Universidad de Alcalá
Escuela Politécnica Superior

Grado Ingeniería Electrónica de Comunicaciones

Trabajo Fin de Grado

Aplicaciones de procesado digital de señal sobre plataforma NUCLEO-L432KC de bajo consumo

Autor: Taslima Diallo Lorente

Tutor: José Manuel Villadangos Carrizo

2019

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Departamento de Electrónica

Grado en Ingeniería Electrónica de la Comunicaciones

Trabajo Fin de Grado

“Aplicaciones de procesado digital de señal sobre plataforma NUCLEO-L432KC”

Taslima Diallo Lorente

2019

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería Electrónica de Comunicaciones

Trabajo Fin de Grado

“Aplicaciones de procesado digital de señal sobre plataforma NUCLEO-L432KC de bajo consumo”

Autor: Taslima Diallo Lorente

Tutor: José Manuel Villadangos

TRIBUNAL:

Presidente: Enrique Santiso Gómez

Vocal 1: Luis Miguel Bergasa Pascual

Vocal 2: José Manuel Villadangos Carrizo

Fecha: ……

Índice

Índice de Figuras

Índice de tablas:

1. Resumen

El proyecto consiste en la implementación del procesado digital de señal en un microcontrolador de bajo consumo del Cortex-M4 (NUCLEO-L432KC) aplicado, en este caso, a un conjunto de filtros FIR (paso bajo, paso alto, paso banda y banda eliminada) diseñados cuyos coeficientes se generan desde Matlab. El proyecto tiene como funcionalidad principal realizar un filtrado digital en tiempo real de una señal analógica de entrada.

Palabras clave: Timer, NUCLEO-L432KC, muestras, conversión, interrupción, filtrado, FIR, ADC y DAC.

2. Abstract

The project consists in the implementation of the digital signal processor in a low consumption microcontroller of Cortex-M4 (NUCLEO-L432KC) applied, in this case, a set of FIR filters (low pass, high pass, band pass and band eliminated) whose coefficients are generated from Matlab. The main function of the project is to perform a real-time digital filtering of an analog input signal.

Keywords: Timer, NUCLEO-L432KC, muestras, conversión, interrupción, filtrado, FIR, ADC and DAC.

3. Introducción

Un DSP es un dispositivo que tiene como funcionalidad principal el procesado digital de señales analógicas. Es un sistema basado en un microprocesador y/o microcontrolador que posee un conjunto de instrucciones matemáticas, un hardware y un software embebido optimizados para aplicaciones que requieran operaciones numéricas a muy alta velocidad como es el caso del procesado de señales en tiempo real.

Un DSP se compone, en líneas generales, de un conversor analógico-digital A/D; memoria de datos, memoria de programa y DMA,; multiplicadores y acumuladores; unidad aritmético-lógica; PLL y un conversor digital-analógico D/A.

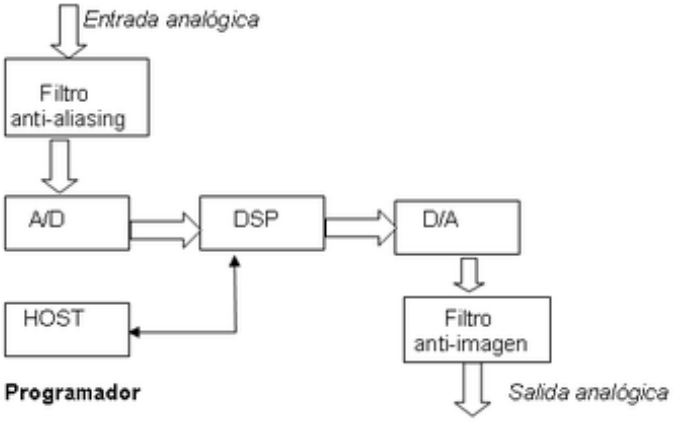

Figura 1. Estructura básica de un DSP.

Para este proyecto en cuestión se utiliza un microcontrolador de la arquitectura de ARM Cortex-M4 como es el STM32L432KC en la plataforma NUCLEO-L432KC. Este microcontrolador, aparte de ser de bajo consumo, reúne, entre otras, todas las características básicas previamente descritas que debería tener un DSP.

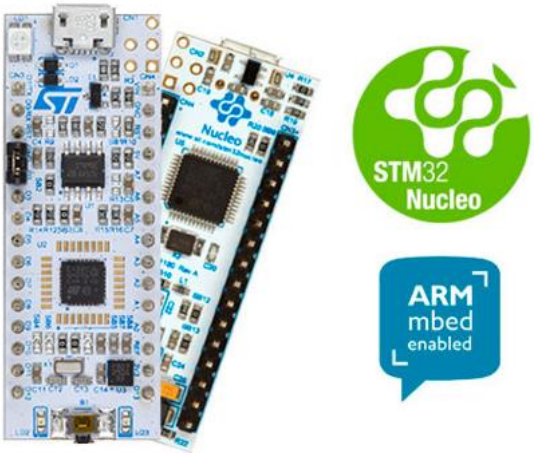

Figura 2. NUCLEO-L432KC.

En el desarrollo de este proyecto con la NUCLEO-L432KC se pretende mostrar el filtrado en tiempo real de señales analógicas del exterior y ver sobre todo las ventajas de trabajar con un microcontrolador que, aunque trabaje en coma flotante, posee instrucciones de procesado de señales que facilitan en gran medida el desarrollo de proyectos de este tipo.

La clase de procesado de señales que se va a llevar a cabo en este proyecto es el filtrado de señales analógicas mediante distintos tipos de filtros FIR, que se trata de un tipo de filtros digitales cuya respuesta al impulso es un número finito de términos no nulos.

Finalmente se muestran los resultados del filtrado de cada uno de filtros implementados en base a un plan de pruebas definido con el fin de mostrar, validar y evidenciar el correcto funcionamiento del sistema diseñado.

4. Plataforma NUCLEO-L432KC

NUCLEO-L432KC es una tarjeta de desarrollo de software embebido que contiene un microcontrolador de 32 bits con arquitectura de ARM Cortex M4 de la familia de microcontroladores STM32 de STMicroelectronics. Las características más destacadas de esta tarjeta son:

- Frecuencia máxima de CPU de 80 MHz.
- VDD de 1.65V a 3.6V
- 256KB de memoria FLASH.
- 64KB de memoria SRAM.
- 4 Temporizadores de propósito general.
- 2 Interfaces de comunicación SPI/I2S.
- 2 Interfaces de comunicación I2C.
- 2 Interfaces de comunicación USART.
- Un ADC de 12 bits con 10 canales.
- 20 pines GPIO configurables como interrupción externa.
- RTC (Real Time Clock).
- Generador Aleatorio (TRNG para entropía HW).

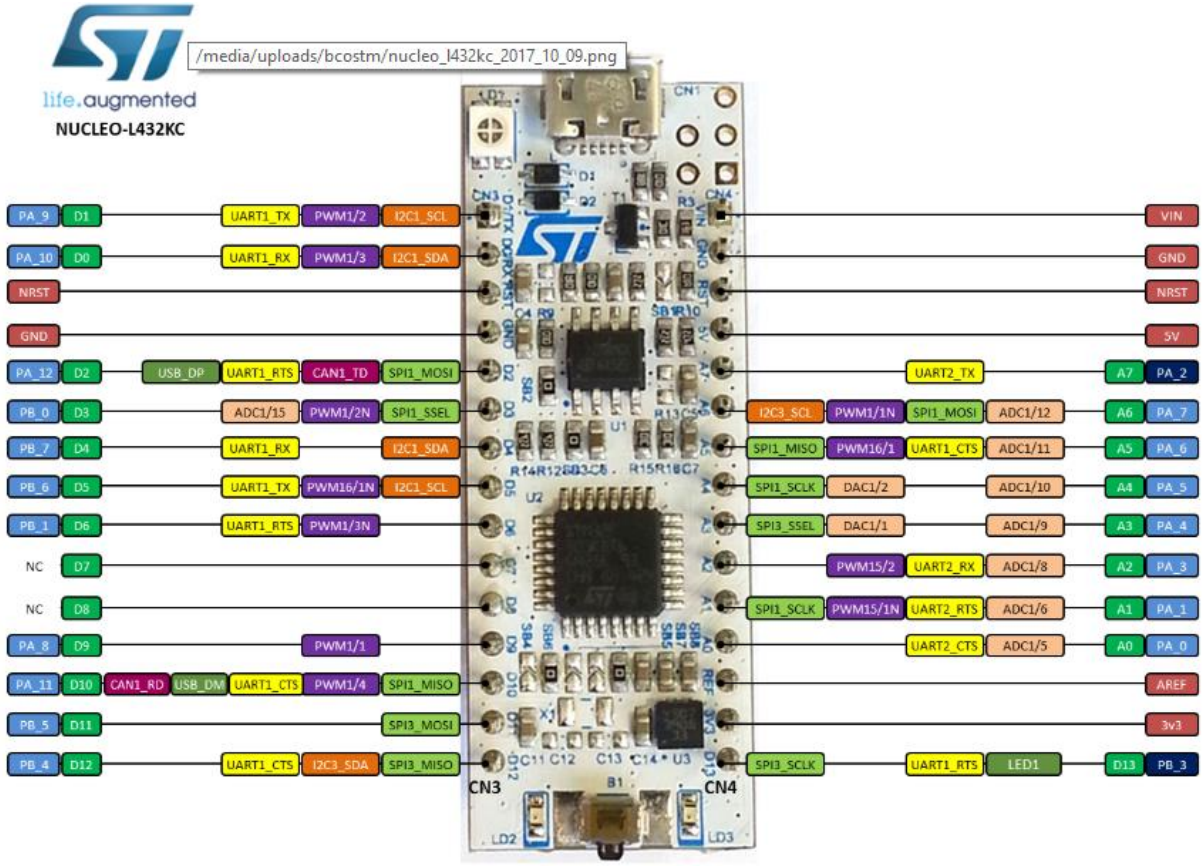

Figura 3. Pinout NUCLEO-L432KC.

A continuación, se va a hacer una descripción de los periféricos y elementos funcionales de la tarjeta que se han utilizado en el proyecto.

4.1 ADC (Analog to Digital Converter)

El conversor analógico-digital que tiene esta tarjeta codifica las muestras convertidas en valores enteros de 12 bits con 10 canales de entrada analógicos. Tiene la capacidad de convertir unitariamente cada canal o la conversión en modo ráfaga de varios canales, pudiendo ser éstos, canales de entrada en modo *single-ended* o en modo diferencial. El rango de tensiones de entrada que es capaz de convertir comprende desde los 0V hasta los 3.6V.

Se puede configurar que el valor de convertido por el ADC tenga los bits a lineados a la izquierda o a la derecha y se le puede asociar un canal del DMA para transferir el valor convertido a memoria o a otro periférico.

Existe la posibilidad de habilitar la interrupción global del ADC al final de cada conversión con el fin de realizar alguna función con el valor de cada muestra.

El inicio de conversión puede configurarse de manera que se indique por software pudiendo tener la opción de que convierta de manera continua, o bien de manera que sea un *Timer* el que marque el inicio de conversión de manera regular.
La frecuencia de conversión del ADC viene condicionada por la fuente de reloj escogida y por los valores de los PLL internos siguiendo el diagrama de bloques de la Figura 4.

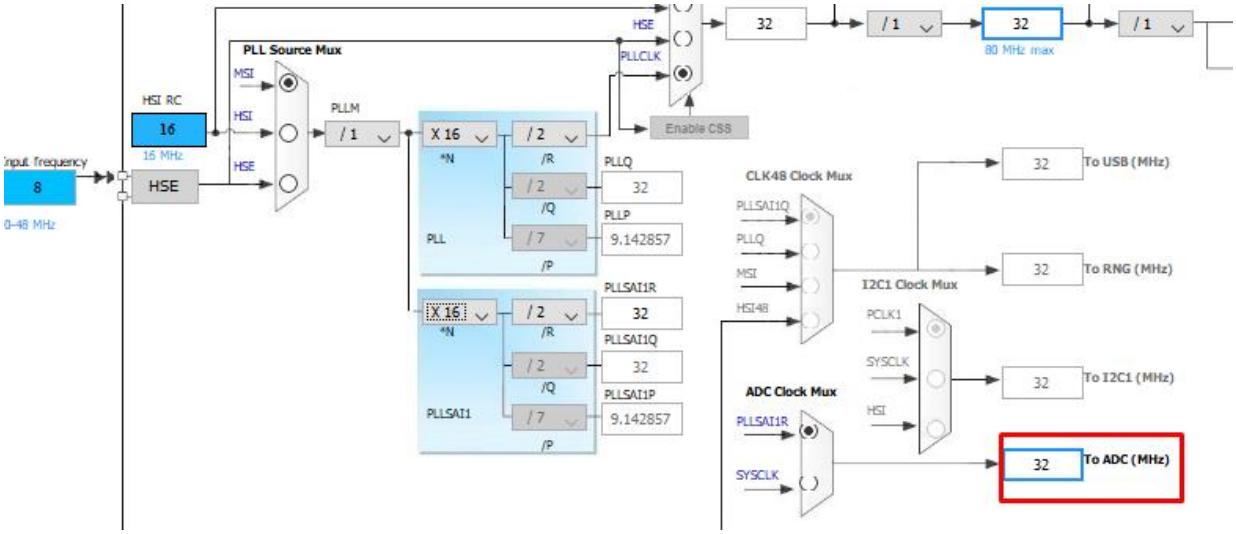

Figura 4. Diagrama de bloques – frecuencia del ADC.

4.2 DAC (Digital to Analogue Converter)

El conversor DAC de este microcontrolador tiene dos salidas analógicas y convierte a unos valores tensión, valores enteros codificados en 8 bits o en 12 bits, con la posibilidad de especificar si los datos a convertir están alineados a la izquierda o a la derecha. Las características más destacables del DAC son:

- Posibilidad de generar ruido.
- Posibilidad de generar una forma de onda triangular.
- Conexión con el DMA.
- El inicio de conversión se puede indicar por software, se puede usar un *Timer* o una interrupción externa asociada a un pin como marcador del inicio de conversión.
- Posibilidad de habilitación de la interrupción global al final de la conversión.

4.3 TIM6 (Timer 6)

El TIM6 es un temporizador/contador de 16bits. La frecuencia de conteo depende de la configuración de reloj que se haya puesto para los *Timer* y del valor de prescaler interno del TIM6 que se haya configurado. Se suele usar como *Trigger* o disparador de eventos periódicos como puede ser el inicio de conversión del ADC y/o DAC o cualquier otra funcionalidad que se programe. Tiene la posibilidad de habilitar la interrupción global cuando el contador alcanza el número de cuentas que se le haya configurado (momento en el que el registro contador TIM6->CNT se resetea para poder volver a contar ascendentemente hasta el mismo valor que se le haya configurado), y así poder realizar funciones periódicas.

4.4 ST-LINK

La tarjeta NUCLEO-L432KC trae incorporado el depurador/programador ST-LINK y es el módulo que se encarga de realizar la carga en la tarjeta del software embebido que se haya programado mediante un conector Micro USB mediante el cual también se realiza la depuración y tiene la posibilidad de funcionar como un puerto Virtual COM.

5. Implementación del proyecto

El desarrollo o implementación del proyecto consta de varias fases de trabajo que se interconectan entre sí según el diagrama de flujo de la Figura 5**.**

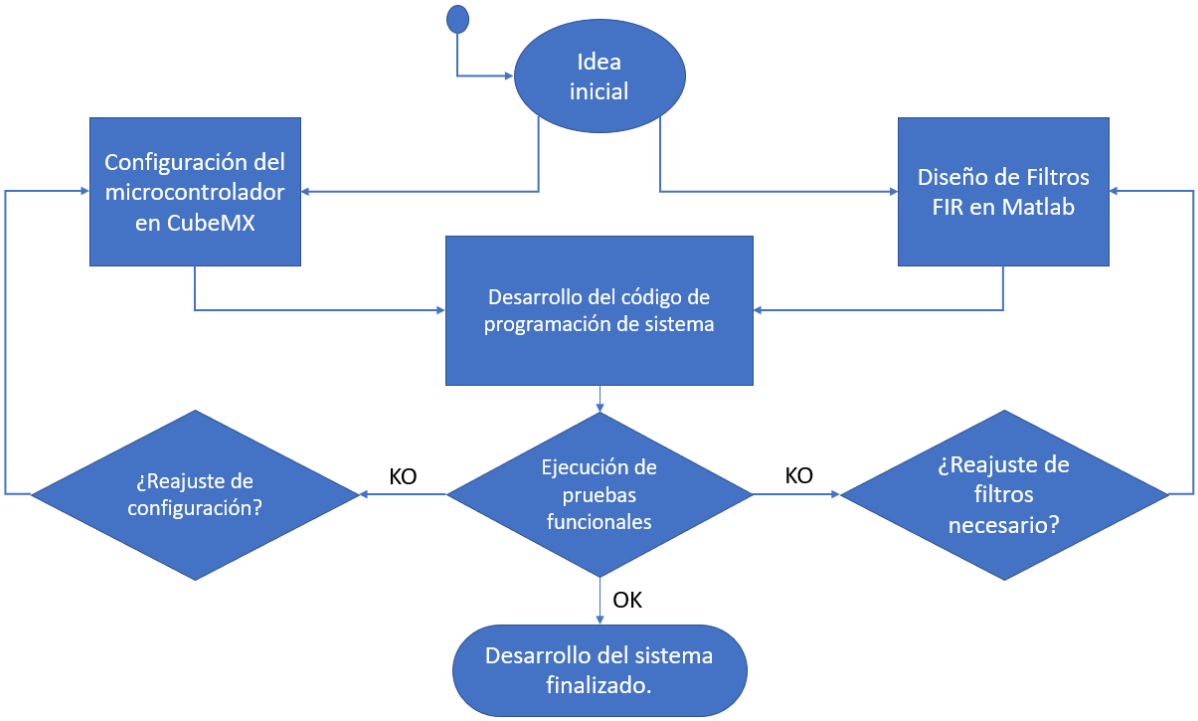

Figura 5. Flujo de trabajo para el desarrollo del proyecto.

Partiendo de una idea de diseño inicial, se procede a realizar la configuración del microcontrolador. Debido a que el microcontrolador que se usa es de la familia STM32 de STMicroelectronics se puede utilizar el software STM32CubeMX para realizar la configuración de pines, periféricos y fuentes de reloj del sistema.

Por otro lado, se debe realizar el diseño de los filtros FIR que se van a emplear en el proyecto. Esta tarea se realiza mediante una herramienta de diseño de filtros FIR e IIR que integra el software Matlab.

En el entorno de Keil uVision 5, se desarrolla el código de programación mediante el cual se va a realizar el filtrado en tiempo real.

Una vez terminado el código de programación se deben realizar pruebas funcionales para poder validar el correcto funcionamiento del sistema. En caso de que alguna de las pruebas que se realicen no concuerden con el resultado esperado, se deberán realizar reajustes de configuración del microcontrolador y/o de los filtros diseñados, y se debe revisar el código de programación en busca de posibles errores que hayan podido provocar que alguna de las pruebas no sea exitosa.

En caso de que todas las pruebas realizadas, fuesen correctas y acordes a los resultados esperados, se podrá validar que el sistema desarrollado funciona.

5.1 Idea de diseño del sistema

Para el desarrollo y/o implementación del proyecto, se parte de que la idea principal es poder procesar en tiempo real una señal analógica proveniente del exterior en un microcontrolador de bajo consumo.

Concretamente, el procesado que se ha pensado para este proyecto es el filtrado digital en tiempo real. Se quieren implementar cuatro tipos distintos de filtros FIR (paso bajo, paso alto, paso banda y banda eliminada) previamente diseñados.

Es necesario tener un módulo o sistema de adquisición de señales analógicas en el microcontrolador, es decir, es necesario un módulo ADC (Analog-to-Digital Converter) que permita obtener de manera proporcional el valor digital de un voltaje de entrada.

Para poder tratar y analizar la señal que se filtre en tiempo real, es necesario que en el microcontrolador haya un conversor D/A, de manera que las muestras digitales filtradas puedan ser sacadas por un pin como un valor de tensión proporcional a su valor digital.

Debido a que el diseño de los filtros FIR dependen de la frecuencia de muestreo que tenga el sistema, es necesario utilizar un temporizador que marque el inicio de conversión del ADC. Se escoge una frecuencia de muestreo de 8 KHz, de manera que se debe configurar el temporizador para que marque el inicio de conversión cada 125 µs y así poder obtener una muestra digital de la señal de entrada también cada 125 µs.

Figura 6. Tarea principal del sistema cada 125 microsegundos.

La señal de entrada a convertir se va a generar con la tarjeta mini-DK2 LPC1768, la salida del DAC de ese microcontrolador, se conecta a una de las entradas analógicas de la NUCLEO-L432KC, pudiendo regular la amplitud de la señal mediante un potenciómetro de 100KΩ. En la salida analógica del L432KC es donde se va hacer la comprobación de que el filtrado se está realizando correctamente con ayuda de una tarjeta de sonido externa (Tecknet) con un conector *jack* de 3.5mm de manera que con el software Visual Analyzer se pueda pueda ver tanto la forma de onda de la señal como su espectro.

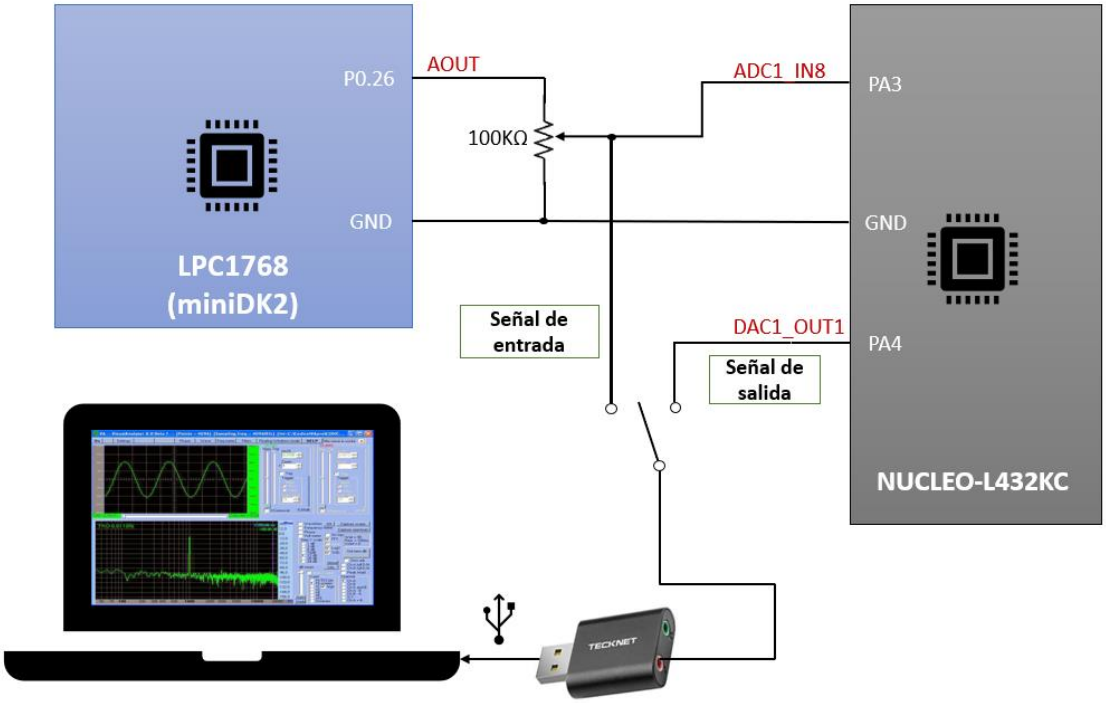

Figura 7. Esquema de bloques del conexionado del proyecto

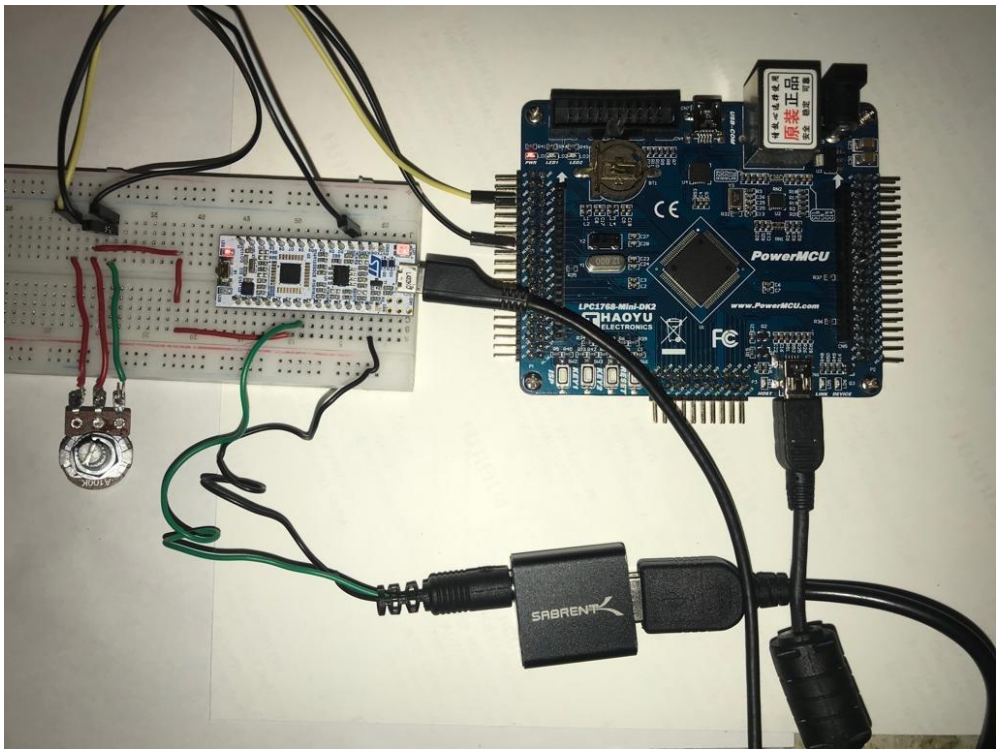

Figura 8. Montaje físico del sistema.

5.2 STM32CubeMX – Preconfiguración del proyecto

CubeMX es un software que se usa como herramienta de preconfiguración de microcontroladores de la familia de STM32, previo al desarrollo del sistema en Keil uVision.

En primer lugar, este software ofrece una primera interfaz de selección del microcontrolador a utilizar. Previo a la elección del microcontrolador, hay que tener cuenta que se puede escoger entre microcontrolador y tarjeta; la diferencia es que a la hora de escoger una tarjeta hay pines que traen una preconfiguración por defecto en base al diseño electrónico de periféricos que se haya realizado para esa tarjeta con el microcontrolador en cuestión; mientras que si se escoge solamente el microcontrolador, el usuario tiene libertad para asignar las funcionalidades a los pines ya que ninguno de estos viene preconfigurado a excepción de los que son NRST, *GND, 3V3* y *5V*. En este caso, procede escoger la tarjeta "NUCLEO-L432KC".

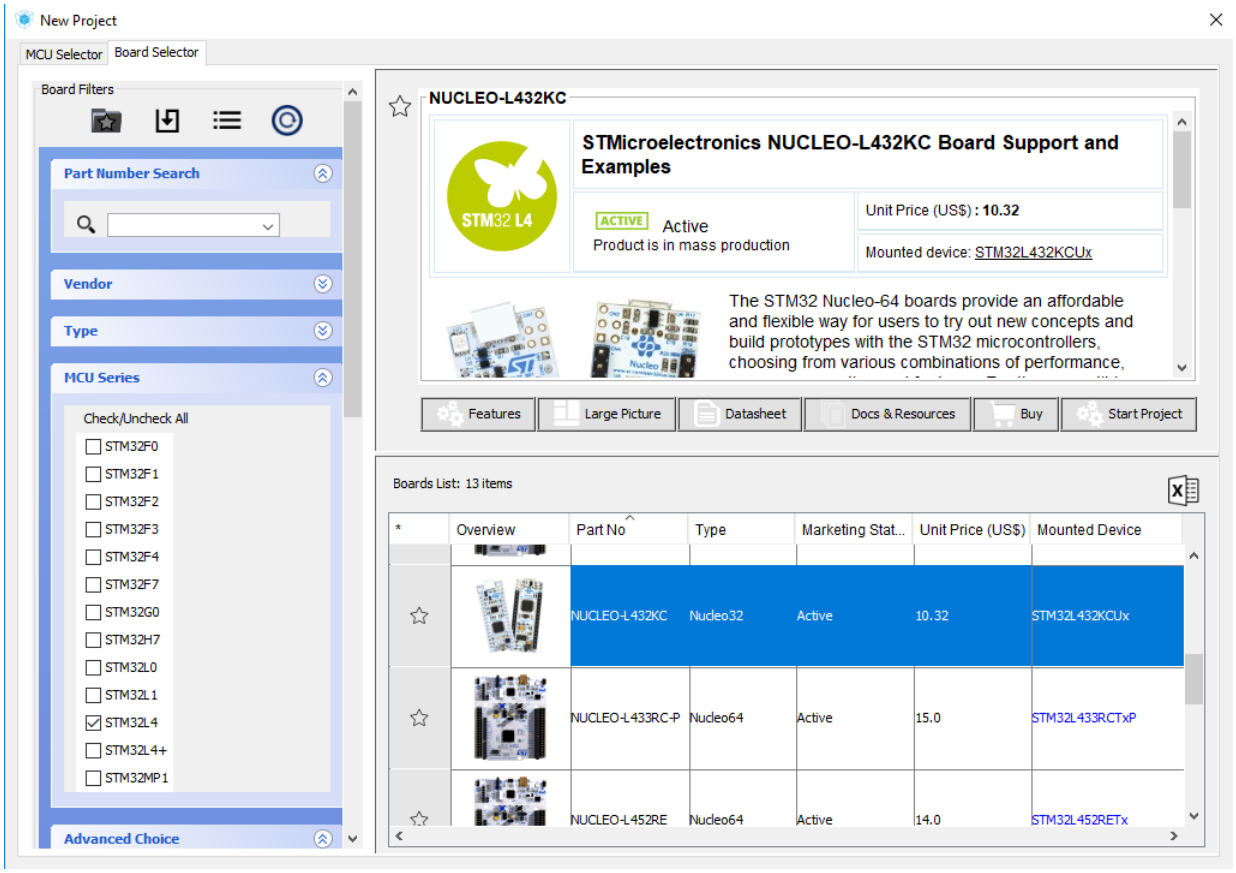

Figura 9. Listado de Tarjetas de STM32.

La Figura 9, muestra la interfaz de usuario para la elección del dispositivo, y como se puede comprobar, se puede realizar un filtrado por serie de los y microcontroladores y tarjetas existentes de la familia STM32 facilitando la búsqueda al usuario.

Una vez escogida la tarjeta, Cube muestra una interfaz como la de la Figura 10, a través de la cual se pueden configurar los periféricos y pines asociados a éstos de la tarjeta en cuestión. Como se puede ver en el panel izquierdo, hay algunos periféricos que se muestran en color rojo y otros con un símbolo de advertencia. Los que aparecen en color rojo, son aquellos que no se pueden utilizar debido a que existe conflictos bloqueantes con los pines que usan los periféricos preconfigurados de la tarjeta, mientras que el símbolo de advertencia indica que hay ciertas funcionalidades del periférico que no se

pueden utilizar por conflicto de pines, pero no impide la utilización del mismo para otras funcionalidades.

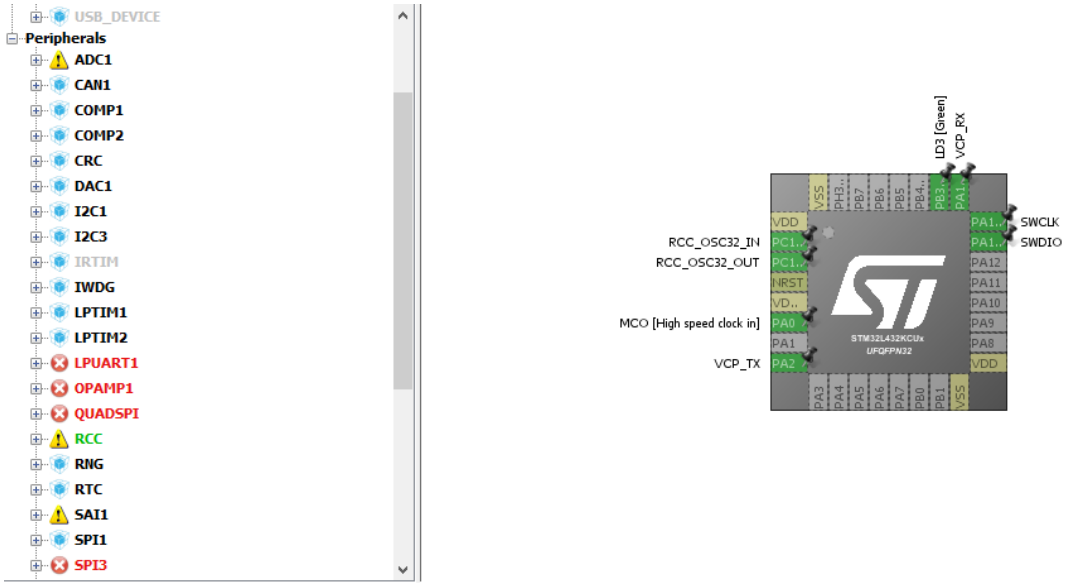

Figura 10. Pinout inicial Nucleo-L432kc

Volviendo al contexto del sistema a desarrollar, para el procesado digital de señales analógicas es necesario el uso de un temporizador que marque el inicio de conversión del ADC a una frecuencia de 8KHz para poder así poder realizar un filtrado en tiempo real y sacar el resultado de las muestras filtradas por el DAC. Los periféricos que se deben configurar son *ADC1, DAC1* y *TIM6.* Se escoge el pin PA3 como entrada analógica para el canal 8 del ADC1 en modo "*Single-ended*" y el pin PA4 como salida analógica para el canal 1 del DAC1.

A la hora de manejar los tiempos de interrupción y frecuencias de conversión, es necesario saber la frecuencia de reloj que llega a cada uno de los periféricos escogidos, esta información viene proporcionada en la pestaña "*Clock Configuration*" tal y como se muestra en la Figura 11.

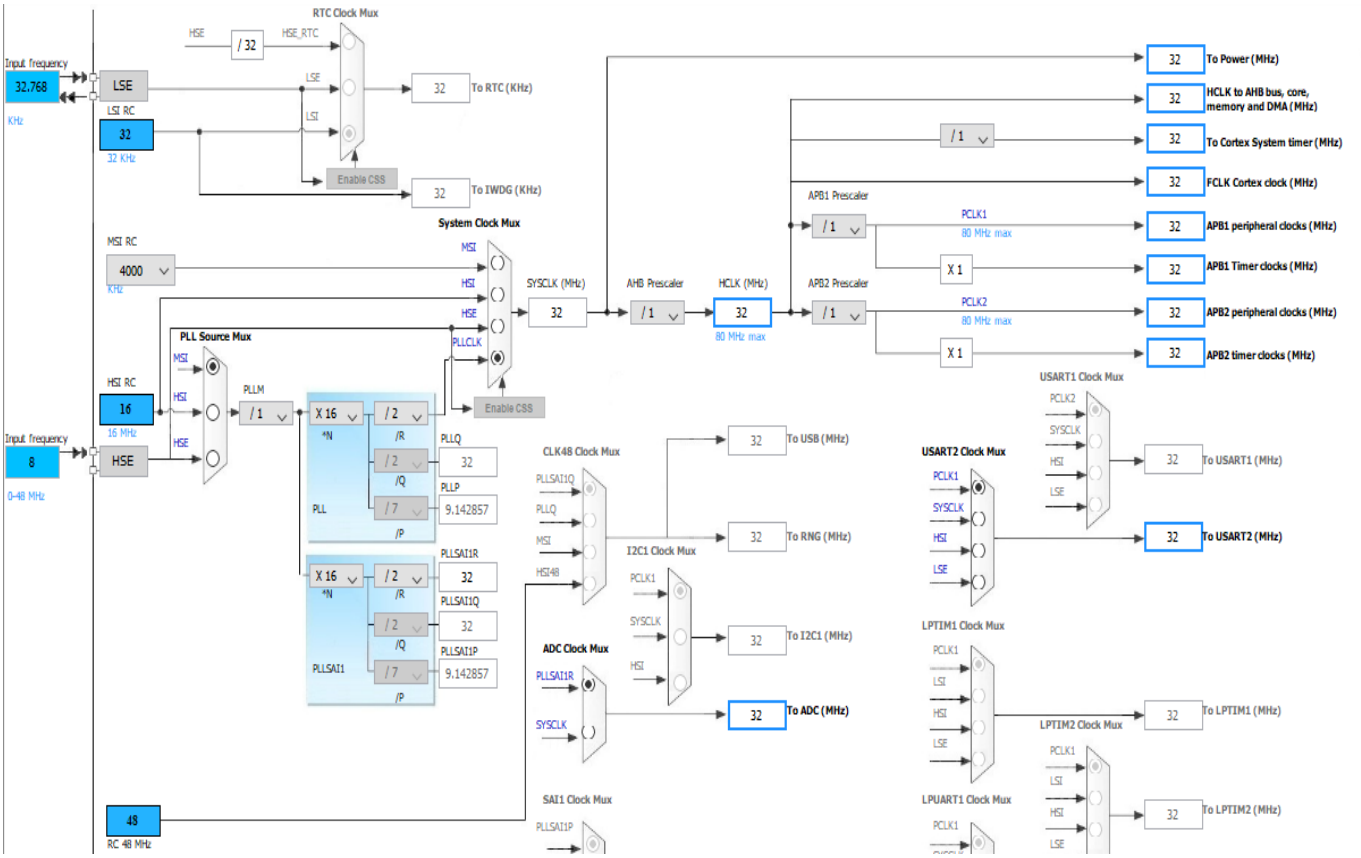

Figura 11. Configuración del reloj y frecuencias de los periféricos

Teniendo en cuenta la frecuencia que llega a cada periférico, se procede a configurar cada uno de estos:

- **TIM6:** Es el encargado de marcar el inicio de conversión del ADC1. El sistema se diseña para que el muestreo de las señales analógicas sea 8KHz. De la Figura 11 se ve que la frecuencia que llega a los *timer* es de 32MHz. Se debe configurar el periodo de cuentas del TIM6 con un valor de 3999 cuentas, ya que la frecuencia de conteo es 32MHz y la frecuencia de muestreo deseada es 8KHz →32MHz/8KHz=4000 cuentas, si se tiene en cuenta que el registro contador CNT de los timer cuenta desde '0' hasta el valor que se le indique, al cálculo anterior de cuentas habría que restarle una unidad, para que la frecuencia de muestreo sea exacta. Teniendo en cuenta que el TIM6 es el encargado de marcar el inicio de conversión, se debe indicar que al alcanzar el registro CNT el valor de cuentas indicado, el TIM6 debe actualizar un evento. Debido a que el diseño del sistema no lo requiere, no se habilita la interrupción global del TIM6.

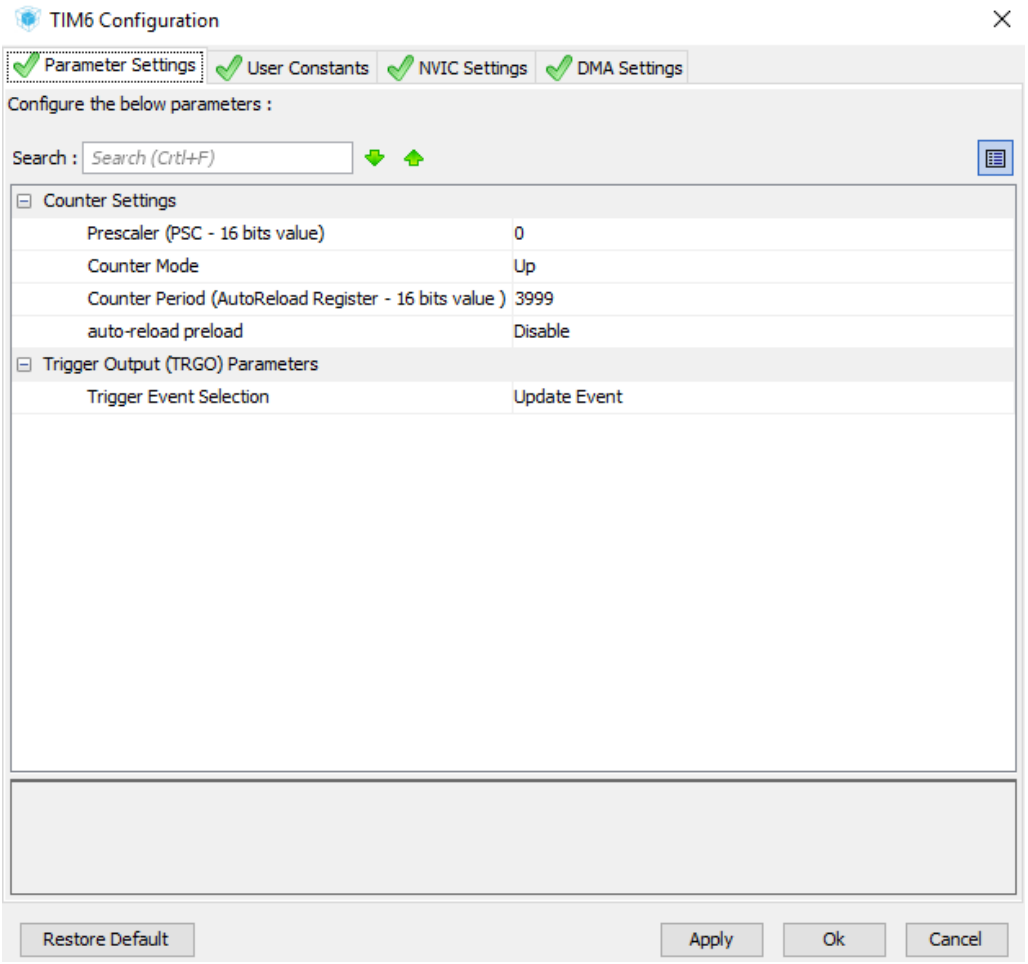

Figura 12. Configuración del tim6.

- **ADC1:** Es el encargado de convertir la señal analógica del canal 8 del ADC1, en muestras digitales. El inicio de conversión lo marca el TIM6 como se ha configurado previamente y se debe habilitar la interrupción global del ADC1 debido, a que en la rutina de atención a esta interrupción es donde se va a realizar el filtrado de cada muestra, y el envío de estas (una vez filtradas) al DAC1.

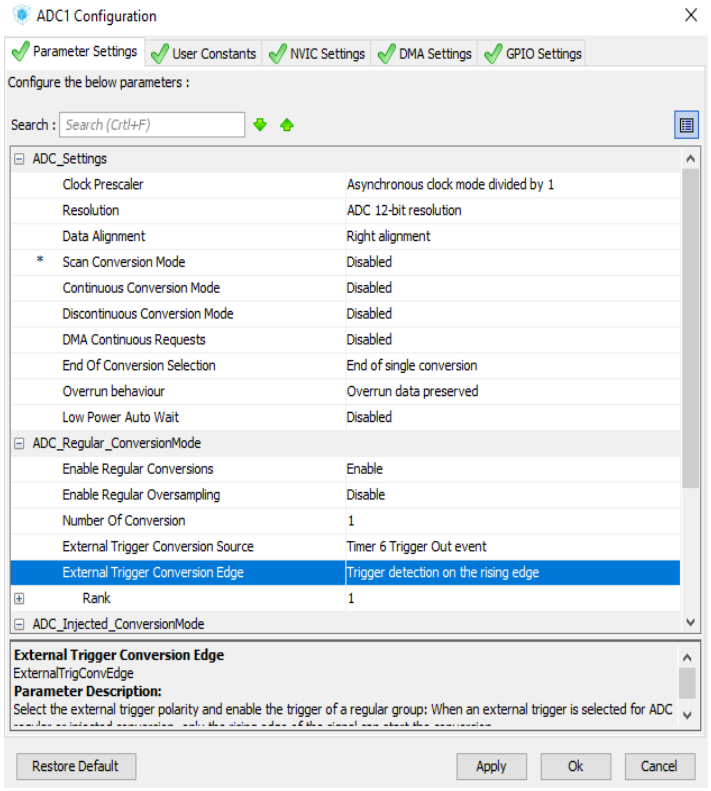

Figura 13. Configuración del ADC1 – Parameter Settings

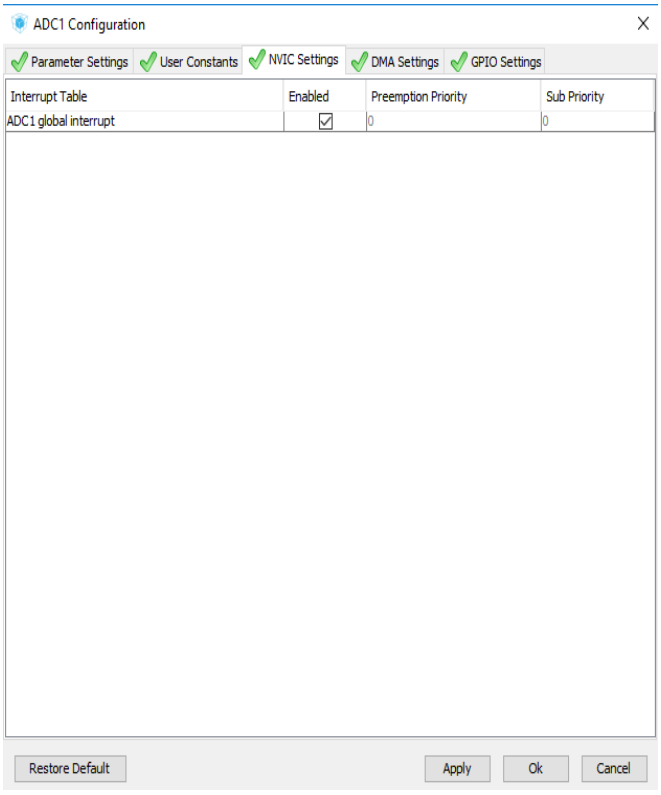

Figura 14. Configuración del ADC1 – NVIC Settings

- **DAC1:** este periférico solo requiere de la habilitación del buffer de salida y no se habilita la interrupción global del mismo, ya que el sistema diseñado no lo requiere.

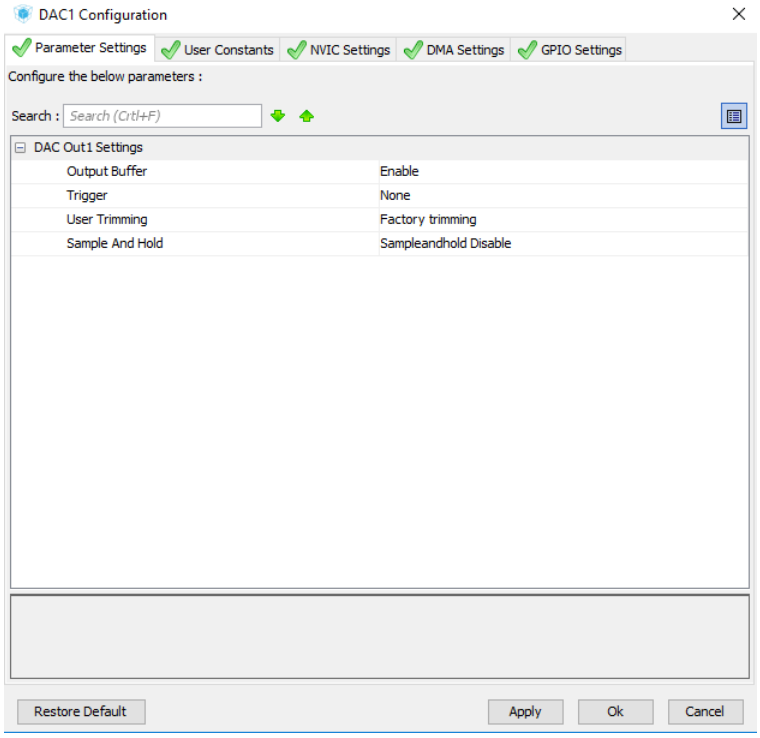

Figura 15. Configuración del DAC1.

Una vez realizadas las configuraciones de los periféricos, se procede a generar el código de programación en C de todo lo configurado en CubeMX. Para ello es importante indicar el entorno de desarrollo del proyecto tal y como se muestra en la Figura 16, en este caso, es el software Keil uVision5.

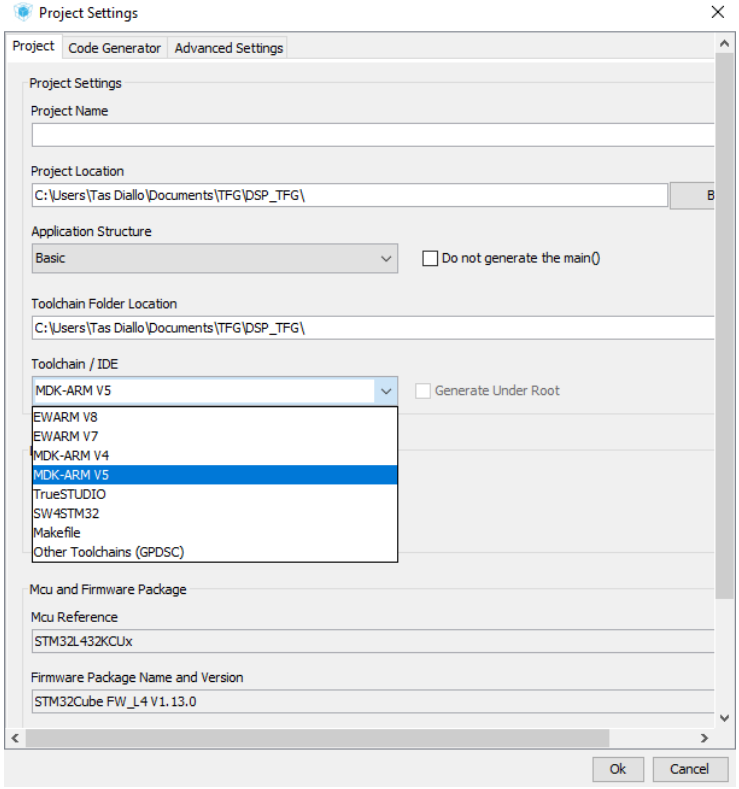

Figura 16. Ventana de configuración para generación del código en c.

Con esta configuración CubeMX, genera un proyecto organizado para el entorno de Keil uVision 5 con un sistema de carpetas y ficheros de texto, entre otros con extensión ".s", ".c" y ".h".

5.3 Generación de la señal de entrada

Es necesario tener una fuente de generación de señales, en este caso senoidales para poder validar el correcto funcionamiento de los filtros que se hayan implementado en la tarjeta NUCLEO-L432KC. Para no tener la necesidad de estar en un laboratorio para generar señales con un generador de señales, se opta por utilizar la tarjeta de desarrollo miniDK2 con el microcontrolador LPC1768 como fuente de generación de señales.

Entre otras características la tarjeta miniDK2 tiene un módulo conversor de digital a analógico DAC y 4 *Timers* y entradas de interrupción externas, que son los módulos de esta tarjeta que van a ser necesarios para generar señales periódicas.

5.3.1 Configuración de interrupciones externas – miniDK2 (LPC1768)

Se utiliza una función de configuración de interrupciones externas con el fin de poder cambiar la frecuencia de salida de la señal senoidal, de manera que con los botones "ISP" y "KEY2", se pueda bajar y subir la frecuencia respectivamente en saltos de 50 Hz.

La función `init_EINTs()` se encarga de realizar la configuración de las interrupciones externas y de los pines asociados a éstas. En la tarjeta miniDK2 los botones "ISP" y "KEY2" están unidos respectivamente con los pines P2.10 y P2.11.

```
void init_EINTs(void){
    LPC_PINCON->PINSEL4|=(0x01<<20); // P2.10 es entrada interrup. EXT 1
    LPC_PINCON->PINSEL4|=(0x01<<24); // P2.11 es entrada interrup. EXT 2
    LPC_SC->EXTMODE|=(1<<2)|(1<<0);   // Por Flanco
    LPC_SC->EXTPOLAR=0;              // de bajada


    NVIC_SetPriority(EINT2_IRQn, 0);            // M□prioritario
    NVIC_SetPriority(EINT0_IRQn, 0);            // M□prioritario

    NVIC_EnableIRQ(EINT2_IRQn); // sin CMSIS: NVIC->ISER[0]=(1<<19);
    NVIC_EnableIRQ(EINT0_IRQn); // sin CMSIS: NVIC->ISER[0]=(1<<20);
}
```

En primer lugar, se especifica con el registro PINSEL4 la funcionalidad de interrupción externa para los pines mencionados anteriormente. Se configura que la manera de interrumpir es por flanco de bajada para ambos pines.

Se configura una prioridad '0' para ambas interrupciones, es decir, mayor prioridad que la interrupción del Timer1 ya que, en este diseño, es más prioritario que el usuario pueda cambiar la frecuencia de la señal siempre que quiera.

Las rutinas de atención a la interrupción de estas interrupciones externas se encargan de reconfigurar el Timer1 pasándole por parámetro a su función de configuración la frecuencia deseada. En este caso la frecuencia deseada es la actual, aumentada o disminuida en 50 Hz, en función de si el botón pulsado es "KEY2" o "ISP" respectivamente.

```
void EINT0_IRQHandler(){
        LPC_SC->EXTINT|=(1<<0);
        if (F_out>50){
            F_out-=50;
            init_TIMER1(F_out);
            LPC_TIM1->TCR = 0x2;//  Reset Timer
            LPC_TIM1->TCR   =   0x1;//  Start Timer
        }
}

void EINT2_IRQHandler(){
        LPC_SC->EXTINT=(1<<2);

        if (F_out<450){
            F_out+=50;
            init_TIMER1(F_out);
            LPC_TIM1->TCR = 0x2;//  Reset Timer
            LPC_TIM1->TCR = 0x1;//  Start Timer
            }
}
```

5.3.2 Configuración del TIM1 – miniDK2 (LPC1768)

En primer lugar, es necesario escoger uno de los 4 *Timers* como referencia de tiempos para generar señales periódicas. Se escoge el TIMER1 y se realiza el código de configuración de éste:

```
void init_TIMER1(uint16_t Freq)
{
    LPC_SC->PCONP|=(1<<2);                          //  Power ON
    LPC_TIM1->TCR = 0x00;                           //  Timer Stopped
    LPC_TIM1->PR = 0x00;                          //  Prescaler =1
    LPC_TIM1->MCR = 0x03;           //  Reset TC on Match, e interrumpe!
    LPC_TIM1->MR0 = (F_pclk/Freq/N_muestras)-1;  // Cuentas hasta el Match
    LPC_TIM1->EMR = 0x00;                         //  No acta sobre el HW
    LPC_TIM1->TCR = 0x01;                             //  Start Timer
    NVIC_EnableIRQ(TIMER1_IRQn);              //  Habilita NVIC
    NVIC_SetPriority(TIMER1_IRQn,1);  //  Nivel 1 prioridad
}
```

Se utiliza una función exclusiva para configurar el TIMER1, llamada en este caso `void init_TIMER1(uint16_t Freq)`. En primer lugar, se debe activar o encender este periférico con el registro PCONP. Teniendo en cuenta que la frecuencia que llega a este periférico es de 25 MHz, la cual viene definida con nombre de "Fpclk", se configura el registro de prescaler (divisor de frecuencia) LPC_TIM1->PR con un valor de '0' de manera que la frecuencia de conteo que lleva el registro LPC_TIM1->TC sea de 25 MHz también.

Teniendo en cuenta que el registro contador, lleva una cuenta ascendente, se debe configurar el valor máximo de cuentas al que se quiere que cuente teniendo en cuenta que este valor se debe configurar en un registro de 32 bits con todos los bits dedicados exclusivamente al valor máximo de cuentas. El registro en el que se configura este valor es el registro LPC_TIM1->MR0, y como se puede observar en la función de configuración se realiza una operación en la que se divide la frecuencia del TIM1 con la frecuencia deseada "Freq" (recibida como parámetro de la función), de la señal senoidal y el número de muestras por ciclo que se desee de la señal senoidal, obteniendo así, un número adimensional de cuentas.

A continuación, se debe especificar lo que debe hacer el TIM1 una vez que su registro contador alcanza el número de cuentas configurado. El registro en el que se consulta la tarea a realizar es el registro LPC_TIM1->MCR, y en este caso se configura de tal manera que cuando se alcance el número de

cuentas, el TIM1 interrumpa y su registro contador resetee su valor de cuentas a '0' para poder volver a empezar a contar.

Ya que se ha configurado que el TIM1 interrumpa, se debe habilitar su rutina de atención a la interrupción y se establece con una prioridad '1'.

5.3.3 TIMER1_IRQHandler – miniDK2 (LPC1768)

Se define la rutina de atención a la interrupción del TIM1 como TIMER1_IRQHandler, y se programa de tal manera que sea donde se va a dar la orden de sacar las muestras almacenadas en el array global por el DAC, sacando una muestra por cada interrupción, volviendo a empezar de nuevo periódicamente una vez se hayan sacado todas:

```
void TIMER1_IRQHandler(void)
{
    static uint16_t indice_muestra;
    LPC_TIM1->IR|= (1<<0);                          // borrar flag
    LPC_DAC->DACR= muestras[indice_muestra++] << 6; // bit6..bit15
    indice_muestra&= N_muestras-1;// contador circular (Si N_muestras
potencia de 2)
}
```

5.3.4 Configuración del DAC – miniDK2 (LPC1768)

El DAC es el periférico por el cual se va a obtener la señal analógica de salida y se debe configurar de tal manera que el pin P0.26 este configurado como salida analógica, sin resistencia de pull-up o pull-down y se debe resetear el registro de control del DAC:

```
void init_DAC(void)
{
    LPC_PINCON->PINSEL1|= (2<<20);      // DAC output = P0.26 (AOUT)
    LPC_PINCON->PINMODE1|= (2<<20);     // Deshabilita pullup/pulldown
    LPC_DAC->DACCTRL=0;                             //
}
```

5.3.5 Función "genera_muestras(uint16_t muestras_ciclo)" – miniDK2 (LPC1768)

Se utiliza una función dedicada a generar muestras digitales de señal senoidal, para así poder sacarlas por el DAC. El código de esta función es el siguiente:

```
void genera_muestras(uint16_t muestras_ciclo)
{
    uint16_t i;
    //se□senoidal
    for(i=0;i<muestras_ciclo;i++)
    muestras[i]=(uint32_t)(511+511*sin(2*pi*i/N_muestras));
}
```

Esta función recibe como parámetro una variable entera sin signo de 16 bits que indica el número de muestras por ciclo que tiene una señal senoidal. Mediante un array declarado de manera global muestras[] se guarda generan mediante un bucle "for" todas las muestras, quedándose almacenadas cada una en su respectiva posición del array.

Teniendo en cuenta que el DAC de este microcontrolador es de 10 bits y que el resultado de la conversión no puede tener valores de tensión negativos, en el bucle donde se generan las muestras se debe aplicar un *offset* a la mitad de su rango con un valor de $2^{10}/2$ -1 = 511, que correspondería con un valor analógico de tensión de 1.65V ya que el rango de tensión de salida del DAC es de 3.3V. Aplicando una amplitud al seno de 511, se consigue ajustar la señal de salida aprovechando todo el rango de tensiones permitido.

5.3.6 Señal de salida – miniDK2 (LPC1768)

Se explica en este apartado el funcionamiento global del software embebido que se ha cargado en la tarjeta miniDK2 para la generación de la señal senoidal.

Se utiliza una interrupción periódica del TIM1 cuya frecuencia de interrupción se puede modificar en función de la frecuencia de la señal que se desee. En la rutina de atención a la interrupción es donde se va a sacar cada una de las muestras que haya almacenadas en el array de muestras.

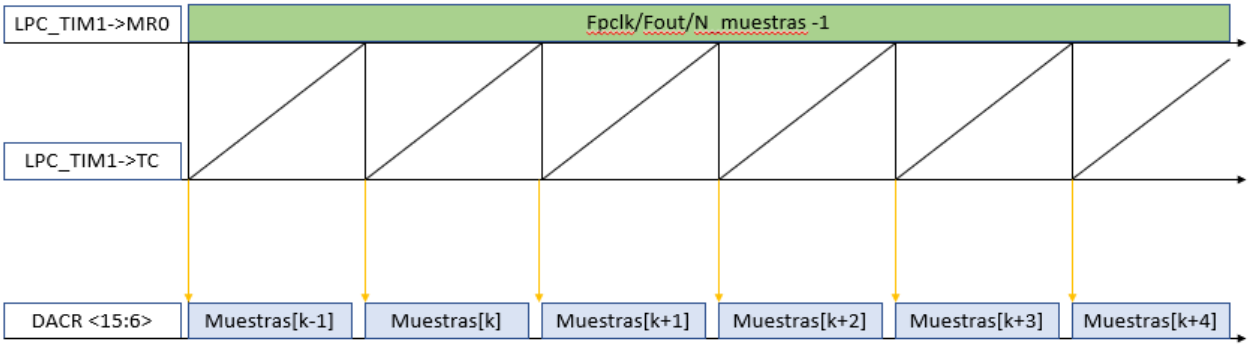

Figura 17. Cronograma de generación de la señal senoidal.

En la siguiente figura se muestra en simulación, en el entorno de desarrollo de Keil uVision 4, la señal que saldría por el pin P0.26 del LPC1768, configurando una frecuencia de la señal sinusoidal de 200 Hz por ejemplo.

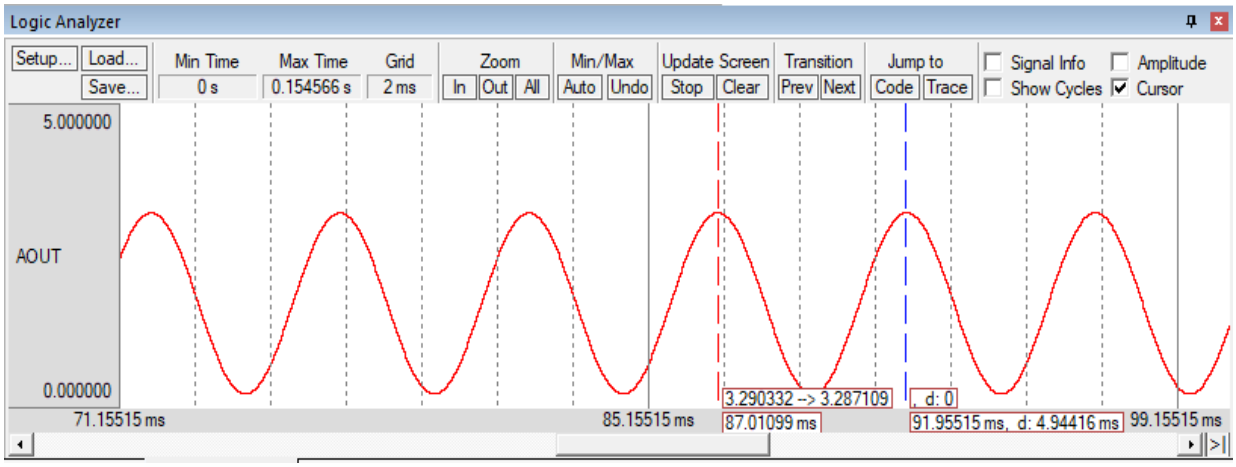

Figura 18. Simulación de la señal de entrada al sistema.

5.4 Diseño y generación de filtros FIR con Matlab

Previo al diseño de cada uno de los filtros, es preciso destacar algunos aspectos acerca de los filtros FIR y del filtrado en general.

Un filtro FIR es un tipo de filtro digital cuya respuesta a una entrada impulso de entrada tendrá un número finito de términos no nulos.

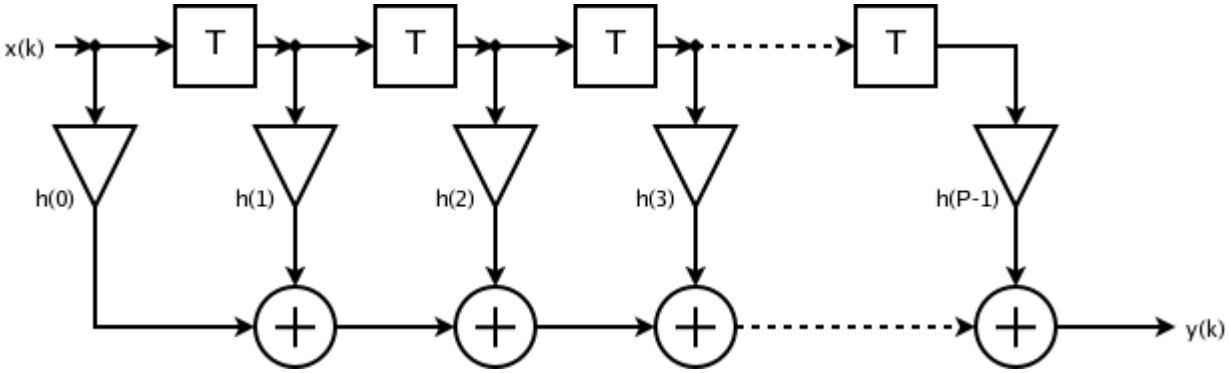

Figura 19. Esquema general de filtros FIR

En el esquema de la Figura 19 se representa el esquema general que tiene un filtro FIR. La señal de entrada al filtro debe ser una señal discreta al tratarse de un filtro digital, y como se puede observar en la Figura 19, salida actual y(k) depende del sumatorio de las multiplicaciones de los coeficientes del filtro con las muestras anteriores y actual de la señal de entrada. La correspondencia entre muestra y coeficiente en las multiplicaciones se define de tal manera que la muestra más antigua se multiplica con el último coeficiente, y la muestra actual con el primer coeficiente. De manera analítica, la salida actual del filtro tiene la siguiente expresión:

$$y_n = \sum_{k=0}^{N-1} b_k x_{n-k}$$

Ecuación 1. Expresión analítica de un filtro FIR.

En la expresión que se muestra en la Ecuación 1, la salida actual se representa como y_n, b_k corresponde a los coeficientes del filtro y x_n corresponde a la muestra actual.

Con respecto a los filtros IIR (filtro de respuesta infinita al impulso), el filtro FIR tiene la desventaja de que, para un mismo tipo de filtrado, por ejemplo, paso bajo, el filtro FIR genera muchos más coeficientes que un filtro IIR, traduciéndose esto, en una mayor carga computacional. A pesar de esta desventaja, se decide optar por el uso de filtros FIR debido a que son estables, ya que tienen todos los polos en el origen y además se pueden diseñar para que tengan una fase lineal.

En el contexto del sistema diseñado, se ha fijado que la frecuencia con la que se muestrea la señal analógica de entrada va a ser 8 KHz. Teniendo en cuenta el Teorema de Nyquist-Shannon acerca del muestreo de señales, se establece que la frecuencia de muestreo debe ser al menos el doble de la frecuencia de la señal a muestrear. Es decir, que para el sistema diseñado las señales a muestrear no deberán tener frecuencias superiores a 4 KHz. Se sabe que de una señal se tiene más información (muestras) de ella cuanto mayor es la frecuencia de muestreo. Teniendo en cuenta esto último, se ha establecido para este proyecto que la frecuencia máxima de señal que se va a admitir va a ser de 500 Hz, que con una frecuencia de muestreo de 8 KHz, se van a tener 16 muestras por periodo. El diseño de los filtros se ve condicionado a que las frecuencias de corte que se establezcan tampoco superen

esa frecuencia de 500 Hz ya que, a la hora de realizar pruebas funcionales del sistema, los resultados podrían resultar confusos y poco fiables.

La frecuencia de muestreo es un parámetro en el diseño de filtros FIR que de manera directa influye en el número de coeficientes generados. Es decir, cuanto mayor es la frecuencia de corte, más coeficientes se generan. Teniendo en cuenta que el filtro se va a implementar en un sistema embebido filtrando en tiempo real y que el margen de tiempo que existe para filtrar es 1/fs, donde fs es la frecuencia de muestreo, puede llegar a suceder que para un número N de coeficientes, el microcontrolador no haya podido terminar de filtrar la muestra k, para cuando ya tiene disponible la muestra k+1. Por esta razón a la hora de configurar el filtro en Matlab siempre se va a marcar la opción de generar el mínimo número de coeficientes posible.

Por otro lado, hay que tener en cuenta que las bandas de transición (zona de cambio entre frecuencia de paso y frecuencia de parada o viceversa) hacen que el filtrado sea más eficaz cuanto más abruptas son. El inconveniente de tener bandas de transición abruptas es que el número de coeficientes del filtro FIR que se genere aumentará considerablemente con respecto a un filtro con una banda de transición menos pronunciada, pudiendo ser éste un problema, ya que el tiempo disponible para filtrar cada muestra, con una frecuencia de muestreo de 8 KHz, es de 125 µsegundos. Se establece, para este sistema que todas las bandas de transición de todos los filtros que se diseñen tengan al menos 100 Hz de anchura.

5.4.1 Herramienta "filterDesigner" - Matlab

A continuación, se explica cómo hacer el diseño y la generación de distintos tipos de filtros FIR en Matlab. Para empezar, se necesita la herramienta para el diseño de filtros con Matlab. Desde la consola de comandos que tiene Matlab se introduce el comando "*filterDesigner*" y se abre la herramienta para diseñar tanto filtros FIR como IIR como se muestra en la Figura 20.

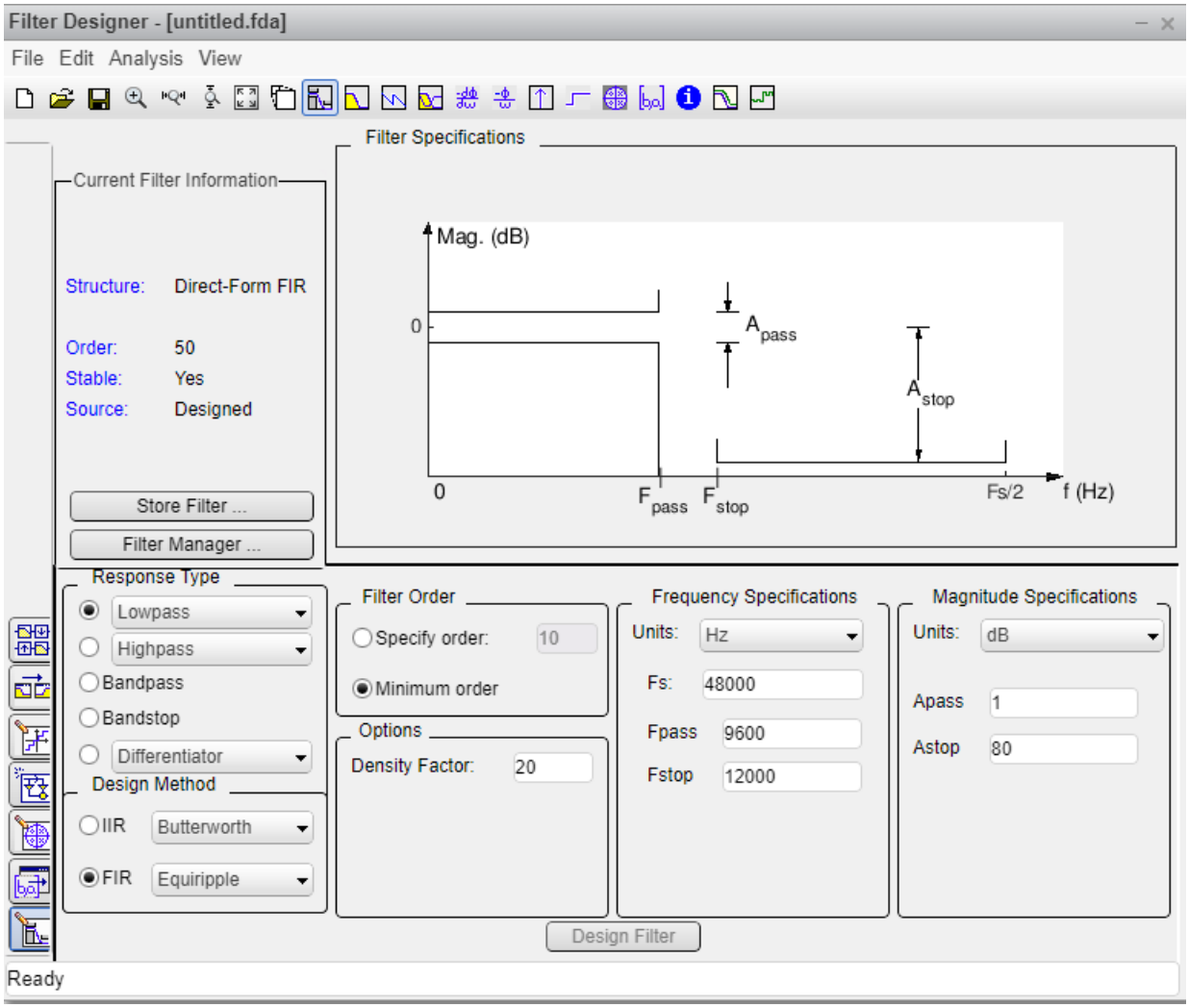

Figura 20. Ventana de diseño de filtros IIR y FIR.

En la mitad inferior de la ventana es la parte en la que se van a especificar los parámetros de diseño de cada filtro.

Por un lado, se puede apreciar que se puede escoger entre varios tipos de filtros (paso bajo, paso alto…). A continuación, se debe especificar el método de diseño, es decir, si el filtro que se quiere diseñar es IIR (Infinite Impulse Resonse) o FIR (Finit Impulse Response).

El siguiente paso es especificar, el número de coeficientes que se desea, por defecto, se genera el número mínimo de coeficientes.

Lo siguiente que se debe de especificar son las bandas de paso y de parada, y por último se deben especificar los parámetros de magnitud que se desea para las bandas de paso y de parada.

En la Figura 21 se muestra un ejemplo de diseño de filtro FIR paso bajo con una frecuencia de muestreo 48 KHz con una banda de paso de 1 KHz y una banda de parada de 2 KHz, manteniendo los parámetros que vienen por defecto de magnitud.

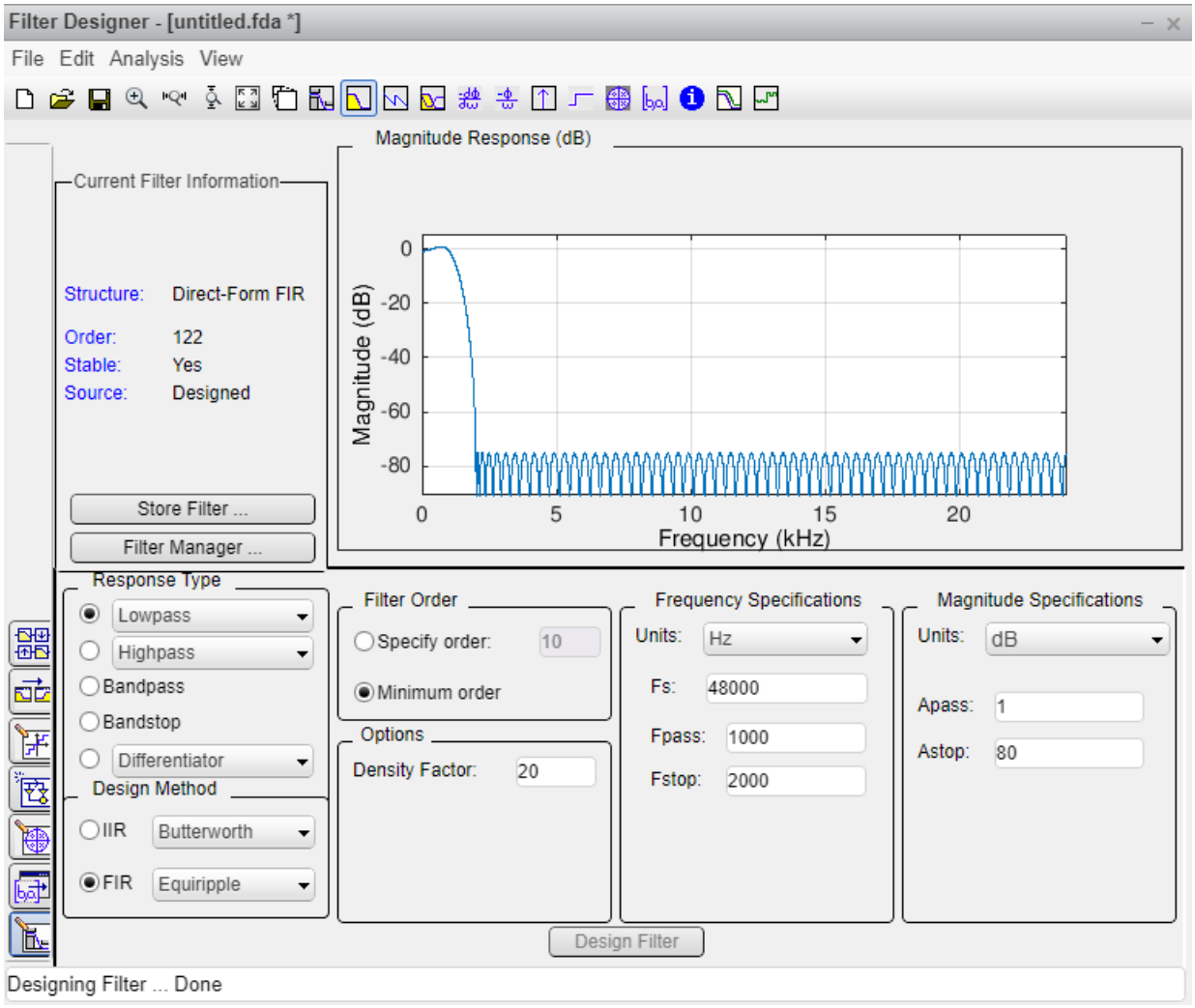

Figura 21. Ejemplo de diseño de filtro FIR.

Como se puede observar en la figura anterior, en la parte superior de la ventana de diseño de filtros, se muestra por un lado el número de coeficientes que tiene el filtro, y por otro lado la representación de la magnitud del filtro en función de la frecuencia. Esta representación indica la atenuación en dB que tiene una señal en función de la frecuencia que tenga. A esta representación se le puede añadir la respuesta de la fase quedando, en este caso, como la Figura 22**.**

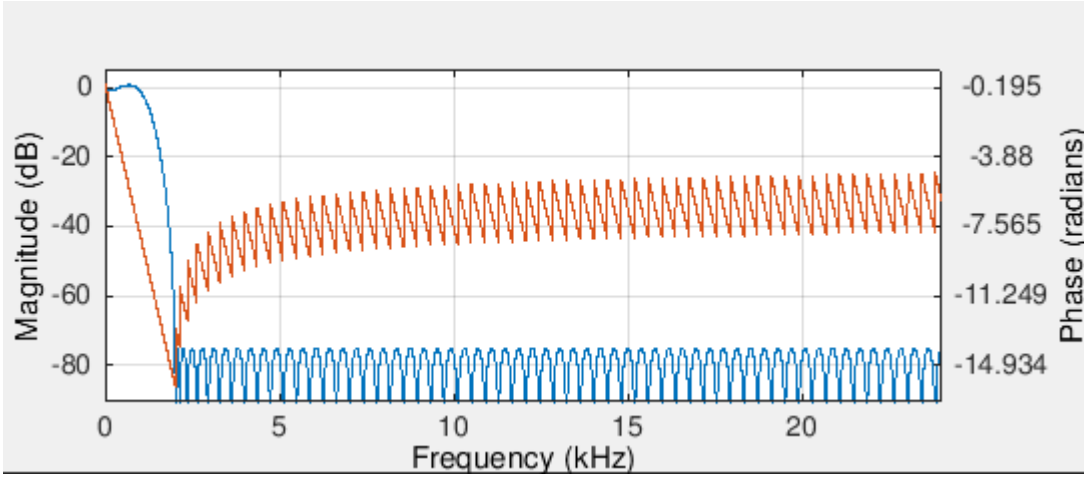

Figura 22. Representación de la magnitud y la fase de un filtro FIR.

Para obtener los coeficientes del filtro diseñado, se debe escoger la opción “Export” del menú “File” de la barra superior de la ventana de diseño de filtros.

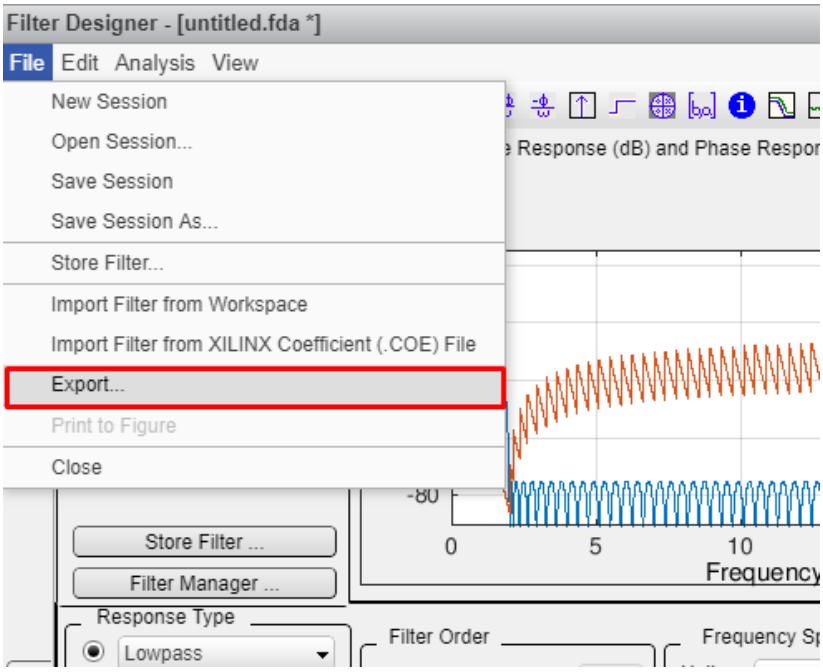

Figura 23. Exportar coeficientes de un filtro – Paso 1

El siguiente paso es especificar la manera en la que se quiere exportar los coeficientes. En este caso, se escoge como opción, exportar al *workspace* de Matlab como una variable de coeficientes con nombre “Num”, tal y como se muestra en la Figura 24.

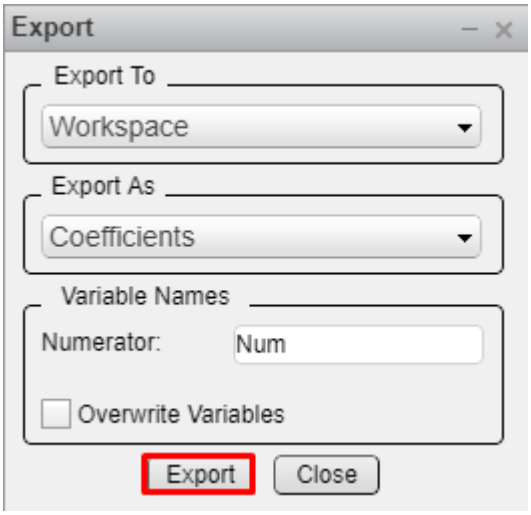

Figura 24. Exportar coeficientes de un filtro – Paso 2

Automáticamente en el *workspace* de Matlab se genera un array de tipo double con una dimensión igual al número de coeficientes que tenía el filtro diseñado. El último paso consiste en coger este fichero array y llevarlo a un fichero de texto plano donde los elementos del array estén separados por comas. Para conseguir esto último, se escribe el comando “csvwrite(‘Coeffs.txt’, Num)” que de manera automática crea un fichero de texto con extensión “.txt” con en el nombre que se le haya indicado (en este caso “Coeffs”), y con los coeficientes separados por comas en el directorio donde se esté trabajando actualmente.

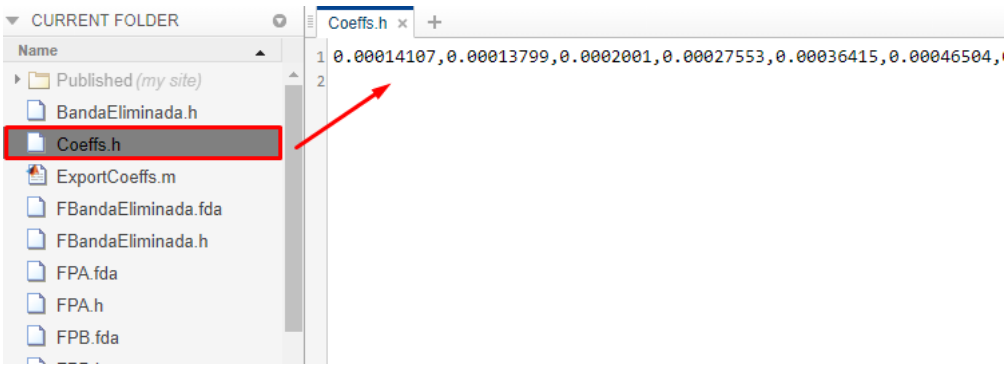

Figura 25. Exportar coeficientes de un filtro – Paso 3

5.4.2 Filtro FIR Paso Bajo - Diseño

El primer filtro que se quiere obtener es un filtro FIR Paso Bajo. En la especificación del orden del filtro, se va a seleccionar la opción de que el orden sea el más pequeño posible. En la especificación de frecuencia, se debe indicar que la frecuencia de muestreo es de 8KHz, que la banda de paso es hasta 100 Hz y que la banda de parada es hasta los 400 Hz. No se modifican los parámetros de magnitud que vienen por defecto ya que son válidos para el sistema diseñado. Por último, se da la orden de diseñar el filtro, obteniendo la representación del filtro diseñado tal y como se muestra en la Figura 26.

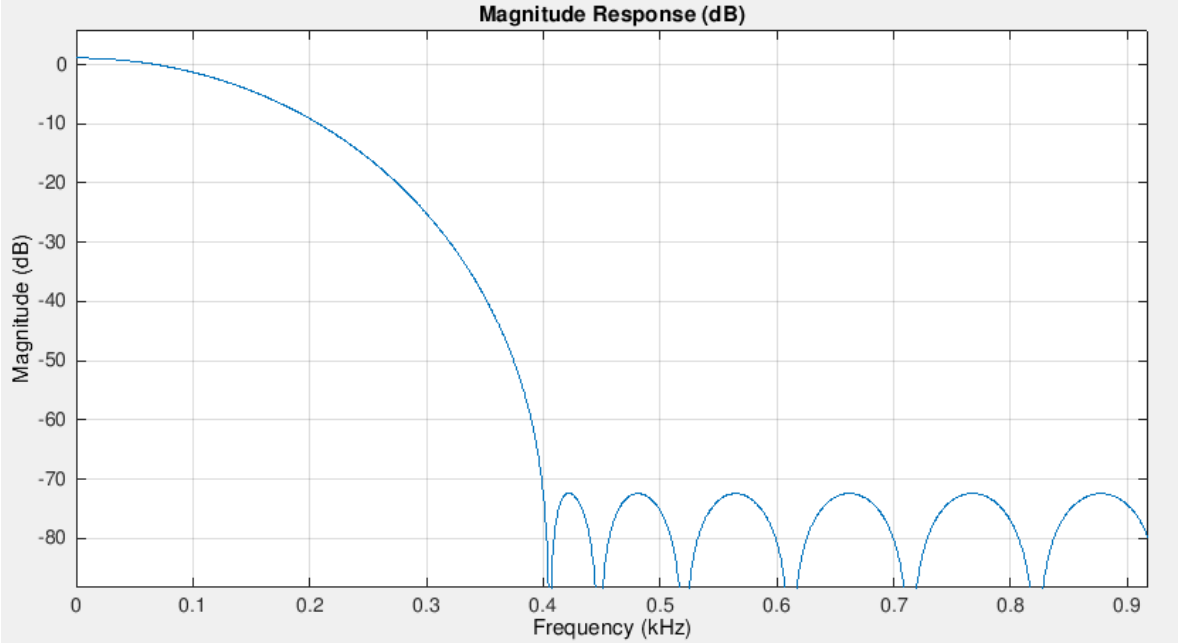

Figura 26. Filtro FIR paso bajo diseñado

5.4.3 Filtro FIR Paso Alto – Diseño

Se configura la frecuencia de muestro a 8 KHz, una banda de parada hasta los 100 Hz y una banda de paso de paso a partir de 400 Hz, y en este caso se obtiene la representación del filtro de la Figura 27.

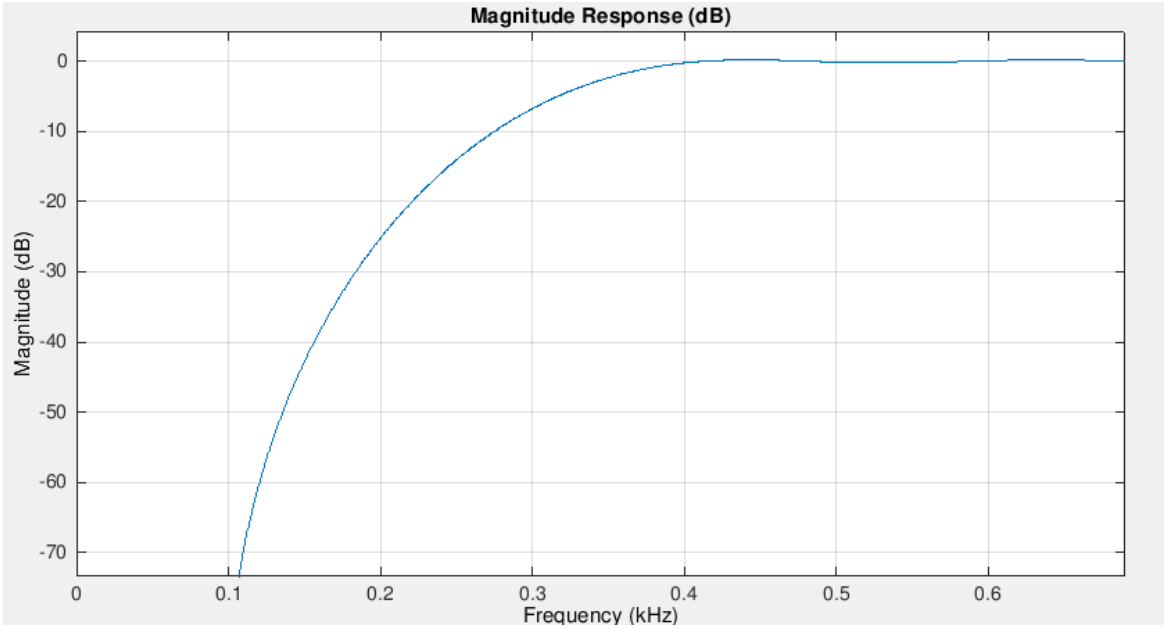

Figura 27. Filtro FIR paso alto diseñado

5.4.4 Filtro FIR Paso Banda - Diseño

Para el caso del filtro paso banda existen más especificaciones de frecuencia que en los dos casos anteriores debido, a que realmente un filtro paso banda es una intersección entre un filtro paso alto y un filtro paso bajo donde la frecuencia de corte del filtro paso bajo debe ser mayor o igual que la del filtro paso alto. Por ese motivo hay que indicar dos frecuencias de paso y dos frecuencias de parada. En este caso, se ha optado poner la primera banda de parada hasta los 100 Hz y la segunda banda de parada a partir de los 400 Hz. La banda de paso se ha configurado para que esté entre los 200 y los 300 Hz.

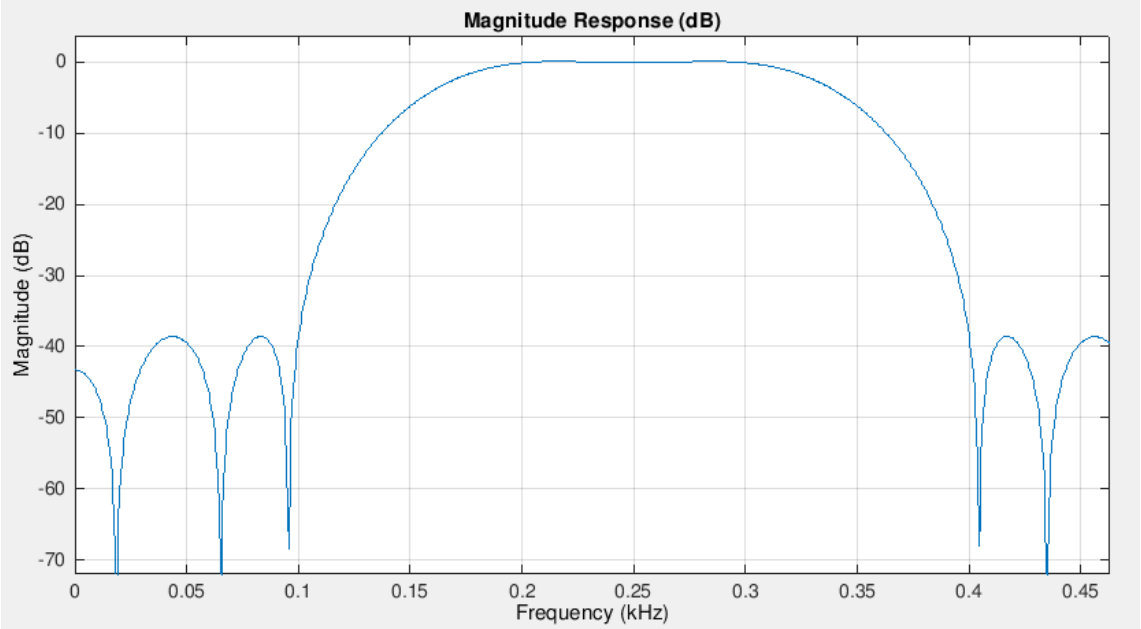

Figura 28. Filtro FIR paso banda diseñado7

5.4.5 Filtro FIR Banda Eliminada (Notch) - Diseño

Un filtro Notch o de banda eliminada se puede interpretar como la unión de un filtro paso alto con un filtro paso bajo, con la particularidad de que la frecuencia de corte del filtro paso alto debe ser mayor que la del filtro paso bajo. Al igual que en el caso del filtro paso banda, hay que especificar dos frecuencias de paso y dos frecuencias de parada. Se fija la banda de paso inferior hasta los 100 Hz y la superior, a partir de los 400 Hz. La banda de parada se configura para que esté entre los 200 y los 300 Hz. Con estos parámetros de configuración se obtiene el filtro de la Figura 29.

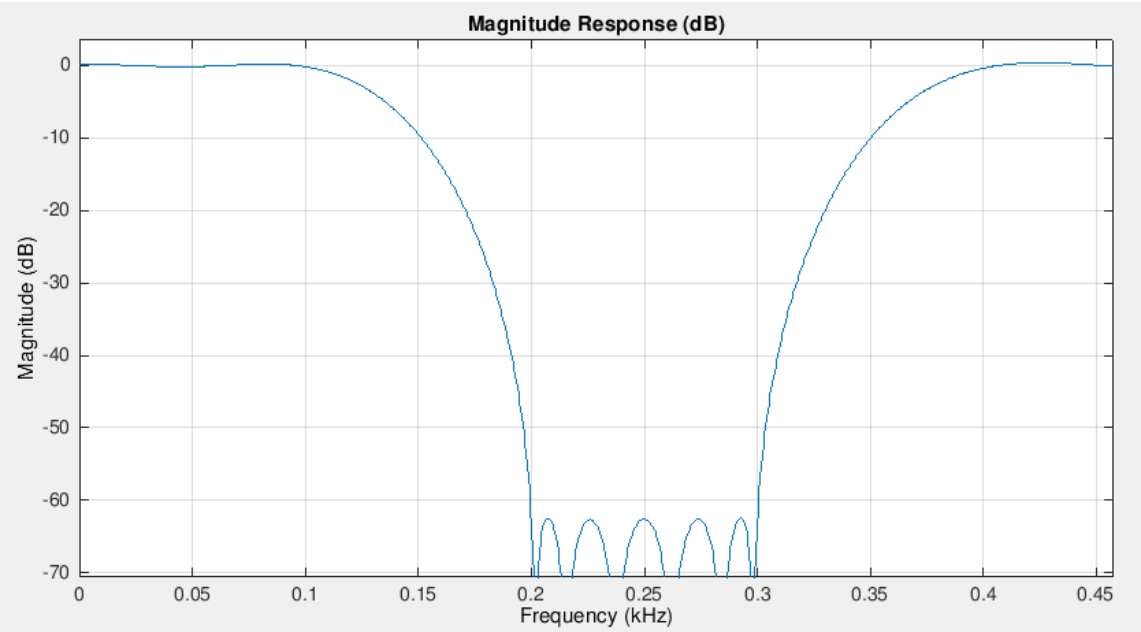

Figura 29. Filtro FIR notch diseñado.

5.5 Desarrollo del proyecto

5.5.1 Entorno de desarrollo Keil uVision 5

Keil uVision 5 es un entorno de desarrollo de software embebido para microcontroladores. En el mismo entorno, este software permite la gestión de proyectos, la edición de código fuente con comprobación dinámica de sintaxis, la depuración de programas. Una de las grandes diferencias que tiene con respecto a su versión predecesora (Keil uVision 4), es que, a la hora de crear un proyecto para un microcontrolador específico, es necesario realizar la instalación de paquetes de software específicos para el microcontrolador en cuestión.

Una vez instalados todos los paquetes necesarios y partiendo de que se generó previamente en CubeMX un proyecto para Keil uVision 5 para la tarjeta NUCLEO-L432KC, al abrir el proyecto en keil se puede observar que está organizado en carpetas que contienen ficheros y librerías necesarias para la compilación e inicialización del microcontrolador.

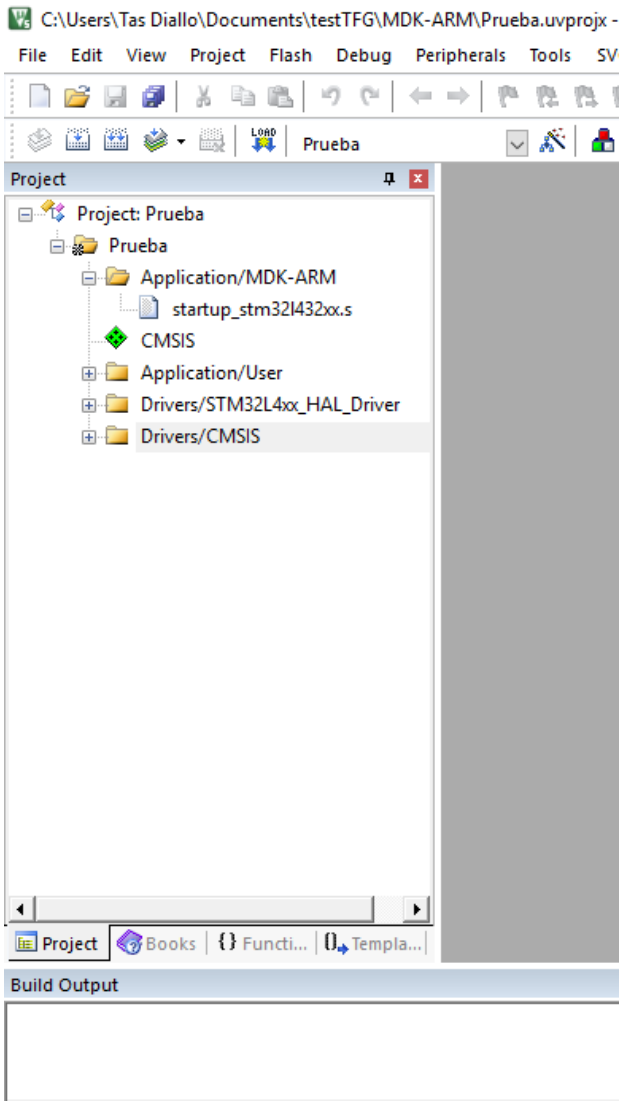

Figura 30. Organización del proyecto - keil uVision 5

Cabe destacar de entre las librerías que se generan, las librerías HAL. Estas librerías permiten el desarrollo en alto nivel a partir de funciones y macros prototipo que facilitan la labor del desarrollador. Los ficheros sobre los que se va a realizar o añadir código, se encuentran en la carpeta "Application /User" y son "main.c" y "stm32l4xx_it.c":

- "main.c": en este fichero es donde se realiza la función principal del proyecto y donde se encuentran todas las funciones de configuración de los periféricos y pines que se configuraron en CubeMX.

- "stm32l4xx_it.c": este fichero contiene principalmente las funciones de atención a las interrupciones de todos los periféricos o módulos para los que se ha habilitado la interrupción global en el sistema.

Es necesario tener en cuenta que todos los ficheros que han sido generados desde CubeMX llevan unos comentarios que indican donde se recomienda añadir el código de programación adicional, por ejemplo:

```
void ADC1_IRQHandler(void)
{
  /* USER CODE BEGIN ADC1_IRQn 0 */

  /* USER CODE END ADC1_IRQn 0 */
  HAL_ADC_IRQHandler(&hadc1);
  /* USER CODE BEGIN ADC1_IRQn 1 */

  /* USER CODE END ADC1_IRQn 1 */
}
```

En este ejemplo se muestra la función de atención a la interrupción del ADC1. Si se desease completar esta función para realizar alguna funcionalidad, es recomendable escribir entre los comentarios.

```
/* USER CODE BEGIN ADC1_IRQn 0 */

/* USER CODE END ADC1_IRQn 0 */
```

Debido a que, si durante el desarrollo del código, surge alguna necesidad de modificar la configuración de la tarjeta, al regenerar CubeMX el proyecto, el código que se añadió previamente hasta el momento, no se será eliminado.

En la barra superior de herramientas que tiene Keil, cabe destacar principalmente tres elementos que son de utilidad:

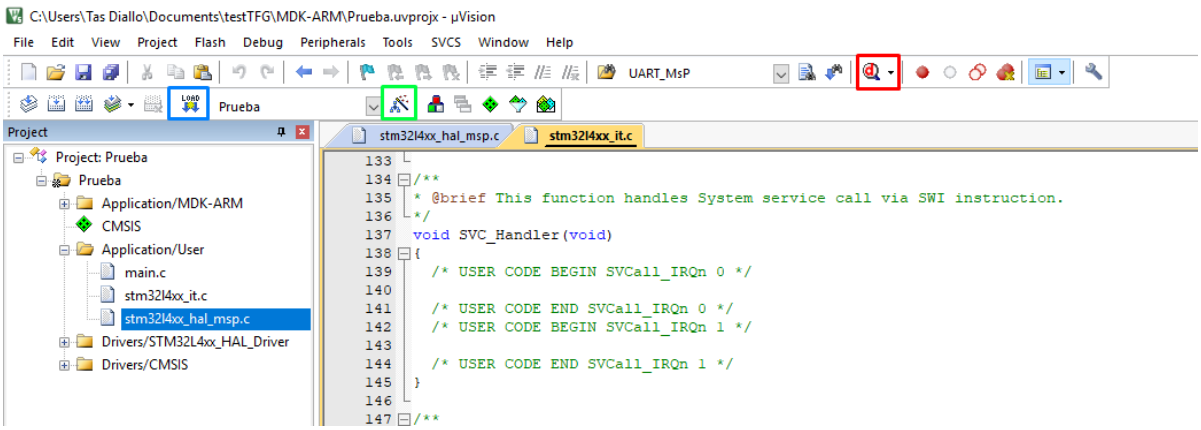

Figura 31. Barra superior de herramientas – keil uVision 5

Con la Figura 31 se observa recuadrado en color rojo el botón de acceso a la depuración del programa. Una vez se haya hecho el código de programación, mediante este botón se puede acceder a la depuración del programa con la posibilidad de hacerlo tanto en simulación (sin la tarjeta conectada) o bien con la tarjeta conectada.

Mediante el botón recuadrado en color azul, se puede cargar el código directamente a la tarjeta sin necesidad de pasar por la depuración. Es imprescindible, de manera lógica, que tanto la tarjeta como el depurador, estén conectados al ordenador. En el caso de NUCLEO-L432KC, el depurador se encuentra integrado en la tarjeta y solo sería necesario conectar la tarjeta.

Por último, el botón recuadrado en color verde permite acceder a la configuración de la tarjeta.

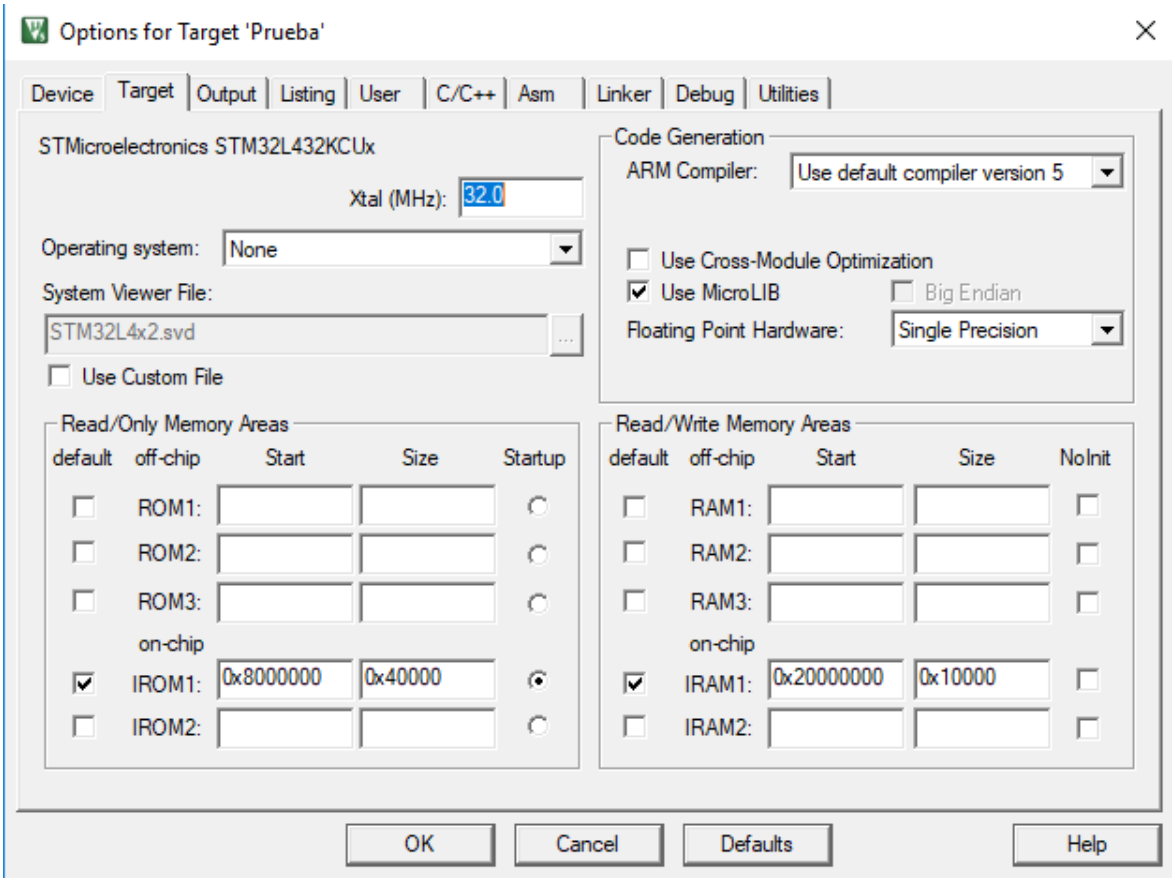

Figura 32. Configuración de la tarjeta – keil uVision 5

Por defecto se abre la ventana de configuración en la pestaña “Target” donde se pueden ver las direcciones de los módulos de memoria que tiene el microcontrolador, donde se puede cambiar el compilador del programa y donde se indica la frecuencia de oscilador de cristal de la tarjeta.

Accediendo a la pestaña “Debug” se puede cambiar la configuración para que la depuración se pueda realizar con el simulador de keil o directamente con la tarjeta.

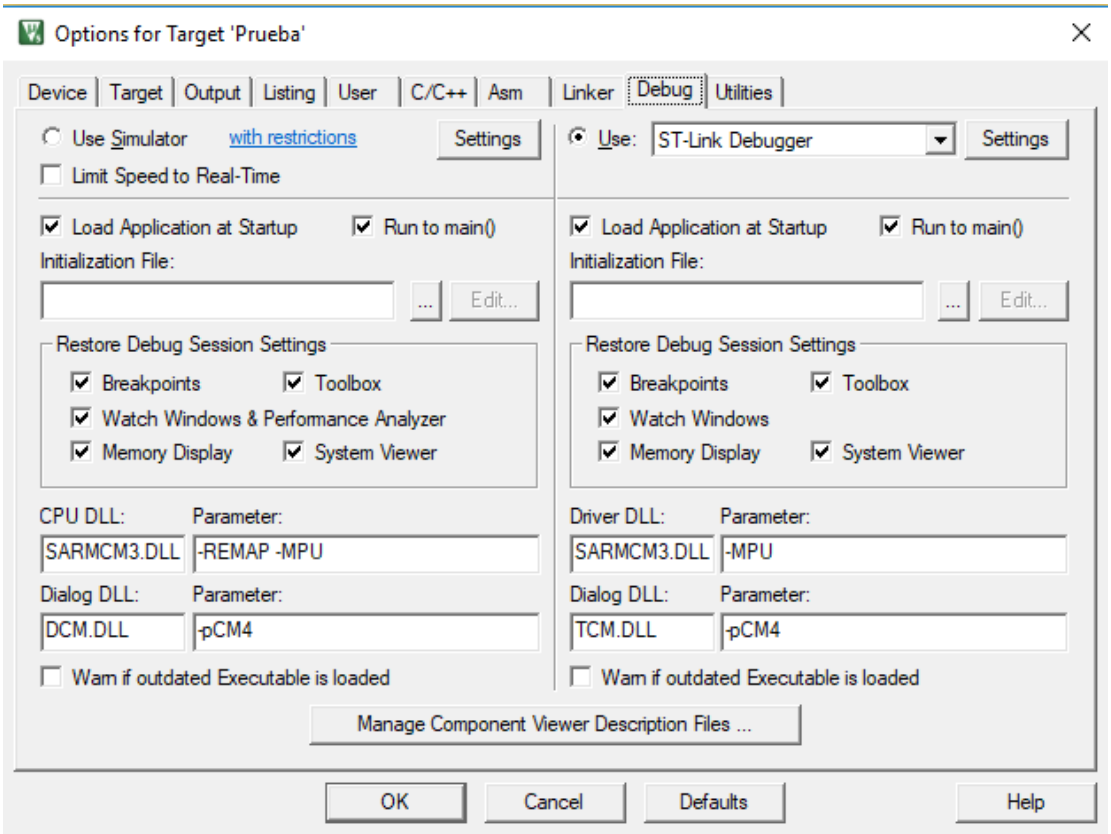

Figura 33. Penstaña “Debug” en la ventana de configuración – keil uVision 5

5.5.2 Desarrollo del código de programación

Teniendo en cuenta que con el programa CubeMX se configuró previamente el ADC haciendo que interrumpa al final de cada conversión, se programa la rutina de atención a la interrupción del ADC de manera que, en el momento de la interrupción, se haga una lectura del valor de la muestra convertida para realizar el filtrado en tiempo real y finalmente enviar el resultado del filtrado por el DAC.

El uso de cualquier filtro FIR implica tener un buffer de entrada de un tamaño igual al número de coeficientes que tiene el filtro en donde se almacenen las ultimas *P* muestras convertidas, donde *P* es el número de coeficientes del filtro. Atendiendo al esquema genérico de filtros FIR de la Figura 34 se pueden apreciar una serie de retardos de la señal de entrada, estos retardos en conjunto con la muestra actual “x(k)” representan el buffer de entrada al que se ha estado haciendo referencia.

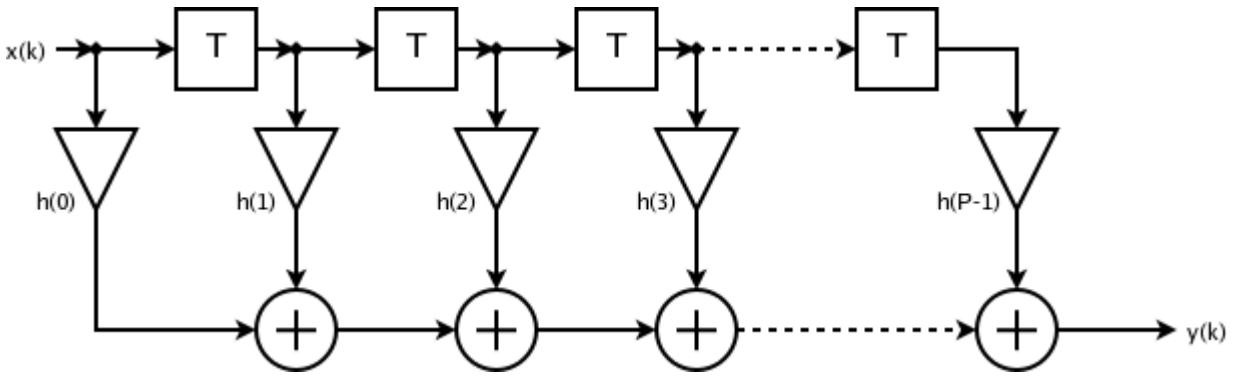

Figura 34. Esquema de filtros FIR

Por otro lado, se puede observar en la Figura 34 que es necesario realizar un bucle donde se realicen las operaciones de multiplicación de cada una de las muestras almacenadas en el buffer de entrada por el coeficiente que le corresponda. Esta correspondencia entre muestra del buffer y coeficiente del filtro se estable de manera que la muestra actual se multiplica con el primer coeficiente del filtro, la muestra anterior a la actual se multiplica con el segundo coeficiente del filtro y así sucesivamente se llega a que la muestra más antigua almacenada en el buffer se debe multiplicar con el último coeficiente del filtro. Cada multiplicación entre muestra y coeficiente se suma de manera acumulativa, obteniendo como resultado, al final del bucle, la muestra actual filtrada "y(k)".

En el código de programación que se muestra a continuación corresponde a una parte de la función de atención a la interrupción que se encarga de realizar el filtrado explicado anteriormente.

```
// bucle de filtrado del filtro FIR
    for(int i=0;i<ORDEN(h);i++){
        y = y + x[ORDEN(h)-i-1]*h[i];
    }

    //buble para desplazar las muestras anteriores del buffer
    for(int j=0;j<ORDEN(h)-1;j++){
        x[j]=x[j+1];
    }
```

En conjunto, ambos bucles *for* ejecutan la ecuación general de filtros FIR de cualquier tipo:

$$y_n = \sum_{k=0}^{N-1} b_k x_{n-k}$$

Dado que el DAC del L432KC es de 12 bits, y solo admite números enteros sin signo (ya que no es posible la conversión D/A para valores de tensión negativos), es necesario convertir a entero sin signo el valor de la muestra actual filtrada y[n] debido a que el resultado del filtrado devuelve un número en formato *float* (con signo y números decimales). El problema de realizar solo esta conversión es que para un valor negativo de y[n] se va a sacar por el DAC su valor entero en valor absoluto lo que se traduce en que la forma de onda de la señal de salida del filtro, puede verse recortada en media onda en el peor de los casos.

Para corregir este inconveniente, antes de sacar la muestra y[n] por el DAC, se le aplica previamente un offset calculado en depuración, el cual se obtiene con el siguiente código de programación:

```
if (y<aux){
        aux=y;
}
```

Se declara una variable auxiliar de tipo *float* y se almacena en esta variable el valor de la muestra y[n] siempre y cuando y[n] sea menor que y[n-k] donde k representa el índice de cada una de las muestras de salida anteriores. De este modo, se consigue el valor más pequeño de salida que puede tener el filtro. Si este valor (*aux*) es positivo, no es necesario aplicar ningún offset. En cambio, si es negativo, es necesario sumar a todas las muestras de salida, el valor absoluto de *aux.* Este valor de offset, es distinto para cada filtro, es decir, es necesario evaluar en depuración cada uno de los filtros.

Una vez aplicado el offset se escribe la sentencia para sacar la muestra y[n] por el DAC:

```
HAL_DAC_SetValue(&hdac1,DAC_CHANNEL_1,DAC_ALIGN_12B_R,(uint16_t)y&(0xFFF));
```

Como se puede observar, se convierte y[n] a un entero sin signo de 16 bits y se le aplica una máscara para enviar únicamente los 12 bits de menor peso.

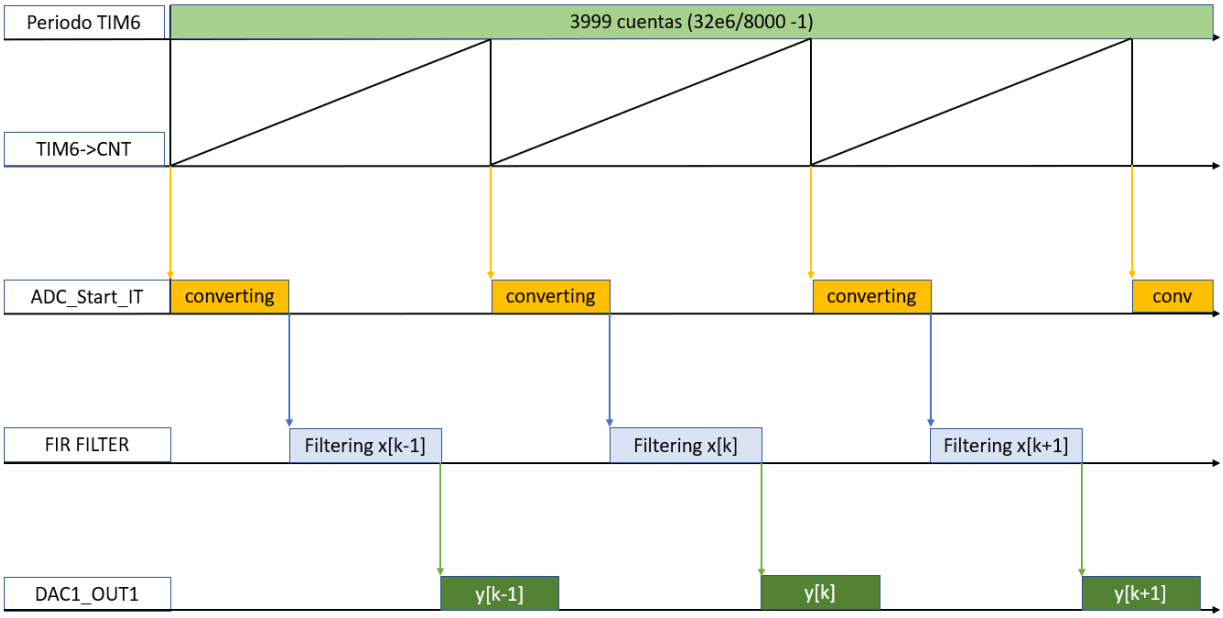

Figura 35. Cronograma explicativo del funcionamiento del sistema diseñado

Con el envío de las muestras filtradas al DAC1 se puede analizar con ayuda de un osciloscopio las características de frecuencia y amplitud de la señal filtrada.

Esto puede resultar ventajoso ya que, de esta manera se pueden realizar pruebas analizando la señal filtrada y en función del resultado de ésta, se puede ver si el filtro se ajusta a las especificaciones de diseño del filtro y en su defecto reajustarlo de nuevo.

Por último, en la función principal del programa se debe indicar que el Timer6 se inicie mediante la sentencia "`HAL_TIM_Base_Start(&htim6);`" debido a que es el encargado de marcar el inicio de conversión del ADC1.

6. Medición de señales – Visual Analyzer 2014

Para la obtención de la representación de las señales de entrada y salida del sistema, se utiliza el software “Visual Analyzer 2014”.

Visual Analyzer es un software que permite medir en tiempo real señales analógicas utilizando la tarjeta de sonido del ordenador o bien una tarjeta de sonido externa. Las características del VA (Visual Analyzer) son las siguientes:

- Funcionalidad de osciloscopio (con dos canales de entrada, división de tiempo y trigger).
- Analizador de espectros con representación en tiempo real de magnitud y fase.
- Generador de funciones (triangular, cuadrada, y sinusoidal) y generador de continua.
- Medición de frecuencia en el dominio del tiempo y de la frecuencia.
- Medición de voltaje (DC, valor eficaz, valor pico a pico y valor medio).
- Posibilidad de usar filtros, para filtrar en tiempo real (paso bajo, paso alto, paso banda,etc).

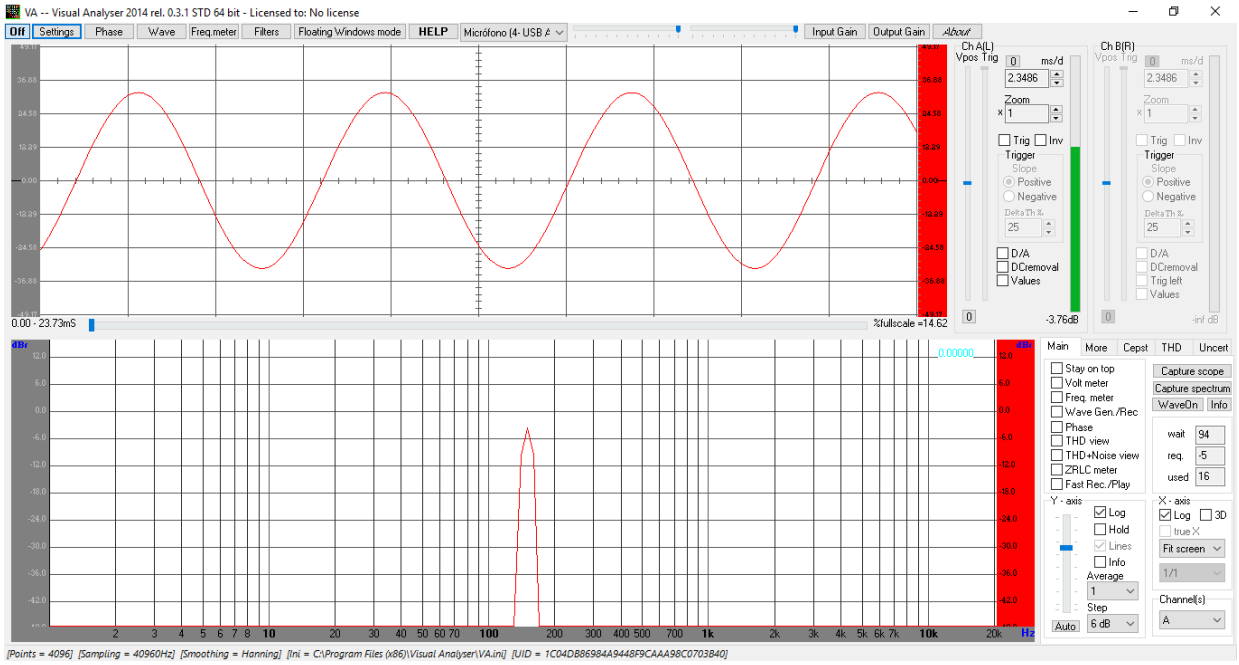

Figura 36. Interfaz de usuario de Visual Analyzer 2014

Cabe destacar como inconveniente, que al usar un conector jack y una tarjeta de sonido para poder medir la señales con este software, los valores medios de señales alternas no se pueden ver representados, siendo imprescindible en el caso de que fuese necesario, el uso de un osciloscopio.

Debido a que el ordenador utilizado para el desarrollo de todo el proyecto es un portátil Acer E5-573G, fue necesario el uso de una tarjeta de sonido externa de bajo coste. La tarjeta de sonido que lleva integrada el ordenador permite el uso del conector jack de 3.5 mm únicamente como salida de audio para el uso de auriculares o altavoces. Por esta razón, ha sido necesario el uso de una tarjeta de sonido externa que permitiese tener entrada y salida de datos. En este caso se ha optado por usar una tarjeta de bajo coste de la marca Tecknet.

Figura 37. Tarjeta de sonido externa – Tecknet

Existe un inconveniente con el uso de tarjetas de sonido externas no profesionales y de bajo coste, y es que en la gran mayoría de estas, no es posible la entrada de audio en modo stereo, es decir, el canal izquierdo y el derecho de audio están unidos en el mismo punto, y esto implica que no se pueden medir dos señales distintas, que en el caso de este proyecto implica que no se pueden ver al mismo tiempo las señales de entrada y salida del sistema. Primero se mostrará la señal de entrada y después la señal de salida.

Para conectar las señales que se miden con la tarjeta de sonido ha sido necesario soldar dos cables (para GND y la señal) a un conector jack macho de 3.5mm.

7. Plan de Pruebas

Este apartado tiene como objetivo establecer unas pautas para la ejecución de pruebas funcionales de los distintos filtros implementados en el desarrollo del proyecto. En primer lugar, se debe diseñar un plan de pruebas que permita abarcar todas las situaciones posibles que puedan provocar un fallo en el funcionamiento del sistema diseñado.

Las características principales que debe tener un plan de pruebas para validar el funcionamiento de un sistema como el diseñado son:

- Precondición/es: es la situación o entorno inicial en el que se debe encontrar el sistema antes de realizar una prueba. Las precondiciones se caracterizan por ser factores que condicionan el resultado de la prueba.
- Objetivo: se indica la finalidad que tiene la prueba.
- Acciones: definen las pautas o pasos a seguir en todo momento durante la ejecución de una prueba.

Para definir los casos de prueba se ha decidido separar o agrupar los casos de prueba según el filtro que se esté probando por comodidad.

7.1 Filtro FIR Paso Bajo - Caso de prueba 1

Precondiciones	
·Debe seleccionarse en el software cargado en la tarjeta NUCLEO-L432KC, el filtro paso bajo. ·La tarjeta miniDK2 debe tener inicialmente una señal sinusoidal de 50 Hz. ·A la salida se aprecia una señal con muy poca atenuación inicialmente.	
Objetivo	
Se busca comprobar que a medida que se va aumentando la frecuencia de la señal de entrada, la señal de salida se ve atenuada a partir de los 100 Hz, llegando a atenuarse por completo a partir de los 400 Hz.	
Pasos	**Resultado Esperado**
1. Ajustar con el potenciómetro la amplitud de la señal de entrada y tomar la referencia de magnitud en la ventana de representación del espectro de la señal en Visual Analyzer.	En Visual Analyzer se observa que la magnitud se puede ajustar al gusto del usuario e inicialmente la frecuencia de la señal es de 50 Hz
2. Pulsar el botón “KEY 2” de la tarjeta miniDK2.	Se observa que la frecuencia de la señal de entrada ha aumentado a 100Hz.
3. Medir el espectro de la señal de salida.	Con respecto a la señal de entrada, la de salida se ve atenuada en pocos dB de magnitud.
4. Pulsar el botón “KEY 2” de la tarjeta miniDK2 y medir el espectro de la señal que se ve a la salida.	La señal de entrada tiene una frecuencia de 50 Hz más que la señal de entrada en el paso 3 y la señal de salida actual se ve más atenuada que la señal de salida vista en el paso 3.
5. Repetir el paso 4, hasta alcanzar una frecuencia de señal de entrada de 450Hz.	A medida que aumenta la frecuencia, se atenúa cada vez más la señal de salida, llegando a ser inapreciable a partir de los 400Hz.

Tabla 1. Caso de prueba 1 – Filtro paso bajo

A continuación, se realiza la ejecución del Caso de prueba 1:

Se parte de la señal de entrada inicialmente a 50 Hz.

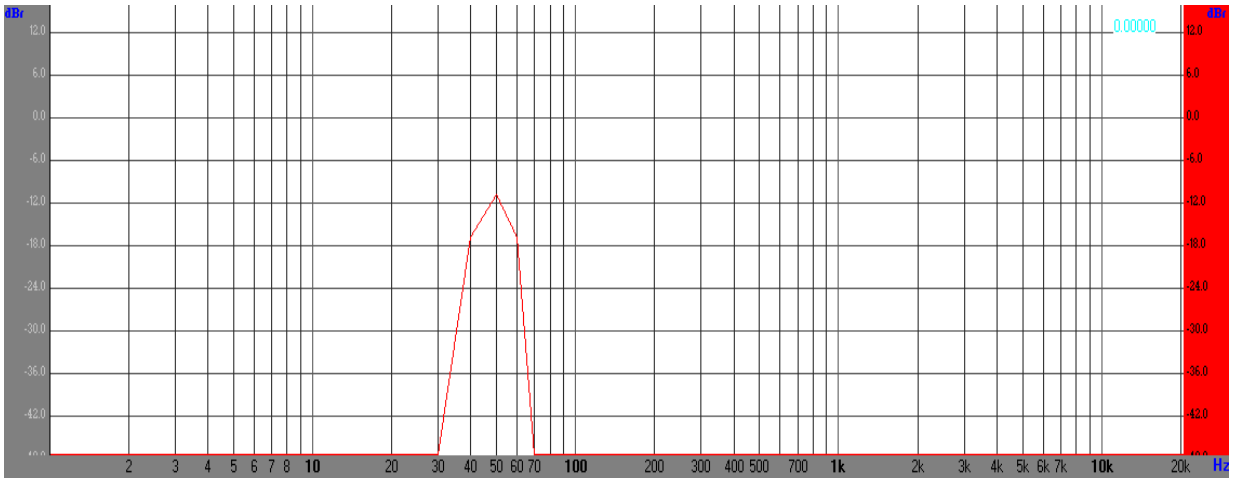

Figura 38. Espectro de entrada a 50 Hz

A la salida, con una entrada como la de la Figura 38 se obtiene el siguiente espectro:

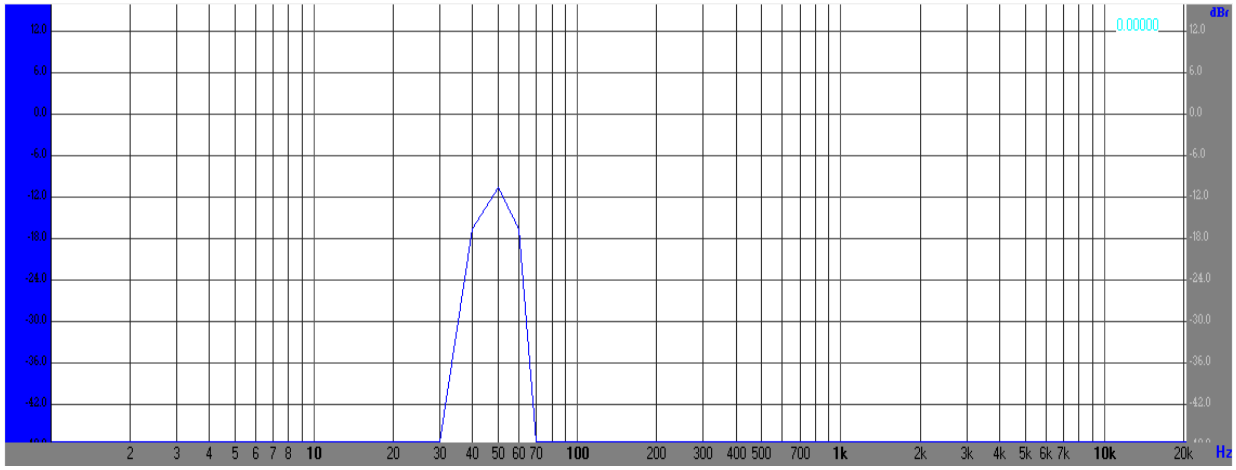

Figura 39. Espectro de salida a 50 Hz – Filtro paso bajo

Pulsando el botón “KEY2” la frecuencia de la señal de entrada aumenta a 100 Hz.

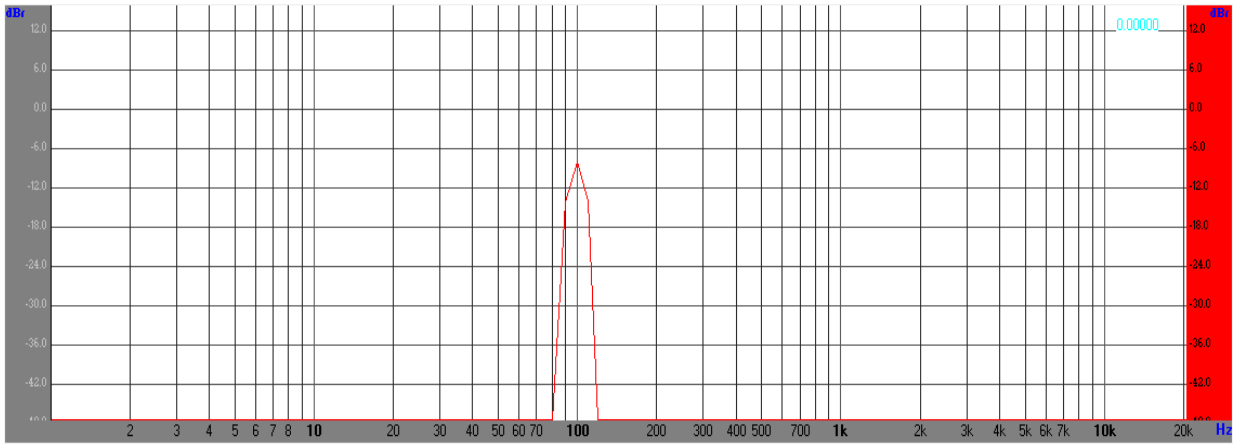

Figura 40. Espectro de entrada a 100 Hz

Ante una entrada a 100 Hz se obtiene una salida más atenuada que con 50 Hz:

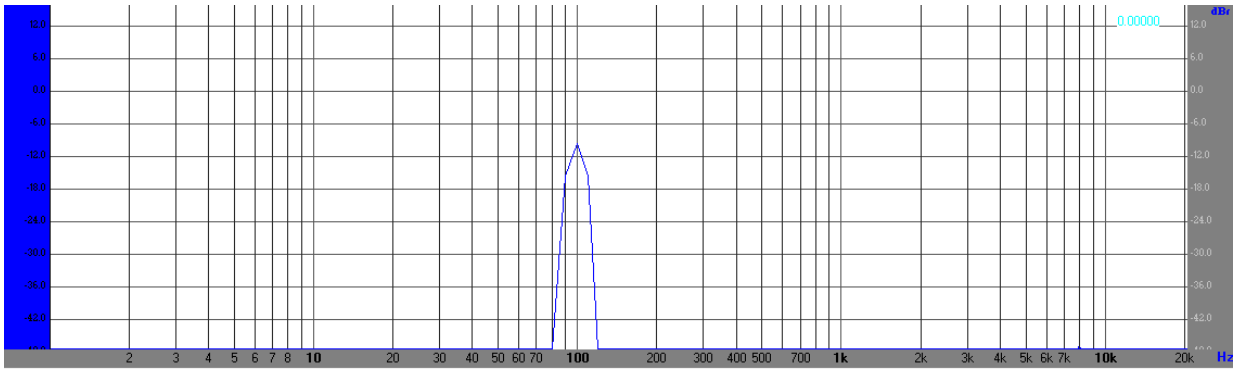

Figura 41. Espectro de salida a 100 Hz – Filtro paso bajo

Desde este punto en adelante, aquellas figuras con representación del espectro en color rojo indican que son señales de entrada al filtro y en color azul su respectiva salida.

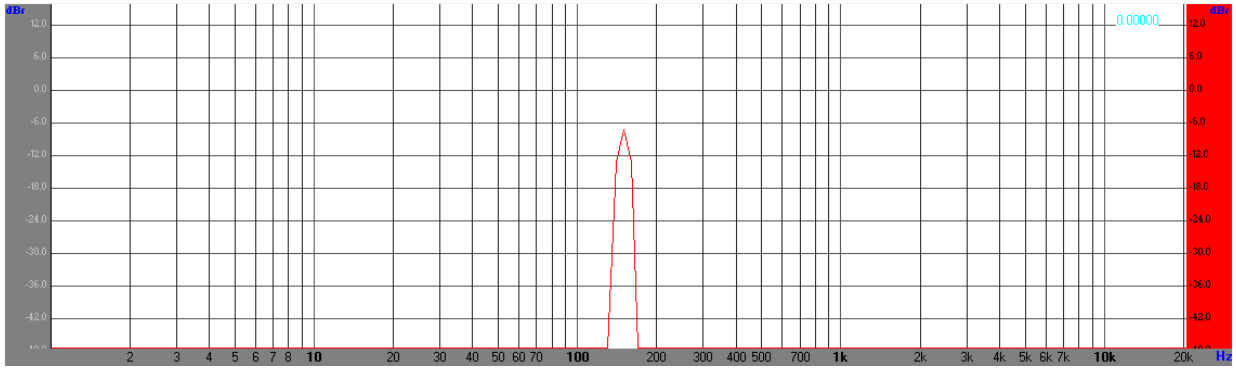

Figura 42. Espectro de entrada a 150 Hz

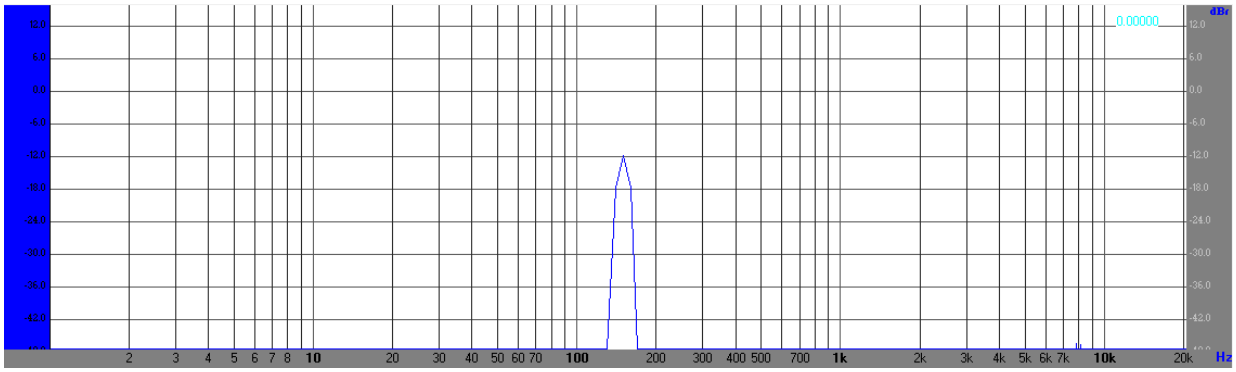

Figura 43. Espectro de salida a 150 Hz – Filtro paso bajo

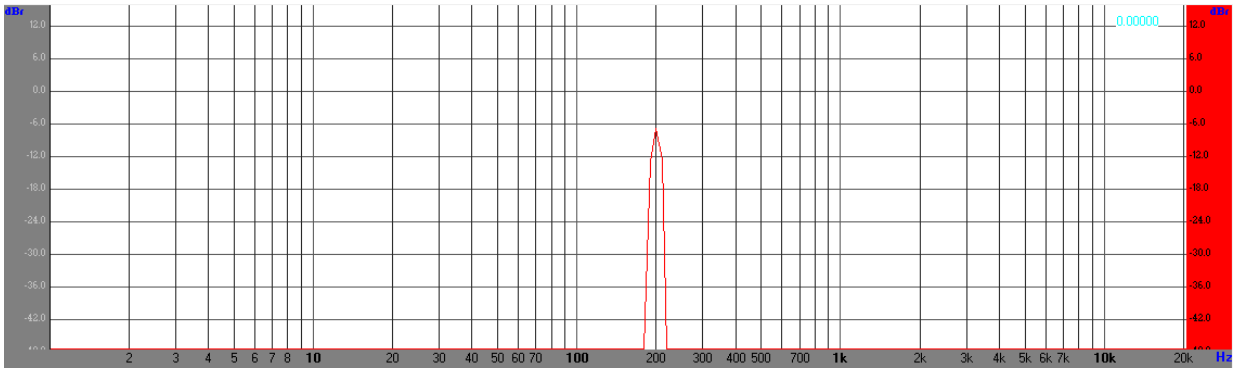

Figura 44. Espectro de entrada a 200 Hz

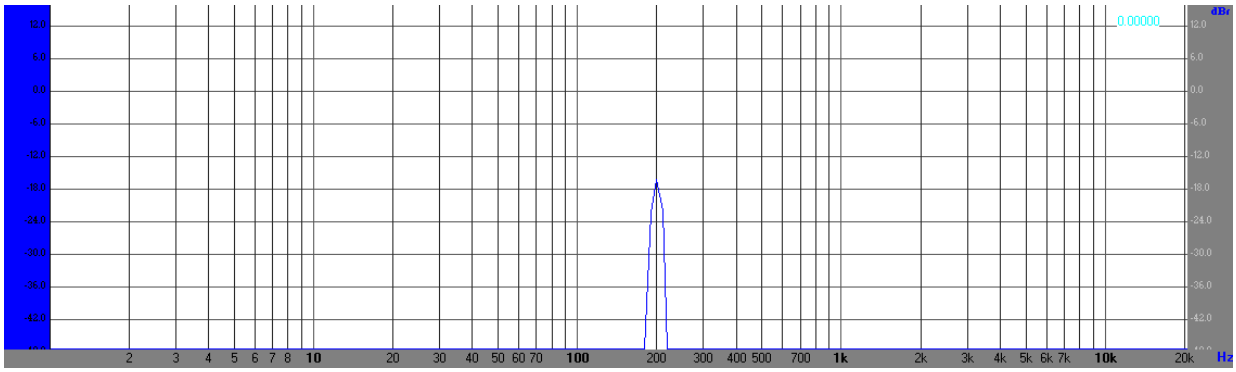

Figura 45. Espectro de salida a 200 Hz – Filtro paso bajo

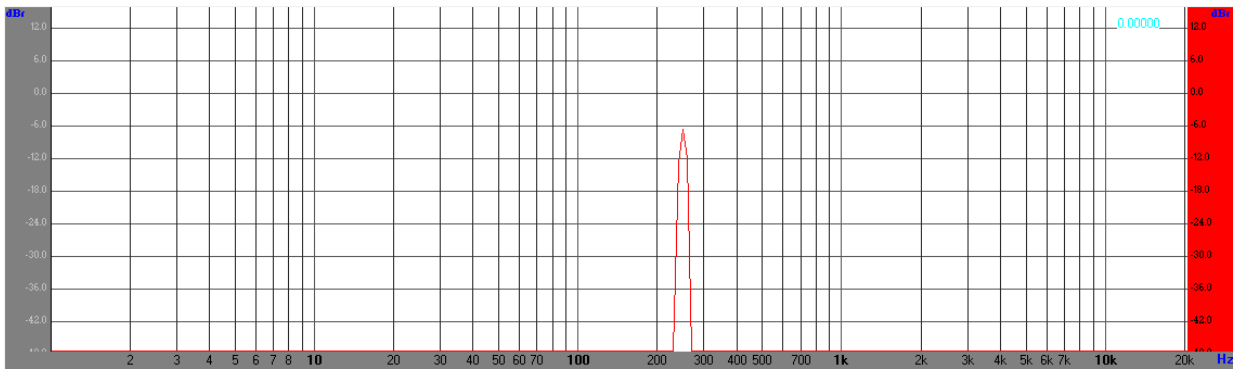

Figura 46. Espectro de entrada a 250 Hz

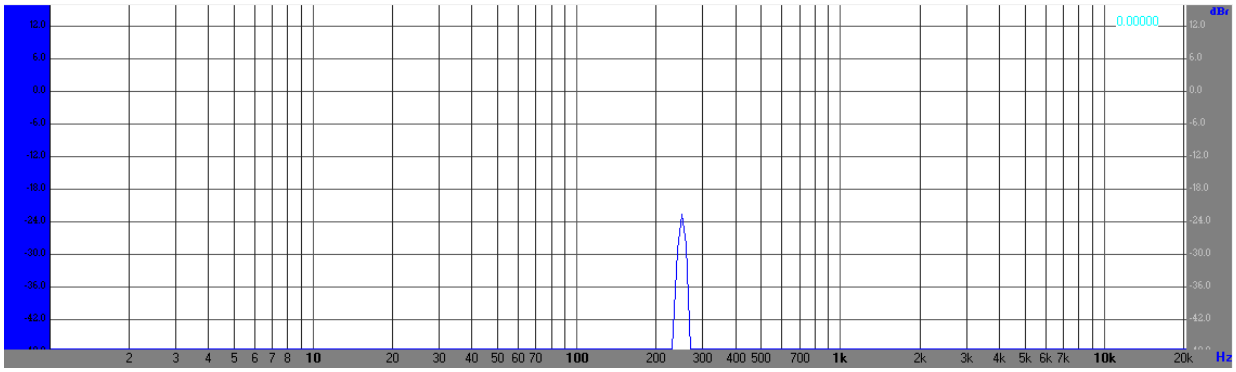

Figura 47. Espectro de salida a 250 Hz – Filtro paso bajo

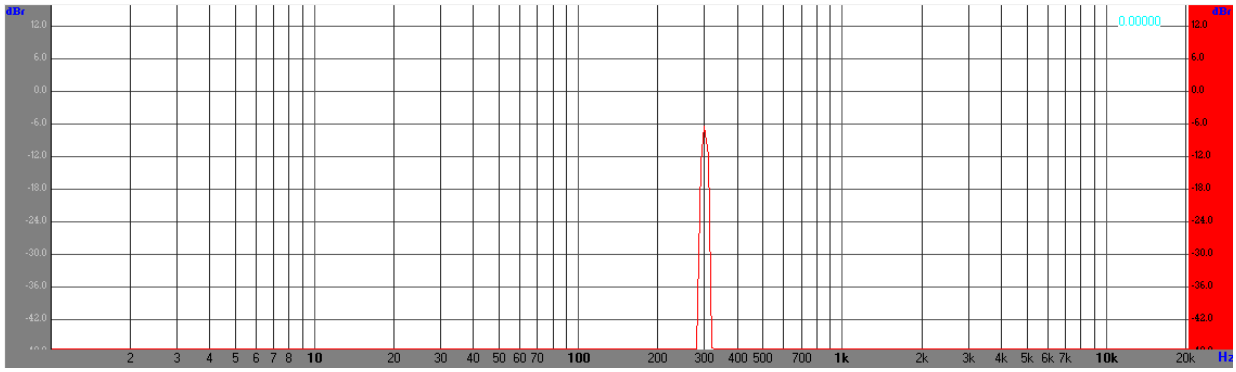

Figura 48. Espectro de entrada a 300 Hz

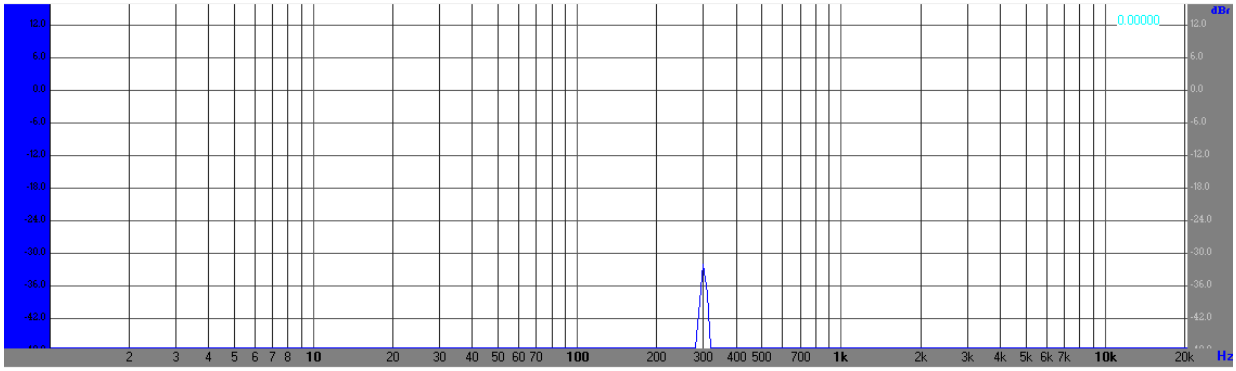

Figura 49. Espectro de salida a 300 Hz – Filtro paso bajo

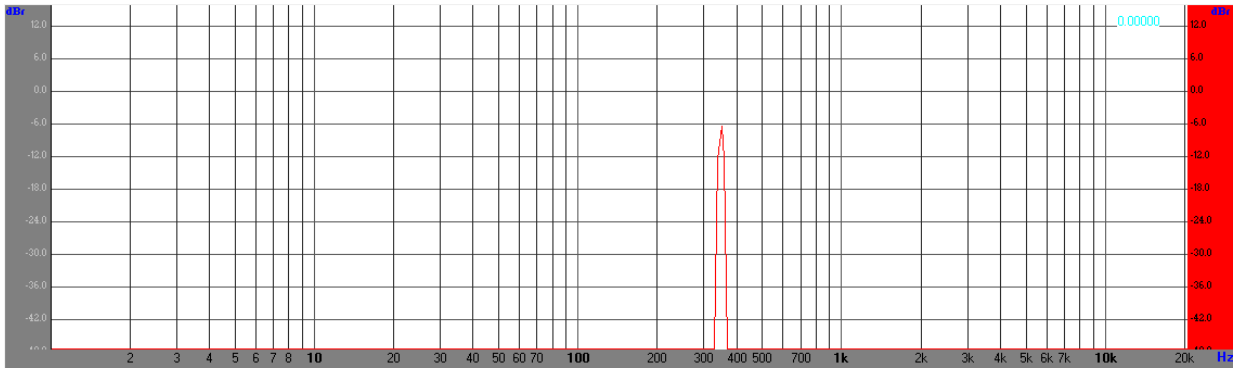

Figura 50. Espectro de entrada a 350 Hz

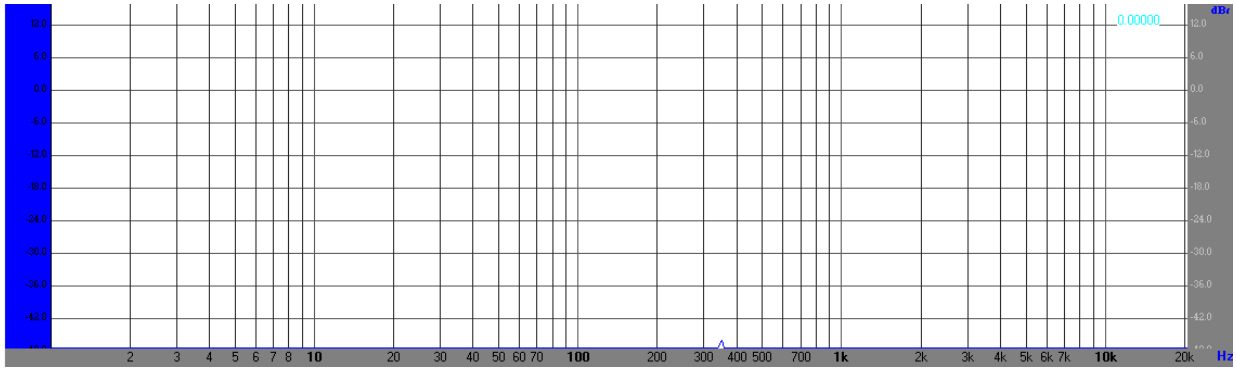

Figura 51. Espectro de salida a 350 Hz – Filtro paso bajo

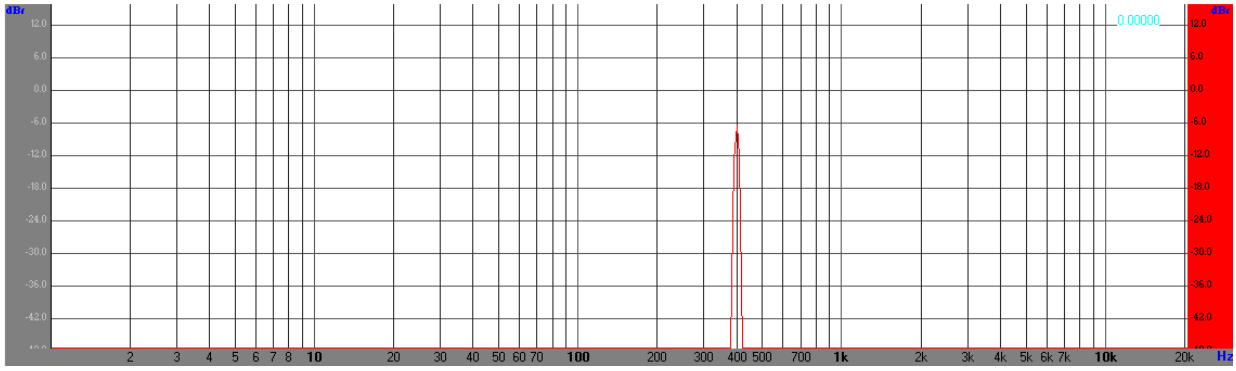

Figura 52. Espectro de entrada a 400 Hz

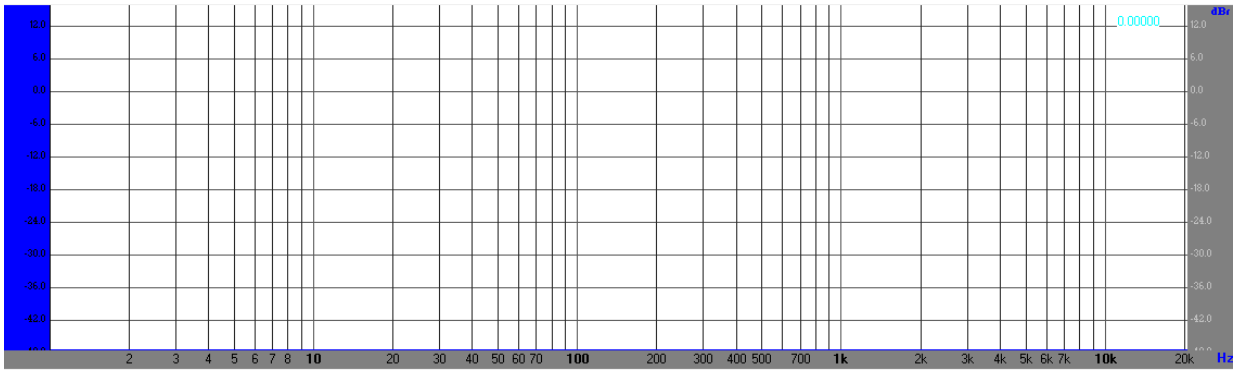

Figura 53. Espectro de salida a 400 Hz – Filtro paso bajo

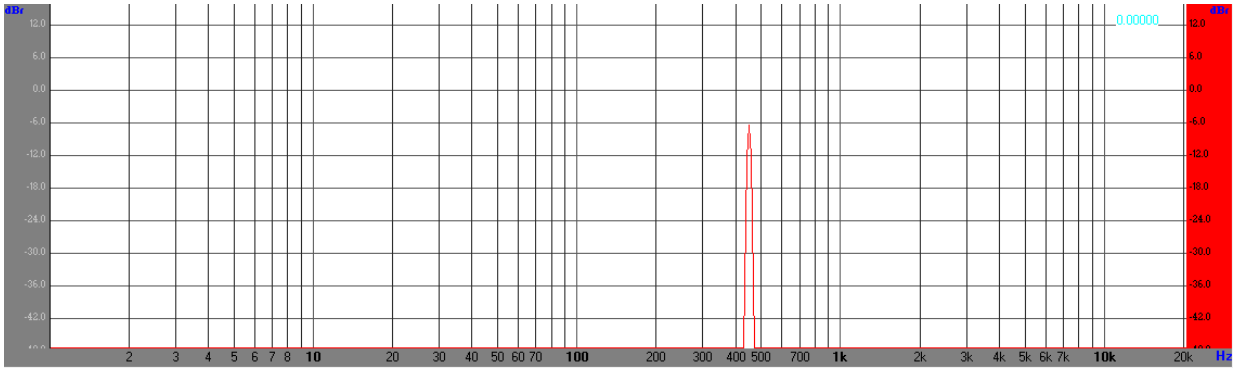

Figura 54. Espectro de entrada a 450 Hz

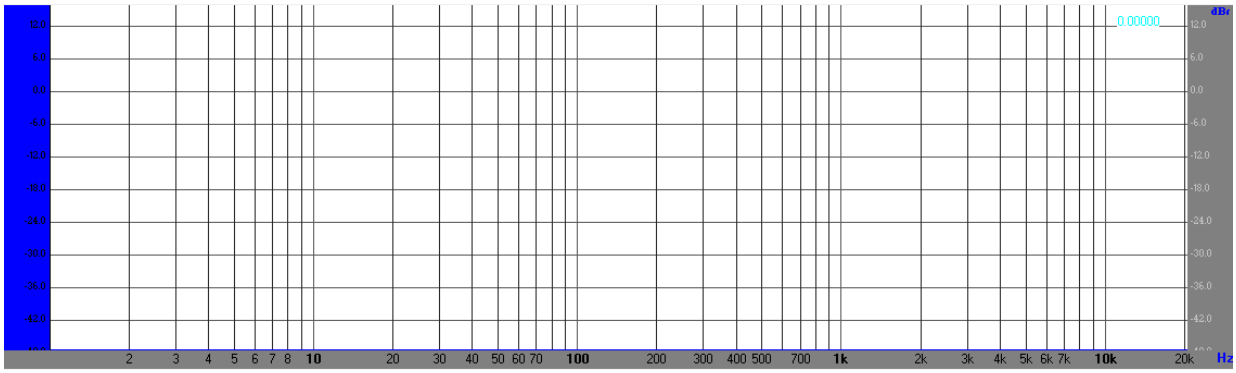

Figura 55. Espectro de salida a 450 Hz – Filtro paso bajo

7.2 Filtro FIR Paso Alto - Caso de prueba 2

Precondiciones	
·Debe seleccionarse en el software cargado en la tarjeta NUCLEO-L432KC, el filtro paso alto. ·La tarjeta miniDK2 debe tener inicialmente una señal sinusoidal de 50 Hz. · A la salida no se puede apreciar señal inicialmente.	
Objetivo	
Se busca comprobar que a medida que se va aumentando la frecuencia de la señal de entrada, la señal de salida se ve aumentada en magnitud a partir de los 100 Hz, llegando a apreciarse, sin prácticamente atenuaciones, a partir de los 400 Hz.	
Pasos	**Resultado Esperado**
1. Ajustar con el potenciómetro la amplitud de la señal de entrada y tomar la referencia de magnitud en la ventana de representación del espectro de la señal en Visual Analyzer.	En Visual Analyzer se observa que la magnitud se puede ajustar al gusto del usuario e inicialmente la frecuencia de la señal es de 50 Hz
2. Pulsar el botón “KEY 2” de la tarjeta miniDK2.	Se observa que la frecuencia de la señal de entrada ha aumentado a 100Hz.
3. Medir el espectro de la señal de salida.	Con respecto a la señal de entrada, la de salida se ve atenuada llegando a no apreciarse con claridad.
4. Pulsar el botón “KEY 2” de la tarjeta miniDK2 y medir el espectro de la señal que se ve a la salida.	La señal de entrada tiene una frecuencia de 50 Hz más que la señal de entrada en el paso 3 y la señal de salida actual se ve con magnitud mayor que la señal de salida vista en el paso 3.
5. Repetir el paso 4, hasta alcanzar una frecuencia de señal de entrada de 450Hz.	A medida que aumenta la frecuencia, aumenta la magnitud de la señal de salida, llegando a no haber prácticamente atenuación a partir de los 400Hz.

Tabla 2. Caso de prueba 2 – Filtro paso alto

Para la ejecución de este caso de prueba y de los casos 3 y 4 las entradas a los filtros van a ser las mismas que en el caso del filtro paso bajo, es decir, las figuras 38, 40, 42, 44, 46, 48, 50, 52 y 54.

Respectivamente para cada una de esas entradas se obtienen para el filtro paso alto las siguientes salidas:

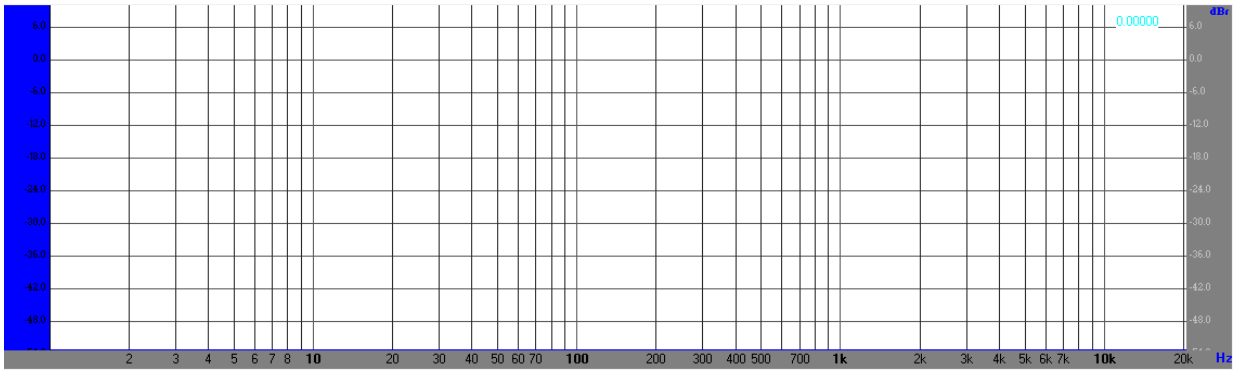

Figura 56. Espectro de salida a 50 Hz – Filtro paso alto

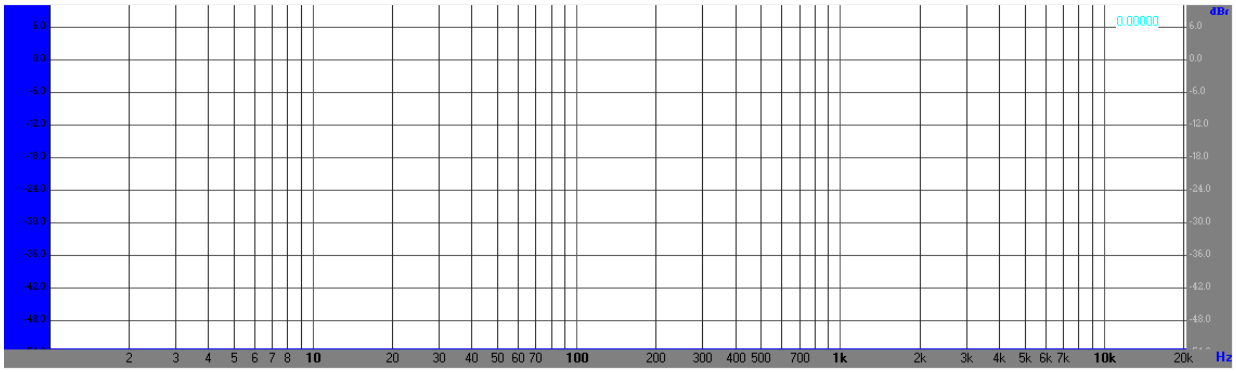

Figura 57. Espectro de salida a 100 Hz – Filtro paso alto

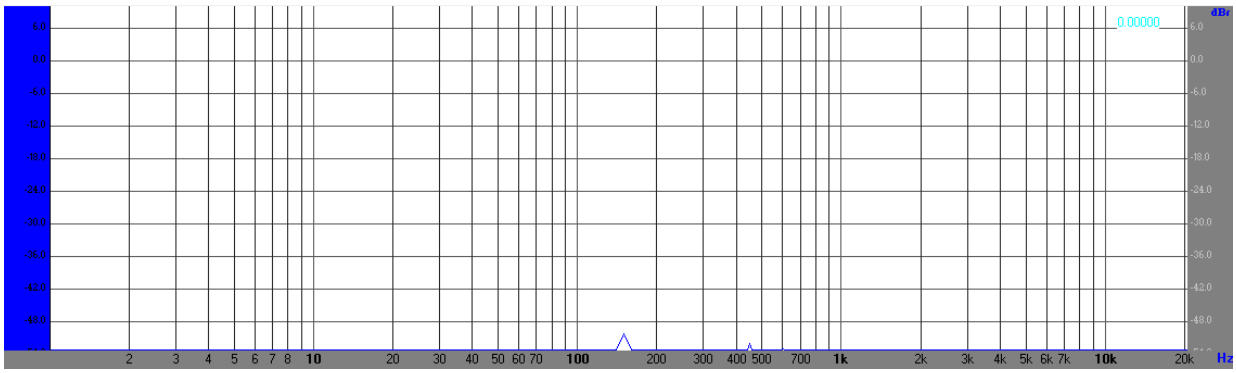

Figura 58. Espectro de salida a 150 Hz – Filtro paso alto

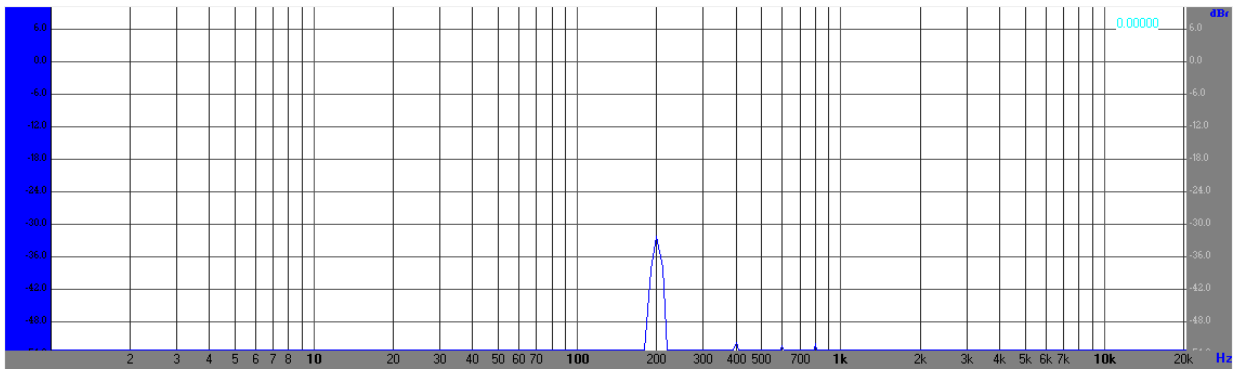

Figura 59. Espectro de salida a 200 Hz – Filtro paso alto

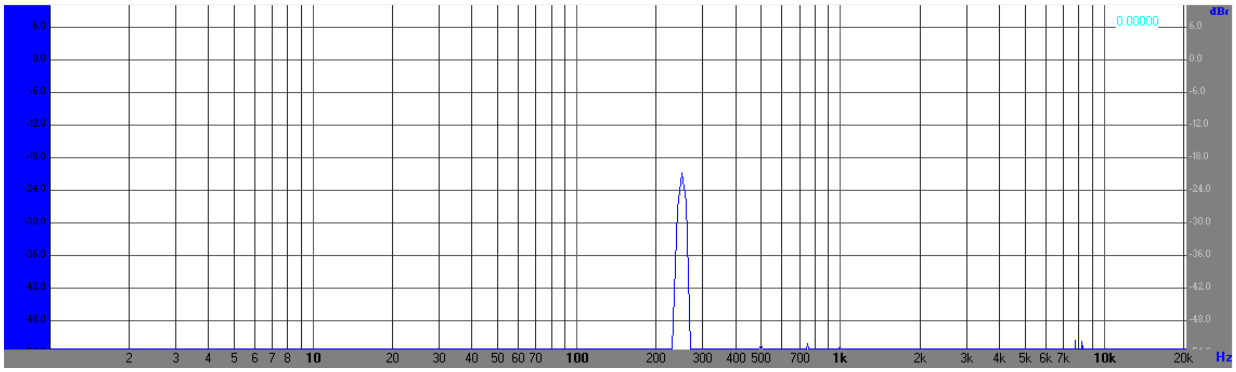

Figura 60. Espectro de salida a 250 Hz – Filtro paso alto

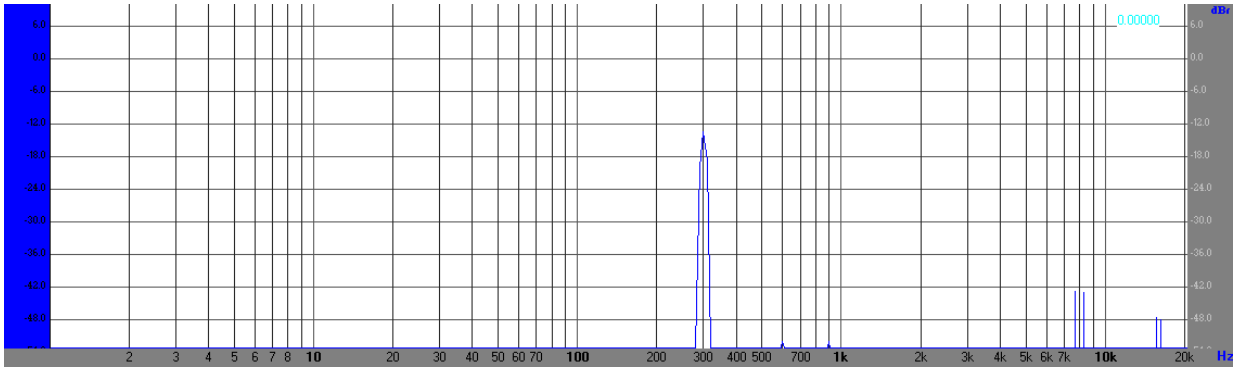

Figura 61. Espectro de salida a 300 Hz – Filtro paso alto

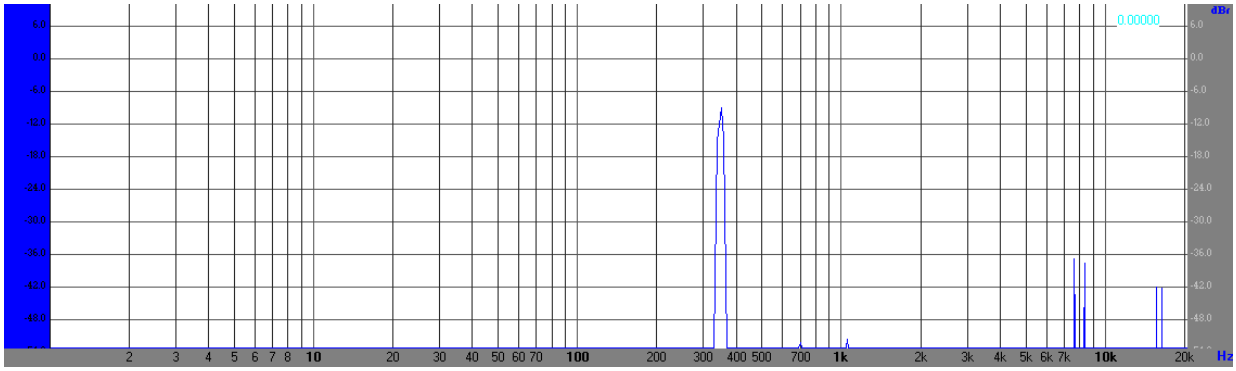

Figura 62. Espectro de salida a 350 Hz – Filtro paso alto

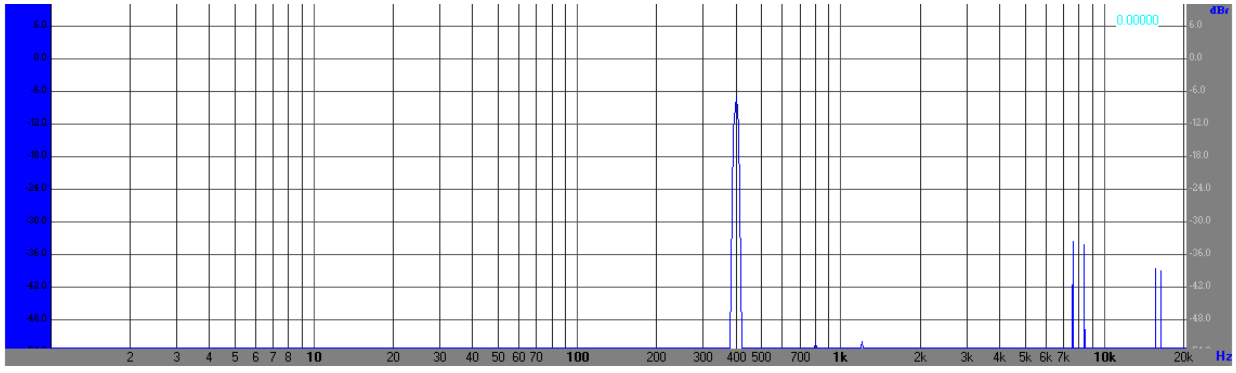

Figura 63. Espectro de salida a 400 Hz – Filtro paso alto

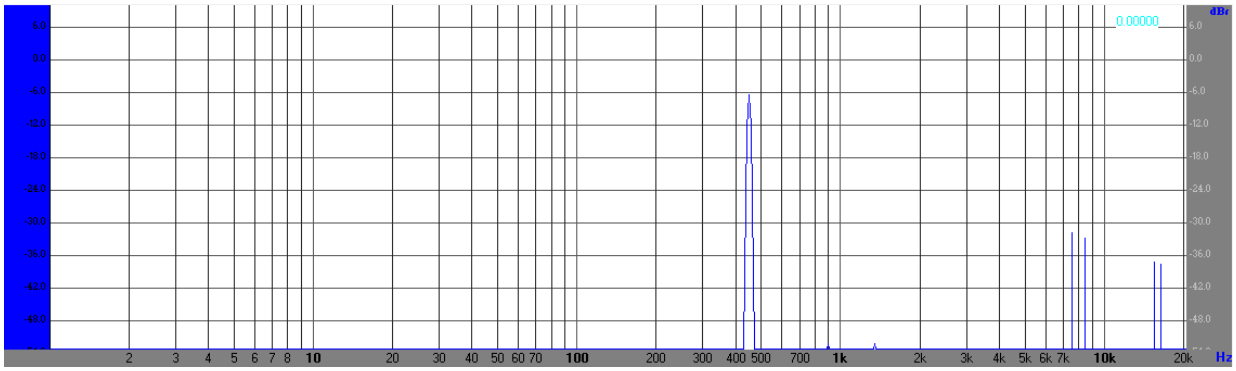

Figura 64. Espectro de salida a 450 Hz – Filtro paso alto

7.3 Filtro FIR Paso Banda - Caso de prueba 3

Precondiciones	
·Debe seleccionarse en el software cargado en la tarjeta NUCLEO-L432KC, el filtro paso banda. ·La tarjeta miniDK2 debe tener inicialmente una señal sinusoidal de 50 Hz. · A la salida no se puede apreciar señal inicialmente.	
Objetivo	
Se busca comprobar que a medida que se va aumentando la frecuencia de la señal de entrada, entre los 100 y los 200 Hz la señal de salida va aumentando su magnitud, llegando a no haber prácticamente atenuación entre los 200 y los 300 Hz. A partir de los 300 Hz vuelve a atenuarse la señal llegando a no apreciarse, prácticamente, a partir de los 400 Hz.	
Pasos	**Resultado Esperado**
1. Ajustar con el potenciómetro la amplitud de la señal de entrada y tomar la referencia de magnitud en la ventana de representación del espectro de la señal en Visual Analyzer.	En Visual Analyzer se observa que la magnitud se puede ajustar al gusto del usuario e inicialmente la frecuencia de la señal es de 50 Hz
2. Pulsar el botón “KEY 2” de la tarjeta miniDK2.	Se observa que la frecuencia de la señal de entrada ha aumentado a 100Hz.
3. Medir el espectro de la señal de salida.	Con respecto a la señal de entrada, la de salida se ve atenuada en pocos dB de magnitud.
4. Pulsar el botón “KEY 2” de la tarjeta miniDK2 y medir el espectro de la señal que se ve a la salida.	La señal de entrada tiene una frecuencia de 50 Hz más que la señal de entrada en el paso 3 y la señal de salida actual se ve más atenuada que la señal de salida vista en el paso 3 siendo prácticamente inapreciable.
5. Repetir el paso 4, hasta alcanzar una frecuencia de señal de entrada de 450Hz.	A medida que aumenta la frecuencia, la magnitud de la salida es muy baja. A partir de los 300 aumenta la magnitud de la señal de salida, llegando a no haber prácticamente atenuación a partir de los 400Hz.

Tabla 3. Caso de prueba 3 – Filtro paso banda

Ejecutando este caso de prueba se obtienen los siguientes espectros de salida usando las mismas señales de entrada que en los dos filtros anteriores:

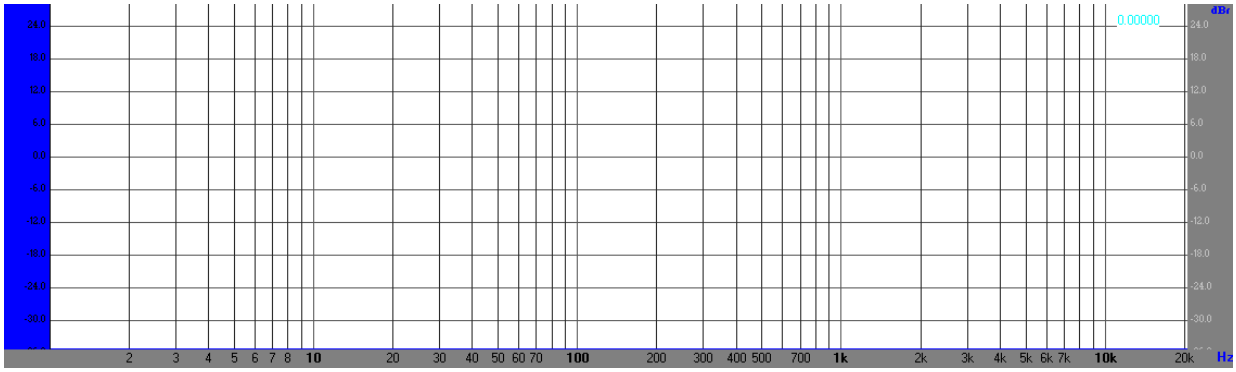

Figura 65. Espectro de salida a 50 Hz – Filtro paso banda

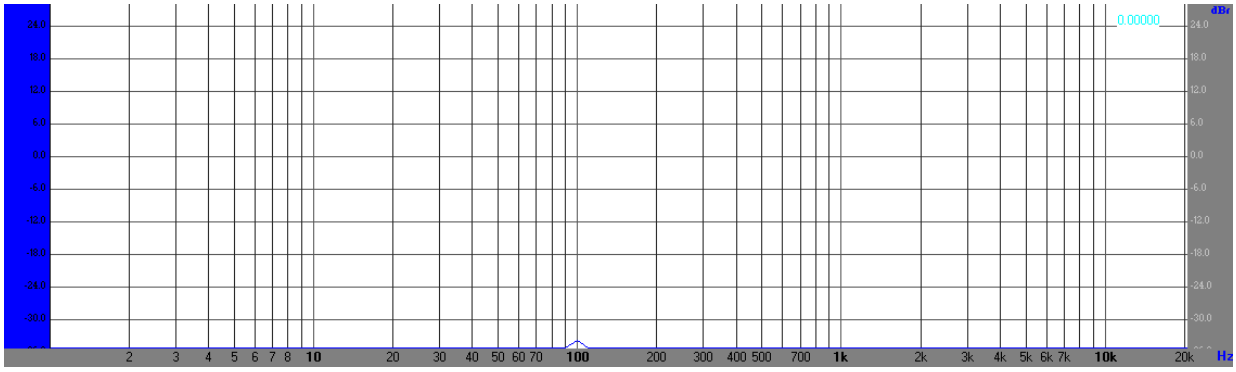

Figura 66. Espectro de salida a 100 Hz – Filtro paso banda

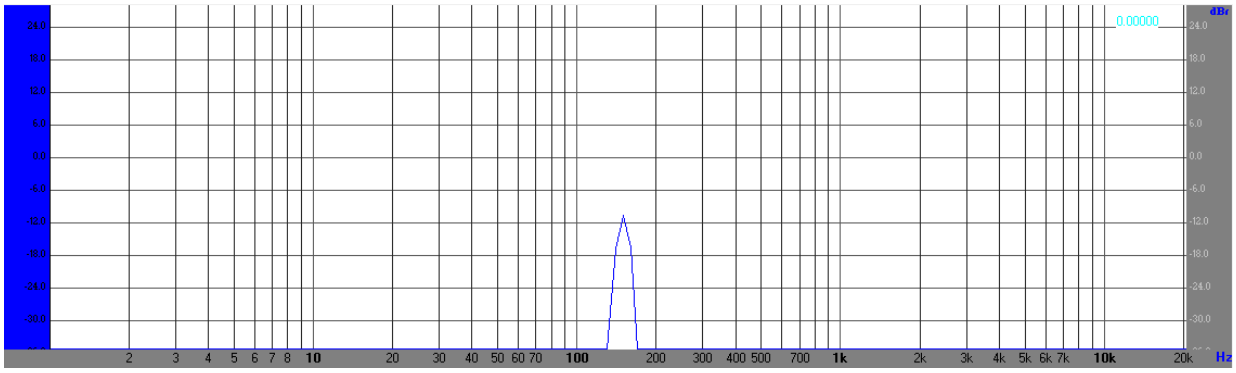

Figura 67. Espectro de salida a 150 Hz – Filtro paso banda

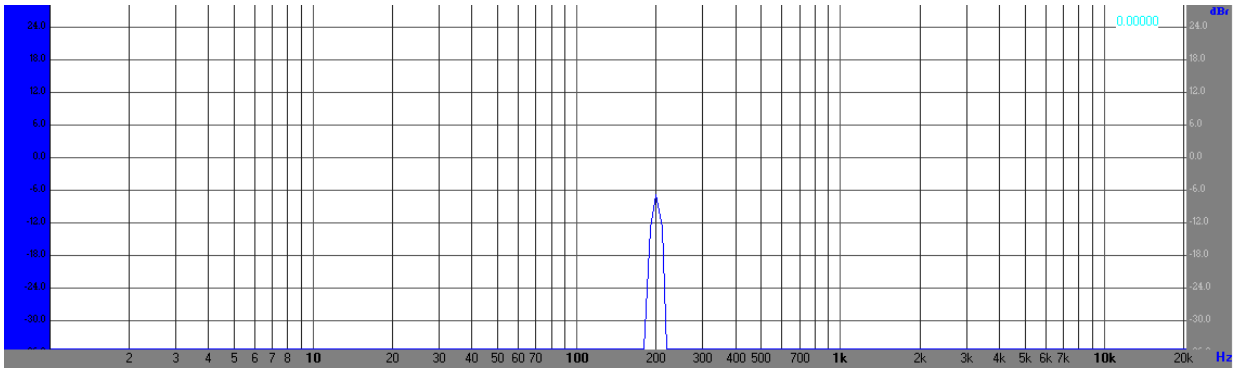

Figura 68. Espectro de salida a 200 Hz – Filtro paso banda

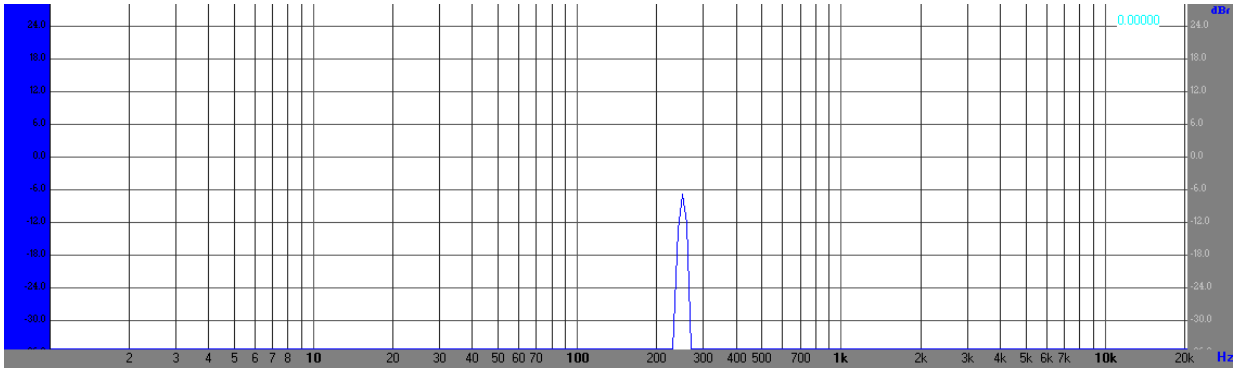

Figura 69. Espectro de salida a 250 Hz – Filtro paso banda

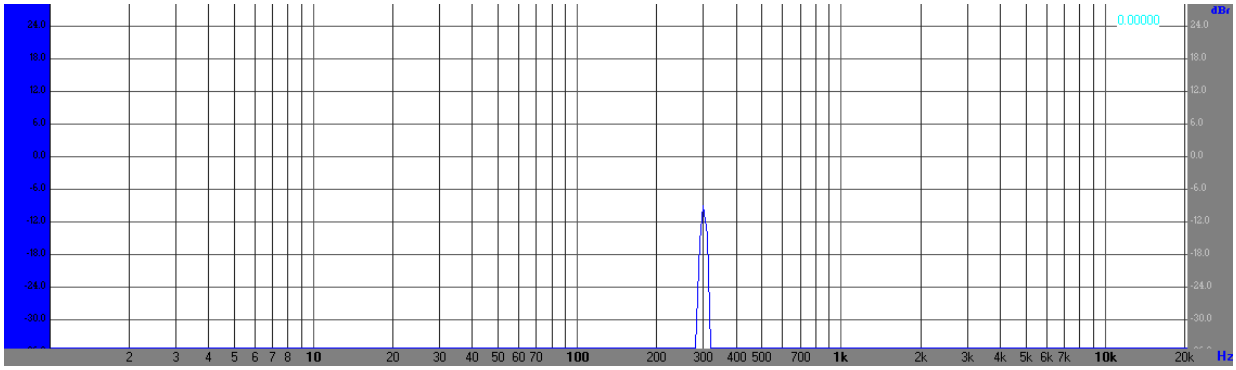

Figura 70. Espectro de salida a 300 Hz – Filtro paso banda

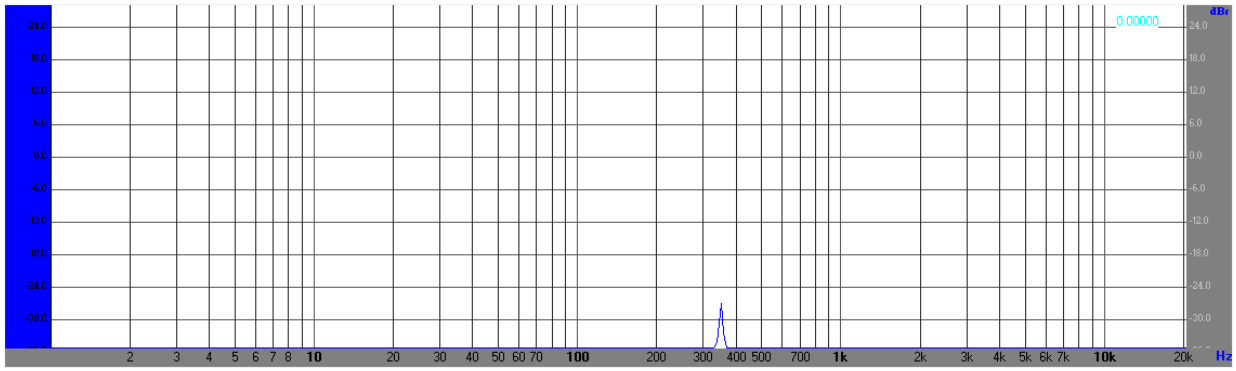

Figura 71. Espectro de salida a 350 Hz – Filtro paso banda

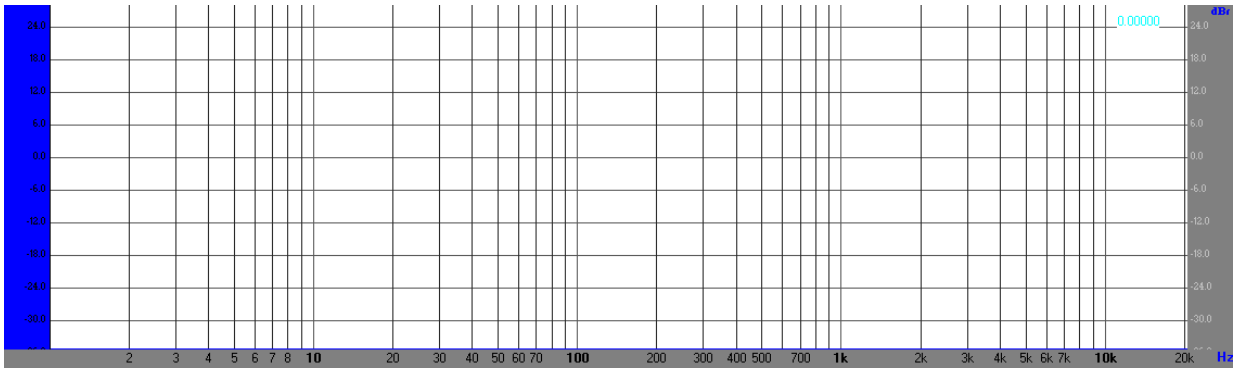

Figura 72. Espectro de salida a 400 Hz – Filtro paso banda

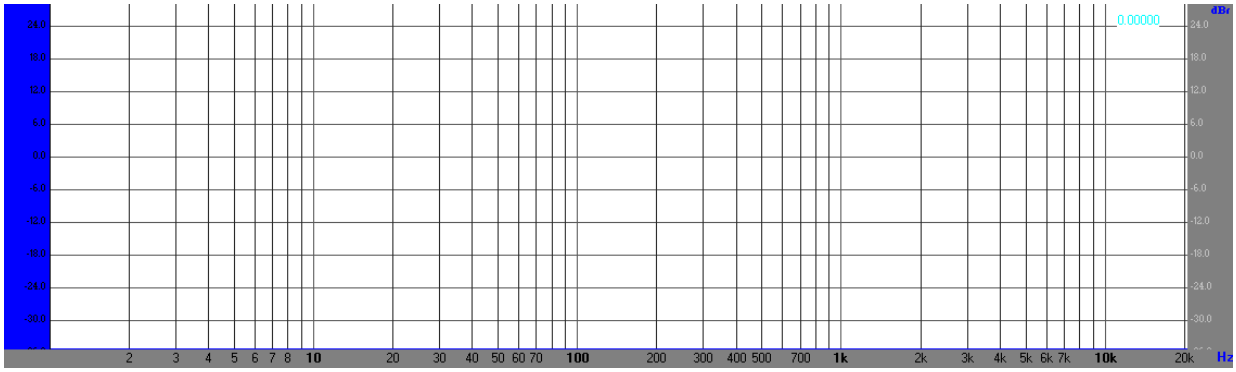

Figura 73. Espectro de salida a 450 Hz – Filtro paso banda

7.4 Filtro FIR Banda Eliminada - Caso de prueba 3

Precondiciones	
·Debe seleccionarse en el software cargado en la tarjeta NUCLEO-L432KC, el filtro banda eliminada. ·La tarjeta miniDK2 debe tener inicialmente una señal sinusoidal de 50 Hz.	
Objetivo	
Se busca comprobar que a medida que se va aumentando la frecuencia de la señal de entrada, entre los 100 y los 200 Hz la señal de salida se va atenuando, llegando a no apreciarse entre los 200 y los 300 Hz. A partir de los 300 Hz vuelve a aumentar la magnitud de la señal llegando a apreciarse, sin prácticamente atenuaciones, a partir de los 400 Hz.	
Pasos	**Resultado Esperado**
1. Ajustar con el potenciómetro la amplitud de la señal de entrada y tomar la referencia de magnitud en la ventana de representación del espectro de la señal en Visual Analyzer.	En Visual Analyzer se observa que la magnitud se puede ajustar al gusto del usuario e inicialmente la frecuencia de la señal es de 50 Hz
2. Pulsar el botón “KEY 2” de la tarjeta miniDK2.	Se observa que la frecuencia de la señal de entrada ha aumentado a 100Hz.
3. Medir el espectro de la señal de salida.	Con respecto a la señal de entrada, la de salida se ve atenuada siendo la atenuación prácticamente inapreciable.
4. Pulsar el botón “KEY 2” de la tarjeta miniDK2 y medir el espectro de la señal que se ve a la salida.	La señal de entrada tiene una frecuencia de 50 Hz más que la señal de entrada en el paso 3 y la señal de salida actual se ve más atenuada que la señal de salida vista en el paso 3 siendo prácticamente inapreciable.
5. Repetir el paso 4, hasta alcanzar una frecuencia de señal de entrada de 450Hz.	A medida que aumenta la frecuencia, la magnitud de la salida es muy baja. A partir de los 300 aumenta la magnitud de la señal de salida, llegando a no haber prácticamente atenuación a partir de los 400Hz.

Tabla 4. Caso de prueba 4 – Filtro banda eliminada

Utilizando las mismas señales de entrada que en el resto de filtros, se muestran a continuación las salidas del filtro banda eliminada:

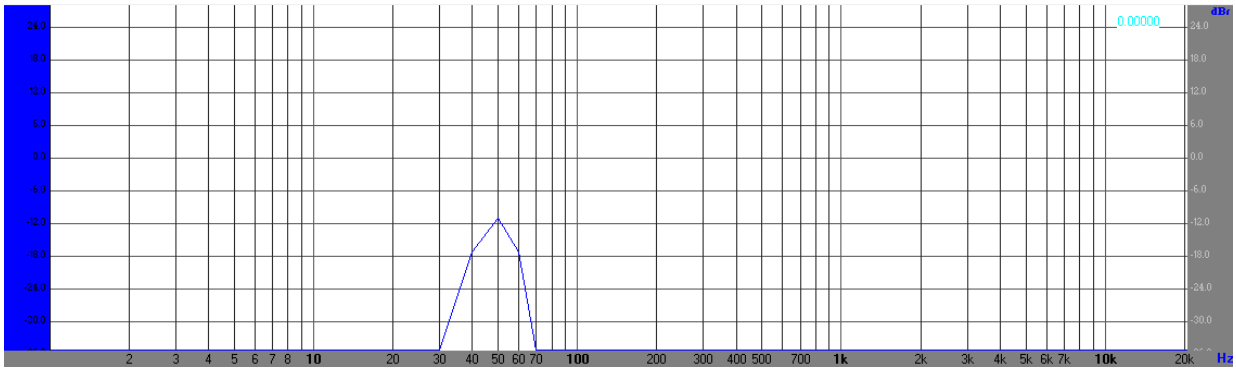

Figura 74. Espectro de salida a 50 Hz – Filtro banda eliminada

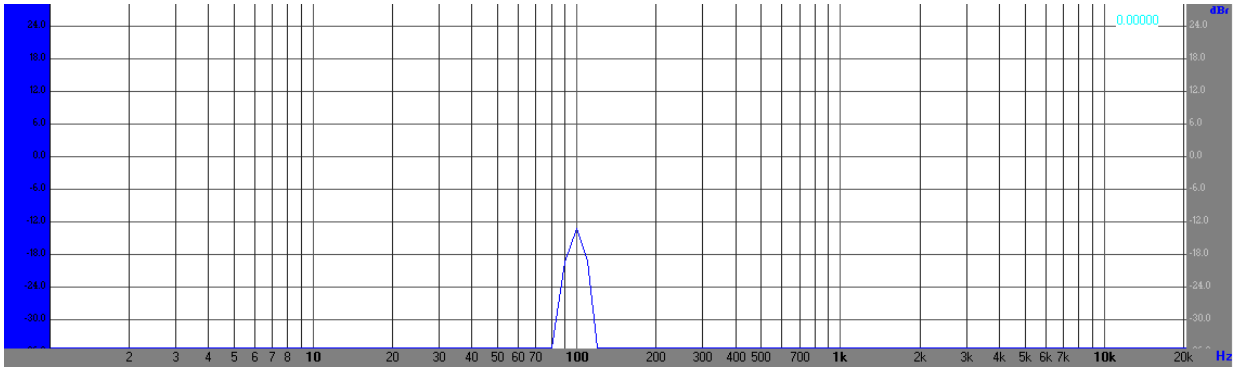

Figura 75. Espectro de salida a 100 Hz – Filtro banda eliminada

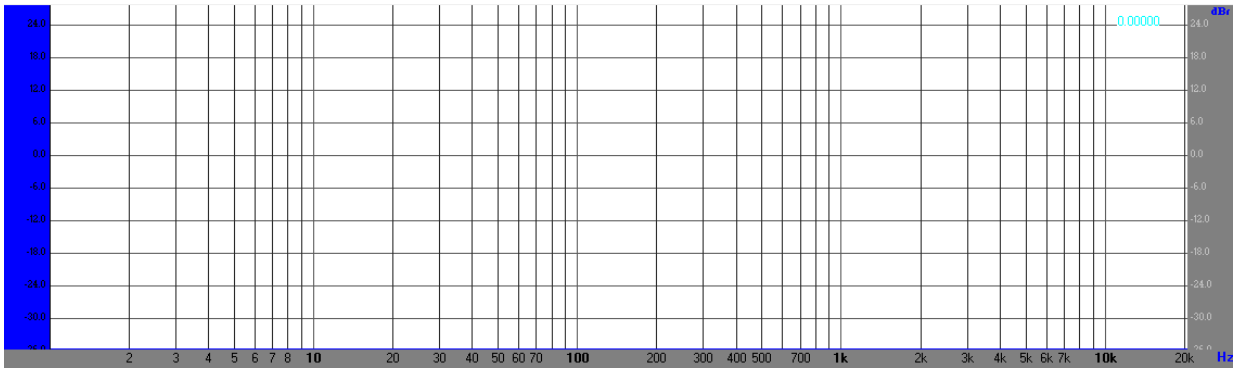

Figura 76. Espectro de salida a 150 Hz – Filtro banda eliminada

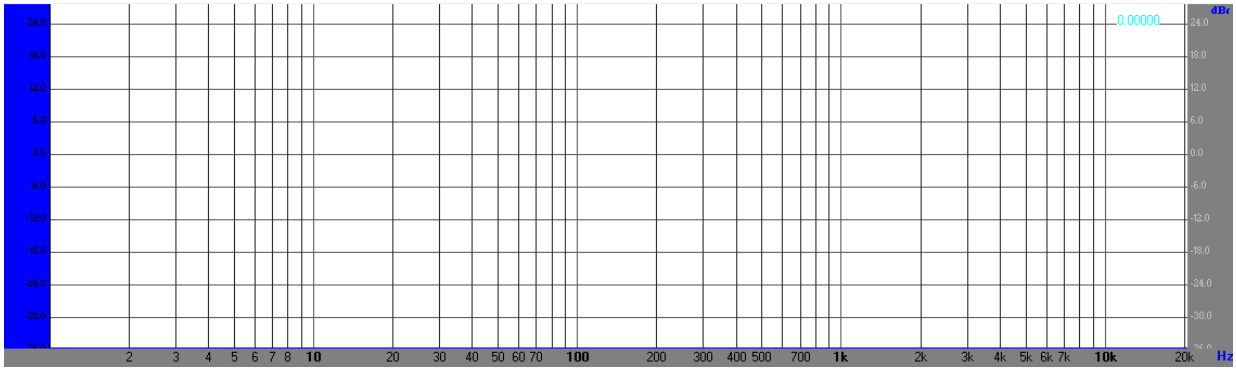

Figura 77. Espectro de salida a 200 Hz – Filtro banda eliminada

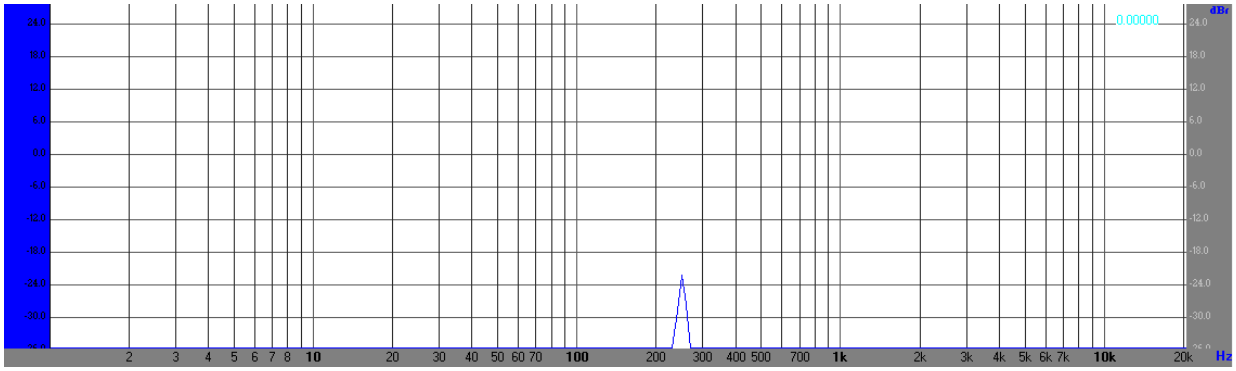

Figura 78. Espectro de salida a 250 Hz – Filtro banda eliminada

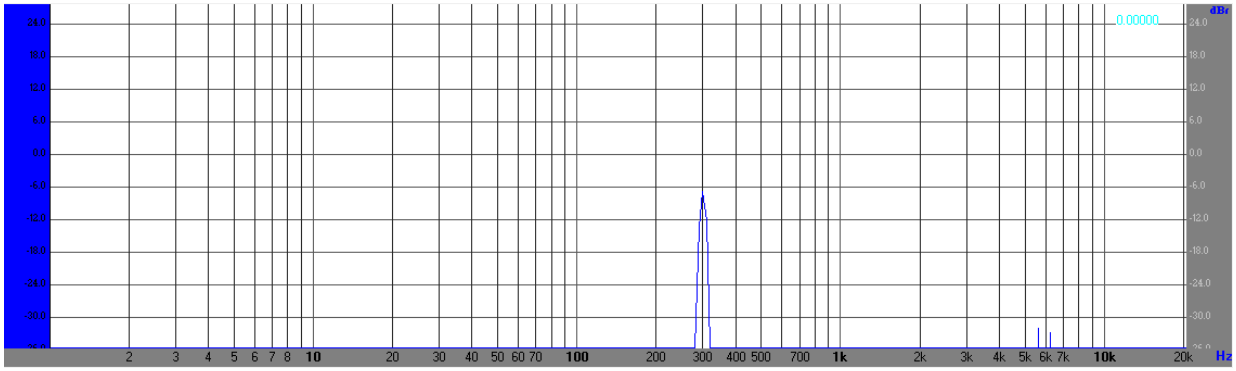

Figura 79. Espectro de salida a 300 Hz – Filtro banda eliminada

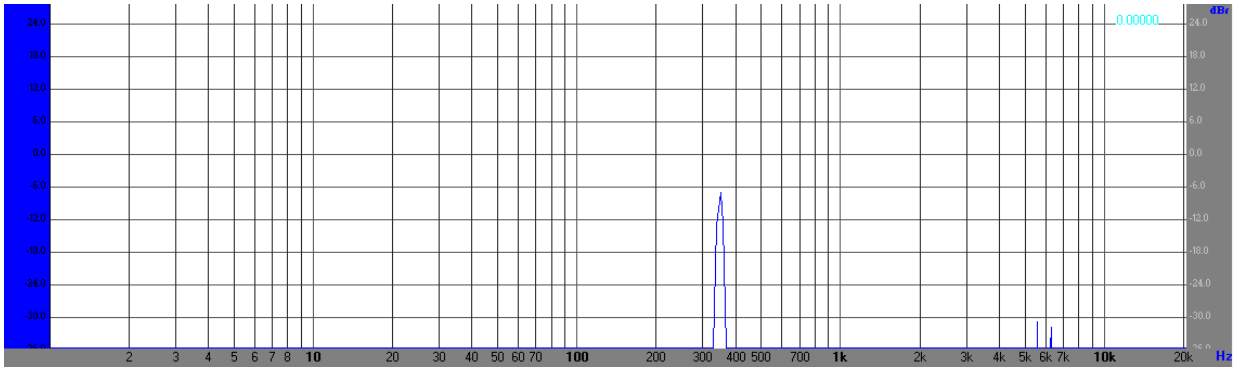

Figura 80. Espectro de salida a 350 Hz – Filtro banda eliminada

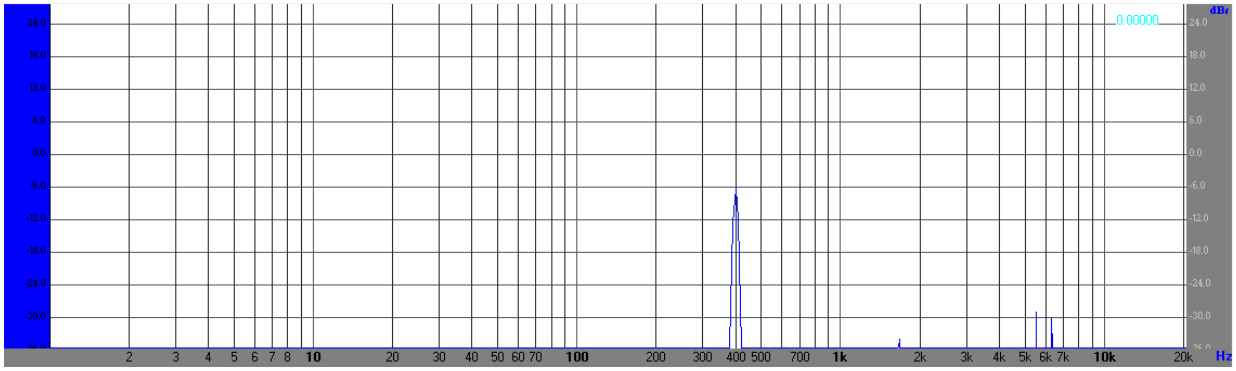

Figura 81. Espectro de salida a 400 Hz – Filtro banda eliminada

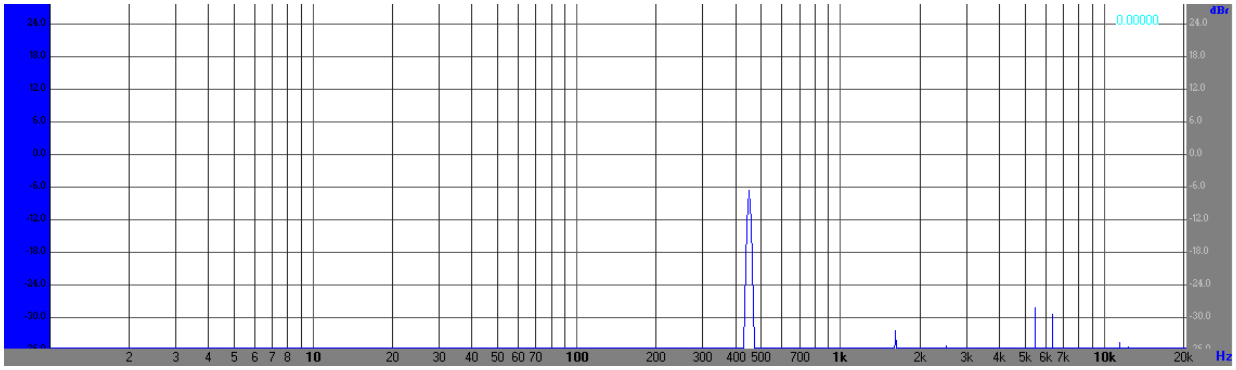

Figura 82. Espectro de salida a 450 Hz – Filtro banda eliminada

8.Resultados y Conclusiones

En base a los resultados obtenidos con respecto a la ejecución del plan de pruebas, se puede decir que los 4 filtros implementados, cumplían con las expectativas de funcionamiento definidas en el plan de pruebas.

Analizando cada filtro por separado, si se observan las figuras de espectros de salida de cada uno de los filtros se puede ver que la salida se iba atenuando o aumentando conforme a lo esperado, respetado las bandas de paso y de parada definidas en el diseño de los filtros. Se puede observar en el caso de los filtros paso banda y banda eliminada que las bandas de transición tienen una anchura de tan solo 100 Hz, con lo que, al aumentar la frecuencia de la señal de entrada al filtro en saltos de 50 Hz solamente se obtiene un resultado por cada banda de transición que tenga el filtro. Por ejemplo, en el caso de los filtros paso alto y paso bajo las bandas de transición tienen una anchura de 300 Hz, pudiendo tener, en este caso, hasta 5 resultados de filtrado en cada banda de transición.

La tarjeta utilizada para este proyecto (NUCLEO-L432KC), el software que interviene en el desarrollo de software embebido para microcontroladores de la familia de STM32, han facilitado en gran medida el poder realizar la parte del funcional del proyecto, con más agilidad y fluidez teniendo en cuenta existen ejemplos de código de uso de periféricos como el ADC, el DAC o el Timer que están basados en las "*HAL Libraries*", que aunque estén diseñados para otros microcontroladores de la familia STM32, permiten al desarrollador, programar a un nivel más alto sin necesidad de conocer a ciencia cierta los registros del microcontrolador que intervienen en los periféricos, ya que generalmente estos periféricos son similares para la familia STM32.

El diseño de filtros se ha visto condicionado por la frecuencia de muestreo escogida. Es decir, a la hora de querer analizar una señal, es necesario tener la mayor cantidad de muestras posible. Es por esta razón por la que no se ha optado por usar señales con frecuencias de más de 500 Hz, ya que con una frecuencia de muestreo de 8 KHz se obtienen 16 muestras por periodo de la señal. En el caso de que se hubiese usado una frecuencia de muestreo de, por ejemplo, 16 KHz y se quisiese obtener un mínimo de 16 muestras por periodo sería necesario que la frecuencia de la señal a muestrear tuviese como máximo 1KHz de frecuencia. Por un lado, puede resultar ventajoso porque, esta situación permite crear filtros digitales con un mayor margen de frecuencias. Por otro lado, puede no resultar ser tan ventajoso si se tiene en cuenta que, al muestrear al doble de frecuencia, se tiene la mitad de tiempo que en la situación anterior y probablemente se tengan más coeficientes al tener una frecuencia de muestreo de mayor, requiriendo así mayor carga computacional.

9. Trabajo futuro

Para seguir en la línea de investigación del procesado digital de señal en tiempo real en microcontroladores, en este apartado se va a proponer una línea futura de investigación.

Esta nueva línea de investigación consiste en desarrollar aplicaciones de procesado de señales de video en tiempo real sobre microcontroladores de la familia STM32 evaluando las ventajas y desventajas que puede tener el uso de un microcontrolador con arquitectura del Cortex-M4 frente uno del Cortex-M7, analizando el rendimiento y la cantidad de recursos utilizados por ambos tipos de microcontroladores.

Como tarjetas de desarrollo se proponen NUCLEO-144 (STM32F439ZI) y por otro lado la STM32F746 Discovery Board. Ambas, al ser de la familia de STM32 pueden ser configuradas mediante CubeMX.

El entorno de desarrollo que se propone, en este caso no es Keil uVision 5, sino Atollic TrueSTUDIO, ya que es un software de desarrollo, basado en Eclipse, que a diferencia de Keil, permite su uso comercial de manera gratuita sin necesidad de adquirir licencia.

10. Pliego de condiciones

En este apartado, se hace referencia a los elementos que son necesarios para poder implementar una aplicación de filtrado de señales en tiempo real como la que se ha descrito en este proyecto.

10.1 Elementos Hardware

Los elementos hardware hacen referencia los elementos físicos que se han utilizado:

- Tarjeta de desarrollo NUCLEO-L432KC.
- Tarjeta de desarrollo LPC1768-Mini-DK2.
- Módulo de depuración J-LINK.
- PC con sistema operativo Windows 10, con procesador de 64 bits, con al menos 4GB de RAM y con 500MB de memoria disponibles.
- Tarjeta de sonido externa Tecknet o similares.

10.1 Elementos Hardware

Los elementos software hacen referencia a los programas y entornos de diseño, desarrollo y de pruebas que se han utilizado:

- Programa STM32CubeMX para la preconfiguración de la tarjeta NUCLEO-L432KC.
- Programa Keil uVision 5 para el desarrollo del código de programación.
- Programa Matlab R2018a para el diseño de filtros FIR y generación de los coeficientes.
- Programa Visual Analyzer 2014 para la visualización de señales.

11. Presupuesto

Para la implementación de una aplicación de filtrado de señales en tiempo real como la que se ha descrito en este proyecto se realiza un estudio de los costes a asumir.

En la tabla siguiente se muestran los costes de equipo y software que son necesarios para la implementación de este proyecto.

Concepto	Precio	Periodo de amortización (meses)	Periodo de uso (meses)	Coste para el proyecto
Programa STM32CubeMX	0€	N/A	5	0€
Programa Keil uVision 5	1535€	18	5	426.39
Programa Matlab R2018a	2000€	18	5	555.56
Programa Visual Analyzer	0€	N/A	5	0€
PC Acer Aspire E5-573G	569,00€	18	5	158,05€
Coste total de equipo y software				1140€

Tabla 5. Coste total de equipo y software

A continuación, se muestra la tabla que refleja los costes del material empleado.

Concepto	Precio [€/ud]	Cantidad	Coste para el proyecto
Tarjeta NUCLEO-L432KC	9,37	1	9,37€
Tarjeta LPC1768-Mini-DK2	23,03	1	23,03€
Depurador J-LINK	8,65	1	8,65€
Tarjeta de sonido Tecknet	6,99	1	6,99€
Coste total del material			48,04€

Tabla 6. Coste total del material

Teniendo en cuenta que el precio por hora de un ingeniero electrónico de software embebido está en torno a 60€/hora, se realiza en la Tabla 7 un desglose con la estimación de horas que necesitaría un empleado para desarrollar una aplicación de este tipo.

Concepto	Cantidad [horas]	Precio [€/hora]	Coste para el proyecto
Investigación acerca del procesado digital de señales	25	60	1500€
Estudio en profundidad de la tarjeta NUCLEO-L432KC	20	60	1200€
Estudio de la tarjeta Mini-DK2	20	60	1200€
Desarrollo del código en C de ambos microcontroladores.	12	60	720€
Diseño y Ejecución de pruebas funcionales	4	60	240€
Coste total de mano de obra			4896€

Tabla 7. Costes de mano de obra

Teniendo en cuenta los costes reflejados en las Tabla 5, 6 y 7, se refleja en la Tabla 8 el coste total de ejecución de material del proyecto.

Concepto	Precio total
Coste total de equipo y software	1140€
Coste total del material	48,04€
Coste total de mano de obra	4896€
Coste de ejecución de material	5529,04€

Tabla 8. Coste total de ejecución de material

A partir del coste total de ejecución de material se genera un presupuesto de contratación.

Concepto	Precio total
Coste total de ejecución de material	5529,04€
Gastos generales (10%)	552,90€
Margen de beneficio (10%)	552,90€
Presupuesto de contratación	6634,85€

Tabla 9. Presupuesto de contratación

Teniendo en cuenta el presupuesto de contratación y el Impuesto de Valor Añadido, se genera el presupuesto total del proyecto.

Concepto	Precio total
Presupuesto de contratación	**6634,85€**
I.V.A. (21%)	**1393,32€**
Presupuesto total del proyecto	**8028,17€**

Tabla 10. Presupuesto total del proyecto

10. Bibliografía

[1] Tarjeta NUCLEO-L432KC: https://www.st.com/en/evaluation-tools/nucleo-l432kc.htmlç

[2] Visual Analyzer 2014: http://www.sillanumsoft.org

[3] Tarjeta de sonido externa TECKNET: https://www.amazon.es/TeckNet-Microfono-Ordenador-Altavoces-Auriculares/dp/B000R5NJD8/ref=asc_df_B000R5NJD8/?tag=googshopes-21&linkCode=df0&hvadid=321105631818&hvpos=1o1&hvnetw=g&hvrand=4771054401842910478&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=1005487&hvtargid=pla-562656984131&psc=1

[4] Conceptos de DSP – Diseñando con el Cortex-M4: https://www.arm.com/files/pdf/DSPConceptsM4Presentation.pdf

[6] Software CubeMX: https://www.st.com/en/development-tools/stm32cubemx.html

[7] Filtros FIR (Wikipedia): https://es.wikipedia.org/wiki/FIR_(Finite_Impulse_Response)

[8] Keil uVision 5: http://www2.keil.com/mdk5/uvision/

[9] Atollic TrueSTUDIO: https://atollic.com/truestudio/

[10] Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C (Third Edition)

[11] Programming Manual for STM32 – CortexM4: https://www.st.com/content/ccc/resource/technical/document/programming_manual/6c/3a/cb/e7/e4/ea/44/9b/DM00046982.pdf/files/DM00046982.pdf/jcr:content/translations/en.DM00046982.pdf

[12] Reference Manual for STM32L4: https://www.st.com/content/ccc/resource/technical/document/reference_manual/02/35/09/0c/4f/f7/40/03/DM00083560.pdf/files/DM00083560.pdf/jcr:content/translations/en.DM00083560.pdf

[13] User Manual for STM32L4 – HAL drivers: https://www.st.com/content/ccc/resource/technical/document/user_manual/63/a8/8f/e3/ca/a1/4c/84/DM00173145.pdf/files/DM00173145.pdf/jcr:content/translations/en.DM00173145.pdf

11. Anexos

11.1 Código para la generación de señales de entrada – LPC1768

```
/*****************************************************************/
/* Genera una se□senoidal con el DAC de frecuencia F_out         */
/* Timer 1 en modo Match (MAT1.0 -> salida habilitamos modo Toggle )*/
/*****************************************************************/

#include <LPC17xx.H>
#include <Math.h>
#define F_cpu 100e6         // Defecto Keil (xtal=12Mhz)
#define F_pclk F_cpu/4  // Defecto despues del reset
#define pi 3.1459
#define N_muestras 256  // 256 muestras/ciclo
#define V_refp 3.3
uint16_t muestras[N_muestras];
uint16_t F_out = 50;  // 200 Hz

void genera_muestras(uint16_t muestras_ciclo)
{
    uint16_t i;
    //se□senoidal
    for(i=0;i<muestras_ciclo;i++)
    muestras[i]=(uint32_t)(511+511*sin(2*pi*i/N_muestras)); // Ojo! el DAC
es de 10bits
}


// Timer 1 interrumpe peri□amente a F = F_out x N_muestras !!!!
// La muestra correpondiente del array se saca al DAC en cada interrupci□
void TIMER1_IRQHandler(void)
{
    static uint16_t indice_muestra;
    LPC_TIM1->IR|= (1<<0);                                    // borrar
flag
    LPC_DAC->DACR= muestras[indice_muestra++] << 6; // bit6..bit15
    indice_muestra&= N_muestras-1;                        // contador circular
(Si N_muestras potencia de 2)
}

void init_DAC(void)
{
    LPC_PINCON->PINSEL1|= (2<<20);      // DAC output = P0.26 (AOUT)
    LPC_PINCON->PINMODE1|= (2<<20);     // Deshabilita pullup/pulldown
    LPC_DAC->DACCTRL=0;                             //
}


/*  Timer 1 en modo Output Compare (reset T0TC on Match 0)
      Counter clk: 25 MHz   MAT1.0 : On match, salida de una muestra hacia
el DAC */
void init_TIMER1(uint16_t Freq)
{
      LPC_SC->PCONP|=(1<<2);                          //  Power ON
      LPC_TIM1->TCR = 0x00;                           //  Start Timer

    LPC_TIM1->PR = 0x00;                          //  Prescaler =1
    LPC_TIM1->MCR = 0x03;                           //  Reset TC on Match, e
interrumpe!
    LPC_TIM1->MR0 = (F_pclk/Freq/N_muestras)-1;  // Cuentas hasta el Match
```

```
    LPC_TIM1->EMR = 0x00;                          //  No acta sobre el HW
    LPC_TIM1->TCR = 0x01;                          //  Start Timer
    NVIC_EnableIRQ(TIMER1_IRQn);                 //  Habilita NVIC
      NVIC_SetPriority(TIMER1_IRQn,1);  //  Nivel 1 prioridad
}
void init_EINTs(void){
    LPC_PINCON->PINSEL4|=(0x01<<20);          // P2.10 es entrada interrup.
EXT 1
    LPC_PINCON->PINSEL4|=(0x01<<24);          // P2.11 es entrada interrup.
EXT 2
    LPC_SC->EXTMODE|=(1<<2)|(1<<0);                          // Por Flanco,
    LPC_SC->EXTPOLAR=0;                                       // de bajada


    NVIC_SetPriority(EINT2_IRQn, 0);             // M□prioritario
    NVIC_SetPriority(EINT0_IRQn, 0);             // M□prioritario

    NVIC_EnableIRQ(EINT2_IRQn); // sin CMSIS: NVIC->ISER[0]=(1<<19);
    NVIC_EnableIRQ(EINT0_IRQn); // sin CMSIS: NVIC->ISER[0]=(1<<20);
}

void EINT0_IRQHandler(){
        LPC_SC->EXTINT|=(1<<0);

        if (F_out>50){
            F_out-=50;
            init_TIMER1(F_out);
            LPC_TIM1->TCR = 0x2;//  Reset Timer
            LPC_TIM1->TCR   =   0x1;//  Start Timer

        }


}

void EINT2_IRQHandler(){
        LPC_SC->EXTINT=(1<<2);

        if (F_out<450){
            F_out+=50;
            init_TIMER1(F_out);
            LPC_TIM1->TCR = 0x2;//  Reset Timer
            LPC_TIM1->TCR = 0x1;//  Start Timer
            }
}


int main(void)
{
//  NVIC_SetPriorityGrouping(2);

    genera_muestras(N_muestras);
    init_DAC();
    init_TIMER1(F_out);
    init_EINTs();
    while(1);
}
```

11.2 Código del fichero “main.c” – NUCLEO-L432KC

```
/**

******************************************************************************
**
  * @file           : main.c
  * @brief          : Main program body

******************************************************************************
**
  ** This notice applies to any and all portions of this file
  * that are not between comment pairs USER CODE BEGIN and
  * USER CODE END. Other portions of this file, whether
  * inserted by the user or by software development tools
  * are owned by their respective copyright owners.
  *
  * COPYRIGHT(c) 2019 STMicroelectronics
  *
  * Redistribution and use in source and binary forms, with or without
modification,
  * are permitted provided that the following conditions are met:
  *   1. Redistributions of source code must retain the above copyright
notice,
  *      this list of conditions and the following disclaimer.
  *   2. Redistributions in binary form must reproduce the above copyright
notice,
  *      this list of conditions and the following disclaimer in the
documentation
  *      and/or other materials provided with the distribution.
  *   3. Neither the name of STMicroelectronics nor the names of its
contributors
  *      may be used to endorse or promote products derived from this
software
  *      without specific prior written permission.
  *
  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS"
  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE
  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE
  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE
  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR
  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER
  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY,
  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF
THE USE
  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
  *

******************************************************************************
**
  */
/* Includes ------------------------------------------------------------
--*/
```

```
#include "main.h"
#include "stm32l4xx_hal.h"

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;

DAC_HandleTypeDef hdac1;

TIM_HandleTypeDef htim6;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/
uint16_t value=0;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_ADC1_Init(void);
static void MX_DAC1_Init(void);
static void MX_TIM6_Init(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------------*/

/* USER CODE END PFP */

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  *
  * @retval None
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
```

```
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_ADC1_Init();
  MX_DAC1_Init();
  MX_TIM6_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start(&htim6);
    HAL_DAC_Start(&hdac1,DAC_CHANNEL_1);
    HAL_ADC_Start_IT(&hadc1);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {

  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */

  }
  /* USER CODE END 3 */

}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_PeriphCLKInitTypeDef PeriphClkInit;

    /**Configure LSE Drive Capability
    */
  HAL_PWR_EnableBkUpAccess();

  __HAL_RCC_LSEDRIVE_CONFIG(RCC_LSEDRIVE_LOW);

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_LSE|RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.LSEState = RCC_LSE_ON;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.MSICalibrationValue = 0;
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_6;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_MSI;
  RCC_OscInitStruct.PLL.PLLM = 1;
  RCC_OscInitStruct.PLL.PLLN = 16;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
  RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
  RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
```

```
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

  PeriphClkInit.PeriphClockSelection =
RCC_PERIPHCLK_USART2|RCC_PERIPHCLK_ADC;
  PeriphClkInit.Usart2ClockSelection = RCC_USART2CLKSOURCE_PCLK1;
  PeriphClkInit.AdcClockSelection = RCC_ADCCLKSOURCE_PLLSAI1;
  PeriphClkInit.PLLSAI1.PLLSAI1Source = RCC_PLLSOURCE_MSI;
  PeriphClkInit.PLLSAI1.PLLSAI1M = 1;
  PeriphClkInit.PLLSAI1.PLLSAI1N = 16;
  PeriphClkInit.PLLSAI1.PLLSAI1P = RCC_PLLP_DIV7;
  PeriphClkInit.PLLSAI1.PLLSAI1Q = RCC_PLLQ_DIV2;
  PeriphClkInit.PLLSAI1.PLLSAI1R = RCC_PLLR_DIV2;
  PeriphClkInit.PLLSAI1.PLLSAI1ClockOut = RCC_PLLSAI1_ADC1CLK;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Configure the main internal regulator output voltage
    */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) !=
HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Configure the Systick interrupt time
    */
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

    /**Configure the Systick
    */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

    /**Enable MSI Auto calibration
    */
  HAL_RCCEx_EnableMSIPLLMode();

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* ADC1 init function */
static void MX_ADC1_Init(void)
{

  ADC_ChannelConfTypeDef sConfig;

    /**Common config
```

```
    */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  hadc1.Init.LowPowerAutoWait = DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_EXTERNALTRIG_T6_TRGO;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.Overrun = ADC_OVR_DATA_PRESERVED;
  hadc1.Init.OversamplingMode = DISABLE;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Configure Regular Channel
    */
  sConfig.Channel = ADC_CHANNEL_8;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_2CYCLES_5;
  sConfig.SingleDiff = ADC_SINGLE_ENDED;
  sConfig.OffsetNumber = ADC_OFFSET_NONE;
  sConfig.Offset = 0;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

}

/* DAC1 init function */
static void MX_DAC1_Init(void)
{

  DAC_ChannelConfTypeDef sConfig;

    /**DAC Initialization
    */
  hdac1.Instance = DAC1;
  if (HAL_DAC_Init(&hdac1) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**DAC channel OUT1 config
    */
  sConfig.DAC_SampleAndHold = DAC_SAMPLEANDHOLD_DISABLE;
  sConfig.DAC_Trigger = DAC_TRIGGER_NONE;
  sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_DISABLE;
  sConfig.DAC_ConnectOnChipPeripheral = DAC_CHIPCONNECT_DISABLE;
  sConfig.DAC_UserTrimming = DAC_TRIMMING_FACTORY;
  if (HAL_DAC_ConfigChannel(&hdac1, &sConfig, DAC_CHANNEL_1) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

}

/* TIM6 init function */
```

```
static void MX_TIM6_Init(void)
{

  TIM_MasterConfigTypeDef sMasterConfig;

  htim6.Instance = TIM6;
  htim6.Init.Prescaler = 0;
  htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim6.Init.Period = 3999;
  htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) !=
HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

}

/* USART2 init function */
static void MX_USART2_UART_Init(void)
{

  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

}

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
static void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD3_GPIO_Port, LD3_Pin, GPIO_PIN_RESET);
```

```
  /*Configure GPIO pin : LD3_Pin */
  GPIO_InitStruct.Pin = LD3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD3_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  file: The file name as string.
  * @param  line: The line in file as a number.
  * @retval None
  */
void _Error_Handler(char *file, int line)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return
state */
  while(1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/**
  * @}
  */

/**
  * @}
  */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF
FILE****/
```

11.3 Código del fichero “stm32l4xx_it.c” – NUCLEO-L432KC

```
/**

******************************************************************************
**
  * @file    stm32l4xx_it.c
  * @brief   Interrupt Service Routines.

******************************************************************************
**
  *
  * COPYRIGHT(c) 2019 STMicroelectronics
  *
  * Redistribution and use in source and binary forms, with or without
modification,
  * are permitted provided that the following conditions are met:
  *   1. Redistributions of source code must retain the above copyright
notice,
  *      this list of conditions and the following disclaimer.
  *   2. Redistributions in binary form must reproduce the above copyright
notice,
  *      this list of conditions and the following disclaimer in the
documentation
  *      and/or other materials provided with the distribution.
  *   3. Neither the name of STMicroelectronics nor the names of its
contributors
  *      may be used to endorse or promote products derived from this
software
  *      without specific prior written permission.
  *
  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS"
  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE
  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE
  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE
  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR
  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER
  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY,
  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF
THE USE
  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
  *

******************************************************************************
**
  */
/* Includes ------------------------------------------------------------
--*/
#include "stm32l4xx_hal.h"
#include "stm32l4xx.h"
#include "stm32l4xx_it.h"

/* USER CODE BEGIN 0 */
#define ORDEN(a) sizeof(a)/sizeof(float)
```

```
extern DAC_HandleTypeDef hdac1;
extern uint16_t value;
float y=0;
float aux=0;
uint8_t cont=0;

//          //FPB
//float h[]={-0.00016188,-9.115e-05,-9.5725e-05,-7.8036e-05,-2.6454e-
05,7.2991e-
05,0.00023655,0.00048253,0.00083079,0.001302,0.0019169,0.0026958,0.0036572,0
.0048167,0.0061866,0.0077741,0.009581,0.011602,0.013827,0.016235,0.018801,0.
021491,0.024264,0.027076,0.029876,0.032609,0.03522,0.037654,0.039857,0.04177
9,0.043375,0.044607,0.045445,0.04587,0.04587,0.045445,0.044607,0.043375,0.04
1779,0.039857,0.037654,0.03522,0.032609,0.029876,0.027076,0.024264,0.021491,
0.018801,0.016235,0.013827,0.011602,0.009581,0.0077741,0.0061866,0.0048167,0
.0036572,0.0026958,0.0019169,0.001302,0.00083079,0.00048253,0.00023655,7.299
1e-05,-2.6454e-05,-7.8036e-05,-9.5725e-05,-9.115e-05,-0.00016188
//};

//          //FPA
//float h[]={0.011207,-0.0033369,-0.0032438,-0.0033278,-0.0034898,-
0.0036336,-0.0036696,-0.0035157,-0.0031084,-0.0024022,-0.0013779,-4.4606e-
05,0.0015638,0.0033699,0.005284,0.0071737,0.0088959,0.010294,0.011202,0.0114
63,0.010929,0.0094866,0.0070485,0.0035711,-0.00093957,-0.0064264,-0.012774,-
0.019822,-0.02737,-0.035187,-0.043001,-0.050505,-0.057461,-0.063567,-
0.068596,-0.072342,-0.074653,0.92457,-0.074653,-0.072342,-0.068596,-
0.063567,-0.057461,-0.050505,-0.043001,-0.035187,-0.02737,-0.019822,-
0.012774,-0.0064264,-
0.00093957,0.0035711,0.0070485,0.0094866,0.010929,0.011463,0.011202,0.010294
,0.0088959,0.0071737,0.005284,0.0033699,0.0015638,-4.4606e-05,-0.0013779,-
0.0024022,-0.0031084,-0.0035157,-0.0036696,-0.0036336,-0.0034898,-
0.0033278,-0.0032438,-0.0033369,0.011207
//};

//

//          //FPBanda
float h[]={-0.0031755,0.0040122,0.0018034,0.00043491,-0.00047973,-
0.0011725,-0.00177,-0.0023299,-0.0028625,-0.0033528,-0.0037677,-0.0040705,-
0.0042222,-0.0041908,-0.0039556,-0.003508,-0.0028561,-0.0020217,-
0.0010424,3.0431e-
05,0.0011359,0.002208,0.0031796,0.0039876,0.0045793,0.0049163,0.0049802,0.00
47726,0.0043178,0.0036635,0.0028772,0.0020413,0.0012466,0.00058914,0.0001581
1,2.8045e-
05,0.00025431,0.0008642,0.0018512,0.0031735,0.0047546,0.0064811,0.0082151,0.
0097974,0.011055,0.011825,0.011949,0.011303,0.0097979,0.007395,0.0041114,2.2
052e-05,-0.0047315,-0.0099643,-0.015419,-0.020833,-0.025887,-0.03028,-
0.033721,-0.03594,-0.036724,-0.035931,-0.033485,-0.029397,-0.023772,-
0.016803,-0.0087572,3.1042e-
05,0.0091739,0.018252,0.02684,0.034529,0.040943,0.045766,0.048757,0.049771,0
.048757,0.045766,0.040943,0.034529,0.02684,0.018252,0.0091739,3.1042e-05,-
0.0087572,-0.016803,-0.023772,-0.029397,-0.033485,-0.035931,-0.036724,-
0.03594,-0.033721,-0.03028,-0.025887,-0.020833,-0.015419,-0.0099643,-
0.0047315,2.2052e-
05,0.0041114,0.007395,0.0097979,0.011303,0.011949,0.011825,0.011055,0.009797
4,0.0082151,0.0064811,0.0047546,0.0031735,0.0018512,0.0008642,0.00025431,2.8
045e-
05,0.00015811,0.00058914,0.0012466,0.0020413,0.0028772,0.0036635,0.0043178,0
.0047726,0.0049802,0.0049163,0.0045793,0.0039876,0.0031796,0.002208,0.001135
9,3.0431e-05,-0.0010424,-0.0020217,-0.0028561,-0.003508,-0.0039556,-
0.0041908,-0.0042222,-0.0040705,-0.0037677,-0.0033528,-0.0028625,-
0.0023299,-0.00177,-0.0011725,-0.00047973,0.00043491,0.0018034,0.0040122,-
0.0031755
};
```

```
//          //FBandaElminada
//float h[]={-0.016531,0.0079581,0.0054221,0.0031565,0.0013082,-2.5556e-05,-
0.00081554,-0.0010849,-0.00090901,-
0.00039675,0.00031821,0.0011006,0.0018263,0.0023957,0.0027405,0.0028303,0.00
26715,0.0023025,0.0017862,0.0012027,0.00063727,0.00017005,-0.00013331,-
0.00022685,-8.9565e-
05,0.00027105,0.00082033,0.0015021,0.0022447,0.0029659,0.0035797,0.0040064,0
.004181,0.0040566,0.0036087,0.0028384,0.0017757,0.00047475,-0.00099018,-
0.0025304,-0.0040445,-0.0054324,-0.0065996,-0.0074686,-0.0079691,-
0.0080897,-0.0077836,-0.0071416,-0.0061583,-0.0049455,-0.0036206,-
0.0022882,-0.0010588,-6.7207e-05,0.00058027,0.00081214,0.00059381,-8.0875e-
05,-0.0011764,-0.0026089,-0.0042474,-0.0059289,-0.0074701,-0.0086782,-
0.0093622,-0.0093483,-0.0084951,-0.0067096,-0.0039591,-
0.00028022,0.0042195,0.0093636,0.014912,0.020573,0.026016,0.030893,0.034858,
0.037591,0.038819,0.038345,0.036055,0.031931,0.026057,0.018621,0.0099092,0.0
0029094,-0.0098039,-0.019898,-0.029498,-0.038122,-0.045344,-0.050782,-
0.054164,0.94469,-0.054164,-0.050782,-0.045344,-0.038122,-0.029498,-
0.019898,-
0.0098039,0.00029094,0.0099092,0.018621,0.026057,0.031931,0.036055,0.038345,
0.038819,0.037591,0.034858,0.030893,0.026016,0.020573,0.014912,0.0093636,0.0
042195,-0.00028022,-0.0039591,-0.0067096,-0.0084951,-0.0093483,-0.0093622,-
0.0086782,-0.0074701,-0.0059289,-0.0042474,-0.0026089,-0.0011764,-8.0875e-
05,0.00059381,0.00081214,0.00058027,-6.7207e-05,-0.0010588,-0.0022882,-
0.0036206,-0.0049455,-0.0061583,-0.0071416,-0.0077836,-0.0080897,-
0.0079691,-0.0074686,-0.0065996,-0.0054324,-0.0040445,-0.0025304,-
0.00099018,0.00047475,0.0017757,0.0028384,0.0036087,0.0040566,0.004181,0.004
0064,0.0035797,0.0029659,0.0022447,0.0015021,0.00082033,0.00027105,-8.9565e-
05,-0.00022685,-
0.00013331,0.00017005,0.00063727,0.0012027,0.0017862,0.0023025,0.0026715,0.0
028303,0.0027405,0.0023957,0.0018263,0.0011006,0.00031821,-0.00039675,-
0.00090901,-0.0010849,-0.00081554,-2.5556e-
05,0.0013082,0.0031565,0.0054221,0.0079581,-0.016531
//};

    int x[ORDEN(h)];

/* USER CODE END 0 */

/* External variables ---------------------------------------------------
--*/
extern ADC_HandleTypeDef hadc1;

/******************************************************************************
***/
/*            Cortex-M4 Processor Interruption and Exception Handlers
*/
/******************************************************************************
***/

/**
* @brief This function handles Non maskable interrupt.
*/
void NMI_Handler(void)
{
  /* USER CODE BEGIN NonMaskableInt_IRQn 0 */

  /* USER CODE END NonMaskableInt_IRQn 0 */
  /* USER CODE BEGIN NonMaskableInt_IRQn 1 */

  /* USER CODE END NonMaskableInt_IRQn 1 */
}

/**
* @brief This function handles Hard fault interrupt.
*/
```

```
void HardFault_Handler(void)
{
  /* USER CODE BEGIN HardFault_IRQn 0 */

  /* USER CODE END HardFault_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_HardFault_IRQn 0 */
    /* USER CODE END W1_HardFault_IRQn 0 */
  }
  /* USER CODE BEGIN HardFault_IRQn 1 */

  /* USER CODE END HardFault_IRQn 1 */
}

/**
* @brief This function handles Memory management fault.
*/
void MemManage_Handler(void)
{
  /* USER CODE BEGIN MemoryManagement_IRQn 0 */

  /* USER CODE END MemoryManagement_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_MemoryManagement_IRQn 0 */
    /* USER CODE END W1_MemoryManagement_IRQn 0 */
  }
  /* USER CODE BEGIN MemoryManagement_IRQn 1 */

  /* USER CODE END MemoryManagement_IRQn 1 */
}

/**
* @brief This function handles Prefetch fault, memory access fault.
*/
void BusFault_Handler(void)
{
  /* USER CODE BEGIN BusFault_IRQn 0 */

  /* USER CODE END BusFault_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_BusFault_IRQn 0 */
    /* USER CODE END W1_BusFault_IRQn 0 */
  }
  /* USER CODE BEGIN BusFault_IRQn 1 */

  /* USER CODE END BusFault_IRQn 1 */
}

/**
* @brief This function handles Undefined instruction or illegal state.
*/
void UsageFault_Handler(void)
{
  /* USER CODE BEGIN UsageFault_IRQn 0 */

  /* USER CODE END UsageFault_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_UsageFault_IRQn 0 */
    /* USER CODE END W1_UsageFault_IRQn 0 */
  }
  /* USER CODE BEGIN UsageFault_IRQn 1 */
```

```
  /* USER CODE END UsageFault_IRQn 1 */
}

/**
* @brief This function handles System service call via SWI instruction.
*/
void SVC_Handler(void)
{
  /* USER CODE BEGIN SVCall_IRQn 0 */

  /* USER CODE END SVCall_IRQn 0 */
  /* USER CODE BEGIN SVCall_IRQn 1 */

  /* USER CODE END SVCall_IRQn 1 */
}

/**
* @brief This function handles Debug monitor.
*/
void DebugMon_Handler(void)
{
  /* USER CODE BEGIN DebugMonitor_IRQn 0 */

  /* USER CODE END DebugMonitor_IRQn 0 */
  /* USER CODE BEGIN DebugMonitor_IRQn 1 */

  /* USER CODE END DebugMonitor_IRQn 1 */
}

/**
* @brief This function handles Pendable request for system service.
*/
void PendSV_Handler(void)
{
  /* USER CODE BEGIN PendSV_IRQn 0 */

  /* USER CODE END PendSV_IRQn 0 */
  /* USER CODE BEGIN PendSV_IRQn 1 */

  /* USER CODE END PendSV_IRQn 1 */
}

/**
* @brief This function handles System tick timer.
*/
void SysTick_Handler(void)
{
  /* USER CODE BEGIN SysTick_IRQn 0 */

  /* USER CODE END SysTick_IRQn 0 */
  HAL_IncTick();
  HAL_SYSTICK_IRQHandler();
  /* USER CODE BEGIN SysTick_IRQn 1 */

  /* USER CODE END SysTick_IRQn 1 */
}

/******************************************************************************
***/
/* STM32L4xx Peripheral Interrupt Handlers
*/
/* Add here the Interrupt Handlers for the used peripherals.
*/
```

```
/* For the available peripheral interrupt handler names,
*/
/* please refer to the startup file (startup_stm32l4xx.s).
*/
/*******************************************************************************
***/

/**
* @brief This function handles ADC1 global interrupt.
*/
void ADC1_IRQHandler(void)
{
  /* USER CODE BEGIN ADC1_IRQn 0 */

  /* USER CODE END ADC1_IRQn 0 */
  HAL_ADC_IRQHandler(&hadc1);
  /* USER CODE BEGIN ADC1_IRQn 1 */


    // bucle de filtrado del filtro FIR
    for(int i=0;i<ORDEN(h);i++){
        y = y + x[ORDEN(h)-i-1]*h[i];
    }

    //buble para desplazar las muestras anteriores del buffer
    for(int j=0;j<ORDEN(h)-1;j++){
        x[j]=x[j+1];
    }

    if (y<aux){
        aux=y;
    }
    if (aux<0){
      y=y-aux;
    }


    //envde la muestra filtrada al DAC

HAL_DAC_SetValue(&hdac1,DAC_CHANNEL_1,DAC_ALIGN_12B_R,(uint16_t)y&(0xFFF));

    //lectura de muestra convertida en el ADC
    value=HAL_ADC_GetValue(&hadc1);

    //Guardar la muestra en el buffer de entrada en la ultima posici
x[ORDEN(h)-1]=value;

    // resetear la salida para la proxima iteraci    y=0;
    /* USER CODE END ADC1_IRQn 1 */
}

/* USER CODE BEGIN 1 */

/* USER CODE END 1 */
/************************ (C) COPYRIGHT STMicroelectronics *****END OF
FILE****/
```

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR