UNIVERSIDAD POLITÉCNICA DE VALENCIA

ESCUELA POLITÉCNICA SUPERIOR DE GANDÍA

I.T.TELECOMUNICACIÓN(SISTEMAS ELECTRÓNICOS)

"DISEÑO E IMPLEMENTACIÓN SOBRE FPGA, DE UN FILTRO NOTCH SINTONIZABLE Y CONTROLADO DESDE PC"

TRABAJO FINAL DE CARRERA

Autor/es:
D. Fermín Raimundo Medina

Director/es:
Dña. Mª Asunción Pérez Pascual

***GANDÍA*, 2010**

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN Y OBJETIVOS:

Muchas son las ocasiones en las que se hace necesario el empleo de filtros NOTCH, también llamados de hendidura, en el campo de la ingeniería electrónica. Dichos filtros permiten eliminar una banda de frecuencias muy estrecha, de una señal con un espectro determinado.

Así por ejemplo, pueden utilizarse filtros NOTCH para eliminar el ruido de red de 50 Hz, presente en una débil señal de electrocardiograma (ECG), o para eliminar el zumbido de baja frecuencia que se genera al reproducir una cinta de cassette de audio, o para cancelar una señal de radiofrecuencia que interfiere en otra que se quiere sintonizar, por citar algunos ejemplos.

Por otro lado, el avance que ha sufrido la tecnología CMOS en los últimos años, así como el desarrollo de nuevas técnicas de procesado digital de la señal, permiten hoy día la implementación de circuitos digitales muy complejos, en dispositivos de reducido tamaño tales como FPGA's, ASIC's o DSP's.

En el presente trabajo final de carrera, se va a diseñar e implementar sobre FPGA, un filtro NOTCH para aplicaciones de audio. Dicho filtro será configurable, de modo que el usuario podrá seleccionar desde un PC la frecuencia central del filtro, así como elegir entre cuatro anchuras pre-configuradas. La interfaz entre el PC y el usuario será una aplicación desarrollada en lenguaje C, y la información de configuración del filtro, se transmitirá hasta la placa que contiene la FPGA a través del puerto paralelo.

La placa utilizada para el proyecto será la denominada XtremeDSP Development Kit–II, de Xilinx.

2. EL ALGORITMO. SIMULACIÓN EN MATLAB:

Para la implementación del filtro NOTCH, se ha optado por un algoritmo adaptativo, cuya estructura se presenta en la siguiente figura :

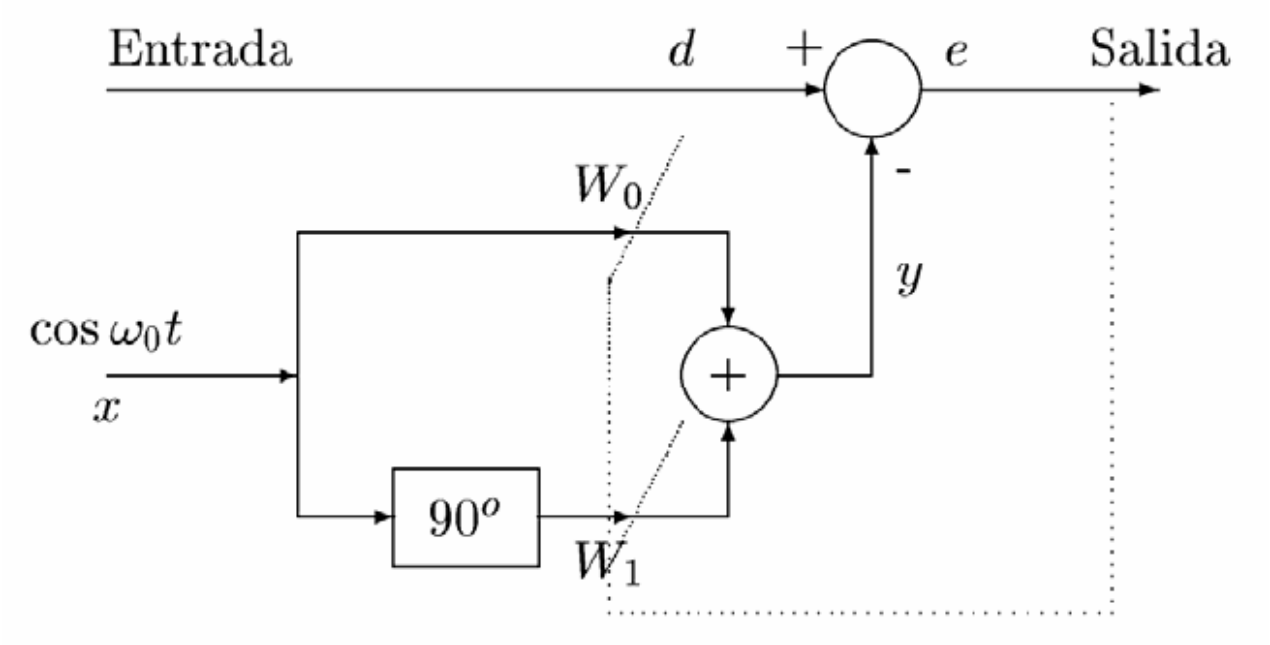

Fig. 2. 1 Esquema del filtro adaptativo para el filtro Notch

Como podemos observar, el filtro consta de dos coeficientes, cada uno de los cuales está multiplicado por un seno y por un coseno respectivamente, de frecuencia igual a la frecuencia que se desea eliminar de la señal de entrada. El seno/coseno será generado en la FPGA mediante un bloque DDS (Síntesis Digital Directa). La combinación del seno y el coseno permite al filtro adaptativo estimar la fase y amplitud del tono que se desea eliminar de la señal de entrada, de modo que los coeficientes van poco a poco adaptándose, hasta conseguir anular dicha frecuencia en la señal de error.

En este algoritmo en que tenemos dos coeficientes, podemos representar la potencia media de la señal de error (o error cuadrático medio, MSE), en función del valor de los coeficientes:

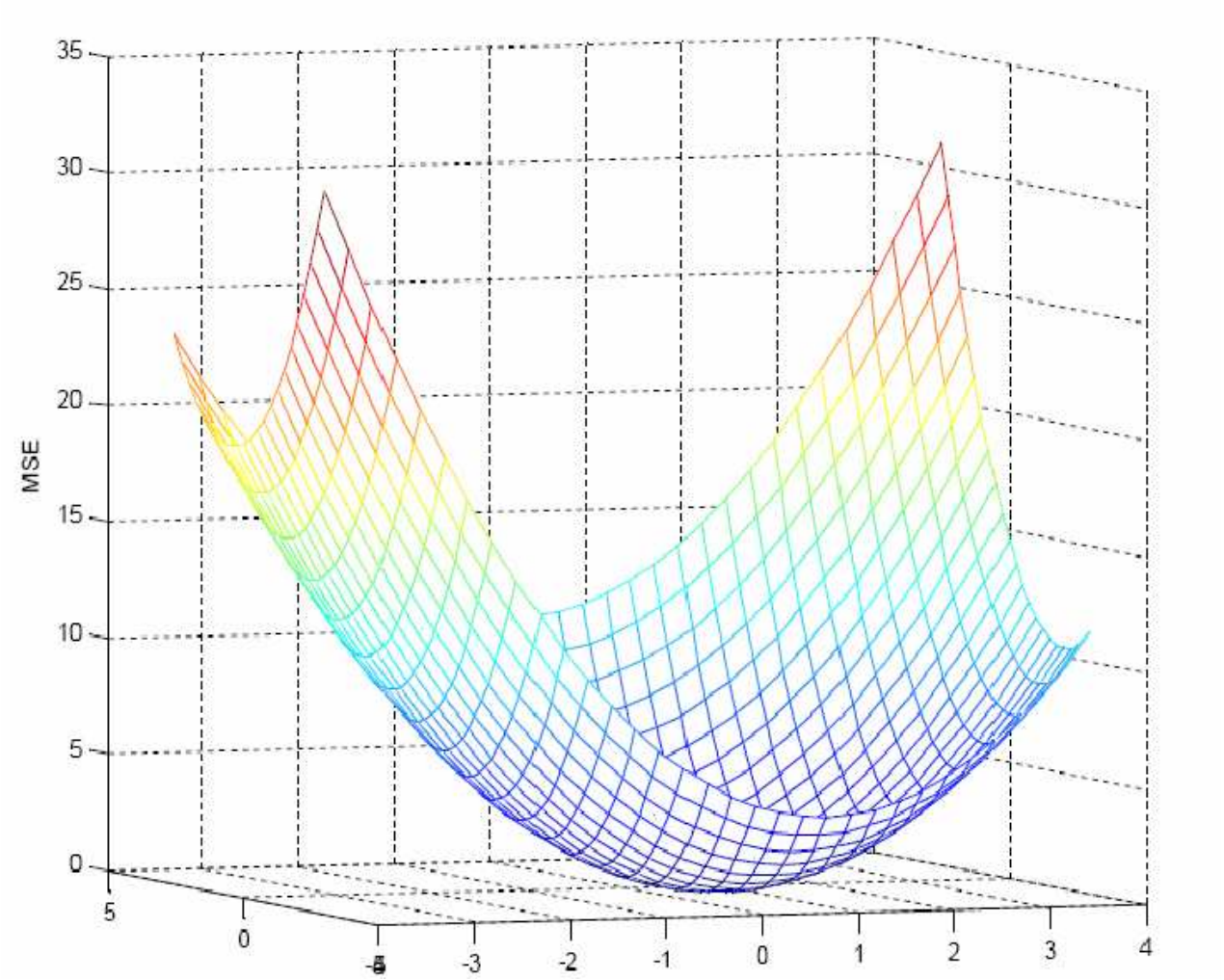

Fig. 2. 2 MSE en función de los coeficientes

Como ya se ha dicho, se trata de llegar al mínimo de la curva anterior, y el tiempo que se tarda en llegar hasta él, depende de un parámetro denominado coeficiente de adaptación μ. Así pues, cuanto mayor sea μ, más rápida será la adaptación. Sin embargo, la potencia del error nunca llega a anularse del todo, oscilando en torno al mínimo, y siendo mayor la amplitud de esas oscilaciones, cuanto mayor es el valor de μ. Por tanto, en el diseño de cualquier filtro adaptativo, habrá que llegar una solución de

compromiso entre rapidez de adaptación y error cuadrático medio residual, una vez se ha llegado a la adaptación.

Para simular el algoritmo en MATLAB, podemos teclear las siguientes instrucciones:

```
L=100;
n=1:L;
fase = pi/3;
frec = 0.05;
d=2*cos(2*pi*n*frec+fase);
x1=cos(2*pi*n*frec);
x2=sin(2*pi*n*frec);
mu=0.1;
w1=0;
w2=0;
y=zeros(L,1);
e=zeros(L,1);
for k=1:L,
y(k)=w1(k)*x1(k)+w2(k)*x2(k);
e(k)=d(k)-y(k);
w1(k+1)=w1(k)+2*mu*e(k)*x1(k);
w2(k+1)=w2(k)+2*mu*e(k)*x2(k);
end
```

Podemos observar, que la señal de entrada tiene la misma frecuencia que las señales seno y coseno, pero distinta amplitud y fase. También vemos que el valor inicial que se ha dado a los pesos es nulo, y que la adaptación se va consiguiendo gracias a un bucle de 100 iteraciones.

El valor que se ha dado al coeficiente de adaptación es de 0’1. Ya se ha comentado la importancia de dicho coeficiente para el diseño de cualquier filtro adaptativo, pero aún es más importante si cabe para nuestro algoritmo, ya que de él va a depender la anchura final del filtro.

A continuación se presentan una serie de gráficas obtenidas en MATLAB, con el propósito de probar el algoritmo para diferentes frecuencias y para

diferentes valores del coeficiente de adaptación. Como señal de entrada se ha tomado una señal delta, cuyo espectro es plano (contiene todas las frecuencias por igual), por lo que resulta idónea para las simulaciones, ya que de este modo, a la salida obtendremos la respuesta en frecuencia del filtro. En el encabezamiento de cada gráfica podemos ver cuál es la frecuencia que se ha querido eliminar, así como la anchura del filtro, que se corresponde con los siguientes valores de μ:

a. Superestrecha → μ = 0.005
b. Estrecha → μ = 0.05
c. Media → μ = 0.2
d. Ancha → μ = 0.4

Estos valores de μ son solo válidos para estas simulaciones. El valor que se ha elegido finalmente para cada una de las anchuras, se ha tomado en función de las simulaciones hechas en System Generator.

Por último indicar que en el anexo 1 se puede consultar el código de MATLAB utilizado para la obtención de estas gráficas:

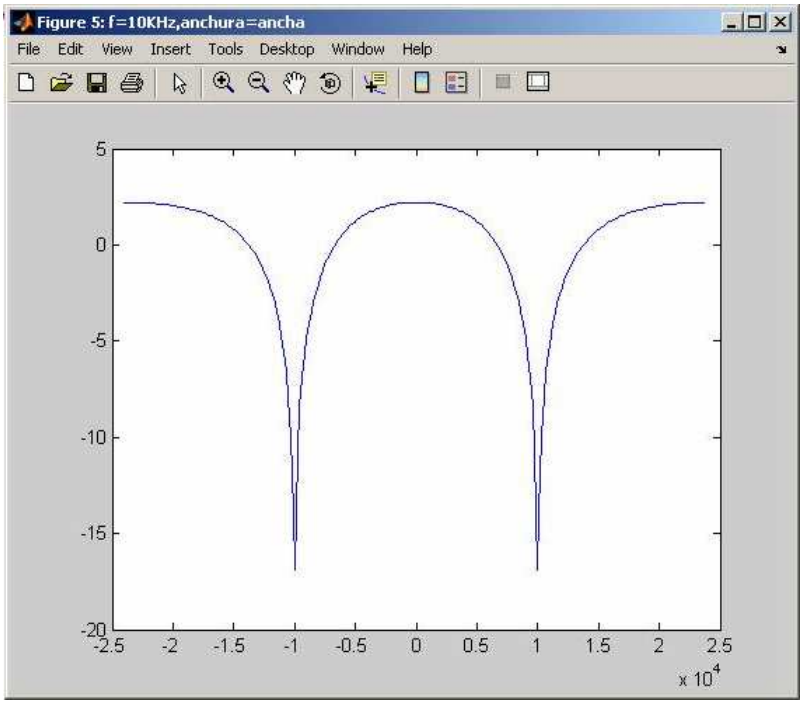

Fig. 2. 3 Respuesta en frecuencia para f = 10 KHz y anchura = ancha

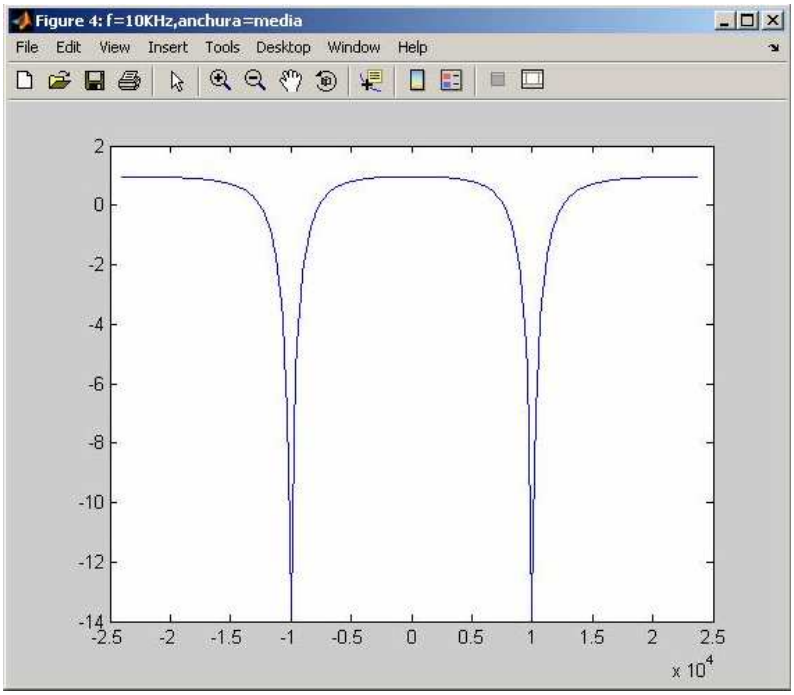

Fig. 2. 4 Respuesta en frecuencia para f = 10 KHz y anchura = media

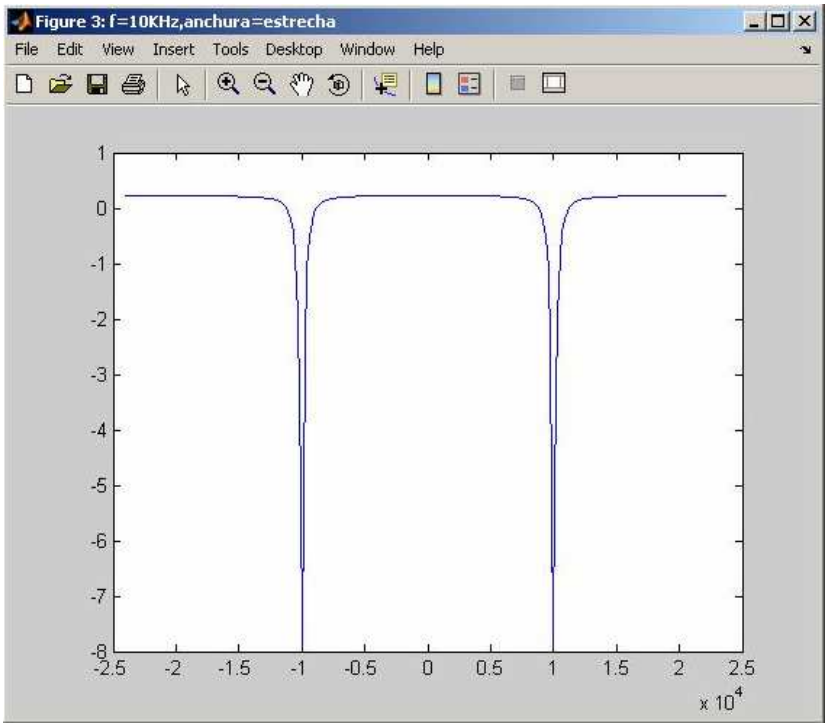

Fig. 2. 5 Respuesta en frecuencia para f = 10 KHz y anchura = estrecha

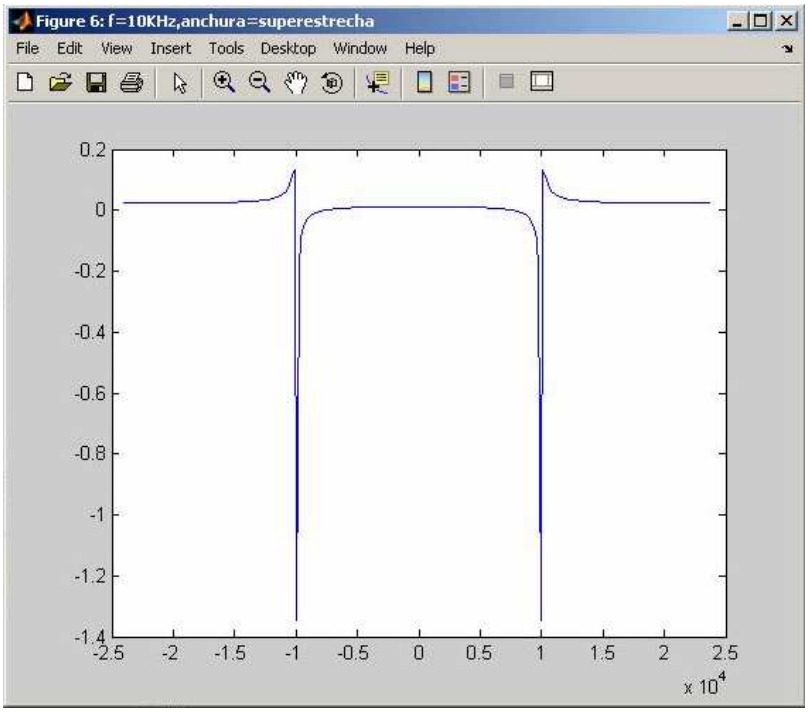

Fig. 2. 6 Respuesta en frecuencia para f = 10 KHz y anchura = superestrecha

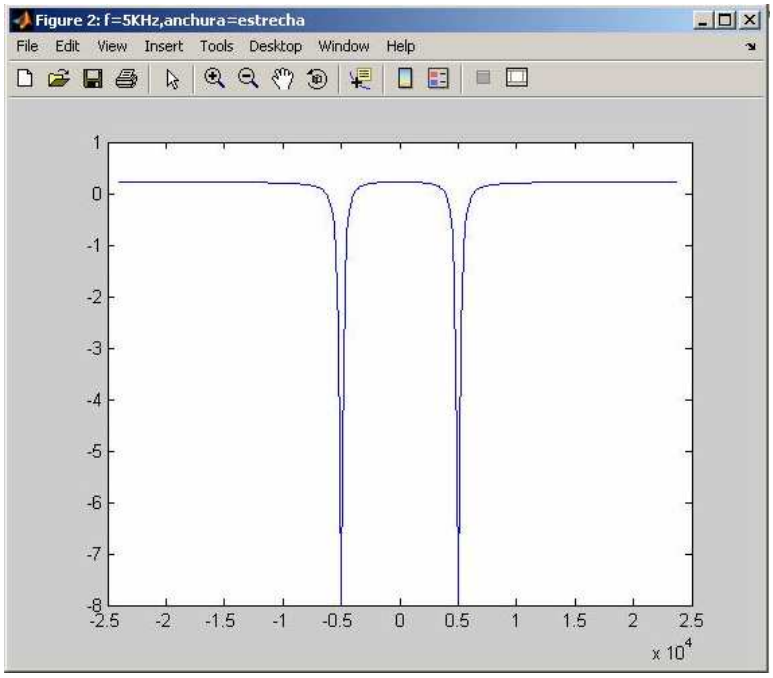

Fig. 2. 7 Respuesta en frecuencia para f = 5 KHz y anchura = estrecha

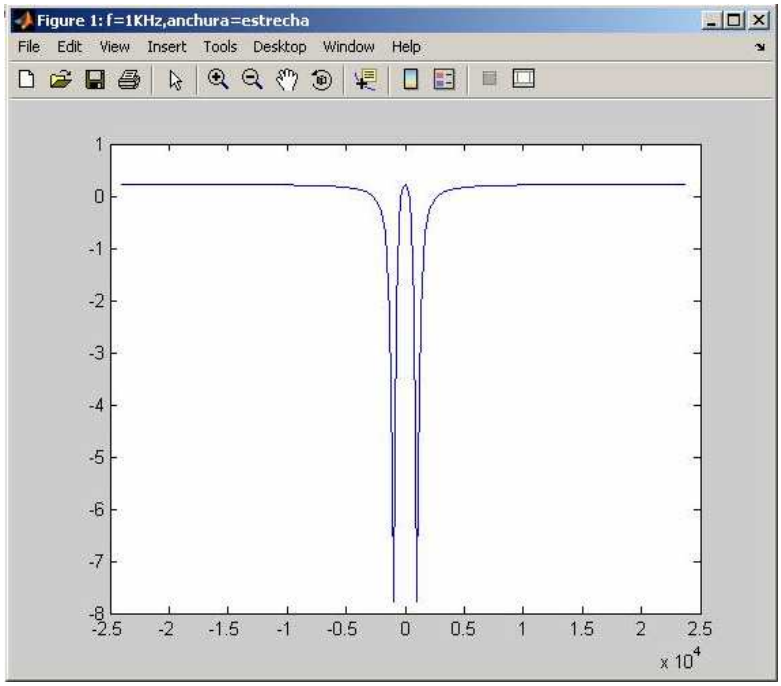

Fig. 2. 8 Respuesta en frecuencia para f = 1 KHz y anchura = estrecha

A la vista de los resultados, se puede concluir que el algoritmo funciona perfectamente, si bien se observa que a menor valor del coeficiente de adaptación, anchura más estrecha, pero también menor atenuación. Además, en el caso en que se selecciona anchura superestrecha, se produce una ligera distorsión en la salida, aunque no parece que sea demasiado problemático.

3. CARACTERÍSTICAS DE LA PLACA XTREMEDSP DEVELOPMENT KIT-II:

En las siguientes figuras se muestra la caja protectora que contiene la placa XtremeDSP, y la propia placa, con una descripción de sus partes más importantes. Obsérvese que la placa contiene conversores analógicos – digitales, y digitales – analógicos, así como dos FPGA's, una de usuario, para programar la función que se desea realizar, y otra para generar la señal de reloj de todo el sistema.

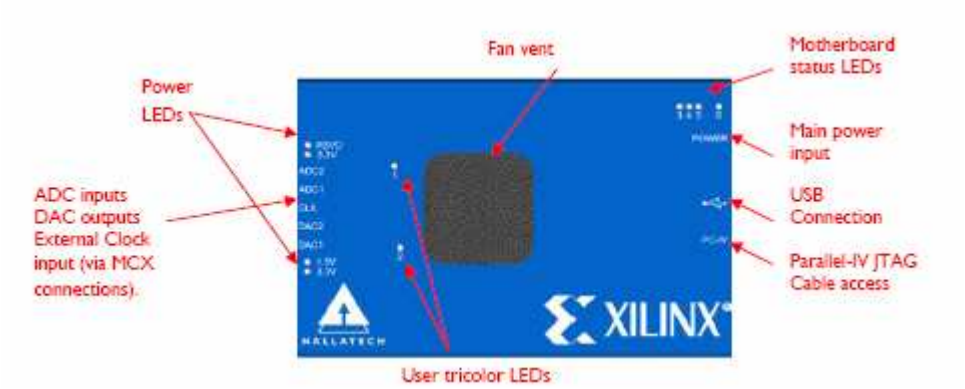

Fig. 3. 1 Caja vista frontal

Fig. 3. 2 Caja lado izquierdo

Fig. 3. 3 Caja lado derecho

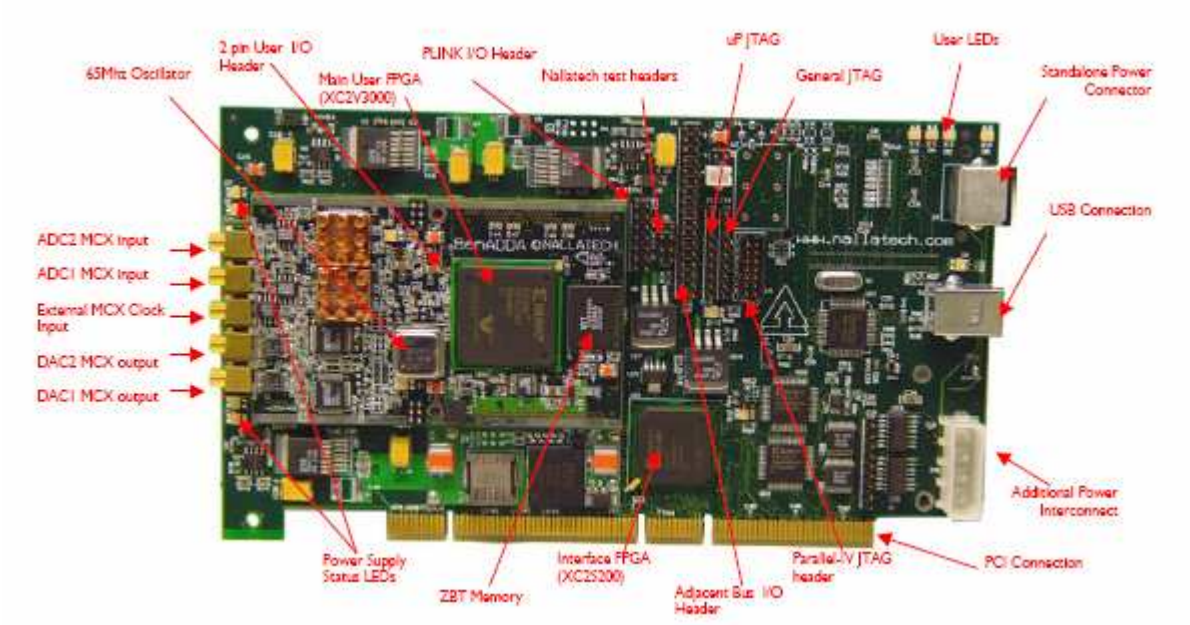

Fig. 3. 4 Vista frontal de la placa

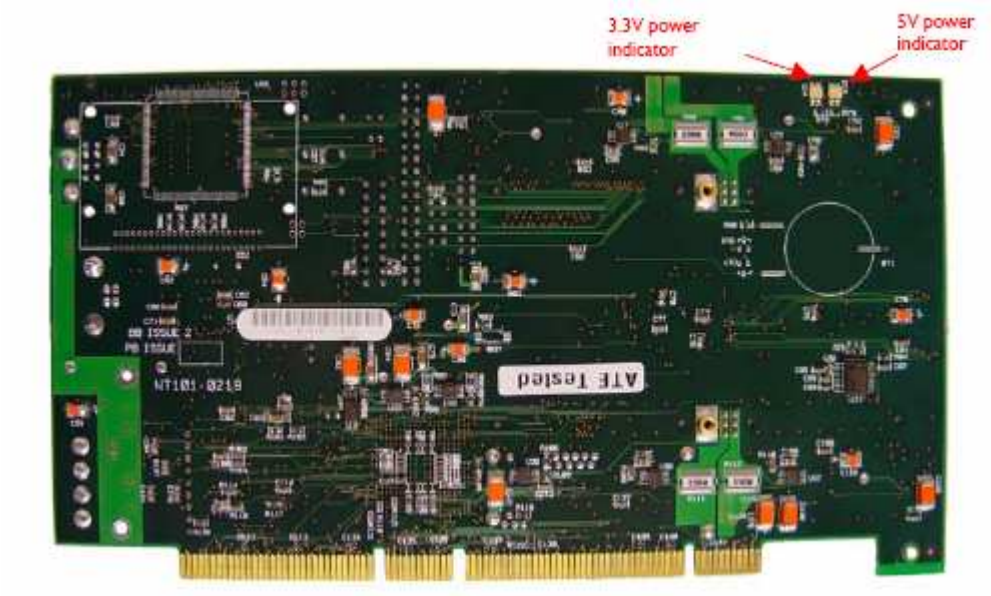

Fig. 3. 5 Vista posterior de la placa

A continuación, se detallan las características técnicas más importantes de la placa :

- BenONE
  - Placa base que soporta el modulo DIME-II
  - FPGA Spartan-II con interface PCI o USB
  - LEDs de estado
  - Pines JTAG de configuración

- BenADDA DIME-II
  - FPGA Virtex-II XC2V3000-4FG676
  - 2 canales ADC AD6644 (14 bits, 65Msps)
  - 2 canales DAC AD9772 (14 bits, 65Msps)
  - Entrada de oscilador externo
  - Banco de Memoria ZBT-SRAM (133MHz, 512x16bits)
  - LEDs de estado

En la siguiente figura se muestra la interface entre un ADC y la FPGA :

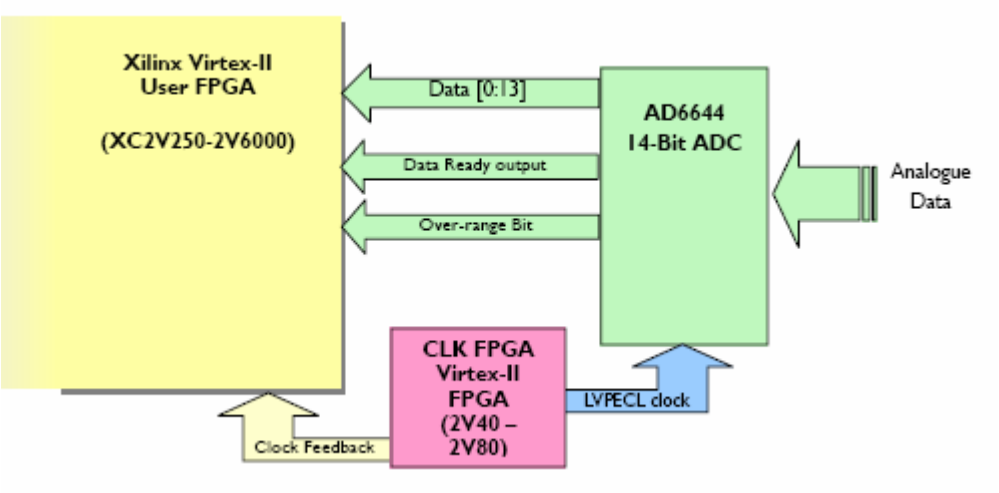

Fig. 3. 6 Interface entre un ADC y la FPGA

Siendo sus características más importantes:

- 14 bits en complemento a dos
- Velocidad de muestreo 65Msps
- Zin = 50 ohm
- Filtro ajustable de 3er orden anti-aliasing (frecuencia de corte por defecto 34.5MHz)
- Rango de entrada 2 Vpp o ± 1 V
- Reloj interno de 65MHz o externo >65MHz
- SNR 74.5 dB

En la siguiente figura se muestra el bloque ADC utilizado en System Generator:

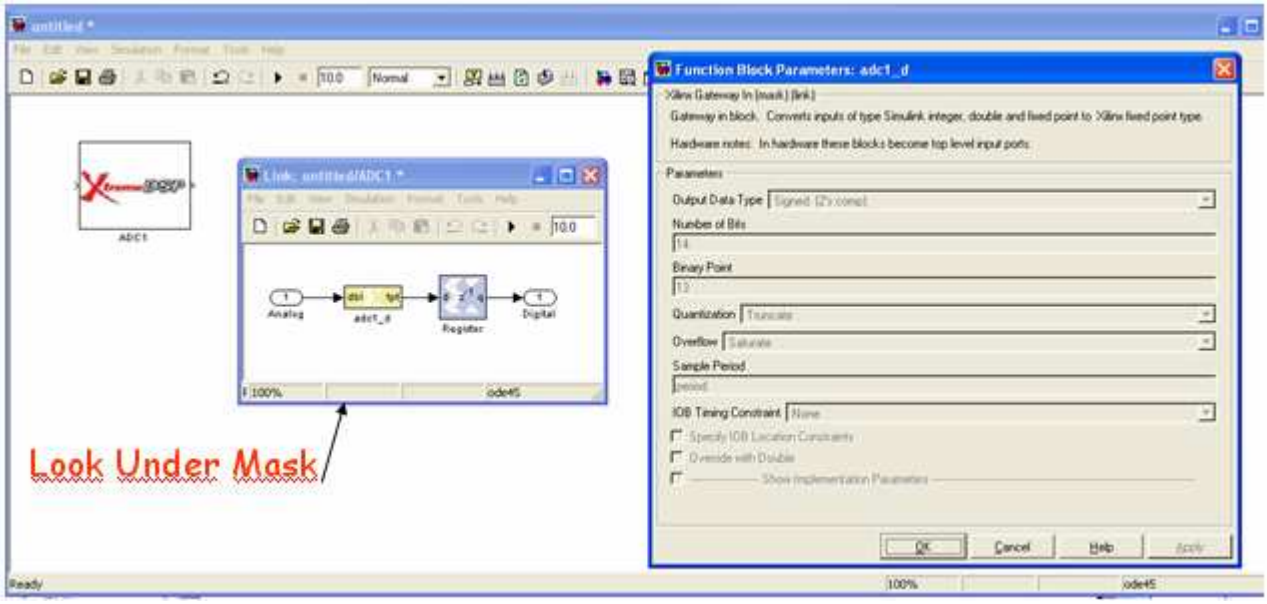

Fig. 3. 7 Bloque ADC en System Generator

En la siguiente figura se muestra la interface entre un DAC y la FPGA:

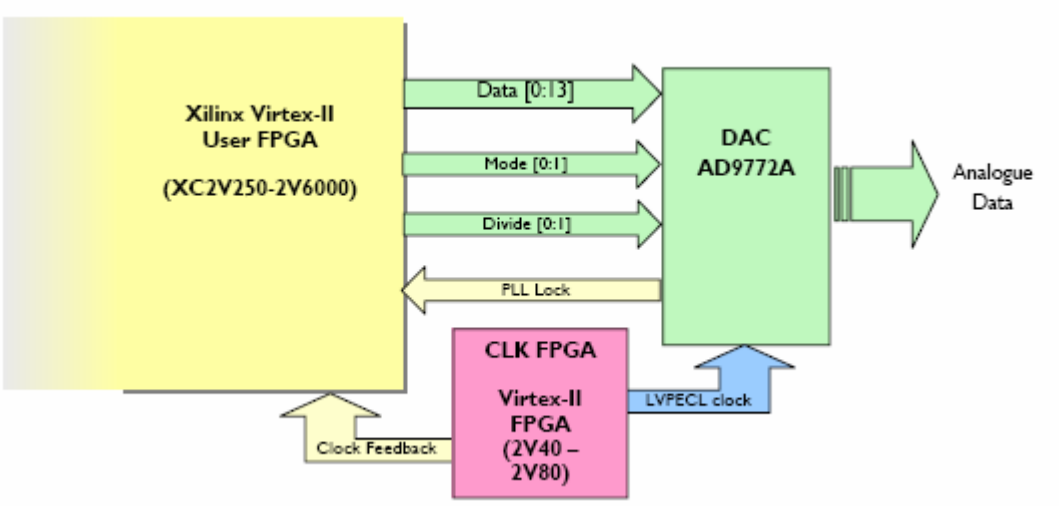

Fig. 3. 8 Interface entre un DAC y la FPGA

Siendo sus características más importantes:

- 14 bits en complemento a dos
- Velocidad de conversión 160Msps

- Zout = 50 ohm
- Filtro interpolador de salida programado a través de MOD0 MOD1
- PLL interno
- La entrada está en formato offset-binary

En la siguiente figura se muestra el bloque DAC utilizado en System Generator:

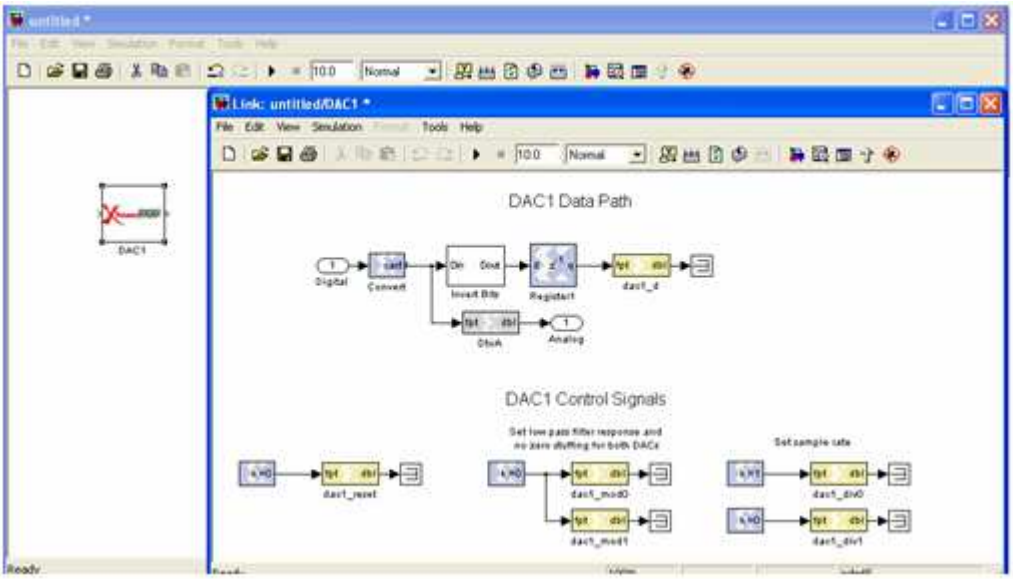

Fig. 3. 9 Bloque DAC de System Generator

Los DAC's pueden ser configurados para trabajar en diferentes modos de operación, según las siguientes tablas proporcionadas por el fabricante:

Digital Mode	MOD0	MOD1	Digital Filter	Zero-Stuffing
Baseband	0	0	LOW	NO
Baseband	0	1	LOW	YES
Direct IF	1	0	HIGH	NO
Direct IF	1	1	HIGH	YES

Input Data Rate (MSPS)	MOD1	DIV1	DIV0	Zero-stuffing	Divide-by-N-ratio
24-100	1	0	0	Yes	1
12-50	1	0	1	Yes	2
6-25	1	1	0	Yes	4
3-12.5	1	1	1	Yes	8

Fig. 3. 10 Configuración modos de operación de los DAC

En el próximo apartado se explicará que es necesario transmitir doce señales digitales para configurar el filtro NOTCH. Estas doce señales se introducirán en la FPGA por el conector JTAG J10 de 14 pines, que se muestra y describe a continuación:

Header Pin Number	Name	User FPGA (2V3000FG676) PIN No
1	PP0LK<0>	E13
2	PP0LK<1>	F13
3	PP0LK<2>	H13
4	PP0LK<3>	G13
5	PP0LK<4>	C12
6	PP0LK<5>	D12
7	PP0LK<6>	E12
8	PP0LK<7>	F12
9	PP0LK<8>	H12
10	PP0LK<9>	G12
11	PP0LK<10>	E7
12	PP0LK<11>	E6
13	GND	N/A
14	GND	N/A

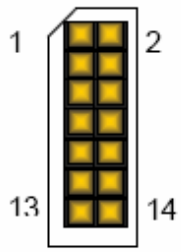

Fig. 3. 11 Descripción del conector JTAJ J10

En la siguiente figura se muestra como se genera y distribuye la señal de reloj por el sistema:

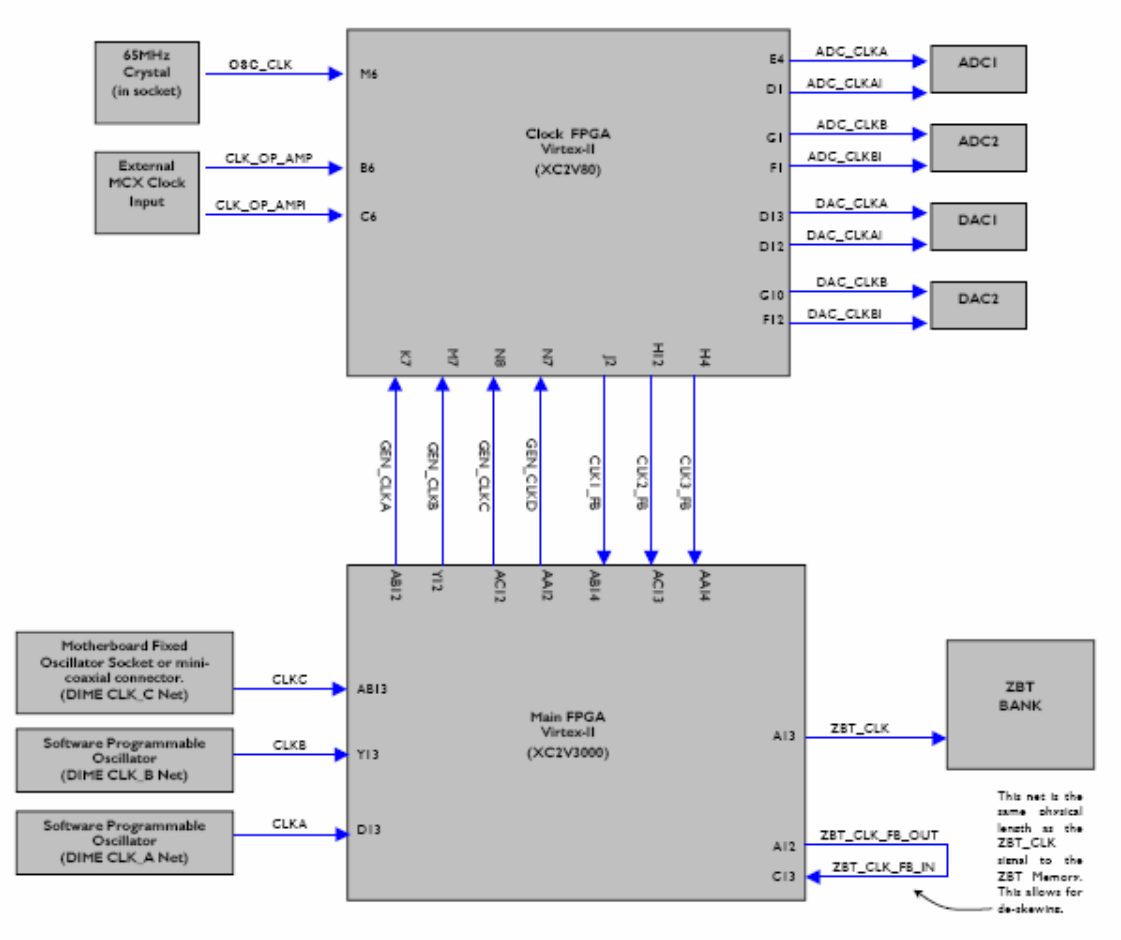

Fig. 3. 12 Distribución de la señal de reloj

El oscilador interno que incorpora la placa es de 65 MHz. La FPGA que proporciona señal de reloj a partir del oscilador interno, es la XC2v80, que se programa con el archivo osc_clk_2v80.bit proporcionado por el fabricante.

Si se desea utilizar una frecuencia de reloj inferior, se puede compilar el archivo osc_clock.vhd, para obtener el .bit correspondiente con el que programar la FPGA XC2v80. El archivo osc_clock.vhd contiene dos variables, M (definida entre 2 y 32) y N (definida entre 1 y 32). La frecuencia de reloj resultante es (M/N) * 65 MHz. En este proyecto se ha utilizado una señal de reloj de (4/13) * 65 MHz = 20 MHz. En el anexo 3 puede verse el archivo osc_clock.vhd empleado.

También es posible utilizar una señal de reloj externa.

En cuanto a la programación de las FPGA's, el fabricante proporciona un software específico denominado *fuse probe.* A continuación se muestran unas figuras con los pasos necesarios para programar las FPGA's utilizando dicha aplicación:

- Abrir el programa FUSE Probe
- Card Control → Open Card

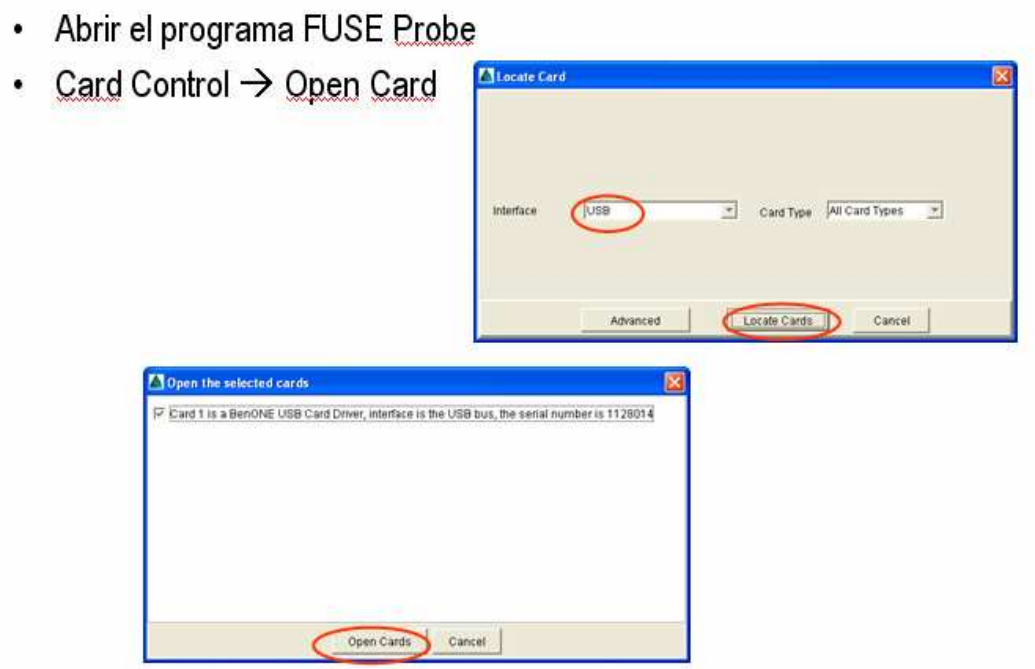

Fig. 3. 13 Localizar las FPGA's con Fuse Probe

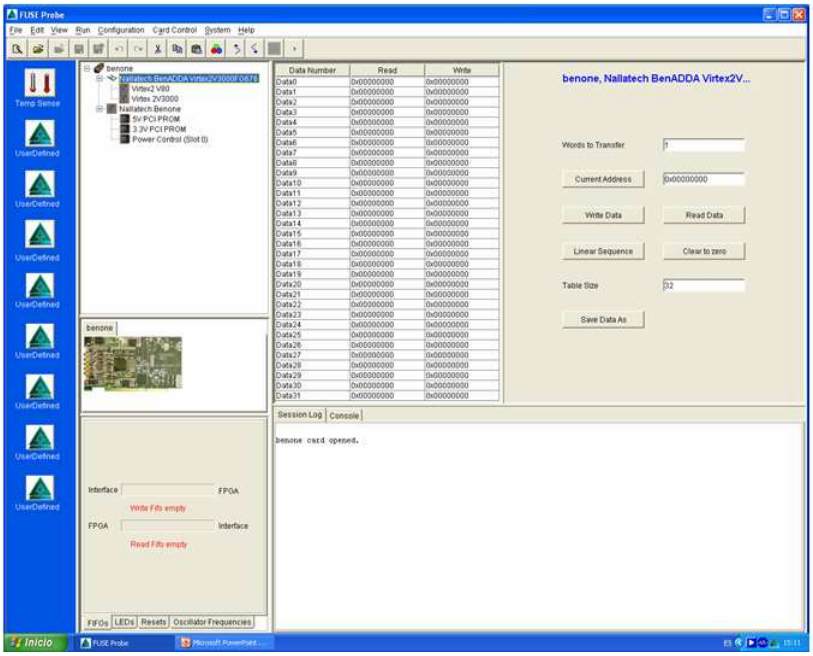

Fig. 3. 14 Ventana principal de Fuse Probe

- Seleccionar la FPGA Virtex2V80 y pulsar Assign Bitfile: 'osc_clock.bit'
- Seleccionar la FPGA Virtex2V3000 y pulsar Assign Bitfile: 'TFC4.bit'
- Configure All Cards

Fig. 3. 15 Configuración de las FPGA's

En la realización de este proyecto se ha utilizado el puerto USB para programar las FPGA's, aunque también es posible hacerlo utilizando el bus PCI que incorpora la tarjeta.

4. DISEÑO DEL FILTRO EN SYSTEM GENERATOR:

En system generator, el algoritmo puede verse en el siguiente diseño:

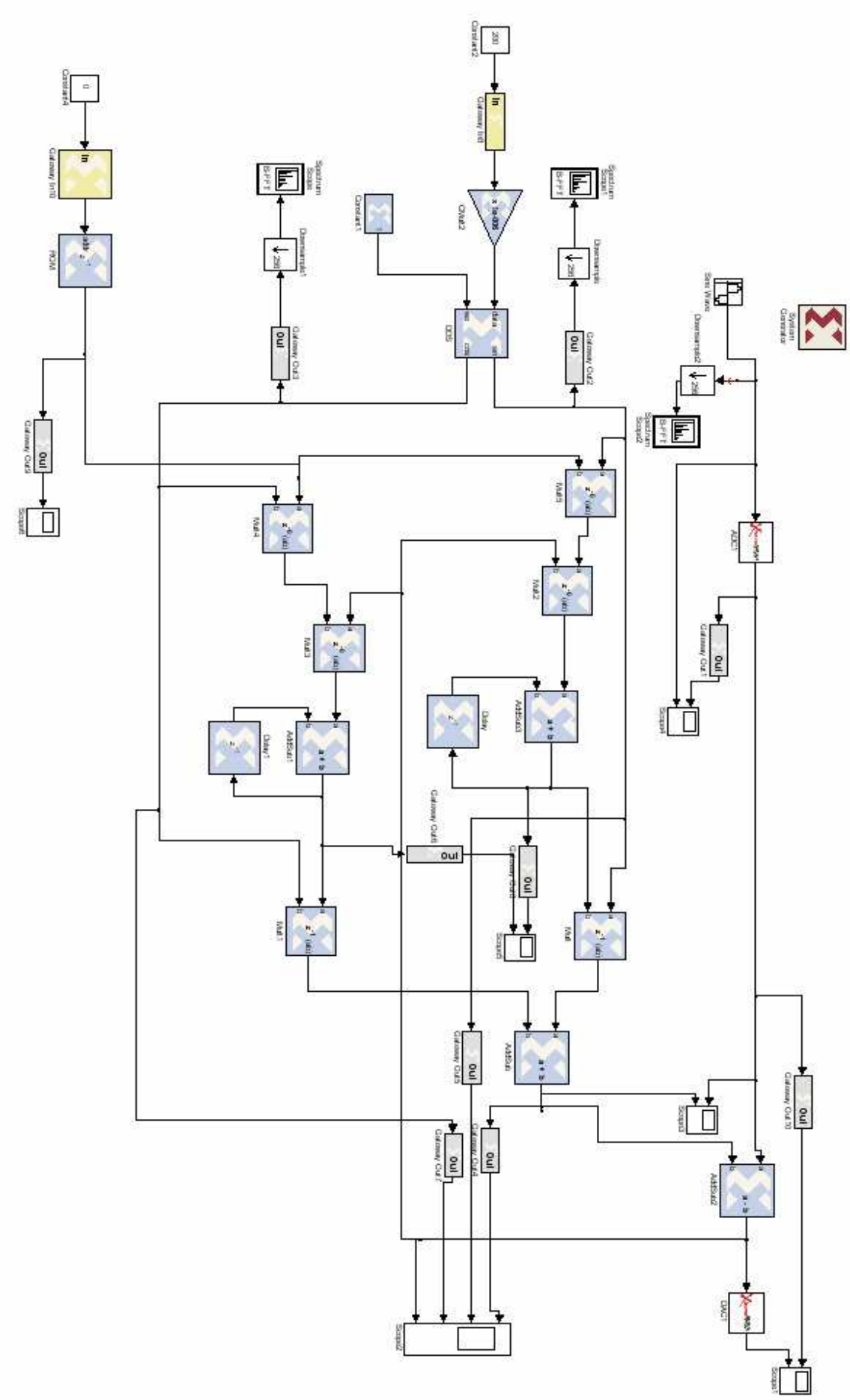

Fig. 4. 1 Diseño en System Generator

A continuación se describe cómo han sido configurados los diferentes bloques:

DDS: Sin lugar a dudas, el bloque más importante de todo el diseño. Genera un seno y un coseno cuya frecuencia es, precisamente, la frecuencia que se desea eliminar de la señal de entrada, es decir, la frecuencia central del filtro. A partir de la siguiente ecuación, se obtiene la frecuencia de la señal sintetizada:

$$f = P * (f_{clk} / 2^{M})$$

En el proyecto se ha optado por una f_{clk} = 20 MHz, y una M = 20. Además se multiplicó todo por un factor de corrección de valor 1.048576, de modo que la ecuación quedara:

$$f = P * 1.048576 * (20*10^{6} / 2^{20}) = P * 20$$

Así pues, variando P de uno en uno y entre 1 y 1000, podremos variar la frecuencia central del filtro desde 20 a 20000 Hz, en saltos de 20 Hz.
Para representar números enteros entre 1 y 1000 en base 2, se necesitan 10 dígitos (se puede representar del 0 al 1023). Por tanto, el puerto paralelo enviará 10 bits a la FPGA, y de este modo se configurará la frecuencia central del filtro.

Los parámetros más importantes del bloque DDS son :

- Output width = 20
- Lookup table input width = 15
- Accumulador = 24
- Memory type = Block Ram

Además de la entrada de datos, el bloque DDS tiene una entrada tipo bool para su habilitación.

Multiplicador por constante Cmult2 : Recoge a través del gateway in8, el valor de los diez bits que envía el puerto paralelo, y lo multiplica por $1.048576 * (1 / 2^{20})$. El resultado va a la entrada de datos del DDS, para que éste genere la frecuencia que el usuario desea eliminar.

Los parámetros más importantes de Cmult2 son :

- Value = $1.048576 / 2^{20}$.
- Number of bits = 24
- Binary point = 24
- Latency = 0
- Output type = unsigned
- Number of bit (output) = 24
- Binary point (output) = 24
- Quantization = Truncate
- Overflow = Wrap

Gateway in8 : Recoge diez bits del puerto paralelo para entregárselos al multiplicador por constante Cmult2. Hay que tener especial cuidado a la hora de asignar el nombre de los pines del conector JTAG J10 (que es por donde se introducen las señales), en el gateway. A continuación se muestra el detalle de dicha asignación de pines:

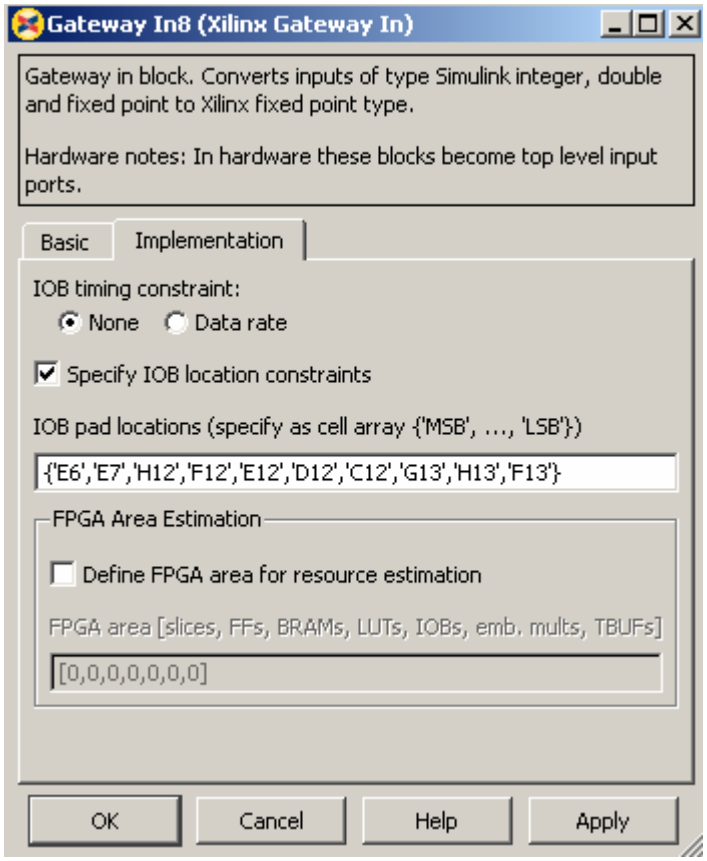

Fig. 4. 2 Asignación de pines en gateway in8

Así por ejemplo, por el pin 'E6', que se corresponde con el pin número 12 del conector JTAG J10 (Ver fig. 3.11.), se introduce el bit más significativo del valor de P.

Otros parámetros de interés del gateway in8 son :

- Output type = unsigned
- Number of bits = 10
- Binary point = 0
- Sampled period = 1 / 20000000

Memoria ROM: Este bloque también es de gran importancia, puesto que nos va a permitir configurar la anchura del filtro. Se trata de una memoria de 4 posiciones (depth = 4), por lo que serán necesarios 2 bits para poder direccionarla. En cada posición de memoria se guarda un valor diferente de 2μ, lo cual implica 4 anchuras del filtro distintas, que son:

- Superestrecha → 2μ = 0.0005
- Estrecha → 2μ = 0.001
- Media → 2μ = 0.01
- Ancha → 2μ = 0.05

Otros parámetros de interés de la memoria ROM son:

- Latency = 1
- Memory type = Block Ram
- Output type = unsigned
- Number of bits = 14
- Binary point = 12
- Optimize for = area

Gateway in10 : Recoge 2 bits del puerto paralelo para direccionar la memoria ROM y de este modo seleccionar una anchura determinada del filtro. En el conector JTAG J10 aun quedan libres dos pines para este propósito. A continuación se muestra el detalle de la asignación de pines:

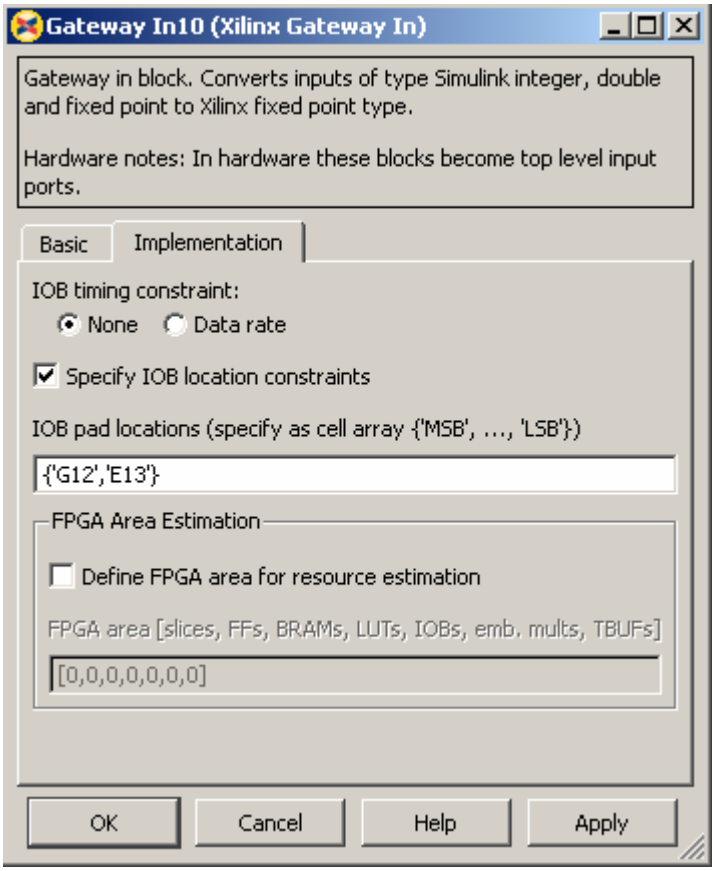

Fig. 4. 3 Asignación de pines en gateway in10

Otros parámetros de interés del gateway in10 son :

- Output type = unsigned
- Number of bits = 2
- Binary point = 0
- Sampled period = 1 / 20000000

Multiplicadores: Todos los multiplicadores se han configurado del sieguiente modo :

- output type = Signed (2's comp)
- Number of bits = 20
- Binary point = 18
- Latency = 0
- Quantization = truncate
- Overflow = Wrap

- Used embedded multipliers
- Optimize for = area

AddSub's: Todos los AddSub's se han configurado de la siguiente manera:

- Output type = Signed (2's comp)
- Number of bits = 20
- Binary point = 18
- Latency = 0
- Quantization = truncate
- Overflow = Wrap
- Use core placement information

Delay's : Los delay's se han configurado del siguiente modo:

- Latency = 1

ADC1 / DAC1: El bloque ADC ya se mostró en el apartado anterior, y el único dato que hay que aportarle es :

- Sampled Period = 1 / 20000000

El DAC1 trabaja con el mismo sampled period, y además se configura del siguiente modo para que trabaje en la zona que nos interesa (ver fig. 3.10.):

- mod 0 = 0
- mod 1 = 0
- div 0 = 1
- div 1 = 0

- **Bloque System Generator :**

A continuación se muestra el detalle de como se ha configurado este bloque:

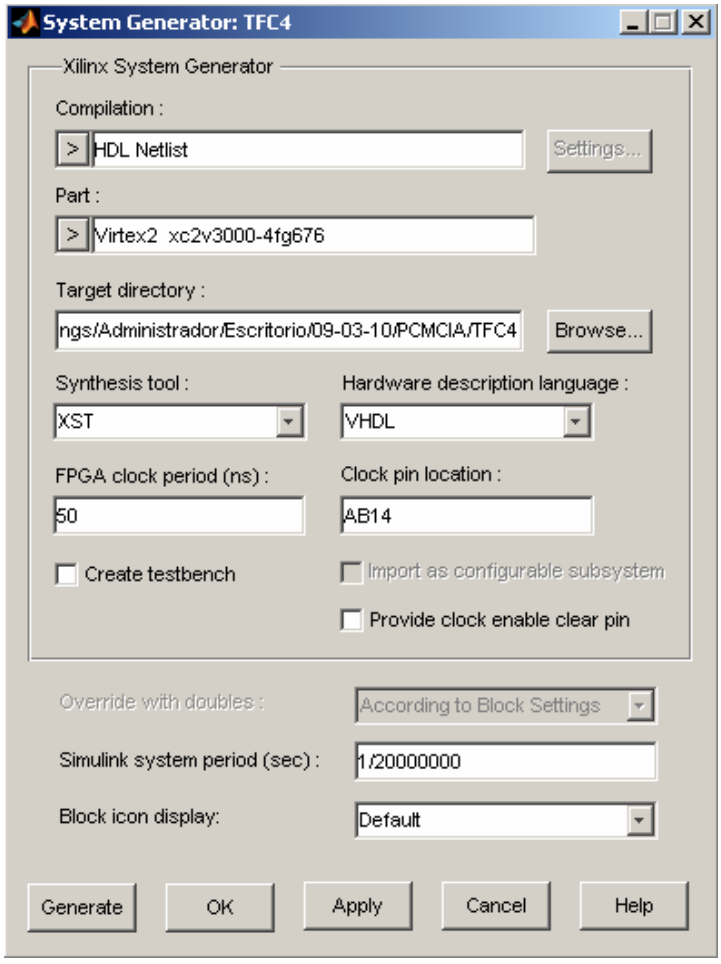

Fig. 4. 4 Configuración bloque System Generator

Para comprobar el correcto funcionamiento del circuito, se introduce una señal de entrada y se da algunos valores a P, se direcciona la memoria ROM, y finalmente se simula el diseño, obteniendo entre otras las siguientes gráficas:

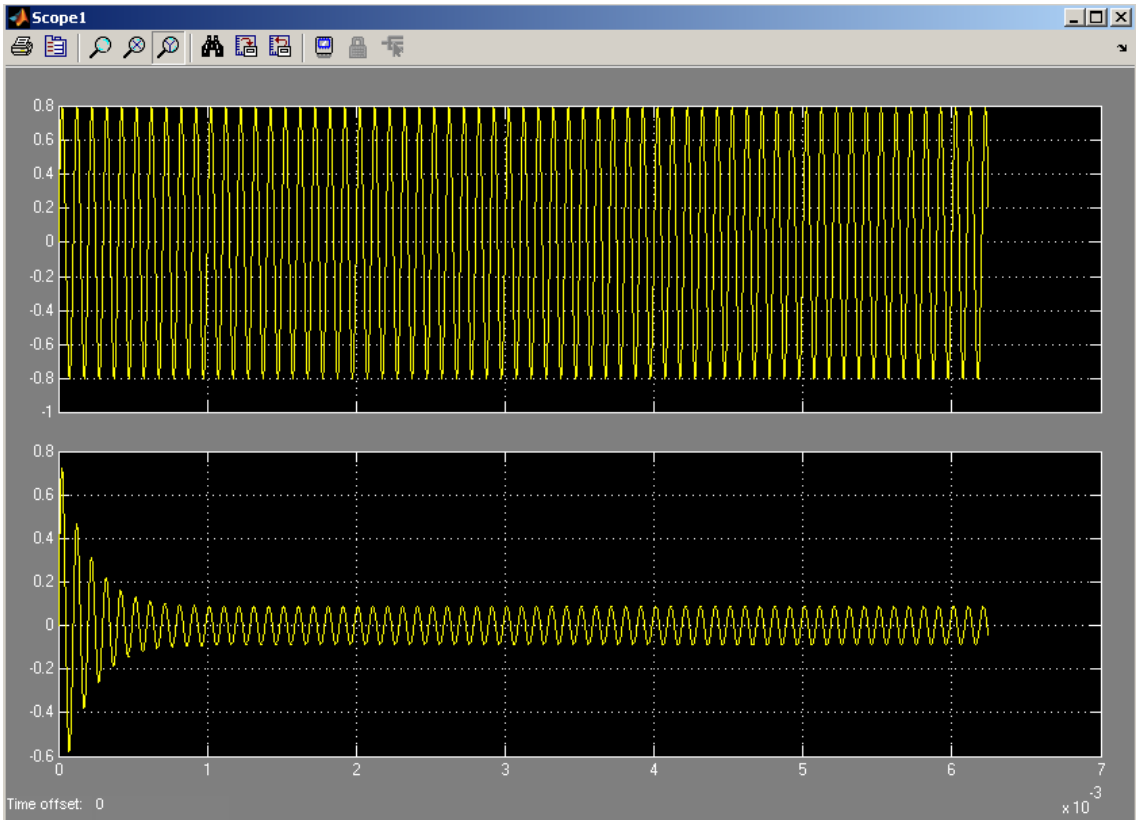

Fig. 4. 5 Entrada y Salida del filtro. fin = fc = 10 KHz. Anchura = Superestrecha

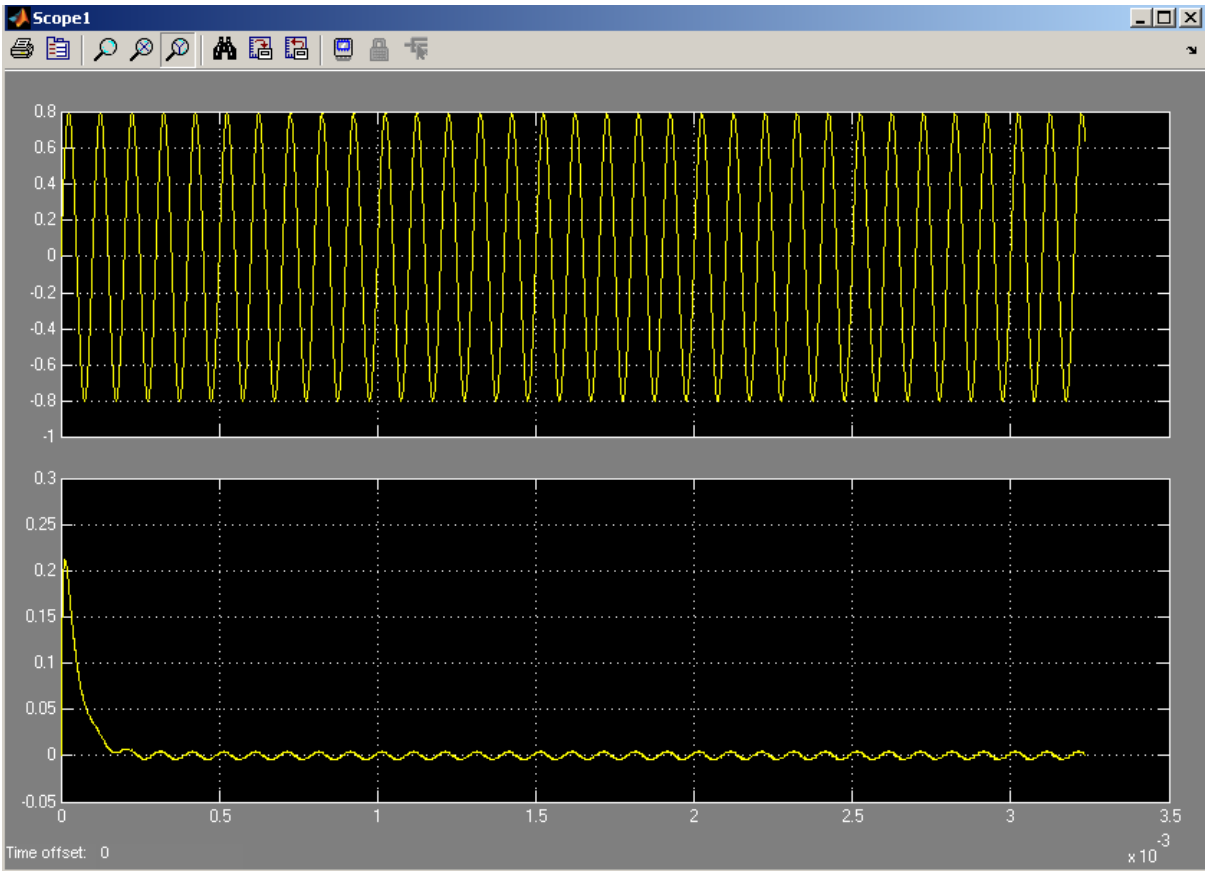

Fig. 4. 6 Entrada y Salida del filtro. fin = fc = 10 KHz. Anchura = estrecha

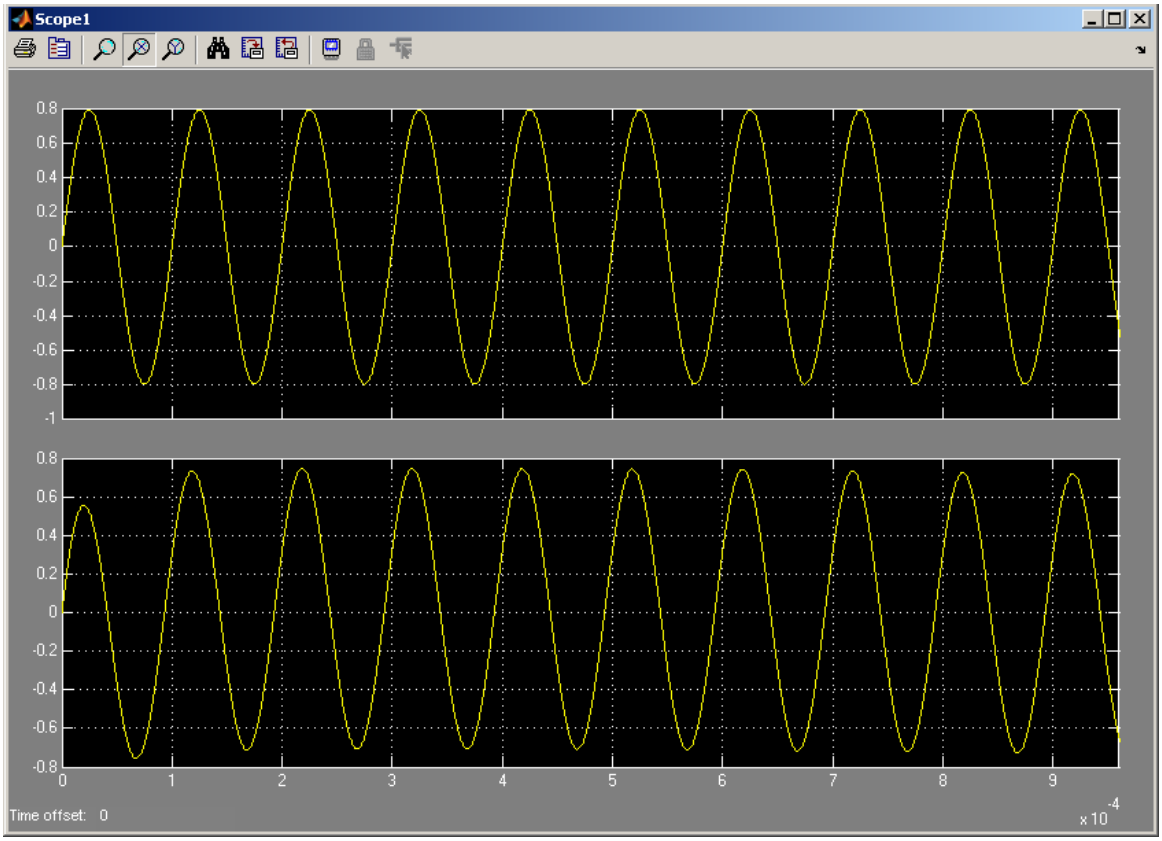

Fig. 4. 7 Entrada y Salida del filtro. fin = 10 KHz. fc = 9 KHz. Anchura = superestrecha

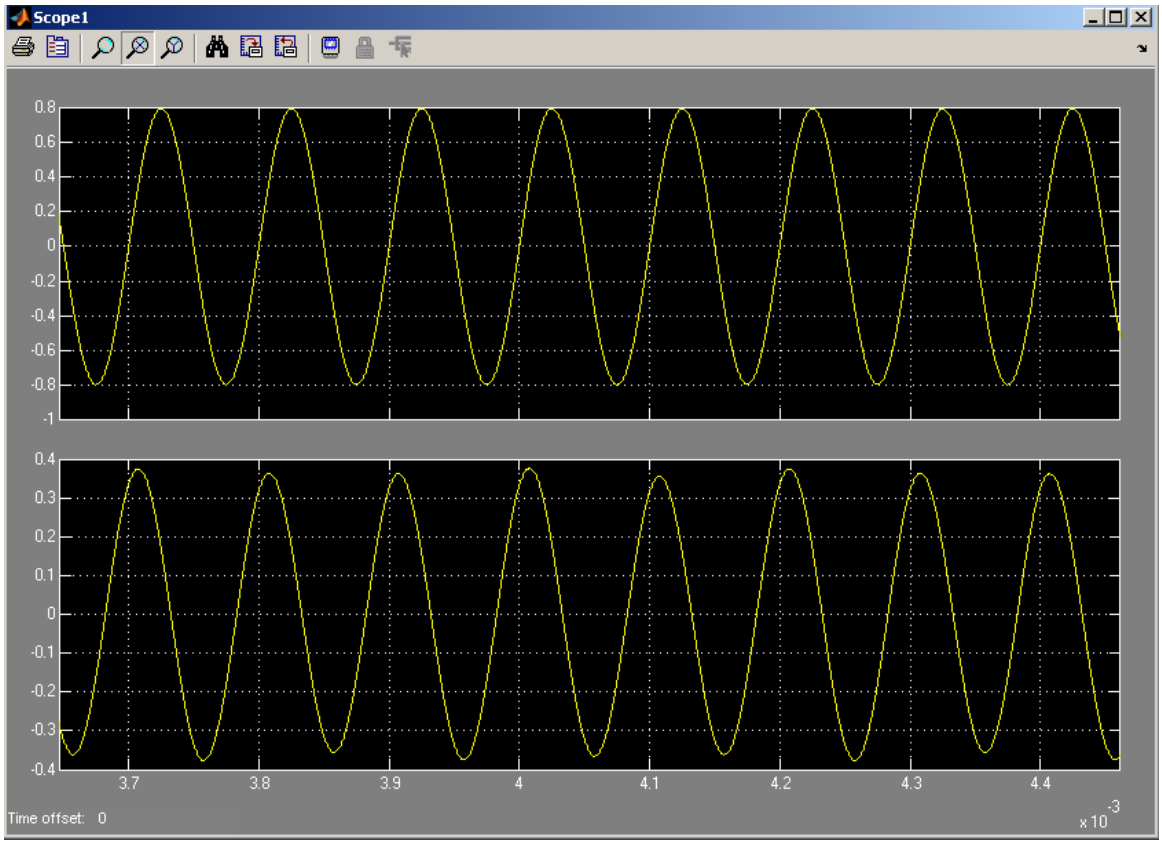

Fig. 4. 8 Entrada y Salida del filtro. fin = 10 KHz. fc = 9 KHz. Anchura = estrecha

En las dos primeras figuras, coinciden la frecuencia de la señal de entrada y la frecuencia que se desea eliminar. En ambos casos la señal es suprimida, pero en la segunda figura se elimina mucho más, ya que se ha seleccionado anchura menor que en la primera. Ya se vio en las simulaciones en MATLAB, que cuando más estrecho es el filtro menos se atenúa la frecuencia central.

En las figuras 4.7 y 4.8, tenemos una señal de entrada de 10 KHz, y decimos al circuito que se quiere eliminar la frecuencia de 9 KHz. En el circuito con anchura superestrecha, el filtro respeta la señal de entrada, como era de esperar. Cuando se selecciona anchura estrecha, la anchura del filtro es tal, que a la salida del filtro tenemos la señal de entrada atenuada a la mitad.

A la vista de estos resultados, se concluye que el circuito funciona perfectamente en simulación, por lo que se procede a la generación del archivo bitstream con el que se programará la FPGA de usuario. Para ello, primero hay que generar el código VHDL a partir del diseño en System Generator, pulsando en el botón *generate* en la ventana mostrada en la fig.4.4. Una vez generado el código VHDL se sintetiza y se implementa mediante el software *project navigator*, para finalmente crear el archivo bitstream. En este punto podemos ver en el synthesis report, los recursos de la FPGA que van a ser utilizados, y en el timming report, el mínimo período de trabajo:

```
Device utilization summary:
---------------------------

Selected Device : 2v3000fg676-4

 Number of Slices:                      441  out of  14336     3%
 Number of Slice Flip Flops:            271  out of  28672     0%
 Number of 4 input LUTs:                708  out of  28672     2%
 Number of IOs:                          47
 Number of bonded IOBs:                  46  out of    484     9%
 Number of BRAMs:                        10  out of     96    10%
 Number of MULT18X18s:                   17  out of     96    17%
 Number of GCLKs:                         1  out of     16     6%
```

Fig. 4. 9 Detalle del synthesis report

```
Timing constraint: TS_clk_76d42bc8 = PERIOD TIMEGRP "clk_76d42bc8" 50 ns HIGH 50%;

 218506218601896 items analyzed, 0 timing errors detected. (0 setup errors, 0 hold errors)
 Minimum period is  58.308ns.
```

Fig. 4. 10 Detalle del timming report

En el synthesis report observamos que apenas hemos consumido recursos de la FPGA, y del timming report obtenemos la frecuencia máxima de trabajo, f = 1 / (50.308 ns) = 19.877 MHz. El hecho de que no coincida con la frecuencia de reloj del sistema, se debe a los retardos de propagación de las señales en la FPGA.

Por último indicar, que debido a la gran cantidad de código generado al pasar de System Generator a VHDL, no se ha incluido éste en el proyecto.

5. CARACTERÍSTICAS DEL PUERTO PARALELO:

Como ya se ha comentado, se va a utilizar el puerto paralelo del PC para comunicar éste con la FPGA.

Las comunicaciones en paralelo propiamente dichas no han sido normalizadas, lo que sí se reconoce es la norma Centronics para la conexión del PC a la impresora, mediante el envío simultáneo de 8 bits de datos (un byte), además de un conjunto de líneas de protocolo.

La norma Centronics hace referencia a las características eléctricas entre el puerto paralelo y una impresora. Las líneas son latcheadas, esto es, mantienen siempre el último valor establecido en ellas mientras no se cambien expresamente, y los niveles de tensión y de corriente coinciden con los niveles de la lógica TTL, cuyos valores típicos son:

-Tensión de nivel alto: 5 V.
-Tensión de nivel bajo: 0 v.
-Intensidad de salida máxima: 2.6 mA.
-Intensidad de entrada máxima: 24 mA.

La norma Centronics establece el nombre y las características de 36 líneas eléctricas para la conexión entre el PC y la impresora.
En realidad, para la transferencia de las señales de datos y de control a través del puerto paralelo sólo se requieren 18 líneas, las restantes son líneas de masa que se enrollan alrededor de los cables de señal para proporcionarles apantallamiento y protección contra interferencias. Por esto, las tarjetas de interface del puerto paralelo en los PC's, suelen incorporar un conector hembra DB-25, mientras que prácticamente todas las impresoras

incorporan un conector hembra tipo Centronics macho de 36 pines (hoy en día, también puerto usb).

Casi todos los ordenadores de sobremesa están equipados, al menos, con una tarjeta de interface paralelo, aunque pueden tener hasta cuatro, denominados LPT1, LPT2 , LPT 3 y LPT4. La dirección de entrada/salida de cada uno de los puertos paralelo así como el número de puertos instalados en un PC se pueden consultar en Windows XP, en la siguiente ruta: Inicio/Configuración/Panel de control/Sistema/Hardware/Administrador de dispositivos, y suele ser frecuente, casi estándar, que las direcciones de los dos primeros puertos paralelos sean las siguientes:

LPT1 = 378 Hexadecimal
LPT2 = 278 Hexadecimal

El puerto paralelo tiene una estructura muy simple. Consta de tres registros: de datos, de estado y de control. Todas las señales que intervienen en el puerto tienen asociado un bit en uno de esos registros:

a) El registro de datos

Consta de 8 bits, y aunque puede ser tanto de entrada como de salida, en este proyecto se utilizará como registro de salida.
En la siguiente figura se muestra la distribución de los bits de este registro, y los pines asociados a cada uno de ellos en el conector DB-25:

REGISTRO DE DATOS							
7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0
PIN 9	PIN 8	PIN 7	PIN 6	PIN 5	PIN 4	PIN 3	PIN 2

Fig. 5. 1 Registro de datos del puerto paralelo

b) El registro de estado

El registro de estado indica la situación actual de la impresora conectada al puerto, de acuerdo con los niveles de tensión que tengan las líneas ACK, BSY, PAP y OF/ON.

Se trata de un **registro de entrada**, por lo que no se utilizará ninguno de sus bits en este proyecto, ni se explicará cual es la función concreta que desempeñan en su funcionamiento habitual en comunicación con una impresora.

Su dirección se obtiene sumando 1 a la dirección base del puerto (**0x379** en LPT1), y los bits que aparecen con el símbolo «/», indican que son activos a nivel bajo.

REGISTRO DE ESTADO							
7	6	5	4	3	2	1	0
BSY	/ACK	PAP	SELIN	ERR	X	X	X
PIN 11	PIN 10	PIN 12	PIN 13	PIN 15	X	X	X

Fig. 5. 2 Registro de estado del puerto paralelo

c) El registro de control

El registro de control permite controlar las transferencias de información con la impresora, y puede ser de lectura y de escritura. La dirección de este registro se obtiene sumando 2 a la dirección base del puerto (0x37A en L PT 1). Los bits de este registro se designan en la siguiente figura, donde el símbolo «/» delante del nombre del bit indica que es activo *a* nivel bajo.

REGISTRO DE CONTROL							
7	6	5	4	3	2	1	0

X	X	X	IRQ	DSL	/INI	ALF	STR
X	X	X	X	PIN 17*	PIN 16	PIN 14*	PIN 1*

Fig. 5. 3 Registro de control del puerto paralelo

El bit 4 es el que se encarga de gestionar las interrupciones del microprocesador, y no tiene asignado ningún pin. El resto de bits los utilizaremos como salidas, y no compete a este proyecto explicar cuál es su función concreta cuando se utilizan para comunicarse con la impresora.

El asterisco de los pines 17, 14 y 1, indica que la salida es complementada, por lo que cuando se escribe en ellos un 1 lógico, a la salida obtendremos un 0 lógico y viceversa. Un detalle éste, que habrá que tener muy en cuenta a la hora de escribir el programa en C que transmita los datos del puerto paralelo a la FPGA.

En la siguiente figura se muestran en detalle los conectores db25 macho y hembra, así como cada de los pines del conector:

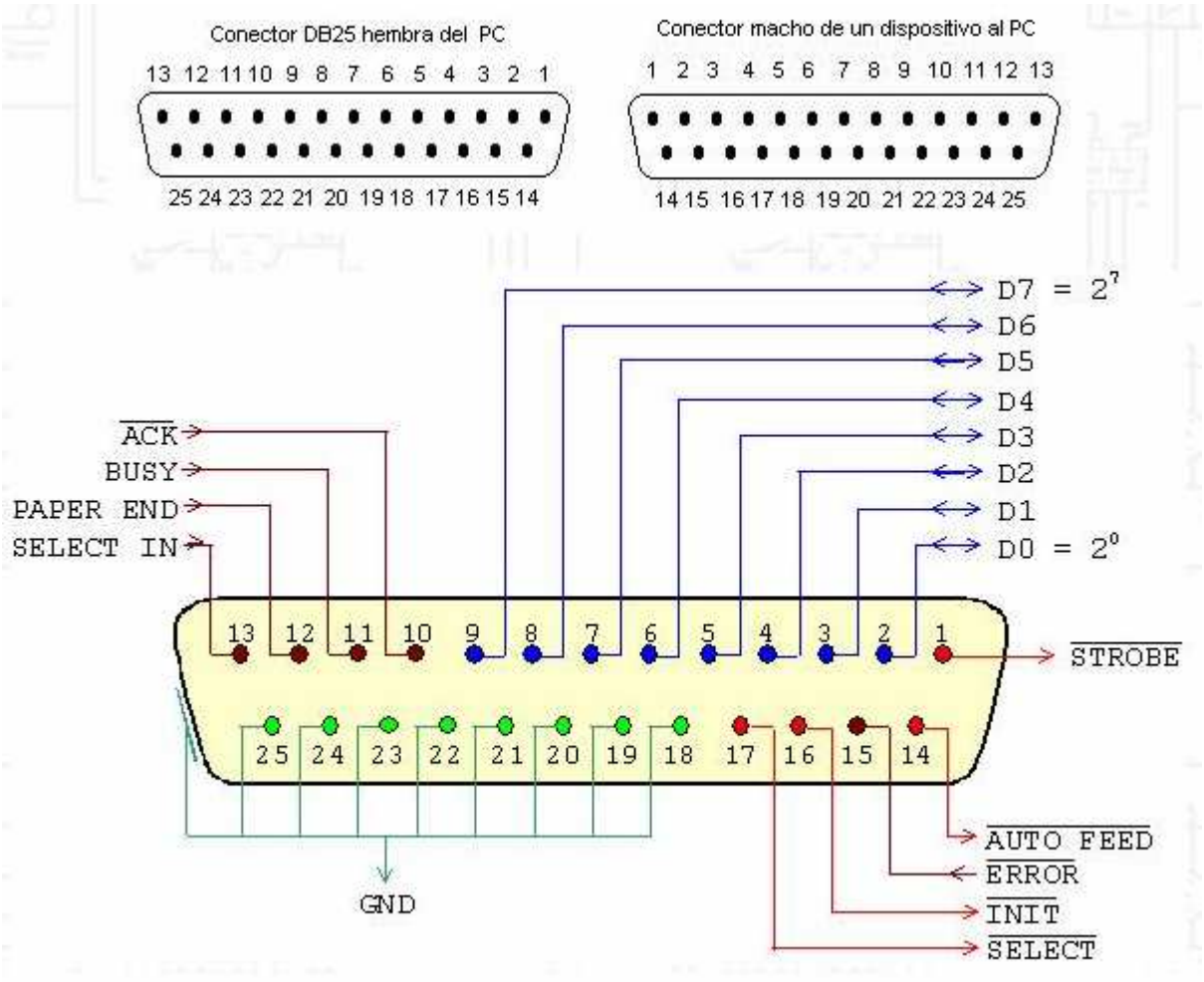

Fig. 5. 4 Conector DB - 25

Así pues, el puerto paralelo nos va a proporcionar doce líneas de salida (ocho de datos más cuatro de control), para poder parametrizar el filtro NOTCH desde el PC. Dado que se ha diseñado el filtro NOTCH para ser configurado mediante doce bits, no será necesario realizar varios envíos consecutivos de datos, lo que implicaría la realización de circuitos de sincronización, con el consiguiente aumento de complejidad en el diseño y utilización de área en la FPGA.

6. COMUNICACIÓN POR PUERTO PARALELO. PROBLEMAS Y SOLUCIONES ADOPTADAS:

Originariamente, el proyecto se pensó para poder configurar los parámetros del filtro NOTCH desde un ordenador portátil. Por tanto, el primer problema con que nos encontramos, es que en la actualidad los ordenadores portátiles no incorporan ningún puerto paralelo. Lo primero en que se pensó, fue en la adquisición de un adaptador usb/paralelo. Sin embargo, ésta no es una solución válida, ya que dicho adaptador solo sirve para comunicar el PC con la impresora, pero no proporciona una dirección de memoria en la que con un programa de usuario poder escribir en los registros de datos y de control. Finalmente, pudo solucionarse el problema mediante el uso de una tarjeta PCMCIA(los portátiles suelen disponer de ranuras de entrada PCMCIA) a puerto paralelo, que se muestra en la siguiente figura:

Fig. 6. 1 Tarjeta PCMCIA a puerto paralelo

Esta solución sí proporciona una dirección de memoria en el mapa de memoria del ordenador, aunque no es la estándar 378 Hex. sino:
LPT1 = 0xFED8 (0x indica que es una cantidad expresada en hexadecimal). Esta es la dirección base, es decir la del registro de datos. La dirección del registro de control será, por tanto, LPT1 + 2 = 0xFEDA.

El segundo problema que se planteó, es que el ordenador portátil con el que se iba a probar el proyecto, tenía el sistema operativo Windows XP. Este sistema operativo(al igual que Windows 2000, Vista…) incorpora una serie de medidas de seguridad que impiden que un programa de usuario pueda acceder a los puertos paralelo y serie del PC. Para solucionar este problema, hubo que descargarse de Internet un driver denominado USERPORT.SYS, y copiarlo en la carpeta Windows/System32/drivers, e instalar una aplicación freeware llamada USERPORT 2.0, cuya interface se muestra en la siguiente figura:

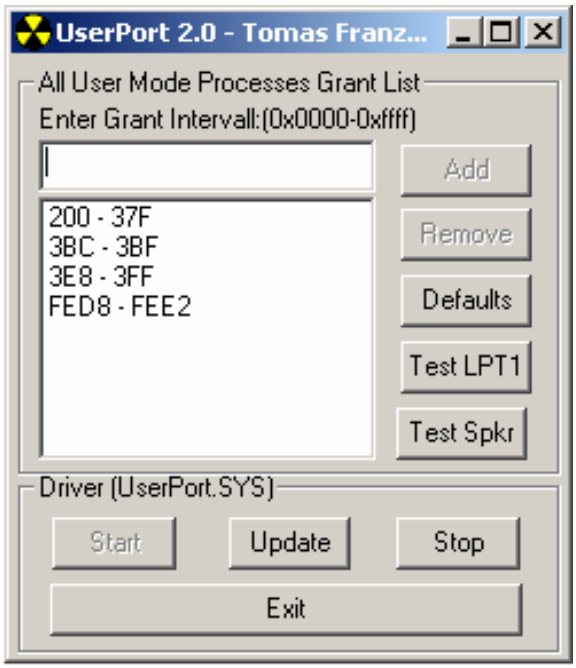

Fig. 6. 2 Interface de la aplicación Userport 2.0

Puede observarse que mediante esta aplicación, se ha habilitado el acceso al intervalo de memoria FED8 – FEE2, que contiene las direcciones de los registros de datos y control.
Así pues, ya estamos en condiciones de escribir en los registros del puerto paralelo, los datos necesarios para configurar correctamente la FPGA. Sin embargo, aun surgió un último problema. Como se ha comentado anteriormente, las salidas a nivel alto del puerto paralelo, se corresponden con un nivel de tensión de 5 voltios, mientras que las entradas de datos de la FPGA, solo admiten niveles de tensión de hasta 3.3 voltios. Por tanto, se tuvo que fabricar una placa para la adaptación de dichas señales. Esta placa consiste en un conector db25 macho (para conectarla al puerto paralelo del PC) y un conector de 14 pines (que se conectará a la FPGA) , y entre ellos, un divisor de tensión para cada una de las doce líneas de transmisión de datos. En un primer momento, se pensó en hacer un divisor puramente resistivo, pero finalmente se optó por un divisor formado por un diodo led y una resistencia. De este modo se puede monitorizar qué líneas del puerto paralelo están a ’1’ y cuales están a ‘0’.

Cuando la línea está a ‘1’, el diodo led está polarizado en directa, y tiene una tensión Vak igual a 1’8 voltios, cayendo en la resistencia 3.2 voltios.
La caída de tensión en las resistencias, es la que se utilizará como entrada de datos a la FPGA. El valor de dichas resistencias fue de 10 KΩ para las líneas del registro de datos. Con este valor, los diodos lucían lo suficiente, y se limitaba bastante la corriente.

Al realizar un montaje previo en una placa de prototipos, se comprobó que las líneas del registro de control no proporcionaban tanta corriente como las del registro de datos, no siendo capaces de entregar los miliamperios que demandaba el circuito. Como consecuencia de ello, la tensión de salida bajaba de 5 voltios a unos 3.4 voltios, repartidos en 1.8 voltios para el led y unos 1.6 voltios para la resistencia. Es por ello que para las cuatro

líneas del registro de control, se utilizaron resistencias de 18 KΩ, que demandaban menos corriente, con lo que la caída de tensión en la resistencia aumentaba hasta 2.8 voltios, valor suficiente para que la FPGA lo interpretara como un '1' lógico.

En las siguientes figuras podemos ver el fotolito que se empleó y como quedó finalmente la placa:

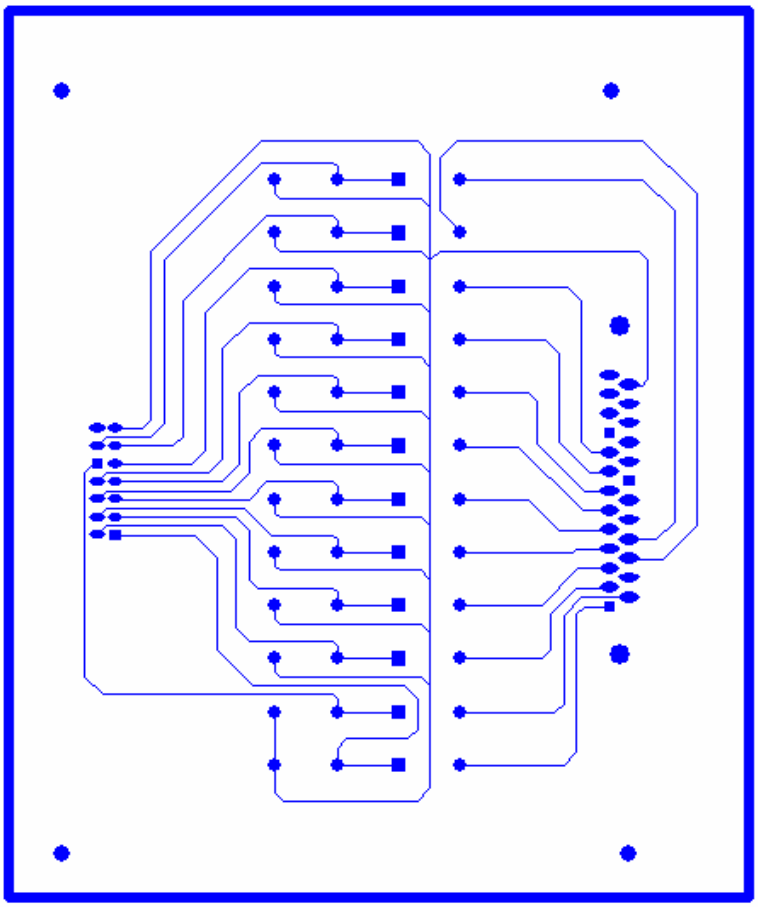

Fig. 6. 3 Fotolito de la placa de adaptación 5v / 3.3 v

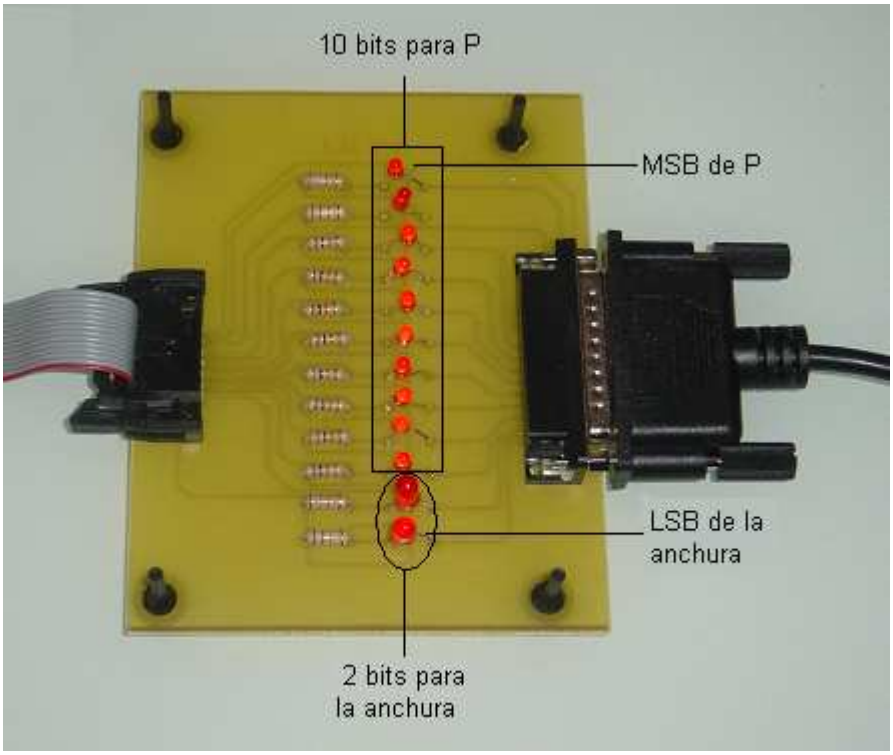

Fig. 6. 4 Placa de adaptación 5v / 3.3 v

7. APLICACION INTERFAZ DE USUARIO:

La aplicación que recoge los datos proporcionados por el usuario, los procesa y saca el resultado por el puerto paralelo para configurar la FPGA, se ha escrito en lenguaje C, utilizando el compilador TurboC, que se desarrolla en entorno MS–2.

A la hora de escribir el programa, se han tenido en cuenta las siguientes premisas :

- El programa debe preguntar qué frecuencia se desea eliminar y proporcionar 4 anchuras preestablecidas.
- El programa debe transmitir 12 bits por el puerto paralelo, que se corresponden con dos palabras de 10 y 2 bits. La palabra de 10 bits es el valor del paso P del DDS, con el que se elige la frecuencia central del filtro, y cuyo valor será P = f / 20. La palabra de dos bits direccionará la memoria con la que se elige el valor de 2μ, del que depende la anchura.
- El puerto paralelo permite transmitir hasta 12 bits. Sin embargo, no es posible escribir el valor de cada uno de los bits por separado, y tampoco escribir una palabra de 10 bits y otra palabra de 2 bits, sino que debe escribirse un valor en el registro de datos (8 bits), y otro valor en el registro de control (4 bits).
- 3 de los 4 bits del registro de control (c3-c0), son complementados, por lo que en ellos deberemos escribir el valor contrario al que queremos transmitir. Estos 3 bits son c3,c1 y c0.

En consecuencia con la anterior, se realiza la siguiente asignación de bits:

Registro de datos (d7-d0): 8 bits menos significativos de P.

2 bits más significativos del Registro de control (c3,c2): 2 bits más significativos de P.

2 bits menos significativos del Registro de control (c1,c0): 2 bits de anchura.

A continuación se presenta una tabla con el valor de los 2 bits de anchura, y con el valor de 2µ, en función de la anchura seleccionada por el usuario, donde el asterisco indica que el bit es complementado:

ANCHURA	C1*	C0*	2µ	OBSERVACIONES
Superestrecha	1	1	0.0005	Se transmite el 00
Estrecha	1	0	0.001	Se transmite el 01
Media	0	1	0.01	Se transmite el 10
Ancha	0	0	0.05	Se transmite el 11

Fig. 7. 1 Valor de los 2 bits de anchura

En la siguiente tabla se muestra el valor que toman todos los bits, en función de los datos introducidos por el usuario:

P (f/20)	ANCHURA	REG.DATOS	REG.CONTROL C3* C2 C1* C0*	OBSERVACIONES DEL REG.CONTROL
0 – 255	Superest.	P	1 0 1 1	Se transmite el 0000
	Estrecha	P	1 0 1 0	Se transmite el 0001
	Media	P	1 0 0 1	Se transmite el 0010
	Ancha	P	1 0 0 0	Se transmite el 0011
256 - 511	Superest.	P – 256	1 1 1 1	Se transmite el 0100
	Estrecha	P – 256	1 1 1 0	Se transmite el 0101
	Media	P – 256	1 1 0 1	Se transmite el 0110
	Ancha	P – 256	1 1 0 0	Se transmite el 0111

512 -767	Superest.	P – 512	0 0 1 1	Se transmite el 1000
	Estrecha	P – 512	0 0 1 0	Se transmite el 1001
	Media	P – 512	0 0 0 1	Se transmite el 1010
	Ancha	P – 512	0 0 0 0	Se transmite el 1011
768-1000	Superest.	P – 768	0 1 1 1	Se transmite el 1100
	Estrecha	P – 768	0 1 1 0	Se transmite el 1101
	Media	P – 768	0 1 0 1	Se transmite el 1110
	Ancha	P – 768	0 1 0 0	Se transmite el 1111

Fig. 7. 2 Valor de todos los bits que intervienen en la transmisión

Las funciones que permiten escribir en los registros del puerto paralelo, son las llamadas outport(), que se encuentran en la librería conio.h.

En el anexo anexo 2, puede verse el código fuente del programa comentado.

8. IMPLEMENTACION SOBRE FPGA. PRUEBAS EN LABORATORIO :

Una vez hecho todo el diseño, solo queda programar las FPGA's tal y como se explicó en el apartado 3, y realizar el siguiente montaje en el laboratorio, donde el flujo de datos va de derecha a izquierda:

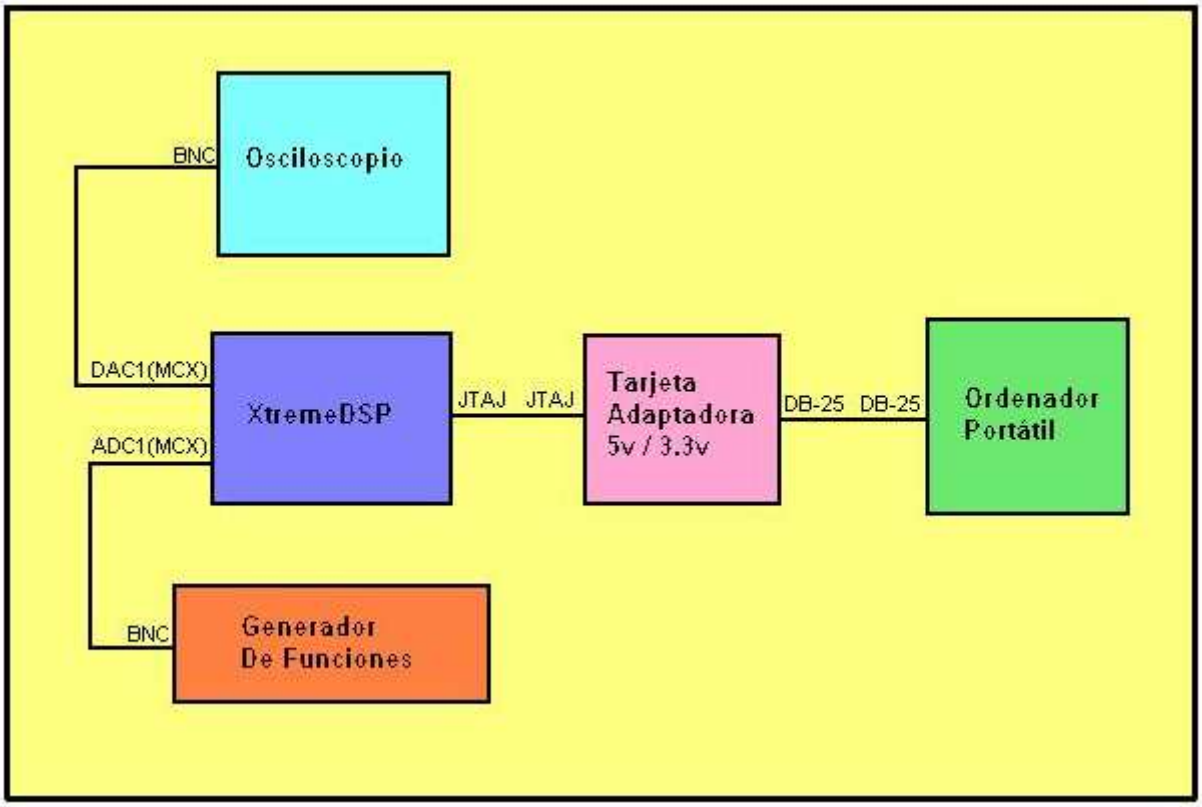

Fig. 8. 1 Diagrama de bloques del diseño en laboratorio

A través del puerto paralelo, el ordenador portátil envía los datos de configuración del filtro, que tras ser adaptados llegan a la FPGA de usuario. Por otro lado, se introduce a la placa XtreamDSP como señal de entrada, una onda senoidal cuya frecuencia se va variando. La salida de la placa se introduce en el osciloscopio, donde se observa cómo se atenúa la señal en las inmediaciones de la frecuencia central del filtro NOTCH, eliminándose casi totalmente, cuando la frecuencia de la señal de entrada coincide con la frecuencia central del filtro.

A continuación se muestra una imagen real del montaje en laboratorio:

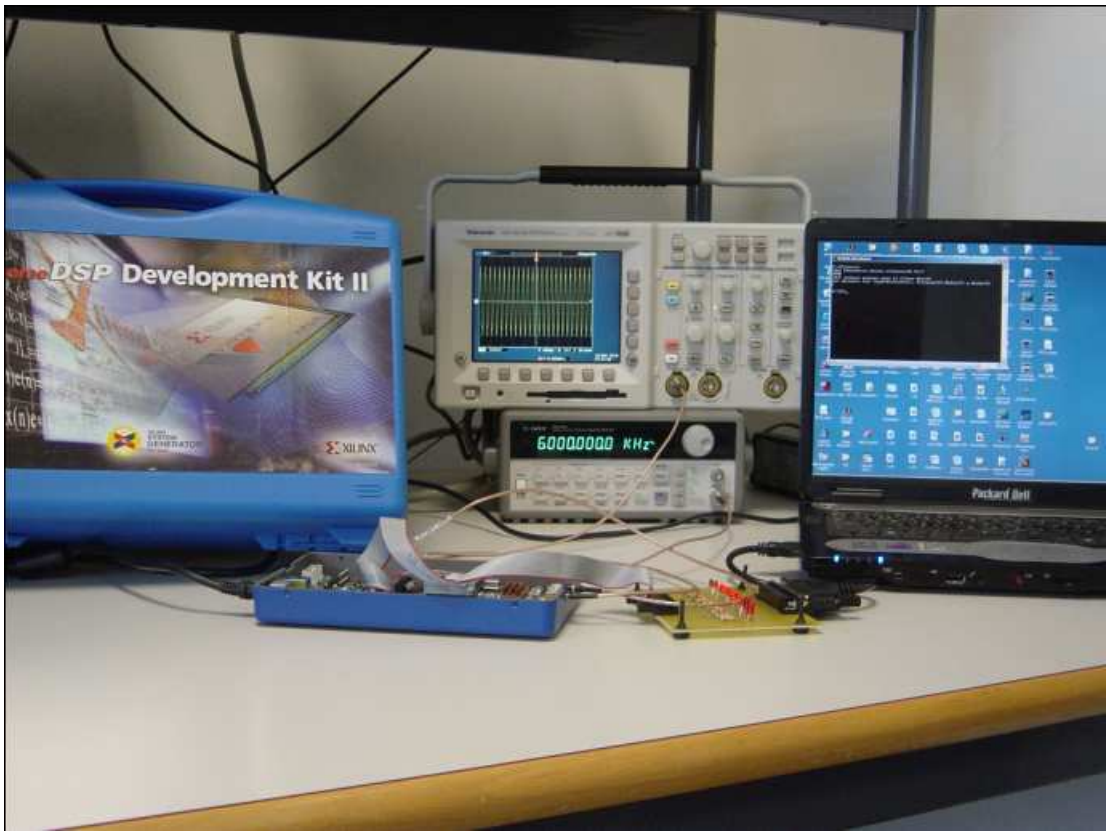

Fig. 8. 2 Imagen real de las pruebas en laboratorio

En la siguiente figura se muestra una vista cenital de la placa :

Fig. 8. 3 Vista cenital de la placa XtremeDSP en el laboratorio

A continuación, se recoge una secuencia de imágenes del osciloscopio, para diferentes frecuencias de la señal de entrada, donde puede observarse el correcto funcionamiento del filtro, configurado con una anchura superestrecha y una fc = 5 KHz:

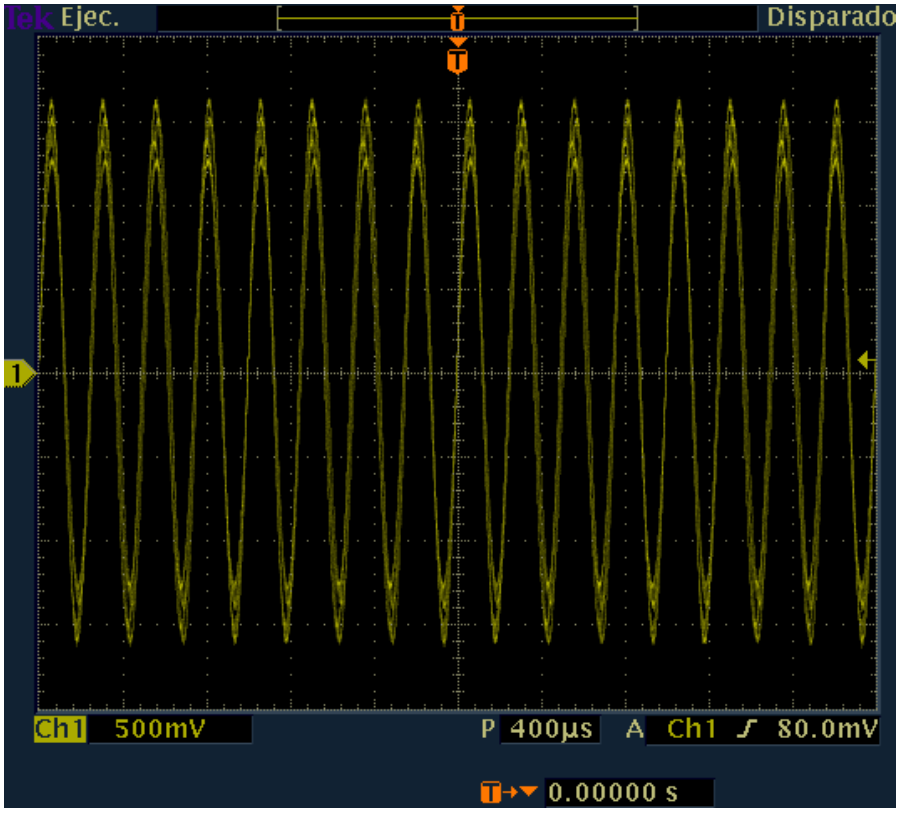

Fig. 8. 4 fin = 4700 Hz

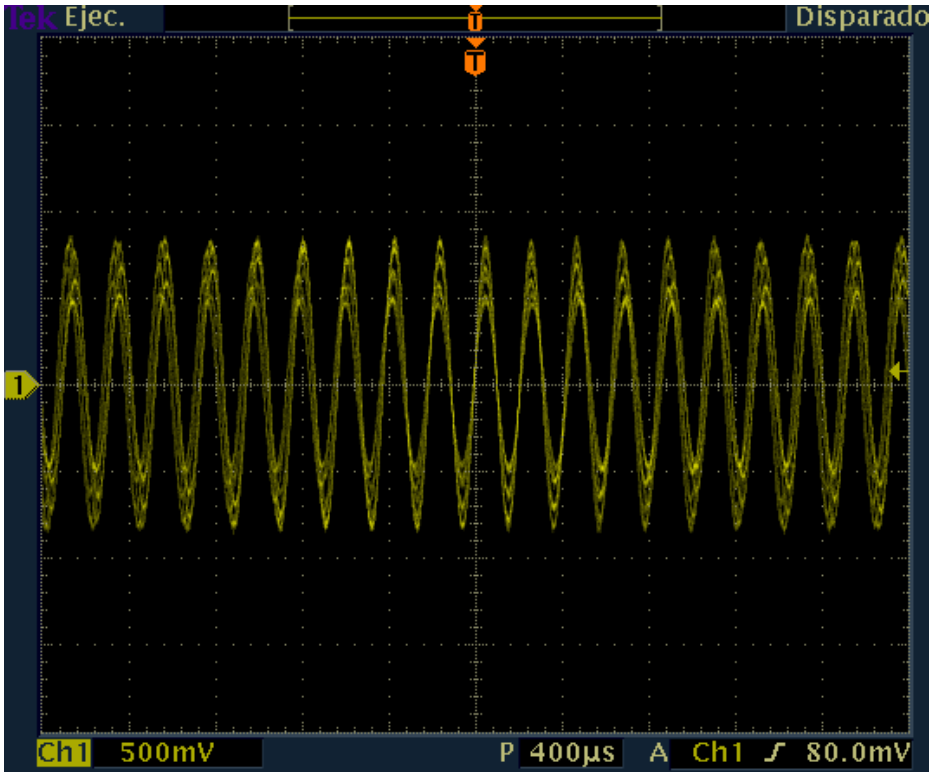

Fig. 8. 5 fin = 4900 Hz

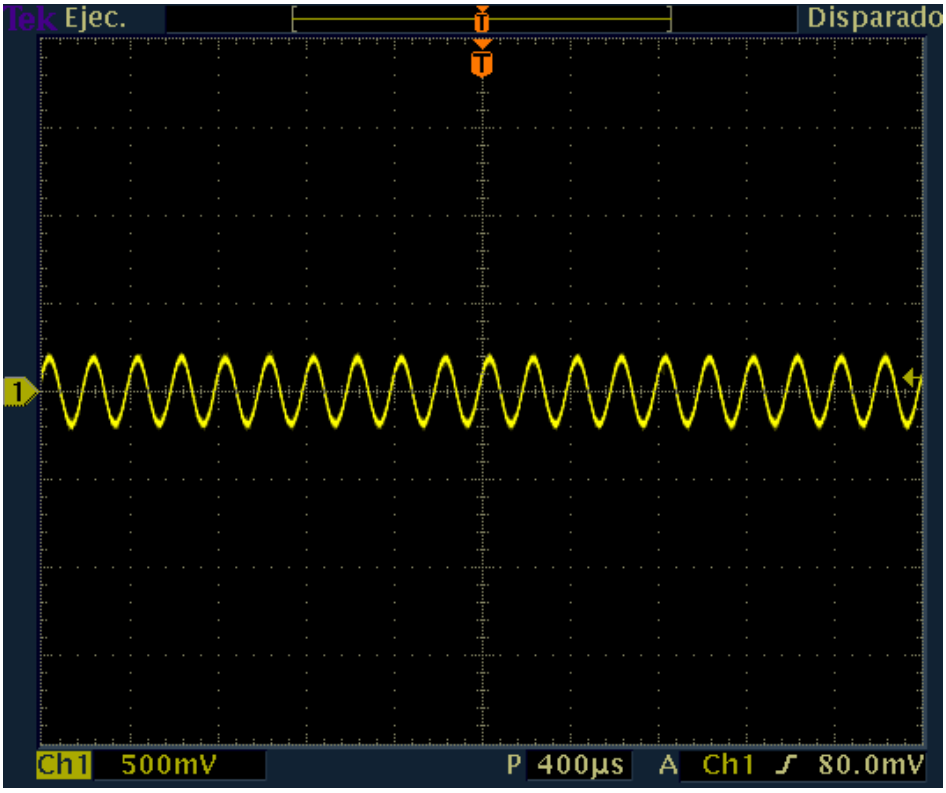

Fig. 8. 6 fin = 5 KHz

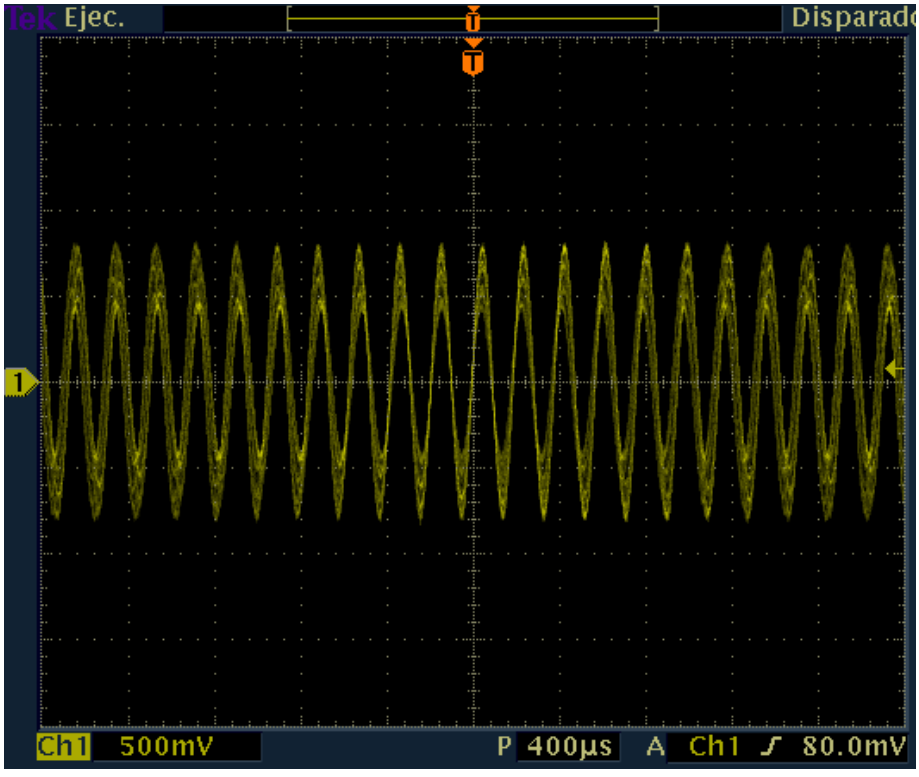

Fig. 8. 7 fin = 5100 Hz

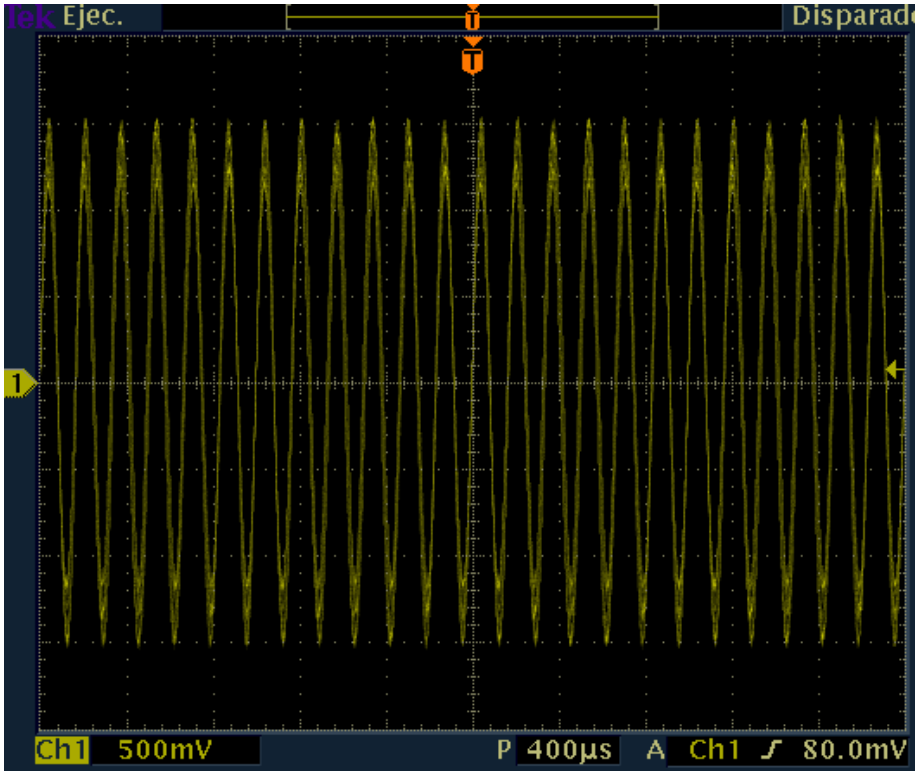

Fig. 8. 8 fin = 5200

9. CONCLUSIONES:

A la vista de los resultados, podemos concluir que la realización del proyecto ha sido un éxito. Se han conseguido todos los objetivos que se habían marcado en un principio, es decir, se ha implementado el filtro NOTCH en la FPGA, y éste puede configurarse desde un PC a través del puerto paralelo.

Además, tal y como puede ocurrir en la elaboración de cualquier proyecto de ingeniería desarrollado por una empresa en el mercado laboral, ha habido que buscar soluciones sobre la marcha a problemas que han ido surgiendo y que no se habían previsto originariamente, tales como las medidas de seguridad que incorpora Windows XP, que impiden que un programa de usuario pueda acceder al puerto paralelo, o la falta de adaptación de las señales entre el puerto paralelo y la FPGA. Estas

soluciones han sido acertadas, y los problemas surgidos no han impedido que el proyecto se llevara a cabo satisfactoriamente.

Por otro lado, me gustaría hacer constar, que aunque el proyecto no es de una elevada complejidad, sí requiere conocimientos de diferentes campos de la ingeniería electrónica, tales como el desarrollo de filtros digitales adaptativos, la implementación de circuitos digitales en FPGA's, la programación en lenguaje C o la transmisión de datos, y que cada una de estas partes diferenciadas del proyecto trabaja coordinadamente con las demás para la consecución de un objetivo común: La implementación sobre FPGA de un filtro NOTCH configurado desde un PC.

También quisiera destacar, que para la realización del proyecto ha habido que desarrollar aspectos tales como la planificación y organización del trabajo, asignando tiempos de desarrollo para cada una de las partes, solucionando problemas no previstos, buscando información, etc. Además se ha conseguido una elevada fluidez en el manejo de herramientas tales como "system generator" y "project navigator", y ha habido que realizar un estudio exhaustivo de la placa XtremeDSP Development kit II para su correcta utilización.

Por último indicar, que aunque el proyecto se realizó para aplicaciones de audio, la misma idea puede utilizarse para cualquier otra aplicación, siempre que se quiera eliminar una determinada banda de frecuencias de una señal concreta.

10. ANEXO 1: CÓDIGO PARA LAS SIMULACIONES EN MATLAB:

```
L=1000;
n=1:L;
delta=[1 zeros(1,999)];%Genero una delta.

fs = 48000;%frecuencia de muestreo.
frec = 1000;%frecuencia analógica a eliminar.
fd = frec/fs;%frecuencia digital a eliminar.

mu=0.05;%Parámetro mu del filtro adaptativo. Con él controlamos
la anchura del filtro.
d=delta;%Introduzco la delta al sistema. A la salida del sistema
tendré la respuesta al
%impulso h(t), o la respuesta en frecuencia H(f).

x1=cos(2*pi*n*fd);%Señales seno y coseno, necesarias para
conseguir la adaptación de los pesos.
x2=sin(2*pi*n*fd);

w1=0;%Valor de los pesos iniciales.
w2=0;
y=zeros(L,1);%Señal que vamos a restar a la señal de entrada
(aproximádamente una delta).
e=zeros(L,1);%Señal de salida (señal de entrada filtrada).
for k=1:L, %Bucle para la adaptación de los pesos.
y(k)=w1(k)*x1(k)+w2(k)*x2(k);
e(k)=d(k)-y(k);
w1(k+1)=w1(k)+2*mu*e(k)*x1(k);
w2(k+1)=w2(k)+2*mu*e(k)*x2(k);
end

f=(0:255)/256-0.5;%Eje de frecuencias digitales de la gráfica.
figure('name','f=1KHz,anchura=estrecha')%Nombre de la gráfica.
plot(f*fs,10*log10(abs(fftshift(fft(e,256)))));%Respuesta      en
frecuencia(analógica)del filtro.

%-----------------------------------------------------------------
----------
%-----------------------------------------------------------------
----------

%Ahora hacemos lo mismo pero para diferentes frecuencias y mu's.

frec = 5000;
fd = frec/fs;
x1=cos(2*pi*n*fd);
x2=sin(2*pi*n*fd);
w1=0;
w2=0;
y=zeros(L,1);
```

```
e=zeros(L,1);
for k=1:L,
y(k)=w1(k)*x1(k)+w2(k)*x2(k);
e(k)=d(k)-y(k);
w1(k+1)=w1(k)+2*mu*e(k)*x1(k);
w2(k+1)=w2(k)+2*mu*e(k)*x2(k);
end
figure('name','f=5KHz,anchura=estrecha')
plot(f*fs,10*log10(abs(fftshift(fft(e,256)))));
%-----------------------------------------------------------------
----------
%-----------------------------------------------------------------
----------
frec = 10000;
fd = frec/fs;
x1=cos(2*pi*n*fd);
x2=sin(2*pi*n*fd);
w1=0;
w2=0;
y=zeros(L,1);
e=zeros(L,1);
for k=1:L,
y(k)=w1(k)*x1(k)+w2(k)*x2(k);
e(k)=d(k)-y(k);
w1(k+1)=w1(k)+2*mu*e(k)*x1(k);
w2(k+1)=w2(k)+2*mu*e(k)*x2(k);
end
figure('name','f=10KHz,anchura=estrecha')
plot(f*fs,10*log10(abs(fftshift(fft(e,256)))));
%-----------------------------------------------------------------
----------
%-----------------------------------------------------------------
----------
frec = 10000;
fd = frec/fs;
mu = 0.2;
x1=cos(2*pi*n*fd);
x2=sin(2*pi*n*fd);
w1=0;
w2=0;
y=zeros(L,1);
e=zeros(L,1);
for k=1:L,
y(k)=w1(k)*x1(k)+w2(k)*x2(k);
e(k)=d(k)-y(k);
w1(k+1)=w1(k)+2*mu*e(k)*x1(k);
w2(k+1)=w2(k)+2*mu*e(k)*x2(k);
end
figure('name','f=10KHz,anchura=media')
plot(f*fs,10*log10(abs(fftshift(fft(e,256)))));
%-----------------------------------------------------------------
----------
%-----------------------------------------------------------------
----------
```

```
frec = 10000;
fd = frec/fs;
mu = 0.4;
x1=cos(2*pi*n*fd);
x2=sin(2*pi*n*fd);
w1=0;
w2=0;
y=zeros(L,1);
e=zeros(L,1);
for k=1:L,
y(k)=w1(k)*x1(k)+w2(k)*x2(k);
e(k)=d(k)-y(k);
w1(k+1)=w1(k)+2*mu*e(k)*x1(k);
w2(k+1)=w2(k)+2*mu*e(k)*x2(k);
end
figure('name','f=10KHz,anchura=ancha')
plot(f*fs,10*log10(abs(fftshift(fft(e,256)))));
%-------------------------------------------------------------------
--------
%-------------------------------------------------------------------
--------
frec = 10000;
fd = frec/fs;
mu = 0.005;
x1=cos(2*pi*n*fd);
x2=sin(2*pi*n*fd);
w1=0;
w2=0;
y=zeros(L,1);
e=zeros(L,1);
for k=1:L,
y(k)=w1(k)*x1(k)+w2(k)*x2(k);
e(k)=d(k)-y(k);
w1(k+1)=w1(k)+2*mu*e(k)*x1(k);
w2(k+1)=w2(k)+2*mu*e(k)*x2(k);
end
figure('name','f=10KHz,anchura=superestrecha')
plot(f*fs,10*log10(abs(fftshift(fft(e,256)))));
```

11. ANEXO 2 : CÓDIGO FUENTE EN LENGUAJE C:

```
#include <conio.h>

#include <stdio.h>

void main()

{

  int puerto1 = 0xFED8;/*direccion del registro de datos*/

  int puerto2 = 0xFEDA;/*direccion del registro de control*/

  int dato1;/*valor que se escribira en el registro de datos*/

  int dato2;/*valor que se escribira en el registro de control*/

  int frec;/*valor de la frecuencia introducida por el usuario*/

  int anchura;/*valor de la anchura introducida por el usuario*/

  int p;/*valor del paso de dds*/

  printf("¨Qu, frecuencia deseas eliminar(En Hz)?\n");

  scanf("%d",&frec);

  printf("¨Qu, anchura quieres para el filtro Notch?\n");

  printf("Las opciones son: SuperEstrecha(1), Estrecha(2),Media(3) y
Ancha(4)\n");

  scanf("%d",&anchura);

  p = frec / 20;

  if (p < 256){

     dato1 = p;
     dato2 = 8;

     } else if (p > 255 && p < 512){

     dato1 = p - 256;
```

```
      dato2 = 12;

      } else if (p > 511 && p < 768){

      dato1 = p - 512;
      dato2 = 0;

      } else {

      dato1 = p - 768;
      dato2 = 4;

      }

      if(anchura==1){

         dato2 = dato2 + 3;

         }else if(anchura==2){

         dato2 = dato2 + 2;

         }else if(anchura==3){

         dato2 = dato2 + 1;

         }else if(anchura == 4) {

         dato2 = dato2 + 0;

         }

   outport(puerto1,dato1);/*se escribe en el registro de datos*/

   outport(puerto2,dato2);/*se escribe en el registro de control*/


}
```

12. ANEXO 3 : CÓDIGO FUENTE EN LENGUAJE VHDL PARA CONFIGURAR EL RELOJ:

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

entity OSC_CLOCK is
    Port ( OSC_IN : in std_logic;
           DAC0_CLKp : out std_logic;
           DAC0_CLKn : out std_logic;
           DAC1_CLKp : out std_logic;
           DAC1_CLKn : out std_logic;
           ADC0_CLKp : out std_logic;
           ADC0_CLKn : out std_logic;
           ADC1_CLKp : out std_logic;
           ADC1_CLKn : out std_logic;
           CLK1_OUTp : out std_logic
           --CLK1_OUTn : out std_logic
            );
end OSC_CLOCK;

architecture Behavioral of OSC_CLOCK is

component BUFG
        port (          I: in std_logic;
                O: out std_logic
                );
end component;

component OBUFDS_LVPECL_33
        port ( O: out std_logic;
                OB: out std_logic;
                I: in std_logic
                );
end component;

component OBUF
        port (          I: in std_logic;
                O: out std_logic
                );
end component;
```

```
component DCM
--
   generic (
          DLL_FREQUENCY_MODE : string := "LOW";
          DUTY_CYCLE_CORRECTION : boolean := TRUE;
          STARTUP_WAIT : boolean := FALSE
         );
--

   port ( CLKIN       : in  std_logic;
          CLKFB       : in  std_logic;
          DSSEN       : in  std_logic;
          PSINCDEC    : in  std_logic;
          PSEN        : in  std_logic;
          PSCLK       : in  std_logic;
          RST         : in  std_logic;
          CLK0        : out std_logic;
          CLK90       : out std_logic;
          CLK180      : out std_logic;
          CLK270      : out std_logic;
          CLK2X       : out std_logic;
          CLK2X180    : out std_logic;
          CLKDV       : out std_logic;
          CLKFX       : out std_logic;
          CLKFX180    : out std_logic;
          LOCKED      : out std_logic;
          PSDONE      : out std_logic;
          STATUS      : out std_logic_vector(7 downto 0)
        );
   end component;

signal GND, RST, LOCK : std_logic;
--
signal CLK2_W: std_logic;
signal CLK2X_W: std_logic;

signal OSC_OUT: std_logic;
signal OSC_OUTI: std_logic;
signal OSC_S: std_logic;

attribute CLKFX_MULTIPLY: integer;
attribute CLKFX_DIVIDE: integer;
attribute CLKFX_MULTIPLY of U_DCM: label is 2; -- variable multiplicación
attribute CLKFX_DIVIDE of U_DCM: label is 25; --variable división
```

```
begin

--H6: BUFG port map (I => OSC_IN, O => OSC_OUT);

-- DCM Instantiation
U_DCM: DCM
  port map (
          CLKIN =>    OSC_IN,
          CLKFB =>    OSC_S,
                  CLK0  =>      OSC_S,
          --CLK270  =>   OSC_OUTI,
                  DSSEN =>    GND,
          PSINCDEC => GND,
          PSEN =>     GND,
          PSCLK =>    GND,
          RST =>      RST,
          --CLK2X =>     OSC_OUT,
          --CLK2X180 =>  OSC_OUTI,
          CLKFX   =>     OSC_OUT,
          CLKFX180  =>         OSC_OUTI,
                  LOCKED =>   LOCK
          );


H1: OBUFDS_LVPECL_33 port map (I => OSC_OUT, O => DAC0_CLKp,
OB => DAC0_CLKn);
H2: OBUFDS_LVPECL_33 port map (I => OSC_OUT, O => DAC1_CLKp,
OB => DAC1_CLKn);
H3: OBUFDS_LVPECL_33 port map (I => OSC_IN, O => ADC0_CLKp, OB
=> ADC0_CLKn);
H4: OBUFDS_LVPECL_33 port map (I => OSC_IN, O => ADC1_CLKp, OB
=> ADC1_CLKn);
H5: OBUF port map (I => OSC_OUT, O => CLK1_OUTp);

end Behavioral;
```