UNIVERSIDAD NACIONAL DE INGENIERIA

FACULTAD DE INGENIERIA ELECTRICA Y ELECTRONICA

"DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE RECONOCIMIENTO DE PALABRAS EN UN FPGA BASADO EN EL ALGORITMO DE LPC"

TESIS

PARA OPTAR AL GRADO DE MAESTRO EN CIENCIAS
MENCIÓN: AUTOMÁTICA E INSTRUMENTACIÓN

PRESENTADA POR
CÉSAR ALBERTO BRICEÑO ARANDA

LIMA, PERÚ
2012

Dedico este trabajo a la memoria de mis padres César e Hilda.

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

LISTA DE FIGURAS

ABREVIATURAS

ADC	*: Analog to Digital Converter (Convertidor Analógico a Digital)*
ANN	*: Artificial Neural Network (Red Neuronal Artificial)*
ASIC	*: Application Specific Integrated Circuit (Circuito Integrado de Aplicación Específica)*
DAC	*: Digital to Analog Converter (Convertidor Digital a Analógico)*
DSP	*: Digital Signal Proccessing (Procesamiento Digital de Señales)*
DTW	*: Dynamic Time Warping (Alineamiento Dinámico en el Tiempo)*
FPGA	*: Field Programmable Gate Array (Arreglo de compuertas programable en campo)*
HAL	*: Hardware Abstraction Layer (Capa de Abstracción de Hardware)*
IDE	*: Integrated Development Environment (Entorno Integrado de Desarrollo)*
JTAG	*: IEEE Joint Test Action Group*
LCD	*: Liquid Crystal Display (Visualizador de Cristal Líquido)*
LED	*: Light Emitting Diode (Diodo Emisor de Luz)*
LPC	*: Linear Predictive Coding (Codificación por Predicción Lineal)*
MATLAB	*: MATrix LABoratory*
PIO	*: Parallel Input-Output (Entrada-Salida Paralela)*
PLL	*: Phase Locked Loop (Lazo de Fase Enganchada)*
RISC	*: Reduced Instruction Set Computer (Computador de Juego Reducido de Instrucciones)*
SoPC	*: System on Programmable Chip (Sistema Programable en Circuito Integrado)*
SDRAM	*: Synchronous Dynamic Random Access Memory (Memoria Dinámica de Acceso Aleatorio Síncrona)*
SSRAM	*: Synchronous Static Random Access Memory (Memoria Estática de Acceso Aleatorio Síncrona)*
UART	*: Universal Asynchronus Receiver Transmitter (Receptor/Transmisor Universal Asíncrono)*
VHDL	*: Very High Speed Integrated Circuit Hardware Description Language (Lenguaje de Descripción de Hardware para Circuitos Integrados de Muy Alta Velocidad)*

RESUMEN

Hace varias décadas el ser humano ha intentado comunicarse con las máquinas de la manera más espontánea, mediante el habla. La dificultad para conseguir que las máquinas entiendan el habla humana es consecuencia de muchos aspectos como variabilidad en los tonos de voz, rapidez de pronunciación, presencia inminente del ruido, acentos regionales, deformación del lenguaje, etc.

Los sistemas más sofisticados de reconocimiento de voz se basan en algoritmos de computación implementados en software, que por tanto tienen limitaciones en la rapidez de procesamiento. Es así que se plantearon soluciones novedosas, como aquellas basadas en hardware, por medio de dispositivos lógicos programables como los FPGA (Field Programmable Gate Array), cuyo uso se está extendiendo en muchos aspectos de la vida cotidiana donde se requiere rapidez y gran capacidad de procesamiento, como es el caso de los sistemas de reconocimiento de voz humana.

El presente trabajo de tesis muestra el proceso de diseño e implementación de un sistema electrónico capaz de reconocer la voz humana mediante la interpretación del significado de varias palabras pronunciadas por un locutor en particular, en una solución implementada en hardware. El sistema aprovecha las características de rapidez de procesamiento de un dispositivo FPGA y la flexibilidad del lenguaje de descripción de hardware empleado para programar dicho dispositivo, así como la versatilidad del FPGA que nos permite generar circuitos digitales, incluso un procesador embebido, que es programado en C o C++, lo que permitió implementar algoritmos secuenciales de reconocimiento de patrones que están diseñados para su uso en sistemas computarizados. En el presente trabajo se usó un códec de audio para la adquisición de datos de voz, el algoritmo LPC para codificar la voz y una red neuronal tipo perceptrón multicapa para el reconocimiento de los patrones de voz codificados.

Palabras clave: Reconocimiento de voz, FPGA, Redes Neuronales Artificiales, Lenguaje C, LPC.

SUMMARY

For several decades, humans have tried to communicate with machines in the most spontaneous way, through speech. The difficulties in getting machines to understand human speech is the result of many aspects: variability in the tone of voice, speed of pronunciation, imminent presence of noise, regional accents, distortion of language, etc.

The most sophisticated speech recognition based on computer algorithms were implemented in software; therefore they are limited in processing speed. Then, new solutions arise, such as those based on hardware, using programmable logic devices such as FPGA (Field Programmable Gate Array), whose use is spreading in many aspects of daily life where you need speed and high capacity of processing, such as systems for human voice recognition.

This document shows the process of designing and implementing a digital system capable of recognizing the human voice by interpreting the meaning of several words of one speaker in particular, as a solution implemented in hardware. The system takes advantage of the processing speed feature of the FPGA device, and the flexibility of the hardware description language used to program the device for generating digital circuits including embedded processors, which are programmed in C or C++, language used to achieve the implementation of sequential pattern recognition algorithms that are designed for computer systems. The project used an audio codec for data voice acquisition, LPC algorithm for voice coding, and a multilayer perceptron artificial neural network for voice-coded pattern recognition.

Key words: Speech Recognition, FPGA, Artificial Neural Networks, C code, LPC.

CAPÍTULO 1
MARCO TEÓRICO

1.1. Introducción

Hoy en día la evolución del diseño electrónico sigue avanzando a pasos agigantados, pues para hacer prototipados de diseños que se realizan a gran escala, se usan dispositivos reconfigurables tales como los FPGAs, que para aplicaciones de un sinfín de exigencias presentan un buen desempeño. Los dispositivos FPGA vienen siendo muy superiores en algunos casos a dispositivos dedicados como los microcontroladores y DSPs, debido a que uno puede implementar la arquitectura de un microcontrolador o de un DSP a medida en ellos, y hardware dedicado para aceleración de procesamiento. Inclusive, se puede diseñar todo un sistema completo de procesador, en el que diseño incluye a un procesador embebido en software como el procesador Nios II de Altera, y que junto a las librería de propiedad intelectual (IP Libraries en inglés) propietaria del fabricante de dispositivos lógicos programables, pueden realizarse aplicaciones complejas con este procesador de propósito general, el cual es un procesdor del tipo RISC de 32 bits de tamaño de instrucción [5].

Desde un punto de vista más general de la clasificación de los procesadores en FPGA, se puede enumerar lo siguiente:

1. Procesadores definidos en hardware: Usan un núcleo de procesador embebido en silicio dedicado y ofrecen ventajas hibridas entre FPGA y ASIC.
2. Procesadores definidos en software: Usan elementos lógicos programables existentes en el FPGA para implementar la lógica del procesador.

La herramienta SoPC Builder de Altera es usada para diseñar un sistema completo basado en procesador en un FPGA usando el procesador definido por software Nios II.

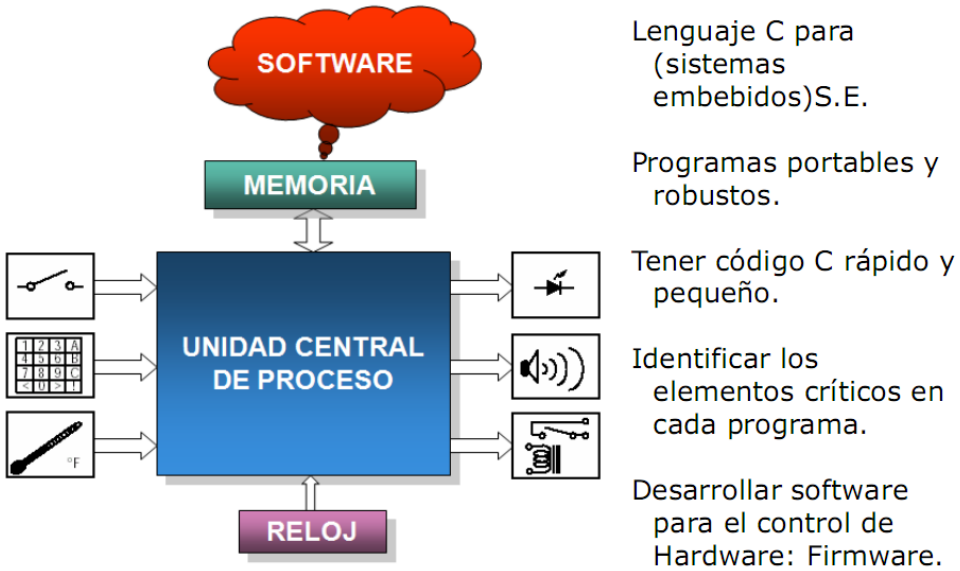


Figura 1.1. Esquema de un sistema basado en procesador.

Así también, existen procesadores embebidos (embedded systems), que es conocido como un circuito electrónico computarizado que está diseñado para cumplir una labor específica en un producto. Se dice sistema embebido porque este sistema está incrustado en un sistema más grande que cumple una tarea específica.

Con lo definido anteriormente se puede decir que el esquema de la Figura 1.1 forma parte de un sistema embebido, si este sistema forma parte de un sistema más grande que está cumpliendo una tarea específica.

Siendo que en este trabajo, se tienen componentes de hardware y software, a continuación, se realiza una descripción resumida de éstos.

1.2. Herramientas de Hardware

El hardware empleado consiste de 2 elementos principales: una PC convencional y una tarjeta de desarrollo DE2 de Altera [13]. Adicionalmente se usó un micrófono y un parlante.

En la Figura 1.2 se muestra una vista panorámica de la tarjeta de desarrollo para FPGA's en conjunto con los accesorios usados, tales como: micrófono, pulsador para iniciar la grabación, salidas de control, codec y pantalla LCD.

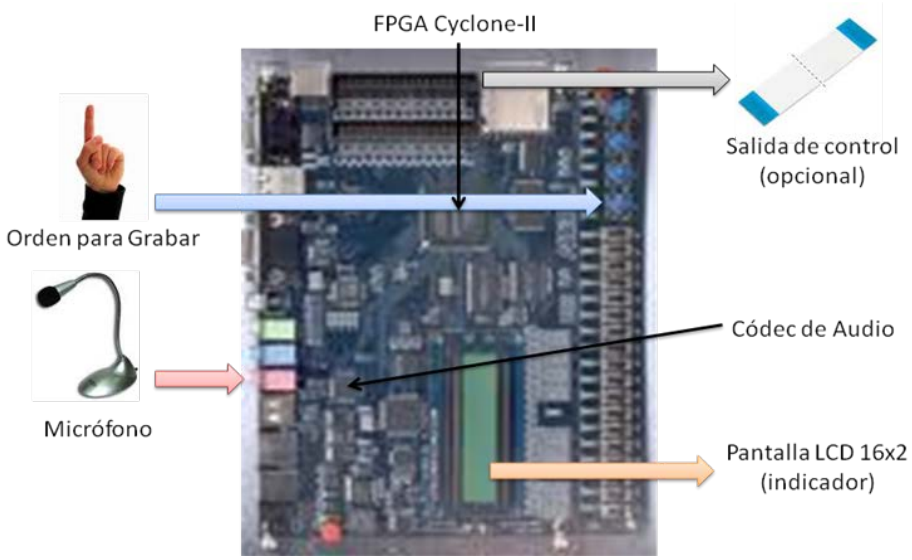


Figura 1.2. Vista panorámica de la tarjeta de desarrollo DE2-70 de Altera.

1.2.1. Tarjeta de desarrollo DE2-70 de Altera

Los elementos de la tarjeta de desarrollo DE2-70 de Altera que se usaron son los siguientes: FPGA Cyclone II EP2C70F896 de Altera, SSRAM de 2MBytes, SDRAM de 32MBytes, CODEC de audio WM8731 de Wolfson, pulsadores, interruptores, LEDs y pantalla LCD 16x2, como se ilustra en la Figura 1.3. Así también, los siguientes dispositivos se encuentran en la tarjeta de desarrollo:

- Dispositivo FPGA Cyclone II 2C70 de Altera.
- Dispositivo de configuración serial EPCS16 de Altera.
- Controlador USB Blaster para programación del FPGA y control API de usuario. Permite los modos de programación JTAG y AS (Active Serial).
- Memoria SSRAM de 2Mbytes.
- 2 memorias SDRAM de 32Mbytes.
- Memoria FLASH de 8Mbytes.
- Socket para memoria SD.
- 4 interruptores de tipo pulsador.
- 18 interruptores de tipo deslizamiento.
- 18 diodos LED de color rojo para el usuario.

- 9 diodos LED de color verde para el usuario.
- 2 fuentes de señal de reloj de 28.63MHz y 50MHz.
- CODEC de audio de 24 bits (calidad de CD) con conectores para micrófono, líneas de entrada y línea de salida.
- Conversor DAC triple para VGA, con puerto de salida VGA.
- 2 decodificadores de TV (NTSC/PAL/SECAM) y conector de TV de entrada.
- Controlador Ethernet de 10/100Mbps con su respectivo conector.
- Controlador USB Host/Slave con conectores USB tipo A y tipo B.
- Transceptor RS-232 y conector serial de 9 pines.
- Conector PS/2 para mouse o teclado.
- Transceptor IrDA.
- 2 puertos de expansión de 40 pines con diodos de protección.

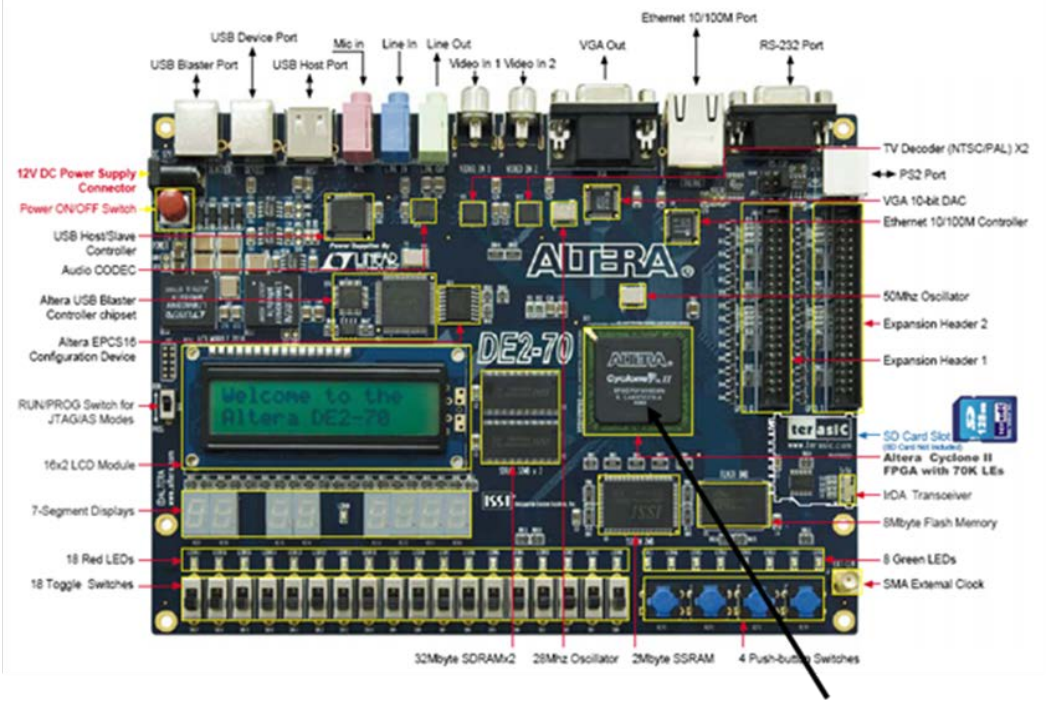


Figura 1.3. Tarjeta de desarrollo DE2-70 de Altera, con FPGA Cyclone II.

1.2.2. Dispositivo FPGA Cyclone II 2C70

En la Figura 1.4, se muestra como se realiza la configuración del FPGA Cyclone II EP2C70F896 [1], que es un dispositivo programable de baja potencia de 68416 elementos lógicos, cuenta con 622 pines para el usuario, de los 896 que tiene en total. Este dispositivo, tiene incluidos bloques pre-diseñados de memoria RAM, 150 multiplicadores y 4 PLLs. Este dispositivo permite realizar la grabación en memoria de los programas desarrollados con las herramientas de software.

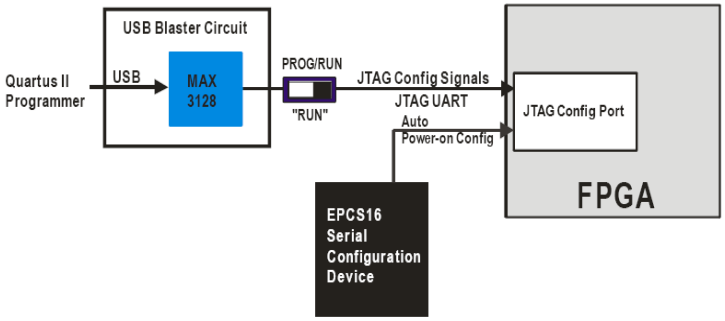


Figura 1.4. Diagrama degrabación y configuración del FPGA.

El FPGA Cyclone II 2C70 de la tarjeta tiene las siguientes características:

- 68416 elementos lógicos.
- 250 M4K bloques (4 Kbits más 512 bits de paridad) de RAM.
- 1152000 bits en total de RAM.
- 150 multiplicadores embebidos.
- 4 PLLs.
- Empaque tipo FBGA (Fine Ball Grid Array) de 896 pines.

1.3. Herramientas de Software

La herramienta de software utilizado en este trabajo, se basa en el paquete de software para FPGAs de Altera (versión 8.1) para sistemas operativos Windows XP SP2, que incluye los siguientes componentes: entornos de desarrollo Quartus II y Nios II, herramientas SoPC Builder y MegaWizard.

La herramienta de software esta disponible en la pagina web de Altera Corp. [1]. Para poder realizar la descarga, previamente se tiene que registrar en dicha página, creando una cuenta de usuario.

Entorno de desarrollo Quartus II

El software Quartus II incluye varias funcionalidades tales como: entrada de diseño mediante programas en código VHDL, Verilog o en diagrama esquemático, compilación, simulación y programación de tarjetas de desarrollo, así como herramientas para crear procesadores embebidos (mediante SoPC Builder) y uso de Librerias parametrizables (usuando la utilidad MegaWizard).

Entorno de desarrollo integrado Nios II IDE

Este software nos permite realizar la programación del procesador embebido en lenguaje C estándar y de los controladores de los periféricos de la tarjeta DE2-70, así como ejecutar la descarga del programa desarrollado al procesador embebido grabado en el FPGA de la tarjeta con ayuda del software Quartus II.

Herramienta SoPC Builder

Esta herramienta nos permite diseñar un sistema completo de procesador basado en el Nios II, el cual es un procesador RISC que combina las ventajas del hardware y del software. Es configurado en hardware usando las compuertas del FPGA y ejecuta un programa en C desarrollado en el entorno Nios II IDE. Este procesador es configurado mediante la herramienta SoPC Builder, en el cual se establecen las conexiones lógicas del procesador con otros elementos del sistema como memorias y periféricos.

1.4. Diagrama de bloques del sistema basado en el procesador Nios II

El sistema basado en el procesador Nios II, consiste en un procesador Nios II configurado en su versión estándar, interfaz de comunicación serial JTAG para la programación del procesador y depuración del software, un PLL y controladores para los dispositivos de la tarjeta externos al FPGA, entre ellos el CODEC de audio, como se ilustra en la Figura 1.5. El referido procesador es utilizado para la implementación del sistema de reconocimiento de palabras.

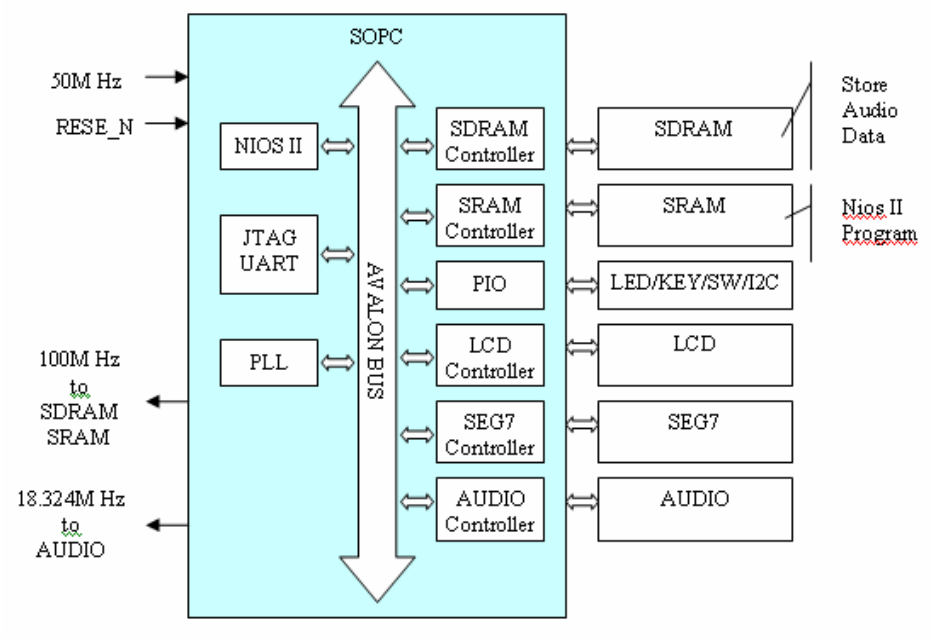


Figura 1.5. Procesador Nios II y módulos de la tarjeta de desarrollo.

1.5. Algoritmos de reconocimiento empleados

Con el propósito de implementar el sistema de reconocimiento de palabras, se implementaron rutinas utilizando el lenguaje de programación C, para tres fines: la conversión de la señal de voz digitalizada por el CODEC, la extracción de características de las señales de voz almacenadas en memoria y el reconocimiento de la palabra entre un grupo de patrones almacenados en memoria. Se analizaron los algoritmos de procesamiento digital de señales, tales como: LPC (Linear Predictive Coding), WT (Wavelet Transform), DTW (Dynamic Time Warping) y ANN (Artificial Neural Networks).

Estas técnicas fueron utilizadas, primeramente, para caracterizar las muestras de voz y crear patrones, usando la Codificación Predictiva Lineal (LPC) así como también el análisis multirresolución provisto por la Transformada Wavelet (WT). Para el reconocimiento automático de patrones se usó el Alineamiento Dinámico en el Tiempo (DTW) así como también la técnica de Redes Neuronales Artificiales (ANN) en su variante de Perceptrón multicapa.

1.6. Procedimiento de implementación del sistema basado en Nios II

A continuación, se presenta los procedimientos para la implementación de un Procesador Nios II, el cual es utilizado en la implementación del sistema de reconocimiento de palabras. Este procesador consta de un CPU (Unidad Central de Procesamiento), Memoria SRAM y DRAM, puertos de entrada y salida (PIO y UART). Asi tambien, es posible adicionar otros componentes disponibles para el procesador embebido.

En las Figuras 1.6, 1.7 y 1.8 se presenta el procedimiento de implementacion del procesador Nios II utilizando la herramienta SoPC Builder. En la Figura 1.6 se muestra como se invoca al SoPC Builder desde Quartus II. Una vez invocado el SoPC Builder aparece una ventana donde se van cargando todas las librerias disponibles al usuario para constuir el sistema, lo cual se aprecia en la Figura 1.7. En la Figura 1.8 se muestra como están establecidas las conexiones lógicas entre el procesador Nios II y los diferentes elementos de hardware como memorias y periféricos.

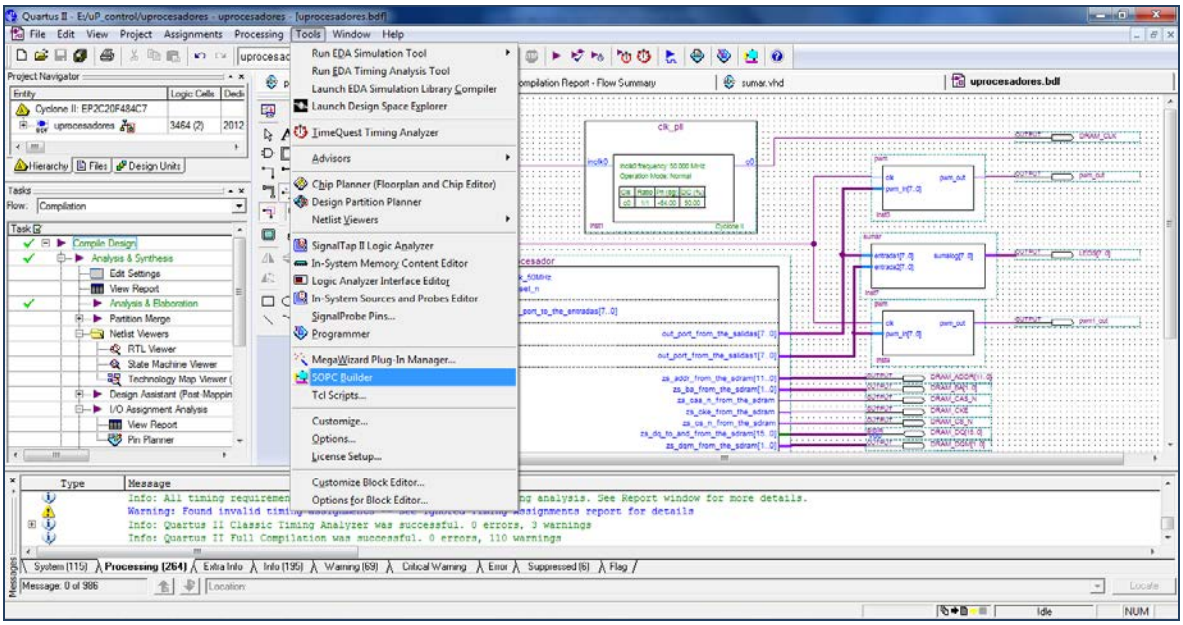


Figura 1.6. Selección de la herramienta SoPC Builder en Quartus II.

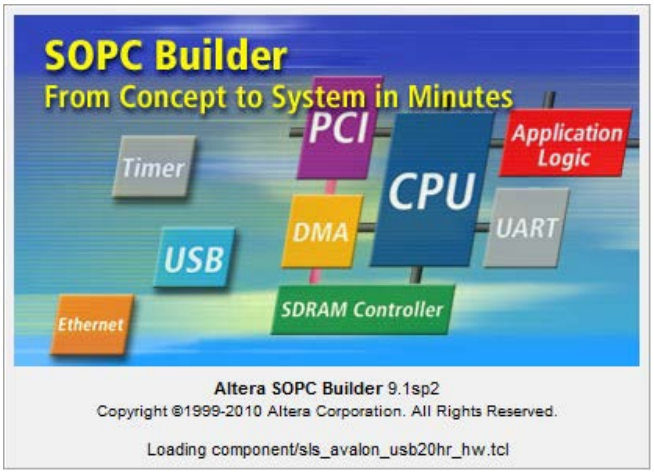


Figura 1.7. Lanzando a la herramienta SoPC Builder.

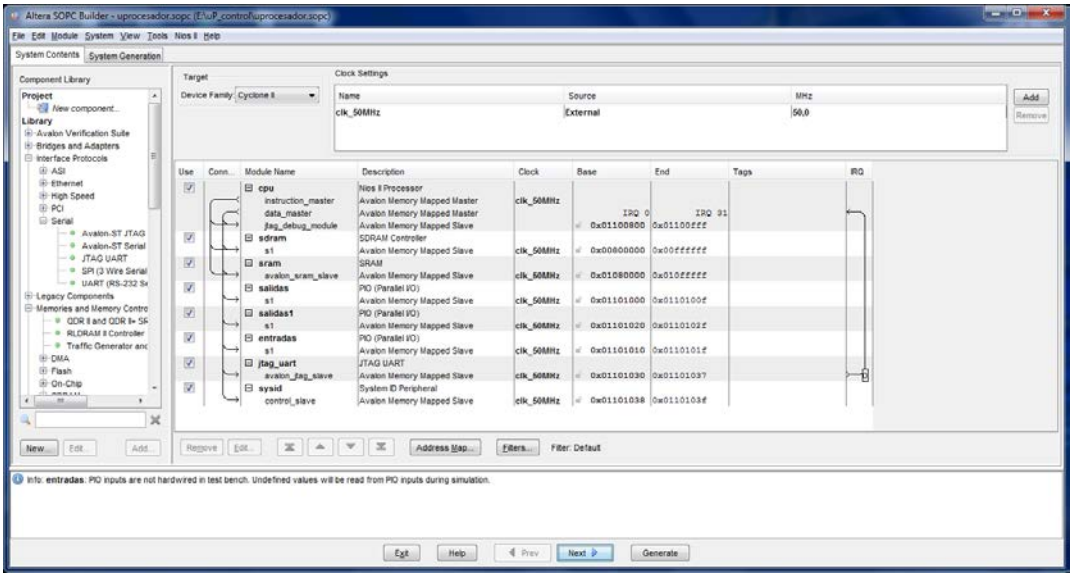


Figura 1.8. Conexiones realizadas en la herramienta SoPC Builder.

Configuraciones del CPU

Las configuraciones disponibles para el procesador embebido Nios II permiten la implementación con recursos suficientes para el sistema de reconocimento de palabras, como se ilustra en las Figuras 1.9 y 1.10. La Figura 1.9 muestra la selección de la versión estándar (Nios II/s) del procesador Nios II, el cual incluye memoria cache de instrucciones, lógica para predicción de ramificaciones, y hardware para multiplicación y división.

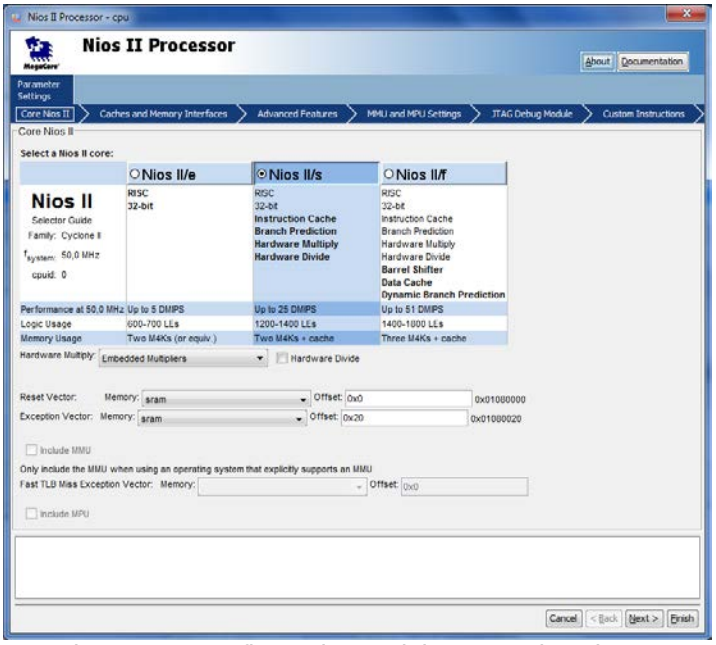


Figura 1.9. Configuraciones del procesador Nios II.

La Figura 1.10 muestra el nivel de depuración a usar, a través de la interfaz JTAG. El nivel seleccionado es el 2, el cual permite bajar el código ejecutable al FPGA, establecer puntos de ruptura de software y 2 puntos de ruptura de hardware.

