UNIVERSIDAD DE SEVILLA

Escuela Superior de Ingenieros de Sevilla

Proyecto Fin de Carrera

Sistema de Reconocimiento de Palabras Aisladas Sobre una FPGA

Estudio E Implementación

Ingeniero de Telecomunicación

Autora: Elena Godino Llani
Tutor: Miguel Ángel Aguirre Echanove
Julio 2001

Capítulo

1

Introducción

ÍNDICE DEL CAPÍTULO

- Introducción
- Sistemas de reconocimiento del habla: Estado del arte, tecnologías del habla, sistemas y circuitos de reconocimiento del habla, y dificultades del reconocimiento del habla
- Objetivos, restricciones y estructura del proyecto
- Generación de la voz

En este capítulo se exponen los objetivos del proyecto y se describen, sin entrar en detalles, la solución adoptada y los pasos seguidos en la misma.

INTRODUCCIÓN

El objetivo del proyecto es el diseño un sistema de reconocimiento del habla completamente hardware. El prototipo del sistema se ha desarrollado sobre una FPGA Virtex 300, y haciendo uso del sistema de prototipado rápido HADES que permite la utilización de las características para el depurado de sistemas hardware de las FPGAs de última generación.

SISTEMAS DE RECONOCIMIENTO DEL HABLA: ESTADO DEL ARTE

El interés que durante décadas han despertado los sistemas de reconocimiento y síntesis del habla, no es otro que el lograr un interfaz hombre-máquina más intuitivo para el ser humano, además de otras ventajas o necesidades que nos podría ofrecer un interfaz basado en la voz. Así, por ejemplo, tradicionalmente las comunicaciones hombre-máquina se basan en un teclado y una pantalla, de forma que para comunicarnos con la máquina necesitamos de al menos una mano libre, así como mantener la vista en la pantalla.

Los grandes avances en nuestra civilización en la segunda mitad de este siglo están estrechamente ligados al desarrollo de las tecnologías de la información. Cada vez más, en nuestra sociedad el acceso a la información tiene mayor importancia. El papel relevante que van tomando las telecomunicaciones y la informática en el manejo y acceso a esta información hace que gran parte se nos presente en forma visual y oral (p.e. sistemas multimedia). Una de las formas de comunicación más eficientes y naturales que poseemos es el habla. Dotar a una máquina de la capacidad de producir y comprender mensajes orales es, no sólo una comodidad, sino un objetivo primordial para la plena integración del mundo tecnológico en la sociedad.

El término "tecnologías del habla" engloba todos los procesos necesarios para la realización de un interfaz oral, entendiendo como interfaz oral aquel dispositivo que nos

permite comunicarnos con una máquina o nos ayuda en la comunicación entre humanos. Por tanto, la Tecnología del Habla engloba a un amplio conjunto de conocimientos y procedimientos de actuación sobre la información representada en la señal de voz. Conocimientos que se articulan con un alto grado de dificultad y especialización, ya que pertenecen a un marco científico-técnico *multidisciplinar*, donde se dan cita diferentes ramas del saber como son: fisiología, acústica, lingüística, procesado de señal, inteligencia artificial, teoría de la comunicación y de la información, y ciencia de la computación.

En el siguiente subapartado "un poco de historia" se presentan cronológicamente los acontecimientos ocurridos en el marco de las tecnologías del habla hasta llegar a la situación presente.

➲ Un poco de historia

Las tecnologías del habla se encuentran en pleno desarrollo e introducción en nuestro entorno cotidiano. El habla, medio de comunicación por excelencia entre seres humanos, comienza a ser utilizada como medio de comunicación con la tecnología desarrollada por el ser humano. La capacidad de interactuar con las máquinas mediante el habla ha sido una constante en el desarrollo tecnológico de la humanidad.

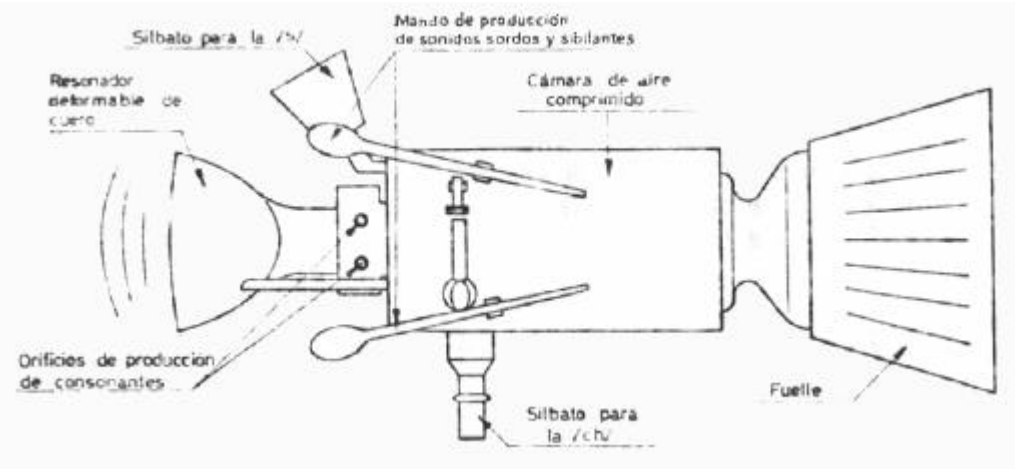


Figura 1.1. Máquina parlante de Wolfgang Von Kempelen.

A partir de finales del siglo XVIII, Wolfgang Von Kempelen, en 1791, construyó un rudimentario sintetizador de voz capaz de pronunciar frases cortas mediante un ingenio mecánico. En el siglo XIX se comenzaron a desarrollar las bases matemáticas en las que se basa el análisis de la señal de voz, como el análisis de Fourier. Durante este siglo se realizaron una serie de inventos que potenciaron la utilización del habla como método de comunicación. El teléfono permitió utilizar la voz para comunicaciones instantáneas y a larga distancia; el fonógrafo permitió grabar registros de voz, siendo lo que podríamos denominar el primer sistema de respuesta oral. Ya en el siglo XX, en el año 1939 se presentó una versión electrónica del sintetizador de Kempelen. El sistema denominado VODER consistía en un teclado muy elaborado que permitía controlar la articulación y generación de sonidos vocálicos y consonánticos. Este sistema pionero sembró las bases de los modernos sintetizadores de voz, basados actualmente en tecnología digital, pero compartiendo la misma teoría sobre el modelo de producción del habla. Ya desde los años 30 podemos decir que comenzó la investigación sobre la señal de voz con el desarrollo de sistemas prácticos de transmisión digital mediante modulación PCM. Con el advenimiento de las comunicaciones digitales y los sistemas de audio digital, se planteó el problema de la codificación eficiente de la señal de voz para reducir al máximo la velocidad de transmisión y el número de bits necesarios para la codificación binaria de la señal de voz. Por el contrario, hasta bien entrado el siglo XX no se tuvo un conocimiento suficiente sobre el

comportamiento del sistema de percepción auditiva del ser humano. Sobre 1940 se desarrollaron los primeros experimentos para estudiar el comportamiento de la membrana basilar.

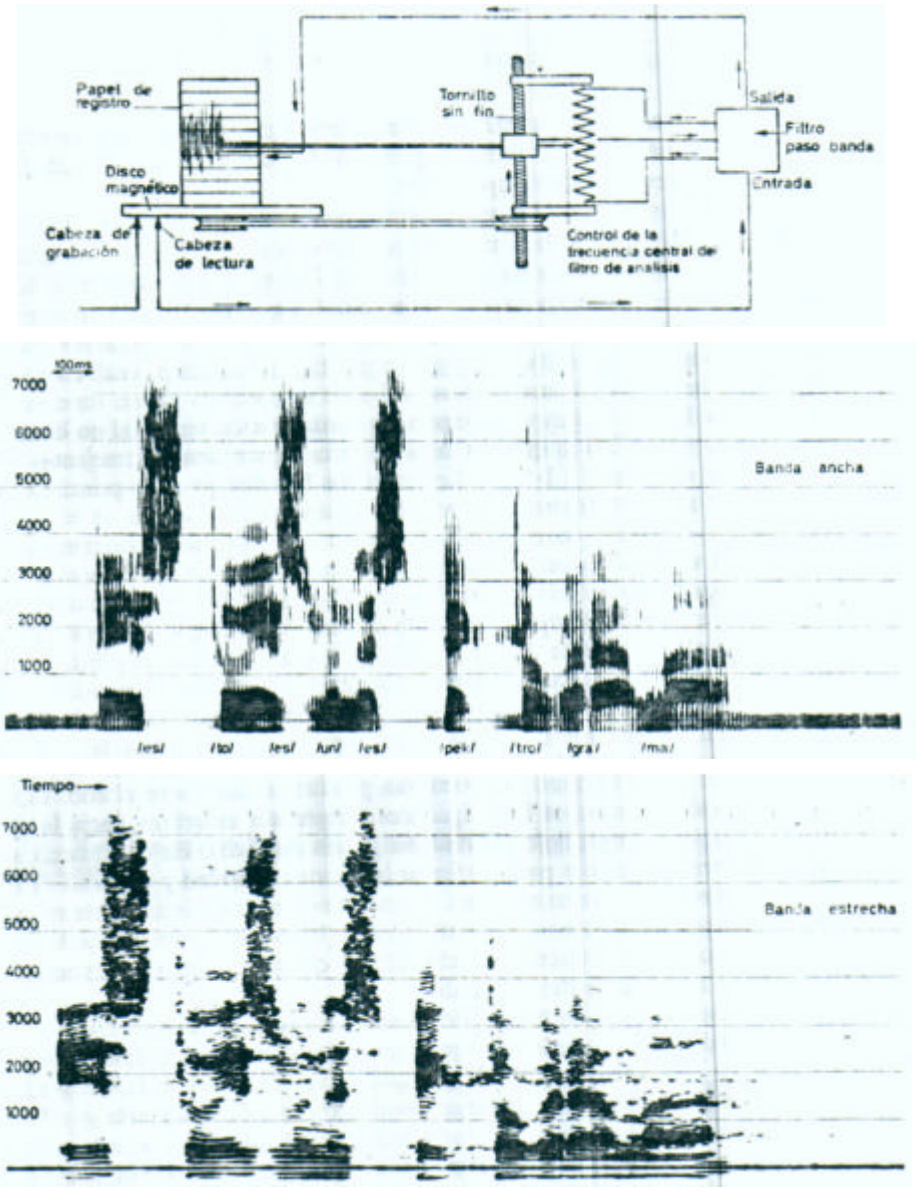


Figura 1.2. Espectrógrafo y espectros de banda ancha y estrecha (Lab. fonética de la Universidad autónoma de Barcelona)

Un hito muy importante dentro de la investigación y desarrollo de las tecnologías del habla fue el invento del espectrógrafo de voz en la década de 1940. El espectrógrafo de voz permitía obtener un registro gráfico de la evolución de la energía de la señal de voz en diferentes bandas frecuenciales. Esta visión tiempo-frecuencia de la señal de voz conjuntamente con los estudios sobre el sistema auditivo humano abrió el campo a la investigación en una de las áreas más fascinantes de las tecnologías del habla: **el reconocimiento automático del habla**. De hecho, en 1952 se presentó un sistema electrónico de reconocimiento de voz en los laboratorios Bell que era capaz de reconocer, dentro de un margen aceptable de error, los dígitos pronunciados en inglés.

La gran revolución en las tecnologías del habla se comenzó a gestar con el advenimiento de los computadoras digitales y la informática. Desde mediados de los años 60 se comenzaron a desarrollar algoritmos de procesado de señal, a desarrollar las bases de

la inteligencia artificial y comenzaron a aparecer los primeros desarrollos e investigaciones en sistemas de reconocimiento automático del habla utilizando computadoras digitales. Pero es en la década de los 70 cuando se da el impulso definitivo con la financiación de grandes proyectos de reconocimiento del habla por parte de países como EE.UU., Japón, Francia, Alemania, etc. En Noviembre de 1971 la Oficina de tecnología para el procesado de la información de la ARPA (Advanced Research Projects Agency of the USA Department of Defense) iniciaba un ambicioso proyecto de investigación con el objetivo de desarrollar varios sistemas de comprensión automática del habla continua para varios locutores cooperantes que hablasen "General American dialect". Cinco años más tarde y una vez finalizado el proyecto inicial, el resultado final fue en cierta forma decepcionante ya que los objetivos no llegaron a alcanzarse pero si que sirvieron de toma de contacto con la realidad del problema y sentaron las bases para posteriores investigaciones. Una de las conclusiones finales, en las que se trataba de evaluar la capacidad computacional necesaria para abordar con garantías el funcionamiento en tiempo real de estos sistemas, situaba la capacidad mínima necesaria alrededor de los 100 MIPS (1MIP=1 Millón de Instrucciones por Segundo).

El requisito de disponer de una capacidad computacional de este orden con sistemas informáticos de bajo coste (Ordenadores Personales, Estaciones de Trabajo...) no se ha alcanzado hasta la actualidad, y aún de una forma parcial, gracias a la aparición en el mercado de forma masiva de la tercera generación de microprocesadores especializados en el procesado digital de señal (DSP-Digital Signal Processors). Los avances en la tecnología de integración de circuitos integrados (VLSI), la aportación de nuevos conceptos en arquitectura de computadores (Arquitecturas RISC y Harward) junto a las decisiones de diseño para operar en un entorno de tiempo real incentivaron la aparición de DSPs de tercera generación.

Esta revolución tecnológica de la década de los 80 y principios de los 90, conjuntamente con la investigación básica realizada durante esta década por multitud de grupos de investigación en todo el mundo en el campo de las tecnologías del habla y de la inteligencia artificial han construido los cimientos del desarrollo actual en sistemas de comunicación oral hombre-máquina, encontrándonos en las puertas de una gran revolución en el desarrollo de sistemas basados en tecnologías del habla.

Hoy en día, gracias a la rápida evolución de las técnicas de procesado digital de señal y el creciente incremento en las prestaciones de cálculo de los ordenadores personales y procesadores digitales de señal, las tecnologías del habla, basadas principalmente en las funciones de codificación, síntesis y reconocimiento, comienzan a ser una realidad como interfaces entre hombre y máquina.

➲ Tecnologías del habla

En esta sección se presentan los principales avances obtenidos en los últimos años en el ámbito de la Tecnología del Habla. Se presta especial atención a las principales líneas de trabajo orientadas hacia el diseño de sistemas de Reconocimiento del Habla, por ser la base del proyecto, aunque dada la extensión del tema nos limitaremos a resumir y estructurar el fondo común de los principales desarrollos e innovaciones.

Asistimos en nuestros días a una progresiva proliferación de aplicaciones basadas en el proceso automático del lenguaje hablado. Así, son cada vez más comunes: las interfaces hombre-máquina controladas por voz, los sistemas de respuesta vocal interactiva, y la automatización de sistemas telefónicos. La ciencia en la que se basan todos ellos es la Tecnología del Habla, que se estructura en cuatro tecnologías básicas principales:

- El Reconocimiento de Voz o Reconocimiento del Habla

Es el proceso de conversión de un mensaje hablado en texto, que permita al usuario una comunicación con la máquina. Se trata de la tecnología que mayor avance ha sufrido en los últimos cinco años, pasándose de poder reconocer tan sólo a un hablante, dentro de un vocabulario muy limitado, a prototipos capaces de reconocer a cualquier hablante sobre vocabularios flexibles compuestos por miles de palabras.

- La Conversión Texto-Voz

Se ocupa de la generación de mensajes hablados mediante la simulación del proceso de lectura de un texto escrito almacenado en formato electrónico. Esta tecnología también cuenta en la actualidad con unos excelentes resultados, planteándose como el mayor reto a conseguir, aumentar la calidad de la voz sintetizada.

- El Reconocimiento de Locutores

Es el proceso de identificación o verificación de la identidad del hablante de forma automática a partir de la señal de voz. El grado de desarrollo de esta tecnología es inferior al de las anteriores, quizás como consecuencia de lo crítico que puede ser las aplicaciones en las que se inserte.

- La Codificación de Voz

Es una tecnología que ha alcanzado un grado de madurez muy elevado, contando en la actualidad con un número importante de procedimientos estandarizados. Su objetivo es la búsqueda de representaciones eficientes en formato digital de la señal de voz para su almacenamiento y/o transmisión, persiguiendo obtener la mayor calidad posible, para el menor número de bits por muestra.

☑ *Estado del arte en el reconocimiento del habla*

Las principales áreas de trabajo que intervienen en el diseño y especificación de sistemas de Reconocimiento del Habla actuales son las siguientes:

- Proceso de la señal de voz.
- Técnicas de reconocimiento de patrones.
- Diferentes estilos de habla.
- Dependencia del locutor.
- Vocabulario de reconocimiento.
- Tarea de reconocimiento.
- Bases de datos para entrenamiento y reconocimiento.

☑ *Proceso de la señal de voz*

La primera operación que debe realizar un reconocedor es procesar la señal de voz de entrada al sistema, con objeto de extraer la información acústica relevante para la tarea que debemos realizar. En este primer nivel del sistema son dos los interrogantes a resolver:

- ¿Qué rasgos o características extraer?
- ¿Qué efectos perturbadores pueden acompañar a la voz? y ¿cómo eliminarlos?

La respuesta a la primera cuestión ha venido precedida de un largo proceso de investigación sobre diferentes procedimientos de parametrización de la voz. Planteándose como solución actual más extendida una parametrización de la envolvente espectral que incluya consideraciones perceptuales a partir del funcionamiento del oído. Para reducir el número de parámetros posibles, la parametrización se combina con la utilización de técnicas discriminativas, seleccionándose el subconjunto con los parámetros más eficientes o distintivos.

En cuanto a la segunda de las preguntas planteadas, la presencia de efectos perturbadores en la señal de entrada, ha generado tres líneas de trabajo principales:

- Detección robusta de voz: Apareciendo innumerables procedimientos de discriminación entre voz o ruido (silencio) para diferentes tipos de ruido.
- Reducción de ruido: Distinguiéndose procedimientos que actúan directamente sobre la señal de voz y procedimientos que buscan compensar el efecto del ruido sobre la parametrización de la voz.
- Cancelación de ecos: Incorporando técnicas de filtrado adaptativo que permitan al usuario comenzar a hablar mientras, desde el terminal remoto, se le está comunicando un mensaje que puede provocar un eco en la voz que entra al reconocedor.

☑ *Técnicas de reconocimiento de patrones*

El reconocimiento de patrones es la técnica más específica de todo sistema de reconocimiento. De ahí que muchos reconocedores se identifiquen a partir de la técnica de reconocimiento de patrones que incorporan. A partir de la representación paramétrica de la voz, este módulo realiza un proceso de clasificación utilizando una serie de patrones. Estos patrones se obtienen en una fase de entrenamiento del sistema y son representativos de un conjunto de unidades lingüísticas (palabras, sílabas, sonidos, fonemas). La peculiaridad más característica de este proceso, que marca su dificultad, es la variabilidad temporal que puede presentar una misma unidad lingüística al ser producida por diferentes modos y/o velocidades de habla. Así pues, las primeras técnicas de reconocimiento de patrones utilizadas fueron las basadas en un Alineamiento Temporal a través de algoritmos de Programación Dinámica, técnicas DTW. Posteriormente se recurrió a la mayor flexibilidad que el modelado de procesos estocásticos permite para representar secuencias de duración variable. Concretamente la alternativa a las técnicas DTW fueron los Modelos Ocultos de Markov, (HMM), que pueden verse como una generalización de algoritmos DTW y han demostrado mejores prestaciones en multitud de sistemas de reconocimiento. También hay que mencionar que, recientemente, la potencia y excelentes capacidades de clasificación mostradas por las denominadas Redes Neuronales Artificiales (RN) las sitúa como posible alternativa frente a los HMM. Hasta el momento las Redes Neuronales han permitido obtener los mejores resultados en Reconocimiento de Locutores, sin embargo en Reconocimiento del Habla encuentran como mayor dificultad la forma de afrontar la variabilidad temporal del habla.

☑ *Modelado dependiente del estilo del habla*

Se distinguen tres modos fundamentales de hablar frente a un sistema de reconocimiento:

- Palabras aisladas

Supone que el usuario pronuncia una sola palabra o comando que el sistema deberá reconocer.

- Habla conectada

El usuario pronuncia de forma fluida un mensaje utilizando un vocabulario muy restringido; el ejemplo más típico sería la pronunciación de un número telefónico.

- Habla continua

Corresponde al modo más avanzado de funcionamiento de un reconocedor, y supone la pronunciación de frases de forma natural para un vocabulario amplio de palabras.

Además de los tres modos fundamentales anteriores, los reconocedores de voz tienen que afrontar, para un modelado robusto del habla, los tres aspectos siguientes:

- Reconocimiento en contexto o "word spotting"

Técnica especialmente utilizada en reconocimiento de palabras aisladas, encaminada a detectar la presencia de palabras del vocabulario a reconocer en el contexto de otras palabras o pronunciaciones. La mayoría de las veces el contexto es resultado de la dificultad que encuentra el usuario para ceñirse a la pronunciación de una única palabra aislada. En otras ocasiones, el reconocimiento en contexto es la solución apropiada para robustecer el reconocimiento en ambientes acústicamente hostiles; por ejemplo, cuando la palabra que pronuncia el usuario viene acompañada de ruidos telefónicos, urbanos, etc. En cualquier caso, se trata de una técnica importante para robustecer los sistemas en aplicaciones reales.

- Rechazo

Otro efecto de la presencia de sonidos indeseados (ruidos, sonidos o palabras fuera del vocabulario), es provocar el reconocimiento de palabras que realmente no han sido pronunciadas. Los procedimientos conocidos como técnicas de rechazo tienen como objetivo permitir incluir entre los resultados de reconocimiento la identificación de esos sonidos indeseados. Nos encontramos ante un problema de gran importancia de cara a la operatividad de un sistema de reconocimiento, que aún hoy por hoy no cuenta con una clara solución.

- Múltiples candidatos

El proceso de reconocimiento de patrones que realiza un reconocedor se basa en identificar el patrón que ofrezca la puntuación más alta para decidir cuál es la mejor palabra o secuencia de palabras reconocida. Este proceso se basa en información exclusivamente acústica, sin tener en consideración otras posibles fuentes de conocimiento que podrían utilizarse para completar las puntuaciones de las diferentes palabras o secuencias candidatas. En la mayoría de los casos, la aplicación en que se encuentra el reconocedor es la que posee la información necesaria que permitiría seleccionar entre varias hipótesis de reconocimiento. Pensemos, por ejemplo, en una aplicación basada en el reconocimiento de números telefónicos; en esa situación, ante las dos hipótesis mejores de reconocimiento, una compuesta de cinco dígitos y otra de siete, la aplicación seleccionaría esta última independientemente de quién obtuviese la mayor puntuación "acústica" en el proceso de clasificación. Los procedimientos que permiten a un reconocedor disponer de la flexibilidad que supone manejar N hipótesis de reconocimiento se denominan N-best.

☑ *Dependencia del locutor*

El grado de dependencia del locutor define si el sistema incorpora patrones de unidades lingüísticas adaptados a un locutor determinado, y, por tanto, sólo funcionará correctamente para él, o si los patrones pretenden ser válidos para cualquier hablante. En el primer caso se habla de reconocimiento dependiente del locutor, mientras que en el segundo de reconocimiento independiente del locutor. A parte de las actividades específicas que se desarrollan para sistemas dependientes e independientes del locutor, existe un importante número de esfuerzos dirigidos a conseguir la adaptación de un reconocedor a un locutor específico con la menor cantidad de voz posible.

☑ *Dependencia del vocabulario*

Las prestaciones de un reconocedor dependen fuertemente del tamaño y grado de dificultad del vocabulario. Es decir, del número de palabras que el sistema es capaz de reconocer, y de la mayor o menor dificultad de su reconocimiento en base a las relaciones de similitud fonética entre palabras. En la actualidad se diseñan sistemas tanto para vocabularios pequeños (menos de 50 palabras) y medios (entre 50 y 500 palabras), como para grandes vocabularios (más de 500 palabras), llegándose hasta 50.000 palabras para aplicaciones de dictado o acceso a bases de datos mediante lenguaje natural.

Otra importante dimensión, en relación con el vocabulario, es la que afecta a la distinción entre vocabularios fijos y flexibles. Una determinada aplicación, cuando esté reconociendo, siempre actuará sobre un vocabulario fijo. Pero en muchos casos ese vocabulario deberá variarse o actualizarse para eliminar y/o dar cabida a nuevas palabras. Tradicionalmente, una variación del vocabulario suponía comenzar un largo y costoso proceso de recogida de una nueva base de datos y re-entrenamiento de los patrones del sistema. En la actualidad hay diversas aproximaciones para conseguir un sistema con vocabulario flexible, que no necesite re-entrenarse para cada nuevo vocabulario.

☑ *Gramáticas del reconocimiento*

Según aumenta el número de palabras del vocabulario, el número de posibles combinaciones crece exponencialmente. Por tanto, se hace imprescindible la incorporación de restricciones, en cuanto al número de combinaciones válidas, según la tarea en que se inserte el sistema. Restricciones que suelen incorporarse en forma de gramáticas basadas en reglas sintácticas y/o semánticas destinadas a reducir el número de palabras susceptibles de ser reconocidas en cada momento. La medida utilizada para definir el grado de dificultad que supone una determinada tarea es la denominada perplejidad, de modo que un nivel de perplejidad bajo supone que en cada momento el número de posibles palabras candidatas es bajo, mientras que una perplejidad alta supone que ese número es alto, y consiguientemente el reconocimiento será más difícil.

➲ Principales sistemas de Reconocimiento del Habla

A principios de los años 80 más de 10 compañías de Estados Unidos ofrecían reconocedores de palabras aisladas dependientes del locutor con un vocabulario de hasta 300 palabras. Sólo las firmas VERBEX y NEC ofrecían un sistema independiente del locutor con posibilidades de reconocimiento de palabras conectadas. En ese momento, la situación del Reconocimiento del Habla podría resumirse como:

- Reconocedores de palabras aisladas dependientes del locutor como tecnología asentada.
- Reconocedores independientes del locutor y reconocedores de palabras conectadas como tecnologías nacientes.

Por otro lado, debido a las limitaciones en el ancho de banda y la sensibilidad frente al ruido, sólo un número muy reducido de estos reconocedores trabajaban sobre la línea telefónica. En esta época sólo encontramos en la literatura referencia a tres aplicaciones del Reconocimiento del Habla dentro del ámbito de las telecomunicaciones:

✓ Dos prototipos de reconocedores de palabras aisladas independiente del locutor para:

- Marcación por voz en la red privada.
- Reconocimiento de letras.

✓ Un reconocedor de palabras aisladas dependiente del locutor aplicado a la marcación de números de teléfono por voz.

Como consecuencia de las investigaciones en Reconocimiento del Habla llevadas a cabo durante los últimos diez años, actualmente son muchas las compañías que cuentan con reconocedores de palabras aisladas (dígitos mas un número reducido de comandos) independiente del locutor. Sistemas diseñados, en su mayor parte, para incorporarse en aplicaciones de telecomunicación.

Las prestaciones obtenidas para palabras aisladas, vocabularios con un número de palabras inferior a 200, e independencia del locutor, dependen en gran medida de las características acústicas de las palabras del vocabulario. Así, mientras que el reconocimiento de los diez dígitos puede presentar una tasa de error de palabra inferior al 2 por 100, el reconocimiento de 39 caracteres alfanuméricos (dígitos y letras) en inglés supone un 7% de error, y el de 129 palabras dentro del ámbito de las compañías aéreas un 2,9 por 100.

El reconocimiento de dígitos conectados es otra de las tareas con mayores posibilidades de utilización en diversas aplicaciones. Los resultados que proporcionan los mejores sistemas desarrollados para el inglés por los Laboratorios Bell de AT&T y por el Centro de Investigación Informática de Montreal (CRIM), suponen una tasa de error de palabra inferior al 1 por 100 cuando trabajan en condiciones de laboratorio. Sin embargo, sobre la red telefónica las prestaciones se reducen de forma importante hasta tasas de error de palabra cercanas al 4 por 100.

La evolución de los sistemas de reconocimiento con mayor proyección de futuro: reconocimiento para grandes vocabularios y habla continua, tiene como mejores representantes a los siguientes sistemas experimentales:

- ✓ BYBLOS

Desarrollado por BBN. Byblos es el nombre de una ciudad fenicia donde se descubrió la primera muestra de escritura fonética. Este detalle marca el énfasis que se pone actualmente en desarrollar sistemas sobre una base fonética. Aunque se trata de un sistema dependiente de locutor, este sistema ha aportado un nuevo y eficiente procedimiento de reconocimiento rápido (búsqueda rápida) basado en algoritmos N-best.

- ✓ TANGORA

Desarrollado en IBM. También se trata de un sistema dependiente de locutor para grandes vocabularios. Su principal interés es un proceso de adaptación a un nuevo locutor que requiere 20 minutos para leer 100 frases de 1.200 palabras, 700 de las cuales son distintas.

- ✓ SPHINX-II

Desarrollado en la Universidad de Carnegie-Mellon (CMU). Es un sistema pionero en reconocimiento independiente de locutor para grandes vocabularios. Su más reciente innovación es el procedimiento VOCIND [11] para hacer al sistema independiente del vocabulario.

- ✓ LINCOLN

Desarrollado en el laboratorio del mismo nombre. Su principal aportación es el modelado de voz rápida, con emoción, tensión, etc.

- ✓ DECIPHER

Desarrollado en SRI International. Su principal novedad fue la representación detallada de aspectos fonéticos importantes, tales como la coarticulación entre palabras.

✓ ATR HMM-LR

Sistema japonés desarrollado en ATR. Está basado en procedimientos específicos de modelado de sonidos que no utilizan estructuras intermedias de modelos de fonema o palabra.

✓ CSELT

Desarrollado en el centro italiano del mismo nombre. Su principal innovación es un sistema de búsqueda rápida basada en un primer descifrado fonético simple y rápido, seguido por una búsqueda más detallada.

✓ PHILIPS

Desarrollado por la empresa del mismo nombre. Es un sistema pionero en procesos de reconocimiento rápidos para habla continua y vocabularios de hasta 10.000 palabras.

✓ Sistemas telefónicos de AT&T y Bell Northern Research (BNR).

Ambos sistemas incorporan procedimientos específicos para aplicaciones de automatización de servicios telefónicos.

Para disponer de una idea general de las prestaciones proporcionadas por los sistemas anteriores, presentamos algunos datos orientativos sobre evaluaciones realizadas para tres bases de datos correspondientes a tres tareas diferentes dentro del programa DARPA (Defence Advanced Research Projects Agency) de Estados Unidos. Hay que dejar claro que las tres bases de datos se componen de voz grabada en condiciones de laboratorio. Una breve descripción de cada base de datos sería:

- Base de datos correspondiente a la gestión de recursos navales (Naval Resource Management), con un vocabulario de 991 palabras.
- Base de datos de información sobre vuelos de líneas aéreas ATIS (Air Travel Information System), con vocabulario de 1.800 palabras.
- Base de datos leída del Wall Street Journal, con un vocabulario de 20.000 palabras.

Cada una de las tareas de reconocimiento cuenta con un nivel de restricción gramatical diferente, siendo mayor para la base de datos ATIS, algo inferior para la de recursos navales, y un orden de magnitud inferior para la del Wall Street Journal.

Entre 1987 y 1991 se consiguió reducir la tasa de error de reconocimiento a nivel de palabra del 20 por 100 al 4,5 por 100, sobre la base de datos de gestión de recursos navales. Para la base de datos ATIS los mejores resultados suponen una tasa de error a nivel de palabra del 4 por 100. Desde 1990 se está trabajando con la base de datos del Wall Steet Journal, que presenta un nivel de dificultad muy superior a las otras dos, consiguiéndose una tasa de error próxima al 13 por 100.

Hay que tener presente que, aún en los casos de un error de palabra del 4 por 100, el error a nivel de frase será próximo al 20 por 100. Por tanto, cara a la incorporación de esta tecnología en aplicaciones reales se hace imprescindible complementar el reconocimiento con técnicas de proceso de lenguaje natural, que permitan extraer información del mensaje hablado aún a partir de frases con errores de reconocimiento. A pesar de esto son muchos los desarrollos que a nivel de prototipo muestran las posibilidades futuras del Reconocimiento del Habla.

➲ Circuitos en el mercado(1995)para el reconocimiento del habla. Resumen de características y prestaciones

En este apartado se exponen los algunos circuitos existentes para el reconocimiento del habla en el mercado en 1995. La razón por la que se ha incluido este apartado es para poner de manifiesto qué características o prestaciones ofrecen los sistemas que no están basados en computadores, aunque algunos de ellos sí hacen uso de DSPs.

Abreviaturas:

- DH : Dependiente del hablante
- IH: Independiente del hablante
- RPA: Reconocimiento de palabras aisladas
- RPC: Reconocimiento de continúo de palabras (Word Spotting)

☑ *Algunos procesadores para el reconocimiento en 1995.*

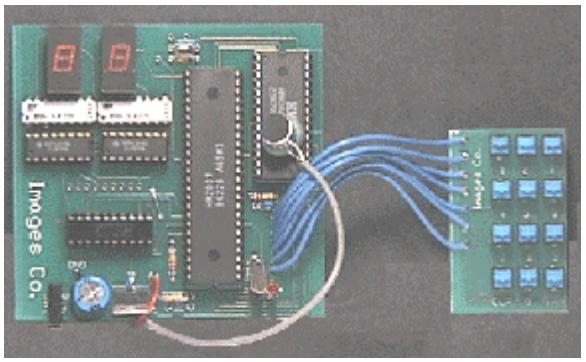
- DVC306 (DSP Communications,Inc.)
- HM2007 (Hualon)
- MSM6679 (OKI semiconductor)
- TC8860F, 64F, 65F (Toshiba)

Nombre : DVC306 (September 95)
Fabricado por : DSP Communications, Inc.
Precio: $10
Número de palabras: 16-128 palabras y un máximo de 8 usuarios en modo DH. (32k-1M SRAMS). Otras características:

- Tiempo de respuesta en IH 500ms, DH 300ms.
- Tamaño del vocabulario activo <30 palabras recomendado.
- IH memoria : 8Kbytes +400 bytes/palabra.
- DH memoria : Aproximadamente 240 bytes/palabra

Especificaciones: RPC, IH y DH. Este DSP dispone de dos modos (IH y DH). Es muy robusto respecto del ruido del entorno(>10dB), el DVC306 permite altas tasas de reconocimiento aún en entornos ruidosos(p.e : en un coche). Además, este producto ofrece síntesis de voz(para realimentación e indicación), Reconocimiento continuo de palabras claves, cancelación adaptativa de audio(para lo que dispone de dos entradas de audio), y una memoria para grabar mensajes cortos.
Características técnicas:

- Alimentación : 3-5V
- Rango de temperaturas: 0-70ºC,
- Encapsulado: 100 pines TQFP. Al que hay que añadirle:
  - Un CODEC- 8 bits que con igual curva que la de la ley PCM o uno lineal de 16 bits. (8Khz Tasa de muestreo/ Rango dinámico de entrada : 40dB).
  - SRAM con configuraciones desde 32K a 1M bytes correspondientes a rangos de vocabulario DH.
  - EPROM/ROM configuraciones de 32K a 1M bytes correspondientes a indicaciones (comprimidas) y a las plantillas independientes del hablante.
  - HOST. Para el que vale cualquier microcontrolador estándar de 4-8 bits. El interfaz consiste en un bus bidireccional de 8 bits y las señales de control.
  - XTAL. a 32 Mhz.



Nombre: HM2007
Fabricado por: HUALON Microelectronics (HM) Corporation (Taiwan).
Precio: $25 for a single quantity, ...
Número de palabras: 40 palabras (0.9 sec cada palabra)o 20 palabras(1.9 sec cada palabra)
Especificaciones: RPA e IH.Este sistema está constituido símplemente por un chip CMOS LSI.
Características técnicas:

- Enpaquetado: 48 pines plastic DIP package
- Alimentación: 5V (simple).
- Intensidades: 6mA mínimo,15mA (máximo).
- Rango de temperaturas:0-70ºC.
- Tiempo de respuesta <300ms.

Trabaja con un micrófono electret. Necesita 8Kbyte Static RAM externa y de un microcontrolador externo.
Un ejemplo de aplicaión de este chip es en juguetes.

Nombre: MSM 6679 (alias VRP 6679)
Fabricado por: OKI SEMICONDUCTOR GROUP
Precio: 20$
Número de palabras: 25 palabras max., sólo con el chip pero se puede añadir una RAM externa (50Bytes/palabra) and have different corpus of 30 words.
Especificaciones: RPA, DH (con dos pasos para el entrenamiento) e IH (es simultáneamente DH y IH), alta tasa de reconocimiento (97% en coches), emplea el algoritmo VCS (Voice Control System from Dallas), basado en "dynamic time warping" y "hidden Markov models".Trabaja con una alimentación simple de 5V, NO necesita CPU externa, rango de temperaturas de –40 á 85ºC.if you want to create a SI with a specific vocabulary, itcosts $200 per new word and $5000 for the globaly "Mask Charges"OKI told me that the MSM6679 is a SD chip - industry first. Ejemplos de aplicaciones : computadores, teléfonos móviles, automóviles, control industrial, equipos de medida, instrumental de diagnóstico, juegos/juguetes...

Nombre: TC 8860F (alias TC 8060F00BS)
Fabricado por: TOSHIBA CORPORATION
Precio: 40.15 FF (French Franc : $1 =3D 5 Fr)
Número de palabras: 10 words
Especificaciones: RPA y HD. Método de alineamiento lineal (algoritmo propio de Toshiba), 4K-bit RAM internos para grabar datos, frecuencia de oscilador : 800kHz, Alimentación : 4.5 - 5.5 V, Intensidad de operación : 4.5mA, Intensidad con el dispositivo a la espera: 0.2microA, enpaquetado: MFP44.
Ejemplo de aplicación : Juguetes..

Lo que se extrae de esta pequeña muestra de componentes es que las prestaciones de los sistemas hardware son mucho menores que las de los sistemas software, ya que aunque los primeros nos ofrecen la posibilidad de un verdadero procesamiento en paralelo, tienen la dificultad de disponer en general de poca memoria en relación a los sistemas software, además los cálculos involucrados en el procesamiento de la señal de voz para

el reconocimiento son muchos, por lo que también necesitaremos implementar muchos operadores para el procesamiento hardware, esto puede verse en sistema como el DVC306, cuyas prestaciones son bastante superiores a las de los demás circuitos, dado que emplea un DSP. Sin embargo, para pequeñas aplicaciones sí que resulta interesante disponer de un sistema hardware para el reconocimiento.

➲ Dificultades en el reconocimiento del habla

Las dificultades que se encuentran en el reconocimiento del habla son muchas y han impedido la plena implantación de estos sistemas, aún hoy día las aplicaciones de estos sistemas son limitadas, y los sistemas con grandes vocabularios no terminan de dar resultados satisfactorios. A continuación listamos los problemas con los que nos encontramos hoy en día:

1. El habla natural es continua.
2. El habla natural tiene variaciones en: la tasa global, tasa local, pronunciaciones entre distintos hablantes, pronunciaciones en el mismo hablante, fonemas con diferentes contextos.
3. En vocabularios largos suele haber confusiones.
4. Siempre hay palabras que quedan fuera del vocabulario, esto es inevitable.
5. La voz grabada es variable por: la acústica de la habitación, características del canal, ruido de fondo.
6. Los tiempos de entrenamiento largos no son prácticos.
7. El usuario tiene unas espectativas iguales o superiores a las de un humano.
8. En la siguiente tabla se resumen las causas principales de las variaciones en el habla.

Ambiente	Ruido correlado con la voz: reverberación, reflexión Ruido incorrelado: Ruido aditivo (estacionario y no estacionario).
Hablante	Características del hablante: Dialecto, género, edad...
	Manera de hablar: Ruido de la respiración y labios, estrés, efecto Lombard, tasa, nivel, *pitch*, cooperactividad.
Equipo de entrada	Micrófono(transductor), Distancia al micrófono, filtrado (antialiasing), Sistema de transmisión (distorsión, ruido, eco), equipo de grabación.

GENERACIÓN DE LA VOZ

En este apartado se van a exponer los fundamentos físicos sobre el proceso por el cual se genera la voz humana, la importancia de este proceso para el reconocimiento del habla, así como del proceso de audición humano, del que sólo se darán algunas pinceladas, es el conocimiento de las características de la señal de voz, dichas características pueden obtenerse de un análisis directo de la señal o estudiando cómo es producida.

➲ El aparato fonador humano

Esta formado por tres elementos: pulmones (generadores de energía), laringe y cuerdas vocales (sistema oscilador) y conducto vocal (cavidad resonante).

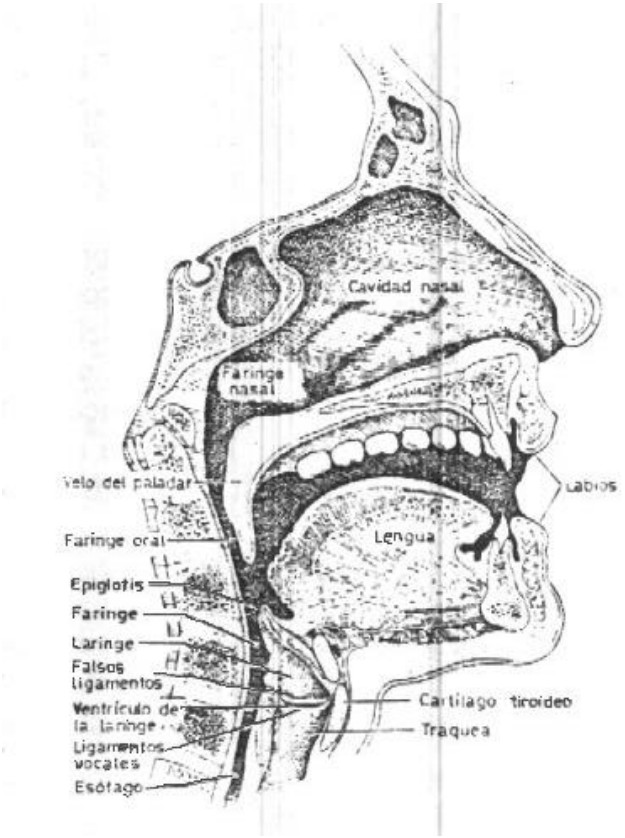


Figura 2.2. Aparato fonador humano.

Modelos para el aparato fonador humano existen muchos, uno de los más simples y extendidos es el siguiente:

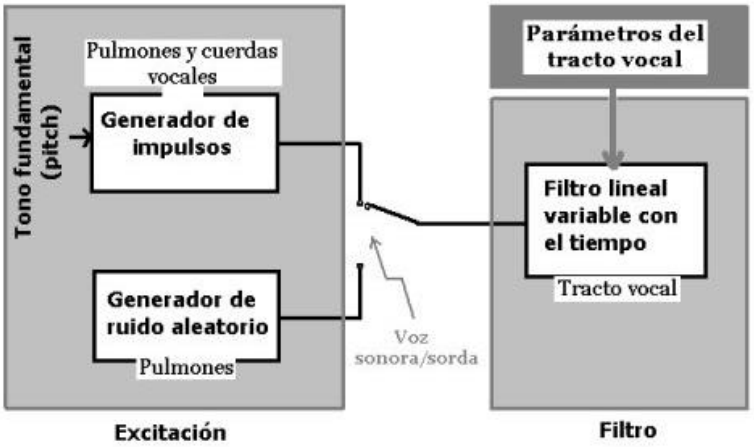


Figura 2.3. Modelo del aparato fonador humano. Existen modelos un poco más complejos que modelan también la epiglotis, para voz sonora, como un filtro y que además introducen un filtro que modela los labios, estos se modelan como un filtro paso bajo de un polo. Sin embargo, para lo que nos interesa explicar este modelo será suficiente.

