RECONOCIMIENTO DE PALABRAS AISLADAS MEDIANTE REDES NEURONALES SOBRE FPGA

JAIME VARELA RINCÓN
JOHAN ERIC LOAIZA PULGARÍN

Universidad Tecnológica de Pereira
Programa de Ingeniería Eléctrica
Facultad de Ingenierías: Eléctrica, Electrónica, Física y de Sistemas
Pereira
2008

RECONOCIMIENTO DE PALABRAS AISLADAS MEDIANTE REDES NEURONALES SOBRE FPGA

JAIME VARELA RINCÓN
JOHAN ERIC LOAIZA PULGARÍN

Trabajo de Grado para optar al título de
Ingeniero Electricista

Directores
Ing. José Alfredo Jaramillo Villegas
Ing. Álvaro Ángel Orozco

Universidad Tecnológica de Pereira
Programa de Ingeniería Eléctrica
Facultad de Ingenierías: Eléctrica, Electrónica, Física y de Sistemas
Pereira
2008

inves

Los sabios son los que buscan la sabiduría,
los necios piensan ya haberla encontrado
Napoleón Bonaparte ($1769 - 1821$*)*

A mi Papá por creer en mí y nunca dejarme desfallecer,
a mi Mamá por su apoyo y amor incondicional en todo el sentido de la palabra,
a mi Hermanita por ser una fuente de cordura en mis momentos de locura,
y en especial a los dos amores de mi vida:
a mi Esposa, que ha sido mi pilar y la fuente de mis energías inagotables,
y a mi bebé, por ser mi razón de ver hacia adelante con muchos deseos de ser alguien
Johan Eric Loaiza P.

Como un testimonio de gratitud ilimitada a mis padres, mi tía y mi hermano.
A quienes sin escatimar esfuerzo alguno,
han sacrificado gran parte de su vida para formarme y educarme.
A quienes la ilusión de su vida ha sido convertirme en persona de provecho.
A quienes nunca podré pagar todos sus desvelos
ni aún con las riquezas más grandes del mundo...
en adelante pondré en práctica mis conocimientos
y el lugar que en mi mente ocuparon los libros, ahora será de ustedes,
esto, por todo el tiempo que les robé pensando en mí.
Gracias por lo que hemos logrado...
Jaime Varela R.

Índice general

Índice de figuras

Índice de cuadros

Resumen

Las redes neuronales artificiales (ANN) se han usado exitosamente para tareas de reconocimiento de voz. El nivel de paralelismo de las ANN es ampliamente significativo, tanto que no es explotado cuando su implementación se realiza en computadores de propósito general. Se busca diseñar un sistema de reconocimiento de palabras aisladas en un computador de propósito general y diseñar en VHDL la ANN del sistema de clasificación apta para la implementación sobre una FPGA, para aprovechar su paralelismo y comparar los resultados del desempeño, eficiencia y confiabilidad.

Se presentan dos desarrollos, ambos con la misma red neuronal pero en un procesador de propósito general y un diseño en VHDL apto para su implementación en una FPGA. Ambos diseños se calificaron teniendo en cuenta su rendimiento tanto en acierto como en velocidad de procesamiento.

El acierto se juzgó haciendo uso del método de validación *leave one out* y la velocidad se juzgó dependiendo del tiempo para procesar paquetes de palabras.
El resultado final son diseños iguales, con aciertos similares pero con velocidades de procesamiento realmente diferentes.

Abstract

Artificial neural networks (ANN) have been used successfully in speech recognition tasks. The parallelism level of the ANN is widely significant, as much as is not exploded when its implementation has been made in general purpose processors. The main task is to design an isolated words recognition system in a general purpose computer and to design on VHDL the ANN of the classification system adequate to its implementation on a FPGA, to take advantage of its parallelism and to compare the performance, efficiency and reliability of both systems.

In this document two developments are presented, both with the same neural network but in a general purpose processor and a design on VHDL skillful to its implementation on a FPGA. Both designs were qualified based on both performance skill and processing speed.

The accuracy (skill) was judged by using the leave one out validation method and the speed were judged based on the time that take the processing of a word package.
The final result are the same designs with similar accuracy but with a processing speed really different from each other.

Agradecimientos

Queremos agradecer a nuestros directores de tesis Ingeniero José Alfredo Jaramillo y al Ingeniero Álvaro Ángel Orozco por su constante apoyo y guía durante el tiempo que transcurrió este proyecto.

También queremos agradecer a los integrantes del *Grupo de investigación Sirius HPC* y al *Grupo de Instrumentación y Control* por su colaboración desinteresada en algunas etapas del proyecto.

Finalmente, queremos agradecer a todas las personas que directa o indirectamente han contribuido a la realización de este trabajo de grado.

Objetivos

General

Desarrollar un sistema de reconocimiento de palabras aisladas sobre un procesador de propósito general y diseñar en VHDL la ANN del sistema de clasificación apta para la implementación sobre una FPGA.

Específicos

- Desarrollar en un procesador de propósito general las etapas de adquisición, preprocesamiento, caracterización y clasificación de las señales de voz.
- Diseñar en VHDL un módulo de clasificación basado en redes neuronales cuya estructura computacional sea adecuada para ser implementada en una FPGA.
- Comparar los rendimientos de los algoritmos de redes neuronales implementados sobre un computador de propósito general con un diseño en VHDL.

Justificación

Un sistema de reconocimiento de voz es básicamente un sistema de reconocimiento de patrones dedicado para detectar voz o, en otras palabras, para identificar un mensaje dentro de una señal de audio adquirida como entrada del medio ambiente.

Tanto las redes neuronales (ANN) como los modelos ocultos de Markov (HMM) se han usado exitosamente para tareas de diferentes complejidades en reconocimiento de voz, desde reconocimiento de palabras aisladas [11] hasta reconocimiento de voz continua. Ambas técnicas, ANN y HMM, tienen un nivel de paralelismo significativo y consisten principalmente de operaciones lógicas sobre operadores de punto fijo [28]. Cuando estas técnicas se implementan en computadores de propósito general, los tiempos de ejecución de los procesos son altos, haciendo que ciertas aplicaciones de reconocimiento de voz, que requieren adecuados niveles de desempeño (en términos de velocidad y confiabilidad), vean reducida su calidad.

Por otra parte, el uso de dispositivos de lógica programable como las FPGAs, dedicados para la aceleración de algoritmos, es una tarea que ha empezado a tomar importancia en el entorno científico. Su gran fortaleza consiste en la posibilidad de implementar de manera sencilla algoritmos que naturalmente son paralelos o que pueden ser fácilmente paralelizables[1].
Los algoritmos que han sido probados para el reconocimiento de patrones de voz tienen un alto nivel de desempeño en computadores basados en procesadores de propósito general, cuya lógica de operación es secuencial. Las necesidades del mundo moderno de acelerar los procesos y reducir costos, han hecho que los desarrollos de investigación se centren en buscar la forma de aplicar modelos teóricos en dispositivos físicos capaces de realizar las tareas solicitadas en el menor tiempo posible, con la mayor confiabilidad que se disponga y con el mayor grado de precisión y desempeño que se pueda alcanzar. Esto hace necesario utilizar elementos de *Hardware* capaces de realizar en forma más eficiente y a altas velocidades procesos que demorarían más tiempo en un procesador de propósito general.

Aunque los niveles de desempeño y precisión de los métodos de reconocimiento de patrones de voz (háblese en este caso de Redes Neuronales), han alcanzado niveles muy altos, se pre-

tende mejorar los niveles de confiabilidad y alcanzar, o probar que, los niveles de precisión teóricos son tan altos como se han mostrado en otros lugares [26].

Con miras a lograr aplicaciones de uso comercial, el desarrollo de este tipo de algoritmos, en dispositivos tales como FPGAs, representa un reto permanente.

Capítulo 1

Resumen del reconocimiento de voz

Si se piensa un poco acerca del reconocimiento de voz, se descubrirá que la cantidad de información que se necesita para realizar este trabajo es muy amplia, es prácticamente imposible obtener una visión completa de todo. Debido a su complejidad matemática, física, informática y lingüística se tendría que hacer un estudio previo de todos estos campos, lo que conllevaría un alto tiempo de desarrollo.
Es por eso que este capítulo se limitará a describir los procesos de una manera superficial, haciendo hincapié en uno de los tantos métodos usados para esta tarea: Las Redes Neuronales Artificiales (**ANN**).

1.1. Fundamentos del reconocimiento de voz

Es bueno empezar por la pregunta más simple y común al momento de comenzar a trabajar en este tema: ¿Qué es el reconocimiento de voz?

Al hablar de reconocimiento de voz, se puede imaginar innumerables campos de aplicación; desde la *domótica* hasta la *inteligencia artificial*, y sobre todo cabe preguntarse: ¿Se empleará reconocimiento de palabras aisladas o de voz continua?, ¿será dependiente o independiente del locutor?, ¿tendrá gramática restringida?. Todo dependerá de la aplicación que se quiera. Por ejemplo, si se quiere un sistema que pueda apagar o encender las luces de la casa, está claro que bastará con un número limitado de palabras que servirán de patrones a identificar con las entradas para satisfacer las necesidades del problema.

Si se piensa en un problema de inteligencia artificial, como hacer que un robot nos entienda, el nivel de complejidad del sistema se elevaría de forma exponencial hasta niveles muy altos. Debido a que se debe tratar un vocabulario completo y no sólo eso, sino que se pueda hablar

con naturalidad y que el sistema identifique las palabras, las frases y el significado [6].

Un sistema de reconocimiento de voz puede trabajar identificando:

- Palabras aisladas. (Que es el caso de este trabajo de grado).
- Fonemas. (Que implican un mayor grado de complejidad).

Cuando se va a analizar una señal de voz, no se hace fonema por fonema, sino que se segmenta la señal, dividiéndola en varias etapas de aproximadamente 20 ms, y se van analizando las frecuencias y sus variaciones. Ésto debido a que la señal de voz puede caracterizarse como un proceso no estacionario sobre intervalos de tamaño variable en función del sonido que se está pronunciando. Al descomponer la señal en estas etapas de 20 ms se consigue que la señal de voz asuma la propiedad de cuasi-estacionariedad [16]

La dificultad empieza a nacer, cuando se puede observar que al pronunciar palabras como *si***e***t***e** y *nu***e***v***e** hay cuatro letras señaladas que parecen ser la misma, pero la pronunciación de al menos dos de ellas es diferente. Aunque sea la **e**, depende mucho de donde se le ubique, qué es lo que la precede y bajo qué estado de ánimo se le pronuncia. Es decir, se debe **predecir** de alguna manera que tipo de **e** es.

1.1.1. Aparato fonador

Partiendo del conocimiento de la producción de sonidos por las cuerdas vocales, hay que tener en cuenta qué es lo que distingue las vocales y las consonantes.
Si se acuden a referencias fonéticas, se encontrará que existen ciertas consonantes oclusivas, fricativas, etc. Y esto influye mucho en el traspaso de la señal en el dominio del tiempo al dominio de la frecuencia. La posición de la lengua, la apertura de la boca y los labios, forman un conjunto fonético que consigue emitir un numero casi infinito de sonidos. En el lenguaje castellano se acotan dichos sonidos para poder construir un lenguaje bastante ordenado. Sin embargo otros idiomas recogen otros sonidos producidos por el aparato fonador que difieren bastante del castellano[6].

Si se emite un sonido constante y sólo se cambia de posición la lengua dentro de la boca, se podrá notar que de cierta manera, se produce el mismo sonido pero se cambian sus distribuciones armónicas, a estas variaciones se les llama **formantes**.

1.1.2. Frecuencia fundamental

Es muy difícil encontrar en la naturaleza un sonido que este compuesto por una única frecuencia, siendo por lo general una suma de frecuencias armónicas, múltiplos de la fundamental. A esta frecuencia fundamental o primer armónico, se le llama tono objetivo del sonido[29].

Los términos tono o altura se refieren a una cualidad de la sensación sonora que nos permite distinguir ente un sonido grave o bajo, de otro agudo o alto. El tono se eleva al aumentar la frecuencia.

Se denomina melodía a un sonido que esta compuesto por una sucesión de tonos de una determinada duración, y que tienen una determinada velocidad de cadencia, de forma que resulta agradable a la mayoría de los oyentes.

Cuando se realiza una mezcla de sonidos con tonos diferentes y simultáneos se produce una armonía; si la mezcla la forman sonidos cuyas frecuencias fundamentales guardan relación numéricas sencillas, la armonía será agradable y recibe el nombre de consonancia; en caso contrario forman lo que se llama una disonancia[6].

1.1.3. Formantes

Son frecuencias que entran en resonancia en las cavidades nasales y orales, saliendo hacia el exterior como la información más importante del habla.
Un formante es el pico de intensidad en el espectro de un sonido; se trata de concentración de energía (amplitud de onda) que se da en una determinada frecuencia. En el habla se determinan por el proceso de filtrado que se produce en el tracto vocal por la configuración de los articuladores.

Los formantes son los elementos que sirven para distinguir componentes del habla humana, sobre todo las vocales y sonidos sonantes. Dichos elementos se producen por la resonancia del tracto vocal, pero algunos tonos silbantes se derivan del colapso periódico de zonas de baja presión debido al efecto Venturi. El formante con la frecuencia más baja se llama F1, el segundo F2, el tercero F3, etc. Normalmente sólo los dos primeros son necesarios para caracterizar una vocal, si bien la pueden caracterizar hasta seis formantes. Los posteriores determinan propiedades acústicas como el timbre.
Los dos primeros se determinan principalmente por la posición de la lengua. F1 tiene una frecuencia más alta cuanto más baja está la lengua, es decir, cuanta mayor abertura tenga una vocal, mayor es la frecuencia en que aparece el F1. El F2 tiene mayor frecuencia cuanto más

hacia delante está posicionada la lengua, es decir, cuanto más anterior es una vocal, mayor es el F2.
No todos los sonidos humanos se componen de formantes definidos, éstos sólo aparecen en sonantes, que incluyen, los sonidos pulmonares que contienen vocales, aproximantes y nasales. Las nasales tienen un formante adicional (F3) en torno a los $1{,}5$ kHz. Las consonantes róticas, por su parte, presentan pequeñas oclusiones y, en el caso de la múltiplie (**r** como en río) aparecen formantes vocálicos.

Si la frecuencia fundamental es mayor que la frecuencia de los formantes, entonces el carácter del sonido se perderá, de manera que, por ejemplo, en el canto de una soprano las vocales suelen ser difíciles de distinguir [6].

Se debe pensar un poco en todo esto y hacer preguntas acerca de las vocales y consonantes, y saber además que la diferencia entre una **a** y una **e** reside en la ubicación de sus formantes. El cerebro humano analiza esas frecuencias y su relación entre ambas, y supone que vocal puede ser. Eso se puede traducir como una operación probabilística que realizan las neuronas concluyendo en una decisión final.

1.2. Reconocimiento de voz utilizando técnica de comparación de patrones

La ventaja inmediata del reconocimiento de voz utilizando técnica de comparación de patrones, reside en que no es necesario descubrir características espectrales de la voz a nivel fonético, lo que evita el tener que desarrollar etapas complejas de detección de formantes, de rasgos distintivos de los sonidos, de los tonos de voz, etc.

Esto está muy bien para un número finito de palabras, cuyo número no sea muy grande. Si se quiere desarrollar este sistema para un complejo entendimiento del lenguaje sería una completa locura, además de ser casi inútil. Por ejemplo si se pronuncia la palabra *queso*, se tendría que hacer exactamente igual que al momento de grabarla. Además se tendría que pronunciar con la misma velocidad y el mismo tono, sino el sistema no podría identificar la palabra. Para ello se necesita un sistema que aprenda por sí mismo y que al final intente estipular que palabra es.

De todos modos, existe la necesidad de aplicar este sistema única y exclusivamente a casos donde el vocabulario sea pequeño y limitado.
También se pueden constituir los grupos de patrones por unidades tales como sonidos básicos

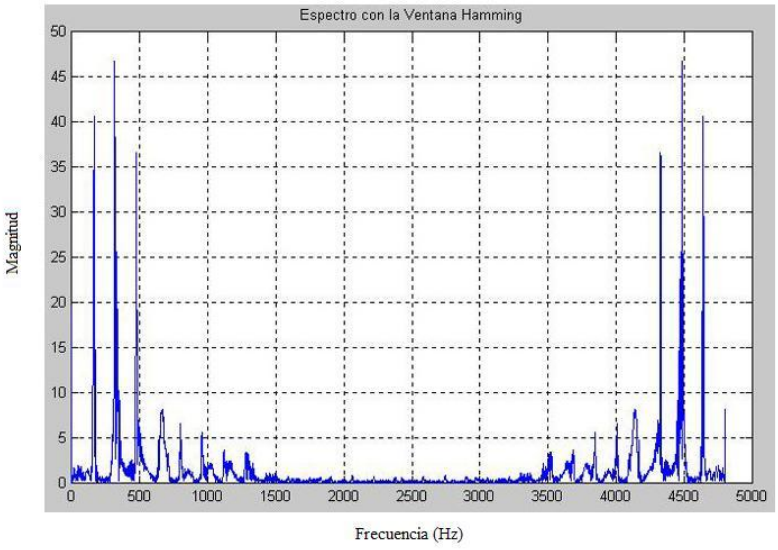

Figura 1.1: Espectro de la ventana Hamming

(fonemas y demás clasificaciones de sonidos cortos). Al grabar estos sonidos en la base de datos, se obtendrán sus características (suele hacerse usando Análisis de Predicción Lineal (LPC) o Coeficientes *Cepstrum* en la escala Mel (MFCC)).

1.2.1. Procesamiento de la voz en el dominio del tiempo

Debido a la naturaleza cambiante de la voz, resulta más conveniente aplicar el análisis a porciones de ésta, ya que el interés primordial es observar la evolución de los distintos parámetros calculados; por ello se procesan segmentos o *ventanas* de la señal.

Pero eso no es todo, cada *ventana* tendrá asociada un peso, es decir, no todas las secciones tendrán la misma importancia. De esta forma las muestras quedan ponderadas con los valores de la función escogida. En la figura 1.1 se ve un ejemplo de función de ventaneo Hamming. Las muestras que se encuentran en los extremos de la ventana tienen un peso mucho menor que las que se hallan en el medio, lo cual es muy adecuado para evitar que las características de los extremos del bloque varíen la interpretación de lo que ocurre en la parte central, que es la más significativa.

Existe un pequeño solapamiento en los extremos de las ventanas, esto se usa para mejorar la calidad de los resultados ya que se hace que los valores de la ventanas en sus extremos

queden muy reducidos. Pero esto repercute en los tiempos de respuesta de los algoritmos utilizados, ya que los alarga ligeramente.

1.2.2. Energía y magnitud

Tanto la energía como la magnitud son útiles para distinguir segmentos sordos y sonoros en la señal de voz.

La "energía" de la voz $E(n)$, se define como la suma de las magnitudes de la voz en intervalos centrados de 10 ms, todo medido sobre el intervalo de medición [23].

$$E(n) = \sum_{i=-50}^{50} |s(n+1)| \tag{1.1}$$

En donde, $s(n)$ son las muestras de voz muestreadas a la frecuencia escogida. El escoger una ventana de 10 ms para el cálculo de la energía y el uso de la función de magnitud en lugar del cuadrado de la magnitud, se realizó por el deseo de realizar los cálculos en aritmética integral y por ende, el aumentar la velocidad de la computación. Existen otras maneras para identificar segmentos sonoros, hay un método denominado *cruces por cero y máximos* donde por ejemplo la **s** provoca que las muestras consecutivas de esa consonante difieren de signo. La tasa de cruces por cero de una grabación de voz $z(n)$ se define como el número de cruces por cero en un intervalo de 10 ms. La función de cruces por cero está dada por la ecuación 4.7 expuesta en [20].

$$Z(n) = \sum_{m=-\infty}^{\infty} |\,[s(m)] - [s(m-1)]\,|r(n-m) \tag{1.2}$$

donde

$$\begin{aligned} [s(n)] &= 1 \quad s(n) \geq 0 \\ &= -1 \quad s(n) < 0 \end{aligned}$$

y

$$\begin{aligned} r(n) &= \frac{1}{2N} \quad 0 \leq n \leq N-1 \\ &= 0 \quad \text{de otra manera} \end{aligned}$$

Aunque la tasa de cruces por cero es altamente susceptible a ruido de 60 Hz, offset, etc; en la mayoría de los casos es una medida muy buena y razonable de la presencia o ausencia de

silencio en la grabación[23].

Generalizando más, se puede decir que una señal clasificada como ruido (la **s** es un ruido de alta frecuencia) provoca en la amplitud un cambio de signo. De esta manera se pueden localizar consonantes fricativas.

1.2.3. Estimación espectral por predicción lineal (LPC)

La importancia de la estimación espectral por predicción lineal (LPC) es mayor que la que su propio nombre sugiere, es una de las técnicas más usadas en el procesamiento de señales de voz.

Esta técnica ha probado ser muy eficiente debido a la posibilidad de parametrizar la señal con un número pequeño de patrones con los cuales es posible reconstruirla adecuadamente. Estos parámetros van cambiando poco a poco en función de la señal en el tiempo. Otra de sus ventajas es que no necesita demasiado tiempo de procesamiento.

El modelo matemático sugiere que el tracto vocal puede modelarse mediante un filtro digital, siendo los parámetros los que determinan la función de transferencia. Esto consiste en que, dado una segmento de palabra, se extraen sus parámetros que en este caso son los *Coeficientes del Filtro.*

El sistema LPC es sin duda una manera acertada de poder hacer un registro de la señal, ya que en vez de registrarla, se transforma todo en un filtro con determinados coeficientes, de manera que al reconstruir la señal, se le ingresa un tren de pulsos periódicos o una fuente de ruido aleatorio que al pasar por el filtro se transformará en la señal original, es decir en la señal de voz. El tren de pulsos producirá señales sonoras y a su vez la fuente de ruido aleatorio producirá señales no sonoras. De esta manera el filtro representa un modelo del tracto vocal.

1.2.4. Modelos de señales

Los modelos que existen para caracterizar señales, pueden clasificarse de modo general en *Determinísticos* y *Estocásticos o Aleatorios*. Esta clasificación se basa en la naturaleza de la señal tratada.

Los modelos determinísticos utilizan propiedades conocidas de la señal para así poder estimarla de forma inmediata. Por ejemplo, una onda senoidal puede caracterizarse por su amplitud, su fase y su frecuencia.

Por otra parte los modelos aleatorios estiman propiedades estadísticas de las señales (Función Gaussiana, Poisson, etc). Se asume que la señal modelada puede caracterizarse como una función de probabilidad, y los parámetros de la función de probabilidad pueden determinarse de manera definida y precisa. Por esto los modelos estocásticos son una aproximación particularmente adecuada para el reconocimiento de voz.

Hasta ahora se han visto los principios básicos, pero quedan muchas cosas por tratar. En este punto es donde se hace necesario mencionar un sistema avanzado como lo son las **Redes Neuronales Artificiales (ANN)**, que será el tema a estudiar en el capítulo siguiente.

Capítulo 2

Introducción a las redes neuronales artificiales (ANN)

En este capítulo se presentará una pequeña introducción a las Redes Neuronales Artificiales (ANN). Después de conocer un poco del desarrollo histórico, se presentarán algunos conceptos fundamentales y se describirán algunos tipos de topologías y los algoritmos de entrenamiento usados en las ANN.

2.1. Desarrollo histórico

El estudio moderno de las Redes Neuronales empieza a finales del siglo XIX, cuando los neurobiólogos empezaron sus estudios acerca del sistema nervioso humano. Santiago Ramón Cajal (1852-1934), determinó, alrededor del año 1892, que el sistema nervioso está compuesto de neuronas discretas, las cuales se comunican entre si, enviando señales eléctricas a través de sus *axones*, que terminan por conectarse con las *dentritas* de miles de otras neuronas; transmitiendo señales eléctricas a través de la *sinapsis*, (la cual es un punto de contacto entre un axón y una dentrita y de una resistencia variable) (figura 2.1)[1]. Este esquema básico fue trabajado durante las décadas siguientes a su descubrimiento, mientras diferentes tipos de neuronas se identificaban y sus patrones de conectividad así como las áreas funcionales del cerebro eran registrados [26]. Mientras muchos de estos neurobiólogos encontraban que era bastante simple determinar el funcionamiento de una sola neurona, se hacia extremadamente difícil determinar como hacían las neuronas para trabajar en conjunto para alcanzar los altos niveles de funcionalidad que eran capaces de dar, entre ellos la capacidad de percibir y aprender. Sin embargo, con el advenimiento de la era de las computadoras de alta velocidad a mediados del siglo XX, finalmente se pudo lograr construir *modelos de sistemas neuronales*,

[1] Imagen tomada de http://www.web-books.com/eLibrary/Medicine/Physiology/Nervous/neuron.jpg

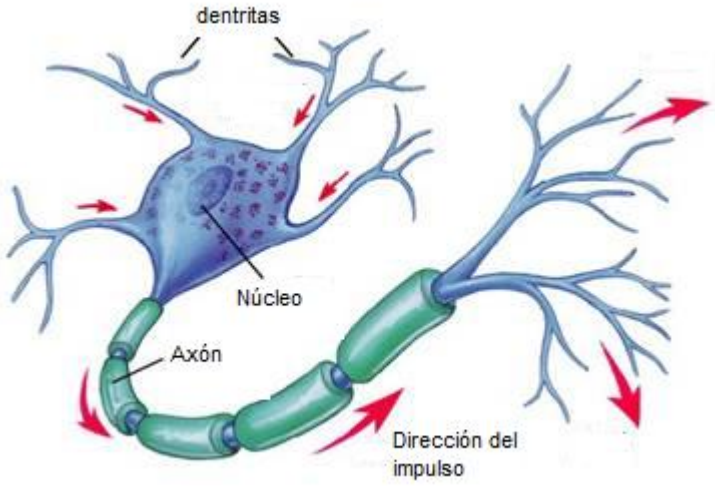

Figura 2.1: Esquema de una neurona biológica

los que permitieron a los investigadores experimentar con dichos sistemas y conocer sus propiedades[5].

Warren McCulloch y Walter Pitts propusieron en el año de 1943 el primer modelo computacional de una neurona llamado *Unidad Binaria de Umbral*, cuyas salidas eran 0 o 1 dependiendo de si la entrada excedía o no un umbral. Este modelo causó gran excitación entre los investigadores de ese entonces, porque se pudo comprobar que un sistema de dichas unidades binarias de umbral, ensambladas en un automata de estados finitos, podía computar cualquier función arbitraria, si se daban valores apropiados a los pesos entre las neuronas [14].

De ahí en adelante fue trabajo de los investigadores el encontrar algún *método de aprendizaje* que pudiera encontrar automáticamente el valor de los pesos entre las conexiones de las neuronas, para que la red pudiera computar cualquier función. Frank Rosenblatt descubrió en 1962 un método iterativo para un tipo particular de red, el *perceptrón de una sola capa* y probó que este método de aprendizaje siempre convergía a un grupo de pesos que reproducían la función deseada, siempre y cuando dicha función pudiera ser procesada por la red[5].