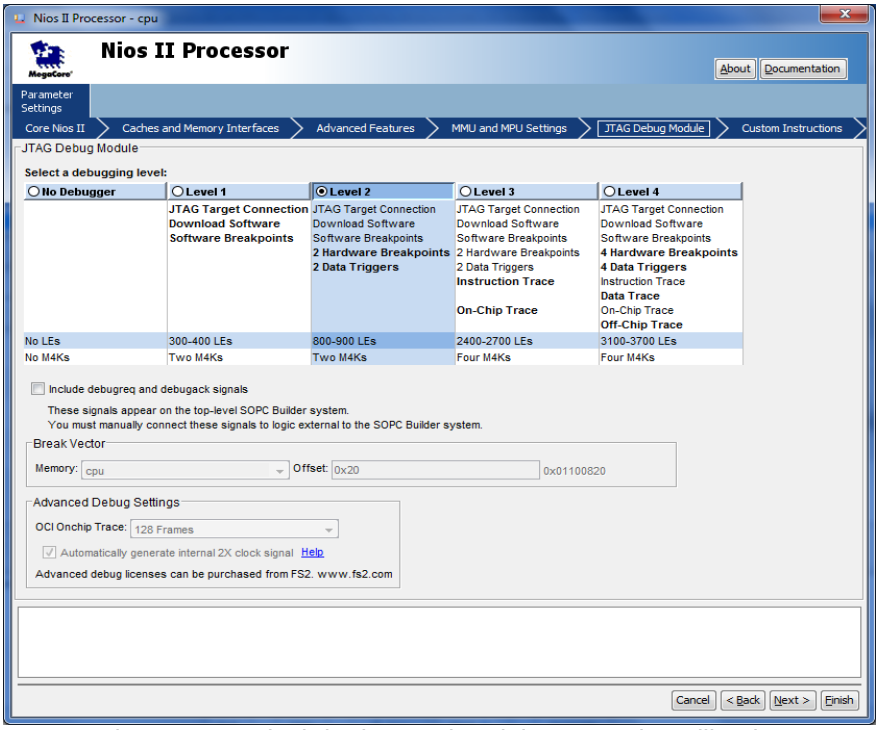


Figura 1.10. Nivel de depuracion del procesador utilizado.

Se debe observar que a mayor nivel de depuración, mayores son los recursos de hardware a usar en el FPGA, así por ejemplo se seleccionó el nivel 2 (Level 2) para la arquitectua de referencia.

Configuraciones de la Memoria SRAM

Con la finalidad de utilizar la memoria SRAM, se hace uso del IP (Intelectual Property) de Altera, el cual permite configurar el tipo de memoria SRAM, como se ilustra en la Figura 1.11.

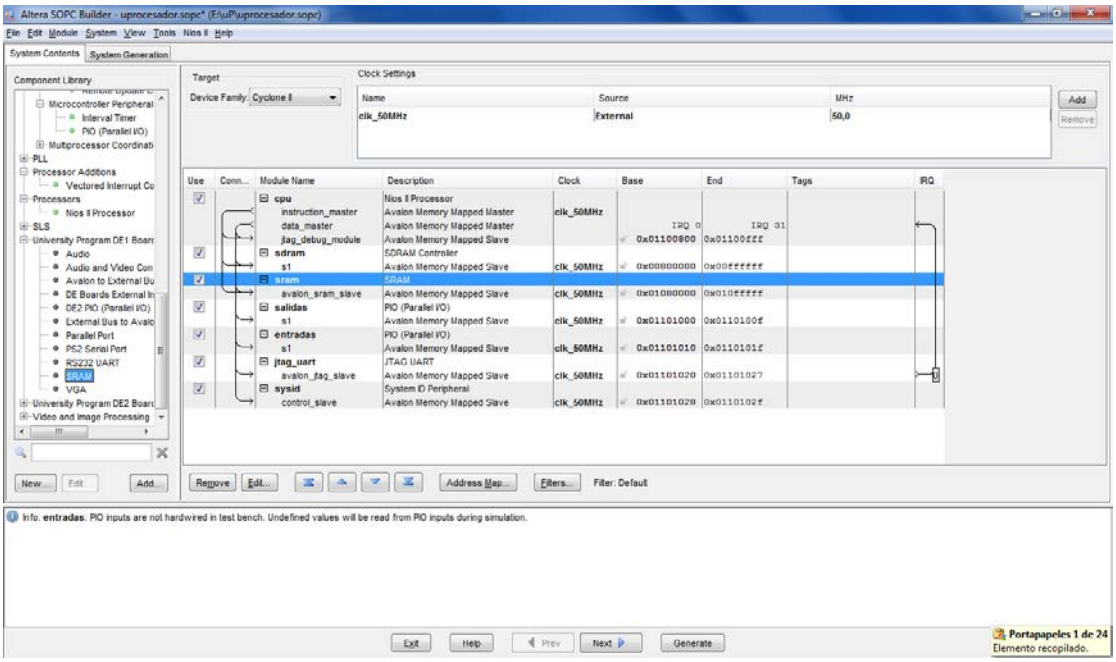


Figura 1.11 Agregando el IP para el uso de la memoria SRAM.

Configuraciones de Entradas y Salidas Paralela (PIO)

Los PIO de salida son utilizados para la comunicación con módulos de memoria externa, son configurados como se muestra en la Figura 1.12.

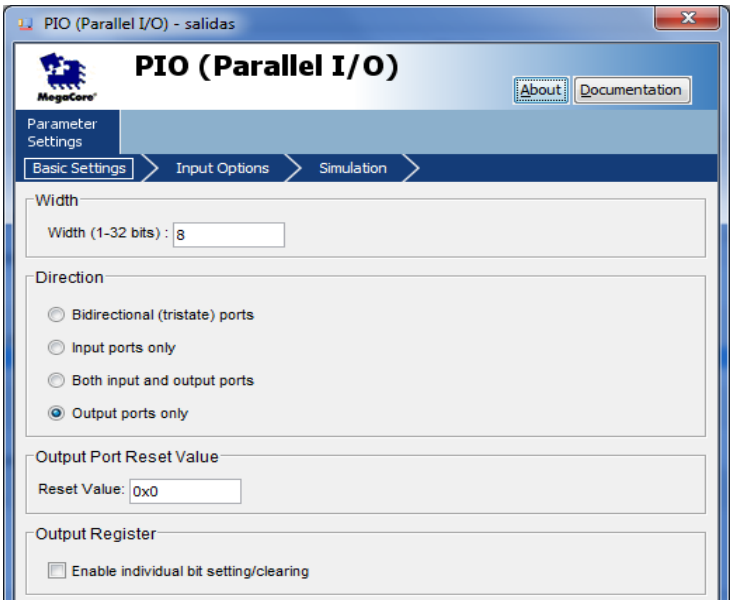


Figura 1.12. Configuración de los PIOS como salidas de 8 bits.

Las entradas PIO, también permiten la comunicación con módulos de memoria, los cuales son configurados como se ilustra en la Figura 1.13.

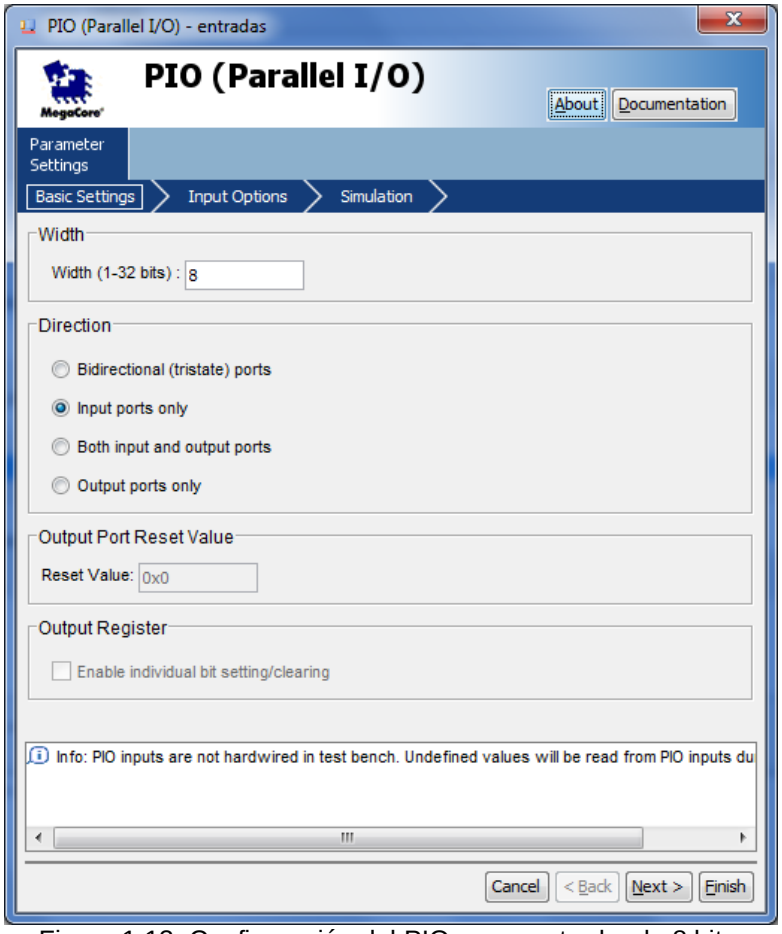


Figura 1.13. Configuración del PIO como entrada, de 8 bits.

Configuraciones del JTAG UART

El JTAG UART se usa para la comunicación entre el FPGA y el módulo del procesador embebido a fin de poder depurar la ejecución de la aplicación via una consola serial, y cuya configuración se ilustra en la Figura 1.14.

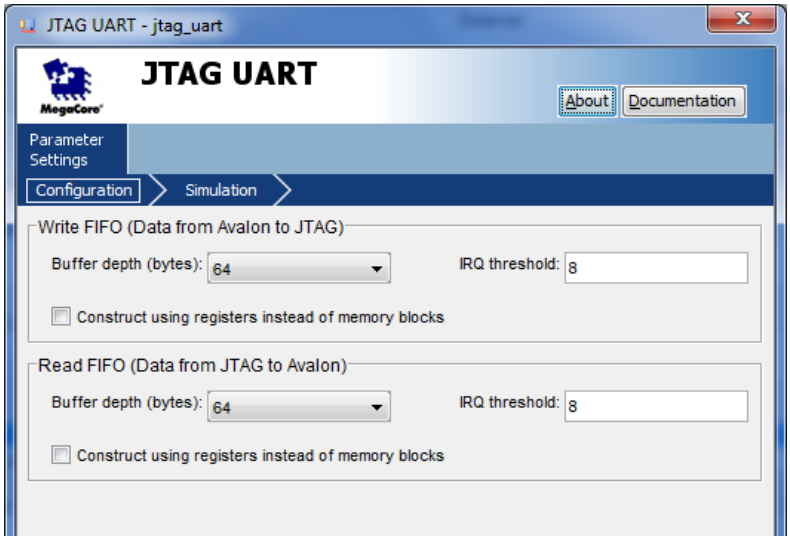


Figura 1.14. Interface de configuración del JTAG UART.

Configuración del SysID

Es necesario configurar el SysID, el cual genera un identificador único para el procesador embebido implementado, como se ilustra en la Figura 1.15. Este valor es verificado cuando se quiera hacer la bajada del código hacia la tarjeta de desarrollo.

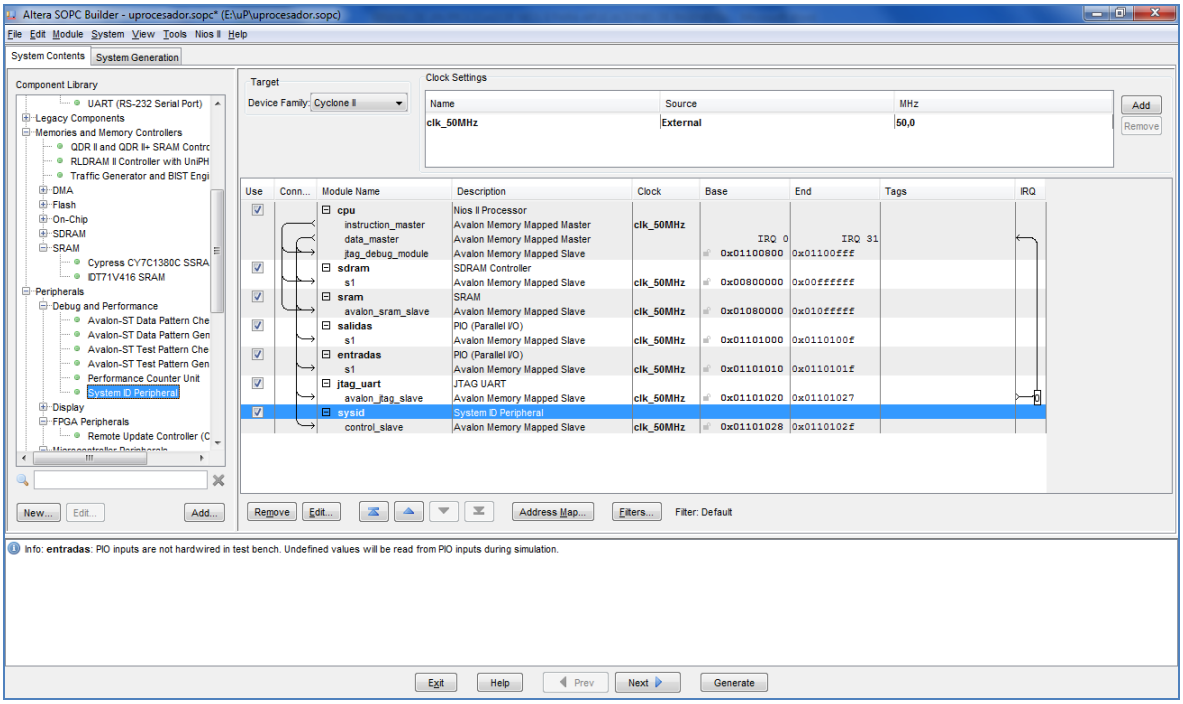


Figura 1.15. Generación del identificador SysID.

Configuración Final

Finalmente, se realiza la configuración de la asignación de las direcciones bases e interrupciones para el procesador embebido, como se ilustran en las Figuras 1.16 y 1.17.

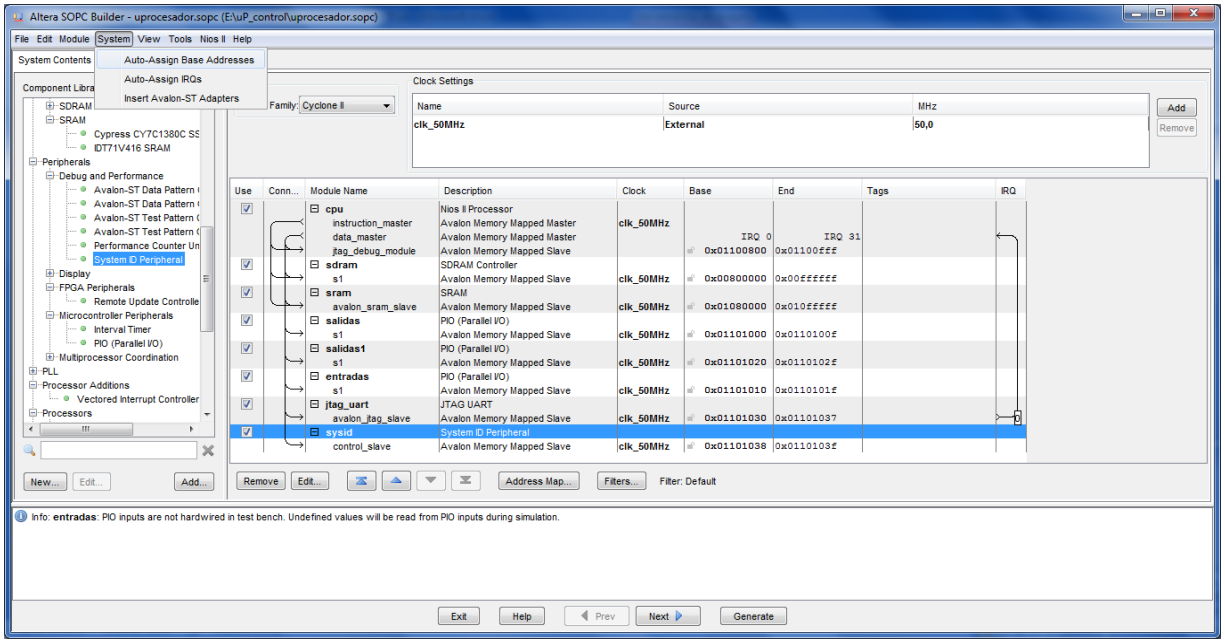


Figura 1.16. Asignación automática de direcciones base en SoPC Builder.

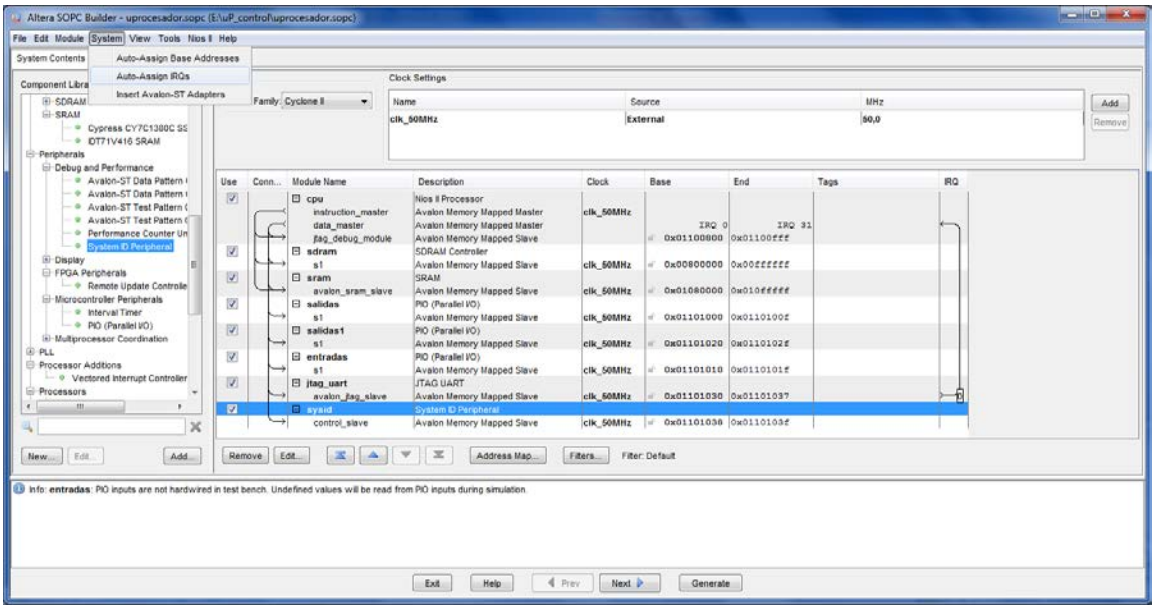


Figura 1.17. Asignación automática de interrupciones en el SoPC Builder.

1.7. Adquisición de Datos

Con la finalidad de realizar la digitalización de las señales de voz, se utiliza el CODEC de audio WM8731 mediante un micrófono en el puerto de entrada ya configurado.

El conversor es del tipo sigma-delta de 24 bits, con entrada para micrófono, entrada y salida de audio. La frecuencia de muestreo se puede programar a un valor entre 8 y 96 kHz. Este dispositivo permite realizar la adquisición de datos de la señal de voz directamente desde un micrófono externo a la tarjeta (ver Figura 1.18).

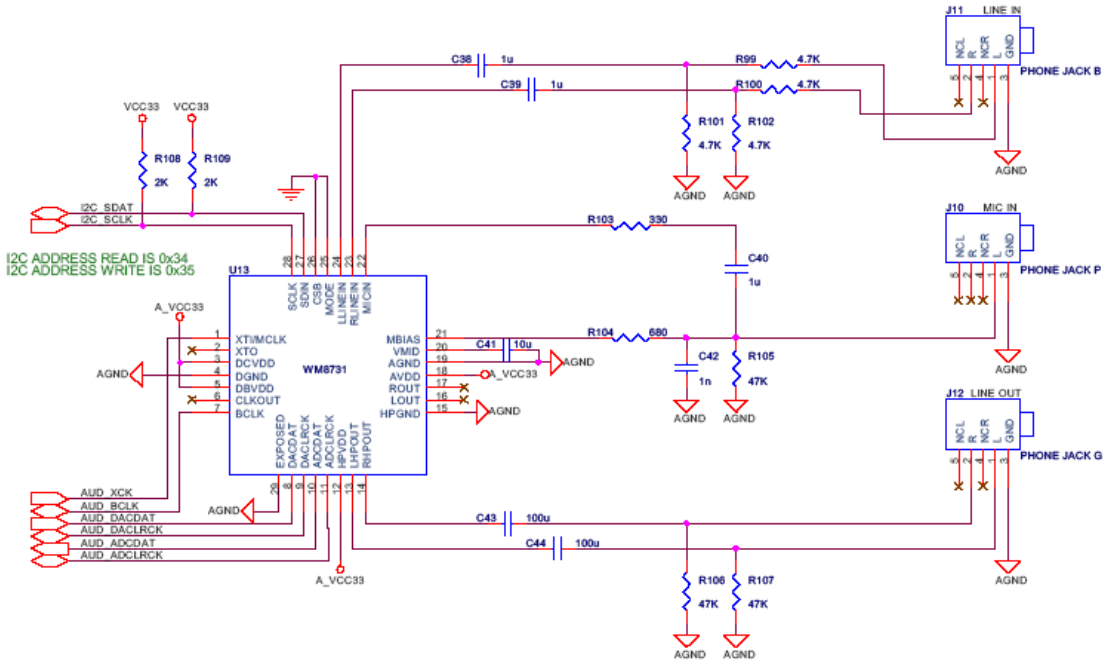


Figura 1.18 Diagrama de conexiones del Códec de audio (WM8731).

El CODEC de audio de la tarjeta tiene las siguientes características:

- CODEC de audio sigma-delta de 24 bits WM8731 de Wolfson.

- Conectores para entrada de micrófono, línea de entrada de audio y línea de salida de audio.
- Frecuencia de muestreo de 8 kHz a 96 kHz.
- Aplicaciones para grabadores y reproductores MP3, teléfonos Smart, grabadores de voz, etc.

1.8. Codificación de la voz

1.8.1. La voz humana

La voz humana se produce voluntariamente por medio del aparato fonatorio. Este está formado por los pulmones como fuente de energía en la forma de un flujo de aire, la laringe, que contiene las cuerdas vocales, la faringe, las cavidades oral (o bucal) y nasal y una serie de elementos articulatorios, los labios, los dientes, el alvéolo, el paladar, el velo del paladar y la lengua (Figura 1.19). Las cuerdas vocales son, en realidad, dos membranas dentro de la laringe orientadas de adelante hacia atrás (Figura 1.20). Por delante se unen en el cartílago tiroides. Por detrás, cada una está sujeta a uno de los dos cartílagos aritenoides, los cuales pueden separarse voluntariamente por medio de músculos. La abertura entre ambas cuerdas se denomina glotis.

Cuando las cuerdas vocales se encuentran separadas, la glotis adopta una forma triangular. El aire pasa libremente y prácticamente no se produce sonido. Es el caso de la respiración. Cuando la glotis comienza a cerrarse, el aire que la atraviesa, proveniente de los pulmones experimenta una turbulencia, emitiéndose un ruido de origen aerodinámico conocido como aspiración (aunque en realidad acompaña a una espiración o exhalación). Esto sucede en los sonidos denominados “aspirados” (como la h inglesa) [11].

Al cerrarse más, las cuerdas vocales comienzan a vibrar a modo de lengüetas, produciéndose un sonido tonal, es decir periódico. La frecuencia de este sonido depende de varios factores, entre otros del tamaño y la masa de las cuerdas vocales, de la tensión que se les aplique y de la velocidad del flujo del aire proveniente de los pulmones. A mayor tamaño, menor frecuencia de vibración, lo cual explica por qué en los varones, cuya glotis es en promedio mayor que la de las mujeres, la voz es en general más grave. A mayor tensión la frecuencia aumenta, siendo los sonidos más agudos. Así, para lograr emitir sonidos en el registro extremo de la voz es necesario un mayor esfuerzo vocal.

También aumenta la frecuencia (a igualdad de las otras condiciones) al crecer la velocidad del flujo de aire, razón por la cual al aumentar la intensidad de emisión se tiende a elevar espontáneamente el tono de voz.

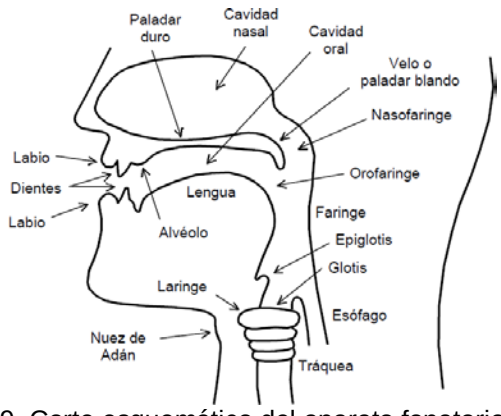


Figura 1.19. Corte esquemático del aparato fonatorio humano.

Finalmente, es posible obturar la glotis completamente. En ese caso no se produce sonido. Sobre la glotis se encuentra la *epiglotis*, un cartílago en la faringe que permite tapar la glotis durante la deglución para evitar que el alimento ingerido se introduzca en el tracto respiratorio. Durante la respiración y la fonación (emisión de sonido) la epiglotis está separada de la glotis permitiendo la circulación del flujo de aire. Durante la deglución, en cambio, la laringe ejecuta un movimiento ascendente de modo que la glotis se apoya sobre la epiglotis.

La porción que incluye las cavidades faríngea, oral y nasal junto con los elementos articulatorios se denomina genéricamente *cavidad supraglótica*, en tanto que los espacios por debajo de la laringe, es decir la tráquea, los bronquios y los pulmones, se denominan *cavidades infraglóticas*. Varios de los elementos de la cavidad supraglótica se controlan a voluntad, permitiendo modificar dentro de márgenes muy amplios los sonidos producidos por las cuerdas vocales o agregar partes distintivas a los mismos, e inclusive producir sonidos propios. Todo esto se efectúa por dos mecanismos principales: el *filtrado* y la *articulación*.

El *filtrado* actúa modificando el espectro del sonido. Tiene lugar en las cuatro cavidades supraglóticas principales: la faringe, la cavidad nasal, la cavidad oral y la cavidad labial. Las mismas constituyen resonadores acústicos que enfatizan determinadas bandas de frecuencia, del espectro generado por las cuerdas vocales, conduciendo al concepto de *formantes*, es decir una serie de picos de resonancia ubicados en fecuencias o bandas de frecuencia que son específicos para cada tipo de sonido.

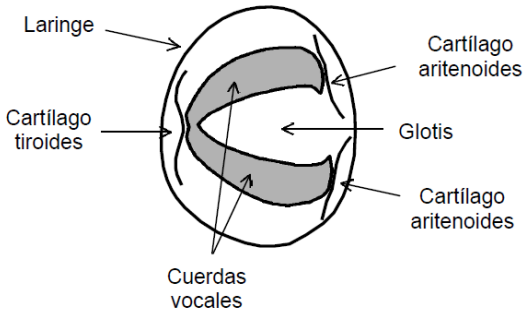


Figura 1.20 Corte esquemático de la laringe según un plano horizontal

La *articulación* es una modificación principalmente a nivel temporal de los sonidos, y está directamente relacionada con la emisión de los mismos y con los fenómenos transitorios que los acompañan. Está caracterizada por el lugar del tracto vocal en que tiene lugar, por los elementos que intervienen y por el modo en que se produce, dan origen a una clasificación fonética de los sonidos.

1.8.2. Modelo de producción de voz

Las señales de voz son producidas cuando una columna de aire desde los pulmones excita el conducto vocal, que se comporta como una cavidad resonante. El conducto vocal es usualmente modelado como una concatenación de tubos acústicos sin pérdidas, con distintas secciones transversales, que comienza en las cuerdas vocales y termina en los labios. La apertura de las cuerdas vocales es denomina glotis. Los diferentes sonidos pueden genéricamente ser clasificados en sonidos tonales (en inglés voiced, como el de las vocales) y sonidos no tonales (en inglés unvoiced, como por ejemplo el de una 's' final de palabra), como se ilustra en la Figura 1.21.

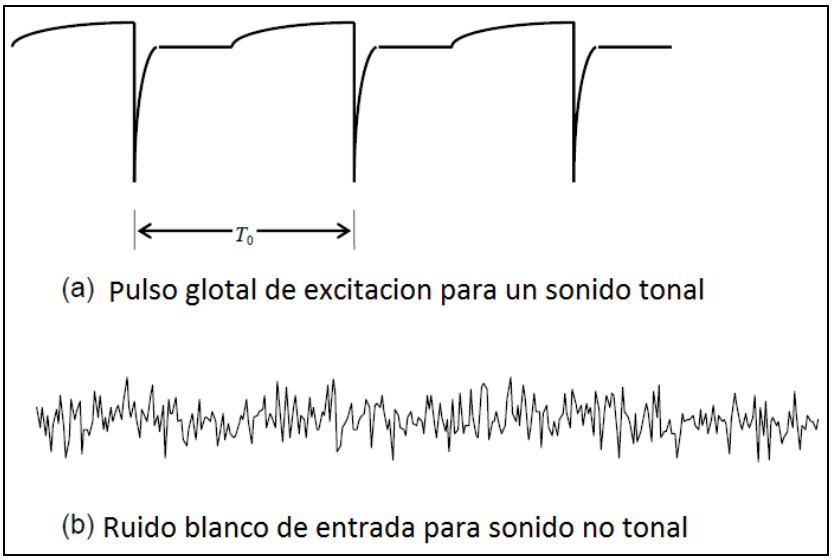


Figura 1.21. Dos tipos de entrada usados para generar sonidos

Los sonidos tonales son producidos al forzar aire a través de la glotis con las cuerdas vocales tensadas de manera que se produce la oscilación relajada de las mismas, excitando de esa forma el conducto vocal con pulsos de aire cuasi-periódicos. Cuanto más grande es la tensión de las cuerdas, más alta es la frecuencia fundamental de la voz producida. Los sonidos no tonales, en tanto, son generados manteniendo las cuerdas abiertas, formando una constricción del conducto vocal, y forzando aire a través de la constricción a una velocidad suficientemente alta para producir turbulencia. En este caso, puede pensarse que el conducto vocal es excitado por una fuente de ruido aleatorio. La Figura 1.22 representa un modelo (en tiempo discreto) del sistema de producción de voz. El conducto vocal se representa por un sistema lineal (en general inestacionario) que es excitado a través de una llave que selecciona entre una fuente de impulsos cuasiperiódicos para el caso de sonidos tonales, o una fuente de ruido aleatorio para el caso de sonidos no tonales. La ganancia apropiada de la fuente, G, es estimada a partir de la señal de voz, y la señal escalada es usada como entrada del modelo del conducto vocal.

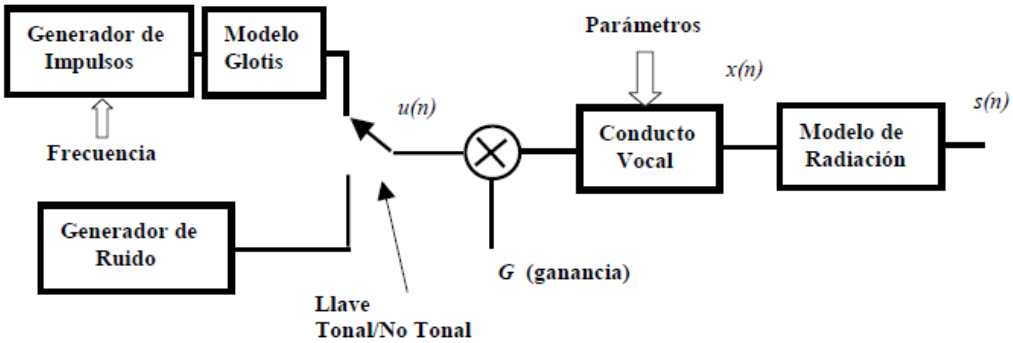


Figura 1.22. Modelo de producción de voz.

