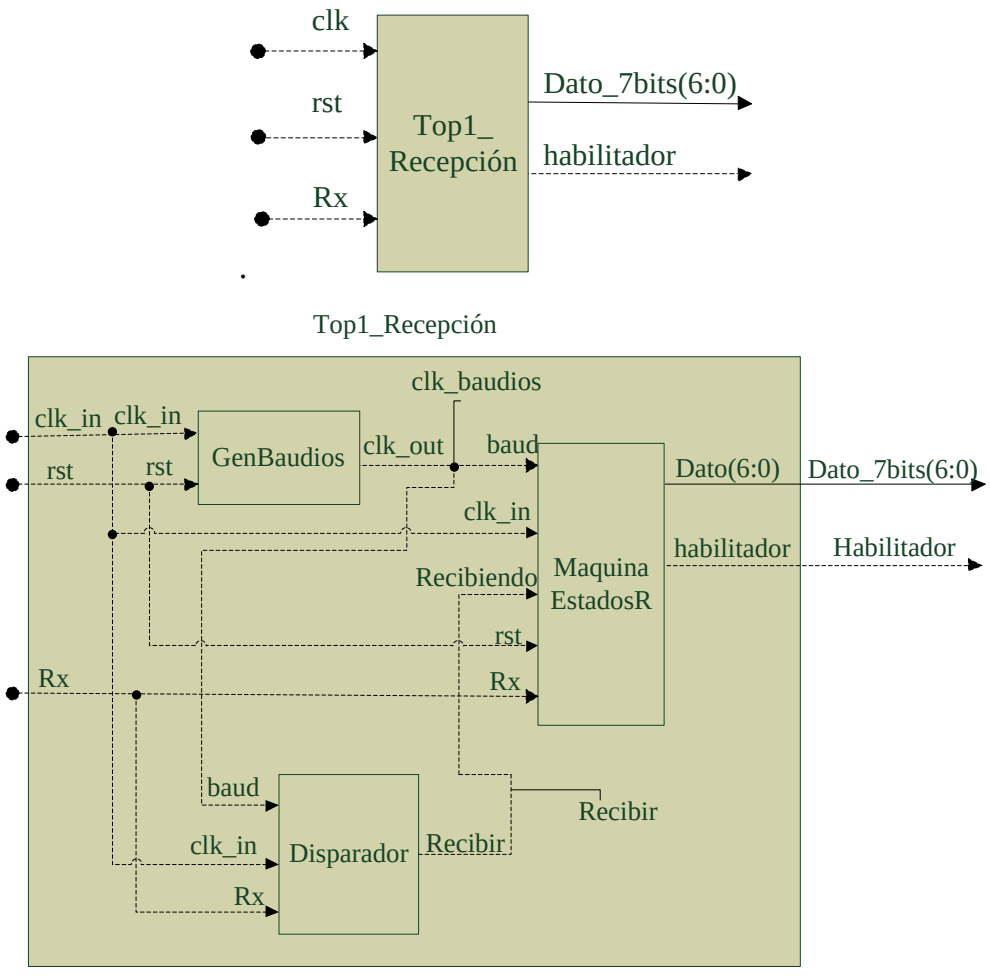


Figura 12.Diagrama RTL de Recepción bit a bit.

El código de programación requerido para este módulo es el siguiente:

```
--------DECLARACIÓN DE LIBRERÍAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
------FIN DECLARACIÓN DE LIBRERÍAS------

------SEÑALES I/O------
---Se realiza la declaración de señales
---de entrada y salida del módulo
entity Top1_Recepcion is
    Port ( clk : in  STD_LOGIC;
           Rx : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           Dato_7bits : out  STD_LOGIC_VECTOR (6 downto 0);
           Habilitador : out  STD_LOGIC);
end Top1_Recepcion;
----FIN SEÑALES I/O----

------------------------ARQUITECTURA------------------------
---Se describe y se modela el hardware mediante código,
---describiendo así el flujo de datos y comportamiento en este módulo.
architecture Behavioral of Top1_Recepcion is

---SEÑALES AUXILIARES INTERNAS-----
---Se declaran señales internas al módulo,
---útiles para la interconexión entre módulos
---internos
signal clk_baudios: std_logic;
signal Recibir: std_logic;
---FIN SEÑALES AUXILIARES INTERNAS---

------MÓDULOS INTERNOS------
---Se Declaran los módulos internos con sus
---correspondientes entradas y salidas

---MÓDULO GenBaudios---
---Dentro del módulo Top1_Recepcion se declara un módulo
---interno denominado GenBaudios, con sus
---correspondientes entradas y salidas
COMPONENT GenBaudios
PORT(
    rst : IN std_logic;
    clk_in : IN std_logic;
    clk_out : OUT std_logic
    );
END COMPONENT;
---FIN MÓDULO GenBaudios---


---MÓDULO Disparador---
---Dentro del módulo Top1_Recepcion se declara un módulo
---interno denominado Disparador, con sus
---correspondientes entradas y salidas
COMPONENT Disparador
    PORT(

        clk_in : IN STD_LOGIC;
        baud : IN std_logic;
        Rx : IN std_logic;
        Recibir : OUT std_logic
        );
END COMPONENT;
---FIN MÓDULO Disparador---

---MÓDULO MaquinaEstadosR---
---Dentro del módulo Top1_Recepcion se declara un módulo
---interno denominado MaquinaEstadosR, con sus
---correspondientes entradas y salidas
```

```
COMPONENT MaquinaEstadosR
    PORT(
        clk_in : in  STD_LOGIC;
        baud : IN std_logic;
        rst : IN std_logic;
        Recibiendo : IN std_logic;
        Rx : IN std_logic;
        Dato : OUT std_logic_vector(6 downto 0);
        habilitador : OUT std_logic
        );
END COMPONENT;
---FIN MÓDULO MaquinaEstadosR---

----INSTANCIACIÓN----
---Se realiza la interconexión interna
---entre módulos internos, mediante sus
---correspondientes señales de entrada y salida.
begin

---CONEXIÓN DEL MÓDULO GenBaudios---
---Se especifica con que señales del exterior
---van a ser conectadas las señales de entrada
---y salida del módulo GenBaudios
Inst_GenBaudios: GenBaudios PORT MAP(
        rst => rst,
        clk_in => clk,
        clk_out => clk_baudios
    );
---FIN CONEXIÓN DEL MÓDULO GenBaudios---

---CONEXIÓN DEL MÓDULO Disparador---
---Se especifica con que señales del exterior
---van a ser conectadas las señales de entrada
---y salida del módulo Disparador
Inst_Disparador: Disparador PORT MAP(
        clk_in => clk,
        baud => clk_baudios,
        Rx => Rx,
        Recibir => Recibir
    );
---FIN CONEXIÓN DEL MÓDULO Disparador---


---CONEXIÓN DEL MÓDULO MaquinaEstadosR---
---Se especifica con que señales del exterior
---van a ser conectadas las señales de entrada
---y salida del módulo Disparador
Inst_MaquinaEstadosR: MaquinaEstadosR PORT MAP(
    clk_in => clk,
    baud => clk_baudios,
    rst => rst,
    Recibiendo => Recibir,
    Rx => Rx,
    Dato => Dato_7bits,
    habilitador => Habilitador
);
---FIN CONEXIÓN DEL MÓDULO Disparador---
end Behavioral;
-----FIN INSTANCIACIÓN----
-----------------------FIN ARQUITECTURA----------------------
```

Módulo GenBaudios

Este módulo es un divisor de frecuencia, que se encarga de generar un número de pulsos por segundo de tal manera que concuerde con la velocidad de comunicación y se muestra a través de la señal **clk_out**. Se tiene como señal entrada **clk_in** la misma que es conectada al reloj interno de la tarjeta; y se ilustra mediante el diagrama RTL de la Figura 13.

Figura 13. Diagrama RTL del Generador de tasa de baudios

Tomando en cuenta que el reloj interno de la tarjeta es de 50MHz, y se requiere tener una tasa de baudios de 2400, entonces el cálculo de la variable principal encargada de realizar la división de la frecuencia se lo realiza de la siguiente manera:

$$cuenta = \frac{f\_reloj}{2 * f_{\_baudios}} \qquad (73)$$

$$cuenta = \frac{50MHz}{2 * 2400bits/s} \qquad (74)$$

$$cuenta = 10416. \qquad (75)$$

Aproximando se puede colocar un valor de $cuenta = 10400$, con lo que se tiene un error mucho menor al 1%, por lo tanto no se esperan errores de ruido.

El funcionamiento de este módulo se ilustra mediante el diagrama de flujo de la Figura 14.

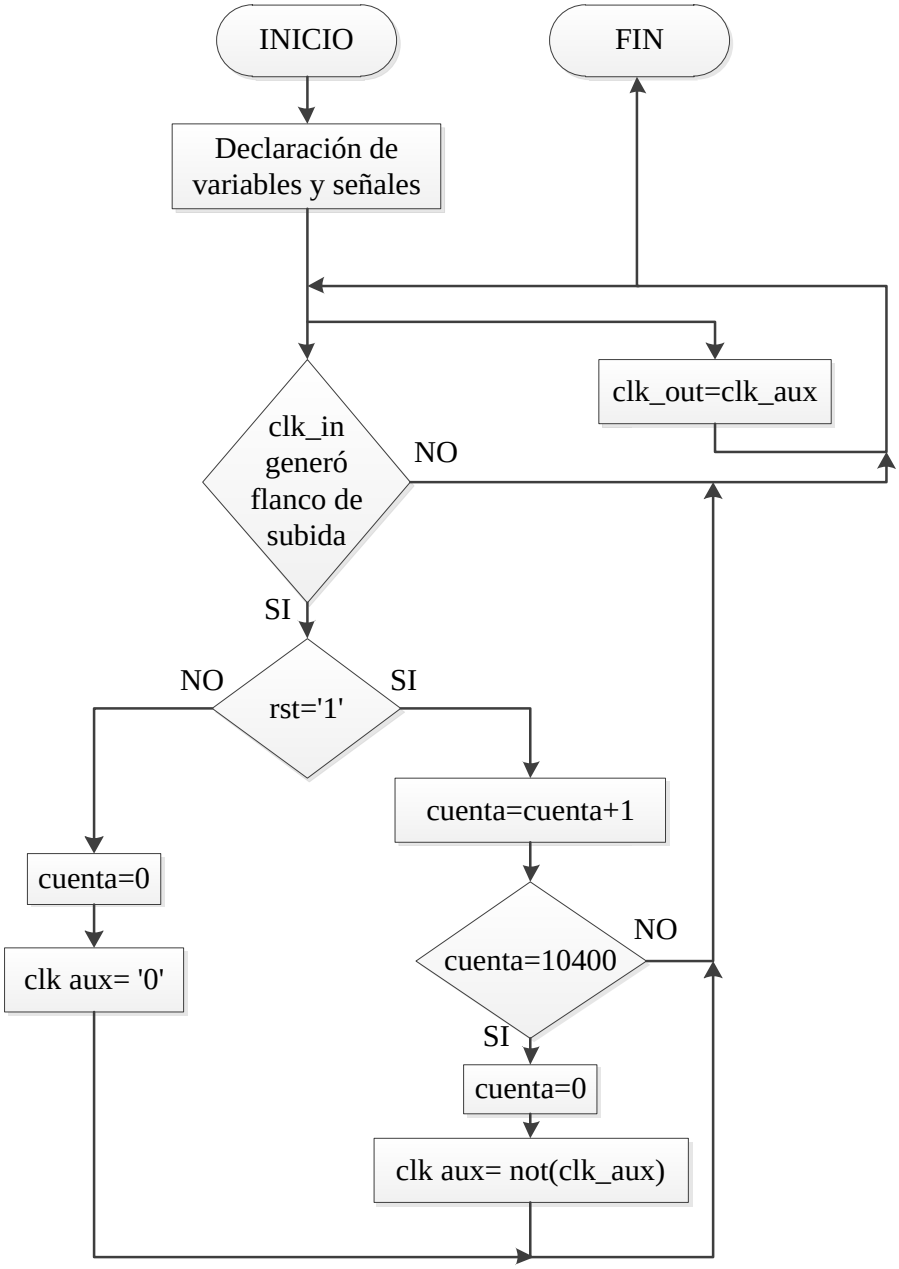


Figura 14. Diagrama de Flujo del Generador de tasa de baudios.

El código de programación requerido para este módulo es el siguiente:

```
--------DECLARACIÓN DE LIBRERÍAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
------FIN DECLARACIÓN DE LIBRERÍAS------

------SEÑALES I/O------
---Se realiza la declaración de señales
---de entrada y salida del módulo
entity GenBaudios is
    Port ( rst : in  STD_LOGIC;
           clk_in : in  STD_LOGIC;
           clk_out : out  STD_LOGIC);
end GenBaudios;
----FIN SEÑALES I/O----
------------------------ARQUITECTURA-------------------------
---Se describe y se modela el hardware mediante código,
---describiendo así el flujo de datos y comportamiento en este módulo.
architecture Behavioral of GenBaudios is
begin ---Inicio del cuerpo de la arquitectura

-------PROGRAMACIÓN PROCESS-----
---Mediante la estructura de un process se puede
---introducir código con una concepción de
---estados secuenciales, sin embargo
---al compilar será convertido en
---una estructura de hardware
process (clk_in)
-------VARIABLES-------
---Se realiza la declaración de variables
---auxiliares para los procesos internos en
---este bloque de programación
variable cuenta: integer:=0;
variable clk_aux: std_logic:='0';
------FIN VARIABLES-----

begin ---Inicio del cuerpo del process
---Se procede a realizar la división de
---frecuencia del reloj interno de la tarjeta,
---cuya frecuencia de es de 50MHZ, con el propósito de
---obtener una tasa de baudios de 2400
    if rising_edge(clk_in) then ---Detección de flanco de subida de clk_in
        if rst= '1' then
            cuenta:=cuenta+1;
            if cuenta= 10400 then ---Valor de la división de frecuencia
                cuenta:=0;
                clk_aux := not clk_aux;
            end if;
        else
            cuenta:=0;
            clk_aux :='0';
        end if;
    end if;
    clk_out <= clk_aux;
end process;
-----FIN PROGRAMACIÓN PROCESS----
end Behavioral;
-----------------------FIN ARQUITECTURA----------------------
```

Módulo Disparador

Este módulo es encargado de proporcionar una señal a la máquina de estados para que ella inicie y finalice el proceso de recepción de cada byte. Basándose en el protocolo de comunicación serial se toma la señal **baud,** que determina la frecuencia de muestreo de la señal **Rx** (canal de Recepción)**,** se detecta el bit de arranque para activar la señal **Recibir,** hasta que se detecte el bit de parada. Mediante el diagrama RTL de la Figura 15, se ilustra este módulo.

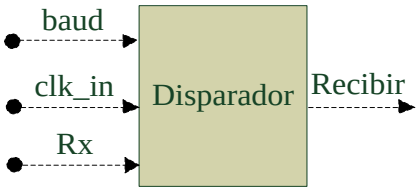


Figura 15. Diagrama RTL del activador de recepción

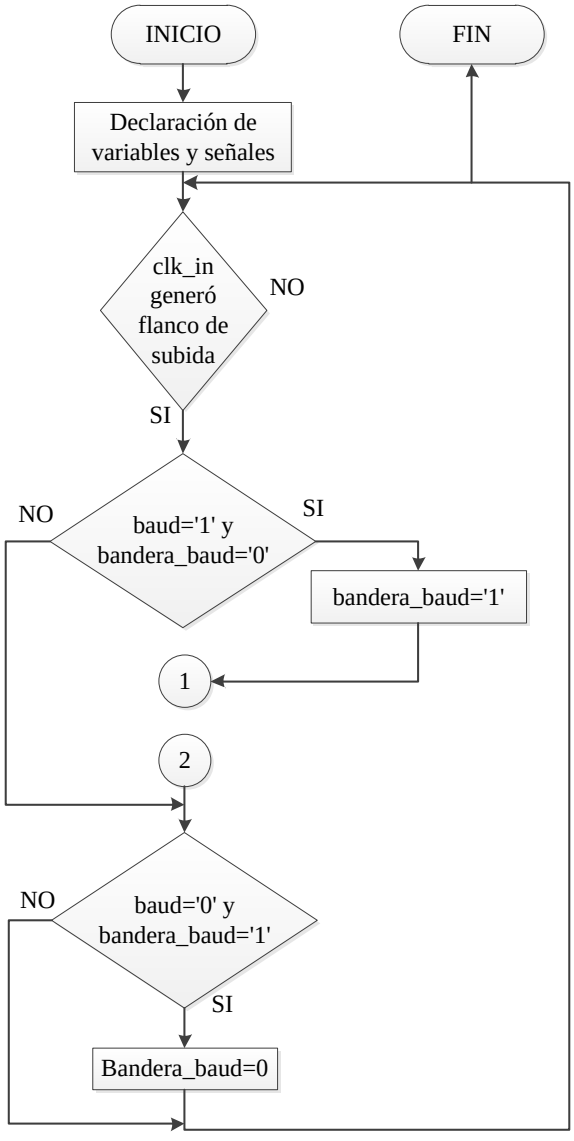


Figura 16. Diagrama de flujo del activador de recepción 1 de 2.

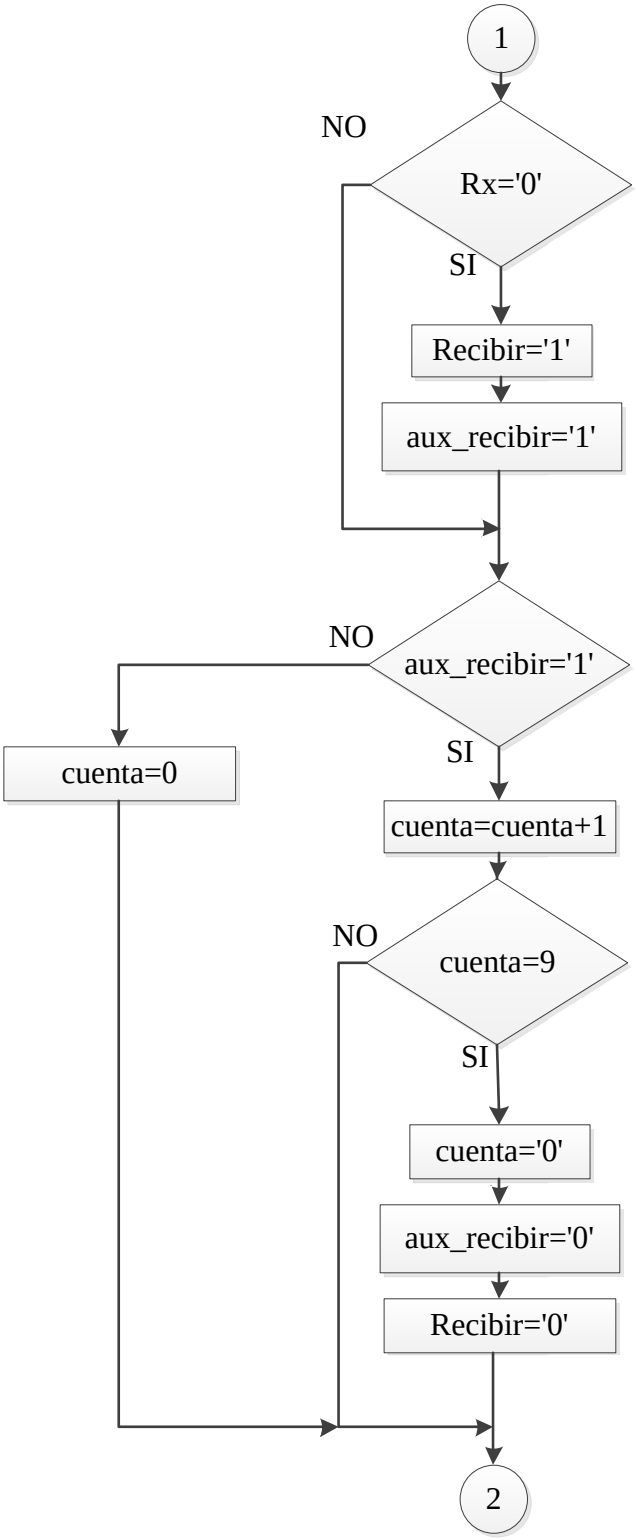


Figura 17. Diagrama de flujo del activador de recepción 2 de 2.

El código de programación requerido para este módulo es el siguiente:

```
--------DECLARACIÓN DE LIBRERÍAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
------FIN DECLARACIÓN DE LIBRERÍAS------

------SEÑALES I/O------
---Se realiza la declaración de señales
---de entrada y salida del módulo
entity Disparador is
    Port ( clk_in : in STD_LOGIC;

              baud : in  STD_LOGIC;
           Rx : in  STD_LOGIC;
           Recibir : out  STD_LOGIC);
end Disparador;
----FIN SEÑALES I/O----

------------------------ARQUITECTURA-------------------------
---Se describe y se modela el hardware mediante código,
---describiendo así el flujo de datos y comportamiento en este módulo.
architecture Behavioral of Disparador is
signal bandera_baud : std_logic:='0'; --Señal auxiliar
begin ---Inicio del cuerpo de la arquitectura

-------PROGRAMACIÓN PROCESS-----
---Mediante la estructura de un process se puede
---introducir código con una concepción de
---estados secuenciales, sin embargo
---al compilar será convertido en
---una estructura de hardware
process (clk_in)
-------VARIABLES-------
---Se realiza la declaración de variables
---auxiliares para los procesos internos en
---este bloque de programación
variable cuenta: integer range 0 to 10:= 0;
variable aux_recibir: std_logic;
------FIN VARIABLES-----

begin ---Inicio del cuerpo del process
---Se procede a generar una señal que permite
---establecer el inicio y fin de una recepción
if rising_edge(clk_in) then ---Detección de flanco de subida de clk_in

if baud = '1' and bandera_baud = '0' then
        bandera_baud <= '1';
        if (Rx='0') then
            Recibir <= '1';
            aux_recibir := '1';
        end if;
        if (aux_recibir = '1') then
            cuenta := cuenta + 1;
            if cuenta = 9 then
                cuenta:=0;
                aux_recibir := '0';
                Recibir <= '0';
            end if;
        else
            cuenta := 0;
        end if;
end if;
if baud = '0' and bandera_baud = '1' then
    bandera_baud <= '0';
end if;
end if;
end process;
-----FIN PROGRAMACIÓN PROCESS----
end Behavioral;
-----------------------FIN ARQUITECTURA----------------------
```

Módulo MaquinaEstadosR

Este módulo es una máquina de estados, y su función principal es realizar la recepción de datos bit a bit, a través de la señal **Rx**, a una frecuencia de muestreo de la señal **baud**, solo cuando la señal **Recibiendo** se encuentre activada. Almacena cada uno de los bits en un arreglo y los muestra a través de la señal **Dato.** Tiene la señal denominada habilitador, la misma que se activa cuando la maquina ha finalizado la recepción de un byte, dicha señal servirá al módulo Top2_Recepcion. El funcionamiento de esta máquina se puede apreciar en su diagrama de estados el cual se muestra en la Figura 18.

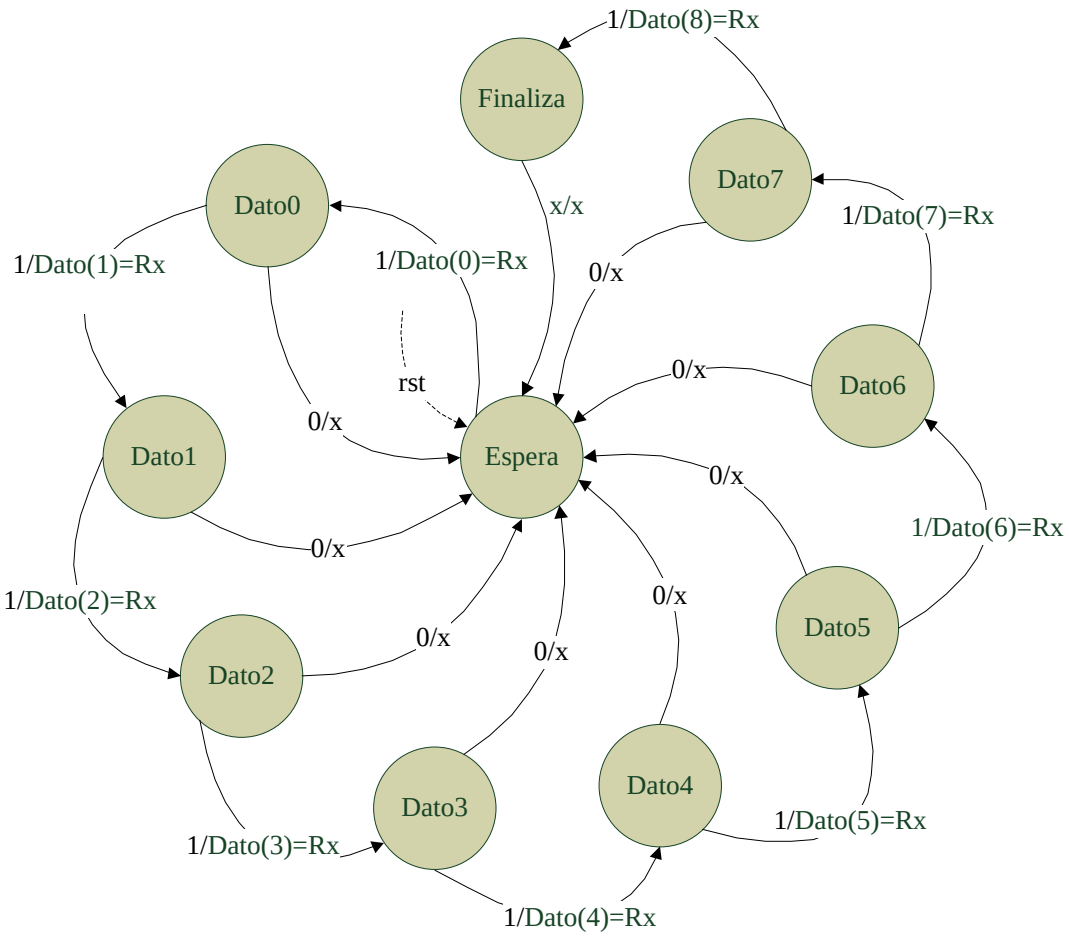


Figura 18. Diagrama de estados de la máquina.

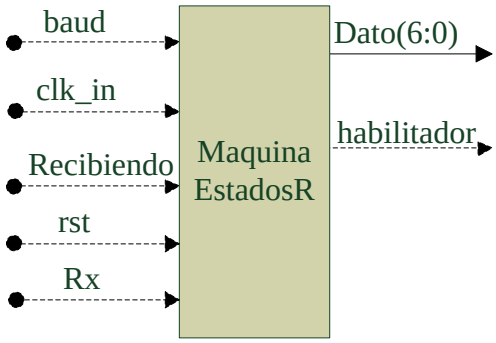


Figura 19. Diagrama RTL de la máquina de estados.

El código de programación requerido para este módulo es el siguiente:

```
--------DECLARACIÓN DE LIBRERÍAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
------FIN DECLARACIÓN DE LIBRERÍAS------

------SEÑALES I/O------
---Se realiza la declaración de señales
---de entrada y salida del módulo
entity MaquinaEstadosR is
    Port ( clk_in : in  STD_LOGIC;
           baud : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           Recibiendo : in  STD_LOGIC;
           Rx : in  STD_LOGIC;
           Dato : out  STD_LOGIC_VECTOR (6 downto 0);
           habilitador : out  STD_LOGIC);
end MaquinaEstadosR;
----FIN SEÑALES I/O----
------------------------ARQUITECTURA-------------------------
---Se describe y se modela el hardware mediante código,
---describiendo así el flujo de datos y comportamiento en este módulo.
architecture Behavioral of MaquinaEstadosR is
---Declaración de estados de la maquina
type Estados is (Espera, Dato0, Dato1, Dato2, Dato3, Dato4, Dato5, Dato6, Finaliza);

------SEÑALES------
---Se realiza la declaración de señales
---auxiliares internas al módulo
signal proxEstado, estadoActual: Estados;
signal Dato_aux : std_logic_vector(6 downto 0);
signal bandera_baud : std_logic:='0';
signal habilitador_aux : std_logic:='0';
------FIN SEÑALES------
begin ---Inicio del cuerpo de la arquitectura


-------------REGISTRO DE ESTADO-------------------
---Se determina bajo qué condiciones
---se va a pasar de un estado a otro
process (clk_in)
begin
if rising_edge(clk_in) then
if baud = '1' and bandera_baud = '0' then
        bandera_baud <= '1';
      if rst='1' then
```

```
        estadoActual <= proxEstado;
      else
        estadoActual <= Espera;
      end if;
end if;

if baud = '0' and bandera_baud = '1' then
    bandera_baud <= '0';
end if;
end if;
end process;
------------FIN REGISTRO DE ESTADO--------------

----------LÓGICA DEL ESTADO SIGUIENTE-------------
---Se determina en qué orden va a realizarse el
---cambio de estado, además se genera una respuesta
---en cada cambio de estado
process (estadoActual, Recibiendo, Rx, baud, Dato_aux)
begin
   case (estadoActual) is

-----------ESTADO Espera--------------
        when Espera =>
        habilitador_aux <= '0';
        if Recibiendo = '1' then
            proxEstado <= Dato0;
            Dato_aux(0) <= Rx;
        else
            proxEstado <= Espera;
            Dato_aux(0) <= '0';
        end if;
---------FIN ESTADO Espera------------

-----------ESTADO Dato0--------------
        when Dato0 =>
        habilitador_aux <= '0';
        if Recibiendo = '1' then
            proxEstado <= Dato1;
            Dato_aux(1) <= Rx;
        else
            proxEstado <= Espera;
            Dato_aux(1) <= '0';
        end if;
---------FIN ESTADO Dato0------------

-----------ESTADO Dato1--------------
        when Dato1 =>
        habilitador_aux <= '0';
        if Recibiendo = '1' then
            proxEstado <= Dato2;
            Dato_aux(2) <= Rx;
        else
            proxEstado <= Espera;
            Dato_aux(2) <= '0';
        end if;
---------FIN ESTADO Dato1------------


-----------ESTADO Dato2--------------
        when Dato2 =>
        habilitador_aux <= '0';
        if Recibiendo = '1' then
            proxEstado <= Dato3;
            Dato_aux(3) <= Rx;
        else
            proxEstado <= Espera;
            Dato_aux(3) <= '0';
        end if;
---------FIN ESTADO Dato2------------

-----------ESTADO Dato3--------------
        when Dato3 =>
        habilitador_aux <= '0';
        if Recibiendo = '1' then
```

```
            proxEstado <= Dato4;
            Dato_aux(4) <= Rx;
        else
            proxEstado <= Espera;
            Dato_aux(4) <= '0';
        end if;
---------FIN ESTADO Dato3------------

-----------ESTADO Dato4--------------
        when Dato4 =>
        habilitador_aux <= '0';
        if Recibiendo = '1' then
            proxEstado <= Dato5;
            Dato_aux(5) <= Rx;
        else
            proxEstado <= Espera;
            Dato_aux(5) <= '0';
        end if;
---------FIN ESTADO Dato4------------

-----------ESTADO Dato5--------------
        when Dato5 =>
        habilitador_aux <= '0';
        if Recibiendo = '1' then
            proxEstado <= Dato6;
            Dato_aux(6) <= Rx;
        else
            proxEstado <= Espera;
            Dato_aux(6) <= '0';
        end if;
---------FIN ESTADO Dato5------------

-----------ESTADO Dato6--------------
        when Dato6 =>
        habilitador_aux <= '0';
        if Recibiendo = '1' then
            proxEstado <= Finaliza;
        else
            proxEstado <= Espera;
        end if;
---------FIN ESTADO Dato6------------

-----------ESTADO Finaliza--------------
        when Finaliza =>
        proxEstado <= Espera;

        habilitador_aux <= '1';
        Dato <= Dato_aux;
---------FIN ESTADO Finaliza------------

   end case;

end process;
----------LÓGICA DEL ESTADO SIGUIENTE-------------

habilitador <= not habilitador_aux;---Se exporta la señal habilitador_aux

end Behavioral;
-----------------------FIN ARQUITECTURA----------------------
```

Módulo Top2_Recepcion

En este módulo se realiza la recepción de dos paquetes de datos de 7 bits, cada paquete ingresa a través de la señal de tipo vector **Dato_7bits**, a una frecuencia de muestreo determinada por la señal **habilitador**. Cada vez que se ha realizado la adquisición de dos paquetes de datos se procede a fusionarlos para formar un solo

paquete de datos mostrado en la señal de tipo vector **Dato_11bits** y la señal **signo**. **Dato_11bits** representa el numero en binario correspondiente a cada peso, y la señal **signo** representa el signo de cada peso. Se genera una señal denominada **habilitante** que indica en que instante se encuentran los datos completos y listos para que se realice de ellos un muestreo. Además se realiza una cuenta de la señal **habilitador**, que al dividirla para dos da como resultado el orden del peso correspondiente que se muestra a través de la señal de tipo vector **DireccionC** que le servirá a la RNA para determinar cuál es el lugar que debe tomar un peso y signo en la RNA.

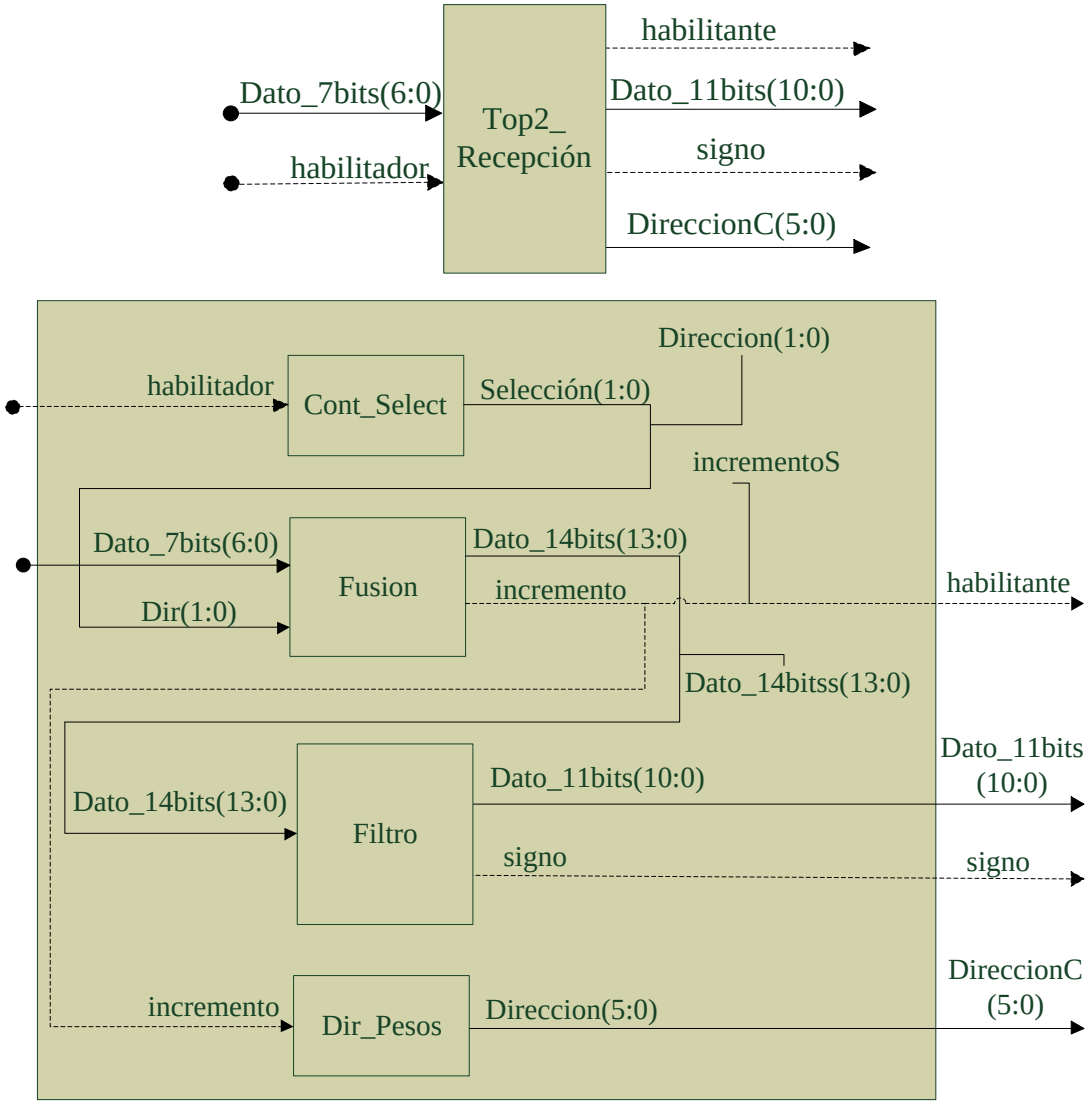


Figura 20. Diagrama RTL de recepción y fusión de paquetes de siete bits.

El código de programación requerido para este módulo es el siguiente:

```
--------DECLARACIÓN DE LIBRERÍAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
------FIN DECLARACIÓN DE LIBRERÍAS------

------SEÑALES I/O------
---Se realiza la declaración de señales
---de entrada y salida del módulo
entity Top2_Recepcion is
    Port ( Dato_7bits : in  STD_LOGIC_VECTOR (6 downto 0);
           habilitador : in  STD_LOGIC;
              signo : out  STD_LOGIC;
              habilitante : out std_logic;
              DireccionC : out  STD_LOGIC_VECTOR (5 downto 0);
           Dato_11bits : out  STD_LOGIC_VECTOR (10 downto 0));
end Top2_Recepcion;
----FIN SEÑALES I/O----

------------------------ARQUITECTURA------------------------
---Se describe y se modela el hardware mediante código,
---describiendo así el flujo de datos y comportamiento en este módulo.
architecture Behavioral of Top2_Recepcion is

---SEÑALES AUXILIARES INTERNAS-----
---Se declaran señales internas al módulo,
---útiles para la interconexión entre módulos
---internos
signal Direccion: std_logic_vector(1 downto 0);
signal Dato_14bitss :  STD_LOGIC_VECTOR (13 downto 0);
signal incrementoS :  STD_LOGIC;
---FIN SEÑALES AUXILIARES INTERNAS---

------MÓDULOS INTERNOS------
---Se Declaran los módulos internos con sus
---correspondientes entradas y salidas

---MÓDULO Cont_Select---
---Dentro del módulo Top2_Recepcion se declara un módulo
---interno denominado Cont_Select, con sus
---correspondientes entradas y salidas
COMPONENT Cont_Select
    PORT(
        habilitador : IN std_logic;
        Seleccion : OUT std_logic_vector(1 downto 0)
        );
    END COMPONENT;
---FIN MÓDULO Cont_Select---

---MÓDULO Fusion---
---Dentro del módulo Top2_Recepcion se declara un módulo
---interno denominado Fusion, con sus
---correspondientes entradas y salidas
    COMPONENT Fusion
    PORT(
        Dato_7bits : IN std_logic_vector(6 downto 0);
        Dir : IN std_logic_vector(1 downto 0);
        incremento : out STD_LOGIC;
        Dato_14bits : OUT std_logic_vector(13 downto 0)
        );
    END COMPONENT;
---FIN MÓDULO Fusion---

---MÓDULO Filtro---
---Dentro del módulo Top2_Recepcion se declara un módulo
---interno denominado Filtro, con sus
---correspondientes entradas y salidas
COMPONENT Filtro
```

```
    PORT(
        Dato_14bits : IN std_logic_vector(13 downto 0);
        Dato_11bits : OUT std_logic_vector(10 downto 0);
        signo : OUT std_logic
        );
    END COMPONENT;
---FIN MÓDULO Fusion---

---MÓDULO Dir_Pesos---
---Dentro del módulo Top2_Recepcion se declara un módulo
---interno denominado Dir_Pesos, con sus
---correspondientes entradas y salidas
COMPONENT Dir_Pesos
    PORT(
        incremento : IN std_logic;
        Direccion : OUT std_logic_vector(5 downto 0)
        --habilitante : OUT std_logic
        );
    END COMPONENT;
---FIN MÓDULO Dir_Pesos---

----INSTANCIACIÓN----
---Se realiza la interconexión interna
---entre módulos internos, mediante sus
---correspondientes señales de entrada y salida.
begin

habilitante <= incrementoS; --Señal auxiliar interna

---CONEXIÓN DEL MÓDULO Cont_Select---
---Se especifica con que señales del exterior
---van a ser conectadas las señales de entrada
---y salida del módulo Cont_Select
Inst_Cont_Select: Cont_Select PORT MAP(
        habilitador => habilitador,
        Seleccion => Direccion
    );
---FIN CONEXIÓN DEL MÓDULO Cont_Select---

---CONEXIÓN DEL MÓDULO Fusion---
---Se especifica con que señales del exterior
---van a ser conectadas las señales de entrada
---y salida del módulo Fusion
    Inst_Fusion: Fusion PORT MAP(
        Dato_7bits => Dato_7bits,
        Dir => Direccion,
        incremento => incrementoS,
        --incremento => habilitante,
        Dato_14bits => Dato_14bitss
    );
---FIN CONEXIÓN DEL MÓDULO Fusion---
---CONEXIÓN DEL MÓDULO Filtro---
---Se especifica con que señales del exterior
---van a ser conectadas las señales de entrada
---y salida del módulo Filtro
    Inst_Filtro: Filtro PORT MAP(
        Dato_14bits => Dato_14bitss,
        Dato_11bits => Dato_11bits,
        signo => signo
    );
---FIN CONEXIÓN DEL MÓDULO Filtro---
---CONEXIÓN DEL MÓDULO Dir_Pesos---
---Se especifica con que señales del exterior
---van a ser conectadas las señales de entrada
---y salida del módulo Dir_Pesos
    Inst_Dir_Pesos: Dir_Pesos PORT MAP(
        incremento => incrementoS,
        Direccion => DireccionC
            );
---FIN CONEXIÓN DEL MÓDULO Dir_Pesos---

end Behavioral;
-----FIN INSTANCIACIÓN----
-----------------------FIN ARQUITECTURA----------------------
```

Módulo Cont_Select

En este módulo se detectan los flancos de bajada de la señal **habilitador** y se incrementa una cuenta que se muestra en la señal de tipo vector **Selección**, y se reinicia a “00” cuando la cuenta alcanza el valor de “10”. La razón por la cual se realiza la cuenta en un vector de dos bits, es porque se necesita saber la ubicación de cada paquete, es decir si un paquete de 7 bits corresponde a un valor LSB o a MSB, con el propósito de unirlos de manera adecuada.

Figura 21. Diagrama RTL del contador identificador de paquetes Msb/Lsb.

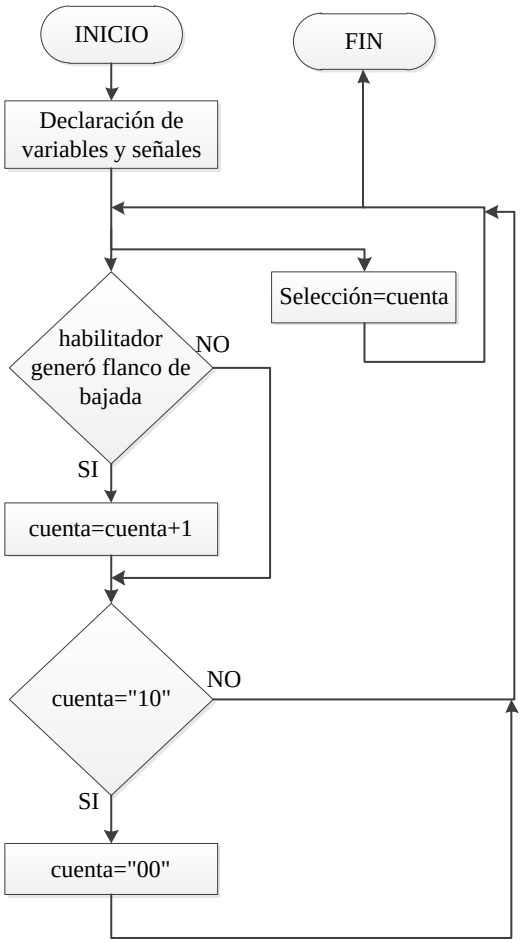


Figura 22. Diagrama de flujo del contador identificador de paquetes Msb/Lsb.

El código de programación requerido para este módulo es el siguiente:

```
--------DECLARACIÓN DE LIBRERÍAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
------FIN DECLARACIÓN DE LIBRERÍAS------

------SEÑALES I/O------
---Se realiza la declaración de señales
---de entrada y salida del módulo
entity Cont_Select is
    Port ( habilitador : in  STD_LOGIC;
           Seleccion : out  STD_LOGIC_VECTOR (1 downto 0));
end Cont_Select;
----FIN SEÑALES I/O----

------------------------ARQUITECTURA------------------------
---Se describe y se modela el hardware mediante código,
---describiendo así el flujo de datos y comportamiento en este módulo.
architecture Behavioral of Cont_Select is
---Señal auxiliar interna
signal cuenta : std_logic_vector(1 downto 0) := "00";
begin ---Inicio del cuerpo de la arquitectura

-------PROGRAMACIÓN PROCESS-----
---Mediante la estructura de un process se puede
---introducir código con una concepción de
---estados secuenciales, sin embargo
---al compilar será convertido en
---una estructura de hardware
process (habilitador, cuenta)
begin ---Inicio del cuerpo del process
---Se genera una cuenta que consta de dos bits de datos
---que sirve como una dirección en el módulo Fusion
---para determinar si se trata de un dato LSB o MSB
   if habilitador='0' and habilitador'event then
      cuenta <= cuenta + 1;
    end if;

    if cuenta = "10" then
      cuenta <= "00";
    end if;

end process;
-----FIN PROGRAMACIÓN PROCESS----

Seleccion <= cuenta; ---Se exporta la señal cuenta
end Behavioral;
----------------------FIN ARQUITECTURA---------------------
```

Módulo Fusion

La unión de los dos paquetes de 7 bits, se realiza en este módulo, cada paquete ingresa a través de la señal de tipo vector **Dato_7bits**, ordenadamente, primero LSB luego MSB, debido a que en el envío se transfiere también en ese orden. Para determinar a qué orden significativo corresponde cada paquete de datos se identifica la dirección mostrada en la señal de tipo vector **Dir.** La unión de ambos paquetes de datos se muestra a través de la señal de tipo vector **Dato_14bits**, la misma que representa un valor de un peso codificado que se lo va a decodificar en el siguiente

bloque. Además este módulo calcula la señal **incremento**, dicha señal será necesaria para realizar una cuenta y determinar que peso está transmitiéndose en un instante determinado.

Figura 23. Diagrama RTL de la fusión de paquetes Lsb con Msb.

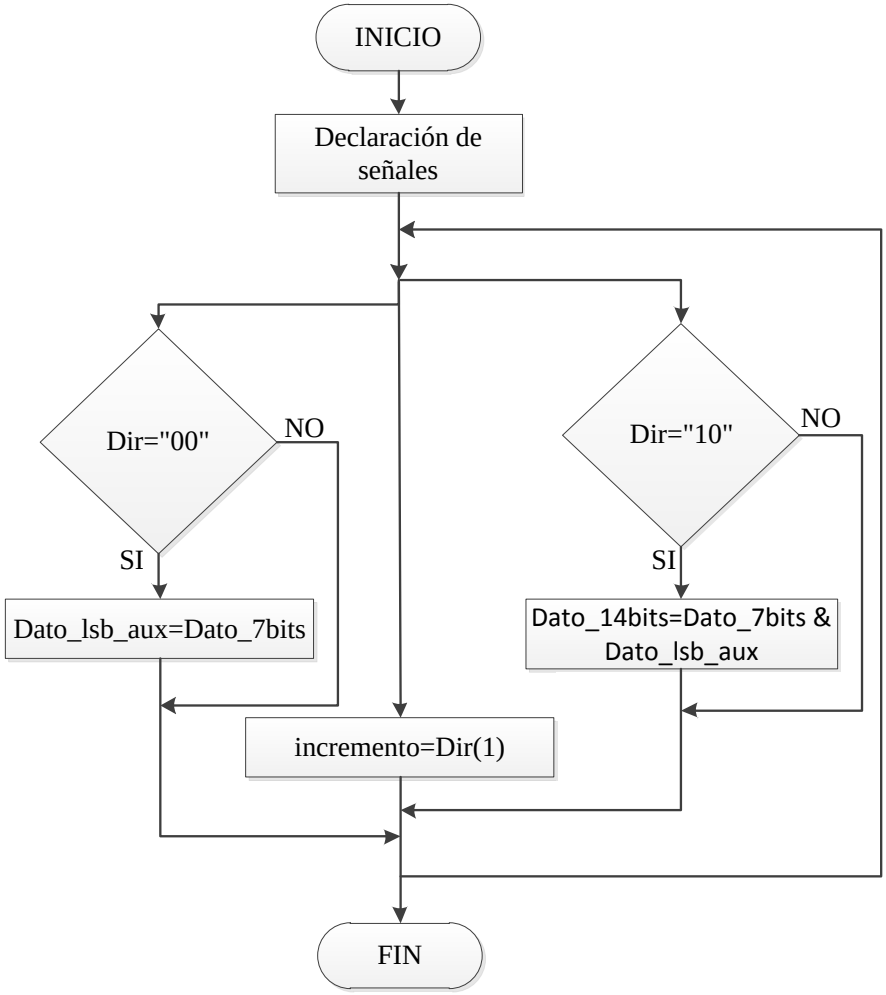


Figura 24. Diagrama de flujo de la fusión de paquetes Lsb con Msb.

El código de programación requerido para este módulo es el siguiente:

```
--------DECLARACIÓN DE LIBRERÍAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
------FIN DECLARACIÓN DE LIBRERÍAS------

------SEÑALES I/O------
---Se realiza la declaración de señales
---de entrada y salida del módulo
entity Fusion is
    Port ( Dato_7bits : in  STD_LOGIC_VECTOR (6 downto 0);
           Dir : in  STD_LOGIC_VECTOR (1 downto 0);
              incremento : out STD_LOGIC;
           Dato_14bits : out  STD_LOGIC_VECTOR (13 downto 0));
end Fusion;
----FIN SEÑALES I/O----
------------------------ARQUITECTURA-------------------------
---Se describe y se modela el hardware mediante código,
---describiendo así el flujo de datos y comportamiento en este módulo
architecture Behavioral of Fusion is
---Declaración de una señal auxiliar
signal Dato_lsb_aux : std_logic_vector(6 downto 0);
begin ---Inicio del cuerpo de la arquitectura
---Se realiza la unión de dos paquetes de datos
---los cuales corresponden a los datos LSB y MSB
    Dato_lsb_aux <= Dato_7bits when Dir = "00";
    Dato_14bits <= Dato_7bits & Dato_lsb_aux when Dir = "10";
    incremento <= Dir(1);
end Behavioral;
------------------------FIN ARQUITECTURA----------------------
```

Módulo Filtro

En este módulo se realizó una decodificación de datos debido a que al enviar desde Matlab los datos “0000000” y “1111111” correspondientes a 0 y 127 respectivamente, se obtenían datos que no correspondían. La señal **Dato_14bits** contiene bits extras que permiten decodificar cada uno de los pesos. A la salida de este módulo se tiene una señal **Dato_11bits** que corresponde al valor de cada peso y una señal **signo** que corresponde al signo que cada peso.

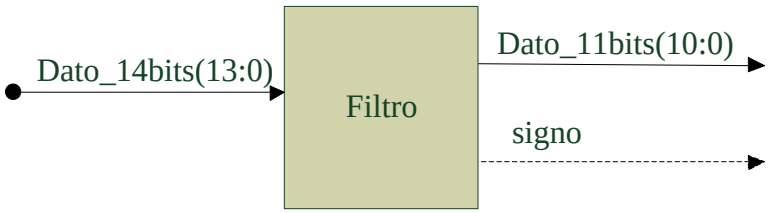


Figura 25. Diagrama RTL del decodificador de pesos.

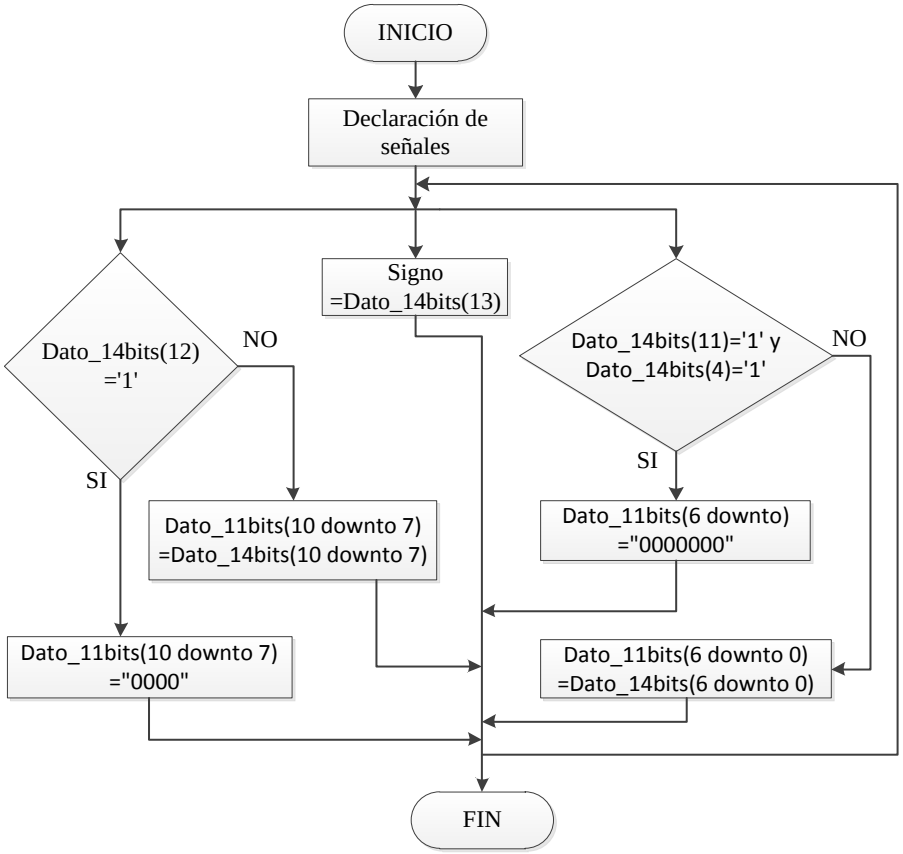


Figura 26. Diagrama de flujo del decodificador de pesos.

El código de programación requerido para este módulo es el siguiente:

```
--------DECLARACIÓN DE LIBRERÍAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
------FIN DECLARACIÓN DE LIBRERÍAS------
------SEÑALES I/O------
---Se realiza la declaración de señales
---de entrada y salida del módulo
entity Filtro is
    Port ( Dato_14bits : in  STD_LOGIC_VECTOR (13 downto 0);
           Dato_11bits : out  STD_LOGIC_VECTOR (10 downto 0);
           signo : out  STD_LOGIC);
end Filtro;
----FIN SEÑALES I/O----
------------------------ARQUITECTURA------------------------
---Se describe y se modela el hardware mediante código,
---describiendo así el flujo de datos y comportamiento en este módulo.
architecture Behavioral of Filtro is
begin ---Inicio del cuerpo de la arquitectura
---Se discrimina los bits que contienen información específica de los
---pesos y se separa el signo de su correspondiente peso
Dato_11bits(10 downto 7) <= "0000" when Dato_14bits(12) = '1' else
                    Dato_14bits(10 downto 7);
Dato_11bits(6 downto 0) <= "0000000" when (Dato_14bits(11) = '1' and Dato_14bits(4) =
'1')  else
Dato_14bits(6 downto 0);
signo <= Dato_14bits(13);
end Behavioral;
------------------------FIN ARQUITECTURA---------------------
```

Módulo DirPesos

Este módulo se encarga de realizar una cuenta de la señal **incremento**, con cada flanco de subida, permitiendo contar los pesos que han ingresado. Cada que se ha ingresado un nuevo peso el vector **Dirección** se actualiza, para que la RNA pueda identificar a un peso y almacenarlo en su lugar correspondiente

Figura 27. Diagrama RTL de la Identificación y Direccionamiento de pesos.

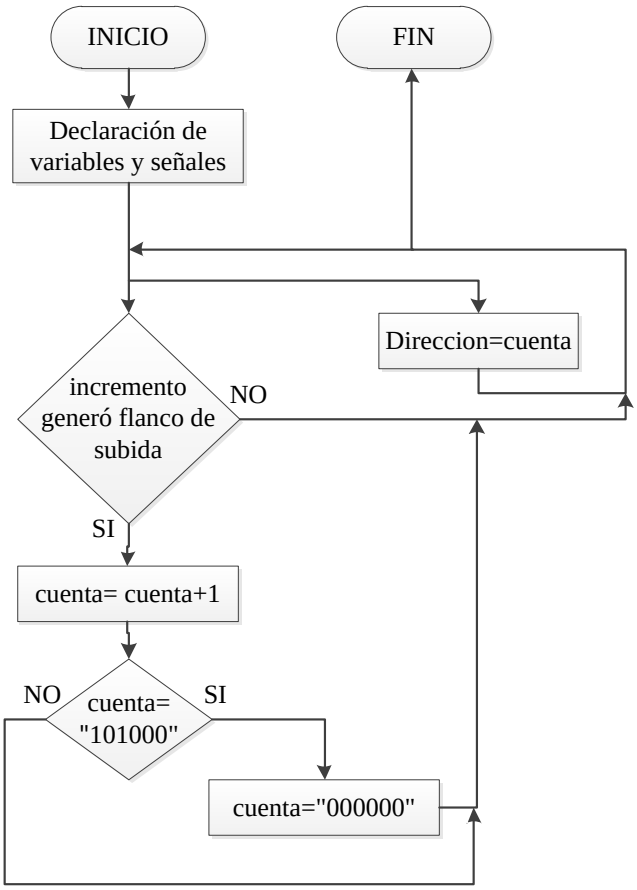


Figura 28. Diagrama de flujo de la identificación y direccionamiento de pesos.

El código de programación requerido para este módulo es el siguiente:

```
--------DECLARACIÓN DE LIBRERÍAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
------FIN DECLARACIÓN DE LIBRERÍAS------

------SEÑALES I/O------
---Se realiza la declaración de señales
---de entrada y salida del módulo
entity Dir_Pesos is
    Port ( incremento : in  STD_LOGIC;
           Direccion : out  STD_LOGIC_VECTOR (5 downto 0)
                     );
end Dir_Pesos;
----FIN SEÑALES I/O----

------------------------ARQUITECTURA------------------------
---Se describe y se modela el hardware mediante código,
---describiendo así el flujo de datos y comportamiento en este módulo.
architecture Behavioral of Dir_Pesos is
---Declaración de una señal auxiliar
signal cuenta : std_logic_vector(5 downto 0) := "000000";
begin ---Inicio del cuerpo de la arquitectura

-------PROGRAMACIÓN PROCESS-----
---Mediante la estructura de un process se puede
---introducir código con una concepción de
---estados secuenciales, sin embargo
---al compilar será convertido en
---una estructura de hardware
process (incremento, cuenta)
begin ---Inicio del cuerpo del process
---Se realiza un conteo de los pesos ingresados,
---dicho conteo se almacena en una señal de tipo vector
---que servirá como una Dirección para que las neuronas
---puedan determinar el peso y signo que les corresponde
   if incremento='1' and incremento'event then
      cuenta <= cuenta + 1;
    if cuenta = "101000" then
      cuenta <= "000000";
    end if;

    end if;

end process;
-----FIN PROGRAMACIÓN PROCESS----
Direccion <= cuenta; ---Se exporta la señal cuenta
end Behavioral;
------------------------FIN ARQUITECTURA----------------------
```

3.4. Acople de Lógica

La tarjeta FPGA trabaja con Lógica inversa en sus entradas y salidas, por lo tanto para facilitar el manejo de los datos se procedió a implementar un módulo que invierte la lógica de las señales **in1** a **in4,** dando como resultado las señales **Out_in1** a **Out_in4,** que son procesadas en el siguiente bloque correspondiente a la RNA.

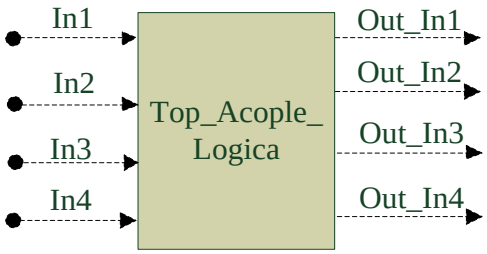


Figura 29. Diagrama RTL del negador de entradas.

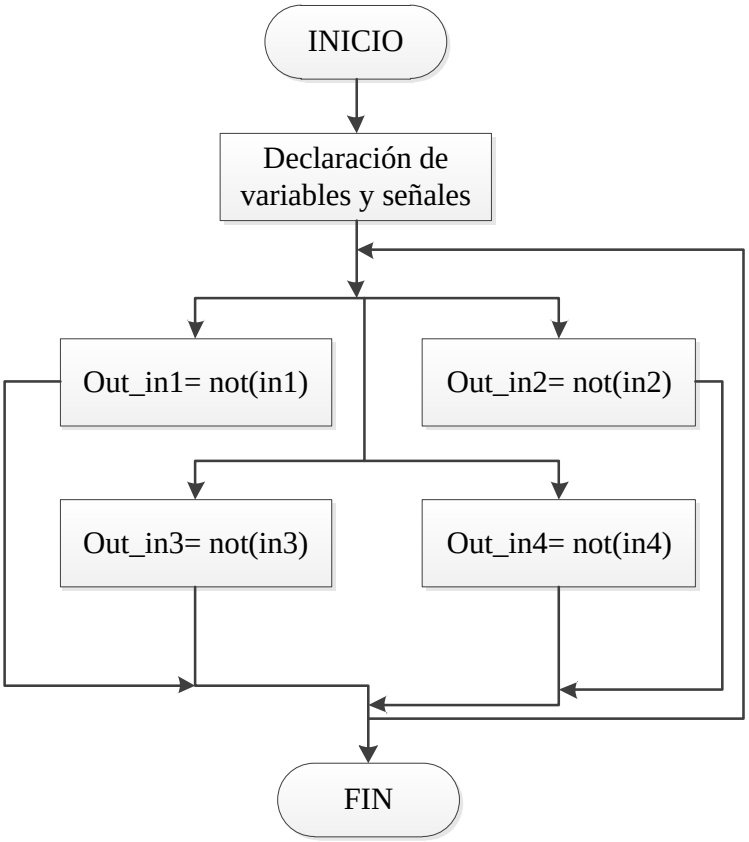


Figura 30. Diagrama de flujo del negador de entradas.

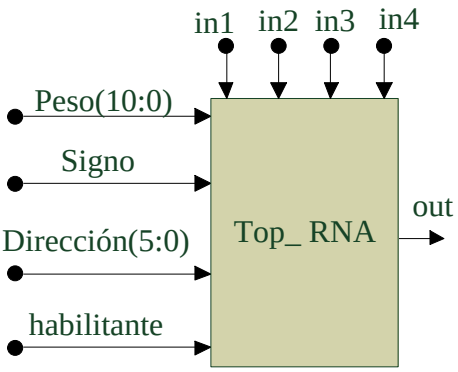
El código de programación requerido para este módulo es el siguiente:

```
--------DECLARACIÓN DE LIBRERÍAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
------FIN DECLARACIÓN DE LIBRERÍAS------

------SEÑALES I/O------
---Se realiza la declaración de señales
---de entrada y salida del módulo
```

```
entity Top_Acople_Logica is
    Port ( In1 : in  STD_LOGIC;
           In2 : in  STD_LOGIC;
           In3 : in  STD_LOGIC;
           In4 : in  STD_LOGIC;
           Out_In1 : out  STD_LOGIC;
           Out_In2 : out  STD_LOGIC;
           Out_In3 : out  STD_LOGIC;
           Out_In4 : out  STD_LOGIC);
end Top_Acople_Logica;
----FIN SEÑALES I/O----

------------------------ARQUITECTURA-------------------------
---Se describe y se modela el hardware mediante código,
---describiendo así el flujo de datos y comportamiento en este módulo.
architecture Behavioral of Top_Acople_Logica is
begin ---Inicio del cuerpo de la arquitectura
---Se realiza la negación de las variables de entrada
---para acoplar a la lógica en bajo de la FPGA utilizada
Out_In1 <= not In1;
Out_In2 <= not In2;
Out_In3 <= not In3;
Out_In4 <= not In4;

end Behavioral;
-----------------------FIN ARQUITECTURA---------------------
```

3.5. Implementación de la red neuronal en FPGA

En este bloque se implementa la arquitectura de la RNA, mediante la conexión de las neuronas correspondientes para que intercambien información conforme se ha mostrado en la Arquitectura que se muestra en el Capítulo 3.1. Cada una de las neuronas realiza su correspondiente recepción y almacenamiento de pesos, lo cual se detalla en los Capítulos 3.5.1 al 3.5.8.



El código de programación requerido para este módulo es el siguiente:

```
--------DECLARACIÓN DE LIBRERÍAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas.
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
------FIN DECLARACIÓN DE LIBRERÍAS------

------SEÑALES I/O------
---Se realiza la declaración de señales
---de entrada y salida del módulo
entity Top_Red_Neuronal is
    Port ( Dato_11bits : in  STD_LOGIC_VECTOR (10 downto 0);
           signo : in  STD_LOGIC;
           Direccion : in  STD_LOGIC_VECTOR (5 downto 0);
           In1 : in  STD_LOGIC;
           In2 : in  STD_LOGIC;
           In3 : in  STD_LOGIC;
           In4 : in  STD_LOGIC;
         habilitante : IN std_logic;
         Salida  : out  STD_LOGIC);
end Top_Red_Neuronal;
----FIN SEÑALES I/O----

------------------------ARQUITECTURA------------------------
---Se describe y se modela el hardware mediante código,
---describiendo así el flujo de datos y comportamiento en este módulo.
architecture Behavioral of Top_Red_Neuronal is

---SEÑALES AUXILIARES INTERNAS-----
---Se declaran señales internas al módulo,
---útiles para la interconexión entre módulos
---internos
signal sN1: std_logic;
signal sN2: std_logic;
signal sN3: std_logic;
signal sN4: std_logic;

signal N5_N8: std_logic;
signal N6_N8: std_logic;
signal N7_N8: std_logic;
---FIN SEÑALES AUXILIARES INTERNAS---

------MÓDULOS INTERNOS------
---Se Declaran los módulos internos con sus
---correspondientes entradas y salidas

---MÓDULO Neurona1---
---Dentro del módulo Top_Red_Neuronal se declara un módulo
---interno denominado Neurona1, con sus
---correspondientes entradas y salidas
COMPONENT Neurona1
    PORT(
        Dato_11bits : IN std_logic_vector(10 downto 0);
        signo : IN std_logic;
        Direccion : IN std_logic_vector(5 downto 0);
        In1 : IN std_logic;
        In2 : IN std_logic;
        In3 : IN std_logic;
        In4 : IN std_logic;
        habilitante : IN std_logic;
        SalidaN1 : OUT std_logic
        );
    END COMPONENT;
---FIN MÓDULO Neurona1---

---MÓDULO Neurona2---
---Dentro del módulo Top_Red_Neuronal se declara un módulo
---interno denominado Neurona2, con sus
---correspondientes entradas y salidas
    COMPONENT Neurona2
    PORT(
        Dato_11bits : IN std_logic_vector(10 downto 0);
        signo : IN std_logic;
        Direccion : IN std_logic_vector(5 downto 0);
        In1 : IN std_logic;
        In2 : IN std_logic;
        In3 : IN std_logic;
```

```
        In4 : IN std_logic;
        habilitante : IN std_logic;
        SalidaN2 : OUT std_logic
        );
    END COMPONENT;
---FIN MÓDULO Neurona2---

---MÓDULO Neurona3---
---Dentro del módulo Top_Red_Neuronal se declara un módulo
---interno denominado Neurona3, con sus
---correspondientes entradas y salidas
    COMPONENT Neurona3
    PORT(
        Dato_11bits : IN std_logic_vector(10 downto 0);
        signo : IN std_logic;
        Direccion : IN std_logic_vector(5 downto 0);
        In1 : IN std_logic;
        In2 : IN std_logic;
        In3 : IN std_logic;
        In4 : IN std_logic;
        habilitante : IN std_logic;
        SalidaN3 : OUT std_logic
        );
    END COMPONENT;
---FIN MÓDULO Neurona3---


---MÓDULO Neurona4---
---Dentro del módulo Top_Red_Neuronal se declara un módulo
---interno denominado Neurona4, con sus
---correspondientes entradas y salidas
    COMPONENT Neurona4
    PORT(
        Dato_11bits : IN std_logic_vector(10 downto 0);
        signo : IN std_logic;
        Direccion : IN std_logic_vector(5 downto 0);
        In1 : IN std_logic;
        In2 : IN std_logic;
        In3 : IN std_logic;
        In4 : IN std_logic;
        habilitante : IN std_logic;
        SalidaN4 : OUT std_logic
        );
    END COMPONENT;
---FIN MÓDULO Neurona4---


---MÓDULO Neurona5---
---Dentro del módulo Top_Red_Neuronal se declara un módulo
---interno denominado Neurona5, con sus
---correspondientes entradas y salidas
    COMPONENT Neurona5
    PORT(
        Dato_11bits : IN std_logic_vector(10 downto 0);
        signo : IN std_logic;
        Direccion : IN std_logic_vector(5 downto 0);
        In1 : IN std_logic;
        In2 : IN std_logic;
        In3 : IN std_logic;
        In4 : IN std_logic;
        habilitante : IN std_logic;
        SalidaN5 : OUT std_logic
        );
    END COMPONENT;
---FIN MÓDULO Neurona5---


---MÓDULO Neurona6---
---Dentro del módulo Top_Red_Neuronal se declara un módulo
---interno denominado Neurona6, con sus
---correspondientes entradas y salidas
    COMPONENT Neurona6
    PORT(
        Dato_11bits : IN std_logic_vector(10 downto 0);
```

```
        signo : IN std_logic;
        Direccion : IN std_logic_vector(5 downto 0);
        In1 : IN std_logic;
        In2 : IN std_logic;
        In3 : IN std_logic;
        In4 : IN std_logic;
        habilitante : IN std_logic;
        SalidaN6 : OUT std_logic
        );
    END COMPONENT;
---FIN MÓDULO Neurona6---


---MÓDULO Neurona7---
---Dentro del módulo Top_Red_Neuronal se declara un módulo
---interno denominado Neurona7, con sus
---correspondientes entradas y salidas
    COMPONENT Neurona7
    PORT(
        Dato_11bits : IN std_logic_vector(10 downto 0);
        signo : IN std_logic;
        Direccion : IN std_logic_vector(5 downto 0);
        In1 : IN std_logic;
        In2 : IN std_logic;
        In3 : IN std_logic;
        In4 : IN std_logic;
        habilitante : IN std_logic;
        SalidaN7 : OUT std_logic
        );
    END COMPONENT;
    ---FIN MÓDULO Neurona7---

---MÓDULO Neurona8---
---Dentro del módulo Top_Red_Neuronal se declara un módulo
---interno denominado Neurona8, con sus
---correspondientes entradas y salidas
    COMPONENT Neurona8
    PORT(
        Dato_11bits : IN std_logic_vector(10 downto 0);
        signo : IN std_logic;
        Direccion : IN std_logic_vector(5 downto 0);
        In1 : IN std_logic;
        In2 : IN std_logic;
        In3 : IN std_logic;
        habilitante : IN std_logic;
        SalidaN8 : OUT std_logic
        );
    END COMPONENT;
---FIN MÓDULO Neurona8---

----INSTANCIACIÓN----
---Se realiza la interconexión interna
---entre módulos internos, mediante sus
---correspondientes señales de entrada y salida.
begin

---CONEXIÓN DEL MÓDULO Neurona1---
---Se especifica con que señales del exterior
---van a ser conectadas las señales de entrada
---y salida del módulo Neurona1
Inst_Neurona1: Neurona1 PORT MAP(
        Dato_11bits => Dato_11bits,
        signo => signo,
        Direccion => Direccion,
        In1 => In1,
        In2 => In2,
        In3 => In3,
        In4 => In4,
        habilitante => habilitante,
        SalidaN1 => sN1
    );
---FIN CONEXIÓN DEL MÓDULO Neurona1---

---CONEXIÓN DEL MÓDULO Neurona2---
```

```
---Se especifica con que señales del exterior
---van a ser conectadas las señales de entrada
---y salida del módulo Neurona2
Inst_Neurona2: Neurona2 PORT MAP(
        Dato_11bits => Dato_11bits,
        signo => signo,
        Direccion => Direccion,
        In1 => In1,
        In2 => In2,
        In3 => In3,
        In4 => In4,
        habilitante => habilitante,
        SalidaN2 => sN2
    );
---FIN CONEXIÓN DEL MÓDULO Neurona2---

---CONEXIÓN DEL MÓDULO Neurona3---
---Se especifica con que señales del exterior
---van a ser conectadas las señales de entrada
---y salida del módulo Neurona3
    Inst_Neurona3: Neurona3 PORT MAP(
        Dato_11bits => Dato_11bits,
        signo => signo,
        Direccion => Direccion,
        In1 => In1,
        In2 => In2,
        In3 => In3,
        In4 => In4,
        habilitante => habilitante,
        SalidaN3 => sN3
    );
    ---FIN CONEXIÓN DEL MÓDULO Neurona3---

---CONEXIÓN DEL MÓDULO Neurona4---
---Se especifica con que señales del exterior
---van a ser conectadas las señales de entrada
---y salida del módulo Neurona4
    Inst_Neurona4: Neurona4 PORT MAP(
        Dato_11bits => Dato_11bits,
        signo => signo,
        Direccion => Direccion,
        In1 => In1,
        In2 => In2,
        In3 => In3,
        In4 => In4,
        habilitante => habilitante,
        SalidaN4 => sN4
    );
---FIN CONEXIÓN DEL MÓDULO Neurona4---


---CONEXIÓN DEL MÓDULO Neurona5---
---Se especifica con que señales del exterior
---van a ser conectadas las señales de entrada
---y salida del módulo Neurona5
Inst_Neurona5: Neurona5 PORT MAP(
        Dato_11bits => Dato_11bits,
        signo => signo,
        Direccion => Direccion,
        In1 => sN1,
        In2 => sN2,
        In3 => sN3,
        In4 => sN4,
        habilitante => habilitante,
        SalidaN5 => N5_N8
    );
---FIN CONEXIÓN DEL MÓDULO Neurona5---


---CONEXIÓN DEL MÓDULO Neurona6---
---Se especifica con que señales del exterior
---van a ser conectadas las señales de entrada
---y salida del módulo Neurona6
    Inst_Neurona6: Neurona6 PORT MAP(
```

```
        Dato_11bits => Dato_11bits,
        signo => signo,
        Direccion => Direccion,
        In1 => sN1,
        In2 => sN2,
        In3 => sN3,
        In4 => sN4,
        habilitante => habilitante,
        SalidaN6 => N6_N8
    );

---FIN CONEXIÓN DEL MÓDULO Neurona6---

---CONEXIÓN DEL MÓDULO Neurona7---
---Se especifica con que señales del exterior
---van a ser conectadas las señales de entrada
---y salida del módulo Neurona7
    Inst_Neurona7: Neurona7 PORT MAP(
        Dato_11bits => Dato_11bits,
        signo => signo,
        Direccion => Direccion,
        In1 => sN1,
        In2 => sN2,
        In3 => sN3,
        In4 => sN4,
        habilitante => habilitante,
        SalidaN7 => N7_N8
    );
    ---FIN CONEXIÓN DEL MÓDULO Neurona7---

---CONEXIÓN DEL MÓDULO Neurona8---
---Se especifica con que señales del exterior
---van a ser conectadas las señales de entrada
---y salida del módulo Neurona8
    Inst_Neurona8: Neurona8 PORT MAP(
        Dato_11bits => Dato_11bits,
        signo => signo,
        Direccion => Direccion,
        In1 => N5_N8,
        In2 => N6_N8,
        In3 => N7_N8,
        habilitante => habilitante,
        SalidaN8 => Salida
    );
---FIN CONEXIÓN DEL MÓDULO Neurona8---
end Behavioral;
-----FIN INSTANCIACIÓN----
----------------------FIN ARQUITECTURA----------------------
```

El diagrama RTL de los componentes internos de este módulo se encuentra ilustrado en la Figura 31.

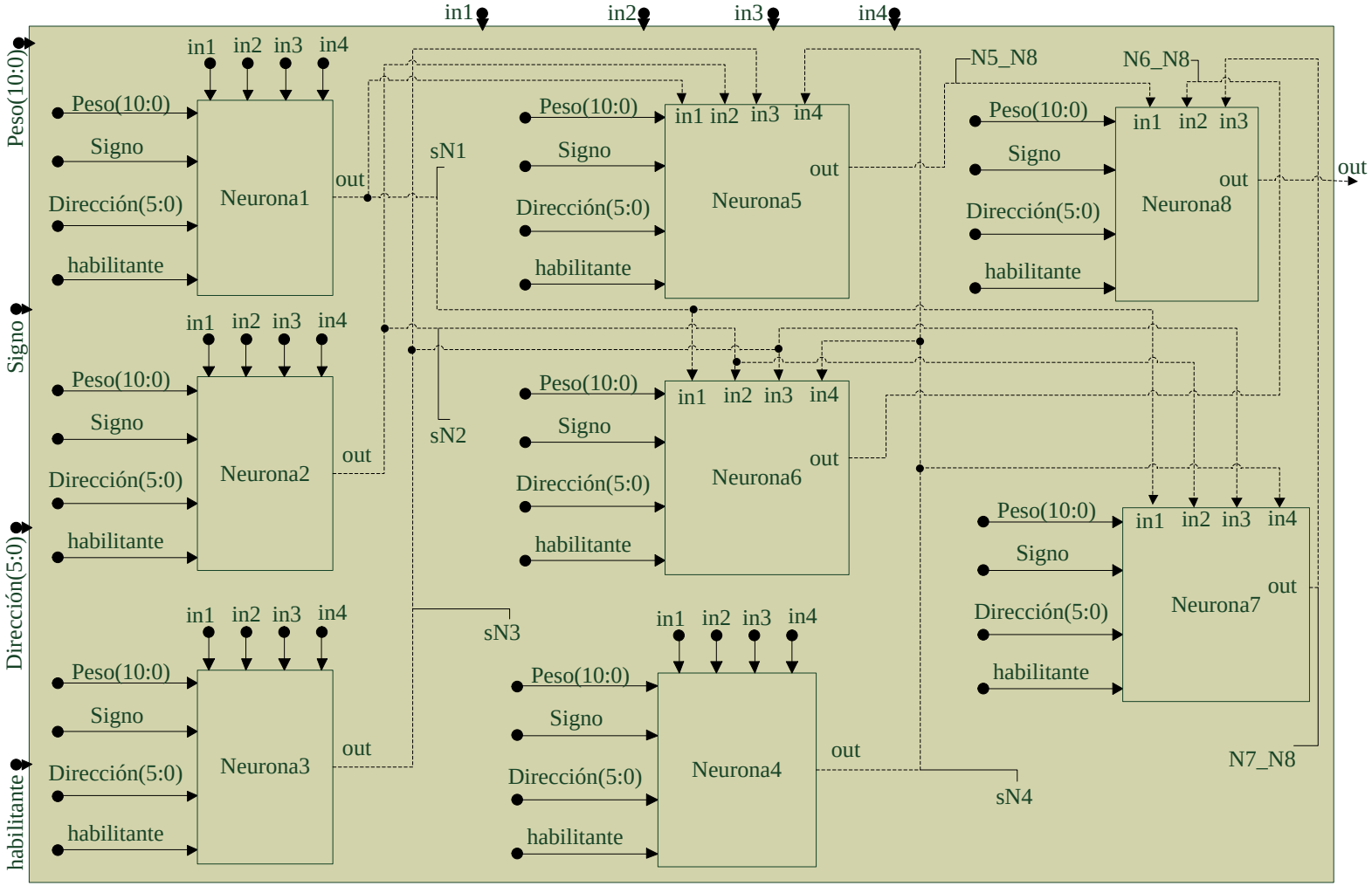


Figura 31. Diagrama RTL de la Red Neuronal Artificial

Las señales **in1** a **in4** en el caso de las neuronas 1 a 7, y las señales **in1** a **in3** en el caso de la neurona 8 son las señales que serán procedas en la puesta en marcha de la RNA como tal. Por la señal de tipo vector **Dato_11bits** y por la señal **signo** se realiza la recepción del peso y signo respectivamente y es almacenado en una variable de tipo vector que le proporciona una memoria interna a cada neurona. Las neuronas 1 a 7 requieren una cantidad de 5 pesos con sus correspondientes signos para cada una; la neurona 8 requiere una cantidad de 4 pesos con sus correspondientes signos. Para almacenar de manera ordenada y con el propósito de que cada neurona tenga bien identificado sus respectivos pesos para la suma ponderada que va a procesar es necesaria la señal de tipo vector **Dirección**. Cada neurona realiza un muestreo y almacenamiento de pesos y signos con una frecuencia determinada por la señal **habilitante**.

Las neuronas de la capa de entrada y de la capa oculta tienen la misma estructura de entradas y salidas y se puede observar en la Figura 32, sin embargo se debe tener en cuenta que no todas las entradas son las mismas para todas las neuronas, ya que cambian dependiendo de la capa en donde se encuentren. Es decir las neuronas 1, 2, 3 y 4 de la capa de entrada tendrán las entradas digitales que provienen del exterior. Las neuronas 5, 6 y 7 de la capa oculta tendrán entradas provenientes de las salidas de las neuronas de la capa de entrada. Dichas conexiones se ilustran en la arquitectura de la Figura 7 y en el diagrama RTL de la Figura 31

Cuando se realiza el entrenamiento se generan pesos específicos para cada neurona, y se envían de manera ordenada para que cada una pueda identificar y almacenar los pesos que le corresponden. Cada neurona está diseñada de tal forma que identifique la posición de los pesos que se le han designado, es decir que cada una difiere de las otras por las direcciones de los pesos que debe identificar. Sin embargo su estructura externa es exactamente igual en las neuronas 1 a 7, por lo tanto se muestra un solo diagrama RTL que ilustra a todos y cada uno de los módulos de las neuronas en la Figura 32.

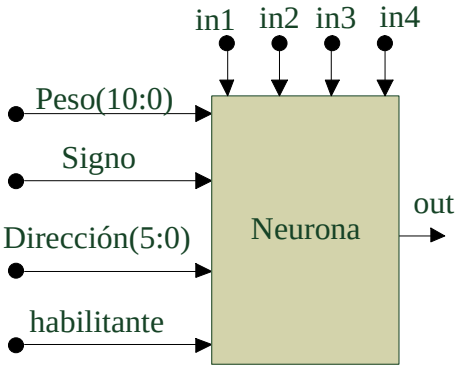


Figura 32. Diagrama RTL generalizado para las neuronas 1 a 7.

La neurona 8 de la capa de salida también se diferencia de las demás por las direcciones de los pesos que debe identificar, sin embargo ella muestra una diferencia adicional en su estructura, por su número de entradas digitales. Como se observa en la Figura 33 esta neurona tiene tres entradas digitales mientras que sus homólogas de la Figura 32 tienen cuatro. Las salidas de las neuronas de la capa oculta son las entradas de esta neurona, tal como se indica en la arquitectura mostrada en la Figura 7 y en el diagrama RTL de la Figura 31.

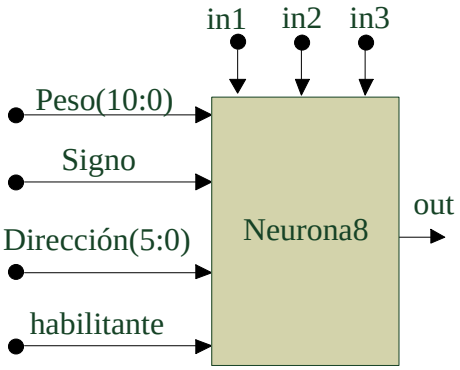


Figura 33. Diagrama RTL de la neurona 8.

Todas y cada una de las neuronas se rigen a los diagramas de flujo mostrados en la Figura 34, Figura 35 y Figura 36, y se asocian a un diagrama de flujo que es propio de cada neurona el cual les permite identificar la posición de los pesos que se les ha designado y por lo tanto no se lo puede generalizar para todas. Se muestran los

diagramas de flujo propios de las neuronas 1 a la 8 en la Figura 37 a la Figura 44, respectivamente.

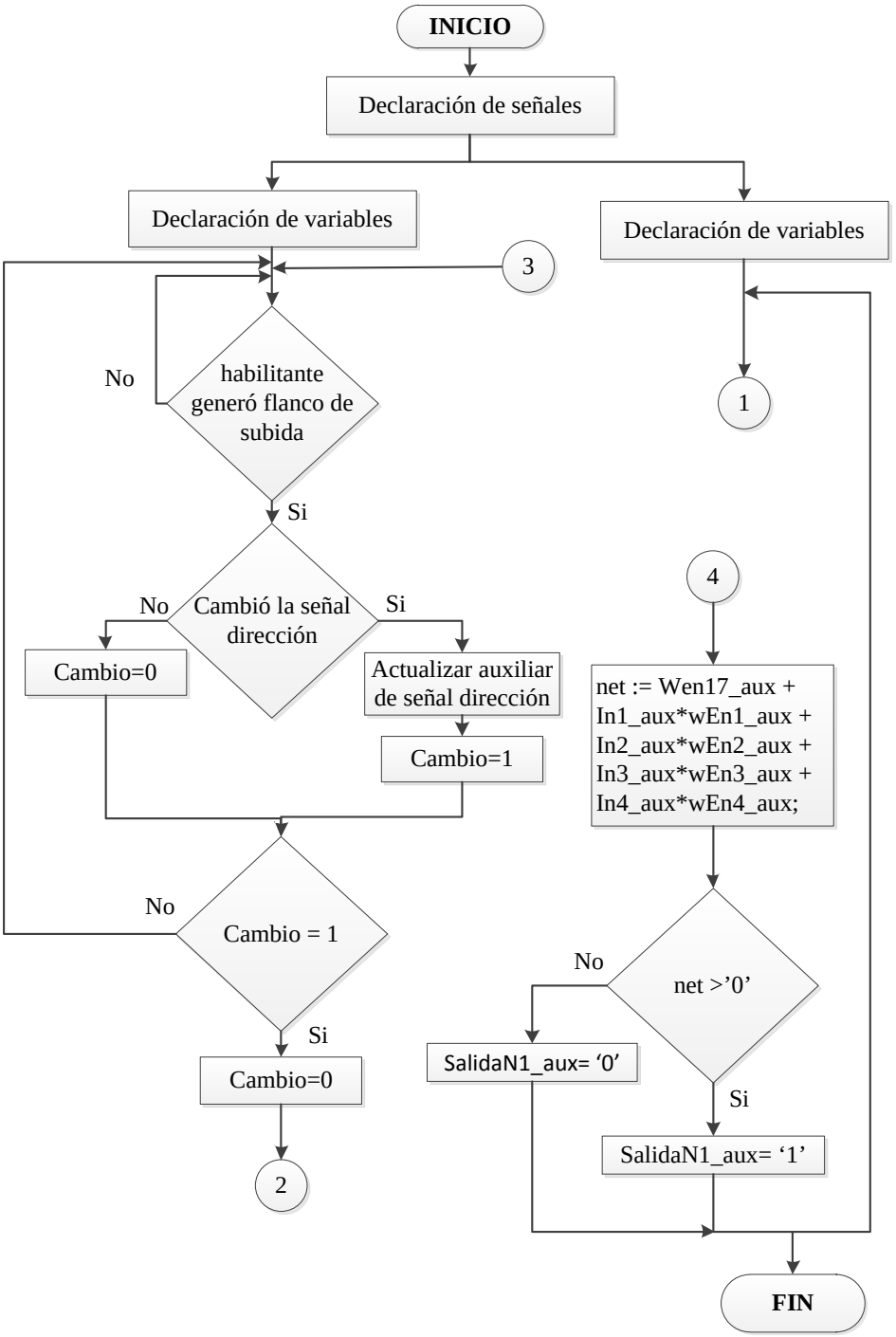


Figura 34. Diagrama de Flujo 1 de 4 del funcionamiento interno de cada neurona.

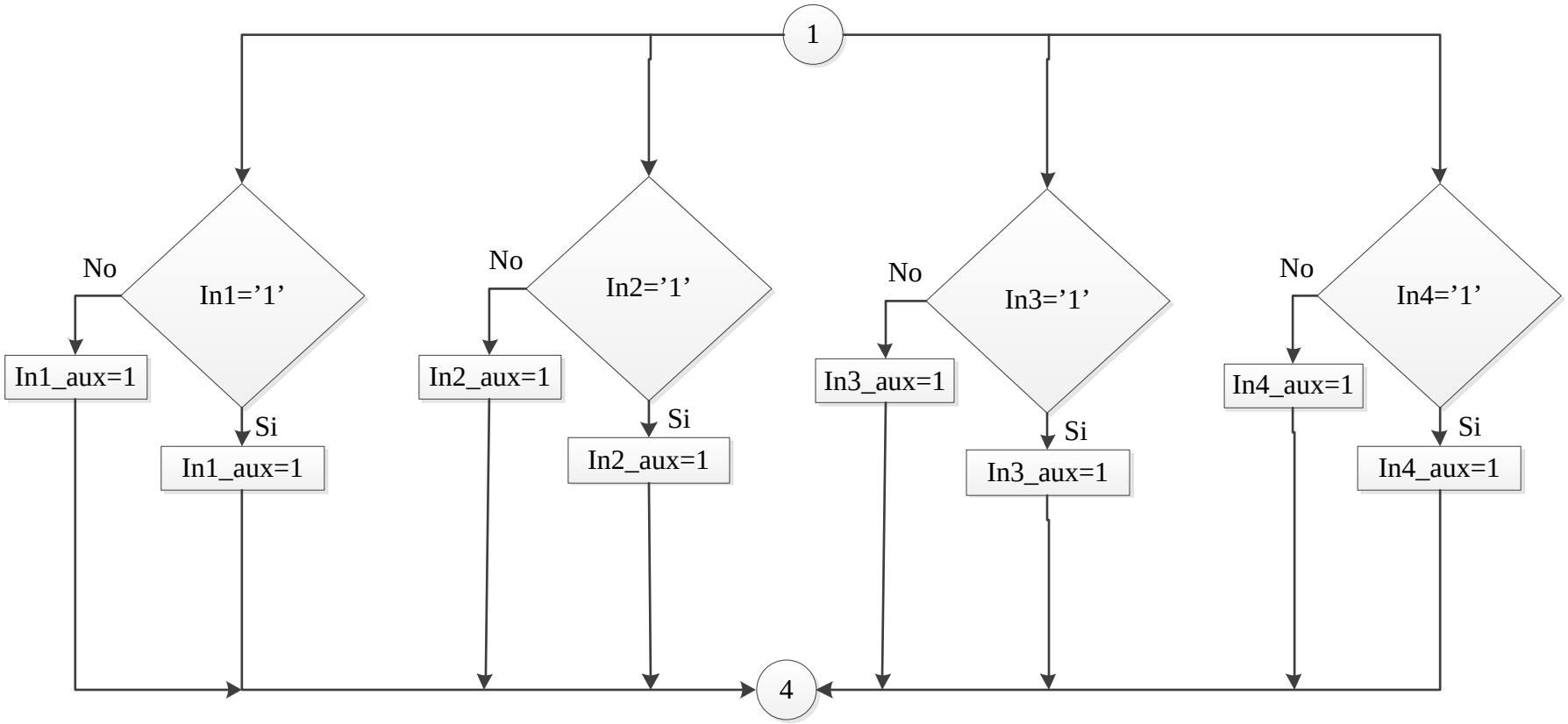


Figura 35. Diagrama de flujo 2 de 4 del funcionamiento interno de cada neurona.

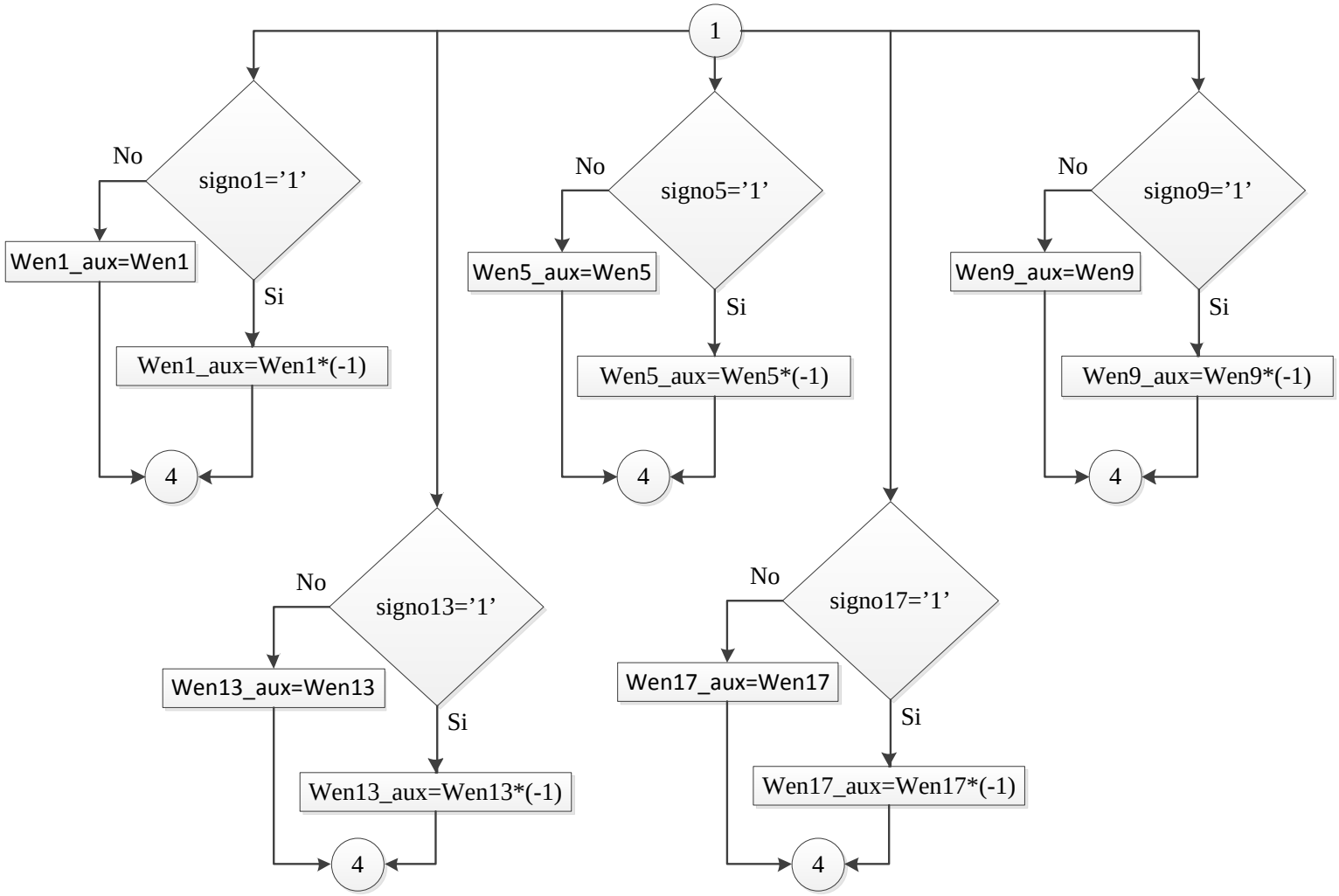


Figura 36. Diagrama de flujo 3 de 4 del funcionamiento interno de cada neurona.

Módulo Neurona 1

A la neurona 1 le corresponde identificar los cinco pesos que se encuentran en las direcciones 1, 5, 9, 13 y 17; estas direcciones son manejadas en números binarios ya que el intercambio de información entre módulos se realiza mediante señales eléctricas.

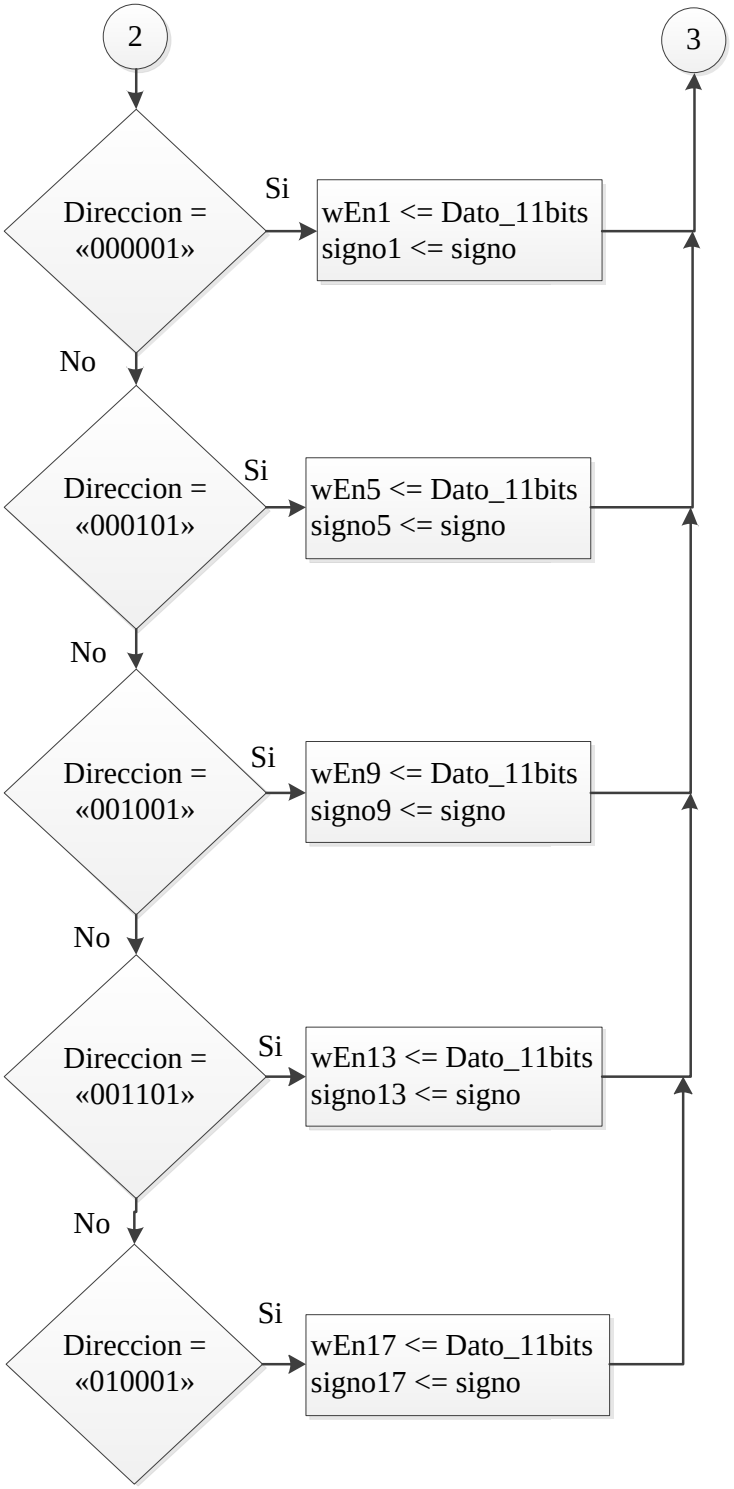


Figura 37. Diagrama de Flujo 4 de 4 del funcionamiento interno de la neurona 1.

