Universidad Autónoma Metropolitana
Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería
Ingeniería en Computación

Reporte parcial de proyecto terminal

Implementación de una red neuronal booleana multicapa en un FPGA

Alumno:

Juan Pablo Hernández Castillo
208303519

Trimestre 14 Otoño
12 enero de 2015

Asesores:

Oscar Alvarado Nava
Eduardo Rodríguez Martínez

Yo, **Oscar Alvarado Nava**, declaro que aprobé el contenido del presente Reporte de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de la UAM Azcapotzalco.

Yo, **Eduardo Rodríguez Martínez**, declaro que aprobé el contenido del presente Reporte de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de la UAM Azcapotzalco.

Rodriguez Martinez E.

Yo, **Juan Pablo Hernández Castillo**, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Matropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de la UAM Azcapotzalco.

Resumen

En el presente trabajo podremos llevar acabo la creación y realizar el proceso de aprendizaje de lo que es una red neuronal, está tendrá el objetivo de aprender las funciones booleanas (NAND y XOR) con el fin de incrementar los tiempos de respuestas.

En el trabajo se puede explicar de forma modular cada uno de los componentes del *perceptrón*. Después podremos realizar las diferentes arquitecturas para realizar el aprendizaje de cada una de las funciones.

Tabla de contenidos

1. Introducción **5**
1.1. Introducción . . . 5
1.2. Justificación . . . 5
1.3. Objetivo general . . . 6
1.4. Objetivo específico . . . 6
1.5. Organización del proyecto . . . 7

2. Marco Teórico **8**
2.1. ¿Qué es una red neuronal? . . . 8
2.2. Funciones de activación . . . 9
2.3. Arquitecturas . . . 11
2.4. Perceptron . . . 12
2.5. Algoritmo de back-propagation . . . 14
2.6. Problemas no lineales . . . 16
2.7. FPGA . . . 17
2.8. Máquina de estados . . . 18
2.9. Uso de la maquina de estados . . . 18

3. Desarrollo de las redes neuronales **20**
3.1. Descripción técnica . . . 20
3.1.1. Alto . . . 20
3.1.2. Mayor . . . 23
3.1.3. Comparador . . . 24
3.2. Especificación técnica . . . 25
3.2.1. Arquitectura e implementación de la función NAND . . . 25
3.2.2. Arquitectura e implementación de la función XOR . . . 26

4. Pruebas y resultados **28**
4.1. Pruebas arquitectura NAND . . . 28
4.2. Pruebas arquitectura XOR . . . 28

5. Conclusiones **30**

Bibliografía **31**

Apéndices **33**

A. Apéndice de tablas **34**
A.1. Prueba de escritorio NAND 34
A.2. Números pseudo-aleatorios 35

B. Apéndice de códigos **39**
B.1. Código Alto 39
B.2. Código aleatorio 41
B.3. Código neurona 42
B.4. Código Mayor 45
B.5. Código Comparador (NAND) 46
B.6. Código Comparador (XOR) 48
B.7. Código perceptrón (NAND) 50
B.8. Código aprende (XOR) 52
B.9. Código perceptrón (NAND) 54
B.10.Código aprende (XOR) 56

Capítulo 1

Introducción

1.1. Introducción

Las redes neuronales artificiales son una propuesta tecnológica en la programación pare el aprendizaje y procesamiento de información. Dichas redes estás basadas en sistemas neuronales de animales,siendo esto un conjunto de redes interconectadas capaces de trabajar entre ellas para producir una salida.

A lo largo de la red neuronal se presentan estructuras conocidas como *perceptron* los cuales se conectan para formar una interconexión, donde las entradas de la capa siguiente, son las salidas de la capa anterior[2].

El perceptrón de la Figura 1.1 intenta modelar el comportamiento de la neurona biológica. Aquí el cuerpo de la neurona se representa como un sumador lineal de los estímulos externos X_j, seguida de una función no lineal $Y_j = f(z_j)$. La función $f(z_j)$ es llamada la función de activación, y es la función que utiliza la suma de estímulos para determinar la actividad de salida de la neurona. En las redes neuronales booleanas se utilizan elementos de lógica booleana como componentes básicos, los cuales no están estructuralmente ligados a funcionamientos observados en la naturaleza[4].

Actualmente se usan las redes neuronales son usadas para uso en aplicaciones para inteligencia artificial, en hardware que cuente con poco recurso y este requiera realizar una tarea en especifico.

Por otra parte un FPGA[1] es un dispositivo que contiene bloques lógicos programables cuya interconexión y funcionalidad pueden ser programados por medio de un lenguaje de descripción de hardware. El uso de dichos dispositivos se puede ubicar en sistemas de comunicaciones, procesamiento de datos o la industria.

1.2. Justificación

La implementación de una red neuronal puede realizarse tanto en hardware o en software. La implementación de las redes neuronales en software, implica una serie de desventajas, como son:

[1] Acrónimo en inglés *Field Programmble Gate Array*, dispositivo programable en campo.

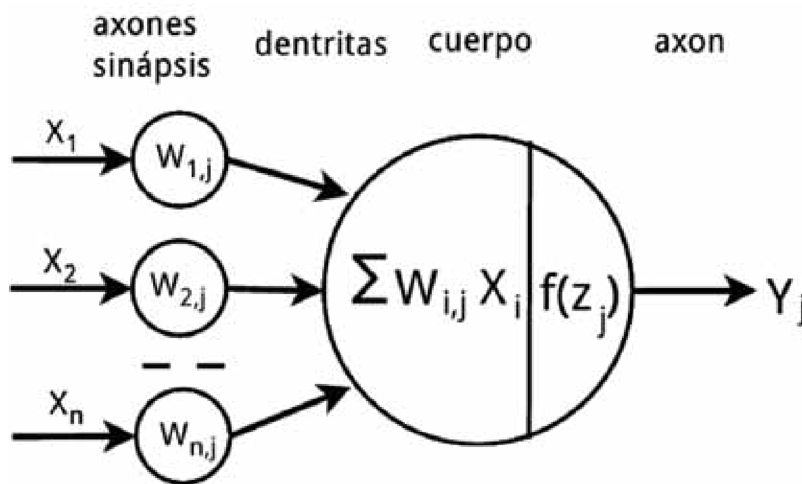

Figura 1.1: Representación de un *perceptron*.

- Bajo rendimiento debido a un alto costo de procesamiento para el entrenamiento de la red.

La implementación de una red neuronal en circuitos digitales tiene las siguientes ventajas:

- No es necesario conocer la estructura física de los componentes básicos.
- Tolerante a variaciones en las propiedades individuales del dispositivo.
- Puede ser fácilmente escalable.
- Puede ser integrado en sistema digital existente.

Actualmente los FPGA ofrecen una la alta densidad de bloques lógicos configurables, los cuales pueden ser reconfigurados en tiempo de ejecución, característica muy útil para la implementar el crecimiento de una red neuronal.

1.3. Objetivo general

Diseñar, describir en un HDL[2] e implementar en un FPGA una red neuronal *booleana* multicapa que sea capaz de realizar de aprender las funciones lógicas NAND y XOR.

1.4. Objetivo específico

- Diseñar una neurona.

[2]Acrónimo en inglés *Hardware Description Language*, lenguaje de descripción de hardware.

- Diseñar un circuito de números pseudo-aleatorios.
- Diseñar un *perceptron* con la arquitectura para el aprendizaje de la función NAND.
- Realizar el aprendizaje con la máquina de estados *booleana* de la función NAND.
- Diseñar un *perceptron* con la arquitectura para el aprendizaje de la función XOR.
- Realizar el aprendizaje con la máquina de estados *booleana* de la función de la función XOR.

1.5. Organización del proyecto

El presente trabajo se organiza de la siguiente manera:

- Capitulo 1 lograremos obtener un concepto general de lo que son las redes neuronales , FPGA.Así como la importancia y objetivos que se persiguen.
- Capitulo 2 se concentra en dar explicación más amplia a todos los conceptos elementales.
- Capitulo 3 se explica de forma modular el desarrollo de la red neuronal, así mismo el desarrollo de las arquitecturas.
- Capitulo 4 se muestran los resultados de las simulaciones así como el de las respectivas implementaciones.
- Capitulo 5 se dan las conclusiones con respecto al presente trabajo.

Capítulo 2

Marco Teórico

2.1. ¿Qué es una red neuronal?

Podemos definir a una red neuronal artificial como un esquema de computación distribuida inspirada en la estructura del sistema nervioso de los seres humanos. La arquitectura de una red neuronal es formada conectando múltiples procesadores elementales, siendo éste un sistema adaptativo que poseé un algoritmo para ajustar sus pesos (parámetros libres) y así alcanzar los requerimientos de desempeño del problema basado en muestras representativas[1]. En la figura 2.1 se muestra el modelo de una neurona no linear, en donde podemos identificar tres elementos principales:

- Un conjunto de sinapsis o conexiones, cada una de estas se caracteriza por tener un peso. Específicamente, una señal x_j en las entradas de sinapsis j conectada a la neurona k que es multiplicado por el peso sipnático w_{kj}.
- Un sumador para sumar las señales de salida, ponderado por los respectivos sinapsis de la neurona; la operación descrita aquí constituye una operación lineal.
- Una función de activación para limitar las amplitudes de la salida de una neurona.

En el modelo de la figura 2.1 también se incluye un *applied bias* externa, descrito por b_k. El *bias* b_k tiene el efecto de incrementarse o bajar la entrada neta de la función de activación dependiendo si este tiene un valor positivo o negativo, respectivamente.

En términos matemáticos, podemos expresar una neurona k como:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j$$

y

$$y_k = \varphi(u_k + b_k)$$

Donde x_1, x_2, ..., x_m son las señales de entrada; w_{k1}, w_{k2}, ..., w_{km} son los pesos sinápticos de la neurona k; u_k es la combinación linear de salidas debido a las señales de entradas de la neurona;

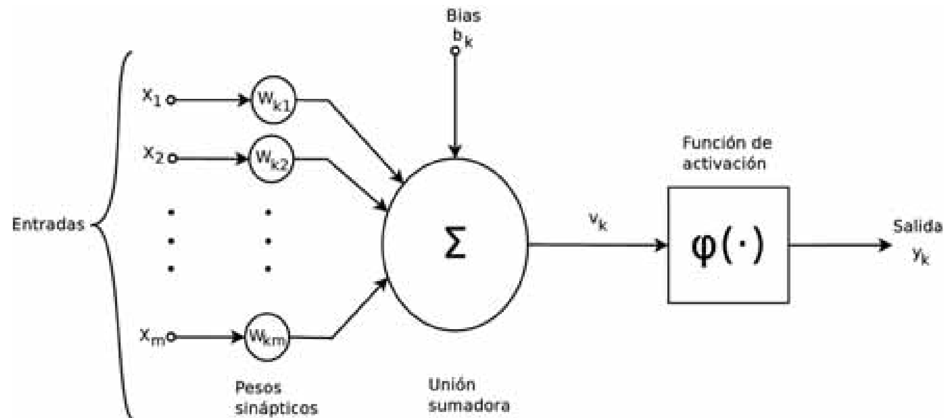

Figura 2.1: Modelo no linear de la neurona.

b_k es la *bias*; $\varphi(\cdot)$ es la función de activación; y_k es la salida de la neurona. El uso del *bias* tiene un efecto de transformación sobre u_k:

$$v_k = u_k + b_k$$

En particular, depende si la *bias* es positiva o negativa, la relación entre el *induced local field* o la activación v_k de la neurona k y la salida del combinador lineal u_k es modificado.

El *bias* b_k es un parámetro externo de la neurona k.Tomando en cuenta su presencia en la expresión de la neurona de la siguiente manera:

$$u_k = \sum_{j=0}^{m} w_{kj} x_j$$

y

$$y_k = \varphi(v_k)$$

Agregamos la nueva sinapsis:

$$x_0 = +1$$

mientras los pesos serian

$$w_{k0} = b_k$$

2.2. Funciones de activación

Podemos definir principalmente tres modelos básicos de función de activación:

1. Función *threshold*. Para este tipo de función descrita en la figura 2.2, tenemos:

$$\varphi(v) = \begin{cases} 1 \text{ si } v \geq 0 \\ 0 \text{ si } v < 0 \end{cases}$$

Correspondiendo, a las salidas de la neurona k, la función *threshold* es expresada como:

$$\varphi(v) = \begin{cases} 1 \text{ si } v_k \geq 0 \\ 0 \text{ si } v_k < 0 \end{cases}$$

Donde v_k es:

$$v_k = \sum_{j=1}^{m} w_{kj} x_j + b_k$$

En esté modelo, la salida de la neurona toma el valor de 1 si el *induced local field* de esa neurona es no negativo, y 0 en el caso contrario.

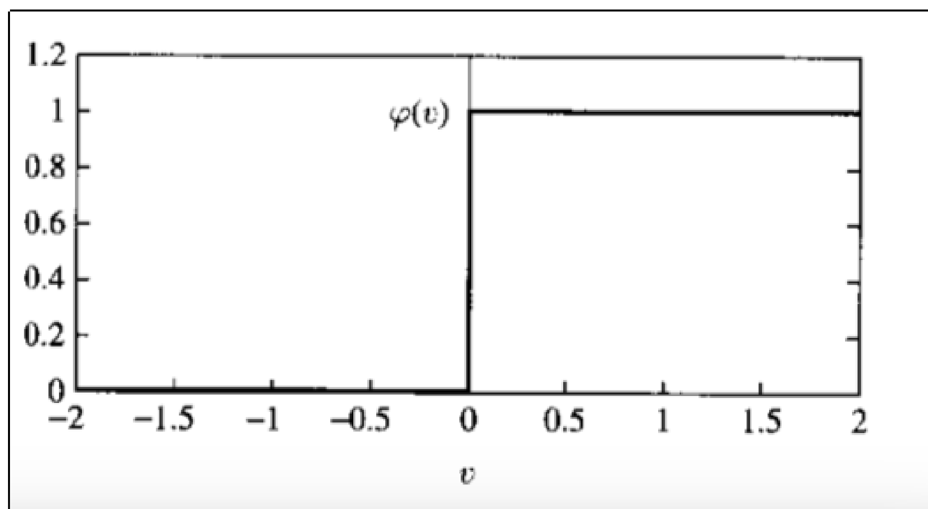

Figura 2.2: Función *threshold.*

2. Función *piecewise-linear* . La descripción de la función se encuentra en la figura 2.3, en donde tenemos:

$$\varphi(v) = \begin{cases} 1,\ v_k \geq +\frac{1}{2} \\ v,\ \frac{1}{2} > v > -\frac{1}{2} \\ 0,\ v \leq -\frac{1}{2} \end{cases}$$

Donde el factor de amplificación dentro de la región lineal de funcionamiento es asumido para ser unitario. Está forma de función de activación puede ser vista como una aproximación a un amplificador no lineal. Las dos situaciones siguientes pueden verse como una forma especial de la función *piecewise-linear.*

- Surge un combinador lineal si la región lineal de operación es mantenida sin caer en la saturación.

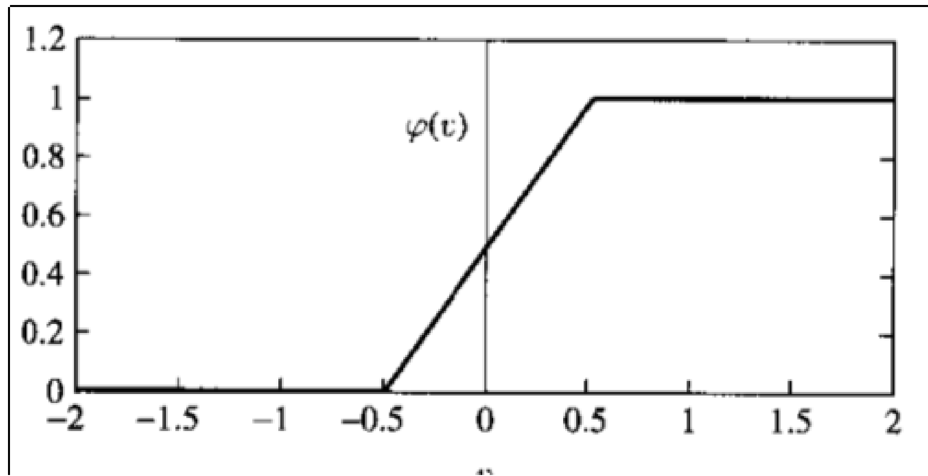

Figura 2.3: Función *piecewise-linear*

- El *piecewise-linear* se reduce a la función *threshold* si el factor de amplificación de la región lineal se vuelve infinitamente largo.

3. Función *sigmoid*. Esta función es la función de activación más usada en la construcción de redes neuronales artificiales. Está definida como una función estrictamente creciente que tiene un balance entre linear y no linear. Un ejemplo de un tipo de está función es la *logistic function*, definida por:

$$\varphi(v) = \frac{1}{1 + exp(-av)}$$

Donde a es el parámetro de la pendiente en la función *sigmoid*. Por variación del parámetro a podemos obtenerla función *sigmoid*, como se ilustra en la figura 2.4. Mientras la función *threshold* asume los valores de 0 o 1, la función *sigmoid* asume valores continuos de 0 a 1. Note también que la función *sigmoid* es diferenciable.

En algunas ocasiones deseables esta función toma los valores del rango de -1 a +1, en esos casos la función asume una forma asimétrica con respecto a la original.

2.3. Arquitecturas

La estructura de la red va depender del algoritmo de aprendizaje usado para entrenar a la red. En general podemos detectar 2 clases diferentes de estructuras.

1. De una capa. En está arquitectura las redes están organizadas con una capa de entrada de los nodos de origen y una capa de salida, correspondiente a las salidas de las neuronas. Está arquitectura se logra apreciar en la figura 2.5.

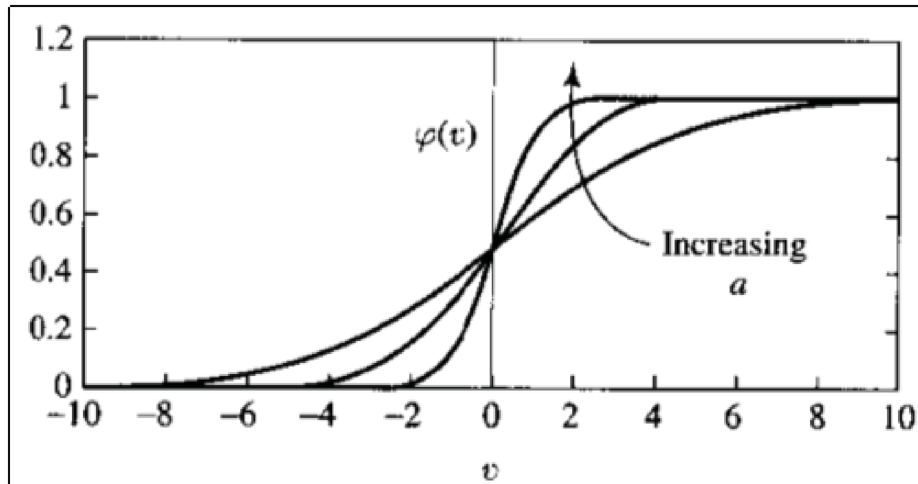

Figura 2.4: Función *sigmoid*

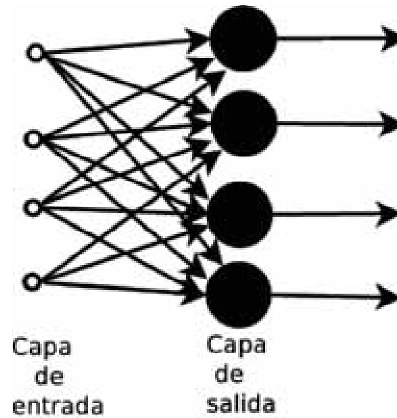

Figura 2.5: Arquitectura de una capa.

2. Multicapas. Esta arquitectura se caracteriza por tener presente una o más capas ocultas. La función de una capa oculta es intervenir entre la capa de entrada y la de salida de alguna manera útil, esto se logra apreciar en la figura 2.6.

2.4. Perceptron

El *perceptron* es construido alrededor de una neurona no lineal. El modelo neuronal consiste de un combinador lineal seguido de un *hard limiter*, como se presenta en la figura 2.7. El nodo de suma del modelo neuronal calcula una combinación linear de entradas aplicada a sus sinapsis, también incorpora el *bias*. El resultado de la suma, que es, el *induced local field*, es aplicado al *hard limiter*. En consecuencia, la neurona produce una salida igual a +1 si la entrada *hard limiter* es positiva, y -1 si es negativa.

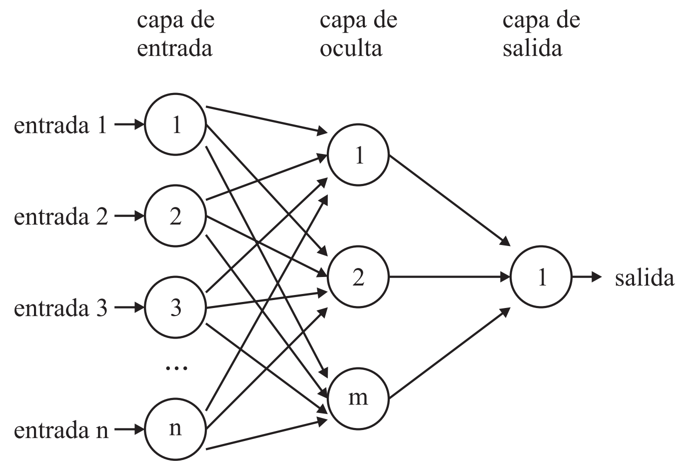

Figura 2.6: Arquitectura multicapa.

En la figura 2.7, los pesos sipnáticos del *perceptron* son denotados por w_1, w_2, ..., w_m. Correspondiendo, a las entradas del *perceptron* son denotadas por x_1, x_2, ..., x_m. La *bia* es denotada por b. Desde el modelo podemos encontrar que el *hard limiter* o *induced local field* de la neurona es:

$$v = \sum_{i=1}^{m} w_i x_i + b$$

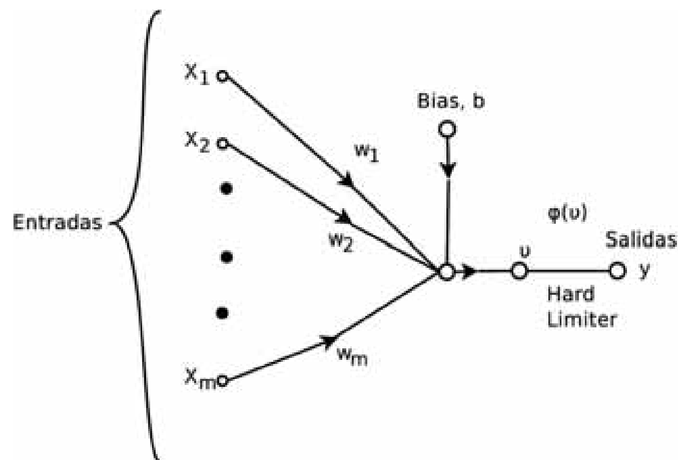

Figura 2.7: Flujo de señal del *perceptron*.

El objetivo del *perceptron* es clasificar correctamente el conjunto de estímulos aplicados externamente (x_1, x_2, ..., x_m) dentro de una o dos clases.ς_1 o ς_2. La regla de decisión para la clasificación es asignar el punto representado por los puntos x_1, x_2, ..., x_m para la clase ς_1 si la salida y del *perceptron* es +1 y para la clase ς_2 si es -1.

Para desarrollar una visión informativa sobre el comportamiento del patrón clasificador, normalmente se traza un mapa de regiones de decisión en el espacio de la señal m-dimension abarcado

por la m variables de entradas. En esta forma simple el *perceptron* tiene dos regiones de decisión separadas por un hiperplano definido por:

$$\sum_{i=1}^{m} w_i x_i + b = 0$$

En la figura 2.8 para el caso de 2 variables de entrada x_1 y x_2. Para la cual el limite de decisión toma forma de una linea recta. Un punto (x_1, x_2) que se encuentra por encima del límite se le asigna la clase ς_1 y un punto (x_1, x_2) que se encuentra por de bajo del límite se le asigna a la clase ς_2. Note también que el efecto del *bias* es solamente para cambiar frontera de decisión más lejos del origen.

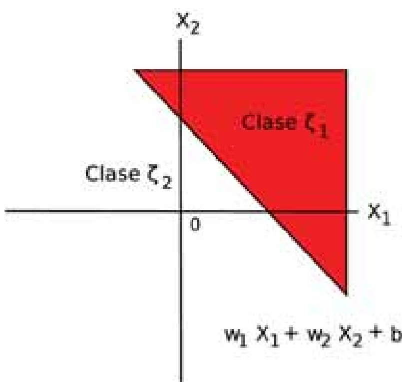

Figura 2.8: Modelo no linear de red neurona.

Los pesos sipnáticos w_1, w_2, ..., w_m pueden ser adaptados sobre las iteraciones.[3]

2.5. Algoritmo de back-propagation

El algoritmo de *back-propagation* es utilizado para el aprendizaje redes neuronales booleanas. En este algoritmo si la salida de las neuronas están mal, cada una ajusta su sinápsis o señal de la capa anterior de acuerdo a un conjunto de probabilidades[10].