Los pulmones proporcionan la diferencia de presión necesaria para crear el flujo de aire que activará a la laringe y demás cavidades del tracto vocal.

En la laringe se producen al vibrar las cuerdas vocales variaciones periódicas de presión en el aire proveniente de los pulmones. Estas variaciones se modelan como un tren de pulsos. El periodo de este tren es el pitch o altura tonal(Fo).

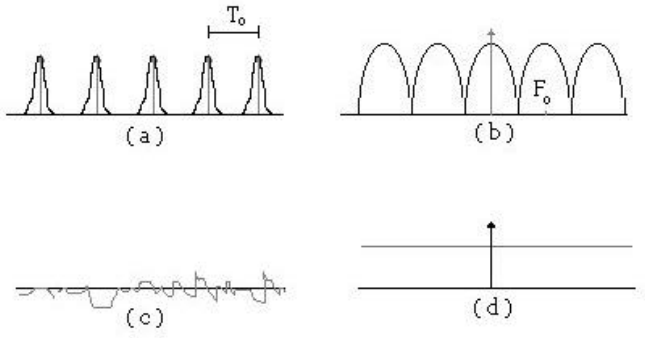


Figura 2.4. (a) tren de pulsos imperfectos: representación matemática de las variaciones de presión del aire en las cuerdas vocales. Este modelo es válido cuando las cuerdas vocales vibran. (b) espectro de la figura anterior. (c) Modelo matemático de la variación en la presión en las cuerdas vocales cuando estas no vibran La excitación se modela como una fuente de ruido cuyo espectro vemos en (d).

Como se ve en la figura 2.4(a) el tren de deltas que modela las variaciones periódicas del aire, esta formado por deltas imperfectas, cuya transformada se muestra en la figura 2.4(b), su transformada tampoco son deltas tienen una cierta anchura espectral, con picos en múltiplo de la frecuencia fundamental (Fo).

La vibración (sonido sonoro) o no (sonido sordo) de las cuerdas vocales , así como la frecuencias de la vibración, son controladas por los músculos correspondientes, para ello establecen la tensión necesaria para cada modo de vibración.

En los sonidos sonoros la energía está básicamente en las bajas frecuencias, al vibrara las cuerdas vocales el pitch se distingue claramente.

Los sonidos sordos tienen menor energía y más distribuida, es un espectro ruidoso en el que no se distingue bien el pitch. La cavidad vocal se comporta como un filtro, a través del que se hace pasar el tren de pulsos(caso que vibren las cuerdas vocales) o la señal de ruido que modela la fuente en los sonidos sordos. La cavidad vocal está constituida por las cavidades de la faringe, boca y nariz, todas ellas deformables mediante los diversos músculos que mueven los elementos articulatorios. Estas cavidades actúan de resonadores variables y a veces conmutables(cavidad nasal) que favorecen o neutralizan componentes del espectro de la onda de presión. Observar que las variaciones en la forma de la cavidad vocal se produce por acciones mecánicas por lo que requiere de un tiempo finito para que varie la respuesta del filtro que es la cavidad vocal de una forma apreciable, esta variación tarda del orden de unos milisegundos.

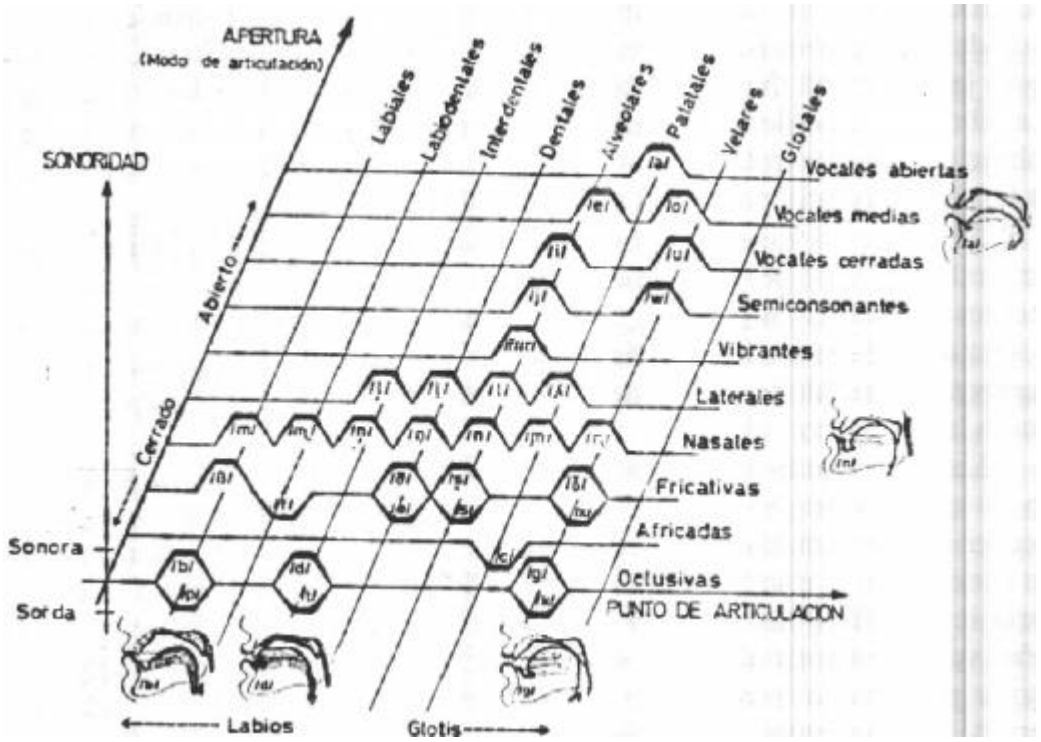


Figura 2.5 (a) Espacio de configuraciones del aparato fonador humano con los sonidos de la fonología española. Los tres ejes de esta representación son: el punto de articulación que se co rresponde con la distancia desde la zona de mayor estrechamiento del conducto vocal a los labios, el modo de articulación que representa el grado de apertura del conducto vocal y/o nasal, y la sonoridad que adopta dos valores: sorda y sonora.

```
Fonos:
        Vocales:Máxima energía.
                Espectro bien definido.
                /a/,/e/,/i/,/o/,/u/.
        Líquidas: se emplea la lengua para su articulación.
                 /l/,/r/,/ʎ/(valla),/^r/(carro).
        Consonantes: se obstruye la salida del aire.
                Oclusivas:
                        Nasales: el aire sale por la nariz.
                                /m/,/n/,/ñ/.
                        Orales: se componen de un silencio
                        seguido de un transitorio muy rápido.
                                Sordas:/p/,/t/,/k/.
                                Sonoras:/b/,/d/,/g/.
                Fricativas: el aire sale por una rendija, se pierde mucha
                energía.
                Sordas:/f/,/s/,/θ/;Sonoras:/j/(valla),/x/(caja).
                Africadas: Sonoras:/E/(brillo);Sordas:/tS/(tacha).
```

Figura 2.5 (b) Esquema de fonos y sus características.

Las resonancias producen señales acústicas en las que la energía está en mayor o menor grado concentrada alrededor de las frecuencias de resonancia correspondientes, a estas concentraciones se las conoce como formantes. En los formantes(en su distribución y estructura) está concentrada la mayor parte de la información psicoacústica que transporta la señal vocal. Las características esenciales de los formantes son sus frecuencias de

resonancia. También la amplitud y el amortiguamiento de los formantes contribuyen a la comprensión de los sonidos.

Para las vocales las frecuencias de los tres primeros formantes, los de más baja frecuencia, constituyen un sistema de referencia absoluto(carta de formantes). Las distintas vocales quedan representadas de forma bastante independiente del locutor como se ve en la figura 2.6.

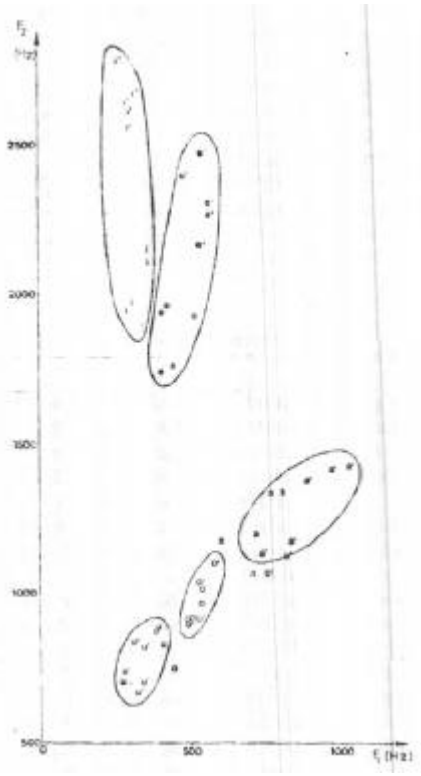


Figura 2.6 :Carta de formantes.

En los sonidos consonánticos las frecuencia de resonancia de los tres primeros formantes sufren rápidas variaciones que enlazan las configuraciones de los formantes de las vocales anterior y posterior. Para las consonantes es la forma de variación la que caracteriza a los sonidos, esta variación no es tan clara como el caso de las vocales.

☑ *Rasgos prosódicos.*

Son las características que distinguen la voz de distintas personas. Estas características dependen mucho del estado emocional y físico del locutor. Las características son pitch, intensidad, y entonación. Por ejemplo: Las mujeres suelen tener un pitch mayor.

☑ *Características de la voz*

La voz es localmente estacionaria, se puede considerar como una onda periódica durante tramas de 30ms.

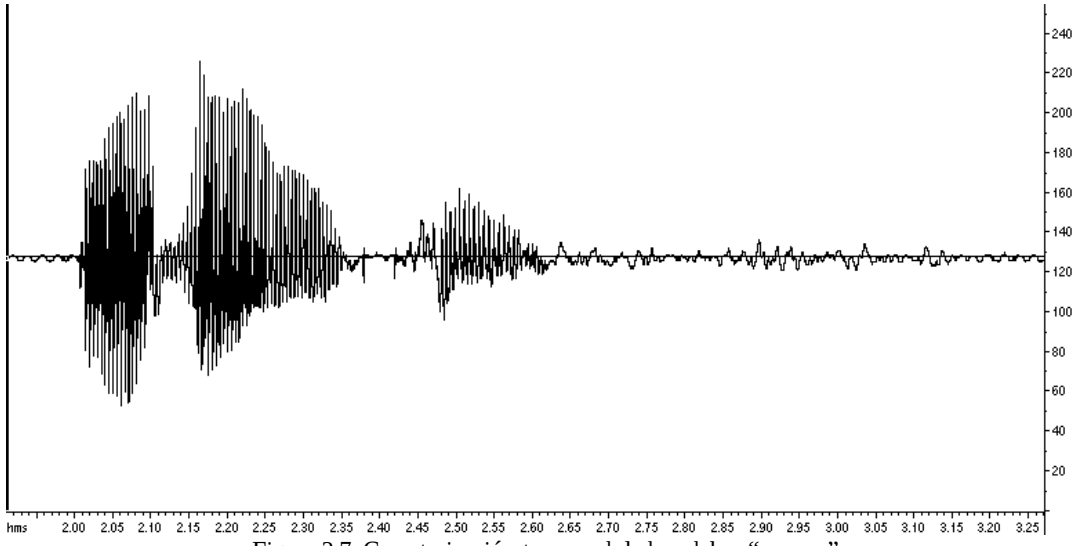


Figura 2.7. Caracterización temporal de la palabra “avanza”.

Las tres concentraciones que se observan en la figura se corresponden con las tres /a/ de la palabra. Vemos que en las consonantes la señal decae. En la siguiente figura se muestra un fragmento de la palabra anterior en el que se puede observar la cuasiperiocidad de la señal.

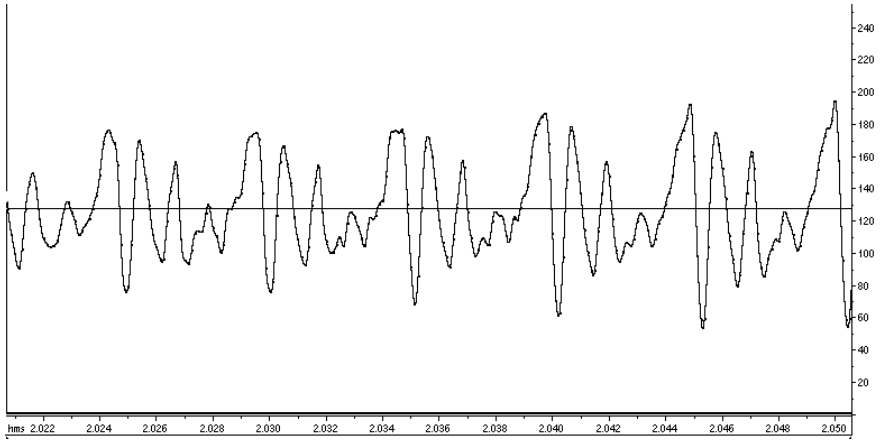


Figura 2.8. Fragmento de 30ms de la /a/ inicial de “avanza”.

Las tramas se clasifican en sonoras, sordas y de transición, siendo estas últimas las más difíciles de analizar, ya que mezclan características de varios fonos: coarticulación. Si ampliásemos las transiciones en la palabra anterior veríamos que se coarticula la /a/ con la /v/ con la /n/ y con la /z/. Además, nos encontramos con el problema de que el rango dinámico varía mucho de una trama a otra y de que la longitud de la trama es muy sensible al fono.

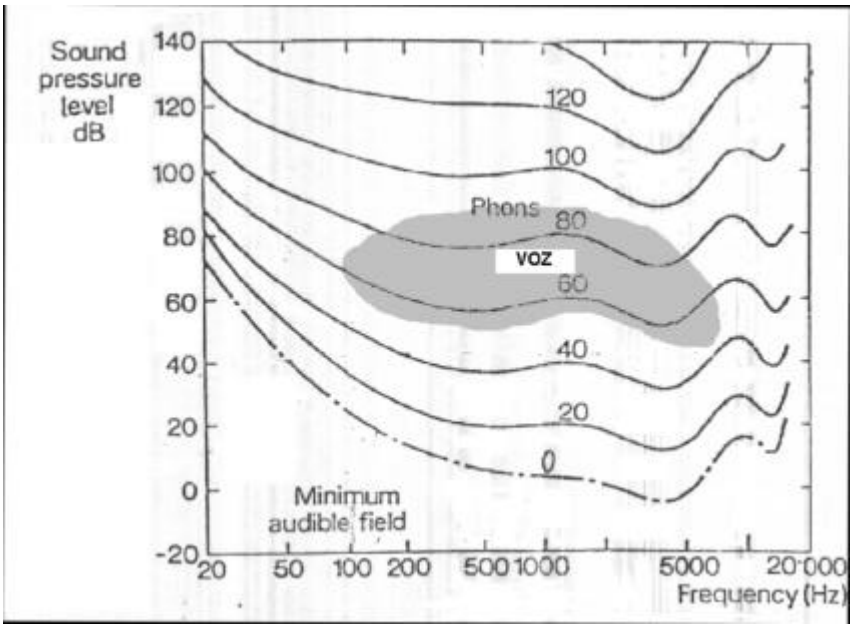


Figura 2.9: Audiometría. En la figura podemos ver entre que frecuencias se distribuye una señal de voz: Entre los 100Hz y los 17KHz.

En las figuras anteriores vemos entre que frecuencias se genera la voz. El espectro se extiende desde 100Hz hasta 17KHz. Sin embargo, como bien es sabido el ancho de banda telefónico se extiende tan sólo hasta los 3.4KHz donde sabemos se encuentra el mayor contenido de energía y por tanto de información.

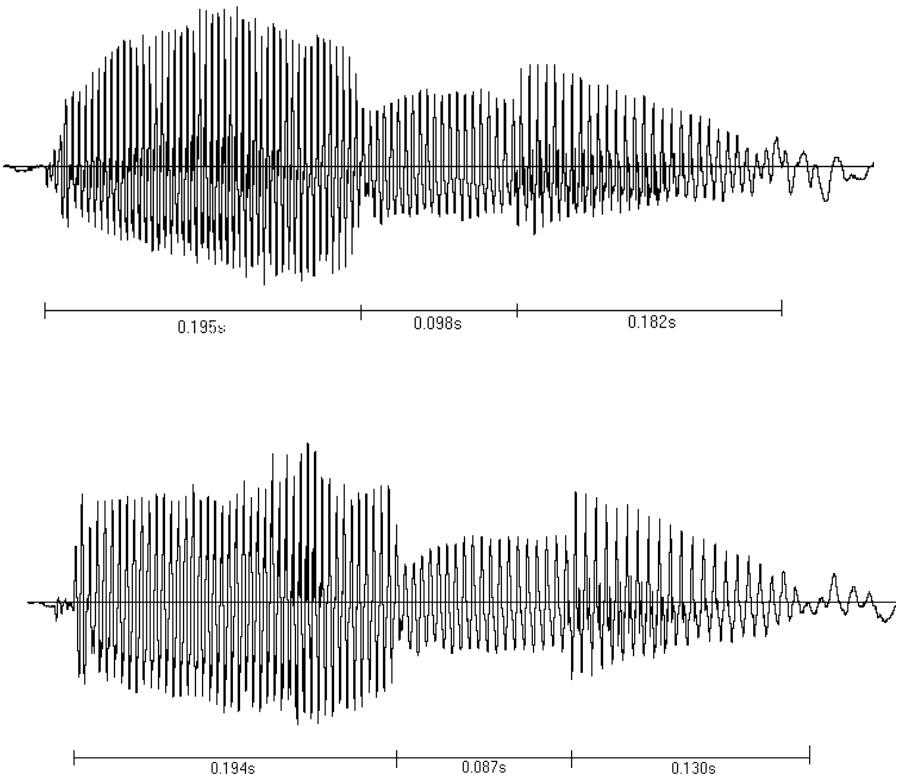


Figura 2.10. Representación temporal de “uno”. Ambas muestras son de la misma palabra pero de pronunciaciones distintas realizadas por un mismo hablante que colabora tratando de pronunciarlo de igual manera en ambos casos. Como puede verse la voz varía para un mismo hablante en las mismas condiciones.

OBJETIVOS, RESTRICCIONES Y ESTRUCTURA DEL PROYECTO

El objetivo del proyecto es el desarrollo de un sistema hardware para reconocimiento del habla. Los sistemas de reconocimiento del habla se han caracterizado tradicionalmente por estar resueltos mediante módulos software, ya sea sobre DSP o procesadores convencionales. El objetivo de este proyecto es la estructuración de estos algoritmos sobre un sistema hardware de manera que se aprovechen las características de paralelismo que ofrece una implementación como ésta. El prototipo del sistema desarrollado se ha implementado en su totalidad sobre una FPGA, tan sólo necesita de un pequeño circuito externo para adquirir y digitalizar la señal de entrada.

Aunque las prestaciones del sistema no se nos impusieron, el marco para el desarrollo del sistema, una FPGA Virtex 300, las limita ya bastante de por sí. Para un sistema de reconocimiento del habla hardware las características que caben esperar son:

- Reconocimiento de palabras aisladas.
- Vocabulario pequeño.
- Dependencia del hablante.

El prototipo del sistema se desarrolló sobre la placa de HADES. Ésta dispone de dos FPGAs:

- ✓ **XC40.** Se emplea para permitir el depurado de sistemas empleando la implementación física del propio circuito, es decir, gestiona el transvase de la información entre la Virtex y el PC a través del puerto paralelo usando el protocolo EPP(la información que se transmite es la que se guarda en la memoria de configuración de la Virtex cuando la condición de disparo que hallamos impuesto se verifique y active el elemento *capture*).

- ✓ **virtex 300.** FPGA sobre la cual se descarga el diseño. La Virtex 300 dispone de 16 bloques de RAM internos de 4096 bits cada uno, además dispone de circuitería que también se puede emplear como RAM (distribuida) aunque sus prestaciones son menores. Es esta característica la que la hace adecuada para este proyecto.

Aún con la RAM que tenemos en la propia FPGA (65Kbits sólo en bloques de RAM), necesitaríamos de RAM externa para podeer disponer de un vocabulario mínimo y razonable, sin embargo como tan sólo se pretende la elaboración de un prototipo que nos permita estudiar la viabilidad y las necesidades de área de un sistema de reconocimiento del habla hardware, no usaremos RAM externa, por lo tanto el vocabulario se verá muy limitado, concretamente sólo cabrán cinco comandos.

Las otras dos características del sistema, reconocimiento de palabras aisladas y dependencia del hablante vienen impuesta por el método de reconocimiento que empleamos, basado en un reconocimiento de formas, donde aquí las *formas* que hay que reconocer son las palabras o comandos, lo cual se realiza mediante la comparación de la entrada con unas plantillas que habremos construido para los comandos, para comparar empleamos el DTW. Para el desarrollo del sistemas se han seguido los pasos normales de un diseño hardware(documentación y delimitación del proyecto, desarrollo o formulación del sistema, implementación software - ajuste de parámetros - y por último implementación hardware - nuevo ajuste de parámetros) con sus respectivas realimentaciones, especialmente en el último paso.

Si bien, es cierto, como ya se ha comentado, que el reconocimiento del habla ha sido objeto de estudio durante décadas, y que por tanto la documentación existente es mucha, ocurre que dicha información, no sólo se centra en logaritmos pensados para una

implementación software, sino que además no es fácil encontrar documentación detallada sobre sistemas totalmente operativos, es decir, la documentación de la que se parte describe una serie de algoritmos y métodos descritos de forma genérica y parametrizada que cubren tan sólo algunos aspectos de los muchos involucrados en todo proceso de reconocimiento del habla. En el apartado anterior se describieron las áreas de trabajo que incluye un sistema de reconocimiento del habla, para la construcción del método de reconocimiento del habla, que se describe con detalle en el capítulo segundo, se procedió como sigue: Para cada una de las áreas implicadas en el reconocimiento se escoge un método, éste se particulariza y se ajusta realizando pruebas con una implementación software del mismo. Una vez probados los bloques software se conectan y se reajustan parámetros para el correcto funcionamiento del conjunto. El criterio para la selección de los valores de los distintos parámetros, no se realizó con criterios de optimalidad, se buscó simplemente un valor razonable, esto es, que diese resultados aceptables en el reconocimiento y que no incrementasen en exceso los recursos necesarios. Una vez obtenido un sistema de reconocimiento operativo se comenzó su implementación hardware. Como primer paso se realiza una primera división del sistema en bloques y se evaluaron los recursos necesarios, especialmente el más crítico, la memoria. El diseño se realizó según la metodología *top-down*, es decir, que se realizaron divisiones sucesivas de los bloques hasta llegar a las unidades mínimas de diseño. Una vez claro el diseño se pasó a escribir el código VHDL. El depurado del código se realiza en paralelo a la escritura del código: Chequeo de sintaxis, síntesis de bloques y simulación lógica, implementación y simulación temporal, también se realizan pruebas sobre la placa. El diseño hardware incluyó el diseño de una PCB para adquisición de datos. Este diseño también se abordó por bloques y se realizaron pruebas en el laboratorio para validar el circuito.

La memoria se ha estructurado en cinco capítulos. En este primero se ha tratado de dar una visión amplia de la de la actualidad en los sistemas de reconocimiento del habla, así como de introducir los términos y conceptos propios de esta disciplina. En el segundo capítulo describiremos los principios físicos de la voz, de forma que nos permita justificar el método de reconocimiento empleado, que también se describe aquí, así como, se presenta la implementación software(matlab) del sistema que luego se ha implementado en hardware. En el tercer capítulo se detalla la adquisición de datos, el circuito empleado y algunos parámetros del sistema de reconocimiento. En el cuarto capítulo se describe con detalle la implementación física del sistema sobre la Virtex 300, se detallan uno a uno los bloques relacionándolos con lo visto en el capítulo 2, también se describe cómo se han obtenido los parámetros restantes(coeficientes de los filtros, umbrales...). Por último, en el quinto capítulo se exponen datos de área, frecuencia, tasas de error, se extraen las conclusiones del diseño y se exponen algunas ampliaciones y mejoras para el sistema.

Capítulo

2

Fundamentos del sistema de reconocimiento del habla

ÍNDICE DEL CAPÍTULO

- Introducción
- Fundamentos del sistema de reconocimiento del habla implementado
- Código Matlab comentado

En este capítulo se describe la solución adoptada en el proyecto desde un punto de vista teórico, también se describe la solución software.

INTRODUCCIÓN

El primer paso para el desarrollo del sistema fue la búsqueda de un método adecuado para el reconocimiento del habla. Para lo cual se tuvo muy en cuenta que el objetivo final era el desarrollo de un sistema hardware y más concretamente, que el prototipo se iba a implementar sobre una FPGA de la familia Virtex a elegir entre la V50 y V300. La principal preocupación era que el método para reconocimiento del habla fuera realizable en el entorno propuesto y en un tiempo razonable por una sólo persona. Por todo lo anterior se buscó la sencillez en el sistema a diseñar. Los sistemas de reconocimiento del habla se caracterizan por ser un sistema de reconocimiento del habla de palabras aisladas, vocabulario pequeño (comandos aislados, de entre un pequeño vocabulario prefijado)y dependientes del hablante (para que la tasa de aciertos sea aceptable el hablante debe entrenar al sistema), así que al buscar la sencillez, son estas las características que cabría esperar del sistema que se desarrollara.

Como no fue posible conseguir una descripción detallada de un sistema de reconocimiento del habla que nos pudiera servir como base para la implementación, se tuvo que diseñar éste en primer lugar. Para lo cual se usaron como base soluciones para determinados aspectos del sistema de las que sí disponíamos con mayor o menor detalle. El sistema que construimos se basa principalmente en las bibliografías [12] y [8] y [10], aunque introdujimos algunas innovaciones y ajustes con el fin de mejorar sus prestaciones: Eficiencia en el reconocimiento y en la implementación.

Aunque el objetivo era un sistema hardware, como paso previo para desarrollar el método y ajustar algunos parámetros realizamos una implementación software, para la dicha implementación software nos planteamos dos entornos C y Matlab. La ventaja del primero es su mayor eficiencia computacional. Por el contrario, tiene el inconveniente de que hay que partir de cero (en realidad existen algunas librerías que pueden ser de ayuda),

en especial, resulta muy engorroso tratar con los archivos wave, mientras que en matlab se cuenta ya con funciones muy especializadas y bien documentadas, en el caso de las wave podemos usar wavread que lee el archivo y nos devuelve una matriz unidimensional o bidimensional según sea la grabación mono o estéreo. Por esto se decidió emplear Matlab, sin embargo, sería recomendable usar ambos, es decir, emplear matlab para aquellas labores en las que disponga de funciones apropiadas y emplear C para cálculos repetitivos que en matlab, y según la potencial del ordenador, puedan tardar bastante.

Un aspecto importante a la hora de realizar el paso del sistema software al sistema hardware es el efecto de la cuantización y asociado a éste la elección del tamaño de la palabra. En el diseño inicial del sistema software no tuvimos en cuenta limitaciones en el tamaño de la palabra, dado que en principio no sabíamos si el sistema funcionaba y este era nuestro objetivo inicial lograr un sistema que funcionase. En cualquier caso, en la medida de lo posible, nos impusimos como limitación inicial usar 8 bits para representar las muestras, también para el sistema software. Esta representación resultó finalmente apropiada. Cuando se realizó la implementación hardware, se estudió en cada caso la resolución más apropiada, así aunque la entrada se represente con 8 bits dentro del banco de filtros se emplea una resolución de 16 bits para los resultados de las secciones y una representación de 32 bits para los cálculos internos en las secciones. Además se realizó el estudio en software para comprobar la desviación entre el sistema sin cuantización y el sistema con cuantización (estas gráficas se encuentran en el capítulo 4) y los resultados fueron realmente buenos.

En este capítulo vamos a ver en primer lugar los fundamentos del sistema de reconocimiento del habla diseñado, y al final veremos su implementación en software, aquí se expone el código matlab, se explica y se muestran algunos resultados obtenidos con ésta implementación, la razón por la que se le presta tanta atención a la implementación software y por la que el código es computacionalmente (para un ordenador) tan ineficiente es porque se ha diseñado un código que sea fácilmente trasladable a VHDL implementable, es decir, que no sólo se ha empleado para ver la viabilidad del método sino también para que sirva de guía a la implementación hardware.

FUNDAMENTOS DEL SISTEMA DE RECONOCIMIENTO DEL HABLA IMPLEMENTADO

➲ Descripción general del sistema

El enfoque del sistema ha sido el de un sistema de reconocimiento de formas, constituido por una serie de bloques o subsistemas. Tenemos un primer bloque que adquiere y adecua la señal, en nuestro caso la voz, para los demás bloques que llevarán a cabo el reconocimiento. Un segundo bloque se encarga de la detección de bordes, que en nuestro sistema consistirá en distinguir palabras del ruido ambiente. Un tercer bloque extraerá de la entrada una serie de parámetros que la caracterice de manera unívoca (esto no es muy cierto, dado que en las vocales tenemos el problema de la no invarianza, la misma característica acústica define varios segmentos de habla y, a la vez, un segmento de habla puede ser definido por diferentes contextos fonéticos) y que serán usadas por el cuarto bloque. El cuarto bloque compara las características obtenidas de la entrada con una base de conocimiento para así decidir qué hemos detectado. En sistemas más complejos que el que hemos elaborado se emplean datos estadísticos y técnicas de inteligencia artificial para la toma de decisión, a parte de alguna medida de *distancia* entre entrada y objetos en la base de conocimiento.

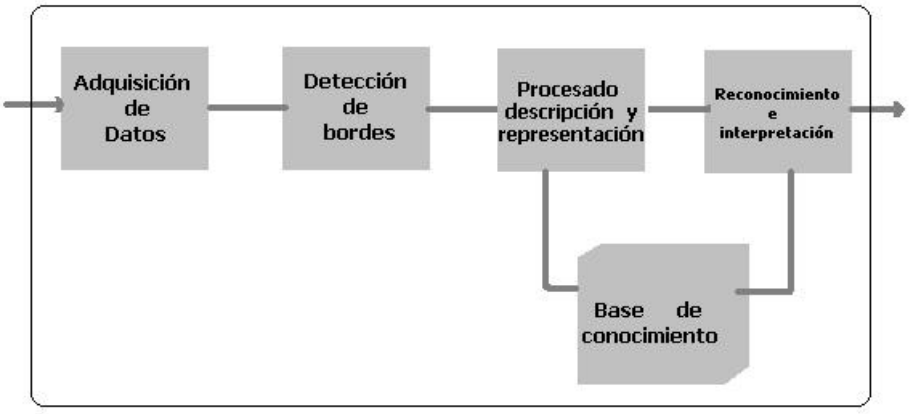


Figura 2.1: Esquema de bloques del sistema.

Antes de hacer una descripción detallada de cada uno de estos bloques haremos una breve descripción de cómo se produce la voz, con el objeto de justificar mejor el diseño del sistema de reconocimiento.

➲ Adquisición de datos, cuantización y muestreo.

Dada la naturaleza de la aplicación el sistema de adquisición se basa en un micrófono. La circuitería necesaria se describe en el siguiente capítulo.

La señal que se obtiene del micrófono es analógica por lo que para someterla a un procesado digital habrá que muestrearla y cuantizarla. Para ello empleamos un convertidor de analógico a digital al que le damos la orden de muestreo cada cierto intervalo de tiempo, que será el periodo de muestreo.

El aparato fonador humano produce señales que llegan hasta los 17Khz, sin embargo, como ya hemos mencionado la cantidad de información que hay ha estas frecuencias es mínima y se obtienen resultados satisfactorios con anchos de banda del orden de 4Khz. Por ello para nuestro sistema se ha seleccionado una frecuencia de muestreo de 8KHz, limitando el ancho de banda de la señal de entrada a unos 3.6KHz para evitar el aliasing, cumplimos con el teorema de Nyquist.

De la experiencia también sabemos que un valor de 8 bits para cuantizar la voz es apropiado y suficiente. Estos son los mismos valores que se utilizan en PCM.

Para una aplicación de reconocimiento del habla algunos autores recomiendan una cuantización de 12 bits y muestreo a 16KHz. Nosotros obtuvimos buenos resultados con los datos indicados en primer lugar.

Para una frecuencia de muestreo de 8KHz tenemos una tasa de llegada de datos al sistema de 1dato(8 bits) cada 125µs. En la siguiente tabla se muestran cuántos ciclos de reloj vamos a tener entre dato y dato:

Frecuencia del Reloj (Mhz)	Ciclos entre muestras
12	1500
50	6250
100	12500

De la tabla anterior se concluye que el tiempo no va a ser un factor crítico en el diseño. Siempre que tengamos un vocabulario reducido ya que si el vocabulario crece luego veremos que el diseño que hemos hecho no es escalable en este sentido. Por otro lado, también veremos luego que el factor más crítico del diseño es la necesidad de RAM.

Para el diseño software hicimos uso de un micrófono electret con un circuito análogo al empleado para la implementación hardware, pero con una alimentación de 1.5v. La señal producida por el micrófono se introdujo en el PC por la entrada mic de una tarjeta de sonido con la opción de amplificación del micrófono desactivada. La señal se grabó con el programa cooledit96 con formato PCM 8bits por muestra y frecuencia de muestreo 8KHz y luego introdujimos la señal en matlab mediante el comando wavread, que nos transforma un archivo .wav en una matriz o vector de muestras.

➲ Descripción y representación

Antes de ver el bloque que realiza la detección de bordes vamos a describir el tercer bloque de la figura 2.1, dado que así se entenderá mejor el bloque 2.

Como ya hemos mencionado la voz es cuasi-estacionaria en tramas de unos 30ms. Es evidente por tanto que es una señal muy redundante. Por tanto no tiene sentido almacenar las muestras en bruto. Desde luego no lo tiene para transmisión y si tenemos en cuenta que una palabra dura en media 832ms y que como luego veremos tendremos que almacenarla entera antes de realizar cualquier procesado, al menos con nuestro método de reconocimiento, haciendo cálculos veremos que para guardar una palabra entera si lo que almacenamos son las muestras en bruto: $832ms * 8000muestras/s * 8bits/muestra = 53{,}248Kbits$ por palabra. Si tenemos en cuenta que la FPGA en la que vamos a implementar el diseño dispone de unos 65Kbits (sólo contamos los blocks RAMs) ya resulta evidente que habrá que valerse de las propiedades de redundancia de la señal de voz para no tener que almacenar tanta información.

Este bloque de descripción y representación hace justamente eso, extrae características de la voz cada cierto tiempo. Estas características son tales que describen la voz en eso periodo de tiempo.

En nuestro caso ese periodo de tiempo se tomo de 32ms por estar en torno al periodo en los que se puede considerar a la voz una onda periódica y por contener 256 periodos de muestreo, hecho que nos facilitará luego los cálculos, dado que dividir por esta cantidad es muy simple.

Por simplicidad hemos tomado un enventanado rectangular, dado que así ahorramos un multiplicador (En el caso de la ventana de Hanning tampoco hace falta un multiplicador, por lo que sería también una alternativa válida), aunque sacrificamos la respuesta espectral, que se ve afectada por el de la ventana, ya que los lóbulos laterales de la ventana rectangular son de bastante importancia.

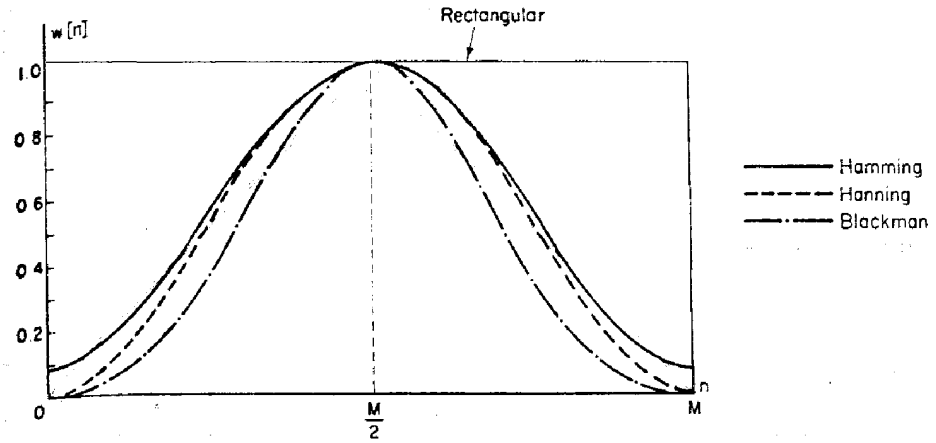


Figura 2.10. Ventanas más usadas.

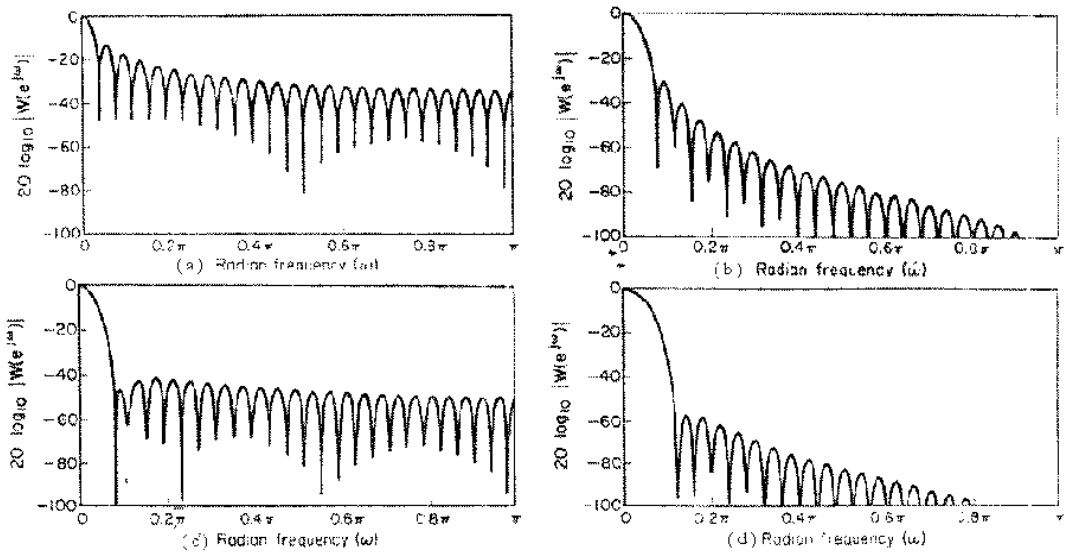


Figura 11. Transformadas de Fourier de las ventanas: (a) Rectangular, (b) Hanning, (c) Hamming, (d) Blackman. En estas figuras podemos ver lo que comentábamos más arriba: Los lóbulos secundarios de la ventana rectangular son de bastante importancia, mientras que al ser el lóbulo principal más estrecho disponemos de mayor resolución espectral.

También por simplicidad no hemos considerado solapes de las ventanas, que es aconsejable para evitar transiciones abruptas en los parámetros que extraemos de la voz y además permite un mejor estudio de las transiciones, sobre todo en caso de que no se use la ventana rectangular, ya que ventanas como la de Hanning ponderan más las muestras del centro de la ventana. Observar que dado que disponemos de suficientes pulsos entre frame y frame añadir esto último no sería un problema, lo que sí habría que acomodar es el número de filtros que posiblemente se pueda reducir, dado que si mantenemos éste fijo pero aumentamos el número de frames (o tramas) por palabra, cosa que ocurre si solapamos las ventanas, el espacio necesario para almacenar los parámetros extraídos de la señal podría incluso superar al espacio para las muestras en bruto. En la figura 2.11 tenemos una representación del análisis hecho.

Figura 2.11. Señal de voz enventanada.

Los parámetros que se pueden extraer de una trama de voz son muy variados y en muchos casos se usan varios a la vez. Los más típicos son la energía (que es la que usaremos), la tasa de cruces por cero (que también lo usaremos), los parámetros LPC (que tiene una fácil interpretación física: Si consideramos que el tracto vocal se puede modelar como un filtro todo polo - aunque la cavidad nasal introduzca ceros en el modelo hay que tener en cuenta que un sistema cero-polo se puede igualar en magnitud a un sistema todo polo, y teniendo en cuenta que la voz no lleva información en la fase veremos que el modelado es correcto – los parámetros LPC son los coeficientes de este filtro todo-polo, modelan la posición la forma del tracto vocal, y si la excitación es un tren de pulsos o un ruido) o el cepstrum (no tiene una interpretación física clara como los LPC. Una interpretación de éste sería la siguiente: Si tenemos en cuenta que la señal de voz es el resultado de pasar la señal de una fuente por un filtro, se puede entender que el espectro de la voz es el producto del de la excitación por el del filtro, si tomamos el logaritmo el producto pasa a ser suma, y si tenemos en cuenta que el filtro que modela el tracto vocal es LP y queda a bajas frecuencias, tendremos una caracterización del tracto vocal).

En cualquier caso, el parámetro que hemos usado para caracterizar la señal de voz ha sido la energía promedio por frame, estimada como amplitud en valor absoluto. Es decir realizamos la suma de las amplitudes en valor absoluto de las 256 muestras que tiene una trama y este es el valor de energía estimado. Como el número de muestras por frame es constante no necesitamos normalizar por éste. En realidad los parámetros usados para caracterizar las tramas de la señal no son simplemente la energía promedio de la señal, sino que son la energía promedio de la señal por bandas. Hacemos pasar la señal de entrada por un banco de filtros paso de banda y a la salida de cada filtro se hace una estimación de la energía promedio en esa banda.

Las bandas que vamos a analizar son 20, este parámetro lo obtuvimos mediante las experiencias realizadas con el sistema software. Disponemos de 20 filtros distribuidos de forma logarítmica desde los 120Hz (frecuencia inferior del filtro de más baja frecuencia) hasta los 3.6KHz (frecuencia superior del filtro a mayor frecuencia). Entre estas frecuencias se encuentra casi la totalidad de la energía de las señales de voz. Hicimos un reparto logarítmico ya que así conseguimos mayor resolución a menor frecuencia donde hay mayor información. Esta solución está inspirada en el oído humano cuya sensibilidad en frecuencia se reparte también de forma logarítmica, con mayor resolución a baja frecuencia. Además los filtros se solapan de manera que no se pierda información, como puede verse en la siguiente figura 2.12, obtenida con matlab. (El código para generarlos se incluye al final).

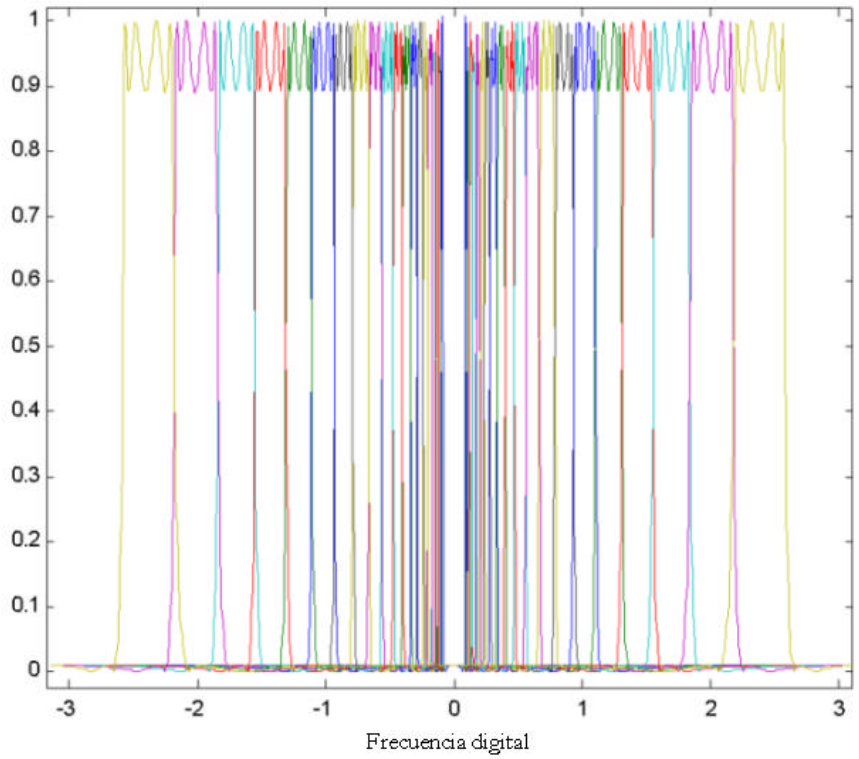


Figura 2.13. Banco de filtros: Imagen generada con Matlab. Los coeficientes de los filtros los hemos generado con Matlab, para el diseño VHDL es necesario cuantificarlos y codificarlos.

En general en voz siempre se usan filtros IIR y en imagen FIR, dado que en voz, la fase no lleva información usar IIR, aunque perturba la fase de la señal no afectará a la percepción de la señal. Por otro lado, es más económico computacionalmente ya que requiere menos coeficientes y por tanto menos memoria y menos operaciones.

Los filtros que hemos empleados son clásicos y en concreto elípticos de orden 8. Para la elección de los filtros hicimos diversas pruebas con filtros clásicos y aún a costa de tener rizado en banda de paso y rechazo la diferencia de orden entre los elípticos y los demás nos hizo inclinarnos por estos.

Por cuestiones de eficiencia en la implementación, escogimos una estructura en cascada de secciones cuadráticas para implementar cada filtro, y cada sección cuadrática se implementa como en forma directa II transpuesta, pero esto se verá en el capítulo dedicado a la implementación hardware, dado que es allí donde se usó ésta.

Para la implementación matlab hicimos uso de la función *filter* que emplea la forma directa II transpuesta, sin preocuparnos de los movimientos que los polos pudieran sufrir al cuantizarse al emplear esta implementación, dado que la resolución que emplea matlab es muy superior a la que usaremos nosotros, aunque esta implementación sea muy sensible al tamaño de la palabra, la resolución que emplea matlab la compensa. Comprobamos experimentalmente que en ningún caso se vuelve inestable. Sin embargo, hay que tener en cuenta que la desviación que sufrirán los polos de los filtros en la implementación software y hardware va a ser distinta, esto es relevante si queremos emplear los modelos generados por software como modelos para la implementación hardware.

Una vez que se ha hecho pasar la señal por el banco de filtros, realizamos una detección de envolvente. Para ver la necesidad de la misma en la figura 2.14 mostramos la salida de los filtros 4, 10 y 20 cuando hacemos pasar por él la señal de la figura 2.7. que podemos ver en la figura 2.14(a) de nuevo.

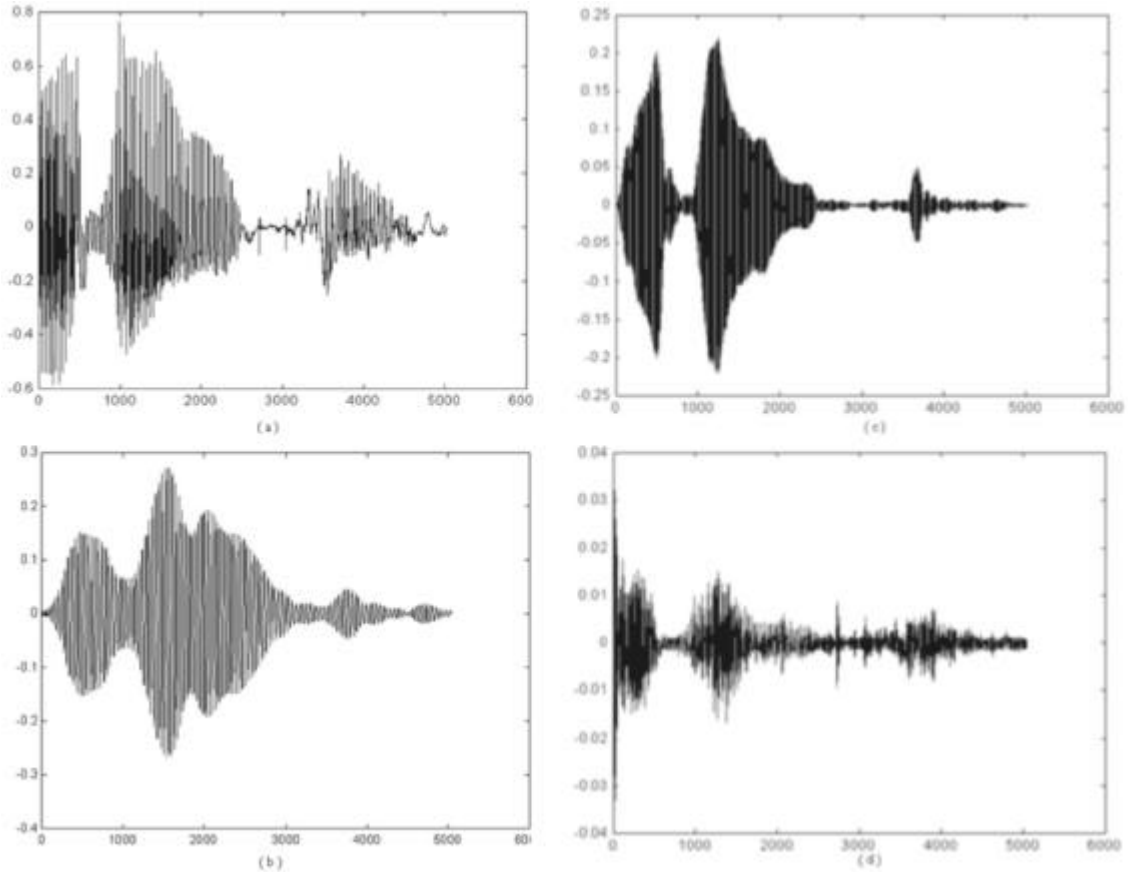


Figura 2.15 (a) Ejemplo de entrada al banco de filtros. (b) Salida del filtro cuarto del banco, empezando a contar como uno al de más baja frecuencia. (c) Salida del décimo filtro del banco. (d) Salida del vigésimo segundo filtro del banco.

Como puede observarse en la figura 2.15 la señal a la salida de los filtros tiene el aspecto de una señal modulada en amplitud. Esto concuerda con el modelo del sistema fonador humano que habíamos presentado antes. La información no se encuentra en la portadora sino en la envolvente. La frecuencia de la portadora distinguirá a un hablante de otro, pero no a una palabra de otra. La única información que se puede decir que aporta es su presencia en sonidos sonoros o su ausencia en sordos.--------

Si bien para caracterizar a la señal vamos a calcular la energía de la envolvente luego veremos que para la detección de bordes calculamos la energía de señal al completo.

El detector de envolvente se basa en el esquema típico: rectificador y filtro paso de baja. El ajuste del polo del filtro paso de baja se realizó empíricamente a 0.05 que redondeamos a 0.0625 para facilitar la implementación hardware. En la figura 2.16 podemos ver la salida del detector de envolvente, hemos usado como entrada la señal mostrada en la figura 2.15 (b). En la figura vemos en rojo la envolvente y celeste la señal. En la figura 2.17 podemos ver la energía por frame de esta misma señal(salida del filtro 4).Esta energía se normaliza por el máximo de la señal total.

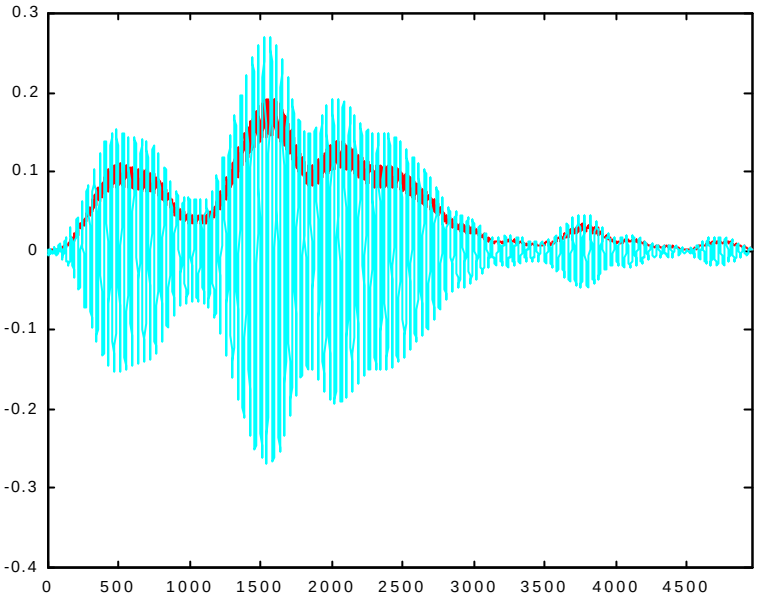


Figura 2.16. Envolvente de la señal.

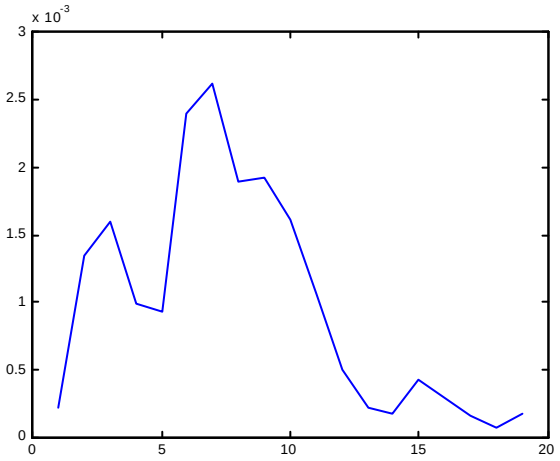


Figura 2.17: Energía promedio de la envolvente.

Esta señal se ha normalizado por el máximo valor de energía de la entrada al completo calculado para un frame. Esta normalización es necesaria para inmunizar al sistema frente a diferentes amplitudes de la señal de entrada.

En resumen, con esta representación de la señal pasamos de 256*8bits por frame a 20*8bits por frame.

➲ Detección de bordes.

El propósito de este bloque es distinguir dónde comienza y acaba una palabra. Para ello debemos fijar unos umbrales de unos parámetros que examinemos de la señal de voz. Los parámetros típicos son la energía promedio y la tasa promedio de cruces por cero.

El uso de la energía como parámetro para distinguir cuando ha comenzado/acabado una palabra es bastante intuitivo, sin embargo la necesidad de la tasa de cruces por cero requiere de una explicación. Si observamos la figura 2.7 veremos que existen tres lóbulos de bastante energía, que se corresponden con las vocales, en este caso las tres *a* de *avanza*. También podemos observar en esta figura que las consonantes llevan

menor energía y que en consonantes como la *z* la energía es aún menor. Muestro sistema de detección de bordes debe tener en cuenta este hecho. Para ello impondremos que la energía de la señal deba mantenerse bajo el umbral durante varios frames para considerar que una palabra ha concluido, esto tiene el inconveniente de que si se tienen dos palabras muy juntas en el tiempo se considerarán como una, por ello a este sistema de reconocimiento de palabras aisladas se le exige una separación mínima entre comandos. Por otro lado para evitar que si el ruido sobrepasa en un momento dado el umbral se tome como comando, se impone un mínimo de tiempo (dos frames) durante los que se debe superar el umbral de energía para poder decir que ha comenzado una palabra. Sin embrago, estas medidas no son suficientes dado que si una palabra comienza o termina por una consonante de poca energía, como por ejemplo la *s*, el sistema determinará que la palabra comienza/termina tras la *s*. Para tener en cuenta esto es para lo que empleamos la tasa de cruces por cero. En consonantes, en especial las fricativas, la tasa de cruces por cero es muy superior a la del ruido. Por ejemplo, en la figura 2.18 tenemos una palabra que comienza por una 'f' la energía media al principio es baja, mientras que la tasa de cruces por cero es alta.

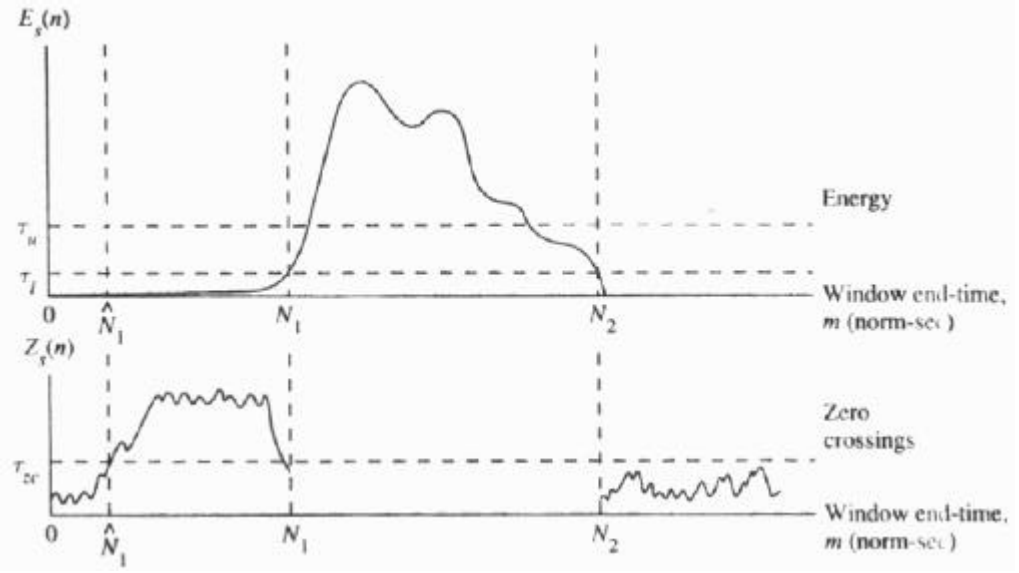


Figura 2.18. Energía promedio por frame y tasa promedio de cruces por cero, para la palabra "four". Discrete-Time Processing of Speech Signals. J.R.Deller, J.G.Proakis, J.H.L.Hansen. Maxwell Macmillan International, 1993. La palabra comienza con una fricativa, cuya energía es baja pero su tasa promedio de cruces por cero alta. En la parte media de la palabra (las vocales) encontramos justo lo contrario baja alta energía pero baja tasa promedio de cruces por cero.

Los umbrales de ambos se determinan a partir de los del ruido ambiente, multiplicándolos por una constante determinada empíricamente. Para esto muestreamos el ruido durante un intervalo suficientemente largo, empíricamente se ajustó a 100ms, para el VHDL se redondeó a 128ms.

Una vez estimados los umbrales, podemos pasar el reconocimiento. Se considera que una palabra comienza cuando supera durante más de un frame el umbral de energía. Como se ha explicado antes, esto indica que la palabra ya ha comenzado, pero puede que lo haya hecho hace más de dos frames, dado que puede haber comenzado con una consonante y por tanto su energía podría no haber sido mayor que el umbral hasta ahora. Para evitar perder frames que aunque de energía baja pertenezcan a la palabra, se hará un estudio de la tasa promedio de cruces por cero de los 10 frames anteriores. Si durante estos 10 frames ocurriera que durante 3 frames seguidos se supera el umbral de cruces por cero, entonces decidiremos que la palabra comenzó en el primero de estos frames. Se exige que sea

durante tres frames seguidos para evitar que se confunda el ruido con una palabra, así un transitorio corto en el ruido será filtrado. Exigir que se mantenga superior al umbral durante tres frames de 32ms cada uno puede ser un poco excesivo en algunos casos, sin embargo nuestras experiencias dieron resultados satisfactorios.

➲ Base de conocimiento

Como el sistema es dependiente del hablante, la base del conocimiento está formada por una parte independiente del hablante y por una parte que recogerá las características de un hablante concreto. Estos datos pertenecientes a un hablante se introducen al sistema durante una fase de entrenamiento.

En nuestro sistema la base de conocimiento está formada por los cinco comandos representados según se indicó en el apartado de representación.

Para los modelos, que es así como nos referiremos a los comandos, hemos establecido una longitud o duración estándar, para facilitar la implementación hardware. Empíricamente comprobamos que una reducción de la duración en los comandos más largos reduce mucho las prestaciones del sistema, mientras que si alargamos un comando más corto apenas se ve afectado, esto tiene su lógica ya que en los comandos largos estamos perdiendo información y en los cortos no. Ya que las duraciones se encontraban entre 10 frames y 30 frames y debido a lo anterior fijamos la longitud estándar en 26 frames.

Con todo esto la necesidad de memoria de la Base del conocimiento es de 5*26*8bits=1040bits.

Para construir la base de conocimiento se sigue un procedimiento idéntico al que se realiza para las palabras de entrada. Como no sería bueno construir la base de conocimiento con una sola pronunciación de una palabra, dado que nunca pronunciamos las palabras de la misma manera. Lo que hacemos es hacer varias pronunciaciones y promediarlas para obtener una pronunciación modelo. Vamos ha describir cómo hemos realizado esto. Lo primero es coger una pronunciación y procesarla igual que lo haremos luego con las palabras de entrada, almacenamos los parámetros extraídos, se toma otra pronunciación y se vuelven a extraer los parámetros. Tomamos ambas pronunciaciones representadas por los parámetros y las alineamos mediante el algoritmo dinamic time warping (ver apartado siguiente). A continuación promediamos los pares de valores según nos indica la alineación. Y pasamos a representar por sus parámetros la siguiente pronunciación, para volver a alinear y promediar y así con tantas pronunciaciones como se quiera. En nuestro caso tomamos tres. Una vez se acabe con las alineaciones y promediados, pasamos a convertir la longitud a 26 frames, para ello simplemente realizamos una proporción lineal.

➲ Reconocimiento e interpretación

Nuestro sistema de reconocimiento se basa en comparar los parámetros extraídos de la entrada con los datos que tenemos en la base de conocimiento. Aquel modelo al que se parezca más la entrada será el que se decida como comando recibido.

Los parámetros que hemos extraído de la señal son sensibles a dos características básicas del habla: La intensidad (que varía por el propio hablante o por la proximidad del hablante al micrófono) y la velocidad a la que pronunciemos la palabra. Estos parámetros también dependen de muchas otras características del habla, como son: entonación, énfasis, estado de ánimo...

Para tratar de paliar los dos primeros (intensidad y velocidad) hemos introducido una serie de medidas. Para la primera la solución escogida es normalizar las energías obtenidas por la máxima energía, promediada en un frame, para la señal al completo. Se hace para la señal al completo porque lo que nos interesa es normalizar por un estimador de la energía media de la señal. (Si tratásemos de hacerlo por la energía de una banda, nos encontraríamos con varios problemas, qué banda escoger, si es la banda de mayor energía la que escogemos, esta irá cambiando según que palabra y qué fonema de la palabra y además el reparto de energías no tiene por qué ser proporcional entre bandas entre dos palabras por lo que las normalizaciones serían distintas.) Por otro lado lo que sí podría haberse hecho es una detección de envolvente para la entrada al completo. Y para la segunda hacemos uso del algoritmo de programación dinámica, *dinamic time warping* que explicamos a continuación.

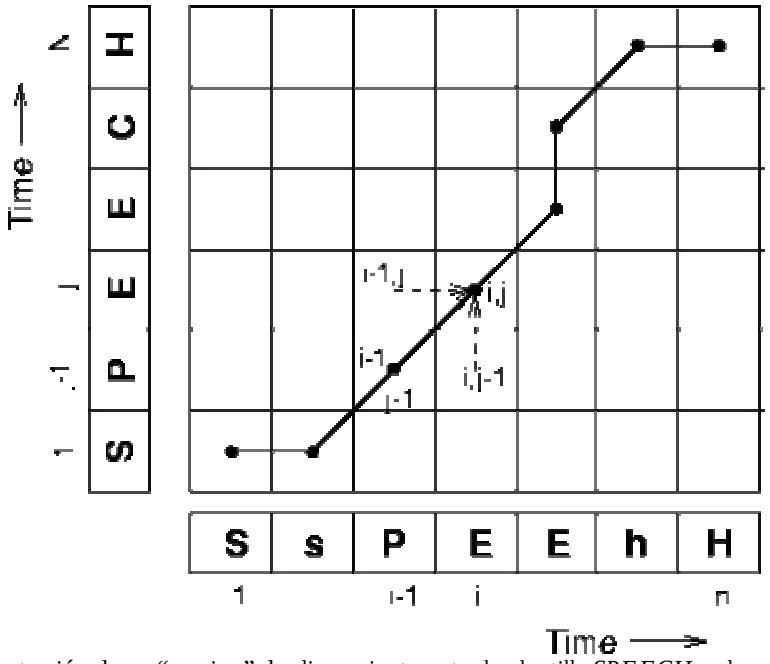


Figura 2.19 Ilustración de un "camino" de alineamiento entre la plantilla *SPEECH* y la entrada ruidosa *SsPEEhH*.

☑ *Dinamic time warping*

El método de reconocimiento del habla que hemos desarrollado se basa en la comparación en el tiempo de la entrada, palabra bajo test, con una plantilla o modelo. Dado que existen variaciones en la velocidad de ejecución de las palabras y estas no produce una prolongación/acortamiento lineal de las mismas, esto es, el cambio en la longitud de la palabra es irregular a lo largo de esta, (por ejemplo, si el tiempo de ejecución aumenta se prolongan más las vocales), lo que es más la misma palabra con la misma duración pueden diferir, debido a que las diferentes partes de la palabra se produzcan a diferentes tasas, necesitaremos realizar un alineamiento no lineal de la entrada y la plantilla o modelo, previo al cómputo de la distancia entre ambos. En la figura 2.19 vemos una ilustración de este problema. La entrada hay que compararla con cada plantilla de la base de conocimientos, siendo el modelo del que diste menos la palabra reconocida. Para resolver este problema hemos usado los principios de la programación dinámica, (que abreviaremos como DP, acrónimo que viene del inglés, *dinamic programming*, DP es un amplio concepto matemático para análisis de procesos en los que hay que tomar decisiones óptimas, no sólo se a aplicado para el reconocimiento sino que su aplicación se extiende a múltiples campos) que cuando se aplica al problema de reconocimiento del habla se suele denominar *dinamic time warping*, (DTW de ahora en adelante) dado que los parámetros

extraídos de la una de las señales se estiran o comprimen en el tiempo(*warp*) para adecuarse a la otra. Nuestra elección de este algoritmo se debió a que se ha empleado con éxito en aplicaciones que requieren algoritmos sencillos y mínimo hardware [10]. Esta técnica se emplea sobre todo para reconocimiento de palabras aisladas, aunque también se ha empleado para reconocimiento de palabras conectadas con ayuda de a otras técnicas. Por otro lado, este método requiere de una plantilla para el reconocimiento de cada producción, razón por la que no resulta escalable y no se emplea para grandes vocabularios.

El algoritmo DTW es una forma de realizar un alineamiento temporal no lineal de palabras, que atiende a criterios de mínima distancia. Como hemos representado las palabras por una serie de vectores de veinte elementos, un vector por frame, cuyos elementos son la energía en cada banda, necesitamos un método para el cálculo de esta distancia. Para medir la distancia entre dos vectores de características de la señal emplearemos la distancia Euclidiana (sin la raíz cuadrada, para evitarnos la complejidad de su implementación), que aunque es computacionalmente más compleja y cara que otras métricas, como la de Itakura, da mayor peso a grandes diferencias en una sola características de los vectores. Empleamos la distancia Euclidiana para medir distancias locales, vector de la entrada frente a vector de la plantilla.

$$d(x,y) = \sqrt{\sum_j \left(x_j - y_j\right)^2} \quad => \quad d(x,y) = \sum_j \left(x_j - y_j\right)^2$$

En la figura 2.19 se ilustra el problema del alineamiento temporal, la representación como una matriz cuyas dos dimensiones son el tiempo es típica, en una dimensión se desglosa la entrada en sus frames y en la otra la plantilla. Aunque en la figura se muestra una descomposición de la palabra por "letras", en realidad esta descomposición es por tramas, en nuestro caso de 32ms, que no tienen por qué englobar un fonema al completo ni sólo parte de un mismo fonema, en muchos casos cogerán las transiciones. La figura trata simplemente de dar una idea de lo que pretendemos. El DTW alineará la entrada con cada plantilla de forma que minimice la distancia entre ambas. En la figura vemos cuál sería el alineamiento que el DTW escogería con las restricciones adecuadas.

Para decidir cuál es el camino de mínima distancia **global** (la distancia global se obtiene como suma de las distancias locales del camino y en algunos casos también hay que sumarle distancias o ponderaciones de las transiciones, aunque este no es nuestro caso) entre una entrada y una plantilla podríamos evaluar todos los posibles caminos entre ambas, pero esto es muy ineficiente ya que el número de caminos crece de forma exponencial con la longitud de la entrada. Sin embargo, podemos imponer una serie de restricciones que evitarán tener que calcular todos los caminos y que son bastante razonables. Las restricciones que hemos impuesto son bastante sencillas:

- El camino no puede volver atrás en el tiempo (camino monótono).
- Todos los frames de la entrada hay que usarlos en el camino.
- Las distancias locales se combinan para dar la distancia global.

De estas condiciones se deduce que el frame inicial de la palabra se asocia siempre al frame inicial de la plantilla y análogamente con el final. Por otro lado, si cada frame en la plantilla y en la entrada debe ser usada en una camino y no podemos volver atrás en le tiempo, esto significa que si tenemos un punto (i,j) en la matriz tiempo-tiempo(donde i indexa la entrada y j indexa la plantilla), entonces un punto previo debe haber sido (i-1,j-1), (i-1,j) o (i,j-1). El DTW en el punto (i,j) continuación el camino de menor distancia desde (i-1,j-1), (i-1,j) o (i,j-1), es decir, el predecesor de (i,j) será el que lleve menor distancia

acumulada de los tres. Esto se fundamenta en el principio de optimalidad de Bellman [10] que reproducimos a continuación:

Sea un camino que comienza y acaba en dos nodos arbitrarios (s,t) y (u,v), respectivamente. Definimos la notación:

$$(s,t)\underset{*}{\rightarrow}(u,v)$$

como el mejor camino (en términos de mínimo coste)que lleva de (s,t) a (u,v).

Denotamos también como:

$$(s,t)\overset{(w,x)}{\underset{*}{\rightarrow}}(u,v)$$

al mejor camino que va de (s,t) a (u,v) y que pasa por (w, x).

En estos términos el principio de optimalidad de Bellman se enuncia como sigue:

$$(s,t)\overset{(w,x)}{\underset{*}{\rightarrow}}(u,v)=(s,t)\underset{*}{\rightarrow}(w,x)\oplus(w,x)\underset{*}{\rightarrow}(u,v)$$

para cualesquiera s, t, u, v, w, x, que cumplan $0\le s,w,u\le I$ *y* $0\le t,x,v\le J$*, donde I y J son los puntos extremos de las palabras y* $\oplus$ *denota concatenación de los segmentos del camino.*

Las consecuencias de este principio son evidentes, para encontrar el mejor camino de (0,0) a (u, v) que pase por un predecesor (s, t) no hay que reexaminar todos los segmentos del camino, sino que basta extender el camino óptimo ya encontrado hasta (u, v) con el segmento óptimo de (s, t) hasta (u,v).

Si D(i,j) es la distancia global hasta (i,j) y d(i,j) denota a la distancia local en (i,j) podemos expresar esta distancia global como:

$$D(i,j)=\min[D(i-1,j-1),\ D(i,j-1),\ D(i-1,j)]+d(i,j)$$

Donde la condición inicial es D(0,0)=d(0,0).

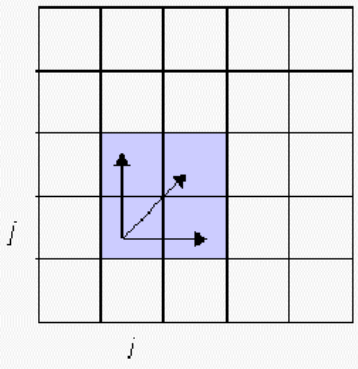


Figura 2.20. Las tres posibles direcciones en las que el mejor camino se puede mover desde la celda (i, j).

Para un reconocimiento básico DP requiere muy poca memoria, sólo necesita, aparte de las plantillas, almacenar dos columnas de la matriz de tiempo.

El algoritmo para encontrar la distancia de mínimo coste es:

1. Calcular la columna 0, comenzando en la celda inferior. El coste global para esta celda es su cote local. Para las demás celdas de la columna el coste global es el coste local más el coste global de la celda situada justo debajo.

2. Calculamos el coste global de la primera celda de la siguiente columna como el local más el coste global de la celda anterior a esta en la fila.

3. Calculamos el coste global para el resto de las celdas de la columna actual como coste local más el mínimo coste global de sus tres predecesores posibles.

4. La columna actual se asigna a la columna predecesora y volvemos al paso 2 hasta que se acaben las columnas.

5. El coste global es el valor almacenado en la celda superior de la última columna cuando termina el algoritmo.

Este proceso se repite con cada plantilla.

Un aspecto que no hemos mencionado hasta ahora pero que es fundamental para el correcto funcionamiento del algoritmo es la normalización de la medida de la distancia total. Observamos que el método descrito penalizará las plantillas que sean más largas, para evitar esta penalización se puede normalizar la distancia global obtenida o, como es nuestro caso, se estandariza la longitud de las plantillas, para lo que realizamos una elongación/contracción lineal de la palabra.

Un aspecto que no hemos contemplado en nuestras restricciones, pero que reduce bastante las operaciones a realizar es el uso de restricciones globales(lo hemos obviado porque para nuestra implementación hardware no iba a reducir la lógica necesaria sino a aumentarla un poco y además complicaba el diseño). Estas restricciones se basan en restringir la zona de la matriz en la que el camino puede estar, para todos los puntos excluidos no habría que calcular distancias, ni se contemplan en los predecesores. Vemos en la figura 2.21 un ejemplo de estas restricciones. Fuera de esta región no se considera ningún camino aunque sea óptimo en coste, esta restricción se le puede imponer a la voz ya que, por ejemplo, no tendría mucho sentido que se alineasen todos los frames salvo el último con el primero de la plantilla y luego el frame último se alinease con el resto de frames.

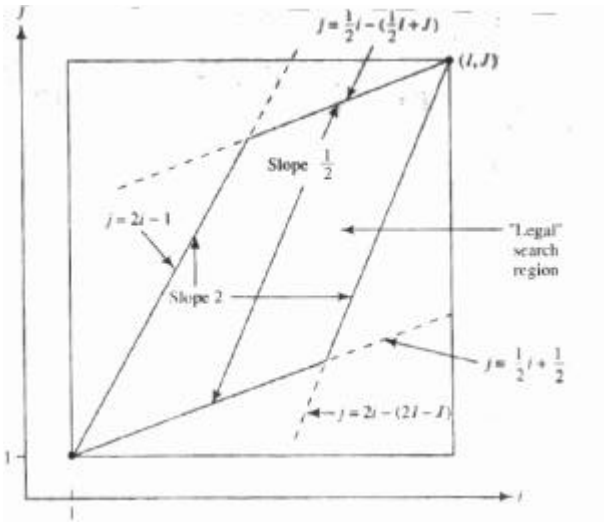


Figura 2.21 La restricción global de Itakura para la búsqueda del camino para una comprensión y expansión máxima de 2.

Estas restricciones reduce el cómputo a la vez que establecen condiciones físicas razonables en el proceso de comparación de entrada y plantilla.

CÓDIGO MATLAB COMENTADO

Por último vamos a exponer y explicar el código en matlab usado para esta primera fase del proyecto, así como los resultados que obtuvimos con él.

Como el objeto de este código es, a parte de ajuste de parámetros, servir de guía para la implementación VHDL, se ha empleado un estilo en el lenguaje próximo a éste en los casos en los que no se complicaba en exceso el código.

Comenzaremos con una serie de funciones para cálculos auxiliares:

➲ Cálculo de la energía

La energía se estima como sumas del valor absoluto de las muestras durante un frame. Esto es lo que implementa la siguiente función *Anf*. Recibe un vector de muestras y devuelve un escalar que representa la energía promedio.

```
function ai=Anf(f);
ai=0;
for i=1:length(f);
  ai=ai+abs(f(i));
end;
ai=ai/length(f);
```

➲ Redefinición de la función signo

Redefinimos signo como cero cuando el escalar de entrada *a* es positivo y uno cuando es negativo.

```
function sig=sgn(a);
if(a>=0);
  sig=0;
else
  sig=1;
end;
```

➲ Cálculo de promedio de cruces por cero

Se estima como el número medio de cambios de signo en un frame. *Zf* Recibe un vector de muestras y devuelve un escalar que representa la tasa media de cruces por cero.

```
function z=Zf(f);
d=length(f);
z=0;
for i=2:d;
  z=z+(abs(sgn(f(i))-sgn(f(i-1))));
end;
z=(1/(2*d))*z;
```

➲ Cálculo de los filtros

Este archivo es básico para el diseño, ya que no sólo servirá para el cálculo de los filtros para la simulación, sino que también emplearemos esta función para obtener los coeficientes, con las transformaciones necesarias, para la implementación hardware de los filtros. La función *bancfil* recibe la frecuencia a la que queremos iniciar el banco de filtros y en la que queremos terminarlo y cuántos filtros queremos que tenga el banco de filtros. Nos devuelve los coeficientes de la función de transferencia de numerados, B, y denominador, A, en forma de matrices donde por filas tenemos los coeficientes de cada filtro del banco. También nos devuelve los polos, p, ceros, z, ganancia, k y función de transferencia al completo, h, organizadas de igual manera que los coeficientes. En la función lo que hacemos es calcular los parámetros de cada filtro (para el cálculo de las frecuencias límites emplea un reparto logarítmico del espacio entre frecuencia inicial y final del banco) y llamar a la función *filtrell* que es la que realiza los filtros.

```
function [B,A,z,p,k,h]=bancfil(fini,ffin,numf);
%function [B,A,z,p,k,h]=bancfil(fini,ffin,numf)
%Entradas:
%   fini=frecuencia del extremo inferior del primer filtro en Hz
%   ffin=frecuencia del extremo superior del último filtro en Hz
%   numf=numero del filtros.
%Salidas:
%    B:coeficientes del numerador donde cada fila se corresponde
%      con un filtro.
%    A:Idem para rl denominador.
%    z,p,k:son los ceros, los polos y la ganancia también dispuestos
%          en filas.
%    H:es la función de transferencia, tb. por fila un filtro.
Rp=1;
%Atenuación en dB máxima en la banda de paso.
```

```
Rs=40;
%Atenuación dB mínima en las bandas de rechazo.
fm=8000;
%fracuencia de muestreo.
%%%%La separación de los filtros va a ser logaritmica
paso=(log10(ffin)-log10(fini))/numf;
for y=1:numf;
  fclog=(y-1)*paso+log10(fini);
  fc(y)=10^fclog;
  fcnlog=y*paso+log10(fini);%%%%%Calculo la siguiente fc
  fs2=10^fcnlog;%%%%%%%La siguiente fc será fs2
  if(y==1);
    %%%then
    fs1=fini-(fs2-fini);
  else
    fs1=fc(y-1);%%%%%%%%fs1 es la anterior fc
  end;
  fs(y,1)=fs1;
  fs(y,2)=fs2;
  %frecuencias límites de la banda de rechazo.
  fp1=((fc(y)-fs1)/2)+fs1;
  fp2=fp1-fs1+fc(y);
  fp(y,1)=fp1;
  fp(y,2)=fp2;
  %frecuencias limítrofes de la banda de paso.
  [Bi,Ai,zi,pi,ki,hi]=filtrell([fp(y,1) fp(y,2)],[fs(y,1) fs(y,2)],Rp,Rs,fm);
          %Filtrell: es otra función creada que calcula filtro a filtro.
  for j=1:length(Bi);
    B(y,j)=Bi(j);
  end;
  for j=1:length(Ai);
    A(y,j)=Ai(j);
  end;
  for j=1:length(zi);
    z(y,j)=zi(j);
  end;
  for j=1:length(pi);
    p(y,j)=pi(j);
  end;
  for j=1:length(ki);
    k(y,j)=ki(j);
  end;
  for j=1:length(hi);
    h(y,j)=hi(j);
  end;
end;
%Si quiere verse una gráfica de todos los filtros superpuestos
%descomentar estas líneas:
%w=-pi:2*pi/1024:pi-2*pi/1024;
%plot(w,abs(h));
```

Esta es la función que calcula los filtros uno a uno, para lo cual emplea las funciones de la librería para el tratamiento digital de señales de matlab, ellipord y ellip, que sirven para obtener un filtro elíptico con las características que se le indican. La implementación software podría haberse hecho en C, pero calculando los filtros con matlab, de esta forma se aprovecharían las características de ambos lenguajes. Los parámetros que hay que pasar a esta función los podemos ver en el archivo que se incluye tras estas líneas.

```
function[B,A,z,p,k,h]=filtrell(fp,fs,Rp,Rs,fm);
%function[B,A,z,p,k,h]=filtrell(fp,fs,Rp,Rs,fm)
%Esta función calcula los filtros que van a ser elípticos.
%Entradas:
%%%%%fm es la frecuencia de muestreo en Hz
```

```
%%%%fp vector 2-d que contiene la frecuencias límites de la banda de paso.
%%%%%primero la menor, y en Hz
%%%%fs análogo para la de rechazo.
%%%%Rp atenuación máxima de la banda de paso en dB
%%%%Rs atenuación mímina de las bandas de rechazo en dB
%Salidas:
%%%%B:coeficientes del numerador.
%%%%A:coeficientes del denominador.
%%%%z,p,k:ceros, polos y ganancia del filtro.
%%%%h:función de transferencia del filtro.
fn=fm/2;
[n,wn]=ellipord(fp/fn,fs/fn,Rp,Rs);
[B,A]=ellip(n,Rp,Rs,wn);
[z,p,k]=ellip(n,Rp,Rs,wn);
h=freqz(B,A,w);
%w=-pi:2*pi/1024:pi-2*pi/1024;
%para ver el filtro, descomentar:
%plot(w,abs(h));
%pause;
```

En el tema dedicado a la implementación hardware veremos como hemos obtenido los coeficientes que empleamos para los filtros implementados sobre la FPGA.

Hasta ahora hemos visto las funciones auxiliares para el tratamiento de la señal, a continuación veremos una serie de funciones que valiéndose de las ya vistas realizan transformaciones sobre la señal siguiendo el proceso que luego implementaremos en la FPGA.

➲ Detección de bordes

Este archivo evalúa los parámetros del ruido a partir de una entrada (ruido) que es una tabla obtenida a partir de una señal de audio. Una vez calculados estos parámetros determina dónde comienza y termina la entrada *s* que no es más que la señal de audio inmersa en ruido. El parámetro N indica el número de muestras por frame. Esta función no devuelve la señal sin ruido, sino que devuelve los índices en los que ha detectado que comienza y acaba.

```
function [N1,N2,max]=ppioyfin(s,ruido,N);
%%%%function [N1,N2,max]=ppioyfin(s,ruido,N)
%%%%ruido es una señal que recoge el ruido ambiente
%%%%A partir de ella vamos a caracterizar el mismo.
%Entradas:
%s:
%señal de la que queremos conocer donde empieza y acaba,
%"bordes" que la separan del ruido ambiente.
%será una matriz obtenida a partir de un archivo de audio por
%ejemplo con wavread.
%ruido:
%Es también una matriz obtenida a partir de una muestra de audio
%del ruido ambiente. De él obtendremos los umbrales para distinguir
%donde comienza la señal de audio.Las caracteristicas que extraeremos
%del ruido son media de energía Y media de cruces por cero.
%N:
%Número de muestras por frame.Nos indica cuantas muestras de la
%entrada agrupar.
%Salidas:
%N1:
%frame en el que consideramos comienza la palabra, que debe haber
%en s y que queremos separar del ruido de ambiente.
%N2:
%Frame en el que acaba la palabra.
%max:
%Por aqui se devuelve el máximo de la señal de entrada, que nos
%servirá luego para inmunizar frente a distintas amplitudes en
%la señal de entrada.
%%para tener una buena carcterización del ruido sería interesante tener
```

```
%%al menos 100ms de muestras.
%%La frecuencia de muestreo estamos suponiendo siempre que es de 8KHz
%%El módulo VHDL, recibe las muestras cada 125us y va calculando en tiempo
%%real los parámetros de interés.
dr=length(ruido);
Meds=0;
MedEr=0;
for j=1:floor(dr/N);
  A(j)=0;
  for t=1:N;
  %%%"media" del ruido
           A(j)=A(j)+(abs(ruido(t+(j-1)*N)));
  %%%Esta es la media de la "estimación"
  %% de la energía => difiere de mean(v)
           end;
  %%Conviene dividir una vez hecha la suma y no sumando a sumando
  A(j)=A(j)/N;
           MedEr=MedEr+A(j);
end;
MedEr=MedEr/floor(dr/N);%divido una vez hecha la suma
%%%%Con el procesado en linea no es posible: MedEr=MedEr/Medr;
%%%%%%%%%si tiene un max muy pequeño esto aumenta mucho
%%%Además no tiene sentido normalizarlo,realmente si lo tiene pq normalizo
%%todo lo q entra eso incluye el ruido anexo a la señal.
%%%Lo que ocurre es que este valor es muy pequeño, luego como cota no vale
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%v=0;
%%%%%%%for j=1:dr;
  %% Varianza muestral
%%%%%%%%%   v=v+((ruido(j)-A)^2)/dr;
  %%%%esta varianza es menor que:(std(v))^2
%%%%%%%end;
%% dado que la estimación de la varianza es en magnitud mucho menor
%% que la media podemos símplemente no calcular la varianza y estimar
%% el ruido por su energía media
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
%%El cálculo de la energía de la señal (Anf.m) se hace frame a frame de
%%N muestras y se normaliza por N. La energía tb. se normaliza por el máximo
%%de la señal en ese frame.
%%%Cruces medio por ceros:
MedZr=0;
for j=1:floor(dr/N);
  z(j)=0;
   %%%"media cruces" del ruido
  z(j)=Zf(ruido(((j-1)*N+1):(j*N)));
   %Zf ya lo devuelve normalizado
           MedZr=MedZr+z(j);
end;
MedZr=MedZr/floor(dr/N);%%%Para no perder resolución
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for j=1:floor(length(s)/N);
  A(j)=0;
  for t=1:N;
  %%%"media" del ruido
           A(j)=A(j)+(abs(s(t+(j-1)*N)));
  %%%Esta es la media de la "estimación"
  %% de la energía => difiere de mean(v)
  end;
MedEs(j)=A(j)/N;
end;
%Divido aquí para no perder resolución.
%Dentro del 2º for anterior:
%Para VHDL:%if(j==1);Meds=A(j);else;
         %if(Meds<A(j)); Meds=A(j);end;end;%end;
[max,o]=max(A);

%%La señal hay que normalizarla para atender que dos señales iguales
%% sean de /= amplitud, pero esta normalización tiene sentido a la
%%salida de los filtros no aquí, además de no ser posible con un procesado
%% en línea.
for j=1:floor(length(s)/N);
  z(j)=0;
   %%%"media cruces" del ruido
                       z(j)=Zf(s(((j-1)*N+1):(j*N)));
```

```
   %%%Esta es la media de la "estimación"
   %% de la energía => difiere de mean(v)
MedZs(j)=z(j);
end;
t=0;
r=0;
for j=2:floor(length(s)/N);
  if(j==2)
    N1=1;
  else
    if(r==0)
          if((MedEs(j-1)>1.8*MedEr)&(MedEs(j)>1.8*MedEr)&(MedEs(j)>2*MedEr))
          N1=j-1;
          r=1;
          for k=j-min([8,3+j]):j-1;
            if((MedZs(k-2)>=2*MedZr)&(MedZs(k-1)>=2*MedZr))
                N1=k-2;
            end;
          end;
       end;
    else
      if(t==0)

if((MedEs(j)<=1.8*MedEr)&(MedEs(j+1)<=1.8*MedEr)&(MedEs(j+2)<=1.8*MedEr)&(MedEs(j+3)<=1.8*MedEr)&(Med
Es(j+4)<=1.8*MedEr)&(MedEs(j+5)<=1.8*MedEr)&(MedEs(j+6)<=1.8*MedEr)&(MedEs(j+7)<=1.8*MedEr)&(MedEs(j
+8)<=1.8*MedEr)&(MedEs(j+9)<=1.8*MedEr))
                                        N2=j;
                                        t=1;
                                        for k=j:min([length(MedZs),j+9])
                                if((MedZs(k)>=2*MedZr)&(MedZs(k+1)>=2*MedZr)&(MedZs(k+2)>=2*MedZr))
                                        N2=k;
                    end;
                    end;
                end;
          end;
    end;
  end;
end;
N1=N1*N;
N2=N2*N;
```

➲ Cálculo de la salida de los filtros

Esta función recibe los coeficientes de un filtro y la señal que queremos filtrar, podría dársele con el ruido pero nosotros lo haremos sin él, como luego veremos. La función se basa en *filter* función de las librerías de matlab que realiza el filtrado usando una forma directa dos transpuesta, que como sabemos es muy sensible a la cuantización de los coeficientes y no será la que usemos para la implementación hardware, como ya habíamos mencionado.

```
function sal=salfil(Bp,Ap,s);
%function sal=salfil(Bp,Ap,s)
%Este función espera recibir los filtros dados por sus
%coeficientes, y llamar a filter sobre la señal que se le pasa
%como entrada.
%La línea comentada del plot sirve para visualizar las salidas
%de los filtros, si lo no queremos ver las salidas conviene
%comentarla para acelerar el proceso de cálculo.
%Entradas:
%Bp:
%Coeficientes del numerador. Se trata de una matriz bidimensional
%en la quecada fila contiene los elementos de un filtro, estos
%elemantos se deben haber obtenido con las funciones que suministra
%matlab o tener el mismo formato.
%Ap:
%igual pero para los coeficientes del denominador.
%s:
%matriz obtenida a partir de una señal de audio y recortada ya del
%ruido ambiente, en realidad este requisito es para nuestra aplicación
```

```
%de reconocimiento del habla.
%sal:
%matriz que contiene las salidas de los filtros por filas.
numf=size(Bp);
for j=1:numf(1);
  aux=filter(Bp(j,:),Ap(j,:),s);
  for p=1:length(aux)
    sal(j,p)=aux(p);
  end;
%plot(sal(j,:));
%pause;
end;
%%%%salen valores muy altos habrá que normalizar de alguna manera.
```

➲ Implementación software para el estudio de la cuantización en la implementación hardware

Realizamos también una implementación cascada de los filtros para hacer un estudio de los resultados que íbamos a obtener con la implementación hardware, con los tamaños de palabra que luego veremos en el capítulo 4.