La programación requerida para la **Neurona1** es la siguiente:

```
--------DECLARACIÓN DE LIBRERÍAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
------FIN DECLARACIÓN DE LIBRERÍAS------

------SEÑALES I/O------
---Se realiza la declaración de señales
---de entrada y salida del módulo.
entity Neurona1 is
    Port ( Dato_11bits : in  STD_LOGIC_VECTOR (10 downto 0);
           signo : in  STD_LOGIC;
           Direccion : in  STD_LOGIC_VECTOR (5 downto 0);
           In1 : in  STD_LOGIC;
           In2 : in  STD_LOGIC;
           In3 : in  STD_LOGIC;
           In4 : in  STD_LOGIC;
           habilitante : IN std_logic;
           SalidaN1 : out  STD_LOGIC);
end Neurona1;
----FIN SEÑALES I/O----

------------------------ARQUITECTURA-------------------------
---Se describe y se modela el hardware mediante código,
---describiendo así el flujo de datos y comportamiento en este módulo.
---Inicio del cuerpo de la arquitectura.
architecture Behavioral of Neurona1 is

----------ARREGLOS-----------
---Se declaran variables auxiliares y arreglos de memoria
---que permiten realizar cálculos y almacenar la información
---correspondiente a los pesos y signos.
signal wEn1 : std_logic_vector(10 downto 0);
signal wEn5 : std_logic_vector(10 downto 0);
signal wEn9 : std_logic_vector(10 downto 0);
signal wEn13 : std_logic_vector(10 downto 0);
signal wEn17 : std_logic_vector(10 downto 0);
signal signo1 : std_logic;
signal signo5 : std_logic;
signal signo9 : std_logic;
signal signo13 : std_logic;
signal signo17 : std_logic;
signal DirCambio : std_logic_vector(5 downto 0) := "000000";
signal SalidaN1_aux : std_logic;
----------FIN ARREGLOS-----------
begin ---Inicio del cuerpo de la arquitectura

-------PROCESS DE ENTRENAMIENTO-----
---Mediante este process se almacenan los valores de los
---pesos en la neurona con sus correspondientes signos.
process (habilitante)
 variable cambio: std_logic:='0'; ---Variable auxiliar
begin ---Inicio del cuerpo del process
   if habilitante = '1' and habilitante'event then

----------DETECCIÓN DE CAMBIO-------------
---Se almacena la dirección actual en un vector, y se compara
---con la dirección almacenada en un flanco de la señal habilitante
---anterior a la actual; si ambas direcciones son iguales significa que
---no se produjo ningún cambio, de lo contrario se almacena en una variable
---discreta la existencia de un cambio en la dirección.
    if Direccion = DirCambio then
    cambio := '0';
    else
    DirCambio <= Direccion;
    cambio := '1';
    end if;
---------FIN DETECCIÓN DE CAMBIO----------
```

```
-------CAMBIO DE DIRECCIÓN-------
---Se accede a la variable discreta cambio que contiene
---la información referente a la existencia o no de un cambio
---en la dirección.
    if cambio = '1' then
    cambio := '0';

-------------MEMORIA-----------------
---En caso de haber un cambio de Dirección se almacena la información
---en arreglos de memoria, dicha información corresponde a los pesos y signos
---de la neurona.
    if Direccion = "000001" then
    wEn1 <= Dato_11bits;
    signo1 <= signo;
    elsif Direccion = "000101" then
    wEn5 <= Dato_11bits;
    signo5 <= signo;
    elsif Direccion = "001001" then
    wEn9 <= Dato_11bits;
    signo9 <= signo;
    elsif Direccion = "001101" then
    wEn13 <= Dato_11bits;
    signo13 <= signo;
    elsif Direccion = "010001" then
    wEn17 <= Dato_11bits;
    signo17 <= signo;
    end if;
-----------FIN MEMORIA-----------
    end if;
-------FIN CAMBIO DE DIRECCIÓN-------
    end if;
end process;
-------FIN PROCESS DE ENTRENAMIENTO-----

-------PROCESS DE PUESTA EN MARCHA-----
---Mediante este process se pone en marcha el funcionamiento de la
---neurona realizando los cálculos para la función net, y su función
---de activación correspondiente para digitalizar su salida.
process (In1, In2, In3, In4)
variable In1_aux, In2_aux, In3_aux, In4_aux, net : integer;
variable wEn1_aux, wEn5_aux, wEn9_aux, wEn13_aux, wEn17_aux : integer;
begin

-------IMPORTACIÓN SEÑALES DE ENTRADA---------
---Debido a que las señales de entrada son señales eléctricas de
---tipo digital presentan una característica y es que no pueden ser
---manejadas como enteros. Para realizar las operaciones necesarias
---se requiere almacenar los valores de las entradas en variables
---de tipo entero.
    if (In1 = '1') then
    In1_aux := 1;
    else
    In1_aux := 0;
    end if;

    if (In2 = '1') then
    In2_aux := 1;
    else
    In2_aux := 0;
    end if;

    if (In3 = '1') then
    In3_aux := 1;
    else
    In3_aux := 0;
    end if;

    if (In4 = '1') then
    In4_aux := 1;
    else
    In4_aux := 0;
    end if;
-------FIN IMPORTACIÓN SEÑALES DE ENTRADA---------
```

```
----ASOCIACIÓN SIGNOS-PESOS--------
---Debido a que los vectores de memoria contienen los pesos y los signos en
---señales diferentes, se requiere asociarle a cada vector de memoria
---el signo que le corresponde para tratarlo como un numero entero.
    if (signo1 = '1') then
    Wen1_aux :=  (-1)*conv_integer(Wen1);
    else
    Wen1_aux :=  conv_integer(Wen1);
    end if;

    if (signo5 = '1') then
    Wen5_aux :=  (-1)*conv_integer(Wen5);
    else
    Wen5_aux :=  conv_integer(Wen5);
    end if;

    if (signo9 = '1') then
    Wen9_aux :=  (-1)*conv_integer(Wen9);
    else
    Wen9_aux :=  conv_integer(Wen9);
    end if;

    if (signo13 = '1') then
    Wen13_aux :=  (-1)*conv_integer(Wen13);
    else
    Wen13_aux :=  conv_integer(Wen13);
    end if;

    if (signo17 = '1') then
    Wen17_aux :=  (-1)*conv_integer(Wen17);
    else
    Wen17_aux :=  conv_integer(Wen17);
    end if;
----FIN ASOCIACIÓN SIGNOS-PESOS--------

-----FUNCION NET------
---Se realiza el cálculo de la función net mediante las
---variables asociadas a los pesos y a las entradas de
---esta neurona.
    net := Wen17_aux + In1_aux*wEn1_aux + In2_aux*wEn5_aux + In3_aux*wEn9_aux +
    In4_aux*wEn13_aux;
---FIN FUNCION NET----

---FUNCION DE ACTIVACIÓN----
---Para el cálculo de la función net se trataron los valores como
---enteros y se requiere llevar la respuesta a un valor digital
---para que sea transferido como una señal eléctrica a la siguiente neurona.
---Esta función de activación corresponde a una función escalón.
    if (net > 0) then
    SalidaN1_aux <= '1';
    else
    SalidaN1_aux <= '0';
    end if;
---FIN FUNCION DE ACTIVACIÓN----
end process;
-------FIN PROCESS DE PUESTA EN MARCHA-----
SalidaN1 <= SalidaN1_aux; ---Exportación de la respuesta SalidaN1_aux
end Behavioral;
------------------------FIN ARQUITECTURA----------------------
```

Módulo Neurona 2

A la neurona 2 le corresponde identificar los cinco pesos que se encuentran en las direcciones 2, 6, 10, 14 y 18; estas direcciones son manejadas en números binarios ya que el intercambio de información entre módulos se realiza mediante señales eléctricas.

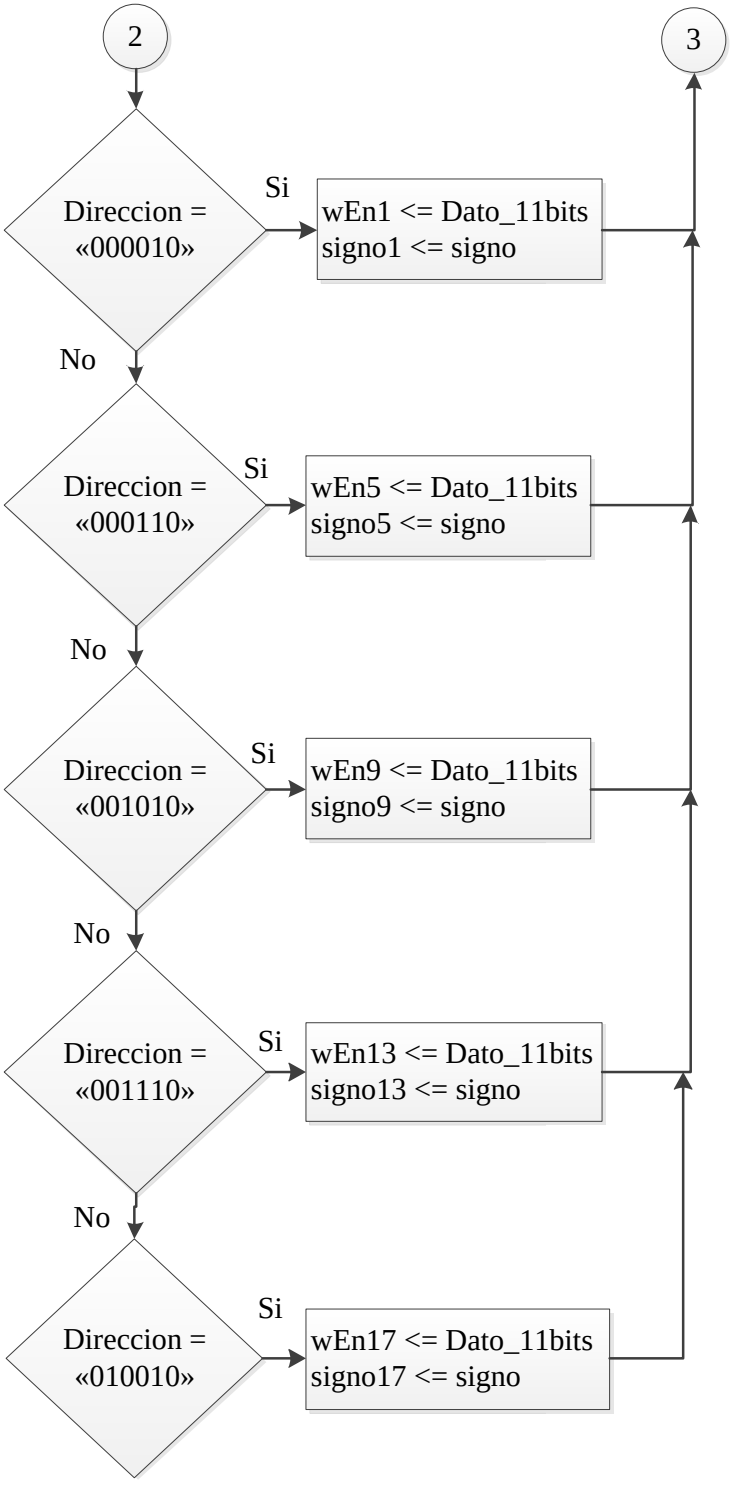


Figura 38. Diagrama de Flujo 4 de 4 del funcionamiento interno de la neurona 2.

La programación requerida para la **Neurona2** es la siguiente:

```
--------DECLARACIÓN DE LIBRERÍAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
------FIN DECLARACIÓN DE LIBRERÍAS------

------SEÑALES I/O------
---Se realiza la declaración de señales
---de entrada y salida del módulo.
entity Neurona2 is
    Port ( Dato_11bits : in  STD_LOGIC_VECTOR (10 downto 0);
           signo : in  STD_LOGIC;
           Direccion : in  STD_LOGIC_VECTOR (5 downto 0);
           In1 : in  STD_LOGIC;
           In2 : in  STD_LOGIC;
           In3 : in  STD_LOGIC;
           In4 : in  STD_LOGIC;
           habilitante : IN std_logic;
           SalidaN2 : out  STD_LOGIC);
end Neurona2;
----FIN SEÑALES I/O----

------------------------ARQUITECTURA-------------------------
---Se describe y se modela el hardware mediante código,
---describiendo así el flujo de datos y comportamiento en este módulo.
architecture Behavioral of Neurona2 is

------------ARREGLOS------------
---Se declaran variables auxiliares y arreglos de memoria
---que permiten realizar cálculos y almacenar la información
---correspondiente a los pesos y signos.
signal wEn1 : std_logic_vector(10 downto 0);
signal wEn2 : std_logic_vector(10 downto 0);
signal wEn3 : std_logic_vector(10 downto 0);
signal wEn4 : std_logic_vector(10 downto 0);
signal wEn17 : std_logic_vector(10 downto 0);
signal signo1 : std_logic;
signal signo2 : std_logic;
signal signo3 : std_logic;
signal signo4 : std_logic;
signal signo17 : std_logic;
signal DirCambio : std_logic_vector(5 downto 0) := "000000";
signal SalidaN1_aux : std_logic;
----------FIN ARREGLOS-----------
begin ---Inicio del cuerpo de la arquitectura.

-------PROCESS DE ENTRENAMIENTO-----
---Mediante este process se almacenan los valores de los
---pesos en la neurona con sus correspondientes signos.
process (habilitante)
 variable cambio: std_logic:='0'; ---Variable auxiliar
begin ---Inicio del cuerpo del process

----------DETECCIÓN DE CAMBIO-------------
---Se almacena la dirección actual en un vector, y se compara
---con la dirección almacenada en un tiempo previo al actual,
---si ambas direcciones son iguales significa que no se
---produjo ningún cambio, de lo contrario se almacena en una variable
---discreta la existencia de un cambio en la dirección.
   if habilitante = '1' and habilitante'event then
    if Direccion = DirCambio then
    cambio := '0';
    else
    DirCambio <= Direccion;
    cambio := '1';
    end if;
---------FIN DETECCIÓN DE CAMBIO----------
```

```
-------CAMBIO DE DIRECCIÓN-------
---Se accede a la variable discreta cambio que contiene
---la información referente a la existencia o no de un cambio
---en la dirección.
    if cambio = '1' then
    cambio := '0';

-------------MEMORIA-----------------
---En caso de haber un cambio de Dirección se almacena la información
---en arreglos de memoria, dicha información corresponde a los pesos y signos
---de la neurona.
    if Direccion = "000010" then
    wEn1 <= Dato_11bits;
    signo1 <= signo;
    elsif Direccion = "000110" then
    wEn2 <= Dato_11bits;
    signo2 <= signo;
    elsif Direccion = "001010" then
    wEn3 <= Dato_11bits;
    signo3 <= signo;
    elsif Direccion = "001110" then
    wEn4 <= Dato_11bits;
    signo4 <= signo;
    elsif Direccion = "010010" then
    wEn17 <= Dato_11bits;
    signo17 <= signo;
    end if;
-----------FIN MEMORIA-----------
    end if;
-------FIN CAMBIO DE DIRECCIÓN-------
    end if;
end process;
-------FIN PROCESS DE ENTRENAMIENTO-----

-------PROCESS DE PUESTA EN MARCHA-----
---Mediante este process se pone en marcha el funcionamiento de la
---neurona, realizando los cálculos para la función net, y su función
---de activación correspondiente.
process (In1, In2, In3, In4) -- ver posibilidad de poner un clk
variable In1_aux, In2_aux, In3_aux, In4_aux, net : integer;
variable wEn1_aux, wEn2_aux, wEn3_aux, wEn4_aux, wEn17_aux : integer;
begin

-------IMPORTACIÓN SEÑALES DE ENTRADA---------
---Debido a que las señales de entrada son señales eléctricas de
---tipo digital presentan una característica, no pueden ser
---manejadas como enteros. Para realizar las operaciones necesarias
---se requiere almacenar los valores de las entradas en variables
---de tipo entero.
   if (In1 = '1') then
    In1_aux := 1;
    else
    In1_aux := 0;
    end if;

    if (In2 = '1') then
    In2_aux := 1;
    else
    In2_aux := 0;
    end if;

    if (In3 = '1') then
    In3_aux := 1;
    else
    In3_aux := 0;
    end if;

    if (In4 = '1') then
    In4_aux := 1;
    else
    In4_aux := 0;
    end if;
-------FIN IMPORTACIÓN SEÑALES DE ENTRADA---------
```

```
----ASOCIACIÓN SIGNOS-PESOS--------
---Debido a que los vectores de memoria contienen los pesos y los signos en
---señales diferentes, se requiere asociarle a cada vector de memoria
---el signo que le corresponde para tratarlo como un numero entero.
    if (signo1 = '1') then
    Wen1_aux :=  (-1)*conv_integer(Wen1);
    else
    Wen1_aux :=  conv_integer(Wen1);
    end if;

    if (signo2 = '1') then
    Wen2_aux :=  (-1)*conv_integer(Wen2);
    else
    Wen2_aux :=  conv_integer(Wen2);
    end if;

    if (signo3 = '1') then
    Wen3_aux :=  (-1)*conv_integer(Wen3);
    else
    Wen3_aux :=  conv_integer(Wen3);
    end if;

    if (signo4 = '1') then
    Wen4_aux :=  (-1)*conv_integer(Wen4);
    else
    Wen4_aux :=  conv_integer(Wen4);
    end if;

    if (signo17 = '1') then
    Wen17_aux :=  (-1)*conv_integer(Wen17);
    else
    Wen17_aux :=  conv_integer(Wen17);
    end if;
----FIN ASOCIACIÓN SIGNOS-PESOS--------

-----FUNCION NET------
---Se realiza el cálculo de la función net mediante las variables
---asociadas a los pesos y a las entradas de esta neurona.
    net := Wen17_aux + In1_aux*wEn1_aux + In2_aux*wEn2_aux + In3_aux*wEn3_aux +
In4_aux*wEn4_aux;
---FIN FUNCION NET----

---FUNCION DE ACTIVACIÓN----
---Para el cálculo de la función net se trataron los valores como
---enteros y se requiere llevar la respuesta a un valor digital
---para que sea transferido como una señal eléctrica a la siguiente neurona.
---Esta función de activación corresponde a una función escalón.
    if (net > 0) then
    SalidaN1_aux <= '1';
    else
    SalidaN1_aux <= '0';
    end if;
---FIN FUNCION DE ACTIVACIÓN----
end process;

-------FIN PROCESS DE PUESTA EN MARCHA-----
SalidaN2 <= SalidaN1_aux;---Exportación de la respuesta SalidaN1_aux
end Behavioral;
-----------------------FIN ARQUITECTURA----------------------
```

Módulo Neurona 3

A la neurona 3 le corresponde identificar los cinco pesos que se encuentran en las direcciones 3, 7, 11, 15 y 19; estas direcciones son manejadas en números binarios ya que el intercambio de información entre módulos se realiza mediante señales eléctricas.

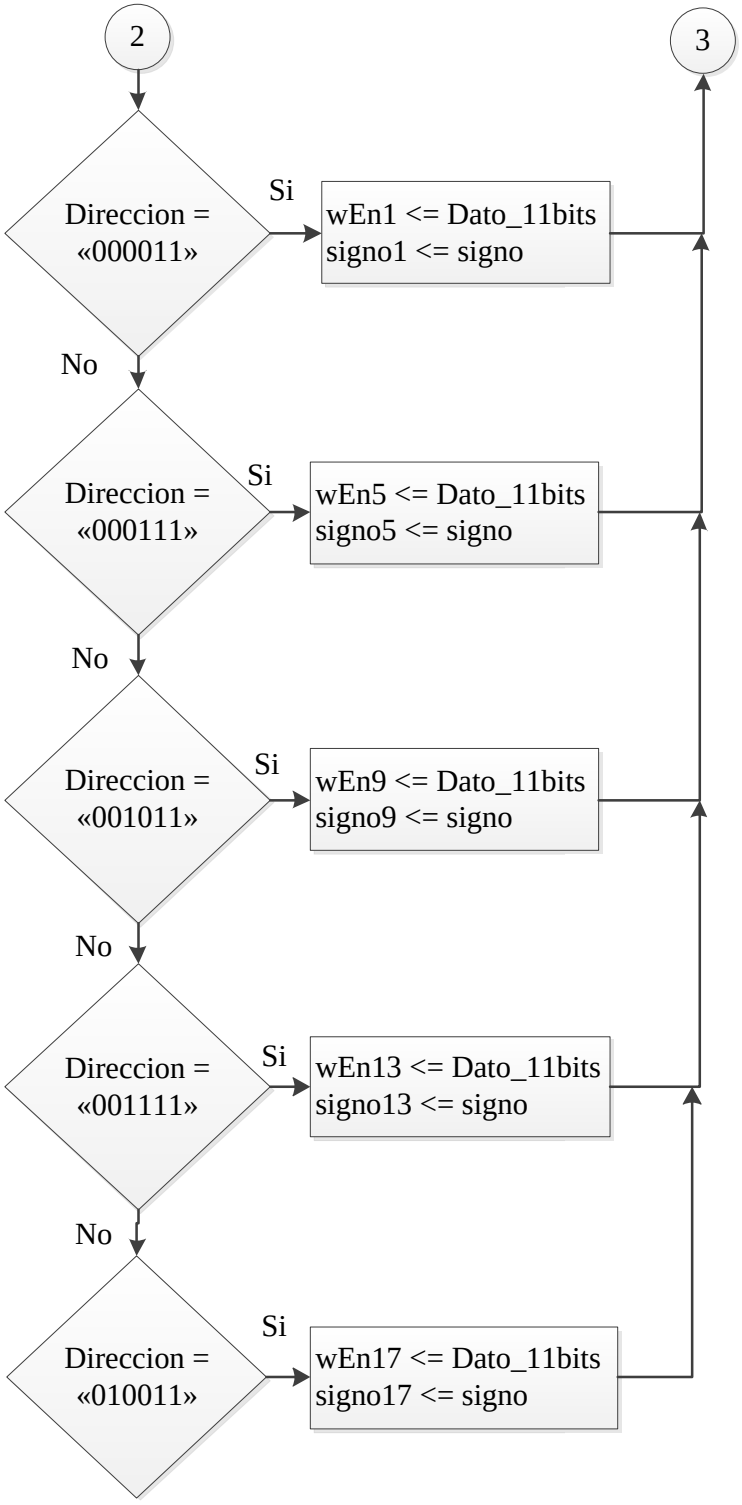


Figura 39. Diagrama de Flujo 4 de 4 del funcionamiento interno de la neurona 3.

La programación requerida para la **Neurona3** es la siguiente:

```
--------DECLARACIÓN DE LIBRERÍAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
------FIN DECLARACIÓN DE LIBRERÍAS------

------SEÑALES I/O------
---Se realiza la declaración de señales
---de entrada y salida del módulo.
entity Neurona3 is
    Port ( Dato_11bits : in  STD_LOGIC_VECTOR (10 downto 0);
           signo : in  STD_LOGIC;
           Direccion : in  STD_LOGIC_VECTOR (5 downto 0);
           In1 : in  STD_LOGIC;
           In2 : in  STD_LOGIC;
           In3 : in  STD_LOGIC;
           In4 : in  STD_LOGIC;
              habilitante : IN std_logic;
              SalidaN3 : out  STD_LOGIC);
end Neurona3;
----FIN SEÑALES I/O----

------------------------ARQUITECTURA-------------------------
---Se describe y se modela el hardware mediante código,
---describiendo así el flujo de datos y comportamiento en este módulo.
architecture Behavioral of Neurona3 is

------------ARREGLOS------------
---Se declaran variables auxiliares y arreglos de memoria
---que permiten realizar cálculos y almacenar la información
---correspondiente a los pesos y signos.
signal wEn1 : std_logic_vector(10 downto 0);
signal wEn2 : std_logic_vector(10 downto 0);
signal wEn3 : std_logic_vector(10 downto 0);
signal wEn4 : std_logic_vector(10 downto 0);
signal wEn17 : std_logic_vector(10 downto 0);
signal signo1 : std_logic;
signal signo2 : std_logic;
signal signo3 : std_logic;
signal signo4 : std_logic;
signal signo17 : std_logic;
signal DirCambio : std_logic_vector(5 downto 0) := "000000";
signal SalidaN1_aux : std_logic;
----------FIN ARREGLOS----------
begin---Inicio del cuerpo de la arquitectura

-------PROCESS DE ENTRENAMIENTO-----
---Mediante este process se almacenan los valores de los
---pesos en la neurona con sus correspondientes signos.
process (habilitante)
 variable cambio: std_logic:='0';---Variable auxiliar
begin---Inicio del cuerpo del process
   if habilitante = '1' and habilitante'event then

----------DETECCIÓN DE CAMBIO-------------
---Se almacena la dirección actual en un vector, y se compara
---con la dirección almacenada en un tiempo previo al actual,
---si ambas direcciones son iguales significa que no se
---produjo ningún cambio, de lo contrario se almacena en una variable
---discreta la existencia de un cambio en la dirección.
    if Direccion = DirCambio then
    cambio := '0';
    else
    DirCambio <= Direccion;
    cambio := '1';
    end if;
---------FIN DETECCIÓN DE CAMBIO----------
```

```
-------CAMBIO DE DIRECCIÓN-------
---Se accede a la variable discreta cambio que contiene
---la información referente a la existencia o no de un cambio
---en la dirección.
    if cambio = '1' then
    cambio := '0';

-------------MEMORIA-----------------
---En caso de haber un cambio de Dirección se almacena la información
---en arreglos de memoria, dicha información corresponde a los pesos y signos
---de la neurona.
    if Direccion = "000011" then
    wEn1 <= Dato_11bits;
    signo1 <= signo;
    elsif Direccion = "000111" then
    wEn2 <= Dato_11bits;
    signo2 <= signo;
    elsif Direccion = "001011" then
    wEn3 <= Dato_11bits;
    signo3 <= signo;
    elsif Direccion = "001111" then
    wEn4 <= Dato_11bits;
    signo4 <= signo;
    elsif Direccion = "010011" then
    wEn17 <= Dato_11bits;
    signo17 <= signo;
    end if;
-----------FIN MEMORIA-----------
    end if;
-------FIN CAMBIO DE DIRECCIÓN-------
    end if;
end process;
-------FIN PROCESS DE ENTRENAMIENTO-----

-------PROCESS DE PUESTA EN MARCHA-----
---Mediante este process se pone en marcha el funcionamiento de la
---neurona, realizando los cálculos para la función net, y su función
---de activación correspondiente.
process (In1, In2, In3, In4) -- ver posibilidad de poner un clk
variable In1_aux, In2_aux, In3_aux, In4_aux, net : integer;
variable wEn1_aux, wEn2_aux, wEn3_aux, wEn4_aux, wEn17_aux : integer;
begin

-------IMPORTACIÓN SEÑALES DE ENTRADA---------
---Debido a que las señales de entrada son señales eléctricas de
---tipo digital presentan una característica, no pueden ser
---manejadas como enteros. Para realizar las operaciones necesarias
---se requiere almacenar los valores de las entradas en variables
---de tipo entero.
   if (In1 = '1') then
    In1_aux := 1;
    else
    In1_aux := 0;
    end if;

    if (In2 = '1') then
    In2_aux := 1;
    else
    In2_aux := 0;
    end if;

    if (In3 = '1') then
    In3_aux := 1;
    else
    In3_aux := 0;
    end if;

    if (In4 = '1') then
    In4_aux := 1;
    else
    In4_aux := 0;
    end if;
-------FIN IMPORTACIÓN SEÑALES DE ENTRADA---------
```

```
----ASOCIACIÓN SIGNOS-PESOS--------
---Debido a que los vectores de memoria contienen los pesos y los signos en
---señales diferentes, se requiere asociarle a cada vector de memoria
---el signo que le corresponde para tratarlo como un numero entero.
    if (signo1 = '1') then
    Wen1_aux :=  (-1)*conv_integer(Wen1);
    else
    Wen1_aux :=  conv_integer(Wen1);
    end if;

    if (signo2 = '1') then
    Wen2_aux :=  (-1)*conv_integer(Wen2);
    else
    Wen2_aux :=  conv_integer(Wen2);
    end if;

    if (signo3 = '1') then
    Wen3_aux :=  (-1)*conv_integer(Wen3);
    else
    Wen3_aux :=  conv_integer(Wen3);
    end if;

    if (signo4 = '1') then
    Wen4_aux :=  (-1)*conv_integer(Wen4);
    else
    Wen4_aux :=  conv_integer(Wen4);
    end if;

    if (signo17 = '1') then
    Wen17_aux :=  (-1)*conv_integer(Wen17);
    else
    Wen17_aux :=  conv_integer(Wen17);
    end if;
----FIN ASOCIACIÓN SIGNOS-PESOS--------

-----FUNCION NET------
---Se realiza el cálculo de la función net mediante las variables
---asociadas a los pesos y a las entradas de esta neurona.
    net := Wen17_aux + In1_aux*wEn1_aux + In2_aux*wEn2_aux + In3_aux*wEn3_aux +
In4_aux*wEn4_aux;
---FIN FUNCION NET----

---FUNCION DE ACTIVACIÓN----
---Para el cálculo de la función net se trataron los valores como
---enteros y se requiere llevar la respuesta a un valor digital
---para que sea transferido como una señal eléctrica a la siguiente neurona.
---Esta función de activación corresponde a una función escalón.
    if (net > 0) then
    SalidaN1_aux <= '1';
    else
    SalidaN1_aux <= '0';
    end if;
---FIN FUNCION DE ACTIVACIÓN----
end process;
-------FIN PROCESS DE PUESTA EN MARCHA-----
SalidaN3 <= SalidaN1_aux; ---Exportación de la respuesta SalidaN1_aux
end Behavioral;
-----------------------FIN ARQUITECTURA----------------------
```

Módulo Neurona 4

A la neurona 4 le corresponde identificar los cinco pesos que se encuentran en las direcciones 4, 8, 12, 16 y 20; estas direcciones son manejadas en números binarios ya que el intercambio de información entre módulos se realiza mediante señales eléctricas.

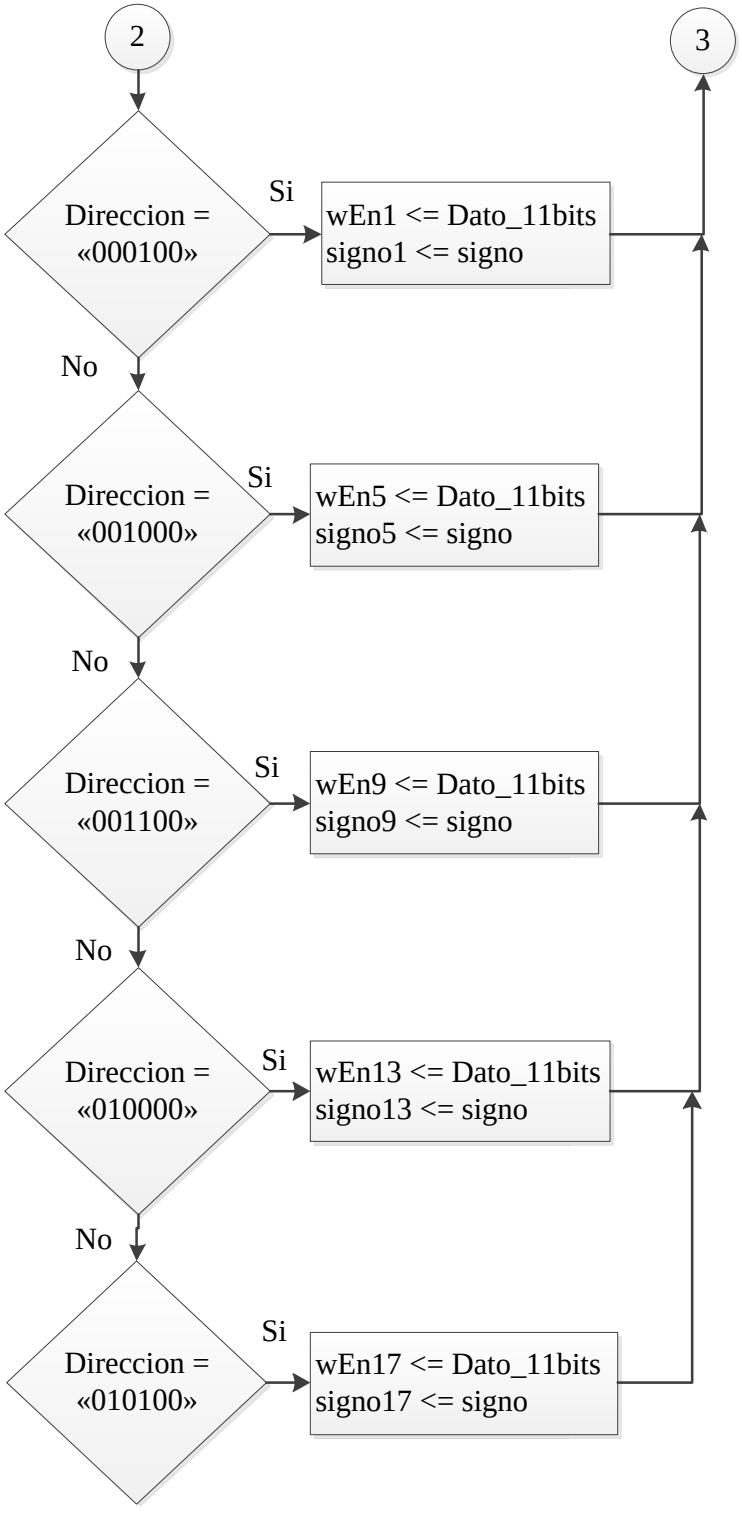


Figura 40. Diagrama de Flujo 4 de 4 del funcionamiento interno de la neurona 4.