Las señales de error de la salida de una neurona j con iteraciones n es definida por:

$$e_j(n) = d_j(n) - y_j(n), \text{ neurona } j \text{ es un nodo de salida}$$

Se define el valor instantáneo de la energía de error para una neurona j como $\frac{1}{2}e_j^2(n)$. Correspondiente, los valores instantáneos $\xi(n)$ del total de la energía de error es obtenida por la suma $\frac{1}{2}x_0 = +1e_j^2(n)$ sobre todas las neuronas en la capa de salida. Por lo que podemos expresarlo:

$$\xi(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n)$$

Donde es el conjunto C que incluye a todas la neuronas en la capa de salida de la red neuronal.La N denota el total de números de patrones contenidos en un entrenamiento. El *avereges squared error energy* es obtenido sumando $\xi(n)$ sobre todo n y luego la normalizamos con respecto al tamaño del conjunto de N, como se muestra a continuación.

$$\xi_{av} = \frac{1}{N}\sum_{n=1}^{N}\xi(n)$$

La energía de error instantáneo $\xi(n)$, y por lo tanto el promedio de la energía de error ξ_{av}, es una función de todos los parámetros libres de la red. Para determinado conjunto de entrenamiento,ξ_{av} representa la función de costo como una medida del rendimiento del aprendizaje. Podemos considerar un método simple de entrenamiento en el cual los pesos se actualizan en un modelo de base patrón hasta una época, que es, una presentación completa de todo el conjunto de entrenamiento ya tratado. Los ajustes de los pesos se hacen de acuerdo con los respectivo errores calculados para cada patrón presentado a la red.

Por lo tanto, la media aritmética de estos cambios de peso individuales sobre el conjunto de entrenamiento es una estimación del verdadero cambio que pudiera resultar de la modificación de los pesos basados en la minimización de la función de costo ξ_{av}

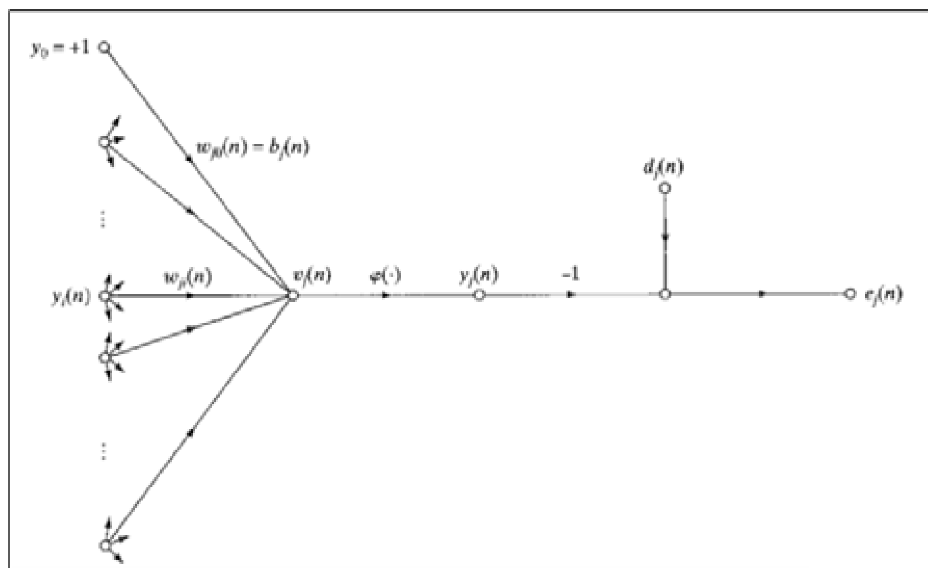

Figura 2.9: Neurona j.

Considere la figura 2.9 que representa una neurona j, la cual se alimenta de las señales producida por la capa anterior. El *induced local field* $v_j(n)$ producida por la entrada de la función activación asociada con la neurona j es la siguiente:

$$v_j(n) = \sum_{i=0}^{m} w_{i,j}(n)y_i(n)$$

Donde m es el total de entradas (incluyendo el *bias*) aplicados a la neurona j. Por lo tanto la función signal $y_j(n)$ aparece en las salidas de la neurona j en cada iteracción n es

$$y_j(n) = \varphi_j(v_j(n))$$

De manera similar al LMS[1], el algoritmo de *back-propagation* aplica una corrección $\nabla w_{i,j}(n)$ para la corrección de los pesos sinápticos $w_j in$, la cual es proporcional a la derivada parcial $\frac{\partial \xi(n)}{\partial w_{i,j}(n)}$.

2.6. Problemas no lineales

En un *perceptron* elemental (una capa) no es necesario tener capas ocultas. En consecuencia, no logra clasificar patrones que sean no lineales. Sin embargo, la mayor cantidad de problemas son no lineales, por ejemplo la OR exclusiva (XOR), la clasificación de sus puntos se encuentran en un hipercubo, cada punto dentro del hipercubo esta entre la clase 0 o la clase 1. Sin embargo para este caso especial solo tendremos que considerar las cuatro esquinas del cuadrado unitario, que corresponden a las entradas del patrón (0,0),(0,1),(1,1) y (1,0). Tanto la primera como tercera entradas del patrón están en la clase 0, como se muestra a continuación:

$$0 \oplus 0 = 0$$

y

$$1 \oplus 1 = 0$$

Donde $\oplus$ es el operador lógico para representar la OR Exclusiva. Las entradas del patrón (0,0) y (1,1) están en esquinas opuestas, pero producen la misma salida. Por otra parte las entradas del patrón (1,0) y (0,1) también están en esquinas opuestas pero se encuentran en la clase 1, como se puede apreciar:

$$0 \oplus 1 = 1$$

y

$$1 \oplus 0 = 1$$

Podemos utilizar una neurona con 2 entradas para generar una linea recta para el limite de decisión en el espacio de entradas. Para todos los puntos sobre un sitio de esta linea, la salida de la neurona es 1; para todos los puntos de otro sitio en esta linea, las salidas son 0. La posición y orientación de la linea en el espacio de entradas esta determinado por los pesos sinápticos de la neurona conectados a la entrada de los nodos. Con las entradas del patrón (0,0) y (1,1) en esquinas opuestas del cuadrado así como las entradas del patrón (0,1) y (1,0) también en esquinas opuesta hace ver que no es posible resolver el problema con una linea recta.

Podemos resolver el problema de XOR agregando una capa oculta con dos neuronas, como se muestra en la figura 2.10. Donde:

- Cada neurona usa la función de activación *threshold*.
- Bit 0 y 1 están representado por los valores 0 y +1, respectivamente.

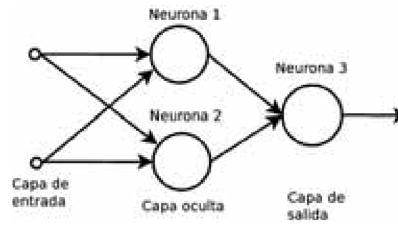

Figura 2.10: Arquitectura para la función XOR.

La neurona 1, que se encuentra en la capa oculta, esta caracterizada como:

$$w_{11} = w12 = +1b_1 = -\frac{3}{2}$$

La pendiente del limite de decisión construido por la neurona 1 en la capa oculta es igual -1 , como se ve reflejada en la figura 2.11A. La características de la neurona 2 que se encuentra en la capa oculta son los siguientes:
La neurona 1, que se encuentra en la capa oculta, esta caracterizada como:

$$w_{21} = w22 = +1b_2 = -\frac{1}{2}$$

La orientación y posición de la neurona 2 se ve reflejado en la figura 2.11B.
La neurona 3 que corresponde a la neurona de salida presenta la siguientes características:

$$w_{31} = -2w_{32} = +1b_3 = -\frac{1}{2}$$

La función de salida de la neurona está construida por la combinación lineal de los limites de decisión formada por las dos neuronas ocultas. El resultado se puede apreciar en la figura 2.11C. Cuando ambas neuronas ocultas están apagadas, esto ocurre cuando las entrada del patrón es (0,0), las salidas de la neuronas permanecen desactivadas. Cuando las neuronas ocultas están encendidas, ocurre cuando las entradas de patrón son (1,1). Cuando la neurona alta de la capa oculta está apagada y la neurona baja de la capa oculta esta apagada cambia la salida de la neurona a un valor positivo, esto ocurre cuando las entradas de patrón son (0,1) o (1,0)[3].

2.7. FPGA

Una FPGA es un dispositivos basado en arreglos de bloques lógicos configurables conectado a través de interconexiones programables[5]. Generalmente el FPGA se puede re-configurar en tiempo de desarrollo y en la actualidad existen herramientas que permiten la re-configuración parcial o total en tiempo de ejecución.

[1]least mean square

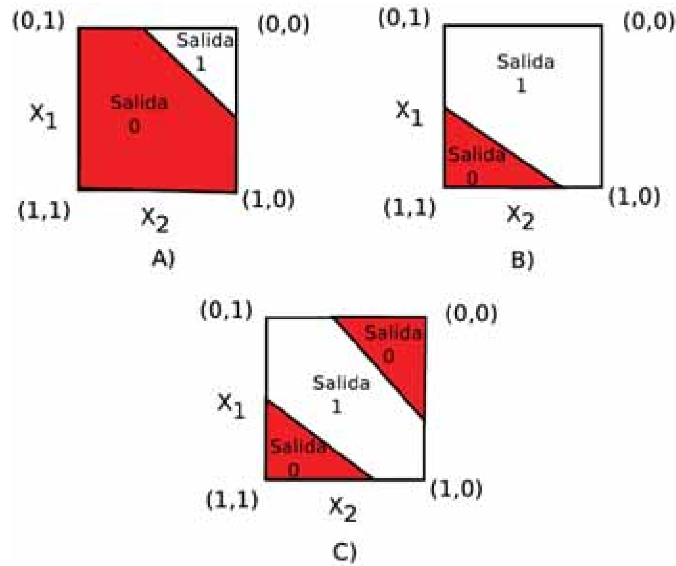

Figura 2.11: Arquitectura para la función XOR.

2.8. Máquina de estados

Una máquina de estado finito con salida M=(S,I,O,f,gs_0) consiste en un conjunto finito de estados S, un alfabeto (conjunto finito no vacío) de entradas I, un alfabeto de salidas O, un estado inicial s_0 una función de transición f: S X I $\longrightarrow$ S y una función de salida g: S X I $\longrightarrow$ O.

Una máquina M=(S,I,O,f,gs_0) puede describirse por una tabla de estados, que indica los valores de las funciones f y g, o por un diagrama de estados, grafo dirigido donde los vértices representan los estados de la máquina, el estado inicial se indica mediante una flecha que no proviene de otro estado y existe una flecha etiquetada por "i,o", del estado s al estado s' si f(s,i)=s' y g(s,i)=o[11].

2.9. Uso de la maquina de estados

El uso de una máquina de estados esta ligado sobre donde se desea programar la red neuronal. Principalmente podemos identificar las redes neuronales en software y redes neuronales en hardware.

Cuando se desea realizar una red neuronal sobre hardware, es costoso realizarlo como una red neuronal de forma clásica. El costo es elevado debido a que el algoritmo de *back-propagation* requiere de derivadas parciales, las cuales para poder resolver se requiere de algún método numérico.

Por ejemplo, tomemos a la función XOR en donde tenemos una capa oculta con 2 neuronas y una de salida. Por tanto tomando en cuenta que para cada ajuste es una derivada y cada ajuste se debe realizar en cada neurona, tenemos un total 3 derivadas parciales, las cuales cada una debe ser resuelta por un método numérico.

En base a todo lo anterior, se implementa una máquina de estados, la cual en cada estado

se controla la salida de la neurona así como los errores generados por las mismas neuronas. La máquina de estados se complementa con un algoritmo de números pseudo-aleatorios que le da un conjunto de probabilidades haciendo de una forma sencilla una aproximación al algoritmo de *back-propagation*.

Capítulo 3

Desarrollo de las redes neuronales

3.1. Descripción técnica

Para desarrollar la red neuronal primero debemos empezar por las estructuras **alto** y **mayor**, ya que sin importar que arquitectura se necesite estos dos módulos no se ven afectados por los cambios de arquitecturas.

3.1.1. Alto

El módulo alto esta compuesta por los módulos **aleatorio** y **neurona** (ver figura 3.1)

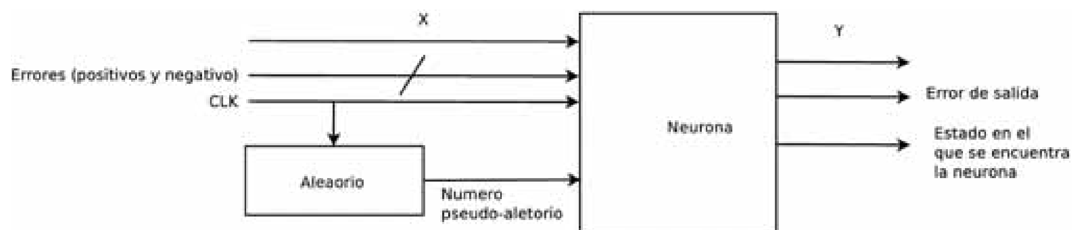

Figura 3.1: Componentes del módulo Alto.

Como se puede observar, el módulo alto solo se encarga de comunicar a la neurona con los números pseudo-aleatorios.

Para ello se hace un portmap de la neurona y el aleatorio, el cual por medio de una señal se enviá la información como se puede apreciar en el código.

```
25
26  --Descripcion
27  architecture beh of alto is
28  signal SAleatorio : std_logic_vector(5 downto 0 );
29  begin
30          Unidad0: Aleatorio port map(CLK,SAleatorio);--Mapeo de la unidad Aletoria
```

```
31          Unidad1: Neurona port map (X,e_pos,e_neg,SAleatorio,CLK,Y,est,e_sal);--Mapeo de la unidad
                Neurona
32  end beh;
```

Cada módulo alto solo puede tener una neurona y un aleatorio, debido a esto el módulo alto solo puede procesar un bit. Por tanto la cantidad de módulos altos se define por el número de bits a utilizar, así como por la arquitectura, por ejemplo, la arquitectura de la NAND para 2 bits solo requiere dos módulos altos, mientras que la arquitectura de XOR para dos bits requiere 4 módulos alto.

Aleatorio

Aquí generamos 73 números no consecutivos, los cuales se se encuentra en la tabla A.2. El circuito solo consta del pulso de reloj como entrada y la única salida que es el número pseudo-aleatorio.

Para poder generar los números pseudo-aleatorios se utilizan 3 contadores:

- Contador de 8 bits, el cual haremos referencia como **contador A**.
- Contador de 6 bits, el cual haremos referencia como **contador B**.
- Contador de 3 bits, el cual haremos referencia como **contador C**.

Todos los contadores se inician en el valor decimal sin signo de 0. Durante cada flanco positivo del pulso de reloj se incrementaran en 1 todos los contadores, al momento en el que **contador A** llegue al valor decimal sin signo de 200 (correspondiente a 11001000 en binario), el **contador A** regresa al valor decimal sin signo de 0, mientras tanto los bits del 2 al 5 y 0 al 1 del **contador B** y del **contador C** correspondientemente se concatenan para crear el número pseudo-aleatorio, él cual es enviado a la neurona.

En tanto el **contador B** regresa al valor 0 decimal sin signo cuando en el conteo se llega al número 111111 (equivalente al 63 decimal sin signo), por su parte el **contador C** debe llegar al valor 111 (equivalente al 7 decimal sin signo).

Neurona

Este módulo contiene la máquina de estados, en dicha máquina de estados se define la salida así como los errores.

La máquina de estados esta definida por 4 estados, los cuales se representan en la figura 3.2, en donde Y = 1 es el estado 00, Y = 0 es el estado 01, Y = NOT X es el estado 10 y Y = X es el estado 11, dichos estados realizan la transición conforme a la tabla 3.1.

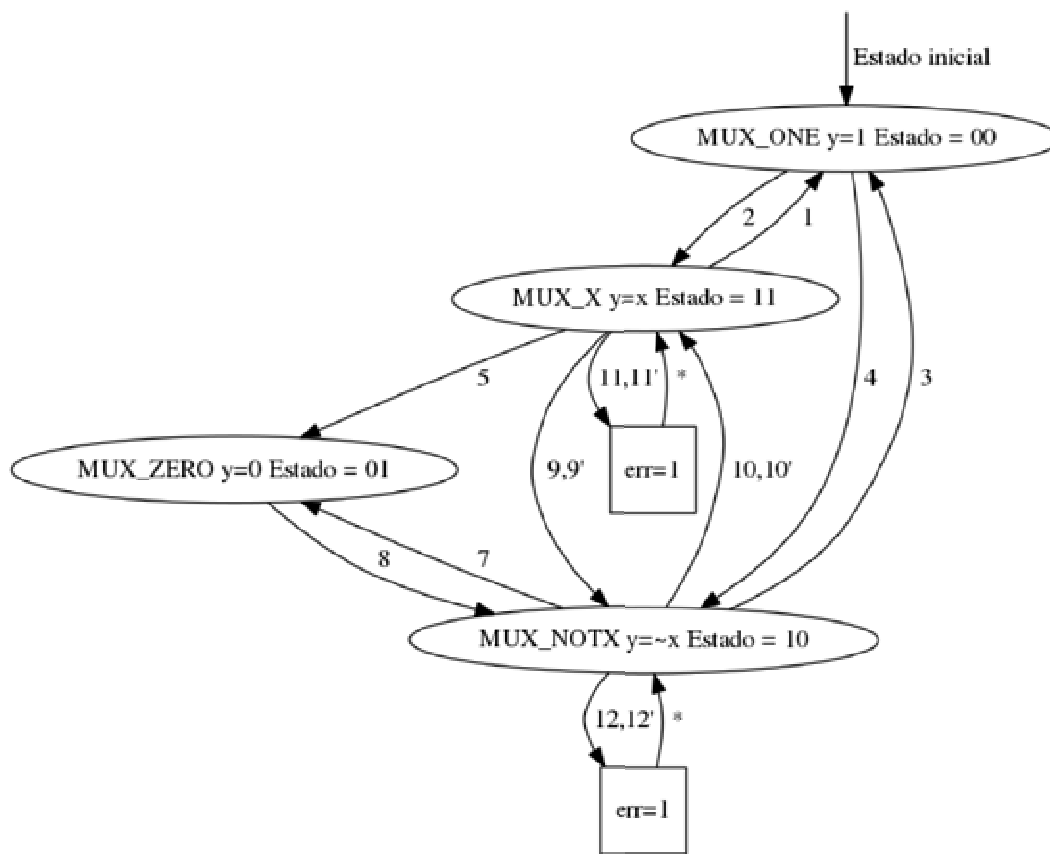

Figura 3.2: Máquina de estados.

Para que la neurona pueda realizar la evaluación de cambio de estado debe existir un flanco positivo en el pulso de reloj y así mismo los valores de los bits 3 al 0 del número pseudo-aleatorio sean iguales a 0 (en la tabla A.2 se aprecian los valores en negritas, los cuales generarán una transición), como se logra apreciar en el siguiente código.

```
32
33                  if(CLK'event and clk='1') then--Si existe un cambio en el pulso de reloj y es 1
34                          sy <= sy;
35                          err <= err;
36                          estado <= estado;
37                          if (Rand (3 downto 0)="0000") then--Se realizara cambio de estado
                                unicamente cuando se encuentre en el estado "0000"
```

Entendiendo todo esto es importante resaltar que el primer estado de la máquina, es el estado 00 (Y = 1) mientras que el error se inicia en 0.

Las neuronas solamente son capaces de procesar un bit, por lo que el número de neuronas va ser directamente proporcional al número de módulos altos en la arquitectura.

No.	err_plus	err_minus	X	Rand
1	1	0	0	00
2	0	1	0	- -
3	1	0	1	00
4	0	1	1	- -
5	0	1	1	00
6	1	0	1	- -
7	0	1	0	00
8	1	0	0	- -
9	1	0	0	01
9’	0	1	1	01
10	1	0	1	01
10’	0	1	0	01
11	1	0	0	10
11’	0	1	1	10
12	1	0	1	10
12’	0	1	0	10
*	- -	- -	- -	- -

Cuadro 3.1: - don’t care, * Transición incondicional

3.1.2. Mayor

El módulo mayor tiene como entradas las n-entradas de las neuronas y la salida obtenida por la función F (como referencia ver la figura 3.3).

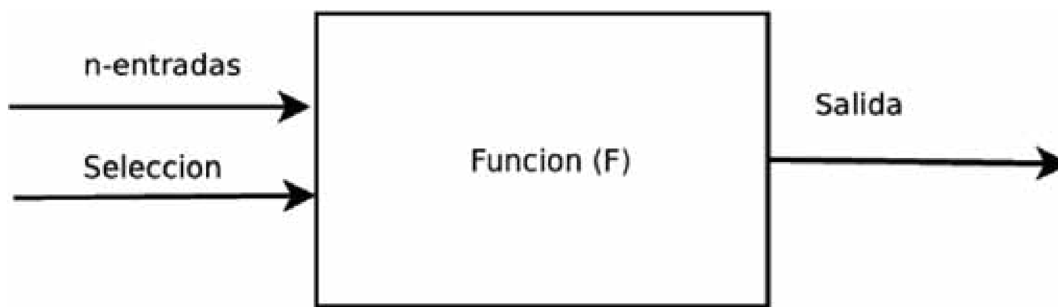

Figura 3.3: Estructura del módulo mayor

Para este caso, se agrego la entrada llamada selección. Esto debido a que dependiendo de la arquitectura (el cual para este proyecto son las funciones lógicas NAND y XOR) se tendrá que determinar si utilizar la función OR o NAND.

La complejidad de las funciones del módulo mayor dependerán de la cantidad de entradas

provenientes de las neuronas, del mismo modo la menara de trabajar de la función depende de la arquitectura que se emplee para la función lógica, por ejemplo, en la función NAND con dos entradas se utilizará la función OR, mientras que para la función XOR se utilizarán tanto la funciones OR y AND.

3.1.3. Comparador

El módulo de comparador recibe los errores de las neuronas, así como el resultado generada por la red neuronal, del mismo modo esté módulo recibe la etiqueta, la cual sirve para poder informar a la neurona si el resultado que obtuvo es el correcto (ver figura 3.4) .

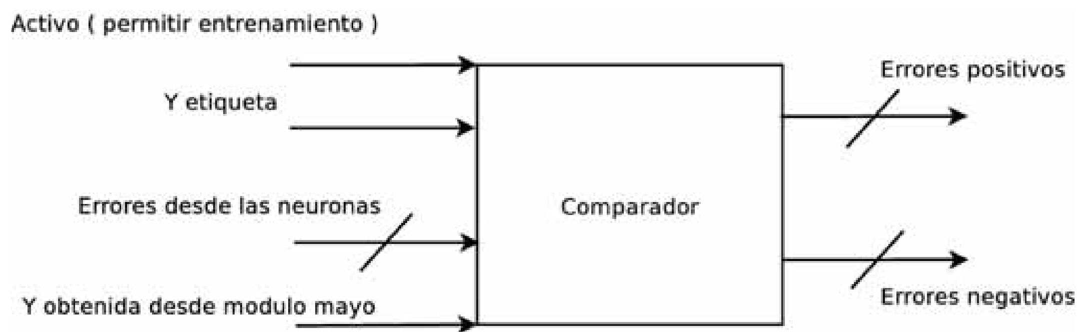

Figura 3.4: Modulo Comparador.

La información obtenida es procesada bajo los siguientes criterios:

- Se envían todos los errores en 0 cuando activo se encuentra en 0.
- Se envían todos los errores en 0 cuando **Y etiqueta** y la **Y obtenida desde la red** son iguales.
- Se envía error en 1 a todas las neuronas si no existe error en ninguna neurona pero **Y etiqueta** y la **Y obtenida desde la red** son diferentes.
- Se envía error en 1 a aquellas neuronas que hayan enviado error.

Para determinar si el error enviado a las neuronas será error positivo o error negativo se siguen los siguientes criterios:

- Si **Y etiqueta** es mayor que **Y obtenida desde la red** se enviá error positivo con valor 1 y error negativo con valor 0.
- Si **Y etiqueta** es menor que **Y obtenida desde la red** se enviá error positivo con valor 0 y error negativo con valor 1.

3.2. Especificación técnica

3.2.1. Arquitectura e implementación de la función NAND

Para realizar la función X_1 NAND X_2, debemos utilizar la ley de De Morgan, en donde X_1 NAND X_2 equivale a NOTX_1 OR NOTX_2.

Tomando en cuenta que los módulos altos solo pueden procesar un bit y que conforme la ley de De Morgan se tiene una función OR, la arquitectura se formará a partir de 2 módulos altos los cuales las salidas correspondientes serán las entradas del módulo mayor, esté ultimo realizará una OR la cual es el resultado generado por la red neuronal, por medio de una señal es enviado al comparador y a un led para su visualización (ver figura 3.5).

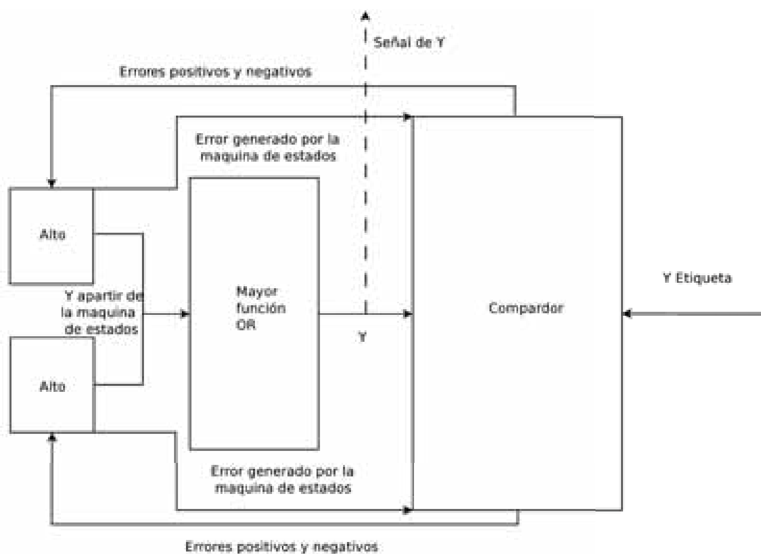

Figura 3.5: Arquitectura NAND

La arquitectura NAND cuenta con una capa de entrada correspondiente a los dos módulos altos y una capa de salida correspondiente al módulo mayor.

Antes de implementarse sobre la FPGA, se realiza la tabla A.1 con una corrida de escritorio. Como logramos apreciar en la corrida de escritorio, la red aparentemente ya aprendió, estando en el estado inicial (Y = 1).

Analizando más profundamente podemos ver que Y = 1, cuando se dan los valores $X_1 = 0$ $X_2 = 0$,$X_1 = 1$ $X_2 = 0$ y $X_1 = 0$ $X_2 = 1$.

Tomando todo esto en cuenta, podemos determinar que la neurona logra el aprendizaje con los valores $X_1 = 1$ $X_2 = 1$, estos valores harán que las neuronas conforme a la tabla 3.1 y la figura 3.2 pasé del estado 00 (Y=0) al estado 10 (Y=NOTX) por medio de la transición 4,quedando los estados de los dos módulos altos en el estado 10 correspondiente a Y=NOTX.

En base a la información anterior y la tabla A.2 podemos determinar dentro de un escenario perfecto que la neurona logrará aprender en el número 010000 (16 Decimal sin signo), transcurriendo

un total de 22 números pseudo-aleatorios, del cual tomando en cuenta que un número pseudo-aleatorio se genera con 400 pulsos de reloj, podemos determinar que la red neuronal aprendería la función NAND en 8800 pulsos de reloj.

Para la implementación sobre la FPGA se utilizó un switch para cada X_n , un switch para la etiqueta, un switch para activar el aprendizaje, un swtich para la selección de la función OR o AND que va utilizar el módulo Mayor. Para poder observar los estados se utilizan un total de 4 leds verdes, mientras se ocupa un led para representar la salida de la red neuronal.

3.2.2. Arquitectura e implementación de la función XOR

La definición de la función X_1XORX_2 es NOTX_1 AND X_2 OR X_1 AND NOTX_2, con conocimiento en está definición se desarrollará la arquitectura de la función XOR. También se debe de tomar en cuenta que la cantidad de módulos alto se incrementa. Entendiendo que cada módulo alto procesa un bit, necesitaremos 2 módulos altos por cada X_n, como se muestra en la figura 3.6.

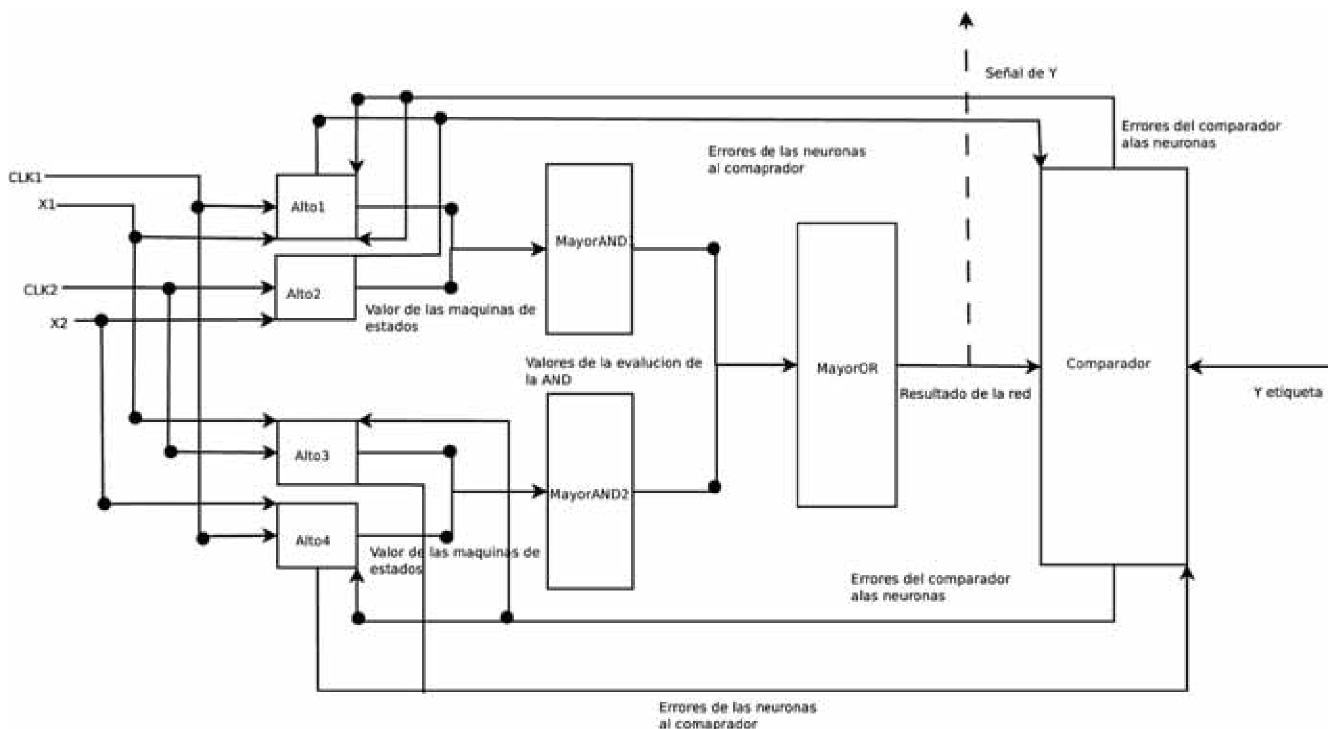

Figura 3.6: Arquitectura XOR.

En donde la capa de entrada corresponden a los módulos altos, la capa oculta corresponde a los módulos mayor en AND mientras la capa salida corresponde al módulo mayor en OR.

La diferencia entre los dos pulsos de reloj genera que los módulos altos 2 y 3 generen un número pseudo-aletoria cuando los módulos altos 1 y 4 ya hayan generado 6 números pseudo-aleatorios.

A partir de lo anterior en un escenario perfecto, en el cual los módulos altos 1 y 4 aprendan en el número pseudo-aleatorio 24 conforme la tabla A.2, mientras que los módulos altos 2 y 3 realicen el cambio de estado hasta el número pseudo-aleatorio 244 (tomando en cuenta que se genera un número pseudo-aleatorio en los módulos altos 3 y 2 cuando ya se generaron 6 números pseudo-aleatorios en los módulos altos 1 y 4), por tanto la red aprendería la función XOR en 52800 pulsos de reloj, en donde los estados de los módulos altos pueden darse de la siguiente manera:

- Los módulos alto 1 y 4 en el estado 11 (Y=X) mientras que los módulos 2 y 3 en el estado 10 (Y=NOTX).
- Los módulos alto 1 y 4 en el estado 10 (Y=NOTX) mientras que los módulos 2 y 3 en el estado 11 (Y=X).

Por lo explicado anteriormente hace complicado que la arquitectura XOR pueda probarse por medio de una simulación.

Para la implementación en la FPGA se utiliza un switch para cada X_n, una switch para activar el aprendizaje y otro para colocar la etiqueta. En está implementación a diferencia a la arquitectura NAND no se utiliza un switch para seleccionar el estado de los módulos mayores, ya que estos se dejan fijos en el código HDL. De modo parecido se utilizan 4 leds verdes para representar los estados de los módulos altos 1 y 2, mientas que para los estados de los módulos 3 y 4 se representan en 4 leds rojos, por ultimo la salida de la red neuronal se representa con un led verde.

Capítulo 4

Pruebas y resultados

4.1. Pruebas arquitectura NAND

Durante la simulación de esta arquitectura se intento el aprendizaje de la neurona con la combinación X_1=1, X_2=1, YEtiqueta=0 y el módulo Mayor en el estado OR. Logrando que la neurona aprende a partir de la primer intento, conforme el escenario perfecto (ver figura 4.1).

Figura 4.1: Simulación de la función NAND.

AL momento de realizar la implementación de la FPGA, se inicia con la combinación X_1=1, X_2=1, YEtiqueta=0 y el módulo Mayor en el estado OR, dando como resultado que se logre el aprendizaje, quedando los estados de los módulos altos en 10 (Y=NOTX).

4.2. Pruebas arquitectura XOR

Como se comento anteriormente, realizar la simulación de esta arquitectura resulta extremadamente compleja. Es por ello que está arquitectura se fue probando directamente en la fpga.

En la implementación en la FPGA se puede observar que la red neuronal aprendía con la segunda combinación, en el peor de los casos lograba aprender hasta la tercer combinación. Del mismo modo los estados de los módulos altos 1 y 4 eran 11(Y=X) mientras en los módulo altos 2

y 3 eran 10(Y=NOTX), esta combinación de estados en los módulos altos podría darse también en la inversa.

Capítulo 5

Conclusiones

Las redes neuronales en la actualidad son herramientas muy útiles para escenarios en donde se requiera poco hardware y se deban realizar ciertas tareas en especifico.

En el caso preciso de este trabajo, el trabajo mas complejo es el aprendizaje de las funciones ya que este proceso es en el que más cuidado se debe tener y también el que lleva mas tiempo.

Una de las cosas que se deben mejorar es el algoritmo de número aleatorios, ya que el algoritmo de números aleatorios utilizados en este trabajo solo se generan 75 números de los cuales 5 números son utilizados para realizar las transiciones entre los estados, se debe buscar un algoritmo más complejo que permita crear una secuencia más largas y con mayor cantidad de opciones para la transición de estados, esto probablemente ayudará a que las transiciones se pueden llevar de una forma más rápida.

Bibliografía

[1] Rodrigo Salas (visto el 12 de febrero de 2013).
Redes Neuronales Artificiales.Disponible en: http://www.inf.utfsm.cl/~rsalas/Pagina_Investigacion/docs/Apuntes/Redes %20Neuronales %20 Artificiales.pdf

[2] J.G. Taylor.*Neural networks and their applications.*While,1996.

[3] Simon S. Haykin.*Neural networks a comprehensive fundation.*Prentice-Hall,1999

[4] Universidad Carlos III de Madrid (visto el 12 de febrero de 2013).
Inteligencia en redes de comunicaciones.Disponible en:http://www.it.uc3m.es/jvillena/irc/practicas/04-05/7mem.pdf

[5] XILINX(visto el 8 de febrero de 2013).
Disponible en : http://www.xilinx.com/training/fpga/fpga-field-programmable-gate-array.htm

[6] Alducin Castillo, Javier,"Sistema de identificación de rostros humanos a través de redes neuronales",proyecto terminal,División de CBI, Universidad Autónoma Metropolitana Azcapotzalco,México,2008.

[7] Robles Sandoval, Rafael,Ïmplementación del Algoritmo de Digesto MD5 en un Sistema Incrustado Basado en un FPGA",proyecto terminal,División de CBI, Universidad Autónoma Metropolitana Azcapotzalco,México,2011

[8] Bautista Noyola,Joel,"Diseño e Implementación de un Co-Procesador Matemático de Aritmética Entera basado en un FPGA",proyecto terminal,División de CBI, Universidad Autónoma Metropolitana Azcapotzalco,México,2011

[9] Roman Kohut,Bernds Steinbach (visto el 21 de enero de 2013).
Boolean neuronal network. Disponible en: http://www.informatik.tu-freiberg.de/prof2/publikationen/BNN_WSEAS.pdf

[10] L. Ionescu, I. Bostan, V. Ionescu (visto el 29 de enero de 2013).
FPGA implementation of a boolean neuronal network (2 de octubre de 2003). Disponible en: http://ieeeexplore.ieee.org/xpl/search/searchresult.jsp?newsearch=true&queryText=FPGA+implementation+of+boolean+neuronal+network&x=-850&y=-194

[11] Universidad de Coruña(visto el 19 de Diciembre de 2014).
Matemáticas discretas, área de álgebra .Disponible en: http://quegrande.org/apuntes/grado/1G/MDG/teor
11/tema_3_-_maquinas_de_estado_finito.pdf

Apéndices

Apéndices A

Apéndice de tablas

A.1. Prueba de escritorio NAND

Ultimos 2 bits A	Ultimos 2 bits B	X_1	X_2	Yneurona1	Yneurona2	Y	Y Etiqueta	Error neurona1	Error neurona2	Error positivo1	Error positivo2	Error negativo1	Error negativo2
.	.	0	0	1	1	1	1	0	0	0	0	0	0
01	01	0	0	1	1	1	1	0	0	0	0	0	0
10	10	0	0	1	1	1	1	0	0	0	0	0	0
10	10	0	1	1	1	1	1	0	0	0	0	0	0
11	11	0	1	1	1	1	1	0	0	0	0	0	0
00	00	1	0	1	1	1	1	0	0	0	0	0	0
01	01	1	0	1	1	1	1	0	0	0	0	0	0
10	10	1	0	1	1	1	1	0	0	0	0	0	0
10	10	1	1	1	1	1	0	0	0	0	0	1	1
11	11	1	1	0	0	0	0	0	0	0	0	0	0
00	00	1	0	0	1	1	0	0	0	ya que en la simulación para introducir los valores correspondientes de X_1, X_2 y la etiqueta se debe detener la simulación, eso hace que el pulso de reloj no logre avanzar y se mantenga en un tiempo constante los cambios. 0	0	0	0

Cuadro A.1: Pruebas de escritorio NAND

A.2. Números pseudo-aleatorios

Número sin signo decimal	Número sin signo decimal	Número sin signo binario
01	50	110010
02	31	011111
03	12	001100
04	62	111110
05	47	101111
06	28	011100
07	13	001101
08	59	111011
09	40	101000
10	25	011001
11	10	001010
12	56	111000
13	41	101001
14	22	010110
15	07	000111
16	53	110101
17	38	100110
18	23	010111
19	04	000100
20	50	110010
21	35	100011

Cuadro A.2: Pseudo-aleatorios parte 1

Número sin signo decimal	Número sin signo decimal	Número sin signo binario
22	**16**	**010000**
23	01	000001
24	51	110011
25	**32**	**100000**
26	13	001101
27	63	111111
28	44	101100
29	29	011101
30	14	001110
31	60	111100
32	41	101001
33	26	011010
34	11	001011
35	57	111001
36	42	101010
37	27	011011
38	04	000100
39	54	110110
40	39	100111
41	20	010100
42	05	000101
43	55	110111
44	**32**	**100000**
45	17	010001
46	02	000010
47	**48**	**110000**
48	33	100001
49	18	010010
50	03	000011
51	45	101101
52	30	011110
53	15	001111
54	61	111101

Cuadro A.3: Pseudo-aleatorios parte 2

Número sin signo decimal	Número sin signo decimal	Número sin signo binario
55	46	101110
56	31	011111
57	08	001000
58	58	111010
59	43	101011
60	24	011000
61	09	001001
62	59	111011
63	36	100100
64	21	010101
65	06	000110
66	52	110100
67	37	100101
68	22	010110
69	03	000011
70	49	110001
71	34	100010
72	19	010011
73	**00**	**000000**

Cuadro A.4: Pseudo-aleatorios parte 3

Apéndices B

Apéndice de códigos

B.1. Código Alto

```
--Juan Pablo Hernandez Castillo
--Modulo alto
--Proyecto terminal , Implementacion de una red Booleana sobre un FPGA

--Bibliotecas de la IEEE
library IEEE;
use IEEE.STD_LOGIC_1164.all;
USE IEEE.NUMERIC_STD.all;
use work.NeuronaPack.all;
use work.AleatorioPack.all;


--Declaramos la entidad de el modulo alto
entity alto is
port(
   X: in std_logic;--Entrada x
   e_pos: in std_logic;--Entrada de error positiva
   e_neg: in std_logic;--Entrada de error negativa
   CLK: in std_logic;--Pulso de reloj
   Y: out std_logic;--Salida y
   est: out STD_LOGIC_VECTOR(1 downto 0);--Estado de la neurona
   e_sal: out std_logic --Error de salida
);
end alto;

--Descripcion
architecture beh of alto is
signal SAleatorio : std_logic_vector(5 downto 0 );
begin
        Unidad0: Aleatorio port map(CLK,SAleatorio);--Mapeo de la unidad Aletoria
        Unidad1: Neurona port map (X,e_pos,e_neg,SAleatorio,CLK,Y,est,e_sal);--Mapeo de la unidad
            Neurona
end beh;



--componente
library IEEE;
use  ieee.std_logic_1164.all;
package AltoPack is
```

```
40          component alto
41                  port(
42                          X: in std_logic;--Entrada x
43                          e_pos: in std_logic;--Entrada de error positiva
44                          e_neg: in std_logic;--Entrada de error negativa
45                          CLK: in std_logic;--Pulso de reloj
46  --                      CLK2: in std_logic;--Pulso de reloj
47                          Y: out std_logic;--Salida y
48              est: out STD_LOGIC_VECTOR(1 downto 0);--Estado de la neurona
49                          e_sal: out std_logic --Error de salida
50                          );
51          end component;
52  end package;
```

B.2. Código aleatorio

```
--Juan Pablo Hernandez Castillo
--Circuito para obtener numeros pseudo-aleatorios
--Proyecto terminal , Implementacion de una red Booleana sobre un FPGA

--Bibliotecas de la IEEE
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.std_logic_signed.all;

--Declaramos la entidad del Aleatorio
entity Aleatorio is
  port(
   CLK: in std_logic;--Pulso de reloj
   salida: out std_logic_vector(5 downto 0)--Salida de los numeros pseudo-aleatorios
  );
end Aleatorio;

--Descripcion del Contador
architecture beh of Aleatorio is
--Declaracion de componentes
signal sumaultimosbits: std_logic_vector (2 downto 0) := "000"; --Senal para el conteo de los 2
    bits menos significativos
signal sum: std_logic_vector (5 downto 0):="000000"; --Senal que nos ayudara en la suma
signal cont: std_logic_vector (7 downto 0):="00000000";--Contador para llegar a 200

begin
  process(CLK)--Si existe un cambio en el pulso de reloj
  begin
    if(CLK'event and clk='0') then --Si existe un cambio en el pulso de reloj y es 1
      if(cont="11001000") then--Si se llega a 200 da el numero
        cont<="00000000";
        salida<=sum(5 downto 2)& sumaultimosbits(1 downto 0);
      else--Sigue el conteo
                  cont<=cont + 1;
        if(sum="111111") then --Si se llega al numero maximo del conteo
          sum<="000000";
        elsif(sumaultimosbits = "111") then
                        sumaultimosbits<="000";
        else--se sigue contando
                         sum<=sum + 1;
                         sumaultimosbits <= sumaultimosbits + 1;
        end if;
      end if;
    end if;
  end process;
end beh;

--componente
library IEEE;
use ieee.std_logic_1164.all;
package AleatorioPack is
  component Aleatorio
    port(
     CLK: in std_logic;
     salida: out std_logic_vector(5 downto 0)
    );
  end component;
end package;
```

B.3. Código neurona

```
1  --Juan Pablo Hernandez Castillo
2  --Mux, para obtener salidas del circuito
3  --Proyecto terminal , Implementacion de una red Booleana sobre un FPGA
4
5  --Bibliotecas de la IEEE
6  library IEEE;
7  use IEEE.STD_LOGIC_1164.all;
8  USE IEEE.NUMERIC_STD.all;
9
10 --Declaramos la entidad del Neurona
11 entity Neurona is
12   port(
13    X: in std_logic;--Entrada x
14    e_pos: in std_logic;--Entrada de error positiva
15    e_neg: in std_logic;--Entrada de error negativa
16    Rand: in std_logic_vector (5 downto 0);--Entrada de numeros pseudoaleatorios
17    CLK: in std_logic;--Pulso de reloj
18    Y: out std_logic;--Salida y
19    est: out STD_LOGIC_VECTOR (1 downto 0);--Salida al display
20    e_sal: out std_logic --Error de salida
21   );
22 end Neurona;
23
24 --Descripcion
25 architecture beh of Neurona is
26 --Declaracion de componentes
27 --Declaracion de senales
28 signal err : std_logic:= '0' ;
29 signal sy : std_logic:= '1';
30 signal estado: STD_LOGIC_VECTOR (1 downto 0):= "00";
31 begin
32         process(CLK)--Si existe un cambio en el pulso de reloj
33         begin
34                 if(CLK'event and clk='1') then--Si existe un cambio en el pulso de reloj y es 1
35                         sy <= sy;
36                         err <= err;
37                         estado <= estado;
38                         if (Rand (3 downto 0)="0000") then--Se realizara cambio de estado
                               unicamente cuando se encuentre en el estado "0000"
39                          if (estado( 1 downto 0) ="00") then
40                              if (e_pos = '0' AND e_neg = '1' AND  X = '0') then--2
41                                  estado <= "11";
42                                  sy <= X;
43                                  err <= '0';
44                              elsif (e_pos = '0' AND e_neg = '1' AND X = '1') then --4
45                                  estado <= "10";
46                                  sy <= NOT X;
47                                  err <= '0';
48                              else
49                                  estado <= "00";
50                                  sy <= '1';
51                                  err <= '0';
52                              end if;
53                          elsif(estado(1 downto 0) = "01") then
54                              if (e_pos = '1' AND e_neg = '0' AND X = '1' ) then--6
55                                  estado <= "11";
56                                  sy <= X;
57                                  err <= '0';
58                              elsif( e_pos = '1' AND e_neg = '0' AND X = '0') then --8
59                                  estado <= "10";
```

```
60                                  sy <= NOT X;
61                                  err <= '0';
62                              else
63                                  estado <= "01";
64                                  sy <=  '0';
65                                  err <= '0';
66                              end if;
67                          elsif(estado (1 downto 0) = "10") then
68                              if (e_pos = '1' AND e_neg = '0' AND X = '1' AND Rand (5 downto 4) = "
                                    00") then --3
69                                  estado <= "00";
70                                  sy <= '1';
71                                  err <= '0';
72                              elsif(e_pos = '0' AND e_neg = '1' AND X = '0' AND Rand (5 downto 4) =
                                     "00") then --7
73                                  estado <= "01";
74                                  sy <= '0';
75                                  err <= '0';
76                              elsif (e_pos = '1' AND e_neg = '0' AND X = '1' AND Rand ( 5 downto 4)
                                     = "01") then--10
77                                  estado <= "11";
78                                  sy <= X;
79                                  err <= '0';
80                              elsif (e_pos = '0' AND e_neg =  '1' AND X = '0' AND Rand (5 downto 4)
                                     = "01") then--10'
81                                  estado <= "11";
82                                  sy <= X;
83                                  err <= '0';
84                              elsif (e_pos = '1' AND e_neg = '0' AND X = '1' AND Rand (5 downto 4)
                                    ="10") then --12
85                                  estado <= "10";
86                                  sy <= NOT X;
87                                  err <= '1';
88                              elsif (e_pos = '0' AND e_neg = '1' AND X = '0' AND Rand (5 downto 4)
                                    ="10") then --12'
89                                  estado <= "10";
90                                  sy<= NOT  X;
91                                  err <= '1';
92                              else  --*
93                                  estado <= "10";
94                                  sy <= NOT X;
95                                  err <= '0';
96                              end if;
97                          else
98                              if(e_pos = '1' AND e_neg = '0' AND X = '0' AND Rand (5 downto 4) ="00
                                    ") then --1
99                                  estado <= "00";
100                         sy <= '1';
101                         err <= '0';
102                             elsif (e_pos = '0' AND e_neg = '1' AND X = '1' AND Rand (5 downto 4)
                                    ="00") then --5
103                                 estado <= "01";
104                         sy <= '0';
105                         err <= '0';
106                             elsif (e_pos = '1' AND e_neg = '0' AND X = '0' AND Rand (5 downto 4)
                                    ="01") then --9
107                                 estado <= "10";
108                         sy <= NOT X;
109                         err <= '0';
110                             elsif (e_pos = '0' AND e_neg = '1' AND X = '1' AND Rand (5 downto 4)
                                    ="01") then --9'
111                                 estado <= "10";
112                     sy <= NOT X;
113                         err <= '0';
```

```
114                                 elsif (e_pos = '1' AND e_neg = '0' AND X = '0' AND Rand (5 downto 4)
                                        ="10") then --11
115                                     err <= '1';
116                                     sy <= X;
117                                     estado <= "11";
118                                 elsif (e_pos = '0' AND e_neg = '1' AND X = '1' AND Rand (5 downto 4)
                                        ="10") then --11'
119                                     err <= '1';
120                                     sy <=  X;
121                                     estado <= "11";
122                                 else --*
123                                     err <= '0';
124                                     sy <=  X;
125                                     estado <= "11";
126                                 end if;
127                             end if;
128                         end if;
129                 end if;
130         Y <= sy;
131         e_sal <= err;
132         est <= estado;
133         end process;
134 end beh;
135 
136 --componente
137 library IEEE;
138 use  ieee.std_logic_1164.all;
139 package NeuronaPack is
140         component Neurona
141                 port(
142                         X: in std_logic;--Entrada x
143                         e_pos: in std_logic;--Entrada de error positiva
144                         e_neg: in std_logic;--Entrada de error negativa
145                         Rand: in std_logic_vector (5 downto 0);--Entrada de numeros
                                pseudoaleatorios
146                         CLK: in std_logic;--Pulso de reloj
147                         Y: out std_logic;--Salida y
148             est: out STD_LOGIC_VECTOR (1 downto 0);--Salida del estado de la neurona
149                         e_sal: out std_logic --Error de salida
150                         );
151         end component;
152 end package;
```

B.4. Código Mayor

```
--Juan Pablo Hernandez Castillo
--Modulo mayor
--Proyecto terminal , Implementacion de una red Booleana sobre un FPGA

--Bibliotecas de la IEEE
library IEEE;
use IEEE.STD_LOGIC_1164.all;
USE IEEE.NUMERIC_STD.all;

--Declaramos la entidad
entity Mayor is
	port (
		Entrada1: in std_logic;
		Entrada2: in std_logic;
		seleccion: in STD_LOGIC;
		Salida: out std_logic
		);
end Mayor;

architecture bhe of Mayor is

begin
	Salida <= Entrada1 and Entrada2 when seleccion='1' else  -- Si la seleccion es 1 se realiza AND
	Entrada1 or Entrada2; --Si la seleccion es 0 se realiza la OR
end bhe;

--Componente
library IEEE;
use  ieee.std_logic_1164.all;
package MayorPack is
	component Mayor
		port (
		Entrada1: in std_logic;
		Entrada2: in std_logic;
	    seleccion: in STD_LOGIC;
			Salida : out std_logic
			);
	end component;
end package ;
```

B.5. Código Comparador (NAND)

```
--Juan Pablo Hernandez Castillo
--Modulo comparador
--Proyecto terminal , Implementacion de una red Booleana sobre un FPGA

--Bibliotecas de la IEEE
library IEEE;
use IEEE.STD_LOGIC_1164.all;
USE IEEE.NUMERIC_STD.all;

entity comparador is
    Port ( --CLK : in STD_LOGIC;
           yprima : in STD_LOGIC;
           y : in STD_LOGIC;
           activo : in STD_LOGIC;
           sepos1 : out STD_LOGIC;
           seneg1 : out STD_LOGIC;
           sepos2 : out STD_LOGIC;
           seneg2 : out STD_LOGIC;
           err1 : in STD_LOGIC;
           err2 : in STD_LOGIC);
end comparador;

architecture Behavioral of comparador is

begin
    process(yprima,y,activo,err1,err2)
    begin
            sepos1 <= '0';
            seneg1 <= '0';
            sepos2 <= '0';
            seneg2 <= '0';
            if(activo = '1') then
                if( yprima /= y) then --Si el resultado es diferente entre la etiqueta y lo obtenido
                    if( err1 = '1' OR err2 = '1') then --Si existe error en cualquiera de las neuronas
                        if (err1 = '1') then --Si el error  esta en la primera neurona
                            if(yprima < y) then --Si el resultado esperado es mayor a la etiqueta
                                seneg1 <= '1';
                            else -- Si el resultado esperado  es menor a la etiqueta
                                sepos1 <= '1';
                            end if;
                        end if;
                        if (err2 = '1') then --si el error esta en la segunda neurona
                            if(yprima < y) then --Si el error  esta en la primera neurona
                                seneg2 <= '1';
                            else -- Si el resultado esperado  es menor a la etiqueta
                                sepos2 <= '1';
                            end if;
                        end if;
                    else -- si no se ve error en nnguna neurona pero el resultado no es lo esperado
                        if(yprima < y) then --Si el reasultado esperado es mayor a la etiqueta
                            seneg1 <= '1';
                            seneg2 <= '1';
                        else -- Si el resultado esperado  es menor a la etiqueta
                            sepos1 <= '1';
                            sepos2 <= '1';
                        end if;
                    end if;
```

```
58                  end if;
59             end if;  --end if;
60  end process;
61  end Behavioral;
62
63
64  --Componente
65  library IEEE;
66  use  ieee.std_logic_1164.all;
67  package ComparadorPack is
68          component comparador
69                  port (--CLK : in STD_LOGIC;
70                 yprima : in STD_LOGIC;
71                 y : in STD_LOGIC;
72                 activo : in STD_LOGIC;
73                 sepos1 : out STD_LOGIC;
74                 seneg1 : out STD_LOGIC;
75                 sepos2 : out STD_LOGIC;
76                 seneg2 : out STD_LOGIC;
77                 err1 : in STD_LOGIC;
78                 err2 : in STD_LOGIC
79                          );
80          end component;
81  end package ;
```

B.6. Código Comparador (XOR)

```
--Juan Pablo Hernandez Castillo
--Modulo comparador
--Proyecto terminal , Implementacion de una red Booleana sobre un FPGA

--Bibliotecas de la IEEE
library IEEE;
use IEEE.STD_LOGIC_1164.all;
USE IEEE.NUMERIC_STD.all;

entity comparador is
    Port ( --CLK : in STD_LOGIC;
           yprima : in STD_LOGIC;
           y : in STD_LOGIC;
           activo : in STD_LOGIC;
           sepos1 : out STD_LOGIC;
           seneg1 : out STD_LOGIC;
           sepos2 : out STD_LOGIC;
           seneg2 : out STD_LOGIC;
           sepos3 : out STD_LOGIC;
           seneg3 : out STD_LOGIC;
           sepos4 : out STD_LOGIC;
           seneg4 : out STD_LOGIC;
           err1 : in STD_LOGIC;
           err2 : in STD_LOGIC;
           err3 : in STD_LOGIC;
           err4 : in STD_LOGIC);
end comparador;

architecture Behavioral of comparador is

begin
    process(yprima,y,activo,err1,err2)
    begin
            sepos1 <= '0';
            seneg1 <= '0';
            sepos2 <= '0';
            seneg2 <= '0';
            sepos3 <= '0';
            seneg3 <= '0';
            sepos4 <= '0';
            seneg4 <= '0';
            if(activo = '1') then
                if( yprima /= y) then --Si el resultado es diferenteentre la etiqueta y lo
                    obtenido
                    if( err1 = '1' OR err2 = '1' OR err3 = '1' OR err4 = '1') then --Si existe
                        error en cualquiera de las  neuronas
                        if (err1 = '1') then --Si el error  esta en la primera neurona
                            if(yprima < y) then --Si el reasultado esperado es mayor a la etiqueta
                                seneg1 <= '1';
                            else -- Si el resultado esperado  es menor a la etiqueta
                                sepos1 <= '1';
                            end if;
                        end if;
                        if (err2 = '1') then --si el error esta en la segunda neurona
                            if(yprima < y) then --Si el error  esta en la primera neurona
                                seneg2 <= '1';
                            else -- Si el resultado esperado  es menor a la etiqueta
                                sepos2 <= '1';
                            end if;
                        end if;
```

```
59                              if (err3 = '1') then --si el error esta en la tercera neurona
60                                  if(yprima < y) then --Si el error  esta en la primera neurona
61                                      seneg3 <= '1';
62                                  else -- Si el resultado esperado  es menor a la etiqueta
63                                      sepos3 <= '1';
64                                  end if;
65                              end if;
66                              if (err4 = '1') then --si el error esta en la cuarta neurona
67                                  if(yprima < y) then --Si el error  esta en la primera neurona
68                                      seneg4 <= '1';
69                                  else -- Si el resultado esperado  es menor a la etiqueta
70                                      sepos4 <= '1';
71                                  end if;
72                              end if;
73                          else -- si no se ve error en nnguna neurona pero el resultado no es lo
                                esperado
74                              if(yprima < y) then --Si el reasultado esperado es mayor a la etiqueta
75                                  seneg1 <= '1';
76                                  seneg2 <= '1';
77                                  seneg3 <= '1';
78                                  seneg4 <= '1';
79                              else -- Si el resultado esperado  es menor a la etiqueta
80                                  sepos1 <= '1';
81                                  sepos2 <= '1';
82                                  sepos3 <= '1';
83                                  sepos4 <= '1';
84                              end if;
85                          end if;
86                      end if;
87                 end if;  --end if;
88  end process;
89  end Behavioral;
90
91
92  --Componente
93  library IEEE;
94  use  ieee.std_logic_1164.all;
95  package ComparadorPack is
96          component comparador
97                  port (--CLK : in STD_LOGIC;
98                 yprima : in STD_LOGIC;
99                 y : in STD_LOGIC;
100                activo : in STD_LOGIC;
101                sepos1 : out STD_LOGIC;
102                seneg1 : out STD_LOGIC;
103                sepos2 : out STD_LOGIC;
104                seneg2 : out STD_LOGIC;
105                sepos3 : out STD_LOGIC;
106                seneg3 : out STD_LOGIC;
107                sepos4 : out STD_LOGIC;
108                seneg4 : out STD_LOGIC;
109                err1 : in STD_LOGIC;
110                err2 : in STD_LOGIC;
111                err3 : in STD_LOGIC;
112                err4 : in STD_LOGIC
113                         );
114         end component;
115 end package ;
```

B.7. Código perceptrón (NAND)

```
1
2   --Juan Pablo Hernandez Castillo
3   --Perceptron
4   --Proyecto terminal , Implementacion de una red Booleana sobre un FPGA
5
6   --Bibliotecas de la IEEE
7   library IEEE;
8   use IEEE.STD_LOGIC_1164.all;
9   USE IEEE.NUMERIC_STD.all;
10  use work.MayorPack.all;
11  use work.AltoPack.all;
12  use work.DivPerPack.all;
13
14  entity perceptron is
15          port(
16                          X1: in std_logic;--Entrada x1
17                          X2: in std_logic;--Entrada x2
18                          e_pos1: in std_logic;--Entrada de error positiva 1
19                          e_pos2: in std_logic;--Entrada de error positiva 2
20                          e_neg1: in std_logic;--Entrada de error negativa 1
21                          e_neg2: in std_logic;--Entrada de error negativa 2
22                          CLK: in std_logic; --Pulso de reloj
23                          seleccion: in STD_LOGIC;
24                          Y: out std_logic;--Salida y
25                          e_sal1: out std_logic; --Error de salida
26                          est1: out STD_LOGIC_VECTOR (1 downto 0);--estado neurona1
27                          est2: out STD_LOGIC_VECTOR (1 downto 0);--estado neurona2
28          e_sal2: out std_logic --Error de salida
29          );
30  end perceptron;
31
32  architecture bhe of perceptron is
33  signal SCLK : std_logic ;
34  signal SY1: std_logic;
35  signal SY2: std_logic;
36  begin
37      SCLK <= CLK;        unidad1: alto port map(X1,e_pos1,e_neg1,SCLK,SY1,est1,e_sal1);--Mapeo a el
         modulo alto1
38          unidad2: alto port map(X2,e_pos2,e_neg2,SCLK,SY2,est2,e_sal2);--Mapeo a el modulo alto1
39          unidad3: Mayor port map(SY1,SY2,seleccion,Y);--Mapeo a el modulo Mayor
40  end bhe;
41
42  --componente
43  library IEEE;
44  use  ieee.std_logic_1164.all;
45  package perceptronPack is
46          component perceptron
47                  port(
48                          X1: in std_logic;--Entrada x1
49                          X2: in std_logic;--Entrada x2
50                          e_pos1: in std_logic;--Entrada de error positiva 1
51                          e_pos2: in std_logic;--Entrada de error positiva 2
52                          e_neg1: in std_logic;--Entrada de error negativa 1
53                          e_neg2: in std_logic;--Entrada de error negativa 2
54                          CLK: in std_logic; --Pulso de reloj
55                  seleccion: in STD_LOGIC;
56                          Y: out std_logic;--Salida y
57                          e_sal1: out std_logic; --Error de salida
58                  est1: out STD_LOGIC_VECTOR (1 downto 0);--estado neurona 1
59                  est2: out STD_LOGIC_VECTOR (1 downto 0);--estado neurona 2
```

```
60                              e_sal2: out std_logic --Error de salida
61                              );
62              end component;
63     end package;
```

B.8. Código aprende (XOR)

```
--Juan Pablo Hernandez Castillo
--Perceptron
--Proyecto terminal , Implementacion de una red Booleana sobre un FPGA

--Bibliotecas de la IEEE
library IEEE;
use IEEE.STD_LOGIC_1164.all;
USE IEEE.NUMERIC_STD.all;
use work.perceptronPack.all;
use work.ComparadorPack.all;
use work.acoludorpulsoderelojPack.all;
use work.displaysPack.all;
use work.dispaynPack.all;

entity Apren is
    Port ( CLOCK_50 : in STD_LOGIC;
           SW : in STD_LOGIC_VECTOR(4 DOWNTO 0);
                          -- SW(4). Selecciona AND (si '1') o OR (si '0') para el modulo mayor
                          -- SW(3). Switch que activa el aprendizaje de la red (si '1')
                          -- SW(2) = X1, SW(1) = X2, y SW(0) = YP, para insertar el ejemplo a
                              aprender
           KEY : in STD_LOGIC_VECTOR(0 DOWNTO 0); -- Push-button para correr paso a paso el
               aprendizaje
                          HEX5 : out STD_LOGIC_VECTOR (6 downto 0);
                          HEX4 : out STD_LOGIC_VECTOR (6 downto 0);
           HEX3 : out STD_LOGIC_VECTOR (6 downto 0);
           HEX2 : out STD_LOGIC_VECTOR (6 downto 0);
           HEX1 : out STD_LOGIC_VECTOR (6 downto 0);
           HEX0 : out STD_LOGIC_VECTOR (6 downto 0);
                          LEDR : out STD_LOGIC_VECTOR (4 DOWNTO 0);
           LEDG : out STD_LOGIC_VECTOR (7 downto 0));
end Apren;

architecture Behavioral of Apren is
signal sep1: std_logic;
signal sep2: std_logic;
signal sen1: std_logic;
signal sen2: std_logic;
signal ses1: std_logic;
signal ses2: std_logic;
signal sy: std_logic;
signal SCLK: std_logic;
signal displayuno: STD_LOGIC_VECTOR ( 5 downto 0);
signal displaydos: STD_LOGIC_VECTOR ( 5 downto 0);
signal dpstd1: STD_LOGIC_VECTOR (1 downto 0);
signal dpstd2: STD_LOGIC_VECTOR (1 downto 0);

begin
    Unidad1: perceptron port map (SW(2),SW(1),sep1,sep2,sen1,sen2,SCLK,SW(4),sy,ses1,dpstd1,dpstd2
        ,ses2);
         -- HEX3 despliega los 4 bits mas significativos del numero pseudoaleatorio de nuerona 1
         -- HEX2 despliega los 2 bits menos significativos del numero pseudoaleatorio de nuerona 1
    Unidad2: displays port map(displayuno,HEX3,HEX2);
         -- HEX1 despliega los 4 bits mas significativos del numero pseudoaleatorio de nuerona 2
         -- HEX0 despliega los 2 bits menos significativos del numero pseudoaleatorio de nuerona 2
    Unidad3: displays port map(displaydos,HEX1,HEX0);
         -- y' y activo e1+ e1- e2+ e2-
    Unidad4: comparador port map (sy,SW(0),SW(3),sep1,sen1,sep2,sen2,ses1,ses2);
         LEDG(5 downto 4) <= dpstd1;
         LEDG(3 downto 2) <= dpstd2;
```

```
58      LEDG(0) <= sy;
59           LEDG(1) <= '0';
60           LEDG(6) <= NOT(KEY(0));
61           LEDG(7) <= SCLK;
62      LEDR <= SW;
63  end Behavioral;
```

B.9. Código perceptrón (NAND)

```
1  --Juan Pablo Hernandez Castillo
2  --Perceptron
3  --Proyecto terminal , Implementacion de una red Booleana sobre un FPGA
4
5  --Bibliotecas de la IEEE
6  library IEEE;
7  use IEEE.STD_LOGIC_1164.all;
8  USE IEEE.NUMERIC_STD.all;
9  use work.MayorPack.all;
10 use work.AltoPack.all;
11 use work.DivPerPack.all;
12
13 entity perceptron is
14         port(
15                         X1: in std_logic;--Entrada x1
16                         X2: in std_logic;--Entrada x2
17                         e_pos1: in std_logic;--Entrada de error positiva 1
18                         e_pos2: in std_logic;--Entrada de error positiva 2
19                         e_pos3: in std_logic;--Entrada de error positiva 3
20                         e_pos4: in std_logic;--Entrada de error positiva 4
21                         e_neg1: in std_logic;--Entrada de error negativa 1
22                         e_neg2: in std_logic;--Entrada de error negativa 2
23                         e_neg3: in std_logic;--Entrada de error negativa 3
24                         e_neg4: in std_logic;--Entrada de error negativa 4
25                         CLK: in std_logic; --Pulso de reloj
26                         Y: out std_logic;--Salida y
27                         e_sal1: out std_logic; --Error de salida
28                         e_sal3: out std_logic; --Error de salida
29                         e_sal4: out std_logic; --Error de salida
30                         est1: out STD_LOGIC_VECTOR (1 downto 0);
31                         est2: out STD_LOGIC_VECTOR (1 downto 0);
32                         est3: out STD_LOGIC_VECTOR (1 downto 0);
33                         est4: out STD_LOGIC_VECTOR (1 downto 0);
34             e_sal2: out std_logic --Error de salida
35         );
36 end perceptron;
37
38 architecture bhe of perceptron is
39 signal SCLKA1 : std_logic ;
40 signal SCLKA2 : std_logic ;
41 signal SCLK : std_logic ;
42 signal SY1: std_logic;
43 signal SY2: std_logic;
44 signal SY3: std_logic;
45 signal SY4: std_logic;
46 signal SYM1: std_logic;
47 signal SYM2: std_logic;
48 signal sx1: std_logic;
49 signal sx2: std_logic;
50 begin
51     SCLK <= CLK;
52     sx1 <= X1;
53     sx2 <= X2;
54         unidad1: alto port map(sx1,e_pos1,e_neg1,SCLK,SCLK,SY1,est1,e_sal1);--Mapeo a el modulo
               alto1
55         unidad2: alto port map(sx2,e_pos2,e_neg2,SCLK,SCLK,SY2,est2,e_sal2);--Mapeo a el modulo
               alto1
56         unidad3: Mayor port map(SY1,SY2,'1',SYM1);--Mapeo a el modulo Mayor
57         unidad4: alto port map(sx1,e_pos3,e_neg3,SCLK,SCLK,SY3,est3,e_sal3);--Mapeo a el modulo
               alto1
```

```
58          unidad5: alto port map(sx2,e_pos4,e_neg4,SCLK,SCLK,SY4,est4,e_sal4);--Mapeo a el modulo
                alto1
59          unidad6: Mayor port map(SY3,SY4,'1',SYM2);--Mapeo a el modulo Mayor
60          unidad7: Mayor port map(SYM1,SYM2,'0',Y);--Mapeo a el modulo Mayor
61
62  end bhe;
63
64  --componente
65  library IEEE;
66  use  ieee.std_logic_1164.all;
67  package perceptronPack is
68          component perceptron
69                  port(
70                          X1: in std_logic;--Entrada x1
71                          X2: in std_logic;--Entrada x2
72                          e_pos1: in std_logic;--Entrada de error positiva 1
73                          e_pos2: in std_logic;--Entrada de error positiva 2
74                          e_pos3: in std_logic;--Entrada de error positiva 3
75                          e_pos4: in std_logic;--Entrada de error positiva 4
76                          e_neg1: in std_logic;--Entrada de error negativa 1
77                          e_neg2: in std_logic;--Entrada de error negativa 2
78                          e_neg3: in std_logic;--Entrada de error negativa 3
79                          e_neg4: in std_logic;--Entrada de error negativa 4
80                          CLK: in std_logic; --Pulso de reloj
81                          Y: out std_logic;--Salida y
82                          e_sal1: out std_logic; --Error de salida
83                          e_sal3: out std_logic; --Error de salida
84                          e_sal4: out std_logic; --Error de salida
85                  est1: out STD_LOGIC_VECTOR (1 downto 0);
86                  est2: out STD_LOGIC_VECTOR (1 downto 0);
87                  est3: out STD_LOGIC_VECTOR (1 downto 0);
88                  est4: out STD_LOGIC_VECTOR (1 downto 0);
89                          e_sal2: out std_logic --Error de salida
90                          );
91          end component;
92  end package;
```

B.10. Código aprende (XOR)

```
1   --Juan Pablo Hernandez Castillo
2   --Perceptron
3   --Proyecto terminal , Implementacion de una red Booleana sobre un FPGA
4
5   --Bibliotecas de la IEEE
6   library IEEE;
7   use IEEE.STD_LOGIC_1164.all;
8   USE IEEE.NUMERIC_STD.all;
9   use work.perceptronPack.all;
10  use work.ComparadorPack.all;
11  use work.acoludorpulsoderelojPack.all;
12
13  entity Apren is
14      Port ( CLOCK_50 : in STD_LOGIC;
15             SW : in STD_LOGIC_VECTOR(17 DOWNTO 0);
16             KEY : in STD_LOGIC_VECTOR(0 DOWNTO 0); -- Push-button para correr paso a paso el
                   aprendizaje
17             HEX5 : out STD_LOGIC_VECTOR (6 downto 0);
18                     HEX4 : out STD_LOGIC_VECTOR (6 downto 0);
19             HEX3 : out STD_LOGIC_VECTOR (6 downto 0);
20             HEX2 : out STD_LOGIC_VECTOR (6 downto 0);
21             HEX1 : out STD_LOGIC_VECTOR (6 downto 0);
22             HEX0 : out STD_LOGIC_VECTOR (6 downto 0);
23                     LEDR : out STD_LOGIC_VECTOR (17 DOWNTO 0);
24             LEDG : out STD_LOGIC_VECTOR (7 downto 0));
25  end Apren;
26
27  architecture Behavioral of Apren is
28  signal sep1: std_logic;
29  signal sep2: std_logic;
30  signal sep3: std_logic;
31  signal sep4: std_logic;
32  signal sen1: std_logic;
33  signal sen2: std_logic;
34  signal sen3: std_logic;
35  signal sen4: std_logic;
36  signal ses1: std_logic;
37  signal ses2: std_logic;
38  signal ses3: std_logic;
39  signal ses4: std_logic;
40  signal sy: std_logic;
41  signal SCLK: std_logic;
42  signal displayuno: STD_LOGIC_VECTOR ( 5 downto 0);
43  signal displaydos: STD_LOGIC_VECTOR ( 5 downto 0);
44  signal dpstd1: STD_LOGIC_VECTOR (1 downto 0);
45  signal dpstd2: STD_LOGIC_VECTOR (1 downto 0);
46  signal dpstd3: STD_LOGIC_VECTOR (1 downto 0);
47  signal dpstd4: STD_LOGIC_VECTOR (1 downto 0);
48
49  begin
50            -- SW(0) = X1, SW(1) = X2
51            --SW(2) = etiqueta
52            --SW(3) = Aprendizaje activo
53
54
55      Unidad1: perceptron port map (SW(0),SW(1),sep1,sep2,sep3,sep4,sen1,sen2,sen3,sen4,CLOCK_50,sy,
            ses1,ses3,ses4,dpstd1,dpstd2,dpstd3,dpstd4,ses2);
56          Unidad2: comparador port map(SW(2),sy,SW(3),sep1,sen1,sep2,sen2,sep3,sen3,sep4,sen4,ses1,
                ses2,ses3,ses4);
57
```

```
58        --leds 4 y 5  estado de la neurona 2
59        --leds 3 y 2  estado de la neurona 1
60             LEDG(5 downto 4) <= dpstd2;
61             LEDG(3 downto 2) <= dpstd1;
62 
63             --led 4 y 5 estado de la neurona 4
64             --led 3 y 2 estado de la neurona 3
65             LEDR(5 downto 4) <= dpstd4;
66             LEDR(3 downto 2) <= dpstd3;
67 
68             LEDG(0) <= sy;
69             LEDG(1) <= '0';
70             LEDG(6) <= NOT(KEY(0));
71             LEDG(7) <= CLOCK_50;
72        --LEDR <= SW;
73             HEX4 <= "1111111";
74             HEX5 <= "1111111";
75   end Behavioral;
```