Este estudio se realizó para el banco de filtros, para ello redefinimos la función *Anf*, que calcula la energía promedio de una trama teniendo en cuenta los valores de saturación, *salcasc* que lleva a cabo el filtrado paso banda con la implementación cascada y la misma resolución que luego se usará para la implementación hardware, y *sección* en la que se apoya la función anterior para dicho cálculo.

```
%%calculo de la energia de un frame
function ai=Anf(f,a,b);
ai=0;
for i=1:length(f);
   ai=ai+abs(f(i));
   if(ai>255);ai=255;end;
end;
ai=ai/length(f);
%%%%%%%% normalizo por el número de muestras de la
%%%%%%%%ventana aunque todos van a usar la misma,para evitar
%%%desbordamiento.

function sal=salcasc(sosc,s);
%sal=salcasc(sosc,s);
%sal: salida de cada filtro por filas
%sosc: coeficientes de la cascada, 1ªdimensión ->qué filtro
%2ªdimensión->qué sección, 3ªdimensión-> qué coeficiente, se han
obtenido como se explica en el capítulo 4.
d=size(s);
for i=1:20; %recorro todos los filtros.
   j=1;
   filtro(:,:)=sosc(i,:,:);%coeficientes del filtro
   for a=1:3;
      for b=1:5;
         antes(a,b)=0;
      end;
   end;   %cada vez que comience un filtro pongo esto a cero
   while j<d(1)+1;
      antes(3,1)=antes(2,1);
      antes(2,1)=antes(1,1);
      antes(1,1)=s(j);
      for k=1:4; %recorro las secciones
         valor=seccion(filtro,k,antes);
```

```
         antes(3,k+1)=antes(2,k+1);
         antes(2,k+1)=antes(1,k+1);
         antes(1,k+1)=valor;
      end;
      sal(i,j)=valor;
      j=j+1;
   end;
end;


function valor=seccion(filtro,seccion,antes);
%filtro size:4 secciones 6 coeficientes
%seccion:indica que sección se está operando
%antes:size 3 filas y 5 columnas las tres salidas/entradas a cada
sección

v1=antes(1,seccion)*filtro(seccion,1);
%como lo que va a entrar ya son enteros no hay necesidad de usar round
%lo que si habrá es que saturar el producto.
if(v1>2^31-1); v1=2^31-1;end;
if(v1<-2^31);v1=2^31;end;
v2=antes(2,seccion)*filtro(seccion,2);
if(v2>2^31-1); v2=2^31-1;end;
if(v2<-2^31);v2=2^31;end;
v3=antes(3,seccion)*filtro(seccion,3);
if(v3>2^31-1); v3=2^31-1;end;
if(v3<-2^31);v3=2^31;end;
v4=(-1)*antes(1,seccion+1)*filtro(seccion,5);
if(v4>2^31-1); v4=2^31-1;end;
if(v4<-2^31);v4=2^31;end;
v5=(-1)*antes(2,seccion+1)*filtro(seccion,6);
if(v5>2^31-1);v5=2^31-1;end;
if(v5<-2^31);v5=2^31;end;
valor=v1+v2;
if(valor>2^31-1);valor=2^31-1;end;
if(valor<-2^31);valor=2^31;end;
valor=valor+v3;
if(valor>2^31-1);valor=2^31-1;end;
if(valor<-2^31);valor=2^31;end;
valor=valor+v4;
if(valor>2^31-1);valor=2^31-1;end;
if(valor<-2^31);valor=2^31;end;
valor=valor+v5;
if(valor>2^31-1);valor=2^31-1;end;
if(valor<-2^31);valor=2^31;end;
valor=valor/filtro(seccion,4);
```

➲ Suavizado

Esta función es parte del detector de envolvente, por el que pasamos la señal antes de calcular su energía por bandas. No es más que un filtro paso de baja con un sólo polo, podríamos haber usado filter, como ya se hizo con el banco de filtros, sin embargo, hemos querido implementarlo de forma que nos facilite su traducción al VHDL. Recibe la señal y el polo y devuelve la señal filtrada.

function fsuavi=suavi(in,u);
%function fsuavi=suavi(in,u)
%Entradas:
%in:señal que queremos filtrar paso de baja, viene dada en un vector.

```
%u:parámetro del filtro.
%Salidas:
%fsuavi:salida del filtro es otro vector.
%%%suaviza picos haciendo el filtrado de la señal rectificada
fsuavi(1)=0;
for j=2:length(in);
  fsuavi(j)=(1-u)*fsuavi(j-1)+u*in(j);
end;
%%%%%%%Lo que hago es un detector de envolvente.
%En matlab podría usarse filter pero lo hacemos así porque es como lo
%implementaremos en VHDL.
```

➲ Cálculo de la energía

Esta función calcula la energía por frame de la señal rectificada y suavizada. Recibe la señal (como matriz, cada fila una banda) de una banda rectificada y suavizada, es decir, la envolvente y devuelve una matriz en la que por filas tenemos una posición por cada frame y en esta posición la energía de la banda.

```
 function [fener,maxi,pos]=ener(srs,N);
%function [fener,maxi,pos]=ener(srs,N)
%srs:señal rectificada y suavizada(la envolvente).Matriz bidimensional
%cada fila es una banda de energía.
%N:número de muestras a promediar.
%fener:energía promedio.
%maxi:máximo de energía de entre todas las bandas.
%pos:posición e la que se halló el máximo.
%%%N es el numero de muestras para las que promediare la energia
%%%%srs es la señal rectificada y suavizada.
for j=1:floor(length(srs)/N);
  for m=1:N
    se(m)=srs(m+(j-1)*N);
  end;
  fener(j)=Anf(se);
  if(j==1)
          maxi=fener(1);
    pos=1;
  else
    if(maxi<fener(j))
      maxi=fener(j);
      pos=j;
    end;
  end;
end;
```

➲ Función que realiza la detección de envolvente y cálculo de las energías por banda

Para lo que se basa en las dos funciones descritas previamente.

```
function [fun,posi]=rsuen(y,N,u,max);
%function [fun,posi]=rsuen(y,N,u,max)
%%%%%rectifica,suaviza y calcula la energía promedio por tramas de N
%Entradas:
%y es la señal de entrada, es una matriz bidimensional en la que
%en nuestra aplicación tenemos por filas las salidas de cada filtro.
%N:número de muestras por frame.
%u:Parámetro del filtro para detección de envolvente.
%max:máximo valor de la entrada.
d=size(y);
 maximo=0;
for j=1:d(1);
  yr=abs(y(j,:));
```

```
  ys=suavi(yr,u);
 %descomentar para ver gráficas:
  %le=1:length(ys);
  %plot(le,ys,'r',le,yr,'y');
  %pause;
  [aux,maxim(j),posi(j)]=ener(ys,N);
  for p=1:length(aux)
    funaux(j,p)=aux(p);
    %%%%%no creo que normalizar cada banda sea bueno
    %%%%pq si lo hago estoy eliminando que en una bandad
    %%haya + energia que en otra
  end;
 %  if(maxim(j)>maximo)
 %    maximo=maxim(j);
 % end;
end;
fun=funaux/max;
%%%%%Devuelvo fun que es una matriz con la salidas de cada filtro,
%%%por filas, debidamentes procesadas:rect,envolvente,energia normalizada.
%%%%posi es un vector que contiene por filas el punto en el que se encuentra
%%(En promedio de frames de N) el maximo de cada banda.
```

➲ DTW

Implementa el algoritmo visto más arriba, los parámetros que hay que pasarle se detallan en el archivo que incluimos más abajo. Esta no es la función que se emplea para el cálculo de los modelos, sólo se emplea para calcular la distancia entre la entrada y un modelo, ambos representados por matrices de energía por banda y frame. Observar que como habíamos comentado sólo necesitamos almacenar dos columnas de la matriz.

```
function dist=dtw(vmodf,vinf);
%dist=dtw(vmodf,vinf)
%Esta función implementa el algoritmo dtw. Para ello debe recibir:
%vmodf:vector cuyos elementos son los valores de energía promedio de
%la envolvente del comando que queremos reconocer por bandas y por
%frames.Para la implementación en matlab no es necesario normalizar
%el tamaño de los modelos como sí será necesario en el VHDL, para
%evitarnos el cociente.
%vinf:idem pero para la entrada que además habrá sido normalizada
%en energía.
%Salida:
%dist:distancia entre la entrada y el modelo una vez alineados conforme
%al algoritmo dtw con las restricciones impuestas.
dimen=size(vmodf);
dim=size(vinf);
Dcolp(1:dimen(2))=0;
for j=1:dim(2);  %columnas
  Dcolat(1:dimen(2))=0;
  dcola(1:dimen(2))=0;
  for n=1:dimen(2);  %filas =>por cada punto de la matriz de alineamiento.
    e=0;
    for k=1:dimen(1);
      e=e+(vmodf(k,n)-vinf(k,j))^2;
    end;
    %dcola(n)=sqrt(e);  %%distancias locales de la columna actual.
    %modificación para ver el efecto de no usar raíz
    dcola(n)=e;
    %e
    if (j==1)
      if(n==1)
        Dcolat(1)=dcola(1);
      else
        Dcolat(n)=dcola(n)+Dcolat(n-1);
      end;
```

```
      %%%%%%%%%%hemos iniciado la 1ª  columna.
    else
      if (n==1);
        Dcolat(n)=dcola(n)+Dcolp(n);
      else
        %%%%%%para el resto de los puntos:
        Dcolat(n)=dcola(n)+min([Dcolat(n-1) Dcolp(n) Dcolp(n-1)]);
      end;
    end;
  end;
  Dcolp=Dcolat;
end;
dist=Dcolat(dimen(2))/(dimen(2)*dim(2));
%% Al mirar el punto (dim(2),dimen(2)), estoy imponiendo que se acabe
%%en él. Miro el camino más corto hasta él.
%%%%%Este es el dtw sin restricciones a novel global, sólo local.
%%%%%Normalizar por la longitud de la entrada POR la longitud del modelo
%%%%%beneficia demasiado a las palabras muy largas.
%Otra opción menos real:dist=Dcolat(dimen(2))/(dimen(2)+dim(2))*10;
```

➲ Modelos

Esta función calcula los modelos que emplearemos para la implementación hardware y para la software. Los pasos que se siguen para calcular los modelos son: detección de bordes de las tres ejecuciones que necesitamos para crear un modelo, filtrarlas pasándolas por el banco, detección de envolvente y alineamiento de dos ejecuciones guardando el camino (esta función la veremos a continuación), haciendo uso del camino se promedian ambas ejecuciones y el resultado se alinea con la tercera ejecución, también guardamos el camino que es lo que nos interesa para promediarla con el resultado del promedio de las dos anteriores, al resultado se le estandariza en longitud.

```
function [model,vs1f,vs2f,vs3f]=modelos(b,a,N,u,s1i,s2i,s3i,rui);
[n1,n2,max1]=ppioyfin2vhdl(s1i,rui,N);
%plot(s1i(n1:n2));
%pause;
s1=s1i(n1:n2);
[n1,n2,max2]=ppioyfin2vhdl(s2i,rui,N);
s2=s2i(n1:n2);
[n1,n2,max3]=ppioyfin2vhdl(s3i,rui,N);
s3=s3i(n1:n2);

s1f=salfil(b,a,s1);
s2f=salfil(b,a,s2);
s3f=salfil(b,a,s3);
vs1f=rsuenvhdl(s1f,N,u,max1);
vs2f=rsuenvhdl(s2f,N,u,max2);
vs3f=rsuenvhdl(s3f,N,u,max3);
[dis,cam12]=dtw2(vs1f,vs2f);
%s1 en el lado, s2 abajo
% real=>fila=>s1, imag=>columnas=>s2
dim=size(cam12);
rl=real(cam12(dim(1),dim(2)));
ig=imag(cam12(dim(1),dim(2)));
%los inicio al predecesor del punto final del camino
p=1;
while (and((rl~=0),(ig~=0)));
  %Todos los caminos comienza en el origen.(ver dtw2.m)
  %El punto que tiene por predecesor el 0,0 es el origen
  inmodel(:,p)=(vs1f(:,rl)+vs2f(:,ig))/2;
  % Por filas salidas de los filtros
  p=p+1;
  %el modelo esta invertido por filas.
  auxi=cam12(rl,ig);
  rl=real(auxi);%fila del predecesor
```

```
  ig=imag(auxi);%columna del predecedor
end;
di=size(inmodel);
for j=1:di(2)
  model12(:,j)=inmodel(:,di(2)-j+1);
end;

[dis,cam123]=dtw2(model12,vs3f);
dim=size(cam123);
rl=real(cam123(dim(1),dim(2)));
ig=imag(cam123(dim(1),dim(2)));
p=1;
while (and((rl~=0),(ig~=0)));
  inmodel(:,p)=(model12(:,rl)+vs3f(:,ig))/2;
  % Por filas salidas de los filtros
  p=p+1;
  %el modelo esta invertido por filas.
  auxi2=cam123(rl,ig);
  rl=real(auxi2);
  ig=imag(auxi2);
end;
di=size(inmodel);
for j=1:di(2)
  modelsn(:,j)=inmodel(:,di(2)-j+1);
end;
% falta normalizar la longitud del modelo.
%supongo que la longitud del modelsn>1.
if(di(2)==16)
  model=modelsn;
else
  for j=1:16
    x1=floor(((di(2)-1)/15)*j-((di(2)-16)/15));
    x2=ceil(((di(2)-1)/15)*j-((di(2)-16)/15));
    if(x1==0)
      model(:,j)=modelsn(:,x2);
    else
      if(x2>di(2));
        model(:,j)=modelsn(:,x1);
      else
                    model(:,j)=(modelsn(:,x1)+modelsn(:,x2))/2;
      end;
    end;
  end;
end;
```

➲ DTW para modelos

Esta función realiza el dtw guardando el camino.

```
function [dist,cam]=dtw(vmodf,vinf);
%%He creado un dtw2.m pq no necesito crear la matriz cam(ino) a no ser que
%vaya a crear los modelos.
dimen=size(vmodf);
dim=size(vinf);
Dcolp(1:dimen(2))=0;
for j=1:dim(2);  %columnas
  Dcolat(1:dimen(2))=0;
  dcola(1:dimen(2))=0;
  for n=1:dimen(2); %filas =>por cada punto de la matriz de alineamiento.
    e=0;
    for k=1:dimen(1);
      e=e+(vmodf(k,n)-vinf(k,j))^2;
    end;
    dcola(n)=sqrt(e); %%distancias locales de la columna actual.
    if (j==1)
      if(n==1)
        Dcolat(1)=dcola(1);
```

```
        cam(n,j)=0+0*i;
        %%%%pongo 0 pq no tiene predecesor
      else
        Dcolat(n)=dcola(n)+Dcolat(n-1);
        cam(n,j)=(n-1)+j*i;
        %%El predecesor es el (n-1,j)
      end;
      %%%%%%%%%hemos iniciado la 1ª  columna.
    else
      if (n==1);
        Dcolat(n)=dcola(n)+Dcolp(n);
        cam(n,j)=n+(j-1)*i;
        %%El predecesor es el (n,j-1)
      else
        %%%%%%para el resto de los puntos:
        [Y In]=min([Dcolat(n-1) Dcolp(n) Dcolp(n-1)]);
        Dcolat(n)=dcola(n)+ Y;
        if(In==1)
          cam(n,j)=(n-1)+j*i;
        end;
        if(In==2);
          cam(n,j)=n+(j-1)*i;
        end;
        if(In==3);
          cam(n,j)=n-1+(j-1)*i;
        end;
      end;
    end;
  end;
  Dcolp=Dcolat;
end;
dist=Dcolat(dimen(2))/(dimen(2)*dim(2))*100;
%% Al mirar el punto (dim(2),dimen(2)), estoy imponiendo que se acabe
%%en él. Miro el camino más corto hasta él.
%%%%%Este es el dtw sin restricciones a novel global, sólo local.
%%%%%Normalizar por la longitud de la entrada POR la longitud del modelo
%%%%%beneficia demasiado a las palabras muy largas.
%Otra opción menos real:dist=Dcolat(dimen(2))/(dimen(2)+dim(2))*10;
```

Capítulo

3

Adquisición de datos

ÍNDICE DEL CAPÍTULO

- Introducción
- El sistema de adquisición
- PCB
- Sobre la placa

Este capítulo trata sobre la parte más física del proyecto el desarrollo de la PCB para la adquisición de datos, esta tarea si bien es la más sencilla, requiere especial cuidado ya que reparar errores es más costoso.

INTRODUCCIÓN

El sistema se va a implementar como un sistema digital, dadas las posibilidades de procesado que éste ofrece frente a los sistemas analógicos, para ello es necesario digitalizar la entrada.

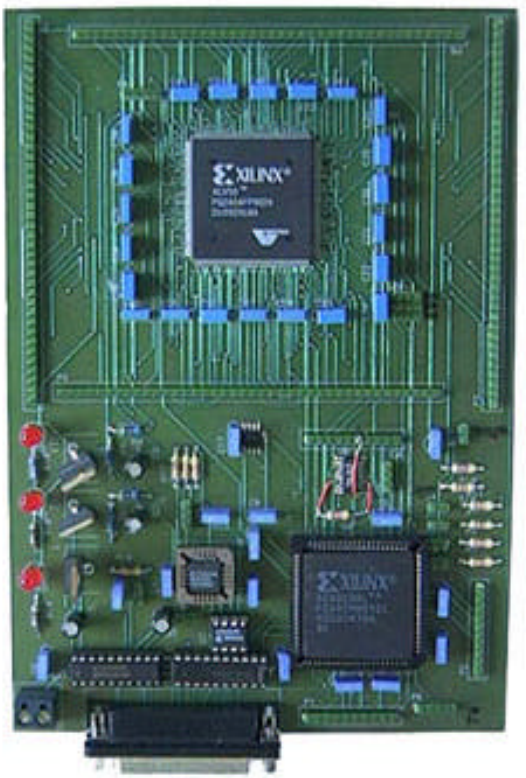


Figura 3.1.Proyecto HADES.

Toda la lógica que implementa el sistema de reconocimiento del habla se aloja en una FPGA, Virtex modelo V300PQ240. Para ello se trabaja con la placa del proyecto HADES[(1)],ver figura 3.1 desarrollada por *M. A. Aguirre* del Departamento de Tecnología Electrónica de la Universidad de Sevilla y tutor de este proyecto.

Para adquirir los datos que procesamos en la FPGA necesitamos añadir circuitería adicional a HADES. Los datos adquiridos por esta circuitería se introducen luego en la FPGA, para ser procesados de manera digital. La conexión de circuitería adicional a HADES se ha concebido de la siguiente forma: En torno a la Virtex, como puede verse en la figura 3.1, se han dispuesto cuatro conectores en hilera que se corresponden con pines de la FPGA de proposito general. Para conectar nuestra placa a HADES simplemente debemos dotarla de unos conectores en peine que encajen en estos conectores, ver figura 3.2.

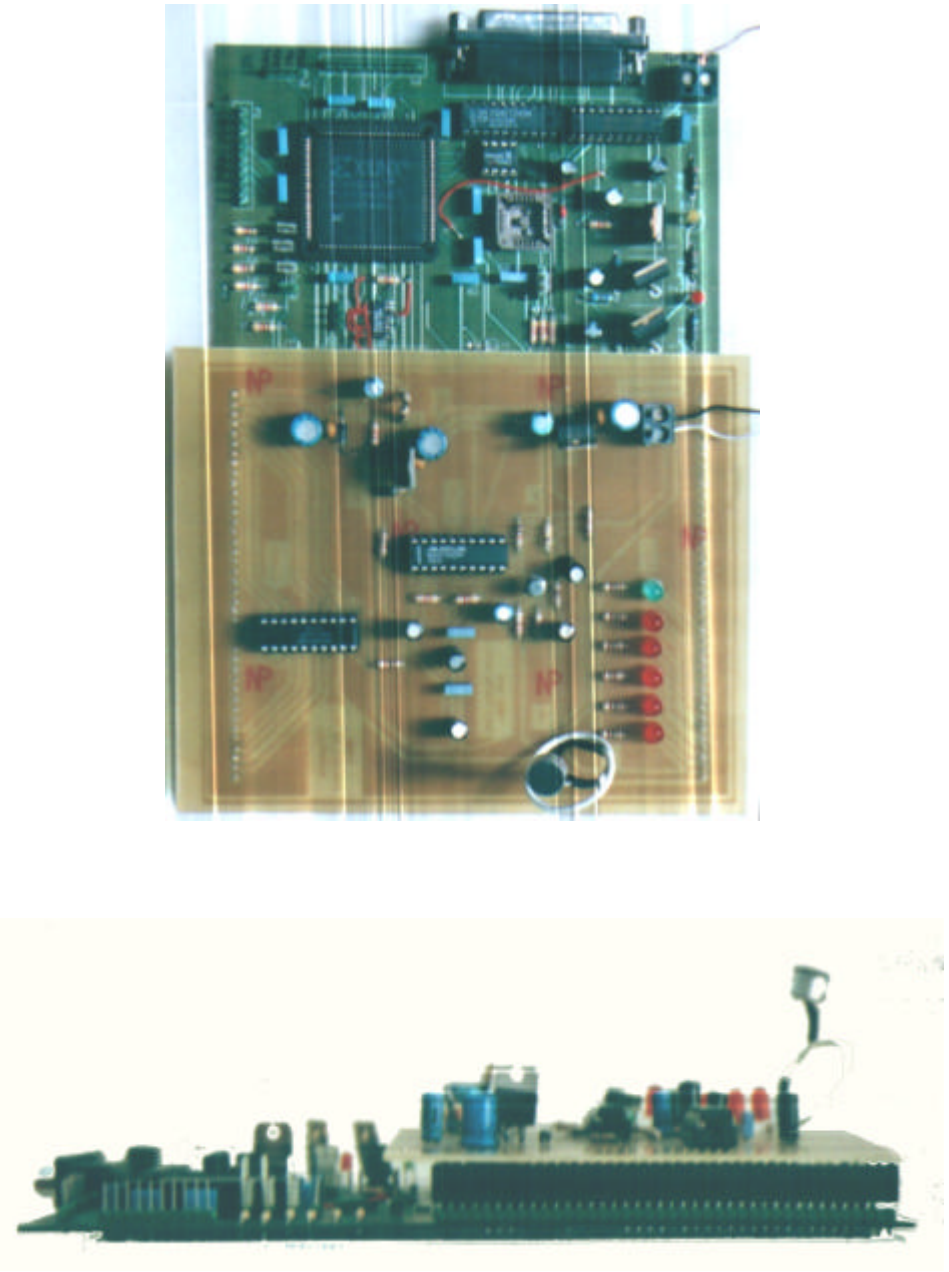


Figura 3.2 Fotografía de la placa para la adquisición de datos unida a HADES-1

Como puede verse en la figura 3.2. para nuestro proyecto sólo hemos necesitado dos conectores peine, aunque por el número de patillas podríamos dejarlo en uno.

EL SISTEMA DE ADQUISICIÓN DE DATOS

El circuito para adquisición de los datos necesarios para el reconocimiento del habla se compone de micrófono, amplificador, filtro anti-aliasing y CAD.

En la figura 3.3 vemos el esquema de bloques del circuito.

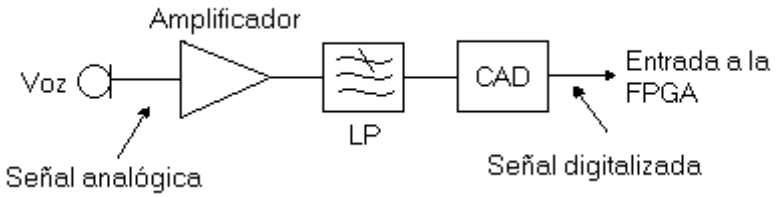


Figura 3.3: Esquema de bloques del circuito para adquisición de datos.

A continuación haremos una descripción detallada de cada uno de estos componentes y finalizaremos describiendo la PCB del circuito completo.

➲ Alimentación del circuito(2)

La placa de la Virtex, HADES, sobre la que vamos a montar nuestro diseño, se alimenta a nueve voltios continuos. Para obtener a partir de esta fuente de alimentación las tensiones que necesitamos para nuestro circuito de adquisición emplearemos dos reguladores de tensión, uno fijo (7805) y otro variable (LM713).

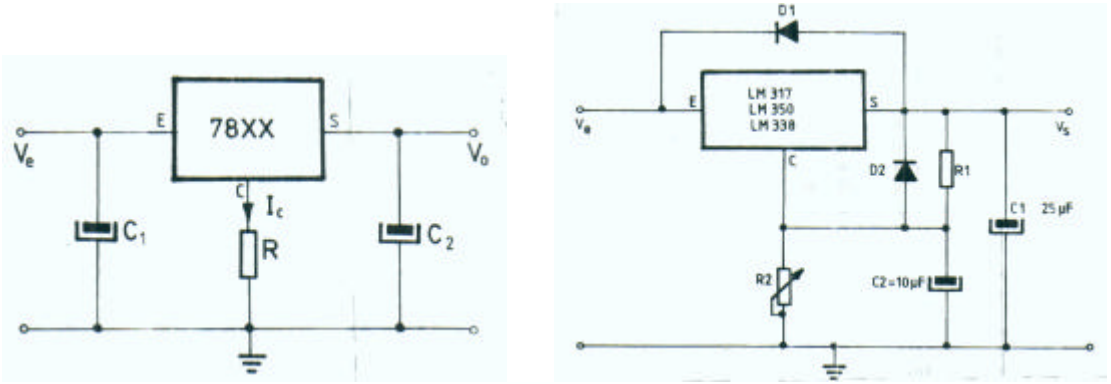


Figura 3.3. Esquemas de los reguladores de tensión. Para el esquema de la izquierda tomamos R=0, C1 y C2 del orden de unos 100µF y en paralelo a C1 colocamos un condensador de 100nF para rechazar el rizado de 50Hz de la alimentación, en realidad no se requieren condensadores tan elevados, ni tampoco un condensador para rechazar los 50Hz si pretendemos alimentar al circuito con una fuente continua, pero tampoco están de más. Para el esquema de la derecha también pusimos un condensador de desacoplo a la entrada de 100µF, aunque nuevamente, y aquí con mayor motivo, dado que se alimenta a partir del regulador de 5v, no es necesario o al menos no se requiere un condensador muy grande. R1 se tomó igual a 220Ω, R2 se sustituyó por dos resistencias en paralelo cuya suma es de 33Ω, C2=10µF, y C1=100µF con un condensador en paralelo de 100nF, al igual que a la entrada.

Los cinco voltios que obtenemos con el 7805 nos sirven para alimentar el circuito del micrófono, el amplificador y el filtro. El circuito del micrófono puede alimentarse hasta con 1.5v, es el típico circuito de los micrófonos electret que se alimentan con una pila de 1.5v. Sin embargo, la onda que obtenemos a la salida del micrófono es de muy poca amplitud. Como nuestro amplificador es un simple BJT obtenemos mejores resultados si aumentamos la alimentación del micrófono.

De igual manera el único dispositivo que requiere los 3.3v es el CAD, todos los demás toleran una alimentación a 5v.

Ambos circuitos llevan a la entrada dos condensadores en serie para filtrar la frecuencia de 50Hz, condensador pequeño, y el otro para filtrar rizados en la señal, si este rizado no existe como ocurre cuando la alimentación es continua puede disminuirse o incluso eliminarse, aunque dejarlo tan sólo repercutirá en costes. También el hecho de tener en serie los dos reguladores de tensión podría hacernos pensar que no es necesario poner condensadores de desacoplo a la salida del 7805 y a la entrada del LM713. Sin embargo, dado que la pista es bastante larga, por exigencias mecánicas, conviene mantener ambos.

➲ Circuito del micrófono[3]

Dada las características de nuestra aplicación debemos usar para adquirir la entrada un micrófono, que no es más que un transductor que nos convierte la onda mecánica que produce el sistema fonador humano, voz, en señales eléctricas proporcionales.

Existen muchos tipos de micrófono: de carbón (teléfonos), dinámicos, de condensador, piezoeléctricos, electret. Es de este último tipo del que hemos usado para el proyecto. Los motivos de su elección son los siguientes: Aunque su fidelidad no es tan buena como la del de condensador no requiere las tensiones tan elevadas de este. Tiene una sensibilidad de unos –70dB, amplia respuesta en frecuencia, margen dinámico de 120dB y aunque su impedancia sea elevada se ve reducida gracias a un circuito basado en un FET que viene incorporado en la propia cápsula y que requiere una alimentación mínima de 1.5v. Y por último son baratos y su tamaño es muy reducido.

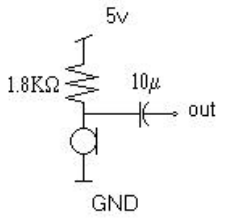


Figura 3.4. Esquema de alimentación del micrófono.

En la figura 3.4 podemos ver el circuito simple que se usa para el micrófono. Aunque la alimentación podría ser 1.5v lo alimentaremos a 5v como se indica en la figura 3.4 para evitar tener que disponer de un regulador más (véase apartado de reguladores).

La tensión que obtenemos a la salida de este circuito no supera los 400mv de amplitud, alimentando a 5v.

➲ Amplificador

El circuito amplificador se basa en un transistor BJT, con este circuito tan simple obtenemos una amplificación tal que la salida se halla entre 0 y 5v.

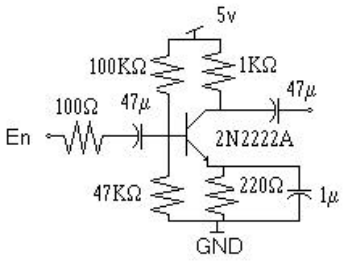
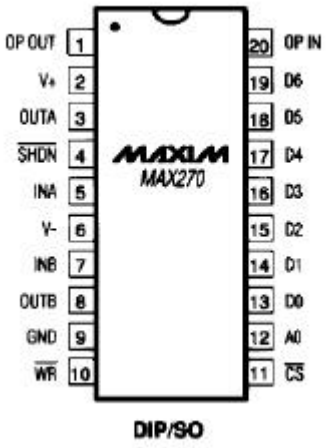


Figura 3.6:Circuito amplificador.

El transistor se encuentra en activa y tan sólo satura si la entrada supera los 400mVpp. La señal a la salida tiene una componente de continua de 2.2v . El condensador en el emisor elimina Re a la frecuencia de trabajo.

➲ Filtro: MAX270[4]

El filtro es un C.I. especificado para anti-aliasing y suavizado de señales a la salida de CDA. Se trata de un filtro butterworth continuo construido en torno a OPAMPs y de condensadores programables para variar la frecuencia de corte, el factor de calidad no se puede programar. Es un filtro de bajas frecuencias de corte, 1Khz á 25Khz, que se configuran fácilmente. Dispone de dos filtros en el chip y de un amplificador, nosotros sólo hemos usado uno de los filtros, el A. En el modo de funcionamiento pin-strap, que es el que hemos usado, las líneas para programar la frecuencia de corte se escriben por hardware uniéndose a positivo o negativo según interese, en nuestro caso la frecuencia de corte queremos que sean los 3.6KHz así que tenemos que escribir un 62 en las patillas de programación: ”1000001”. Las patillas /CS y /WE quedan deshabilitadas en este modo así que las uniremos a GND. El modo de ahorro de energía del que dispone este chip no lo emplearemos por no complicar el diseño, auque dadas las características del sistema, baja frecuencia de muestreo, sería interesante su uso tanto para el filtro como para el CAD. Para deshabilitar este modo basta conectar a nivel alto /SHDN. El CI admite alimentación bipolar o simple, esta última será la que empleemos.



MAX270

PIN #	NAME	FUNCTION
1	OP OUT	Uncommitted Op-Amp Output
2	V+	Positive Supply Voltage
3	OUTA	Filter A Output
4	SHDN	SHUTDOWN Control. Low level floats OUTA, OUTB, and OP OUT and places device into shutdown mode.
5	INA	Filter A Input
6	V-	Negative Supply Voltage
7	INB	Filter B Input
8	OUTB	Filter B Output
9	GND	Ground
10	WR	WRITE Control Input. A low level writes data D0-D6 to program memory addressed by A0. High level latches data.
11	CS	CHIP SELECT Input. Must be low for WR input to be recognized.
12	A0	Three-Level Address Input– logic high: addresses filter A logic low: addresses filter B connect to V-: pin-strap mode
13-19	D0-D6	7-Bit Data Inputs. Allows programming of 128 cutoff frequencies in a 1kHz to 25kHz range.
20	OP IN	Uncommitted Op-Amp Input

Figura 3.7: MAX270: patillaje.

A la salida del filtro tenemos la señal de entrada filtrada y un poco atenuada. El offset de esta señal no está centrado en el rango dinámico del CAD, por lo tendremos que adaptarlo.

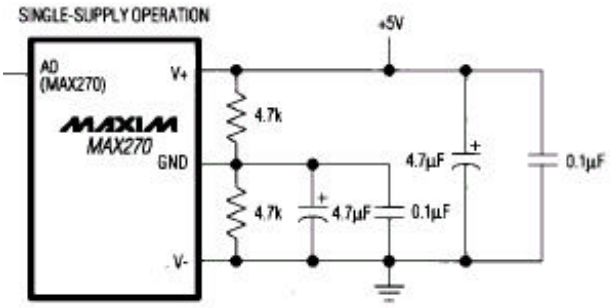


Figura 3.8: Alimentación del filtro.

➲ CAD: MAX152[5]

Es un convertidor analógico digital de 8bits y capaz de hacer hasta 100K muestras por segundo. Se alimenta de forma bipolar o simple a 3.0 á 3.6v (en valor absoluto). Dispone de un solo canal, así como de un mecanismo de bajo consumo para cuando no está operando, que tampoco vamos a emplear. El CAD tiene varios modos de funcionamiento, nosotros lo vamos a usar en modo lectura, en este modo se le indica que mustree y convierta bajando la señal /RD (para ello debe estar baja /CS, que hemos conectado a tierra en la PCB), cuando el CAD acaba la conversión indica la presencia del dato a su salida con INT a nivel bajo, una vez que subamos /RD el dato se quita de la salida – en la figura 3.9 vemos los cronogramas de la operación descrita -. Requerimos por tanto de una pequeña circuitería lógica para el manejo del CAD. Como el circuito incluido

en la FPGA debe saber cuando hemos recibido nuevo dato, para que lo procese y esto debe hacerlo cada vez que se cumple el tiempo de muestreo, le damos la orden de muestreo y conversión al CAD ciclos antes de que se llegue a los 125µs de muestreo, para que cuando INT se ponga a nivel bajo se cumpla aproximadamente el tiempo de muestreo, esto se hace para sincronizarnos con el contador de tramas.(ver capítulos posteriores).

Para adaptar los niveles de la salida del filtro anti-aliasing a la entrada del CAD usamos un divisor de tensiones con alimentación de 3.3v, hay que tener en cuenta que el filtro atenúa la señal, por lo que su rango dinámico disminuye de 0-5v á 0-3v, aproximadamente.

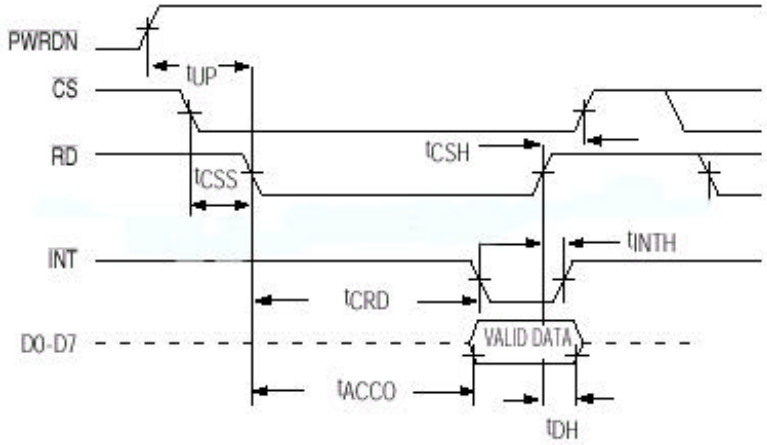
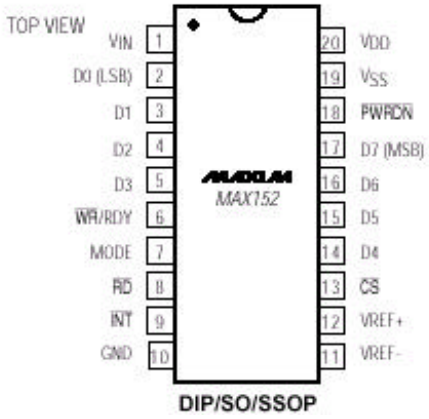


Figura 3.9: Cronogramas del CAD. Donde tACCO(max)=150ns (para una carga de 100nF).



Pin Description

PIN	NAME	FUNCTION
1	V_{IN}	Analog Input. Range is VREF- ≤ V_{IN} ≤ VREF+.
2	D0	Three-State Data Output (LSB)
3-5	D1-D3	Three-State Data Outputs
6	$\overline{WR}$/RDY	Write Control Input/Ready Status Output*
7	MODE	Mode Selection Input is internally pulled low with a 15µA current source. MODE = 0 activates read mode MODE = 1 activates write-read mode*
8	$\overline{RD}$	Read Input must be low to access data.*
9	$\overline{INT}$	Interrupt Output goes low to indicate end of conversion.*
10	GND	Ground
11	VREF-	Lower limit of reference span. Sets the zero-code voltage. Range is V_{SS} ≤ VREF- < VREF+.
12	VREF+	Upper limit to reference span. Sets the full-scale input voltage. Range is VREF- < VREF+ ≤ V_{DD}.
13	$\overline{CS}$	Chip-Select Input must be low for the device recognize $\overline{WR}$ or $\overline{RD}$ inputs.
14-16	D4-D6	Three-State Data Outputs
17	D7	Three-State Data Output (MSB)
18	PWRDN	Powerdown Input reduces supply current when low.
19	V_{SS}	Negative Supply. Unipolar: V_{SS} = 0V, Bipolar: V_{SS} = -3V.
20	V_{DD}	Positive Supply, +3V.

Figura 3.9: Patillas del MAX152.

P C B

Para el desarrollo de la PCB se ha empleado el programa Tango en su versión para windows, aunque éste entorno permite introducir un esquemático a partir del que obtener la PCB automáticamente no se hizo uso del mismo, sino que se realizó un rutado manual dada la simplicidad de la placa.

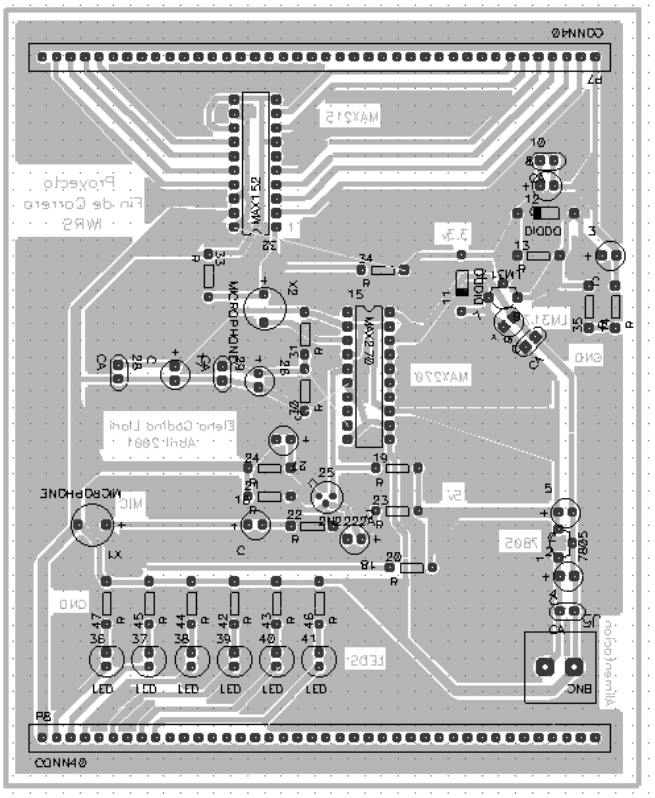


Figura 3.10. PCB. Esta figura tiene como objetivo mostrar el aspecto de la PCB, su resolución no es adecuada y su escala es tan sólo aproximada.

S O B R E L A P L A C A

La visualización de la respuesta del sistema se ha hecho a través de una serie de LEDs, se dispone de cinco LEDs rojos que se emplean para los comandos reconocidos, y uno verde que se utiliza para indicar situaciones de error. Si el LED verde se enciende mientras hablamos y se apaga al acabar el comando nos habrá indicado que la longitud del mismo ha excedido la longitud máxima prevista. Por razones de espacio ésta se redujo a 1.024s. Si LED verde permanece encendido después de acabar el comando o se enciende tras el final de éste, nos indica que la distancia entre todos los modelos y la entrada excede el umbral impuesto. Si se enciende más de un LED rojo se nos estará indicando otro error, que dos distancias coinciden.

Capítulo

4

Implementación física del sistema

ÍNDICE DEL CAPÍTULO

- Introducción
- Características del sistema y del entorno: Virtex 300
- Memorias
- Bloques del sistema
- DLL
- *Readback*: El elemento Capture

En este capítulo se expone la implementación física del sistema de reconocimiento del habla, además de las justificaciones para su elección . Esta implementación se ha acomodado a los recursos de los que dispone la Virtex 300.

INTRODUCCIÓN

Este capítulo es el centro del proyecto, en el se aborda la implementación hardware del sistema de reconocimiento del habla que es la meta de este proyecto. Sin embargo, para un completo entendimiento de lo que en él se expone es necesario un estudio profundo del capítulo dos, dado que la implementación física está constituida por bloques de circuitería que llevan a cabo las funcionalidades que se describieron en él.

Este capítulo se ha estructurado en cuatro puntos; el primero describe el entorno de trabajo de forma que conozcamos aquellos recursos de lo que disponemos; el segundo punto denominado memorias pone de manifiesto la importancia de este elemento en el sistema y describe cómo se han repartido los recursos de los que disponíamos; el tercer punto es el más extenso y detalla los distintos bloques que forman el sistema de reconocimiento del habla, aunque en primer lugar daremos una visión general del circuito de manera que sirva de guía para el resto del apartado; por último los puntos cuatro y cinco describen otros dos bloques que aunque no formen parte directamente de nuestro sistema son necesarios uno para su funcionamiento y otro para su depurado.

CARACTERÍSTICAS DEL SISTEMA Y DEL ENTORNO: VIRTEX 300

El prototipo del proyecto se ha desarrollado sobre una FPGA Virtex 300 de Xilinx. Estas FPGAs de última generación disponen de una serie de nuevas características, muy especialmente en la lógica de configuración de la FPGA (lógica que permite la carga de un diseño en la FPGA, es decir, recibe el flujo de bits que se le envía y lo guarda en la memoria de configuración de la Virtex, esta era la única funcionalidad que hasta ahora presentaba), dado que permite la lectura de la memoria de configuración de la FPGA. La lectura de la memoria de configuración de la Virtex se puede emplear tanto para comprobar que la configuración de la FPGA es correcta como para leer el estado las CLBs, registros de las IOBs, los valores de las LUT RAMs y de los bloques de RAM. Para esto segundo se emplea el elemento *capture*, como se verá en el apartado quinto de este capítulo. El sistema

para prototipado rápido HADES hace uso de todas estas nuevas características, por lo que permite aprovecharlas para el desarrollo rápido de prototipos. Este entorno fue el que se nos propuso para el desarrollo del proyecto.

➲ HARDWARE DEBUGGING SYSTEM

Se trata de un proyecto desarrollado por el departamento de ingeniería electrónica de la universidad de Sevilla que permite el depurado hardware de los prototipos implementados sobre FPGAs. Este sistema en su primera versión se empleo para realizar el prototipo del proyecto.

Las ventajas de un depurado hardware son evidentes para cualquier diseñador hardware, no sólo está el hecho de que lo que se verifique sea el propio sistema hardware y no un modelo para simulación, sino que permite usar patrones de entrada reales que en ocasiones son imposibles de emular y además se obtiene la respuesta en tiempo real, y no tras largas horas de simulación. Por otro lado, en ocasiones la implementación sobre la FPGA no es más que un paso previo en el desarrollo del sistema y no su forma final, por lo que este depurado sólo servirá para la fase previa.

HADES-1 se comunica con el PC a través del puerto paralelo empleando el protocolo EPP, lo que permite una tasa de transferencia de más de 500Kbytes. El enlace se usa en ambas direcciones, en la dirección PC-HADES-1 se transmite la información para la configuración de la Virtex y del reloj (DALLAS 1075), el sentido contrario se emplea para mandar hacia el PC los datos capturados de la Virtex, es decir, el estado del sistema en el instante que hayamos programado su captura como se verá en el punto quinto. Cabe destacar que la comunicación con el PC no la administra ningún microcontrolador como cabría esperar sino que se hace a través de una FPGA XC40000, la configuración de esta FPGA está guardada en una PROM y se carga en la FPGA durante el arranque de la misma.

El proyecto HADES-1 puede trabajar con todas las FPGAs de la familia Virtex, siempre que su encapsulado sea el PQ240. En el momento de la realización del proyecto se disponían de placas con la de cincuenta mil y trescientas mil puertas, dadas las dimensiones del sistema que se iba a desarrollar la más adecuada es la segunda opción, la V300PQ240. Esta FPGA no sólo dispone de más circuitería para la lógica del sistema sino que también dispone de más memoria interna a la FPGA.

➲ La familia de FPGAs Virtex: V300PQ240.

Device	System Gates	CLB Array	Logic Cells	Maximum Available I/O	Block RAM Bits	Maximum SelectRAM+™ Bits
XCV50	57,906	16x24	1,728	180	32,768	24,576
XCV100	108,904	20x30	2,700	180	40,960	38,400
XCV150	164,674	24x36	3,888	260	49,152	55,296
XCV200	236,666	28x42	5,292	284	57,344	75,264
XCV300	322,970	32x48	6,912	316	65,536	98,304
XCV400	468,252	40x60	10,800	404	81,920	153,600
XCV600	661,111	48x72	15,552	512	98,304	221,184
XCV800	888,439	56x84	21,168	512	114,688	301,056
XCV1000	1,124,022	64x96	27,648	512	131,072	393,216

Tabla 4.1.Miembros de la familia Virtex de FPGAs.

La familia **Virtex** del fabricante Xilinx la componen una serie de FPGAs de alta capacidad y prestaciones. En la siguiente tabla podemos ver los miembros de la familia así como algunas de sus características.

Los parámetros más importantes a la hora de escoger el miembro de la familia que se va a usar son los que se muestran en la tabla. En el sistema de reconocimiento del habla el parámetro del número de pines no es una restricción a la hora de escoger la FPGA, al menos en lo que en funcionamiento normal del circuito se refiere, para el depurado sí podría ser interesante disponer de más salidas.

MEMORIAS

La familia Virtex de FPGAs de Xilinx dispone de bloques del chip dedicados para crear memorias RAM síncronas de doble puerto, cada uno de estos bloques disponen de 4096 células de memoria. Además también se dispone de la RAM distribuida que ya estaba disponible en la familia XC4000.

La Virtex 300 cuenta con 16 de estos bloques de RAM, dispuestos en dos columnas de 8 bloques a ambos lados de la FPGA.

➲ Bloques de memoria del sistema

Para la elaboración del sistema de reconocimiento del habla se requirió de tres ROMS, una FIFO y tres RAMs. Las ROMs se emplearon para contener los modelos o plantillas de los comandos que el sistema reconocerá y para el banco de filtros, éste por haberse dividido en dos para realizar un procesamiento paralelo, por cuestiones de tiempo, necesita de dos ROMs (ROM1 y ROM2) independientes que puedan ser leídas a la par. La FIFO se emplea en el detector de envolvente. Las RAMs se emplean para almacenar la entrada una vez parametrizada(dtwRAM) y para realizar cálculos tanto en los filtros como en el dtw, estos dos últimos comparten los mismos bloques de RAM (RAM_res). En la tabla podemos ver los requerimiento de cada una de estas memorias y su ocupación en el prototipo teniendo en cuenta que una memoria ocupa al menos un bloque de RAM, dado que las Virtex disponen de circuitería interna especializada para memorias y dicha circuitería se agrupa en bloques de tamaño 4096 bits que comparten toda los circuitos de acceso.

Memoria		Nº palabras	Nº bits por palabra	Bloques
FIFO		20	8	1
DtwRAM		640	8	2
Md		2600	8	6
ROM1/ROM2 (cada una)		80	16	1
RAM_res	**Filtros**		16	1
	dtw	32	64	

Tabla 4.2. Memorias del sistema. El total de bloques de RAM ocupados es 13. En los tres bloques de RAM que sobran podrían emplearse para aumentar el vocabulario hasta 7 palabras, estaría por ver si el aumento de circuitería, que en principio no será mucho, cabe en a FPGA.

Todas las memorias se crearon usando la aplicación *Core generation* que suministra el fabricante para crear memorias usando la circuitería específica de la FPGA.

➲ Obtención de ROM1.mif y ROM2.mif

En la implementación software del sistema se obtuvieron los coeficientes de los filtros con la función *ellip* de la librería para el tratamiento digital de señales de matlab 5.1. Dicha función nos devuelve los coeficientes para una implementación directa II

transpuesta de los filtros, esta realización de los filtros, como ya se comentó en el capítulo 2, puede ser apropiada para matlab pero no para la implementación hardware de los filtros dada la cuantización de los coeficientes. Por ello, se empleará para la implementación hardware una realización cascada de los filtros, usando secciones de orden 2 en forma directa dos transpuesta. La implementación cascada es la más extendida para la realización de filtros clásicos, ya que en general sólo funcionará mal si algún polo está próximo al 1 o al –1, en cualquier caso siempre se podrá sustituir la forma directa II transpuesta por la forma acoplada para la realización de las secciones con polos próximos al 1 ó -1. Aunque la estructura lattice tiene mejores prestaciones, en cuanto a efectos de la cuantización, su complejidad hace que se prefiera, en general, la cascada, salvo en casos muy críticos donde los efectos de la cuantización sean de gran importancia para el sistema.

El proceso que hay que seguir para obtener los coeficientes para la implementación del filtro en cascada, comienza con la descomposición de la función de transferencia en un producto de , en este caso, cuatro secciones de segundo orden.:

$$H(z)=\frac{b_0+b_1z^{-1}+b_2z^{-2}+b_3z^{-3}+b_4z^{-4}+b_5z^{-5}+b_6z^{-6}+b_7z^{-7}+b_8z^{-8}}{a_0+a_1z^{-1}+a_2z^{-2}+a_3z^{-3}+a_4z^{-4}+a_5z^{-5}+a_6z^{-6}+a_7z^{-7}+a_8z^{-8}}$$

$$H(z)=k*\prod_{x=1}^{4}\frac{1+b_{1x}z^{-1}+b_{2x}z^{-2}}{1+a_{1x}z^{-1}+a_{2x}z^{-2}}$$

Para conseguir esta representación los pasos a seguir son: representar la función de transferencia del filtro completo en la forma polo, cero y ganancia; agrupar los polos complejos conjugados; asociarlos el par de ceros más cercanos; ordenar las secciones; y por último repartir la ganancia entre las secciones. En la figura 4.1 podemos ver la representación de los polos y los ceros de la funciones de transferencia de los 20 filtros, como la implementación es clásica los ceros caen todos sobre el circulo unidad. En cuanto a los polos como se puede ver está todos dentro del círculo unidad y lejos de los puntos críticos.

Para realizar todas estas transformaciones emplearemos la función de la librería de control de matlab: *zp2sos*. Ésta transforma la representación polo-cero-ganacia en una representación en cascada de secciones de segundo orden, para ello lleva a cabo el emparejamiento de polos y ceros, el escalado por secciones y reordena las secciones, para lo que le indicaremos que deseamos que la sección de polos más próximos vaya al final, de manera que se evite una resonancia que pueda producir una saturación a la entrada, de las demás secciones. Ejecutando en la línea de comandos lo siguiente obtendremos los coeficientes para la realización cascada:

```
for i=1:20;sos(i,:,:)=zp2sos(z(i,:),p(i,:),k(i)); end;
```

La variable *sos* es una matriz en tres dimensiones (20,4,6), donde la primera dimensión nos indica el filtro, la segunda la sección y la tercera el coeficiente. Cada sos(i,:,:) se organiza como sigue:

$$H(z)=\prod_{k=1}^{L}H_k(z)=\prod_{k=1}^{L}\frac{b_{0k}+b_{1k}z^{-1}+b_{2k}z^{-2}}{a_{0k}+a_{1k}z^{-1}+a_{2k}z^{-2}}$$

$$sos = \begin{bmatrix} b_{01} & b_{11} & b_{21} & a_{01} & a_{11} & a_{21} \\ b_{02} & b_{12} & b_{22} & a_{02} & a_{12} & a_{22} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ b_{0L} & b_{1L} & b_{2L} & a_{0L} & a_{1L} & a_{2L} \end{bmatrix}$$

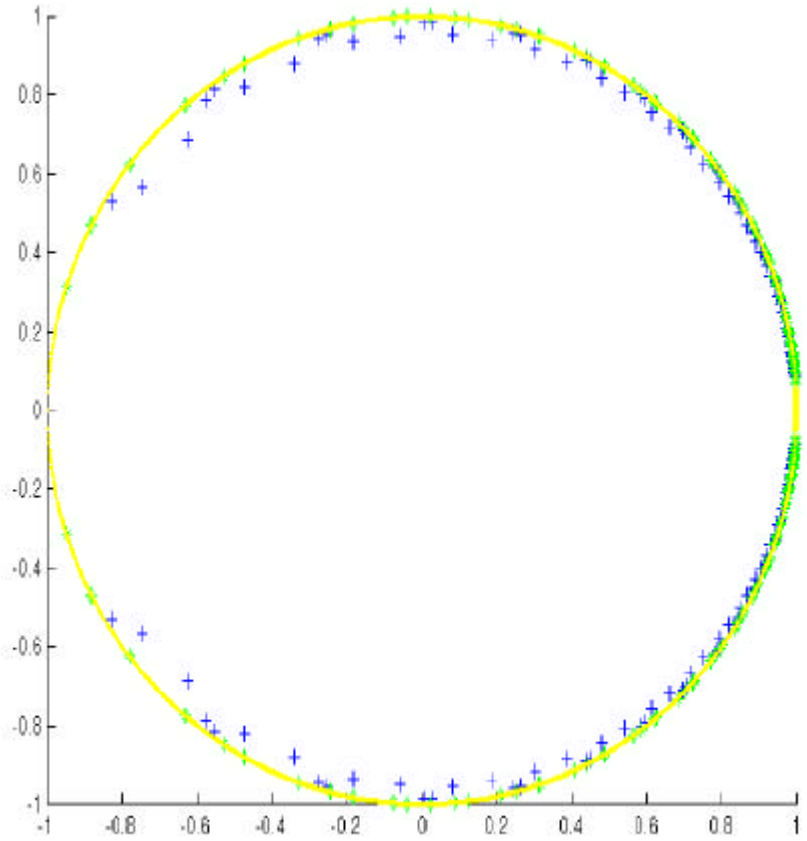


Figura 4.1. Polos y ceros de la función de transferencia antes de la cuantización a 16 bits. En azul los polos y en verde los ceros, en amarillo se ha representado el círculo unidad.

La variable *sos* contendrá, por tanto, los coeficientes de la implementación en cascada sin cuantizar ni codificar aún. Para realizar la cuantización hayamos el máximo en valor absoluto de *sos*, que resultó ser –1.9910. Para representar los coeficientes en el circuito hardware nos pareció apropiado emplear al menos 16 bits, dado que con 8 bits el paso de cuantización resultaba demasiado grande para representar un rango tan grande. Con 16 bits para codificar un rango de 4, el paso resulta de $6.1035*10^{-5}$. En la figura 4.1 se han representado los polos y ceros en el plano Z de todos los filtros, como puede verse a baja frecuencia, los polos están mucho más próximos, no sólo entre sí, por ser los filtros más estrechos, sino al círculo unidad que delimita la barrera para la estabilidad del filtro. La condición de estabilidad para un sistema de segundo orden es $/a_2/<1$ y $/a_1/<1+a_2$. en la tabla 4.3 se presentan los coeficientes del filtro a más baja frecuencia de banco. El valor más crítico es el coeficiente a2 de la cuarta sección, 0.9983. Cuando cuanticemos estos coeficientes a2 deberá permanecer menor que uno, esto se puede asegurar cuantizando a 16 bits, también se asegura con menos bits, pero nos quedaremos con 16 bits como compromiso entre prestaciones y recursos necesarios.

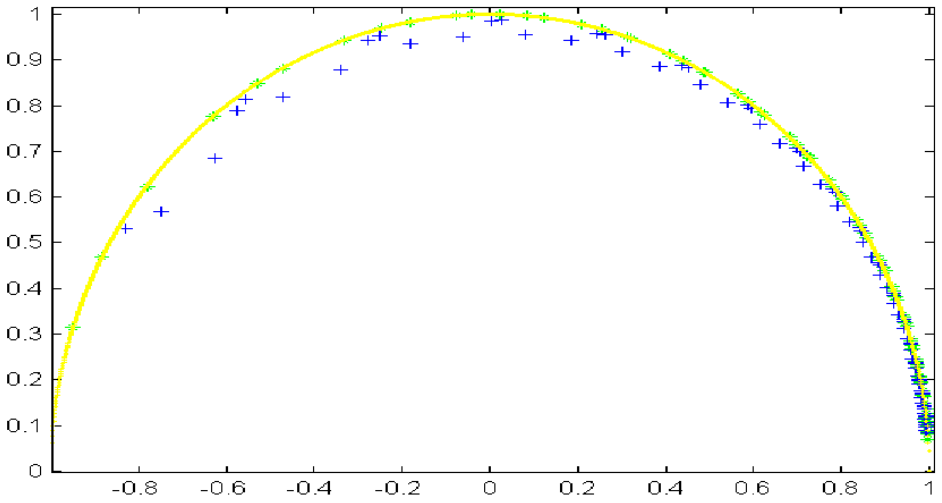


Figura 4.2 Posición de polos y ceros respecto del círculo unidad tras cuantiza a 16 bits, los coeficientes de la implementación cascada. Hemos empleado la función bancfil que se ha creado para generar los filtros, nos devuelve polos, ceros y ganancia de la función de transferencia al completo, empleamos zp2sos para obtener los coeficientes de la implementación cascada, los cuantizamos a 16 bits y empleamos sos2zp para obtener los polos, ceros y ganancia, que es lo que se representa en la figura..

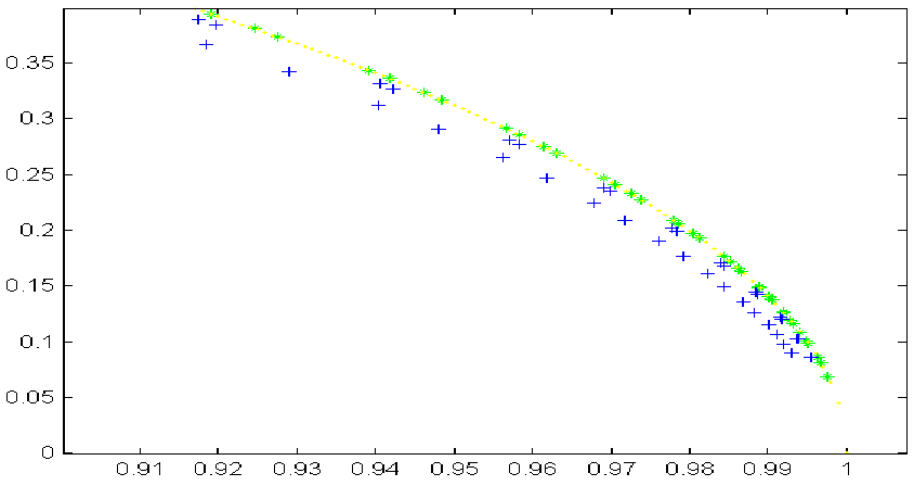


Figura 4.3 En esta figura vemos la ampliación de la zona próxima al 1. Se comprueba que los polos siguen dentro del círculo unidad.

Polos antes de cuantizar	Polos después de cuantizar
0.9937±0.1026i	0.9937±0.1026i
0.9919±0.0976i	0.9919±0.0976i
0.9930±0.0893i	0.9930±0.0895i
0.9955±0.0854i	0.9955±0.0855i

Tabla 4.4. Polos del filtro a más baja frecuencia antes y después de cuantizar. Como puede verse con 16 bits el efecto de la cuantización es muy pequeño.

Una vez que se tienen lo coeficientes y el tamaño de la palabra de cuantización pasamos a obtener ROM1.mif y ROM2.mif. Para lo cual, se escalan los coeficientes por $2^{16}/4$ y se redondean al entero más próximo, para lo que se emplea la función *round* de matlab. Por último, se codifican en complemento a dos, para ello elaboramos la función convc2.m y archc2.m que escribe el fichero con el mapa de unos y ceros para la ROM.(Esta función al igual que otras auxiliares se incluyen en el anexo 2), para lo cual, convierten a complemento a 2 los coeficientes b0, b1, -a1 y –a2 de cada sección que son los que se escriben luego en el archivo. Observar que por emplear filtros elípticos b0=b2 (Esto se seguirá cumpliendo aunque se cuantice, razón por la que aún tras ésta los ceros siempre permanecen sobre el círculo unidad) y a0=1 (al escalar y cuantizar será igual a $2^{16}/4$), estas propiedad se ha usado para ahorrar memoria, aunque en realidad no tiene mucho sentido ya que en cualquier caso se tienen que emplear bloques de RAM al completo. Una sección tan sólo necesita de cuatro coeficientes para representarse: b0, b1, a1, a2. Para eliminar a0 hay que tener en cuenta que la salida de cada sección tendrá que escalarse por $4/2^{16}$.

Coeficiente	Sección1	Sección 2	Sección 3	Sección 4
b0	0.0912	0.3108	0.4720	0.7427
b1	-0.1808	-0.6201	-0.9384	-1.4806
b2	0.0912	0.3108	0.4720	0.7427
a0	1	1	1	1
a1	-1.9838	-1.9859	-1.9874	-1.9910
a2	0.9934	0.9939	0.9980	0.9983

Tabla 4. 3. Coeficientes del filtro de más baja frecuencia para la implementación en cascada, sin cuantizar.

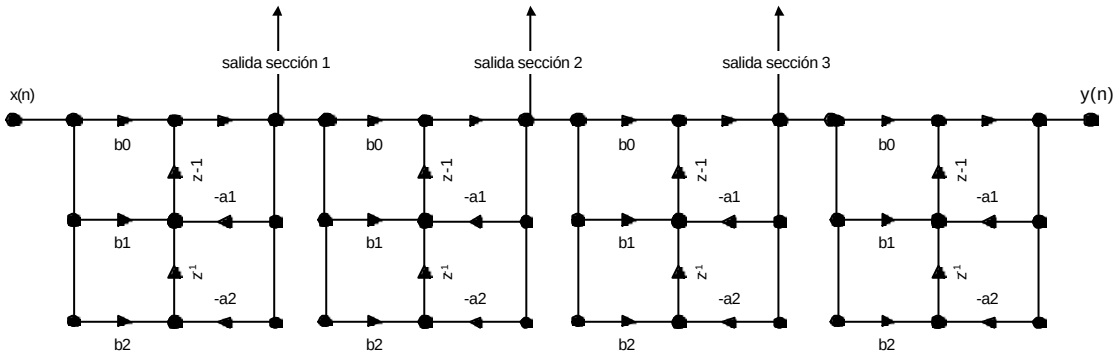


Figura 4.4. Estructura implementada para los filtros en su realización hardware.

Las memorias ROM1 y ROM2 tienen dimensiones: palabras de 32 bits por 128 palabras. En cada palabra se guardan dos coeficientes del numerador o del denominador de la función de transferencia de una sección. En ROM1 se incluyen los coeficientes de los diez filtros a más baja frecuencia y en ROM2 se incluyen los de más alta frecuencia. Como luego se verá los coeficientes se reparten en dos memorias para poder ser accedidos a la vez ya que se realiza el filtrado en paralelo. LA salida del bloque que realiza el filtrado va a ser salida del banco **i** en paralelo con salida del banco 1**i**. En la figura 4.3 podemos ver el reparto de las ROMs que se comentó antes, como puede verse los coeficientes del denominador se introducen ya con el signo negado.

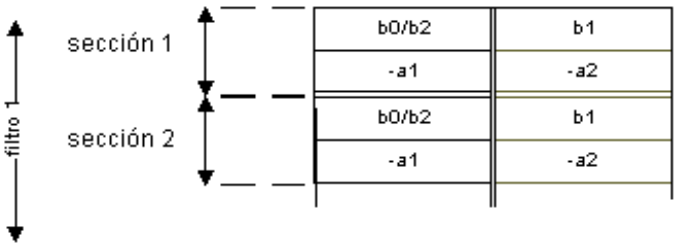


Figura 4.5. Estructura de las memorias ROM para los coeficientes del banco de filtros.

El archivo .mif está formado por una columna de ceros y unos (en el *Core generation* indicaremos que está en binario), en cada fila va el contenido de una palabra de la ROM, a la izquierda los bits menos significativos y a la derecha los más significativos. Según se ve en la figura 4.5, b0 se encuentra en los bits menos significativos de la palabra.

El espacio real ocupado por cada ROM es de 2*4*10=80 palabras por 32bits cada una son 2560 bits, como hay que usar al menos un bloque de RAM por memoria, se desperdician 48 palabras de 32 bits.

En la implementación de los filtros no hemos tenido en cuenta el escalado con objeto de evitar la saturación en los sumadores del filtro, dado que aunque se emplee complemento a 2, no se corre el riesgo de oscilaciones indeseadas dado que se han creado los sumadores con control para el desbordamiento, se controla saturando por arriba o por abajo.

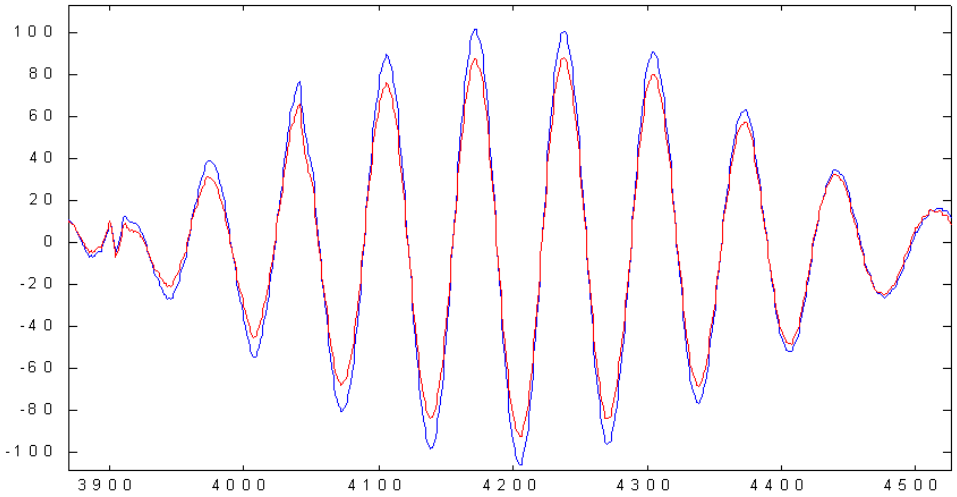


Figura 4.5. Comparación entre la salida del banco de filtros implementada en forma directa II y con resolución infinita en las operaciones (en azul) y en rojo la salida de una implementación cascada con las mismas limitaciones en la representación que la implementación hardware, es decir, la misma cuantización en los coeficientes y saturando productos y sumas a 2^31, se han creado salcasc.m y seccion.m, para este estudio. En esta gráfica se observa una ligera diferencia entre ambas, en otras zonas de la palabra en las que la las subidas y bajadas no son tan rápidas ni tan picudas ambas señales se ajustan la una a la otra. Esta señales son un fragmente de la palabra avanza en su banda primera.

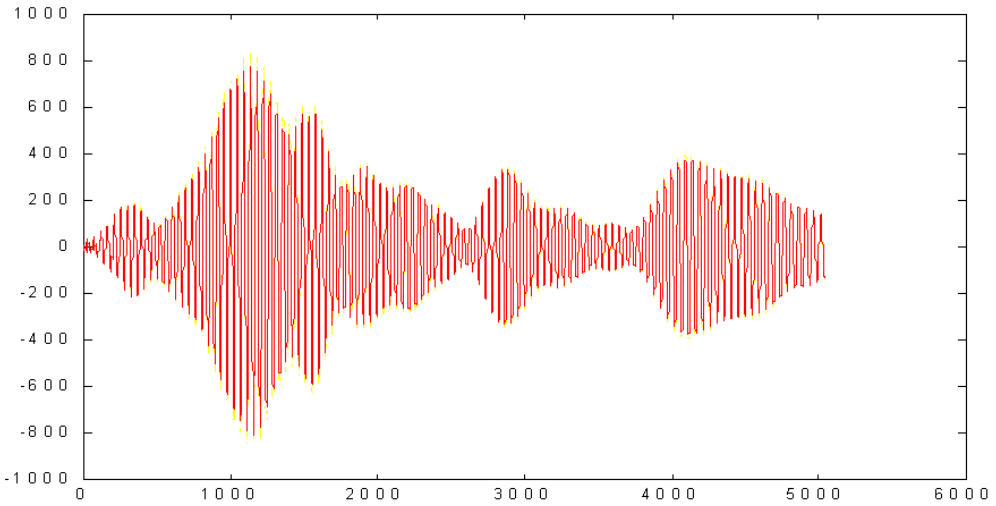


Figura 4.6 Comparación de la salida del filtro en forma directa II con resolución infinita y cascada con las limitaciones del sistema hardware. En la figura vemos la palabra avanza al completo en su banda tercera, en rojo la salida de la realización cascada y en amarillo la salida de la directa. Esta banda contiene mucha más energía. La hemos presentado al completo para mostrar como el resultado es bastante bueno.

Como conclusión de los resultados obtenidos podemos extraer que emplear los filtros en forma directa y sin tener en cuenta la resolución que se emplea en cada operación en el hadware, salvo en el resultado final no va a ser un foco de error grave a la hora de elaborar las plantillas para el hadware.

➲ Obtención de md.mif.

Este apartado es de suma importancia, dado que en él se describe como construir un archivo con las plantillas para el sistema de reconocimiento del habla. Como se ha descrito ya, la realización de los modelos mediante el propio circuito no nos ha sido posible dado que el circuito consume el 99% de la circuitería para la lógica y la totalidad de los bloques de RAM, aunque como ya se ha dicho en realidad hay RAM desperdiciada, ni aún aprovechándola toda tendríamos bastante, para crear los modelos en el propio circuito necesitaríamos de RAM externa, dado que para el cálculo de los modelos necesitamos guardar como mínimo dos entradas que se corresponderán con dos pronunciaciones de una palabra, para realizar el DTW entre ellas, este DTW se puede realizar con sólo dos columnas, como se hace en el prototipo, pero habrá que guardar una matriz con los predecesores de cada punto de manera que se pueda recuperar el camino para promediar las dos entradas. En cualquier caso, aunque pudiésemos aprovechar la RAM interna, por ejemplo usando RAM distribuida para la FIFO, no se dispondría de puertas suficientes para implementar la lógica, ya que aunque se pueden reutilizar los bloques para el cálculo del DTW, los bloques nuevos para el control y la normalización de la longitud, ocuparán bastante, además sería aconsejable poder oír la entrada antes de que se emplee para crear un modelo o imponerle un mínimo de energía respecto del estimado para ruido ambiente.

Por tanto para crear las plantillas para el prototipo hay que emplear el software. Los programas que realizan el procesado se han desarrollado en Matlab y son los mismos que se emplearon para la prueba software, *modelosvhdlv4.m*. Para emplear este archivo, que se incluye en el capítulo 2, tan sólo es necesario crear los filtros con *bancfil.m* y cargar la grabación de la voz en una matriz columna (por ejemplo, si la grabación es wav podemos usar wavread para conseguir la matriz), dicha grabación deberá haberse hecho en mono, a 8KHz y con 8 bits por muestra, se necesitan tres pronunciaciones de una palabra para crear

un modelo, estas grabaciones se introducen en variables distintas en matlab. También será necesaria una muestra de ruido obtenida en las mismas condiciones.

La función *modelosvhdlv4.m* nos devuelve el modelo y la parametrización de las tres pronunciaciones introducidas para crear el modelo.

La variable con el modelo obtenida es una matriz bidimensional, de 20 filas y 26 columnas, cada fila se corresponde con un filtro. Los elementos son la energía promedio de una trama de la envolvente de la salida del filtro correspondiente. Estos valores son flotantes y habrá que cuantizarlos y codificarlos para poderlos introducir en el hardware. Una forma más exacta de crear estas plantillas sería cuantizando los coeficientes de los filtros y la entrada y empleando *round* tras cada operación, además de saturar el valor teniendo en cuenta la resolución empleada por el diseño hardware en cada momento. En cualquier caso, se ha realizado de la primera forma por lo que tan sólo se escalan y redondean los resultados finales para ello realizamos un procesado parecido al de ROM1 y ROM2. La primera opción es: Buscar el máximo de los cinco comandos, que resulta ser 0.003, por lo que se tendrá que escalar por 85800, para codificar luego empleamos el binario base con 8 bits, dado que la energía promedio de la envolvente siempre es positiva, para ello empleamos la función *convbinbase.m*, que se incluye en el anexo; la segunda opción que es la que empleamos al final, consiste en escalar por el mismo factor que se escala en el circuito hardware, en éste la entrada al circuito digital tiene un rango entre –128 y 127, esta entrada se escala por 64 antes de entrar en el banco de filtros (pero no para el cálculo de la energía promedio de la entrada), este escalado se realizó para evitar quedarnos sin señal a la salida del banco de filtros y su ajuste fue experimental, el último escalado se realiza cuando se normaliza por el máximo valor de energía, donde en realidad se emplea 128/Emax para evitar quedarnos sin señal, por otro lado la entrada al sistema software una vez convertida en matriz en matlab se tiene un rango dinámico de –1 á 1, en el software no se realiza el escalado inicial por 64 y cuando se normaliza por Emax tampoco se multiplicó por 128, por ello para conseguir la energía promedio por bandas de los modelos normalizamos la salida obtenida con el software por $64*128^2$, para luego emplear *round* y limitar el rango dinámico a [0 255], por último empleamos *convbinbase* para obtener los modelos codificados.

ESQUEMA DEL SISTEMA

El sistema lo forman cinco bloques básicos y varios bloques más pequeños que realizan funciones auxiliares, estos bloques son los que se detallan a continuación:

- Bloque que maneja el convertidor analógico-digital, le da la orden de muestreo cuando corresponde y lee el dato convertido que éste le entrega. Además adecúa la representación numérica e indica al resto del sistema la presencia de una nueva muestra para que la procesen.
- Bloque que detecta la actividad, es decir, distingue cuándo sólo hay ruido de cuándo tenemos algo más a la entrada, para lo cual debe caracterizar el ruido, extrayendo algunos estadísticos de una muestra significativa del mismo. Una vez caracterizado el ruido debe analizar la entrada y compararla con las estimaciones de los estadísticos que ha obtenido del ruido para determinar si hay o no actividad.
- Banco de filtros. Como se expuso en el capítulo 2, cada trama de 32ms de la señal de voz se caracteriza por la energía promedio de la envolvente de veinte bandas tomadas desde los 120Hz hasta los 3.6KHz. Como paso previo a la obtención de la energía promedio hacemos pasar la señal de entrada por un banco de veinte filtros IIR paso de banda, con lo que obtenemos la señal en cada banda a su salida.
- Detección de envolvente y promediador de energía. Este bloque que en realidad lo forman dos sub-bloques opera sobre las salidas del banco de filtros, realizando sobre estas señales lo que su nombre indica, una detección de envolvente, para lo cual hay que

rectificar y filtrar paso de baja, y un promediado de energía. La energía la vamos a estimar, como también se vio, en el capítulo 2, como suma de las amplitudes en valor absoluto.

- DTW. Este bloque implementa en algoritmo de *dinamic time warping* que se expuso en el segundo capítulo. Éste bloque actúa sobre los parámetros extraídos de la entrada, mediante los dos bloques previamente mencionados, y que se habrán almacenado en RAM.
- Otros bloques menores que llevan a cabo pequeños cálculos o acciones que requieren los demás bloques.

En la figura 4.6 podemos ver un esquema del sistema al completo (de la parte que se ubica en la FPGA).

En el diseño del sistema se trato de aprovechar la posibilidad de procesado en paralelo que ofrece una implemetación hardware. Sin embargo, por razones de espacio el diseño tuvo que modificarse y pasar a una implementación serie en algunos bloques.

En la figura 4.4 se aprecia la existencia de un bloque de control central, que simplemente se ocupa de administrar y sincronizar los distintos bloques del sistema.

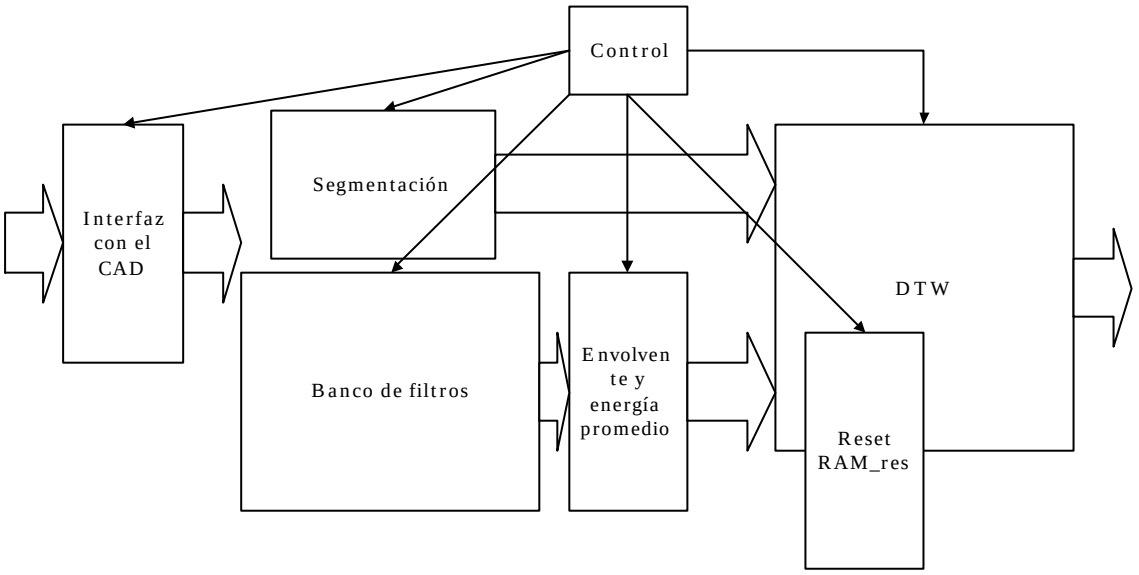
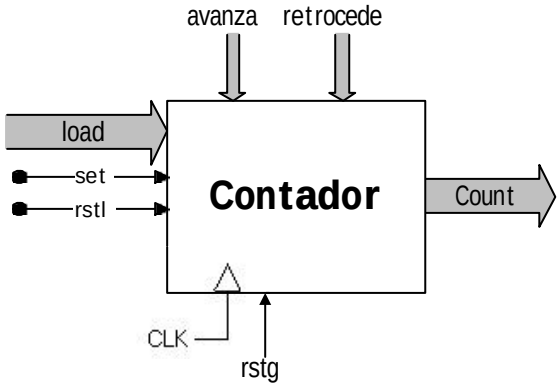


Figura 4.4. Esquema del sistema de reconocimiento del habla.

Para seguir la descripción bloque a bloque del sistema es aconsejable hacerlo con los archivos vhd presente, así como después de haber leido el capítulo 2.

BLOQUES ELEMENTALES



➲ Contador

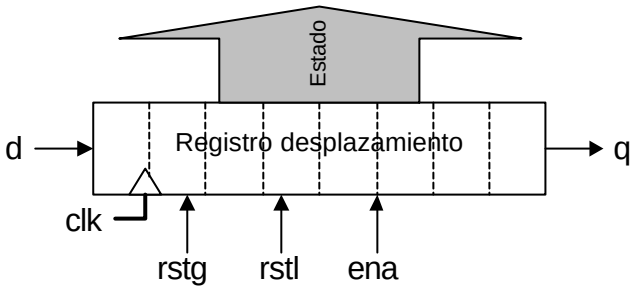
El contador es el bloque más empleado en el diseño. Su realización ha sido lo más genérica posible para evitar tener que disponer de varias versiones del mismo o de su estructura.

Está constituido por varios procesos en paralelo, el más importante el proceso de sincronismo con el reloj, otros que se basan en lógica combinacional, entre ellos está el proceso de cuenta hacia delante y hacia

atrás, con carga en paralelo y reset local, que es habitual en muchos contadores, lo peculiar de este contador, es que puede contar más de una unidad hacia delante o hacia atrás, para lo que dispone de otros cuatro procesos, un decodificador para avanza, dado que esta entrada no indica directamente cuánto avanzar, mientras que retroceder sí que lo hace, luego tenemos los procesos suma y resta para calcular cuánto avanza o retrocede el contador sin tener en cuenta el rango de cuenta del contador, de esto se cuida el proceso combinacional de operación normal de un contador. Por último, tenemos un proceso que simplemente compara el valor de la cuenta con un valor introducido como *generic* y si es igual lo indica.

➲ Registro de desplazamiento

El registro de desplazamiento se emplea tan sólo para la detección de actividad. Como se puede ver en la figura en este registro de desplazamiento se puede visualizar el contenido al completo.



➲ Valor absoluto

Este bloque se ocupa de eliminar el signo de aquellas señales en las que no nos es necesario. Para ello se emplea la función de la librería *arith abs*.

➲ Librería mistipos

Su nombre no indica bien su utilidad actual. Cuando se comenzó el diseño se trató de aprovechar la posibilidad de un procesado paralelo, razón por la que creamos algunos tipos nuevos para facilitar el uso de *generate*, sin embargo, las versiones posteriores no necesitaron de estos, y la librería ha quedado reducida a un listado de constantes. En realidad podría eliminarse.

DESCRIPCIÓN GENERAL DEL FUNCIONAMIENTO DEL CIRCUITO LÓGICO

El sistema atraviesa una serie de etapas en su funcionamiento. La primera es una etapa de arranque en la que se evalúa el ruido ambiente, para realizar un sistema que se adapte al ruido ambiente se necesitarían al menos dos micrófonos. El resto de las etapas se suceden en el funcionamiento normal del circuito, éste parte de un estudio de la entrada, de la que extrae parámetros y la compara con los umbrales estimados en la fase de arranque, por otro lado y en paralelo está parametrizando la entrada y almacena los vectores de señal en memoria, cuando se detecte un principio y fin de palabra se pasa a la fase de reconocimiento en sí misma. En la fase de reconocimiento hay que comparar la entrada con cada una de las plantillas, cuando esta comparación finalice se indicará qué palabra se ha reconocido y se pasará a la última fase del ciclo en la que se resetean las RAM_res que comparten el bloque del dtw y el banco de filtros.

➲ Contadores de muestreo y de tramas

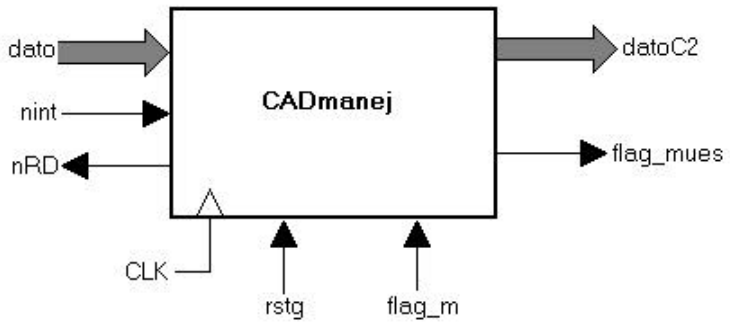
Durante las fases de arranque y parametrización-segmentación de la entrada son muy importantes dos temporizaciones, la del tiempo entre muestras y la temporización de las tramas. La primera se tomo de 125μs y la segunda de 32ms como se apuntó en el capítulo 2.

Para indicar ambos eventos se emplearon contadores independientes, si bien podría usarse para el contador de tramas un contador de menor tamaño que contase las ocurrencias de los instantes de muestreo. La razón por la que no se realizó de esta manera es por que se pretendió en un primer momento dar mayor libertad a la hora de escoger los parámetros del sistema de reconocimiento.

Estos dos contadores se ubican dentro de la entity *rdppfin*. Si bien la salida del contador de tramas llega al resto de los bloques que lo requieren, la salida del contador de muestreo sólo llega al bloque que maneja el CAD, para indicarle que capture y convierta una muestra, siendo la señal que proviene del CAD para indicarnos que ya ha realizado la conversión, la que se hace llegar al resto de los bloques, de forma que se les indica l a presencia de una nueva muestra.

➲ Bloque que resuelve el interfaz con el CAD

Este bloque está formado por una máquina de estados muy simple, que espera la orden de captura(flag_m), manda al CAD capturar(nrd) y realiza una espera activa hasta nint=0, momento en el que comunica al resto del sistema la presencia de un nuevo dato flag_cad=1. este bloque también realiza la transformación de los datos que le llegan del CAD a complemento a dos, observar que tan sólo hace falta un inversor para dicha transformación.



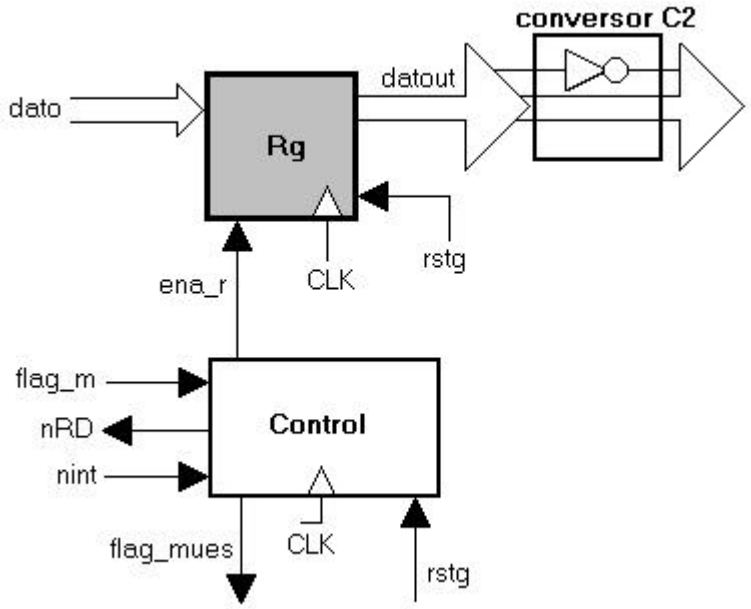


Figura 4.5 Esquema del bloque que maneja el CAD. Arriba vemos el interfaz y abajo vemos en detalle cómo está hecha.

➲ Bloque que evalúa el ruido en la fase de arranque

Esta funcionalidad la desarrolla parte de la entity *rdppfin.* Para evaluar el ruido, tras el reset global el circuito toma muestras del entorno durante cuatro tramas, de estas 4*256 muestras extrae dos parámetros para caracterizar el ruido la energía promedio y la tasa de cruces por cero. Ambas las promedia dividiéndolas por 4, en realidad tan sólo se dividen por dos, dado que el umbral se toma como el parámetro del ruido estimado escalado por dos. El promedio de energía se calcula sumando los valores absolutos de la entrada, para lo que contamos con un registro acumulador, una vez concluidas las cuatro tramas se escala el valor del registro y se le suma dos. Este sumando se ha introducido porque al ser la sensibilidad del micrófono muy baja, a no ser que el ruido ambiente sea considerable, su energía promedio será cero en general, razón por la que al pasar a reconocer palabras se disparaba enseguida el principio de palabra. La tasa promedio de cruces por cero se estima como la suma de veces que cambia el signo de la entrada divido por dos. Para su cálculo usamos un biestable que guarda el signo de la muestra anterior, y de una pequeña máquina de estados que se encarga de actualizar un contador y el biestable cuando se produce un cambio de signo. Para el cálculo del umbral también dividimos por dos y le sumamos una constante.

La estimación de estos parámetros para el ruido son los mismos que hay que realizar para la entrada para realizar la detección de actividad. Como ocurre que los dos cálculos no son necesarios a la par, se ha reutilizado la circuitería, además la energía promedio también se emplea para hallar el máximo de la energía promedio de la entrada que se emplea para normalizar los valores de energía promedio por bandas de la entrada.

El promedio de energía lo calcula la componente *promedio* y el promedio de la tasa de cruces por cero lo calcula la entity *promedz*

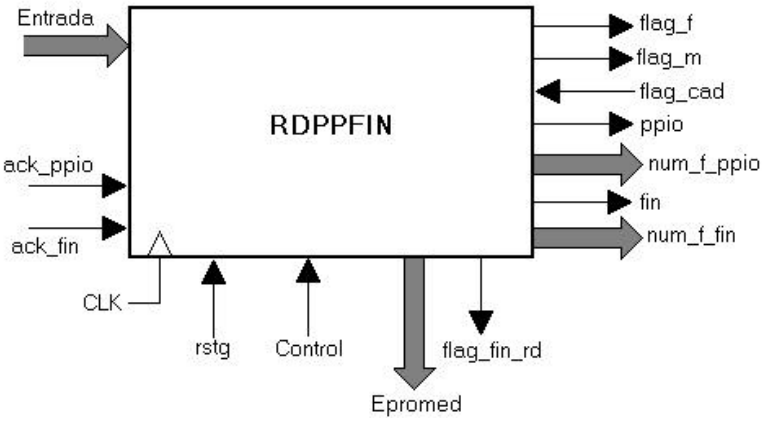


Figura 4.6. Interfaz del bloque rdppfin.

De la figura anterior cabe destacar la línea control que le indica al bloque que el circuito está en la fase uno el ciclo que comentábamos atraviesa el sistema, no hay ninguna señal que le indique que el sistema está en fase de arranque, de hecho cuando el sistema arranca "pensará" que está en la fase de parametrización-segmentación, siendo el único bloque que actúa de manera distinta éste. Durante las cuatro tramas iniciales flag_f (indica el fin de un frame o trama y viene de contador de tramas) no será directamente la salida del contador de tramas, sino que se mantiene a cero para el resto de los bloques y para parte del bloque rdppfin (aquellas partes que no intervienen en la evaluación del ruido) hasta la evaluación finalice. La máquina que controla el proceso de arranque es *fsm_rd*.

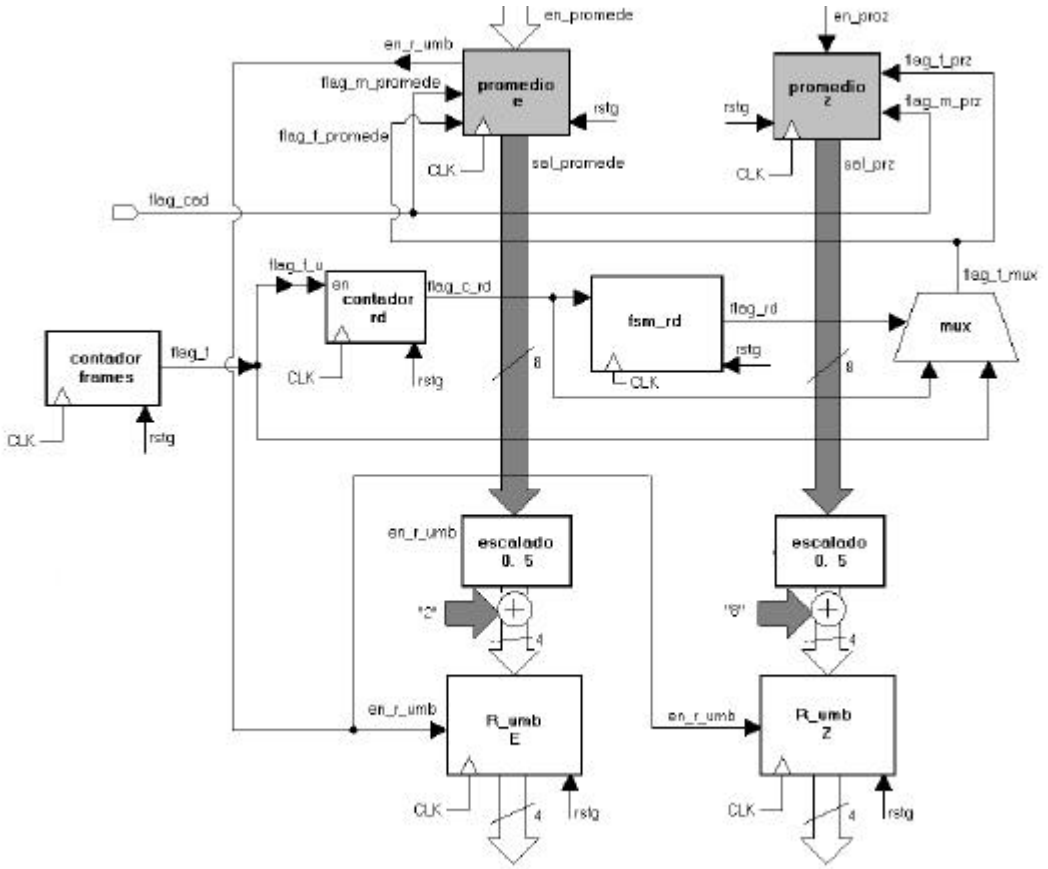


Figura 4.16 Esquema de la parte de rdppfin que evalúa el ruido.

SEGMENTACIÓN

➲ Detección de actividad

Esta funcionalidad la desarrolla el resto del bloque rdppfin. Para lo cual, a parte de la circuitería que comparte con la parte que evalúa el ruido, dispone de cuatro máquinas de estado, un registro de desplazamiento y un contador de módulo 10. Además para generar el número de tramas que hace que comenzó una palabra, cuando estamos detectando el principio, o el número de tramas que hace que concluyó cuando estamos detectando el fin, se incluyen dos procesos.

El contador nos servirá para contar el número de tramas seguidas durante las que se ha sobrepasado el umbral de energía, por arriba cuando es el principio de palabra y por debajo cuando es el fin de palabra.

El registro de desplazamiento se emplea para saber si ha ocurrido que se sobrepasó el umbral de cruces por cero tres tramas seguidas, una vez que se ha superado el umbral de energía durante un número de tramas seguidas estipulado.

Las cuatro máquinas de estado (fsm_comp_ppio, fsm_ppio_pab, fsm_comp_fin, fsm_fin_pab) se reparten en dos para la detección del principio y dos para la detección del final, su funcionamiento es totalmente análogo por lo que sólo describiremos el de las dos que detectan el principio de palabra. Fsm_comp_ppio sale de su estado de reposo cuando se termina de evaluar el ruido, a partir de este momento esperará a que se le indique el final de una trama para comparar los valores de energía promedio y tasa de cruces por cero de esa trama con los umbrales, si los superan por arriba avanzará el contador e introducirá un uno en el registro de desplazamiento, si no supera alguno de los dos o no supera ninguno, no realizará la operación correspondiente, es decir, no contará y/o introducirá un cero en el registro de desplazamiento. Una vez finalizada su operación da paso a fsm_ppio_pab, esta máquina comprueba si el contador es mayor que dos, que es el número de tramas seguidas que se impone para considerar que ha comenzado la palabra; si no lo es la palabra no se considera que haya comenzado, se vuelve al reposo y se devuelve el control; si sí lo es, se indica el principio de palabra, se resetean el contador y el registro de desplazamiento, esto es necesario ya que las máquinas para detección del fin de palabra harán uso de ellos, y se queda a la espera del fin de palabra, cuando llegue el fin de palabra, se resetean contador y registro de desplazamiento para fsm_comp_ppio (se ha preferido que sea sólo esta máquina la que maneje el reset síncrono para ahorrar un par de multiplexores.

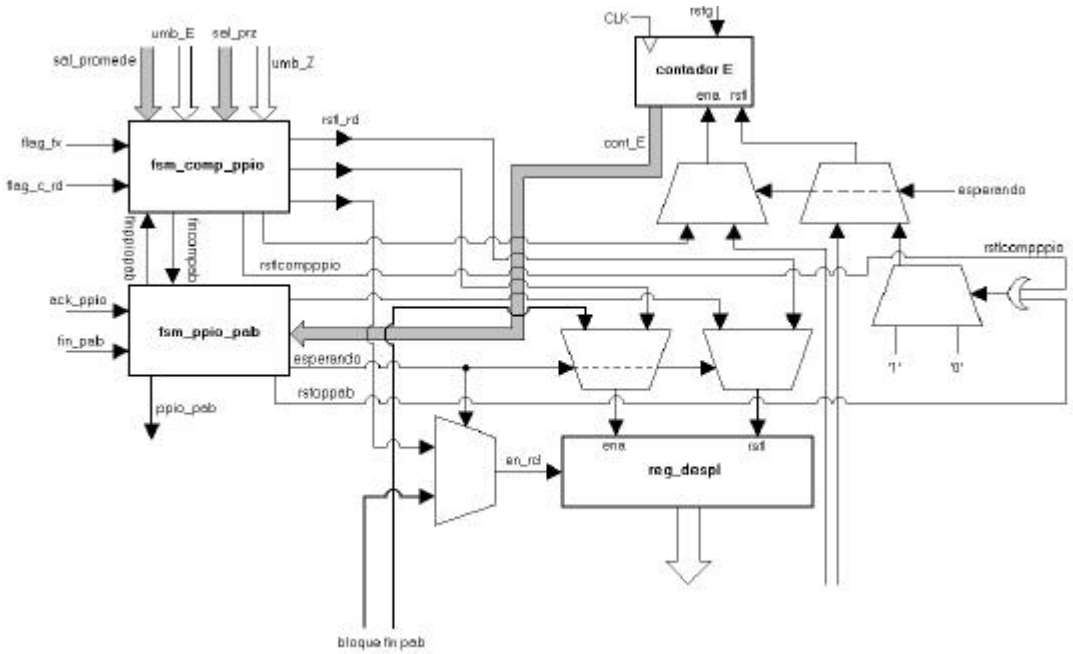


Figura 4.6. Conexión de las máquinas de estado para la detección del principio de palabra con el contador y el registro de desplazamiento.

Si la palabra comienza o termina en consonantes como las fricativas aunque su energía haya decaído o no haya superado aún el umbral es posible que ya haya comenzado, para ello evaluamos la tasa de cruces por cero y empleamos el registro de desplazamiento. Una vez que se cumple la condición de la energía promedio, y se ha indicado el principio o fin, se estudia el contenido del registro de desplazamiento, para ver si se cumple la condición que nos indica que la palabra comenzó algunas tramas antes de la actual o que finalizó algunas tramas después de la indicada, en realidad el final de palabra se detecta diez tramas después de que ocurra que la energía pase a ser menos que el umbral, razón por la que en el caso del fin, lo que indicará el registro de desplazamiento es cuántas tramas sumarle a la dirección actual menos 10.

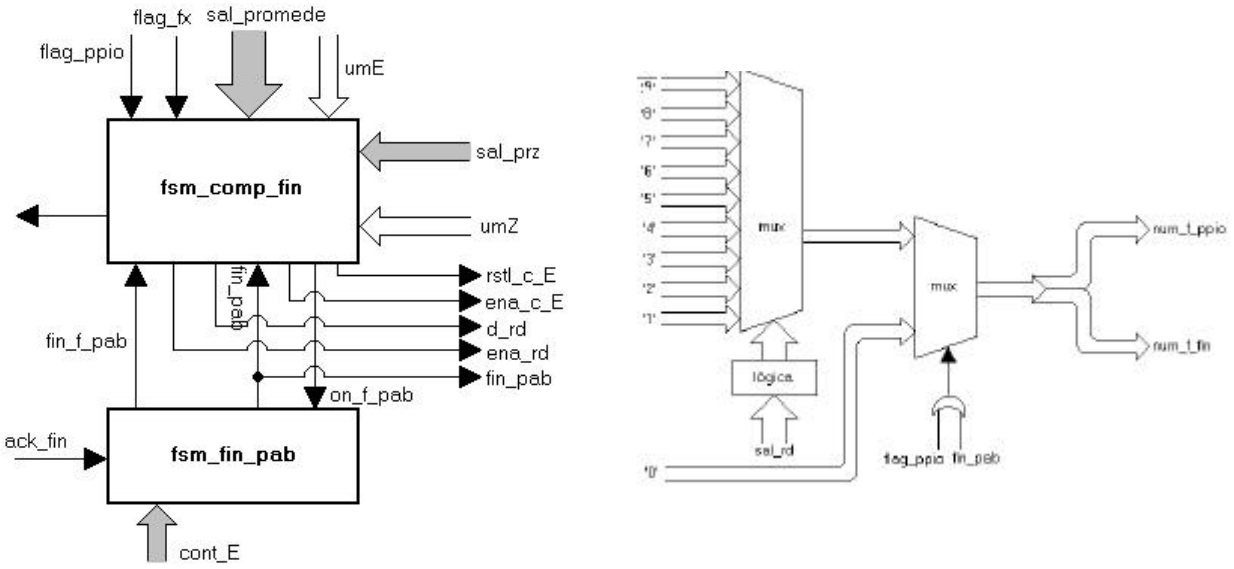


Figura 4.8. (a) Máquinas que detectan el fin de palabra. Para la detección del fin de palabra la energía debe caer durante diez tramas seguidas. (b) Detalle de la circuitería para obtener el número de tramas que hace que comenzó o finalizó una palabra a partir del contenido del registro de desplazamiento.

Por último, la detección de actividad no se limita tan sólo a indicar si la palabra ha comenzado o acabado, sino que también calcula las tramas que hace de ello, ya que como hemos comentado la tasa de cruces por cero se emplea para precisar si la palabra es algo más "larga" una vez que la energía promedio nos informa de su comienzo o conclusión, para ello se emplea la circuitería que se ve en la figura 4.8 (b), como puede verse es la misma tanto para el principio de una palabra como para el final. Luego se verá que la parametrización de la entrada está funcionando todo el tiempo que estamos en el primer ciclo del circuito, aunque todavía no se haya detectado el principio de una palabra, la entrada parametrizada se guarda en RAM (dtwRAM). Cuando se detecte el principio de palabra se calculará la dirección de comienzo del comando sobre esta RAM y cuando se detecte el final se calculará la dirección de la última trama de la palabra, con estas dos direcciones sabremos dónde se ubica la palabra que hay que reconocer. El cálculo de estas direcciones se lleva a cabo entre parte del bloque rdppfin (figura 4.8 (b)) y el bloque dirppyfin. Este bloque recibe el número de tramas que hace que empezó o finalizó la palabra, además de las indicaciones de tal comienzo y fin y de la dirección en la que actualmente se está almacenando en la dtwRAM. Hay que tener en cuenta que la dirección que se le pasa a este bloque, aunque todavía no se haya acabado de procesar la última muestra de la trama por parte del bloque que parametriza, es la dirección en la cuál se va a almacenar, dado que fsm_almacena_dtw_RAM, como luego se verá, actualiza la dirección tras almacenar. El bloque dirppyfin se encarga de restar a la dirección actual el número de tramas que hace que comenzó la palabra (o 10 menos el número de tramas, caso de que lo que se busque sea la dirección de fin) teniendo en cuenta que las direcciones dentro de la dtwRAM sólo pueden moverse entre 0 y 31, cuando se nos indique el principio o fin se realiza la captura de la dirección en un registro que la guardará para el DTW.

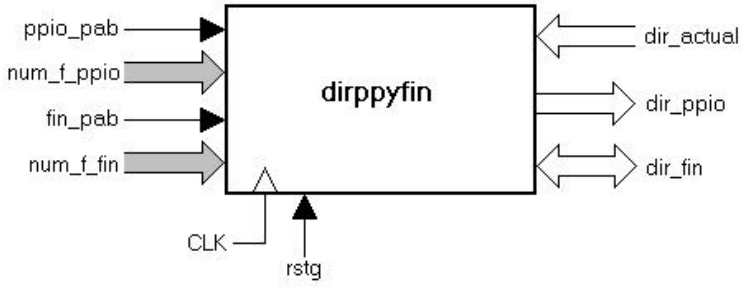


Figura 4.9. Interfaz del bloque dirppyfin.

INMUNIDAD A LA INTENSIDAD DE LA ENTRADA

El sistema de reconocimiento propuesto se basa en el cálculo de la semejanza entre la entrada y unas plantillas, si la amplitud de la entrada fuera muy superior o inferior a la de las plantillas podría ocurrir que la entrada no se reconociese de manera correcta. Para inmunizarnos contra este hecho, normalizamos la entrada y las señales empleadas para crear las plantillas. El factor de normalización escogido fue la energía promedio máxima evaluada en una trama de la señal al completo (sin descomponer en bandas) y sin tomar la envolvente. Este valor se obtiene simplemente estudiando las energías promedio calculadas para la detección de actividad y quedándonos con el máximo valor.

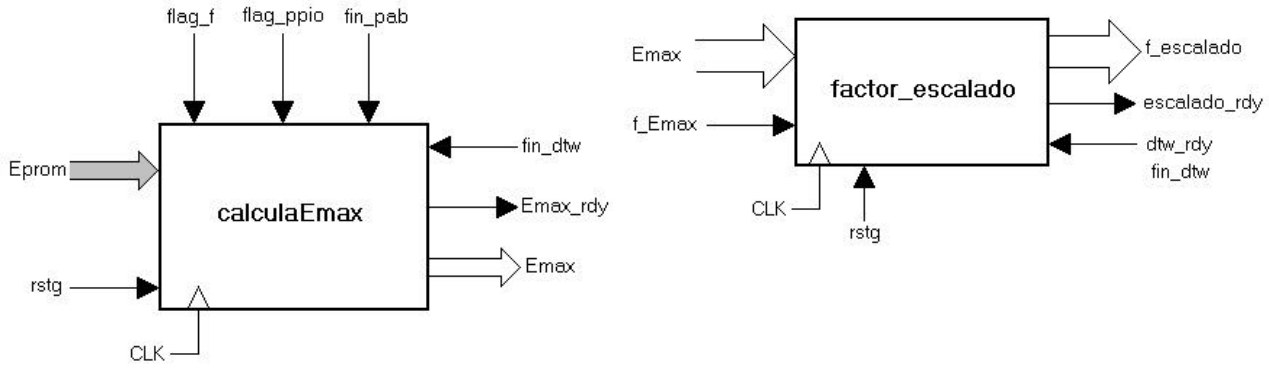


Figura 4.10 Detalle del interfaz de calcula Emax y detalle del interfaz factor de escalado.

El valor máximo de la energía promedio de la entrada no es directamente el factor de normalización o escalado que emplearemos para normalizar las energías promedio de la entrada en el DTW, sino que el factor de escalado es 128/Emax. La razón por la que multiplicamos por 128, es evidente, para no quedarnos sin señal.

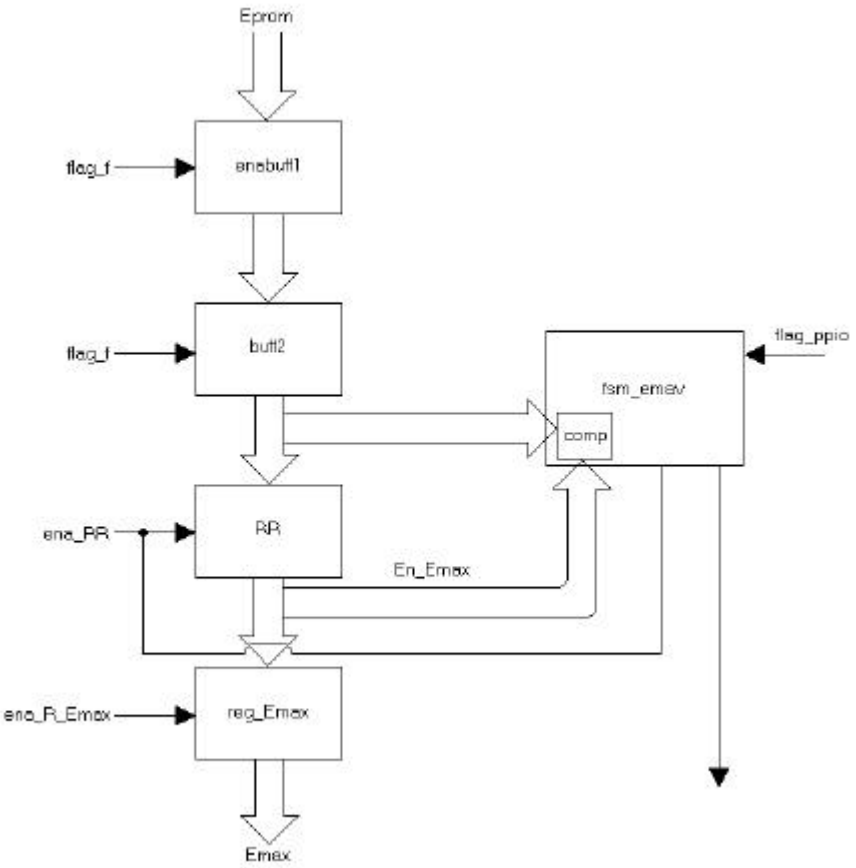


Figura 4.11. detalle del interior de calcula Emax. Los dos registros buffer se emplean para guardar el valor anterior y el actual de la energía promedio, ya que este bloque sólo compara energías promedio una vez ha comenzado la palabra, emplea el buffer porque la señal de principio de palabra puede llegar una trama después de que ocurra ya que hay que superar el umbral durante dos tramas para considerara que comenzó la palabra.

El cálculo del cociente se realizó de manera iterativa, esto es, buscando el factor que al multiplicar por Emax nos dé 128. Para este cálculo necesitamos un contador (k) y un comparador. El pseudocódigo del proceso es el siguiente:

En la figura siguiente se muestra el subsistema que implementa el pseudocódigo.

```
inicio k=0;
Bucle
Hacer k*Emax
            si menor que 128 hacer k=k+1 y seguir en e bucle.
            sino salir del bucle
      fin bucle
Si k*Emax=128 fin devolver K;
Sino
      k*Emax=k1;
      k1-128=q1;
      (k-1)*Emax=k2;
      128-k2=q2;
      Si q1=<q2 entonces devolver k;
      Sino devolver k-1;
Fin
```

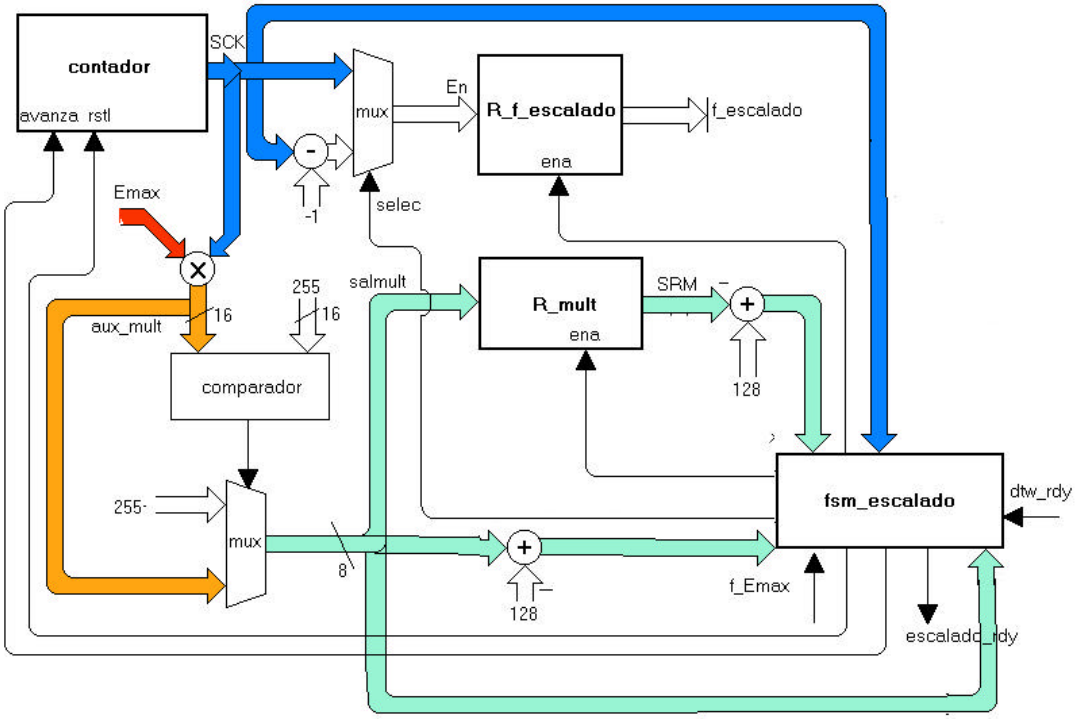


Figura 4.11. Esquema para el cálculodel factor de escalado.

Hemos empleado un registro (SRM) para guardar el producto anterior de manera que ni haya que repetir el cálculo ni se necesiten dos multiplicadores. El comparador que hay junto al multiplicador sirve para saturar el producto a 255. El comparador de q1 y q2 no se ha representado, pero se encontrará dentro de la máquina de estados. Este bloque espera a que calcula Emax termine para actuar y su ciclo no finaliza hasta que no termina el DTW, de manera que mantiene el factor de escalado todo el tiempo que dure éste.

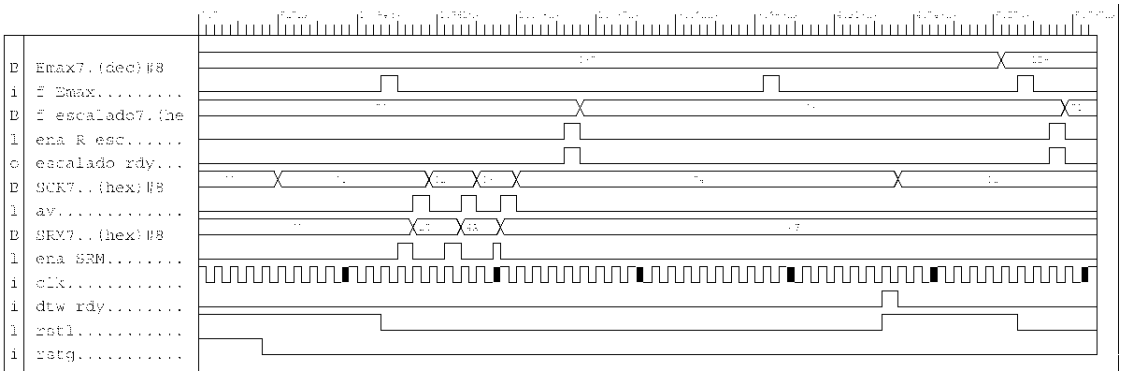


Figura 4.12 Cronograma funcional del bloque factor de escalado. Aquí vemos que aunque el diseño de la máquina de estados no es muy habitual funciona.

P A R A M E T R I Z A C I Ó N

Hasta ahora los bloques que se han visto se corresponden con la segmentación de la señal de entrada. Los siguientes bloques se encargan de extraer los parámetros que vamos a emplear para comparar la entrada con las plantillas. Este bloque se puede subdividir a su vez en cuatro tres bloques, *banco de filtros*, *secuenciador* y *energ_envolv_bandas.* El primero y el último son de tamaño considerable, mientras que el segundo es un pequeño bloque de adaptación entre ambos.

➲ Banco de filtros

Este bloque implementa veinte filtros paso de banda, a través de los que se hace pasar la señal de entrada, a las veinte salidas de este bloque se les calculará luego a energía promedio. Los filtros escogidos, como es habitual en voz, son IIR, dado que la voz no lleva información en la fase y los filtros IIR requieren menor número de coeficientes (por lo que su implementación consumirá menos recursos y además serán más rápidos) y se obtienen mejores respuestas espectrales, ya que al introducir los filtros IIR realimentaciones pueden implementar ceros y polos. Se escogió una implementación elíptica de estos filtros por ser a su vez la que menor orden daba con diferencia (orden 8). Como se comentó cuando hablamos de las memorias para la implementación hardware de los filtros se ha empleado una estructura cascada, con cada sección realizada en forma directa dos transpuesta (véase figura 4.4).

En un principio se intentó una implementación con un sólo multiplicador pero el tiempo de procesado requerido resultó excesivo, de las posibles soluciones que nos planteamos y dado que el tiempo de procesado no era mayor que dos por el tiempo entre muestras, la que se seleccionó fue la división de banco de filtros en dos, es decir, una implementación paralelo, que divide por la mitad el tiempo de procesado que pasó a ser de 95μs. La ventaja de esta implementación es que se pueden reutilizar todas las máquinas de estado que teníamos antes con unas pocos modificaciones. Además, la implementación en paralelo tan sólo requiere disponer de dos multiplicadores y disponer de dos ROMs y dos RAMs (antes necesitábamos sólo una de cada, pero de tamaño doble de las actuales), el resto (las máquinas de estado) son comunes.

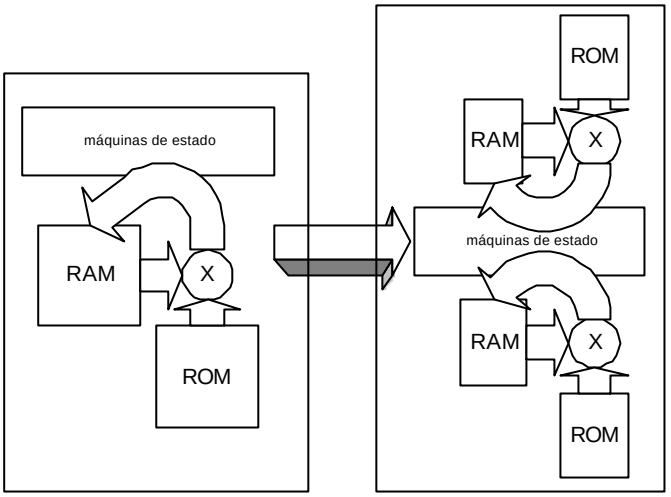


Figura 4.12 Esquema del banco de filtros implementación inicial y final.

Para controlar el flujo de datos se emplean varias máquinas de estado. Antes de pasar a verlas explicaremos la lógica usada para la descomposición de acciones en dichas máquinas. Para cada uno de los dos bancos de filtros disponemos de un multiplicador, de una memoria ROM con los coeficientes y con una memoria RAM en la que tenemos los resultados parciales o finales anteriores de cada sección y en la que almacenar los resultados parciales o finales que obtengamos. Las operaciones hay que realizarla para cada uno de los diez filtros (se realizan en paralelo para dos filtros: el 1º y el 11º, el 2º y el 12º...). Para cada filtro hay que evaluar cuatro secciones, las operaciones dentro de las secciones son siempre las mismas, tan sólo varían los coeficientes y las entradas. Para todas las secciones menos la cuarta las operaciones que hay que realizar una vez calculado el resultado de la sección son las mismas, guardar el resultado en RAM (RAM_res) y en el caso de la 4ª sección guardar el resultado en la RAM y reordenar los resultados entre secciones.

Las máquinas que componen el banco de filtros son:

- Start_f
- Filtrar10v
- Filtoper
- Sección
- Ram_I
- Ram_II
- Generador de direcciones RAM

☑ *Start_f*

Esta máquina se encarga de capturar la nueva muestra e introducirla en la RAM. El banco de filtros comparte con el DTW la RAM_res. Cuando se planteó la implementación del filtro en paralelo no se disponía de bloques de memoria suficientes para separar la ROM y la RAM en dos memorias independientes, sin embargo, como el DTW empleaba dos bloques de memoria para sus cálculos, se estudió la posibilidad de compartir éstos bloques entre el banco de filtros y el DTW. El DTW exigía un ancho de dato de 64 bits, mientras que el banco de filtros sólo necesitaba un ancho de dato de 16 bits. La estructura que adopta la RAM_res cuando la está usando el banco de filtros se puede ver en la figura 4.13. En esta figura se aprecia la ubicación de la entrada, la nueva muestra se introduce en ambas RAMs(la de cada filtro) en la posición que se ve en la figura, para ello antes de meter la nueva muestra hay que desplazar las anteriores, descartando la más antigua, este proceso lo lleva a cabo start_f. Las muestra de la entrada sólo se guardan en la dirección cero, por lo que para usarla en los demás filtros hay que acceder a esta dirección.

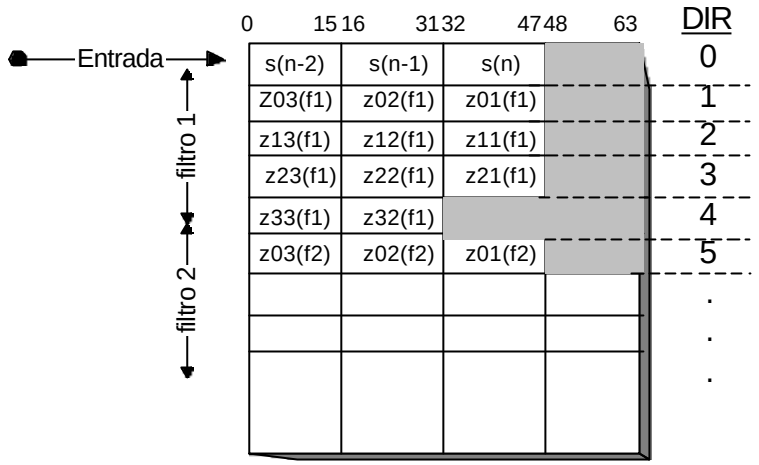


Figura 4.13 Estructura de la RAM_res cuando la emplea el banco de filtros.

Start_f funcionará, mientras el bloque de control_central le dé paso, cada vez que llegue una nueva muestra y siempre que el banco de filtros haya terminado (finbanco=1).

Start_f emplea tres registros (en cada filtro), uno para capturar la entrada, y los otros dos se emplean para la reorganización de la memoria.

☑ *Filtrar10v*

Esta máquina se encarga de que se realicen las operaciones correspondientes a los diez filtros. Para ello, da paso a filtoper y queda a la espera de que acabe incrementa un contador y vuelve a dar paso a filtoper, así hasta que el contador llegue a diez momento en el que actividad del banco de filtros habrá concluido.

☑ *Filtoper*

Esta máquina se encarga de que se realicen las operaciones correspondientes a las cuatro secciones del filtro, para ello va dando paso a tres máquinas sección, ram_I y ram_II, que realizan las operaciones básicas. Para las tres primeras secciones da paso a sección y cuando esta le devuelve e l control da paso a RAM_I y para la última da paso a sección y cuando le devuelve el control se lo pasa a RAM_II.

Para la primera sección filtoper pide la dirección de la RAM, dado que en las secciones siguientes esta dirección no hay que pedirla ya que es la última que pidió RAM_I, a la que debe acceder la sección.

☑ *Sección*

Esta máquina de estados administra la salida de datos de las memorias y la captura de los resultados en registros. Para ello una vez que filtoper se lo indica solicita un dato (sólo de la ROM ya que para la RAM ya se está accediendo a la dirección adecuada), él no genera la dirección directamente, sino que solicita que se genere a las máquinas adecuadas, lo que sí genera son los enables de las memorias. Una vez solicitado el dato espera un tiempo suficiente para que se impongan las direcciones, se obtengan los datos y se opere con ellos antes de realizar la captura en un registro acumulador. Las operaciones que hay que realizar por sección son cinco productos y cuatro sumas, como puede verse en la figura 4.2. Dichas operaciones se realizan en secuencia. El cuello de botella no se encuentra en los operadores aritméticos, en este caso, sino que lo que nos ralentiza es la extracción de datos de las memorias. Sin embargo, por la estructura de la memoria con una sólo lectura se extraen

tres datos, y simplemente es necesario un mutiplexor para seleccionar qué dato pasa al operador, por ello aunque la lectura inicial del dato sea algo lenta las operaciones posteriores son bastante más rápidas. Si bien de la RAM con una lectura se obtienen tres datos (en la primera lectura) podría pensarse que de la ROM sólo se obtiene dos coeficientes, pero si se observa la figura 4.3, y recordamos que los filtros son elípticos veremos que nos basta con estos dos coeficientes ya que el primero y el tercero son idénticos. Los datos para el cuarto y quito producto se piden cuando se está guardando el resultado del tercero.

☑ *RAM_I*

Esta máquina guarda en el tercer grupo de 16 bits del dato de la última dirección que solicitó sección el contenido del acumulador, es decir, el resultado de la sección y pone a cero el acumulador. Este dato almacenado junto con los que ya había en memoria constituirán la entrada a la siguiente sección. Este bloque no tiene que pedir la dirección, se limita a imponer la habilitación de lectura.

El registro acumulador es de 32 bits, ampliamos la resolución para los cálculos intermedios de la sección, además el sumador controla el *overflow*. Los dieciséis bits que metemos en memoria son los bits 14 á 30 del registro acumulador, este escalado se corresponde con el escalado de los coeficientes que comentamos en el apartado memorias.

☑ *RAM_II*

Esta máquina realiza las mismas operaciones que la anterior pero además reordena la memoria para cuando llegue la siguiente muestra. Como se ha visto en RAM_I, ésta almacena el resultado de la sección en el tercer grupo de 16 bits del dato, como vimos en sección para las operaciones cuarta y quinta sólo necesita dos operandos que en la RAM espera encontrar en los grupos 1º y 2º de 16 bits del dato, es en este orden en el que RAM_II deja la memoria descartando el dato más antiguo de los tres. RAM_II reordena los datos filtro a filtro.

Lo primero que organiza son los datos de la última sección ya que así no tiene que pedir esta dirección, para los demás tendrá que pedirla. El proceso es simple, lee el dato, fabrica el nuevo descartando el primer grupo de 16 bits y lo escribe en memoria, para la última sección no lo tiene que leer. Este bloque también pone a cero el acumulador.

Hay que tener en cuenta que el dato nuevo que este bloque mete en RAM es la salida del filtro i, así que no sólo se almacenará en RAM sino que también se le pasa al bloque secuenciador para que vaya operando con él. Como son dos bloques de filtros en paralelo a secuenciador se le pasarán dos resultados a la par.

☑ *Generador de direcciones RAM*

Tanto para la ROM como para la RAM disponemos de un contador que es el que impone la dirección en la memoria. En el caso de la ROM no se necesita de ninguna máquina que controle dicho contador, dado que siempre que se desee una nueva dirección bastará con incrementar el contador. En el caso de la RAM sí que es necesario una máquina que lo controle. Si bien en sección tan sólo habrá que avanzar el contador, cuando interviene RAM_II hay avances y retrocesos. Además no sólo es el contador el que impone direcciones en la RAM, sino que cuando se accede a las muestras de la entrada se impone la dirección de forma directa.

El funcionamiento del generador de direcciones RAM es el siguiente, la primera petición de dirección lo saca del reposo, pero se limita a inicializa un contador para ahorrar estados, ya que en estos momentos impone la dirección start_f, se queda a la espera de una

segunda petición a partir de ésta sí maneja su contador e indica que será ella la que imponga la dirección. La secuencia de avances y retrocesos es la siguiente : 2a, a, a, a, 3r, a, a (donde la a=avanza y la r=retrocede). Esta secuencia se explica en la siguiente figura.

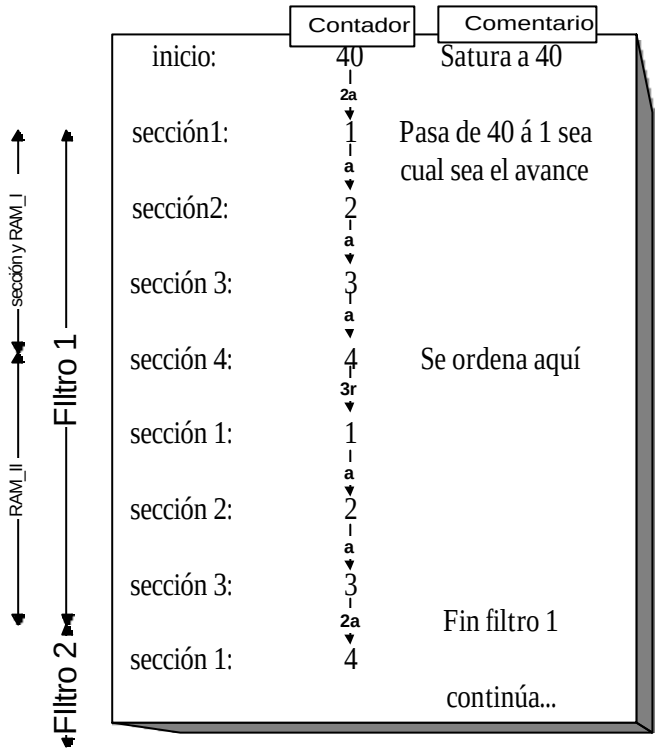


Figura 4.15 Secuencia de acceso a RAM.

Una vez estudiadas las máquinas de estado que controlan el flujo de datos del banco de filtros es fácil de entender su funcionamiento. El subsistema espera a que llegue una nueva muestra, esta muestra es escalada y almacenada en RAM (la muestra se escala para ganar resolución). A continuación comienza el filtrado para lo que actúan todas las máquinas vistas, las salidas de los filtros se obtiene secuencialmente en el tiempo a pares, según el orden, 1-11, 2-12, 3-13... Esto habrá que tenerlo en cuenta en los bloques que reciben los datos. En la página siguiente se incluye un esquema del banco de filtros.

➲ Secuenciador

Este es un bloque muy simple que sirve para adaptar los interfaces del banco de filtros y del energ_envolv_bandas, es decir, pasa de las dos salidas simultáneas del banco de filtros a dos salidas multiplexadas en el tiempo. Para poder hacer esto permitimos que energ_envolv_bandas nos avise cuando termine de procesar la muestra para enviarle la siguiente, obviamente el procesado de energ_envolv_bandas debe tardar menos que el del banco de filtros, como ocurre en realidad.

El secuenciador captura la salida del segundo banco de filtro, mientras que la del primero la deja pasar.

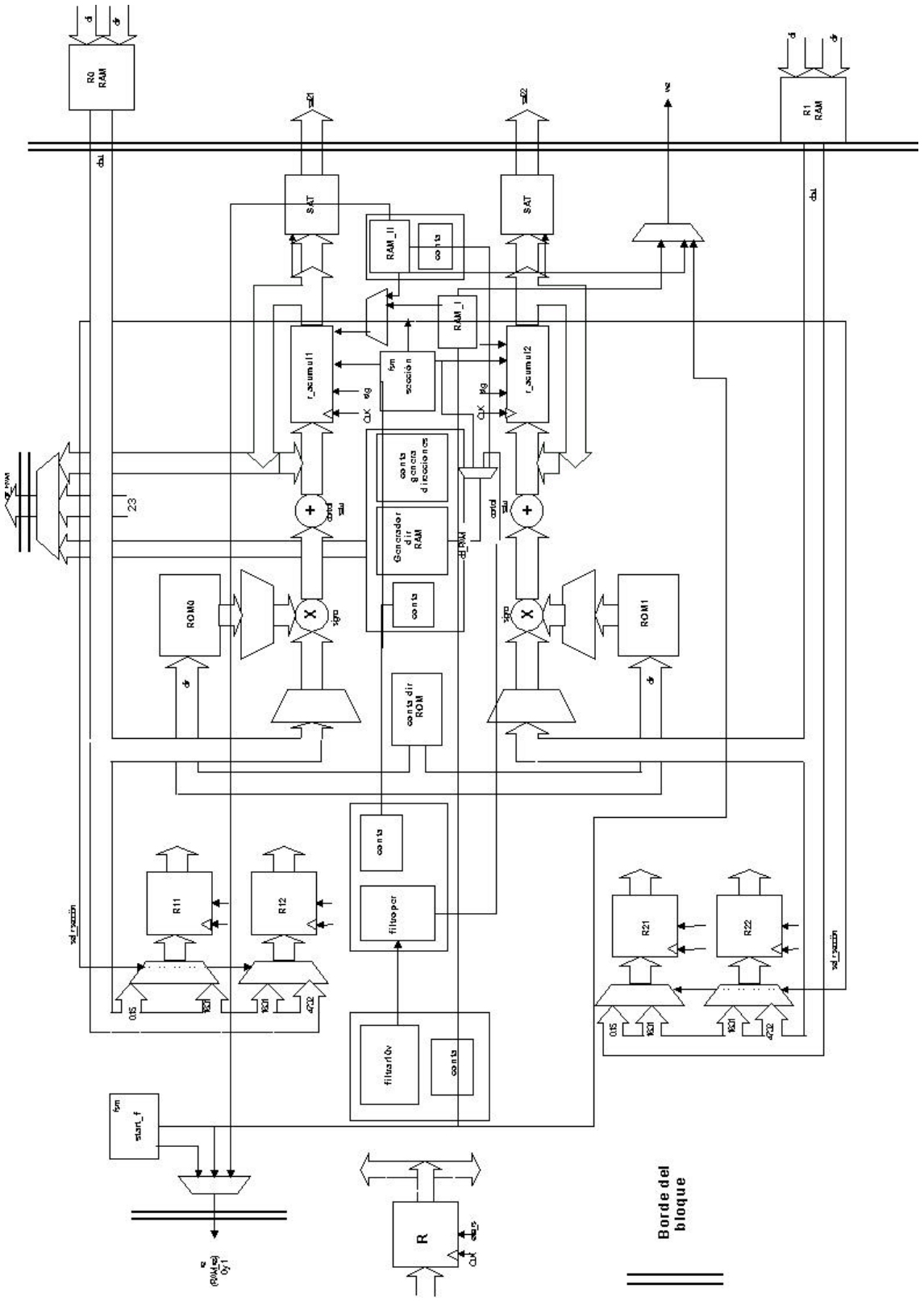


Figura 4.16 Esquema del banco de filtros

➲ Energ_envolv_bandas

Este bloque se encarga de hacer la detección de envolvente y calcular la energía promedio de ésta en cada banda. Está compuesto por varios bloques, un rectificador y un filtro paso de baja para la detección de envolvente y el bloque proener para el cálculo de la energía promedio.

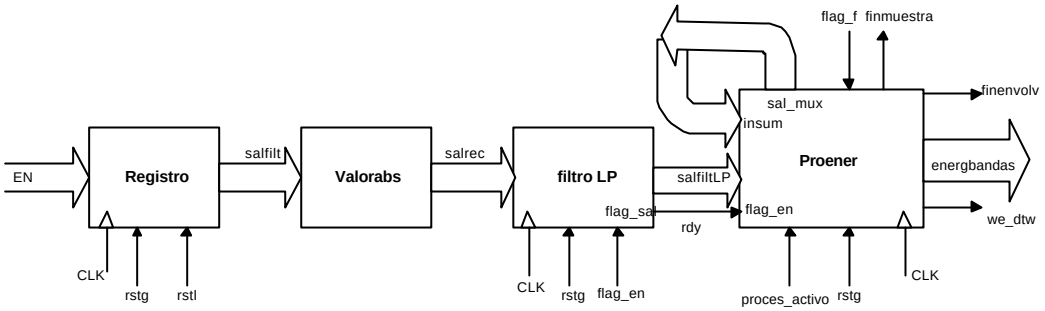


Figura 4.20 Esquema de energ_envolv_bandas

Lo primero que se hace en este bloque es capturar la entrada, para luego rectificarla y pasarla por un filtro LP, común para todas las bandas, se emplea una FIFO para ello. La salida del filtro pasa al promediador de energía que no es más que una serie de registros acumuladores en los que se irá acumulando las salidas de cada filtro una vez hecha la detección de envolvente. Cuando concluya una trama y antes de introducir el resultado en RAM lo dividimos por el número de muestras (256).

☑ *El filtro LP*

Este forma parte del detector de envolvente. Dado que la entrada nos va llegando multiplexada en el tiempo no hay necesidad de disponer de la circuitería del filtro LP para cada entrada, sino que empleando una FIFO nos lo podemos ahorrar. Esto junto con la elección del polo del filtro que deja la ecuación en diferencias como sigue:

Salida[n]=(1-0.0625)*salida[n-1]-0.0625*entrada[n]

El factor 0.05 calculado empíricamente lo redondeamos a 0.0625 para evitar el multiplicador. El problema de emplear una FIFO es que si se construye con bloques de RAM se estará desperdiciando mucha, por lo que sería más conveniente hacer uso de la RAM distribuida. En la figura siguiente podemos ver la implementación del filtro paso de baja. Se emplea un registro acumulador para hacer el cálculo antes de introducirlo en la FIFO.

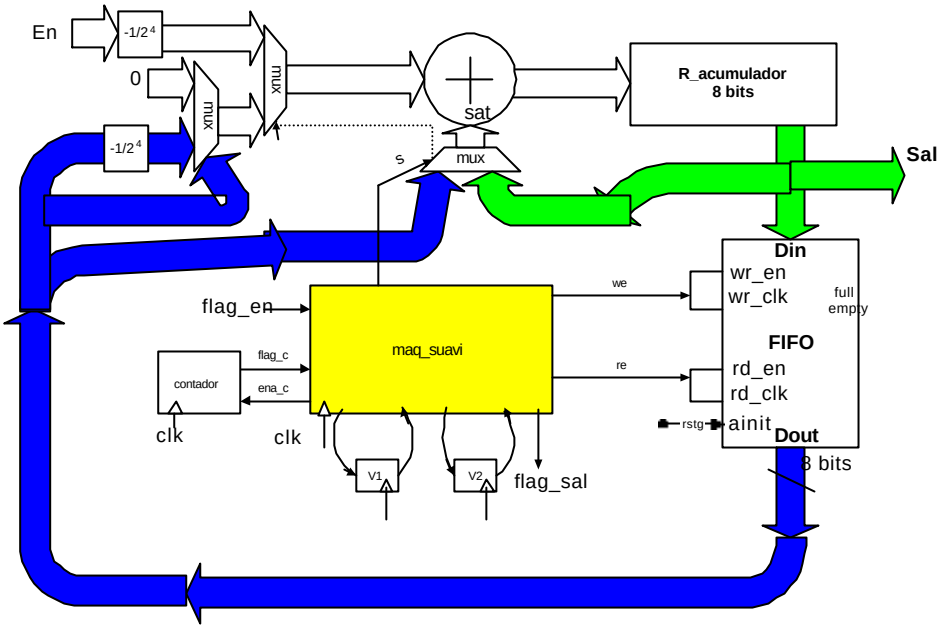


Figura 4.21 Esquema del filtro LP.

Al emplear la FIFO nos vemos obligamos a introducir tras el reset global veinte ceros en ella, dado que si no se han metido datos no se pueden sacar y esto obligaría a crear un modo de operación inicial distinto en el filtro, labor mucho más complicada que introducir esta fase de arranque.

La máquina que controla el filtrado paso de baja tiene algunas peculiaridades, como son el uso de vi1 y vi2, que se emplean para ahorrar estados. A continuación presentamos el diagrama de bolas de maq_suaviza.

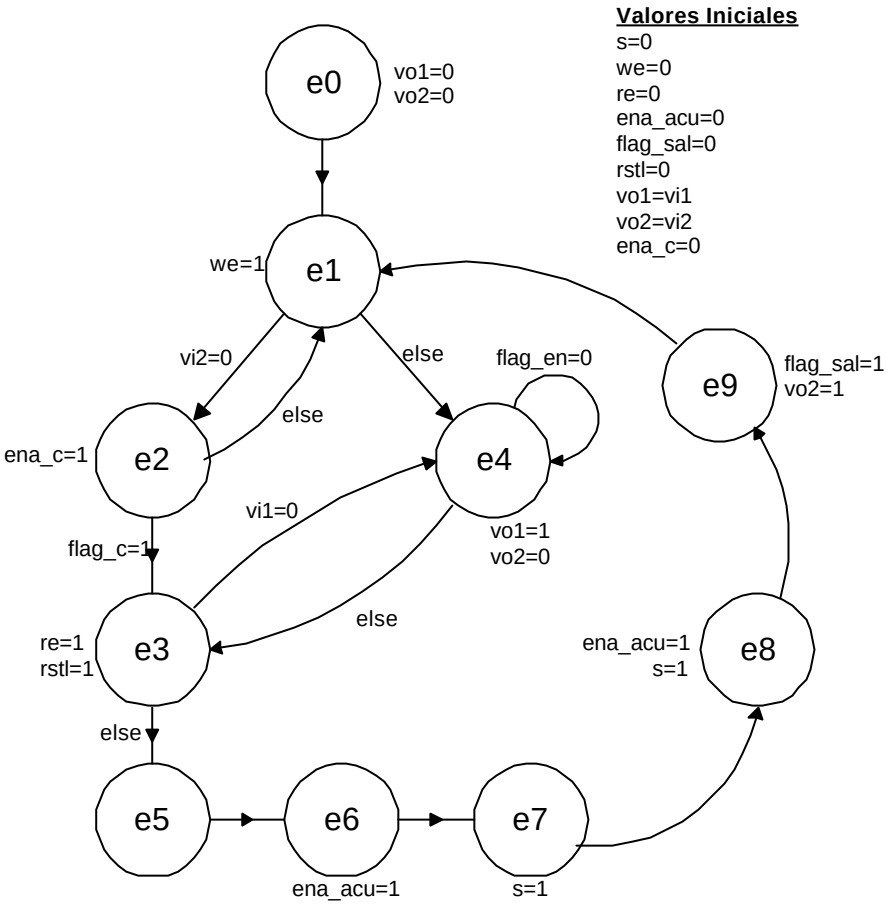


Figura 4.22 Diagrama de bolas de maq_suavi.

☑ *Proener*

Como ya se ha dicho este bloque consta de una serie de registros acumuladores, en los que va realizando la suma acumulativa de las salidas del filtro LP. Las salidas del bloque filtroLP, es un bus en el que se multiplexan en el tiempo los datos de la envolvente de las veinte bandas, para realizar la suma acumulativa hay que separar estas bandas para ello se empleó un contador de módulo veinte y un demultiplexor, de forma que cada vez que llega un nuevo dato se almacena en el registro correspondiente, el contador hace de seleccionador del demultiplexor, por último se incrementa el contador, para el siguiente dato.

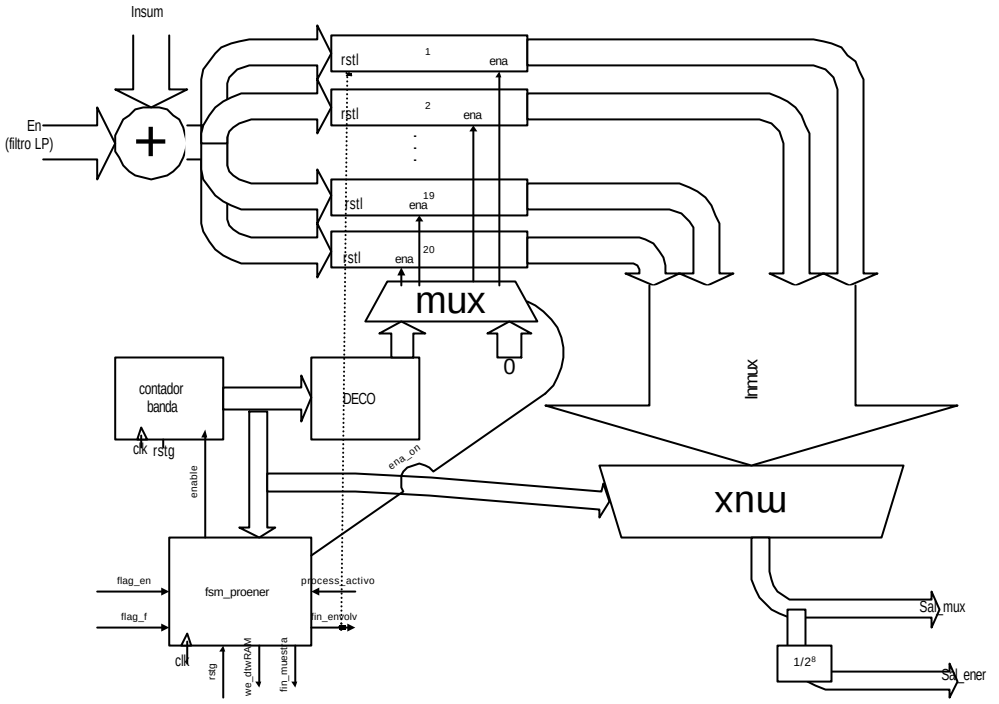


Figura 4.23. Esquema de Proener.

Una vez que concluye una trama se pasa a almacenar en la dtwRAM los valores de la entrada parametrizados, para ello solicita a almacena_en_RAM_DTW que escriba el dato que le pasa. Para enviarle los datos a almacena_en_RAM_DTW se emplea de nuevo el contador y el demultiplexor.

En la implementación de este bloque hemos sacado fuera de él la realimentación que hay de la salida del demultiplexor a la entrada al sumador, dado que esto mejora la implementación del sistema.

ALMACENA EN RAM DTW

Este bloque se encarga de almacenar en RAM las salidas del bloque parametrizador y de extraer de la RAM los valores de las entradas cuando el DTW se lo solicite, básicamente lo que hace él es generar la dirección, así como los habilitadores de escritura o lectura.

Las dimensiones de la dtwRAM son de 32*20 palabras de 8 bits cada una. La memoria se ha estructurado por en filtros, es decir, los valores de energía de cada banda se colocan juntos en la memoria, a una palabra se le permite una duración máxima de 32 tramas (1.024s). en primer lugar irán los 32 valores promedio de energía del primer filtro, en segundo lugar las 32 del segundo filtro y así sucesivamente. Esta organización de la memoria facilita el cálculo de la dirección de comienzo de la palabra ya que este cálculo se hace respecto de la primera banda, de forma que la trama inicial de las demás bandas se localiza fácilmente, sumándole una cantidad fija a esta dirección.

Almacena_en_RAM cuenta con dos contadores uno que le indica la dirección sobre la banda primera y otro que junto con un decodificador indica la banda. La forma de acceso a la memoria cuando se almacena en la dtwRAM y cuando se extrae de ella es la misma, se accede a los valores de energía de una trama, es decir, que el orden de acceso es “saltando” de banda a banda. Cuando está activa la parametrización se permite la escritura pero no la lectura de datos de la dtwRAM, y al revés cuando lo que está activo es el DTW.

En la dtwRAM se escriben los valores de energía promedio que se obtuvieron en proener, y se leen estos al completo por cada columna del DTW, es decir, que se leen 26 veces.

En este bloque encontramos dos máquinas de estado, fsm_almacena_RAM_dtw y fsm_almaRAM2. La segunda se encarga de indicarle a la primera si ha comenzado una palabra. La primera controla los contadores, la secuencia de acciones que lleva a cabo es la siguiente, espera a que se le indique que se quiere escribir o leer, espera a que se realice la operación, y actualiza los contadores (aquí aunque parece que no se ha puesto ningún seguro por si se le indique una lectura o escritura cuando no es posible, no es así ya que de esto se encargará control_central), si sólo hay que actualizar el que nos lleva de banda en banda no hay nada más que hacer, volvemos al estado inicial, si hay también que actualizar el de las tramas, (esto lo sabremos por el nuevo valor del contador de bandas, cuando acabamos de recorrer las bandas es cuando hay que cambiar de valor de energía promedio de trama) habrá que controlar cuando lo que estemos haciendo sea escribir y hayamos ya detectado el principio de una palabra, que no se sobrescriban valores de la entrada porque esta sea más larga de lo permitido (1.024s), si esto ocurre se indicará con la luz de error, hasta que comience el DTW, es decir, se indicará el error pero no se evitará que con las primeras 32 tramas de la entrada se lleve a cabo el DTW, que en general al no contener la palabra al completo será un resultado erróneo.

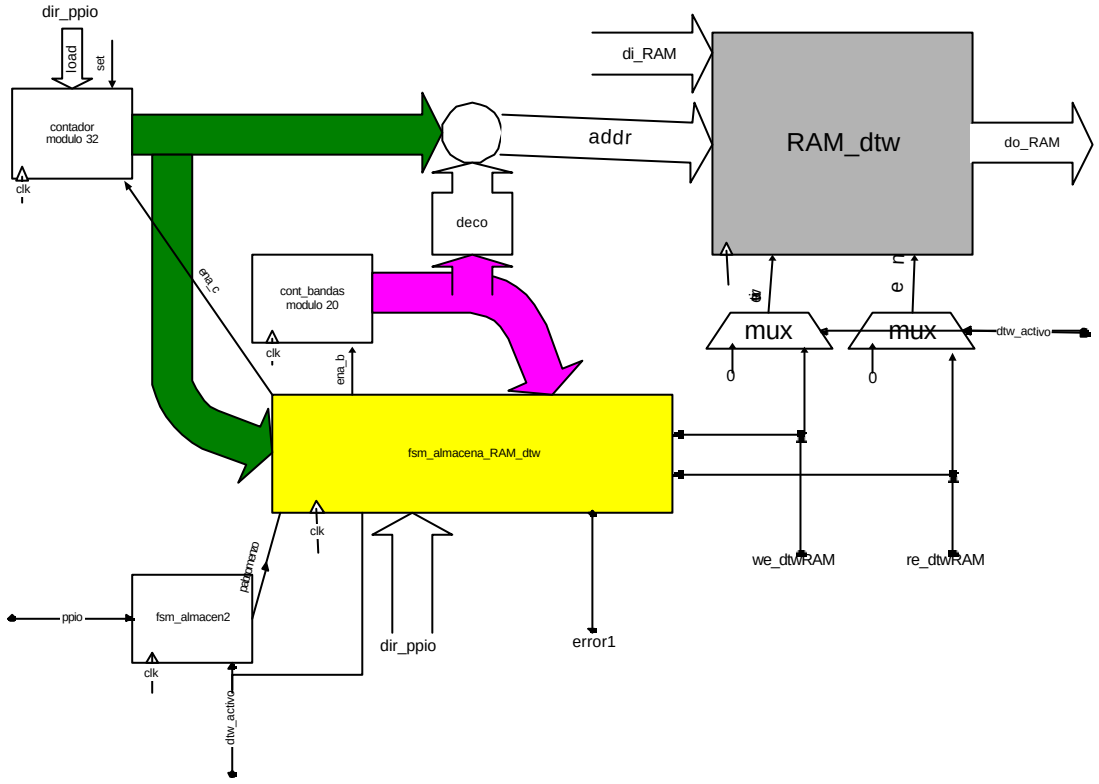


Figura 4.21 Esquema de almacena_RAM_dtw.

CONTROL_CENTRAL

Este bloque se ocupa de sincronizar y controlar las tres fases que atraviesa el circuito en su funcionamiento normal, la primera, que ya la hemos visto, es la parametrización de la entrada y la detección de actividad, la segunda es el reconocimiento de patrones para lo cual se emplea el DTW, que se verá en último lugar en este apartado, y la tercera parte del ciclo es la inicialización de la memoria RAM_res antes de volver a pasar a parametrización, esto se verá en el siguiente apartado.

El pensar en distintas fases en el funcionamiento del circuito puede "chocar" un poco, dado que una de las mayores ventaja de una implementación hardware es el hecho de poder disponer de verdad de distintos procesos en paralelo. Sin embargo, el hecho de que hablemos de fases no significa que no haya procesos paralelos en el circuito, como ya se ha estudiado los procesos de parametrización y detección de bordes son totalmente paralelos. Lo que ocurre en el caso del DTW y la parametrización es que por el propio sistema de reconocimiento necesitamos disponer de toda la entrada para poder dar comienzo al reconocimiento de patrones. En cualquier caso si dispusiésemos de más RAM se podría duplicar la dtwRAM de manera que mientras se realiza el DTW, se pueda proceder con la detección de una nueva entrada y su parametrización, las modificaciones necesarias en el circuito serían muy pocas, bastaría añadir la nueva dtwRAM, permitir que se fuera turnando, es decir, que cada vez se almacene la entrada en una dtwRAM diferente e incorporar algún seguro por si el DTW no ha concluido cuando lo hace la nueva entrada, también habría que capturar el factor de escalado y permitir que se volviese a evaluar para la nueva entrada. Además, dado que la duración del DTW es del orden de unos 8.5 ms por modelo, esto es, para cinco modelos 42.5 ms (a 20 MHz de frecuencia de reloj), y una palabra aislada tiene una duración media de 500ms, en general sería factible.

Control_central está formado por una máquina de estados, un par de biestables y circuitería lógica. Básicamente, este bloque indica a un bloque que puede proceder y espera a que este le indique el fin de su actividad para pasar a otra fase. Los biestables se emplean para el paso de la fase de parametrización y segmentación a la de reconocimiento de plantillas, son necesarios porque son varios los procesos que deben indicarnos su fin y pueden hacerlo en orden distinto.

RESET RAM_RES

Este bloque se encarga de poner a cero la RAM_res para que se pueda utilizar en el banco de filtros. Sería interesante inicializar la memoria en lugar de a cero al valor medio de ruido una vez se le ha hecho pasar por los filtros, este valor podría obtenerse de las últimas muestras del ruido tomadas, las que nos permitieron detectar el fin de palabra, el problema está en que habría que guardarlas, esto se podría hacer disponiendo de un buffer del tamaño de unas pocas de muestras filtradas, con las tres últimas salidas de cada filtro sería suficiente. En la implementación realizada se han empleado ceros para rellenar la memoria por lo que se sufrirá siempre de un pequeño transitorio.

Este bloque está formado por una máquina de estado y un contador, gracias al que se recorre toda la RAM, mientras que la máquina impone los habilitadores de lectura y escritura (el de lectura es necesario imponerlo también para escribir), el dato a escribir es fijo, ceros.

DINAMIC TIME WARPING

Este bloque no es más importante que los que ya se han visto, sin embargo, es éste método el que más caracteriza a nuestro sistema de reconocimiento dado que la parametrización siempre es necesaria, pero puede emplearse casi cualquier parametrización con cualquier método de reconocimiento de patrones.

Para abordar la implementación del DTW se ha seguido un razonamiento análogo al que se empleó para el diseño del banco de filtros, dado que una implementación que no serialice las operaciones no cabe en la FPGA que empleamos para el prototipo.

Las operaciones necesarias en el DTW son fácilmente estructurables en una matriz en la que el bloque combinacional se repite en el espacio, como se representa en la figura

4.22. Sin embargo, cada unidad es lo suficientemente cara en espacio como para que no nos sea posible la implementación de toda la matriz, o una columna.

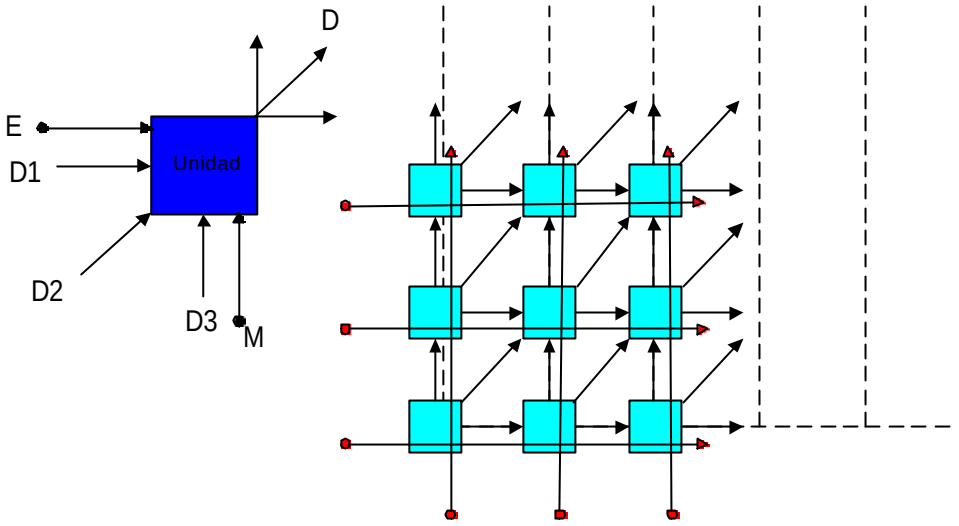


Figura 4.22 Esquema de transmisión de la información en el algoritmo *dinamic time warping.*

Razón por la que se ha implementado tan sólo una unidad, en el bloque llamado celda, y lo que se hace es serializar las operaciones en el tiempo, empleando registros para que se pidan los datos siguientes mientras que se opera con los actuales.

El DTW trabaja con cuatro memorias distintas; la ROM md, donde están guardadas las plantillas, que habremos construido mediante software; y las RAMs dtwRAM donde se guarda la entrada una vez parametrizada, y las dos memorias RAM_res que se emplean para guardar las distancias globales calculadas para la columna actual y la anterior.

La ROM md se ha estructurado por modelos, cada modelo se ha insertado en la memoria en el mismo orden en el que se va a extraer para no complicar la lógica de direccionamiento, este orden es por tramas, es decir, que nos encontramos los valores de energía promedio de las veinte bandas para una misma trama seguidos en la memoria. Las dimensiones de la ROM son 5*26*20 palabras de 8 bits cada una, donde el cinco viene de los cinco modelos que reconocemos, el 26 del tamaño estándar de estos modelos y el 20 de las bandas por trama.

La dtwRAM ya se vio en apartados anteriores y no se va a repetir aquí.

Las RAM_res aunque se comentaron en banco de filtros ya que éste las reutiliza para sus cálculos, su organización es aquí distinta. Cada palabra es de 64 bits y alberga a distancia global calculada para la unidad que representan. De las 64 palabras sólo se necesitan 32, como máximo, en el DTW, ya que esta es la longitud máxima que podrá tener una columna, ya que es la que se le ha permitido a la entrada.

Los archivos en los que podemos encontrar implementado este bloque se listan a continuación.

Archivo	Entity	Nivel
Topdtw2.vhd	Topdtw_s	TOP (nivel 1)
Dtw2.vhd	Dtw	2
Fsm_dtw.vhd	Fsm_dtw	3

Logcompara.vhd	Logcompara	3
Fsm_dtwr.vhd	Fsm_dtwr	3
Matrizdtwv5.vhd	Matrizdtw	2
Fsm_matrizdtw.vhd	Fsm_matrizdtw	3
Columna2.vhd	Columna_s	2
Fsm_columna_s2.vhd	Fsm_columna_s	3
Fsm_columna0.vhd	Fsm_columna0	3
Celda_s.vhd	Celda_s	2
Fsm_celda.vhd	Fsm_celda	3

Tabla 4.3 Lista de archivos del DTW.

➲ TOPDTW

El topdtw_s se ha empleado sólo para unir los bloques de nivel 2 que integran el DTW, además de resolver las señales de la RAM_res que tiene más de un driver en el DTW. Aquí acceden a la RAM_res tanto columna como dtw, la primera para guardar las nuevas direcciones calculadas y leer las de los predecesores y dtw para leer la distancia final entre la entrada y un modelo.

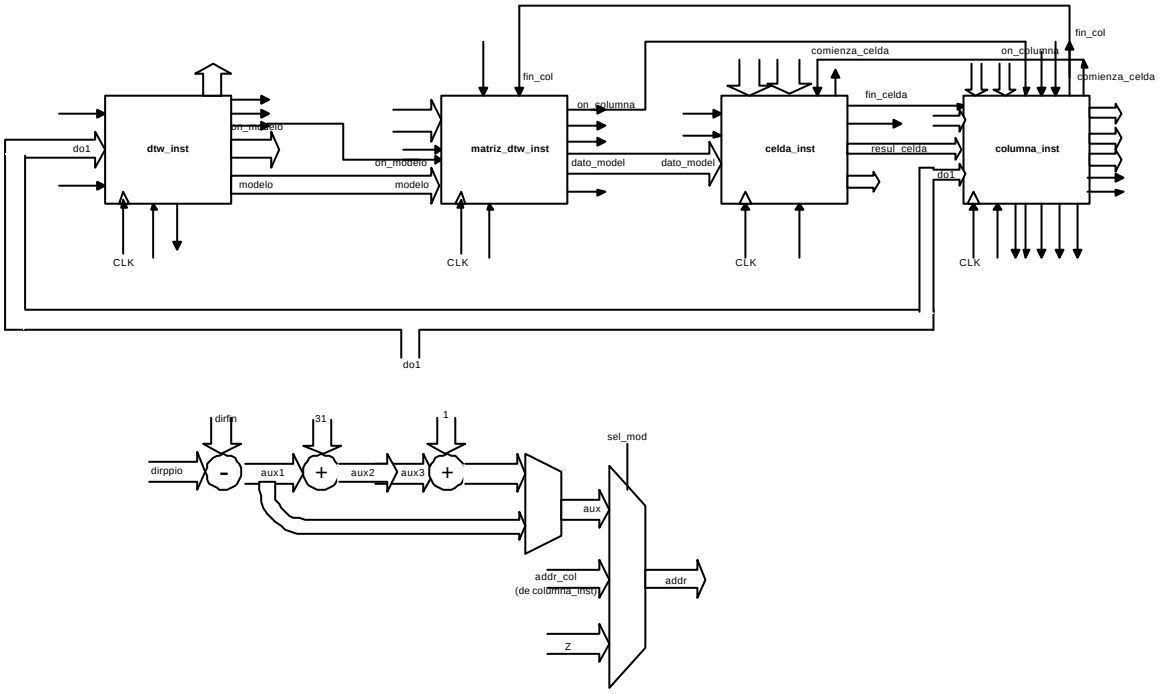


Figura 4.23 Esquema topdtw.

➲ DTW

Este bloque que hemos denominado igual que el algoritmo se va a encargar de que se realice este para cada uno de los modelos, para ello emplea un contador que unido a un decodificador sirve también par indexar los modelos en md. También, se ocupa de almacenar la distancia final entre la entrada cada modelo, así como de cuando finaliza el

dtw con todos los modelos de realizar la comparación de distancias, para lo cual emplea logcompara, y mantener el resultado del dtw hasta que se complete un nuevo reconocimiento.

Logcompara es un bloque combinacional constituido por una serie de comparadores. Este bloque va a devolver el modelo reconocido o una indicación de error que puede ser de dos tipos. En este bloque se dispone de un valor constante denominado umbral, dicho valor es la máxima distancia que vamos a permitir entre la entrada y un modelo para determinar que ha sido éste el comando reconocido. Este umbral se determina empíricamente, se ajusto a 150000. Si la distancia entre cualquier modelo y la entrada es superior a ese umbral, no se reconocerá ningún modelo y se indicará el error, en el prototipo éste se visualiza con el LED verde, con éste LED también se ha visualizado el caso en el que la entrada superaba la longitud máxima permitida, sin embargo, si bien, con el anterior error el sistema seguía adelante con éste error el sistema se queda bloqueado, lo hemos diseñado para que ocurra así y habrá que volver a descargar el diseño. Por otro lado, si algunas de las distancias son menores que éste umbral, y se da el caso de que hay más de dos iguales esto también se indica, en este caso no se ha indicado en cuáles ocurre, sino en cuántos ocurre, no sería difícil modificar logcompara para que nos indicase cuáles son iguales y para que sólo lo hiciese en el caso de que sean las de menos longitud, es decir, el comando reconocido.

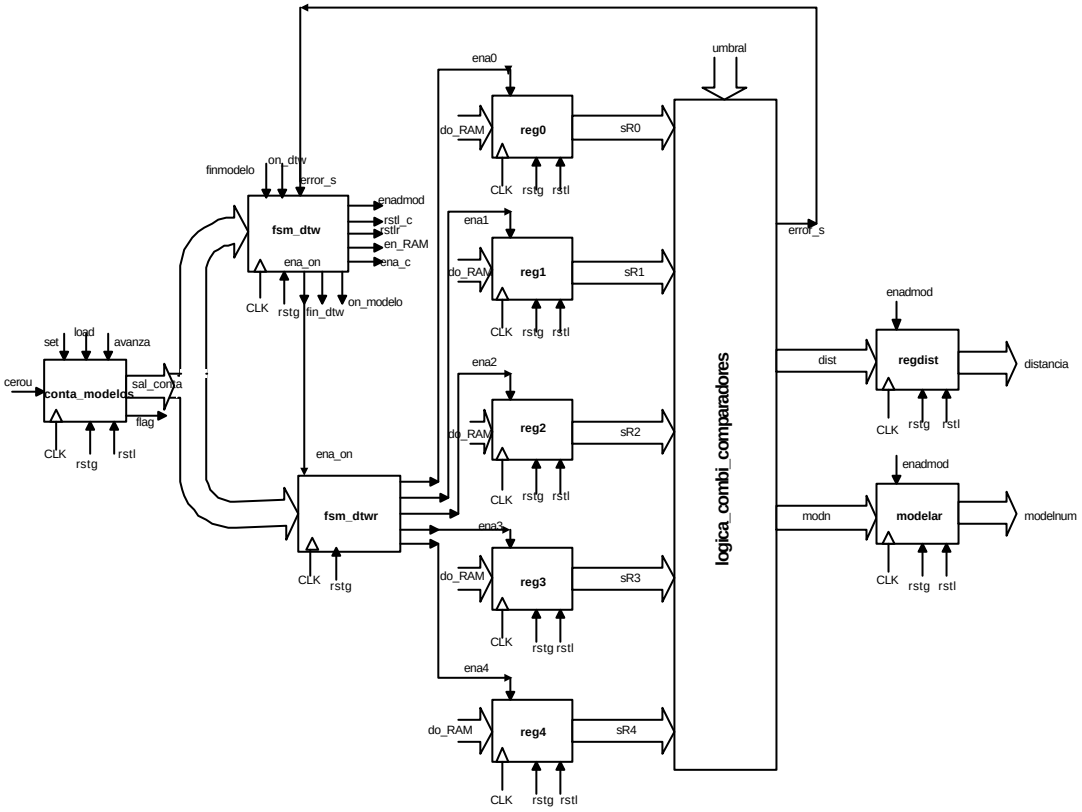


Figura 4.24 Esquema dtw.

➲ Matrizdtw

Este bloque se encarga de que se realicen las operaciones para el cálculo de la distancia entre un modelo, el que le indique el bloque dtw, y la entrada. Este bloque no es el que realiza las operaciones sino que llama a columna para que lo haga. Como se puede ver en la tabla anterior el bloque columna cuenta con dos máquinas de estado columna0 y columna, la primera se emplea para realizar las operaciones de la primera columna, esta

opción se prefirió a la de inicializar la columna anterior a algún valor por que de todas formas haría falta otra máquina de estados que lo hiciese, en caso de llevarse a cabo la inicialización habría de hacerse poniendo en todos los elementos el máximo valor posible salvo en el primero que habría que inicializarlo a cero.

Las dos memorias RAM_res se emplean para guardar las distancias de la columna actual y anterior. Estas memorias van alternando sus papeles, es decir, para una iteración de columna la actual es la denominada RAM0 y para la siguiente la actual es la denominada RAM1, esta alternancia la controla matrizdtw por medio de la señal col_actual, cuando vale cero la columna actual es RAM0 y cuando vale uno RAM1.

Matrizdtw también dispone de unas señales de control con las que activa a la máquina columna0 o columna. A la primera la llama cada vez que se inicia un modelo y lo hace con col_actual=0, para el resto de las columnas emplea la segunda máquina de estados, alternando el valor de col_actual. En la figura siguiente podemos ver el diagrama de bolas de fsm_matrizdtw en él se explica con detalle la operación de esta máquina.

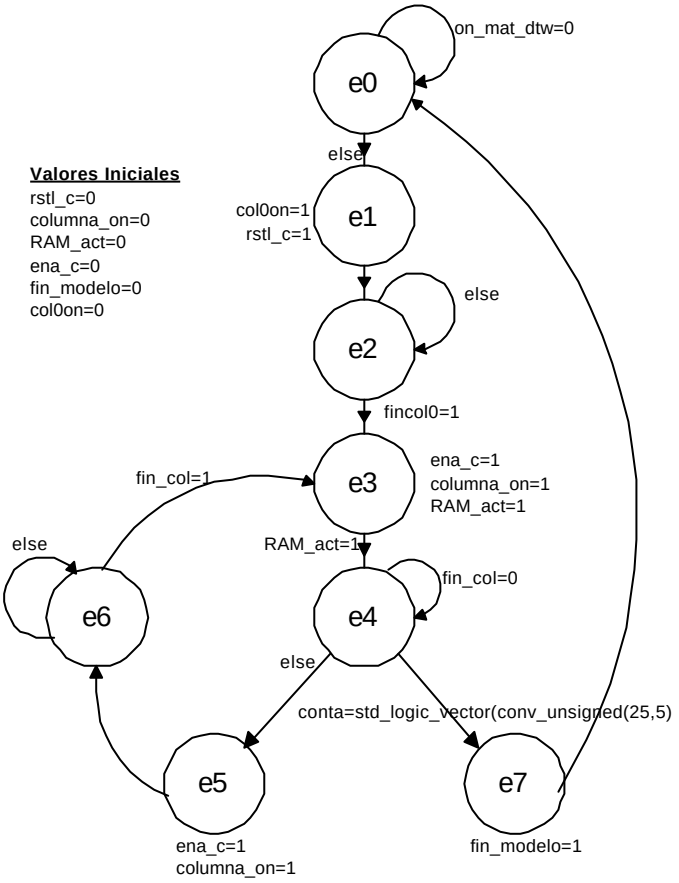


Figura 4.25 Diagrama de bolas de matrizdtw.

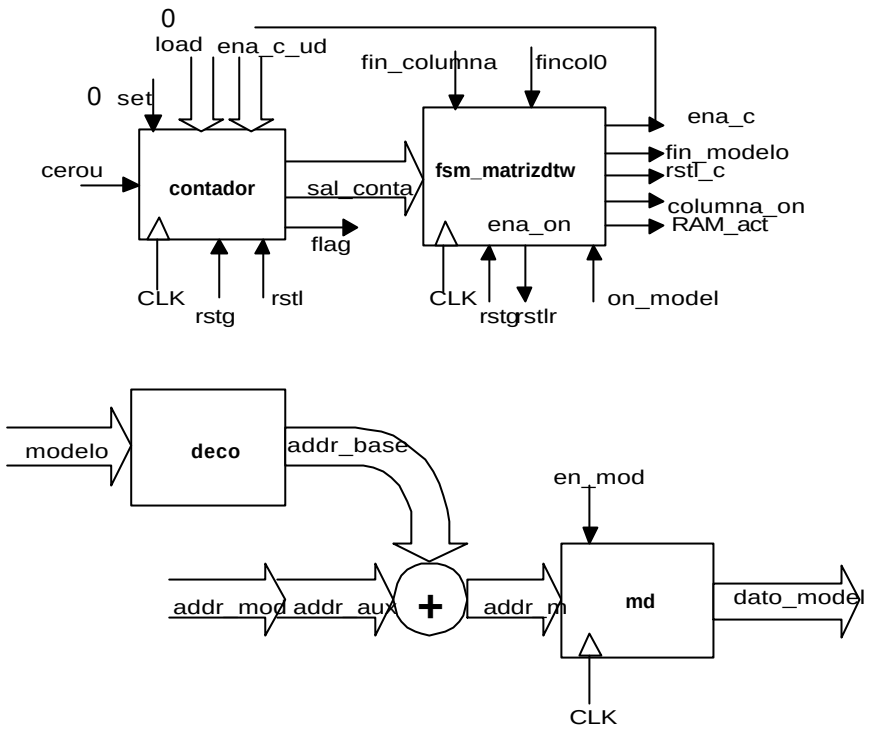


Figura 4.26 Esquema de matrizdtw.

El número de columnas que debe llevar a cabo es veintiséis que es el tamaño fijo al que hemos normalizado la longitud del modelo para ello se emplea un contador.

➲ Columna

Este bloque realiza las operaciones necesarias de para una columna de la matriz del DTW, para ello se apoya en celda que se encarga de calcular las distancias locales, se corresponde con la unidad que se denominó unidad en la figura 4.22.

Las funciones que debe desempeñar celda son el cálculo de las distancias globales de toda la columna actual, para ello tiene que calcular la distancia global y sumarle la distancia del mejor de sus predecesores. Las distancias globales de los predecesores están almacenadas en RAM_res. Como ya se vio en el capítulo 2 vamos a tener en cuenta sólo tres sucesores posibles.

Cuando comienza el DTW con un nuevo modelo, la columna anterior no existe, por lo que habrá que inicializarla de forma que el sistema se comporte como si no existiese o habrá que actuar de manera distinta cuando nos ocupemos de esta primera columna, esto segundo es lo que se ha hecho. Fsm_columna0 se encarga de calcular la primera columna. También es distinta la manera de proceder para la primera fila, de esto segundo nos ocuparemos dentro de fsm_columna0 y fsm_columna.

Dentro de columna encontramos el contador de filas, este contador se encarga de generar la dirección para acceder a RAM_res desde columna. Lo manejan tanto fsm_columna0 como fsm_columna.

Tres registros se emplean en columna para guardar las distancias globales de los predecesores que se habrán leído de la RAM_res. Fsm_columna hace uso de los tres registros, mientras que a fsm_columna0 le basta con uno, empleará el registro llamado R3. Para la primera columna sólo el único predecesor posible que hay es el que está en la misma columna en la posición anterior, salvo para el elemento de la primera fila que no

tiene predecesores. También para los demás elementos de la primera fila sólo hay un predecesor (salvo para el de la primera columna que ya se ha dicho que no tiene), el elemento de la columna anterior que está en su misma posición.

Antes de explicar el funcionamiento del bloque exponemos su diagrama.

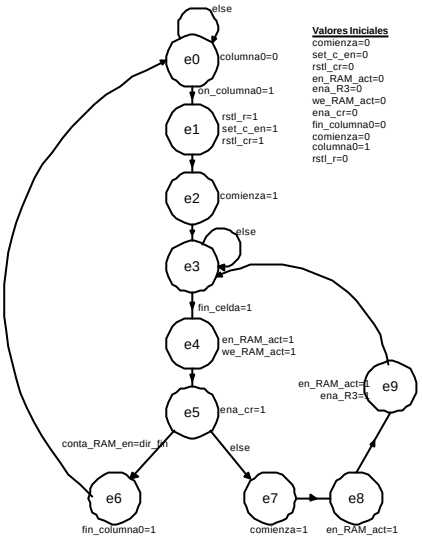


Figura 4.25 Diagrama de columna.

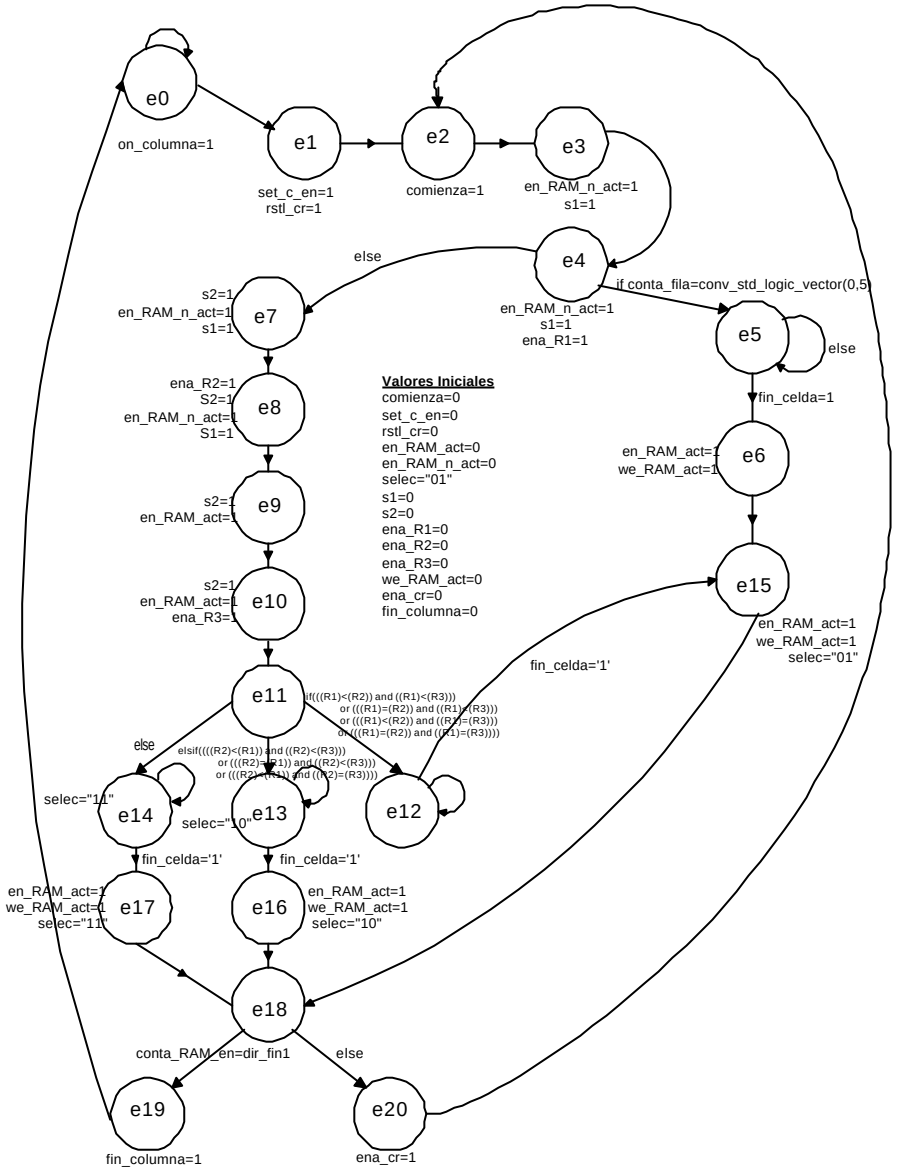


Figura 4.26 Diagramas de bolas de fsm_columna y fsm_columna0.

Cuando se le indica que puede proceder lo primero que hace es inicializar el contador de filas a cero e inicializar el contador de almacena_RAM_dtw al valor calculado de la dirección de principio de la palabra, de esta manera independizamos a almacena_RAM_dtw del funcionamiento del bloque dtw, en tanto a que no deberá saber que cada vez que finaliza una columna hay que inicializar el contador a la dirección de principio. A continuación ordenamos a celda que calcule la distancia local con el valor de la señal que le hemos provisto desde columna y con el valor del modelo que se le facilita desde matrizdtw. Observar que en cada columna sólo se recorren los valores de energía de las bandas de una trama, mientras que para la señal de entrada se recorren todos los valores, desde la dirección de principio hasta la dirección de fin de la palabra, ambas han sido calculadas y almacenadas en registros para su uso por el DTW. Mientras que celda realiza el cálculo de la distancia local, columna va extrayendo de RAM_res los valores de las distancias globales de los predecesores y los almacena en los tres registros antes mencionados, una vez hecho esto comprueba quién es el que tiene menor distancia global y lo elige como predecesor, es decir, es a éste al que hace pasar al sumador donde se le añade la distancia local antes de almacenarlo en RAM_res y pasar al siguiente elemento de la columna, a no ser que la

última distancia procesada se correspondiera con la de la última trama de la entrada. Para la columna y fila inicial la única diferencia es que sólo hay que sacar un elemento de RAM_res, su único predecesor posible, para el primer elemento de la primera columna no hay que sumar nada a la distancia local.

La alternancia entre columna actual y anterior se ha resuelto desde matrizdtw con la línea col_actual, que actúa sobre una serie de multiplexores que hacen el resto del trabajo para independizar a la circuitería de control de columna de quién es en ese momento la columna actual.

Fsm_columna0 y fsm_columna comparten los contadores y registros por ello es necesario emplear multiplexores para resolver los casos en los que hay más de un driver.

➲ Celda

Se ocupa del cálculo de las distancias locales. Para realizar éste cálculo necesita realizar las operaciones siguientes:

$$d = \sum_{k=1}^{20} \left(E\bmod_K - Es_K * Norma \right)^2$$

El diseño de este bloque ha sido nuevamente secuencial, se van extrayendo los valores de la memoria dtwRAM y md, se almacenan en sendos registros para ir pidiendo los siguientes mientras que operamos. Se trama de una secuencia de operaciones que se repiten veinte veces, para ello empleamos el contador que direcciona la RAM, cuando éste indique que se ha terminado una banda será cuando celda finalice su operación, el contador de este bloque se emplea para generar las direcciones para md que junto con la dirección base del modelo, que nos la da matrizdtw direccionan la memoria md. Las direcciones en la RAM no las impone este módulo él simplemente pide un dato y almacena_RAM_dtw se lo entrega. Los resultados intermedios se van almacenando en un registro acumulador, cuando llegue la indicación de que ha terminado una banda se pasan a un registro de salida y el acumulador se pone a cero, al pasar fsm_celda al reposo. Este archivo celda describe un "datapath" para las operaciones, en él es muy importante utilizar la representación adecuada e interpretarla adecuadamente, en concreto es importante fijarse en el bloque que eleva al cuadrado, debe tener en cuenta que está multiplicando dos números con signo ya que si le pasamos un unsigned y la entrada es negativa, cosa que puede ocurrir al realizar la diferencia, el producto se realizará de manera errónea, también hay que leer con cuidado la librería arith para saber que sobrecarga de las funciones tenemos definidas realmente.

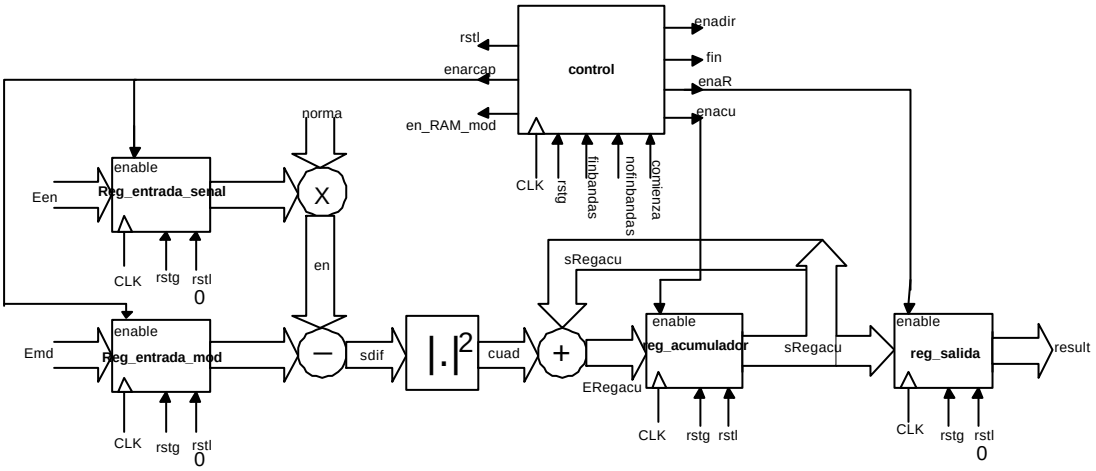


Figura 4.27 Esquema de celda.

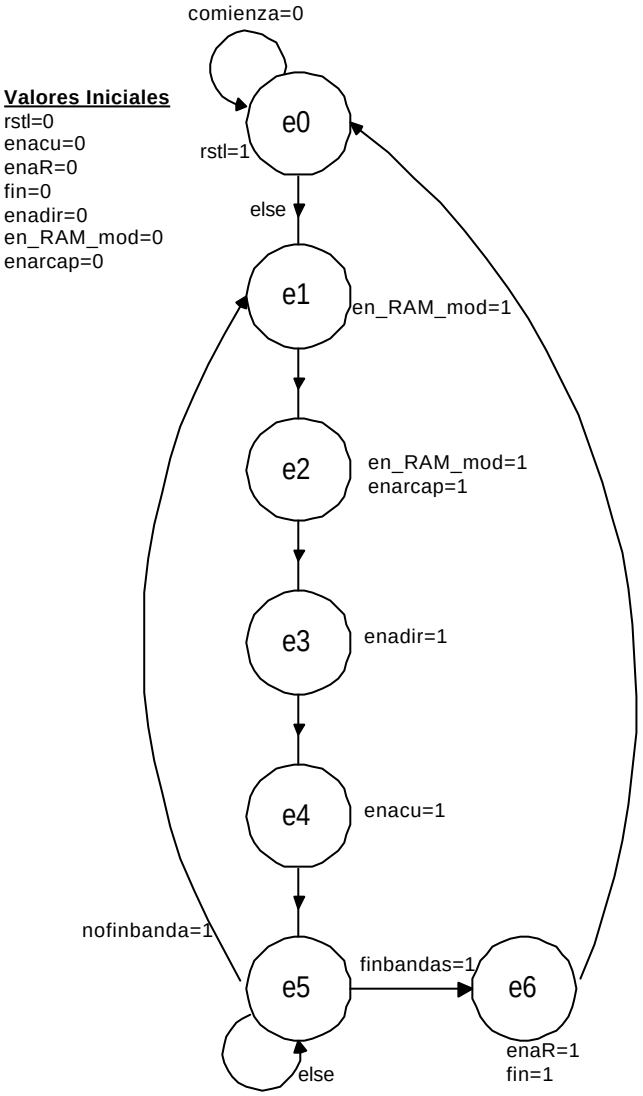


Figura 4.28 diagrama de bolas de celda.

D L L

La familia Virtex incorpora hasta ocho DLLs en las FPGAs con las que se consigue un retraso cero en el reloj y un desplazamiento del reloj muy bajo entre las salidas síncronas distribuidas a lo largo del chip. Estas DLLs se ubican en las cuatro esquinas de la FPGA. Sin partes de la FPGA dedicadas a éstas en exclusiva. Permiten disponer de hasta ocho diseños con relojes independientes.

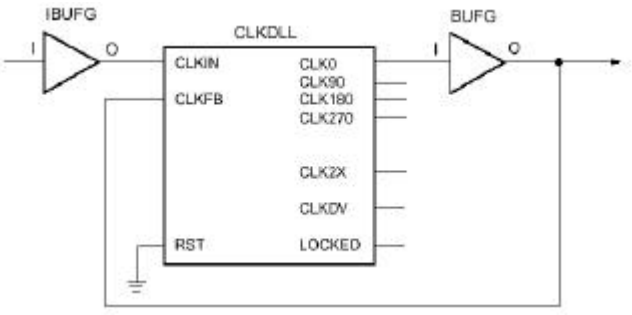


Figura 4.29 Esquema de BUFGDLL.

La DLL resulta indispensable para el funcionamiento de los diseños descargados sobre la FPGAs de la familia Virtex, ya que es tal el tamaño de la FPGA que sin un control en los desplazamientos en el reloj los diseños no resultan viables. Es por ello que la hemos ubicado en el TOP del diseño (iwrsser.vhd).

Otra utilidad de las DLLs es que permite multiplicar por 2 la frecuencia del reloj o dividirla por distintos factores, en la xapp132.pdf se describen otras aplicaciones así como su funcionamiento.

READBACK: EL ELEMENTO CAPTURE

El elemento capture aprovecha las nuevas características para el depurado de la familia de FPGAs Virtex. Este elemento permite imponer la condición de captura que deben cumplir las señales que se le indiquen del diseño para realizar la captura en la memoria de configuración del estado del chip. El sistema HADES-1 nos permite la lectura de esta memoria de configuración y presenta los valores de las señales internas de nuestro circuito.

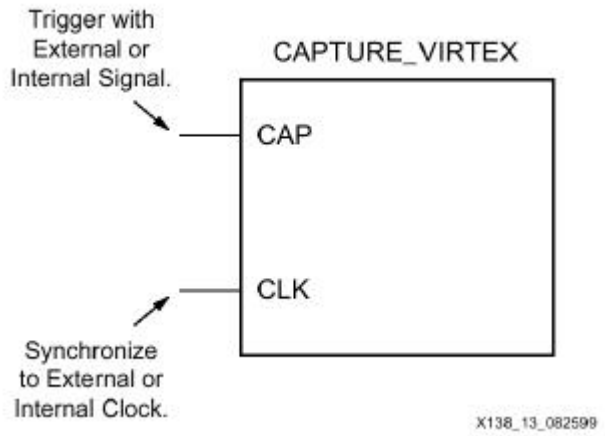


Figura 4.30 Elemento Capture de la librería de la Virtex.

¿Cómo permitir el “readback”?

Como el readback se realiza a través del interfaz SelecMAP después de la configuración de la FPGA, habrá que configurar este puesto para que continúe activo, para ello hay que establecerlo en el proceso de configuración de la FPGA, con la orden: bitgen –g persist:X8.

La captura e interpretación de los datos que nos envía la FPGA la realizamos a través del sistema HADES-1.

Capítulo

5

Conclusiones, comentarios y ampliaciones

ÍNDICE DEL CAPÍTULO

- Introducción
- Conclusiones
- Futuras líneas de desarrollo
- Agradecimientos

En este último capítulo con el que concluye la memoria se recogen algunos aspectos menos técnicos, con los que tratamos de hacer una crítica objetiva al proyecto que hemos realizado.

INTRODUCCIÓN

A lo largo de los cuatro capítulos anteriores hemos expuesto el punto de partida y la solución del proyecto. Para terminar la presente memoria, con este capítulo se pretende valorar la solución adoptada y comentar algunos aspectos que surgieron durante el proyecto pero que no hemos encontrado un lugar más adecuado que este para su exposición.

CONCLUSIONES

A continuación se van a exponer varios aspectos: por un lado, se estudian los resultados de la implementación, esto es, los recursos consumidos por el sistema, así como la velocidad de trabajo, frecuencia de reloj que admite el diseño,... ; por otro, se comentan algunos aspectos de las soluciones adoptadas, algunos de los cuales ya se han visto a lo largo de la memoria. Asimismo, se extraerán conclusiones sobre el proceso de diseño y de depurado, así como de la facilidad de testeabilidad del sistema, veremos qué métodos y sistemas se han empleado, cuáles son sus ventajas, inconvenientes y su alcance y utilidad.

➲ Prestaciones del sistema

Como cabe esperar de un sistema en sus etapas iniciales de desarrollo, las prestaciones son limitadas: reconoce palabras aisladas, el vocabulario es reducido, es dependiente del hablante (aunque en varias experiencias su comportamiento no pareció depender del hablante, lo que constituye una interesante línea de desarrollo) y su tasa de

error es relativamente elevada, además de variar con el comando (en algunos comandos es bastante buena).

Es muy importante incidir en que el grupo de comandos que se escoja es muy determinante y deben tener diferencias espectrales apreciables por el sistema para que éste sea capaz de diferenciarlos. Experimentalmente comprobamos que el sistema no se comporta bien con comandos en los que hay pausas entre sílabas (esta deficiencia afecta a la parte del sistema que se encarga de la detección de bordes).

Las distintas experiencias inducen a pensar que la relativamente elevada tasa de error que presenta el sistema tiene como causa el hecho de haber generado los modelos mediante software, ya que, aunque hemos comentado en el capítulo 4 la escasa diferencia entre uno y otro procesado, sí existen diferencias en el procesado no digital de la palabra (con esto nos referimos al sistema de adquisición de datos: micrófono, amplificador y filtro anti-aliasing).

➲ Recursos del sistema

El sistema completo consume casi todos los recursos de memoria de los que dispone la FPGA, así como casi la totalidad de los recursos que tiene para los bloques lógicos. En la siguiente tabla podemos ver el consumo de recursos. El número de recursos necesarios puede reducirse un poco empleando los comandos adecuados que mejoran en área la implementación. En todo caso es evidente que para llevar a cabo ampliaciones en este diseño en casi todos los casos necesitaremos de una FPGA superior a la Virtex 300. Las estadísticas tras la ubicación y rutado de la lógica nos indican que la máxima frecuencia de funcionamiento del sistema será de 25.587Mhz, siendo el máximo retraso de 8.852ns. A esta frecuencia el único problema lo encontraríamos en el banco de filtros, ya que a esta frecuencia no puede realizar los 20 filtrados antes de que llegue la siguiente muestra, razón por la cual se desdobló.

```
Design Summary:
   Number of errors:      0
   Number of warnings:   41
   Number of Slices:              3,070 out of  3,072   99%
   Number of Slices containing
      unrelated logic:              389 out of  3,070   12%
   Number of Slice Flip Flops:    1,606 out of  6,144   26%
   Total Number 4 input LUTs:     5,786 out of  6,144   94%
      Number used as LUTs:                    5,752
      Number used as a route-thru:               34
   Number of bonded IOBs:            17 out of    166   10%
   Number of Tbufs:                  38 out of  3,200    1%
   Number of Block RAMs:             15 out of     16   93%
   Number of GCLKs:                   1 out of      4   25%
   Number of GCLKIOBs:                1 out of      4   25%
   Number of DLLs:                    1 out of      4   25%
Total equivalent gate count for design:  315,435
Additional JTAG gate count for IOBs:  864
```

```
Device utilization summary:

   Number of External GCLKIOBs         1 out of 4
25%
   Number of External IOBs            17 out of 166
10%

   Number of BLOCKRAMs                15 out of 16
93%
   Number of SLICEs                 3070 out of 3072
99%
```

Tabla 5.1 Consumo de recursos del sistema completo.

Bloque	Slice(total:3072)	Bloques de RAM (total 16)	Número de puertas equivalentes	Frecuencia Máxima (Mhz)
Bloque que estudia el ruido y segmenta.	174	0	3145	76.086
Bloques que parametriza la entrada.	375	1(FIFO)	23185	35.841
Banco de filtros	651	6(Comparte 4 con el DTW)	111663	24.536
DTW	1525	11 (Comparte 4 con el banco de filtros)	177385	29.349

Tabla 5.2 Ocupación por bloques. La frecuencia es algo banco de filtros en la tabla es algo menor que la del sistema total porque para implementar esta versión del banco de filtro se hicieron unos ajustes que afectaron a su velocidad.

En la tabla anterior hemos separado el bloque que lleva a cabo la parametrización en el banco de filtros y el resto de los bloques (secuenciador y energ_envolv_bandas). Observamos que el banco de filtros requiere casi tantos recursos como el DTW y casi un tercio de la FPGA. Este banco de filtros contiene dos multiplicadores de 16 por 16, con el resultado en 32 bits. El DTW consume más de un tercio de la FPGA, éste cuenta con dos multiplicadores de 8 por 8, pero trabaja con datos de hasta 64 bits.

Por otro lado el bloque crítico en velocidad del sistema es el banco de filtros, que reduce la frecuencia de trabajo máxima a 24MHz. En realidad el diseño completo se queda en unos 25Mhz, y la implementación se realizará para un reloj de 20MHz. Sin embargo, este bloque nos obligó además a desdoblarlo ya que era imposible conseguir el filtrado antes de la llegada de la siguiente muestra, e incluir un buffer no tiene sentido dado el tamaño que tendría que tener, ya que tendría que esperar a que se pasase al DTW para vaciar el buffer.

La velocidad de respuesta del sistema la determinan el DTW y la detección de bordes. El banco de filtros no afecta a la velocidad de respuesta del sistema, ya que durante la parametrización el banco de filtros permanece siempre activo, y en unos 95μs ha realizado los veinte filtrados. Por otro lado, hasta 10 tramas, es decir, 320ms (como máximo) tras el final de una palabra no se detecta su fin, y por tanto no comenzará el DTW. Una columna del DTW, según el simulador, tarda unos 330μs, es decir, que como por comando tenemos que evaluar 26 columnas, por modelo serán 8.58ms, luego para los cinco modelos el tiempo total será 42.9ms. Si nos fijamos en el resultado anterior veremos que lo que determina el tiempo de respuesta es el detector de actividad, mientras que el

número de modelos no sea muy elevado. Estos tiempos de respuesta no son apreciables por el ser humano.

➲ El diseño

El tiempo empleado en la elaboración del proyecto se dividió, porcentual y aproximadamente, en un 15% en la búsqueda de información, clasificación y estudio de la documentación obtenida; un 35% en la definición del proyecto, elaboración de una solución y en la implementación software de la misma; otro 15% en la elaboración hardware del sistema; y el 35% restante en el depurado del sistema hardware, incluidas las realimentaciones y modificaciones de la implementación hardware.

Los problemas que nos encontramos a lo largo del proyecto fueron muchos y variados, y a continuación los comentaremos someramente, principalmente con el fin de prevenir a cualquiera que desee ahondar en el tema.

En la búsqueda de información se plantea el problema de clasificar los grandes volúmenes de información existentes sobre el tema. En efecto, cuando nos encontramos con un proyecto de reconocimiento del habla el problema no es la falta de información sino el desbordamiento de la misma, y por desgracia en muchos casos esta información es vaga e inútil, cuando no desconcertante y contradictoria. La búsqueda en internet debe estar bien orientada y es aconsejable conocer con antelación el tema, ya que a menudo encontramos soluciones sólo para determinados aspectos del reconocimiento. La búsqueda sobre sistemas comerciales sólo servirá para obtener algunas de sus características, pero en general no se nos describirá la solución adoptada. Una buena idea es acceder a universidades y/o contactar con profesionales de la enseñanza para que nos guíen. El punto de comienzo que recomendamos es la lectura del estado del arte en tecnologías del habla, en especial el capítulo 1 de la presente memoria, que trata sobre el reconocimiento. Además del problema anterior, la documentación para un proyecto como éste tiene otra dificultad, como es que la elaboración de un proyecto sobre reconocimiento del habla requiere de conocimientos multidisciplinares, y si además este proyecto se va a implementar sobre hardware hay que añadir esta nueva disciplina a la lista. Por último destacamos una última dificultad que nos parece relevante: según qué características decidamos estudiar de la voz habrá que tener en cuenta que la documentación que utilicemos sea, en nuestro caso, para el español, ya que si por ejemplo, usamos los modelos ocultos de Markov (HMM) las probabilidades que empleemos deberán estar referidas a nuestro idioma.

La definición de las características del sistema que se ha elaborado viene fijada por la propia solución adoptada. Cuando nos decantamos por esta solución valoramos el hecho de que es un método no experimental, por tanto bien documentado (de hecho, se usó mucho para reconocedores entre los setenta y los ochenta). Además disponíamos de la información necesaria para implementarlo y no como en el caso de HMM, donde encontrar las probabilidades no es fácil. De igual forma, conceptualmente tanto la parametrización como el reconocimiento de patrones empleados son simples, lo que facilitará la búsqueda de fallos (especialmente en la parametrización se prefirió el uso de un banco de filtros para extraer las características espectrales directamente al uso de FFT o al empleo de otros parámetros como LPC o Cepstrum).

En la implementación software del proyecto nos encontramos en primer lugar con la elección del entorno, y una vez tomada esta decisión, el problema se centró en la poca eficiencia del mismo, unido a que no se depuró el software por no ser este el objetivo del proyecto.

En la elaboración del sistema hardware se encontraron dificultades con el espacio disponible en la FPGA, ya que como hemos comentado en principio se abordó una implementación paralelo que resultó inviable.

En el depurado del sistema tuvimos problemas con el alcance de las herramientas de las que disponíamos, lo que nos obligó a fabricar varios sistemas de prueba para comprobar el correcto funcionamiento bloque a bloque, así como a emplear una simulación a alto nivel con modelsim.

Un problema con el que nos encontramos a la hora de realizar la implementación hardware del sistema, a la hora de la síntesis e implementación, es que un proyecto de estas dimensiones requiere de un computador de gran capacidad, ya que, a modo de ejemplo, en un Pentium a 350MHZ con 32 MB RAM, el proyecto completo requiere 7 horas de procesado, mientras que en un Pentium a 550MHz con 64MB de RAM "tan sólo" necesita 2.5 horas de procesado. Y cuando se trata de depurar el diseño, el simulador a nivel de funcionalidad lógica, a nivel de puertas o teniendo en cuenta la implementación física, son totalmente inviables para ver largos procesados, como los que requiere un sistema de reconocimiento del habla, aún en el caso de que permitan leer de un fichero la entrada (el empleo de ficheros para describir el entorno tampoco es algo muy depurado por las herramientas y en caso de Xilinx no se soporta), requieren de días para la obtención de resultados, y además la simplificación de señales dificulta su estudio (una solución es sacar la señal del bloque o registrarla, pero esto obliga a tener que hacer modificaciones en el diseño, y además podemos encontrarnos con que no tenemos patillas suficientes ni aún en el encapsulado mayor). Una alternativa a todo esto es el sistema HADES, que por desgracia no se encontraba totalmente operativo, además tiene el problema de que sólo permite, al menos por el momento, una condición de trigger y una captura, y para cada captura distinta hay que volver a procesar todo el proyecto. También requeriría de algún mecanismo para controlar la entrada de manera que podamos realizar un estudio real de lo que deberíamos obtener y lo que obtenemos. En cualquier caso siempre será preferible, ya que el procesado del circuito nos da resultados en tiempo real. Otra alternativa para el depurado es emplear simuladores de alto nivel, tipo modelsim, que presentan tres ventajas fundamentales: obtienen resultados en muy poco tiempo; pueden describirse entornos y leer las entradas de ficheros, así como escribirnos en ellos las salidas; y mantiene los nombres de las señales. Como desventaja a estos simuladores está el hecho de que su utilidad se reduce a búsqueda de fallos en la lógica, es decir, debemos estar seguros que el sistema sea implementable ya que el simulador no va a detectar este fallo, dado que simplemente simula el VHDL como si fuera un lenguaje de programación.

FUTURAS LÍNEAS DE DESARROLLO

Bajo este título trataremos también diversos puntos: en primer lugar, qué aspectos son susceptibles de mejora; y en segundo lugar, qué se podría añadir para mejorar las prestaciones o para conseguir otras metas (y cuáles serían estas).

➲ Mejoras y variaciones del sistema

Una primera línea de desarrollo serían cambios en el parametrizado y en el reconocimiento de patrones. Proponemos como cambios en el parametrizado sustituir la estimación de la energía por bandas en función del tiempo por parámetros LPC o Cepstrum, o también estimar la energía por bandas en función del tiempo empleando la FFT. También sería interesante un estudio más profundo del tamaño óptimo de las tramas y el número de parámetros por trama. Estas variaciones que se han propuesto no afectan ni

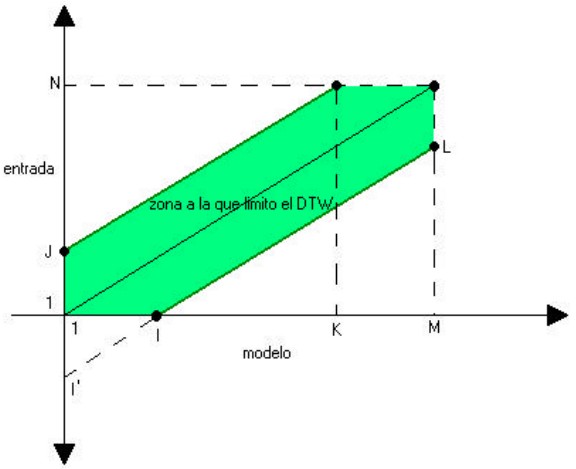
a la detección de actividad ni al reconocimiento de patrones, tan sólo al bloque del banco de filtros y al que calcula la energía promedio de las bandas.

Las modificaciones en el reconocimiento de patrones que proponemos se pueden basar en: utilizar otras restricciones en el DTW, como por ejemplo restricciones en la pendiente, lo que mejoraría bastante el tiempo de respuesta del sistema; o en utilizar métodos distintos como sería el uso de cadenas ocultas de Markov. Esta segunda propuesta requerirá de un estudio previo de viabilidad, en tanto a necesidad de recursos, y en esta línea proponemos también el uso de redes neuronales.

Otras partes del sistema que merecen un mayor análisis son la detección de bordes y el tratamiento del ruido. El sistema implementado no se comporta bien con palabras en las que hay una pausa apreciable entre sílabas, por lo que buscar e implementar métodos más finos de detectar los bordes la palabra sería otra posible línea de desarrollo. En cuanto al tratamiento del ruido, en el sistema se ha resuelto con un análisis del ruido tras el arranque, dado que la sensibilidad del micrófono es pequeña. Esto no ocasionó problemas, pero sería aconsejable realizar un estudio del ruido continuo, para lo cual debería dotarse al sistema de un segundo micrófono. Para aumentar las prestaciones del sistema en cuanto a inmunidad al ruido otras técnicas que se pueden estudiar son un filtrado en banda del ruido y preenfatizar la entrada.

En el sistema falta también la elaboración de los modelos en el propio circuito, aunque en un principio pensamos que no sería posible introducir este cálculo en la V300 y que se necesitaría de una FPGA mayor, un estudio más detenido del problema nos hace pensar que sería posible que cupiese, aunque habría que ajustar mucho el diseño. Para construir los modelos, podríamos guardar sólo dos muestras a la par. El procedimiento que habría que seguir para crear los modelos, sería parametrizar la primera pronunciación tal y como se ha visto, y a continuación estandarizar su longitud, para lo cual necesitaríamos calcular la pendiente dada por la longitud de la pronunciación dividida por la longitud estándar (este cociente se haría iterando de igual manera a como se calculó el factor de escalado). Para estandarizar la longitud lo que hacemos es una proyección lineal de la entrada, es decir, que la estiramos o acortamos linealmente. Lo que hay que hacer es calcular los valores de cada trama de la 0 a la 25, para ello, si **y** es el índice de la entrada (indica la trama de la entrada), y **x** es el índice del modelo estandarizado en longitud, sabremos que trama de la entrada se lee por la fórmula: $y = pendiente \cdot x$. Según estandarizamos la entrada se almacena en md, que ahora será una memoria RAM. Basta tener estandarizada la primera pronunciación ya que las demás se estandarizan al realizar el DTW con el modelo. Concluido este paso, se realiza el DTW con la siguiente pronunciación guardando la matriz de predecesores(en una nueva memoria que habrá que crear), una vez que termina se realizaría el promedio siguiendo el camino y así con cuantas pronunciaciones queramos usar para cada modelo, repitiéndose este proceso en cada modelo.

También podemos contemplar una mejora al DTW. Se trata de imponer unas restricciones a la pendiente a la hora de alinear. Físicamente tiene sentido ya que se eliminan casos como asociar todos los frames de la entrada con el primero del modelo (salvo el último que se asocia por restricción siempre al último). Llevar a cabo esta mejora no es muy complicado. Dado que realizamos el



DTW elemento a elemento, siempre podemos saber en que columna y fila nos encontramos. Tal y como se ha implementado se realiza columna a columna de manera que cuando acabamos una se comienza otra. Por ello es fácil ver que sabiendo en qué columna estoy partiré de una fila (mínimo) y llegaré hasta una fila (máximo) marcadas por la limitación en la pendiente. Podemos ver esto en la figura anterior.

N y M son conocidos antes de realizar el DTW: M es fijo y N dependerá de la entrada. Esta mejora tan sólo necesita de unos cálculos previos al DTW y tiene la ventaja de que estos cálculos valdrán para todas las alineaciones de esta palabra con la entrada. Según la limitación de pendiente que usemos tendremos un compromiso entre rapidez y prestaciones, pero estas últimas no se verán seriamente afectadas a no ser que la restricción en la pendiente sea muy fuerte. De forma práctica, debemos calcular J, I, K y L (ver gráfica anterior), tras lo cual se le pasan al DTW de forma que se tienen en cuenta para saber cuándo cambiar de columna y por qué fila comenzar. Los cálculos de J, I, K y L en función de M, N y h (restricción en pendiente) son los siguientes:

- J: $J = h$
- I: $I = h$
- K: $\begin{cases} j - 1 = N \\ \frac{N}{M} K + J = N \Rightarrow K = (N - h)\frac{M}{N} \end{cases}$
- L: $\begin{cases} i - 1 = M \\ \frac{N}{M} M + \frac{N}{M} I = L \Rightarrow L = N - \frac{N}{M} I \end{cases}$

El único inconveniente es el cálculo de M/N ya que aunque podemos tomar M potencia de 2, N no es de nuestra elección, por lo que habrá que hacer el cálculo iterando.

Como última ampliación se propone realizar síntesis de voz. Esto sería de utilidad para mejorar el interfaz del sistema de reconocimiento, empleándose para verificar la construcción de los modelos. Se puede aprovechar la estructura del banco de filtros para la síntesis de voz.

➲ Aplicaciones

La aplicación más clara del sistema es como reconocedor telefónico, es decir, para selección de opciones a través de la línea telefónica. El uso de nuestro sistema sería viable dado que el ancho de banda telefónico es el que empleamos, aunque habría que dotarlo de independencia del hablante.

Otra aplicación posible es como sistema de marcación por voz, para lo que habría que aumentar el número de comandos.

En otro ámbito, sería aplicable al control de electrodomésticos de pocas funciones, por ejemplo, microondas, lavadora... En general en cualquier interfaz sencillo puede ser utilizado, aunque habrá que escoger con cuidado los comandos.

A G R A D E C I M I E N T O S

Son muchas las personas y algunas las empresas a las que hay que agradecer algún aporte a este proyecto. En primer lugar, un agradecimiento muy especial al tutor de este proyecto D. Miguel Ángel Aguirre Echanove, por su inestimable ayuda a lo largo del diseño, especialmente en la parte de la FPGA, así como por guiarme aquellas veces que parecía no haber solución.

En otro orden de cosas, a la empresa Maxim, por su ayuda en la obtención de componentes para el circuito, así como por la diligencia en hacerlo.

Igualmente quiero agradecer su colaboración a D. Jon Tombs, que me orientó de manera muy acertada en la investigación teórica del método de reconocimiento, siendo sus indicaciones las que finalmente condujeron a información productiva.

Agradecer también a D. Jorge Bohórquez del Departamento de Ingeniería Eléctrica y Electrónica de la Universidad de los Andes por facilitarme la documentación que necesitaba.

Me gustaría dar las gracias a los profesores del Área de Teoría de la Señal y Comunicaciones de la Escuela Superior de Ingenieros D. José Ignacio Acha Catalina y D. Rubén Martín Clemente por su aportación en aspectos concretos relativos al diseño teórico del proyecto.

Quiero agradecer muy especialmente su ayuda a José Manuel Cordero García, tanto por su aporte de medios para la síntesis/implementación del proyecto, como por su colaboración, tanto en los muchos esquemas que se encuentran en esta memoria como su aporte de conocimientos, y por su apoyo en los momentos difíciles.

Por último, y sin duda no menos, muchas gracias a mi familia por su paciencia y comprensión durante todo el tiempo de desarrollo del proyecto.

Bibliografía

1. http://woody.us.es/~aguirre/hades.html

2. Circuitos integrados lineales sus aplicaciones. M.Torres Portero. Paraninfo, Madrid 1989.

3. Artículo de Radiorama – Marzo 1991: Alimentación para micrófonos electret. José Luis Cardona,

4.Gran enciclopedia de la electrónica tomo 1. Ediciones Nueva Lente, Madrid 1984.
5.Transductores y acondicionamiento de señal, Ramón Pallás, Marcombo 1989.

6. Datasheet MAX270.

7. Datasheet MAX152.

8. Reconocimiento automático del habla. F.Casacuberta y E.Vidal. Marcombo, 1987.

9. Speech Analysis. Tony Robinson, 1998.

10. Discrete-Time Processing of Speech Signals. J.R.Deller, J.G.Proakis, J.H.L.Hansen. Maxwell Macmillan International, 1993.

11. Application of digital signal processing. A.V.Oppenheim. Prentice-Hall, 1987.

12. Texto del grupo de noticias: comp.speech. De Douglas G. Danforth, NASA Ames Research Center. Subject: Very simple speech recognition Alg. wanted. Message-ID: <1992Nov12.180625.13886@riacs.edu>.

13. Message-Id: <199511291433.PAA29324@suncrie.paris.ensam.fr>. Jean-Pierre. Lereboullet. Subject: VoiceRecognitionProcessorsDATABASE29/11/95.

14. http://www.sensoryinc.com/

15. Survey of the State of the Art in Human Language technology. Ronald A. Cole y otros,1995.

16. Speech Recognition by Dynamic Time Warping. http://www.dcs.shef.ac.uk/

17. http://www.acoustics.hut.fi/~slemmett/dippa/chap3.html. Phonetics and Theory of Speech Production.

18. Tratamiento digital de señales.Principios, algoritmos y aplicaciones. J.G.Proakis, D.G.Manolakis, Prentice Hall, 3ª Edición.

19. Speech analysis and synthesis using a glottal excites AR model with dtw-based glottal determination. G. Cohen and D. Malah.

20. Tecnologías del Habla.

http://www.gtc.cps.unizar.es/~eduardo/investigacion/voz/intro.html

21. L. Hernández Gómez, F. J. Caminero Gil, UNIVERSIDAD POLITÉCNICA DE MADRID. C. de la Torre Munilla, L. Villarrubia Grande, TELEFÓNICA INVESTIGACIÓN Y DESARROLLO. Estado del arte en Tecnología del Habla. http://www.tid.es/presencia/publicaciones/comsid/esp/articulos/vol52/artic1/1.html

22.Diseño de circuitos integrados de aplicación específica (ASIC).Jean-Pierre Deschamps.Paraninfo 1994.

23.Modern VLSI design- systems on silicom. Wayne Wolf.Second Edition,1998 Prentice-Hall.

24.Digital Signal Processing: A practical Approach. Emmanuel C.Ifeachor. Barrie W.Jervis. Ed: Addison Wesley, 1993.

25. Matlab Signal Processing Toolbox. User Guide Versión 4.1

26. M.A.Aguirre, J. Tombs, A. Torralba, L. Franquelo. *HADES –1 : A Rapid Prototyping Environment based on Advanced FPGA´s*. SCIS´01, Nov. 2001, Oporto (Portugal).

27. Datasheet Virtex FPGAs.

28. Notas de aplicación de Virtex: Xapp132 (v.2.3). Using the Virtex Delay Locked Loop.

29. Notas de aplicación de Virtex: Xapp138. (v.2.3). Virtex FPGA series configurations and readback.

30. Notas de aplicación de Virtex: Xapp215. (v.1.0). Design Tips for HDL Implementation of Arithmetic Functions.

31. VHDL. Douglas L. Perry. McGrawHill. 2ª Edición.