La programación requerida para la **Neurona4** es la siguiente:

```
--------DECLARACIÓN DE LIBRERÍAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
------FIN DECLARACIÓN DE LIBRERÍAS------

------SEÑALES I/O------
---Se realiza la declaración de señales
---de entrada y salida del módulo.
entity Neurona4 is
    Port ( Dato_11bits : in  STD_LOGIC_VECTOR (10 downto 0);
           signo : in  STD_LOGIC;
           Direccion : in  STD_LOGIC_VECTOR (5 downto 0);
           In1 : in  STD_LOGIC;
           In2 : in  STD_LOGIC;
           In3 : in  STD_LOGIC;
           In4 : in  STD_LOGIC;
           habilitante : IN std_logic;
           SalidaN4 : out  STD_LOGIC);
end Neurona4;
----FIN SEÑALES I/O----

------------------------ARQUITECTURA-------------------------
---Se describe y se modela el hardware mediante código,
---describiendo así el flujo de datos y comportamiento en este módulo.
architecture Behavioral of Neurona4 is

------------ARREGLOS------------
---Se declaran variables auxiliares y arreglos de memoria
---que permiten realizar cálculos y almacenar la información
---correspondiente a los pesos y signos.
signal wEn1 : std_logic_vector(10 downto 0);
signal wEn2 : std_logic_vector(10 downto 0);
signal wEn3 : std_logic_vector(10 downto 0);
signal wEn4 : std_logic_vector(10 downto 0);
signal wEn17 : std_logic_vector(10 downto 0);
signal signo1 : std_logic;
signal signo2 : std_logic;
signal signo3 : std_logic;
signal signo4 : std_logic;
signal signo17 : std_logic;
signal DirCambio : std_logic_vector(5 downto 0) := "000000";
signal SalidaN1_aux : std_logic;
----------FIN ARREGLOS----------
begin---Inicio del cuerpo de la arquitectura

-------PROCESS DE ENTRENAMIENTO-----
---Mediante este process se almacenan los valores de los
---pesos en la neurona con sus correspondientes signos.
process (habilitante)
 variable cambio: std_logic:='0';---Variable auxiliar
begin---Inicio del cuerpo del process
   if habilitante = '1' and habilitante'event then

----------DETECCIÓN DE CAMBIO-------------
---Se almacena la dirección actual en un vector, y se compara
---con la dirección almacenada en un tiempo previo al actual,
---si ambas direcciones son iguales significa que no se
---produjo ningún cambio, de lo contrario se almacena en una variable
---discreta la existencia de un cambio en la dirección.
    if Direccion = DirCambio then
    cambio := '0';
    else
    DirCambio <= Direccion;
    cambio := '1';
    end if;
---------FIN DETECCIÓN DE CAMBIO----------
```

```
-------CAMBIO DE DIRECCIÓN-------
---Se accede a la variable discreta cambio que contiene
---la información referente a la existencia o no de un cambio
---en la dirección.
    if cambio = '1' then
    cambio := '0';

-------------MEMORIA-----------------
---En caso de haber un cambio de Dirección se almacena la información
---en arreglos de memoria, dicha información corresponde a los pesos y signos
---de la neurona.
    if Direccion = "000100" then
    wEn1 <= Dato_11bits;
    signo1 <= signo;
    elsif Direccion = "001000" then
    wEn2 <= Dato_11bits;
    signo2 <= signo;
    elsif Direccion = "001100" then
    wEn3 <= Dato_11bits;
    signo3 <= signo;
    elsif Direccion = "010000" then
    wEn4 <= Dato_11bits;
    signo4 <= signo;
    elsif Direccion = "010100" then
    wEn17 <= Dato_11bits;
    signo17 <= signo;
    end if;
-----------FIN MEMORIA-----------
    end if;
-------FIN CAMBIO DE DIRECCIÓN-------
    end if;
end process;
-------FIN PROCESS DE ENTRENAMIENTO-----

-------PROCESS DE PUESTA EN MARCHA-----
---Mediante este process se pone en marcha el funcionamiento de la
---neurona, realizando los cálculos para la función net, y su función
---de activación correspondiente.
process (In1, In2, In3, In4) -- ver posibilidad de poner un clk
variable In1_aux, In2_aux, In3_aux, In4_aux, net : integer;
variable wEn1_aux, wEn2_aux, wEn3_aux, wEn4_aux, wEn17_aux : integer;
begin

-------IMPORTACIÓN SEÑALES DE ENTRADA---------
---Debido a que las señales de entrada son señales eléctricas de
---tipo digital presentan una característica, no pueden ser
---manejadas como enteros. Para realizar las operaciones necesarias
---se requiere almacenar los valores de las entradas en variables
---de tipo entero.
   if (In1 = '1') then
    In1_aux := 1;
    else
    In1_aux := 0;
    end if;

    if (In2 = '1') then
    In2_aux := 1;
    else
    In2_aux := 0;
    end if;

    if (In3 = '1') then
    In3_aux := 1;
    else
    In3_aux := 0;
    end if;

    if (In4 = '1') then
    In4_aux := 1;
    else
    In4_aux := 0;
    end if;
-------FIN IMPORTACIÓN SEÑALES DE ENTRADA---------
```

```
----ASOCIACIÓN SIGNOS-PESOS--------
---Debido a que los vectores de memoria contienen los pesos y los signos en
---señales diferentes, se requiere asociarle a cada vector de memoria
---el signo que le corresponde para tratarlo como un numero entero.
    if (signo1 = '1') then
    Wen1_aux :=  (-1)*conv_integer(Wen1);
    else
    Wen1_aux :=  conv_integer(Wen1);
    end if;

    if (signo2 = '1') then
    Wen2_aux :=  (-1)*conv_integer(Wen2);
    else
    Wen2_aux :=  conv_integer(Wen2);
    end if;

    if (signo3 = '1') then
    Wen3_aux :=  (-1)*conv_integer(Wen3);
    else
    Wen3_aux :=  conv_integer(Wen3);
    end if;

    if (signo4 = '1') then
    Wen4_aux :=  (-1)*conv_integer(Wen4);
    else
    Wen4_aux :=  conv_integer(Wen4);
    end if;

    if (signo17 = '1') then
    Wen17_aux :=  (-1)*conv_integer(Wen17);
    else
    Wen17_aux :=  conv_integer(Wen17);
    end if;
----FIN ASOCIACIÓN SIGNOS-PESOS--------

-----FUNCION NET------
---Se realiza el cálculo de la función net mediante las variables
---asociadas a los pesos y a las entradas de esta neurona.
    net := Wen17_aux + In1_aux*wEn1_aux + In2_aux*wEn2_aux + In3_aux*wEn3_aux +
    In4_aux*wEn4_aux;
---FIN FUNCION NET----

---FUNCION DE ACTIVACIÓN----
---Para el cálculo de la función net se trataron los valores como
---enteros y se requiere llevar la respuesta a un valor digital
---para que sea transferido como una señal eléctrica a la siguiente neurona.
---Esta función de activación corresponde a una función escalón.
    if (net > 0) then
    SalidaN1_aux <= '1';
    else
    SalidaN1_aux <= '0';
    end if;
---FIN FUNCION DE ACTIVACIÓN----
end process;
-------FIN PROCESS DE PUESTA EN MARCHA-----
SalidaN4 <= SalidaN1_aux; ---Exportación de la respuesta SalidaN1_aux
end Behavioral;
-----------------------FIN ARQUITECTURA----------------------
```

Módulo Neurona 5

A la neurona 5 le corresponde identificar los cinco pesos que se encuentran en las direcciones 21, 24, 27, 30 y 33; estas direcciones son manejadas en números binarios ya que el intercambio de información entre módulos se realiza mediante señales eléctricas.

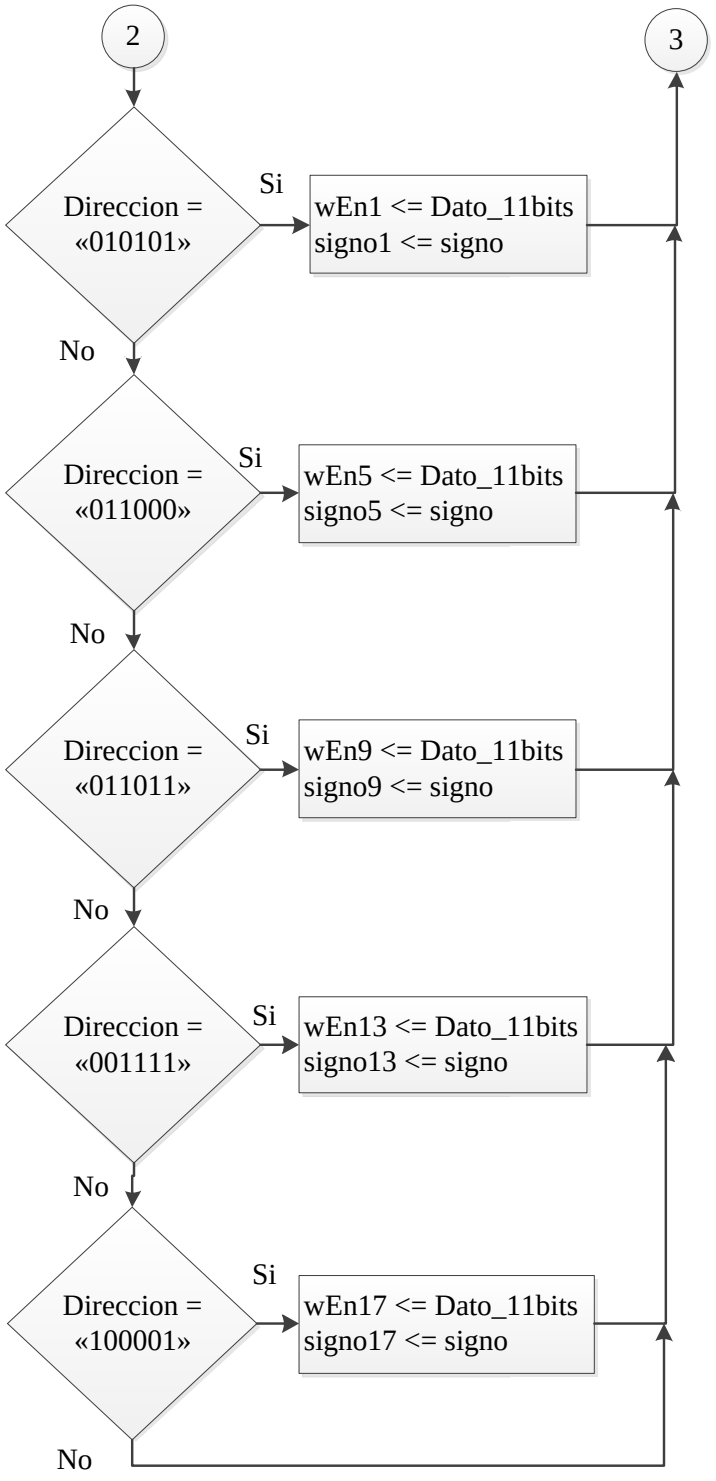


Figura 41. Diagrama de Flujo 4 de 4 del funcionamiento interno de la neurona 5.

La programación requerida para la **Neurona5** es la siguiente:

```
--------DECLARACIÓN DE LIBRERÍAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
------FIN DECLARACIÓN DE LIBRERÍAS------

------SEÑALES I/O------
---Se realiza la declaración de señales
---de entrada y salida del módulo.
entity Neurona5 is
    Port ( Dato_11bits : in  STD_LOGIC_VECTOR (10 downto 0);
           signo : in  STD_LOGIC;
           Direccion : in  STD_LOGIC_VECTOR (5 downto 0);
           In1 : in  STD_LOGIC;
           In2 : in  STD_LOGIC;
           In3 : in  STD_LOGIC;
           In4 : in  STD_LOGIC;
              habilitante : IN std_logic;
              SalidaN5 : out  STD_LOGIC);
end Neurona5;
----FIN SEÑALES I/O----

------------------------ARQUITECTURA-------------------------
---Se describe y se modela el hardware mediante código,
---describiendo así el flujo de datos y comportamiento en este módulo.
architecture Behavioral of Neurona5 is

------------ARREGLOS------------
---Se declaran variables auxiliares y arreglos de memoria
---que permiten realizar cálculos y almacenar la información
---correspondiente a los pesos y signos.
signal wEn1 : std_logic_vector(10 downto 0);
signal wEn2 : std_logic_vector(10 downto 0);
signal wEn3 : std_logic_vector(10 downto 0);
signal wEn4 : std_logic_vector(10 downto 0);
signal wEn17 : std_logic_vector(10 downto 0);
signal signo1 : std_logic;
signal signo2 : std_logic;
signal signo3 : std_logic;
signal signo4 : std_logic;
signal signo17 : std_logic;
signal DirCambio : std_logic_vector(5 downto 0) := "000000";
signal SalidaN1_aux : std_logic;
----------FIN ARREGLOS----------
begin---Inicio del cuerpo de la arquitectura

-------PROCESS DE ENTRENAMIENTO-----
---Mediante este process se almacenan los valores de los
---pesos en la neurona con sus correspondientes signos.
process (habilitante)
 variable cambio: std_logic:='0';---Variable auxiliar
begin---Inicio del cuerpo del process
   if habilitante = '1' and habilitante'event then

----------DETECCIÓN DE CAMBIO------------
---Se almacena la dirección actual en un vector, y se compara
---con la dirección almacenada en un tiempo previo al actual,
---si ambas direcciones son iguales significa que no se
---produjo ningún cambio, de lo contrario se almacena en una variable
---discreta la existencia de un cambio en la dirección.
    if Direccion = DirCambio then
    cambio := '0';

    else
    DirCambio <= Direccion;
    cambio := '1';
    end if;
---------FIN DETECCIÓN DE CAMBIO----------
```

```
-------CAMBIO DE DIRECCIÓN-------
---Se accede a la variable discreta cambio que contiene
---la información referente a la existencia o no de un cambio
---en la dirección.
    if cambio = '1' then
    cambio := '0';

-------------MEMORIA-----------------
---En caso de haber un cambio de Dirección se almacena la información
---en arreglos de memoria, dicha información corresponde a los pesos y signos
---de la neurona.
    if Direccion = "010101" then
    wEn1 <= Dato_11bits;
    signo1 <= signo;
    elsif Direccion = "011000" then
    wEn2 <= Dato_11bits;
    signo2 <= signo;
    elsif Direccion = "011011" then
    wEn3 <= Dato_11bits;
    signo3 <= signo;
    elsif Direccion = "011110" then
    wEn4 <= Dato_11bits;
    signo4 <= signo;
    elsif Direccion = "100001" then
    wEn17 <= Dato_11bits;
    signo17 <= signo;
    end if;
-----------FIN MEMORIA-----------
    end if;
-------FIN CAMBIO DE DIRECCIÓN-------
    end if;
end process;
-------FIN PROCESS DE ENTRENAMIENTO-----

-------PROCESS DE PUESTA EN MARCHA-----
---Mediante este process se pone en marcha el funcionamiento de la
---neurona, realizando los cálculos para la función net, y su función
---de activación correspondiente.
process (In1, In2, In3, In4) -- ver posibilidad de poner un clk
variable In1_aux, In2_aux, In3_aux, In4_aux, net : integer;
variable wEn1_aux, wEn2_aux, wEn3_aux, wEn4_aux, wEn17_aux : integer;
begin

-------IMPORTACIÓN SEÑALES DE ENTRADA---------
---Debido a que las señales de entrada son señales eléctricas de
---tipo digital presentan una característica, no pueden ser
---manejadas como enteros. Para realizar las operaciones necesarias
---se requiere almacenar los valores de las entradas en variables
---de tipo entero.
   if (In1 = '1') then
    In1_aux := 1;
    else
    In1_aux := 0;
    end if;

    if (In2 = '1') then
    In2_aux := 1;
    else
    In2_aux := 0;
    end if;

    if (In3 = '1') then
    In3_aux := 1;
    else
    In3_aux := 0;
    end if;

    if (In4 = '1') then
    In4_aux := 1;
    else
    In4_aux := 0;
    end if;
-------FIN IMPORTACIÓN SEÑALES DE ENTRADA---------
```

```
----ASOCIACIÓN SIGNOS-PESOS--------
---Debido a que los vectores de memoria contienen los pesos y los signos en
---señales diferentes, se requiere asociarle a cada vector de memoria
---el signo que le corresponde para tratarlo como un numero entero.
    if (signo1 = '1') then
    Wen1_aux :=  (-1)*conv_integer(Wen1);
    else
    Wen1_aux :=  conv_integer(Wen1);
    end if;

    if (signo2 = '1') then
    Wen2_aux :=  (-1)*conv_integer(Wen2);
    else
    Wen2_aux :=  conv_integer(Wen2);
    end if;

    if (signo3 = '1') then
    Wen3_aux :=  (-1)*conv_integer(Wen3);
    else
    Wen3_aux :=  conv_integer(Wen3);
    end if;

    if (signo4 = '1') then
    Wen4_aux :=  (-1)*conv_integer(Wen4);
    else
    Wen4_aux :=  conv_integer(Wen4);
    end if;

    if (signo17 = '1') then
    Wen17_aux :=  (-1)*conv_integer(Wen17);
    else
    Wen17_aux :=  conv_integer(Wen17);
    end if;
----FIN ASOCIACIÓN SIGNOS-PESOS--------

-----FUNCION NET------
---Se realiza el cálculo de la función net mediante las variables
---asociadas a los pesos y a las entradas de esta neurona.
    net := Wen17_aux + In1_aux*wEn1_aux + In2_aux*wEn2_aux + In3_aux*wEn3_aux +
In4_aux*wEn4_aux;
---FIN FUNCION NET----

---FUNCION DE ACTIVACIÓN----
---Para el cálculo de la función net se trataron los valores como
---enteros y se requiere llevar la respuesta a un valor digital
---para que sea transferido como una señal eléctrica a la siguiente neurona.
---Esta función de activación corresponde a una función escalón.
    if (net > 0) then
    SalidaN1_aux <= '1';
    else
    SalidaN1_aux <= '0';
    end if;
---FIN FUNCION DE ACTIVACIÓN----
end process;
-------FIN PROCESS DE PUESTA EN MARCHA-----
SalidaN5 <= SalidaN1_aux; ---Exportación de la respuesta SalidaN1_aux
end Behavioral;
-----------------------FIN ARQUITECTURA----------------------
```

Módulo Neurona 6

A la neurona 6 le corresponde identificar los cinco pesos que se encuentran en las direcciones 22, 25, 28, 31 y 34; estas direcciones son manejadas en números binarios ya que el intercambio de información entre módulos se realiza mediante señales eléctricas.

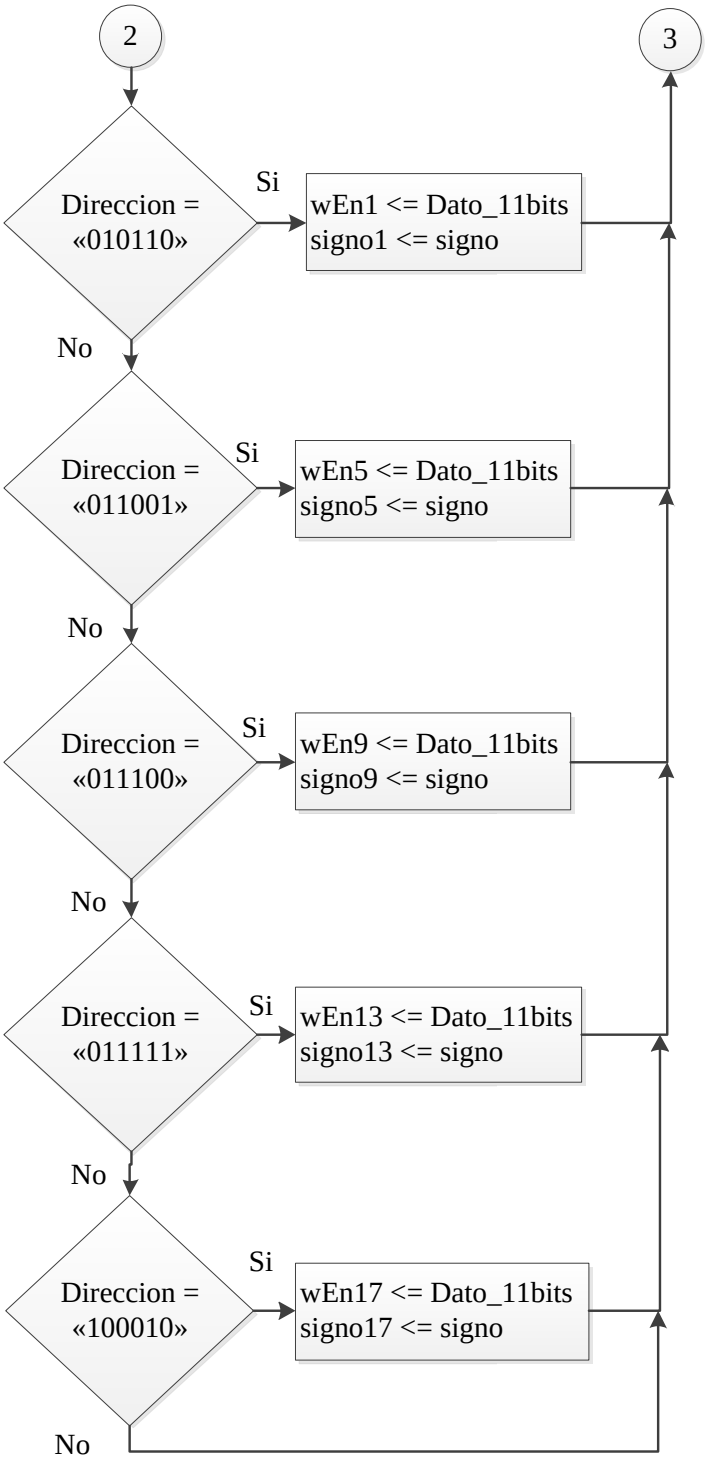


Figura 42. Diagrama de Flujo 4 de 4 del funcionamiento interno de la neurona 6.

La programación requerida para la **Neurona6** es la siguiente:

```
--------DECLARACIÓN DE LIBRERÍAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
------FIN DECLARACIÓN DE LIBRERÍAS------

------SEÑALES I/O------
---Se realiza la declaración de señales
---de entrada y salida del módulo.
entity Neurona6 is
    Port ( Dato_11bits : in  STD_LOGIC_VECTOR (10 downto 0);
           signo : in  STD_LOGIC;
           Direccion : in  STD_LOGIC_VECTOR (5 downto 0);
           In1 : in  STD_LOGIC;
           In2 : in  STD_LOGIC;
           In3 : in  STD_LOGIC;
           In4 : in  STD_LOGIC;
              habilitante : IN std_logic;
              SalidaN6 : out  STD_LOGIC);
end Neurona6;
----FIN SEÑALES I/O----

------------------------ARQUITECTURA-------------------------
---Se describe y se modela el hardware mediante código,
---describiendo así el flujo de datos y comportamiento en este módulo.
architecture Behavioral of Neurona6 is

------------ARREGLOS------------
---Se declaran variables auxiliares y arreglos de memoria
---que permiten realizar cálculos y almacenar la información
---correspondiente a los pesos y signos.
signal wEn1 : std_logic_vector(10 downto 0);
signal wEn2 : std_logic_vector(10 downto 0);
signal wEn3 : std_logic_vector(10 downto 0);
signal wEn4 : std_logic_vector(10 downto 0);
signal wEn17 : std_logic_vector(10 downto 0);
signal signo1 : std_logic;
signal signo2 : std_logic;
signal signo3 : std_logic;
signal signo4 : std_logic;
signal signo17 : std_logic;
signal DirCambio : std_logic_vector(5 downto 0) := "000000";
signal SalidaN1_aux : std_logic;
----------FIN ARREGLOS----------
begin---Inicio del cuerpo de la arquitectura

-------PROCESS DE ENTRENAMIENTO-----
---Mediante este process se almacenan los valores de los
---pesos en la neurona con sus correspondientes signos.
process (habilitante)
 variable cambio: std_logic:='0';---Variable auxiliar
begin---Inicio del cuerpo del process
   if habilitante = '1' and habilitante'event then

----------DETECCIÓN DE CAMBIO-------------
---Se almacena la dirección actual en un vector, y se compara
---con la dirección almacenada en un tiempo previo al actual,
---si ambas direcciones son iguales significa que no se
---produjo ningún cambio, de lo contrario se almacena en una variable
---discreta la existencia de un cambio en la dirección.
    if Direccion = DirCambio then
    cambio := '0';
    else
    DirCambio <= Direccion;
    cambio := '1';
    end if;
---------FIN DETECCIÓN DE CAMBIO----------
```

```
-------CAMBIO DE DIRECCIÓN-------
---Se accede a la variable discreta cambio que contiene
---la información referente a la existencia o no de un cambio
---en la dirección.
    if cambio = '1' then
    cambio := '0';
-------------MEMORIA-----------------
---En caso de haber un cambio de Dirección se almacena la información
---en arreglos de memoria, dicha información corresponde a los pesos y signos
---de la neurona.
    if Direccion = "010110" then
    wEn1 <= Dato_11bits;
    signo1 <= signo;
    elsif Direccion = "011001" then
    wEn2 <= Dato_11bits;
    signo2 <= signo;
    elsif Direccion = "011100" then
    wEn3 <= Dato_11bits;
    signo3 <= signo;
    elsif Direccion = "011111" then
    wEn4 <= Dato_11bits;
    signo4 <= signo;
    elsif Direccion = "100010" then
    wEn17 <= Dato_11bits;
    signo17 <= signo;
    end if;
-----------FIN MEMORIA-----------
    end if;
-------FIN CAMBIO DE DIRECCIÓN-------
    end if;
end process;
-------FIN PROCESS DE ENTRENAMIENTO-----

-------PROCESS DE PUESTA EN MARCHA-----
---Mediante este process se pone en marcha el funcionamiento de la
---neurona, realizando los cálculos para la función net, y su función
---de activación correspondiente.
process (In1, In2, In3, In4) -- ver posibilidad de poner un clk
variable In1_aux, In2_aux, In3_aux, In4_aux, net : integer;
variable wEn1_aux, wEn2_aux, wEn3_aux, wEn4_aux, wEn17_aux : integer;
begin

-------IMPORTACIÓN SEÑALES DE ENTRADA---------
---Debido a que las señales de entrada son señales eléctricas de
---tipo digital presentan una característica, no pueden ser
---manejadas como enteros. Para realizar las operaciones necesarias
---se requiere almacenar los valores de las entradas en variables
---de tipo entero.
   if (In1 = '1') then
    In1_aux := 1;
    else
    In1_aux := 0;
    end if;

    if (In2 = '1') then
    In2_aux := 1;
    else
    In2_aux := 0;
    end if;

    if (In3 = '1') then
    In3_aux := 1;
    else
    In3_aux := 0;
    end if;

    if (In4 = '1') then
    In4_aux := 1;
    else
    In4_aux := 0;
    end if;
    -------FIN IMPORTACIÓN SEÑALES DE ENTRADA---------
```

```
----ASOCIACIÓN SIGNOS-PESOS--------
---Debido a que los vectores de memoria contienen los pesos y los signos en
---señales diferentes, se requiere asociarle a cada vector de memoria
---el signo que le corresponde para tratarlo como un numero entero.
    if (signo1 = '1') then
    Wen1_aux :=  (-1)*conv_integer(Wen1);
    else
    Wen1_aux :=  conv_integer(Wen1);
    end if;

    if (signo2 = '1') then
    Wen2_aux :=  (-1)*conv_integer(Wen2);
    else
    Wen2_aux :=  conv_integer(Wen2);
    end if;

    if (signo3 = '1') then
    Wen3_aux :=  (-1)*conv_integer(Wen3);
    else
    Wen3_aux :=  conv_integer(Wen3);
    end if;

    if (signo4 = '1') then
    Wen4_aux :=  (-1)*conv_integer(Wen4);
    else
    Wen4_aux :=  conv_integer(Wen4);
    end if;

    if (signo17 = '1') then
    Wen17_aux :=  (-1)*conv_integer(Wen17);
    else
    Wen17_aux :=  conv_integer(Wen17);
    end if;
----FIN ASOCIACIÓN SIGNOS-PESOS--------

-----FUNCION NET------
---Se realiza el cálculo de la función net mediante las variables
---asociadas a los pesos y a las entradas de esta neurona.
    net := Wen17_aux + In1_aux*wEn1_aux + In2_aux*wEn2_aux + In3_aux*wEn3_aux +
In4_aux*wEn4_aux;
---FIN FUNCION NET----

---FUNCION DE ACTIVACIÓN----
---Para el cálculo de la función net se trataron los valores como
---enteros y se requiere llevar la respuesta a un valor digital
---para que sea transferido como una señal eléctrica a la siguiente neurona.
---Esta función de activación corresponde a una función escalón.
    if (net > 0) then
    SalidaN1_aux <= '1';
    else
    SalidaN1_aux <= '0';
    end if;
---FIN FUNCION DE ACTIVACIÓN----
end process;
-------FIN PROCESS DE PUESTA EN MARCHA-----
SalidaN6 <= SalidaN1_aux; ---Exportación de la respuesta SalidaN1_aux
end Behavioral;
-----------------------FIN ARQUITECTURA----------------------
```

Módulo Neurona 7

A la neurona 7 le corresponde identificar los cinco pesos que se encuentran en las direcciones 23, 26, 29, 32 y 35; estas direcciones son manejadas en números binarios ya que el intercambio de información entre módulos se realiza mediante señales eléctricas.

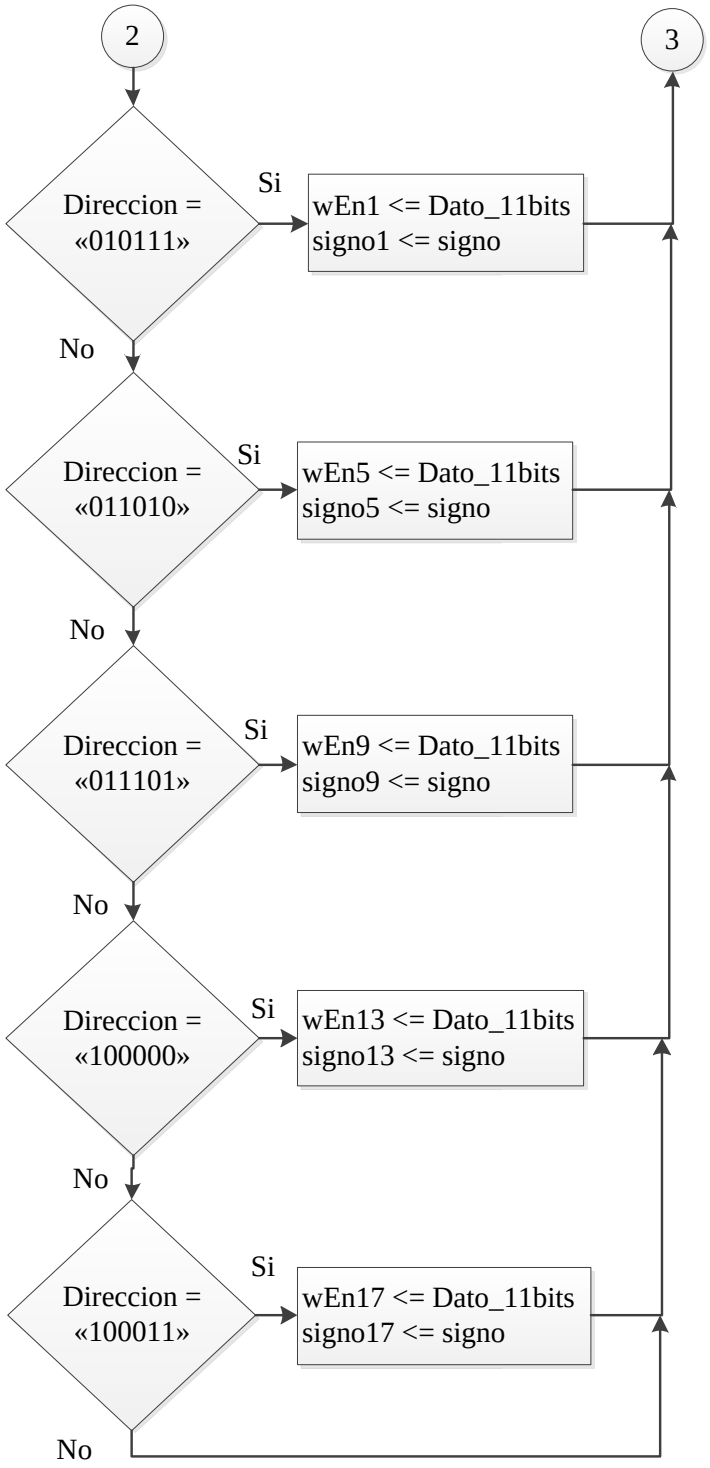


Figura 43. Diagrama de Flujo 4 de 4 del funcionamiento interno de la neurona 7.