1.8.3. Modelo de radiación

En la Figura 1.22, el modelo de radiación describe la impedancia de radiación vista por la presión de aire cuando abandona los labios, que puede ser razonablemente aproximada por una ecuación en diferencias de primer orden, o equivalentemente por una función de transferencia en el dominio de la variable z de la siguiente forma:

$$R(z) = (1 - z^{-1}) \tag{1.1}$$

1.8.4. Modelo de glotis

Existen diferentes modelos de la glotis, que han sido propuestos en diversa literatura, para el caso en que ésta es excitada por pulsos. Un modelo simple es el denominado modelo exponencial representado por una función de transferencia G(z) de la forma:

$$G(z) = \frac{-aeln(a)z^{-1}}{(1-az^{-1})^2} \tag{1.2}$$

Donde: *e* es la base de los logaritmos neperianos. El numerador en (1.2) se selecciona de Manera que $g(n)$ tenga un valor máximo aproximadamente igual a 1.

El modelo está inspirado en mediciones de la respuesta de la glotis a impulsos, que se asemejan a la respuesta de un sistema de segundo orden. Una respuesta típica se representa en la Figura 1.23.

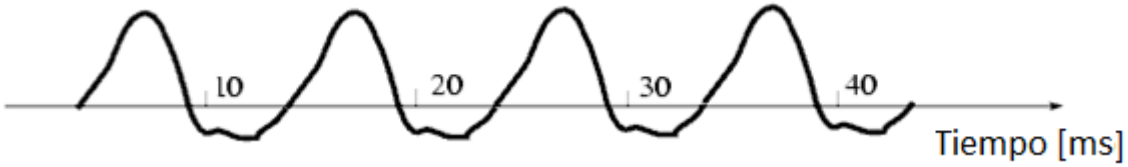


Figura 1.23. Respuesta típica de la glotis a una excitación con tren de impulsos.

1.8.5. Codificación Predictiva Lineal (LPC)

Modelo Predictivo Lineal del Conducto Vocal

Un modelo matemático frecuentemente usado para el conducto vocal + radiación es una ecuación en diferencias (modelo *auto-regresivo*) que se obtiene asumiendo que cada muestra de la señal de voz está estrechamente relacionada con las muestras anteriores, de manera que el valor presente de la señal se puede obtener como una combinación lineal de, por ejemplo, *p* muestras anteriores, es decir:

$$s(n) \cong \sum_{k=1}^{p} a_k * s(n-k) \tag{1.3}$$

Incluyendo un término de excitación G*u(n), la última ecuación puede escribirse como una igualdad de la forma:

$$s(n) = \sum_{k=1}^{p} a_k * s(n-k) + G * u(n) \tag{1.4}$$

Este modelo se denomina modelo de predicción lineal (LPM Linear Predictive Model) para producción/síntesis de voz [7,20], siendo los coeficientes a_k los denominados coeficientes de predicción lineal (LPC), y G, la ganancia de excitación. El modelo LPC puede derivarse discretizando un modelo continuo de transmisión acústica basado en la concatenación de tubos acústicos sin pérdidas.

En el dominio $\mathbf{Z}$ la ecuación (1.4) puede escribirse como:

$$S(z) = \sum_{k=1}^{p} a_k * z^{-k} S(z) + G * U(z) \tag{1.5}$$

Lo que conduce a una función de transferencia:

$$H(z) = \frac{S(z)}{GU(z)} = \frac{1}{1 + \sum_{k=1}^{p} a_k * z^{-k}} = \frac{1}{A(z)} \tag{1.6}$$

Que es del tipo *all pole*, modelado en diagrama de bloques por la Figura 1.24.

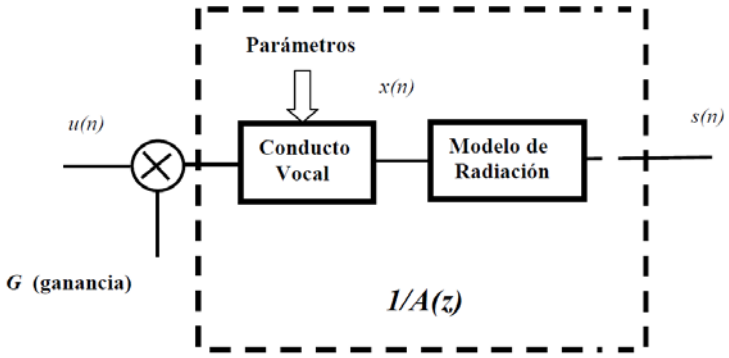


Figura 1.24. Modelo de producción de voz basado en LPC.

Básicamente, *H(z)* representa la función transferencia de un modelo lineal del conducto vocal + radiación**.** Los parámetros del filtro digital *H(z)*, son controlados por la señal de voz que está siendo producida. Los parámetros del modelo completo de Figura 1.22 son dados por la clasificación entre sonidos tonales y no tonales, la frecuencia de los sonidos tonales, la ganancia *G*, y los coeficientes del filtro digital *H(z)*.

Estimación de los LPC

A partir de la ecuación (1.3) vemos que una estimación (o predicción) de s(n) basada en p muestras anteriores, puede calcularse como.

$$\tilde{s}(n) = \sum_{k=1}^{p} a_k * s(n-k) \tag{1.7}$$

Y el error de estimación (predicción) puede entonces definirse como:

$$\varepsilon(n) = s(n) - \tilde{s}(n) \tag{1.8}$$

Resultando entonces en:

$$\varepsilon(n) = s(n) - \sum_{k=1}^{p} a_k * s(n-k) \tag{1.9}$$

Claramente cuando la señal es generada por un sistema lineal como el de la Figura 1.24, entonces el error de estimación $\varepsilon(n)$ deberá igualarse al término de excitación $G * u(n)$.

El modelo predictor puede usarse en telecomunicaciones para incrementar el número de señales de voz que puede transmitirse por un canal. Si los coeficientes a_k son conocidos en el transmisor y en el receptor, entonces só lo necesita transmitirse el error y la señal de voz puede ser reconstruida en el receptor usando la ecuación de diferencias. En el transmisor, $s(n)$ es la entrada al filtro de predicción en tanto que $\varepsilon(n)$ es la salida del filtro. Transmitir solo la señal de error resulta en ahorro substancial de ancho de banda del canal.

El modelo de predicción descrito se modificará para su uso en síntesis de voz. En este caso el problema básico se reduce al cálculo de los parámetros del modelo, es decir de los coeficientes de predicción lineal y de la ganancia de excitación. En la práctica, los coeficientes de predicción deben ser computados a partir de muestras de la señal de voz que se quiere sintetizar. Como la señal es inestacionaria, en el sentido que la configuración del conducto vocal cambia con el tiempo (de acuerdo al sonido que se está emitiendo), el conjunto de coeficientes se debe estimar en forma adaptable sobre cortos intervalos (típicamente de 10 ms a 30 ms de duración) denominados tramas (en inglés *frames*) donde se asume que la señal es estacionaria, y los LPC son constantes.

Típicamente los coeficientes LPC se obtienen minimizando un criterio cuadrático en los errores de predicción $\varepsilon(n)$, para cada *frame* (cuadro) en que es dividido el segmento de voz.

Los valores de $s(n)$ se puede aproximar por una combinación lineal de las p últimas muestras, dados por:

$$\tilde{s}(n) = a_1 s(n-1) + a_2 s(n-2) + \cdots + a_p s(n-p) = \sum_{k=1}^{p} a_k \,.\, s(n-k) \tag{1.10}$$

Con: $1 \leq k \leq p$

Método de la Autocorrelación

De la ecuación 1.8 se sabe que el Error es:

$$\varepsilon(n) = s(n) - \tilde{s}(n) \tag{1.11}$$

Minimizando el error cuadrático medio se calcula los coeficientes a_k:

$$E = \sum_n \varepsilon(n)^2 = \sum_n [s(n) - \tilde{s}(n)]^2 = \sum_n \left[s(n) - \sum_{k=1}^{p} a_k \,.\, s(n-k) \right]^2 \tag{1.12}$$

$$= \sum_n \left[s(n) - [a_1 s(n-1) + a_2 s(n-2) + \cdots + a_p s(n-p)] \right]^2 \tag{1.13}$$

Igualando a cero la derivada parcial de E con respecto a a_1, se obtiene:

$$\frac{\partial E}{\partial a_1} = 0 = \sum_n \left\{2\left[s(n) - \left[a_1 s(n-1) + a_2 s(n-2) + \cdots + a_p s(n-p)\right]\right]^1 . [0 - s(n-1)]\right\}$$

$$\frac{\partial E}{\partial a_1} = -2\sum_n \left\{\left[s(n) - \sum_{k=1}^{p} a_k \, . \, s(n-k)\right] . [s(n-1)]\right\} \tag{1.14}$$

Igualando a cero la derivada parcial de E con respecto a a_2, se obtiene:

$$\frac{\partial E}{\partial a_2} = 0 = \sum_n \left\{2\left[s(n) - \left[a_1 s(n-1) + a_2 s(n-2) + \cdots + a_p s(n-p)\right]\right]^1 . [0 - s(n-2)]\right\}$$

$$\frac{\partial E}{\partial a_2} = -2\sum_n \left\{\left[s(n) - \sum_{k=1}^{p} a_k \, . \, s(n-k)\right] . [s(n-2)]\right\} \tag{1.15}$$

En general anulando la derivada parcial de E con respecto a cualquier a_k .

$$\frac{\partial E}{\partial a_j} = 0 = -2\sum_n \left\{\left[s(n) - \sum_{k=1}^{p} a_k \, . \, s(n-k)\right] . [s(n-j)]\right\} \tag{1.16}$$

Lo que da lugar a la expresión:

$$\sum_n s(n) * s(n-j) = \sum_{k=1}^{p} \left\{a_k \sum_n [s(n-j) * s(n-k)]\right\} \tag{1.17}$$

Para: $j = 1,2,\ldots,p$ y $k = 1,2,\ldots,p$

Que es un sistema lineal de p ecuaciones con p incógnitas, siendo a_k las incógnitas y $s(n), s(n-1$),…, etc. los coeficientes.

Para $j = 1$

$$\sum_{k=1}^{p} \left\{a_k \sum_n [s(n-1) * s(n-k)]\right\} = \sum_n s(n)s(n-1) \tag{1.18}$$

Para $j = 2$

$$\sum_{k=1}^{p} \left\{a_k \sum_n [s(n-2) * s(n-k)]\right\} = \sum_n s(n)s(n-2) \tag{1.19}$$

…

Para $j = p$

$$\sum_{k=1}^{p} \left\{a_k \sum_n [s(n-p) * s(n-k)]\right\} = \sum_n s(n)s(n-p) \tag{1.20}$$

De otra forma, se puede expresar las ecuaciones anteriores como sigue:

$$a_1\sum_n s(n-1)s(n-1)+\ a_2\sum_n s(n-1)s(n-2)+\ \ldots+a_p\sum_n s(n-1)s(n-p)=\sum_n s(n)s(n-1)$$

$$a_1\sum_n s(n-2)s(n-1)+\ a_2\sum_n s(n-2)s(n-2)+\ \ldots+a_p\sum_n s(n-2)s(n-p)=\sum_n s(n)s(n-2)$$

….

$$a_1\sum_n s(n-p)s(n-1)+\ a_2\sum_n s(n-p)s(n-2)+\ \ldots+a_p\sum_n s(n-p)s(n-p)=\sum_n s(n)s(n-p)$$

Una vez definida la autocorrelación de una secuencia, es posible expresar el sistema de ecuaciones lineales anterior (expresión 1.17) de la siguiente manera:

$$\sum_{k=1}^{p}\left\{a_k\sum_n[s(n-j)*s(n-k)]\right\}=\sum_n s(n)*s(n-j) \tag{1.21}$$

Con $k=1,2,\ldots,p$ y n variando desde $-\infty$ hasta $+\infty$. Haciendo el cambio de variable $n\to n+j$, el primer miembro de la expresión 1.21 anterior queda:

$$\sum_{k=1}^{p}\left\{a_k\sum_n[s(n-j)*s(n-k)]\right\}=\sum_n s(n)*s(n+j-k) \tag{1.22}$$

Sin embargo, $\sum_n s(n)*s(n+j-k)$, es precisamente, la autocorrelación en $j-k$, es decir, se tiene:

$$\sum_{k=1}^{p}\left\{a_k\sum_n[s(n-j)*s(n-k)]\right\}=\sum_{k=1}^{p}a_k\,R(j-k) \tag{1.23}$$

Por otro lado, el segundo miembro de la expresión 1.21 queda directamente como la autocorrelación en –j que coincide con la autocorrelación en j al ser esta una función par.

$$R(-j)=R(j) \tag{1.24}$$

Con ello, el método de la autocorrelación se puede expresar de forma compacta como:

$$\sum_{k=1}^{p}a_k\,R(j-k)=R(j)\text{ con }j=1,2,\ldots,p\text{ y con }k=1,2,\ldots,p \tag{1.25}$$

Que se puede expresar como:

para $j=1$ $\quad a_1R(0)+a_2R(-1)+a_3R(-2)+\cdots+a_pR(1-p)=R(1)$

para $j=2$ $\quad a_1R(1)+a_2R(0)+a_3R(-1)+\cdots+a_pR(2-p)=R(2)$

…

para $j=p$ $\quad a_1R(p-1)+a_2R(p-2)+a_3R(p-3)+\cdots+a_pR(0)=R(p)$

Teniendo en cuenta la simetría de la autocorrelación, el anterior sistema de ecuaciones lineales quedaría de la forma:

$$\textbf{para}\quad j=1 \qquad a_1R(0)+a_2R(1)+a_3R(2)+\cdots+a_pR(p-1)=R(1)$$

$$\textbf{para}\quad j=2 \qquad a_1R(1)+a_2R(0)+a_3R(1)+\cdots+a_pR(p-2)=R(2)$$

…

$$\textbf{para}\quad j=p \qquad a_1R(p-1)+a_2R(p-2)+a_3R(p-3)+\cdots+a_pR(0)=R(p)$$

El problema se reduce a resolver un sistema lineal de p ecuaciones con p incógnitas, que se pueden llevar a cabo por cualquiera de los procedimientos habituales del algebra lineal o bien por un procedimiento matricial.

Matricialmente se puede expresar como:

$$\begin{bmatrix} R(0)R(1) & \cdots & R(p-1) \\ R(1)R(0) & \cdots & R(p-2) \\ \vdots & \ddots & \vdots \\ R(p-1) & R(p-2) & \cdots & R(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} = \begin{bmatrix} R(1) \\ R(2) \\ \vdots \\ R(p) \end{bmatrix} \tag{1.26}$$

Despejando (multiplicando por la izquierda a la matriz cuadrada):

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} = \begin{bmatrix} R(0)R(1) & \cdots & R(p-1) \\ R(1)R(0) & \cdots & R(p-2) \\ \vdots & \ddots & \vdots \\ R(p-1) & R(p-2) & \cdots & R(0) \end{bmatrix}^{-1} \begin{bmatrix} R(1) \\ R(2) \\ \vdots \\ R(p) \end{bmatrix} \tag{1.27}$$

La matriz cuadrada es una matriz que tiene las siguientes particularidades:

- Los elementos de la diagonal principal son iguales $R(0)$).
- Es simétrica respecto a la diagonal principal.
- Se conoce como matriz Toeplitz.

En resumen, los coeficientes de predicción lineal a_k se pueden calcular realizando la inversa de la matriz Toepelitz, y multiplicándola por la matriz columna **(R**). Los elementos de ambas matrices son exclusivamente los valores de la autocorrelación R, desde 0 hasta p-1 en la matriz Toeplitz, y desde 1 hasta p en la matriz columna **(R**).

A continuación, se presenta un ejemplo del método de autocorrelación, donde se presentan 10 muestras s(n) y 10 coeficientes LPC R(n), dados por:

$$s(0)=-0.2578 \qquad R(0)=0.8448$$

$$s(1)=-0.28125 \qquad R(1)=0.7790$$

$$s(2)=-0.2890 \qquad R(2)=0.6980$$

$$s(3)=-0.2969 \qquad R(3)=0.6086$$

$s(4) = -0.3047$ $R(4) = 0.5152$

$s(5) = -0.3203$ $R(5) = 0.4182$

$s(6) = -0.3125$ $R(6) = 0.3218$

$s(7) = -0.3047$ $R(7) = 0.2299$

$s(8) = -0.28125$ $R(8) = 0.1428$

$s(9) = -0.2500$ $R(9) = 0.06445$

Utilizando la ecuación (1.27), resulta en:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} R(0) & R(1) & R(2) & R(3) \\ R(1) & R(0) & R(1) & R(2) \\ R(2) & R(1) & R(0) & R(1) \\ R(3) & R(2) & R(1) & R(0) \end{bmatrix}^{-1} \begin{bmatrix} R(0) \\ R(1) \\ R(2) \\ R(3) \end{bmatrix} = \begin{bmatrix} 1.0441 \\ -0.0491 \\ -0.0211 \\ -0.0824 \end{bmatrix}$$

$a_1 = 1.0441 \quad a_2 = -0.0491 \quad a_3 = -0.0211 \quad a_4 = -0.0824$

Método recursivo de Levinson-Durbin

Es un método alternativo de calcular los coeficientes a_k de predicción lineal.

$$E^{(0)} = R(0)$$

$$K_i = \frac{R(i) - \sum_{j=1}^{i-1} \alpha_j^{(i-1)} R(i-j)}{E^{(i-1)}} \text{con} \quad 1 \leq i \leq p$$

$$\alpha_i^{(i)} = K_i$$

$$\alpha_j^{(i)} = \alpha_j^{(i-1)} - K_i \alpha_{i-j}^{(i-1)} \text{con } 1 \leq j < i$$

$$E^{(i)} = \left(1 - K_i^{\,2}\right) E^{(i-1)}$$

Los resultados son los mismos que produce el método de la autocorrelación, presentados a través del siguiente ejemplo:

$R(0) = 0.8448$

$R(1) = 0.7790$

$R(2) = 0.6980$

$R(3) = 0.6080$

$R(4) = 0.5152$

$R(5) = 0.4182$

$R(6) = 0.3218$

$$R(7) = 0.2299$$

$$R(8) = 0.1428$$

$$R(9) = 0.06445$$

Para i=1.

$$E^{(0)} = R(0) = 0.8448$$

$$K_1 = \frac{R(1) - 0}{E^{(0)}} = 0.9221$$

$$\alpha_1^{(1)} = K_1 = 0.9221$$

$$E^{(1)} = (1 - K_1^{\,2})E^{(0)} = 0.126475$$

Para i=2.

$$K_2 = \frac{R(2) - \sum_j \alpha_j^{(i-1)} R(i-j)}{E^{(1)}} = \frac{R(2) - \alpha_1^{(1)} R(1)}{E^{(1)}} = -0.1607 \quad para \quad 1 \le j \le 1$$

$$\alpha_2^{(2)} = K_2 = -0.1607$$

$$\alpha_1^{(2)} = \alpha_1^{(1)} - K_2 \alpha_1^{(1)} = 1.0703$$

$$E^{(2)} = (1 - K_2^{\,2})E^{(1)} = 0.1232$$

Para i=3.

$$K_3 = \frac{R(3) - \sum_j \alpha_j^{(i-1)} R(i-j)}{E^{(2)}} = \frac{R(3) - [\alpha_1^{(2)} R(2) + \alpha_2^{(2)} R(1)]}{E^{(2)}} = -0.1078 \quad para \quad 1 \le j \le 2$$

$$\alpha_3^{(3)} = K_3 = -0.1078$$

$$\alpha_2^{(3)} = \alpha_2^{(2)} - K_3 \alpha_1^{(2)} = -0.04534$$

$$E^{(3)} = (1 - K_3^{\,2})E^{(2)} = 0.1218$$

Para i=4.

$$K_4 = \frac{R(4) - \sum_j \alpha_j^{(i-1)} R(i-j)}{E^{(3)}} = \frac{R(4) - [\alpha_1^{(3)} R(3) + \alpha_2^{(3)} R(2) + \alpha_3^{(3)} R(1)]}{E^{(3)}} =$$

$$= -0.0823 \quad para \quad 1 \le j \le 3$$

$$\alpha_4 = \alpha_4^{(4)} = K_4 = -0.0823$$

$$a_3 = \alpha_3^{(4)} = \alpha_3^{(3)} - K_4\alpha_1^{(3)} = -0.02107$$

$$a_2 = \alpha_2^{(4)} = \alpha_2^{(3)} - K_4\alpha_2^{(3)} = -0.04907$$

$$a_1 = \alpha_1^{(4)} = \alpha_1^{(3)} - K_4\alpha_3^{(3)} = 1.0441$$

Los coeficientes de predicción lineal son los últimos valores de α calculados. Por ejemplo para p=4, $a_1 = \alpha_1^{(4)}$; $a_2 = \alpha_2^{(4)}$; $a_3 = \alpha_3^{(4)}$; $a_4 = \alpha_4^{(4)}$

1.9. Reconocimiento de voz

Para el reconocimiento de patrones de voz se usan comúnmente uno de estos 3 métodos: alineamiento dinámico del tiempo (DTW), modelos ocultos de Markov (HMM), o redes neuronales artificiales (ANN). En este trabajo, se ha optado por las redes neuronales artificiales, donde los patrones a reconocer son señales de voz codificadas.

Redes Neuronales Artificiales (ANN)

Redes neuronales artificiales son redes interconectadas masivamente en paralelo de elementos simples (usualmente adaptativos) y con organización jerárquica, las cuales intentan interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso biológico.

Modelo de Neurona Biológica

Una neurona consta de un cuerpo celular más o menos esférico, de 5 a 10 µm de diámetro, del que sale una rama principal, el axón, y varias ramas cortas, llamadas dentritas. Una de las características que diferencian a las neuronas del resto de las células vivas es su capacidad de comunicarse, como se ilustra en la Figura 1.25. En términos generales, las dentritas y el cuerpo celular reciben señales de entrada; el cuerpo celular las combina e integra y emite señales de salida. El axón transporta esas señales a los terminales axónicos, que se encargan de distribuir información a un conjunto de neuronas. Se calcula que en el cerebro humano existen $1x10^{15}$ conexiones entre neuronas [12].

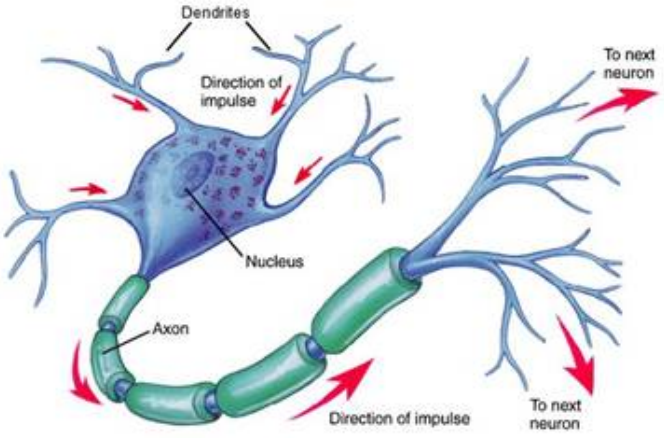


Figura 1.25. Representación de una neurona biológica.

Las señales que se utilizan son de de naturaleza eléctrica y química. La señal generada por la neurona y transportada a lo largo del axón es un impulso eléctrico, mientras

que la señal que se transmite entre los terminales axónicos de una neurona y las dentritas de las neuronas siguientes es de origen químico; concretamente, se realiza mediante moléculas de sustancias transmisoras (neurotransmisores) que fluyen a través de unos contactos especiales, llamados sinapsis, que tienen la función de receptor y están localizados entre los terminales axónicos y las dentritas de la neurona siguiente (espacio sináptico: entre 50 y 200 Angstroms, 1 Angstrom = $1x10^{-10}$ m).

La generación de las señales eléctricas está íntimamente relacionada con la composición de la membrana celular. Existen muchos procesos complejos relacionados con la generación de dichas señales, sin embargo se puede simplificar así, la neurona, como todas las células, es capaz de mantener en su interior un líquido cuya composición difiere marcadamente de la composición del líquido exterior. La diferencia más notable se da en relación con la concentración de iones sodio y potasio. El medio externo es unas 10 veces más rico en sodio que el interno, mientras que el medio interno es unas 10 veces más rico en potasio que el externo. Esta diferencia de concentración en iones sodio y potasio a cada lado de la membrana produce una diferencia de potencial de aproximadamente 70 mV, negativa en el interior de la célula. Llamado potencial en reposo de la célula nerviosa, como se ilustra en la Figura 1.26.

La llegada de señales procedentes de otras neuronas a través de las dentritas (recepción de neurotransmisores) actúa acumulativamente, bajando ligeramente el potencial de reposo. Dicho potencial modifica la permeabilidad de la membrana, de manera que cuando llega a cierto valor crítico comienza una entrada masiva de iones sodio que invierten la polaridad de la membrana. La inversión del voltaje de la cara interior de la membrana cierra el paso a los iones de sodio y abre el paso a los iones de potasio hasta que se restablece el equilibrio en reposo. La emisión del voltaje, conocida como potencial de acción, se propaga a lo largo del axón y, a su vez, provoca la emisión de los neurotransmisores en los terminales axónicos. Después de un pequeño periodo refractario, puede seguir un segundo impulso. El resultado de todo esto es la emisión por parte de la neurona de trenes de impulsos cuya frecuencia varía en función (entre otros factores) de la cantidad de neurotransmisores recibidos.

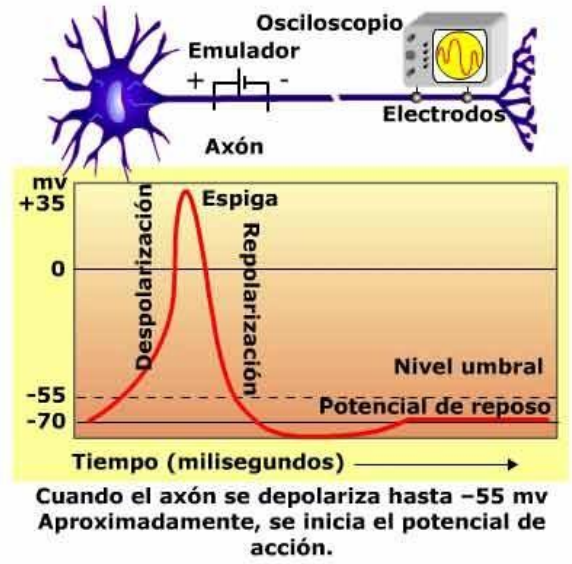


Figura 1.26. Potencial de acción en una neurona biológica.

Existen 2 tipos de sinapsis: a) las sinapsis excitadoras, cuyos neurotransmisores provocan disminuciones de potencial en la membrana de la célula postsináptica, facilitando la generación de impulsos a mayor velocidad, y b) las sinapsis inhibidoras, cuyos neurotransmisores tienden a estabilizar el potencial de la membrana, dificultando la emisión

de impulsos. En cada instante, algunas neuronas estarán activas y otras se hallarán en reposo; la suma de los efectos excitadores e inhibidores determina si la célula será o no estimulada; es decir, si emitirá o no un tren de impulsos y a qué velocidad [12].

Modelo de Neurona Artificial

Las redes neuronales artificiales (ANN) [8], son modelos que intentan reproducir el comportamiento del cerebro. Como tal modelo, realiza una simplificación, averiguando cuáles son los elementos relevantes del sistema, pude ser porque la cantidad de información de que se dispone es excesiva o bien porque es redundante. Una elección adecuada de sus características, más una estructura conveniente, es el procedimiento convencional utilizado para construir redes capaces de realizar determinada tarea.

Cualquier modelo de red neuronal consta de dispositivos elementales de proceso: las neuronas. A partir de ellas, se pueden generar representaciones específicas, de tal forma que un estado conjunto de ellas puede significar una letra, un número o cualquier otro objeto. Generalmente se pueden encontrar tres tipos de neuronas: aquellas que reciben estímulos externos (neuronas de entrada), aquellas que se ocupan del procesamiento (neuronas ocultas), y aquellas que dan la respuesta del sistema (neuronas de salida).

Perceptrón Simple

En la Figura 1.27, se ilustra el modelo del perceptrón simple consiste en 2 capas de neuronas: una con N neuronas de entrada, y otra con 1 neurona de salida. Esta única entrada permite diferenciar dos clases de patrones que son linealmente separables.

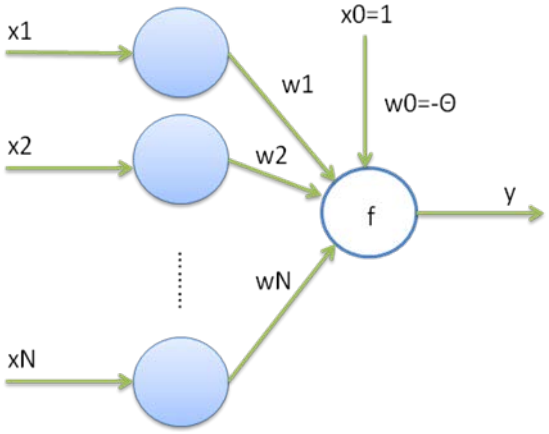


Figura 1.27. Representación de una ANN tipo perceptrón simple.

Las entradas x_i multiplicadas por sus pesos w_i se suman, esto incluye la resta ocasionada por el umbral Θ. La salida "y" es igual a la función de transferencia "f" aplicada a dicha suma.

$$f(x) = \begin{cases} 1, x > 0 \\ 0, x < 0 \end{cases} \tag{1.28}$$

$$y = f\left(\sum_{i=1}^{N} w_i x_i - \theta\right) \tag{1.29}$$

La definición de la función de transferencia de la neurona de salida muestra la diferenciación de clases por parte de la red: si la suma de estímulos supera un umbral Θ, entonces la entrada (x_1, x_2) pertenece a una clase A; y si la suma ponderada de entradas no supera el umbral Θ, entonces el patrón de entrada pertenece a una clase B. Por tanto, la

frontera entre las 2 clases que puede diferenciar el perceptrón simple se define cuando la suma ponderada de las entradas es igual al umbral. Por ejemplo, para N=2:

$$N = 2 \rightarrow y = f(x_1 w_1 + x_2 w_2 - \theta) \tag{1.30}$$

$$Frontera: x_2 = \frac{-w_1}{w_2} x_1 + \frac{\theta}{w_2} \tag{1.31}$$

Entrenamiento del Perceptrón Simple

Entrenar una red neuronal artificial significa hallar los valores de los parámetros de la red tales que se verifiquen los pares (entrada, salida deseada) con los cuales se entrena la red. Para ello se inicializan los parámetros de la red de manera aleatoria y se van ajustando en función del error en la salida respecto del valor deseado.

$$e(t) = d_i - f[\sum_{i=1}^{N} x_i w_i(t) - \theta(t)] \tag{1.32}$$

$$w_i(t+1) = w_i(t) + \alpha\, e(t) x_i \tag{1.33}$$

El factor de ganancia α (está en el rango de 0.0 a 1.0) es ajustado experimentalmente de forma que satisfaga tanto los requerimientos de aprendizaje rápido como la estabilidad de las estimaciones de los pesos. El error $e(t)$ es la diferencia entre la salida deseada d_i y la salida "y" de la red ante la entrada de entrenamiento x_i. Para calcular los pesos de la red se itera hasta que el valor del error $e(t)$ sea cero o muy pequeño, entonces se dice que el sistema está entrenado para los pares de entrada-salida usados en las iteraciones.

El perceptrón simple puede entonces clasificar un vector de entrada en dos clases A y B que son linealmente separables. Sin embargo, existen casos en los cuales dos clases no son linealmente separables. Para ello se usan neuronas adicionales ubicadas en capas ocultas de la red; esto da lugar al perceptrón multicapa, el cual es usado en este trabajo.

Perceptrón Multicapa

Un perceptrón multinivel o multicapa es una red de tipo hacia-adelante compuesta de varias capas de neuronas entre las de entrada y salida, como se ilustra en la Figura 1.28. Esta red permite establecer regiones de decisión mucho más complejas que las de 2 semiplanos, como hace el perceptrón de un solo nivel (perceptrón simple). Para entrenar un perceptrón multicapa se usa el agoritmo de propagación hacia atrás (backpropagation).

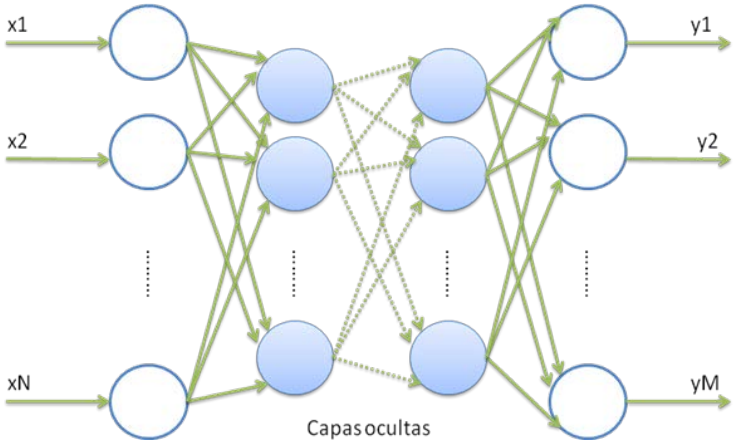


Figura 1.28. Representación de una ANN tipo perceptrón multicapa.

En MATLAB [6], una red perceptrón consiste en una capa simple de neuronas perceptrón conectadas a R entradas a través de un conjunto de pesos w_{ij}. Los índices de red i, j indican que w_{ij} es la intensidad de la conexión de la j-ésima entrada a la i-ésima neurona, como se ilustra en la Figura 1.29.

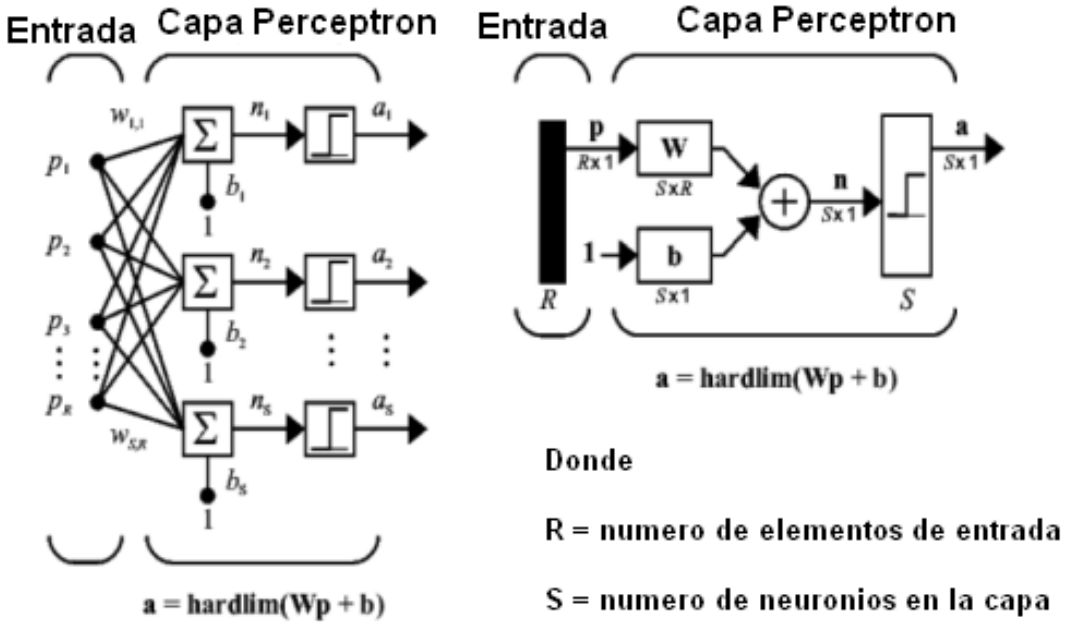


Figura 1.29. Capa perceptrón de ANN en MATLAB.

La función de transferencia hard-limit retorna un 0 o 1 si la sumatoria de sus excitaciones a su entrada es negativa o positiva respectivamente, como se ilustra en la Figura 1.30.

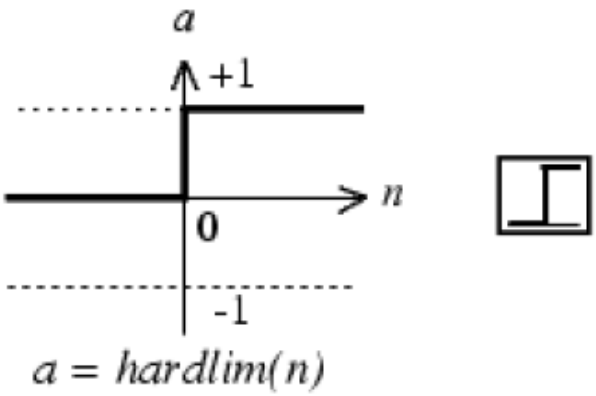


Figura 1.30. Función de activación hardlim.

Una red hacia-adelante (feedforward) simple consiste en una capa simple de neuronas logsig que tiene R entradas. La función logsig genera salidas entre 0 y 1 cuando las neuronas de entrada van de menos infinito a más infinito. Como una alternativa, las redes multicapa pueden usar la función de transferencia tansig o la función de transferencia lineal purelin en redes de propagación hacia atrás, como se ilustra en la Figura 1.31.

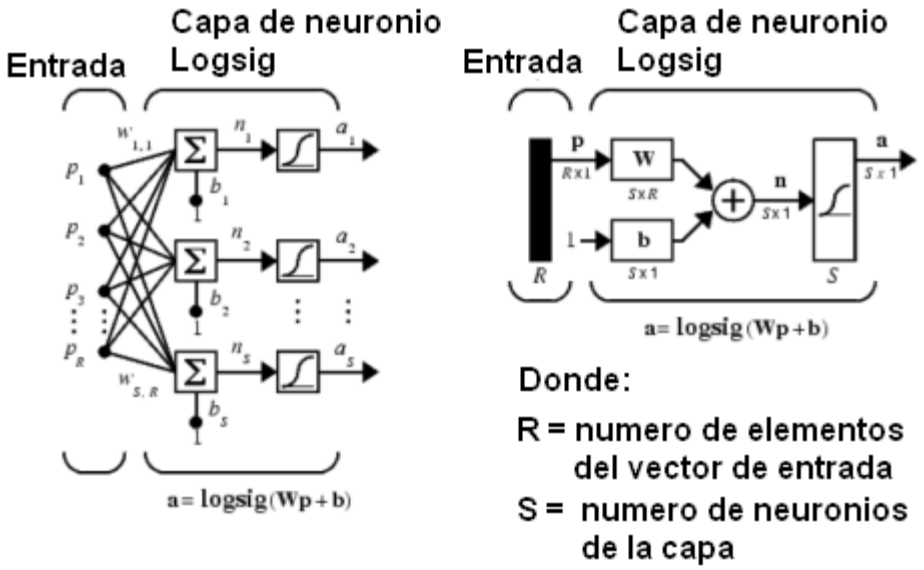


Figura 1.31. Capa logsig de RNA en MATLAB.

Las redes hacia-adelante usualmente tienen una o más capas ocultas de salida de neuronas lineales. Capas múltiples de neuronas con funciones de transferencia no lineales permiten que la red aprenda relaciones lineales y no lineales entre los vectores de entrada y salida. La capa de salida lineal hace que la red produzca valores fuera del rango de -1 a 1. Por otro lado, si se quiere restringir las salidas de la red (por ejemplo entre 0 y 1), la capa de salida debe usar una función de transferencia sigmoidal (como logsig), como se ilustra en la Figura 1.32.

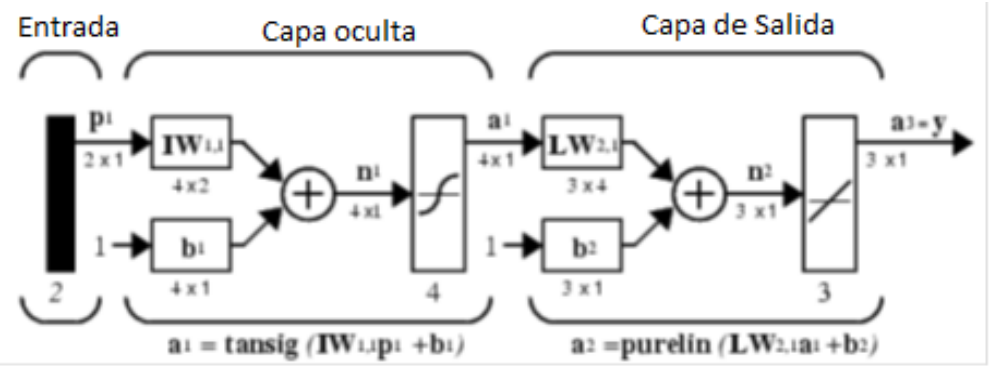


Figura 1.32. Red neuronal artificial en MATLAB.

Algoritmo Backpropagation

Hay muchas variantes del algoritmo de propagación hacia atrás (backpropagation por su término en inglés), una de las cuales se describen a continuación. La implementación más simple del aprendizaje por propagación hacia atrás actualiza los pesos de la red y los umbrales en la dirección que la función rendimiento (performance) disminuye más rápido, el negativo del gradiente. Una iteración de este algoritmo puede ser escrita como sigue:

$$x_{k+1} = x_k - \alpha_k g_k \tag{1.34}$$

Donde: x_k es un vector de pesos y umbrales actuales, g_k es el gradiente actual y α_k es la taza de aprendizaje.

Hay dos formas diferentes en las que este algoritmo de disminución de gradiente puede ser implementado, modo incremental y modo batch. En el modo incremental, el gradiente es calculado y los pesos son actualizados después que cada entrada es aplicada a la red. En el modo batch, todas las entradas son aplicadas a la red antes de que los pesos sean actualizados.

En modo batch, los pesos y umbrales de la red son actualizados solo después que el conjunto completo de entrenamiento se haya aplicado a la red. Los gradientes calculados en cada ejemplo de entrenamiento son agregados todos para determinar el cambio en los pesos y umbrales. La función de entrenamiento de gradiente descendente en MATLAB es traingd, que actualiza los pesos y umbrales en la dirección del gradiente negativo de la función de rendimiento.

La función traingd puede entrenar cualquier red siempre que su peso, red de entrada y función de transferencia tenga funciones derivativas. La propagación hacia atrás (backpropagation) se usa para calcular las derivadas del rendimiento perf respecto de las variables peso y umbral X. Cada variable es ajustada de acuerdo al gradiente descendente:

$$dX = lr * \frac{d}{dX} perf \tag{1.35}$$

$$perf : funcion\ \ performance$$

En el presente capítulo se ha cubierto el marco teórico necesario para desarrollar el trabajo de tesis, empezando por explicar las herramientas de hardware y software a usar con el fin de diseñar e implementar un sistema basado en un procesador configurable en FPGA. Luego, se han cubierto los conceptos necesarios para el tratamiento de la voz humana, desde la adquisición, codificación, procesamiento y reconocimiento. En los subsiguientes capítulos se cubrirá la definición del problema, objetivos que se persiguen con el presente trabajo, así como la propuesta de solución, para luego proceder a presentar los resultados de la simulación y posterior implementación del sistema en la tarjeta desarrollo seleccionada.

CAPÍTULO 2
DEFINICIÓN DEL PROBLEMA

En el presente capítulo se describe el problema a ser resuelto por el presente trabajo de tesis, para luego plantear los objetivos generales y específicos que se persiguen con el desarrollo del presente trabajo, mencionando las limitaciones del mismo.

2.1. Planteamiento del problema

El reconocimiento de la voz humana ha sido y es un problema difícil de resolver, debido a que la producción de voz en el ser humano es un proceso complejo. Además, el reconocimiento de palabras habladas, que es objetivo del presente trabajo, depende de muchas variables como tipo de hablante (hombre o mujer, adulto o niño), rapidez de pronunciación, estado de ánimo, ruido del medio, cantidad de palabras que puede reconocer el sistema. Se ha planteado soluciones restringiendo el problema, por ejemplo, a un solo hablante, y se han usado diversos dispositivos de hardware y algoritmos en software para su implementación.

El trabajo que se describe a continuación trata de afrontar el problema de reconocimiento de voz, en el sentido de diferenciar palabras habladas y sus significados, usando dispositivos modernos, como un FPGA, y técnicas modernas, como Redes Neuronales Artificiales, con el fin de obtener un buen rendimiento. Las aplicaciones que existen son diversas, control de máquinas, por voz, que cuenten con la interfaz planteada, implementación de artefactos inteligentes capaces de comunicarse con el ser humano por voz, entre otros.

El problema consiste en reconocer patrones de señales de voz mediante un método de reconocimiento adecuado para tener el máximo de acierto al momento de hacer las pruebas de verificación.

2.2. Objetivos generales

Realizar el reconocimiento de palabras habladas por un locutor en particular, utilizando un sistema basado en un procesador configurable en dispositivo FPGA, realizando la comparación con patrones de voz guardados anteriormente en un dispositivo de almacenamiento.

2.3. Objetivos específicos

Implementar el sistema de reconocimiento de palabras aisladas en una tarjeta de desarrollo basada en FPGA que permita realizar el tratamiento de la voz, desde su adquisición, codificación, posterior tratamiento, y reconocimiento, usando técnicas modernas.

Realizar la adquisición de las señales de voz (mediante el Codec de audio brindado por la tarjeta de desarrollo), realizar su compresión (codificación predictiva lineal, LPC) para procesarla posteriormente, desarrollando para esto un sistema basado en un procesador configurable definido por software en un FPGA.

Lograr el reconocimiento de los patrones de voz obtenidos, esto mediante el uso de las redes neuronales, que serán entrenadas para poder obtener sus parámetros característicos y así la evaluación de la red con las entradas dadas por las salidas del LPC.

Usar algoritmos implementados en lenguaje C para tres fines: la conversión de la señal de voz digitalizada por el CODEC (implementando una libreria), la extracción de características de las señales de voz que se almacenarán en memoria y el reconocimiento de palabras entre un grupo de patrones almacenados en memoria.

2.4. Limitaciones

El sistema a desarrollar en el presente trabajo de tesis está enmarcado bajo las siguientes limitaciones:

- Reconocimiento de palabras aisladas.
- Reconocimiento de palabras de un locutor único.
- Reconocimiento de palabras está definido por comparación contra patrones anteriormente almacenados en memoria.
- Reconocimiento de palabras está restringido a la cantidad de palabras que se pueda almacenar en memoria semiconductora de tarjeta de desarrollo.
- Técnica de reconocimiento de patrones usando el algoritmo de backpropagation para una red neuronal artificial de tres capas (capa de entrada, capa de salida y capa escondida).

En el siguiente capítulo se muestra la propuesta de solución para el problema planteado, enfocándose en la metodología de diseño a seguir para el tratamiento de la voz, desde su adquisición, codificación, y posterior reconocimiento.

CAPÍTULO 3
PROPUESTA DE SOLUCIÓN

3.1. Metodología de diseño

La señal de voz es captada con un micrófono convencional, que convierte las diferencias de presión del sonido en variaciones de voltaje. El micrófono está conectado a una tarjeta de desarrollo DE2-70, que cuenta con un códec de audio de 24 bits Wolfson WM8731, el cual convierte la tensión eléctrica generada por el micrófono en muestras digitalizadas, con una frecuencia de muestreo de 8 kHz y una representación de 16 bits por muestra para cada canal L y R. En la Figura 3.1 se ilustra el diagrama de bloques del sistema reconocedor de palabras

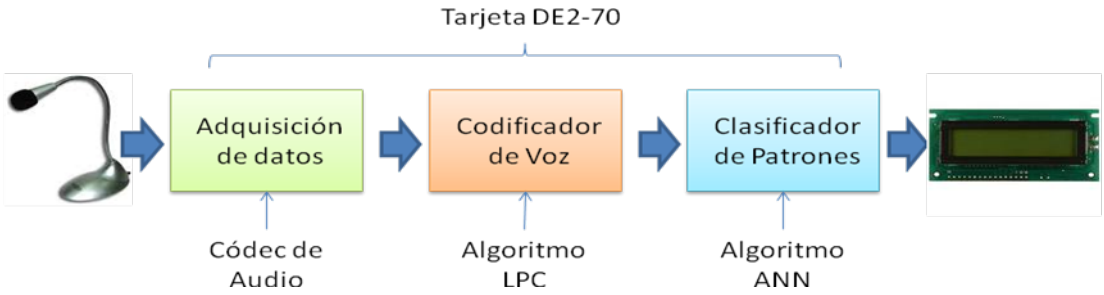


Figura 3.1. Diagrama de Bloques del Reconocedor de Palabras.

Para controlar los bloques de adquisición de datos, codificación LPC y reconocimiento usando la read neuronal artificial (ANN) se usa una máquina de estados, como se ilustra en la Figura 3.2. En un inicio, el sistema se encuentra en estado de reposo, esperando la indicación de inicio de grabación; al presionar un pulsador, el sistema pasa al estado de procesamiento donde primero se hace una grabación de 4000 muestras, luego se hace la codificación LPC y finalmente el reconocimiento usando la ANN con la muestra de la palabra reconocida, para luego regresar al estado de reposo. En este momento, se puede reproducir lo grabado, o grabar una nueva palabra y procesarla para reconocer a que palabra del vocabulario pertenece.

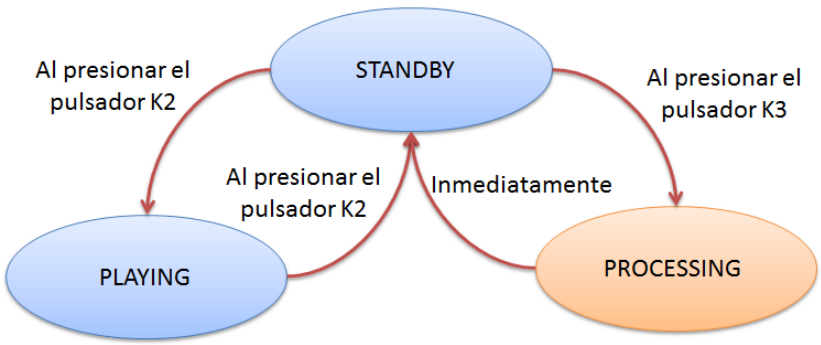


Figura 3.2. Diagrama de Estados del Reconocedor de Voz.

3.2. Adquisición de datos

Para la adquisición de datos se hace uso del Códec de audio proporcionado por la tarjeta de desarrollo y del micrófono como accesorio que será conectado a la entrada de micrófono.

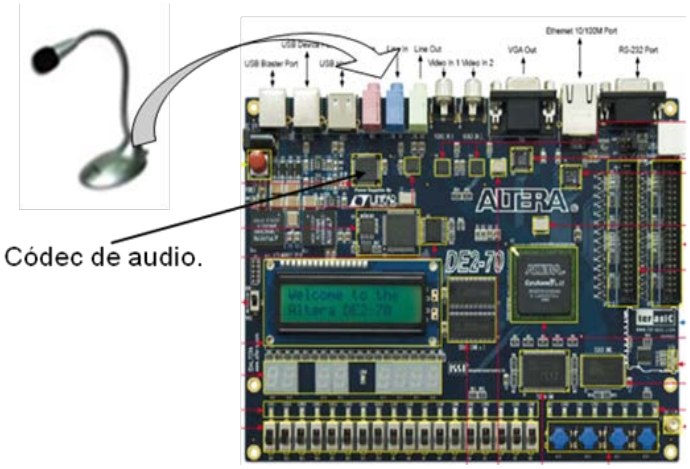


Figura 3.3. Diagrama de conexión para la adquisición de datos.

Un componente de software para el control del chip del audio, es embebido en el SoPC, a traves de una librería de propiedad intelectual (IP Library core). Dicho componente es desarrollado por Terasic Technologies Inc [13].

Componente de audio en SoPC Builder

El componente es diseñado para proveer un interfaz para el procesador Nios II, recibiendo audio del convertidor analógico a digital (ADC por sus siglas en inglés de Analog to Digital Converter) o enviando audio al conversor digital a analógico (DAC por sus siglas en inglés de Digital to Analog Converter). En la Figura 3.4 se ilustra el diagrama de bloques del sistema de audio, en el cual se observa el uso de colas tipo FIFO (First-In First Out) para la transferencia de los datos adquiridos.

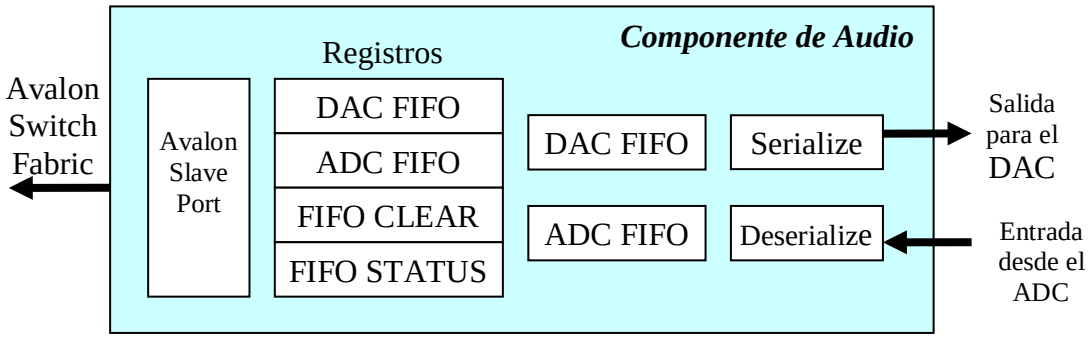


Figura 3.4. Componente de audio.

En el lado izquierdo de la Figura 3.4 se muestra el bloque “Avalon Slave Port” que establece la interfaz del componente de audio con el bus propietario que usa el sistema basado en el procesador Nios II, conocido como Avalon.

Configuración del componente de audio en SoPC Builder

Se debe en primer lugar, configurar el chip de audio a través del protocolo I2C. El chip de audio debe ser configurado de la manera siguiente :

- Modo Maestro.
- Justificado a la izquierda, de 16 bits.

Controlar el audio con la siguiente MACRO se define en el AUDIO_HAL.h

Nombre de la Macro	Función
AUDIO_DAC_FULL()	Verifica si el DAC-FIFO está lleno
AUDIO_DAC_WRITE(data)	Escribe 32-bits de datos hacia el DAC-FIFO. Nota. Llamar a AUDIO_DAC_FULL() para asegurarse que el FIFO no esté lleno antes de escribir datos en él.
AUDIO_ADC_EMPTY()	Verifica si el ADC-FIFO está vacio
AUDIO_ADC_READ()	Leer una palabra de 32-bits de dato desde el ADC-FIFO Nota. Llamar a AUDIO_ADC_EMPTY() para asegurarse que el FIFO no esté vacio antes de leer desde él.
AUDIO_FIFO_CLEAR()	Borrar ambos el ADC-FIFO y el DAC-FIFO
Nota. La palabra doble de 32-bits contiene una muestra izquierda de 16-bits y una muestra derecha de 16-bits. La muestra izquierda está en la parte más significativa y la muestra derecha está en la parte menos significativa de la palabra.	

Figura 3.5. Funciones para usar DAC y ADC.

Integración del componente de audio en SoPC Builder

En las Figuras 3.6 y 3.7 se ilustra las configuración del procesador Nios II embebido y la configuración del audio en el SoPC Builder.

En la Figura 3.6 se muestra una vista parcial del conexionado logico de los componentes del sistema basado en el procesador Nios II usando la herramienta de software SoPC Builder, donde se resalta el componente de audio.

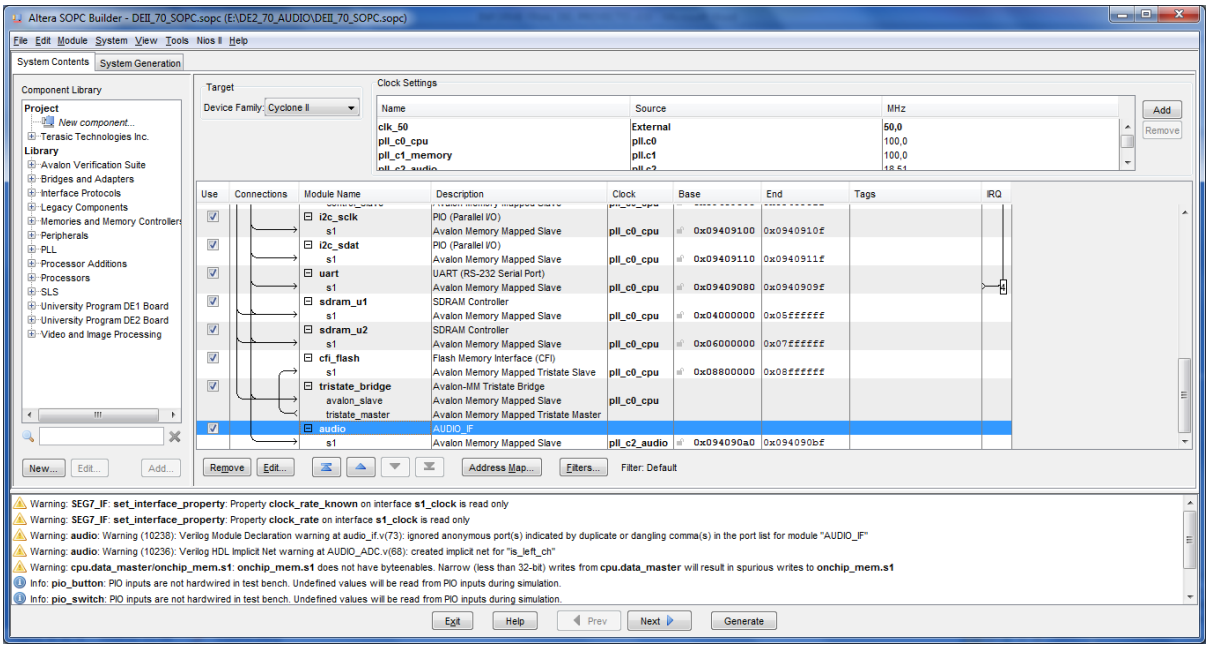


Figura 3.6. Conexionado en la herramienta SoPC Builder.

En la Figura 3.7 se muestra de manera particular el componente de audio que se instancia en la herramienta SoPC Builder. Una vez que todos los componentes de hardware (memorias, periféricos, librerias de propiedad intelectual, bloques hechos a la medida, etc) han sido instanciados en el SoPC Builder, se procede a generar el sistema en hardware, para luego proceder a desarrollar el programa en lenguaje C que se ejecutará en el sistema.

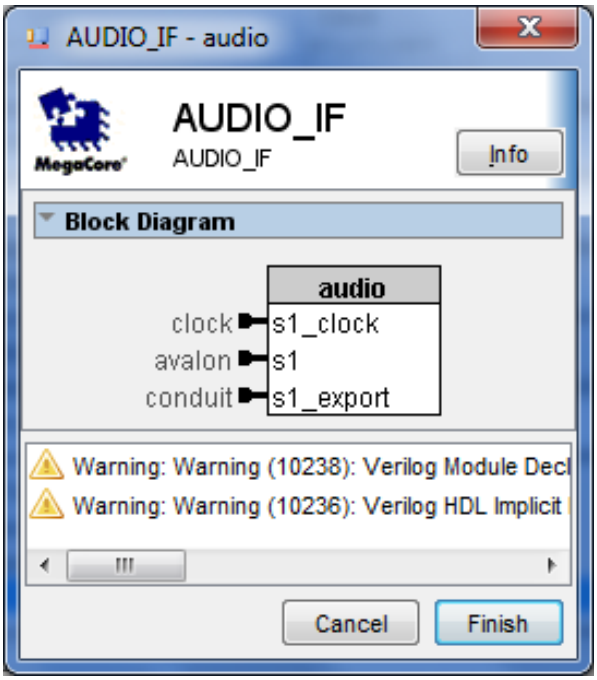
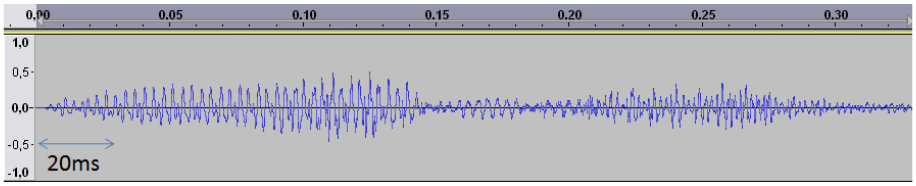


Figura 3.7. Añadiendo el bloque AUDIO_IF.

3.3. Codificación de la voz

Como el micrófono es de un solo canal (mono-oral), el códec de audio genera la misma representación en ambos canales, por lo que es necesario usar solo uno. Como la codificación LPC de la señal de voz se cumple para segmentos pequeños de voz, se fragmenta la palabra grabada cada T = 20 ms, que con la frecuencia de muestreo f_s = 8 kHz da un total de 160 muestras por trama, como se ilustra en la Figura 3.8. Dicha trama se codifica por LPC a M=10 parámetros por trama; como la cantidad de tramas para 0.6 s, de grabación efectiva, es de 600/20=30 tramas, entonces la cantidad de parámetros LPC que representan la grabación de voz es de 30(10)=300 parámetros.



fs=8KHz → N=160

M=10→ c=1/16

$X=[x(0)\ x(1)\ \ldots\ x(n)\ \ldots\ x(N-1)]$ → Bloque LPC (enC) → $A=[a(1)\ a(2)\ \ldots\ a(k)\ \ldots\ a(M)]$

M=10→ c=M/N=1/16

Figura 3.8. Representación de la Codificación LPC de la voz.

3.4. Reconocimiento de voz

Los 300 parámetros LPC que representan a la señal de voz (palabra grabada durante 1.0 seg) serán las plantillas almacenadas en memoria que constituyen el vocabulario del sistema, las neuronas de entrada de la red que se encargarán de hacer el reconocimiento, previo entrenamiento con varias grabaciones de las palabras del vocabulario. Para el presente trabajo, la red tiene 300 neuronas de entrada, 10 neuronas ocultas y 3 neuronas de

salida, en vista de que puede reconocer 3 palabras. Sin embargo, el consumo de memoria es de cerca del 10%, entonces el vocabulario se puede extender a muchas más palabras.

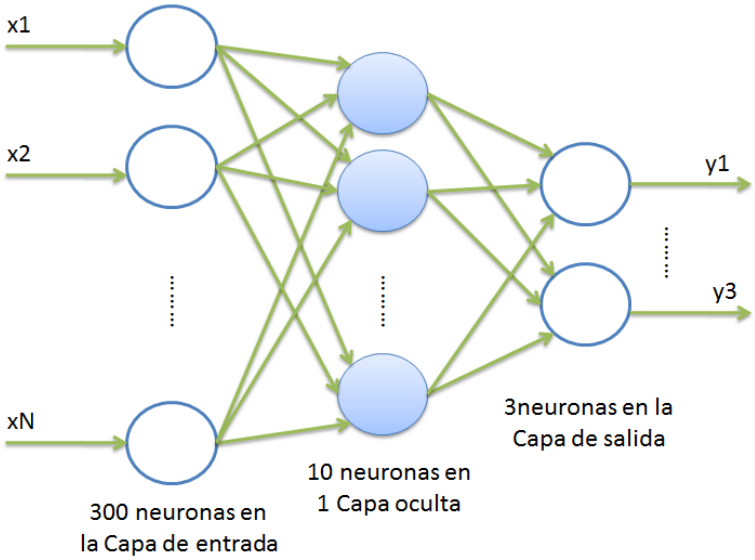


Figura 3.9. Estructura de la Red Neuronal usada en el Reconocedor.

En el presente capítulo se ha mostrado la propuesta de solución para el problema planteado, enfocándose en la metodología de diseño a seguir para el tratamiento de la voz, desde su adquisición, codificación, y posterior reconocimiento. En el siguiente capítulo se mostrarán los resultados obtenidos de la simulación del sistema utilizando la herramienta de software MATLAB.

CAPÍTULO 4
SIMULACIÓN DEL ALGORITMO DE RECONOCIMIENTO DE PALABRAS

En el presente capítulo se muestran los resultados obtenidos de la simulación del sistema reconocedor de palabras utilizando la herramienta de software MATLAB. Este paso es importante, previo al desarrollo del sistema en hardware, a fin de validar los resultados de los algoritmos empleados.

4.1. Introducción

Con la finalidad de realizar las simulaciones computacionales, en la Figura 4.1, se presenta un escenario de pruebas, el cual consta de señales de voz generadas por un locutor, definidas por las palabras: “do”, “re” y “mi”. Referidas palabras de entradas son codificadas por el algoritmo LPC, el cual genera 300 coeficientes, R(i), para cada palabra respectivamente. A continuación, se hace uso de una red neuronal artificial (ANN) con la finalidad de implementar un algoritmo de aprendizaje – backpropagation, cuyas patrones entrada-salida son definidos por los coeficientes LPC y una codificación binaria: “100”, “010” y “001”, respectivamente.

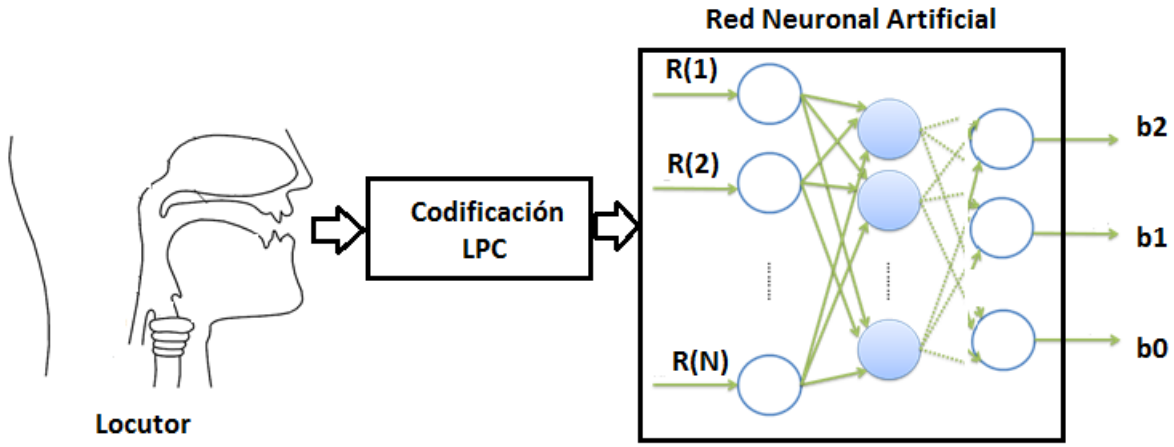


Figura 4.1. Escenario de pruebas para las simulaciones computacionales

La Tabla 4.1, contiene las codificaciones entrada y salida para la ANN.

Tabla 4. 1 Codificación Entrada-Salida de la RNA

Palabra	Codificación LPC	Codificacion Binaria de Salida
do	$R_1(1)$……$R_1(300)$	100
re	$R_2(1)$……$R_2(300)$	010
mi	$R_3(1)$……$R_3(300)$	001

Así también, se debe considerar que, para la generación de las palabras a partir del locutor, se tiene una ventana de grabación de 1 s, con frecuencia de muestreo de 8 kHz, con codificación PCM de 16 bits por canal L-R estereo.

4.2. Construcción de la Base de Datos para Simulación

Para el escenario de simulación propuesto en la Figura 4.1, se ha realizado la construcción de una base de datos compuesta por 90 grabaciones, de los cuales, 30 pertenecen a la palabra “do”, 30 a la palabra “re” y 30 a la palabra “mi”.

Para cada subconjunto (palabra), se ha divido en 20 grabaciones para el proceso se entrenamiento de la Red Neuronal Artificial (ANN) y 10 grabaciones para el proceso de prueba y verificación, como se ilustra en la Figura 4.2.

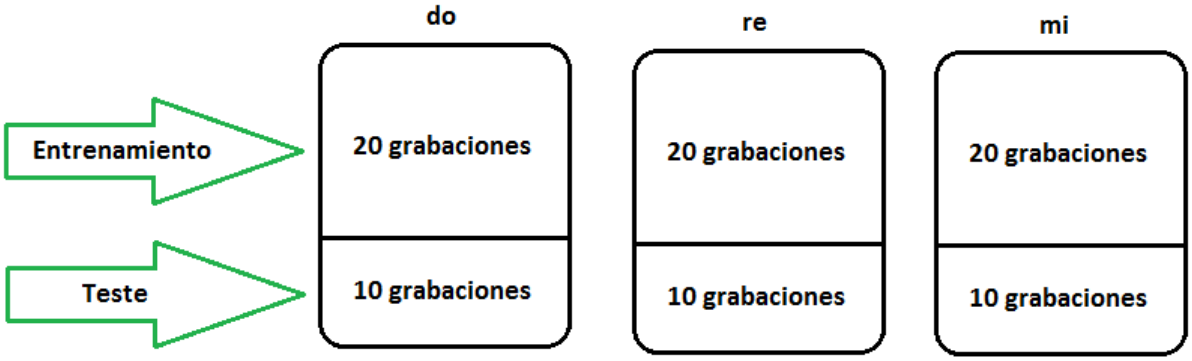


Figura 4.2. Construcción de la Base de Datos para Simulación

A partir de las grabaciones realizadas, para cada 20 ms de grabación, se obtienen 10 parametros LPC, consecuentemente, para 1 s de grabación se tienen 500 parametros LPC. Sin embargo, cabe resaltar, que se aplica un procedimiento basado en Energia de la Señal para cancelar los momentos de silencio, considerandose finalmente 600 ms de ventana que corresponden a 300 parametros LPC.

4.3. Patrones Entrada-Salida de la ANN

A partir de la base de datos generada por el locutor, se construye las matrices correspondientes a los patrones de entrada-salida de la ANN. Para este proposito, a continuación se presenta un ejemplo, de generación de las matrices de patrones entrada-salida para tres palabras “do”, “re” y “mi”

$$\begin{matrix} "mi" & "re" & "do" \end{matrix} \qquad Codificación$$

$$\begin{bmatrix} R_3(1) & R_2(1) & R_1(1) \\ R_3(1) & R_2(1) & R_1(2) \\ \dots & & \\ R_3(300) & R_2(300) & R_1(300) \end{bmatrix} \Leftrightarrow \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Como todo proceso de inteligencia artificial el sistema así formado aprenderá de la experiencia que le daremos con las voces grabadas anteriormente, debiendo verificarse que catalogue a todas las grabaciones como reconocidas de uno u otro grupo de nuestro diccionario, esto asegura que pequeñas variaciones de nuestros comandos puedan clasificarse bien entre los grupos que existan.

Para verificar que nuestro entrenamiento resultó exitoso, el parámetro “Performance” debe tender a cero, de no cumplirse debemos repetir nuestro entrenamiento o en caso extremo grabar de nuevo las voces, como se ilustra en la Figura 4.3.

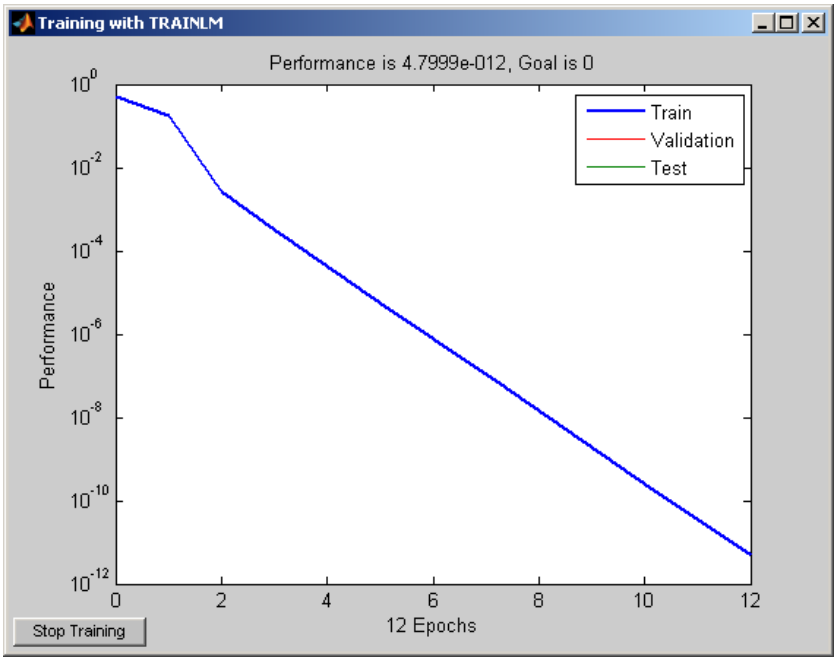


Figura 4.3. Performance de la red neuronal

4.4. Resultados de Simulación

Para la simulación del sistema reconocedor de palabras se implementaron los programas en MATLAB, cuyo diagrama de flujo se muestra en la figura 4.4. El listado de los programas desarrollados en MATLAB es presentado en el Anexo B.

En la Tabla 4.2, se tienen los resultados de testeo del Grupo 1, para las palabras “do”, “re” y “mi”, y cuya codificacion binaria de salida corresponde a “100”, “010” y “001”, respectivamente. A partir de los resultados de esta simulación, se tiene un porcentaje de acierto del 70% para la palabra “do”, 70% para la palabra “re” y 100% para la palabra “mi”.

Debe mencionarse, que como toda red neuronal artificial, ésta debe ser entrenada con una cantidad suficiente de patrones, a fin que los pesos (conexiones entre neuronas entre capas sucesivas) estén ajustados de manera que en la presencia de un patrón ligeramente distorsionado pueda ser reconocido. Igualmente, para el algoritmo de backpropagation es importante el límite inferior de error a satisfacer y la cantidad máxima de iteraciones del algoritmo, para tomar la decisión que se ha reconocido un patrón dentro de los que la red pueda recordar o que el patrón presentado no está dentro de los que la red pueda recordar, cuya información esta básicamente en los pesos de la red. Para el presente trabajo se ha trabajado con tres palabras aisladas y una red backpropagation de tres capas, cuyo desempeño de acuerdo a las Tablas 4.2 y 4.3 es bastante aceptable.

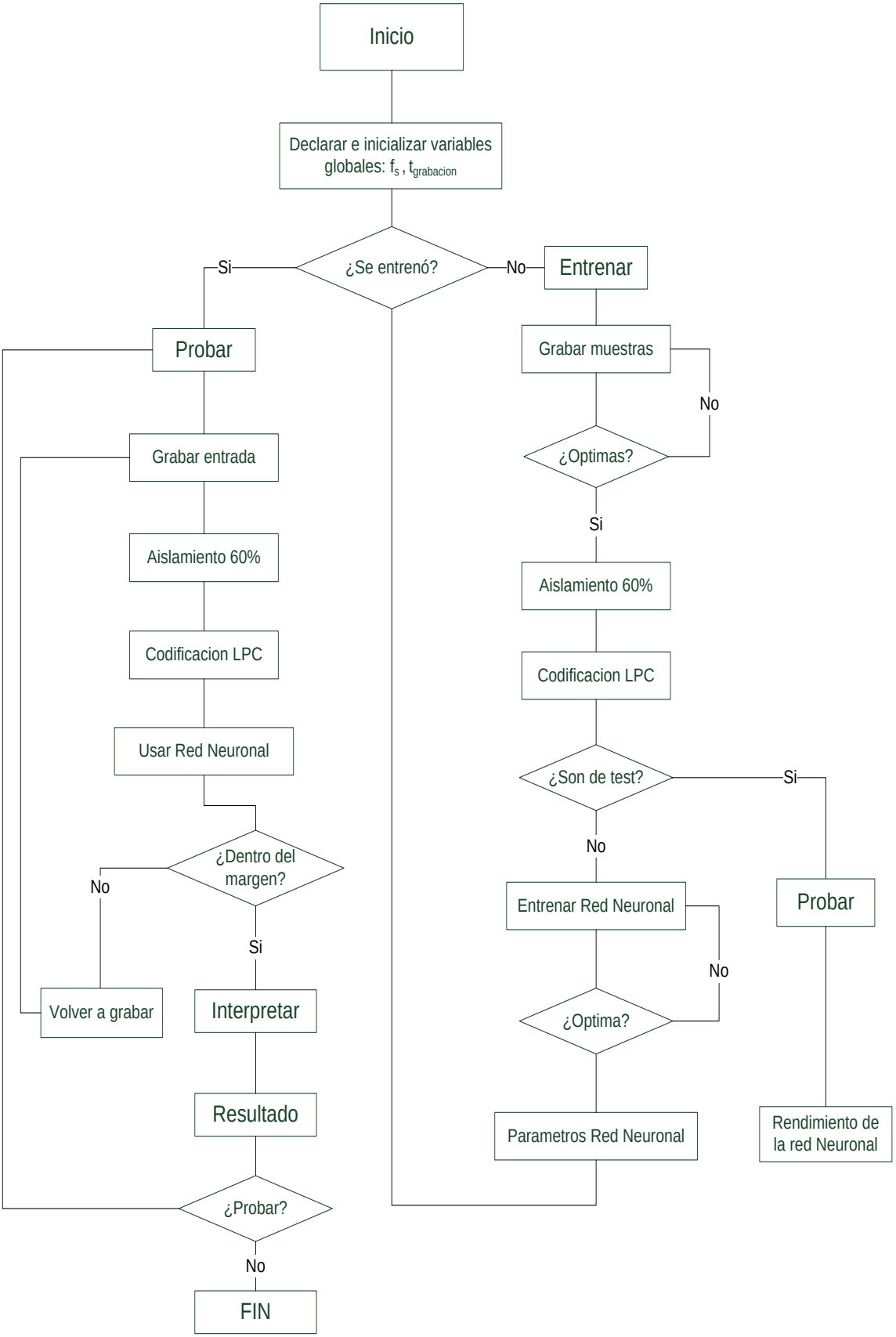


Figura 4.4. Diagrama de flujo de simulación del reconocedor de palabras en MATLAB

Tabla 4. 2 Resultados de Prueba – Grupo 1

Prueba1	Salida1	Salida2	Salida3	Acierto
DO_1	1.0000	0.0000	0.0000	OK
DO_2	1.0000	0.0000	0.0001	OK
DO_3	1.0000	0.0000	0.0001	OK
DO_4	1.0000	0.0001	0.0000	OK
DO_5	0.0000	0.0000	0.1257	ERROR
DO_6	1.0000	0.0000	0.0826	OK
DO_7	0.0000	0.0000	0.9995	ERROR
DO_8	0.0000	0.1359	0.0000	ERROR
DO_9	0.9944	0.0179	0.0000	OK
DO_{10}	0.9980	0.0062	0.0000	OK
RE_1	0.0000	1.0000	0.0000	OK
RE_2	0.0000	1.0000	0.0000	OK
RE_3	0.0000	1.0000	0.0000	OK
RE_4	0.0044	0.9992	0.0000	OK
RE_5	0.0000	1.0000	0.0000	OK
RE_6	0.9951	0.0162	0.0000	ERROR
RE_7	0.0480	0.9903	0.0000	OK
RE_8	0.0002	1.0000	0.0000	OK
RE_9	0.9948	0.0167	0.0000	ERROR
RE_{10}	0.8267	0.4584	0.0000	ERROR
MI_1	0.0000	0.0000	1.0000	OK
MI_2	0.0000	0.0000	1.0000	OK
MI_3	0.0000	0.0000	1.0000	OK
MI_4	0.0000	0.0000	1.0000	OK
MI_5	0.0000	0.0000	1.0000	OK
MI_6	0.0000	0.0000	1.0000	OK
MI_7	0.0000	0.0000	1.0000	OK
MI_8	0.0000	0.0000	1.0000	OK
MI_9	0.0000	0.0000	1.0000	OK
MI_{10}	0.0000	0.0000	1.0000	OK

En seguida, se realizó una segunda prueba, con un grupo 2 de palabras bajo las misma condiciones, cuyos resultados se muestran en la Tabla 4.2. A partir de los resultados de esta simulación, se tiene un porcentaje de acierto del 80% para la palabra “do”, 90% para la palabra “re” y 70% para la palabra “mi”.

Tabla 4. 3 Resultados de Prueba – Grupo 2

Prueba2	Salida1	Salida2	Salida3	Acierto
DO_1	0.9961	0.0000	0.1364	OK
DO_2	0.0000	0.0000	0.8980	ERROR
DO_3	1.0000	0.0000	0.0000	OK
DO_4	1.0000	0.0000	0.4116	OK
DO_5	1.0000	0.0000	1.0000	ERROR
DO_6	1.0000	0.0000	0.0000	OK
DO_7	1.0000	0.0000	0.0001	OK
DO_8	1.0000	0.0000	0.0000	OK
DO_9	1.0000	0.0000	0.0001	OK
DO_{10}	1.0000	0.0000	0.0001	OK
RE_1	0.0000	1.0000	0.0000	OK
RE_2	0.0000	1.0000	0.0000	OK
RE_3	0.0000	1.0000	0.0000	OK
RE_4	0.0000	1.0000	0.0000	OK
RE_5	0.0000	1.0000	0.0000	OK
RE_6	0.9271	0.2319	0.0000	ERROR
RE_7	0.0000	1.0000	0.0000	OK
RE_8	0.0178	0.9966	0.0000	OK
RE_9	0.0000	1.0000	0.0000	OK
RE_{10}	0.0003	0.9999	0.0000	OK
MI_1	0.0000	1.0000	0.0000	ERROR
MI_2	0.0000	0.0000	1.0000	OK
MI_3	0.0000	0.0000	1.0000	OK
MI_4	0.0000	1.0000	0.0001	ERROR
MI_5	0.0000	0.0000	1.0000	OK
MI_6	0.0000	0.0000	1.0000	OK
MI_7	0.0000	1.0000	0.0000	ERROR
MI_8	0.0000	0.0000	1.0000	OK
MI_9	0.0000	0.0000	1.0000	OK
MI_{10}	0.0000	0.0001	1.0000	OK

A partir de los resultados de simulación, se concluye que el sistema presenta un porcentaje de acierto de aproximadamente 80%, observándose que los errores en el reconocimiento son productos basicamente de los niveles de ruidos y la dependencia de la pronunciación del locutor al construir la base de datos.

En el presente capítulo se han mostrado los resultados obtenidos de la simulación del sistema reconocedor de palabras utilizando la herramienta de software MATLAB, obteniéndose resultados bastante aceptables. En el siguiente capítulo se va a detallar la implementacion y pruebas del sistema desarrollado en FPGA.

CAPÍTULO 5
IMPLEMENTACIÓN Y PRUEBAS DEL SISTEMA EN UN FPGA

5.1. Introducción

La implementación del sistema de reconocimiento de palabras propuesto en el presente trabajo se realizó en una tarjeta de desarrollo DE2-70 fabricada por Terasic Technologies Inc. [17], que contiene un FPGA Cyclone II 2C70 fabricado por Altera [1]. Además del FPGA, se usa el códec de audio, la memoria SRAM, interruptores pulsadores y la pantalla LCD de la tarjeta DE2-70. La figura 5.1 muestra la configuración de hardware del sistema, donde el micrófono (MIC) es el dispositivo de entrada y la pantalla LCD es el dispositivo de salida.

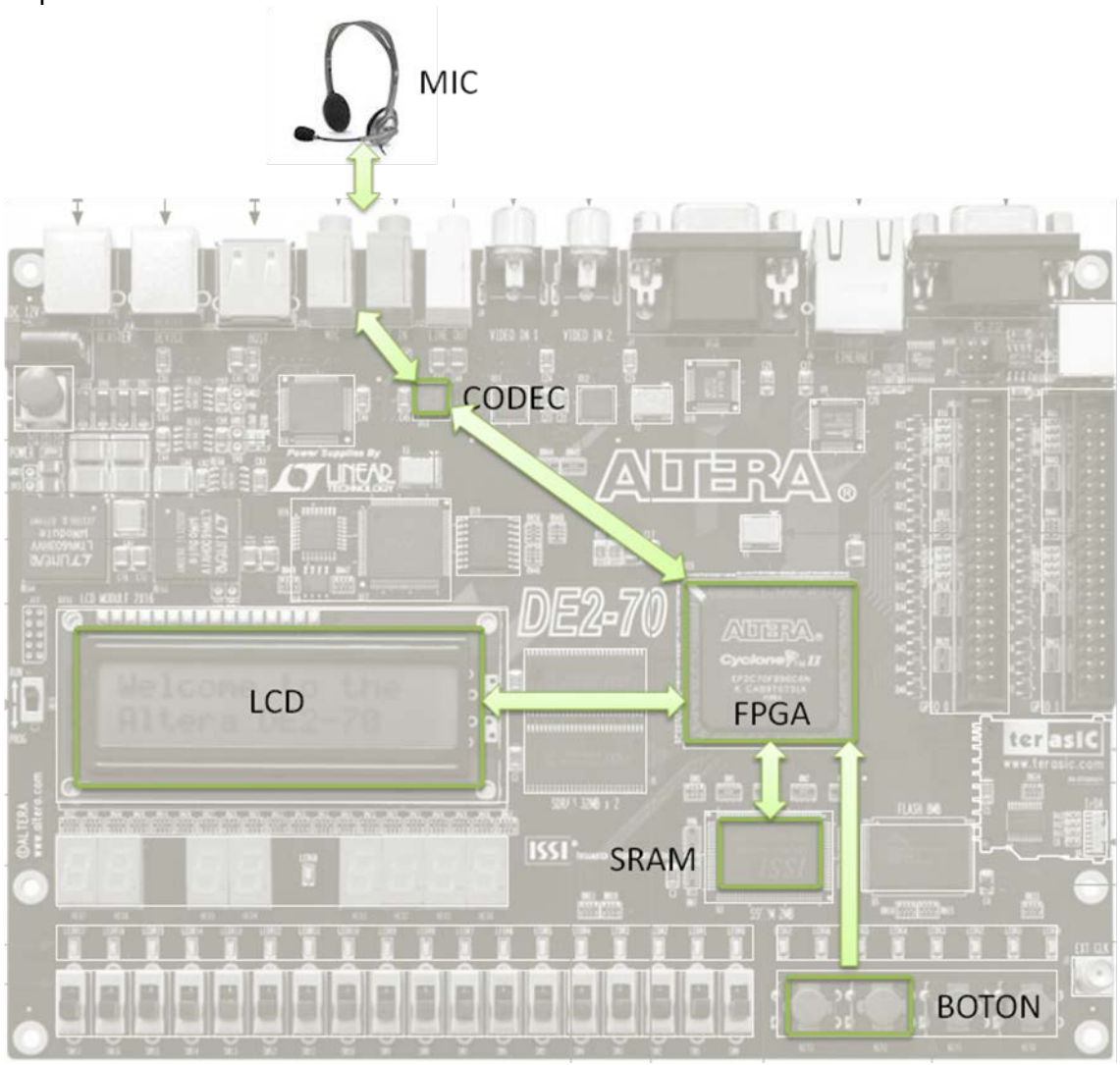


Figura 5.1: Configuración de hardware del sistema propuesto.

El mecanismo de aprendizaje del sistema es off-line [19], es decir, se realiza previamente a la utilización del mismo. Los parámetros de entrenamiento del sistema están almacenados en la memoria de la tarjeta. Para iniciar la grabación de una nueva palabra con el códec de audio se presiona el botón KEY3; el sistema está programado para una grabación de 0,6seg. El procesador Nios II contenido en el FPGA realiza el procesamiento de la voz grabada y muestra en la pantalla LCD la palabra reconocida, ya sea la primera, segunda, tercera o ninguna de ellas.

Con la finalidad de realizar la adquisición de la señal de voz a través de la entrada de micrófono de la plataforma de desarrollo FPGA, se realizó la implementación de una librería en C, basada en un código desarrollado por el fabricante Terasic; dicha librería permite adquirir y transferir por el bus de datos las informaciones de la señal de voz para el procesador Nios II y procesarlo a traves del algoritmo LPC y red neuronal artificial (RNA).

En la Figura 5.2, se ilustra el diagrama de flujo del programa que corresponde a la referida librería (ver codificación en el Anexo D). Esta codificación, utiliza la declaración de variables globales y la verificación del codec de audio. En seguida, se tienen tres estados posibles del sistema: reposo, grabación/procesamiento y reproducción.

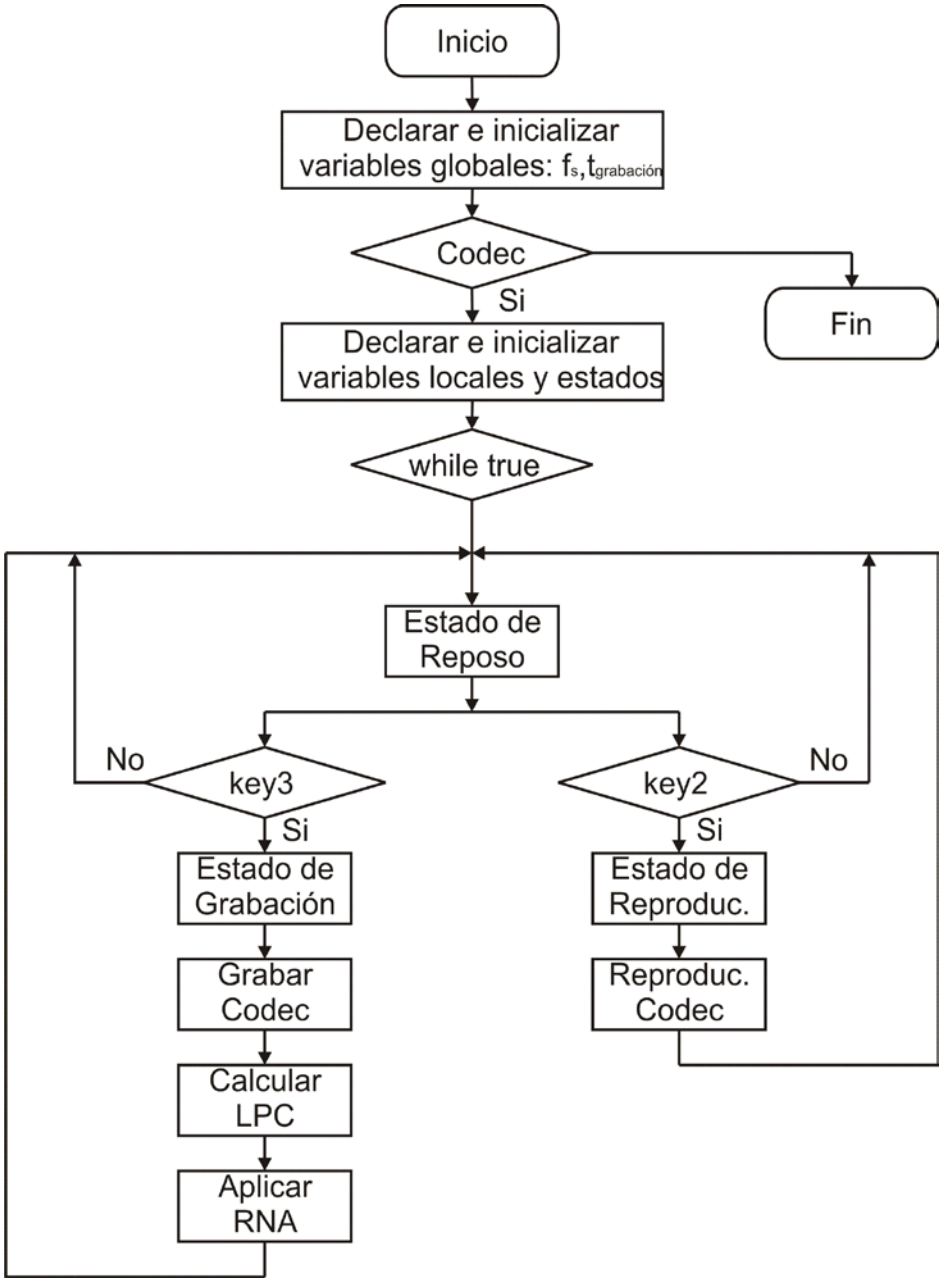


Figura 5.2. Diagrama de flujo de la librería de reconocimiento de palabras.

El estado en reposo, espera una interrupción de hardware (pulsador), el cual dará inicio al estado de grabación/procesamiento, en caso se habilite el pulsador 1.

El estado de grabación/procesamiento, inicia la grabación utilizando el codec, y el procesamiento, realiza la codificación de los parametros LPC, y el reconocimiento de la palabra mediante la Red Neuronal Artificial (ANN).

El estado de reproducción, es activado en caso en el estado de resposo se habilite el pulsador 2. Este estado reproduce la palabra con el codec de audio y reinicia nuevamente el sistema.

5.2. Adquisición de voz con el Códec

El códec de audio Wolfson usado es de 24 bits y 2 canales, por tanto, representa cada canal de audio con 12 bits. El procesador embebido Nios II es de 32 bits, razón por la cual emplea 16 bits por canal para representar una señal de audio. Además, la representación de números negativos en el códec es por codificación en complemento a 2. Se hizo un análisis para hallar la fórmula de conversión de binario a real, lo cual se presenta en la Tabla 5.1.

Tabla 5. 1 Representaciones de las muestras de señal de voz obtenidas con el CODEC.

Complemento a 2	Entero (con signo)	Entero (sin signo)	Real
0000011111111111	4095	4095	1.0
0000011111111110	4094	4094	
…	…	…	…
0000000000000001	1	1	
0000000000000000	0	0	0.0
1111111111111111	-1	65535	
1111111111111110	-2	x	y
…	…	…	…
1111100000000001	-4095	32769	
1111100000000000	-4096	32768	-1.0

La fórmula de conversión de notación con complemento a 2 hacia número real sería:

$$y = \begin{cases} \dfrac{x}{2^{11}-1}, 0 \le x < 2^{11} \\ \dfrac{x-2^{16}}{2^{11}}, 2^{11} \le x < 2^{16} \end{cases} \quad (5.1)$$

Donde “x” es la representación binaria de la muestra de voz en el códec de audio, “y” es el equivalente en número real.

Experimentalmente, el códec de audio digitaliza con un desfazaje en amplitud. Por ejemplo, cuando se graba el ruido del medio, la representación promedio del códec de audio

es de 64060, cuando debería variar entre 0 y 65535 como muestra la tabla en la representación entera sin signo. Asímismo, la representación binaria positiva máxima es de 5000 aproximadamente, cuando debería ser 4095. Se ha modificado la ecuación para compensar los errores mencionados, quedando:

$$y = \begin{cases} \dfrac{x+1476}{8191}, x < 3500 \\ \dfrac{x-64060}{8192}, 60400 \le x \end{cases} \tag{5.2}$$

5.3. Base de datos

Para probar el sistema se usó la misma base de datos usada en la simulación, con la finalidad de comparar los resultados de la simulación con los resultados del sistema propuesto implementado en un FPGA. Se usaron además muestras adicionales grabadas directamente con el códec de audio de la tarjeta DE2-70, con el fin de hallar el rendimiento real del sistema. La figura 5.3 muestra una representación de la base de datos usada para probar la implementación del sistema en un FPGA; en donde se observa las 10 grabaciones adicionales (marcadas en negro) realizadas con el códec de audio.

Figura 5.3: Base de datos para pruebas del Sistema

La figura 5.4 muestra una grabación en MATLAB de la base de datos de prueba, cuya duración es de 1.0 seg. Recordar que en la simulación se ha considerado un aislamiento de 0.6 seg para descartar la porción de silencio inicial, sin embargo, en la implementación se realiza directamente una grabación de 0.6 seg, después de presionar el botón de inicio de grabación.

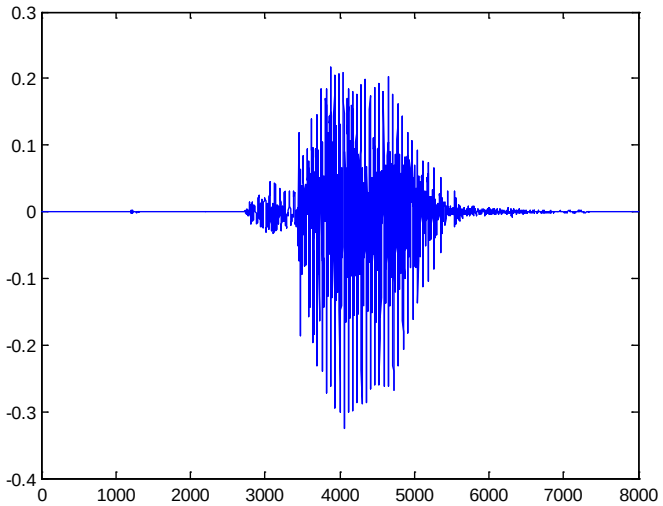


Figura 5.4: Grabación de prueba de la palabra DO con MATLAB

Con el factor de corrección de la ecuación (5.2) hallado para el códec de audio con el fin de convertir las muestras de grabación de voz de binario a número real, se obtienen formas de onda muy parecidas a las que obtienen con una tarjeta de sonido de computadora. La figura 5.5 muestra la gráfica de una grabación hecha con el códec de audio de la tarjeta DE2-70.

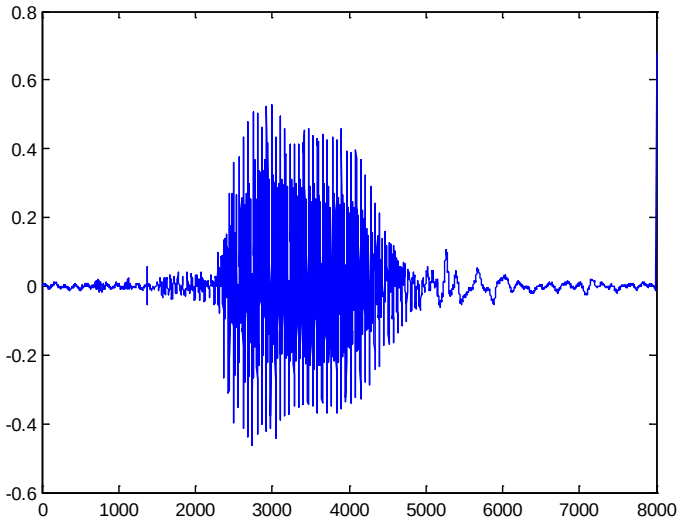


Figura 5.5: Grabación de la palabra DO con el códec de audio

5.4. Reconocimiento de voz

Entrenamiento de la Red en MATLAB

En vista que los procesamientos realizados en MATLAB y en el FPGA dan resultados parecidos, se realizó por simplicidad el entrenamiento del sistema con grabaciones de MATLAB, es decir, usando la tarjeta de sonido de una computadora. Se encontró que el rendimiento del sistema implementado en un FPGA es parecido al hallado por simulaciones en MATLAB, aún cuando los parámetros de entrenamiento del sistema se hallaron con grabaciones de MATLAB y no con el códec de audio que representa la situación más real.

La red neuronal artificial propuesta para el sistema de reconocimiento de voz es un perceptrón multicapa con función de transferencia sigmoidal, que se podría considerar también como una red de propagación hacia adelante. La red consta de una capa de entrada con N=300 neuronas, una capa oculta con H=10 neuronas, y una capa de salida con M=3 neuronas. La entrada de la red es la codificación LPC de una señal de voz digitalizada, y las salidas son indicadores del reconocimiento de patrones.

Para entrenar la red se han desarrollado programas en MATLAB denominados *Entrenamiento_1, Entrenamiento_2, Entrenamiento_3*. El código se encuentra en el Anexo B.

El programa *incorporar.m* genera la arquitectura de la red neuronal, realiza el entrenamiento y guarda los valores calculados de los pesos de la red en archivos de datos para que puedan ser usados en la implementación en hardware de la red neuronal artificial. El código se encuentra en el Anexo B.

Estrictamente, la grabación de voz con el códec de audio es diferente a la realizada en PC con MATLAB. Por tanto, el entrenamiento de la red neuronal se debería realizar en MATLAB, con muestras de audio tomadas de la misma tarjeta. Esto es posible modificando el

programa en Nios II para que imprima los valores muestreados con el códec. La voz grabada se convierte a archivo wav en MATLAB. Se probó un entrenamiento con 20 grabaciones de las palabras “do”, “re” y “mi”, entonadas en su nota musical respectiva, con resultados parecidos pero con mayor trabajo para realizar el entrenamiento. Se encontró que el reconocedor de palabras es dependiente de la frecuencia de la voz.

Implementación del Sistema en lenguaje C

Para la implementación de la codificación LPC en C se opto por resolver la ecuación (1.26) mediante el método de Levinson Durbin debido a su menor coste computacional. Se cargan las muestras por trama, se calcula la autocorrelación de las muestras, luego se implementa el método de Levinson Durbin, finalmente se guardan los coeficientes LPC respectivos. Los códigos se encuentran en el Anexo C.

La arquitectura de red neuronal planteada tiene una función de transferencia equivalente dada por la siguiente operación de matrices:

$$Out = \log sig(netLW * \tan sig(netIW * In + netB1) + netB2) \qquad (5.3)$$

Utilizando el entorno integrado Nios II IDE se implementó en C las funciones de acción de la red neuronal artificial, las cuales fueron añadidas junto con la implementación de la codificación LPC, al programa del demo de audio de la tarjeta DE2-70, que fue la base desde donde se implementó el trabajo.

La función de transferencia fue dividida para su implementación en C, en cuatro operaciones matriciales a saber: primerMultip(), primerSumop(), segundoMultip() y segundoSumop(). Además se ha comentado parte del código orientado a la depuración, por ejemplo la impresión de los cálculos parciales. Los códigos se encuentran en el Anexo C.

5.5. Pruebas del sistema de reconocimiento de voz

Las funciones usadas en la adquisición de voz y procesamiento han sido empaquetadas en el archivo voice.c que contiene a lpc_ann.c y otros, de modo que la función main.c realiza la grabación y procesamiento de la voz usando solo la función record_voice() cuyos parámetros son la frecuencia de muestreo y el tiempo de grabación. Los códigos C completos se encuentran en el Anexo D.

A continuación, usando grabaciones de MATLAB de la base de datos, se realizarán comparaciones de valores hallados con MATLAB y el FPGA. Una comparación entre los parámetros LPC hallados en MATLAB y en el FPGA se muestra en la Tabla 5.2, para los primeros 20 ms de la palabra “do” grabada con MATLAB. Se observa que la precisión obtenida con el procesador Nios II embebido en el FPGA es del orden de 1e-4, es decir, la diferencia es de un diez-milécimo como máximo.

Tabla 5. 2 Comparación de parámetros LPC obtenidos mediante MATLAB y FPGA

CODEC-FPGA	CODEC-MATLAB
-0.4353	-0.4353
-0.6184	-0.6184
-0.0648	-0.0649
0.0820	0.0820
-0.0398	-0.0398
-0.0246	-0.0246
-0.0331	-0.0330
0.4344	0.4344
-0.0116	-0.0116
-0.1629	-0.1629

Para visualizar mejor la similitud en los cálculos, se hallaro n los parámetros LPC de toda la grabación y se grafican en las figuras 5.6 y 5.7, donde se observan los 300 parámetros LPC hallados con MATLAB (primera gráfica, datos en color azul) y con el códec (gráfica inferior, datos en color verde). Ambas gráficas corresponden a una grabación de la palabra “mi” realizada con MATLAB.

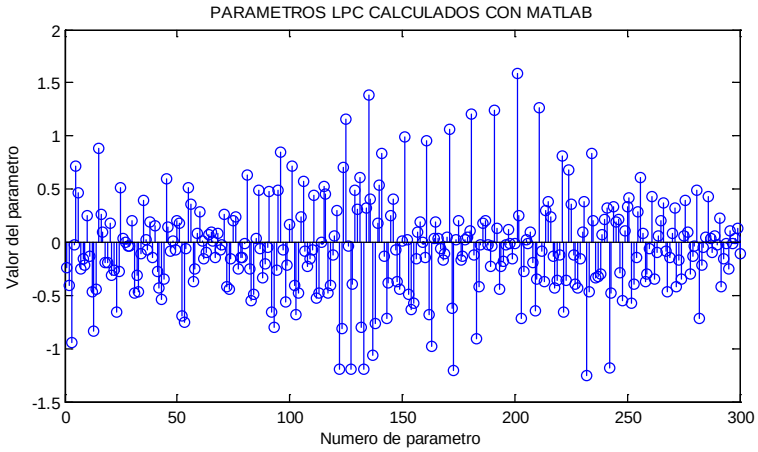


Figura 5.6: Parámetros LPC calculados con MATLAB

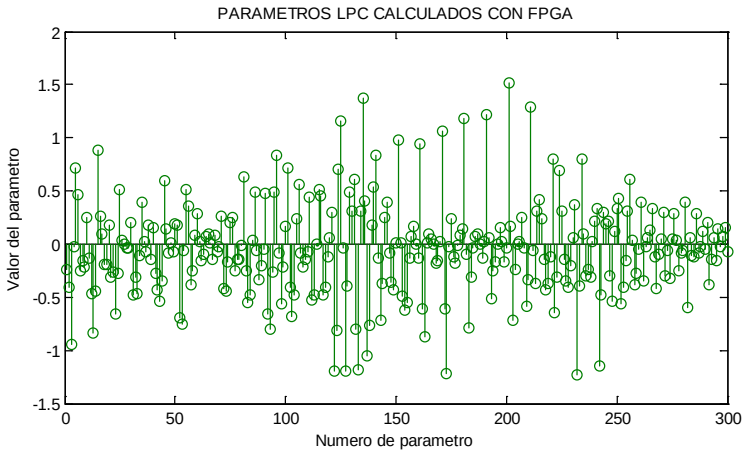


Figura 5.7: Parámetros LPC calculados con el FPGA

La figura 5.8 muestra las 2 gráficas anteriores trazadas de manera continua, donde se observa que los valores son muy cercanos (las gráficas casi se superponen). La ligera diferencia en los últimos términos se debe a la propagación del error por precisión en el cálculo, al ser Nios II un procesador que opera en punto fijo y no flotante.

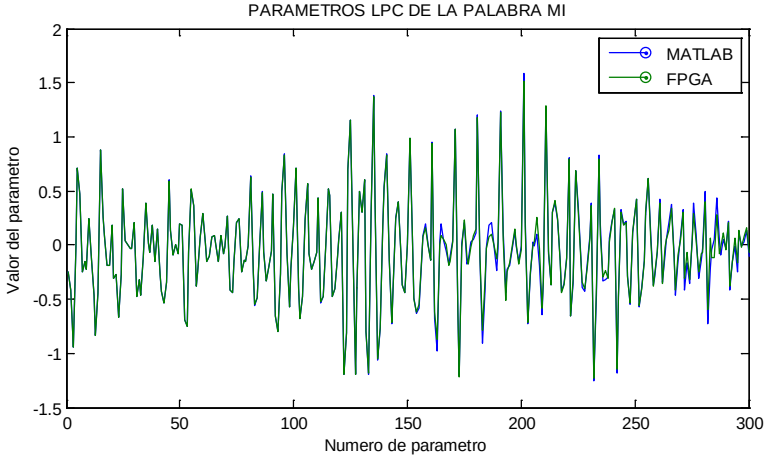


Figura 5.8: Comparación de parámetros LPC calculados con MATLAB y FPGA

Las pruebas del sistema de reconocimiento de palabras habladas se clasificaron en 2 etapas: la primera donde se ingresan grabaciones realizadas con MATLAB, y la segunda donde las grabaciones ingresan directamente a la tarjeta por un micrófono. Se han hecho pruebas con las 30 palabras grabadas con el códec y usadas en la simulación, y con 30 palabras adicionales grabadas con el códec de audio de la tarjeta.

La primera etapa de pruebas consistía en ingresar las palabras grabadas con MATLAB y usadas en la simulación, en la memoria SRAM de la tarjeta para que el procesador embebido en el FPGA realice el proceso de reconocimiento de voz. Los resultados fueron exactamente iguales a los de la simulación, a excepción de un caso donde la grabación contenía ceros que originaban indeterminación en el cálculo con el FPGA.

La tabla 5.3 muestra 5 ejemplos de salidas del sistema en MATLAB (tabla superior) y con el FPGA (tabla inferior), correspondientes a grabaciones de prueba de la palabra mi realizadas con MATLAB. En la última columna se muestra el total de aciertos y la eficiencia hallada con MATLAB y el FPGA. Los resultados son los mismos, dado que provienen de una misma grabación de voz.

Tabla 5. 3 Salidas del sistema ante grabaciones con MATLAB

Grabación	3e_30	3e_29	3e_28	3e_27	3e_26	Total
DO	0.000000	0.000000	0.000000	0.000000	0.000000	7/10
RE	0.000065	0.000002	0.000000	1.000000	0.000003	7/10
MI	1.000000	0.999994	0.999998	0.000003	0.999992	10/10
FPGA	ok	ok	ok	error	ok	80%
DO	0.0000	0.0000	0.0000	0.0000	0.0000	7/10
RE	0.0001	0.0000	0.0000	1.0000	0.0000	7/10
MI	1.0000	1.0000	1.0000	0.0000	1.0000	10/10
MATLAB	ok	ok	ok	error	ok	80%

La segunda etapa de pruebas consistió en grabar la palabra directamente con el códec de audio de la tarjeta, y realizar el procesamiento con parámetros de entrenamiento hallados en MATLAB. La tabla 5.4 muestra 5 ejemplos de las salidas del sistema ante grabaciones realizadas con el códec de audio; en cada columna se encuentra el grado de acierto para las 3 posibilidades, y en la última columna se encuentra el número de aciertos para cada palabra y el rendimiento del sistema implementado en FPGA, para una prueba 1 (tabla superior) y una prueba 2 (tabla inferior). El rendimiento o porcentaje de aciertos promedio encontrado es de 80% aproximadamente.

Tabla 5. 4 Salidas del sistema ante grabaciones con CODEC

Grabación	DO1	DO2	DO3	DO4	DO5	Total
DO	1.000000	0.000006	0.996923	1.000000	1.000000	8/10
RE	0.000000	0.999999	0.010960	0.000000	0.000000	8/10
MI	0.000001	0.000000	0.000000	0.000049	0.000051	5/10
Prueba1	ok	error	ok	ok	ok	70%
DO	1.000000	0.997602	0.976301	0.999999	0.999296	10/10
RE	0.000000	0.007646	0.080589	0.000002	0.002064	10/10
MI	0.000055	0.000000	0.000000	0.000000	0.000000	7/10
Prueba2	ok	ok	ok	ok	ok	90%

5.6. Resultados de la implementación del sistema

Al ejecutar el programa incorporar.m en MATLAB, se entrena la red con las grabaciones de las 3 palabras realizadas externamente o dentro de MATLAB, el programa genera las matrices de pesos y umbrales de la red y éstos se copian en el software del sistema para que reconozca las palabras con las que fue entrenado. El rendimiento de la red neuronal del sistema es de $1.04x10^{-11}$ aprox, como se ilustra en la Figura 5.9. El porcentaje de aciertos del sistema implementado es de 80% aproximadamente.

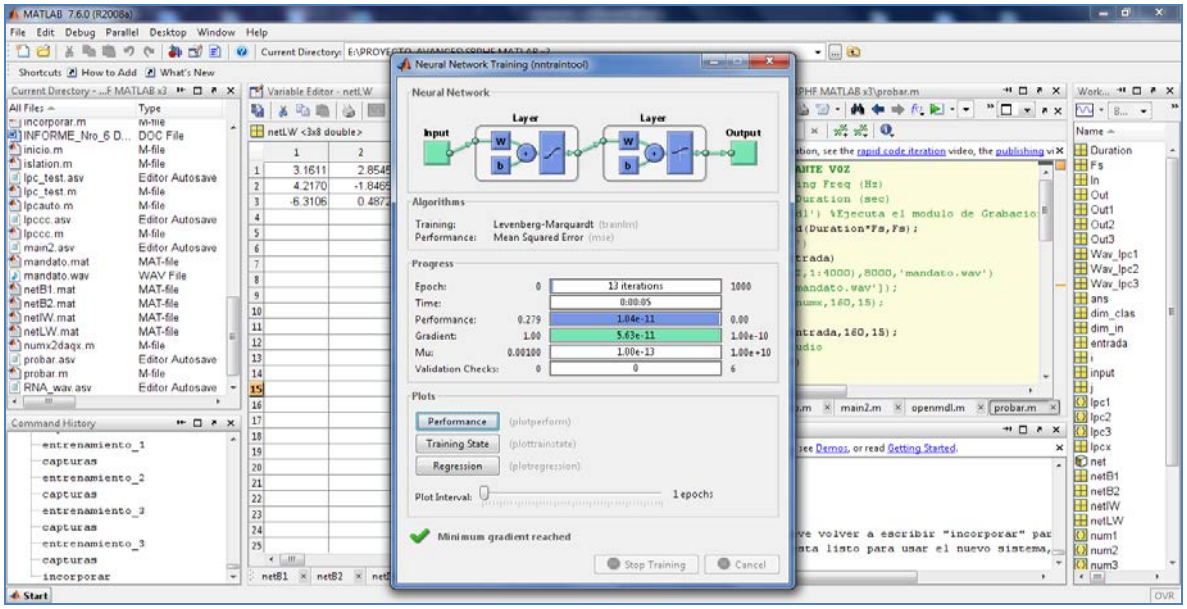


Figura 5.9. Rendimiento del entrenamiento de la ANN (visto en MATLAB v2008a)

Al abrir el proyecto “DE2P_TOP”con QuartusII se puede ver la estructura hardware del sistema. El archivo fuente principal está escrito en Verilog y contiene las instanciaciones a los elementos hardware del sistema, entre los cuales se encuentra el procesador Nios II desarrollado a medida, como se ilustra en la Figura 5.10.

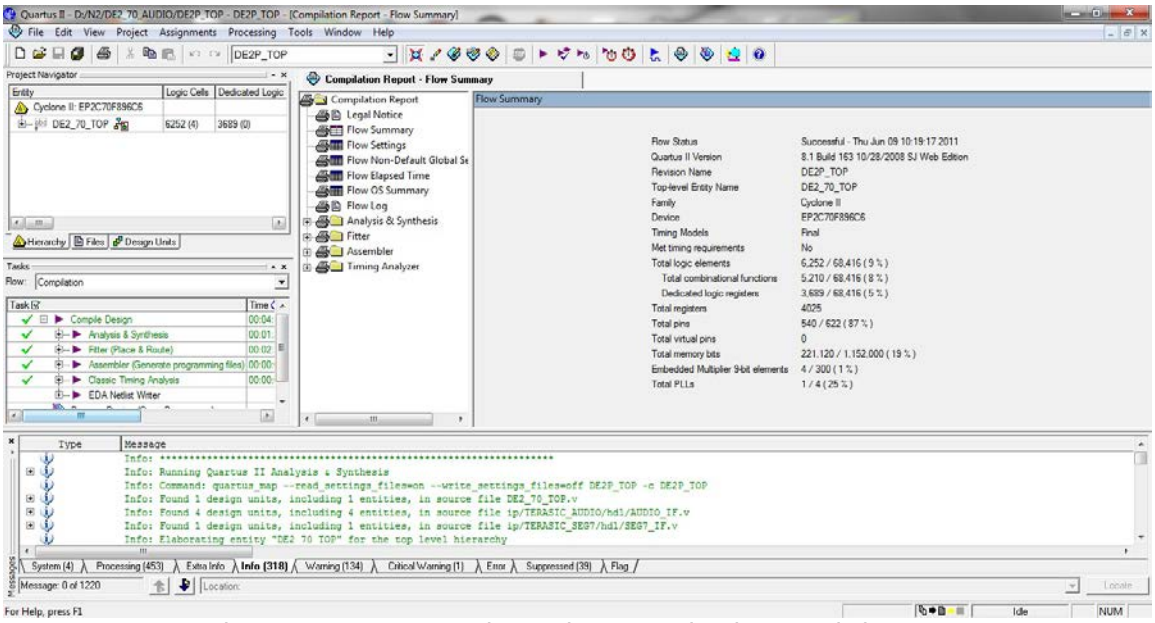


Figura 5.10. Recursos de Hardware empleados por el sistema.

Con la herramienta SoPC-Builder de Quartus II se puede ver la estructura del procesador Nios II usado en este trabajo, que consta de procesador y controladores para memorias, códec de audio, y demás periféricos de la tarjeta DE2-70. Se puede realizar modificaciones y volver a generar un procesador, luego se debe actualizar el archivo fuente principal del proyecto y compilar para poder usar la nueva configuración de hardware, como se ilustra en la Figura 5.11.

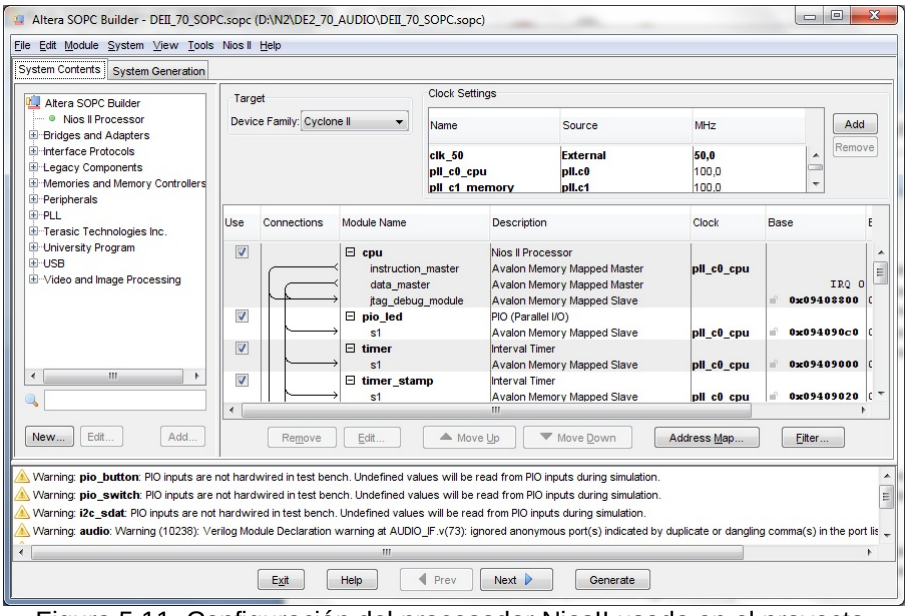


Figura 5.11. Configuración del procesador NiosII usado en el proyecto.

La configuración de software se realiza en el entorno integrado Nios II IDE, donde se selecciona como espacio de trabajo la carpeta “software” que está dentro de la carpeta general de proyecto. Dentro de dicha carpeta se encuentran los programas en C o C++ que serán ejecutados por el procesador Nios II especificado en la configuración de hardware, como se ilustra en la Figura 5.12. Para realizar la grabación, en el Nios II IDE, dar clic derecho en la carpeta srphf_nios y seleccionar la opción “Run As… NiosII Hardware”. Con esto el sistema está listo para ser probado, por lo que se presionará una vez el pulsador K3 y se pronunciará las palabras. La ventana de consola del Nios II IDE mostrará el resultado del proceso de reconocimiento de voz.

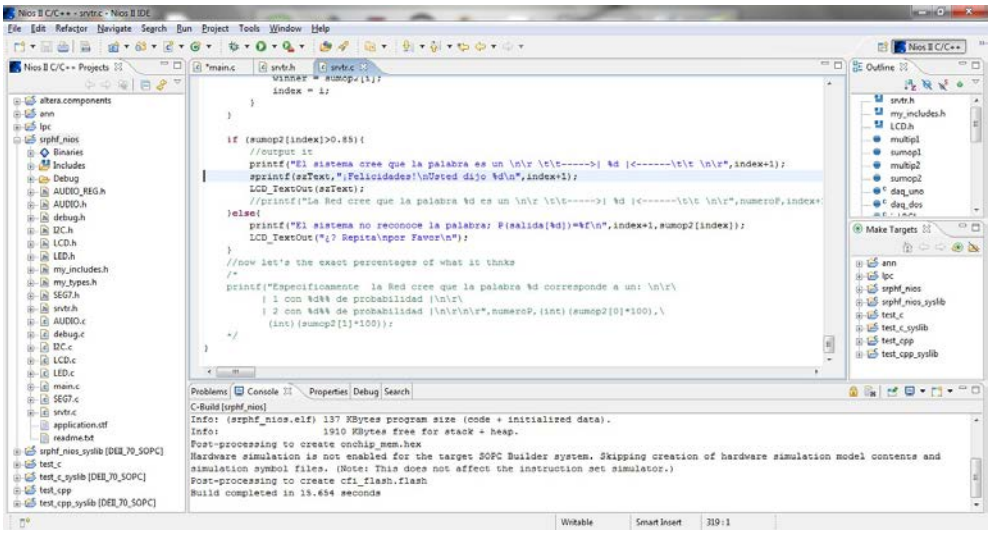


Figura 5.12. Recursos de software empleados por el sistema.

Para las pruebas se entrenó el sistema con 3 palabras: do, re y mi. Como el sistema es genérico, “do” es la 1ª palabra, “re” es la 2ª palabra y “mi” es la 3ª palabra. Fotos de los resultados se muestran en las Figuras 5.13 a 5.15. El porcentaje de aciertos del sistema es de 80% aproximadamente.

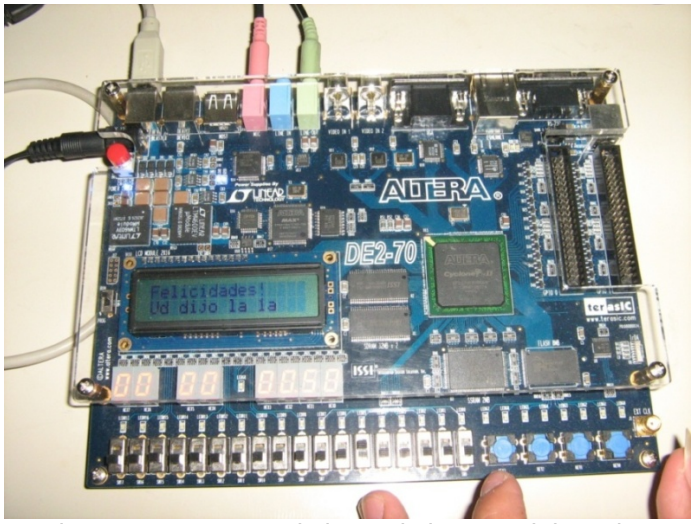


Figura 5.13. Reconocimiento de la 1ra palabra “do”.

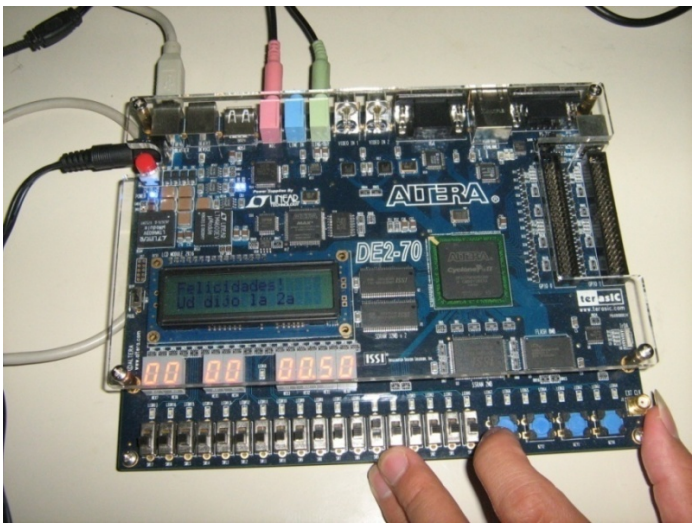


Figura 5.14. Reconocimiento de la 2da palabra “re”.

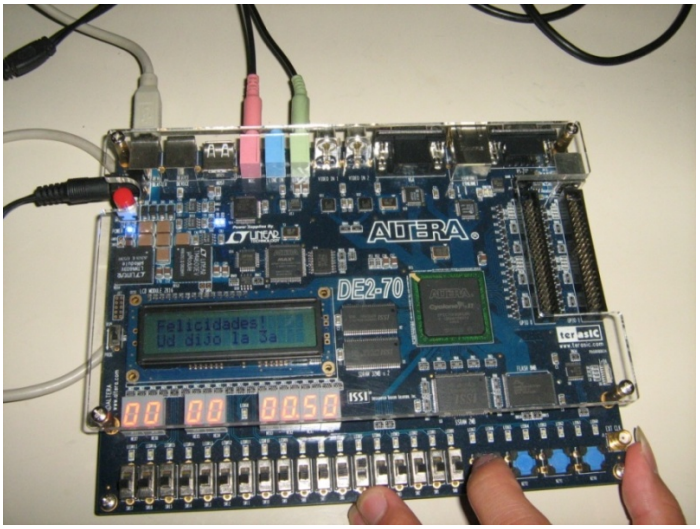


Figura 5.15. Reconocimiento de la 3ra palabra “mi”.

En el presente capítulo se ha mostrado el desarrollo e implementación del sistema reconocedor de palabras aisladas en FPGA utilizando la tarjeta de desarrollo DE2-70 de Altera Corp. Los resultados de las pruebas son comparables a los obtenidos en la simulación con la herramienta de software MATLAB, lo cual valida la metodología de diseño empleada y la correcta aplicación de los algoritmos para el reconocimiento de las palabras de manera bastante aceptable.

A continuación incluiremos las conclusiones a las que se ha llegado luego del desarrollo e implementación del sistema propuesto, para luego continuar con las observaciones y recomendaciones para futuras ampliaciones.

CONCLUSIONES

En este trabajo se ha presentado un sistema de reconocimiento de voz (palabras habladas), probado con diferentes hablantes. A partir de las pruebas se puede concluir que el sistema previamente entrenado con un hablante puede reconocer otros hablantes siempre y cuando pronuncien las palabras con la misma entonación (tonalidad de la voz y acentuación en la pronunciación). Esto se debe a que el algoritmo de codificación de la voz LPC, usado en el proyecto, se usa también para caracterizar el tono de voz de los hablantes.

El sistema desarrollado es transportable a FPGA's Altera de diferentes densidades que puedan contener el procesador Nios II usado en el proyecto. La portabilidad del sistema se puede dar con FPGA's de diferentes fabricantes, si todo el desarrollo se hubiera hecho usando un lenguaje de descripción de hardware estandarizado (VHDL o Verilog). Aunque el Nios II está elaborado en lenguaje de descripción de hardware, su arquitectura utiliza una definición de bajo nivel, que explota las características únicamente en FPGA's de Altera.

El uso de un procesador embebido en software en el FPGA (soft-core processor), permite la configuración del sistema de reconocimiento de acuerdo a los requerimientos establecidos durante el modelado del sistema.

Los FPGA's dan la posibilidad de desarrollar diferentes aplicaciones, combinando en una misma aplicación a elección del diseñador, las ventajas del uso de hardware en el FPGA propiamente y del uso de software con un procesador embebido.

El consumo de recursos hardware en el FPGA de la tarjeta DE2-70 se resume en:

Recurso	Consumo
Cantidad de elementos lógicos usados del FPGA Cyclone II	6252, de un total de 68416 (9%)
Cantidad de pines usados del FPGA Cyclone II	540, de 622 disponibles (87%)
Cantidad de bits de memoria usados del FPGA Cyclone II	221120, de un total de 1152000 (19%)
Cantidad de multiplicadores embebidos usados del FPGA	4, de un total de 300 (1%)
Cantidad de PLLs usados del FPGA Cyclone II	1, de un total de 4 (25%)

El consumo de recursos de software del sistema fue de:

Recurso	Consumo
Tamaño de programa srphf_nios.elf	137 KBytes de un total de 1910 KBytes (7.173%)

En los 2 puntos anteriores se observa que el consumo de recursos es relativamente pequeño, es decir se puede añadir más bloques hardware o procesadores embebidos si se desea,siempre y cuando no crezca el consumo de pines.

El siguiente cuadro, se hace una comparación con 2 productos comerciales, de los cuales el BlueAnt S4, es un kit hardware para hacer y recibir llamadas solo usando la voz, a bajo costo. Como se puede observar la gran ventaja del sistema desarrollado es la flexibilidad que ofrece, lo cual básicamente se debe a la utilización, para el hardware, de un FPGA en lugar de por ejemplo un DSP. La idea del presente trabajo fue realizar una investigación sobre el tema, fundamentalmente en lo que concierne a la implementación de los algoritmos usados.

Categoría	Dragon	BlueAnt	Sistema desarrollado
Costo ($)	200	100	279*
Rendimiento	Bueno	Bueno	Regular
Hardware/Software	Software	Hardware	Hardware
Flexibilidad	Baja	Baja	Alta

*Precio académico de la tarjeta de desarrollo.

El rendimiento se puede definir como la razón entre la cantidad de aciertos y la cantidad de intentos en el reconocimiento de una palabra; en el sistema propuesto depende bastante del entrenamiento realizado. Con un buen entrenamiento de la red neuronal, que implica tener una pronunciación constante de la palabra, se mejora el rendimiento. El rendimiento calculado en el sistema es de 80% aproximadamente. Asi también, la cantidad de palabras y el contenido del vocabulario se pueden modificar.

La red neuronal usada en el sistema se entrena en MATLAB para conseguir los parámetros que se van a guardar en la memoria del sistema desarrollado. La performance de la red neuronal es la evaluación de una función que cuantifica la confiabilidad de la red, probándola con cientos de entradas con cantidades de ruido de 0 a 0.5 de desviación estándar, como se ilustra en la Figura 6.1.

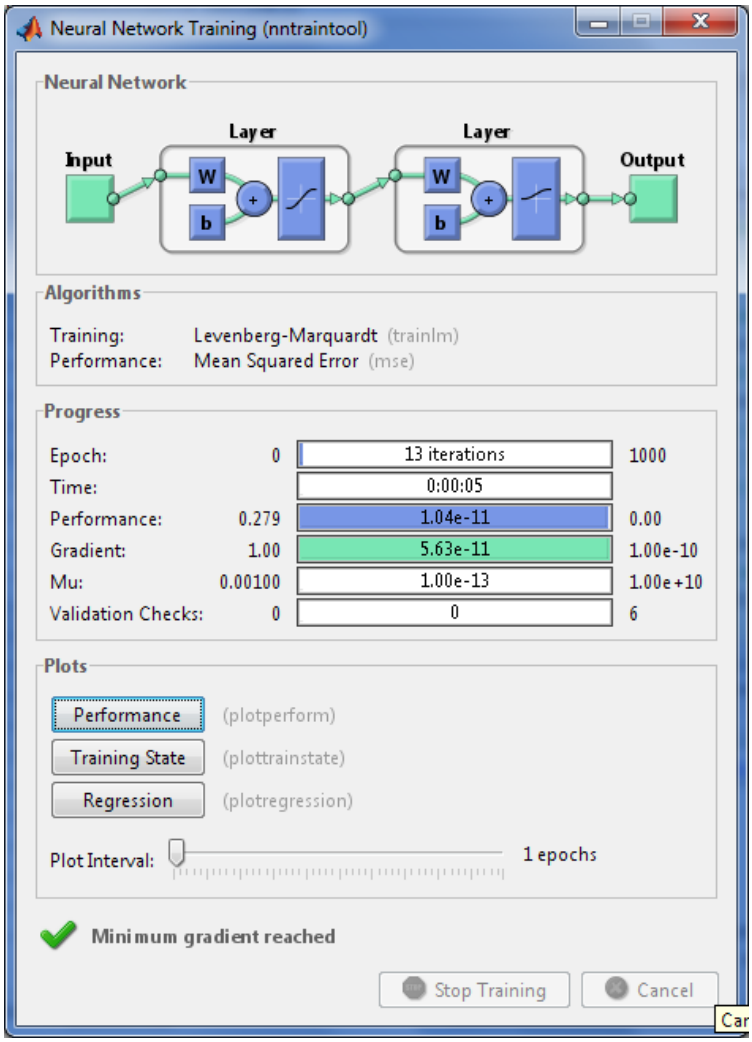


Figura 6.1. Entrenamiento del Sistema en MATLAB.

Como antecedente se puede mencionar el trabajo de investigación titulado “Diseño e implementación de un Sistema de Reconocimiento de Palabras en un FPGA” del Instituto de Investigación de la Facultad de Ingeniería Eléctrica y Electrónica de la Universidad Nacional de Ingeniería [18], que consistió en diseñar el sistema utilizando la tarjeta de desarrollo Spartan-3E Starter Kit que utiliza el FPGA X3S500E del fabricante de dispositivos lógicos programables Xilinx, el software ISE 10.1, y el System Generator 10.1, éste trabaja en el ambiente Simulink de Mat-Lab utilizando bloques de Xilinx que se pueden combinar con los bloques de Simulink, la ventaja del uso de este software es que estando en el ambiente Simulink se puede crear, a partir de los bloques, un archivo .vhd, que puede ser compilado en el software ISE 10.1, además se puede programar el FPGA desde Simulink usando la Co-simulación.

Para la codificación de la voz se usó el algoritmo LPC, y para el reconocimiento el algoritmo DTW se tuvo que implementar un controlador en VHDL para el manejo de los LEDs de salida. Las pruebas se realizaron utilizando 2 LEDs y el vocabulario fueron las palabras uno, dos, tres y cuatro. Un comando de voz uno enciende el LED0, el comando dos lo apaga. El comando tres enciende el LED1 y cuatro lo apaga.Se hicieron pruebas con varios tipos de secuencias “uno”, “uno-tres”, “uno-tres”, “uno-tres-dos”, “uno-dos-tres”, etc no se se obtuvo un rendimiento, definido como la razón entre la cantidad de aciertos y la cantidad de intentos de la implementeción final pero si se hizo para las simulaciones obteniendo un rendimiento de 70 %.

OBSERVACIONES Y RECOMENDACIONES

- Se debe observar que para el desarrollo del presente trabajo de tesis se decidió por el fabricante Altera Corp., por proveer herramientas de software de diseño de prototipado de hardware usando FPGAs y su fácil utilización para diseños que involucran un procesador embebido definido por software como el Nios II. La configuración del procesador Nios II/s se seleccionó en función de los recursos de hardware del FPGA necesarios y que no impacten en el desempeño de todo el sistema. Para simplificar el trabajo se utilizó un demo del fabricante de la tarjeta para el manejo del códec.

- Se debe destacar que se hicieron pruebas a nivel de simulación del sistema con la herramienta de software MATLAB, previo a la implementación del diseño en hardware en la tarjeta de desarrollo DE2-70, y cuyos resultados son muy parecidos a los obtenidos en hardware.

- Luego de realizada la parte experimental del proyecto, y de los resultados obtenidos, se recomienda extender el número de palabras de vocabulario del sistema reconocedor de palabras. La cantidad máxima de palabras a almacenar depende de la capacidad de la memoria de programa de la tarjeta. Para ello se debe modificar el código de entrenamiento de la red hecho en MATLAB.

- Se recomienda hacer más interactivo el proceso de entrenamiento de la red. Para ello, se debe imprimir las muestras tomadas por el códec de audio y añadir un UART al procesador NiosII para enviar dichas muestras por el puerto serie de la PC, donde se puede modificar el programa de entrenamiento para adquirir los datos directamente del puerto serial, en lugar de archivos de texto, y devolver las matrices de pesos y umbrales también por el puerto serial, donde el sistema reconocedor almacenará dichos valores para su uso en el nuevo reconocimiento.

- Es recomendable automatizar el reconocimiento de voz encendiendo un LED de la tarjeta durante el tiempo de grabación y apagándolo durante el tiempo de procesamiento, esta secuencia repetida periódicamente. El sistema podrá cambiar a modo de entrenamiento con un interruptor de la tarjeta, con lo que se consigue una interacción hombre-máquina con las ventajas que esto supone.

- El prototipo de reconocedor de voz implementado se puede aplicar, por ejemplo, al control por voz de un dispositivo móvil a distancia, con el uso de una radio o teléfono celular. Otra aplicación es la domótica, con el control por voz en casa de funciones tales como abrir puertas, encender luces, etc.

- Por otro lado, se recomienda independizar el sistema reconocedor de voz de la PC, ya que un componente actual es propietario y requiere estar conectado a la PC en su versión de prueba. Se puede comprar la licencia del IP SSRAM, o usar una memoria embebida para el almacenamiento del programa en lugar de la SSRAM.

REFERENCIAS

[1] Altera Corp. http://www.altera.com/

[2] Centro de descargas de Altera Corp. https://www.altera.com/download/dnl-index.jsp

[3] Foro de Altera: http://www.alteraforum.com

[4] Literatura de Altera: http://www.altera.com/literature/lit-index.html

[5] Literatura sobre procesador Nios II: http://www.altera.com/literature/lit-nio2.jsp

[6] Mathworks Inc. Matlab:The language of technical computing, disponible en: http://www.mathworks.com/products/matlab/

[7] N .S. Neheand R.S. Holambe, "New Feature Extraction Methods Using DWT and LPC for Isolated Word Recognition", IEEE Region 10 Conference, 2008.

[8] José Hilera, Víctor Martínez, "Redes Neuronales Artificiales Fundamentos, Modelos y Aplicaciones", Addison-Wesley Iberoamericana 1995.

[9] John G. Ackenhusen, L. R. Rabiner, "Microprocessor Implementation of an LPC-based Isolated Word Recognizer". IEEE, 1981.

[10] http://www.cas.mcmaster.ca/~lawford/3TB4/ref/nios2-2.pdf

[11] http://www.fceia.unr.edu.ar/acustica/biblio/fonatori.pdf

[12] http://www.psicologia-online.com/ebooks/general/neuronas.htm

[13] Terasic Technologies Inc. DE2-70 User manual. Disponible en: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=39&No=226&PartNo=4

[14] Lawrence Rabiner and Ronald W. Schafer, "Digital Processing of Speech Signals", Prentice Hall, 1978.

[15] http://www.bearcave.com/misl/misl_tech/wavelets/

[16] http://www.progmat.uaem.mx:8080/articulos/vol1no1articulo7.pdf

[17] http://www.terasic.com

[18] "Diseño e implementación de un Sistema de Reconocimiento de Palabras en un FPGA", Instituto de Investigación FIEE, 2010.

[19] "Diseño e implementación en FPGA de un Sistema para Reconocimiento de Voz en Tiempo Real", Instituto de Investigación FIEE, 2011.

[20] John R. Deller, John H.L. Proakis, H. L. Hansen, "Discrete Time Processing of Speech Signals" IEEE, 2000.

ANEXO A
Configuración del procesador embebido Nios II

La Figura A.1 muestra el esquema generado por la herramienta de software Quartus II, donde el elemento central es el procesador embebido Nios II generado con la herramienta SoPC Builder. El procesador Nios II es programado usando el entorno integrado NiosII IDE.

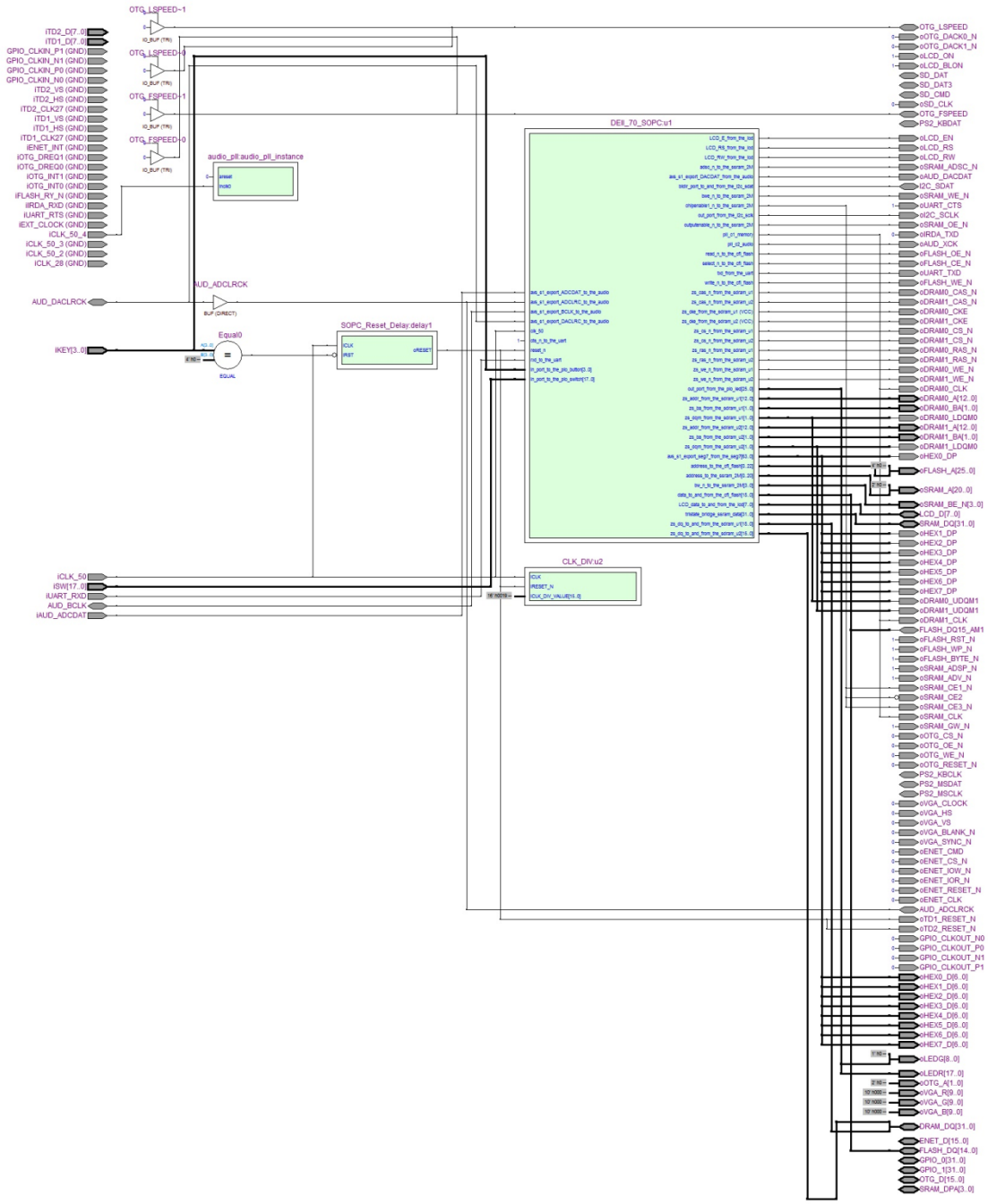


Figura A.1. RTL de la parte hardware del proyecto

El gráfico de la Figura A.1 es generado mediante el archivo tope hecho en verilog (Figura A.3), el cual tiene como principal elemento al procesador Nios II que fue generado por SoPC Builder.

En la Figura A.2 se muestran las conexiones lógicas del procesador Nios II con los diferentes elementos de hardware como memorias y periféricos.

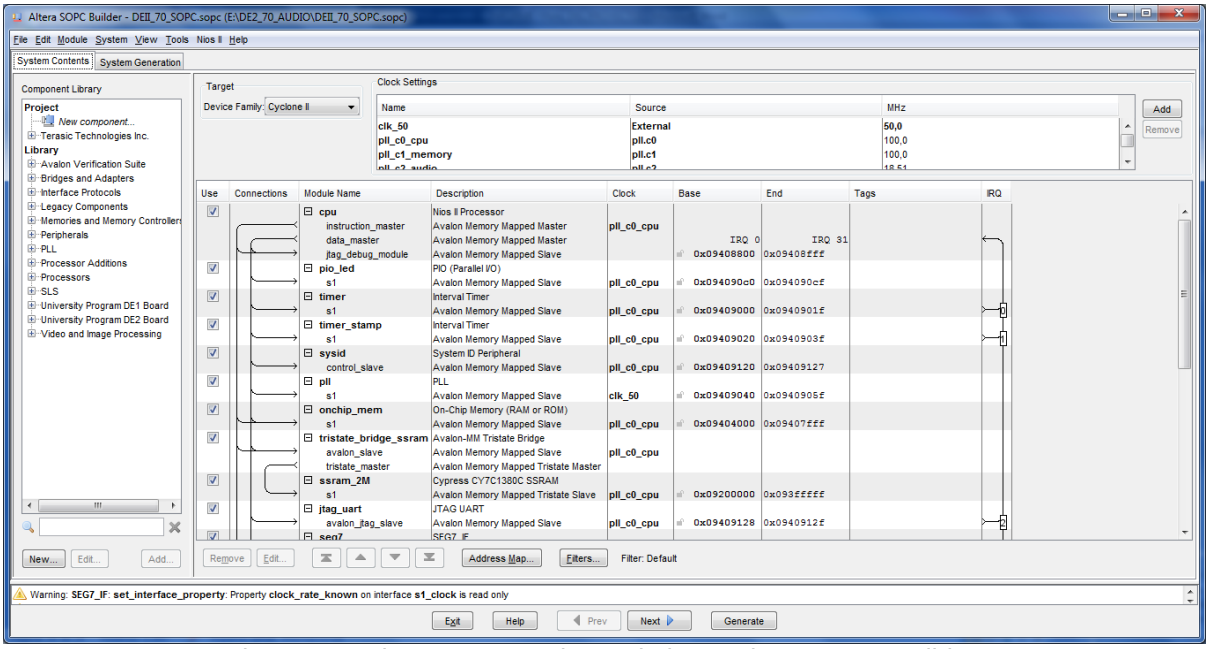


Figura A.2. Bloques agregados en la herramienta SoPC Builder.

En la Figura A.3 se muestra el tope de jerarquía del diseño de hardware, especificado con el lenguaje de descripción de hardware Verilog.

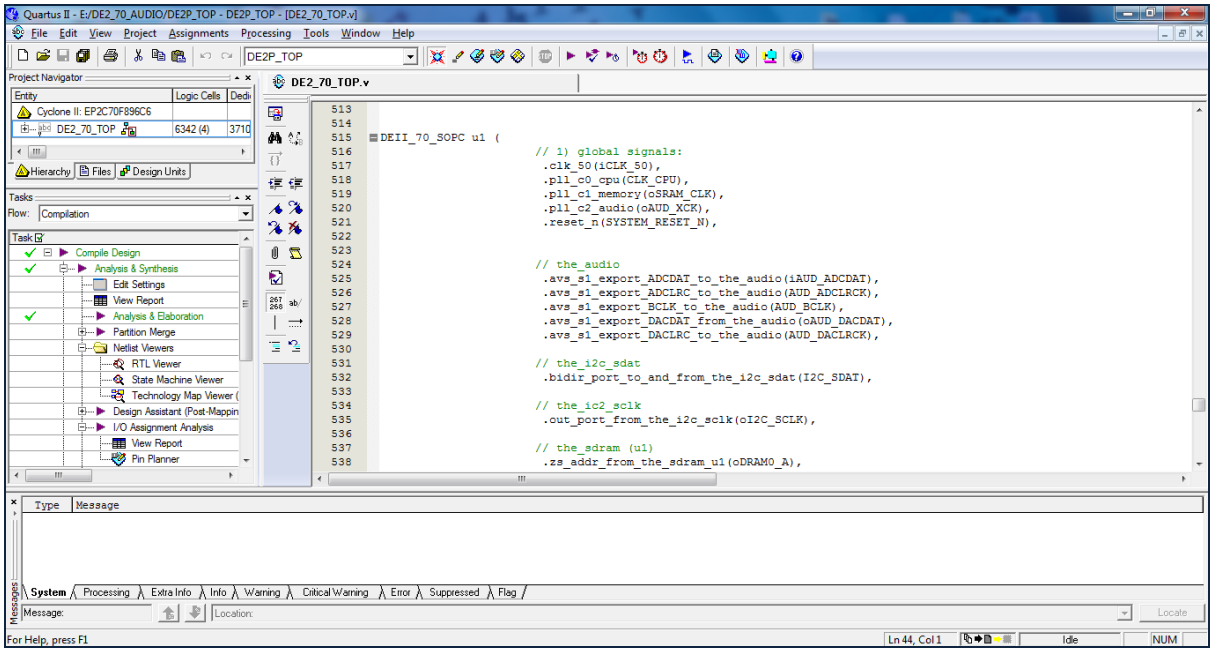


Figura A.3. Archivo tope generado en verilog.

ANEXO B
Algoritmos para la Simulación en MATLAB

A continuación, se presentan los listados de código referente a la programación del escenario de simulación usando la herramienta de software MATLAB.

```
%% ACTIVACION MEDIANTE VOZ

Fs = 8000; % Sampling Freq (Hz)
Duration = 1.0; % Duration (sec)

estacc=20; %Milisegundos que se consideran estables
nlpc=10; %Numero de parametros lpc
aislar=0.6;%Porcion de la muestra que se deberá aislar

%Grabacion y procesado
entrada = wavrecord(Duration*Fs,Fs);
figure(2); plot(entrada)
estaccc=floor(estacc*Fs/1000);
pasos=floor(aislar*Duration*Fs/estaccc);
numxx = islation(entrada,estaccc,pasos);

%lee la pista de audio
wavplay(numxx,Fs)

%Codificacion LPC
lpcx = lpccc(numxx,estaccc,nlpc,pasos);

%Carga de la estructura de Red Neuronal
load netIW.mat
load netLW.mat
load netB1.mat
load netB2.mat

%% Función clasificadora:
In=lpcx';
Out=logsig(netLW*tansig(netIW*In+netB1)+netB2);
Out'

if max(Out) <= 0.90 || min(Out) >= 0.10 || sum(Out) >= 1.1
   error('Pruebe a hablar mas alto y claro')
end

%Salida

if Out(1) == max(Out)
   disp('La palabra reconocida fue la "primera"')
end
if Out(2) == max(Out)
   disp('La palabra reconocida fue la "segunda"')
end
if Out(3) == max(Out)
   disp('La palabra reconocida fue la "tercera"')
end
```

Suponiendo que el locutor temporal ha entrenado al sistema con las grabaciones de los comandos con su voz, resumidamente, se comienza por lanzar el comando “probar” que graba el comando a usar por un lapso de 1.0s, se carga lo grabado a Matlab, se hace un procedimiento de aislamiento de la palabra que se pudiera encontrar en la grabación buscando el segmento de energía predominante, se verifica el comando pronunciado al escuchar su voz, se hace la caracterización del comando mediante LPC, se cargan los

valores de arquitectura de la RED NEURONAL y se procede al reconocimiento con una salida tanto binaria como textual.

El código para el aislamiento de la palabra es el que se muestra a continuación, el cual busca el segmento de tiempo que contenga la mayor cantidad de energía:

```
function X2 = islation(X,m,paso)
[bx,n]=size(X);
bx0=floor((bx/m));

for i=1:bx0
frame= X((i-1)*m+1:i*m) %-mean(numx((i-1)*160+1:i*160));
frame_energy(i) = log(sum(frame.*frame)+eps);
end

frame_energyy = frame_energy-min(frame_energy);

for i=bx0-paso+1
frame_energyyy(i)=sum(frame_energyy(i:i+paso-1));
end

ub_max = find(frame_energyyy >= max(frame_energyyy));

X2 =  X((ub_max-1)*m+1:(ub_max+paso-1)*m);
```

El código de caracterización con LPC es el que muestra a continuación, el cual trabaja sobre un segmento fijo de 4800 puntos ó 0.6 seg de grabación, específicamente calcula 10 coeficientes LPC para 30 tramas ó segmentos de voz estacionarios de 160 muestras o 20 ms cada uno.

```
function XXM=lpccc(X,m,p,paso)

[bx,n]=size(X);

ZM=ones(m*paso,1);
ZM(1:bx,1)=X;

bx1=(m*paso/m);

XM=zeros(p+1,bx1);

for i=1:bx1
   XM(:,i)=lpc(ZM(m*(i-1)+1:m*i,1)',p)';
end

XM=XM(2:p+1,:);

XXM=[];
for i=1:bx1
   XXM=[XXM XM(:,i)'];
end
```

Eso sería todo lo necesario para poder empezar el reconocimiento de voz, aparte de los archivos *.mat que describen la arquitectura de la red neuronal que implícitamente contiene nuestro diccionario de voces.

Si no hemos entrenado nuestra red neuronal o queremos probar el sistema con un locutor diferente, se empieza por grabar al diccionario con el código que se muestra a continuación:

```
%% ENTRENAMIENTO

%%Parametros a ajustar:
Fs = 8000; % Sampling Freq (Hz)
Duration = 1.0; % Duracion (sec)
Muestr= 12; %Numero de muestras por grabacion
estacc=20; %Milisegundos que se consideran estables
nlpc=10; %Numero de parametros lpc
aislar=0.6;%Porcion de la muestra que se deberá aislar

%% GRABACIONES

disp('Grabará 3 palabras por 12 veces seguidas, siga las instrucciones ')
disp('Si está en un ambiente ruidoso tendra que hablar fuerte y claro" ')
pause(2)
disp('En 3 seg dirá "Su 1º palabra" , hable cuando vea "comience" o "siguiente" ')
pause(1)
disp('En 2 seg dirá "Su 1º palabra" , hable cuando vea "comience" o "siguiente" ')
pause(1)
disp('En 1 seg dirá "Su 1º palabra" , hable cuando vea "comience" o "siguiente" ')
pause(1)
disp('Comience!')


clear num1
for i=1:Muestr
    y = wavrecord(Duration*Fs,Fs);
    wavwrite(y,Fs,['1e_' num2str(i) '.wav'])
    pause(0.5)
    disp('Siguiente')
    pause(0.0)
end

disp('En 2 seg dirá "Su 2º palabra"')
pause(2)
disp('Comience!')

clear num2
for i=1:Muestr
    y = wavrecord(Duration*Fs,Fs);
    wavwrite(y,Fs,['2e_' num2str(i) '.wav'])
    pause(0.5)
    disp('Siguiente')
    pause(0.0)
end

disp('En 2 seg dirá "Su 3º palabra"')
pause(2)
disp('Comience!')

clear num1
for i=1:Muestr
    y = wavrecord(Duration*Fs,Fs);
    wavwrite(y,Fs,['3e_' num2str(i) '.wav'])
    pause(0.5)
    disp('Siguiente')
    pause(0.0)
end

disp('Para escuchar sus grabaciones escriba "capturas"!')
```

En la ventana de comandos aparecerán las instrucciones para empezar a grabar, normalmente siendo 12 grabaciones para cada palabra a reconocer cuyo número puede variar a voluntad.

El siguiente código nos servirá para verificar que hallamos grabado bien escuchando nuestras grabaciones, además de hacer el proceso de aislamiento y caracterización LPC.

```
%% CAPTURAS
clear num1
for i=1:Muestr
    num1{i} = wavread(['1e_' num2str(i) '.wav']);
    %wavplay(num1{i},Fs)
end
clear num2
for i=1:Muestr
    num2{i} = wavread(['2e_' num2str(i) '.wav']);
    %wavplay(num2{i},Fs)
end
clear num3
for i=1:Muestr
    num3{i} = wavread(['3e_' num2str(i) '.wav']);
    %wavplay(num2{i},Fs)
end

estaccc=floor(estacc*Fs/1000);
pasos=floor(aislar*Duration*Fs/estaccc);

clear numm1
for i=1:Muestr
    numm1{i} = islation(num1{i},estaccc,pasos);
    wavplay(numm1{i},Fs)
end
clear numm2
for i=1:Muestr
    numm2{i} = islation(num2{i},estaccc,pasos);
    wavplay(numm2{i},Fs)
end
clear numm3
for i=1:Muestr
    numm3{i} = islation(num3{i},estaccc,pasos);
    wavplay(numm3{i},Fs)
end

%% LPC
clear lpc1
for i=1:Muestr
    lpc1{i}=lpccc(numm1{i},estaccc,nlpc,pasos); %lpccc(X,m,p)
end
clear lpc2
for i=1:Muestr
    lpc2{i}=lpccc(numm2{i},estaccc,nlpc,pasos); %lpccc(X,m,p)
end
clear lpc3
for i=1:Muestr
    lpc3{i}=lpccc(numm3{i},estaccc,nlpc,pasos); %lpccc(X,m,p)
end


disp('Si esta conforme con sus grabaciones escriba "incorporar" para guardarlas en el sistema,')
disp('de lo contrario escriba "entrenamiento" para grabar otra vez')
```

De no estar conformes con lo grabado, se procede a grabar de nuevo.

El siguiente código es el generador de la arquitectura de nuestra red neuronal entendiéndose como el entrenamiento de nuestro sistema, el que se sigue para obtener los parámetros que se usan en las pruebas.

```
%% Generador de Arquitectura de Red Neuronal
tic
clear Wav_lpc1 Wav_lpc2 Wav_lpc3
for i=1:Muestr
   Wav_lpc1(i,:)  = lpc1{i} ; %CLASE 1
end
for i=1:Muestr
   Wav_lpc2(i,:)  = lpc2{i} ; %CLASE 2
end
for i=1:Muestr
   Wav_lpc3(i,:)  = lpc3{i} ; %CLASE 3
end

dim_clas=3; %numero de salidas
uso_clas=Muestr-2; %numero de pruebas por cada clase

if dim_clas==3
input=[Wav_lpc1(2:uso_clas+1,:)' Wav_lpc2(2:uso_clas+1,:)' Wav_lpc3(2:uso_clas+1,:)'];
end
if dim_clas==4
input=[Wav_lpc1(2:uso_clas+1,:)' Wav_lpc2(2:uso_clas+1,:)' Wav_lpc3(2:uso_clas+1,:)'
Wav_lpc4(2:uso_clas+1,:)'];
end

dim_in=size(input,1); %numero de entradas

%% %% %% %% %%
target=zeros(uso_clas,dim_clas);
for j=1:dim_clas
   for i=1:uso_clas
      target(i+uso_clas*(j-1),j)=1;
   end
end

clear pt
for i=1:dim_in
   pt(i,:)=[-3 3];%intervalo de entradas
end

%pt=minmax(input)

clear net
net = newff(pt,[dim_clas+7 dim_clas],{'tansig','logsig'});

%output= sim(net,input2)
%net.trainParam.min_grad=1.0e-20

net  = train(net,input,target');
%Guardar pesos y bias
netIW=net.IW{1,1};
netLW=net.LW{2,1};
netB1=net.b{1,1};
netB2=net.b{2,1};

save('netIW.mat','netIW')
save('netLW.mat','netLW')
save('netB1.mat','netB1')
save('netB2.mat','netB2')

%% Función clasificadora:
%In=Wav_lpc2(7,:)';
```

```
%Out=logsig(netLW*tansig(netIW*In+netB1)+netB2)

clear Out1
for j=[2:Muestr-1]
    In=Wav_lpc1(j,:)';
    Out1(:,j)=logsig(netLW*tansig(netIW*In+netB1)+netB2);
end
clear Out2
for j=[2:Muestr-1]
    In=Wav_lpc2(j,:)';
    Out2(:,j)=logsig(netLW*tansig(netIW*In+netB1)+netB2);
end
clear Out3
for j=[2:Muestr-1]
    In=Wav_lpc3(j,:)';
    Out3(:,j)=logsig(netLW*tansig(netIW*In+netB1)+netB2);
end

Out1(:,[2:Muestr-1])
Out2(:,[2:Muestr-1])
Out3(:,[2:Muestr-1])
toc
disp('Revise sus resultados, pueve volver a escribir "incorporar" para ver si mejoran')
disp('Si esta conforme ahora ya esta listo para usar el nuevo sistema, escriba "probar"')
```

ANEXO C
Algoritmo en C del Sistema de Reconocimiento para el Nios II

A continuación, se presentan los listados de código referente a la programación en C para el FPGA.

Código lpc_ann.h

```
#ifndef LPC_ANN_H
#define LPC_ANN_H

#ifdef  __cplusplus
extern "C" {
#endif


#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define logsig(value) (1/(1+exp(-value)))
#define tansig(value) (2/(1+exp(-2*value))-1)

double trama[160]; //Trama=20ms (267, 15: 03-12-2011)
double rauto[11]; //lpcxT=10
double lpc_uno[300], lpc_dos[300], lpc_ent[300]; //(150: 03-12-2011)
double Am[11][11];

void cargarTrama(int muestrasxT, int numeroT, int numeroP, double daq_ent[]);
void calcularRs(int lpcxT, int muestrasxT);
void levinsonDurbin(int lpcxT);
void resultadoLpc(int lpcxT, int numeroT);

void primerMultip(int numeroP,int hiddenN, int inputN);
void primerSumop(int hiddenN);
void segundoMultip(int outputN, int hiddenN);
void segundoSumop(int outputN);
void resultadoAnn(int numeroP, int outputN);


#ifdef  __cplusplus
}
#endif

#endif  /* LPC_ANN_H */
```

Código lpc_ann.c

```
#include "lpc_ann.h"
#include "voice.h"
#include "LCD.h"

//int N;
double multip1[10]; //(8: 03-12-2011)
double sumop1[10]; //(8: 03-12-2011)
double multip2[3];
double sumop2[3];
//float daq_ent[4000];

const double netB1[10] = { //(8x1: 03-12-2011)
-0.7243,
-1.4188,
-0.4584,
0.9126,
1.289,
-2.7264,
-1.1792,
```

```
1.4271,
0.9093,
0.6831
};

const double netB2[3] = { //3x1 (2x1)
1.6097,
-2.5307,
3.6008
};

const double netLW[30] = { //3x10 (3x8: 03-12-2011)
0.7013, 4.569, -0.5212, -4.355, -2.3707, 4.2971, 2.0315, -9.1676, -2.2997, -0.4699,
-2.6434, -0.7624, 3.4577, 4.4247, -4.1439, -4.7265, 2.851, 9.6398, -15.2141, 3.4687,
-0.1966, -6.5612, -0.9745, -7.6511, -0.906, 6.0072, 1.3829, 5.5175, 13.1642, 0.3223
};

void cargarTrama(int muestrasxT, int numeroT, int numeroP, double daq_ent[]){
    int i;

    for(i=0; i<muestrasxT; i++){
        //if(numeroP==1)
            //trama[i]=daq_uno[muestrasxT*numeroT+i];
        //else if(numeroP==2)
            //trama[i]=daq_dos[muestrasxT*numeroT+i];
        //else
            trama[i]=daq_ent[muestrasxT*numeroT+i];
        //trama[i]=daq_tst[muestrasxT*numeroT+i];
    }

/*
    for(i=0; i<muestrasxT; i++){
        //printf("trama[%d]: %f\n",i+1,trama[i]);
        printf("%f\n",trama[i]);
    }
*/

}

void calcularRs(int lpcxT, int muestrasxT){
    int i,j;
    //r(k)=sum(0,N-1-k,x[n]*x[n+k])
    for(i=0; i<lpcxT+1; i++){
        rauto[i]=0;
        for(j=0; j<muestrasxT-i; j++){
            rauto[i] += trama[j]*trama[j+i];
            //rauto[i] = rauto[i] + trama[j]*trama[j+i];
        }
        //Biased autocorrelation
        //rauto[i]=rauto[i]/muestrasxT;
    }

/*
    for(i=0; i<lpcxT+1; i++){
        printf("rauto[%d]: %f\n",i,rauto[i]);
    }
*/
}

void levinsonDurbin(int lpcxT){
    int i,j=0;
    double Em[lpcxT+1];
    double Qm[lpcxT+1];
    double Km[lpcxT+1];

    //Valores iniciales
    Em[0]=rauto[0];
    Km[1]=rauto[1]/Em[0];
    Am[1][1]=Km[1];
    Em[1]=Em[0]*(1-Km[1]*Km[1]);

    //Recursion para m>=2
    i=1;
    do{
        i++;
        Qm[i]=rauto[i];
        for(j=1;j<i;j++){
            Qm[i] -= Am[j][i-1]*rauto[i-j];
```

```
        }
        Km[i]=Qm[i]/Em[i-1];
        Am[i][i]=Km[i];
        for(j=1;j<i;j++){
            Am[j][i]=Am[j][i-1]-Km[i]*Am[i-j][i-1];
        }
        Em[i]=Em[i-1]*(1-Km[i]*Km[i]);
    }while(i<lpcxT);
}

void resultadoLpc(int lpcxT, int numeroT){
    int m=0;
    //int n=0;
    for(m=0; m<lpcxT; m++){
        //MATLAB definition with -
        lpc_ent[numeroT*lpcxT+m]=-Am[m+1][lpcxT];
        //printf("%f\n",lpc_ent[numeroT*lpcxT+m]);
        //printf("lpc_uno[%d]: %f\n",numeroT*lpcxT+m+1,lpc_uno[numeroT*lpcxT+m]);
    }

/*
    printf("Matriz Am[][]: \n");
    for(m=0;m<lpcxT+1;m++){
        for(n=0;n<lpcxT+1;n++){
            printf("%#+0.3f ",Am[m][n]);
        }
        printf("\n");
    }
*/
}


//Para calcular la salida del Perceptron Multicapa:
//Out=logsig(netLW*tansig(netIW*In+netB1)+netB2)
//logsig(n) = 1 / (1 + exp(-n))
//tansig(n) = 2/(1+exp(-2*n))-1
//In y Out son vectores (entrada y salida)

//multip1[8][1]=netIW[8][150]*In'[150][1]
void primerMultip(int numeroP, int hiddenN, int inputN){
    int i,j=0;

//    for (i=0; i<hiddenN; i++){
//        for (j=0; j<inputN; j++){
//            multip1.at(inputN*i+j) = 0;
//        }
//    for (i=0; i<hiddenN*inputN; i++){
//        multip1.at(i) = 0;
//    }

    for (i=0; i<hiddenN; i++){
        multip1[i] = 0;
        //multip1.at(i) = 0;
        for (j=0; j<inputN; j++){
            if (numeroP==1){
                multip1[i] += netIW[inputN*i+j]*lpc_uno[j];
                //multip1[i] += netIW[inputN*i+j]*inLPC1[j];
                //multip1.at(i) += netIW[inputN*i+j]*inLPC1[j];
            }else if (numeroP==2){
                multip1[i] += netIW[inputN*i+j]*lpc_dos[j];
                //multip1[i] += netIW[inputN*i+j]*inLPC2[j];
                //multip1.at(i) += netIW[inputN*i+j]*inLPC2[j];
            }else{
                multip1[i] += netIW[inputN*i+j]*lpc_ent[j];
                //multip1[i] += netIW[inputN*i+j]*inLPC2[j];
                //multip1.at(i) += netIW[inputN*i+j]*inLPC2[j];
            }
        }
    }

/*
    printf("primerMultip:\n");
    for (i=0; i<hiddenN; i++){
        printf("%f\n",multip1[i]);
    }
*/
}
```

```
//sumop1[8][1]=tansig(netB1[8][1]+multip1[8][1])
void primerSumop(int hiddenN){
    int i=0;
    float suma=0;
    for (i=0; i<hiddenN; i++){
        sumop1[i] = 0;
        //sumop1.at(i) = 0;
    }
    for (i=0; i<hiddenN; i++){
        suma = netB1[i]+multip1[i];
        //suma = netB1[i]+multip1.at(i);
        sumop1[i] = tansig(suma);
        //sumop1.at(i) = tansig(suma);
        //printf("sumop1.at(%d): %f\n",i+1,sumop1.at(i));
    }
//    printf("primerSumop:\n");
//    for (i=0; i<hiddenN; i++){
//        printf("%f\n",sumop1.at(i));
//    }
}

//multip2[3][1]=netLW[3][8]*sumop1[8][1]
void segundoMultip(int outputN, int hiddenN){
    int i,j=0;

//    for (i=0; i<hiddenN; i++){
//        for (j=0; j<inputN; j++){
//            multip2.at(inputN*i+j) = 0;
//        }
//    for (i=0; i<outputN; i++){
//        multip2.at(i) = 0;
//    }

    for (i=0; i<outputN; i++){
        multip2[i]=0;
        //multip2.at(i)=0;
        for (j=0; j<hiddenN; j++){
            multip2[i] += netLW[hiddenN*i+j]*sumop1[j];
            //multip2.at(i) += netLW[hiddenN*i+j]*sumop1.at(j);
        }
    }

//    printf("segundoMultip:\n");
//    for (i=0; i<outputN; i++){
//        printf("%f\n",multip2.at(i));
//    }
}

//sumop2[4][1]=logsig(netB2[4][1]+multip2[4][1])
void segundoSumop(int outputN){
    int i=0;
    float suma=0;
    for (i=0; i<outputN; i++){
        sumop2[i]=0;
        //sumop2.at(i)=0;
    }
    for (i=0; i<outputN; i++){
        suma = netB2[i]+multip2[i];
        //suma = netB2[i]+multip2.at(i);
        sumop2[i]=logsig(suma);
        //sumop2.at(i)=logsig(suma);
    }

    //printf("segundoSumop:\n");
    for (i=0; i<outputN; i++){
        printf("%f\n",sumop2[i]);
    }
}

void resultadoAnn(int numeroP, int outputN){
    float winner = -1;
    int index = 0;
    int i;
    char szText[32]; //for printing on LCD

    //find the best fitting output
    for(i =0; i < outputN; i++)
    {
```

```
        if(sumop2[i] > winner)
        {
            winner = sumop2[i];
            index = i;
        }
    }

    if (sumop2[index]>0.85){
        //output it
        printf("El sistema cree que la palabra es la \n\r \t\t----->| %da |<------\t\t
\n\r",index+1);
        sprintf(szText,"¡Felicidades!\nUd dijo la %da\n",index+1);
        LCD_TextOut(szText);
        //printf("La Red cree que la palabra %d es un \n\r \t\t----->| %d |<------\t\t
\n\r",numeroP,index+1);
    }else{
        printf("El sistema no reconoce la palabra;
P(salida[%d])=%f\n",index+1,sumop2[index]);
        LCD_TextOut("¿? Repita\npor Favor\n");
    }
    //now let's the exact percentages of what it thnks
    /*
    printf("Especificamente  la Red cree que la palabra %d corresponde a un: \n\r\
          | 1 con %d%% de probabilidad |\n\r\
          | 2 con %d%% de probabilidad |\n\r\n\r",numeroP,(int)(sumop2[0]*100),\
           (int)(sumop2[1]*100));
    */
}
```

ANEXO D
Librería en C para el Codec

Código main.c

```
// functions and variables for voice recording and recognition
#include "voice.h"

// global variables
int record_sample_rate;
int record_time_ms;

int main()
{
    record_time_ms = 600;
    show_menu();
    if (!init()) //verify if system is ok
        return 0;
    record_voice(record_sample_rate, record_time_ms);
}
```

Código voice.h

```
// --------------------------------------------------------------------
// Copyright (c) 2007 by Terasic Technologies Inc.
// --------------------------------------------------------------------
//
//Modificacion: 2011

#ifndef VOICE_H_
#define VOICE_H_

#include <stdio.h>
#include <stdlib.h> // malloc, free
#include <string.h>
#include <stddef.h>
#include <unistd.h>  // usleep (unix standard?)
#include <io.h>
#include <math.h>
#include "sys/alt_flash.h"
#include "sys/alt_flash_types.h"
#include "alt_types.h"  // alt_u32
#include "altera_avalon_pio_regs.h" //IOWR_ALTERA_AVALON_PIO_DATA
#include "sys/alt_irq.h"  // interrupt
#include "sys/alt_alarm.h" // time tick function (alt_nticks(), alt_ticks_per_second())
#include "sys/alt_timestamp.h"
#include "sys/alt_stdio.h"
#include "my_types.h"
#include "system.h"
#include <fcntl.h>
#include "debug.h"

#include "AUDIO.h"
#include "LCD.h"
#include "LED.h"
#include "SEG7.h"

//#include "voice.h"
#include "lpc_ann.h"

#ifdef DEBUG_APP
    #define APP_DEBUG(x)    DEBUG(x)
#else
    #define APP_DEBUG(x)
#endif


////////////////////////////////////////////////////////////////////////////
////////////// Internal function prototype & data structure ////////////////////
```

```
////////////////////////////////////////////////////////////////////////////
//================= internal function prototype & data structure definit =====
#define RECORD_BUTTON   0x08
#define PLAY_BUTTON     0x04
#define PLAYRING_BUTTON 0x02
#define RECORD_BLOCK_SIZE   250    // ADC FIFO: 512 byte
#define PLAY_BLOCK_SIZE     250    // DAC FIFO: 512 byte
#define MAX_TRY_CNT         1024
#define LINEOUT_DEFUALT_VOL 0x79  // 0 dB
#define USE_SDRAM_FOR_DATA

#ifndef USE_SDRAM_FOR_DATA
    #define BUF_SAMPLE_NUM     (96000*5)  // 5 second @ 96K
#endif


typedef enum{
    MIC_RECORD,
    LINEIN_RECORD,
    LINEOUT_PLAY
}AUDIO_FUNC;

void show_menu(void);
bool init(void);
void show_power(short sample);
void dump_record_data(alt_u32 *pData, alt_u32 len);

void button_monitor_isr(void* context, alt_u32 id);
bool button_monitor_start(volatile alt_u32 *pPressedMask);
bool init_audio(AUDIO_FUNC audio_func, int record_sample_rate);
void display_time_elapsed(alt_u32 sample_num, int record_sample_rate);

// ui config
bool ui_is_mic_boost(void);
bool ui_is_mic_record(void);
bool ui_is_dac_zero_cross(void);
int  ui_get_sample_rate(void);

void record_voice(int record_sample_rate, int record_time_ms);

#define DEBUG_DUMP  /*printf */

//int message_dump(const char *format[, argument, ...]);
//int message_dump(const char *format[, argument, ...]){}

#endif /*VOICE_H_*/
```

Código voice.c

```
// --------------------------------------------------------------------
// Copyright (c) 2007 by Terasic Technologies Inc.
// --------------------------------------------------------------------
//
//Modificacion: 2011

#include "voice.h"

////////////////////////////////////////////////////////////////////////////
////////////// Internal function implement(body) ///////////////////////////
////////////////////////////////////////////////////////////////////////////
bool ui_is_mic_record(void){
    bool bMicRecord;
    bMicRecord = (IORD(PIO_SWITCH_BASE, 0) & 0x01)?FALSE:TRUE;
    return bMicRecord;
}

bool ui_is_mic_boost(void){
    bool bMicBoost;
    bMicBoost = (IORD(PIO_SWITCH_BASE, 0) & (0x01 << 1))?FALSE:TRUE;
    return bMicBoost;
}

bool ui_is_dac_zero_cross(void){
```

```
    bool bZeroCross;
    bZeroCross = (IORD(PIO_SWITCH_BASE, 0) & (0x01 << 2))?TRUE:FALSE;
    return bZeroCross;
}

int ui_get_sample_rate(void){
    int sample_rate = 96000;
    alt_u32 mask;
    mask = IORD(PIO_SWITCH_BASE, 0);
    mask = (mask >> 3) & 0x07;
    if (mask == 1)
        sample_rate = 48000;
    else if (mask == 2)
        sample_rate = 44100;
    else if (mask == 3)
        sample_rate = 32000;
    else if (mask == 4)
        sample_rate = 8000;
    return sample_rate;
}

void button_monitor_isr(void* context, alt_u32 id){
    volatile alt_u32* pPressedMask = (volatile alt_u32*)context;
    *pPressedMask |= IORD_ALTERA_AVALON_PIO_EDGE_CAP(PIO_BUTTON_BASE) & 0x0F;  // 4-button

    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(PIO_BUTTON_BASE,0);
}

bool button_monitor_start(volatile alt_u32 *pPressedMask){
    bool bSuccess = TRUE;
    // enable interrupt
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(PIO_BUTTON_BASE, 0x0F); // 4-button

    // Reset the edge catpure register
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(PIO_BUTTON_BASE,0);

    // register IRQ
    if (bSuccess && (alt_irq_register(PIO_BUTTON_IRQ, (void *)pPressedMask,
button_monitor_isr) != 0)){
        printf("[SW-MONITOR]register button IRQ fail\r\n");
        bSuccess = FALSE;
    }

    return bSuccess;
}


bool init_audio(AUDIO_FUNC audio_func, int record_sample_rate){
    bool bSuccess = TRUE;
    AUDIO_InterfaceActive(FALSE);
    //
    if (audio_func == MIC_RECORD){
        bool bMicBoost;
        bMicBoost = ui_is_mic_boost();
        AUDIO_SetInputSource(SOURCE_MIC);
        AUDIO_DacEnableSoftMute(TRUE);
        AUDIO_AdcEnableHighPassFilter(FALSE);
        AUDIO_MicBoost(bMicBoost);
        AUDIO_MicMute(FALSE);
        AUDIO_LineInMute(TRUE);
    }else if (audio_func == LINEIN_RECORD){
        AUDIO_SetInputSource(SOURCE_LINEIN);
        AUDIO_DacEnableSoftMute(TRUE);
        AUDIO_AdcEnableHighPassFilter(FALSE);
        AUDIO_MicMute(TRUE);
        AUDIO_LineInMute(FALSE);
        AUDIO_SetLineInVol(0x17, 0x17);  // max 0x1F, min:0; 0x17: 0dB (assume max input is
2.0v rms)
    }else if (audio_func == LINEOUT_PLAY){
        AUDIO_DacEnableSoftMute(TRUE);
        AUDIO_MicBoost(FALSE);
        AUDIO_MicMute(TRUE);
        AUDIO_LineInMute(FALSE);
        AUDIO_DacEnableSoftMute(FALSE);
        //AUDIO_DacDeemphasisControl(DEEMPHASIS_48K);
        AUDIO_DacEnableZeroCross(ui_is_dac_zero_cross());
        AUDIO_SetLineOutVol(LINEOUT_DEFUALT_VOL, LINEOUT_DEFUALT_VOL); // max 7F, min: 30,
0x79: 0 db
```

```
        AUDIO_DacEnableSoftMute(FALSE);
    }

    if (record_sample_rate == 8000)
        AUDIO_SetSampleRate(RATE_ADC8K_DAC8K);
    else if (record_sample_rate == 32000)
        AUDIO_SetSampleRate(RATE_ADC32K_DAC32K);
    else if (record_sample_rate == 48000)
        AUDIO_SetSampleRate(RATE_ADC48K_DAC48K);
    else if (record_sample_rate == 44100)
        AUDIO_SetSampleRate(RATE_ADC44K1_DAC44K1);
    else
        AUDIO_SetSampleRate(RATE_ADC96K_DAC96K);
    //
    AUDIO_InterfaceActive(TRUE);

    return bSuccess;
}


void display_time_elapsed(alt_u32 sample_num, int record_sample_rate){
    // assume sample rate is 48K
    alt_u32 time;
    time = sample_num * 100 / record_sample_rate;
    SEG7_Decimal(time, 0x04);
}

void show_power(short sample){
    static alt_u32 sum = 0;
    static alt_u16   cnt = 0;
    alt_u16 power = 0;
    sum += (sample >= 0)?sample:-sample;
    cnt++;
    if (cnt == 32){
        power = sum >> 14;
        LED_LightCount(power);
        sum = 0;
        cnt = 0;
    }
}


bool init(void){
    bool bSuccess = TRUE;

    SEG7_Clear();
    LCD_Open();
    LCD_TextOut("RECONOCEDOR\nDE PALABRAS\n");
    //LCD_TextOut("Welcome\nAudio Demo\n");

    SEG7_Decimal(0x00000000, 0x00);

    // prepare
    if (!AUDIO_Init()){
        LCD_TextOut("Audio Error\n\n");
        printf("Audio Init Error\r\n");
        bSuccess = FALSE;
    }


    return bSuccess;
}

void dump_record_data(alt_u32 *pData, alt_u32 len){
    short sample_l, sample_r, sample_max = 0;
    alt_u32 data;
    int i;
    //return ;
    for(i=0;i<len;i++){
        data = *pData++;
        sample_l = (short)((data >> 16) & 0xFFFF);
        sample_r = (short)(data & 0xFFFF);
        //printf("[%d]%d/%d\n", i, sample_l, sample_r);
        if (sample_l > 0 && sample_max <  sample_l)
            sample_max = sample_l;
        if (sample_l < 0 && sample_max <  -sample_l)
            sample_max = -sample_l;
        if (sample_r > 0 && sample_max <  sample_r)
```

```
            sample_max = sample_r;
        if (sample_r < 0 && sample_max <  -sample_r)
            sample_max = -sample_r;
    }
    printf("max=%d\n", sample_max);
}


const char szMenu[][128] = {
    "======== Terasic Audio Demo [11/21/2007]==========\n",
    "operation guide:\n",
    "  KEY3: Record Start/Stop (Auto Stop when buffer is full)\n",
    "  KEY2: Play Start/Stop (Audo Stop when no data to play)\n",
    "  SW0:  Audio Source Selection. DOWN-->MIC, UP-->LINE-IN\n",
    "  SW1:  MIC Boost Control. DOWN-->BOOST ON UP-->BOSST OFF\n",
    "  SW2:  Zero-Cross detect for Playing: DOWN-->OFF, UP-->ON\n",
    "  SW5/SW4/SW3: Sample Rate Control.\n",
    "               DOWN/DOWN/DOWN-->96K\n",
    "               DOWN/DOWN/UP---->48K\n",
    "               DOWN/UP/DOWN---->44.1K\n",
    "               DOWN/UP/UP------>32K\n",
    "               UP/DOWN/DOWN---->8K\n",
    "  Status Information:\n",
    "  LCD:  Display status\n",
    "  LED:  Display audio singal strength.\n",
    "  SEG7: Display time elapsed for playing/recording.\n"
    "\n\n"
};

void show_menu(void){
    int i;
    for(i=0;i<sizeof(szMenu)/sizeof(szMenu[0]);i++)
        printf(szMenu[i]);
}

void record_voice(int record_sample_rate, int record_time_ms){
typedef enum{
        ST_STANDY,
        ST_RECODING,
        ST_PLAYING
    }STATE;
    STATE state = ST_STANDY;
    volatile alt_u32 button_mask=0;
    bool bRecordPressed, bPlayPressed, bError = FALSE;
    alt_u32 *pBuf, *pPlaying, *pRecording, RecordLen, PlayLen, data, try_cnt,
buf_sample_size;
    alt_u16 ch_right, ch_left;
    char szText[128];

    /*******Declaracion de las variables auxiliares********/
    alt_u32 direccion = 1;
    alt_u32 dato = 0;
    alt_u16 muestra = 0;
    double daq_xxx[4800]; //(4000: 03-12-2011)

    int muestrasxT=160; //Trama=20ms (267, 15: 03-12-2011)
    int lpcxT=10;
    int totalT=30; //Palabra=60%1.0seg (15, 0.5: 03-12-2011)
    int numeroT=0;

    int longitudE = 300; //(150: 03-12-2011)
    int neuronasO = 10; //(8: 03-12-2011)
    int totalP = 3; //2
    int numeroP = 1;
    /****************Fin de declaracion******************/

//    show_menu();
//    if (!init())
//        return 0;


#ifdef USE_SDRAM_FOR_DATA
    pBuf = (alt_u32 *)SDRAM_U1_BASE;
    buf_sample_size = SDRAM_U1_SPAN/sizeof(alt_u32);
#else
    // alloc memory to store PCM data
    buf_sample_size = BUF_SAMPLE_NUM;
    pBuf = malloc(buf_sample_size * sizeof(alt_u32));
```

```
    if (!pBuf){
        LCD_TextOut("malloc fail\n\n");
        printf("malloc fail\r\n");
        return 0;
    }
#endif
    button_monitor_start(&button_mask);  // button IRQ
    printf("ready\n");
    LCD_TextOut("Presione KEY3, y\ndiga una Palabra\n");

    // test
    record_sample_rate = ui_get_sample_rate();
    RecordLen = buf_sample_size;
    //

    // infinite loop
    while(1){
        //
        bRecordPressed = (button_mask & RECORD_BUTTON)?TRUE:FALSE;
        bPlayPressed = (button_mask & PLAY_BUTTON)?TRUE:FALSE;
        if (bPlayPressed || bRecordPressed)
            button_mask = 0;
        if (state == ST_STANDY){
            if (bRecordPressed){
                bool bMicRecord;
                record_sample_rate = ui_get_sample_rate();
                bMicRecord = ui_is_mic_record();
                AUDIO_FifoClear();
                init_audio(bMicRecord?MIC_RECORD:LINEIN_RECORD, record_sample_rate);
                AUDIO_FifoClear();
                state = ST_RECODING;
                if (bMicRecord){
                    bool bMicBoost = ui_is_mic_boost();
                    printf("MIC %srecording (sample rate = %d)...\r\n", bMicBoost?"Boost
":"", record_sample_rate);
                    sprintf(szText,"MIC %s (%dK)\nRecording...\n",bMicBoost?"Boost":"",
record_sample_rate/1000);
                    LCD_TextOut(szText);
                }else{
                    printf("LINE-IN recording (sample rate = %d)...\r\n",
record_sample_rate);
                    sprintf(szText,"LINE-IN (%dK)\nRecording...\n",record_sample_rate/1000);
                    LCD_TextOut(szText);
                }
                pRecording = pBuf;
                RecordLen = 0;
            }else if (bPlayPressed){
                if (RecordLen == 0){
                    printf("Please record before play audio\r\n");
                    LCD_TextOut("Record First\nBefore Play\n");
                }else{
                    bool bZeroCross = ui_is_dac_zero_cross();
                    AUDIO_FifoClear();
                    init_audio(LINEOUT_PLAY, record_sample_rate);
                    state = ST_PLAYING;
                    printf("playing (sample rate = %d)...\r\n", record_sample_rate);
                    sprintf(szText,"Playing (%dK)\n%s...\n",record_sample_rate/1000,
bZeroCross?"Zero Cross":"");
                    LCD_TextOut(szText);
                    pPlaying = pBuf;
                    PlayLen = 0;
                }
            }
            bError = FALSE;
        }else if (state == ST_RECODING){

            //Limite de la cantidad de muestras grabadas a 4000 (fs=8000)
            //Se debe establecer el SW5 de  la tarjeta en 1 logico (arriba)
            if (bRecordPressed || (RecordLen >= (record_sample_rate/1000)*record_time_ms) ||
bError){ //(0.5)
            //if (bRecordPressed || (RecordLen >= buf_sample_size) || bError){

                // stop record
                printf("record %d samples\n", (int)RecordLen);
                sprintf(szText, "record\n%d samples\n", (int)RecordLen);
                LCD_TextOut(szText);

                /*****Procesamiento de la señal adquirida******/
```

```
//                while ((direccion > 0) && (direccion < 4000)){
//                    printf("Ingrese el numero de muestra que desea ver (0 para salir): ");
//                    scanf("%lu",&direccion);
//                    dato = *(pRecording+direccion-4000);
//                    printf("La muestra solicitada es: %u\n",(unsigned)(dato & 0xFFFF));
//                    //printf("La muestra solicitada es: %lu I=%u
D=%u\n",dato,(unsigned)((dato>>16) & 0xFFFF),(unsigned)(dato & 0xFFFF));
//                }

                //printf("Iniciando carga de valores grabados con el codec...\n");
                for(direccion=0; direccion<(record_sample_rate/1000)*record_time_ms;
direccion++){ //(4800: 04-12-2011)
                    dato = *(pRecording+direccion-
(record_sample_rate/1000)*record_time_ms+1); //(4799: 04-12-2011)
                    muestra = (unsigned short)(dato & 0xFFFF);
                    //printf("%u\n", muestra);
                    if(muestra < 32768){ //(32768: 04-12-2011)
                        daq_xxx[direccion]=(float)((unsigned)((dato & 0xFFFF)+1476))/8191.0;
//0, 32767
                        //printf("%f\n",(float)((unsigned)((dato & 0xFFFF)+1476))/8191.0);
//0, 32767
                        //printf("%u\n",(unsigned)(dato & 0xFFFF));
                    }else if(muestra >= 32768){ //(32768: 04-12-2011)
                        daq_xxx[direccion]=(float)((short)((dato & 0xFFFF)-64060))/8192.0;
//65536, 32768
                        //printf("%f\n",(float)((short)((dato & 0xFFFF)-64060))/8192.0);
//65536, 32768
                        //printf("%hi\n",(short)((dato & 0xFFFF)-64060));
                    }
                    //printf("%f\n",daq_xxx[direccion]);
                }
                //printf("Fin de la carga de valores grabados con el codec...\n");

    //printf("Iniciando codificacion de voz grabada...\n");
    for (numeroT=0; numeroT<totalT; numeroT++){
        cargarTrama(muestrasxT,numeroT,0,daq_xxx);
        calcularRs(lpcxT,muestrasxT);
        levinsonDurbin(lpcxT);
        resultadoLpc(lpcxT,numeroT);
    }
    //printf("Fin de la codificacion de voz grabada...\n");

    numeroP=1;
    while(numeroP != 0){
        //printf("Dar el numero de palabra almacenada (1 - %d), o 0 salir: ",totalP);
        //scanf("%d",&numeroP);
        numeroP=0;
        //if(numeroP == 0)
            //break;

        primerMultip(numeroP,neuronasO,longitudE);
        primerSumop(neuronasO);
        segundoMultip(totalP,neuronasO);
        segundoSumop(totalP);
        resultadoAnn(numeroP,totalP);
    }
                /**************Fin del procesamiento**************/

                state = ST_STANDY;
                //usleep(1000000);
                //state = ST_PLAYING;
                LED_AllOff();
                dump_record_data(pBuf, RecordLen);

            }else{
                // continue recoding
                int i = 0;
                while ((i < RECORD_BLOCK_SIZE) && (RecordLen < buf_sample_size)){
                    try_cnt = 0;
                    while (!AUDIO_AdcFifoNotEmpty() && try_cnt < MAX_TRY_CNT){ // wait while
empty
                        try_cnt++;
                    }
                    if (try_cnt >= MAX_TRY_CNT){
                        bError = TRUE;
                        break;
                    }
```

```
                    AUDIO_AdcFifoGetData(&ch_left, &ch_right);
                    data = (ch_left << 16) | ch_right;
                    *pRecording++ = data;
                    RecordLen++;
                    i++;
                    show_power(data & 0xFFFF);
                }  // while
                display_time_elapsed(RecordLen, record_sample_rate);
            }
        }else if (state == ST_PLAYING){
            //if (PlayLen >= RecordLen){
            if (bPlayPressed || (PlayLen >= RecordLen) || bError){
                // stop playing
                printf("Play Stop %s\r\n", bError?"(Error)":"");
                LCD_TextOut("Standby\n\n");
                state = ST_STANDY;
                LED_AllOff();
            }else{
                // continue playing
                int i = 0;
                while ((i < PLAY_BLOCK_SIZE) && (PlayLen < RecordLen)){
                    try_cnt = 0;
                    while (!AUDIO_DacFifoNotFull() && try_cnt < MAX_TRY_CNT){ // wait while
full
                        try_cnt++;
                    }
                    if (try_cnt >= MAX_TRY_CNT){
                        bError = TRUE;
                        break;
                    }
                    data = *pPlaying++;

                    //data = 0;
                    ch_left = data >> 16;
                    ch_right = data & 0xFFFF;
                    AUDIO_DacFifoSetData(ch_left, ch_right);
                    i++;
                    PlayLen++;
                    show_power(data & 0xFFFF);
                    //printf("[%d] %d/%d\n", PlayLen, (short)((data >> 16) & 0xFFFF),
(short)(data & 0xFFFF));
                }
                display_time_elapsed(PlayLen, record_sample_rate);
            }
        }
    }
}
```