Sin embargo, en un análisis más profundo Marvin Minsky y Seymour Papert demostraron en 1969 que el grupo de funciones computables por el perceptrón es bastante limitada. Dicho descubrimiento hizo que todos los investigadores dejaran sus estudio y sus desarrollos en

Redes Neuronales durante 15 años.

El interés en las ANN fue gradualmente aumentando cuando en 1982 J.J Hopfield sugirió que una Red Neuronal puede ser analizada en términos de una *Función de Energía*. Más tarde se popularizó un método de entrenamiento mucho más rápido conocido como *Backpropagation*, el cual podía entrenar un *Perceptrón Multicapa* para computar cualquier función deseada, demostrando que el pesimismo de Minsky y Papert era sólo infundado. Con el advenimiento del backpropagation, las redes neuronales han aumentado su desarrollo y sobre todo su amplia gama de aplicaciones[26].

2.2. Fundamentos de las redes neuronales artificiales

Existen diferentes tipos de redes neuronales, pero todas poseen cuatro atributos fundamentales:

- *Un grupo de unidades procesadoras*.
- *Un grupo de pesos*.
- *Un método de computación*.
- *Un método de entrenamiento*.

2.2.1. Unidades procesadoras

Una red neuronal contiene un número potencialmente alto de unidades de procesamiento simple, que son bastante análogas a las neuronas en el cerebro humano. Todas estas unidades trabajando simultáneamente, proveen un paralelismo inmenso al momento de procesar información. Toda la computación en el sistema es realizada por estas unidades, no existe ningún procesador que esté pendiente de su funcionamiento[2]. En cada instante de tiempo, cada unidad simplemente computa una función escalar de sus entradas locales y trasmite su resultado a las unidades vecinas.

Las unidades en una Red Neuronal están divididas en *unidades de entrada*, las que se encargan de recibir los datos del ambiente; *unidades ocultas*, las cuales internamente transforman la representación de los datos y las *unidades de salida*, las cuales representan decisiones o señales de control.

[2]Claro está, a no ser de que dicha red se simule en algún computador convencional, en vez de ser implementada directamente en un *Hardware*, (llámese en este trabajo de grado, la FPGA).

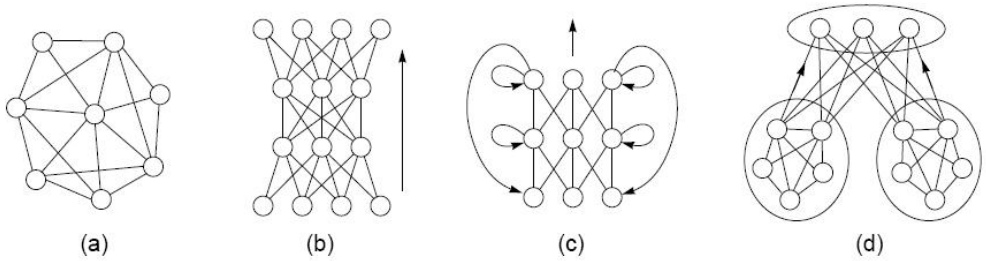

Figura 2.2: (a) Red Auto-asociativa. (b) Red en Capas. (c) Redes Recurrentes. (d) Redes Modulares.

2.2.2. Pesos

Todas las unidades en una red están organizadas en una topología dada por un grupo de *conexiones* o *pesos*, que se muestran como líneas en un diagrama. Cada peso posee un valor real que normalmente está en el rango de $-\infty$ a $+\infty$, aunque lo más común y mejor para la implementación de la ANN, es limitar este rango. El valor de este peso, básicamente lo que indica es que tanta influencia tiene la unidad en su vecina, un peso positivo lo que hace es excitar otra unidad mientras que un peso negativo lo que hace es inhibirla.

El valor de los pesos determina la reacción computacional de la red ante cualquier patron arbitrario a la entrada, esto quiere decir que los pesos se encargan de almacenar *la memoria de largo plazo*, o en otras palabras el *aprendizaje* de la red. Los pesos pueden cambiar como resultado del entrenamiento, pero muchas veces lo hacen de forma lenta, ya que el aprendizaje acumulativo cambia lentamente.

Una red se puede conectar con cualquier tipo de topología. Las más comunes se muestran en la figura 2.2. Cada tipo de topología es usado para algún tipo particular de aplicación. Se pueden mencionar:

- Red Auto-asociativa. Es la más usada para los problemas de completar patrones. (Devuelve un patrón aprendido con sólo mostrarle una parte de este).
- Red en Capas. Es muy usada para la asociación y reconocimiento de patrones.
- Redes Recurrentes. Son bastante útiles para realizar secuencias de patrones.
- Redes Modulares. Sirven para la construcción de sistemas complejos a partir de componentes simples.

La conexión entre las neuronas o unidades de dos capas distintas puede ser de tres tipos: *Completa*, en la cual todas las neuronas están conectadas unas con otras. *Aleatoria*, sólo se conectan algunas con otras y *Locales*, en la que se conecta una con su vecina únicamente. Una red completamente conectada, tiene lo que se conoce como *mayor grado de libertad*, lo que permite que este tipo de redes tenga la capacidad teórica de aprender más funciones que una red más restringida. Sin embargo esto conlleva a un problema grave: El sobreentrenamiento. Supóngase que se invita a un botánico experto en árboles, el cual se ha entrenado durante toda su vida para identificar cualquier árbol por el número de hojas que posee. Su experiencia o su *entrenamiento* le ha hecho *aprender* que para que un árbol sea un árbol debe tener sólo X número de hojas.

Sin embargo se le presenta un espécimen que posee $X + 1$ hojas. El no sabe que es un árbol pero el resto de botánicos si, pero debido a su sobreentrenamiento para él eso no es un árbol. O puede ocurrir lo contrario hablando de entrenamientos, puede ser de que ese botánico este tan mal entrenado para reconocer árboles que el criterio que aplica es que sean verdes. Por lo tanto cualquier elemento verde que se le muestre, para él será un árbol. Este es un problema grave que presentan las Redes Neuronales Artificiales, cuando se sobreentrenan o cuando su entrenamiento es muy pobre.

2.2.3. Computación

La computación siempre inicia presentando una patrón de entrada a la red o imponiendo un patrón de activación en las unidades de entrada. Luego la activación de las unidades siguientes es computada, ya sea de manera *síncrona* (todas al tiempo en un sistema en paralelo) o *asíncrona* (una a una en un sistema secuencial). En las Redes de Capas este proceso se conoce como *propagación hacia adelante*.

Una unidad dada es normalmente actualizada en dos etapas: Primero se computa la *Entrada de la Red*, y luego la *Activación de la salida*, como función de la entrada de la red. En los casos más comunes la entrada global a la red x_j para la unidad j es sólo la suma de las entradas simples pesadas:

$$x_j = \sum_j p_i w_{ji}, \qquad j = 1, 2, 3, ...n \tag{2.1}$$

en donde p_i es la entrada de una neurona y w_{ij} es el peso de la conexión entre la neurona i y la neurona j. Esta es la típica función de entrada de una neurona en donde el operador es la suma $\sum$, pero existen otro tipo de funciones de activación como la productoria $\prod$ y el máximo de las entradas pesadas.

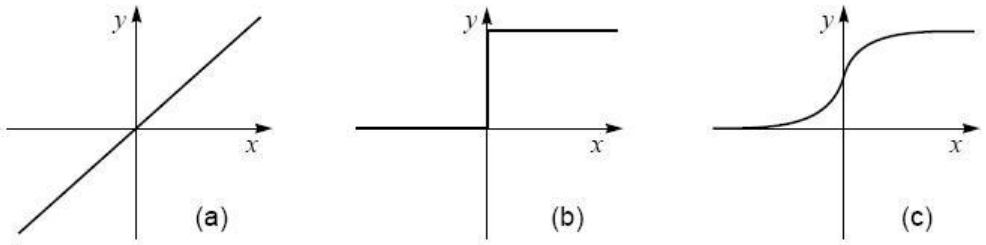

Figura 2.3: (a) Función de activación lineal. (b) Función de activación limitador fuente. (c) Función de activación sigmoidal.

Una vez se tiene calculada la entrada global a la red x_j, se calcula ahora la salida a_j, como un función de x_j. Esta función es conocida como la *función de activación* y puede ser tanto determinística como estocástica, y pueden ser locales o no locales.

Las funciones de activación determinísticas y locales usualmente toman una de tres formas: *lineal, limitador fuente o sigmoidal*, como se pueden ver en la figura 2.3. En el caso lineal se tiene que $a = x$. Esta no es muy usada debido a que no es muy fuerte: Capas multiples de unidades lineales pueden ser reemplazadas por una sola con la misma funcionalidad. En orden de construir funciones no lineales (caso más común de los eventos naturales), una red requiere unidades no lineales. La forma más simple de no linealidad la provee la función de activación *limitador fuente*:

$$a = \begin{cases} 0 & \text{si x} \leq 0 \\ 1 & \text{si x} > 0 \end{cases} \tag{2.2}$$

Ésta es mucho más fuerte que una función lineal, una red multicapa de unidades *limitador fuente* puede, teóricamente, computar cualquier función booleana. Sin embargo es un poco trabajoso entrenar una red de este tipo debido a que la discontinuidad de la función implica que encontrar el grupo de pesos deseados implica una búsqueda exponencial. Existen numerosas aplicaciones en donde las salidas continuas son preferidas a las salidas binarias. En consecuencia la función de activación más común es la *Sigmoidal*.

$$a = \frac{1}{1 + e^{-x}} \quad \text{o} \quad y = \tanh(x) \tag{2.3}$$

Las funciones sigmoidales tienen la ventaja de la no linealidad, la continuidad en el tiempo y la diferenciabilidad, permitiendo a una red multicapa procesar cualquier función real arbitraria. Mientras permite la implementación de un algoritmo práctico de entrenamiento,

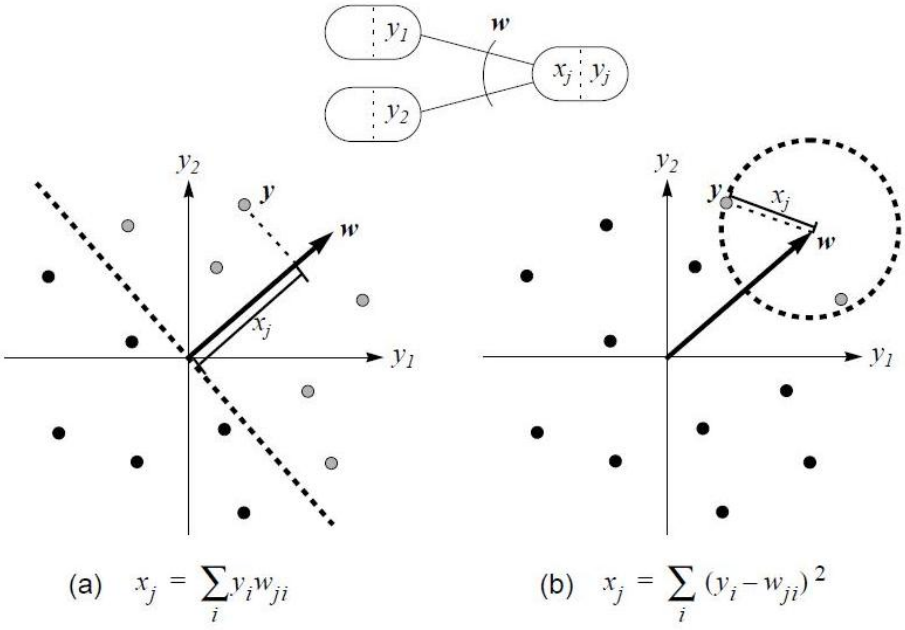

Figura 2.4: Cálculo de la entrada a la red (a) Producto punto = hiperplanos y (b) Diferencias = hiperesferas. Imagen tomada de [26]

backpropagation, basado en el descenso de un gradiente.

En este punto se han tratado las funciones de activación que tienen la forma general:

$$a_j = f(x_j) \quad donde \quad x_j = \sum_j p_i w_{ji} \tag{2.4}$$

Esta es la forma más común de funciones de activación. Sin embargo, otro tipo de redes como las **LVQ** (Learned Vector Quantization) y las **RBF** (Radial Basis Function), incluyen unidades que se basan en otro tipo de funciones de activación, cuya forma general es:

$$a_j = f(x_j) \quad donde \quad x_j = \sum_j (p_i - w_{ji})^2 \tag{2.5}$$

La diferencia entre estos dos tipos de unidades tiene una interpretación geométrica simple que se muestra en la figura 2.4. En el primer caso x_j es el producto punto entre un vector de entradas $\vec{p}$ y un vector de pesos $\vec{w}$; lo que quiere decir que x_j es la proyección de y sobre w. Esta proyección puede apuntar o no en la misma dirección que w, es decir puede estar en uno

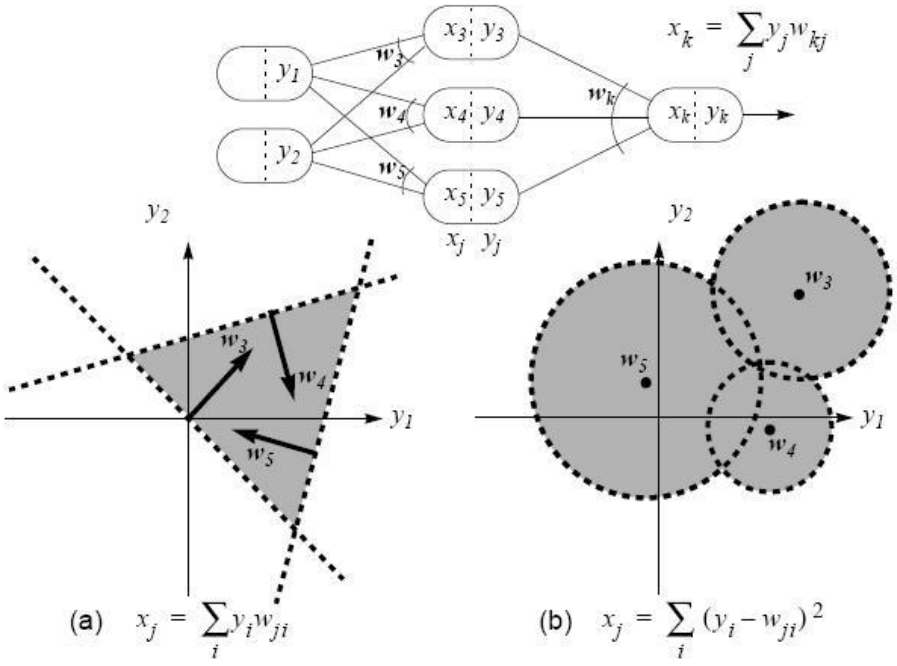

Figura 2.5: Construcción de funciones complejas a partir de (a) hiperplanos y (b) hiperesferas. Imagen tomada de [26]

de dos lados de un hiperplano perpendicular a w. Las entradas que estén en el mismo lado del hiperplano tendrán $x_j > 0$ mientras que las que estén en el otro lado tendrán $x_j < 0$. Es decir, si $a_j = f(x_j)$ es una función limitador fuente como en la ecuación 2.2, la unidad clasificará cada entrada en términos de en cual lado del hiperplano se encuentra.

En contraste, el segundo caso es una distancia Euclidiana entre el vector de entradas y y el vector de pesos w. En este caso, el peso representa el centro de una distribución esférica en el espacio de entradas. La función de distancia puede ser invertida por una función como $y_j = f(x_j) = e^{-x_j}$, por lo que una entrada en la región del centro de la región circula tendrá una activación $y_j = 1$, mientras que una entrada a una distancia lejana tendrá una activación $y_j = 0$.

En cualquiera de los casos, ambas regiones de decisión, los hiperplanos y las hiperesferas (ver figura 2.5) , pueden ser posicionadas en cualquier lugar del espacio de entrada y ser usado para dividirlo en formas arbitrarias. Entonces, cada región de decisión puede ser com-

binada para construir cualquier función arbitraria y compleja, incluyendo al menos, una capa de unidades *limitador fuente* o *sigmoidales*. Esta es la tarea del entrenamiento, ajustar los hiperplanos y/o las hiperesferas para formar un modelo más preciso de la función deseada.

2.2.4. Entrenamiento

Entrenar una red, en el sentido general, significa adaptar las conexiones de ésta para que reproduzca el comportamiento computacional deseado para los patrones de entrada. Este proceso incluye, normalmente, variar los pesos de la red (es decir, mover los hiperplanos y/o las hiperesferas); pero en otras ocasiones puede incluir la modificación de la topología de la red (básicamente añadir o eliminar conexiones entre neuronas). En un sentido general, la modificación de los pesos es más común que la modificación de la topología, ya que una red de muchas conexiones puede "aprender" a poner valores de pesos en cero, lo que asemejaría a eliminar una conexión.

Encontrar un grupo de pesos que permita a una red procesar una función dada es un procedimiento para nada trivial. Una solución analítica existe solo en el caso simple de la asociación de patrones. Por ejemplo cuando la red es lineal y la tarea es asociar un grupo ortogonal de vectores de entrada con unos de salida. En este caso los pesos están dados por:

$$w_{ji} = \sum_{p} \frac{p_i^n t_j^n}{\|p^n\|^2} \tag{2.6}$$

en donde p es el vector de entradas, t el vector de salida esperada y n el índice del patrón.

Pero en casos generales, el proceso de entrenamiento puede dividirse en dos grandes grupos: *Aprendizaje Supervisado* y *Aprendizaje no Supervisado*.

Aprendizaje Supervisado

En este tipo de entrenamiento la red es entrenada entregándole un patrón de entrada y un patrón de salida objetivo. Estas parejas de entrada salida pueden ser entregadas por un entrenador externo o por el sistema que contiene la red [14].

Dentro del aprendizaje supervisado se pueden encontrar tres tipos de aprendizaje: *Aprendizaje por corrección de error*, *Aprendizaje por refuerzo* y *Aprendizaje estocástico*.

1. **Aprendizaje por corrección de error**.
   Básicamente consiste en ajustar los pesos de las conexiones de la red en función de la diferencia entre los valores esperados y los que se obtienen a la salida de la red, es

decir, en función del *error* cometido a la salida.

Se distinguen tres tipos básicos de entrenamiento y se explican a continuación:

- *Regla de Aprendizaje del Perceptrón*. Enunciada por Rosenblatt en 1968 se basa en que:

$$\Delta w_{ij} = \sigma.a_j.\delta_i \tag{2.7}$$

en donde:

$$\begin{aligned} \sigma &= Aprendizaje \\ \delta_i &= (t_i - a_i) \\ t_i &= Salida\ esperada \end{aligned}$$

- *Regla de Aprendizaje Delta* o regla del mínimo error cuadrado. También usa la desviación del objetivo pero tiene en cuenta las neuronas predecesoras que tiene la neurona de salida. Esto permite cuantificar el error global de la red en cualquier momento durante el proceso de entrenamiento de la red, lo cual es importante, ya que cuanto más información se tenga sobre el error cometido, más rápido se puede aprender. Por lo tanto el error calculado (δ) es igualmente repartido entre las conexiones de las neuronas predecesoras.
- *Regla de Backpropagation*. También es conocida como la regla LMS multicapa, la cual es una generalización de la regla de Aprendizaje Delta. Esta regla permite realizar cambios en los pesos de la conexiones de la capa oculta.

2. **Aprendizaje por Refuerzo**.
Se trata de un aprendizaje supervisado, más lento que el anterior, que se basa en la idea de no disponer de un ejemplo completo del comportamiento deseado, es decir, de no indicar durante el entrenamiento exactamente la salida que se desea que proporcione la red ante una determinada entrada.

   En el aprendizaje por refuerzo la función del supervisor se reduce a indicar mediante una señal de refuerzo si la salida obtenida en la red se ajusta a la deseada (éxito $= +1$ o fracaso $= -1$), y en función de ello se ajustan los pesos basándose en un mecanismo de probabilidades. Se podría decir que en este tipo de aprendizaje la función del supervisor se asemeja más a la de un crítico (que opina sobre la respuesta de la red) que a la de un maestro (que indica a la red la respuesta concreta que debe generar), como ocurría en el caso de supervisión por corrección del error.

3. **Aprendizaje Estocástico**
   El aprendizaje estocástico consiste básicamente en realizar cambios aleatorios en los valores de los pesos de las conexiones de la red y evaluar su efecto a partir del objetivo deseado y de distribuciones de probabilidad.

   En el aprendizaje estocástico se suele hacer una analogía en términos termodinámicos, asociando a la red neuronal con un sólido físico que tiene cierto estado energético. En el caso de la red, la energía de la misma representaría el grado de estabilidad de la red, de tal forma que el estado de mínima energía correspondería a una situación en la que los pesos de las conexiones consiguen que su funcionamiento sea el que más se ajusta al objetivo deseado.

   Según lo anterior, el aprendizaje consistiría en realizar un cambio aleatorio de los valores de los pesos y determinar la energía de la red (habitualmente la función energía es una función de Liapunov). Si la energía es menor después del cambio, es decir, si el comportamiento de la red se acerca al deseado, se acepta el cambio; si, por el contrario, la energía no es menor, se aceptaría el cambio en función de una determinada y preestablecida distribución de probabilidades.

Aprendizaje no Supervisado

En el cual una unidad de salida es entrenada para responder a un grupo de patrones dentro de la entrada. En este tipo de entrenamiento el sistema debe descubrir estadísticamente las características del grupo de entradas. A diferencia del entrenamiento supervisado no existe un grupo de categorías *a priori* en el cual los patrones puedan ser clasificados. Por tanto el sistema debe de ser capaz de desarrollar su propia representación de la señal de entrada [13].

Básicamente no requieren influencia externa para el ajuste de los pesos de las conexiones. Estas deben encontrar las características, correlaciones o categorías que puedan haber entre los datos presentados en la entrada a la red. La interpretación depende exclusivamente del algoritmo de aprendizaje usado.

La salida puede mostrar el grado de similitud entre la entrada y la información que le haya presentado a la red en el pasado. En otros casos se puede presentar un establecimiento de categorías, indicando la red a la salida a que categoría pertenece la información presentada a la entrada, siendo la propia red quien debe encontrar la categoría apropiada a partir de las correlaciones entre las informaciones presentadas.

En cuanto a los tipos de entrenamiento se pueden distinguir dos tipos, que son los de uso

más general:

- Aprendizaje Hebbiano.
- Aprendizaje Competitivo y Comparativo.

1. Aprendizaje Hebbiano. Es la técnica más usada y la base de otras muchas. Se basa en medir la familiaridad de las características o extraer las mismas de la entrada. El fundamento es bastante simple, si dos neuronas N_i y N_j toman un mismo estado de forma simultánea, el peso de su conexión se incrementa.
2. Aprendizaje Comparativo y Competitivo. Se caracteriza por clusterización o clasificación de los datos de entrada. Se puede decir que si un patrón nuevo se determina que pertenece a una clase reconocida con anterioridad, entonces la inclusión de este nuevo patrón a la clase, realzará la representación de la misma. Si no se determina su pertenencia a alguna clase, la red reajusta sus pesos para reconocer la nueva clase.

2.3. Redes de propagación hacia adelante (*Feed forward*)

2.3.1. *Red backpropagation*

Una de las redes más comúnmente usadas es la red multicapa de propagación hacia adelante. Esta se encuentra dentro de la categoría de “Redes para Clasificación y Predicción”.

Las redes de propagación hacia adelante son muy ventajosas ya que son los modelos más rápidos para ejecutar y además son usados como los aproximadores universales de funciones. La principal desventaja de este tipo de redes, es que no se ha podido desarrollar un algoritmo confiable y lo suficientemente rápido por lo que se vuelve extremadamente lento para entrenar. Por ende, este tipo de redes debe de ser escogido cuando se necesiten altas velocidades de ejecución, pero los altos tiempos de entrenamiento no sean problema[7].

- Arquitectura Básica
  Las redes *backpropagation* usualmente consisten de tres a cuatro capas en las cuales las neuronas son organizadas lógicamente. La primera y la última capa son las capas de entrada y salida respectivamente, y existen una o más capas ocultas entre las capas mencionadas anteriormente. La experiencia indica que un mínimo de tres capas (una sola capa oculta) se requieren para resolver problemas complejos. El término propagación hacia adelante significa que la información viaja sólo en una dirección. Esto significa que la salida de una capa se vuelve la entrada de la capa siguiente, y así hacia

adelante. Para que esto ocurra, cada capa está completamente conectada con la capa siguiente (Cada neurona está conectada por un peso con la neurona en la capa siguiente).

Debe de ser entendido que la capa de entrada tiene parte dentro de los cálculos, en vez de solo recibir las entradas. Los datos son computados y las funciones de activación son aplicadas en todas las capas. Este proceso ocurre hasta que los datos llegan a la última capa, donde son clasificados dentro de alguna categoría.

- Función de Activación
  La función de activación debe de ser una función no lineal, que cuando se aplique a la suma pesada de las entradas, se determine la salida de la red.

- Entrenamiento *Backpropagation*
  El entrenamiento *Backpropagation* está basado en la Regla Delta (*Delta Rule*) que básicamente dice que si la diferencia (Delta) entre la salida deseada por el programador y la salida actual de la red debe de ser minimizada, los pesos deben de ser continuamente modificados. El resultado de la función de transferencia cambia el *error delta* en la capa de salida. El error en la capa de salida debe de ser ajustado o fijado, y por ende puede ser usado para cambiar los pesos en las conexiones de entrada para que la salida deseada se logre. Esta es la razón de por qué el nombre *Back Propagation*, porque el error se propaga hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida.

  La mayor importancia que posee este proceso es que a medida que se entrena la red, las neuronas de las capas intermedias se organizan a si mismas de tal modo que las neuronas aprenden a reconocer diferentes características del espacio total de la entrada. La idea final es que después del entrenamiento, cuando se les presente un patrón arbitrario de entrada que contenga ruido o este incompleta, la red responderá con una salida activa siempre que la entrada contenga un patrón que se asemeje a aquellas características que las neuronas individuales hayan aprendido a reconocer durante su entrenamiento.

  Para iniciar el entrenamiento se le presenta a la red un patrón de entrenamiento, el cual tiene q componentes como se describe en la ecuación 2.8

$$P = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_i \\ \vdots \\ p_q \end{bmatrix} \tag{2.8}$$

Cuando se le presenta a la red un patrón para su entrenamiento, este se propaga a través de las conexiones que existen entre las neuronas, produciéndose así una entrada neta n en cada una de las neuronas de la capa siguiente, la entrada neta a la neurona j de la capa que sigue debido a la presencia de un patrón de entrenamiento en la entrada, está dada por la ecuación 2.9, se puede notar que la entrada es el valor justo antes de pasar por la función de transferencia.

$$n_j^o = \sum_{i=1}^{q} W_{ji}^o p_i + b_j^o \tag{2.9}$$

W_{ji}^o: Peso que une la componente i de la entrada con la neurona j de la primera capa oculta.
p_i: Componente i del vector p que contiene el patrón de entrenamiento de q componentes.
b_j^o: Ganancia de la neurona j de la capa oculta.

En donde $(^o)$ representa la capa a la que pertenece cada parámetro, en este caso la capa oculta.

Cada una de las neuronas que componen la capa oculta tienen como salida b_j^o que está dada por la ecuación 2.10

$$a_j^o = f^o \left(\sum_{i=1}^{q} W_{ji}^o p_i + b_j^o \right) \tag{2.10}$$

En donde f^o es la función de transferencia de las neuronas de la capa oculta.

Las salidas a_j^o de las neuronas de la capa oculta (de l componentes) son las entradas a los pesos de las conexiones con la capa de salida de la red, $a_j^o \Rightarrow n_k^s$ el comportamiento se puede observar más claramente en la ecuación 2.11

$$n_k^s = \sum_{j=1}^{m} W_{kj}^s a_j^o + b_k^s \tag{2.11}$$

W_{kj}^s: Peso que une la neurona j de la capa oculta con la neurona k de la capa de salida de la red, la que está compuesta por s neuronas.
p_j^o: Salida de la neurona j de la capa oculta, la que está compuesta por m neuronas.
b_k^s: Ganancia de la neurona k de la capa de salida.
n_k^s: Entrada a la neurona k de la capa de salida.

La salida final de la red se describe en la ecuación 2.12

$$a_k^s = f^s\left(n_k^s\right) \tag{2.12}$$

En donde f^s es la función de transferencia de las neuronas de la capa de salida.

La salida de la red de cada neurona a_k^s se compara con la salida que se espera de la red t_k para calcular el error en cada neurona de salida 2.13

$$\delta_k = \left(t_k - a_k^s\right) \tag{2.13}$$

El error que se obtiene debido a cada patrón p mostrado y luego propagado está dado por 2.14

$$ep^2 = \frac{1}{2}\sum_{k=1}^{s} \left(\delta_k\right)^2 \tag{2.14}$$

Este proceso se repite para el número total de patrones de entrenamiento que se tengan (r), para un proceso de aprendizaje que sea satisfactorio el objetivo final del algoritmo es actualizar todos los pesos y ganancias de la red neuronal minimizando el error medio cuadrático 2.15

$$e^2 = \sum_{p=1}^{r} ep^2 \tag{2.15}$$

El error que genera una red neuronal en función de sus pesos, genera un espacio n dimensional, donde n es el número de pesos de conexión que posee la red, al evaluar el gradiente del error en un punto de esta superficie n dimensional se podrá obtener la dirección en la cual la función del error tendrá un mayor crecimiento, como el objetivo del proceso de aprendizaje es lograr que ese error sea mínimo, debe tomarse la dirección negativa para obtener la dirección en la cual el decremento del error sea máximo y lograr así su minimización, que es la condición que se requiere para hacer la actualización de los pesos en el algoritmo de *backpropagation* 2.16

$$W_{k+1} = W_k - \alpha \nabla ep^2 \tag{2.16}$$

El gradiente negativo de ep^2 se denotará como $-\nabla ep^2$ y se calcula como la derivada del error respecto a todos los pesos que hay en la red.

En la capa de salida el gradiente negativo del error respecto a los pesos es:

$$-\frac{\partial ep^2}{\partial W^s_{kj}} = -\frac{\partial}{\partial W^s_{kj}} \left(\frac{1}{2} \sum_{k=1}^{l} \left(t_k - a^s_k\right)^2 \right) = (t_k - a^s_k) \times \frac{\partial a^s_k}{\partial W^s_{kj}} \tag{2.17}$$

- $-\frac{\partial ep^2}{\partial W^s_{kj}}$: Componente del gradiente respecto al peso de la conexión existente de la neurona de la capa de salida y la neurona j de la capa oculta W^s_{kj}.

- $\frac{\partial a^s_k}{\partial W^s_{kj}}$: Derivada de la salida de la neurona k de la capa de salida con respecto al peso W^s_{kj}.

Teniendo en cuenta el desarrollo matemático propuesto en [12] se llega al siguiente valor:

$$-\frac{\partial ep^2}{\partial W^s_{kj}} = (t_k - a^s_k) \times f'^s\left(n^s_{kj}\right) \times a^o_j \tag{2.18}$$

Como se puede observar y deducir fácilmente, las funciones de transferencia utilizadas en este tipo de red neuronal deben de ser continuas para que la derivada exista en todo el intervalo de cálculo, ya que el término $f'^s\left(n^s_{kj}\right)$ es requerido para el cálculo del error.

El algoritmo de *Backpropagation* utiliza la misma técnica que emplea el algoritmo LMS, es decir, aproximación en pasos descendentes [12]. La única complicación que

se puede observar es el cálculo del gradiente, el cual es indispensable para el cálculo del error y la propagación de la sensitividad.

Capítulo 3

Nociones y fundamentos de las FPGA

Cuando se adentra en la tarea de diseñar un sistema electrónico y surge la necesidad de implementar todo o una parte del mismo en *Hardware dedicado*, son varias las posibilidades que hay. En la figura 3.1 se representan las principales aproximaciones ordenándolas en función de los parámetros *Costo*, *Flexibilidad*, *Prestaciones* y *Complejidad*. Como se puede observar, las mejores prestaciones las entrega un diseño *Full-Custom*, consiguiéndose a costa de elevados costos y enorme complejidad de diseño. En el otro extremo del abanico de posibilidades se encuentra la implementación *Software*, que es muy barata y flexible, pero que en determinados casos no es válida para alcanzar un nivel de prestaciones relativamente alto.

Entre estas dos opciones se puede elegir la fabricación de un circuito electrónico realizado mediante diseño *semi-custom*, utilizando las llamadas células estándar, o recurrir a un circuito ya fabricado que se pueda "programar" in situ, como es el caso de las **FPGAs**. De estas dos opciones, la primera proporciona mejores prestaciones , aunque es más cara y exige un ciclo de diseño relativamente largo. Por otro lado, los dispositivos lógicos programables constituyen una buena oferta para realizar diseños electrónicos digitales con una buena relación *Costo - Prestaciones*. Y lo que es mejor, permite obtener una implementación en un tiempo de diseño asombrosamente corto.
Otro aspecto para tener en cuenta al momento de decidirse por este tipo de implementación es que el costo de realización es relativamente bajo, lo que suele ser una buena opción al momento de realizar prototipos [30].

Es por eso que este capítulo está dedicado a describir de forma somera en que consisten este tipo de dispositivos, haciendo énfasis en la familia Xilinx.

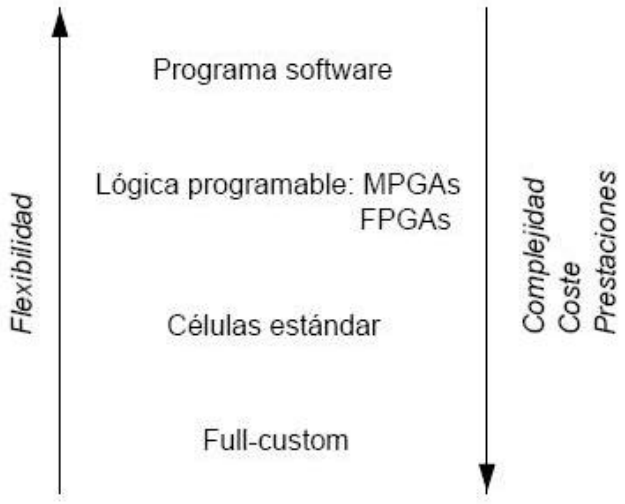

Figura 3.1: Diseños de *software* en función de costos, flexibilidad, prestaciones y complejidad

3.1. Evolución de los dispositivos programables

Se entiende por dispositivo programable aquel circuito de propósito general que posee una estructura interna que puede ser modificada por el usuario final, para implementar una gran variedad de aplicaciones. El primer dispositivo que cumplió estas características era una memoria **PROM**, *Programmable Read-Only Memory*, que puede realizar un comportamiento de circuito utilizando las líneas de direcciones como entradas y las de datos como salidas (implementa una tabla de verdad). Hay dos tipos básicos de **PROM**:

1. Programables por máscara (desde la fábrica), proporciona mejores prestaciones. Son las denominadas de conexiones *Hardwired*.

2. Programables en el campo *field* por el usuario final. Son las *EPROM (Erasable Programmable Read-Only Memory)* y las *EEPROM (Electrically-erasable programmable read-only memory)*. Proporcionan peores prestaciones, pero son menos costosas para volúmenes pequeños de producción y se pueden programar de manera inmediata.

El **PLD**, *Programmable Logic Device*, es una matriz de puertas *AND* conectada a otra matriz de puertas *OR* más unos biestables. Cualquier circuito lógico se puede implementar, por tanto, como suma de productos. La versión más básica de este circuito es una *PAL (Programmable Array Logic)*, con un plano de puertas AND y otro fijo de puertas OR. Las salidas de estas últimas se pueden pasar por un biestable en la mayoría de los circuitos que existen en el mercado.

- Ventaja: Son bastante eficientes si se implementan circuitos no superiores a unos centenares de puertas.
- Inconvenientes: Arquitectura rígida, y está limitado por un número fijo de biestables y entradas/salidas (I/O).

Después de este tipo de elementos se encuentra algo más sofisticado y general: Una matriz de elementos variados que se pueden interconectar libremente. Este es el caso de las **MPGA**, *Mask Programmable Gate Array*, cuyo principal representante está constituido por un conjunto de transistores más circuitería de **I/O**. Se unen mediante pistas de metal que se trazan de forma óptima, siendo esta, la máscara que se debe enviar al fabricante para su construcción.

Las **FPGA**, *Field Programmable Gate Array*, introducidas por Xilinx en 1985, son el dispositivo programable por el usuario de más general espectro. También se denominan **LCA**, *Logic Cell Array*. Consisten en una matriz bidimensional de bloques configurables que se pueden conectar mediante recursos generales de interconexión. Estos recursos incluyen segmentos de pista de diferentes longitudes, más unos conmutadores programables para enlazar bloques a pistas o pistas entre sí. En realidad lo que se programa en una **FPGA** son los conmutadores que sirven para realizar las conexiones entre los diferentes bloques, más la configuración de los bloques.

3.2. Field Programmable Gate Array (FPGA)

3.2.1. Estructura general de las FPGAs

El proceso de diseño de un circuito digital utilizando una matriz lógica programable puede descomponerse en dos etapas básicas:

1. Dividir el circuito en bloques básicos, asignándolos a los bloques configurables del dispositivo.
2. Conectar los bloques de lógica mediante los conmutadores necesarios.

Para ello el fabricante proporciona las herramientas de diseño adecuadas.
Los elementos básicos que constituyen una FPGA como las de Xilinx se pueden ver en la figura 3.2 y son los siguientes:

- Bloques lógicos, cuya estructura y contenido se denomina arquitectura. Existen variados tipos de arquitecturas, que varían en complejidad [1]. Suelen incluir biestables para facilitar la implementación de circuitos secuenciales.

[1] Desde una simple puerta hasta módulos más complejos o estructuras PLD

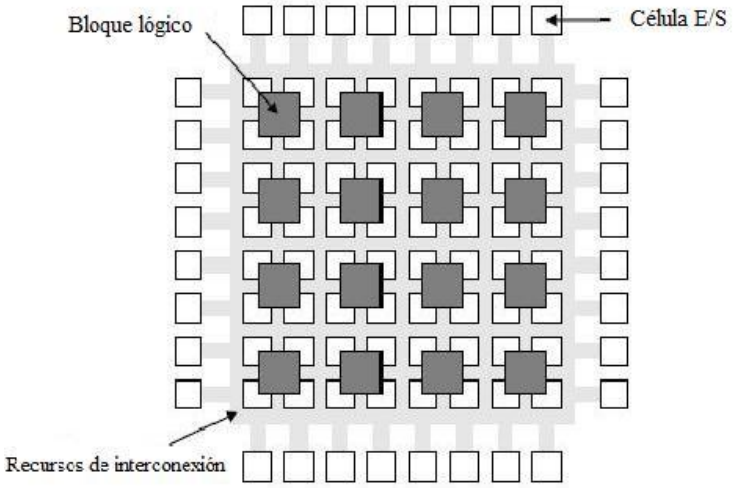

Figura 3.2: Estructura General de una FPGA. Imagen tomada de [30]

- Bloques de Entrada / Salida.
- Recursos de interconexión, cuya estructura y contenido se denomina arquitectura de rutado.
- Memoria RAM, que se carga durante el *reset* para configurar bloques y conectarlos.

Entre las numerosas ventajas que proporciona el uso de FPGAs se destacan principalmente dos: El bajo costo de realizar prototipos y el corto tiempo de producción. Pero no todo son ventajas. Entre los inconvenientes de su utilización están la baja velocidad de operación y baja densidad lógica en un chip simple.

Pero no todas las FPGAs son iguales. Dependiendo del fabricante se pueden encontrar diferentes soluciones. Las que actualmente se encuentran en el mercado se pueden clasificar en cuatro grandes familias. Estas estructuras se pueden observar en la figura 3.3 y son:

1. Matriz simétrica, como son las de **XILINX**
2. Basadas en canales, las de **ACTEL**
3. Mar de puertas, como las de **ORCA**
4. PLD jerárquica, como las de **ALTERA**

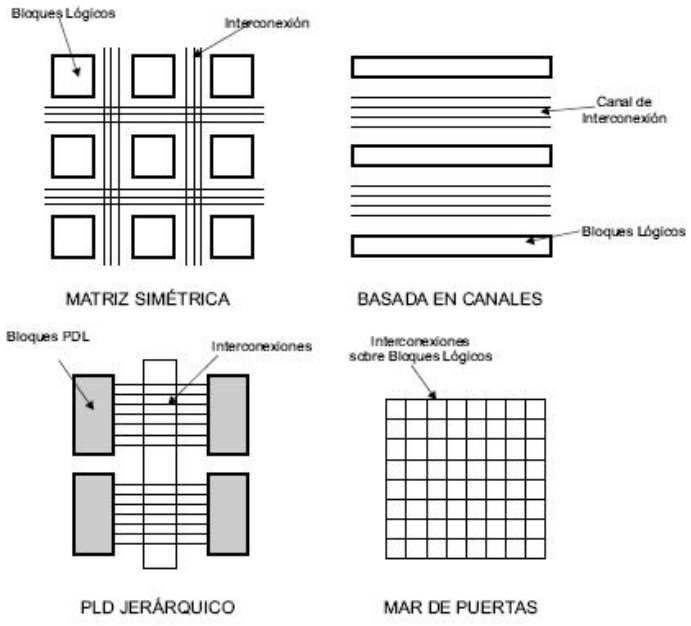

Figura 3.3: Tipos de FPGAs. Imagen tomada de [30]

3.2.2. Arquitectura de las FPGA de Xilinx

Tecnología de programación

Antes de comenzar con conocimientos más avanzados acerca de las FPGAs de Xilinx, hay que aclarar la realización del proceso de programación.
En primer lugar, si se piensa que el número de dispositivos de conexión que hay en una FPGA es muy grande (usualmente por encima de $100{,}000$), es necesario que dichos dispositivos cumplan las siguientes propiedades:

- Ser lo más pequeños posible.
- Tener la resistencia **ON** lo más baja posible, mientras la **OFF** ha de ser lo más alta posible (para que funcione como conmutador).
- Se deben de poder incorporar al proceso de fabricación de la FPGA.

El proceso de programación no es único, sino que se puede realizar mediante diferentes "tecnologías", como las células RAM estáticas, transistores EPROM y EEPROM, etc. En el caso de las FPGAs de Xilinx los elementos de programación se basan en células de memorias

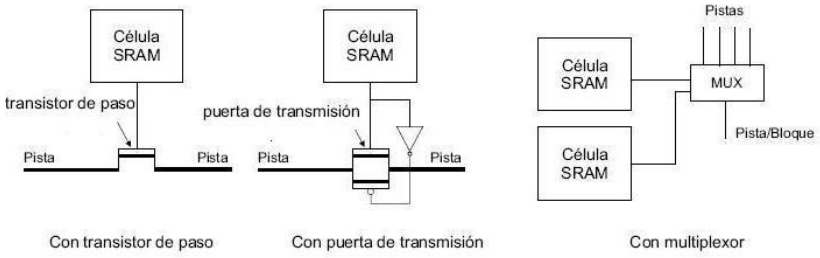

Figura 3.4: Tipos de conectores utilizados por Xilinx. Imagen tomada de [30]

SERIE	Tipo CLB	Nº de CLBs	Puertas Equivalentes
XC2000	1 LUT, 1 FF	64-100	1.200-1.800
XC3000	1 LUT, 2 FF	64-484	1.500-7.500
XC4000XL	3 LUT, 2 FF	64-3.136	1.600-180.000

Figura 3.5: Familias del fabricante XILINX

RAM que controlan transistores de paso, puertas de transmisión o multiplexores. En la figura 3.4 se puede ver el esquema de cómo son.

Es importante destacar que si se utilizan células SRAM la configuración de la FPGA será válida únicamente mientras esté conectada la alimentación, pues es memoria volátil. En los sistemas finales está claro que hace falta algún mecanismo de almacenamiento no volátil que cargue las células de RAM. Esto se puede conseguir utilizando EPROMs o disco.
Este elemento de programación es relativamente grande (aproximadamente 5 transistores), pero se puede implementar en el proceso normal del circuito, utilizando tecnología CMOS. Además permite realizar una reconfiguración rápida de la FPGA.

Descripción de las principales familias

Hay multiples familias lógicas dentro de Xilinx. Las primeras que surgieron fueron las $XC2000$, $XC3000$ y las $XC4000$, correspondientes respectivamente a la primera, segunda y tercera generación de dispositivos, que se distinguen por el tipo de bloque lógico configurable (**CLB**) que contienen. La figura 3.5 muestra la cantidad de CLBs que puede haber en cada FPGA de las familias base y ese mismo valor expresado en puertas equivalentes. El

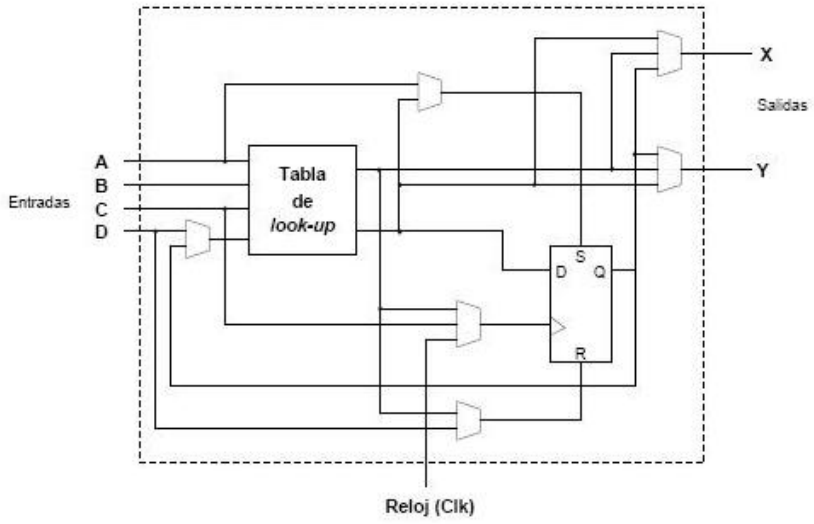

Figura 3.6: Arquitectura del CLB de la XC2000. Imagen tomada de [30]

bloque lógico ha de ser capaz de proporcionar una función lógica en general y reprogramable. La mejor forma de realizar esto es mediante una tabla de valores "pre-asignados" o *Look up Tables* (**LUT** en adelante). Una LUT es básicamente una memoria, con un circuito de control que se encarga de cargar los datos. Cuando se aplican en una dirección las entradas de la función booleana, la memoria devuelve un dato, lo que se hace corresponder con la salida requerida. Falta añadir los componentes necesarios para desempeñar funciones no implementables con una memoria, tales como una batería de registros, multiplexores, buffers, etc. Estos componentes están en posiciones fijas del dispositivo.

La ventaja de la utilización de este tipo de tablas es su gran flexibilidad: Una LUT de k entradas puede implementar cualquier función booleana de k variables, y hay 2^{2^k} funciones posibles. El inconveniente es obvio: ocupan mucho espacio y no son muy aprovechables. Los bloques lógicos configurables de la familia XC2000 se componen de una *Look up table* con cuatro entradas y un biestable, con lo que puede generar cualquier función de hasta cuatro variables o dos funciones de tres variables. El de la familia XC3000 es más complejo, permite implementar funciones de cinco variables o dos funciones de cuatro variables. Además contiene dos biestables y más lógica. Y así, a medida que se aumenta de nivel en las familias, mayor es el nivel de recursos y de tecnología para ser usados.
En general, los recursos de interconexión son de tres tipos [30]:

- Conexiones directas, permiten la conexión de las salidas del CLB con sus vecinos más

directos.

- Interconexiones de propósito general, para distancias superiores a un CLB (es decir, más allá del vecino inmediato). Son pistas tanto horizontales como verticales, del tamaño de un CLB, pero que se empalman para generar pistas más largas.

- Líneas de largo recorrido, suelen cubrir el ancho o largo de la pastilla. Permiten conexiones con retardos inferiores a la unión de las pistas del item anterior.

Aquí es donde aparece el concepto de camino crítico, que es aquel recorrido que, desde una entrada hasta una salida presenta el máximo retardo.

3.3. Diseño con FPGAs

3.3.1. Conceptos del VHDL

El **VHDL** (o por sus siglas en inglés *Very High Speed Hardware Description Language*), es un lenguaje para la descripción de sistemas electrónicos digitales. Este lenguaje de programación nace del programa del gobierno norteamericano iniciado en 1980 de circuitos integrados de alta velocidad (*Very High Speed Integrated Circuits*, ***VHSIC***). En el desarrollo de este programa se encontró la necesidad de un lenguaje estandarizado para la descripción de la estructura y la función de los circuitos integrados (**IC**). Y fue allí donde el *VHSIC Hardware Description Language (VHDL)* fue desarrollado, y subsecuentemente adoptado como un estándar por el *Institute of Electrical and Electronic Engineers (IEEE)*, en los Estados Unidos [2].

El VHDL fue diseñado para llenar un gran número de necesidades en el proceso de diseño. Primero, permite la descripción de la estructura que es la manera de descomponer el total en una serie de sub-diseños y como se interconectan éstos para formar el ente total. Segundo, permite la especificación de la función de diseños usando formas familiares de lenguajes de programación. Y tercero, como resultado, permite al diseño ser simulado antes de ser construido, así los diseñadores pueden rápidamente comparar alternativas y probarlas para realizar correcciones sin el retraso de realizar cada vez prototipos en *hardware*.

3.3.2. Descripción estructural

Un sistema electrónico digital puede ser descrito como un módulo con entradas y/o salidas. Los valores eléctricos en las salidas son función de los valores en las entradas, la figura

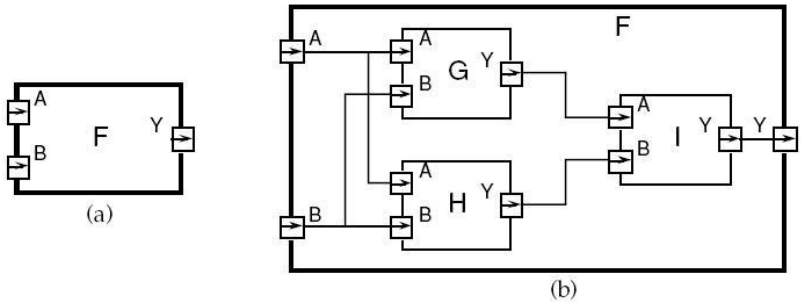

Figura 3.7: Ejemplo de una descripción estructural

3.7(a) muestra un ejemplo de este tipo de vista de un sistema digital. El módulo F posee dos entradas A y B, y una salida Y. Usando terminología del VHDL, el módulo F será llamado una **entidad**, y las entradas y salidas serán llamadas **puertos**.

Una manera de describir la función de un módulo es describiendo como está compuesto en sub-módulos. Cada uno de los sub-módulos es una **instancia** de alguna entidad, y los puertos de las instancias son conectados usando **señales**.
La figura 3.7(b) muestra como la entidad F puede estar compuesta de instancias de entidades G, H e I. Este tipo de descripción es llamada *Descripción Estructural*. Nótese que cada una de las entidades G, H e I pueden tener igualmente una descripción estructural.

3.3.3. Descripción comportamental

En muchos casos no es apropiado describir un módulo estructuralmente. Uno de estos casos es un módulo que se encuentre al final de la jerarquía de alguna descripción estructural. Por ejemplo, si se está diseñando un sistema usando paquetes de **IC** comprados en una tienda de **IC**, no hay necesidad de describir la estructura interna del IC. En estos casos, una descripción de la función realizada por el módulo si es requerida, sin hacer referencia a la estructura interna actual. Dicha descripción es llamada una descripción *funcional* o de *comportamiento*.

Para ilustrar mejor este concepto, supóngase que la función de la entidad F en en la figura 3.7(a) es la función XOR, Luego, una descripción del comportamiento de la función F podría ser la función booleana:

$$Y = \bar{A} \cdot B + A \cdot \bar{B} \tag{3.1}$$

Comportamientos más complejos no pueden ser descritos puramente como una función de entradas. En sistemas con realimentación, las salidas son también función del tiempo. VHDL resuelve este problema permitiendo la descripción del comportamiento en la forma de un programa ejecutable.

3.3.4. Modelado de eventos discretos en el tiempo

Una vez que la estructura y el comportamiento de un módulo ha sido especificada, es posible simular el modulo ejecutando su descripción de comportamiento. Ésto se logra simulando el paso del tiempo en pasos discretos. En algún tiempo de simulación, la entrada al módulo puede ser estimulada cambiando el valor de un puerto de entrada. El módulo reacciona corriendo el código de su descripción de comportamiento y programando nuevos valores para ser ubicados en las señales conectadas a sus puertos de salida en un tiempo de simulación después. Ésto se conoce como programar un arreglo a la señal. Si el nuevo valor es diferente del valor anterior en la señal, un *evento* ocurre, y otros módulos con puertos de entrada conectados a la señal, serán activados.

La simulación inicia con una *fase de inicialización*, y luego procede repitiendo un *ciclo de simulación* de dos etapas. En la fase de inicialización, a todas las señales se les dan valores iniciales, el tiempo de simulación se inicia en cero, y cada programa de comportamiento de los módulos es ejecutado.

En la primera etapa de un ciclo de simulación, el tiempo de simulación es avanzado hasta el tiempo más temprano en el que un cambio ha sido programado.
En la segunda etapa, todos los módulos que reaccionaron a eventos ocurridos en la primera etapa, ejecutan su programa de comportamiento. Estos programas usualmente fijan futuros cambios en sus señales de salida. Cuando todos los programas de comportamiento se han terminado de ejecutar, el ciclo de simulación se repite. Si no hay más transacciones programadas, la simulación es completada.
El propósito de la simulación es obtener información acerca de los cambios en el sistema sobre el tiempo. Ésto se logra corriendo la simulación bajo el control de un *monitor de la simulación*. El monitor permite a las señales y otra información de los estados, ser vistas o almacenadas en un archivo para análisis posterior. Además permite pasos interactivos en el proceso de simulación.

3.3.5. Ejemplo rápido

A continuación se mostrará un pequeño ejemplo de una descripción en VHDL de un contador de dos bits, para dar a conocer el lenguaje y como funciona. Se inicia con la de-

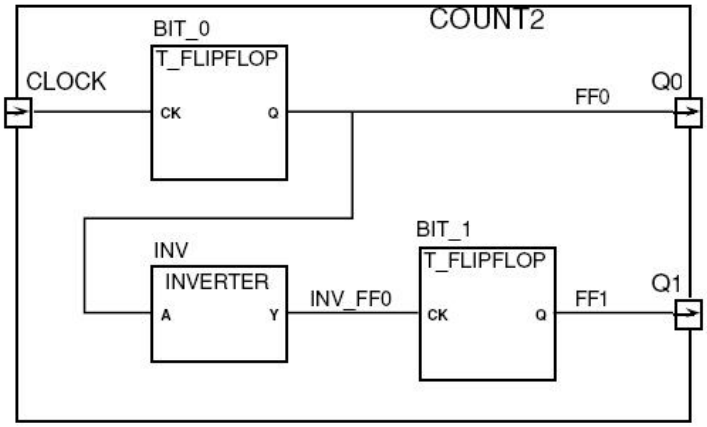

Figura 3.8: Estructura de la entidad count2

scripción de una entidad, especificando su interface externa, la cual incluye una descripción de sus puertos. El contador puede ser definido así:

```
entity count2 is
        generic(pdelay : Time := 10 ns);
        port (clock in bit;
              q1, q0 out bit);
end count2;
```

Este sistema especifica que la entidad **count2** tiene una entrada y dos salidas, los cuales son todos valores de un sólo bit, esto es que pueden tomar valores entre ’0’ y ’1’. Además se define una constante genérica llamada **pdelay** la cual puede ser usada para controlar la operación de la entidad (en este caso es una propagación del retraso). Si ningún valor está explícitamente dado por este valor cuando la entidad es usada en un diseño, el valor por defecto de 10 ns será usado.

Una implementación de la entidad es descrita en un cuerpo de arquitectura.
Existe más de una arquitectura correspondiente a una entidad específica, cada una de las cuales es válida y muestra diferentes puntos de vista de la entidad.
Por ejemplo, una descripción de comportamiento del contador puede ser escrita como:

```
architecturebehaviour ofcount2 is
```

```
begin
    countup: process (clock)
                variable count-val:natural := 0;
    begin
        if clock = '1' then
            count-val := (count-val + 1)mod4;
            q0 <= bit'val(count-valmod2) after pdelay
            q1 <= bit'val(count-val/ 2) after pdelay
        end if
    end processcountup
end behaviour
```

En esta descripción del contador, el comportamiento es implementado por un proceso llamado **countup**, el cual es sensible a la entrada de reloj *clock*. Un proceso (o *process* en **VHDL**) es un cuerpo de código el cual es ejecutado siempre y cuando cualquiera de las señales a las que es sensible, cambie su valor. Este proceso posee una variable llamada **count-val** para almacenar el estado actual del contador. La variable es inicializada en cero, y retiene su valor entre las diferentes etapas de activación del proceso. Cuando la entrada de reloj (*clock*) cambia de '0' a '1', el estado de la variable es incrementado en uno, y los cambios son asignados a los puertos de salida, basados en el nuevo valor.
Las asignaciones usan la constante genérica **pdelay** para determinar que tan largo debe ser el tiempo, para que luego de que el reloj cambie, el cambio sea ejecutado. Cuando el control alcanza el final del cuerpo del proceso, el proceso se suspende hasta que ocurra otro cambio en la entrada de reloj.

El contador de 2 bits, también puede ser descrito como un circuito compuesto por dos flip/flops y un inversor como se ve en la figura 3.8.

Capítulo 4

Preprocesamiento y caracterización de la señal de voz

El desarrollo de este capítulo está encaminado a mostrar el preprocesamiento de la señal de voz adquirida, los pasos y elementos matemáticos que hacen posible la digitalización de la señal y su posterior procesamiento, tanto para la lógica secuencial del procesador de propósito general PC, como para su implementación sobre la FPGA.
El objetivo primordial que se busca con la etapa de preprocesamiento, es hacer que la señal de voz adquirida quede tan limpia, como el sistema de caracterización lo requiera. Lo que se logra en esta etapa es hacer que el ruido se elimine, la señal se vuelva lo más plana posible espectralmente y las tendencias espectrales de tiempo largo se remuevan de la señal de voz, todo esto para obtener finalmente la mayor inmunidad posible a las imperfecciones en la medición.

4.1. Adquisición y preprocesamiento de la señal de voz

La selección de las palabras y la adquisición de las muestras, se realizó teniendo en cuenta los parámetros estipulados en [11]. Las palabras usadas fueron 15, la cuales se incluyen en el anexo A de este trabajo de grado. Dichas palabras fueron pronunciadas por 100 personas diferentes (cada persona pronunció las 15 palabras, es decir una muestra de cada palabra por persona), con edades que oscilan entre los 10 y los 61 años.

Las condiciones bajo las cuales se adquirieron las muestras y se procedió a crear la base de datos fueron:

- Ambiente semi-controlado, con la mayor relación posible voz a ruido.

- Micrófono Conexant HD.
- Frecuencia de muestreo de 22050 Hz, adquisición monaural, resolución de 16 bits por muestra en la conversión análoga-digital.
- *Software* de procesamiento Matlab.

Tras la obtención de las muestras de las palabras (1500 en total), usando un programa realizado en Matlab, el cual se incluye en medio magnético, se realiza el preprocesamiento de la señal digitalizada.
El preprocesamiento de dicha señal incluye los pasos de segmentación, filtrado de pre-énfasis, normalización en amplitud y ventaneo.

4.1.1. Segmentación

El problema de la localización del inicio y final de una muestra de voz en un fondo acústico de silencio es de vital importancia en muchas áreas del procesamiento de voz.
En particular, el problema del reconocimiento de palabras está basado en la suposición de que se puede hallar la región de la muestra donde hay voz, para ser reconocida. Una extraordinaria ventaja de una buena localización de las regiones de voz, es que pueden reducir substancialmente la cantidad de procesamiento requerido para la aplicación que se busca.

La tarea de separar la voz del espacio de silencio no es una tarea tan simple y trivial como puede parecer, esto puede suceder sólo en el caso de ambientes acústicos con tasa extremadamente alta de señal a ruido; como ejemplo de dichos ambientes tenemos las cámaras a prueba de eco, o cámaras a prueba de ruido donde son hechas las grabaciones de alta calidad. Para dichos ambientes con alta relación señal a ruido, la energía de los sonidos de voz de más bajo nivel (como los fricativos débiles, las porciones de voz de bajo nivel, etc) excede la energía del ruido del fondo, por ende una simple medición de la energía es suficiente para realizar las mediciones.

Sin embargo, dichas condiciones ideales de grabación no son prácticas para las aplicaciones del mundo real en sistemas de procesamiento de voz. Por ende, la simple medición de la energía se convierte en una condición necesaria mas no suficiente para separar los fricativos débiles del fondo de silencio.

En este trabajo de grado se utilizó el algoritmo propuesto por *Rabiner y Sambur* en [21] para la localización de los puntos de inicio y final de grabación de voz, dicho algoritmo es muy útil y funciona muy bien en ambientes con una relación señal a ruido de al menos 30

dB. Dicho algoritmo está basado en dos mediciones de la voz, la *energía de tiempo corto* y la *tasa de cruces por cero*.

Algoritmo de localización de inicio y final de voz

Las metas del algoritmo de localización de inicio y final de grabación son:

1. Procesamiento simple y eficiente.
2. Localización confiable de eventos acústicos significantes.
3. Capacidad de ser aplicado a una amplia variedad de fondos de silencio.

La primera meta implica que sólo mediciones simples se le realizarán a la forma de onda de la voz como una base para la toma de decisiones. Si la velocidad y la simplicidad no son temas de importancia, se pueden utilizar algoritmos de procesamiento más complejos, para lograr así mayor exactitud en los resultados.

Con las consideraciones anteriores en mente, el algoritmo de localización de los límites se implementó y desarrolló con base en dos mediciones simples, energía y tasa de cruces por cero, y al final se usa lógica simple para la toma de decisión final. Ambos métodos son simples y rápidos de computar, y además pueden dar indicaciones bastante exactas de la presencia o ausencia de voz.

Pero, cómo se mide la energía de tiempo corto y la tasa de cruces por cero?, este será el tema a tratar a continuación.

Energía de tiempo corto La "energía" de la voz $E(n)$, se define como la suma de las magnitudes de la voz en intervalos centrados de 10 ms, todo medido sobre el intervalo de medición [23].

$$E(n) = \sum_{i=-50}^{50} |s(n+1)| \tag{4.1}$$

En donde, $s(n)$ son las muestras de voz muestreadas a la frecuencia escogida (que para este trabajo de grado fue de 22050 Hz). El escoger una ventana de 10 ms para el cálculo de la energía y el uso de la función de magnitud en lugar del cuadrado de la magnitud, se realizó por el deseo de realizar los cálculos en aritmética integral y por ende, el aumentar la velocidad de la computación.

Tasa de cruces por cero La tasa de cruces por cero de una grabación de voz $z(n)$ se define como el número de cruces por cero en un intervalo de 10 ms. Aunque la tasa de cruces por cero es altamente susceptible a ruido de 60 Hz, offset, etc; en la mayoría de los casos es una medida muy buena y razonable de la presencia o ausencia de silencio en la grabación.[23]

La principal suposición que se realiza al momento de aplicar este algoritmo es que los primeros 100 ms de grabación son silencio. Es por eso, que en este intervalo de tiempo, las estadísticas de silencio en el fondo de la grabación son medidas.
Estas mediciones incluyen: La media de los valores de los datos digitalizados y la desviación estándar de la tasa de cruces por cero, así como la energía promedio.

Si alguna de estas medidas es excesiva, el algoritmo se detiene y avisa al usuario. De otra manera, un límite de cruces por cero, **IZCT**, para la zona donde no existe voz, es escogido como la suma de la tasa media de cruces por cero durante el silencio, $\overline{IZC}$, más dos veces la desviación estándar de la tasa de cruces por cero durante el silencio

$$IZCT = (\overline{IZC} + 2\sigma_{IZC}) \tag{4.2}$$

La función de energía $E(n)$ es calculada para todo el intervalo. La energía pico, IMX, y la energía en el silencio, IMN, son usadas para obtener dos nuevos límites, ITL e ITU, de acuerdo a la regla

$$\begin{aligned} I1 &= 0{,}03 \cdot (IMX - IMN) + IMN && (4.3)\\ I2 &= 4 \cdot IMN && (4.4)\\ ITL &= \text{máx}(I1, I2) && (4.5)\\ ITU &= 5 \cdot ITL && (4.6) \end{aligned}$$

La ecuación 4.3 muestra que $I1$ se encuentra a un nivel que es 3 por ciento la energía pico (ajustado para la energía del silencio), de la misma manera 4.4 muestra que $I2$ se encuentra a un nivel puesto a cuatro veces la energía del silencio. El límite bajo, ITL, es el máximo de estos dos límites de conservación de energía. Y el límite alto, ITU, es cinco veces el límite bajo.

Algoritmo de energía media . El algoritmo empieza buscando desde el inicio del intervalo hasta que el límite bajo sea excedido. Este punto es preliminarmente etiquetado como el inicio de la grabación hasta que la energía caiga por debajo de ITL antes de que crezca por encima de ITU.
Si esto ocurre, un nuevo punto de inicio es obtenido encontrando el primer punto al cual la energía excede ITL, y luego excede ITU antes de caer por debajo de ITL;

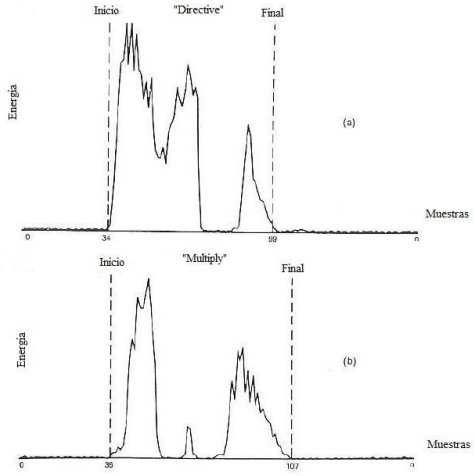

Figura 4.1: Gráficas típicas de la energía, con marcadores de inicio (beginning) y final (end) de cada palabra

eventualmente un punto de inicio debe existir.
Un algoritmo similar es usado para definir un estimado preliminar del punto final de la grabación. Estos son los llamados puntos de inicio y final, N_1 y N_2 respectivamente.

Hasta ahora, sólo se han usado medidas de la energía para encontrar los límites de la grabación. En este punto, se puede asumir que, aunque parte de la grabación puede estar fuera del intervalo (N_1, N_2), los puntos actuales de inicio y final no se encuentran allí. Con relación a esto aparece el algoritmo de cruces por cero.

Algoritmo de cruces por cero . El algoritmo procede a examinar el intervalo desde N_1 hasta $N_1 - 25$, y contar el número de intervalos en donde la tasa de cruces por cero exceda el límite $IZCT$. Si el número de veces que el límite es excedido es de tres veces o más, el punto de inicio de voz es puesto en el primer punto donde se excedió el límite. De otra manera, el punto inicial se mantiene en N_1. La función de cruces por cero está dada por la ecuación expuesta en [20]

$$Z(n) = \sum_{m=-\infty}^{\infty} |\,[s(m)] - [s(m-1)]\,| r(n-m) \tag{4.7}$$

donde

$$\begin{aligned} [s(n)] &= 1 \quad s(n) \geq 0 \\ &= -1 \quad s(n) < 0 \end{aligned}$$

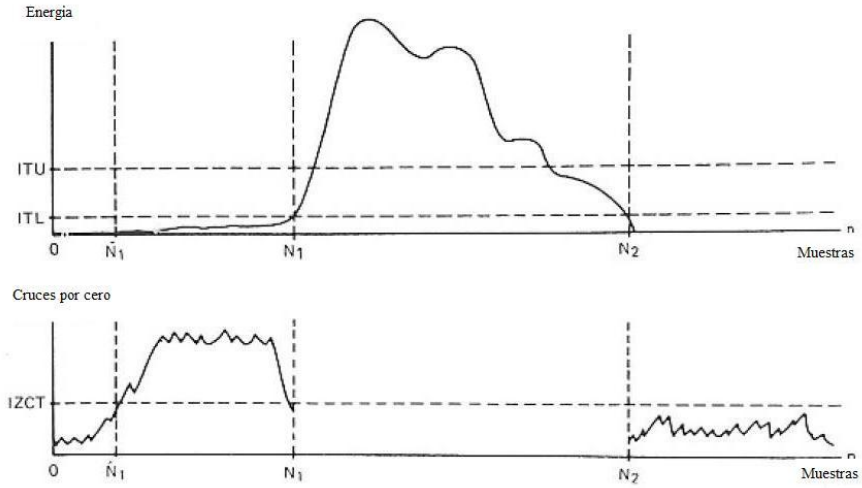

Figura 4.2: Ejemplo típico de energía y cruces por cero para una palabra fricativa

y

$$\begin{aligned} r(n) &= \frac{1}{2N} \quad 0 \leq n \leq N-1 \\ &= 0 \quad \text{de otra manera} \end{aligned}$$

La idea detrás de esta estrategia es que para todos los casos de interés, el exceder un límite ajustado en la tasa de cruces por cero es una fuerte y confiable indicación de la energía del silencio (figura 4.2). Es muy posible que un sonido fricativo débil no pase la prueba y por ello se pierda. Sin embargo, en estos casos no existe un método muy *confiable* para distinguir dichos sonidos fricativos débiles del fondo de silencio.

Un procedimiento de búsqueda similar a este es usado para determinar si existe energía de silencio en el intervalo desde N_2 hasta $N_2 + 25$ y así hallar el punto final de la grabación de voz. El punto final es reajustado basado en los resultados de la prueba de cruces por cero[21].

4.1.2. Filtrado de pre-énfasis

Una vez que se ha logrado segmentar la grabación de la voz, se hace completamente necesario realizar un filtrado a la señal digitalizada y segmentada, principalmente para hacer

sobresalir las componentes de alta frecuencia de la muestra. Un filtrado simple usa un filtro FIR (Filtro bi-dimensional de respuesta finita al impulso) cuya fórmula matemática de descripción es

$$H_{pre}(z) = 1 + a_{pre}z^{-1} \tag{4.8}$$

Un rango de valores típico para a_pre se encuentra entre $[-1{,}0, -0{,}4]$. Existen dos explicaciones simples para utilizar este filtro. Primero, las secciones sonoras de la señal poseen una atenuación natural de pendiente negativa de unos 20 dB por década, y esto debido a las características fisiológicas del sistema humano de producción de voz. Este filtro compensa la pendiente natural de la voz antes del análisis espectral, mejorando de esta manera la eficiencia final del análisis a realizar [17].
Otra explicación alternativa es más fisiológica, como se sabe, el oído es mucho más sensible en la región por encima de 1 kHz. El diseño de este filtro hace que se amplifique esta área del espectro, ayudando a que el algoritmo de análisis espectral modele los aspectos perceptuales más importantes del espectro de voz [17].

4.1.3. Normalización en amplitud

La finalidad tras esta etapa es lograr que el rango de variación de las muestras de voz adquiridas, se unifique; algo que no es igual al momento de la grabación. Básicamente la idea tras esta etapa es lograr hacer la media de los valores igual a cero y normalizar la amplitud a un rango $[-1, 1]$; de esta manera se logra que en las etapas que vienen a continuación, es decir extracción de características y clasificación, se pierda la sensitividad a los valores máximos de la amplitud [11]. En este caso se hizo uso de la normalización *premnmx*, la cual se encarga de preprocesar los datos para que el valor mínimo sea -1 y el mayor 1. Este preprocesamiento se aplica al conjunto de muestras adquiridas, segmentadas y filtradas, normalizando para que caigan dentro del intervalo $[-1, 1]$. El algoritmo empleado es el siguiente

$$pn = 2 \times \left(\frac{p - minp}{maxp - minp} \right) - 1 \tag{4.9}$$

4.1.4. Ventaneo

La señal de la voz se puede caracterizar como un proceso no estacionario sobre intervalos de tamaño variable en función del sonido que se está pronunciando [11]. Al descomponer la señal en bloques relativamente pequeños (del orden de 20 a 40 ms), se puede lograr que la señal de voz adquiera características de cuasi-estacionariedad [16]. Es así como la señal de voz que ha pasado por el filtro de pre-énfasis, $\tilde{s}(n)$, se divide en bloques de N muestras cada uno, con tramas adyacentes separadas por M muestras. Si se puede realizar una notación y

llamar $x_l(n)$ al ***l-ésimo*** bloque, y existen L bloques en la señal de voz, entonces:

$$x_l(n) = \tilde{s}(Ml + n), \quad n = 0, 1, ..., N-1, \quad l = 0, 1, 2, ..., L-1 \tag{4.10}$$

Para la realización de este proyecto de grado se usaron ventanas de 30 ms calculadas cada 10 ms, como se planteó en [11].
La división de la señal en tramas se realiza mediante un corte abrupto que genera discontinuidades al inicio y final de cada trama. Es por eso, que en este paso se busca disminuir el efecto e la división por tramas, multiplicando cada bloque por una ventana que sea adecuada y cuyo objetivo es disminuir paulatinamente hasta cero su valor al inicio y final de cada bloque. Definiendo cada bloque como $w(n)$, $0 \leq n \leq N-1$, el resultado de realizar el ventaneo es la nueva señal $\tilde{x}_l(n)$ que no es más que la señal de voz pre-enfatizada y normalizada en amplitud, multiplicada por la ventana elegida.

En reconocimiento de voz, existen diferentes tipos de ventanas, pero la más usada es la ventana Hamming [15] cuya forma matemática es

$$w(n) = 0{,}54 - 0{,}46 \cos\left(\frac{2\pi n}{N-1}\right) \tag{4.11}$$

4.2. Caracterización de la señal de voz

La etapa de caracterización de la señal, es básicamente la estimación de variables, llamadas vectores de características o parámetros, a partir de otro conjunto de variables. La elección de la mejor representación paramétrica de la información acústica adquirida, es la principal, si no la más importante tarea, al momento de realizar un reconocimiento de voz.

Básicamente, los objetivos buscados al seleccionar una representación paramétrica son, en primer lugar, lograr realizar una compresión de la señal de voz, eliminando así la información no pertinente para el posterior análisis fonético de los datos pre-procesados. Segundo, busca realzar aquellos aspectos importantes de la señal que puedan contribuir de manera significante a la detección de las diferencias fonéticas.

Como es bien sabido, existen numerosas formas de representar paramétricamente los datos pre-procesados de la señal de voz adquirida. En este trabajo de grado, las representaciones paramétricas pueden dividirse en dos grupos principales, en aquellas basadas en el **Espectro de Fourier** y aquellas que basan su representación en el **Espectro de Predicción Lineal**.

Como se explica en [11], al primer grupo pertenecen los coeficientes *Cepstrum* derivados

del análisis de predicción lineal (*Linear Predictive Cepstrum Coefficients* ó **LPCC**) y en el segundo se encuentran los coeficientes *Cepstrum* sobre la escala de frecuencias Mel (*Mel-Frequency Cepstrum Coefficients*) ó **MFCC** y sobre las cuales se enfocará este trabajo.

Estas características obtenidas no poseen ningún tipo de significado físico, y mucho menos alguna relación con las características de producción de voz o también llamadas *Características Acústicas* [27], como son el pitch, el jitter, etc.

4.2.1. Coeficientes Cepstrum sobre la Escala de Frecuencias Mel (MFCC)

Los **MFCC** son una representación motivada perceptualmente al igual que los **PLP**. Se definen como el *Cepstrum real de una señal de tiempo corto **ventaneada** y que ha sido derivada de la FFT de esa señal*. Se usa una escala de frecuencia no lineal para aproximar el comportamiento del sistema auditivo.
Los pasos básicos para realizar un análisis acústico haciendo uso de los **MFCC** son:

- ***Espectro de Potencia***: Este se obtiene de cada uno de los bloques ventaneados usando la **TDF** y el Análisis espectral de la señal, es decir, se realiza la **TDF** de los bloques. De allí, las componentes real e imaginaria del espectro de voz de tiempo corto se elevan al cuadrado y se suman para la obtención final del espectro de potencia de tiempo corto

$$P(w) = \mathbb{R}[S(w)]^2 + \mathbb{I}[S(w)]^2 \tag{4.12}$$

- ***Banco de filtros en la escala Mel***: En esencia, es un banco de filtros con M filtros $(m = 1, 2, ..., M)$ en donde m es un filtro triangular dado por la siguiente ecuación

$$H_m[k] = \begin{cases} 0 & k < f[m-1] \\ \frac{2(k-f[m-1])}{(f[m+1]-f[m-1])(f[m]-f[m-1])} & f[m-1] \leq k \leq f[m] \\ \frac{2(f[m+1]-k)}{(f[m+1]-f[m-1])(f[m+1]-f[m])} & f[m] \leq k \leq f[m+1] \\ 0 & k > f[m+1] \end{cases} \tag{4.13}$$

  Dichos filtros se encargan del cálculo del espectro promedio alrededor de cada frecuencia central con anchos de banda que aumentan de manera progresiva.

  Se define f_l y f_h como las frecuencias más baja y más alta del banco de filtros dadas en Hz, la frecuencia de muestreo está dada por F_s, el número de filtros es M y el tamaño de la **FFT** está dado por N. (ver Anexo D)

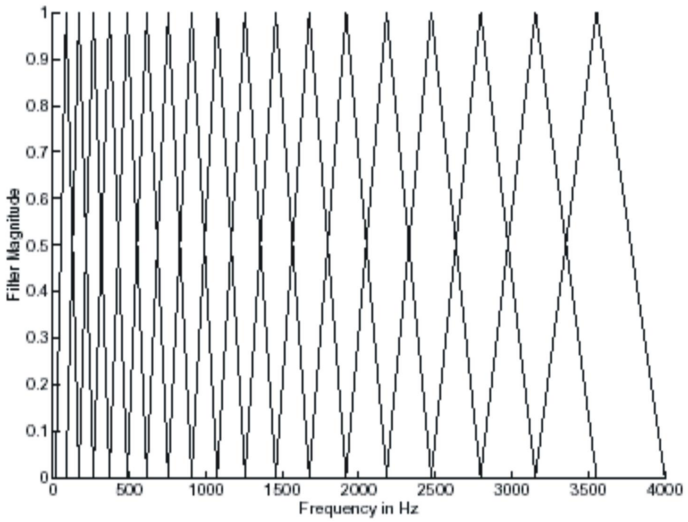

Figura 4.3: Banco de filtros en la escala mel

Los puntos límites se encuentran separados de manera uniforme en la escala *Mel* de la siguiente manera:

$$f[m] = \left(\frac{N}{F_s}\right) B^{-1}\left(B(f_l) + m\frac{B(f_h) - B(f_l)}{M+1}\right) \tag{4.14}$$

En donde la escala *Mel* B está dada por

$$B(f) = 1125ln\left(1 + \frac{f}{700}\right) \tag{4.15}$$

Y su inversa como

$$B^{-1}(b) = 700(e^{\frac{b}{1125}} - 1) \tag{4.16}$$

- ***Cálculo de la energía log***: Este cálculo se realiza a la salida de cada filtro, básicamente para tratar de simular la relación no lineal existente entre la intensidad emitida y la intensidad percibida, cuya naturaleza es netamente logarítmica

$$Y[m] = ln\left[\sum_{k=0}^{N-1} |X_a[k]|^2 H_m[k]\right], \qquad 0 < m \leq M$$

donde $X_a[k]$ es la **TDF** de la señal de entrada.

- ***Coeficientes Cepstrum***: Los coeficientes *Cepstrum* derivados de la escala *Mel* se obtienen a partir de la transformada discreta del coseno de las salidas de los M filtros

$$c[n] = \sum_{m=0}^{M-1} Y[m] cos(\frac{\pi n(m - \frac{1}{2})}{M}), \qquad 0 \leq n \leq M$$

  donde M tiene variaciones entre 24 y 40 dependiendo de la implementación.

En este punto, la caracterización está encaminada al análisis discreto de la señal de voz. Esto se logra limitando el número de coeficientes *Cepstrum* a 13, es decir los coeficientes de la energía de la señal en el dominio de la frecuencia. Otro tipo de análisis que se puede extraer con los coeficientes, es el de las características dinámicas de la señal [19].
Aunque puede conllevar a un análisis más prolongado, el paso inicial es aumentar el número de coeficientes a 39 haciendo ahora un análisis no sólo a la energía, sino también a la primera y segunda derivada de ésta.

Capítulo 5

Diseño e implementación del sistema de clasificación

La idea tras este capítulo es mostrar el desarrollo que se siguió para el diseño, entrenamiento, pruebas y final implementación en **VHDL** del sistema de clasificación, tras el cual se ha desarrollado este trabajo de grado: las redes neuronales artificiales **(ANN)**

5.1. Diseño del sistema de clasificación basado en redes neuronales artificiales

La primera etapa de desarrollo de este sistema se basó en la apropiada selección de la red neuronal. La idea primordial se basa en encontrar un diseño lo más óptimo, simple y sobretodo eficiente para su apropiada implementación en VHDL.

5.1.1. Topología de la Red

Aún hoy con el alto avance existente en el desarrollo de la ciencia, y sobretodo en el campo de la matemática aplicada, no se ha llegado a un concenso generalizado sobre cuál es la mejor manera de escoger la topología de la red, es decir, el número de capas y el número de neuronas por capas. Es por eso que el método de “ensayo y error” sigue siendo la herramienta más útil al momento de seleccionarla.

La topología escogida, tanto por su simplicidad como por su precisión al momento de implementarse en un sistema de reconocimiento de patrones, fue inicialmente la de un perceptrón

multicapa, con entrenamiento simple. Pruebas posteriores demostraron una baja efectividad para el caso concreto del reconocimiento de patrones de voz; por lo que se optó por dar un paso más en los escalones de complejidad y se escogió una **Red MLP *feed forward back-propagation***, es decir con entrenamiento de propagación del error hacia atrás, dada su alta efectividad al momento de implementarse en sistemas de reconocimiento y clasificación.

En trabajos de investigación previos se logró descubrir que si un arreglo de red del tipo perceptrón multicapa o *backpropagation* se entrena asintóticamente como un clasificador 1 de N usando el error medio cuadrático (**MSE**)o algún criterio similar, las salidas se aproximarán a la probabilidad de clase posterior $P(clase|entrada)$, con una precisión que aumenta con el tamaño del set de entrenamiento. Este importante hecho ha sido probado por Gish (1990), Bourlard & Wellekens (1990), Hampshire & Pearlmutter (1990), Ney (1991) y otros a lo largo de estos años.

Las posteriores series de experimentos con la redes fueron un intento por responder la pregunta importante en estos casos "¿Cuál es la arquitectura óptima para la red neuronal en un sistema de reconocimiento de patrones de voz?".

El beneficio de la capa oculta.

Al momento de optimizar el diseño de una red neuronal, la primera pregunta a considerar es si la red debe o no tener una capa oculta.
En teoría, una red sin capas ocultas (un perceptrón simple o **SLP** por sus siglas en inglés) sólo puede formar regiones de decisión lineales, pero se garantiza la obtención de un $100\,\%$ de precisión en la clasificación, si y sólo si el set de entrenamiento es linealmente separable.
En contraste, una red con una o más capas ocultas (un perceptrón multicapa, o **MLP**) puede formar regiones de decisión no lineales, pero se incurre en el problema de quedar atascada en algún mínimo local que puede ser inferior al mínimo global.

Comúnmente se asume que un **MLP** es mejor que un **SLP** para el reconocimiento de voz, porque la voz se encuentra asentada en un dominio altamente no lineal. La experiencia ha demostrado que el problema de los mínimos locales tiene poca injerencia en este tipo de problemas, con excepción de tareas artificiales.
Pero, ¿por qué se dice que una sola capa oculta basta para el diseño?. Esto es debido a:

1. Se ha demostrado, como en[3], que cualquier función que puede computarse por un **MLP** con múltiples capas ocultas también puede ser computada por un **MLP** con una sola capa oculta, si ésta posee un número suficiente de neuronas.

2. La experiencia de los diseñadores de redes neuronales ha enseñado que el tiempo de

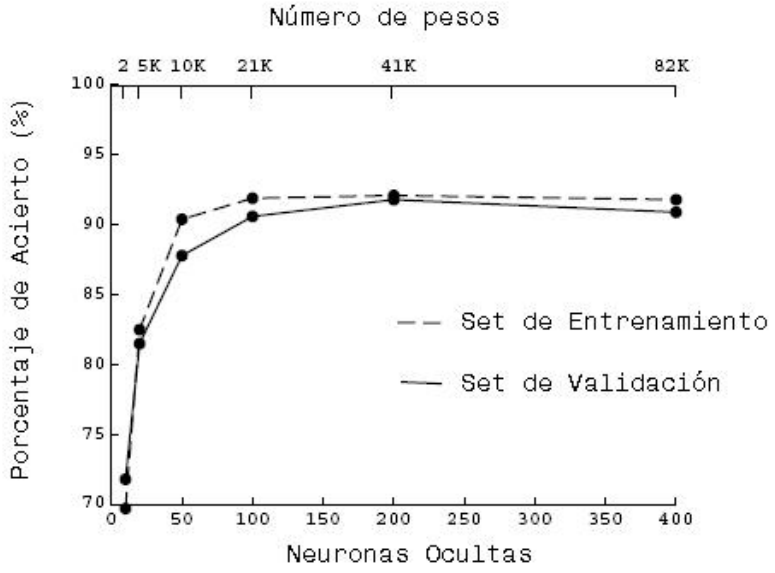

Figura 5.1: El desempeño mejora con el número de neuronas en la capa oculta

entrenamiento se incrementa substancialmente para redes con multiples capas ocultas.

Número de neuronas en la capa oculta

El número de neuronas en la capa oculta tiene un alto impacto en el comportamiento de un **MLP**. Entre más unidades ocultas tenga una red, más complejas son las superficies de decisión que pueda formar y por ende mejor precisión se puede obtener al momento de la clasificación. Sin embargo, más allá de un número de neuronas ocultas, la red puede poseer un amplio poder de modelado, tanto que puede modelar la idiosincrasia de los datos de entrenamiento si se entrena por mucho tiempo, minando por ello el desempeño en los datos de prueba.

La figura 5.1 muestra la precisión del reconocimiento de voz como una función del número de neuronas ocultas, tanto para el set de entrenamiento como para el set de validación. Se puede observar que en esta medición hecha por Tebelskis y cuyo proceso se expone en [26], la precisión aumenta a medida que más neuronas son añadidas a la capa oculta, (por lo menos 400 neuronas). Ésto indica que existe tanta variabilidad en la voz que es virtualmente imposible para una red neuronal memorizar todo el set de entrenamiento.

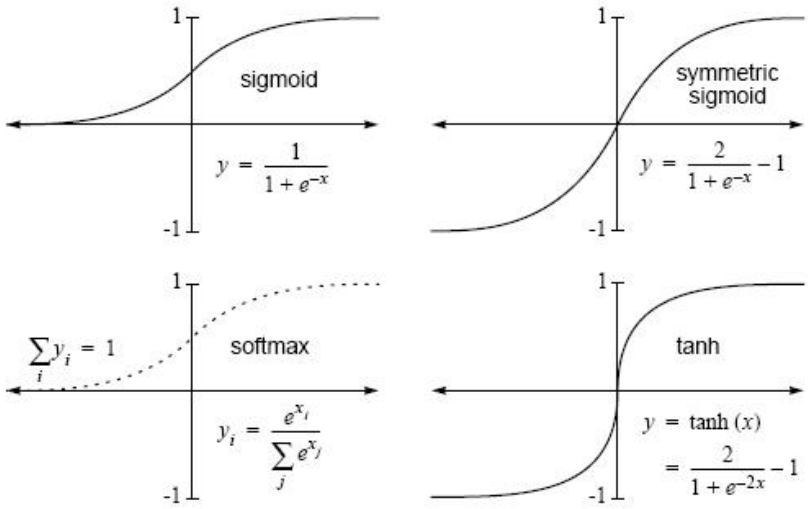

Figura 5.2: Las funciones de transferencia no lineales más populares

Además se espera que el desempeño de la red continue aumentando en una tasa gradual, por encima de las 400 neuronas. Pero el problema empieza en este punto, cuando se desea implementar el sistema en una FPGA. Cada aumento del número de neuronas, aumenta de igual manera el tiempo de computación y por ende el consumo de recursos y de lógica en la tarjeta. Es por ello que en implementaciones normales en computadores secuenciales se limitan dichas neuronas a 100, manteniendo así un excelente compromiso entre precisión en el reconocimiento y requerimientos de computación.

Teniendo en cuenta estas consideraciones, se optó por el diseño de una red con tres capas: Una capa de entrada, una capa oculta y una capa de salida.

La selección de la función de transferencia

La selección de la función de transferencia puede tener una diferencia significativa en el desempeño de la red. Dado que las funciones de transferencia lineales en capas de neuronas lineales no son muy usadas ya que se pueden reemplazar por una sencilla neurona lineal, se ha optado por una función no lineal, continua y derivable para activar las neuronas. Pero, ¿qué función usar? De nuevo una pregunta que sólo puede tener respuesta después de muchos

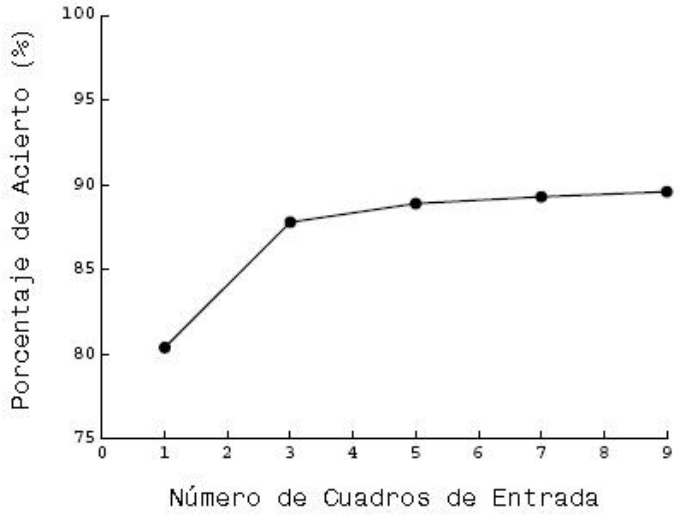

Figura 5.3: Aumentando la ventana de entrada se aumenta la sensibilidad contextual, y por ello la precisión

ensayos y sobre todo de la aplicación en que se quiera usar.

En la figura 5.2 se pueden observar los tipos de funciones más usados al momento de trabajar con redes neuronales, pero sobre todo al trabajar con un sistema de entrenamiento en el que las funciones derivables son requisito fundamental al momento de realizar el ajuste de pesos de la red.

Basado en las consideraciones planteadas en [26], se escogieron 2 funciones para el análisis de la red, la función tangente hiperbólica (llamada de ahora en adelante *tansig*) y la función logarítmica (en adelante conocida como *logsig*). La selección de estas funciones está basada en el hecho ya reconocido de que las redes aprenden mucho más eficientemente cuando se usan funciones de transferencia simétricas (por ejemplo en rangos de $[-1, 1]$) en todas las neuronas que no sean de salida (incluyendo las neuronas de la capa de entrada).

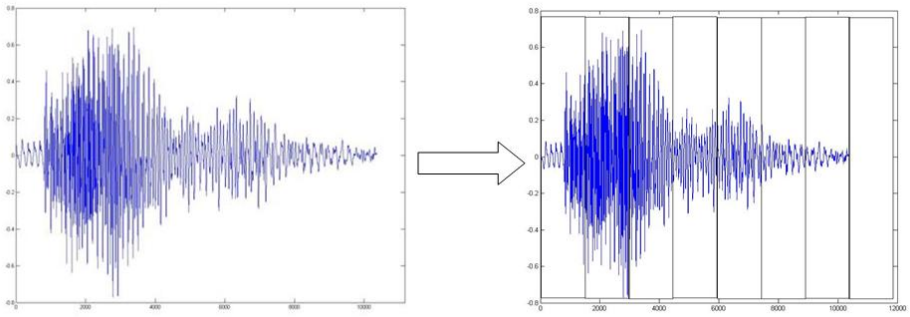

Figura 5.4: Palabra Baile en el dominio del tiempo

5.1.2. Estructura de los datos de entrada

La precisión del sistema de reconocimiento mejora con la sensibilidad contextual del modelo acústico. La mejor manera de mejorar la sensibilidad contextual es mostrar el modelo acústico no como un solo cuadro de voz, sino como una ventana compuesta de muchos cuadros (*enframe*).

Una red neuronal tiene una ventaja sobre los **HMM** y es que la red puede fácilmente mirar a cualquier número de cuadros de entrada, por ende aún modelos de fonemas de contexto independiente pueden volverse arbitrariamente de sensibilidad contextual. Esto indica que puede volverse trivial el incrementar la precisión de la red simplemente incrementando el tamaño de la ventana de entrada.

Como bien se sabe, la voz debe de ser representada como una secuencia de cuadros, resultado de algún tipo de análisis a una señal de voz aplicada a una forma de onda. Sin embargo, no existe un consenso en que tipo de análisis provee el mejor rendimiento; la representación óptima tiende a variar de sistema a sistema.

El algoritmo aplicado para el ventaneo de la señal de voz, muestrea la señal en función del tamaño o número de muestras que se posea de la misma como se puede ver en la figura 5.4 .

Una vez realizado el ventaneo de la señal se procede a la extracción de las características *Cepstrum* de la misma. Estás están consignadas en una matriz de $M \times N$ en donde M indica el número de ventanas o frames de la palabra y N indica el número de coeficientes escogidos, que para el caso de esta aplicación fueron 13 coeficientes.

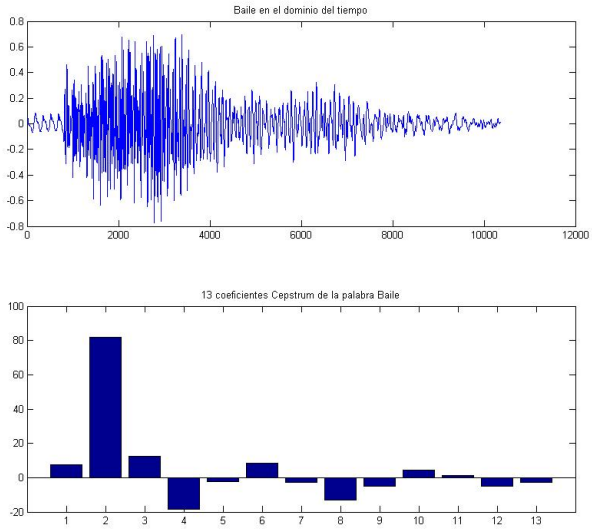

Figura 5.5: Los coeficientes Cepstrum son el resultado de analizar el espectro del espectro

Dichos coeficientes *Cepstrum* analizan el comportamiento "espectral del espectro" y proveen la matriz indicada anteriormente. Pero dado que la implementación de un diseño basado en un matriz de semejante tamaño es poco eficiente, se procedió a reducir la dimensionalidad de los datos de entrada para poseer un solo vector que indique una muestra del comportamiento de la matriz, y esto se logra sacando la media de todos los 13 coeficientes, hasta obtener sólo un vector de $1 \times N$. La figura 5.5 muestra la comparación de tener la señal en el dominio del tiempo a tener un vector que indica el comportamiento del espectro en el espectro de dicha muestra de la palabra.

Normalización de las entradas

Teóricamente, el rango de los valores de entrada no debe afectar el rendimiento asintótico de la red, dado que la red aprende a compensar entradas escaladas con escalas inversas en los pesos, y además aprende a compensar para una media desviada ajustando el *bias* de las unidades ocultas. Sin embargo, es bien sabido que las redes aprenden más eficientemente si las entradas son normalizadas simétricamente alrededor de 0.

Para ello se usó la función *prestd* que posee el paquete Matlab. Esta función pre-procesa

los datos de manera que posean una media de 0 y una desviación estándar de 1.
Lo que hace en definitiva es tomar cada dato, restarlo de la media y dividir el resultado por la desviación estándar.

$$pn = \left(\frac{p - \bar{x}}{\sigma} \right) \tag{5.1}$$

5.1.3. Proceso de entrenamiento y prueba de acierto de la red neuronal

En este trabajo de grado, como se expuso anteriormente, se usaron redes *backpropagation* como topología para el entrenamiento, pero con este marco de trabajo se exploraron muchas variaciones del procedimiento de entrenamiento. En esta sección se mostrará un resumen de los resultados obtenidos con las innumerables variaciones realizadas.

Etapa de pruebas de topologías

La primera etapa del proceso de construcción de la red, fue una larga tarea de "*ensayo y error*", tratando de obtener una buena eficiencia de la red. La primera etapa inicio con pruebas para una red de tres capas, con función de transferencia **tansig** en las dos primeras capas y la función **logsig** en la capa oculta.
Además las pruebas se realizaron con 100 muestras de cada palabra y 39 coeficientes *Cepstrum* para la caracterización. Para el número de neuronas en la capa oculta se uso una fórmula planteada en [5] y que plantea el número de neuronas en función del número de palabras y el número de muestras de la misma.

$$\begin{aligned} CAPA1 &= 2 \times (k + 2) \\ CAPA2 &= k + m \\ CAPA3 &= k \end{aligned}$$

En donde *k* es el número de palabras, y *m* es el número de muestras de la palabra.
Este fue el formato elegido para el entrenamiento de la primera red, de allí en adelante se procedió a variar el número de neuronas en las diferentes capas.
A continuación se muestran los resultados de las primeras 3 redes entrenadas, estas con el número total de muestras dividido a la mitad, una para entrenar y la otra para validar.

Red	**# Neuronas en las capas**	**MSE**	**Epochs**
Red 1	34 - 65 - 15	0.115569855	2587
Red 2	34 - 95 - 15	0.1001201558	3165
Red 3	34 - 125 - 15	0.0979458521	4211

Terminada esta primera etapa y viendo que aumentando el número de neuronas no modificaba significativamente la respuesta esperada de la red, se procedió a empezar a modificar las

neuronas de la capa de entrada y la capa oculta. Las siguientes 3 neuronas se entrenaron realizando una modificación en las capas de entrada y ahora repartiendo los datos de entrada en 4 grupos de 25 muestras cada uno, 3 para entrenamiento y 1 para validar.

Red	# Neuronas en las capas	MSE	Epochs
Red 4	39 - 65 - 15	0.13456589	1614
Red 5	39 - 95 - 15	0.110124568	2154

En este punto se realizó un alto en el camino, pues al realizar la validación con muestras de entrenamiento y con muestras fuera del entrenamiento se observaron los siguientes resultados de acierto.

Red	Test CT	Test OT
Red 1	78 %	13 %
Red 2	78 %	16 %
Red 3	81 %	16 %
Red 4	83 %	22 %
Red 5	87 %	25 %

En esta tabla la medida **CT** se refiere al porcentaje de acierto promedio en el test cerrado (la validación se realiza con los mismos datos que se realiza el entrenamiento) y la medida **OT** indica el porcentaje de acierto promedio en el test abierto (la validación se realiza con muestras diferentes con las que se realiza el entrenamiento).
Al obtener estos valores tan bajos y con tantas neuronas se realiza un replanteo completo de la red. Para ello se proponen y se implementan, como primera medida, los siguientes cambios:

- Reducción del número de coeficientes de caracterización (*Cepstrum*) de 39 a 13, es decir, trabajar sólo con las características estáticas de la señal.
- Se hará una división de las muestras en 10 grupos, y se usarán 8 para entrenamiento y 2 para validación.
- El número de palabras se reducirá de 15 a 8 para aumentar el rendimiento de la red.

De nuevo los resultados obtenidos del entrenamiento de las redes con estas características se muestran a continuación.

Red	# Neuronas en las capas	MSE	Epochs
Red 6	20 - 20 - 8	0.0109375	114
Red 7	13 - 13 - 8	0.0246094	226
Red 8	13 - 30 - 8	0.0119141	100
Red 9	13 - 10 - 8	0.0548016	1000

Los valores de aciertos de las redes entrenadas con estos nuevos parámetros son los siguientes:

Red	Test CT	Test OT
Red 6	91.7188 %	38.125 %
Red 7	81.72 %	38.125 %
Red 8	91.0938 %	39.375 %
Red 9	57.8125 %	31.25 %

De nuevo, aunque los valores de acierto en Test Cerrado son significativamente altos, los de validación con Test Abierto aún no llenan las expectativas.
El siguiente cambio a realizar fue el de modificar las funciones de activación, y sólo trabajar con funciones **Logsig** en todas las neuronas manteniendo la topología de la *Red 7*. Los resultados del entrenamiento fueron:

Red	# Neuronas en las capas	MSE	Epochs
Red 10	13 - 13 - 8	0.0289063	167
Red 11	13 - 23 - 8	0.0222656	267

Y los de acierto que se obtuvieron son:

Red	Test CT	Test OT
Red 10	80.3125 %	41.25 %
Red 11	80.3125 %	41.25 %

Con estos resultados, aunque se mejoró en algo el acierto, no se obtuvo el rendimiento esperado. Por ello, se procedió a realizar una última modificación.

- Reducir el número de muestras, es decir, realizar una limitante de género y trabajar sólo con muestras de hombres.

En este punto se realizó el entrenamiento para una sola red, en este caso, es una red con funciones **Logsig** en todas sus neuronas, se trabajó con dos grupos cada uno de 25 muestras uno para entrenamiento y el segundo para validación. Los resultados que se obtuvieron para el entrenamiento fueron:

Red	# Neuronas en las capas	MSE	Epochs
Red 12	13 - 13 - 8	0.025	275

Teniendo en cuenta el no tan bajo **MSE** se procedió a realizar la última modificación:

- Conformar 5 grupos de 5 muestras por palabra, en donde serán usados 4 para entrenamiento y 1 para validación.

En el entrenamiento de la red se obtuvo lo siguiente:

Red	**# Neuronas en las capas**	**MSE**	**Epochs**
Red 13	13 - 13 - 8	0.00859375	113

Y finalmente los valores de acierto obtenidos fueron:

Red	**Test CT**	**Test OT**
Red 13	94.0625 %	55 %

Estos valores fueron ampliamente satisfactorios, y cumplían en gran medida las expectativas que se tenían acerca del rendimiento de la red.
Pero además se decidió trabajar con esta topología, por algo muy importante y que no se ve a simple vista, pero que juega un papel más que importante en la implementación de la red en VHDL y posteriormente en la tarjeta: La *Universalidad* de este diseño.

Pero, ¿por qué *Universalidad* en el diseño de la neurona?. Muy simple, porque si se observa con detenimiento la arquitectura de la neurona, así como la de los datos de entrada y salida se puede observar:

1. El número de neuronas de la capa de entrada es igual al número de coeficientes *Cepstrum* que provienen de cada palabra. Lo que aumenta considerablemente el rendimiento de la red.

2. Las neuronas de la capa de salida, son iguales al número de posibles clases de opciones que tiene la red, es decir, igual al número de palabras que se tienen para la clasificación. Esto hace que la red posea una salida binaria, a la que a cada clase le corresponde un bit, es decir:

$$
\begin{array}{ccc}
00000000 & \Rightarrow & \text{Ninguna Palabra} \\
10000000 & \Rightarrow & \text{Baile} \\
01000000 & \Rightarrow & \text{Bola} \\
00100000 & \Rightarrow & \text{Coco} \\
00010000 & \Rightarrow & \text{Choza} \\
00001000 & \Rightarrow & \text{Gato} \\
00000100 & \Rightarrow & \text{Gol} \\
00000010 & \Rightarrow & \text{Jugo} \\
00000001 & \Rightarrow & \text{Mano}
\end{array}
$$

   Lo cual permite a la red generar mejores hiperplanos, es decir, dividir el posible universo de respuestas en clases significativas, evitando el traslapo de superficies de clasificación, y dándole a la red mejor capacidad de acierto.

3. Las neuronas de la capa oculta, aunque son significativamente pocas, al tener las mismas características de las neuronas de la capa de entrada, refuerzan el nivel de entrenamiento de la red, permitiendo así una mejor clasificación.

4. Las funciones de transferencia de las neuronas de las tres capas son las mismas, lo cual no modifica la topología de la red al momento de la implementación, y mejor aún, limita desde la salida de la primera capa, los valores a estar entre 0 y 1.

5. Y el punto más importante de todos: Si se piensa en implementación, se debe de pensar en una fuente *Universal*, es decir, lograr que un diseño sea capas de ser implementado o interconectado muchas veces sin hacerle modificaciones. Este es el caso de las neuronas de la red. Todas las 34 neuronas son las mismas, todas, debido a la topología de la red, deben de tener 13 entradas, las funciones de activación son las mismas, la única variación se encuentra en los pesos, que es la única variable de la neurona.

5.1.4. Preparación de la red y sus datos para la implementación

El desarrollo que compete la siguiente sección, no es más que el preparar tanto la red como los datos para su posterior implementación en la FPGA.

Ya con la red 13 escogida como la red a implementar por todas las características anteriormente expuestas, se procede a extraer toda la información necesaria para su posterior implementación, es decir, se procede a extraer los valores de los *pesos* y los *bias* de todas las neuronas.

Toda esta etapa se realizó usando de nuevo Matlab, y usando los programas generados, que se incluyen con este trabajo.

Para tener en cuenta, la notación usada será:

- Las entradas corresponderán a un vector columna de 13×1 que corresponden a los 13 coeficientes de la palabra. Enumeradas desde in_0 hasta in_12.

- Los pesos de cada neurona serán expresados en función de la capa a la que pertenezca la neurona, el número de la neurona, y el peso. Todos se enumerarán como en electrónica digital, iniciando desde el cero. Como ejemplo, el peso 4 de la neurona 7 de la capa oculta: l1_n6_w3. En total se tendrán 442 pesos más 34 bias.

- Las salidas pertenecen a un nuevo vector columna de 8×1 que corresponden a las 8 posibles palabras.

El paso siguiente es el desarrollo mediante una algoritmia simple de la llamada *Red Visible*.

Este es un programa creado en Matlab que permite observar el comportamiento de los datos que entran a la red, a través de su paso por todas las neuronas y todas las capas.

Todo este proceso de observación de datos se implementó con un simple propósito: lograr que la implementación es la FPGA sea lo más óptima posible, debido a que la traducción de los valores numéricos reales debe pasarse a alguno de los formatos de números binarios existentes y por ende se debe de tener en claro y conocer los rangos de los posibles valores que pueden tomar los datos al pasar por la red, para de allí realizar una mejor discretización de los mismos.

Los resultados obtenidos para los 5 grupos de trabajo se muestran a continuación, en ellos están expuestos los valores máximos y mínimos obtenidos en los multiplicadores y los sumadores en cada capa.

- **GRUPO 1** (76 Aciertos)

Valor	**Mult_l0**	**Sum_l0**	**Mult_l1**	**Sum_l1**	**Mult_l2**	**Sum_l2**
Max	94.5647	21.5378	57.3045	81.2506	96.8456	95.8848
Min	-126.4479	-20.8952	-42.2215	-56.2326	-97.4663	-288.1226

- **GRUPO 2** (76 Aciertos)

Valor	**Mult_l0**	**Sum_l0**	**Mult_l1**	**Sum_l1**	**Mult_l2**	**Sum_l2**
Max	94.5647	23.0674	57.6002	76.9047	96.8456	127.5003
Min	-126.5060	-25.5829	-42.1885	-72.9785	-97.4663	-304.3012

- **GRUPO 3** (74 Aciertos)

Valor	**Mult_l0**	**Sum_l0**	**Mult_l1**	**Sum_l1**	**Mult_l2**	**Sum_l2**
Max	94.4784	21.8576	57.5439	74.0545	96.8456	96.2398
Min	-126.3905	-23.9375	-42.3641	-82.8041	-97.4663	

- **GRUPO 4** (75 Aciertos)

Valor	**Mult_l0**	**Sum_l0**	**Mult_l1**	**Sum_l1**	**Mult_l2**	**Sum_l2**
Max	94.6915	23.2520	57.7883	84.8046	96.8456	98.8652
Min	-126.6756	-21.7679	-41.9212	-54.5705	-97.4663	-294.8359

- **GRUPO 5** (44 Aciertos)

Valor	Mult_l0	Sum_l0	Mult_l1	Sum_l1	Mult_l2	Sum_l2
Max	94.6915	23.2520	57.7883	84.8046	96.8456	98.8652
Min	-126.6756	-21.7679	-41.9212	-54.5705	-97.4663	-294.8359

Observando detenidamente los valores obtenidos se pueden encontrar los valores máximos y mínimos que se tendrán con los datos de entrada en las neuronas y estos son:

	Mult	Sum
Max	96.8456	127.5003
Min	-126.6756	-323.9557

Tabla 5.1: Valores máximos y mínimos a lo largo de la red

Ya con estos datos, se procede a la configuración de la FPGA, y el trabajo en la implementación de la red neuronal.

5.2. Implementación de la red neuronal en la FPGA

El proceso de la implementación de la Red Neuronal conocida en este trabajo de grado como la **Red 13**, estuvo dividida en dos etapas:

Ciclo 1 La etapa inicial estuvo enfocada a implementar una neurona y posteriormente una red neuronal pequeña con los datos y valores en formato *punto flotante precisión simple.*

Ciclo 2 La etapa final del proceso de implementación. En esta se implementa la red neuronal **Red 13** sobre la FPGA, con datos y valores en formato *punto fijo*.

Cada uno de los ciclos se explicará en las secciones siguientes.

5.2.1. Implementación de una red neuronal en punto flotante

Antes de iniciar, vale hacer una pequeña aclaración. Durante mucho tiempo, la programación en diferentes lenguajes (háblese de C++, Java, etc) se ha hecho mediante el desarrollo de algoritmos de lógica secuencial, es decir, los procesos requieren que se ejecute una acción

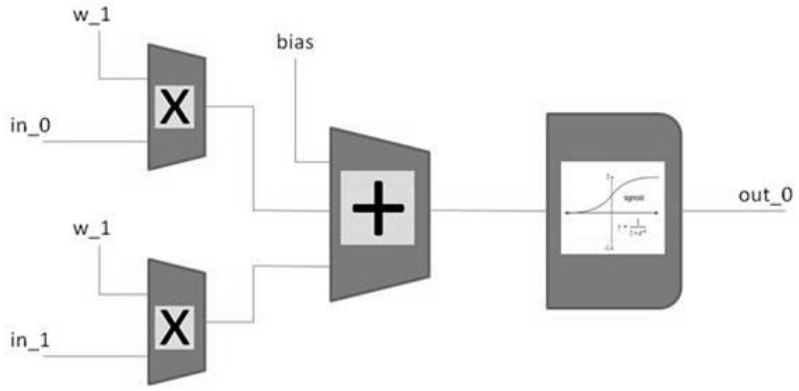

Figura 5.6: Una neurona vista en términos de sus componentes

antes de proseguir con otra. Cuando se hace el cambio a un lenguaje de descripción de *Hardware* (VHDL en este caso), se debe dejar de pensar de manera secuencial.
Esa fue la primera dificultad que se encontró al iniciar esta etapa. Cuando se habla de una red, se piensa, al momento de programar, en un dato que se propaga a lo largo de la misma, sometido a diferentes cambios debido a multiplicaciones, sumas y la aplicación de funciones matemáticas a la salida de la red.
En VHDL no. Ahora se debe ver la red como un grupo de sumadores, multiplicadores, registros y comparadores, todos interconectados y trabajando en paralelo para generar a la salida un valor deseado.

Por eso el desarrollo inicial se enfocó a programar sólo una neurona, un perceptrón simple con función de transferencia *hardlim*. La figura 5.6 muestra como se debe de pensar en una neurona al momento de tratar de implementarla en VHDL.

Punto Flotante en VHDL

Como se explica en el apéndice B el formato de punto flotante en precisión simple es una herramienta de representación binaria de datos que permite la representación de cualquier número real. Ahora se explicará como se realizó la implementación de la neurona en punto flotante en VHDL.

En la figura 5.7 se ilustra la implementación de la neurona en VHDL. Dicha neurona está compuesta de multiplicadores, sumadores, comparadores y registros. Cada uno de ellos implementado para trabajar a 32 bits haciendo uso de los *IPcores* de Xilinx para el trabajo con

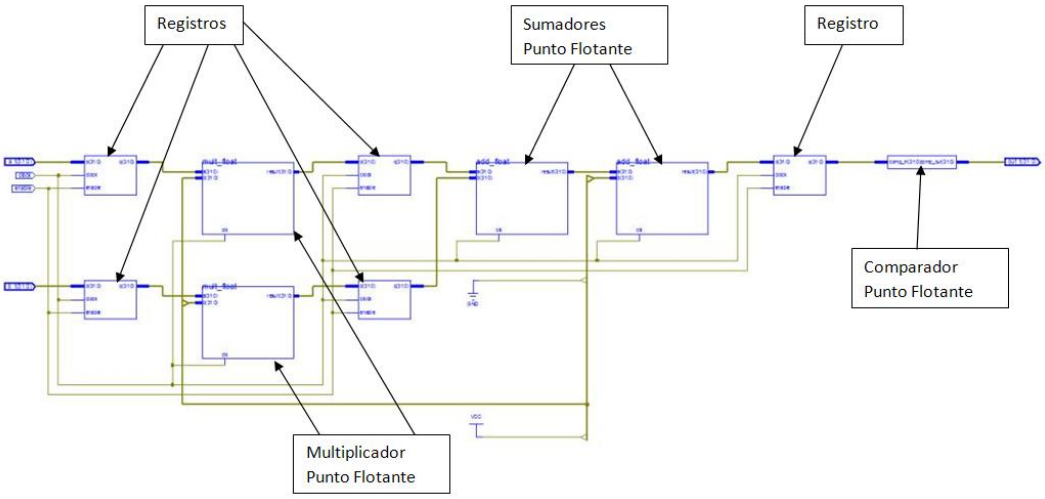

Figura 5.7: Neurona implementada en VHDL en formato Punto Flotante

punto flotante.
El diseño es sencillo en cuanto a su conceptualización pero usa muchos recursos de la lógica embebida de la FPGA. Es decir que para el desarrollo de las operaciones tanto aritméticas (suma, multiplicación) como lógicas (comparación), asi como para los registros, se debe de hacer uso de la lógica interna de la FPGA.

¿Cómo funciona esta neurona?

La figura 5.7 permite una percepción amplia de como funciona la red, pero para mayor entendimiento se dirá que: La neurona posee dos entradas físicas (es decir entradas a pines de la FPGA) los cuales se encargan de entrar las dos señales de 32 bits a la entrada de la red. Además posee también una salida física de 32 bits que se encarga de dar el valor a salida de la red.

Al momento de los datos entrar a la red se hacen pasar por un registro, que no es más que un simple flip/flop tipo D, que se encarga de sostener los datos durante un ciclo de reloj, y mantener todo el flujo de datos sincronizado. Posteriormente los datos entran al primer bloque de cálculo, en este caso dos multiplicadores. Dicho bloque es una instancia de multiplicadores en punto flotante generados mediante el **IPcore** de *Xilinx*, estos se encargan de tomar el dato de entrada válido, y sostenido en este ciclo de reloj, y multiplicarlo por el valor del peso obtenido en el entrenamiento.

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	261	28800	0%
Number of Slice LUTs	1397	28800	4%
Number of fully used Bit Slices	128	1530	8%
Number of bonded IOBs	98	220	44%
Number of BUFG/BUFGCTRLs	1	32	3%
Number of DSP48Es	8	48	16%

Figura 5.8: Resultados de la neurona en punto flotante

En el siguiente ciclo de reloj, los resultados de las multiplicaciones pasan por otro registro para mantener síncrono el flujo de datos y enseguida ingresan en un primer sumador de punto flotante, de nuevo generado mediante los **IPcore's** de *Xilinx*, y el resultado pasa a otro sumador, (en el siguiente ciclo de reloj), donde se opera con el *bias* de la red.

Dado que el perceptrón se diseño usando una función *Hardlim* como función de activación, su implementación fue bastante simple, se limitó a un simple comparador; para valores mayores que 0 la salida es 1 y para valores menores la salida es 0. Este último implementado directamente con VHDL.

La figura 5.8 muestra los resultados de implementación de esta neurona sencilla. De los resultados se pueden concluir varias cosas, aunque se cumplen con las limitaciones de cantidad de lógica para la FPGA, una proyección de estos resultados para la red completa, a simple vista se deduce que es inviable su implementación.

Para corroborar esta conclusión, se procedió a realizar un experimento simple, la implementación y prueba de un perceptrón multicapa capaz de implementar la función OR exclusiva (**XOR**).

Aprovechando la capacidad de realizar instancias de componentes anteriormente creados, se procedió a generar una red neuronal simple, de sólo dos capas, dos neuronas en la capa de entrada y una en la capa de salida. Con este diseño, la red está en capacidad de generar planos de clasificación independientes y mucho más complejos. En la figura 5.9 se puede observar como la neurona previamente creada, ha sido instanciada tres veces para crear el perceptrón multicapa. Los valores de los pesos y los bias fueron obtenidos previamente haciendo uso de

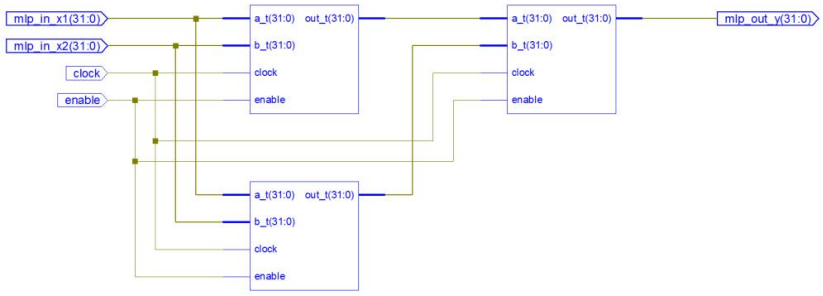

Figura 5.9: Red simple de 3 neuronas y 2 capas

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	550	27,392	2%	
Number of 4 input LUTs	3,338	27,392	12%	
Logic Distribution				
Number of occupied Slices	1,950	13,696	14%	
Number of Slices containing only related logic	1,950	1,950	100%	
Number of Slices containing unrelated logic	0	1,950	0%	
Total Number of 4 input LUTs	3,559	27,392	12%	
Number used as logic	3,338			
Number used as a route-thru	221			
Number of bonded IOBs	98	416	23%	
Number of MULT18X18s	24	136	17%	
Number of BUFGMUXs	1	16	6%	

Figura 5.10: Recursos usados por la red de 3 neuronas y 2 capas

la red visible desarrollada en Matlab.

Los resultados del uso de recursos en la FPGA se puede observar en la figura 5.10. Es en este caso donde se puede concluir que aunque a pesar de que la red funciona y genera los resultados esperados, el consumo de recursos es demasiado alto y hace inviable una implementación de redes de mayor tamaño.

Es por eso que un replanteo del formato de los datos se hace necesario en este punto de la investigación. El problema al momento de seleccionar un formato de datos radica en su eficiencia al momento de la implementación, su facilidad al momento de trabajar y a su bajo consumo en recursos.

Como se pudo observar con el punto flotante, aunque permite una amplia gama de rangos

de trabajo en la representación de cualquier número real, (desde $-\infty$ hasta $+\infty$), también genera un alto consumo de recursos en la FPGA. Ahora, la representación binaria simple o BCD, no genera posibilidad de desarrollo del sistema debido a su limitada representación de datos, el punto flotante aunque ilimitado en su representación, consume muchos recursos; es por eso que se procede a hacer uso del punto fijo.

5.2.2. Implementación de la red neuronal final en punto fijo

En la mayoría de los procesadores disponibles comercialmente no existe un soporte de *hardware* para aritmética en punto flotante debido a los costos que el silicio extra le impondría al costo total del procesador. Aplicaciones que necesiten este tipo de representación deberían hacer uso de *software* de emulación de aritmética en punto flotante. Este tipo de *software* puede limitar significativamente la tasa a la cual los algoritmos son ejecutados.

Implementando algoritmos que usen matemáticas en punto fijo, un significativo aumento en la velocidad de ejecución se puede observar debido a la naturaleza entera de los valores en este formato y que son soportados en la mayoría del *hardware* de procesadores actuales. El tema principal a tener en cuenta es que el mejoramiento en la velocidad trae consigo el costo de reducir el rango de trabajo y la exactitud de las variables en los algoritmos.

Es por esta razón que se optó por la representación de punto fijo para la implementación de la red neuronal, debido a su bajo consumo de recursos y a la alta fiabilidad al momento de representar valores.

La siguiente etapa del proceso, se centró en la selección de un formato de representación apropiado (es decir, que número de bits se usarían para la parte entera (BI) y cuantos se usarían para la parte decimal (BF), para más detalle ver Anexo C). Para ello se hizo uso de los valores hallados anteriormente en la tabla 5.1.

Posteriormente se procedió a determinar el valor entero máximo. En este caso -323, a su vez usando las fórmulas para calcular **BI** (Anexo C), se tiene que la parte entera debe de tener 9 bits que equivalen a:

$$\text{BI} = 101000011 = 323$$

Con esta cantidad de bits se podría tener una representación hasta el número 511, que es significativamente mayor que el valor máximo a propagar por la red. El número de bits de la parte entera más el bit de signo me da la primera parte del formato $\%10$.BF, en donde 10 me indica que existen 9 bits en la parte entera más un bit de signo.

Ahora para la parte decimal se propuso realizar una serie de pruebas con la red visible creada en Matlab, para verificar que tanto varía el rendimiento de la red en cuanto a exactitud dependiendo de el número de decimales, y dependiendo de este número de decimales tendremos un número definido de bits (BF).

RESULTADOS DISCRETIZACIÓN

Resultados iniciales obtenidos de Matlab

$$\begin{aligned} \text{Grupo 1} &= 76 \\ \text{Grupo 2} &= 76 \\ \text{Grupo 3} &= 74 \\ \text{Grupo 4} &= 75 \\ \text{Grupo 5} &= 44 \end{aligned}$$

A continuación se muestran los resultados obtenidos de la cantidad de aciertos al variar el número de decimales en la red visible en Matlab, para poder conocer el número de bits en la parte decimal que se van a usar (BF).

- Con 6 decimales equivalentes a 6 bits en BF (pasos de $0{,}015625$)

$$\begin{aligned} \text{Grupo 1} &= 59 \\ \text{Grupo 2} &= 58 \\ \text{Grupo 3} &= 63 \\ \text{Grupo 4} &= 61 \\ \text{Grupo 5} &= 39 \end{aligned}$$

- Con 7 decimales equivalentes a 7 bits en BF (pasos de $0{,}0078125$)

$$\begin{aligned} \text{Grupo 1} &= 66 \\ \text{Grupo 2} &= 66 \\ \text{Grupo 3} &= 67 \\ \text{Grupo 4} &= 66 \\ \text{Grupo 5} &= 42 \end{aligned}$$

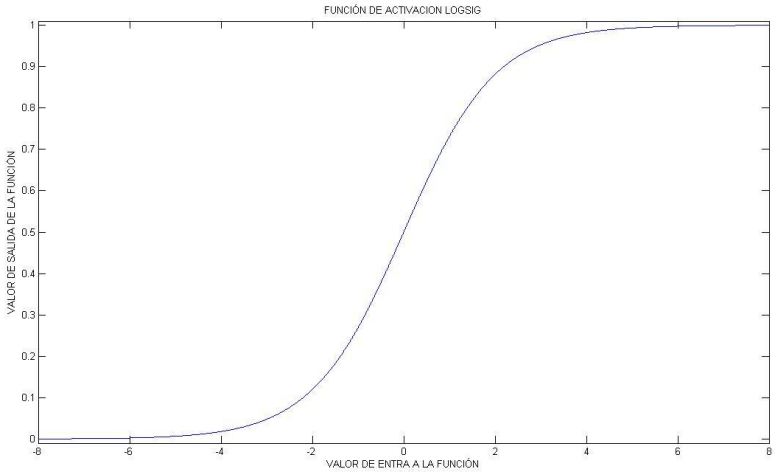

Figura 5.11: Función Logsig

- Con 8 decimales equivalentes a 8 bits en BF (pasos de $0{,}00390625$)

$$\begin{aligned} \text{Grupo 1} &= 73 \\ \text{Grupo 2} &= 74 \\ \text{Grupo 3} &= 73 \\ \text{Grupo 4} &= 73 \\ \text{Grupo 5} &= 46 \end{aligned}$$

Finalmente con estos resultados se estableció que el número óptimo de decimales para obtener un rendimiento deseado es de 8, que equivalen a 8 bits en la parte decimal (de nuevo usando las fórmulas del Anexo C) es decir 256 pasos de $0{,}00390625$.
Ahora ya se puede definir el formato en el que está representada la red en punto fijo $\%10{,}8$, el cual indica que se tiene 1 bit de signo, 9 bits en la parte entera y 8 bits en la parte decimal, lo que en resumen indica que se trabajarán datos de 18 bits.

Una vez establecido el formato de trabajo y habiendo verificado que tanto el rendimiento de la red como la precisión de la respuesta no bajaron, se procedió a realizar la implementación de los componentes de la **Red** 13. Al igual que como se hizo para la red neuronal en punto flotante, en este caso se implementaron uno a uno los componentes de la red (multiplicador, sumador, función de transferencia y registros). Las principales diferencias están en los siguientes puntos:

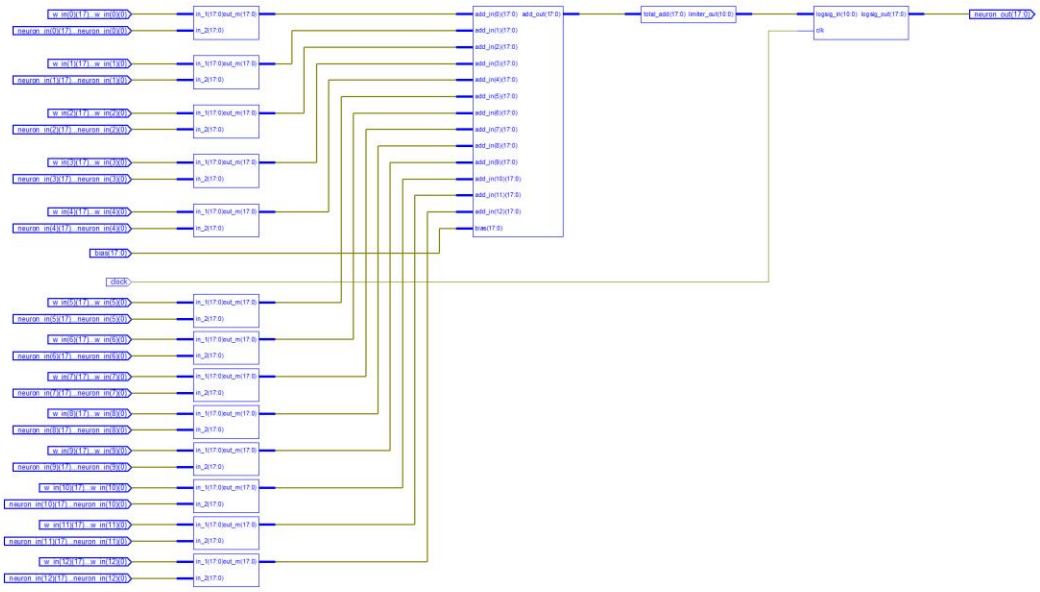

Figura 5.12: Neurona final diseñada en VHDL en punto fijo

- El diseño de los componentes básicos es muy simple, no hay que hacer uso de IpCores de Xilinx ni ningún tipo de fuentes complejas para realizarlo.
- La función de transferencia no es tan sencilla como un simple comparador mayor que, ahora se convierte en una función continua y derivable.

El desarrollo de la implementación fue similar al de la implementación de la red en punto flotante, con la ventaja ahora que los diseños estaban simplificados a simples sumas y multiplicaciones binarias (debido a escoger el formato punto fijo), lo complicado en este caso es la implementación de la función de transferencia, que se implementó desarrollando una memoria ROM.

La idea de utilizar la memoria ROM como función de transferencia nace de la naturaleza inicial de este trabajo de grado, la necesidad de implementar con el mínimo gasto de recursos.

La implementación se basó en un proceso que se llamará “Análisis de discretización”. El objetivo de este proceso al igual que la primera etapa de discretización es lograr encontrar la representación binaria más apropiada para la implementación de la función de transferencia en cuestión. Al observar la figura 5.11, se pueden determinar dos puntos frontera que para el

análisis de este trabajo de grado serán -4 y 4. Se puede concluir que valores inferiores a -4 serán automáticamente asignados a el valor 0 y valores por encima de 4 serán asignados a la salida 1. El problema se encuentra en el rango $(-4, 4)$ pues como se puede ver en la figura 5.11 es en este intervalo donde existe un cambio significativo de valores en la función.

De nuevo haciendo uso del análisis de discretización, se procedió a realizar el diseño en memoria ROM del intervalo mencionado anteriormente. Se tuvo en cuenta los siguientes puntos:

- Se analizarán los valores ubicados entre $-3{,}99609375$ y $3{,}99609375$.
- La representación binaria de estos valores usando las ecuaciones de el Anexo C es:

$$\begin{aligned} -3{,}99609375 &= 11111111111 \\ 3{,}99609375 &= 01111111111 \end{aligned}$$

- La entrada a la memoria ROM sólo hará uso de los 11 bits menos significativos del vector de 18 bits que genera el sumador.
- Dichos valores de entrada a la ROM serán valores entre $-3{,}99609375$ y $3{,}99609375$, es decir, 00000000000 y 11111111111 en valores binarios.
- Los valores de salida estarán en el rango 00000101 que es equivalente a $0{,}0195$ y 11111011 que equivale a $0{,}9805$

Teniendo en cuenta estos items, se concluye que se tendrá una memoria ROM de $2048x8$, es decir, una salida de 8 bits y 2048 salidas posibles. Teniendo siempre en cuenta que los valores negativos que entran a cualquiera de los módulos de la red estarán en complemento a dos.

La etapa siguiente es un comparador de salida. La labor de dicho comparador es entregar unos o ceros a la salida, para que la clasificación se haga sobre los valores asignados a las palabras por identificar.

Esta tarea estuvo a cargo de un comparador con un valor limite, el cual se encarga de limitar los valores de salida a 1 y 0. La salida se asignó, haciendo uso de la red visible de Matlab, encontrando como valor límite $0000000000011010111 = 0{,}83803$. El limitador al final entregará 1 a la salida con valores que entregue la ROM por encima de $0{,}83803$ y a los demás les asignará 0.

Un diseño jerárquico de la red neuronal, haciendo uso de las instancias creadas individualmente, se puede observar en la figura 5.13. En esta gráfica se expone el diseño en VHDL de la

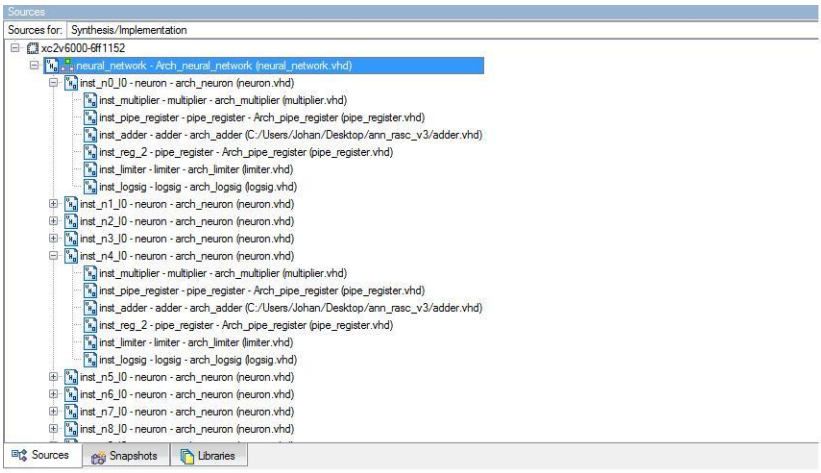

Figura 5.13: Neurona final diseñada en VHDL en punto fijo

red neuronal mediante la creación de los módulos, anteriormente descritos, de manera individual y que finalmente son unidos en la red final, para su diseño y posterior implementación.

Capítulo 6

Resultados y conclusiones

6.1. Condiciones de prueba

- **Normalización estadística**. Se hizo uso del método *premnmx* para la normalización en amplitud de las muestras adquiridas.
- **Estrategia de validación**. Se implementó la validación simple, en la cual el conjunto total de muestras se divide aleatoriamente en dos partes: una se usa como el conjunto de entrenamiento para ajustar los parámetros del modelo del clasificador. El otro conjunto, llamado el conjunto de validación, se usa para estimar la probabilidad de error del clasificador [18].
- **Extracción de parámetros**. Se utilizaron 13 características estáticas para el análisis MFCC. A estas características iniciales se le agregaron características dinámicas de primer y segundo orden, sólo para análisis en Matlab, obteniendo vectores de 39 parámetros.
- **Especificación de desempeño**: La medida CT indica el porcentaje de acierto promedio en el test cerrado (la validación se realiza con las mismas muestras con que se realiza el entrenamiento). La medida OT indica el porcentaje de acierto promedio en el test abierto (la validación se realiza con muestras diferentes con las que se realiza el entrenamiento).

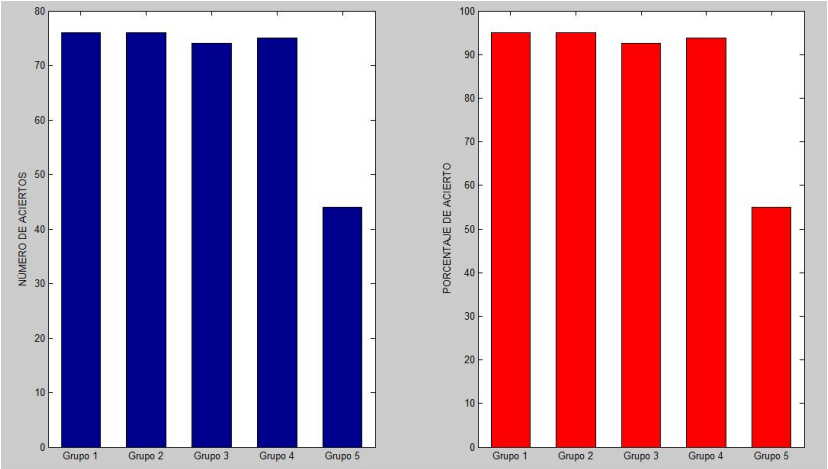

Figura 6.1: Resultados en aciertos y porcentaje de aciertos de la red 13

6.2. Resultados del clasificador en Matlab

6.2.1. Resultados del clasificador con muestras de entrenamiento y de validación

El sistema clasificador implementado en Matlab se desarrollo usando la red13 que se expuso en el capítulo anterior. A dicha red entrenada se le ingresaron las muestras de entrenamiento y validación para su clasificación, obteniendo los resultados de la figura 6.1.

Los grupos 1 al 4 pertenecen a las muestras de entrenamiento y el grupo 5 está compuesto por las muestras de validación. En la tabla 6.1 se encuentran los datos de entrenamiento de

RED	NEURONAS	EPOCHS	MSE
Red13	13 - 13 - 8	113	0,00859375

Tabla 6.1: Resumen de entrenamiento Red13

la red neuronal usada. El entrenamiento de realizó en Matlab en 113 epochs y se obtuvo un MSE de 0,00859375. Se realizó la prueba de validación simple, tanto para los datos de entrenamiento como para los de validación.

En la tabla 6.2 se tienen los resultados obtenidos al realizar la validación de los datos de entrenamiento y de validación. Los aciertos obtenidos en el test CT se obtuvieron con relación

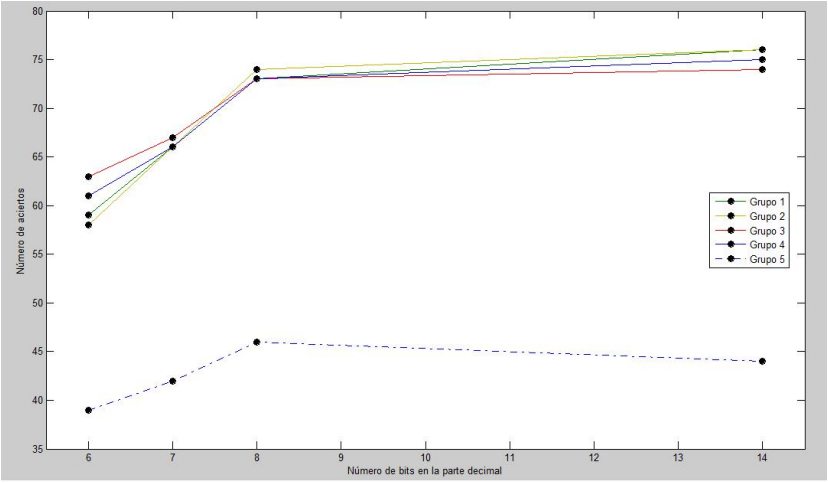

Figura 6.2: Resultados del número de aciertos en función del número de bits en la parte decimal (BF)

al número total de muestras de entrenamiento (320) y los aciertos en el test OT se obtuvieron con relación a las muestras de validación (80). La figura 6.2 muestra los resultados obtenidos

PARÁMETROS DE LA RED	**TEST CT**		**TEST OT**	
	ACIERTOS	%	ACIERTOS	%
MFCC 13	301	94,0625	44	55

Tabla 6.2: Resultados de la validación simple

al momento de realizar el análisis de discretización para el formato binario a escoger. En ella se puede observar como el acierto varía de forma significativa respecto a la cantidad de bits en la parte decimal. El valor mayor en el eje de las abscisas es el número de bits con el que trabaja Matlab para realizar los cálculos. Para obtener estos resultados, de nuevo se hizo uso de la red visible que se anexa a este trabajo de grado.

Además se realizó una prueba de comparación del rendimiento de la red en cuanto a acierto, utilizando un Bayesiano lineal para la clasificación de los 8 eventos. Los resultados en validación se obtuvieron usando el método 70-30 (70 % de los datos para entrenamiento y 30 % para validación). Se realizaron 11 pruebas para sacar una media de los errores. La tabla 6.3muestra un resumen de los resultados.

Para el análisis final y con el fin de realizar la comparación de rendimiento en términos de

BAYESIANO LINEAL	
PRUEBA REALIZADA	**PORCENTAJE DE ERROR**
Prueba 1	58,33 %
Prueba 2	70 %
Prueba 3	60,83 %
Prueba 4	56,67 %
Prueba 5	55 %
Prueba 6	63,33 %
Prueba 7	60 %
Prueba 8	61,67 %
Prueba 9	56,67 %
Prueba 10	52,50 %
Prueba 11	50 %
Error Medio	58,634 %

Tabla 6.3: Resultados de la validación utilizando Bayes

velocidad de procesamiento, se realizó una prueba con las muestras tomadas.

6.2.2. Resultados de la prueba de velocidad

Con la red entrenada en Matlab y bajo los parámetros anteriormente expuestos, se le entregó a la red varios paquetes de datos, cada uno, con un número diferente de muestras, con el fin de determinar la velocidad (en términos de tiempo) a la cual rinde la red neuronal artificial en un procesador de propósito general.

La primera prueba se realizó entregándole a la red las muestras en forma de serie es decir, se usó un algoritmo para entregar las muestras, de manera que cada vez se entrega una muestra más que al momento anterior, es decir:

$$\begin{aligned} f(t) &= n \\ f(t+1) &= n+1 \\ f(t+2) &= (n+1)+1 \\ f(t+3) &= ((n+1)+1)+1 \end{aligned}$$

en donde n es el número de muestras en un momento dado. Así, hasta completar las 400 muestras. Para el cálculo de tiempo se hizo uso de la función *tic toc* de Matlab. Esta función

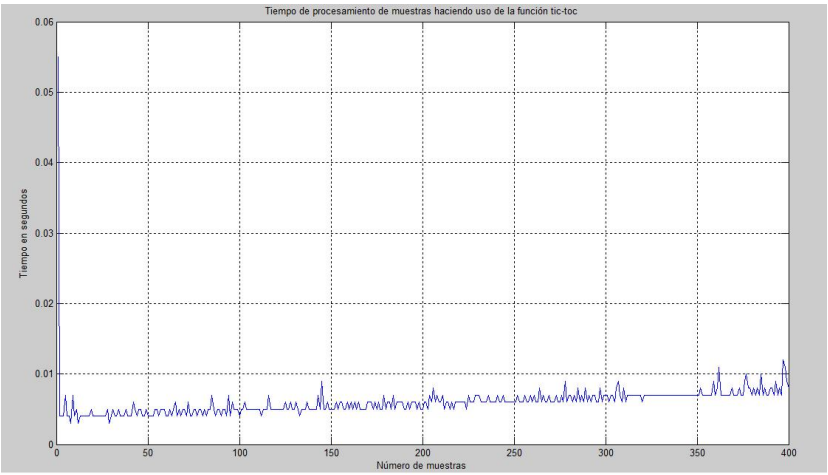

Figura 6.3: Resultados del tiempo empleado para procesar las muestras

se encarga de medir el tiempo transcurrido desde que se usa *tic* hasta que se usa *toc*. En este caso, *tic* se incorpora al programa al momento de ingresar la muestra y *toc* al momento de terminar la simulación de la red con dicha muestra.La figura6.3 muestra los resultados de tiempo empleado por numero de muestras.

El pico de tiempo usado por el procesador de propósito general está en la primera muestra, en donde el tiempo empleado es de $0{,}055s$. En la tabla 6.4 se puede observar los valores pico de tiempo empleados por el procesador de propósito general para obtener resultados de la red neuronal.

MUESTRAS	**TIEMPO (s)**
1	$0{,}055S$
145	0,009
362	0,011
397	0,012

Tabla 6.4: Valores pico de tiempo al procesar muestras

La segunda prueba se realizó de la misma manera que la anterior sólo que haciendo uso de la función *cputime*. Esta función se encarga de entregar el tiempo total de CPU (en segundos) usado por Matlab desde el momento en que se inicia. La figura 6.4 muestra los resultados del tiempo empleado para procesar muestras. De nuevo el mayor pico de tiempo se encuentra

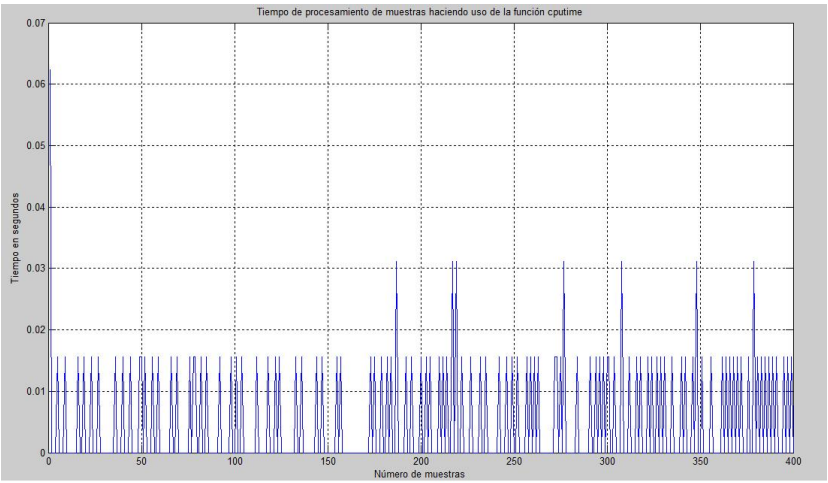

Figura 6.4: Resultados del tiempo empleado para procesar las muestras

para la primera muestra y el valor es de $0{,}0624s$.
La tabla 6.5 muestra los valores picos de tiempo que se encuentran al procesar cierta cantidad de muestras.

MUESTRAS	**TIEMPO (s)**
1	0,0624
5	0,0156
187	0,0312

Tabla 6.5: Valores pico de tiempo al procesar muestras con *cputime*

De estos resultados se pueden sacar dos conclusiones importantes:

- La primera es que el tiempo empleado para procesar una sola muestra es en promedio de $0{,}055$ segundos, tomado directamente de la observación del procesamiento de muestras.
- La segunda es que el valor del procesamiento de una sola muestra se puede calcular del valor al procesar 400 muestras, siendo este valor de$0{,}00002$ segundos es decir 20 μs.

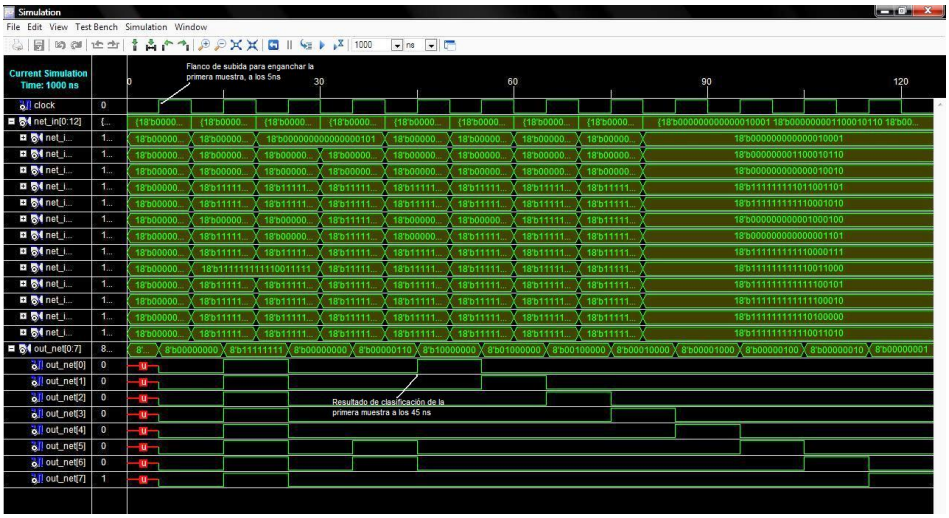

Figura 6.5: Resultados de la red al presentarle las 8 palabras

6.3. Resultados del clasificador en VHDL

Con la idea de realizar la implementación de la red neuronal en VHDL es que se realizó el desarrollo de este trabajo de grado, y es así como se llegó a un diseño final óptimo para su implementación en una FPGA.

El código del clasificador se anexa junto con este trabajo de grado y los resultados obtenidos en cuanto a acierto y velocidad se muestran a continuación, se tiene en cuenta que se trabajó con un reloj de 100 MHz para la red.

6.3.1. Resultados prueba de acierto en VHDL

La primera prueba a la que se enfrentó el diseño del clasificador, (en este caso la red neuronal Red13), fue la de acierto. Para esta tarea, se le ingresaron a la red diferentes muestras de la base de datos para su clasificación. Las muestras fueron tomadas aleatoriamente para no hacer pruebas con muestras de algún grupo determinado (ya sean de entrenamiento o de validación).

La figura 6.5 muestra los resultados del clasificador en VHDL, en donde se han introducido las 8 palabras de alguna de las muestras de entrenamiento y en donde se pueden evidenciar tanto los tiempos de enganche de la primera muestra como los tiempos de salida de de las

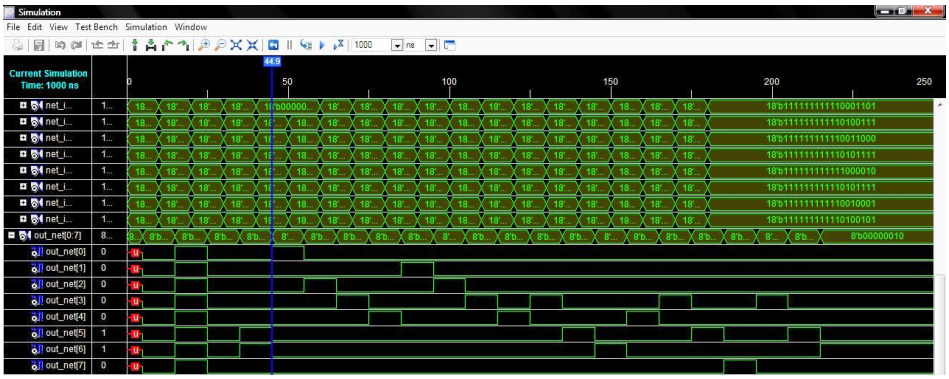

Figura 6.6: Resultados de la red al presentarle muestras aleatorias

muestras subsiguientes.

El acierto que presenta la red neuronal en VHDL es el esperado para un diseño con formato reducido de bits en la parte decimal (BF) y al igual que se había calculado en la red visible de Matlab, los resultados reales se aproximan mucho a los esperados (tabla 6.6).

RED	**TEST CT**		**TEST OT**	
	ACIERTOS	%	ACIERTOS	%
Red13 con BF 8 bits	291	90,9375	46	58

Tabla 6.6: Resultados de la red en VHDL

La siguiente etapa de prueba fue ingresarle a la red una serie de muestras aleatorias (incluidas de entrenamiento y validación) para realizar su clasificación. De nuevo se visualizan tiempos de enganche de muestras y las salidas son secuenciales (valga el juego de palabras) en un diseño paralelo. Como se observa en las figuras 6.5 y 6.6, los resultados de clasificación son entregados en cada ciclo de reloj, lo que muestra en esencia, la naturaleza paralela del diseño de redes neuronales, las neuronas procesan información siempre en cada ciclo de reloj.

6.3.2. Resultados prueba de velocidad en VHDL

La idea con esta prueba es poder comparar el rendimiento en velocidad entre el diseño secuencial del procesador del propósito general y el diseño paralelo que se realizó en VHDL.

En la figura 6.7 se puede observar el ingreso y salida de datos en función de los ciclos

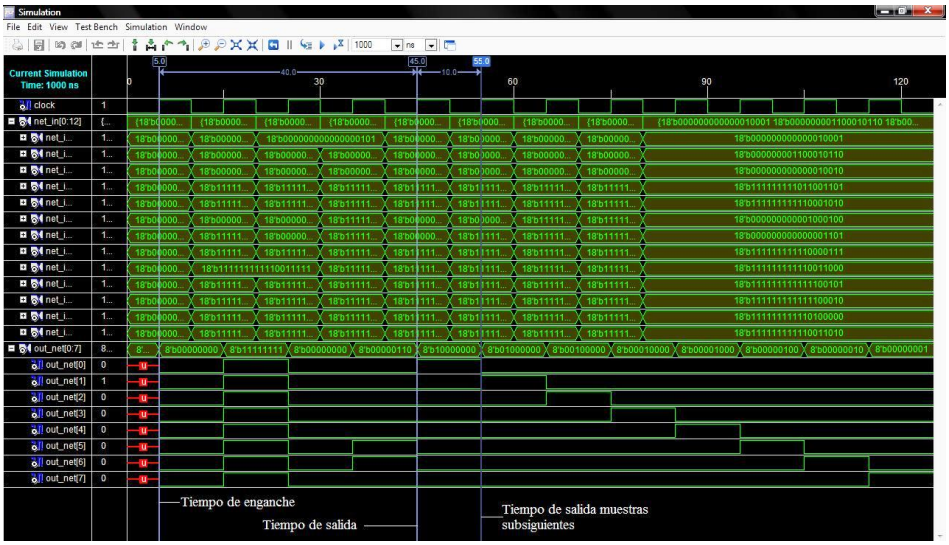

Figura 6.7: Tiempos de procesamiento de la red

de reloj. En ésta se pueden observar tres características temporales importantes: El tiempo de enganche de la primera muestra, el tiempo de salida de esa muestra y los tiempos de salida de las muestras subsiguientes.

Tiempo de enganche Es el tiempo en el que realmente los datos ingresan a la red, ya que el diseño y la segmentación del clasificador toma los datos con flancos de subida del reloj, permitiendo trabajar a los diferentes niveles de lógica simultáneamente (5 ns).

Tiempo de salida de la primera muestra Es el tiempo total que tarda en propagarse por todos los niveles de lógica la señal de la primera muestra (45 ns).

Tiempo de salida de muestras subsiguientes Aprovechando el diseño segmentado de la red neuronal, los datos se procesan paralelamente, permitiendo así que los resultados de las clasificaciones siguientes no se tomen el mismo tiempo de la muestra inicial, sino que exista una diferencia de sólo un ciclo de reloj entre resultados de las muestras subsiguientes. (10 ns).

La idea de comparación de tiempos entre los diseños hechos en diferentes plataformas de procesamiento, no es más que probar el alto uso y aprovechamiento de la capacidad de paralelismo de las FPGA y sus diseños en VHDL

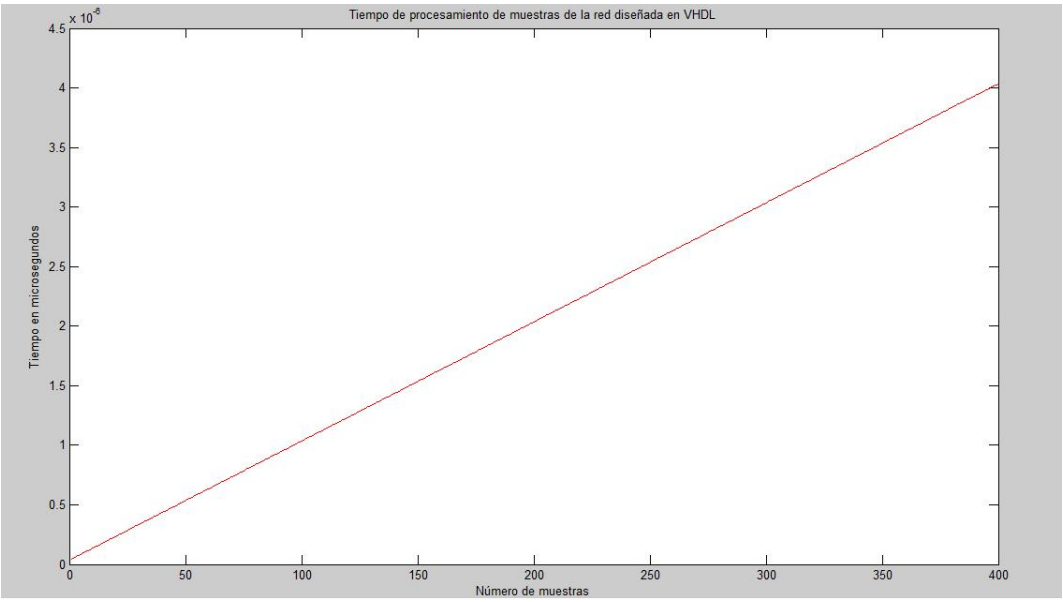

Figura 6.8: Tiempos empleados para procesar paquetes de palabras en VHDL

La figura 6.8, muestra los tiempos que emplea el diseño en VHDL apto para la implementación en FPGA, en procesar paquetes de palabras. Para la calificación de la velocidad de procesamiento se utilizaron los mismos criterios de evaluación de velocidad que en Matlab.

En este punto se puede concluir que para el procesamiento de paquetes de 400 muestras, el procesador de propósito general tarda $6999{,}99 \approx 7000$ μs en procesar los datos, mientras que el diseño en VHDL tarda sólo 4 μs en realizar la misma tarea, lo que en definitiva es un diseño 1733 veces más rápido y con un nivel de acierto casi igual al del procesador de propósito general.

6.4. Conclusiones y trabajo futuro

De los resultados anteriores se puede concluir que:

- El desarrollo del sistema de preprocesamiento y caracterización, basado en los algoritmos planteados en [21] y [11], se implementó de tal forma que permite de manera eficiente la localización y posterior extracción de la información inherente a la palabra

en la señal grabada además de la extracción de las características MFCC necesarias para la posterior etapa de clasificación.

- El uso de las redes neuronales como clasificador, permite un mejor desempeño en nivel de cierto que un clasificador bayesiano lineal. Debido a que las redes neuronales permiten desarrollar espacios de clasificación más complejos y sobre todo en muestras linealmente no separables.

PARÁMETRO	**ANN**		**BAYESIANO**	
	CT(%)	OT(%)	CT(%)	OT(%)
MFCC 13	94,0625	55	87,34	41,366

Tabla 6.7: Resultados de los clasificadores

- Una apropiada selección del formato de representación binaria de los datos en VHDL, permite el uso eficiente de la lógica de la FPGA. Esto debido a que representaciones en punto flotante, hacen inviable la implementación de redes neuronales con topologías elevadas.

- La estructura paralela de la red neuronal, aprovechada en el diseño de este trabajo de grado, a pesar del consumo elevado de lógica, permite la aceleración de la algoritmia en comparación con diseños secuenciales expuestos con anterioridad en trabajos como en [1]

- La utilización de *Very High Speed Integrated Circuit Hardware Description Language* (VHDL), para el correcto diseño de un clasificador con arquitectura de red neuronal ofrece las ventajas de un diseño en hardware: computación paralela de información y operación a altas velocidades.

- Los experimentos desarrollados mostraron que el sistema está diseñado para que su implementación obtenga un alto rendimiento en las tareas de reconocimiento de patrones.

- El diseño en *Hardware* de la ANN, permite que la aceleración del algoritmo sea de tal magnitud que los datos se procesan alrededor de 1700 veces más rápido que en un procesador de propósito general.

- Se ha comprobado, que el diseño final en VHDL de la red neuronal es fácilmente reconfigurable para ser usado en cualquier otra aplicación.

Algunos puntos a los que a mediano plazo se podría ampliar este trabajo de grado son los siguientes:

- Implementar el diseño del clasificador para el reconocimiento de patologías asociadas al tracto vocal basado en algoritmos de caracterización desarrollados por el *Grupo de Instrumentación y Control* del programa de Ingeniería Eléctrica en la Universidad Tecnológica de Pereira.
- Desarrollar por parte del *Laboratorio Sirius HPC* algoritmos basados en redes neuronales para la aplicación de la red diseñada en VHDL y su posterior implementación en dispositivos de lógica reconfigurable.
- Aprovechar la alta velocidad de procesamiento y la computación paralela para implementar clasificadores de voz continua, siendo esta la meta primordial de todos los trabajos de reconocimiento de voz.

Bibliografía

[1] Fredy Segura y Antonio García Alvaro Varela, Johann Osma. Neurona digital optimizada: Modelamiento, implementación y validación, 2002.

[2] Peter J. Asheden. *The VHDL Cookbook*. University of Adelaide. South Australia, 1990.

[3] G. Cybenko. Approximation by superpositions of a sigmoid function. In *Mathematics of Control, Signals, and Systems, vol. 2*, pages 303–314, 1989.

[4] Min Xu et al. Hmm-based audio keyword generation. *Advances in Multimedia Information Processing - PCM 2004: 5th Pacific Rim Conference on Multimedia*, 2004.

[5] Martin Hasler Jean Hennebert and Hervé Dedieu. Neural networks in speech recognition, 2006.

[6] Pedro Gómez Vilda Jesús Bernal Bermúdez, Jesús Bobadilla Sancho. Reconocimiento de voz y fonética acústica, 2003.

[7] M. Stinchcombe K. Hornik and H. White. Multilayer feedforward networks are universal approximators. In *Neural Networks 2*, pages 359–366.

[8] W. Sung K.-I. Kum, J. Kang. A floating-point to fixed-point c converter for fixed-point digital signal processors. In *Second SUIF Compiler Workshop*, 2004.

[9] W. Sung K.-I. Kum, J. Kang. A floating-point to integer c converter with shift reduction for fixed-point digital signal processors. In *Proceedings of the International Conference on Acoustics, Speech and Signal Procesing*, pages 2163–2166, March 1999.

[10] Trent W. Lewis and David M. W Powers. *Spoken Language Processing - A Guide to Theory, Algorithm and System Development*. Prentice Hall, 2001.

[11] M.A. Álvarez López. Reconocimiento de voz sobre diccionarios reducidos usando modelos ocultos de markov. Master's thesis, Universidad Nacional de Colombia sede Manizales, 2004.

[12] Camilo A. Zuluaga M. Maria Isabel Acosta B, Harold Salazar I. *Tutorial de Redes Neuronales*. Universidad Tecnológica de Pereira, 2000.

[13] A Mellouk and P Gallinari. A discriminative neural prediction system for speech rcognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 1993.

[14] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, 1969.

[15] S.K. Mitra. *Digital Signal Processing: A computer-Based Approach*. McGraw-Hill, 2nd ed edition, 2001.

[16] C. F. Ojeda. *Extracción de características usando transformada wavelet en la identificación de voces patológicas*. 2003.

[17] J. Picone. Signal modeling techniques in speech recognition, 1993.

[18] P. E. Hart R. O. Duda and D. G. Stork. *Pattern Classification*. John Wiley and Sons, Inc, 2001.

[19] L. Rabiner and B. Juang. *Fundamentals of Speech Recognition*. 1993.

[20] L. Rabiner and R. W. Schafer. *Digital Processing of Speech Signals*. Prentice Hall, 1978.

[21] L. R. Rabiner and M. R. Sambur. An algorithm for determining the endpoints of isolated utterances. *The Bell System Technical Journal.*, 54:297–315, 1975.

[22] R. Rao and R. Mersereau. Lip modeling for visual speech recognition. *28th. Annual Asimolar Conference on Signals, Systems and Computers*, 2, 1994.

[23] R. W. Schafer and L. R. Rabiner. Parametric representations of speech. In *Proc. IEEE Speech Recognition Symposium*.

[24] John Volkman Stanley Smith Stevens and Edwin Newman. A, scale for the measurement of the psychological magnitude of pitch. *Journal of the Acoustical Society of America*, 8:185–190, 1937.

[25] et al Steinbiss, V. The philips research system for large-vocabulary continuos-speech recognition. In *Proc. of the European Conf. on Speech Communication and Technology*, pages 2125–2128.

[26] Joe Tebelskis. Speech recognition using neural networks. Master's thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1995.

[27] J. F. Vargas. Selección de características en el análisis acústico de voces, 2003.

[28] Acero X. Huang, A and H. W. Hon. *Spoken Language Processing*. Prentice Hall, 2001.

[29] C. García y D. Tapias. La frecuencia fundamental de la voz y sus efectos en reconocimiento de habla continua, year =.

[30] M.L. López Vallejo y J. L. Ayala Rodrigo. Fpga: Nociones básicas e implementación. *Laboratorio de Diseño Microelectrónico, 4° Curso, P94*, 2004.

Anexo A

Palabras que componen el diccionario reducido

A continuación se exponen las palabras que componen el diccionario reducido de voz.

Baile

Bola

Coco

Choza

Gato

Gol

Jugo

Mano

Anexo B

Nociones básicas del punto flotante

El estándar de la IEEE para aritmética en coma flotante (IEEE 754) es el estándar más extendido para las computaciones en punto flotante, y es seguido por muchas de las mejoras de CPU y FPU. El estándar define formatos para la representación de números en punto flotante (incluyendo el cero) y valores desnormalizados, así como valores especiales como infinito y NaN, con un conjunto de operaciones en punto flotante que trabaja sobre estos valores. También especifica cuatro modos de redondeo y cinco excepciones (incluyendo cuando dichas excepciones ocurren, y que sucede en dichos momentos).

IEEE 754 especifica cuatro formatos para la representación de valores en punto flotante: precisión simple (32 bits), precisión doble (64 bits), precisión simple extendida (≥ 43 bits, no usada normalmente) y precisión doble extendida (≥ 79 bits, usualmente implementada con 80 bits). Sólo los valores de 32 bits son requeridos por el estándar, los otros son opcionales. Muchos lenguajes especifican qué formatos y aritmética de la IEEE implementan, a pesar de que a veces son opcionales. Por ejemplo, el lenguaje de programación C, ahora permite pero no requiere la aritmética de la IEEE (el tipo de C float es típicamente usado para la precisión simple de la IEEE y el tipo double usa la precisión doble del la IEEE).

El título completo del estándar es **IEEE Standard for Binary Floating-Point Arithmetic** (ANSI/IEEE Std 754-1985), y también es conocido por IEC 60559:1989, **Binary floating-point arithmetic for microprocessor systems**.

El formato seleccionado para realizar el trabajo inicial en la neurona en punto flotante es el de 32 bits, es decir, los números se expresaran en formato punto flotante de precisión simple.

donde S es el bit de signo y Exp es el campo exponente. (Para el signo: 0=Positivo ; 1= Nega-

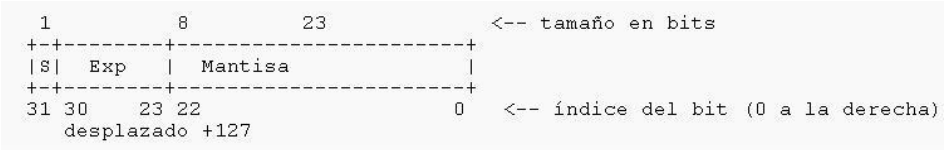

Figura B.1: Formato punto flotante precisión simple

tivo). El exponente es desplazado en un número en precisión simple, un exponente en el rango -126 a +127 es desplazado mediante la suma de 127 al valor para obtener un número en el rango 1 a 254 (0 y 255 tienen valores especiales descritos más adelante). Cuando se interpreta el valor en punto flotante, el número es desplazado de nuevo para obtener el exponente real.

El conjunto de valores posibles pueden ser divididos en los siguientes:

- Ceros
- Números normalizados
- Números desnormalizados
- Infinitos
- NaN

En la figura B.2 se distinguen las clases que pueden tomar los valores expresados en este formato. Se diferencian principalmente por el valor del campo Exp, siendo modificada ésta por el campo fracción. Considera Exp y Fracción como campos de números binarios sin signo (Exp se encuentra en el rango 0 a 255). Para números normalizados, los más comunes, Exp es el exponente desplazado y Fracción es la parte fraccional de la mantisa (o significando). El número tiene valor v y se expresa en la ecuación B.1

$$v = s \times 2^e \times m \tag{B.1}$$

Donde
S=$+1$(números positivos) cuando S es 0
S=-1(números negativos) cuando S es 1
e=Exp - 127 (en otras palabras, el exponente se almacena con 127 sumado a él, también llamado “biased with 127” en inglés)
m=1. Fracción en binario (esto es, el significando es el número binario 1 seguido por el punto decimal seguido por los bits de Fracción). Por lo tanto, $1 \leq m < 2$.

Clase	Exp	Fracción
Ceros	0	0
Números desnormalizados	0	distinto de 0
Números normalizados	1-254	cualquiera
Infinitos	255	0
NaN (Not a Number)	255	distinto de 0

Figura B.2: Clases de valores expresados en punto flotante

Para tener en cuenta ciertos conceptos de este formato, se enumeran los principales aspectos para no olvidar:

1. Los números desnormalizados son iguales excepto que $e = -126$ y $m = 0$.Fracción. (e NO es -127 : el significando ha de ser desplazado a la derecha por un bit más, de forma que incluya el bit principal, que no siempre es 1 en este caso. Esto se balancea incrementando el exponente a -126 para el cálculo.)

2. -126 es el menor exponente para un número desnormalizado.

3. Hay dos ceros. $+0$ (S es 0) y -0 (S es 1).

4. Hay dos infinitos $+\infty$ (S es 0) y $-\infty$ (S es 1).

5. Los NaN pueden tener un signo y un significando, pero estos no tienen otro significado que el que puedan aportar en pruebas de diagnóstico; el primer bit del significando es a menudo utilizado para distinguir *NaNs señalizados* de *NaNs silenciosos*

6. los NaNs y los infinitos tienen todos los bits a 1 en el campo Exp.

Se realizará un ejemplo sencillo para ilustrar la conversión de un numero real al formato binario punto flotante precisión simple(32 bits):

Se codificará el número decimal -225.625 usando el sistema de la IEEE 754.

Se necesita obtener el signo, el exponente y la fracción.Dado que es un número negativo, el signo es “1”. Ahora los demás valores:

Primero, se escribe el número el número (sin signo) usando notación binaria. El resultado es $11100001{,}101$.

Ahora, movamos el punto decimal a la izquierda, dejando sólo un 1 a su izquierda.

$11100001{,}101 = 1{,}1100001101 \times 2^7$. Esto es un número en punto flotante normalizado.

La mantisa es la parte a la derecha del punto decimal, rellenada con ceros a la derecha hasta que obtengamos todos los 23 bits. Es decir
11000011010000000000000.

El exponente es 7, pero necesitamos convertirlo a binario y desplazarlo (de forma que el exponente más negativo es 0, y todos los exponentes son solamente números binarios no negativos). Para el formato IEEE 754 de 32 bits, el desplazamiento es 127, así es que $7 + 127 = 134$. En binario, esto se escribe como 10000110.
Poniendo todo junto se tiene:

$$-225{,}625 = 11000011011000011010000000000000$$

Anexo C

Representación de datos en punto fijo

C.1. Representación en punto fijo

Para la construcción precisa de un algoritmo, los valores de los datos y coeficientes usados deben estar en representación de punto flotante precisión simple o doble (32 y 64 bits respectivamente). Sin embargo, como se explicó anteriormente, existe un importante uso de procesador para realizar cálculos basados en punto flotante, resultado de la falta de *Hardware* basado en dicha representación. En muchos casos como el de procesadores embebidos de bajo poder no existe ni siquiera compiladores que soporten números en punto flotante precisión doble. El punto flotante impone límites para la tasa efectiva de iteraciones de un algoritmo.

Para incrementar la tasa de ejecución de algoritmos o la eficiencia matemática del mismo, los cálculos pueden realizarse usando representaciones en *Complemento a dos signadas en punto fijo*. Este tipo de representación requiere que el programador cree un lugar decimal virtual entre dos bits, para un tamaño de dato dado.
Para propósito de este trabajo de grado se notará por convención los números en punto fijo de la siguiente manera:

$$B[BI][BF] \tag{C.1}$$

en donde: B = Bit de Signo
BI = Número de bits en la parte entera
BF = Número de bits en la parte decimal

El bit de signo (B) más número de bits en la parte entera (BI) más el número de bits en la parte decimal, forman el número total de bits usados para representar un número. La suma $B + BI + BF$ es conocida como tamaño de palabra (WL por sus siglas en inglés **Word Length**).

C.1.1. Rango de la parte entera del punto fijo

Para representar un número en punto flotante en formato punto fijo, se debe de ver como dos distintas partes, el contenido entero y el contenido decimal. El rango de la parte entera de una variable en punto flotante, (por ejemplo su rango de mínimo a máximo), en un algoritmo determina el número de bits (BI) requerido para representar la parte entera de un número. Teniendo en cuenta que BI solamente puede tener valores enteros debido a la naturaleza binaria de los bits (existe o no existe).

Existen dos métodos diferentes [9] para el cálculo de el número de bits requeridos para la parte entera (BI) para cada tipo de número, signados y no signados.

Rango de punto fijo para enteros no signados

La relación para números positivos está definido por el mínimo y el máximo de cualquier número de BI-bits, como se muestra en la ecuación C.2.

$$0 \leq \alpha \leq \left(2^{BI} - 1\right) \tag{C.2}$$

Método 1 Resolviendo la ecuación C.2 para el número de bits BI requeridos:

$$\begin{aligned} BI &\geq ceiling\left(\log_2\left(\alpha + 1\right)\right) \\ BI &= ceiling\left(\log_2\left(\alpha + 1\right)\right) \end{aligned} \tag{C.3}$$

En donde α es la variable (número real o en punto flotante) y *ceiling* redondea alrededor de $+\infty$. Nótese que el valor de $\log_2()$ en la ecuación C.3 nunca puede ser negativo, lo que implica que BI siempre es ≥ 1 [8].

Por ejemplo, un número positivo dado, $\alpha = 5{,}4321$, se puede determinar en los siguientes números de bits

$$BI = ceiling\left(\log_2\left(5{,}4321 + 1\right)\right) = ceiling(2{,}6835) = 3$$

Por tanto, se requieren 3 bits para la parte entera de α.

Método 2 Aunque el método anterior es una posible solución de calcular BI para valores no signados, existe otra posible solución igual de buena, especialmente cuando se trabaja con números que pueden ser menores que $|1$ que deben ser implementados en variables tipo estándar.

Tomando de nuevo la desigualdad inicial de la ecuación C.2, la desigualdad puede ser reescrita de la siguiente manera:

$$0 \leq \alpha < 2^{BI} \tag{C.4}$$

Nótese que el límite superior cambia de $\leq$ a $<$ y el valor del límite cambia a un valor superior que el máximo BI-bit del número no signado.

Resolviendo la ecuación C.4 para BI, se tiene

$$\begin{aligned} \alpha &< 2^{BI} \\ \log_2(\alpha) &< BI \\ BI &> (\log_2(\alpha)) \end{aligned} \tag{C.5}$$

Conociendo que BI es un número de bits enteros, se puede crear una ecuación que compute un BI que satisfaga la ecuación C.5 añadiendo 1 y truncando el resultado (alrededor de $-\infty$). Una ecuación para un número entero requerido de bits se puede generalizar para este método de la siguiente manera:

$$BI = floor((\log_2(\alpha) + 1)) \tag{C.6}$$

Rango de punto fijo para enteros signados

Si se necesita representar variables signadas, las soluciones previas para BI cambian debido a que los límites del rango de los valores signados cambian. Esta relación para el contenido de valores enteros signados $(\pm\alpha)$ es definido por el valor mínimo y máximo que un número signado de BI-bits puede manejar. Este número se muestra en la ecuación C.7

$$\left(-2^{BI-1}\right) \leq \alpha \leq \left(2^{BI-1} - 1\right) \tag{C.7}$$

Cabe recordar que existe una asimetría alrededor de cero para variables enteras signadas (por ejemplo, un número de 8 bits signado varía entre -128 y $+127$. Esta asimetría conlleva dos posibles métodos para calcular el número de bits [9].

Método 3 Resolviendo BI para la restricción negativa de la ecuación C.7 (ejemplo cuando α es negativo).

$$\begin{aligned} \left(-2^{BI-1}\right) &\leq \alpha \\ 2^{BI-1} &\geq -\alpha \\ BI - 1 &\geq \log_2(-\alpha) \\ BI &\geq \log_2(-\alpha) + 1 \end{aligned} \tag{C.8}$$

Resolviendo para BI para la restricción positiva de la ecuación C.7 (ejemplo cuando α es positivo):

$$\begin{aligned} \alpha &\leq \left(2^{BI-1}-1\right) \\ \alpha+1 &\leq 2^{BI-1} \\ \log_2(\alpha+1) &\leq BI-1 \\ BI &\geq (\log_2(\alpha+1)+1 \end{aligned} \tag{C.9}$$

Por ejemplo si:

$$\begin{gathered} \alpha_{min}=-2, \alpha_{max}=2 \\ BI|_{\alpha_{min}} \geq \log_2(-\alpha_{min})+1=\log_2(2)+1=2 \\ BI|_{\alpha_{max}} \geq \log_2(\alpha_{max}+1)+1=\log_2(3)+1=2{,}5850 \end{gathered}$$

La restricción positiva es la más estrecha de las dos restricciones, debido a la asimetría de los enteros signados alrededor de cero. No es poco común que los programadores definan magnitud de variables que son asimétricas alrededor de cero. (por ejemplo: $-3 \leq \alpha \leq 3$). Los cálculos para el número requerido de bits enteros puede ser generalizado por este método:

$$BI = ceiling\left(\log_2\left(\text{máx}\left(abs\left[\alpha_{max}, \alpha_{min}\right]\right)+1\right)\right)+1 \tag{C.10}$$

En donde α es la variable real a la que se halla el rango y *ceiling* redondea alrededor de $+\infty$.

Un ejemplo para el cálculo de BI de una variable signada se expone a continuación con: $-5{,}4321 \leq \alpha \leq 5{,}4321$

$$\begin{gathered} BI = ceiling\left(\log_2\left(\text{máx}\left(abs\left[-5{,}4321, 5{,}4321\right]\right)+1\right)\right)+1 \\ BI = ceiling\left(\log_2(6{,}4321)\right)+1 = ceiling(2{,}6853)+1 = 3+1 \\ BI = 4 \end{gathered}$$

Método 4 Aunque el método 3 es una forma viable de calcular el valor BI para valores signados, existe de nuevo otra vía de cálculo que es igual de buena, especialmente cuando se trata de lidiar con números que pueden ser más pequeños que $|1|$ y que deben ser implementados en variables estándar.
Tomando los límites iniciales de la ecuación C.7

$$\left(-2^{BI-1}\right) \leq \alpha \leq \left(2^{BI-1}-1\right)$$

La desigualdad puede ser escrita de la siguiente forma:

$$-2^{BI-1} \leq \alpha < 2^{BI-1} \tag{C.11}$$

Resolviendo la inecuación C.11 para BI en la restricción negativa (es decir cuando α es negativo):

$$\begin{gathered} \left(-2^{BI-1}\right) \leq \alpha \\ 2^{BI-1} \geq -\alpha \\ BI - 1 \geq \log_2(-\alpha) \\ BI \geq \log_2(-\alpha) + 1 \end{gathered}$$

Resolviendo ahora la inecuación C.11 para BI en la restricción positiva (es decir cuando α es positivo):

$$\begin{gathered} \alpha < 2^{BI-1} \\ \log_2(\alpha) < BI - 1 \\ BI > (\log_2(\alpha)) + 1 \end{gathered}$$

La restricción para el número requerido de bits en la parte entera puede generalizarse de la siguiente manera:

$$BI > \log_2\left(\text{máx}\left(abs\left[\alpha_{max}, \alpha_{min}\right]\right)\right) + 1 \tag{C.12}$$

Conociendo que BI es un número entero de cierto número de bits se puede crear una ecuación para calcular BI que satisfaga la restricción en la ecuación C.5, simplemente añadiendo 1 y truncando el resultado (redondeando hacia $-\infty$). La ecuación quedaría finalmente:

$$BI = floor\left(\log_2\left(\text{máx}\left(abs\left[\alpha_{max}, \alpha_{min}\right]\right)\right) + 2\right) \tag{C.13}$$

C.1.2. Resolución en punto fijo para la parte decimal

La resolución para una variable en punto fijo está determinado por el número de bits en la parte decimal (BF) usados en la variable en punto fijo. La resolución ε, de un número en punto fijo está dado por la ecuación [8]:

$$\varepsilon = \frac{1}{2^{BF}} \tag{C.14}$$

Por ello, el número de bits en la parte decimal requeridos para una resolución particular están definidos por:

$$BF = \log_2\left(\frac{1}{\varepsilon}\right) \tag{C.15}$$

Sin embargo, dado que BF sólo posee valores enteros (ya que solo se puede usar un número entero de bits), se usa la aproximación por encima (*ceiling*) del logaritmo, es decir:

$$BF = ceiling\left(\log_2\left(\frac{1}{\varepsilon}\right)\right) \tag{C.16}$$

Por ejemplo una variable signada $\alpha = -5{,}4321, \varepsilon \leq 0{,}0001$

$$BF = ceiling\left(\log_2\left(\frac{1}{0{,}0001}\right)\right)$$
$$BF = ceiling\left(\log_2(10000)\right) = ceiling(13{,}288) = 14$$

No es raro que los programadores encuentren el número de bits en la parte entera requeridos (BI) y queden con la resolución dada por el número de bits restantes para una longitud de palabra dada (WL) en una variable. Si se requiere de una mayor resolución, la variable binaria o la longitud de palabra debe de ser aumentada para brindar la resolución esperada [8].

C.1.3. Escalamiento de un número real a punto fijo

Una vez que se ha calculado un apropiado formato en punto fijo basado en la longitud de palabra (WL), el rango de la parte entera y la resolución del valor real, la aproximación en punto fijo de dicho valor se puede ahora calcular. Esta relación está gobernada por la ecuación:

$$FxdPt = \left(real \times 2^{BF}\right)|_{\text{alrededor de 0}} \tag{C.17}$$

Dado que la representación en punto fijo de un número real sólo puede tener valores enteros, se debe de hacer uso del truncamiento del valor real escalado [8]. Esto significa, redondear alrededor de 0, por ejemplo $-1{,}4$ se vuelve -1 y $1{,}4$ se vuelve 1.

Del ejemplo anterior, una variable signada $\alpha = -5{,}4321$, $\varepsilon \leq 0{,}0001$, BF$= 14$. La representación en punto fijo para la variable α es:

$$\alpha_{FxdPt} = \left(\alpha_{real} \times 2^{BF_\alpha}\right)|_{\text{alrededor de 0}} = \left(-5{,}4321 \times 2^{14}\right)|_{\text{alrededor de 0}}$$
$$\alpha_{FxdPt} = \left(-5{,}4321 \times 16384\right)|_{\text{alrededor de 0}} = (-88999{,}5264)|_{\text{alrededor de 0}}$$
$$\alpha_{FxdPt} = -88999$$

En este caso el valor de α_{FxdPt} entrega el valor entero del valor real para ser convertido a binario en formato punto fijo. Realizando la conversión en formato $[1][3][14]$ es decir 1 bit de signo (**B**), 3 bits para la parte entera (**BI**) y 14 bits para la parte decimal (**BF**), se tiene un dato con longitud de palabra (**WL**) [18] bits.

$$\alpha_{\text{punto fijo}} = 110101101110100111$$

Anexo D

Coeficientes Cepstrum en la escala de frecuencia Mel

D.1. Cepstrum

Un *Cepstrum* es el resultado de aplicar la transformada de Fourier (FT) del espectro en decibelios como si está fuera una señal. Su nombre proviene de cambiar el orden de las cuatro primeras letras de la palabra *"spectrum"*.

El *Cepstrum* fue definido en 1963, por Bogert, Healey & Tukey, como:

- **Verbalmente**: El *cepstrum* de una señal es la transformada de Fourier del logaritmo (sin fase envuelta) de la transformada de Fourier de una señal. Algunas veces llamado el espectro de un espectro.
- **Matemáticamente**: El *cepstrum* de una señal se define en la ecuación D.1 como:

$$Cps = FT\left(\log\left(|FT(f(t)|\right) + 2j\pi m\right) \tag{D.1}$$

  Donde m es el entero requerido para envolver apropiadamente el ángulo de la parte imaginaria de la función logarítmica compleja.

El *cepstrum* real usa la función logarítmica definida para valores reales. El *cepstrum* complejo usa la función compleja logarítmica definida para los valores complejos. El *cepstrum* complejo posee información acerca de la magnitud y la fase del espectro inicial, permitiendo la reconstrucción de la señal. El *cepstrum* real usa sólo información de la magnitud del espectro.

El *cepstrum* es usado como un excelente vector de características para la representación de

la voz humana y las señales musicales. Para estas aplicaciones, el espectro es transformado inicialmente usando la escala Mel. El resultado es llamado *Cepstrum* en la escala Mel o MFC por sus siglas en inglés (los coeficientes resultantes son llamados Coeficientes cepstrales en la frecuencia Mel, o MFCC). Son ampliamente usados para la identificación de voz, detección del pitch y otros.

D.2. Escala mel

La escala mel, propuesta por Stevens, Volkman y Newman [24] en 1937 es una escala perceptual de pitches (o frecuencia fundamental audible de un sonido) juzgado por oyentes para ser igual en distancia uno de otro. El punto de referencia entre esta escala y la medida de la frecuencia normal es definida equiparando a un tono de 1000 Hz, 40 dB por encima del límite audible, con un pitch de 1000 mels. Por encima de alrededor de 500 Hz, intervalos más y más grandes son juzgados para producir igual incrementos del pitch. Como resultado, cuatro octavas en la escala de Hertz por encima de 500 Hz son juzgados para comprender alrededor de dos octavas en la escala mel. El nombre mel proviene de la palabra melodía para indicar que la escala está basada en comparaciones de pitch.

El cambio entre escalas *mel* y *Hertz* está dada por la ecuación D.1

$$m = 1127{,}01048 \ln\left(1 + \frac{f}{700}\right) \tag{D.2}$$

y representado en la figura D.1

D.3. Cepstrum en la escala de frecuencia mel

En el procesamiento de las señales de audio, los **Cepstrum en la frecuencia mel (MFC)**, son una representación del espectro de potencia de tiempo corto de un sonido, basados en la transformada lineal del coseno de un espectro de potencia logarítmica en una escala de frecuencia no lineal, o escala mel, como se explicó anteriormente.

Los **Coeficientes Cepstrum en la escala de frecuencia mel** son coeficientes que colectivamente forman un MFC. Éstos son derivados desde un tipo de representación cepstral de un clip de audio (un “espectro de un espectro”). La diferencia entre los *cepstrum* y los *cepstrum* en la escala de frecuencia mel, es que en los MFC, las bandas de frecuencia son igualmente espaciados en la escala mel, los cuales aproximan de una manera más cercana la respuesta del sistema auditivo humano, a diferencia de las bandas de frecuencia linealmente espaciadas

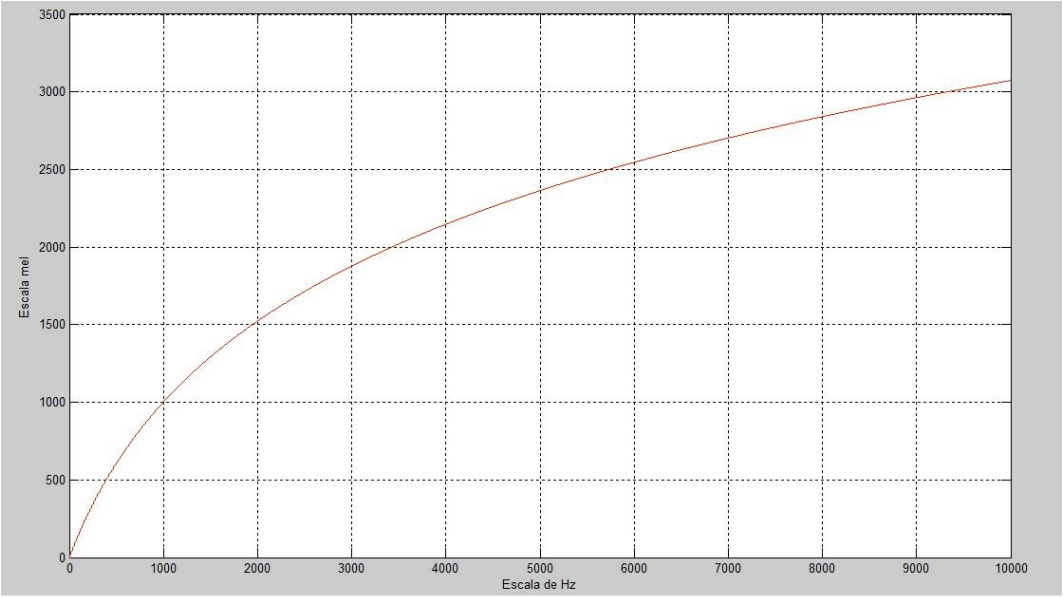

Figura D.1: Cambio de escala entre Hertz y mel

usadas en los *cepstrum* normales.

La forma más común de calcular los MFCC, es la siguiente [4]:

1. Calcular la transformada de Fourier de una señal ventaneada.
2. Mapear las potencias de los espectros obtenidos en la escala mel, usando ventanas triangulares traslapadas.
3. Tomar los logaritmos de cada una de las potencias en la frecuencia mel.
4. Calcular la transformada discreta del coseno de la lista de potencias logarítmicas en la frecuencia mel, como si fueran una señal.
5. Los Coeficientes *cepstrum* (MFCC) son las amplitudes del espectro resultante.