La programación requerida para la **Neurona7** es la siguiente:

```
--------DECLARACIÓN DE LIBRERÍAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
------FIN DECLARACIÓN DE LIBRERÍAS------

------SEÑALES I/O------
---Se realiza la declaración de señales
---de entrada y salida del módulo.
entity Neurona7 is
    Port ( Dato_11bits : in  STD_LOGIC_VECTOR (10 downto 0);
           signo : in  STD_LOGIC;
           Direccion : in  STD_LOGIC_VECTOR (5 downto 0);
           In1 : in  STD_LOGIC;
           In2 : in  STD_LOGIC;
           In3 : in  STD_LOGIC;
           In4 : in  STD_LOGIC;
              habilitante : IN std_logic;
              SalidaN7 : out  STD_LOGIC);
end Neurona7;
----FIN SEÑALES I/O----

------------------------ARQUITECTURA-------------------------
---Se describe y se modela el hardware mediante código,
---describiendo así el flujo de datos y comportamiento en este módulo.
architecture Behavioral of Neurona7 is

------------ARREGLOS------------
---Se declaran variables auxiliares y arreglos de memoria
---que permiten realizar cálculos y almacenar la información
---correspondiente a los pesos y signos.
signal wEn1 : std_logic_vector(10 downto 0);
signal wEn2 : std_logic_vector(10 downto 0);
signal wEn3 : std_logic_vector(10 downto 0);
signal wEn4 : std_logic_vector(10 downto 0);
signal wEn17 : std_logic_vector(10 downto 0);
signal signo1 : std_logic;
signal signo2 : std_logic;
signal signo3 : std_logic;
signal signo4 : std_logic;
signal signo17 : std_logic;
signal DirCambio : std_logic_vector(5 downto 0) := "000000";
signal SalidaN1_aux : std_logic;
----------FIN ARREGLOS-----------
begin---Inicio del cuerpo de la arquitectura

-------PROCESS DE ENTRENAMIENTO-----
---Mediante este process se almacenan los valores de los
---pesos en la neurona con sus correspondientes signos.
process (habilitante)
 variable cambio: std_logic:='0';---Variable auxiliar
begin---Inicio del cuerpo del process
   if habilitante = '1' and habilitante'event then

----------DETECCIÓN DE CAMBIO-------------
---Se almacena la dirección actual en un vector, y se compara
---con la dirección almacenada en un tiempo previo al actual,
---si ambas direcciones son iguales significa que no se
---produjo ningún cambio, de lo contrario se almacena en una variable
---discreta la existencia de un cambio en la dirección.
    if Direccion = DirCambio then
    cambio := '0';
    else
    DirCambio <= Direccion;
    cambio := '1';
    end if;
---------FIN DETECCIÓN DE CAMBIO----------
```

```
-------CAMBIO DE DIRECCIÓN-------
---Se accede a la variable discreta cambio que contiene
---la información referente a la existencia o no de un cambio
---en la dirección.
    if cambio = '1' then
    cambio := '0';

-------------MEMORIA-----------------
---En caso de haber un cambio de Dirección se almacena la información
---en arreglos de memoria, dicha información corresponde a los pesos y signos
---de la neurona.
    if Direccion = "010111" then
    wEn1 <= Dato_11bits;
    signo1 <= signo;
    elsif Direccion = "011010" then
    wEn2 <= Dato_11bits;
    signo2 <= signo;
    elsif Direccion = "011101" then
    wEn3 <= Dato_11bits;
    signo3 <= signo;
    elsif Direccion = "100000" then
    wEn4 <= Dato_11bits;
    signo4 <= signo;
    elsif Direccion = "100011" then
    wEn17 <= Dato_11bits;
    signo17 <= signo;
    end if;
-----------FIN MEMORIA-----------
    end if;
-------FIN CAMBIO DE DIRECCIÓN-------
    end if;
end process;
-------FIN PROCESS DE ENTRENAMIENTO-----

-------PROCESS DE PUESTA EN MARCHA-----
---Mediante este process se pone en marcha el funcionamiento de la
---neurona, realizando los cálculos para la función net, y su función
---de activación correspondiente.
process (In1, In2, In3, In4) -- ver posibilidad de poner un clk
variable In1_aux, In2_aux, In3_aux, In4_aux, net : integer;
variable wEn1_aux, wEn2_aux, wEn3_aux, wEn4_aux, wEn17_aux : integer;
begin

-------IMPORTACIÓN SEÑALES DE ENTRADA---------
---Debido a que las señales de entrada son señales eléctricas de
---tipo digital presentan una característica, no pueden ser
---manejadas como enteros. Para realizar las operaciones necesarias
---se requiere almacenar los valores de las entradas en variables
---de tipo entero.
   if (In1 = '1') then
    In1_aux := 1;
    else
    In1_aux := 0;
    end if;

    if (In2 = '1') then
    In2_aux := 1;
    else
    In2_aux := 0;
    end if;

    if (In3 = '1') then
    In3_aux := 1;
    else
    In3_aux := 0;
    end if;

    if (In4 = '1') then
    In4_aux := 1;
    else
    In4_aux := 0;
    end if;
-------FIN IMPORTACIÓN SEÑALES DE ENTRADA---------
```

```
----ASOCIACIÓN SIGNOS-PESOS--------
---Debido a que los vectores de memoria contienen los pesos y los signos en
---señales diferentes, se requiere asociarle a cada vector de memoria
---el signo que le corresponde para tratarlo como un numero entero.
    if (signo1 = '1') then
    Wen1_aux :=  (-1)*conv_integer(Wen1);
    else
    Wen1_aux :=  conv_integer(Wen1);
    end if;

    if (signo2 = '1') then
    Wen2_aux :=  (-1)*conv_integer(Wen2);
    else
    Wen2_aux :=  conv_integer(Wen2);
    end if;

    if (signo3 = '1') then
    Wen3_aux :=  (-1)*conv_integer(Wen3);
    else
    Wen3_aux :=  conv_integer(Wen3);
    end if;

    if (signo4 = '1') then
    Wen4_aux :=  (-1)*conv_integer(Wen4);
    else
    Wen4_aux :=  conv_integer(Wen4);
    end if;

    if (signo17 = '1') then
    Wen17_aux :=  (-1)*conv_integer(Wen17);
    else
    Wen17_aux :=  conv_integer(Wen17);
    end if;
----FIN ASOCIACIÓN SIGNOS-PESOS--------

-----FUNCION NET------
---Se realiza el cálculo de la función net mediante las variables
---asociadas a los pesos y a las entradas de esta neurona.
    net := Wen17_aux + In1_aux*wEn1_aux + In2_aux*wEn2_aux + In3_aux*wEn3_aux +
In4_aux*wEn4_aux;
---FIN FUNCION NET----

---FUNCION DE ACTIVACIÓN----
---Para el cálculo de la función net se trataron los valores como
---enteros y se requiere llevar la respuesta a un valor digital
---para que sea transferido como una señal eléctrica a la siguiente neurona.
---Esta función de activación corresponde a una función escalón.
    if (net > 0) then
    SalidaN1_aux <= '1';
    else
    SalidaN1_aux <= '0';
    end if;
---FIN FUNCION DE ACTIVACIÓN----
end process;
-------FIN PROCESS DE PUESTA EN MARCHA-----
SalidaN7 <= SalidaN1_aux; ---Exportación de la respuesta SalidaN1_aux
end Behavioral;
-----------------------FIN ARQUITECTURA----------------------
```

Módulo Neurona 8

A la neurona 8 le corresponde identificar los cuatro pesos que se encuentran en las direcciones 36, 37, 38 y 39; estas direcciones son manejadas en números binarios ya que el intercambio de información entre módulos se realiza mediante señales eléctricas.

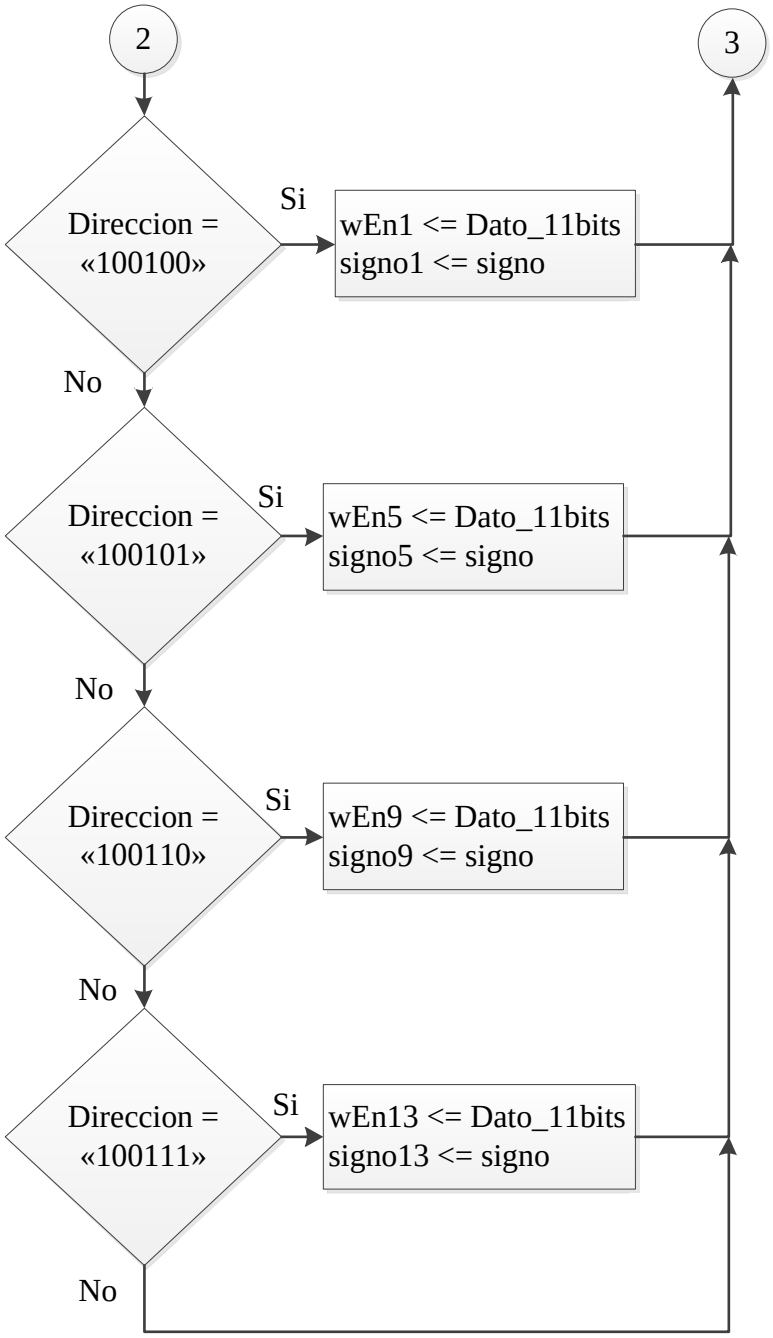


Figura 44. Diagrama de Flujo 4 de 4 del funcionamiento interno de la neurona 8.

La programación requerida para la **Neurona8** es la siguiente:

```
--------DECLARACIÓN DE LIBRERÍAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
------FIN DECLARACIÓN DE LIBRERÍAS------

------SEÑALES I/O------
---Se realiza la declaración de señales
---de entrada y salida del módulo.
entity Neurona8 is
    Port ( Dato_11bits : in  STD_LOGIC_VECTOR (10 downto 0);
           signo : in  STD_LOGIC;
           Direccion : in  STD_LOGIC_VECTOR (5 downto 0);
           In1 : in  STD_LOGIC;
           In2 : in  STD_LOGIC;
           In3 : in  STD_LOGIC;
           habilitante : IN std_logic;
           SalidaN8 : out  STD_LOGIC);
end Neurona8;
----FIN SEÑALES I/O----

------------------------ARQUITECTURA------------------------
---Se describe y se modela el hardware mediante código,
---describiendo así el flujo de datos y comportamiento en este módulo.
architecture Behavioral of Neurona8 is

------------ARREGLOS------------
---Se declaran variables auxiliares y arreglos de memoria
---que permiten realizar cálculos y almacenar la información
---correspondiente a los pesos y signos.
signal wEn1 : std_logic_vector(10 downto 0);
signal wEn2 : std_logic_vector(10 downto 0);
signal wEn3 : std_logic_vector(10 downto 0);
signal wEn4 : std_logic_vector(10 downto 0);
signal signo1 : std_logic;
signal signo2 : std_logic;
signal signo3 : std_logic;
signal signo4 : std_logic;
signal DirCambio : std_logic_vector(5 downto 0) := "000000";
signal SalidaN1_aux : std_logic;
----------FIN ARREGLOS----------
begin---Inicio del cuerpo de la arquitectura

-------PROCESS DE ENTRENAMIENTO-----
---Mediante este process se almacenan los valores de los
---pesos en la neurona con sus correspondientes signos.
process (habilitante)
 variable cambio: std_logic:='0';---Variable auxiliar
begin---Inicio del cuerpo del process
   if habilitante = '1' and habilitante'event then

----------DETECCIÓN DE CAMBIO-------------
---Se almacena la dirección actual en un vector, y se compara
---con la dirección almacenada en un tiempo previo al actual,
---si ambas direcciones son iguales significa que no se
---produjo ningún cambio, de lo contrario se almacena en una variable
---discreta la existencia de un cambio en la dirección.
    if Direccion = DirCambio then
    cambio := '0';
    else
    DirCambio <= Direccion;
    cambio := '1';
    end if;
---------FIN DETECCIÓN DE CAMBIO----------
```

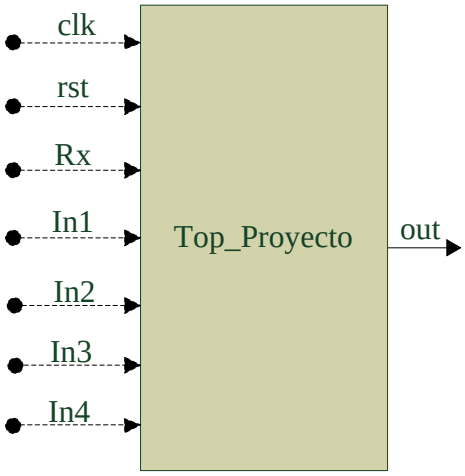
```
-------CAMBIO DE DIRECCIÓN-------
---Se accede a la variable discreta cambio que contiene
---la información referente a la existencia o no de un cambio
---en la dirección.
    if cambio = '1' then
    cambio := '0';

-------------MEMORIA-----------------
---En caso de haber un cambio de Dirección se almacena la información
---en arreglos de memoria, dicha información corresponde a los pesos y signos
---de la neurona.
    if Direccion = "100100" then
    wEn1 <= Dato_11bits;
    signo1 <= signo;
    elsif Direccion = "100101" then
    wEn2 <= Dato_11bits;
    signo2 <= signo;
    elsif Direccion = "100110" then
    wEn3 <= Dato_11bits;
    signo3 <= signo;
    elsif Direccion = "100111" then
    wEn4 <= Dato_11bits;
    signo4 <= signo;
    end if;
-----------FIN MEMORIA-----------
    end if;
-------FIN CAMBIO DE DIRECCIÓN-------
    end if;

end process;
-------FIN PROCESS DE ENTRENAMIENTO-----

-------PROCESS DE PUESTA EN MARCHA-----
---Mediante este process se pone en marcha el funcionamiento de la
---neurona, realizando los cálculos para la función net, y su función
---de activación correspondiente.
process (In1, In2, In3) -- ver posibilidad de poner un clk
variable In1_aux, In2_aux, In3_aux, net : integer;
variable wEn1_aux, wEn2_aux, wEn3_aux, wEn4_aux : integer;
begin

-------IMPORTACIÓN SEÑALES DE ENTRADA---------
---Debido a que las señales de entrada son señales eléctricas de
---tipo digital presentan una característica, no pueden ser
---manejadas como enteros. Para realizar las operaciones necesarias
---se requiere almacenar los valores de las entradas en variables
---de tipo entero.
   if (In1 = '1') then
    In1_aux := 1;
    else
    In1_aux := 0;
    end if;

    if (In2 = '1') then
    In2_aux := 1;
    else
    In2_aux := 0;
    end if;

    if (In3 = '1') then
    In3_aux := 1;
    else
    In3_aux := 0;
    end if;
-------FIN IMPORTACIÓN SEÑALES DE ENTRADA---------
```

```
----ASOCIACIÓN SIGNOS-PESOS--------
---Debido a que los vectores de memoria contienen los pesos y los signos en
---señales diferentes, se requiere asociarle a cada vector de memoria
---el signo que le corresponde para tratarlo como un numero entero.
    if (signo1 = '1') then
    Wen1_aux :=  (-1)*conv_integer(Wen1);
    else
    Wen1_aux :=  conv_integer(Wen1);
    end if;

    if (signo2 = '1') then
    Wen2_aux :=  (-1)*conv_integer(Wen2);
    else
    Wen2_aux :=  conv_integer(Wen2);
    end if;

    if (signo3 = '1') then
    Wen3_aux :=  (-1)*conv_integer(Wen3);
    else
    Wen3_aux :=  conv_integer(Wen3);
    end if;

    if (signo4 = '1') then
    Wen4_aux :=  (-1)*conv_integer(Wen4);
    else
    Wen4_aux :=  conv_integer(Wen4);
    end if;
----FIN ASOCIACIÓN SIGNOS-PESOS--------

-----FUNCION NET------
---Se realiza el cálculo de la función net mediante las variables
---asociadas a los pesos y a las entradas de esta neurona.
    net := Wen4_aux + In1_aux*wEn1_aux + In2_aux*wEn2_aux + In3_aux*wEn3_aux;
---FIN FUNCION NET----

---FUNCION DE ACTIVACIÓN----
---Para el cálculo de la función net se trataron los valores como
---enteros y se requiere llevar la respuesta a un valor digital
---para que sea transferido como una señal eléctrica a la siguiente neurona.
---Esta función de activación corresponde a una función escalón.
    if (net > 0) then
    SalidaN1_aux <= '1';
    else
    SalidaN1_aux <= '0';
    end if;
---FIN FUNCION DE ACTIVACIÓN----
end process;
-------FIN PROCESS DE PUESTA EN MARCHA-----
SalidaN8 <= not SalidaN1_aux; ---Exportación de la respuesta SalidaN1_aux
end Behavioral;
-----------------------FIN ARQUITECTURA----------------------
```

3.6. Sistema completo en FPGA

El sistema completo de la tarjeta se puede representar mediante un solo módulo el mismo que contiene a todos los demás, dicho módulo se ilustra en la Figura 45.

.



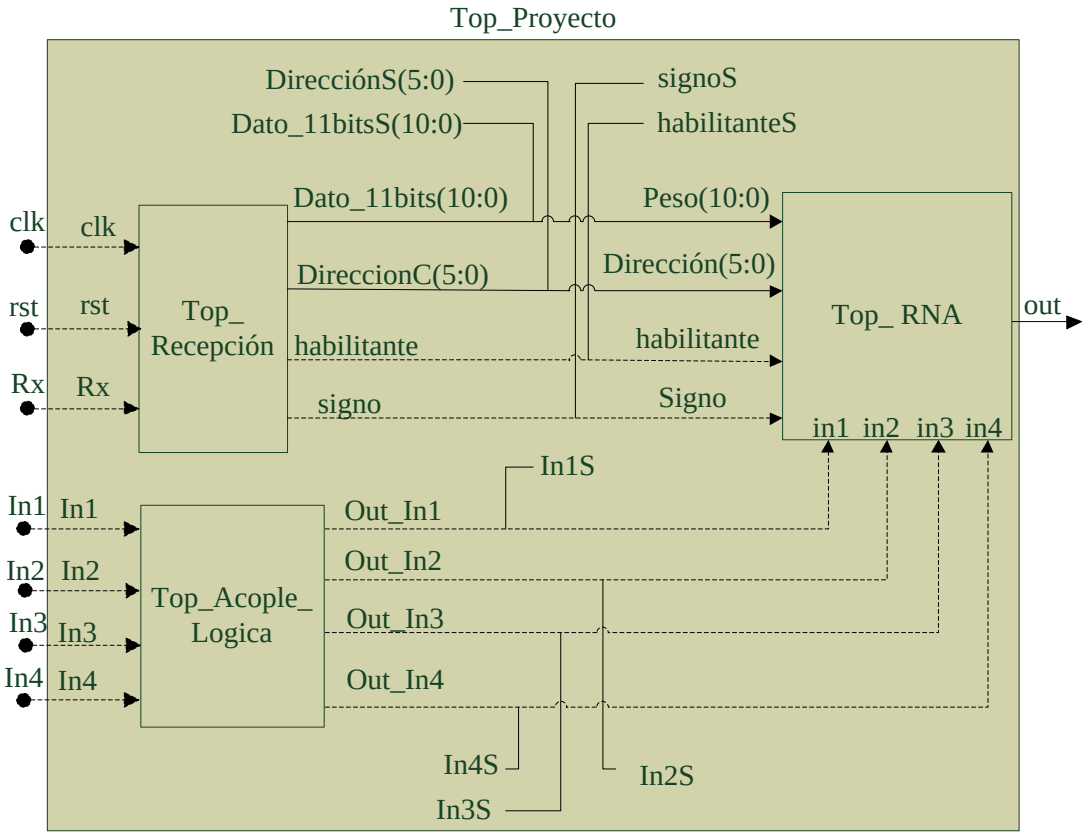


Figura 45. Diagrama RTL de todo el sistema embebido.

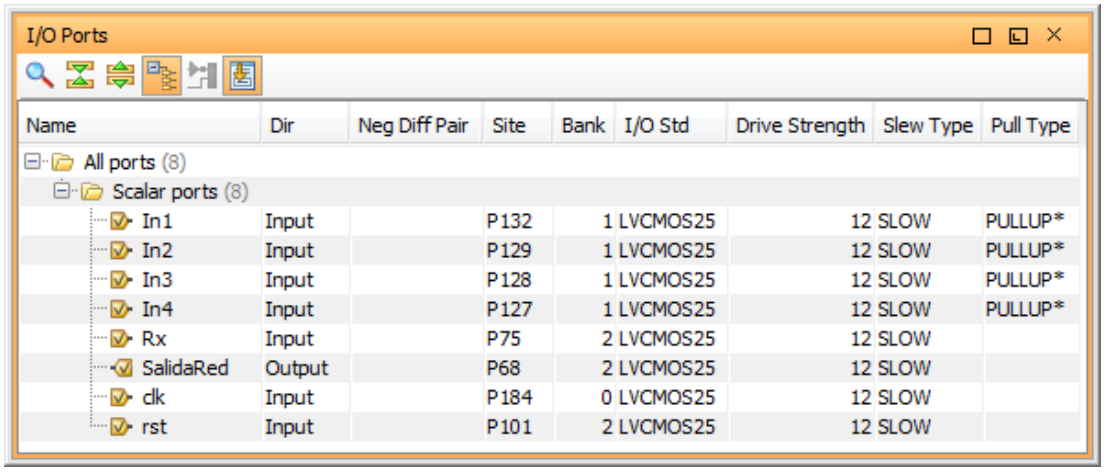
En la programación de este módulo es la siguiente:

```
--------DECLARACIÓN DE LIBRERÍAS--------
---Son necesarias porque contienen paquetes y archivos de
---configuración con sus correspondientes arquitecturas.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
------FIN DECLARACIÓN DE LIBRERÍAS------

------SEÑALES I/O------
---Se realiza la declaración de señales
---de entrada y salida del módulo
entity Top_Proyecto is
    Port ( clk : in  STD_LOGIC;
           Rx : in  STD_LOGIC;
           rst : in  STD_LOGIC;
              In1 : IN std_logic;
              In2 : IN std_logic;
              In3 : IN std_logic;
              In4 : IN std_logic;

           SalidaRed : out  STD_LOGIC);
end Top_Proyecto;
----FIN SEÑALES I/O----

------------------------ARQUITECTURA------------------------
---Se describe y se modela el hardware mediante código,
---describiendo así el flujo de datos y comportamiento en este módulo.
architecture Behavioral of Top_Proyecto is

---SEÑALES AUXILIARES INTERNAS-----
```

```
---Se declaran señales internas al módulo,
---útiles para la interconexión entre módulos
---internos
signal Dato_11bitsS: std_logic_vector(10 downto 0);
signal signoS: std_logic;
signal DireccionS: std_logic_vector(5 downto 0);
signal In1S: std_logic;
signal In2S: std_logic;
signal In3S: std_logic;
signal In4S: std_logic;
signal habilitanteS : std_logic;
---FIN SEÑALES AUXILIARES INTERNAS---

------MÓDULOS INTERNOS------
---Se Declaran los módulos internos con sus
---correspondientes entradas y salidas

---MÓDULO Top_Recepcion---
---Dentro del módulo Top_Proyecto se declara un módulo
---interno denominado Top_Recepcion, con sus
---correspondientes entradas y salidas
COMPONENT Top_Recepcion
    PORT(
        clk : IN std_logic;
        Rx : IN std_logic;
        rst : IN std_logic;
        signo : OUT std_logic;
        habilitante : OUT std_logic;
        DireccionC : OUT std_logic_vector(5 downto 0);
        Dato_11bits : OUT std_logic_vector(10 downto 0)
        );
    END COMPONENT;
---FIN MÓDULO Top_Recepcion---

---MÓDULO Top_Red_Neuronal---
---Dentro del módulo Top_Proyecto se declara un módulo
---interno denominado Top_Red_Neuronal, con sus
---correspondientes entradas y salidas
COMPONENT Top_Red_Neuronal
    PORT(
        Dato_11bits : IN std_logic_vector(10 downto 0);
        signo : IN std_logic;
        Direccion : IN std_logic_vector(5 downto 0);
        In1 : IN std_logic;
        In2 : IN std_logic;
        In3 : IN std_logic;
        In4 : IN std_logic;
        habilitante : IN std_logic;
        Salida : OUT std_logic
        );
    END COMPONENT;
---FIN MÓDULO Top_Red_Neuronal---

---MÓDULO Top_Acople_Logica---
---Dentro del módulo Top_Proyecto se declara un módulo
---interno denominado Top_Acople_Logica, con sus
---correspondientes entradas y salidas
COMPONENT Top_Acople_Logica
    PORT(
        In1 : IN std_logic;
        In2 : IN std_logic;
        In3 : IN std_logic;
        In4 : IN std_logic;
        Out_In1 : OUT std_logic;
        Out_In2 : OUT std_logic;
        Out_In3 : OUT std_logic;
        Out_In4 : OUT std_logic
        );
    END COMPONENT;
---FIN MÓDULO Top_Acople_Logica---

----INSTANCIACIÓN----
---Se realiza la interconexión interna
---entre módulos internos, mediante sus
```

```
---correspondientes señales de entrada y salida.
Begin

---CONEXIÓN DEL MÓDULO Top_Recepcion---
---Se especifica con que señales del exterior
---van a ser conectadas las señales de entrada
---y salida del módulo Top_Recepcion
Inst_Top_Recepcion: Top_Recepcion PORT MAP(
        clk => clk,
        Rx => Rx,
        rst => rst,
        signo => signoS,
        habilitante => habilitanteS,
        DireccionC => DireccionS,
        Dato_11bits => Dato_11bitsS
    );
---FIN CONEXIÓN DEL MÓDULO Top_Recepcion---

---CONEXIÓN DEL MÓDULO Top_Red_Neuronal---
---Se especifica con que señales del exterior
---van a ser conectadas las señales de entrada
---y salida del módulo Top_Red_Neuronal
Inst_Top_Red_Neuronal: Top_Red_Neuronal PORT MAP(
        Dato_11bits => Dato_11bitsS,
        signo => signoS,
        Direccion => DireccionS,
        In1 => In1S,
        In2 => In2S,
        In3 => In3S,
        In4 => In4S,
        habilitante => habilitanteS,
        Salida => SalidaRed
    );
---FIN CONEXIÓN DEL MÓDULO Top_Red_Neuronal---

---CONEXIÓN DEL MÓDULO Top_Acople_Logica---
---Se especifica con que señales del exterior
---van a ser conectadas las señales de entrada
---y salida del módulo Top_Acople_Logica
Inst_Top_Acople_Logica: Top_Acople_Logica PORT MAP(
        In1 => In1,
        In2 => In2,
        In3 => In3,
        In4 => In4,
        Out_In1 => In1S,
        Out_In2 => In2S,
        Out_In3 => In3S,
        Out_In4 => In4S
    );
---FIN CONEXIÓN DEL MÓDULO Top_Red_Neuronal---

end Behavioral;
-----FIN INSTANCIACIÓN----
----------------------FIN ARQUITECTURA---------------------
```

La configuración de los puertos de entradas y salidas de la tarjeta FPGA se muestran en la Figura 46.



Name	Dir	Neg Diff Pair	Site	Bank	I/O Std	Drive Strength	Slew Type	Pull Type
All ports (8)								
Scalar ports (8)								
In1	Input		P132	1	LVCMOS25	12	SLOW	PULLUP*
In2	Input		P129	1	LVCMOS25	12	SLOW	PULLUP*
In3	Input		P128	1	LVCMOS25	12	SLOW	PULLUP*
In4	Input		P127	1	LVCMOS25	12	SLOW	PULLUP*
Rx	Input		P75	2	LVCMOS25	12	SLOW	
SalidaRed	Output		P68	2	LVCMOS25	12	SLOW	
clk	Input		P184	0	LVCMOS25	12	SLOW	
rst	Input		P101	2	LVCMOS25	12	SLOW	

Figura 46. Configuración de pines de Entradas/Salidas.

La codificación para especificar la ubicación de las señales de entrada y salida en los puertos correspondientes de la FPGA se muestra a continuación:

```
NET "Rx" LOC = P75;
NET "SalidaRed" LOC = P68;
NET "clk" LOC = P184;
NET "rst" LOC = P101;


NET "In1" LOC = P132;
NET "In2" LOC = P129;
NET "In3" LOC = P128;
NET "In4" LOC = P127;


NET "In1" PULLUP;
NET "In2" PULLUP;
NET "In3" PULLUP;
NET "In4" PULLUP;
```

Para la implementación de hardware se requirió realizar la conexión entre un módulo de pulsadores con la FPGA, a través del cual se van a ingresar los valores digitales necesarios para evaluar la RNA implementada. Además se conectó el módulo PL2303 a la FPGA, el cual posibilita la comunicación serial desde una computadora portátil que no posee puertos seriales, ya que el dispositivo PL2303 crea un puerto serial virtual y convierte la comunicación USB en comunicación UART.

En la parte superior de la tarjeta se observa el puerto 8I/Os_2, el cual se conecta con un módulo de 8 pulsadores, cuya conexión se ilustra en la Figura 47.

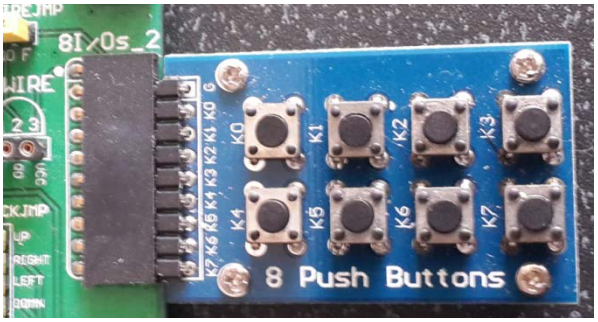


Figura 47. Conexión FPGA-Modulo de Pulsadores

La parte posterior del puerto 8I/Os_2 en la FPGA se ilustra en la Figura 48.

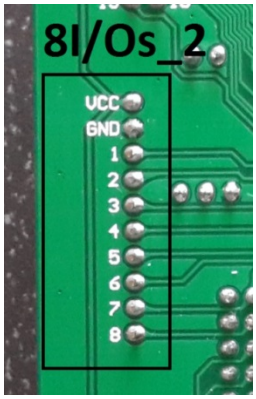


Figura 48. Pines del Puerto 8I/Os_2

Donde se observa una numeración y denominación de pines, la conexión entre estos pines y el módulo de pulsadores se ilustra en la *Tabla 13*.

Tabla 13.

Conexiones FPGA-Módulo de pulsadores

Pines de la FPGA del puerto 8I/Os_2	Pines del Módulo 8 Push Buttons
Vcc	Sin conectar
Gnd	G
1	K0
2	K1
3	K2
4	K3

5	K4
6	K5
7	K6
8	K7

En la parte superior de la tarjeta se observa el puerto 8I/Os_1, que se conecta con el módulo PL2303, el cual genera un puerto serial virtual en la PC para el envío de datos hacia la FPGA, cuya conexión se ilustra en la Figura 49.

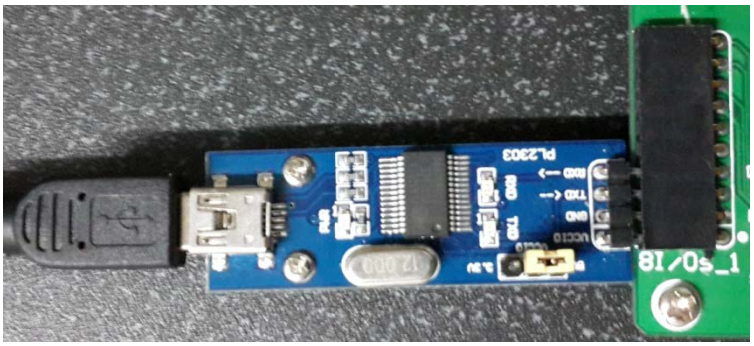


Figura 49. Conexión FPGA-PL2303

La parte posterior del puerto 8I/Os_1 en la FPGA se ilustra en la Figura 50.

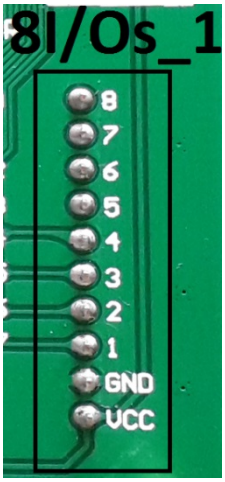


Figura 50. Pines del Puerto 8I/Os_1

Donde se observa una numeración y denominación de pines, la conexión entre estos pines y el módulo PL2303 se ilustra en la *Tabla 14*.

Tabla 14.

Conexiones FPGA-PL2303

Pines de la FPGA del puerto 8I/Os_1	Pines del Módulo PL2303
8	Sin conectar
7	Sin conectar
6	Sin conectar
5	Sin conectar
4	Sin conectar
3	Sin conectar
2	RXD
1	TXD
Gnd	GND
Vcc	VCCI0

CAPÍTULO IV: EVALUACIÓN Y ANÁLISIS

4.1. Escenarios de evaluación

A la RNA se la expuso a los escenarios, que se explican mediante la *Tabla 15*; con funciones lógicas conocidas y no conocidas, entre las cuales se incluye la función or exclusivo por su característica de no ser linealmente separable y por lo tanto poseer un mayor grado de dificultad. Además la RNA fue expuesta a funciones incompletas. Estas pruebas se realizaron para el entrenamiento, transferencia y puesta en marcha de la RNA.

Tabla 15.

Descripción de la las Funciones de Respuesta de la RNA

Función	Equivalencia	Ubicación
F1_AUX1	in1 Xor in2 Xor in3 Xor in4	Variable auxiliar
F1_AUX2	F1_AUX Or in4	Variable auxiliar
F1	F1_AUX1 Nand F1_AUX2	*Tabla 16*
F2	Es la misma F1 pero incompleta	*Tabla 16*
F3	in3 And in4	*Tabla 16*
F4	in3 Or in4	*Tabla 16*
F5	in1 Xor in4	*Tabla 17*
F6	in2 Xor in3	*Tabla 17*
F7	F5 Or F6	*Tabla 17*
F8	F5 And F6	*Tabla 17*
F9	in3 Xor in4 Con valores incompletos	*Tabla 18*
F10	in3 Xor in4 Con valores de Ceros	*Tabla 18*
F11	in3 Xor in4 Con valores de Unos	*Tabla 18*
F12	Función aleatoria incompleta	Tabla 18
F13	Función con dos respuestas	*Tabla 19*
F14	Función de única respuesta	*Tabla 19*
F15	Función incompleta con unos	*Tabla 19*
F16	Función incompleta con ceros	*Tabla 19*

En F1 se resuelve la función booleana Nand entre F1_AUX1 y F1_AUX2. En F2 se resuelve la misma función F1, con la considerable diferencia de que se restringe la información y se proporcionan tres datos menos. En F3 se resuelve una función

booleana “AND” entre in3 e in4, asumiendo que estas son las únicas entradas, por lo cual se generan cuatro datos de respuesta y resto queda sin completarse. En F4 se resuelve una función booleana “OR” entre in3 e in4, asumiendo que estas son las únicas entradas, por lo cual se generan cuatro datos de respuesta y el resto queda incompleto.

