Universidad del País Vasco
Euskal Herriko Unibertsitatea

ZTF-FCT
Zientzia eta Teknologia Fakultatea
Facultad de Ciencia y Tecnología

Trabajo Fin de Grado
Grado en Ingeniería electrónica

Procesado digital de señales mediante circuitos integrados reconfigurables.

Filtros de respuesta impulso finita (FIR)

Autora:
Itsaso Artetxe Querejeta
Directora:
Inés del Campo Hagelstrom

Leioa, 24 de junio de 2014

ÍNDICE

INTRODUCCIÓN Y OBJETIVOS

El procesado digital de señales es una técnica ampliamente usada en distintas disciplinas entre las que se encuentran las telecomunicaciones, el control de procesos, la exploración espacial y la medicina, entre otras. Hoy en día, esta afirmación adquiere incluso más relevancia con la televisión digital, los sistemas de información y el entretenimiento multimedia.

Los continuos avances en las tecnologías de integración de circuitos electrónicos como la microelectrónica han permitido reemplazar ciertos circuitos analógicos por circuitos digitales, hasta el punto de poder integrar sistemas en un único chip, SoC (System on Chip). El procesado digital de señales se concentra en el análisis y en el procesamiento de las señales representadas en forma digital, es decir, discretizadas en el tiempo y en la amplitud. Los primeros microprocesadores permitieron desarrollar el procesado digital de señales. Actualmente, existen microprocesadores específicos para este tipo de aplicaciones conocidos como DSPs (Digital Signal Processors). Sin embargo, muchas aplicaciones actuales requieren un hardware específico de alta velocidad. En estos casos pueden ser los dispositivos programables , en concreto las FPGAs (Field Programmable Gate Array), los que mejor se adapten a las necesidades de velocidad, área y consumo para realizar el procesado digital de señales. Estos dispositivos permiten realizar complejos algoritmos que pueden ser modificados una vez ha sido implementado el diseño.

Los principales objetivos del presente trabajo son el estudio del procesado digital de señales usando dispositivos reconfigurables de alta velocidad, como son las FPGAs, y su aplicación a un ejemplo concreto. Con este fin se propone el diseño de un sistema, su especificación usando un lenguaje de descripción del hardware VHDL (Very High Description Language) y su verificación experimental sobre un circuito integrado de tipo FPGA. En el procesado digital de señales el uso de filtros suele ser necesario. Por ello se propone la implementación de un sistema de filtrado de señales procedentes de un electrocardiograma (ECG). Otros de los objetivos del trabajo son conocer los entornos de desarrollo que los fabricantes de dispositivos de lógica programable ponen a disposición de los diseñadores y manejar herramientas de software matemático como Matlab.

El primer capítulo tiene como objetivo principal el estudio de las características fundamentales del procesado digital de señales. También tiene como objetivo el estudio de los filtros digitales, en concreto los filtros de respuesta impulso finita FIR (Finite Impulse Response). El segundo capítulo se centra en el análisis del hardware necesario para realizar adecuadamente el procesado digital. Se estudian en detalle las tecnologías de los dispositivos de lógica programable y se analiza la familia de dispositivos FPGA que se empleará para la

realización práctica. Los objetivos del tercer capítulo son el diseño realizado de los filtros FIR para señales ECG. Por último, en el cuarto capítulo, se realiza la implementación del sistema de filtros diseñado sobre un dispositivo concreto. Se analizan las dos arquitecturas posibles para implementar el sistema y se exponen los resultados obtenidos tras las implementaciones de dichas arquitecturas para la aplicación práctica propuesta.

CAPÍTULO 1: INTRODUCCIÓN AL PROCESADO DIGITAL DE SEÑALES. FILTROS FIR.

Las señales analógicas son funciones continuas en el tiempo y normalmente toman valores en un rango continuo. Estas señales pueden procesarse mediante sistemas analógicos con el fin de extraer la información deseada o cambiar sus características. El tratamiento digital de señales proporciona un método alternativo de procesar una señal analógica. Se aplica en un amplio rango de disciplinas: procesamiento de voz, transmisión de señales a través de canales telefónicos, procesamiento y transmisión de imágenes, sismología, geofísica y detección de explosiones nucleares, entre otros [1].

Para poder realizar un procesado digital de una señal, ésta tiene que ser digital como se indica en la Figura-1. Una señal digital es una señal discreta en el tiempo que sólo puede tomar un conjunto de valores discretos. Si la señal que se va a procesar es una señal analógica, se convierte en una señal digital muestreándola en instantes discretos de tiempo y cuantificando sus valores mediante un conversor analógico/digital. Seguidamente se procesa la señal. La señal procesada será una señal digital, en caso de necesitar una señal analógica, se empleará un conversor digital/analógico a la salida [1].

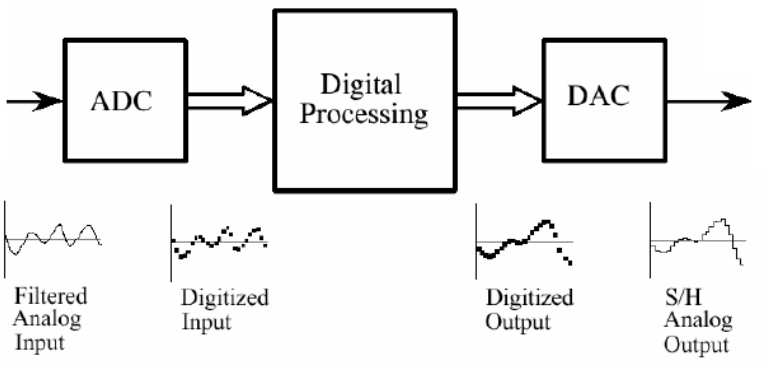


Figura-1: etapas del tratamiento digital de una señal analógica.

En casos donde las señales pueden ser digitalizadas sin perder información de forma considerable y se cuenta con suficiente tiempo y potencia para realizar las conversiones y los cálculos necesarios, es preferible utilizar un sistema digital. Esto se debe a varias razones:

- Flexibilidad. El hardware de procesamiento digital permite reconfigurar las operaciones del procesado modificando el programa. Sin embargo, la reconfiguración de un sistema analógico implica el rediseño del sistema y la verificación del correcto funcionamiento del mismo.

- Precisión. Los componentes de circuitos analógicos hacen difícil que el diseñador del sistema pueda controlar la precisión de un sistema. Los digitales no muestran este problema.
- Procesado en tiempo no real. Las señales digitales se almacenan en dispositivos con memoria. De tal manera que se pueden hacer procesos en tiempo "no real" en un laboratorio remoto.
- Mayor complejidad. El procesado digital de señales también permite la implementación de algoritmos de tratamiento de señales más sofisticados. En sistemas analógicos suele ser difícil realizar ciertas operaciones matemáticas.
- Inmunidad al ruido. Una señal digital transmitida como una secuencia de '1's y '0's se puede reconstruir sin error, puesto que el ruido normalmente no es suficiente para evitar la identificación de los '1's y '0's.

  Aún así presenta algunos inconvenientes:
- Mayor consumo. En algunos casos los circuitos digitales consumen más energía que los circuitos analógicos, produciendo un aumento de temperatura de los sistemas. Suelen incorporar sistemas de ventilación o refrigeración.
- Errores de cuantificación. Estos errores se deben tanto a la conversión A/D como a la D/A de la señal.

1.1 SEÑALES DIGITALES

Para obtener señales digitales [2] a partir de señales analógicas primero es necesario convertirlas a una señal en tiempo discreto y luego cuantificar las amplitudes.

1.1.1 Muestreo

El muestreo consiste en tomar muestras de la señal continua en el tiempo *x(t)* en instantes discretos de tiempo (*nT*)

$$x(t) \Rightarrow x(nT) \tag{1.1}$$

donde *T* es el periodo de muestreo y se determina a partir del teorema de Nyquist-Shanon que dice que la frecuencia de muestreo, f_m, tiene que ser mayor que dos veces la frecuencia más alta contenida en la señal analógica, f_{max} [3]

$$f_m \geq 2f_{max} \tag{1.2}$$

Si se utiliza una frecuencia menor a la establecida por el teorema de Nyquist, se produce una distorsión conocida como aliasing [2]. El aliasing impide recuperar correctamente la señal cuando las muestras de ésta se obtienen en intervalos de tiempo demasiado largos.

1.1.2 Cuantificación

La cuantificación es la conversión de una señal con valores continuos en una señal con valores discretos [2]. Para ello se divide el rango dinámico de la señal en *N* niveles de cuantificación y a cada muestra se le asigna el valor del nivel de cuantificación correspondiente. La cuantificación es un proceso irreversible y siempre se produce una pérdida de información.

Los valores cuantificados pueden representarse usando diferentes formatos. En la representación de coma fija se asigna un número de bits a la parte entera y a la fraccional tal que el número de bits disponibles sean b. El número de niveles N es potencia de 2, $N = 2^b$. Por tanto la resolución de la cuantificación en la representación en coma fija vendrá limitada por el número de bits disponibles.

1.2 SISTEMAS DIGITALES

A continuación, se presentan distintas clasificaciones de los sistemas digitales [1], [2]:

- Sistemas lineales y no lineales
    - Lineal: satisface el principio de superposición. Si al aplicar como entradas al sistema las señales $x_1(t), x_2(t) \dots x_n(t)$ obtenemos como salida del sistema las señales $y_1(t), y_2(t) \dots y_n(t)$, la respuesta del sistema a una señal de entrada $x(t)$ formada por la combinación lineal de dos o más señales.

$$x(t) = ax_1(t) + bx_2(t) + \cdots + kx_n(t) \tag{1.3}$$

      es igual a la combinación lineal de la suma de las respuestas del sistema a cada una de las señales.

$$y(t) = ay_1(t) + by_2(t) + \cdots + ky_n(t) \tag{1.4}$$

    - No lineal: no satisface el principio de superposición (1.3), (1.4).
- Sistemas invariantes o variantes en el tiempo
    - Invariante en el tiempo. El sistema es constante en el tiempo.
    - Variante en el tiempo. El sistema depende del tiempo.
- Sistemas estáticos y dinámicos
    - Un sistema estático depende de la muestra de entrada en dicho instante pero no de muestras pasadas ni futuras de la entrada. En caso contrario se dirá que es dinámico.
- Sistemas causales y no causales
    - Se dice que un sistema es causal si la salida en cualquier instante dado sólo depende de las entradas actuales y pasadas pero no depende de las entradas futuras.
    - Si depende de señales futuras es no causal.

- Sistemas estables e inestables
  - Se dice que un sistema es estable sí y sólo sí toda entrada acotada genera una señal de salida acotada [4].
- Sistemas discretos recursivos y no recursivos
  - Se dice que un sistema discreto es recursivo si su salida en un instante dado depende de los valores de las salidas en tiempos anteriores.
  - Si no depende de los valores de salidas en tiempos anteriores entonces se dice que es no recursivo.

Tras haber clasificado los sistemas respecto de una serie de propiedades, nos vamos a centrar en el análisis de los sistemas lineales, invariantes en el tiempo (sistemas LTI). Los motivos son [2]:

1- Muchos sistemas prácticos son sistemas LTI o pueden aproximarse fácilmente a ellos.
2- Existen gran variedad de métodos matemáticos para el análisis de estos sistemas.

1.2.1 Sistemas lineales invariantes en el tiempo

A continuación veremos que cualquier señal discreta se puede representar como combinación lineal de impulsos unitarios retardados. Esto permite caracterizar cualquier sistema LTI mediante su respuesta al impulso unitario.

$$\delta(n) = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases} \tag{1.5}$$

Primero se va a demostrar que una señal discreta puede expresarse como combinación lineal de impulsos unitarios retardados. En la Figura-2 se observa un impulso unitario.

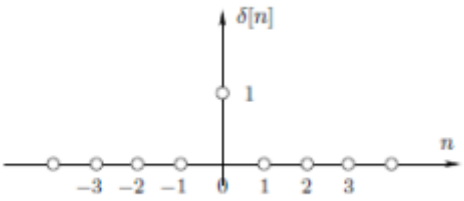
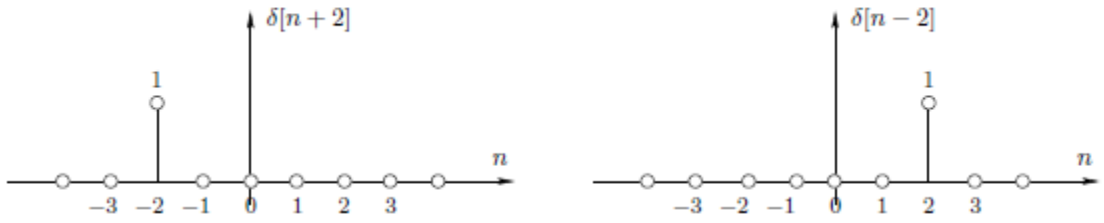


Figura-2: señal impulso unitario.

Si se realizan desplazamientos en el tiempo se obtienen las funciones representadas en las Figuras-3(a) y 3(b).



(a) (b)

Figura-3: señales impulso desplazadas: (a) dos unidades a la izquierda y (b) dos unidades a la derecha

Cualquier señal *x(n)* se puede escribir como combinación lineal de impulsos desplazados. En la Figura-4 puede verse un ejemplo.

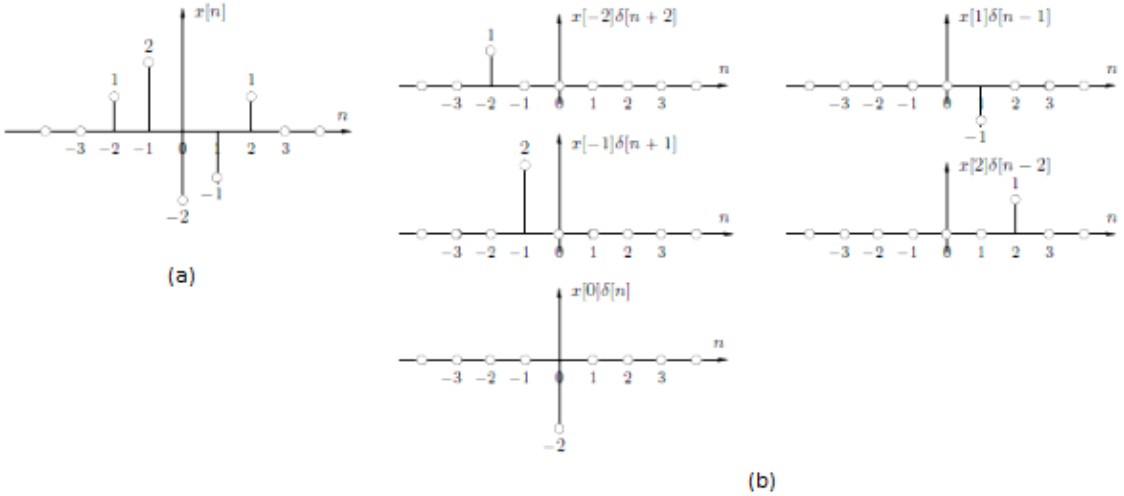


Figura-4: descomposición de una señal (a) en señales impulso desplazadas (b).

La señal de entrada $x(n)$, se expresa como suma de impulsos $\delta(n-k)$ de acuerdo a (1.5). Las amplitudes de los impulsos de la señal de entrada están determinadas por $x(k)$.

$$x(n) = \sum_{k=-\infty}^{\infty} x(k)\delta(n-k) \tag{1.6}$$

Teniendo en cuenta que el sistema es lineal podemos expresar la respuesta del sistema ante la entrada *x(n)* como la suma de las respuestas de las señales elementales donde *H* representa la función de transferencia del sistema y $h(n,k)$ es la respuesta al impulso unitario [5].

$$y(n) = H[x(n)] = H\left[\sum_{k=-\infty}^{\infty} x(k)\delta(n-k)\right] = \sum_{k=-\infty}^{\infty} x(k)\,H[\delta(n-k)] = \sum_{k=-\infty}^{\infty} x(k)h(n,k) \tag{1.7}$$

Se puede simplificar la ecuación (1.7) teniendo en cuenta que es un sistema invariante en el tiempo [5]

$$H[\delta(n-k)] = h(n-k) \tag{1.8}$$

y sustituyendo (1.8) en (1.7)

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k) \tag{1.9}$$

A la expresión (1.9) se le denomina suma de convolución [5]. Debido a la propiedad distributiva de la convolución es indiferente cuál de las dos funciones será la que se traslade, y cuál se quede fija. Por tanto (1.9) también puede escribirse como

$$y(n) = \sum_{k=-\infty}^{\infty} x(n-k)h(k) \tag{1.10}$$

Ahora hay que tener en cuenta que la causalidad es necesaria en cualquier aplicación de tratamiento de señales en tiempo real. En cualquier instante dado no se tiene acceso a valores futuros de la señal de entrada. Por tanto, un sistema LTI es causal sí y sólo sí su respuesta al impulso es cero para los valores negativos de *k*.

$$y(n) = \sum_{k=0}^{\infty} x(n-k)h(k) \tag{1.11}$$

La siguiente característica a analizar es la estabilidad. Un sistema es estable si ante una entrada acotada la salida también lo es. Se puede demostrar que un sistema LTI es estable si su respuesta al impulso es absolutamente sumable [2], [4]

$$\sum_{k=0}^{\infty} |h(k)| \leq \infty \tag{1.12}$$

La condición (1.12) es suficiente y necesaria para que este tipo de sistemas sean estables.

Por último queda estudiar la recursividad. Se pueden distinguir dos tipos de sistemas:

- Sistemas LTI recursivos: Son sistemas cuya salida depende de salidas anteriores (recursivo) y al ser estimulados con una entrada impulso su salida no vuelve al reposo. En este tipo de sistemas la estabilidad debe comprobarse ya que son sistemas realimentados. Un caso particular de sistemas LTI recursivos es el sistema IIR (Infinite Impulse Response) [6]. Son sistemas que tienen una respuesta al impulso de duración infinita.
- Sistemas LTI no recursivos: Se basan en obtener la salida a partir de las entradas actuales y anteriores. Los sistemas FIR (Finite Impulse Response), son

sistemas LTI no recursivos que tienen una respuesta al impulso de duración finita [6]. Por tanto, en este caso (1.12) es una suma finita de *N* términos.

$$y[n] = \sum_{k=0}^{N-1} h(k)x(n-k) \tag{1.13}$$

Los sistemas FIR siempre son siempre estables, ya que se verifica (1.12). La salida *y(n)* puede escribirse como la convolución de la entrada *x(n)* con la respuesta impulso *h(n).*

Una vez analizadas todas las características de los sistemas LTI, se tratarán los filtros digitales, concretamente los filtros FIR, puesto que cualquier aplicación de procesado digital de señal los requiere. En el procesado digital, tras las conversiones A/D , el uso de filtros suele ser necesario. Además los filtros FIR nos garantizan la estabilidad del sistema. Por todo ello, se analizarán las ventajas y desventajas de los filtros digitales frente a los filtros analógicos y se explicará un método de diseño de estos filtros.

1.3 FILTROS DIGITALES

Los filtros digitales actúan sobre valores numéricos asociados a muestras de señales analógicas previamente digitalizadas por conversores A/D o simplemente sobre un conjunto de números almacenados en una memoria.

En la Tabla-1 se observan las características de los filtros digitales, en concreto de los filtros FIR [5]. Hay varias características importantes a tener en cuenta. Una de ellas es el efecto de longitud de palabra finita. Partiendo de la expresión (1.13), se definen los coeficientes del filtro b_k como la respuesta al impulso.

$$y[n] = \sum_{k=0}^{N-1} h[k]x[n-k] = \sum_{k=0}^{N-1} b_k x[n-k] \tag{1.14}$$

Los coeficientes del filtro implementado serán distintos de los calculados teóricamente si la representación numérica que se utiliza para implementar el filtro no es de precisión infinita. En los dispositivos digitales la precisión es siempre finita. Hay que establecer el número bits de las señales. En coma fija se especifica cuántos de ellos se emplean para la parte fraccional y cuantos para la parte la entera.

Por otra parte, hay distintas técnicas para la obtención de los coeficientes del filtro. Con estas técnicas se intentan obtener los coeficientes del filtro que más se aproximan al filtro ideal deseado. Los métodos de diseño se analizan en el siguiente apartado.

	Características favorables	Características desfavorables
Precisión	• **Gran margen dinámico**. En filtros analógicos es difícil lograr atenuaciones mayores de 60–70dB. • **Fase lineal.** Los filtros FIR se pueden diseñar para tener una respuesta de fase estrictamente lineal (distorsión de fase nula), lo que es importante en muchas aplicaciones, como transmisión de datos, audio digital y procesamiento de imágenes.	• **Obtención de coeficientes.** Los métodos para obtener los coeficientes de los filtros pueden ser complejos. • **Efecto de longitud de palabra finita.** En digital la precisión es finita. La pérdida de precisión afecta tanto a las señales de entrada y salida como a los coeficientes del filtro.
Producción	• Posibilidad de reconfigurar el dispositivo. • Menor tamaño. • Producción en serie.	
Otros	• **Memoria**. Las señales filtradas y sin filtrar pueden almacenarse en memoria para uso o análisis posterior. • **Mayor fiabilidad** (por ejemplo no hace falta realizar calibraciones)	• **Consumo**. El consumo de los circuitos digitales aumenta al aumentar la frecuencia de operación.

Tabla-1: características de los filtros FIR.

1.3.1 Diseño de filtros FIR

Existen distintos tipos de métodos de diseño de filtros FIR. Uno de los más usados es el método de las ventanas y es el que se explicará a continuación [5], [6].

1.3.1.1 Método de las ventanas

En este método, a partir de las especificaciones de la respuesta en frecuencia deseada $H_d(\omega)$, determinamos la correspondiente respuesta al impulso unidad $h_d(n)$ [5], [6]. En (1.14) se definían los coeficientes del filtro b_k como las respuestas al impulso unidad $h_d(n)$. Así, $h_d(n)$ y $H_d(\omega)$ están relacionados por la transformada de Fourier.

$$H_d(\omega) = \sum_{n=0}^{\infty} h_d(n) e^{-i\omega n} \tag{1.15}$$

Donde

$$h_d(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H_d(\omega) e^{-i\omega n} d\omega \tag{1.16}$$

Por tanto, dada $H_d(\omega)$ podemos determinar la respuesta al impulso unidad $h_d(n)$ evaluando la integral en (1.16).

La respuesta al impulso unidad $h_d(n)$ obtenida por la integral es infinita y tiene que truncarse en algún punto. Si se trunca el filtro en *n=M-1* se dice que es un filtro FIR de longitud *M*. Truncar $h_d(n)$ a una longitud *M* es equivalente a multiplicar $h_d(n)$ por una ventana $w(n)$, por ejemplo la ventana rectangular [10].

$$w(n) = \begin{cases} 1 & 0 \le n \le M-1 \\ 0 & n \ge M \end{cases} \tag{1.17}$$

Entonces la respuesta al impulso unidad del filtro FIR es,

$$h(n) = \begin{cases} w(n)h_d(n) & 0 \le n \le M-1 \\ 0 & n \ge M \end{cases} \tag{1.18}$$

Analicemos el efecto de la función de la ventana sobre la respuesta en frecuencia deseada, $H_d(\omega)$. Se recuerda que un producto en el dominio temporal se transforma en una convolución en el dominio frecuencial. Por tanto, el producto $w(n)$ por $h_d(n)$ es equivalente a realizar la convolución de $H_d(\omega)$ con $W(\omega)$, ver Figura-5, siendo $W(\omega)$ la representación en el dominio de la frecuencia de la función ventana [6]. Así se obtiene la respuesta en frecuencia del filtro FIR.

$$\begin{aligned} H(w) = H_d(w) * W(w) &= \left(\sum_{n=0}^{\infty} h_d(n)e^{-i\omega n}\right)\left(\sum_{n=0}^{\infty} w(n)e^{-i\omega n}\right) = \\ = \sum_{n=0}^{M-1} h_d(n)w(n)e^{-i\omega n} &= \sum_{n=0}^{M-1} h(n)e^{-i\omega n} \end{aligned} \tag{1.19}$$

donde el operando " $*$ " indica convolución.

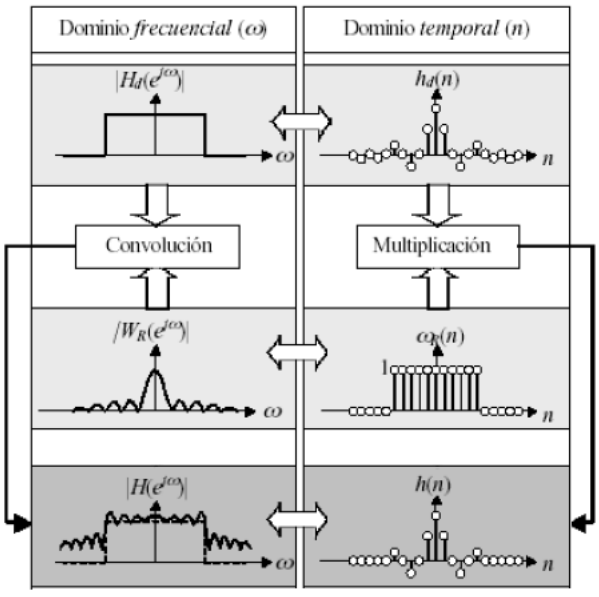


Figura-5: efecto del enventanado del filtro ideal.

Efectos del enventanado:

- La anchura del lóbulo principal está relacionada con la banda de transición en el filtro como puede verse en la Figura-6 [6]. Cuanto mayor sea la anchura del lóbulo principal mayor será la banda de transición del filtro. Se puede demostrar que la anchura del lóbulo principal depende del número de términos *M*, [5]. La anchura de la ventana rectangular es $\frac{4\pi}{M}$, esto implica que cuanto mayor sea *M* menor será la anchura y por tanto la banda de transición más estrecha.

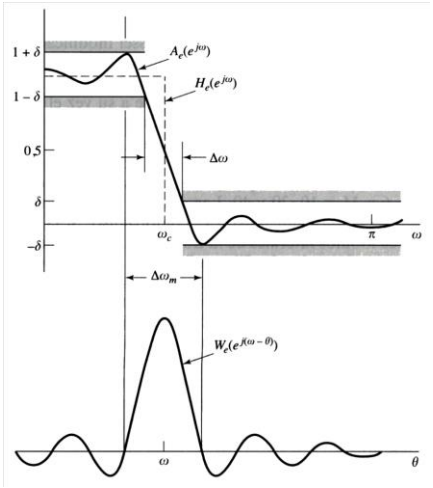


Figura-6: relación entre el lóbulo principal y la banda de transición.

- Los lóbulos secundarios son los responsables de la aparición de un rizado u oscilaciones en la respuesta en frecuencia, en la banda pasante y atenuada. Analicemos la aparición de los lóbulos secundarios desde el punto de vista matemático. En las series de Fourier se determinó que si una función con una discontinuidad (filtro ideal) era aproximada mediante series de Fourier aparecen sobreoscilaciones en las proximidades de la discontinuidad. A medida que el número de términos *M* aumenta el nivel de oscilación va disminuyendo, hasta hacerse cero cuando $M \to \infty$, excepto en la discontinuidad en la que sí aparece una sobreoscilación. Este comportamiento oscilatorio en las proximidades de la discontinuidad se conoce como efecto Gibbs [6].

A la vista de este análisis, se puede tratar de mejorar las prestaciones del filtro real aumentando el número de puntos *M*, sin embargo el incremento de la longitud del filtro eleva su carga computacional. Se pueden diseñar ventanas cuyos extremos se anulen progresivamente, consiguiendo que los lóbulos secundarios se atenúen pero el lóbulo principal se ensanchará.

Existen ventanas como la Kaiser, definidas con dos parámetros *N* y *b*. Ofrecen un grado más de libertad para establecer las características del filtro pero el diseño resulta más complejo. En cualquier caso existen varias de ventanas de un único parámetro y se puede seleccionar la que mejor se adapte a las necesidades del sistema. Por ello, las ventanas con dos parámetros no se suelen emplear. En la Figura-7 se muestran algunas ventanas y en la Tabla-2 las características de algunas de ellas [6].

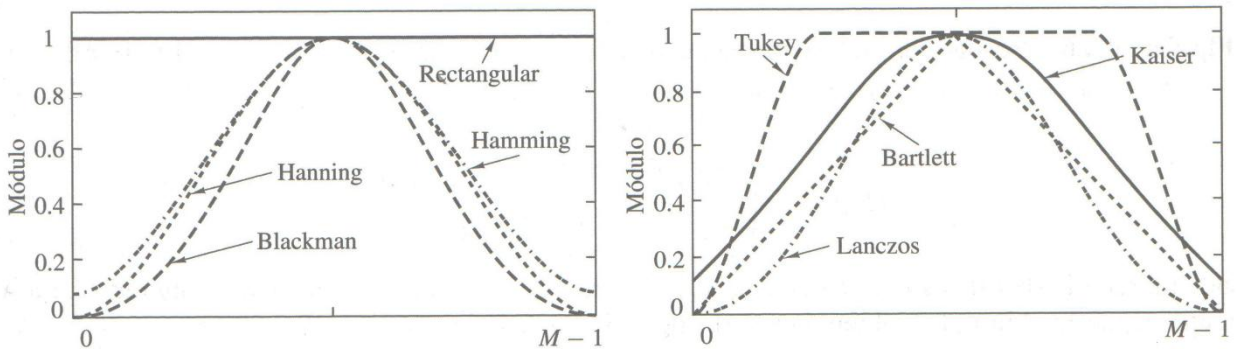


Figura-7: funciones de distintos tipos de ventanas.

Ventana	Anchura del lóbulo principal de la ventana	Anchura de la banda de transición del filtro deseado	Pico lóbulo secundario de la ventana (dB)	Atenuación del filtro diseñado con esta ventana (dB)
Rectangular	$\frac{4\pi}{N}$	$\frac{1.8\pi}{N}$	-13	-21
Barlett (triangular)	$\frac{8\pi}{N}$	$\frac{6.1\pi}{N}$	-25	-25
Hamming	$\frac{8\pi}{N}$	$\frac{6.6\pi}{N}$	-41	-53
Blackman	$\frac{12\pi}{N}$	$\frac{11\pi}{N}$	-57	-74

Tabla-2-: características de algunas ventanas de la Figura 5.

El siguiente capítulo trata sobre los dispositivos que permiten realizar procesado digital de señales. El capítulo se centrará en los dispositivos de lógica programable, en particular las FPGAs (Field Programmable Gate Array) debido a que permiten procesar a alta velocidad, reducir el consumo y el tamaño.

CAPÍTULO 2: PROCESADO DIGITAL DE SEÑALES CON DISPOSITIVOS DE LÓGICA PROGRAMABLE

Una alternativa para realizar el procesado digital de señales es usar los microprocesadores. De hecho, se ha desarrollado una clase de microprocesadores específicos para el procesado digital, conocidos como DSP (Digital Signal Processors) [7]. Los DSPs realizan de forma eficiente funciones típicas del procesado digital de señales como la suma de productos (1.14). Sin embargo, en algunos casos, la velocidad, el área y el consumo de energía no son adecuados para el desarrollo de sistemas de procesado digital.

Otra alternativa son los circuitos integrados. Se pueden clasificar los circuitos integrados dependiendo de su metodología de diseño y fabricación. Existen tres grandes grupos como se observa en la Figura-8 [8]: circuitos integrados estándar, circuitos integrados de aplicación específica, (ASICs, Application Specific Integrated Circuit) y dispositivos de lógica programable, (PLDs, Programmable Logic Device).

Los circuitos integrados estándar poseen características lógicas y eléctricas perfectamente definidas. Este tipo de circuitos presenta la ventaja de su reducido coste y su gran fiabilidad debido a que se fabrican en grandes series. Sin embargo, para poder realizar circuitos un tanto complejos se necesitarán un gran número de dispositivos de este tipo. Además es muy común que pueda haber copias no autorizadas.

Los ASICs son circuitos que se diseñan para realizar una función específica en una aplicación concreta. Existen varios tipos de ASICs, siendo los más utilizados los de tecnología full-custom y semi-custom. En los circuitos full-custom, el diseñador tiene control total en el diseño de los dispositivos. Son los más rápidos y eficientes, pero son los que requieren más tiempo de diseño y tienen mayores costes. Los semi-custom, dan cierta libertad de diseño, pudiendo, por ejemplo, especificar las conexiones que deben ser realizadas entre los transistores [9]. Sin embargo, tanto los full-custom como los semi-custom requieren de la intervención del fabricante en la fase final de producción.

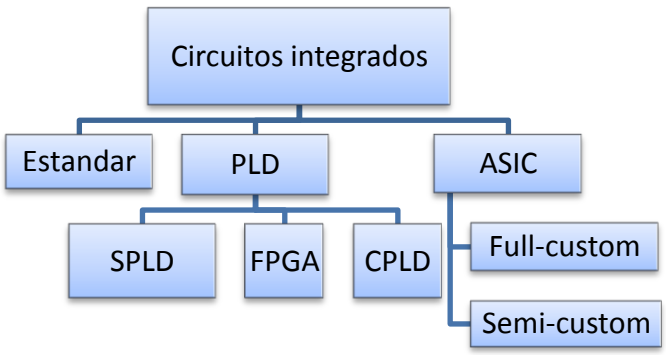


Figura-8: Clasificación de los circuitos integrados.

Por último, en los dispositivos de lógica programable el fabricante dispone de una gran cantidad de circuitos básicos en un sustrato de silicio. Estos circuitos básicos pueden utilizarse para diferentes funciones cambiando sus conexiones internas, lo cual puede hacer el diseñador desde el exterior del dispositivo. Los primeros dispositivos de lógica programable fueron los SPLDs (Simple Programmable Logic Device). Su estructura interna está formada por un conjunto de matrices de puertas AND y puertas OR. El avance en el desarrollo de estos dispositivos ha dado lugar a dos grupos: CPLD (Complex Programmable Logic Device) y FPGA (Field Programmable Gate Array).

Los CPLDs y FPGAs contienen múltiples copias de elementos lógicos LE (Logic Element) o celdas. Los elementos lógicos se distribuyen en filas y columnas en el chip. Para llevar a cabo funciones más complejas los elementos lógicos pueden conectarse mediante una red de interconexión programable.

Los CPLDs tienen tiempos de propagación más rápidos y más predecibles mientras que las FPGAs ofrecen una mayor densidad de integración. Aparte de estas diferencias, la diferencia fundamental entre las FPGAs y los CPLDs es su arquitectura y la tecnología de programación. Las FPGAs son los dispositivos que mejor se adaptan a las necesidades de velocidad, área y consumo para realizar el procesado digital de señales, permitiendo realizar modificaciones una vez implementado el diseño.

El objetivo del siguiente apartado es el estudio de las tecnologías empleadas en las FPGAs. Para ello se analiza previamente la evolución de las tecnologías de los dispositivos de lógica programable.

2.1 TECNOLOGÍAS DE DISPOSITIVOS DE LÓGICA PROGRAMABLE

2.1.1 Tecnología PROM

Los primeros PLDs fueron los SPLDs (Simple Programable Logic Devices). Los SPLDs constan de dos matrices de puertas lógicas: una AND, fija, y otra OR, programable como puede verse en la Figura-9, [10],[11].

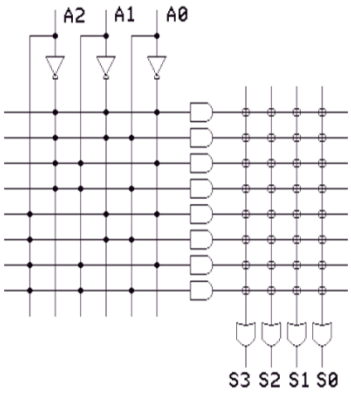


Figura-9: Matriz AND fija - OR programable."•" conexión fija. "o" conexión programable.

Las interconexiones programables están formadas por fusibles (NiCr o Titanio-Tungsteno) y diodos unidos en serie como observamos en la Figura-10.

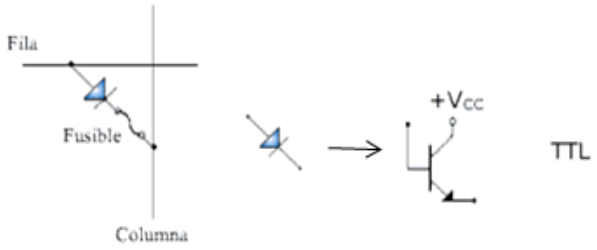


Figura-10: elementos programables de tipo fusible.

2.1.2 Tecnología EPROM

La tecnología que permitió el gran desarrollo de los SPLD hasta el punto de ser denominados CPLDs fue la tecnología EPROM. Los dispositivos de tecnología EPROM (Erasable Programmable Read Only Memory) son dispositivos no volátiles reprogramables [10], [11]. La configuración del dispositivo se almacena en una celda EPROM. La celda consiste en un único transistor NMOS con una puerta flotante, también conocido como FAMOS (Floating-gate Avalanche-injection MOS), Figura-11 y Figura-12. La puerta flotante es una capa de polisilicio rodeada de óxido, aislante, que se coloca entre la puerta y el sustrato. Cuando se almacenan electrones en la capa flotante, se incrementa la tensión umbral del transistor, lo que provoca que quede polarizado en corte y el transistor queda programado.

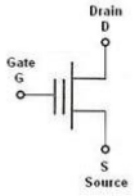


Figura-11: elemento de matriz EPROM.

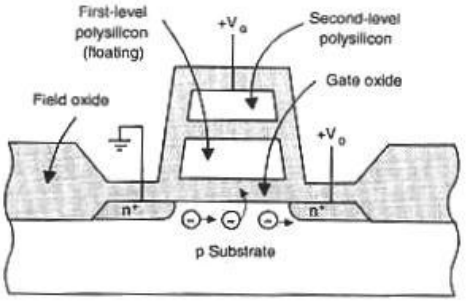


Figura-12: sección lateral de una celda EPROM

El gran inconveniente de esta tecnología es el proceso de borrado ya que se realiza mediante radiación ultravioleta. Con el fin de eliminar este problema se desarrolló la tecnología EEPROM.

2.1.3 Tecnología EEPROM

Una celda de EEPROM (Electically Erasable Programmable Read Only Memory) consta de dos transistores: un MOSFET y un FLOTOX (Floating Gate Tunnel Oxide) que es un transistor de puerta flotante con una fina capa de óxido sobre la zona del drenador [11]. En la Figura-13, puede observarse el símbolo de la celda básica. En la Figura-14, se observan las tensiones necesarias para que el transistor sea programado.

Figura-13: elemento programable EEPROM compuesto por un transistor NMOS y un FLOTOX.

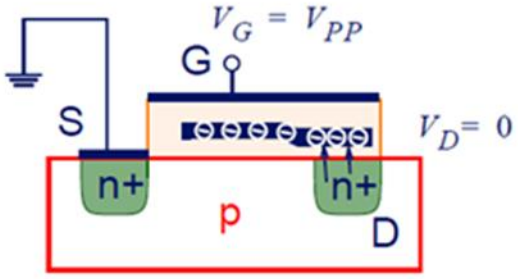


Figura-14: representación del proceso de programación en el transistor FLOTOX .

2.1.4 Tecnología FPGA antifusible

Una de las tecnologías empleadas en las FPGAs es el antifusible. Otra es la tecnología SRAM que se analizará en el siguiente apartado. Los antifusibles, Figura-15, están compuestos por dos capas conductoras separadas por un aislante de alta impedancia [12]. Al aplicar una tensión elevada entre las dos zonas conductoras el aislante se convierte en un conductor, creando así un “link” o zona de paso. De esta forma se reduce la impedancia. Esta situación es irreversible y por tanto no son reprogramables.

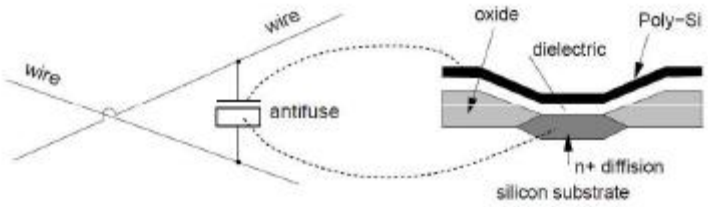


Figura-15: símbolo electrónico y sección lateral de un antifusible ONO.

2.1.5 Tecnología FPGA SRAM

La celda básica de la tecnología SRAM se compone cuatro transistores, formando un biestable [11], [12]. Un biestable es un circuito que almacena un bit de memoria, puede estar en uno de los dos estados lógicos. Además se utilizan otros dos transistores adicionales, como puede verse en la Figura-16, para controlar el acceso al biestable durante las operaciones de lectura y escritura. Se trata por tanto de una celda reprogramable volátil, ya que la información se pierde al desconectar la alimentación del dispositivo.

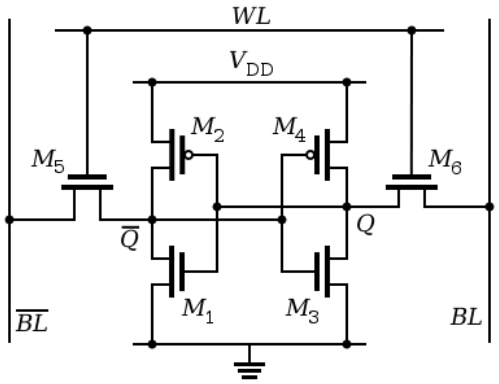


Figura-16: celda de la memoria SRAM compuesta por seis transistores**.**

2.1 RESUMEN COMPARATIVO

La Tabla-3 resume las características principales de cada tecnología. El uso de las tecnologías analizadas depende de las aplicaciones que las requieren. Por una parte, los CPLD por su reducido tamaño y su bajo consumo se emplean a menudo en aplicaciones portátiles junto con pequeñas baterías. Ejemplo de ello es la "ropa inteligente", es decir, prendas que incorporan sensores biológicos (humedad), sensores de movimiento (zapatillas de deportistas) y de vibraciones (pulsaciones), entre otros. La información proveniente de estos sensores se procesa con la ayuda de CPLDs. Cabe destacar que los CPLDs de tecnología EEPROM tienen mayor uso que los EPROM debido al método de borrado.

Dispositivo (PLD)	Tecnología del elemento programable	Elemento programable	Símbolo	Método de programación	Reprogramable	Método de borrado	Volátil	Fabricantes
SPLD	Bipolar	Fusible en serie con diodo		Eléctrica en corriente	No	-	No	ATMEL, RHOM semiconductor
CPLD	EPROM	Transistor FAMOS		Eléctrica en tensión	Si	Luz ultravioleta	No	ATMEL, Dinies
CPLD	EEPROM	Transistor FLOTOX		Eléctrica en tensión	Si	Eléctrico	No	Microchip, ATMEL, RHOM semiconductor
FPGA	Antifusible	Antifusible		Eléctrica en tensión	No	-	No	Actel, Quicklogic
FPGA	SRAM	Celda SRAM		Eléctrica en tensión	Si	Eléctrico	Si	Altera, Xilinx

Tabla-3: características principales de las tecnologías de dispositivos de lógica programable.

Por otra parte, el rango de aplicaciones de las FPGAs es muy amplio debido a la gran variedad de familias y tamaños disponibles en el mercado. Distinguiendo el tipo de aplicaciones según la tecnología empleada, las FPGAs basadas en antifusibles tienen una gran utilidad en los sectores espacial (radioastronomía) y militar (cifrado) puesto que en caso de usar las de tecnología SRAM podrían darse situaciones en las que se diera el borrado parcial o total de la configuración del dispositivo. La principal aplicación de las FPGAs de tecnología SRAM, como ya se ha explicado, es el procesamiento de señales debido a su la alta frecuencia de trabajo. El procesamiento de señales mediante las FPGAs se emplea, por ejemplo, en sistemas de visión artificial (video-vigilancia, robots), en sistemas de imágenes médicas o en codificación y cifrado. Las prestaciones de las FPGAs de tecnología SRAM también permiten su uso en el control industrial. Las utilidades de las FPGAs, tanto por el procesamiento de señal, como por los sistemas de control, hacen que estos dispositivos tengan un papel relevante en el sector del automóvil. En definitiva, las FPGA de tecnología SRAM son una herramienta básica para el procesado digital de señales.

En este trabajo se llevará a cabo la implementación de un filtro digital mediante una FPGA de tecnología SRAM de la empresa Xilinx [13]. La arquitectura de este dispositivo será analizada en el siguiente apartado.

2.3 ARQUITECTURA DE LA FAMILIA VIRTEX 5

Una de las familias de FPGAs más utilizadas por los diseñadores en los últimos años ha sido la familia Virtex-5 de Xilinx [14], [15]. La familia de FPGA Virtex-5 necesita una alimentación entre 1.2V y 3.3V dependiendo del dispositivo y la frecuencia máxima de operación puede alcanzar 550MHz. Las Virtex-5 se basan en la arquitectura ASMBL (Advanced Silicon Modular Block) con una interconexión diagonal que proporciona rutas más rápidas para la propagación de las señales internas.

Los elementos que contiene una Virtex-5 son:

- Bloques lógicos configurables, CLBs (Configurable Logic Block): son los principales recursos de la lógica para la implementación de circuitos secuenciales y combinacionales. Un elemento CLB está divido en dos segmentos, SLICE, Figura-17. Cada SLICE contiene 4 LUTs (Look-up table), 4 flip-flops, multiplexores, un elemento para lógica de acarreo, memoria RAM y registros de desplazamiento. Los dos últimos sólo se encuentran en uno de los dos SLICEs.

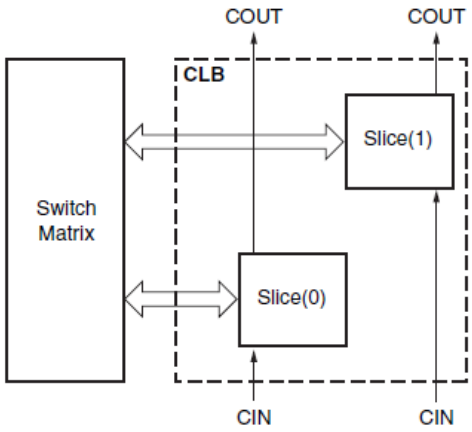


Figura-17: bloque CLB compuesto por dos Slice.

- Entradas/salidas (I/O): hasta 1,200 pines de I/O programables.
- Memoria RAM: las estructuras de bloques de RAM tienen una capacidad de 36 Kbit y operan a 550 MHz.
- Reloj: con bloques para la gestión del reloj (CMT - Clock Management Tiles), compuestos por 2 bloques DCM (Digital Clock Manager) y un bucle de enganche de fase (PLL -Phase-Locked Loop). Maxima frecuencia de operación de 550MHz.
- DSP (Digital Signal Processing) [16]: los DSPs son bloques programables compuestos por multiplexores, registros, multiplicadores y sumadores. Se pueden realizar diversas funciones: multiplicar, multiplicar y acumular, sumar y multiplicar, sumar tres entradas, multiplexar grandes buses, realizar funciones bit a bit, comparar magnitudes e implementar contadores. Proporcionan una gran flexibilidad y permiten reducir el consumo y aumentar la frecuencia de operación. En la Figura-18 se observa un esquema simplificado de la arquitectura interna de un DSP. A continuación, se analizan las señales que se observan en la Figura-18:

- A,B son entradas de primer nivel de 30 y 18 bits respectivamente. Aunque A sea de 30 bits el multiplicador es de 25x18bits, el multiplicador selecciona los 25 bits más significativos.
- C es una entrada de segundo nivel de 48 bits.
- A:B es la concatenación de A y B como entrada de segundo nivel.
- PCIN es una señal que permite enlazar dos DSPs.
- P es la señal de salida de otro DSP. Para que esta señal sea la entrada de otro DSP PCIN debe estar activada.
- OPMODE CARRYNSEL y ALUMODE son señales de control. Permiten definir qué operación se desea realizar.

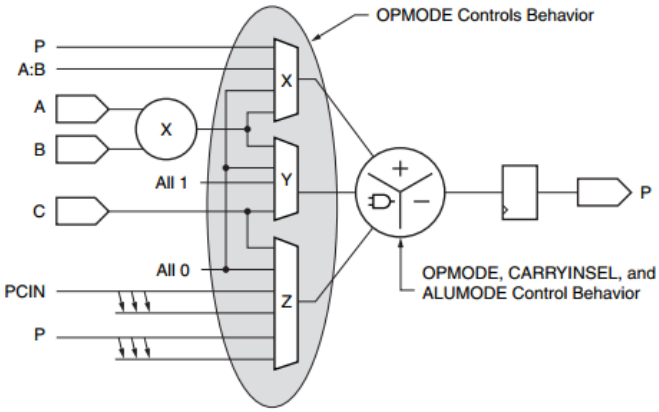


Figura-18: arquitectura simplificada de un elemento DSP de una FPGA.

En este trabajo el dispositivo que se va a utilizar es la FPGA xc5vsx50T de la familia Virtex-5 caracterizada por [15]:

- 120x34 CLBs. 8.160 SLICEs.
- 132 bloques de memoria RAM de 36kB, en total 1.752kB de memoria.
- 280 DSP.
- Frecuencia máxima 550MHz.
- 480 I/O.

Tanto el elevado número de DSPs como la alta velocidad de operación hacen que sea un dispositivo de gran utilidad para el procesado digital de señales.

En el Capítulo 3 se usarán técnicas del procesado digital de señales para una aplicación práctica y en el Capítulo 4 se llevará a cabo su implementación en una FPGA. En concreto, se diseñarán tres filtros FIR, basándose en el método de las ventanas, con el fin de filtrar señales procedentes de un electrocardiograma.

CAPÍTULO 3: DISEÑO DE FILTROS PARA SEÑALES ECG

En el cuerpo humano se generan una amplia variedad de señales eléctricas, provocadas por la actividad química que tiene lugar en los nervios y músculos que lo conforman. El corazón, por ejemplo, produce un patrón característico de variaciones de voltaje. El registro y análisis de estos eventos bioeléctricos son importantes desde el punto de vista de la práctica clínica y de la investigación. Los potenciales se generan a nivel celular, es decir, cada una de las células es un diminuto generador de voltaje.

3.1 SEÑALES ECG

Un electrocardiograma (ECG) es una prueba física ampliamente utilizada para valorar el estado del corazón. El ECG es la representación gráfica de la actividad bioeléctrica del músculo cardíaco.

Para realizar un electrocardiograma se emplea un electrocardiógrafo. Éste registra señales de amplitudes de 1mV aproximadamente y se obtiene aplicando electrodos de registro de biopotenciales [17]. Debido a que la señal capturada tiene una amplitud muy baja, se encuentra muy contaminada de ruido a 50Hz procedente de la red eléctrica. Además, la señal resulta mezclada por señales de baja frecuencia como consecuencia, por ejemplo, de la respiración del paciente. Por último, aparece también una cierta cantidad de ruido blanco que se genera en el propio sistema amplificador. Por tanto, para obtener una señal de electrocardiograma el proceso de filtrado es muy importante.

El espectro de frecuencias de la señal electrocardiográfica normalmente no tiene componentes por encima de los 60Hz en pacientes normales, por lo que se considera adecuado un ancho de banda entre 0.5Hz y 100Hz.

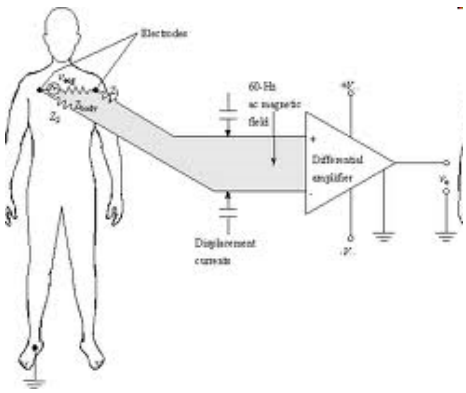


Figura-19: colocación de los electrodos para medidas diferenciales en ECG.

En este capítulo se analizará una señal ECG y se diseñará un sistema de filtrado digital formado por filtros FIR y basándose en el método de las ventanas descrito en el Capítulo 1.

Para realizar el diseño se empleará Matlab (MATrix LABoratory). Matlab es una herramienta de software matemático que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio. Se puede analizar datos, desarrollar algoritmos y crear modelos o aplicaciones. Además, disponen de una gran cantidad de herramientas denominadas toolboxes. En este trabajo se ha hecho uso del toolbox "Filter Design & Analysis Tool" [18].

3.2 ANÁLISIS DE LAS SEÑALES ECG

En la Figura-20 se representa la señal ECG obtenida de una base de datos de señales físicas [19]. La señal ECG será la entrada del sistema de filtrado, en la ecuación (1.4) será *x(n)*. Con el fin de determinar las características de los filtros, se analizan las fuentes de ruido que afectan a la señales ECG mediante un análisis de Fourier Figura-21. Del análisis se concluye que:

- La señal ECG se encuentra muy contaminada por ruido de 50 Hz procedente de la red eléctrica. Para eliminar este ruido se diseñará un filtro elimina-banda a 45-55Hz.
- La señal ECG se encuentra contaminada por componentes de baja frecuencia (entorno a 0.5Hz) generados generalmente por la respiración del paciente, que suelen estar comprendidos entre DC y 1Hz. Este ruido se filtrará mediante un filtro pasa-alto con frecuencia de corte 1Hz.
- La señal ECG se encuentra contaminada, en menor intensidad, por componentes de alta frecuencia. Se usará un filtro pasa-bajo con frecuencia de corte 100Hz.

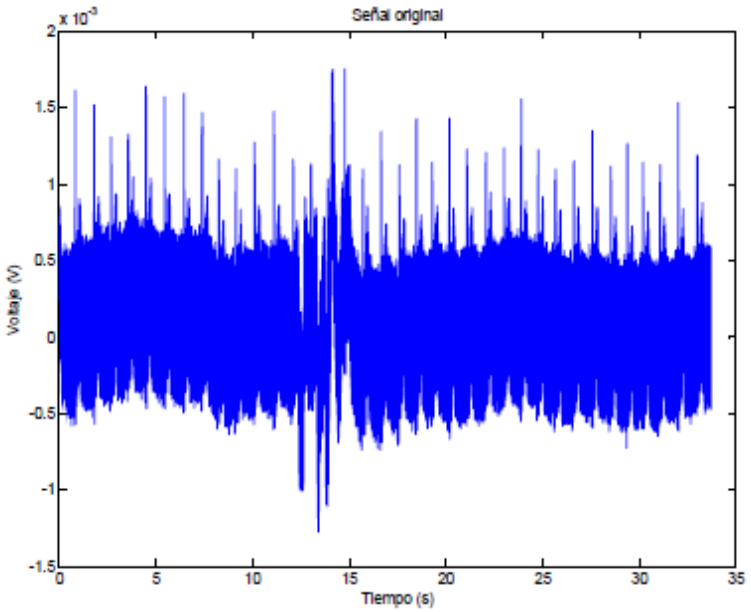


Figura-20:señal de electrocardiograma sin filtrar.

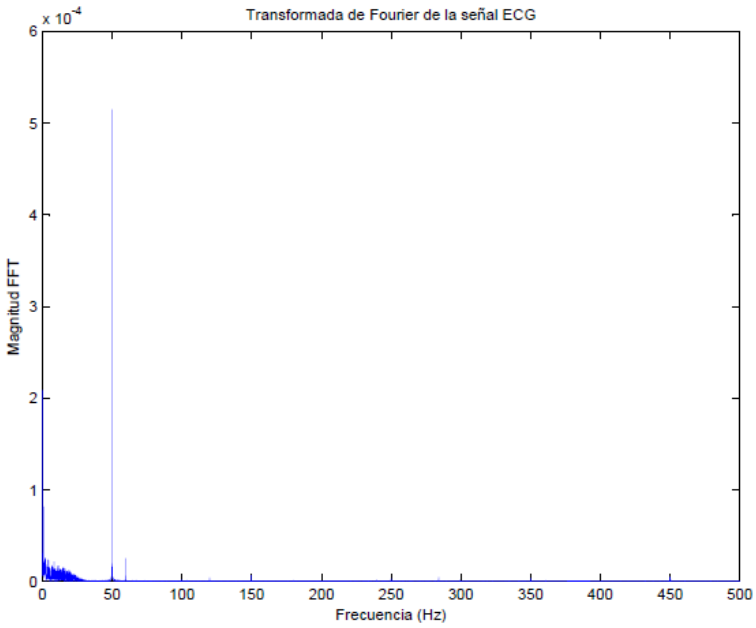


Figura-21: transformada de Fourier de la señal ECG.

3.3 DISEÑO DE FILTROS

El diseño de los filtros FIR se ha hecho teniendo en cuenta las siguientes especificaciones:

- Tipo de filtro: pasabanda, pasa alta, pasa baja, eliminabanda.
- Frecuencias de corte.
- Método de diseño. Seleccionando FIR:
    - Orden del filtro
    - Tipo de ventana.

En la Figura puede verse la captura de las especificaciones del filtro mediante "Filter Design & Analysis Tool". Con esta herramienta se realizaron distintos tipos de filtros variando tanto el orden del filtro como el tipo de ventanas. Se observó que con un orden de 2500 (en cualquier ventana) los filtros apenas mostraban distorsiones. Pero 2500 es un orden demasiado elevado para luego poder implementarlo. Se estableció como requisito fundamental que los filtros no debían superar un orden de 300. Teniendo en cuenta esta restricción se establecen las especificaciones que deben cumplir cada filtro. Seguidamente se presentará el diseño de cada filtro.

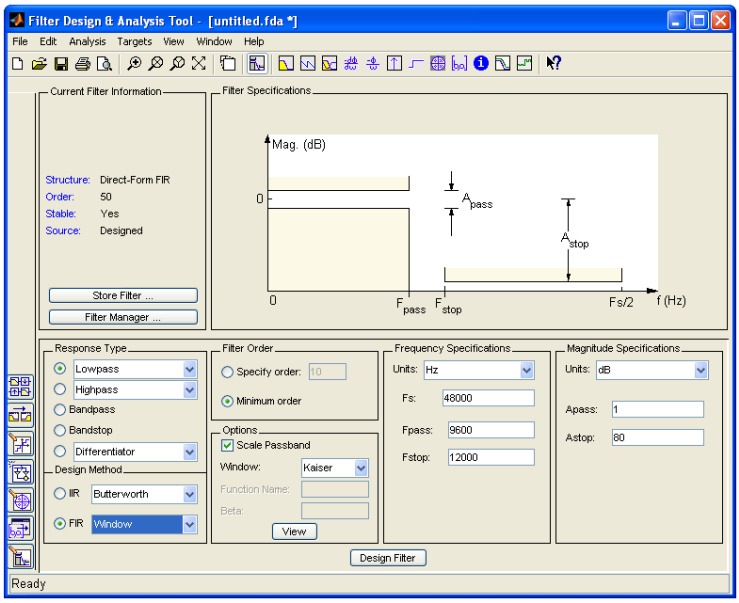


Figura-22: toolbox Filter Design & Analysis Tool de Matlab.

3.3.1 Filtro Notch

El primer filtro que se va a diseñar es el filtro Notch [12], [13]. Este filtro elimina el ruido procedente de la red eléctrica a 50Hz. Las frecuencias de corte son 45Hz y 55Hz. Observando la transformada de Fourier de la Figura-21 se establece que la condición que debe cumplir este filtro es que la componente frecuencial a 50Hz no debe superar en más de un 30% las componentes frecuenciales útiles. Teniendo en cuenta esta condición y que el orden debe ser menor que 300, se diseña el filtro usando la ventana Hamming y un orden de 250. La ventana Hamming está representada en la Figura-7 del Capítulo 1.

Tras pasar la señal ECG por el filtro Notch, se realiza la transformada de Fourier de la señal de salida, ver Figura-23. En ella se observa que la componente frecuencial a 50Hz de la señal ECG se ha reducido en un 93% y que supera en un 27% a las componentes frecuenciales útiles. En la Figura-24 se muestra la señal ECG sin 50Hz.

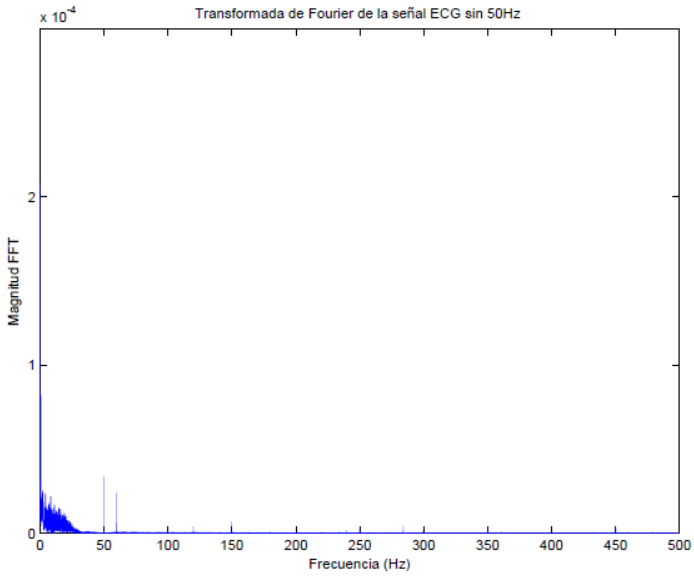


Figura-23: transformada de Fourier de la señal filtrada por el filtro Notch.

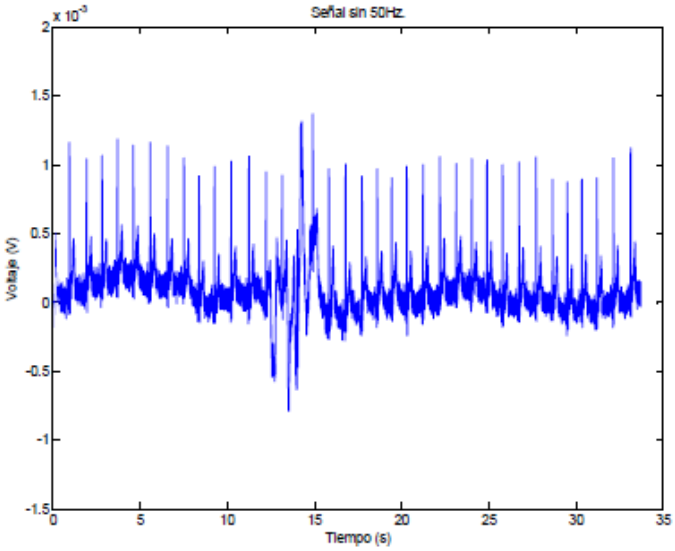


Figura-24: señal sin la componente frecuencial de 50Hz

3.3.2 Filtro pasa-alta

El segundo filtro que se va a diseñar es el filtro pasa-alta [12], [13]. Este filtro elimina el ruido de las componentes de baja frecuencia generados normalmente por la respiración del paciente. La frecuencia de corte es 1Hz. En este filtro no se puede establecer la misma condición que en el filtro Notch porque el orden del filtro sería superior a 300. Observando la transformada de Fourier de la Figura-23 y que el orden debe ser menor que 300, se establece que la condición que debe cumplir este filtro es que reduzca las componentes frecuenciales menores de 1Hz en un 50%.

Teniendo en cuenta las especificaciones establecidas, se diseña el filtro usando la ventana Tukey y un orden de 250. La ventana Tukey está representada en la Figura-7 del

Capítulo 1. Tras pasar la señal ECG sin 50Hz por el filtro pasa-alta, se realiza la transformada de Fourier de la señal de salida, ver Figura-25. En ella se observa que las componentes en torno a 0.5Hz se reducen en un 70%. En la Figura-26 se muestra la señal sin 50Hz ni las componente de baja frecuencia.

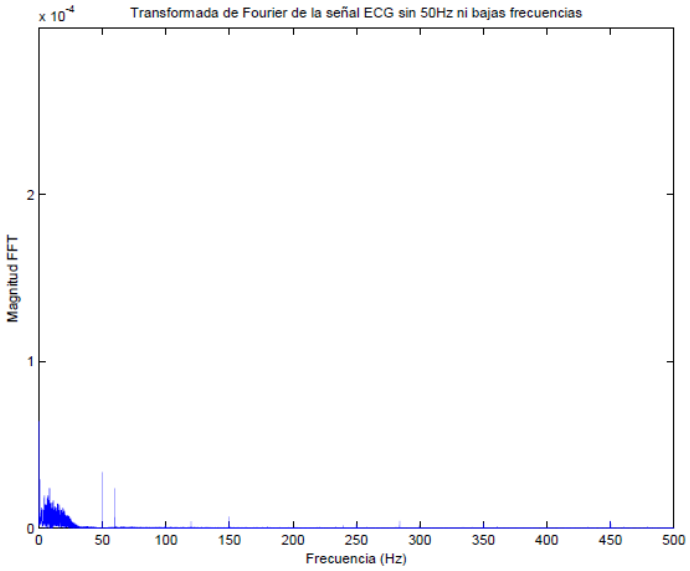


Figura-25: señal ECG sin 50Hz filtrada por el filtro pasa-altas.

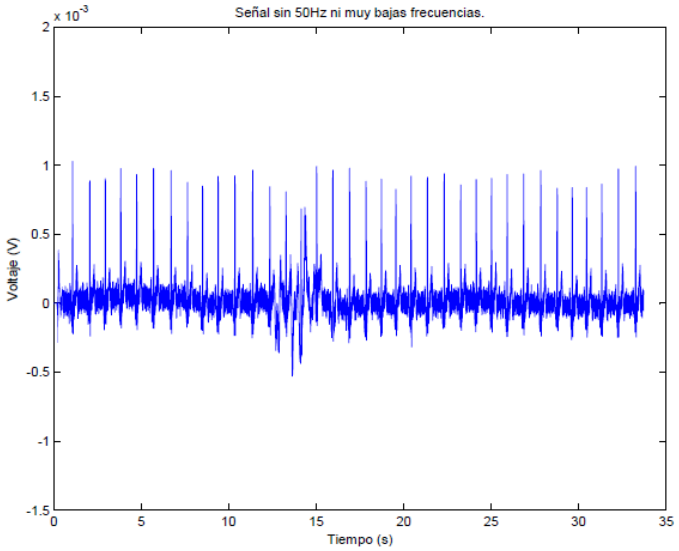


Figura-26: señal ECG sin componente frecuencial de 50 Hz ni DC.

3.3.3 Filtro pasa-baja

El último filtro que se va a diseñar es el filtro pasa-baja [12], [13]. Este filtro elimina el ruido de las componentes de alta frecuencia. La frecuencia de corte es 100Hz. En la Figura-25 se observa que la magnitud de las componentes frecuenciales a altas frecuencias es muy baja,

por ello se establece como condición que sean eliminadas completamente. Se diseña el filtro usando la ventana Hamming y un orden de 30. La ventana Hamming está representada en la Figura-7 del Capítulo 1.

Tras pasar la señal ECG sin 50Hz ni las componentes de baja frecuencia por el filtro pasa-baja, se realiza la transformada de Fourier de la señal de salida, ver Figura-27. En ella se observa que se consiguió eliminar las componentes de alta frecuencia. En la Figura-28 se muestra la señal filtrada.

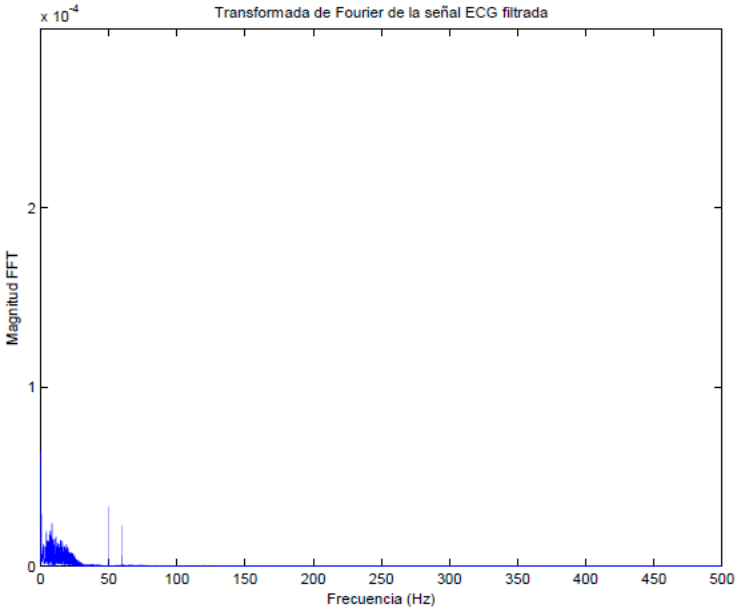


Figura-27: transformada de Fourier de la señal ECG filtrada.

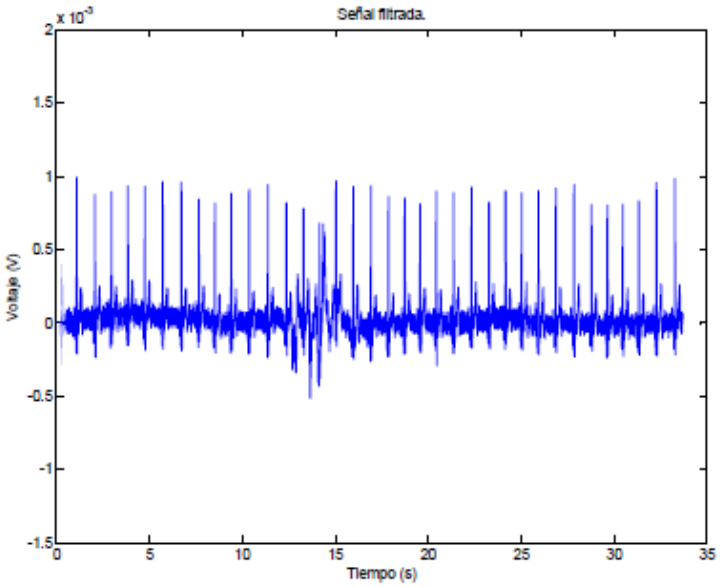


Figura-28: señal ECG filtrada.

En la Tabla-4 se muestran las características de los tres filtros diseñados y su respuesta en frecuencia. Se puede observar los efectos del enventanado, explicados en el Capítulo 1, al reducir el orden del filtro.

El siguiente paso una vez diseñados los filtros es la implementación en la FPGA. Para ello se seleccionará una arquitectura y luego se definirá el circuito para su posterior síntesis e implementación. Existen distintos métodos para definir el circuito y los más utilizados son la captura de esquemas y el lenguaje estándar de descripción de hardware VHDL (Very High Description Language).

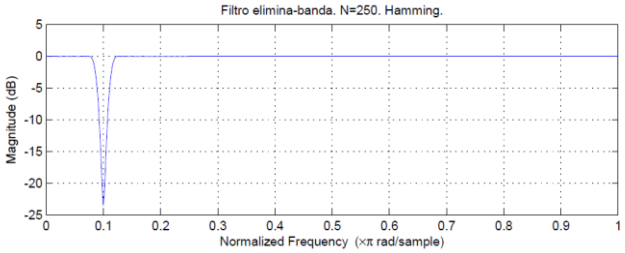
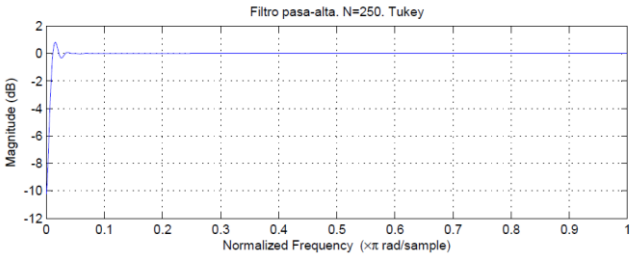
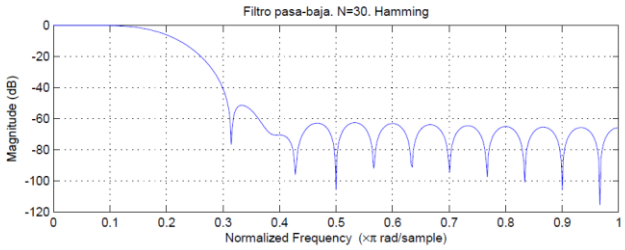
Filtro	Frecuencias de corte (Hz)	N	Ventana	Respuesta en frecuencia
Notch	45-55	250	Hamming	
Pasa-alta	1	250	Tukey	
Pasa-baja	100	30	Hamming	

Tabla-4: características de los filtros FIR diseñados.

CAPÍTULO 4: IMPLEMENTACIÓN DE FILTROS PARA SEÑALES ECG SOBRE FPGAS

En este capítulo se muestra la implementación del sistema de filtrado descrito en el Capítulo 3. En los siguientes apartados se explicarán en detalle las dos arquitecturas utilizadas, la arquitectura paralela y la arquitectura serie.

A continuación se va a mostrar la estructura del filtrado y a definir las señales del sistema, independientemente de la arquitectura con la que se esté trabajando.

4.1 DESCRIPCIÓN DEL SISTEMA DE FILTROS

En la Figura-29 se representa el diagrama de bloques del sistema de filtrado. Está compuesto por tres filtros FIR y dos registros. La utilidad de los dos registros es garantizar que el paso de datos de un filtro a otro se realiza correctamente. Se utilizan para sincronizar las señales y evitar retardos de propagación demasiado largos.

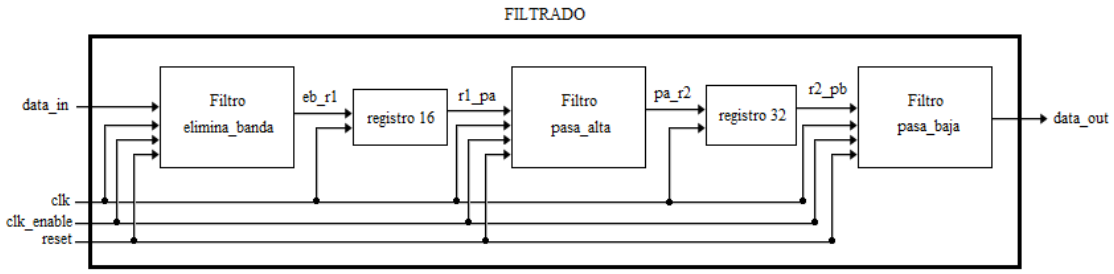


Figura-29: diagrama de bloques del sistema de filtrado.

En la Tabla-5 puede verse la longitud de palabra y el número de bits fraccionales de las señales que se observan en la Figura-29. La selección de dichos parámetros se ha hecho teniendo en cuenta el compromiso precisión/complejidad. Por un lado, la precisión será mayor cuando se trabaje con un mayor número de bits. Por otro lado, un exceso de bits aumenta innecesariamente el uso de recursos y se reduce la frecuencia de operación.

Filtro	Entrada	Salida
elimina_banda	16 (15 fraccionales)	16 (15 fraccionales)
pasa-alta	16 (15 fraccionales)	32 (31 fraccionales)
pasa-baja	32 (31 fraccionales)	32 (31 fraccionales)

Tabla-5: número de bits de las entradas y salidas de cada filtro.

Las señales de entrada y salida son las que se observan en la Figura-29:

- data_in (16 bits): señal ECG sin filtrar.

- clk: señal de reloj.
- clk_enable: señal de habilitación. Indica que hay un dato preparado para la recepción del sistema.
- reset: señal de reinicialización asíncrona.
- data_out (32 bits): señal ECG filtrada.

En el diagrama de bloques de la Figura-29 también se observan las señales internas del sistema:

- eb_r1 (16 bits): señal de salida del filtro elimina-banda y de entrada del registro de 16 bits.
- r1_pb (16 bits) señal de salida del registro interno de 16 bits y de entrada del filtro pasa-baja.
- pb_r2 (32 bits) señal de salida del filtro pasa-baja y de entrada del registro de 32 bits.
- r2_pa (32 bits) señal de salida del registro interno de 32 bits y de entrada del registro de 16 bits.

En la Tabla-6 se resume las señales del sistema. En los siguientes apartados de analizarán en profundidad dos arquitecturas, las arquitectura paralelo y la arquitectura serie.

NOMBRE DEL MÓDULO	ENTRADAS		SALIDAS
"Filtrado"	Clk clk_enable	Reset data_in (15 bits)	data_out (32 bits)
"elimina_banda"	Clk clk_enable	Reset data_in (15 bits)	eb_r1 (16 bits)
"registro16"	Clk	eb_r1 (16 bits)	r1_pb (16 bits)
"pasa_baja"	Clk clk_enable	Reset r1_pb (16 bits)	pb_r2 (32 bits)
"registro32"	Clk	pb_r2 (32 bits)	r2_pa (32 bits)
"pasa_alta"	Clk clk_enable	Reset r2_pa(32bits)	data_out (32bits)

Tabla-6: resumen de las señales del sistema

En el siguiente apartado se explicará en detalle las dos arquitecturas empleadas.

4.2 ARQUITECTURAS DE LOS FILTROS FIR

4.2.1 Arquitectura paralela

La arquitectura paralelo procesa todos los bits de cada palabra en paralelo. Los coeficientes están almacenados en memoria y los datos se van almacenando según van entrando. Supongamos que en un instante *n* han sido almacenados *N* datos en un vector *(x(n),x(n-1)…,x(2),x(1))* y en un vector columna están los *N* coeficientes del filtro.

Esta estructura realiza el producto vectorial de los dos vectores

$$y(n) = (x(n) \quad x(n-1) \quad \dots \quad x(2) \quad x(1)\,) \begin{pmatrix} b(0) \\ b(1) \\ \dots \\ b(N-2) \\ b(N-1) \end{pmatrix} \tag{4.1}$$

$$y(n) = b(0)x(n) + b(1)x(n-1) + \cdots + b(N-1)x(1) \tag{4.2}$$

Primero se realizan todos los productos de la expresión (4.2) y luego se suman. Tanto las multiplicaciones como las sumas se realizan mediante bloques DSP. El resultado obtenido es el mismo que si se desarrolla el sumatorio de la expresión (1.14) obtenida en el apartado *1.3 Filtros FIR*. En la Figura-30 se observa un esquema de la arquitectura.

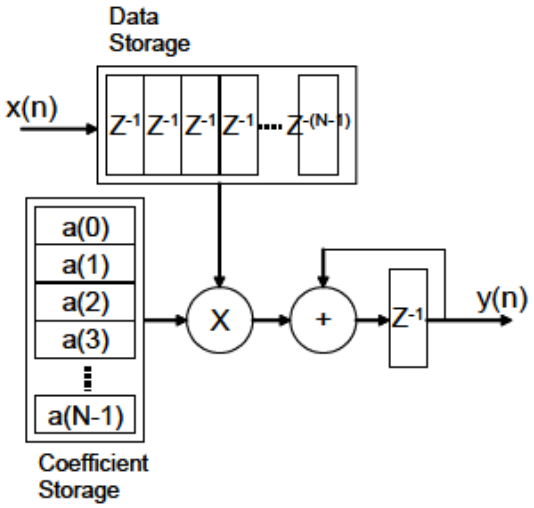


Figura-30: arquitectura paralela.

Esta arquitectura permite hacer uso de la simetría de los filtros. Se reduce el número de coeficientes a la mitad y por tanto el número de productos a realizar también. Se va a analizar la estructura de un filtro simétrico con orden par ya que así son los filtros que se han diseñado. De la expresión (4.2), teniendo en cuenta que el filtro es simétrico se obtiene

$$y(n) = b(0)[x(n) + x(1)] + b(1)[x(n-1) + x(2)] + \cdots + b(N/2)[x\left(\frac{N}{2}\right) + x\left(\frac{N}{2} - 1\right)] \quad (4.3)$$

Una arquitectura simétrica, Figura-31, realiza las sumas de los corchetes de (4.3) , luego multiplica por su correspondiente coeficiente y por último se suman.

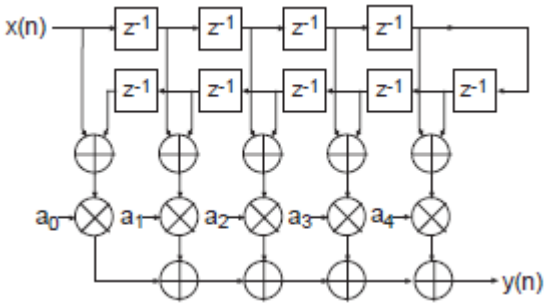


Figura-31: arquitectura simétrica.

4.2.2 Arquitectura serie DA (Distributed Arithmetic)

La arquitectura DA procesa todos los bits de cada palabra en serie. La arquitectura en serie no emplea DSPs, realiza el algoritmo mediante LUTs, sumadores y registros de desplazamiento.

El primer paso cuando entra un dato, en paralelo, es convertirlo en serie. Se realiza mediante un conversor paralelo-serie, PSC (Parallel-to-Serial Converter). Cada bit que sale del PSC se almacena en un registro de un bit. Vemos en la Figura-32 que los datos de entrada se van almacenando en cascada en los registros, siempre bit a bit. Al conjunto de registros que contienen los bits de los datos se le llama TSB (Time Skew Buffer). Los nodos de conexión del TSB son las entradas de la LUT. La LUT almacena los productos parciales de las entradas por los coeficientes y por ultimo en Scaling Accumulator se realizan la suma de los productos parciales.

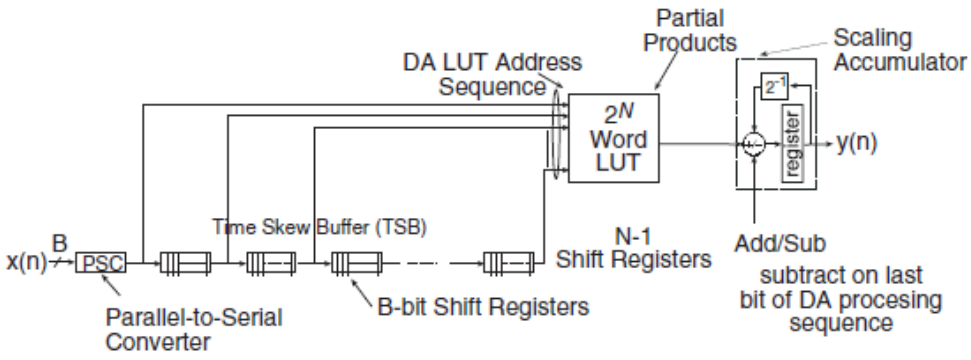


Figura-32: Arquitectura serie.

A continuación, se va a analizar el algoritmo que representa esta arquitectura. Suponiendo los coeficientes conocidos y que la variable de entrada *x(n)* puede escribirse como

$$x[n] = \sum_{b=0}^{B-1} x_b[n] \times 2^b \tag{4.4}$$

donde $x_b[n]$ representa el valor del bit b del número en representación binaria. Sustituyendo la expresión (4.4) en (1.14)

$$y[n] = \sum_{n=0}^{N-1} c[n] \sum_{b=0}^{B-1} x_b[n] \times 2^b \tag{4.5}$$

Desarrollando el sumatorio de (4.5)

$$\begin{aligned} y[n] = {} & c[0] \times (x_{B-1}[0]\, 2^{B-1} + x_{B-2}[0]\, 2^{B-2} + \cdots x_0[0]\, 2^0) \\ & + c[1] \times (x_{B-1}[1]\, 2^{B-1} + x_{B-2}[1]\, 2^{B-2} + \cdots x_0[1]\, 2^0) \ldots \\ & + c[N-1] \times (x_{B-1}[N-1]\, 2^{B-1} + x_{B-2}[N-1]\, 2^{B-2} + \cdots x_0[N-1]\, 2^0) \end{aligned} \tag{4.6}$$

Reorganizando los términos de (4.6)

$$\begin{aligned} y[n] = {} & 2^{B-1} \times (c[0]x_{B-1}[0] + c[1]x_{B-1}[1] + \cdots c[N-1]x_{B-1}[N-1]) \\ & + 2^{B-2} \times (c[0]x_{B-2}[0] + c[1]x_{B-2}[1] + \cdots c[N-1]x_{B-2}[N-1]) \ldots \\ & + 2^0 \times (c[0]x_0[0] + c[1]x_0[1] + \cdots c[N-1]x_0[N-1]) \end{aligned} \tag{4.7}$$

La expresión (4.7) puede expresarse de manera más compacta como

$$y[n] = \sum_{b=0}^{B-1} 2^b \sum_{n=0}^{N-1} c[n] \times x_b[n] \tag{4.8}$$

El segundo sumatorio de (4.8) puede realizarse mediante una LUT puesto que los valores de los coeficientes son conocidos y $x_b[n]$ sólo puede ser "0" o "1" y el producto de una potencia de dos no es más que un desplazamiento como se indica en la Figura-32.

En los siguientes apartados se analizará la implementación de cada una de ellas. Se mostrarán los recursos empleados y las simulaciones de cada arquitectura.

Las implementaciones se llevarán a cabo en el chip xc5vsx50T de la familia Virtex 5 cuyas características se han explicado en el apartado *2.3 Arquitectura de la familia Virtex 5*. Se empleará también el entorno de desarrollo ISE Design Suite 13.4 de la compañía Xilinx. Este entorno es una herramienta para la síntesis y el análisis de diseños de circuitos digitales que

permite al desarrollador compilar los diseños, realizar análisis temporales, examinar diagramas RTL, simular reacciones frente a distintos estímulos y configurar dispositivos.

4.3 IMPLEMENTACIÓN Y SIMULACIÓN DE LA ARQUITECTURA PARALELA

Para llevar a cabo la implementación de una arquitectura es necesario definir el circuito. Se hará mediante código VHDL. VHDL es el acrónimo que representa la combinación de VHSIC y HDL, donde VHSIC es el acrónimo de Very High Speed Integrated Circuit y HDL es a su vez el acrónimo de Hardware Description Language. Es un lenguaje definido por el IEEE (Institute of Electrical and Electronics Engineers) usado por ingenieros para describir circuitos digitales. Aunque puede ser usado para describir cualquier circuito digital se usa principalmente para programar PLDs, FPGAs y ASICs. Otros métodos para diseñar circuitos son la captura de esquemas (con herramientas CAD) y los diagramas de bloques, pero éstos no son prácticos en diseños complejos. La estructura del código consta de llamadas a librerías, una entidad (entity) que consiste en la declaración de las entradas y salidas de un módulo y la arquitectura (architecture) que es la descripción detallada de la estructura interna del módulo o de su comportamiento.

Para realizar la implementación, primero se genera el código VHDL de cada filtro mediante Matlab y se escribe el código de los dos registros. Cada uno de los filtros se implementa usando la arquitectura paralela. Luego se forma el sistema mediante instanciaciones de los componentes (los tres filtros y los dos registros) conectándolos como se indica en el diagrama de bloques de la Figura-29. En el *Anexo 1. Arquitectura paralela*, puede verse el código VHDL de mayor jerarquía.

4.3.1 Recursos y frecuencia de operación máxima

Después de realizar la síntesis y la implementación del diseño, se puede obtener la configuración RTL (Register Transfer Level) del diseño. En la Figura-33 se observan las señales de entrada y salida del sistema de filtrado de las señales ECG y puede verse el esquema RTL del sistema.

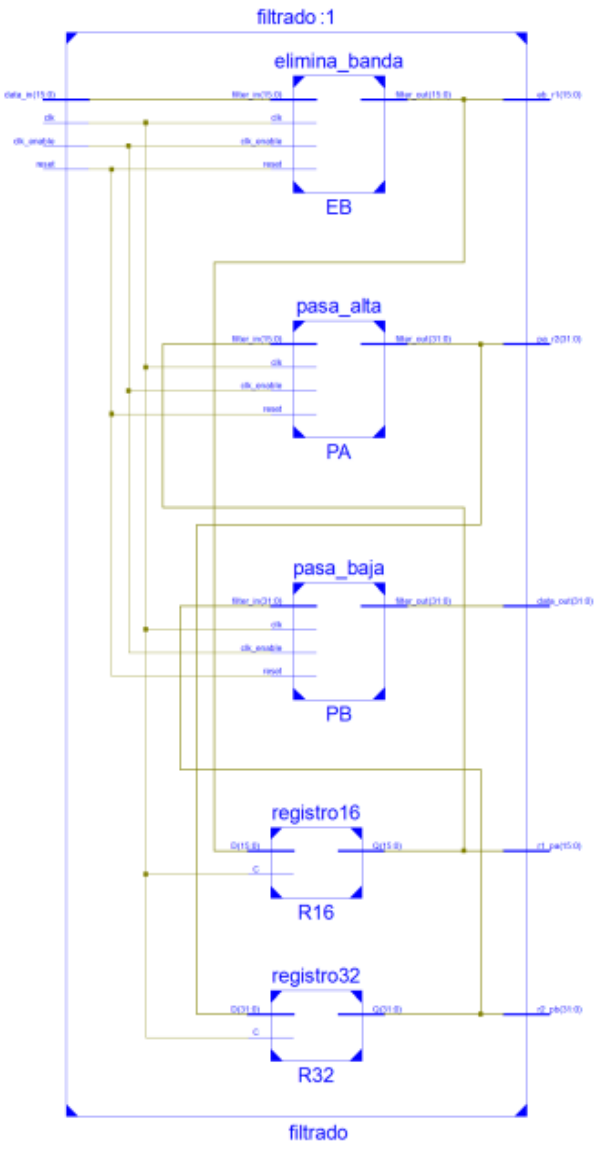


Figura-33: esquema RTL interno del sistema de filtrado.

También se obtiene de los informes elaborados por el entorno de diseño ISE, los recursos utilizados y la frecuencia de operación máxima. La Tabla-7 recoge los recursos utilizados en la FPGA.

Recurso	Utilizado	Disponible	Porcentaje
Registros	8.964	32.640	27%
LUT	13.250	32.640	40%
SLICE	4.752	8.160	58%
DSP	217	288	75%
Pines I/O	147	480	30%

Tabla-7: recursos empleados en la arquitectura paralela.

En el Capítulo 1 se explicó que al aumentar el orden de los filtros se aumentaba la carga computacional y en el apartado *4.2.1 Arquitectura paralela* se explicó que las operaciones se realizaban con DSPs, ambas ideas quedan reflejadas en el alto porcentaje de DSPs empleados.

El período mínimo admitido es de 111,243ns lo que supone una frecuencia de operación máxima igual a 8,989MHz.

Una vez estudiados los resultados de la síntesis y de la implementación hay que comprobar que el diseño realiza correctamente la tarea para la que se ha diseñado. Por ello se procede a realizar una simulación.

4.3.2 Simulación

Para realizar la simulación, teniendo en cuenta que el período mínimo admitido es 111,243ns, se estableció un periodo de muestreo de 120ns, es decir, una frecuencia de 8,333MHz.

La señal de entrada data_in de la simulación es la señal ECG que se empleó para hacer el diseño en el Capítulo 3. Se recuerda que se obtuvieron de una base de datos de señales físicas [11]. La señal de salida data_out está expresada en hexadecimal.

En la simulación realizada, ver Figura-34, se observa que:

- se transfiere un dato por período de reloj.
- la primera salida del primer filtro se da dos períodos después de la entrada del primer dato a los 660ns.
- la salida del primer dato del sistema se produce a los 2820ns. Teniendo en cuenta que el primer dato entra a 420ns se determina que el sistema tarda en filtrar un dato 2400ns, es decir, 20 períodos de reloj.

Por último, se comprobó que los datos de salida del filtro son correctos comparándose con los resultados obtenidos con Matlab en el Capítulo 3.

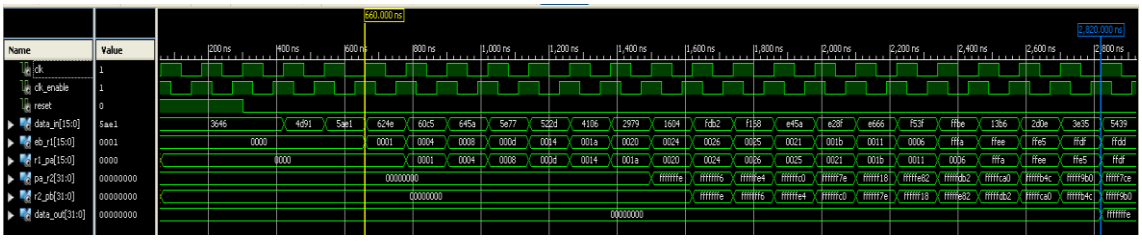


Figura-34: simulación del sistema de filtrado con la arquitectura paralela.

4.4 IMPLEMENTACIÓN Y SIMULACIÓN DE LA ARQUITECTURA SERIE

Para realizar la implementación, primero se genera el código VHDL de cada filtro mediante Matlab y se escribe el código de los dos registros. Cada uno de los filtros se implementa usando la arquitectura serie DA. Luego se forma el sistema mediante

instanciaciones de los componentes (los tres filtros y los dos registros) conectándolos como se indica en el diagrama de bloques de la Figura-29. En el *Anexo 2. Arquitectura serie*, puede verse el código VHDL de mayor jerarquía.

4.4.1 Recursos y frecuencia de operación máxima

Después de realizar la síntesis y la implementación del diseño, se puede obtener el esquema RTL. En este caso no se mostrará el esquema RTL puesto que es igual que en la arquitectura paralela, Figura-33. También se obtiene de los informes elaborados por el entorno de diseño ISE los recursos concretos utilizados y la frecuencia de operación máxima. La Tabla-8 recoge los recursos utilizados en la FPGA.

Recurso	Utilizado	Disponible	Porcentaje
Registros	1.821	32.640	5%
LUT	3.114	32.640	9%
SLICE	1.005	8.160	12%
DSP	0	288	0%
Pines I/O	147	480	30%

Tabla-8: recursos empleados en la arquitectura serie.

En el Capítulo 1 se explicó que al aumentar el orden de los filtros se aumentaba la carga computacional, esto queda reflejado en el elevado número de recursos empleados. Además, el uso de LUTs y no de bloques DSP, es característico de este tipo de arquitectura como se vio en el apartado *4.2.2 Arquitectura serie DA(Distributed Arithmetic)*.

El período mínimo admitido es de 12,028ns lo que supone una frecuencia de operación máxima igual a 83,139MHz.

Igual que para la arquitectura paralela, tras finalizar la síntesis y la implementación se realiza la simulación con el fin de comprobar que funciona correctamente.

4.4.2 Simulación

Para realizar la simulación, teniendo en cuenta que el período mínimo admitido es 12,028ns, se estableció un período de muestreo de 20ns, es decir, una frecuencia de 50MHz.

La señal de entrada data_in de la simulación es la señal ECG que se empleó para hacer el diseño en el Capítulo 3. Se recuerda que se obtuvieron de una base de datos de señales físicas [11]. La señal de salida data_out está expresada en hexadecimal.

En la simulación, ver Figura-35, se observa que:

- se transfiere un dato por 17 períodos de reloj.

- la primera salida del primer filtro se da 51 períodos después de la entrada del primer dato, a los 1110ns.
- la salida del primer dato del sistema se produce a los 8010ns. Teniendo en cuenta que el primer dato entra a 90ns se determina que el sistema tarda en filtrar un dato 7920ns, es decir, 396 períodos de reloj.

Los resultados obtenidos coinciden con Matlab y con los que resultan de la simulación de la arquitectura paralela. Por tanto la simulación se ha realizado correctamente.

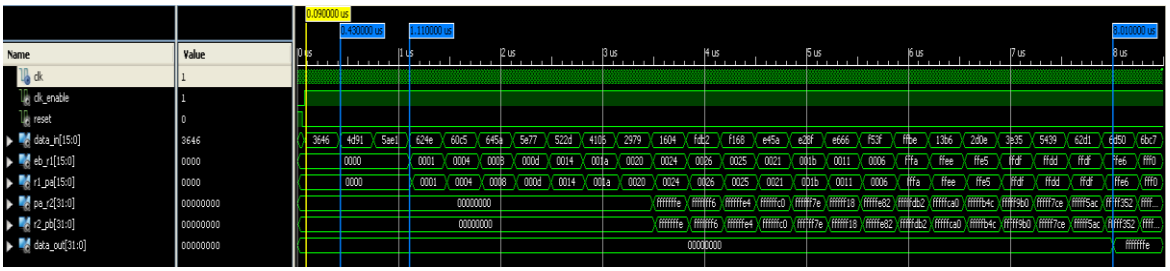


Figura-35 simulación del sistema de filtrado con la arquitectura serie.

Por último en el siguiente apartado se compararán las dos arquitecturas implementadas.

4.5 COMPARACIÓN DE ARQUITECTURAS

La Tabla-10 recoge un resumen de los resultados obtenidos en las dos arquitecturas. Por un lado se observa que la arquitectura serie emplea un menor número de recursos. Por tanto es de esperar que la arquitectura paralela consuma más energía. Además comprobamos que la FPGA posee los recursos necesarios para una implementación adecuada del sistema. Por otro lado, la arquitectura paralela, a pesar de tener una menor frecuencia de operación, es más rápida realizando el filtrado. Esto se debe a la propia arquitectura y al elevado número de DSPs empleados. En cualquier caso, se comprueba que el hardware seleccionado cumple la necesidad de realizar el procesado digital de señales en tiempo real.

		Arquitectura paralela	Arquitectura serie
RECURSOS	Registros	8.964	1.821
	LUT	13.250	3.114
	SLICE	4.752	1.005
	DSP	217	0
	Pines I/O	147	147
Frecuencia máxima de operación		8,989MHz	83,139MHz
Tiempo de filtrado		2400ns	7920ns

Tabla-10: comparativa de las arquitecturas paralela y serie.

CONCLUSIONES

En primer lugar, se han cumplido los dos objetivos principales de este trabajo, estos son: el estudio del procesado digital de señales usando dispositivos reconfigurables de alta velocidad (como son las FPGAs) y su aplicación a un ejemplo práctico. Concretamente, se han estudiado las características de los filtros FIR, permitiendo llevar a cabo el diseño y la implementación de una aplicación típica de instrumentación médica como es el procesado de las señales de electrocardiograma (ECG).

Se analizaron distintas arquitecturas hardware con el fin de poder realizar el procesado digital de señales en tiempo real. Se concluyó que las FPGAs son los dispositivos que mejor se adaptan a las necesidades de velocidad, área y consumo para realizar el procesado digital de señales, permitiendo realizar modificaciones una vez se ha implementado el diseño.

Se diseñaron tres filtros FIR para filtrar el ruido que contaminaba la señal de electrocardiograma. Fue importante establecer un compromiso entre el orden y la atenuación de los filtros. Se comprobó que al aumentar el orden de un filtro se reducía la zona de transición y el rizado de las bandas pasante y atenuada. Sin embargo el alto orden de los filtros elevaba la carga computacional. El uso de la simetría en las arquitecturas de los filtros fue la clave que permitió disminuir la carga computacional. De esta manera se consiguió reducir a la mitad los productos de los datos por los coeficientes del filtro.

Para poder implementar el diseño basado en la arquitectura paralela ha sido fundamental la elección de una FPGA con un elevado número de DSPs y una alta velocidad. En la implementación de la arquitectura paralela se vio el elevado número de recursos necesarios. El elevado porcentaje de recursos empleados limita la frecuencia de operación máxima.

La arquitectura serie DA emplea un número menor de recursos y la frecuencia de operación es mayor. Después de la implementación, se comprobó que esta arquitectura no emplea bloques DSPs, solo usa CLBs.

Comparando la arquitectura paralela y la arquitectura serie DA también se concluye que el uso de DSPs permite realizar operaciones como sumas y productos a una velocidad mayor.

Aunque se ha verificado el correcto funcionamiento mediante simulaciones se podría extender el trabajo realizando una simulación "hardware in the loop". Esta simulación consiste en verificar el correcto funcionamiento de la FPGA en tiempo real. Se puede realizar usando las herramientas del fabricante o mediante LabVIEW (Laboratory Virtual Instrumentation Engineering Workbench).

En resumen, el procesado digital de señales en dispositivos programables, como las FPGAs, permite realizar sistemas complejos con alta frecuencia de operación, reconfigurables, inmunes al ruido y de bajo consumo. Además, actualmente es posible desarrollar sistemas

completos en FPGAs denominados SOPCs (System on a Programmable Chip). Los SOPCs son la última evolución de las FPGAs, que permiten integrar junto con la lógica, uno o más microprocesadores (embebidos o configurados por el usuario), bloques de memoria y periféricos de entrada /salida, en un único chip.

BIBLIOGRAFÍA

[1] Steven W. Smith. *(*1997-1999). *The Scientist and Engineer's Guide to Digital Signal Processing*. [on line]. Disponible: http://www.analog.com/static/imported-files/tech_docs/dsp_book_frontmat.pdf

[2] Jhon G. PRoakis y Dimitris G. Manolakis, "Tratamiento digital de señales", Madrid, España: Pearson Prentice-Hall, 2007.

[3] Claude E. Shannon, "Communication in the Presence of Noise", Proceedings of the IRE, Volumen 37 , Enero, 1949.

[4] Ibone Lizarraga. Apuntes Control automático I.

[5] Marcelino Martinez Sober, Antonio J. Serrano Lopéz y Juan Gómez López. (2009-2010). *Introducción al procesado Digital de Señales.*[on line]. Disponible: http://ocw.uv.es/ingenieria-y-arquitectura/1-1/Course_listing

[6] M. Martínez, L. Gómez, A.J. Serrano, J. Villa y J. Gómez.(2009-2010). *3.- Diseño de filtros FIR.* [on line]. Disponible*:* http://ocw.uv.es/ingenieria-y-arquitectura/filtros-digitales/tema_3._diseno_de_filtros_fir.pdf

[7] Roger Woods, Jhon Mcallister, Ying Yi y Gaye Lightbody, "FPGA-based implementation of signal processing systems", Reino Unido: Willey 2008.

[8] Peter J. Ashenden, "Digital Design:An Embedded Systems Approach Using VHDL", California: Morgan Kaufmann, 2008.

[9] Jose L. Martín Gonzalez, "Electronica digital", Madrid, España: Delta publicaciones, 2007.

[10] Adel S. Sedra y Kenneth C. Smith, "Microelectronic Circuits" , Nueva York: Oxford University Press, 2010.

[11] Kevin Skahill, "VHDL for programmable logic " Massachusetts :Addison-Esley, 1996.

[12] Cristian Sisterna. *Field programmable gate arrays(FPGAS)*. [on line]. Disponible: dea.unsj.edu.ar/sisdig2/

[13] Web: www.xilinx.com

[14] Xilinx. (2012, Marzo, 16). *Virtex-5 FPGA User Guide*. [on line]. Disponible: http://www.xilinx.com/support/documentation/user_guides/ug190.pdf

[15] Xilinx. (2009, Febrero, 6). *Virtex-5 Family Overview Virtex.* [on line]. Disponible: http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf

[16] Xilinx. (2012, Enero, 26). *Virtex-5 FPGA XtremeDSP Design Considerations.User Guide*. [on line]. Disponible: http://www.xilinx.com/support/documentation/user_guides/ug193.pdf

[17] Web: http://www.electrocardiografia.es

[18] Guide Filter Design & Analysis Tool [on line]. Disponible: http://www.mathworks.es/es/help/signal/examples/introduction-to-the-filter-design-and-analysis-tool-fdatool.html

[19] Base de señales fisiológicas: www.physionet.org/physiobank/

[20] Bhumika Chandrakar, O.P.Yadav and V.K.Chandra, "A survey of noise removal techniques for ECG signals", IJARCE, Volumen 2, Marzo 2013.

[21] Geeta Kadam y P.C. Bhaskar, "Reduction of power line interference in ECG signal using FIR filter", IJCER, Volumen 2, Marzo-Abril 2012.

[22] Xilinx. (2011, March, 1). *IP LogiCORE FIR Compiler v5.0.* [on line]. Disponible: http://www.xilinx.com/support/documentation/ip_documentation/fir_compiler_ds534.pdf

ANEXO 1. ARQUITECTURA PARALELA

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity filtrado_FP is
    Port ( clk : in  STD_LOGIC;
           clk_enable : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           data_in : in  STD_LOGIC_VECTOR (15 downto 0);
           data_out : out  STD_LOGIC_VECTOR (31 downto 0);
                  eb_r1,r1_pa : inout STD_LOGIC_VECTOR (15
downto 0);
                  pa_r2,r2_pb : inout STD_LOGIC_VECTOR (31
downto 0));
end filtrado_FP;

architecture Behavioral of filtrado_FP is

     COMPONENT elimina_banda
     PORT(
          clk : IN std_logic;
          clk_enable : IN std_logic;
          reset : IN std_logic;
          filter_in : IN std_logic_vector(15 downto 0);
          filter_out : OUT std_logic_vector(15 downto 0)
          );
     END COMPONENT;

     COMPONENT registro16
     PORT(
          C : IN std_logic;
          D : IN std_logic_vector(15 downto 0);
          Q : OUT std_logic_vector(15 downto 0)
          );
     END COMPONENT;

     COMPONENT pasa_alta
     PORT(
          clk : IN std_logic;
          clk_enable : IN std_logic;
          reset : IN std_logic;
          filter_in : IN std_logic_vector(15 downto 0);
          filter_out : OUT std_logic_vector(31 downto 0)
          );
     END COMPONENT;

     COMPONENT registro32
     PORT(
          C : IN std_logic;
          D : IN std_logic_vector(31 downto 0);
          Q : OUT std_logic_vector(31 downto 0)
          );
     END COMPONENT;
```

```
	COMPONENT pasa_baja
	PORT(
		clk : IN std_logic;
		clk_enable : IN std_logic;
		reset : IN std_logic;
		filter_in : IN std_logic_vector(31 downto 0);
		filter_out : OUT std_logic_vector(31 downto 0)
		);
	END COMPONENT;


begin

	EB: elimina_banda PORT MAP(
		clk => clk ,
		clk_enable => clk_enable,
		reset => reset,
		filter_in =>data_in ,
		filter_out => eb_r1
	);

	R16: registro16 PORT MAP(
		C => clk,
		D => eb_r1,
		Q => r1_pa
	);

	PA: pasa_alta PORT MAP(
		clk => clk,
		clk_enable => clk_enable,
		reset => reset,
		filter_in => r1_pa,
		filter_out => pa_r2
	);

	R32: registro32 PORT MAP(
		C => clk,
		D => pa_r2,
		Q => r2_pb
	);

	PB: pasa_baja PORT MAP(
		clk => clk,
		clk_enable => clk_enable ,
		reset => reset ,
		filter_in => r2_pb ,
		filter_out => data_out
	);


end Behavioral;
```

ANEXO 2. ARQUITECTURA SERIE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity filtrado_serie is
    Port ( clk : in  STD_LOGIC;
           clk_enable : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           data_in : in  STD_LOGIC_VECTOR (15 downto 0);
           data_out : out  STD_LOGIC_VECTOR (31 downto 0);
                  eb_r1,r1_pa : inout STD_LOGIC_VECTOR (15
downto 0);
                  pa_r2,r2_pb : inout STD_LOGIC_VECTOR (31
downto 0));
end filtrado_serie;

architecture Behavioral of filtrado_serie is

     COMPONENT elimina_bandaS
     PORT(
          clk : IN std_logic;
          clk_enable : IN std_logic;
          reset : IN std_logic;
          filter_in : IN std_logic_vector(15 downto 0);
          filter_out : OUT std_logic_vector(15 downto 0)
          );
     END COMPONENT;

     COMPONENT registro16
     PORT(
          C : IN std_logic;
          D : IN std_logic_vector(15 downto 0);
          Q : OUT std_logic_vector(15 downto 0)
          );
     END COMPONENT;

     COMPONENT pasa_altaS
     PORT(
          clk : IN std_logic;
          clk_enable : IN std_logic;
          reset : IN std_logic;
          filter_in : IN std_logic_vector(15 downto 0);
          filter_out : OUT std_logic_vector(31 downto 0)
          );
     END COMPONENT;

     COMPONENT registro32
     PORT(
          C : IN std_logic;
          D : IN std_logic_vector(31 downto 0);
          Q : OUT std_logic_vector(31 downto 0)
          );
```

```
	END COMPONENT;

	COMPONENT pasa_bajaS
	PORT(
		clk : IN std_logic;
		clk_enable : IN std_logic;
		reset : IN std_logic;
		filter_in : IN std_logic_vector(31 downto 0);
		filter_out : OUT std_logic_vector(31 downto 0)
		);
	END COMPONENT;


begin
	EB: elimina_bandaS PORT MAP(
		clk => clk ,
		clk_enable => clk_enable,
		reset => reset,
		filter_in =>data_in ,
		filter_out => eb_r1
	);

	R16: registro16 PORT MAP(
		C => clk,
		D => eb_r1,
		Q => r1_pa
	);

	PA: pasa_altaS PORT MAP(
		clk => clk,
		clk_enable => clk_enable,
		reset => reset,
		filter_in => r1_pa,
		filter_out => pa_r2
	);

	R32: registro32 PORT MAP(
		C => clk,
		D => pa_r2,
		Q => r2_pb
	);

	PB: pasa_bajaS PORT MAP(
		clk => clk,
		clk_enable => clk_enable ,
		reset => reset ,
		filter_in => r2_pb ,
		filter_out => data_out
	);


end Behavioral;
```