Tabla 16.

Respuestas esperadas de las funciones F1 a F4.

				F1	**F2**	**F3**	**F4**
in1	in2	in3	in4	F1	F2	in3 AND in4	in3 OR in4
0	0	0	0	1	1	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	0	1
0	0	1	1	1	1	1	1
0	1	0	0	0	0	-	-
0	1	0	1	1	1	-	-
0	1	1	0	1	1	-	-
0	1	1	1	1	1	-	-
1	0	0	0	0	0	-	-
1	0	0	1	1	1	-	-
1	0	1	0	1	-	-	-
1	0	1	1	1	-	-	-
1	1	0	0	1	1	-	-
1	1	0	1	1	1	-	-
1	1	1	0	1	1	-	-
1	1	1	1	1	-	-	-

En F5 se resuelve la función booleana “XOR” entre in1 e in4 sin tomar en cuenta las otras dos entradas. En F6 se resuelve la función booleana “XOR” entre in2 e in3 sin tomar en cuenta las otras dos entradas. En F7 se resuelve la función booleana

"OR" entre las funciones previamente calculadas F5 y F6. En F8 se resuelve la función booleana "AND" entre las funciones previamente calculadas F5 y F6.

Tabla 17.

Respuestas esperadas de las funciones F5 a F8.

				F5	**F6**	**F7**	**F8**
in1	in2	in3	in4	in1 Xor in4	in2 Xor in3	F5 Or F6	F5 And F6
0	0	0	0	0	0	0	0
0	0	0	1	1	0	1	0
0	0	1	0	0	1	1	0
0	0	1	1	1	1	1	1
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	1
0	1	1	0	0	0	0	0
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	0
1	0	0	1	0	0	0	0
1	0	1	0	1	1	1	1
1	0	1	1	0	1	1	0
1	1	0	0	1	1	1	1
1	1	0	1	0	1	1	0
1	1	1	0	1	0	1	0
1	1	1	1	0	0	0	0

En F9 se resuelve la función booleana "XOR" entre in3 e in4, asumiendo que son las únicas entradas por lo que se tienen solamente cuatro respuestas, dejando así incompleta la tabla global. En F10 se resuelve la función booleana "XOR" entre in3 e in4, asumiendo que son las únicas entradas por lo que se tienen solamente cuatro

respuestas lógicas, sin embargo se procede a llenar los espacios vacíos con ceros lógicos. En F11 se resuelve la función booleana “XOR” entre in3 e in4, asumiendo que son las únicas entradas por lo que se tienen solamente cuatro respuestas lógicas, sin embargo se procede a llenar los espacios vacíos con unos lógicos. En F12 se resuelve una función aleatoria que además se encuentra incompleta en sus respuestas.

Tabla 18.

Respuestas esperadas de las funciones F9 a F12.

				F9	**F10**	**F11**	**F12**
in1	in2	in3	in4	in3 XOR in4	in3 XOR in4 y Ceros	in3 XOR in4 y Unos	F12
0	0	0	0	0	0	0	-
0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	1
0	0	1	1	0	0	0	0
0	1	0	0	-	0	1	1
0	1	0	1	-	0	1	1
0	1	1	0	-	0	1	-
0	1	1	1	-	0	1	-
1	0	0	0	-	0	1	1
1	0	0	1	-	0	1	-
1	0	1	0	-	0	1	-
1	0	1	1	-	0	1	1
1	1	0	0	-	0	1	0
1	1	0	1	-	0	1	1
1	1	1	0	-	0	1	-
1	1	1	1	-	0	1	1

En F13 se resuelve una función que solamente tiene dos respuestas, y el resto queda incompleto. En F14 se resuelve una función con el mínimo de respuestas requeridas, es decir con una sola respuesta, quedando el resto incompleto. En F15 se resuelve una función incompleta cuyas respuestas son solamente unos lógicos. En

F16 se resuelve una función incompleta cuyas respuestas son solamente ceros lógicos.

Tabla 19.

Respuestas esperadas de las funciones F13 a F16.

in1	in2	in3	in4	F13	F14	F15	F16
0	0	0	0	1	-	-	-
0	0	0	1	-	-	-	-
0	0	1	0	-	-	-	-
0	0	1	1	-	-	-	-
0	1	0	0	-	-	-	-
0	1	0	1	0	-	-	-
0	1	1	0	-	-	-	-
0	1	1	1	-	-	1	0
1	0	0	0	-	-	-	-
1	0	0	1	-	-	-	-
1	0	1	0	-	1	-	-
1	0	1	1	-	-	1	0
1	1	0	0	-	-	-	-
1	1	0	1	-	-	1	0
1	1	1	0	-	-	1	0
1	1	1	1	-	-	-	-

4.2. Resultados

Al ingresar la respuesta requerida en la interfaz HMI y al proceder a realizar el entrenamiento de la RNA, el algoritmo de retro-propagación respondió con las gráficas de la Figura 51 a la Figura 66 donde se muestra en número de épocas con relación al error de cada época de entrenamiento.

En la Figura 51, se observa el entrenamiento de la RNA para la función lógica F1, los valores de arranque de los pesos generan un 4% de error inicial, y gran parte de la convergencia ocurre hasta la época 1000, 4000 épocas más tarde alcanza la convergencia total. No se observan oscilaciones considerables ni estancamientos en el transcurso del entrenamiento.

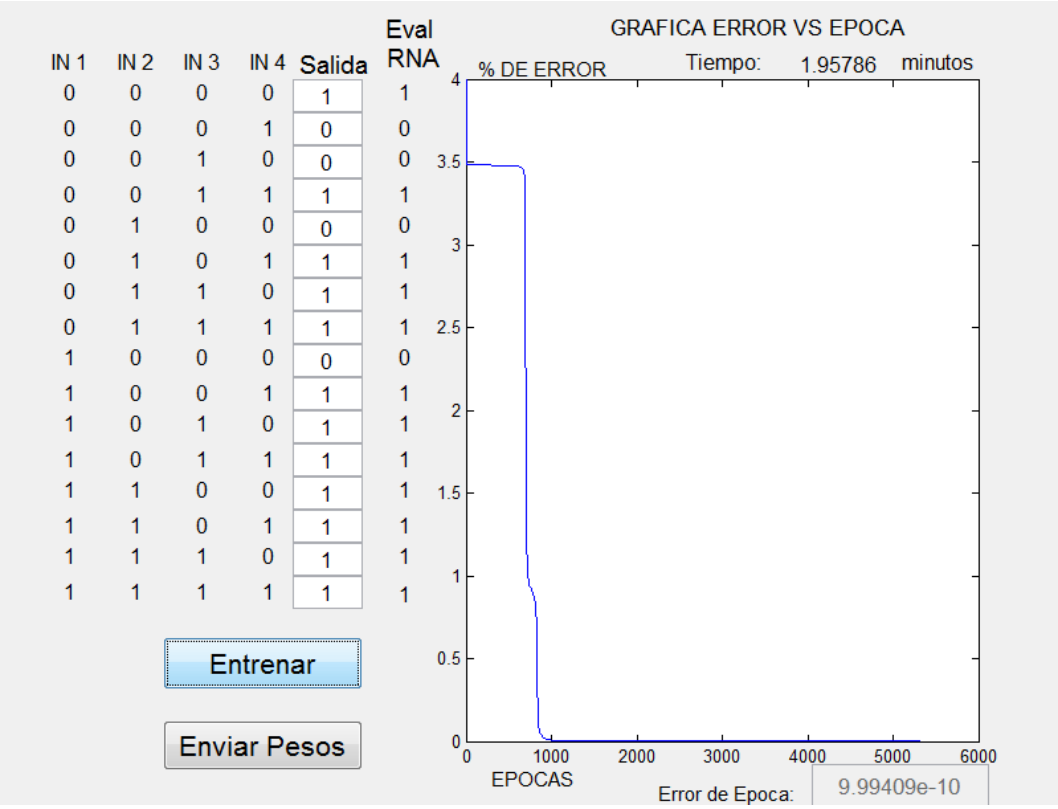


Figura 51. Respuesta a la función lógica F1.

En la Figura 52, se observa el entrenamiento de la RNA para la función lógica F2, los valores de arranque de los pesos generan un 4% de error inicial, gran parte de la convergencia ocurre hasta la época 800, 4000 épocas más tarde alcanza la convergencia total. No se observan oscilaciones considerables ni estancamientos en el transcurso del entrenamiento. A pesar de que esta grafica es una función incompleta se observa un similar comportamiento a su homóloga F1.

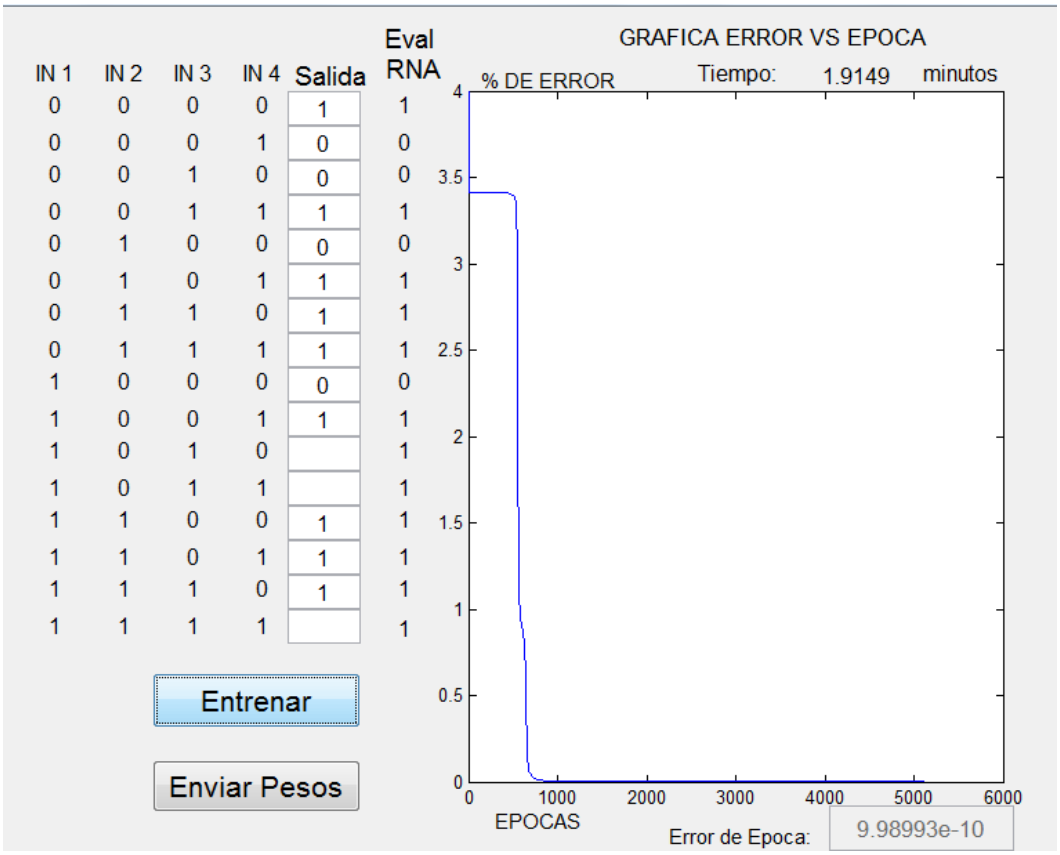


Figura 52. Respuesta a la función lógica F2.

En la Figura 53, se observa el entrenamiento de la RNA para la función lógica F3, los valores de arranque de los pesos generan un 3% de error inicial, se produce una convergencia casi inmediata hasta un error de 0.9%; posteriormente se observa un comportamiento constante en el error hasta la época 1200, donde ocurre un gran decremento hasta época 1300. 2500 épocas más tarde se termina el entrenamiento alcanzando la convergencia total. No existen oscilaciones, sin embargo el comportamiento horizontal de la gráfica demuestra que se mantuvo un error constante durante 1200 épocas.

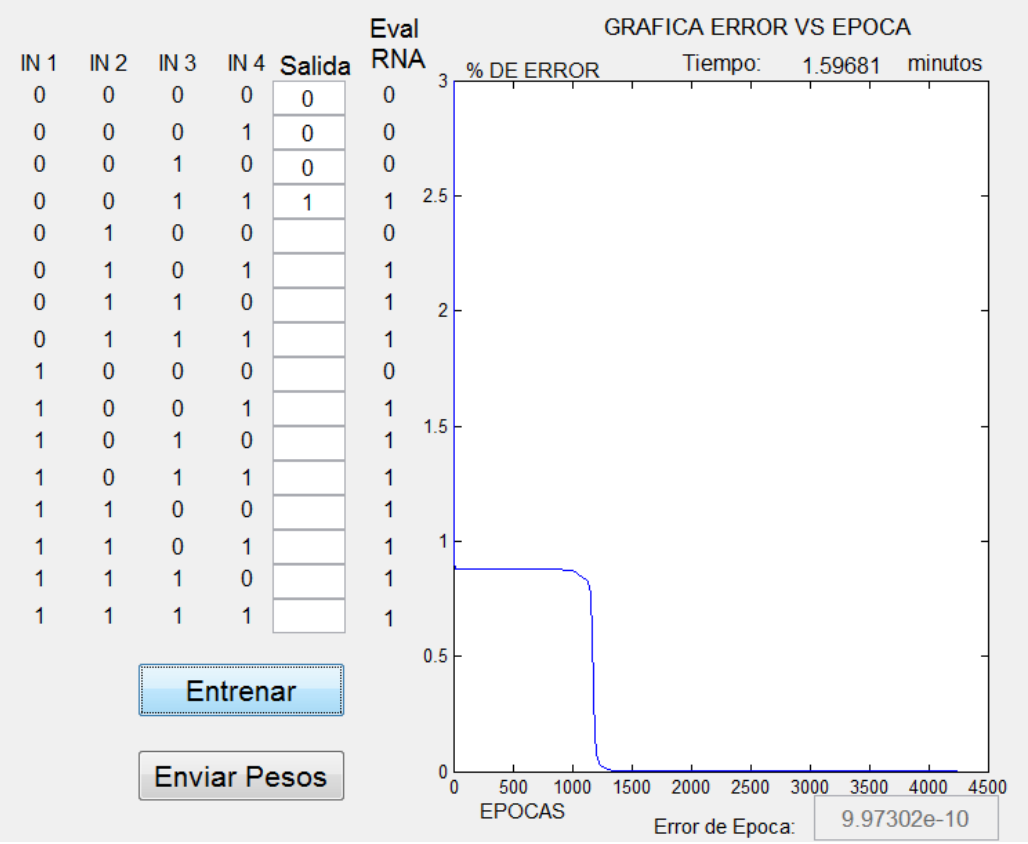


Figura 53. Respuesta a la función lógica F3.

En la Figura 54, se observa el entrenamiento de la RNA para la función lógica F4, los valores de arranque de los pesos generan un 1% de error inicial y gran parte de la convergencia ocurre hasta la época 1200, 4000 épocas más tarde alcanza la convergencia total. No existen oscilaciones ni estancamientos es decir graficas horizontales extensas en el transcurso del entrenamiento. Se observa un similar comportamiento a las gráficas de F1 y F2.

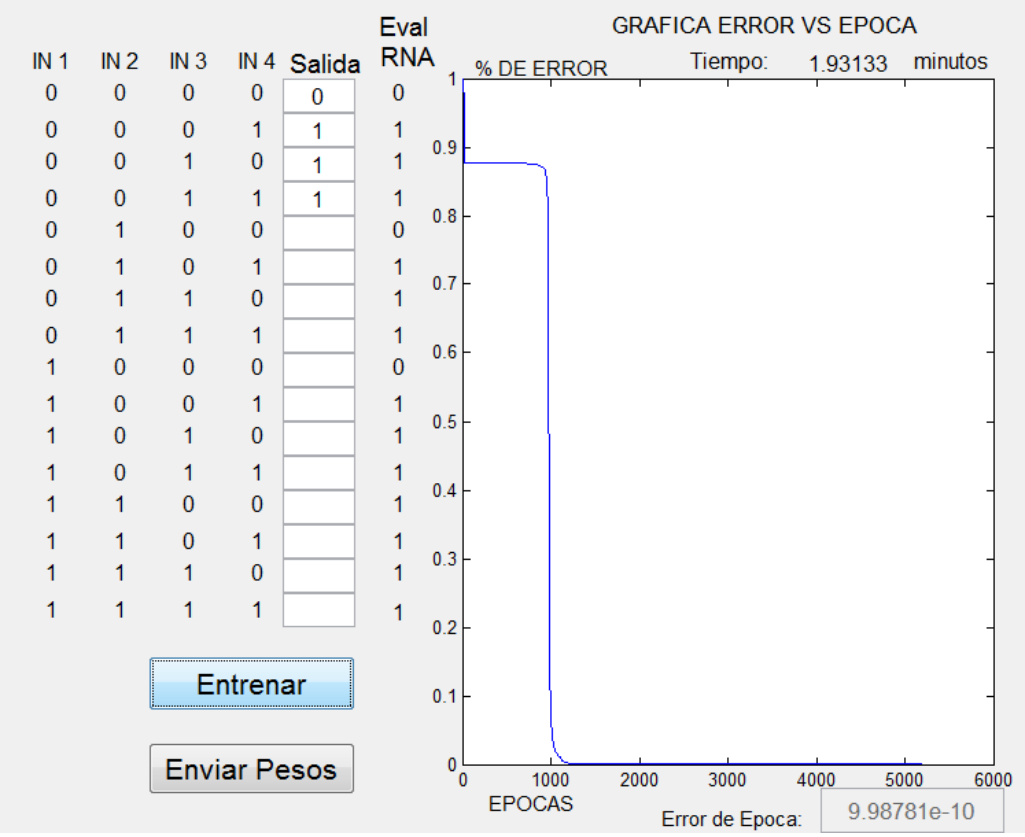


Figura 54. Respuesta a la función lógica F4.

En la Figura 55, se observa el entrenamiento de la RNA para la función lógica F5, los valores de arranque de los pesos generan un 8% de error inicial y gran parte de la convergencia ocurre hasta la época 600, 3000 épocas más tarde alcanza la convergencia total. Se tiene una pequeña oscilación entre la época 400 y 500, no existen estancamientos es decir no se observan valores de error constantes en el transcurso del entrenamiento.

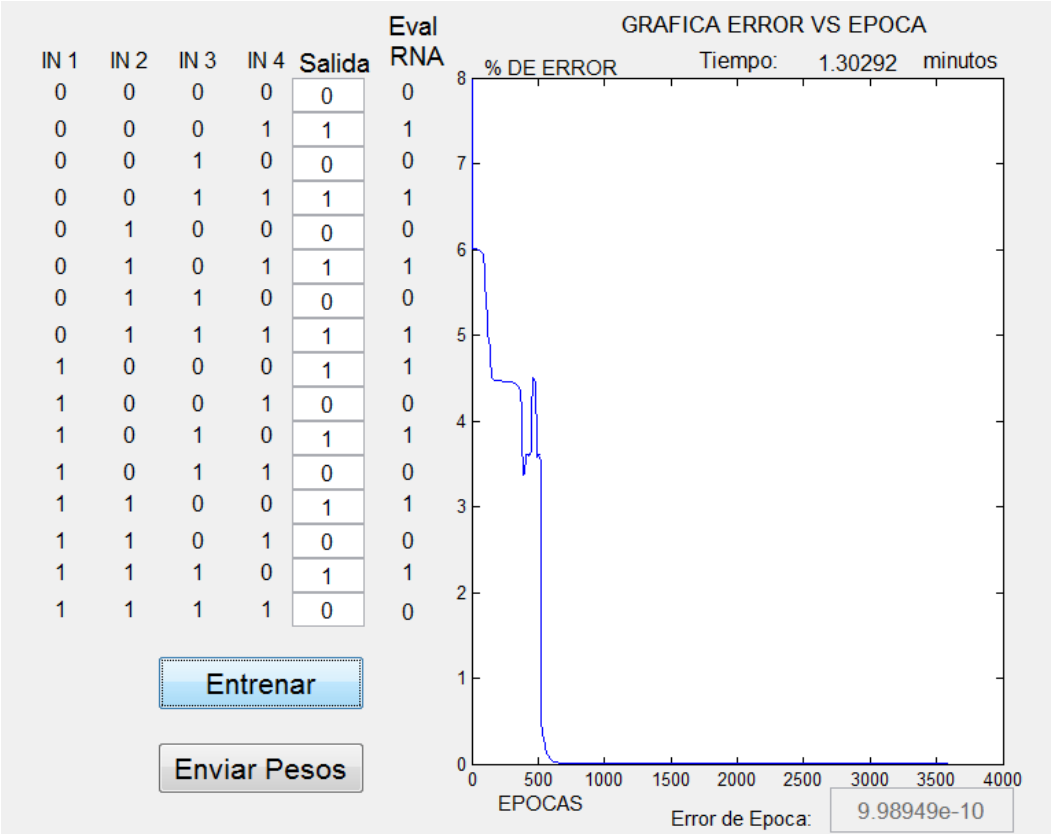


Figura 55. Respuesta a la función lógica F5.

En la Figura 56, se observa el entrenamiento de la RNA para la función lógica F6, los valores de arranque de los pesos generan un 8% de error inicial y gran parte de la convergencia ocurre hasta la época 600, 1200 épocas más tarde alcanza la convergencia total. No existen oscilaciones ni estancamientos en el transcurso del entrenamiento.

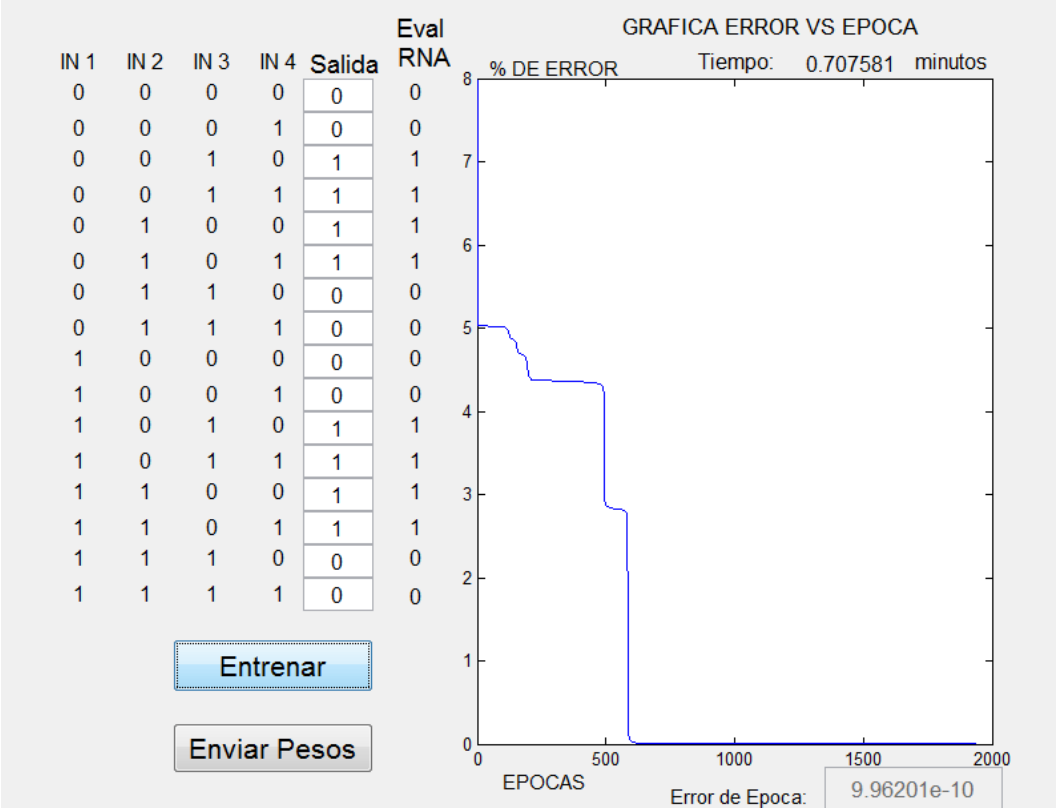


Figura 56. Respuesta a la función lógica F6.

En la Figura 57, se observa el entrenamiento de la RNA para la función lógica F7, los valores de arranque de los pesos generan un 4% de error inicial y gran parte de la convergencia ocurre hasta la época 700, 2000 épocas más tarde alcanza la convergencia total. Se tiene una pequeña oscilación entre la época 500 y 600, no existen estancamientos es decir no se observan valores de error constantes en el transcurso del entrenamiento.

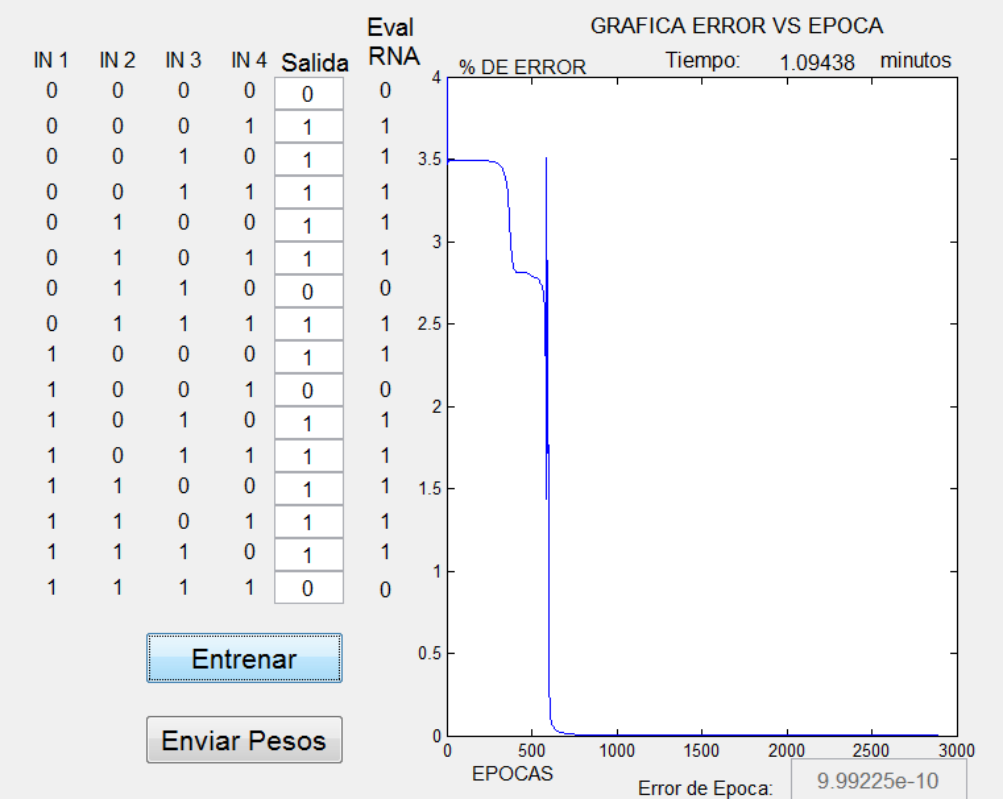


Figura 57. Respuesta a la función lógica F7.

En la Figura 58, se observa el entrenamiento de la RNA para la función lógica F8, los valores de arranque de los pesos generan un 4% de error inicial y gran parte de la convergencia ocurre hasta la época 1800, 3100 épocas más tarde alcanza la convergencia total. Se tiene una pequeña oscilación entre la época 1700 y 1800, no existen estancamientos es decir no se observan valores de error constantes en el transcurso del entrenamiento.

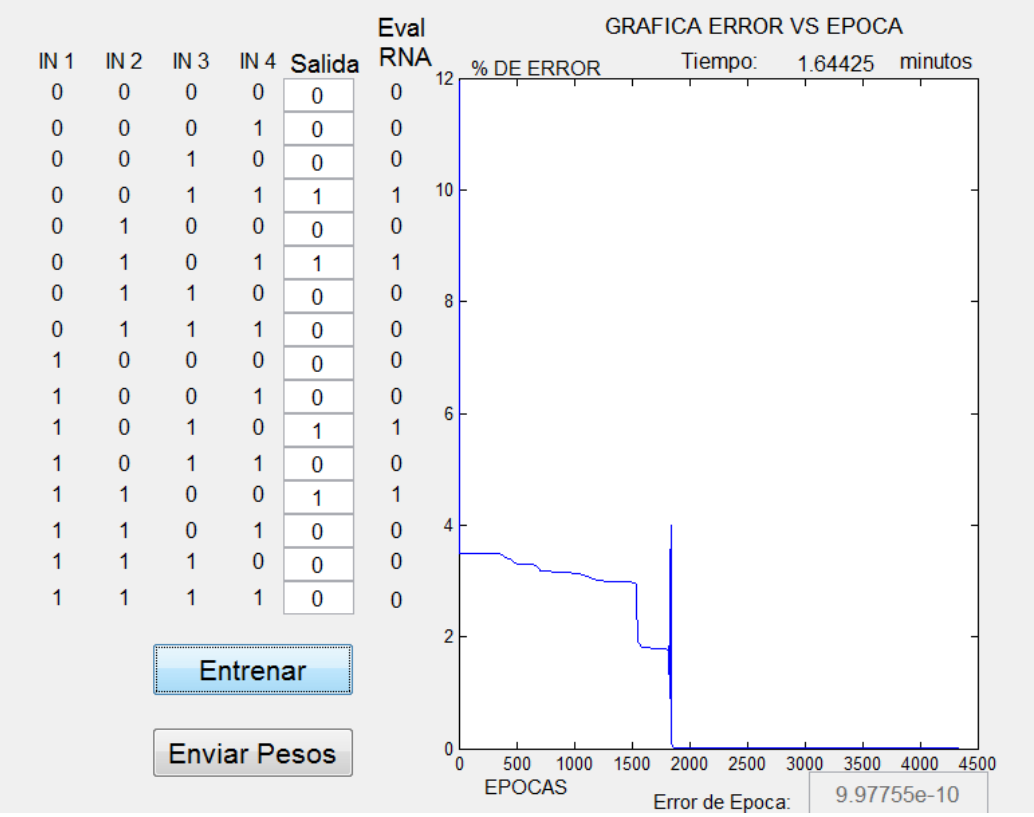


Figura 58. Respuesta a la función lógica F8.

En la Figura 59, se observa el entrenamiento de la RNA para la función lógica F9, los valores de arranque de los pesos generan un 2% de error inicial y gran parte de la convergencia ocurre hasta la época 21000, 1500 épocas más tarde alcanza la convergencia total. Se tiene una pequeña oscilación entre la época 20000 y 20500. Se observa un error constante con una duración de aproximadamente 15000 épocas, claramente el algoritmo no sería capaz de salir de este mínimo local, sin embargo al alcanzar las 20000 épocas el algoritmo asume un estancamiento y por lo tanto reinicia los valores de arranque de los pesos con otros valores diferentes, con los cuales se logra alcanzar una convergencia total.

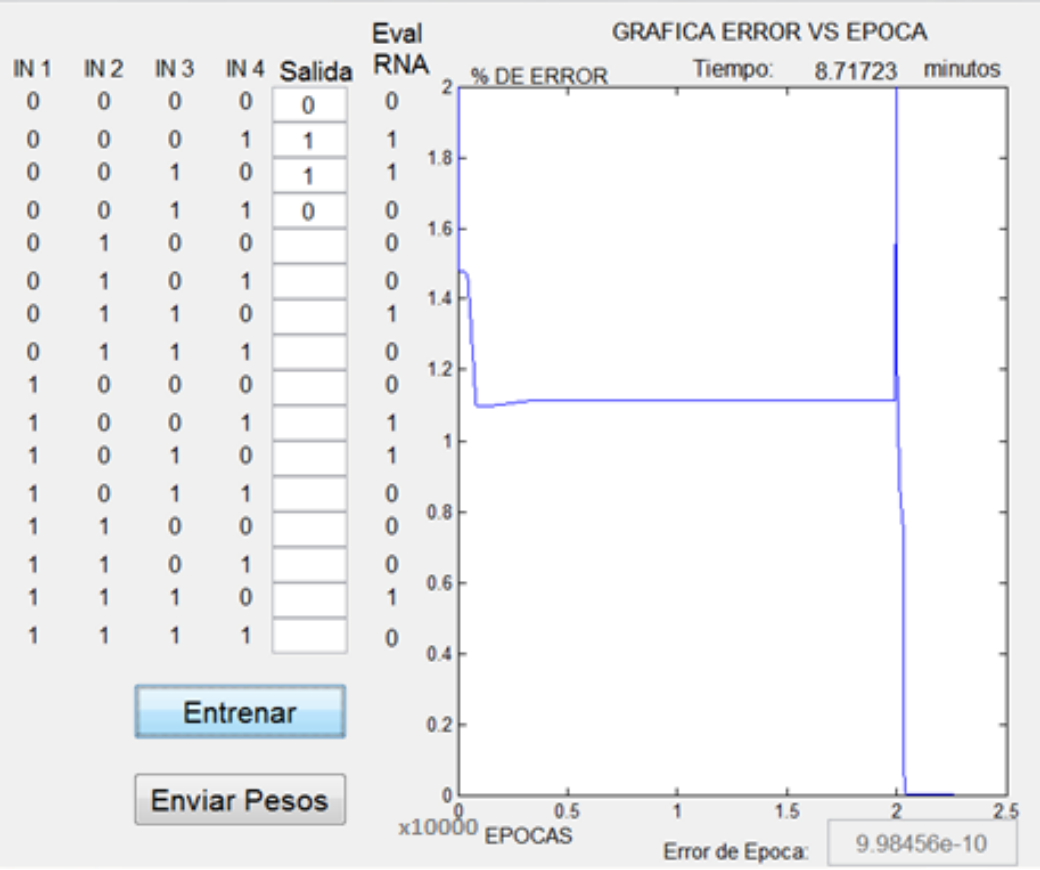


Figura 59. Respuesta a la función lógica F9.

En la Figura 60, se observa el entrenamiento de la RNA para la función lógica F10, los valores de arranque de los pesos generan un 14% de error inicial y gran parte de la convergencia ocurre hasta la época 1800, 3100 épocas más tarde se alcanza la convergencia. No se observan oscilaciones considerables, se observa un error constante en las 1300 épocas iniciales, sin embargo no representa un gran problema ya que 3600 épocas más tarde se alcanza la convergencia total.

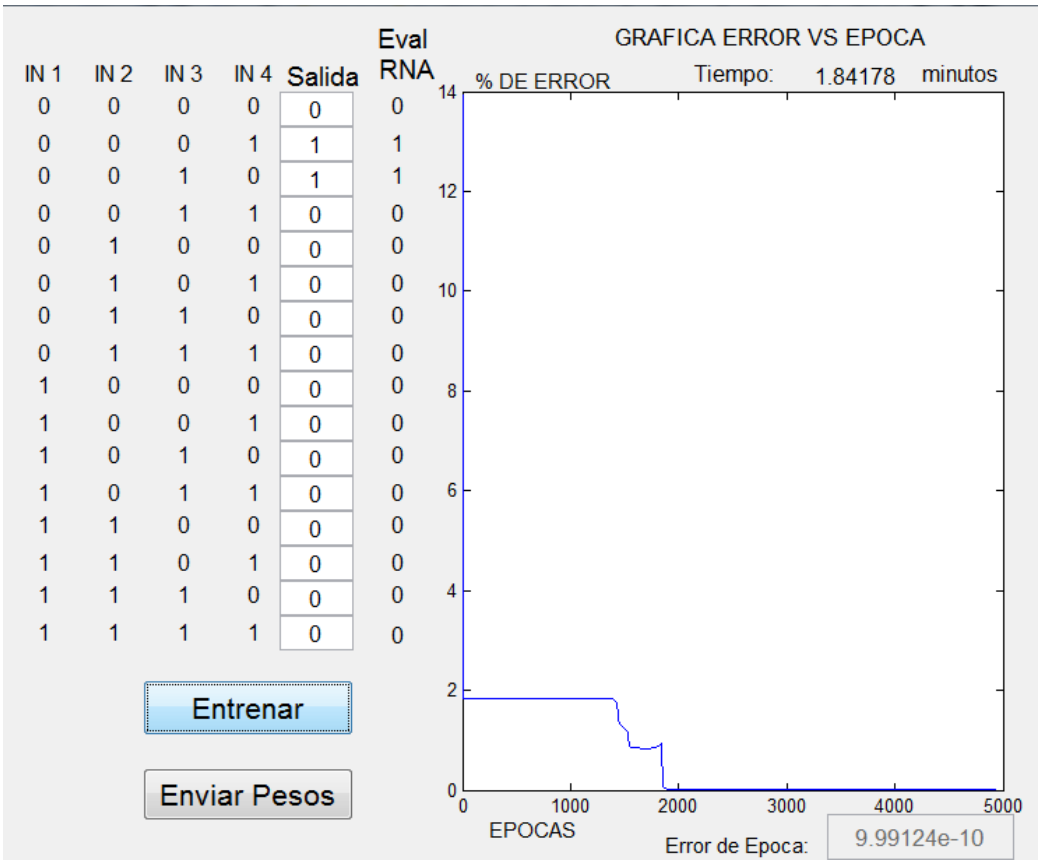


Figura 60. Respuesta a la función lógica F10.

En la Figura 61, se observa el entrenamiento de la RNA para la función lógica F11, los valores de arranque de los pesos generan un 1.9% de error inicial y gran parte de la convergencia ocurre hasta la época 1300, 2200 épocas más tarde se alcanza la convergencia total. Se observa una pequeña oscilación, un error constante en las 500 épocas iniciales, no representa ningún problema ya que en la época 3500 se alcanza la convergencia total.

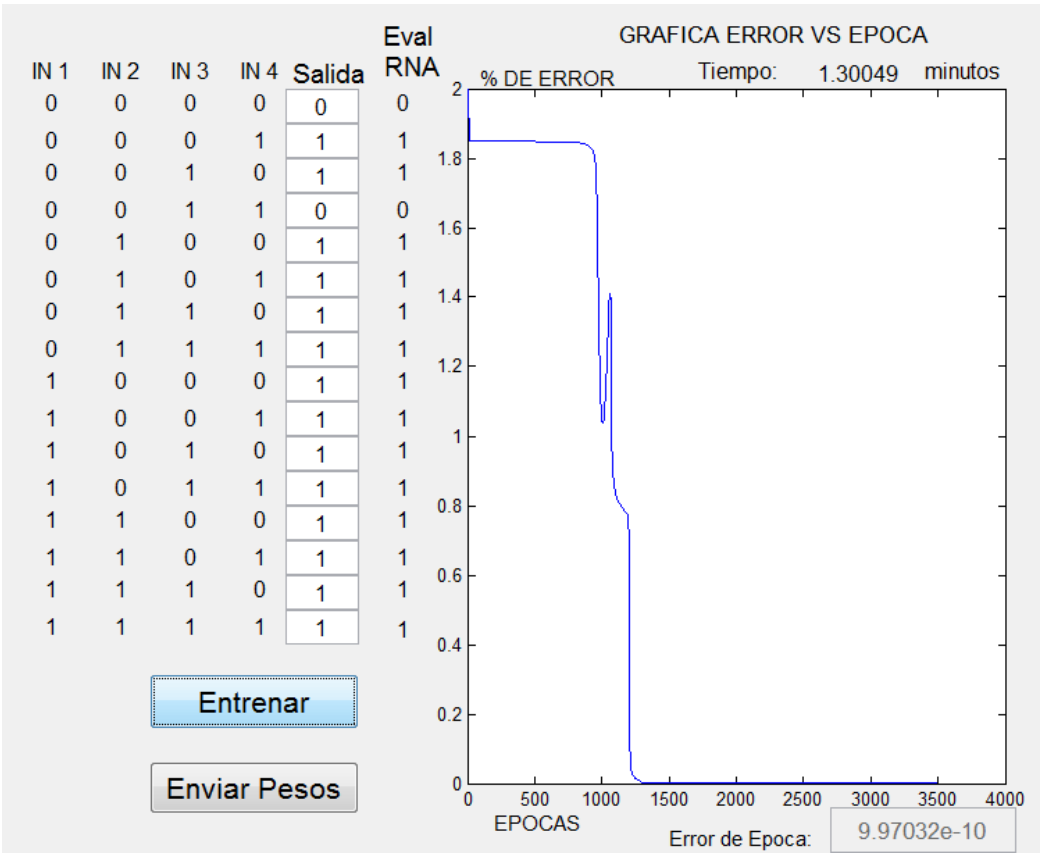


Figura 61. Respuesta a la función lógica F11.

En la Figura 62, se observa el entrenamiento de la RNA para la función lógica F12, los valores de arranque de los pesos generan un 2% de error inicial y gran parte de la convergencia ocurre hasta la época 22000, 1000 épocas más tarde alcanza la convergencia total. Se tiene una pequeña oscilación entre la época 20000 y 20500. Se observa un error constante con una duración de aproximadamente 16000 épocas, claramente el algoritmo no sería capaz de salir de este mínimo local, sin embargo al alcanzar las 20000 épocas el algoritmo asume un estancamiento y por lo tanto reinicia los valores de arranque de los pesos con otros valores diferentes, con los cuales se logra alcanzar una convergencia total en la época 22500.

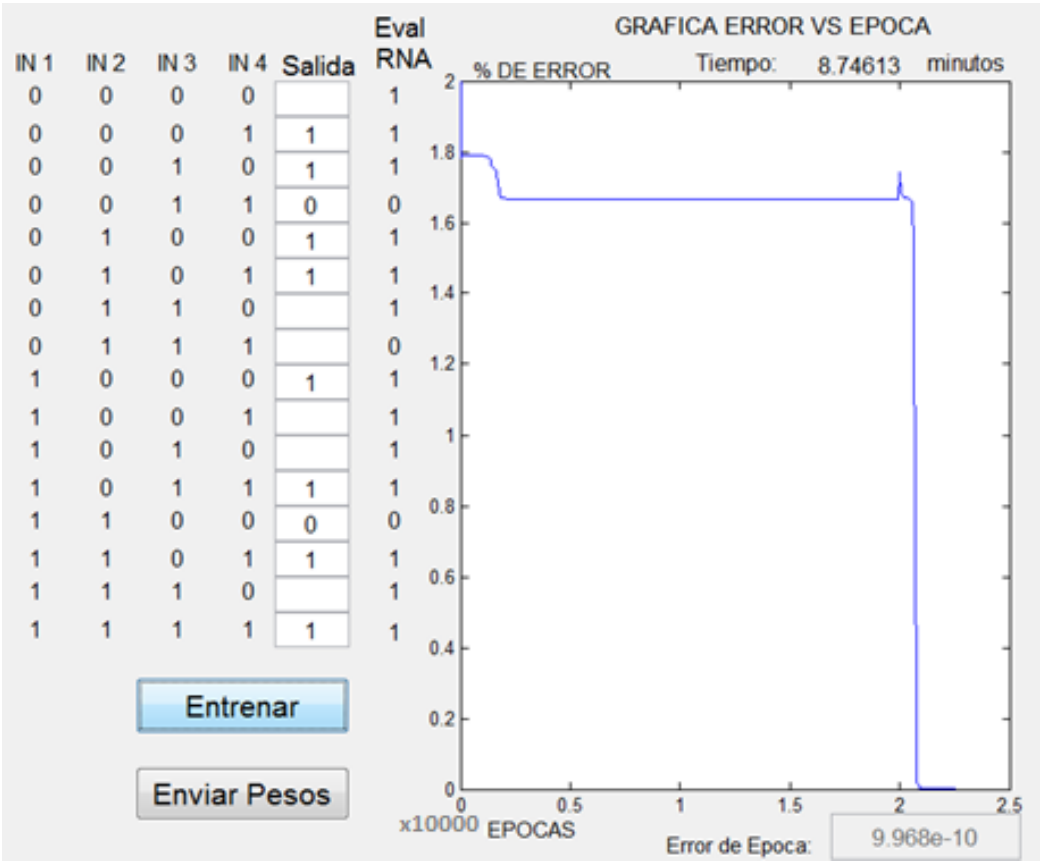


Figura 62. Respuesta a la función lógica F12.

En la Figura 63, se observa el entrenamiento de la RNA para la función lógica F13, los valores de arranque de los pesos generan un 1% de error inicial y gran parte de la convergencia ocurre hasta la época 900, 2200 épocas más tarde se alcanza la convergencia total. Se observa una pequeña oscilación entre la época 0 y 100, un error constante entre las épocas 100 y 300, no representa ningún problema ya que en la época 3100 se alcanza la convergencia total.

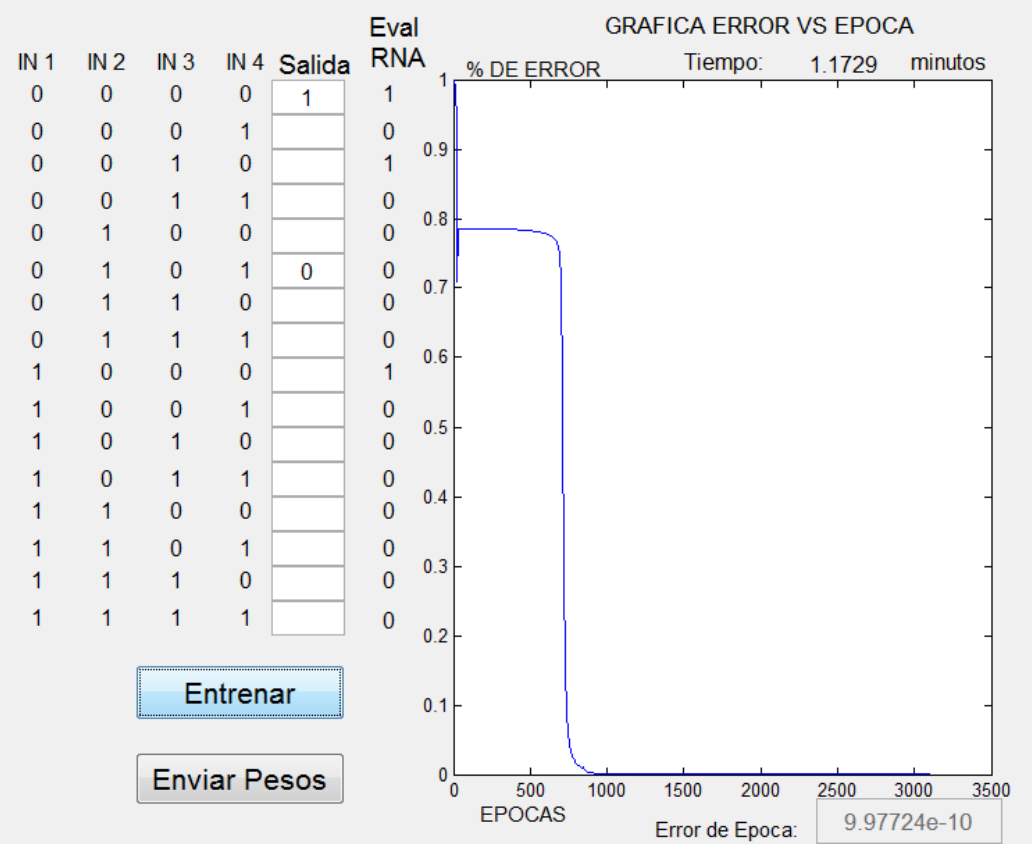


Figura 63. Respuesta a la función lógica F13.

En la Figura 64, se observa el entrenamiento de la RNA para la función lógica F14, los valores de arranque de los pesos generan un error inicial mínimo y la convergencia total ocurre hasta la época 250. No se observa ninguna oscilación, el error es decreciente en el transcurso del entrenamiento.

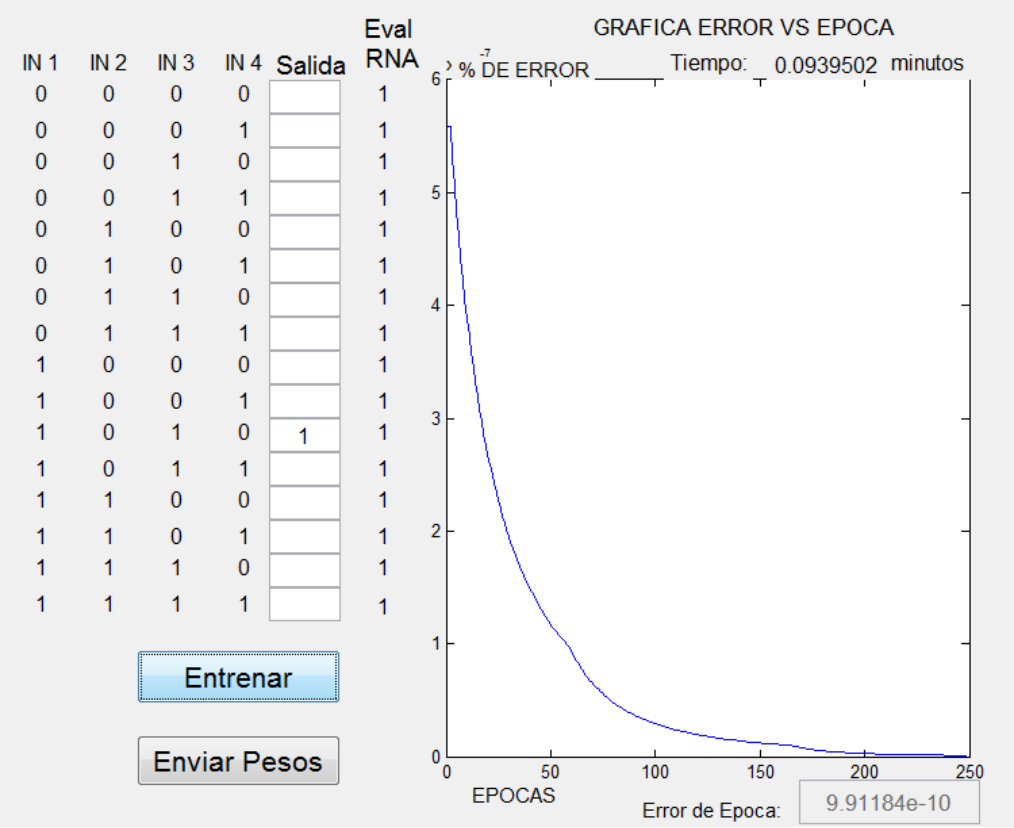


Figura 64. Respuesta a la función lógica F14.

En la Figura 65, se observa el entrenamiento de la RNA para la función lógica F15, los valores de arranque de los pesos generan un error inicial mínimo y la convergencia total ocurre hasta la época 140. No se observa ninguna oscilación, el error es decreciente en el transcurso del entrenamiento.

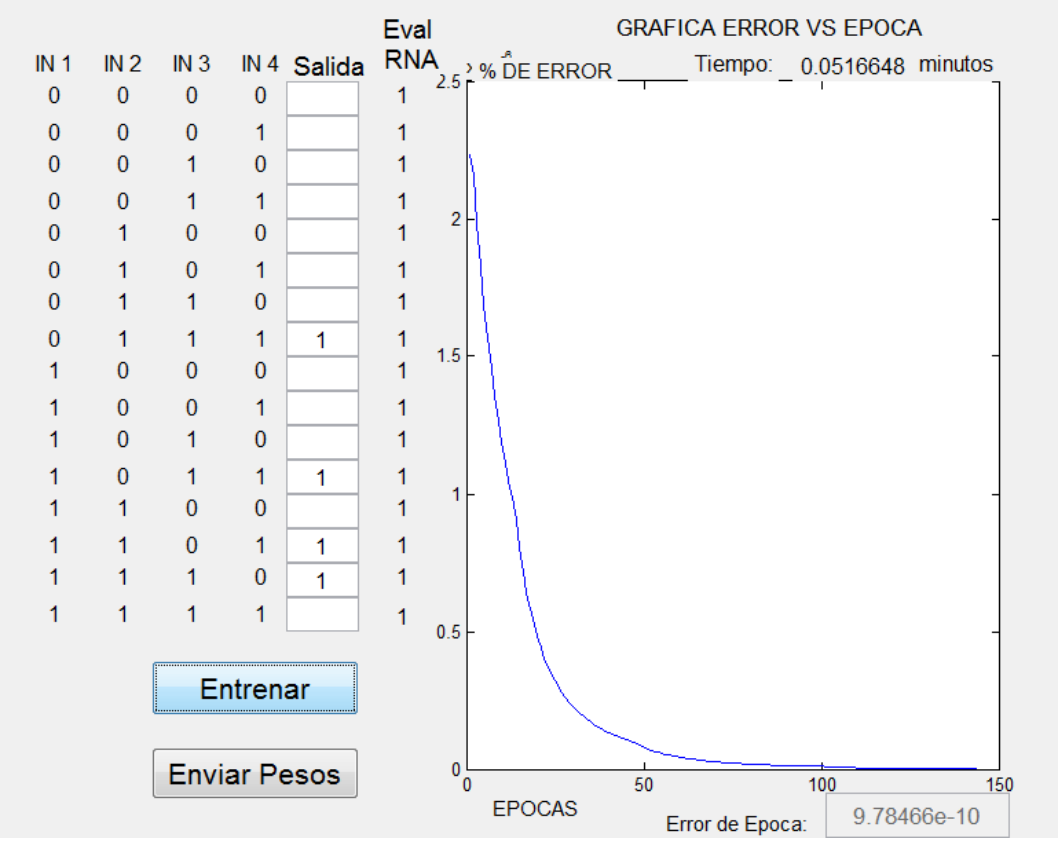


Figura 65. Respuesta a la función lógica F15.

En la Figura 66, se observa el entrenamiento de la RNA para la función lógica F16, los valores de arranque de los pesos generan un error inicial del 4%, sin embargo la convergencia es casi inmediata produciéndose en las 20 primeras épocas, en las siguientes 200 épocas se alcanza la convergencia total. No se observa ninguna oscilación, el error es decreciente en el transcurso del entrenamiento.

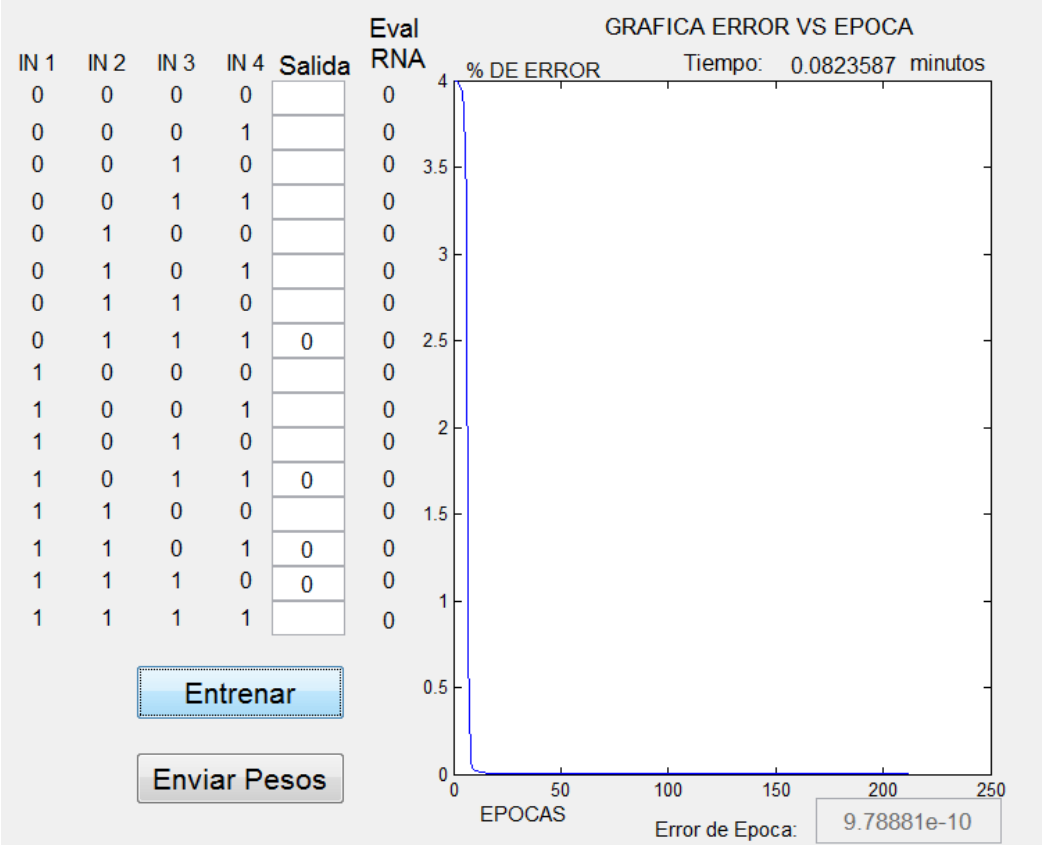


Figura 66. Respuesta a la función lógica F16.

Al realizar la transferencia de los parámetros de funcionamiento de la RNA y realizar las respectivas pruebas en la tarjeta se pudo verificar que la respuesta esperada es la adecuada, en otras palabras la FPGA respondió ante las entradas in1, in2, in3, in4, con una respuesta correspondiente a la función en estudio es decir a cualquiera de las funciones desde F1 hasta F16, y otras.

4.3. Análisis

Las funciones ante las cuales fue expuesta la RNA tomaron un tiempo de entrenamiento y una cantidad de épocas, dichos datos se muestran en la Tabla 20.

Tabla 20.

Rendimiento de las funciones de los escenarios de evaluación.

Función	Tiempo de entrenamiento		Núm. Épocas
F1	1.95	minutos	5400
F2	1.91	minutos	5200
F3	1.59	minutos	4300
F4	1.93	minutos	5250
F5	1.30	minutos	3600
F6	0.70	minutos	1900
F7	1.09	minutos	2900
F8	1.64	minutos	4400
F9	8.71	minutos	23000
F10	1.84	minutos	4900
F11	1.30	minutos	3500
F12	8.74	minutos	23000
F13	1.17	minutos	3100
F14	0.09	minutos	250
F15	0.05	minutos	140
F16	0.08	minutos	210

Al observar los tiempos de entrenamiento de la funciones F14 a F16, se pude determinar que el entrenamiento responde de manera rápida a funciones que requieran solamente valores lógicos de ‘0’ o solamente valores lógicos de ‘1’.

Las funciones F9 y F12 son las que más tiempo demandaron en el entrenamiento. Se observa una línea constante en las gráficas ilustradas en la Figura 59 y Figura 62, esto indica claramente que el algoritmo cayó en un mínimo local, sin posibilidad de salir de él. Sin embargo sale del estancamiento en la época 20 000,

esto se debe a que el algoritmo implementado tiene un condicionamiento para sacar a las funciones que caigan en mínimos locales cuando no se haya cumplido la convergencia al mínimo global en una cantidad de 20 000 épocas. La forma de sacar al algoritmo del mínimo local es reiniciando los valores de los pesos pero con diferentes valores al del entrenamiento inicial.

La función F6 a pesar de ser un Or exclusivo es resuelta de manera relativamente rápida. Todas las demás funciones son resueltas d forma rápida y eficaz según se observa en sus correspondientes gráficas.

Se puede observar de acuerdo a las gráficas generadas, que el aprendizaje requirió una cantidad diferente de etapas de entrenamiento para alcanzar la convergencia al error mínimo global. Se puede afirmar entonces que el tipo de respuesta requerido puede resultar en un aprendizaje más o menos complejo, es decir que el aprendizaje depende también del tipo comportamiento que se espera de una RNA.

Una vez realizada la transferencia de los pesos, y debido a que éstos han sido calculados y transmitidos adecuadamente, la RNA respondió satisfactoriamente en relación a la respuesta esperada. Teniendo en cuenta que la respuesta que la RNA proporciona está basada en procedimientos completamente matemáticos, al estar correctamente diseñada e implementada se espera siempre una reacción de acuerdo a los pesos que se le envían, por lo tanto un buen entrenamiento con una adecuada convergencia al mínimo global garantiza un funcionamiento adecuado de la RNA implementada en la FPGA.

CAPÍTULO V: CONCLUSIONES Y RECOMENDACIONES

5.1. Conclusiones

Al utilizar la metodología de diseño Top-Down, subdividiendo todo el sistema en los módulos, Recepción, Acople de Lógica y RNA, se logró tener un mejor entendimiento de los procesos, porque permite englobar todo el diseño y particularizar en diseños más pequeños. Además estos módulos también fueron subdivididos en otros, permitiendo enfocarse de manera sistemática en cada una de las etapas del proyecto.

La Recepción serial, es un proceso en donde se involucra un conjunto de datos, cuyo principio de organización está relacionado con el tiempo. La máquina de estados resultó proveer uno de los mejores mecanismos para la recepción serial de los pesos de la RNA, debido a que tiene la particularidad de presentar su respuesta en relación de dependencia no solo a las entradas actuales sino también a las entradas previas al instante actual, permitiendo así recibir y organizar un conjunto de datos diacrónicos para convertirlos en información.

La pendiente de la función “sigmoidea” conocida como factor de estiramiento tiene una enorme influencia al momento de realizar el entrenamiento de la RNA, tras haber realizado varias pruebas de entrenamiento se ha observado que el valor más adecuado es 4. Una pendiente muy elevada en la función sigmoidea hace que ésta pierda paulatinamente su característica analógica, lo cual afecta al entrenamiento de la RNA, ya que las respuestas de evaluación se verán limitadas a un rango más

restringido, dificultando así un ajuste fino de los pesos y por lo tanto provocando oscilaciones sucesivas en torno a un mínimo local. Por otra parte si la pendiente es muy baja, la característica analógica de la función sigmoidea incrementa su rango con la relación a las respuestas de evaluación de la RNA, generando así un ajuste extremadamente fino, lo cual puede incrementar exponencialmente el tiempo de entrenamiento de la RNA.

El factor de aprendizaje es un valor que se encuentra en un rango entre 0 y 1, que permite atenuar la magnitud de cambio de la variación de los pesos en el proceso de aprendizaje de la RNA. La variación de los pesos es directamente proporcional a la magnitud del error de época, por lo tanto un error de época pequeño debe atenuarse en menor magnitud que un error de época grande, entonces es importante incrementar el factor de aprendizaje cuando el error de época llega a tornarse muy pequeño, para que no se vea disminuida la velocidad de convergencia del error a un mínimo global.

Dos o más combinaciones de entradas iguales con diferentes salidas harán que la red neuronal no alcance nunca la convergencia del error a un mínimo global, ya que existiría una incongruencia imposible de resolver ya que la red se encontraría frente a una contradicción lógica. Por eso se debe limitar al usuario para que esté no pueda cometer este tipo de errores.

Disminuir el número de neuronas con respecto al número de entradas hace que la RNA paulatinamente pierda su capacidad para resolver funciones que no son linealmente separables. En una RNA la cantidad de neuronas de la capa de entrada debe ser igual a la cantidad de entradas a dicha red, para otorgar a la RNA la capacidad de resolver tanto funciones linealmente separables como funciones linealmente no separables.

5.2. Recomendaciones

Es indispensable plasmar en un diagrama de bloques la idea fundamental y general, que ilustre con claridad el propósito del diseño de la maquinaria de funcionamiento del proyecto, con el objetivo de tener un punto de partida desde lo general hacia lo particular, para proyectar el diseño a una metodología Top Down.

En el proceso de diseño se recomienda que cada uno de sus componentes sea lo más genérico posible para que su flexibilidad le permita adaptarse a cambios imprevistos en el proceso de diseño e implementación.

En el proceso de implementación es recomendable enfocarse de manera sistemática en cada uno de los componentes del diseño, guiándose por las características teóricas plasmadas en dicho diseño.

La interfaz humano máquina, debe ser implementada de tal forma que los errores humanos sean restringidos en la mayor medida posible, en este proyecto se restringe al usuario ingresar las entradas a la RNA, en su lugar se muestran las entradas ya generadas para que el usuario solamente tenga que ingresar las respuestas esperadas. Entradas iguales con diferentes respuestas harían que el software de entrenamiento no pueda encontrar la convergencia del error al mínimo global.

Se deben tener claros los protocolos de comunicación para interpretar correctamente la distribución y organización de la transferencia de datos, con el objetivo de evitar errores y pérdidas de tiempo en la implementación del diseño.

Para trabajos futuros se puede considerar todo el bloque de RNA como una sola neurona e interconectarlo en base a determinada arquitectura, en cuyo caso habría que otorgarle a cada RNA una dirección para que se diferencie de las demás. Para que los tiempos de entrenamiento no se eleven exponencialmente, es importante que la nueva arquitectura sea de una sola capa, permitiendo conocer las entradas y salidas de cada RNA, y por ende realizar entrenamientos sucesivos de manera individual para cada uno de los bloques de neuronas. Se podría utilizar el mismo software de entrenamiento con algunas modificaciones.

Para trabajos futuros se puede incorporar otro modulo después del módulo de recepción para incrementar el valor de los números a recibir según se requiera. El sistema de recepción serial de este proyecto está en la capacidad de recibir números empaquetados en 11 bits, con un bit adicional para el signo, es decir el módulo de recepción puede recibir números en un rango de -2047 hasta +2047.

Bibliografía

Dharmendra S. Modha. (7 de Agosto de 2014). *IBM Research.* Recuperado el 23 de Febrero de 2015, de Introducing a Brain-inspired Computer: http://www.research.ibm.com/articles/brain-chip.shtml

Fundación Wikimedia Inc. (26 de Febrero de 2015). *WIKIPEDIA*. Recuperado el 28 de Febrero de 2015, de Red neuronal artificial: http://es.wikipedia.org/wiki/Red_neuronal_artificial

IBM Corporation. (9 de Octubre de 2014). *IBM*. Recuperado el 20 de Febrero de 2015, de IBM Research: http://research.ibm.com/cognitive-computing/neurosynaptic-chips.shtml#fbid=HWl38t5J4f6

Introducción a las redes neuronales aplicadas. (20 de 10 de 2007). Obtenido de Introducción a las redes neuronales aplicadas: http://halweb.uc3m.es/esp/Personal/personas/jmmarin/esp/DM/tema3dm.pdf

Maxines, D. G., & Alcalá, J. (2002). *VHDL El arte de programar sistemas digitales.* Mexico: Continental.

Jorge A. Quiñones R. (2011). Diseño de controladores digitales en FPGA para aplicaciones de realidad virtual. (Tesis de Maestría). Centro de Investigación y Desarrollo de Tecnología Digital. Instituto Politécnico Nacional. Tijuana, B.C., México.

FECHA DE ENTREGA

El proyecto fue entregado al departamento de Eléctrica y Electrónica y reposa en la Universidad de las Fuerzas Armadas – ESPE, desde:

Sangolquí, 21 DE ABRIL DE 2015

ELABORADO POR:

Jorge Luis Ramírez Torres
CI: 1104088255

AUTORIDADES:

Ing. Luis Orozco, MSc.
DIRECTOR DE LA CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL.