ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA TELEMÁTICA

IMPLEMENTACIÓN DE UN PROCESADOR DIGITAL DE SEÑAL CON BITSCOPE

Autor: Javier de la Paz Garcillán

Directores: Javier Matanza Domingo, Gregorio López López

Madrid

Julio 2017

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Implementación de un Procesador Digital con BitScope

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico (2016/17) es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Javier de la Paz Garcillán Fecha: 10 / 07 / 2017

Autorizada la entrega del proyecto

LOS DIRECTORES DEL PROYECTO

Fdo.: Javier Matanza Domingo Fecha: 10 / 07 / 2017

Fdo.: Gregorio López López Fecha: 10 / 07 / 2017

Vº Bº del Coordinador de Proyectos

Fdo.: David Contreras Bárcena Fecha: ……/ ……/ ……

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Javier de la Paz Garcillán DECLARA ser el titular de los derechos de propiedad intelectual de la obra: Implementación de un Procesador Digital de Señal con el osciloscopio y analizador BitScope, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente).*

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

- El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 10 de julio de 2017

ACEPTA

Fdo Javier de la Paz Garcillán

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

Agradecimientos

Me gustaría empezar agradeciendo a las personas que más tengo que agradecer en la vida, mis padres.

A ellos va dedicado todo lo que he conseguido, y conseguiré en mi vida. Han confiado siempre en mí, me han mostrado su apoyo incondicional y nunca me han hecho echar nada en falta en mi vida.

A partir de que termine mi ciclo en la universidad, empezará una nueva etapa en mi vida en la que me tendré que valer por mí mismo y poner en práctica todo lo que me han enseñado durante estos años. Deberé demostrar lo que valgo, que ha valido la pena todo lo que han apostado por mí y devolver con creces todo lo que se ha invertido en tiempo, esfuerzo y dinero. (Por suerte, sé que, si me caigo, tengo, ya no solo mis padres, sino a una maravillosa familia detrás para ayudarme en lo que sea).

Tranquilos, tened por seguro que no os fallaré.

También quiero aprovechar para agradecer a ICAI que me haya brindado la oportunidad de estudiar y formarme como ingeniero en una universidad de semejante categoría. Agradecer a cada uno de los profesores que he tenido durante estos cuatro años, con especial énfasis en los tutores que además de enseñarme en clase, me han ayudado en lo personal fuera de ellas; y, por supuesto, a mis compañeros, a los que muchos de ellos puedo llamar ahora amigos, sin los cuales, tal vez no habría llegado hasta aquí.

Gracias a David Contreras, coordinador de proyecto, tutor y profesor mío en estos años de grado, por como su cercanía y amabilidad con la clase, su disponibilidad, su ayuda y su entrega con todos y cada uno de nosotros.

Por último, no puedo olvidar en el apartado de agradecimientos a mi director de proyecto, Javier Matanza. Darte las gracias por todo el tiempo que le has dedicado al proyecto y a mí, por muy pesado que haya sido estos meses; por los mil correos que me has tenido que contestar y nunca con tardanza; por las quinientas veces que me has tenido que ver en tu despacho; y por tu, aún con todo esto, amabilidad para ayudarme.

A todos vosotros os dedico mi proyecto; un proyecto que parecía que empezaba a nacer a finales de febrero, cuando me vi sin prácticas y sin TFG, y del que 4 meses después no puedo estar más orgulloso, y del que he disfrutado en el camino y con el resultado final.

Gracias.

IMPLEMENTACIÓN DE UN PROCESADOR DIGITAL DE SEÑAL CON BITSCOPE

Autor: de la Paz Garcillán, Javier.
Director: Matanza Domingo, Javier y López López, Gregorio
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

En el presente proyecto se ha desarrollado una aplicación capaz de hacer cometidos de osciloscopio, analizador de espectros y con funcionalidad de procesador digital de señales analógicas y digitales: esta es el filtrado según distintos criterios (coeficientes o especificaciones). Dicha implementación ha sido llevada a cabo utilizando una plataforma hardware conocida como BitScope [1]. La aplicación, que es compatible con cualquier sistema operativo basado en Windows, Linux y Mac, utiliza el lenguaje Python [4] tanto para la comunicación con el dispositivo hardware como para la representación de señales muestreadas.

Palabras clave: BitScope, Python, Procesador digital de señal, Osciloscopio.

1. Introducción

Nos encontramos ante una situación en la cual se observa que las escuelas de ingeniería trabajan el tratado de señales de una manera muy artificial. Se utilizan programas informáticos, principalmente MATLAB [5] u Octave [6], para ejercicios prácticos de procesado digital de señal. Sin embargo, este tipo de programas, aunque cumple con las expectativas en cuanto a resultados proporcionados, no hace lo propio con el realismo de dichas actividades de procesado de señal. La principal razón es que se trabajan exclusivamente con señales digitales sintéticas, y que el trabajo con señales en tiempo real es complicado de implementar.

2. Definición del proyecto

Se quiere llevar a cabo un proyecto cuyo resultado sea representar en tiempo y en frecuencia una señal analógica muestreada previamente. Se busca, por tanto, una aplicación capaz de ser ejecutada en cualquier entorno, que simule el funcionamiento de un osciloscopio y un analizador de espectros.

Además, se busca añadir a la aplicación funcionalidad de un procesador digital de señal, de modo que, además de representar la señal, se pueda trabajar con ella. Se opta por añadir, por tanto, el filtrado de señal con el método de solape y almacenamiento, como prueba de concepto de los diferentes tipos de técnicas de procesado digital de señal que se podrían llevar a cabo en la herramienta desarrollada. Dicho filtrado se podría hacer, ya sea mediante la introducción por parte del usuario de los coeficientes de dicho filtro, o con las especificaciones del filtro que se desea implementar.

De este modo, se dispondría de una aplicación, capaz de diferenciarse de cualquier otro sistema existente, por su versatilidad, y su capacidad de aunar las funcionalidades de representación y filtrado de señales reales.

Además del objetivo principal, se pretende que la herramienta desarrollada no suponga una complejidad electrónica excesiva desde el punto de vista hardware. Este ha sido uno de los principales motivos por los que se ha decidido utilizar BitScope, pues plantea un entorno totalmente funcional donde la digitalización de la señal, se encuentra totalmente resuelta.

3. Descripción del modelo/sistema/herramienta

Según todo lo anterior, la aplicación que se propone será una ventana creada con Qt Designer [3], y con la ayuda de la librería de PyQtGraph [2] que se dividirá en dos partes:

- La parte de osciloscopio, contará con un cuadro para el control del osciloscopio y con dos subdivisiones.
    - En el superior, se mostrará la representación temporal de la señal.
    - En el inferior, se mostrará la representación en frecuencia de la señal.
- La parte de filtrado, que se activará una vez pulsado el botón de filtrar, también se dividirá en dos partes:
    - La izquierda, donde se encontrará el cuadro de texto en el que se podrán introducir los coeficientes del filtro que se desee aplicar.
    - La derecha, donde se podrán introducir: frecuencia de corte, el ancho deseado en el paso de transición y la atenuación deseada en la banda de stop en dB.

Para que el correcto funcionamiento del sistema se debe:

1. Conectar una señal al CHA del BitScope.
2. Conectar el BitScope mediante un puerto USB a un ordenador o Raspberry-pi que disponga de un intérprete de Python y de la librería que proporciona BitScope (bitlib)
3. Ejecutar la aplicación.

BitScope muestrea la señal que le llega con la frecuencia de muestreo que se le indique. Dichas muestras son transferidas al ordenador/raspberry que esté ejecutando la aplicación y son tratadas de modo que se pueda recomponer la señal y trabajar con ella.

4. Resultados

Para hablar de los resultados obtenidos, nos apoyaremos en la Ilustración 1:

- Se representa la señal introducida en el BitScope. Se muestra de color verde la señal original, tanto en la representación temporal como en el espectro de frecuencia.
- Se puede controlar el nivel del trigger, que nos permite capturar las señales y representarlas de manera estática cuando se cumple la condición. Además, pulsando el botón Not Enabled del trigger, se pueden congelar las gráficas. De este modo se puede ampliar, reducir y mover la señal a nuestro gusto, y así ser

analizada sin que la señal esté siendo continuamente refrescada y ajustada a su rango. Para volver a este último caso, basta con volver a pulsar el botón.

- Se puede filtrar la señal ya sea a través de los coeficientes del filtro que se desea aplicar, o introduciendo frecuencia de corte, ancho de transición y atenuación en banda de stop. Para ver cómo sería el diagrama de Bode del filtro diseñado, se puede pulsar el botón Plot en cualquiera de las dos opciones. Una vez pulsado el botón Filter, aparecerá, si el usuario así lo desea, la señal filtrada, en rojo. Todos los datos introducidos en las cajas de texto, serán validados y en caso de no ser válidos, se mostrará un mensaje de error descriptivo al usuario.
- Se podrá elegir representar la señal: en tiempo, en frecuencia, la señal original, la señal filtrada, o cualquier combinación deseada de estas cuatro.

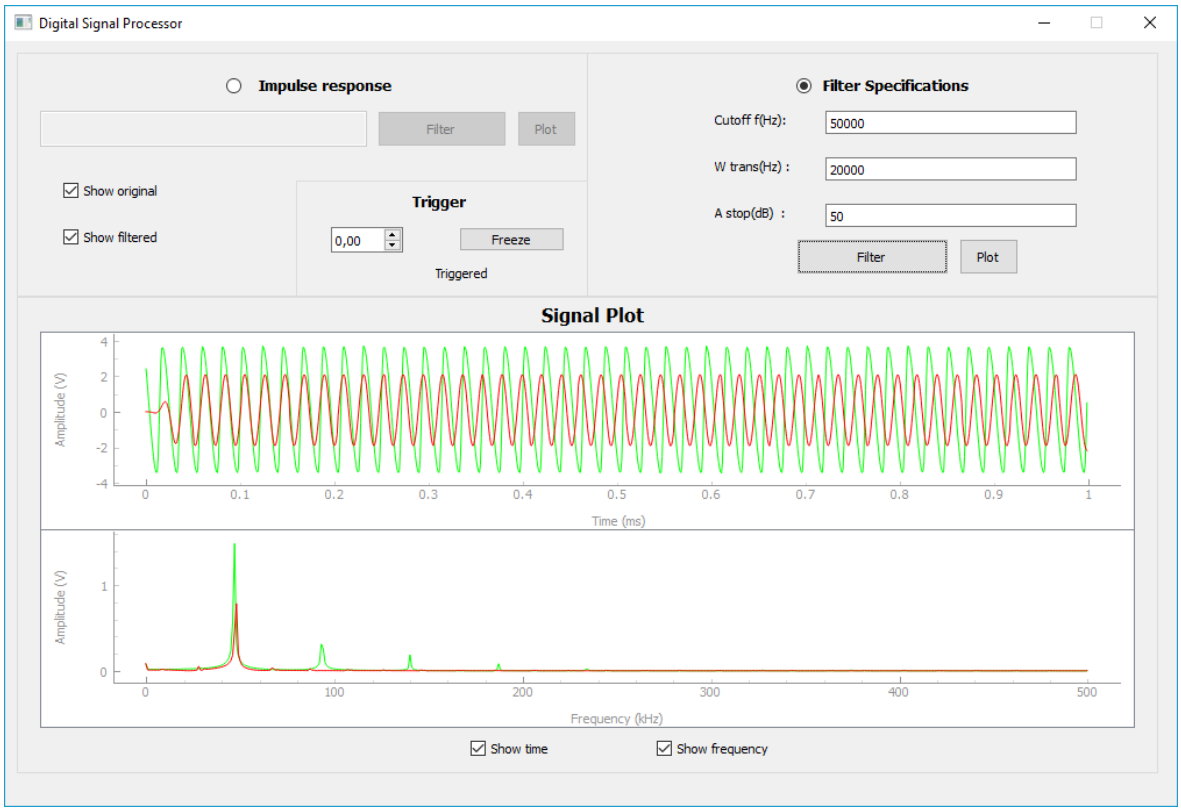

Ilustración 1. Ventana de aplicación en funcionamiento

5. Conclusiones

Se logra el objetivo de juntar la representación y el procesado digital de señales analógicas. Gracias a BitScope, se ha sido capaz de ver funcionar un sistema en tiempo real. El muestreo se hace en tiempo real y se va representando según se procesan las muestras. De este modo se ha mostrado una pequeña parte del gran potencial de un muestreador sencillo, barato y programable, con una API extensa, llena de posibilidades y que ayuda al propósito por el que nació este proyecto de dar mayor realismo a las prácticas en tratado de señal en las escuelas de ingeniería.

6. Referencias

[1] Página Oficial BitScope - http://www.bitscope.org/

[2] Página Oficial PyQtGraph - http://www.pyqtgraph.org/

[3] Página Oficial Qt Designer - https://www.qt.io/ide/

[4] Página Oficial Python 2.7 - https://www.python.org/download/releases/2.7/

[5] Página Oficial MATLAB - https://es.mathworks.com/products/matlab.html

[6] Página Oficial GNU Octave - https://www.gnu.org/software/octave/

DIGITAL SIGNAL PROCESSOR IMPLEMENTATION WITH BITSCOPE

Author: de la Paz Garcillán, Javier.
Supervisor: Matanza Domingo, Javier and López López, Gregorio

ABSTRACT

This project has developed an application capable of behaving as an oscilloscope, spectrum analyser and digital processor of analogue and digital signals: filtering according to different criteria (coefficients or specifications). This implementation has been carried out using a hardware platform known as BitScope [1]. The application, which is compatible with any operating system based on Windows, Linux and Mac, uses Python [4] language both for communication with the hardware device and for the representation of sampled signals.

Keywords: BitScope, Python, Digital signal processor, Oscilloscope.

1. Introduction

We are faced with a situation in which it is observed that the engineering schools work the signalling in a very artificial way. Computer programs, mainly MATLAB [5] or Octave [6], are used for practical exercises in digital signal processing. However, this kind of programs, while meeting expectations in terms of results provided, does not do the same with the realism of such signal processing activities. The main reason is that they work exclusively with synthetic digital signals, and that working with real-time signals is complicated to implement.

2. Project definition

We want to carry out a project which result is to represent in time and in frequency a previously sampled analogue signal. It is therefore sought an application capable of being executed in any computer environment, which simulates the function of an oscilloscope and a spectrum analyser.

In addition, we are looking for adding to the application functionality of a digital signal processor, so that, in addition to representing the signal, you can work with it. It has been opted to add, therefore, the signal filtering with overlap-add method, as proof of concept of the different types of techniques of digital signal processing that could be carried out in the developed tool. The filtering could be done either by introducing the coefficients or by the specifications of the filter to be implemented by the user.

In this way, an application would be available, able to differentiate itself from any other existing system, for its versatility, and its ability to combine the functionalities of representation and filtering of real signals.

In addition to the main objective, it is intended that the tool developed does not imply an excessive electronic complexity from the hardware point of view. This has been one of the main reasons why it has been decided to use BitScope, since it proposes a fully functional environment where the digitalization of the signal is totally resolved.

3. System description

According to all the above, the application that is proposed will be a window, created with Qt Designer [3] and with PyQtGraph [2] library, that will be divided into two parts:

- The oscilloscope part, will have a frame for the oscilloscope control and two subdivisions:
    - In the upper one, the temporal representation of the signal will be displayed.
    - In the lower one, the frequency representation of the signal will be displayed.
- The filtering part, which will be activated once the filter button is pressed, will also be divided into two parts:
    - On the left, where you will find the text box in which you can enter the coefficients of the filter to be applied.
    - On the right, where you can enter: cutting frequency, the desired width in the transition step and the desired attenuation in the stopband in dBs.

For the correct functioning of the system we should:

1. Connect a signal to the CHA of the BitScope.
2. Connect the BitScope via a USB port to a computer or Raspberry-pi which has a Python interpreter and the library that provides BitScope (bitlib)
3. Run the application.

BitScope, samples the entering signal with the sampling frequency instructed. Such samples are transferred to the computer/raspberry that is running the application and are treated so that the signal can be recomposed and work with it.

4. Results

To talk about the results obtained, we will reference to Figure 1:

- The signal entered in the BitScope is displayed. The original signal is shown green in both the temporal and frequency spectrum.
- It is possible to control the level of the trigger, which allows us to capture the signals and to represent them in a static way when the condition is fulfilled. In addition, by pressing the button *Not Enabled* of the trigger, the graphs can be frozen. In this way, the signal can be enlarged, reduced and moved to our liking, and thus be analysed without the signal being continuously refreshed and adjusted to its range. To return to the latter case, simply press the button again.

- You can filter the signal either through the coefficients of the filter you want to apply, or by inputting cut-off frequency, transition width and stopband attenuation. To see what the Bode diagram of the designed filter would look like, you can press the *Plot* button in either of the two options. Once the Filter button is pressed, the filtered signal will appear in red if the user so wishes. All data entered in the text boxes will be validated and if not valid, a descriptive error message will be displayed to the user.

- You can choose to represent the signal: in time, in frequency, the original signal, the filtered signal, or any desired combination of these four.

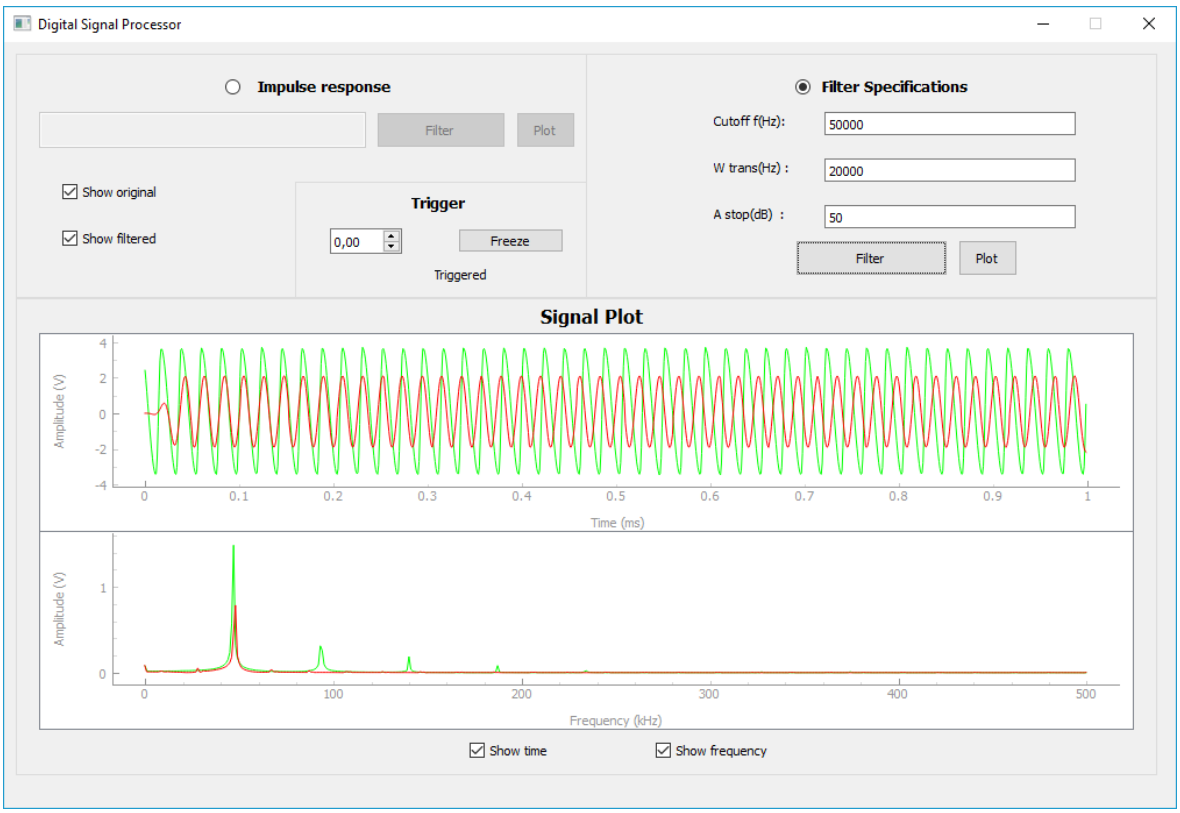

Figure 1. Application window working

5. Conclusions

The objective of joining digital representation and processing of analogue signals together is achieved. Thanks to BitScope, it has been able to see a system work in real time. Sampling is done in real time and the samples are plot as they are processed. In this way, it has been shown a small part of the great potential of a simple, cheap and programmable sampler with an extensive API, full of possibilities and which helps the purpose which this project was born: giving more realism to the signal practices in engineering universities.

6. References

[1] BitScope Official Website - http://www.bitscope.org/

[2] PyQtGraph Official Website - http://www.pyqtgraph.org/

[3] Qt Designer Official Website - https://www.qt.io/ide/

[4] Python 2.7 Official Website - https://www.python.org/download/releases/2.7/

[5] MATLAB Official Website - https://es.mathworks.com/products/matlab.html

[6] GNU Octave Official Website - https://www.gnu.org/software/octave/

Índice de la memoria

Índice de Ilustraciones

Índice de tablas

Índice de Acrónimos

A/D – Analógico / Digital

API - Application Programming Interface

DFT - Discrete Fourier Transform

HPF – High Pass Filter

IDE - Integrated Development Environment

I/O – Input / Output

FFT - Fast Fourier Transform

FIR – Finite Impulse Response

GUI - Graphical User Interface

LPF – Low Pass Filter

OEM - Original Equipment Manufacturer

PBF – Pass-Band Filter

PDS – Procesado Digital de Señal

Q/A – Quality Assurance

TIC - Tecnologías de la Información y la Comunicación

Capítulo 1. INTRODUCCIÓN

Hoy en día, el análisis de señales y el procesado digital de las mismas, se considera un tema imprescindible en el aprendizaje de cualquier carrera de tecnologías de la información y la comunicación (TIC). La digitalización global en la que nos encontramos actualmente, confiere una gran importancia a este tema y una gran demanda de personas con conocimientos sobre el mismo. La digitalización de señales (o transcripción de señales analógicas en señales digitales), ha facilitado su procesamiento, además de inmunizarla contra ruido e interferencias.

Principalmente en ingenierías electrónicas y de telecomunicaciones, el estudio de señales, tanto analógicas como digitales, es de obligado cumplimiento. Entender los principios básicos de la señal, la conversión A/D, las ventajas e inconvenientes de esta conversión y las aplicaciones que se le pueden dar, se antoja algo fundamental en nuestra sociedad, en pleno siglo XXI, y ante la evolución del mundo digital y la cada vez menor presencia de las tecnologías analógicas.

Vayámonos, por tanto, al mundo académico, a los estudiantes de hoy en día y a su formación para el mundo de ahora. En las escuelas de ingeniería, se les confiere mucha importancia a las asignaturas de señales. En torno a 24-30 créditos son destinados en las carreras de Telecomunicaciones a enseñar acerca de este tema.

Las ingenierías, como carreras prácticas que se presuponen, suelen ser enfocadas en las universidades con dos perspectivas: la teórica y la práctica. Es por esto, que la gran mayoría de las asignaturas se dividen en clases teóricas y laboratorios. El objetivo de los laboratorios

no es otra que enseñar al alumno como funciona lo aprendido en clase en la práctica; demostrar que la teoría se puede llevar a la práctica; que lo que se explica, tiene aplicación real y que los estudiantes entiendan, que al final, lo importante en las carreras de ingeniería es saber aplicar a la vida real, lo aprendido día a día.

Por todo esto, en los laboratorios se busca el mayor realismo posible, e intentar reproducir lo más fielmente que se pueda, un caso real, o, en caso de no ser posible, simularlo. Y esto último, simularlo, es lo que más se hace en las prácticas de procesado de señal.

En las escuelas de ingeniería se trabaja el tratado de señales, de manera muy artificial. Se utilizan programas informáticos, principalmente MATLAB [13] y GNU Octave [14], para ejercicios prácticos de procesado digital de señal; sin embargo, dichos programas, aunque cumplen con las expectativas en cuanto a resultados proporcionados, no hacen lo propio con el realismo de dichas actividades de procesado de señal.

MATLAB no trabaja con señales analógicas (reales) de manera sencilla, ni trabaja en tiempo real.

Por otro lado, las escuelas disponen de instrumentos capaces de trabajar con señales analógicas y dar resultados en tiempo real, como osciloscopios y analizadores de espectro, pero, además de su alto precio, su principal laguna es que permiten el estudio de la señal, pero no trabajar con ella, debido a que carecen de características de procesado digital de señal.

Se debe buscar una solución que aúne todas las características previamente citadas y cumpla el objetivo de dar mayor realismo a las prácticas de procesado digital de señal, de forma sencilla y económica, de esta búsqueda surge BitScope, que supondrá ser la base de este proyecto.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

2.1 PROCESAMIENTO DIGITAL DE SEÑAL

Manipulación matemática de una señal de información para modificarla o mejorarla en algún sentido [1]. Se hace uso del procesamiento digital, para realizar una amplia variedad de operaciones de procesamiento de señal.

Se caracteriza por la representación en el dominio discreto (de tiempo o frecuencia) de señales por medio de una secuencia de números o símbolos y el procesado de esas señales.

Para dicho fin, se suelen utilizar sistemas basados en un procesador o microprocesador con instrucciones, hardware y software optimizados para aplicaciones que requieran operaciones numéricas a alta velocidad.

El propósito de procesar una señal puede ser disminuir el nivel de ruido o mejorar la presencia de determinados matices [2]. Es por esto que algunas de las aplicaciones del procesamiento digital de señales son procesamientos de: audio, voz, imágenes, vídeo o datos digitales

2.1.1 MUESTREO DIGITAL

Fase del proceso de digitalización de una señal [5]. Éste consiste en tomar muestras de una señal analógica a una frecuencia de muestreo constante (como se ve en la Ilustración 1). Se reduce una señal continua en el tiempo a señal discreta.

- Una muestra es un valor o serie de valores en un punto en tiempo y/o espacio.
- Un muestreador es un subsistema que extrae muestras de una señal continua. F_s

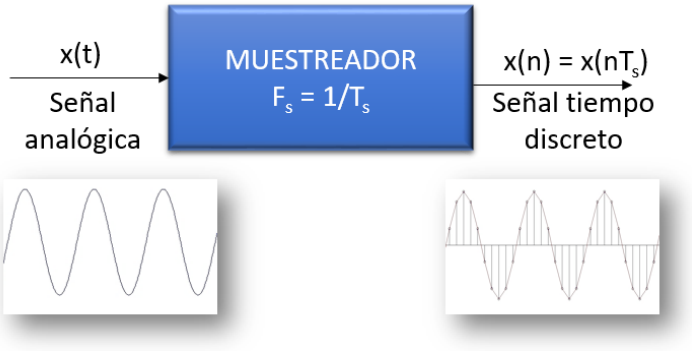

Ilustración 1. Muestreo digital de una señal

Una vez realizado, se pasa a la cuantificación de la misma y posteriormente a la codificación, como se muestra en la Ilustración 2. De este modo se completa una conversión de una señal analógica a digital.

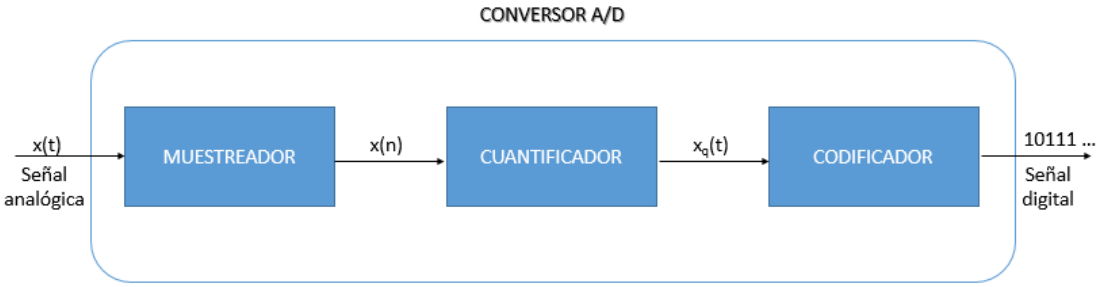

Ilustración 2. Diagrama de bloques de Conversor A/D

Es útil en la digitalización de señales [1], por consiguiente, en las telecomunicaciones y en la codificación del sonido en formato digital.

Tiene su base en el teorema de muestreo de Nyquist-Shannon: una señal analógica puede ser reconstruida, sin error, de muestras tomadas en iguales intervalos de tiempo. La razón de muestreo debe ser igual, o mayor, al doble de su ancho de banda de la señal analógica [15].

El muestreo práctico, difiere del muestreo teórico en tres puntos [5]:

- La onda con la que se muestrea, está constituida por trenes de impulsos de duración no nula.

- Los filtros prácticos de reconstrucción no pueden ser ideales.
- Los mensajes a los que se aplica el teorema no están estrictamente limitados en banda, ya que se trata de señales limitadas en tiempo.

2.1.2 **FFT**

Un algoritmo de transformada rápida de Fourier (FFT), calcula la transformada discreta de Fourier (DFT) de una secuencia, o su inversa (IFFT). [8]

El análisis de Fourier convierte una señal de su dominio original (a menudo tiempo o espacio) en una representación en el dominio de la frecuencia y viceversa (por ejemplo, permite pasar de la parte superior de la Ilustración 3, a la inferior y viceversa). Una FFT calcula rápidamente tales transformaciones por factorización de la matriz DFT en un producto de escasos factores.

Las transformadas rápidas de Fourier son ampliamente utilizadas para muchas aplicaciones en ingeniería, ciencia y matemáticas. Algunos ejemplos de ello son: tratamiento de imagen y audio, reducción de ruido en señales, análisis en frecuencia de señales discretas, análisis de vibraciones, de materiales y estadística, etc.

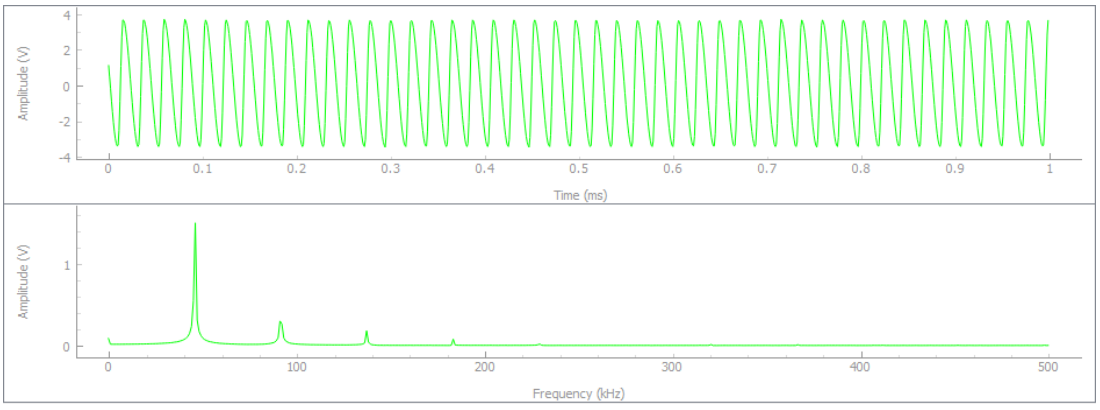

Ilustración 3. Señal en tiempo y frecuencia

2.2 PYTHON

Es un lenguaje de programación de alto nivel, ampliamente utilizado hoy en día para propósitos generales de programación. [12]

Es un lenguaje interpretado, usa tipado dinámico, es multiplataforma y con una sintaxis que favorece el código legible.

Además, se trata de un lenguaje de programación multiparadigma: soporta orientación a objetos, programación imperativa y funcional.

2.2.1 PYQTGPRAPH

En la web oficial de PyQtGraph se dice de esta librería que es un gráfico puramente Python y una GUI construida sobre PyQt4/PySide y NumPy. Está diseñado para usarse en aplicaciones matemáticas, científicas o de ingeniería. A pesar de ser escrita totalmente en Python, la biblioteca es muy rápida debido a su gran influencia de NumPy para el cálculo numérico y el framework GraphicsView de Qt para una visualización rápida. PyQtGraph se distribuye bajo la licencia MIT open-source. [7]

Se puede correr en cualquier sistema operativo (Linux, Windows, and OSX) que tenga Python 2.7 and 3+, PyQt 4.8+ o PySide y NumPy.

2.3 BITSCOPE

BitScope Mixed Signal Systems, son descritos en su página web oficial como: osciloscopios programables basados en PC, analizadores lógicos, analizadores de espectro, generadores de formas de onda y sistemas de adquisición de datos para Windows, Mac OS X, Linux y

Raspberry Pi. Ofrecen soluciones integrales en pruebas, medidas, monitoreo y control para ingenieros en la industria, educación, I+D y servicio. [6]

Económicos y todos ellos están basados en la BitScope Virtual Machine. La BitScope Virtual Machine permite actualizar y personalizar dinámicamente el software.

Son compatibles con el software más actualizado.

Son capaces de capturar señales digitales y analógicas simultáneamente. Tienen un ancho de banda de 100 MHz y hasta 40 MS/s de captura digital. Incorporan generadores de onda y reloj, decodificadores de protocolo, I/O digitales y son capaces de conducir circuitos externos de baja potencia.

Los BitScopes son completamente programables por el usuario para poder ser utilizados en aplicaciones altamente personalizadas o incluso integrados en productos de terceros y sistemas de software en OEM. Pueden programarse directamente a nivel de máquina virtual utilizando guías de usuario de publicación o mediante la BitScope Library, que pone a disposición todas las funciones de adquisición y generación de datos de señal mixta de BitScope mediante una API de llamada de función fácil de usar

Algunas de sus aplicaciones son la creación de prototipos, pruebas y depuración de todo tipo de sistemas de señales mixtas del mundo real, desde circuitos analógicos, lógica digital e informática integrada hasta sistemas de comunicaciones, sensores y servos en robótica o sistemas de control y control de procesos industriales.

2.4 SOFTWARE

2.4.1 QT DESIGNER

Extraído de la información oficial de Qt Designer, se dice de ella que es la herramienta de Qt para diseñar y construir interfaces gráficas de usuario (GUI) a partir de componentes de

Qt. Con esta herramienta, se puede componer y personalizar los widgets o diálogos de una manera que se pueda ver lo que se obtiene, y probarlo utilizando diferentes estilos y resoluciones. [8]

Los widgets y formularios creados con Qt Designer se integran con el código programado, utilizando el mecanismo de señales y ranuras de Qt, que le permite asignar fácilmente el comportamiento a elementos gráficos. Todas las propiedades establecidas en Qt Designer se pueden cambiar dinámicamente dentro del código.

Funciones como la promoción de widgets y complementos personalizados permiten utilizar componentes propios con Qt Designer.

Qt Designer devuelve un archivo .ui, que debe ser convertido en .py con la ayuda del fichero pyuic.py.

2.5 OTRAS TECNOLOGÍAS

2.5.1 THREADS

Un hilo de ejecución es la secuencia más pequeña de instrucciones programadas que puede ser administrada independientemente por un planificador, que es típicamente una parte del sistema operativo. [6]

Pueden existir múltiples hilos dentro de un proceso, ejecutándose simultáneamente y compartiendo recursos como la memoria. En particular, los hilos de un proceso comparten su código ejecutable y los valores de sus variables en un momento dado.

Un programa con varios hilos permite que, mientras se está ejecutando la parte visible al usuario y con la que se puede interactuar, en segundo plano, se estén ejecutando otros procesos en paralelo.

2.5.2 TIMERS

Un *timer* es un tipo especializado de reloj para medir intervalos de tiempo.

Software timers: estos tipos de *timers* no son dispositivos ni partes de dispositivos. Sólo existen en líneas de código. Se basan en la precisión de un oscilador de reloj normalmente incorporado en un dispositivo de hardware que ejecuta el software. [7]

Permiten que se ejecute una acción cada vez que transcurre el tiempo establecido por el programador. Se suele utilizar de manera cíclica, para que se mantenga realizando una acción cada intervalo de tiempo determinado.

Capítulo 3. ESTADO DE LA CUESTIÓN

Este proyecto contribuirá en gran medida, en un futuro muy cercano, a que nuevos estudiantes, tengan la oportunidad de trabajar con señales de una manera distinta a la que tuvieron promociones anteriores. Se busca el avance, el desarrollo y la evolución, de manera que según avancen los años, se pueda dotar a los alumnos de mejores medios para el aprendizaje.

Hasta ahora, en la Universidad Pontificia Comillas ICAI, y en la gran mayoría de las escuelas de ingeniería a las que se ha podido tener acceso a su información, trabajan de dos maneras distintas el tratado de señal:

- Parte de análisis: la parte del estudio de la señal. Analizar, describir y sacar las características de una señal en un circuito. Se construyen circuitos con objetivos concretos:
    - El alumno debe comprobar que la señal que sale de cada etapa es correcta y lógica.
    - El alumno debe construir un circuito para que, dada una señal de entrada, en la salida haya una señal de unas características dadas.
    - El alumno debe analizar la señal a la salida de un circuito ya dado y entender que ha sucedido
    - Etc.

  Para estas situaciones, se utilizan equipos de medición y visualización electrónica como por ejemplo osciloscopios y analizadores de espectro.

    - El osciloscopio se utiliza para la representación gráfica de señales eléctricas que pueden variar en el tiempo. Muestra los valores de las señales eléctricas en forma de coordenadas en una pantalla, en la que el eje X representa tiempos y el eje Y las tensiones. La forma de onda observada puede analizarse

para propiedades tales como amplitud, frecuencia, tiempo de subida, intervalo de tiempo, distorsión y otros. El osciloscopio se puede ajustar para que las señales repetitivas se puedan observar como una forma continua en la pantalla. [3]

- El analizador de espectro permite visualizar en una pantalla los componentes espectrales en un espectro de frecuencias de las señales de entrada, pudiendo ser éstas de cualquier tipo: eléctricas, acústicas u ópticas.

  En el eje de ordenadas se representa, en una escala logarítmica, el nivel en dBm del contenido espectral de la señal. En el eje de abscisas, se representa la frecuencia, en una escala que es función de la separación temporal y el número de muestras capturadas.

  El equipo permite medir valores de potencia o tensión de señal eléctrica. Su uso principal es la medición de potencia del espectro de señales conocidas y desconocidas.

  Mediante el análisis de los espectros de las señales eléctricas se pueden observar frecuencias dominantes, potencia, distorsión, armónicos, ancho de banda y otros componentes espectrales de una señal que no son fácilmente detectables en las formas de onda del dominio del tiempo. [4]

Ambos equipos, tienen una muy amplia y completa funcionalidad en su campo, el análisis; sin embargo, se observan en ambos casos las mismas desventajas para el fin que busca este proyecto. En primer lugar, los precios de estos equipos son muy elevados, por lo que, aunque no es gran cantidad de dinero para una universidad y suelen encontrarse en los laboratorios de cualquier escuela de ingeniería, si es un elevado precio si el que quiere comprarlo es un estudiante para tener en casa. En segundo lugar, carece de funcionalidad de procesado digital de la señal; esto significa que se puede observar todas las características de una señal, pero nunca modificarlas o trabajar con ellas.

- Parte de procesado digital de señal: la parte de trabajo con la señal. Se pueden hacer multitud de prácticas de procesado digital como filtrado, cuantificación, cambios de frecuencia, etc. Para estas situaciones, se utilizan programas informáticos como son: MATLAB (en ICAI y predominante en las universidades) y también Octave (en menor medida).
    - MATLAB: (matrix laboratory) es una herramienta de software matemático de cómputo numérico multi-paradigma que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (M). [13]

      MATLAB permite manipulaciones de matriz, trazado de funciones y datos, implementación de algoritmos, creación de interfaces de usuario e interfaz con programas escritos en otros lenguajes.

      Está disponible para las plataformas Unix, Windows, Mac OS X y Linux.

      Es un software ampliamente utilizado en universidades y en centros de investigación y desarrollo. En los últimos años ha aumentado el número de prestaciones, como, por ejemplo, se ha añadido la posibilidad de programar directamente procesadores digitales de señal.

      Algunas de las utilidades de MATLAB son:

        - Computación Paralela
        - Matemáticas, Estadísticas y Optimización
        - Sistemas de control
        - **Procesamiento de Señales y Comunicaciones**
        - Procesamiento de imágenes y visión por ordenador
        - Prueba y medición
        - Finanzas Computacionales
        - Biología computacional
        - Generación y Verificación de Código
        - Implementación de aplicaciones

- Conectividad y generación de informes de bases de datos
- Generador de Reportes MATLAB

Nos centraremos en su amplia funcionalidad en procesamiento de señales y comunicaciones. MATLAB dispone de Signal Processing Toolbox, que como MathWorks indica en la descripción oficial de dicho paquete: "proporciona funciones y aplicaciones para generar, medir, transformar, filtrar y visualizar señales. La toolbox incluye algoritmos para remuestrear, suavizar y sincronizar señales, diseñar y analizar filtros, estimar espectros de potencia y medir picos, ancho de banda y distorsión. La toolbox también incluye algoritmos de modelado predictivo paramétrico y lineal. Puede usar Signal Processing Toolbox para analizar y comparar señales en los dominios de tiempo, frecuencia y tiempo-frecuencia, identificar patrones y tendencias, extraer características y desarrollar y validar algoritmos personalizados que le permitan conocer sus datos en profundidad."

Sin embargo, MATLB trabaja con señales sintéticas, utiliza archivos de audio digitales comprimidos (.mp3, .wav). Las señales analógicas son muy complejas de implementar.

- GNU Octave - es un software libre con un lenguaje de programación de alto nivel, destinado principalmente a cálculos numéricos. Octave ayuda a resolver problemas lineales y no lineales numéricamente y para realizar otros experimentos numéricos utilizando un lenguaje que es en su mayoría compatible con Matlab. También puede utilizarse como un lenguaje orientado por lotes. Dado que forma parte del Proyecto GNU, es un software libre bajo los términos de la Licencia Pública General GNU. [14]

  Octave es una de las alternativas libres principales a Matlab.

  Octave dispone de paquetes destinados a imágenes, mapping, optimización, señales y estadísticas.

El paquete de señales de Octave dispone de herramientas de procesamiento de señal, incluyendo filtrado, ventana y funciones de visualización. Sin embargo, al igual que MATLAB, carece de realismo funcional.

El principal problema que encontramos en ambos programas destinados al ámbito académico es el poco realismo de su trabajo con señales. Se trabaja con señales sintéticas y los resultados no se producen en tiempo real.

Al final, esto hace que se pierda ese realismo tan importante que se busca en las sesiones prácticas de laboratorio en las universidades.

3.1 PROYECTOS RELACIONADOS

En una búsqueda por Internet, se han encontrado algunos proyectos con alguna similitud al que se presenta en este documento:

- *PicBerry Oscilliscope and Function Generator with a PIC32 and Raspberry Pi*: este proyecto consiste en un osciloscopio digital y un generador de funciones que son capaces de trazar y producir (al mismo tiempo) señales eléctricas generadas (voltaje) que cambian con el tiempo a lo largo de una escala calibrada. El dispositivo proporciona la funcionalidad básica de osciloscopio digital de trazar el voltaje variable en un monitor de escritorio y la capacidad básica del generador de funciones para producir ondas periódicas de amplitudes y frecuencias variables. [17]

  El usuario del dispositivo es capaz de guardar gráficos generados en una tarjeta SD y la medición de pico a pico de voltaje de un gráfico. Algunas de las funciones de generación de funciones más avanzadas incluyen: frecuencia, amplitud y el tipo de onda (seno, cuadrado y diente de sierra). Todas estas funciones de osciloscopio y Generación de Funciones están disponibles a través de una GUI intuitiva e interactiva.

Es, en conclusión, un proyecto similar al propuesto. La idea de un osciloscopio digital está presente, aunque no se utiliza BitScope como muestreador. Sin embargo, difiere en este proyecto, en que no busca funcionalidad de procesador digital de señal, como el filtrado, si no que se centra en la generación de ondas.

Otros proyectos más similares que podemos encontrar, son utilizando PiScope. Sin embargo, todavía está en desarrollo temprano. También utiliza una pantalla LCD para mostrar (en lugar de un monitor) las señales.

- *BitScope PiLab* – este proyecto, es descrito por los desarrolladores como un laboratorio de ciencia de código abierto para Raspberry Pi y BitScope.

  PiLab es una plataforma de programación de código abierto fácil de usar escrita en Python estándar para usar con Raspberry Pi. Está diseñado para permitir la creación de aplicaciones personalizadas de prueba, medición y adquisición de datos en campos educativos, de ingeniería y científicos. [19]

  PiLab utiliza una sencilla interfaz gráfica de usuario PNG que no requiere bibliotecas de widgets complicadas.

  PiLab construye un sistema de prueba y verificación altamente optimizado para probar el funcionamiento de BitScope Micro en producción.

  Se incluyen en PiLab las aplicaciones Q/A: test y verificación.

  Es un proyecto que, al igual que el que se expone, utiliza BitScope para el tratado de señal, pero con objetivos distintos.

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

La principal motivación de este proyecto es cambiar la forma que hay actualmente en las universidades de ingeniería, de trabajar en casos prácticos el tratado de señal.

Se busca una alternativa a los programas informáticos actuales (MATLAB y GNU Octave), que son la forma predominante en la actualidad para estas prácticas.

Se busca el realismo; hacer de las prácticas en el laboratorio una experiencia más real. Se busca poder trabajar con señales reales analógicas, en vez de digitales sintéticas; se busca trabajar y obtener resultados en tiempo real.

Se quería conseguir una aplicación capaz de aunar los campos del análisis y el procesado digital de señal. Un sistema que permita, no solo la representación de señales en tiempo y en frecuencia, si no que permita trabajar con estas señales, dotar al sistema de aplicaciones de procesado digital de señal como es el filtrado, y hacer, de esta manera, la experiencia práctica de tratado de señales, una experiencia más real y completa.

Esta aplicación se quiere diferenciar del resto por juntar las funcionalidades de distintos sistemas, (osciloscopio, analizador de espectros y procesador digital de señal) en uno solo.

BitScope es un hardware especial diseñado para ser un osciloscopio programable basado en PC, un analizador de lógica, un generador de onda y reloj y un analizador de espectro

Además, se ha elegido BitScope, ya que se pretende que la herramienta desarrollada no suponga una complejidad electrónica excesiva desde el punto de vista hardware. BitScope, plantea un entorno totalmente funcional donde la digitalización de la señal, se encuentra totalmente resuelta. Así mismo, su coste en comparación con los sistemas a los que pretender reemplazar es muy bajo, y su tamaño de bolsillo, lo hace fácilmente portable. Estas dos

ventajas, añadidas a todas las descritas anteriormente, hacen de esta aplicación, una aplicación totalmente única.

4.2 OBJETIVOS

El objetivo del proyecto es: conseguir una aplicación programada en Python, que se pueda ejecutar en cualquier sistema operativo y que sea capaz de:

- Representar señales digitales y analógicas en:
  - Tiempo
  - Frecuencia
- Controlar el trigger de la gráfica de la señal representada en tiempo e información de si nos encontramos en estado de: *Triggered, Frozen o Armed*
- Congelar, mover y ampliar y reducir, las representaciones de las señales
- Sacar los coeficientes de un filtro a partir de sus especificaciones
- Ver la representación logarítmica en dB de los filtros creados
- Filtrar la señal de entrada dados:
  - Los coeficientes del filtro
  - La frecuencia de corte, el ancho de banda de paso y la atenuación en la banda de stop.
- Representación de la señal filtrada
- Poder decidir representar la señal: en tiempo, en frecuencia, la señal original, la señal filtrada, o cualquier combinación deseada de estas cuatro.
- Validación de los datos de entrada y notificación descriptiva en caso de error

Además, para el muestreo digital de la señal entrante, se busca conseguir un hardware:

- Sencillo (que la herramienta desarrollada no suponga una complejidad electrónica excesiva)
- Barato (en comparación con instrumentos similares)

- Programable (con API propia para ser configurado y personalizado mediante la programación)

4.3 METODOLOGÍA

En este apartado, tan solo se mencionará con que tecnología se abordará cada punto de los objetivos.

- Digitalización de las señales entrantes → BitScope
- Interfaz gráfico (ventana) → Qt Designer
    - Introducción de datos por ventana → QLineEdit
    - Selección de señal a representar → QCheckBox
    - Filtrado y representación del filtro → QPushButton
    - Congelado de señal → QPushButton
    - Representación de señales → QGraphicsView
    - Elegir tipo de filtrado → QRadioButton
    - Control de valor del trigger → QDoubleSpinBox
    - Notificación del estado del trigger → QLabel
- Pasar de código XML del Qt Designer a Python → pyuic.py
- Comunicación con el BitScope → Python y Bitlib (librería en Python de BitScope) [9]
- Funcionalidad de los elementos de la ventana → Python
- Obtención de muestras del Bitscope en segundo plano a la vez que funciona el resto de la aplicación → Threads
- Refresco de las gráficas → Timer

Se entrará en más detalle de cómo funciona cada cosa, en el apartado 5 del documento, que es donde corresponde la descripción en profundidad del diseño.

4.4 PLANIFICACIÓN Y ESTIMACIÓN ECONÓMICA

4.4.1 PLANIFICACIÓN

A la hora de llevar a cabo el proyecto, se ha intentado seguir un calendario de hitos y objetivos (ilustrado en la Ilustración 4 con un Diagrama de Gantt), que, aunque no se ha cumplido estrictamente, ha sido muy orientativo y ha servido de guía para una buena planificación y organización con el mismo.

Este proyecto comenzó a finales del mes de febrero del año 2017 y ha finalizado con su documentación y presentación ante tribunal el 10 de julio de 2017.

Se desarrollará un poco más en profundidad la planificación llevada a cabo, las actividades y su distribución:

- Inicio del proyecto – Planificación inicial: se solicita llevar a cabo este proyecto (Implementación de un Procesador Digital con BitScope), y se acepta por parte del Director del Proyecto. Se exponen los objetivos y se marcan las primeras fechas.
- Conocimiento e instalación del entorno de trabajo: se encarga el hardware necesario para el sistema del proyecto (el que viene listado en el apartado 4.4.2 del documento), se descarga e instala el software requerido (Bitlib, PyQtGraph, Qt Designer, Python) y en paralelo, se hace la parte de documentarse acerca de la tecnología que se va a utilizar, aprendiendo así que posibilidades nos ofrece y cuál es su funcionamiento, para que, una vez instalado todo y teniendo todo el hardware disponible, no perder tiempo y poder iniciar directamente el trabajo con lo que ya se conoce.
- Diseño: se da forma visual a la idea. Se propone un boceto de cómo podría ser la ventana de la aplicación. Éste es llevado a cabo de manera provisional con Qt Designer; será un primer diseño, que cumpla los requisitos que se han establecido hasta el momento, pero dejando lugar a futuras modificaciones y añadidos.
- Implementación: se incorpora toda la funcionalidad a la interfaz gráfica con Python y PyQtGraph. Se crean las distintas clases, se hace la separación en código de la parte de interfaz y la parte funcional. Una vez finalizado, se observa el resultado y se

proponen optimizaciones (mejorar la velocidad de la aplicación utilizando Threads) y nuevas ideas a implementar (control del trigger, formas alternativas de filtrado, representación de filtros) y se llevan a cabo. Por último, se hace la validación de los datos entrantes y se toman medidas para avisar al usuario de que está introduciendo datos no válidos, mediante notificaciones descriptivas de error.

- Documentación – una vez ya se tiene todo el código terminado se pasa a la redacción de la memoria. Se documenta todo lo realizado en el proyecto siguiendo las pautas del guion dado. Una vez finalizado, se hace la presentación y se práctica la defensa del proyecto. Dándose así, por finalizado completamente, el día de la exposición ante el Tribunal.

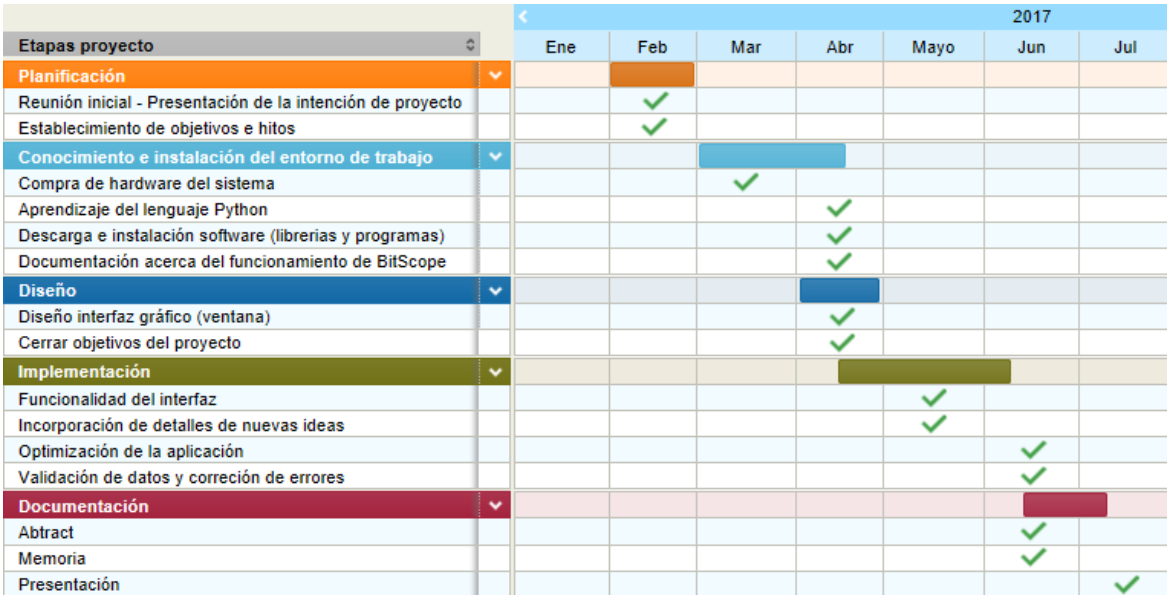

Ilustración 4. Diagrama de Gantt del proyecto

4.4.2 ESTIMACIÓN ECONÓMICA

A continuación, en la Tabla 1, viene un listado y descripción del hardware que se ha necesitado para llevar a cabo este proyecto.

El coste total del hardware ha sido de 191,07 €.

Componente	*Precio unitario*	*Cantidad*	*Precio total*	*Código RS/Farnell*	*Link*
Cable HDMI	12,00 €	1	12,00 €	489-412	http://es.rs-online.com/web/p/products/489-412/
Alimentación de Raspberry-pi	8,68 €	1	8,68 €	909-8126	http://es.rs-online.com/web/p/fuente-de-alimentacion-enchufable/9098126/?origin=PSF_430702|acc
Raspberry-pi 3	32,99 €	1	32,99 €	896-8660	http://es.rs-online.com/web/p/kits-de-desarrollo-de-procesador-y-microcontrolador/8968660/

Adaptador BNC	24,00 €	1	24,00 €	2455505	http://es.farnell.com/bitscope/mp01a/bnc-adapter-bitscope-micro-oscilloscope/dp/2455505?ost=bitscope&categoryId=700000022505
BitScope	98,00 €	1	98,00 €	2432906	http://es.farnell.com/bitscope/bitscope-micro/oscilloscope-2-6ch-20mhz-40msps/dp/2432906
Tarjeta MicroSD	15,40 €	1	15,40 €	121-3897	http://es.rs-online.com/web/p/tarjetas-sd/1213897/?origin=PSF_437585|acc

Tabla 1. Coste de componentes hardware

Capítulo 5. SISTEMA DESARROLLADO

5.1 ANÁLISIS DEL SISTEMA

El sistema pensado a seguir es el descrito en el esquema de la Ilustración 5. En este apartado se hará una descripción del sistema, a muy alto nivel. Ya en el apartado 5.2 de diseño, se profundizará en los componentes que se han escogido, las decisiones que se han tomado y el porqué.

- Señal: debe haber una fuente que genere las señales de entrada o se puede utilizar cualquier otra señal salida de otro sistema (ya sea analógica o digital).
- Muestreador: se debe muestrear la señal y enviar las muestras al PC/Raspberry-pi.
- PC/Raspberry-pi: las muestras deben llegar al dispositivo final a través de un puerto USB. En el monitor del dispositivo final, cuando se ejecute la aplicación, aparecerá una ventana en la que se podrá ver representada la señal entrante en tiempo y en frecuencia y podrá ser filtrada.

Ilustración 5. Conexión del sistema a alto nivel

5.2 DISEÑO

En este apartado se describirá como, dados los objetivos y el sistema planteado, se tomaron las decisiones de diseño y porqué. Los elementos elegidos finalmente son los que figuran en la Ilustración 6.

- Generador de señales: se ha elegido un generador de señales, ya que nos ofrece distintas posibilidades de señal, diseñada a la medida del usuario. Sin embargo, como se ha comentado con anterioridad, cualquier señal, analógica o digital, valdría.

- BitScope: para la digitalización de las señales entrantes, se ha decidido utilizar BitScope. Éste plantea un entorno totalmente funcional donde la digitalización de la señal, se encuentra totalmente resuelta, por lo que el muestreo de las señales es transparente al usuario. Solo se debe inicializar con las condiciones deseadas al activar el dispositivo hardware y tras esto, nos podemos centrar únicamente en las muestras que devuelve de la señal. Además, permite la entrada de señales tanto analógicas como digitales. BitScope cumple con los requisitos establecidos y nos sirve como base para alcanzar los objetivos propuestos. Su funcionalidad viene descrita en el punto 2.4 de este documento. Se introduce la señal en el muestreador, ya sea mediante los hilos de prueba sonda, que vienen junto con el BitScope; o con cables BNC, aprovechando el adaptador BNC del que se dispone. El BitScope muestrea la señal y envía las muestras al PC/Raspberry-pi. Para comunicarse, configurar y programar el BitScope, se utilizará la librería propia que dispone, Bitlib y lenguaje Python.

- PC/Raspberry-pi: las muestras llegan al dispositivo final a través de un puerto USB. El dispositivo final, ya sea PC con Windows, Linux o MAC, o Raspberry, debe tener Python instalado y la librería Biltlib para poder ejecutar la aplicación (otra de las

ventajas de BitScope es la posibilidad de trabajar en cualquier entorno). Para crear el interfaz gráfico en forma de ventana de ordenador con el que pueda interactuar el usuario, se utiliza Qt Designer. Este programa, permite generar una ventana de manera visual en cuanto a la elección y colocación de los componentes que van a formar parte de la misma. Una vez finalizada y estando conforme con el resultado, se puede obtener el código de la misma. En el monitor del dispositivo final, cuando se inicie la aplicación, aparecerá una ventana en la que se podrá representar la señal entrante en tiempo y en frecuencia; se podrá controlar el trigger de la representación temporal; se podrá congelar la señal; la señal podrá ser filtrada por un filtro a diseñar con coeficientes o especificaciones del filtro; y se podrá ver el Diagrama de Bode del filtro diseñado. El diseño de la ventana, aún sin funcionalidad, será el que se observa en la Ilustración 7.

Ilustración 6. Conexión del sistema detallado

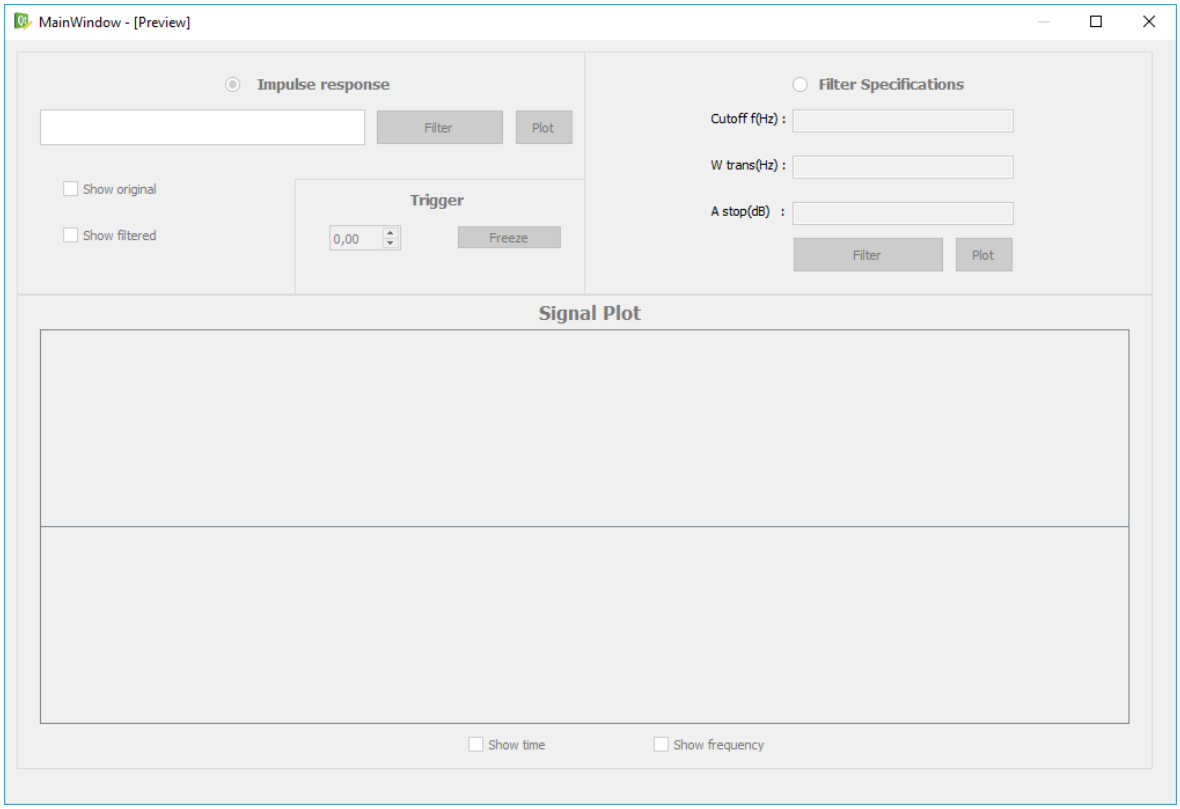

Ilustración 7. Diseño de la ventana de la aplicación

5.3 IMPLEMENTACIÓN

Empezaremos por la creación de la ventana de la aplicación, que tendrá el diseño mostrado anteriormente en la Ilustración 7. Una vez creada con Qt Designer, se guarda y se obtiene un fichero .ui con formato XML. Para pasar de XML a código Python, se utiliza un fichero Python, pyuic, que permite esta transformación.

```
python pyuic.py -o nombre_fichero.py nombre_fichero.ui
```

Todo lo relacionado con el diseño inicial de la ventana, se guardará en un fichero con extensión .py que no debe ser modificado nunca manualmente. En caso de querer proceder a realizar un cambio, se hará con el software de Qt Designer, y se hará un reemplazo entre el fichero antiguo y el nuevo.

Lo primero se lleva a cabo en el fichero principal (main) es la inicialización de la ventana, de modo que cuando se ejecute dicho fichero, lo primero que se produzca sea la creación de la misma.

A continuación, se inicializan las variables auxiliares y las variables para lectura de datos entrantes.

Se configuran punteros de otras clases hacia el main.

Se crean instancias de Sampler (que se encargará de todo lo relacionado con la conexión y comunicación con BitScope); y de Plotter, que será la clase encargada de refrescar la ventana.

La secuencia que se debe llevar siempre con el BitScope debe ser:

(1) Initialize → (2) Setup → (3) Trace → (4) Acquire → (5) Close

Lo primero que se realiza por tanto es la inicialización y el setup del BitScope. Para ello se tiene un método *open_scope* en la clase Sampler. Antes de empezar a dar funcionalidad a la ventana, haremos el *open_scope*. (1) Utilizando la API de BitScope, se aprende que el muestreador se inicializa de la siguiente manera:

- `BL_Initialize()` - inicializa la librería.
- `BL_Open("", 1)` - abre el dispositivo.

(2) Una vez inicializado, se hace el setup:

- `BL_Count(BL_COUNT_ANALOG)`

  `BL_Count(BL_COUNT_LOGIC)`

Detecta el número de canales analógicos y canales lógicos.

- `BL_Select(BL_SELECT_DEVICE, MY_DEVICE=0)`

  `BL_Select(BL_SELECT_CHANNEL, MY_CHANNEL=0)`

  `BL_Select(BL_SELECT_SOURCE, BL_SOURCE_POD)`

  Se seleccionan el dispositivo, el canal y la fuente. El primer argumento especifica qué tipo de entidad seleccionar. El canal 0 elegido, es el CHA.

- `BL_Mode(MY_MODE=BL_MODE_FAST)` - Cuando se selecciona el dispositivo por primera vez, también debe seleccionarse su modo de trace. Debe ser llamado después de seleccionar el dispositivo, pero antes de seleccionar el canal ya que el número de canales disponibles puede ser menor que el número físico que el dispositivo admite en algunos modos.
- `BL_Range(BL_Count(BL_COUNT_RANGE))` – selecciona el rango de canales. En nuestro caso, se selecciona el rango máximo.
- `BL_Offset(BL_ZERO)` – asigna el offset del canal (0 en nuestro caso).
- `BL_Enable(TRUE)` – cambia el estado del canal a habilitado.

Con esto se podría dar por finalizado el setup, sin embargo, se ha decidido configurar también en el *open_scope* el número de muestras a leer en cada vez del BitScope (es decir, el número de puntos), y la frecuencia de muestreo que se utiliza en el mismo.

- `BL_Rate(self.MY_RATE=1000000)` – se establece la frecuencia de muestreo a 1MHz
- `BL_Size(self.MY_SIZE=1000)` – se reciben las muestras en bloques de 1000 (que serán los puntos con los que se representará la señal).

En segundo plano se pretende que el muestreador esté cogiendo continuamente muestras. Para ser exactos, se desea obtener muestras cada 50 ms, sin que esto afecte al funcionamiento principal y visual de la aplicación. Para hacer esto, se ha decidido crear un hilo. En dicho hilo, se inicializa un timer que llama al método *scope_acquire*, y que se configura para actuar cada 50 ms. El hilo se establece como secundario (daemon) y por tanto, actúa en paralelo.

El hilo (thread) queda configurado de la siguiente manera:

```
thread1 = threading.Thread(self.my_sampler.startAcquire())

thread1.setDaemon(True)

thread1.start()

thread1.join()
```

Y la configuración del timer y llamada a *scope_acquire* es la siguiente:

```
self.timer = QtCore.QTimer()

self.timer.timeout.connect(lambda: self.scope_acquire())

self.timer.start(50)
```

En el *scope_acquire* es donde se realizan las fases (3) y (4).

- `BL_Trigger(self.TRIGGER_VALUE,BL_TRIG_RISE)-` establece el nivel del trigger. En nuestro caso el nivel del trigger se leerá de una double spin box de la ventana. Sin embargo, inicialmente, se ha decido poner a 0, hasta que es usuario decida cambiarlo.
- `BL_Trace(0.1, False) -` comienza el trace y captura la señal. Se puede hacer de manera síncrona o asíncrona. Se ha decidido hacerlo de manera síncrona con un timeout de 100 ms. En este caso se garantiza que BL_Trace regrese dentro del tiempo especificado, pero la traza puede o no haber terminado en ese tiempo. Devolverá TRUE si tiene y FALSE en caso contrario. Si se ha completado, se podrá mostrar el disparo capturado, si no, empezará a disparar cada 0.1s que no se haya conseguido disparo, y por tanto no se mostrará una imagen fija, si no, la consecución de disparos realizados. Para que el usuario sepa en qué situación se encuentra de las dos, se le informará con una label que explicaremos más adelante como funciona.
- `self.ch1_data = BL_Acquire() -` adquiere datos del dispositivo. Una vez que se ha completado el trace, los datos (muestras) pueden ser adquiridos. BL_Acquire

carga los datos del dispositivo de un canal a la vez. En nuestro caso, se devolverá una list de 1000 muestras cada vez que sea llamado. Sin embargo, y para no saturar la aplicación, éstas serán guardadas en una variable de clase *self.ch1_data* para que cuando vayan a ser utilizados (ya sea para refrescar la representación o para cualquier otro cometido), esté disponible la más actualizada, y así evitar que cada vez que se hiciera un *BL_Acquire()*, se interrumpiera en proceso principal.

Ahora se pasa a dar funcionalidad a los botones del interfaz. Para conectar los botones con su método, se utilizará la siguiente sintaxis:

```
self.ui.nombre_boton.clicked.connect(self.nombre_metodo)
```

Las funcionalidades de la ventana serán:

- Representación temporal de la señal original: cuando los check box *Show original* y *Show time*, estén seleccionadas, el widget Graphics View superior, estará habilitado y se mostrará la señal entrante en color verde. Los ejes X e Y representarán tiempo y tensión (amplitud) respectivamente, y se adaptarán automáticamente a los cambios de rango de la señal que sea introducida.
- Representación en frecuencia de la señal original: cuando los check box *Show original* y *Show frequency*, estén seleccionadas, el widget Graphics View inferior, estará habilitado y se mostrará el espectro en frecuencia de la señal entrante en color verde. Para pasar de tiempo a frecuencia, se ha realizado la FFT de las muestras de la señal. Para ello, utilizando la librería *numpy.fft*, se aplica el algoritmo de la FFT a cada una de las muestras de la señal:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{i2\pi kn}{N}} \qquad k = 0, \dots, N-1$$

Se mostrará el valor absoluto del resultado y exclusivamente la parte positiva en frecuencia de los armónicos (la parte de la derecha del cero). Los ejes X e Y representarán frecuencia y tensión (amplitud) respectivamente, y se adaptarán automáticamente a los cambios de rango de la señal que sea introducida.

- Filtrado:
  - A partir de los coeficientes del filtro que se desea aplicar. Se introducen los coeficientes de la forma [b_1, b_2, …, b_k] en la caja de texto de la sección "Impulse response".
  - A partir de las especificaciones del filtro que se desea aplicar. Se introduce la frecuencia de corte (donde se produce la caída a la mitad del valor original de amplitud – 6 dB) en Hertzios, ancho deseado en el paso de transición en Hertzios y la atenuación deseada en la banda de stop en dB. A partir de estas características y utilizando la librería *scipy.signal*, se ha sido capaz de sacar los coeficientes del filtro capaz de cumplir con lo requerido. Con el método kaiserord se consigue diseñar una ventana Kaiser para limitar el rizado y el ancho de transición de una región. La ventana de Kaiser es una ventana w_k muy cercana a la ideal utilizada para procesamiento digital de señales definida por la fórmula:

$$w_k = \begin{cases} \dfrac{I_0\left(\pi\alpha\sqrt{1-\left(\frac{2k}{n}-1\right)^2}\right)}{I_0(\pi\alpha)} & 0 \le k \le n \\ 0 & resto \end{cases}$$

I_0 = función de Bessel de primer tipo de orden cero

α = número real arbitrario que determina la forma de la ventana

n = número natural que determina el tamaño de la ventana

Con *firwin* se calcula los coeficientes del filtro FIR correspondiente. El filtro tendrá fase lineal.

Una vez que se obtienen los coeficientes, ambos casos se tratan de la misma manera. Para realizar el filtrado se utiliza el método de solape y almacenamiento. Este método permite realizar el filtrado en escenarios donde la señal a filtrar es una señal en "streaming", es decir, donde no se tiene desde un principio todas las muestras de la

señal. En estos se necesita hacer filtrado "al vuelo", como el que aquí se propone con esta técnica. La idea es partir la señal en trozos y aplicar la convolución circular a cada uno de los bloques. Se puede hacer ya que la convolución a trozos equivale a la total:

$$y[n] = x[n] * h[n] = \left(\sum_i X_i[n]\right) * h[n] = \sum_i x_i[n] * h[n] = \sum_i y_i[n]$$

Siendo L el tamaño de los bloques y P la longitud de los coeficientes del filtro, la técnica programada consiste en lo siguiente:

1. Número de bloques: tamaño de las muestras (nº de puntos) / L-P+1
2. Filtro:
    a. Coeficientes + (L-P) ceros al final
    b. Se hace la FFT del resultante
3. Se añaden P-1 ceros delante de los valores de las muestras de la señal
4. Por cada bloque se hace:
    a. Se cogen desde [k*(L-(P-1)) : k*(L-(P-1))+L], siendo k el nº de bloque.
    b. Se hace la FFT de cada bloque
    c. Se multiplica cada bloque con el filtro
    d. Se hace la IFFT de la señal resultante
    e. Se almacena en una variable que se queda únicamente con los elementos [P : L].

Para hacer esta explicación más visual, se ha decido añadir 2 ilustraciones. En la Ilustración 8, viene la representación de los puntos 1-4.b y en la Ilustración 9 vienen los puntos 4.c-4.e.

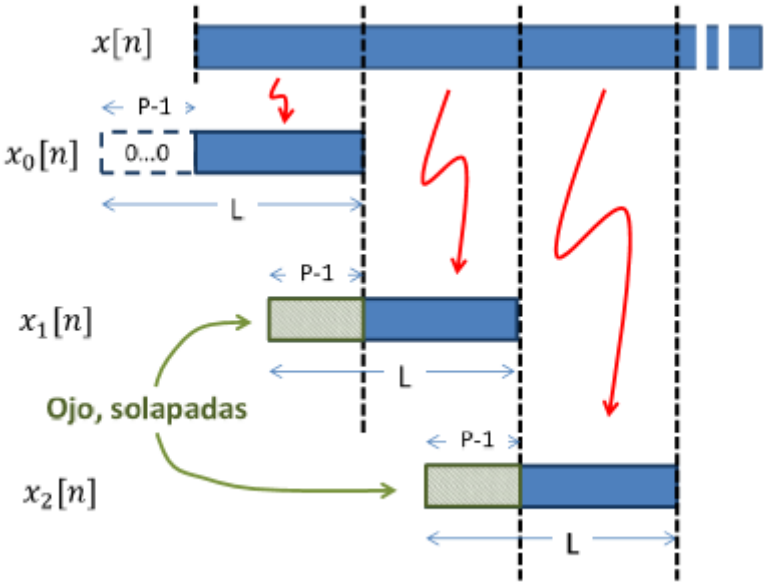

Ilustración 8. Solape y almacenamiento 1

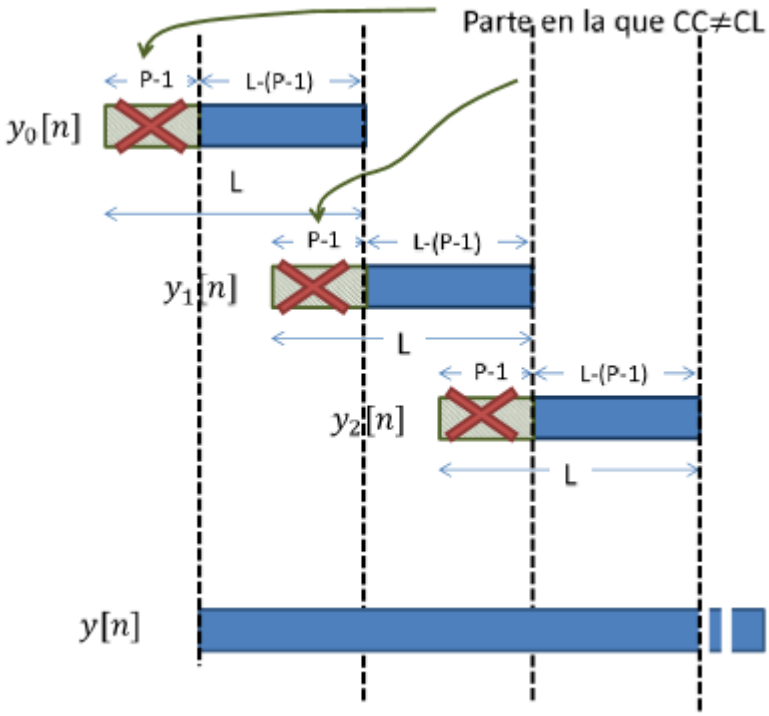

Ilustración 9. Solape y almacenamiento 2

Una vez introducidos los criterios del filtro y pulsado el botón *Filter*, si están pulsados los check boxes *Show time*, *Show frequency* y *Show filtered*, aparecerá la señal filtrada en tiempo y frecuencia en color rojo representada.

Además, en el filtrado se dispone de la opción de ver representado, en escala logarítmica en dB, el diagrama de Bode del filtro que vamos a introducir o que ya hemos introducido. Para esta representación, se siguen los mismos pasos que en el filtrado de la señal, sin embargo, en vez de, una vez calculado el filtro, multiplicarlo por la señal de entrada, se hace el $20\log_{10}(h[n])$ y se representa en una ventana a parte, cómo podemos ver en la Ilustración 10.

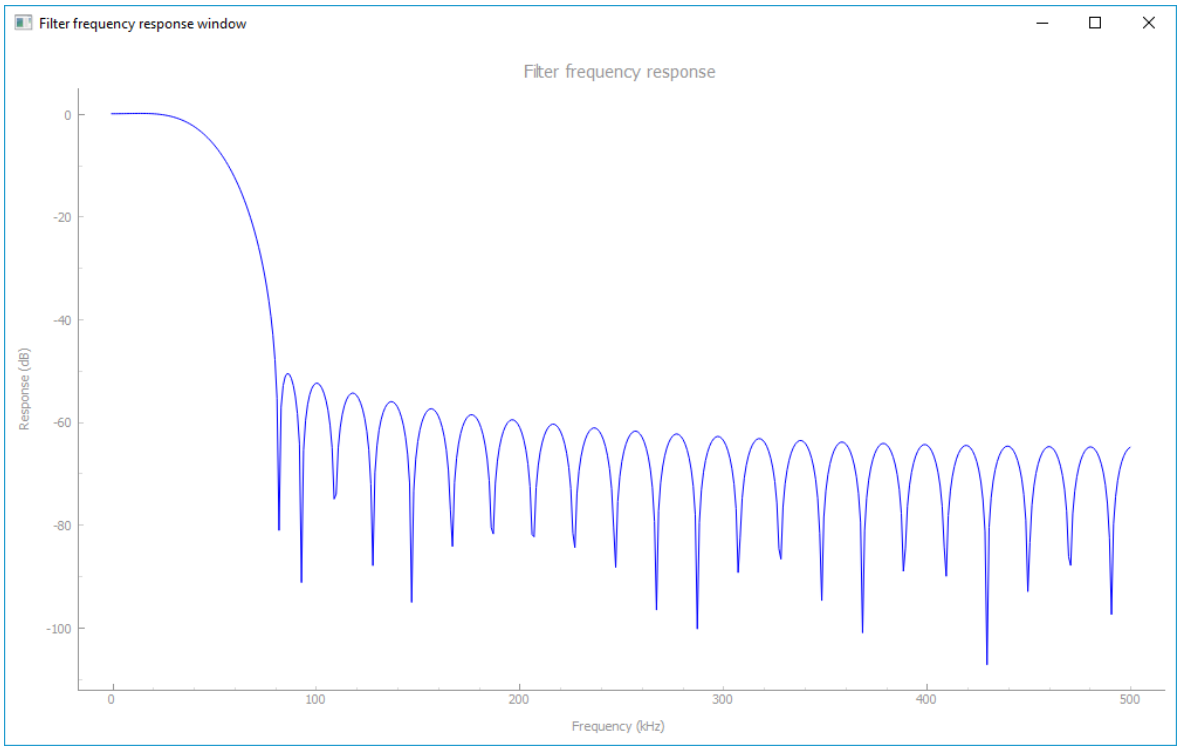

Ilustración 10. Diagrama de Bode del filtro

- Control del trigger - se puede controlar el nivel del trigger, que nos permite capturar las señales y representarlas de manera estática cuando se cumple la condición. Para

determinar el nivel del trigger, se dispone de un double spin box que permite cambiar el nivel.

Además, pulsando el botón *Freeze* del trigger, se pueden congelar las gráficas. De este modo se puede ampliar, reducir y mover la señal a nuestro gusto, y así ser analizada sin que la señal esté siendo continuamente refrescada y ajustada a su rango. Para volver a este último caso, basta con volver a pulsar el botón.

Todo lo anteriormente descrito no es algo estático. Esta aplicación se está continuamente refrescando. En el main se establece un timer, que actúa cada 500 ms y que conecta con la clase *Plotter* para la actualización de la ventana con la siguiente sintaxis:

```
self.timer = QtCore.QTimer()

self.timer.timeout.connect(lambda: self.my_plotter.update())

self.timer.start(500)
```

En el update se actualiza:

1. La representación en tiempo y en frecuencia tanto de la señal original como de la filtrada. Se envían las nuevas muestras sacadas por el muestreador.
2. La representación u ocultación de las señales.
3. El nivel del trigger
4. El estado de trigger: *Triggered, Armed* o *Frozen*.

Cuando se pulsa el botón rojo de cerrar ventana se:

- Cierra la ventana gráfica
- Cierra el Bistcope (5)
    - `BL_Close()` - Cierra todos los dispositivos abiertos (no es posible cerrar sólo uno).

El resultado final de la aplicación en funcionamiento es el que se muestra en la Ilustración 11.

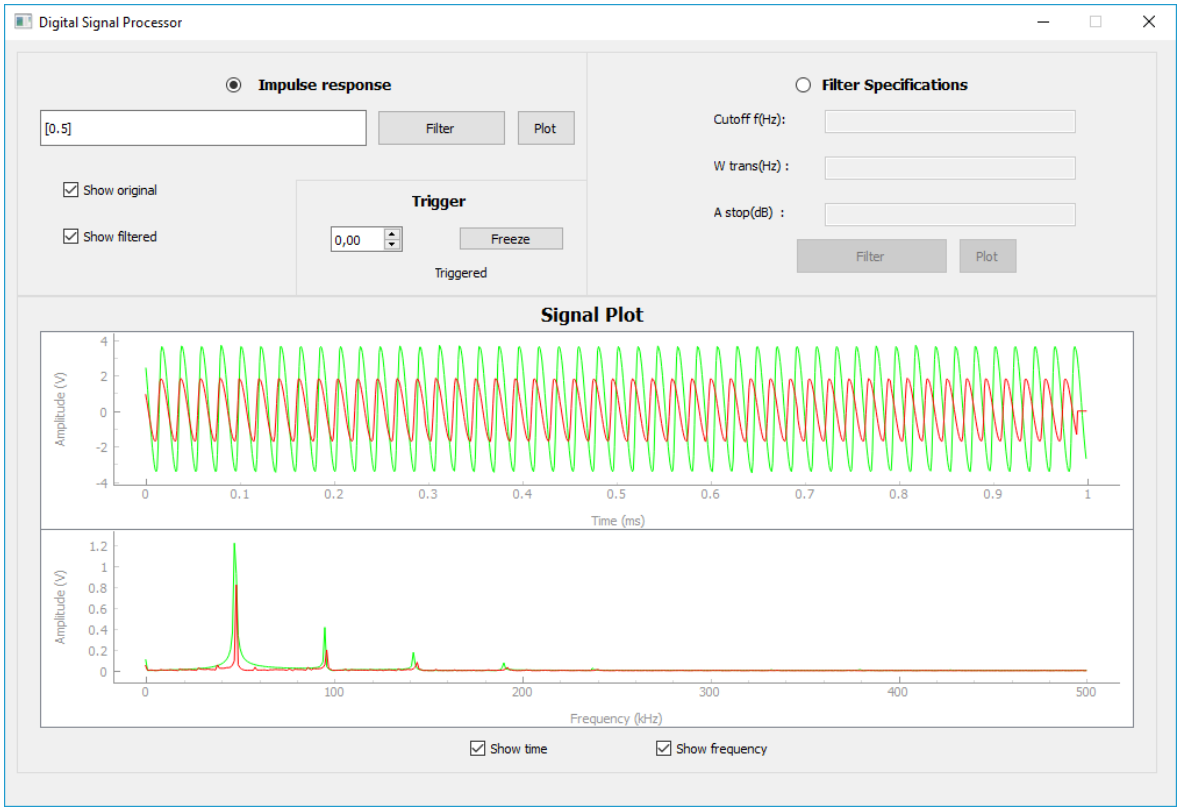

Ilustración 11. Ventana de aplicación en funcionamiento

Capítulo 6. ANÁLISIS DE RESULTADOS

Una vez finalizado el proyecto, se puede determinar que se ha conseguido el objetivo inicial, incluso se ha sido capaz de incrementar el alcance y la funcionalidad del proyecto.

A continuación, se mostrará con una Ilustración la consecución de cada objetivo.

- Representación de señales digitales y analógicas en tiempo y frecuencia (Ilustración 12). Se ve representada la señal de entrada (senoidal de 46 KHz de frecuencia y 4 V de amplitud) en verde. Al ser la f_s 1 MHz y el número de puntos 1000, la ventana temporal va de 0 a 1 ms. En este caso se representarán 46 ciclos. También apreciamos que la señal no es seniodal pura, y se ven otros valores de frecuencias y valor medio, aunque de muy baja amplitud.

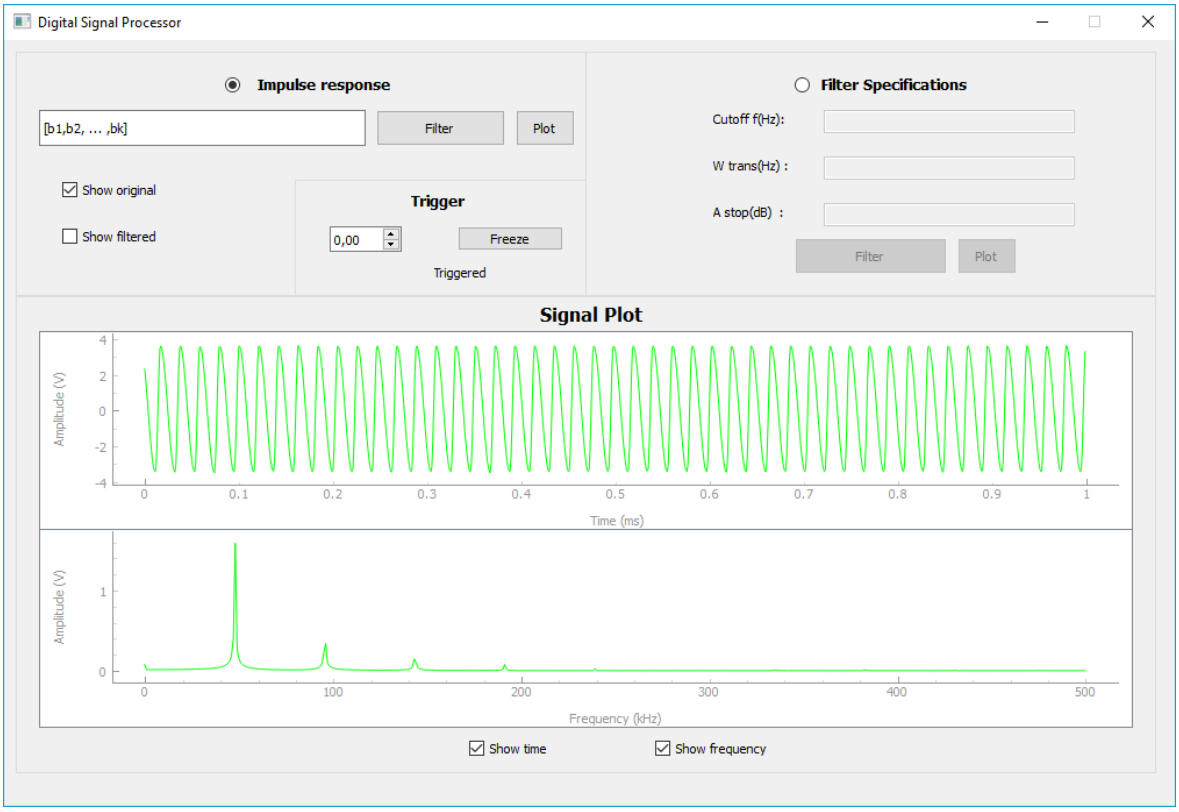

Ilustración 12. Representación temporal y en frecuencia de señal entrante

- Control del trigger de la gráfica de la señal representada en tiempo e información de si nos encontramos en estado de: *Triggered* (Ilustración 12 – amplitud de la señal 4 V y trigger en 0 V → Triggered), *Armed* (Ilustración 13 - amplitud de la señal 4 V y trigger en 5 V → Armed) *o Frozen* (Ilustración 14 – botón *Freeze* pulsado → Frozen).

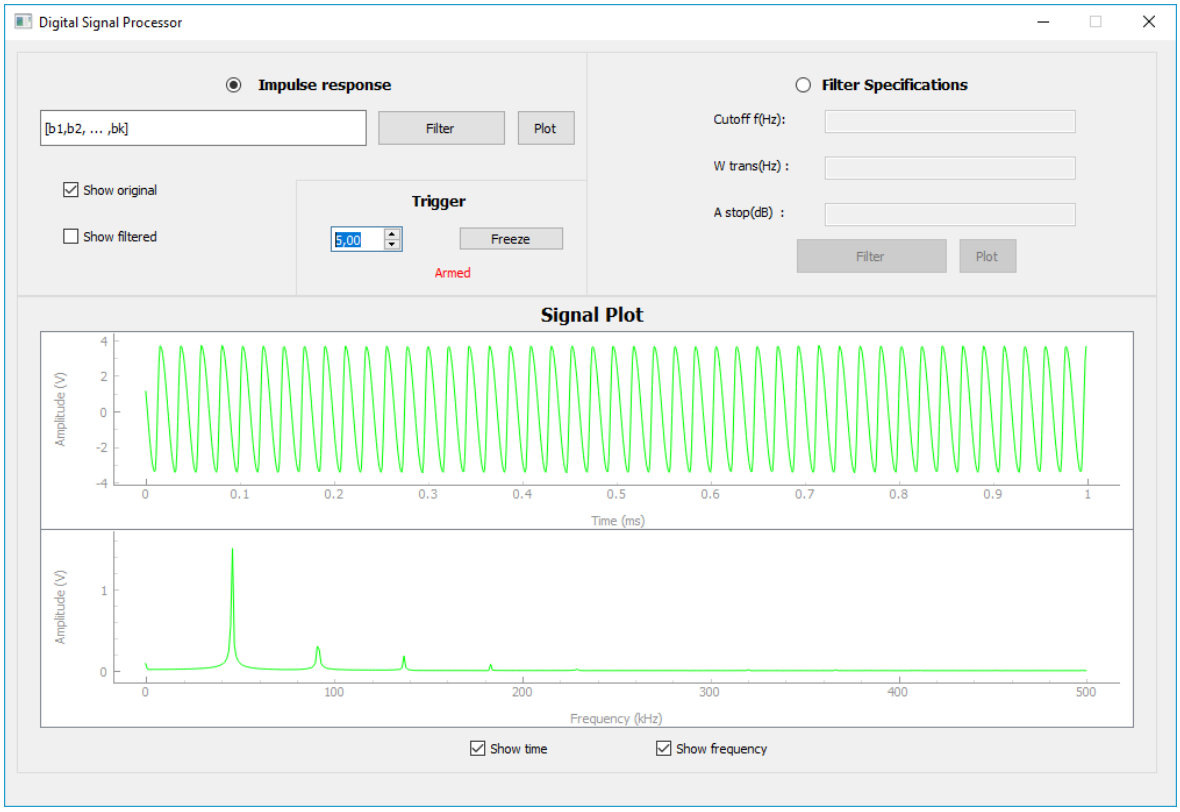

Ilustración 13. Trigger Armed

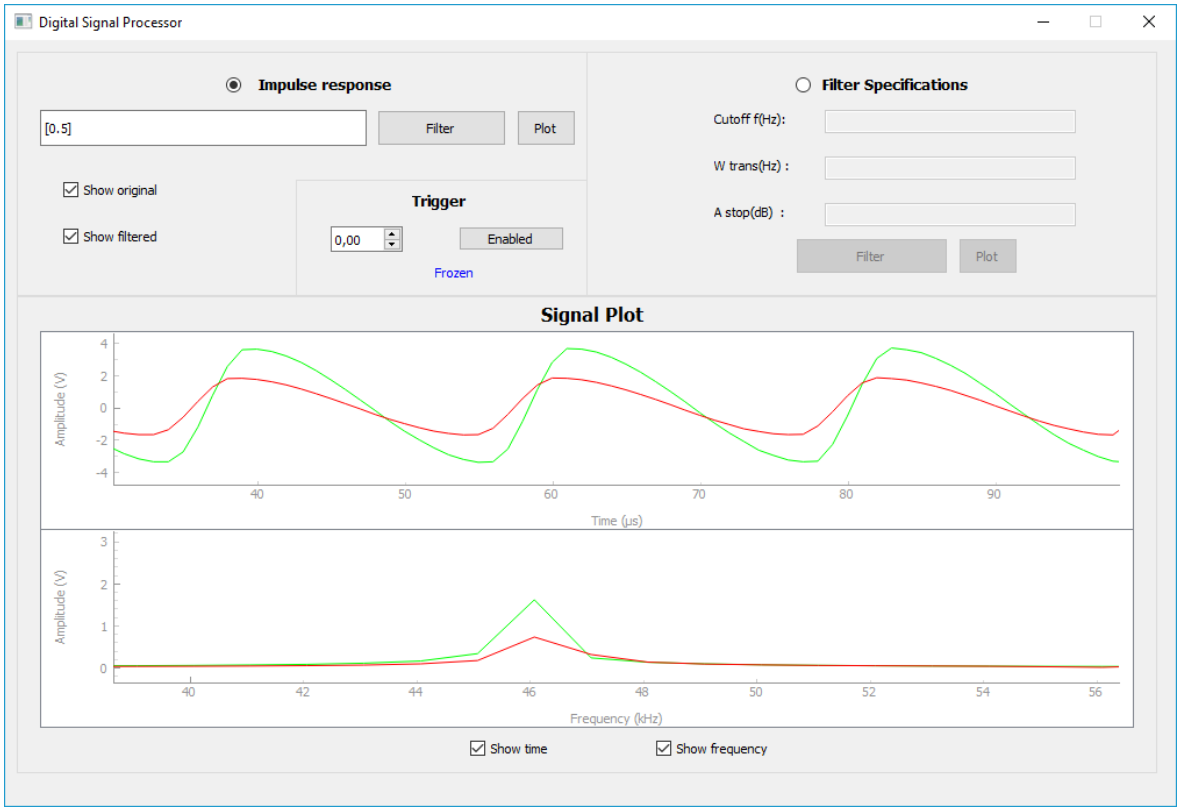

Ilustración 14. Trigger Frozen

- Control del usuario para congelar, mover y ampliar y reducir, las representaciones de las señales (Ilustración 14). Una vez vuelto a pulsar el botón *Enabled*, se recupera el autorange y se vuelve a actualizar la gráfica con nuevas muestras entrantes.

- Representación logarítmica en dB de los filtros creados (Ilustración 15 e Ilustración 16). Se pueden ampliar, mover y reducir, de forma que si se quiere ver que caída se tiene para cada frecuencia, se puede conseguir, ampliando para dicha frecuencia.

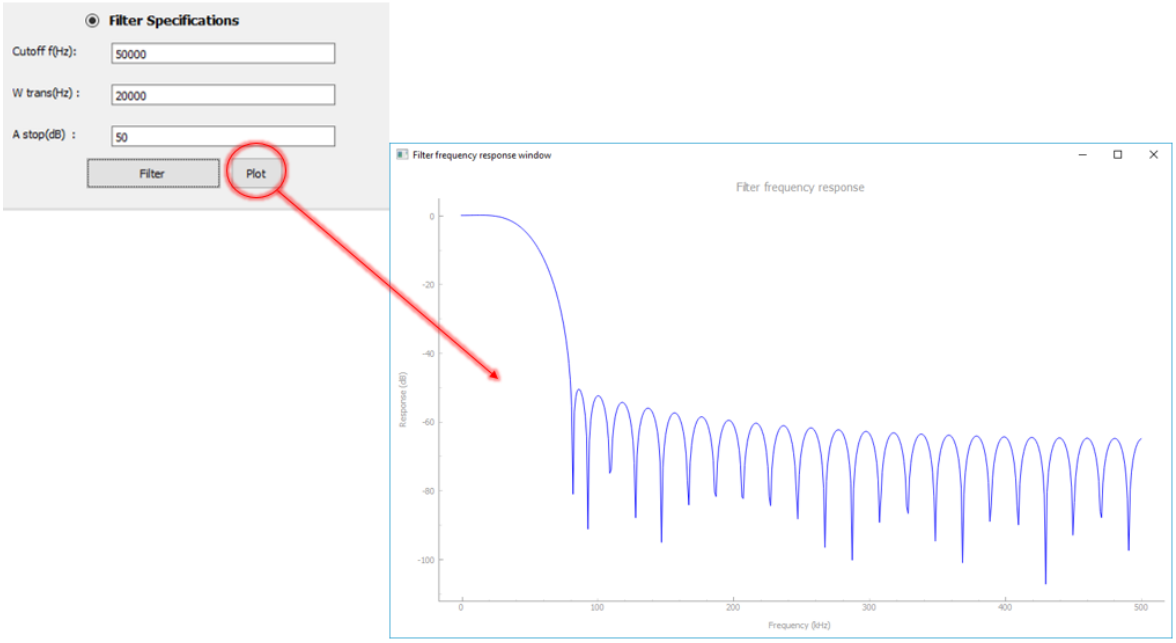

Ilustración 15. Representación de filtro (especificaciones)

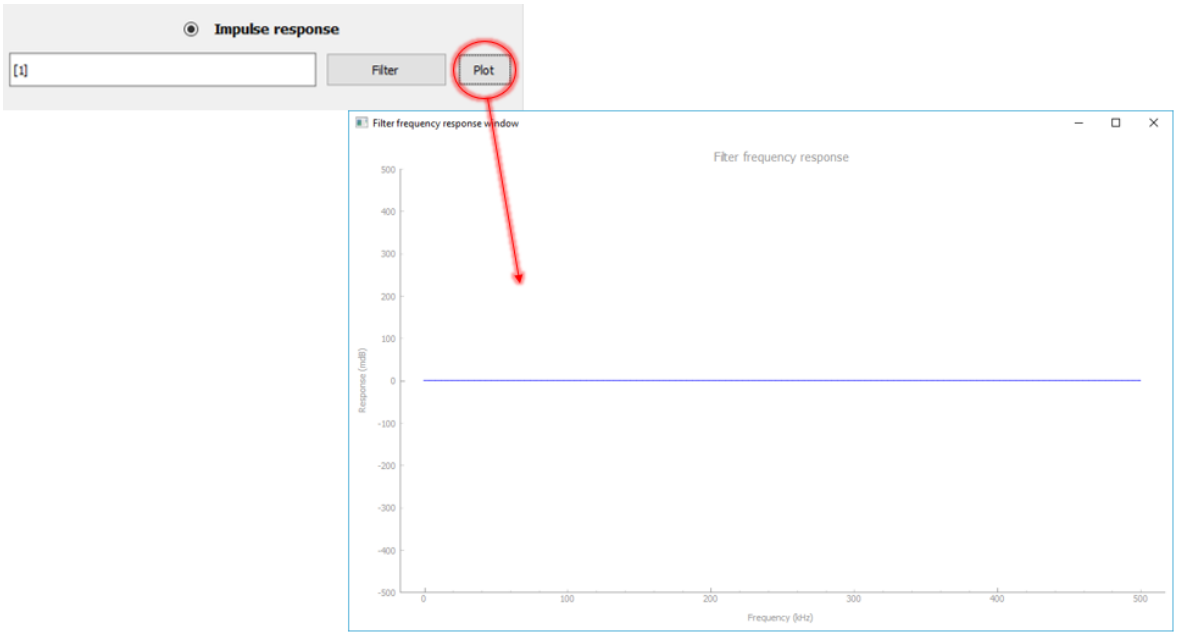

Ilustración 16. Representación de filtro (coeficientes)

- Filtrado de la señal de entrada, y representación. Dados:
    - Los coeficientes del filtro. (Ilustración 18)
    - La frecuencia de corte, el ancho de banda de paso y la atenuación en la banda de stop. (Ilustración 17) Primero se sacan los coeficientes y luego se representa (solo en el caso de tener el check box *Show filtrered* activo).

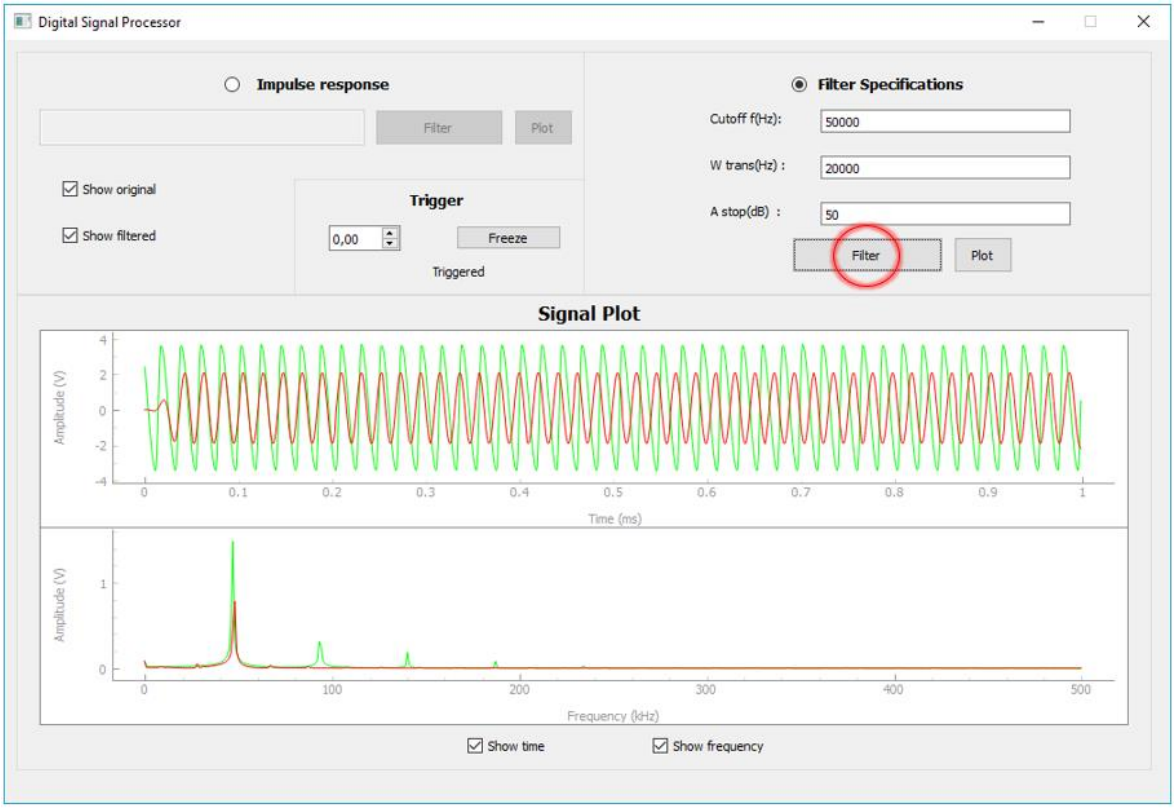

Ilustración 17. Representación del filtrado con especificaciones

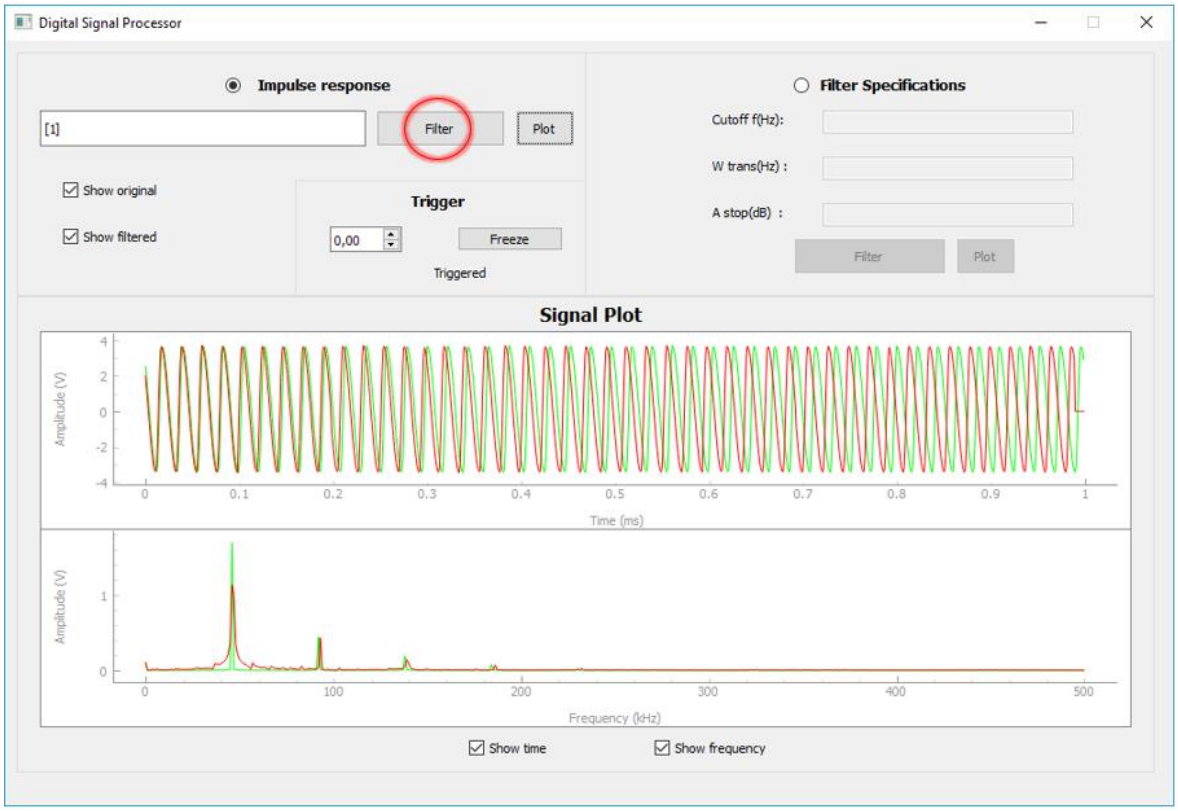

Ilustración 18. Representación del filtrado con coeficientes

Se observa en ambos casos, como el filtrado con filtros FIR, produce un desfase en la salida. Esto se debe a que los filtros FIR diseñados, no tienen fase lineal.

- Poder decidir representar la señal con los check box: en tiempo (*Show time*), en frecuencia (*Show frequency*), la señal original (*Show original*), la señal filtrada (*Show filtered*), o cualquier combinación deseada de estas cuatro. (Ilustración 19)

Ilustración 19. Elección de representaciones

- Validación de los datos de entrada y notificación descriptiva en caso de error. (Ilustración 20). Datos fuera del límite de rango → out of range. Datos mal escritos, con letras o vacíos → sintax error.

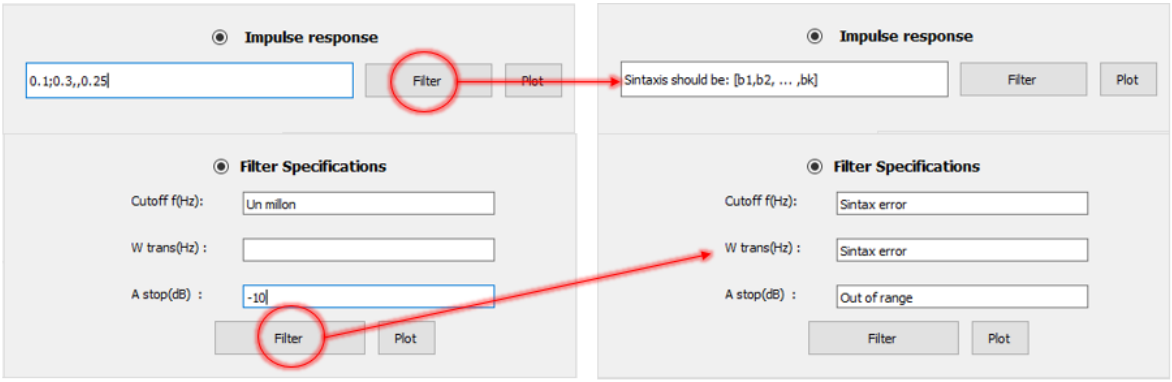

Ilustración 20. Validación de datos

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

El resultado final de este proyecto es una aplicación, compatible para ser ejecutado en cualquier sistema operativo, capaz de cumplir todos los objetivos comentados al inicio del documento; y todo ello sin una complicación excesiva de hardware, y sin grandes costes.

Se han cubierto los siguientes objetivos:

- Representar señales digitales y analógicas en:
  - Tiempo.
  - Frecuencia.
- Controlar el trigger de la gráfica de la señal representada en tiempo e información de si nos encontramos en estado de: *Triggered*, *Frozen o Armed*.
- Congelar, mover y ampliar y reducir, las representaciones de las señales.
- Sacar los coeficientes de un filtro a partir de sus especificaciones.
- Ver la representación logarítmica en dB de los filtros creados.
- Filtrar la señal de entrada dados:
  - Los coeficientes del filtro.
  - La frecuencia de corte, el ancho de banda de paso y la atenuación en la banda de stop.
- Representación de la señal filtrada.
- Poder decidir representar la señal: en tiempo, en frecuencia, la señal original, la señal filtrada, o cualquier combinación deseada de estas cuatro.
- Validación de los datos de entrada y notificación descriptiva en caso de error.

Y todo ello con un sistema sencillo, barato y programable como se buscaba (basado en BitScope).

Este programa tiene una interfaz de usuario simple, de modo que cualquier usuario con conocimientos básicos pueda utilizarlo.

La aplicación ya está preparada para reemplazar, en parte, a MATLAB en prácticas académicas de laboratorio. No tiene una funcionalidad completa, para cubrir todo tipo de prácticas y temario, pero si para una parte de ello.

Sin embargo, la librería Bitlib, nos da muchas más posibilidades a explotar en este ámbito.

Se podría ampliar la funcionalidad en la parte del osciloscopio:

- Añadiendo un botón para cambiar las divisiones por segundo en la representación temporal (modificando la frecuencia de muestreo).
- Añadiendo un cuadro con las medidas (valores) de la señal.

En la parte de procesador digital de señal, se podrían:

- Añadir distintos tipos de filtro (LPF, HPF, BPF).
- Se podría pasar la señal por distintos filtros consecutivos.
- Se podría pasar la señal por diezmadores e interpoladores, cambiando así la frecuencia de muestreo de la señal.
- Generación de señales.

En la Tabla 2, podemos ver, todo lo que nos ofrece BitScope, en comparación con otros sistemas:

	BitScope	*Osciloscopio / Analizador de espectros*	*MATLAB / Octave*
Programable	✓		✓
Trabajo sencillo con señales analógicas	✓	✓	
Funcionalidad de procesador digital de señal	✓		✓
Trabajo en tiempo real	✓	✓	
No necesaria licencia	✓	✓	
Económico	✓		✓

Tabla 2. Tabla comparativa BitScope

Las conclusiones que se han podido sacar de este proyecto, es que se puede ser capaz de juntar en una sola aplicación las funcionalidades de osciloscopio, analizador de espectros y procesador digital de señal gracias a BitScope y Python con sus librerías (PyQtGraph, PyQt4, NumPy, SciPy, Math, Threading, Sys).

Se ha llevado a cabo un proyecto que ha juntado conceptos de señal y programación. Se han abordado distintos ámbitos de las telecomunicaciones como el Teorema de muestreo de Nyquist-Shannon (para el muestreo de las señales entrantes), la Transformada de Fourier (para el cálculo del espectro en frecuencia de las señales), timers (para la actualización de la ventana de la aplicación), hilos (para optimizar la funcionalidad de la aplicación del BitScope, ya que puede ser bloqueante), Python (para la comunicación con el BitScope y

para la funcionalidad de la ventana), filtros FIR (para el filtrado de la señal), convolución circular y solape y almacenamiento (filtrado), punteros (para acceder de una clase a otra), etc.

Todo ello ha hecho de este proyecto un proyecto completo y enriquecedor, que ha permitido juntar conocimientos de distintas materias para distintas funciones, pero con un solo fin: la creación de esta aplicación.

Capítulo 8. BIBLIOGRAFÍA

REFERENCIAS WIKIPEDIA

[1] Conversión analógica - digital. (s.f.). En Wikipedia. Recuperado el 13 de junio de 2017 de https://es.wikipedia.org/wiki/Conversi%C3%B3n_anal%C3%B3gica-digital

[2] Procesamiento digital de señales. (s.f.). En Wikipedia. Recuperado el 13 de junio de 2017 de https://es.wikipedia.org/wiki/Procesamiento_digital_de_se%C3%B1ales

[3] Osciloscopio. (s.f.). En Wikipedia. Recuperado el 13 de junio de 2017 de https://es.wikipedia.org/wiki/Osciloscopio

[4] Analizador de espectro. (s.f.). En Wikipedia. Recuperado el 13 de junio de 2017 de https://es.wikipedia.org/wiki/Analizador_de_espectro

[5] Muestreo digital. (s.f.). En Wikipedia. Recuperado el 13 de junio de 2017 de https://es.wikipedia.org/wiki/Muestreo_digital

[6] Thread. (s.f.). En Wikipedia. Recuperado el 15 de junio de 2017 de https://en.wikipedia.org/wiki/Thread_(computing)

[7] Timer. (s.f.). En Wikipedia. Recuperado el 13 de junio de 2017 de https://en.wikipedia.org/wiki/Timer

[8] FFT. (s.f.). En Wikipedia. Recuperado el 13 de junio de 2017 de https://es.wikipedia.org/wiki/Transformada_r%C3%A1pida_de_Fourier

PÁGINAS DE PRODUCTOS OFICIALES

[9] Página Oficial BitScope - http://www.bitscope.org/

[10] Página Oficial PyQtGraph - http://www.pyqtgraph.org/

[11] Página Oficial Qt Designer - https://www.qt.io/ide/

[12] Página Oficial Python - https://www.python.org/downloads/

[13] Página Oficial MATLAB - https://es.mathworks.com/products/matlab.html

[14] Página Oficial GNU Octave - https://www.gnu.org/software/octave/

OTRAS PÁGINAS DE REFERENCIA

[15] Teoria del muestreo de Nyquist. (s.f.) Eveliux. http://www.eveliux.com/mx/Teoria-del-muestreo-de-Nyquist.html

[16] Filter Design. (s.f) http://www.utdallas.edu/~raja1/EE4361%20Spring%2014/Lecture%20Notes/FIR%20Windows.pdf

PROYECTOS RELACIONADOS

[17] PicBerry Oscilliscope and Function Generator with a PIC32 and Raspberry Pi - http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/f2016/ak634_jmw483_dm797/ak634_jmw483_dm797/ak634_jmw483_dm797/index.html

[18] Main Control System Test - https://twiki.ph.rhul.ac.uk/twiki/pub/Public/MainFiles/main_control_system_test.py.txt

[19] BitScope Pilab - https://bitbucket.org/bitscope/pilab

APIs

[20] BitLib - http://www.bitscope.com/software/library/API.html

[21] PyQt4 - http://pyqt.sourceforge.net/Docs/PyQt4/

[22] PyQtGraph - http://www.pyqtgraph.org/documentation/apireference.html

[23] NumPy - https://docs.scipy.org/doc/numpy-1.12.0/reference/

[24] SciPy - https://docs.scipy.org/doc/scipy/reference/api.html

[25] Math - https://docs.python.org/2/library/math.html

[26] Sys - https://docs.python.org/2/library/sys.html

[27] Threading - https://docs.python.org/2/library/threading.html

ANEXO A. CÓDIGO FUENTE

filterDesign

```
import math
from scipy import signal
from numpy import cos, sin, pi, absolute, arange
from scipy.signal import kaiserord, lfilter, firwin, freqz

class FilterDesigner():
    #Pasar de especificaciones a coeficientes del filtro
    @staticmethod
    def design(sample_rate,width,ripple_db,cutoff_hz):
        nyq_rate = sample_rate / 2.0
        width = width/nyq_rate
        N, beta = kaiserord(ripple_db, width)

        if N>=50:
            N=49

        taps = firwin(N, cutoff_hz/nyq_rate, window=('kaiser', beta))
        taps = taps/max(taps)

        return taps
```

filtro

```
import math
import numpy as np

class Filtro(object):

    #Solape y almacenamiento
    @staticmethod
    def filtrar(coeficientes,data,tipo_filtrado):
        coeficientes = np.asarray(coeficientes)
        data = np.asarray(data)
```

```
        L = 100
        P = len(coeficientes)

        #Calcular los trozos
        num_trozos = math.ceil(len(data)/(L-(P-1)))

        #Filtro
        h_n = np.concatenate((coeficientes, np.zeros(L-P)), axis=0)
        H_n = np.fft.fft(h_n)/len(h_n)

        num_zeros_ult_bloq = (L-P+1-(L-P+1)*(len(data)%(L-P+1))/100)
        data_add = np.concatenate((data, np.zeros(num_zeros_ult_bloq)), axis=0)
        #Senial
        x_n_pad = np.concatenate((np.zeros(P-1), data_add), axis=0)

        #Reservar memoria para la salida
        g_n=[]

        #Algoritmo de troceado, DFT, multiplicacion y almacenamiento
        for k in range(0, int(num_trozos+1)):
            #Trozo k-esimo
            x_k = x_n_pad[k*(L-(P-1)) : k*(L-(P-1))+L]
            #DFT
            X_k = np.fft.fft(x_k)/len(x_k)
            #Multiplicacion
            Z_k = np.multiply(X_k,H_n)
            #DFT inversa
            z_k = np.fft.ifft(Z_k)*math.pow(len(Z_k),2)
            #Almacenamiento
            if(k==(num_trozos)):
                decimal = len(data)/float(L-P)-int(len(data)/(L-P))
                ultimo = int(decimal*(L-P+1))
                g_n[k*(L-(P-1)) : (k+1)*(L-(P-1))-1-(L-P-ultimo)] = z_k[P : L-(L-
P-ultimo)]
                #if(len(coeficientes)>=50):
                #    g_n = np.concatenate((g_n,(np.zeros(1000-len(g_n)))),axis=0)
            else:
                g_n[k*(L-(P-1)) : (k+1)*(L-(P-1))-1] = z_k[P : L]


        if(tipo_filtrado==4):
```

```
            a = np.fft.fft(h_n,len(g_n)*2)
            a = np.abs(a[:len(g_n)])
            primero = a[0]
            for n in range(len(a)):
                g_n[n] = g_n[n]/primero

        return np.real(g_n)
```

main_ui

```
import sys
import pyqtgraph as pg
import threading
import math
import numpy as np
import time
from PyQt4 import QtGui, QtCore
from windows import VentanaFiltro
from bitlib import *
from sampler import Sampler
from plotter import Plotter
from parse import Parse
from filterDesign import FilterDesigner

class Main(QtGui.QMainWindow):
    def __init__(self):
        #Ventana inicializacion
        QtGui.QMainWindow.__init__(self)
        self.setFixedSize(1007, 658)
        self.ui = VentanaFiltro()
        self.ui.setupUi(self)
        self.setWindowTitle('Digital Signal Processor')

        #Auxiliares
        self.auxi_time = False
        self.auxi_freq = False
        self.auxi_filtrado = False
        self.auxi_ori = False
        self.auxi_filtered = False
        self.auxi_tigger = False
```

```
self.tipo_filtrado = 4

#Leido
self.txt = '[]'
self.coeficientes = [1]

#Datos que me introducirian en la ventana
self.MY_RATE = 1000000 #Frecuencia de muestreo
self.MY_SIZE = 1000 #Numero de puntos (tamanio)

#Metodos botones
self.ui.lineEdit_5.setText('[b1,b2, ... ,bk]')
self.ui.btnSpecifications.clicked.connect(self.btnSpecifications_Clicked)
self.ui.radioButton.clicked.connect(self.btnCoefficients_Clicked)
self.ui.checkBox_3.clicked.connect(self.checkBox_3_Clicked)
self.ui.checkBox_4.clicked.connect(self.checkBox_4_Clicked)
self.ui.pushButton.clicked.connect(self.pushButton_Clicked)
self.ui.pushButton_2.clicked.connect(self.pushButton_2_Clicked)
self.ui.checkBox.clicked.connect(self.checkBox_Clicked)
self.ui.checkBox_2.clicked.connect(self.checkBox_2_Clicked)
self.ui.pushButton_3.clicked.connect(self.pushButton_3_Clicked)
self.ui.pushButton_5.clicked.connect(self.pushButton_5_Clicked)
self.ui.pushButton_4.clicked.connect(self.pushButton_4_Clicked)

#Punteros hacia el GUI
self.my_sampler = Sampler(self.MY_SIZE)
self.my_sampler.setUI(self)
self.my_plotter = Plotter()
self.my_plotter.setUI(self)
self.my_plotter.setSampler(self.my_sampler)

#Abrir Bitscope
self.my_sampler.open_scope()


#Crear graficas de tiempo y frecuencia
win = pg.GraphicsWindow()
win.setWindowTitle('Bitscope analysis')
pg.setConfigOptions(antialias=True, background='w')
self.p6 = pg.PlotWidget()
self.p7 = pg.PlotWidget()
```

```
        self.p6.setLabel('bottom', 'Time', units='s',unitPrefix=None)
        self.p6.setLabel('left', 'Amplitude', units='V',unitPrefix=None)
        self.p6.resize(919,169)
        self.p7.setLabel('bottom', 'Frequency', units='Hz',unitPrefix=None)
        self.p7.setLabel('left', 'Amplitude', units='V',unitPrefix=None)
        self.p7.resize(919,169)
        self.curve = self.p6.plot(pen='g', name='t_original')
        self.curve3 = self.p6.plot(pen='r', name='t_filtered')
        self.curve2 = self.p7.plot(pen='g')
        self.curve4 = self.p7.plot(pen='r')
        self.scene = QtGui.QGraphicsScene()
        self.scene.addWidget(self.p6)
        self.scene2 = QtGui.QGraphicsScene()
        self.scene2.addWidget(self.p7)

        #Ventana filtro
        self.win3 = None
        self.p_plot = None
        self.curve_plot = None

        #Coger muestras continuamente en otro hilo secunadario (daemon)
        thread1 = threading.Thread(self.my_sampler.startAcquire())
        thread1.setDaemon(True)
        thread1.start()
        thread1.join()

      #Timer cada 0.5s para refrescar (update) de la ventana
        self.timer = QtCore.QTimer()
        self.timer.timeout.connect(lambda:
self.my_plotter.update([self.curve,self.curve2,self.curve3,self.curve4],[self.p6,
self.p7],self.MY_RATE,self.MY_SIZE,
self.auxi_filtrado,self.coeficientes,self.auxi_ori,self.auxi_filtered,self.auxi_t
igger,self.ui.doubleSpinBox.value(),self.ui.label_9,self.tipo_filtrado))
        self.timer.start(500)

    #RadioButton especificaciones filtro
    def btnSpecifications_Clicked(self):
        self.ui.lineEdit.setEnabled(True)
        self.ui.lineEdit_2.setEnabled(True)
        self.ui.lineEdit_3.setEnabled(True)
        self.ui.radioButton.setChecked(False)
```

```
        self.ui.lineEdit_5.setEnabled(False)
        self.ui.lineEdit_5.setText('')
        self.ui.pushButton.setEnabled(False)
        self.ui.pushButton_2.setEnabled(True)
        self.ui.pushButton_4.setEnabled(True)
        self.ui.pushButton_5.setEnabled(False)
        self.ui.lineEdit.setText('')
        self.ui.lineEdit_2.setText('')
        self.ui.lineEdit_3.setText('')


    #RadioButton coeficientes filtro
    def btnCoefficients_Clicked(self):
        self.ui.lineEdit.setEnabled(False)
        self.ui.lineEdit_2.setEnabled(False)
        self.ui.lineEdit_3.setEnabled(False)
        self.ui.lineEdit.setText('')
        self.ui.lineEdit_2.setText('')
        self.ui.lineEdit_3.setText('')
        self.ui.lineEdit_5.setText('[b1,b2, ... ,bk]')
        self.ui.btnSpecifications.setChecked(False)
        self.ui.lineEdit_5.setEnabled(True)
        self.ui.pushButton.setEnabled(True)
        self.ui.pushButton_2.setEnabled(False)
        self.ui.pushButton_4.setEnabled(False)
        self.ui.pushButton_5.setEnabled(True)

    #CheckBox show time
    def checkBox_3_Clicked(self):
        if(self.auxi_time==True):
            self.auxi_time = False
            self.ui.graphicsView.setScene(None)
        else:
            self.auxi_time = True
            self.ui.graphicsView.setScene(self.scene)

    #CheckBox show frequency
    def checkBox_4_Clicked(self):
        if(self.auxi_freq==True):
            self.auxi_freq = False
            self.ui.graphicsView_2.setScene(None)
```

```
        else:
            self.auxi_freq = True
            self.ui.graphicsView_2.setScene(self.scene2)

    #Boton filter coeficientes
    def pushButton_Clicked(self):
        try:
            self.coeficientes                                                    =
Parse.parseQStringToList(self.ui.lineEdit_5.text())
            self.tipo_filtrado = 5
            self.auxi_filtrado = True
        except Exception as e:
            self.ui.lineEdit_5.setText('Sintaxis should be: [b1,b2, ... ,bk]')
            self.ui.lineEdit_5.setStyleSheet("color: black")


    #Boton filter especificaciones
    def pushButton_2_Clicked(self):
        self.cutoff = None
        self.width = None
        self.astop = None
        try:
            self.cutoff = Parse.parseQStringToFloat(self.ui.lineEdit.text())
            if(self.cutoff<0 or self.cutoff>500000):
                self.ui.lineEdit.setText('Out of range')
        except Exception as e1:
            self.ui.lineEdit.setText('Sintax error')
        try:
            self.width = Parse.parseQStringToFloat(self.ui.lineEdit_2.text())
            if(self.width<0 or (self.cutoff+self.width)>500000):
                self.ui.lineEdit_2.setText('Out of range')
        except Exception as e2:
            self.ui.lineEdit_2.setText('Sintax error')
        try:
            self.astop = Parse.parseQStringToFloat(self.ui.lineEdit_3.text())
            if(self.astop<0 or self.astop>1000):
                self.ui.lineEdit_3.setText('Out of range')
        except Exception as e3:
            self.ui.lineEdit_3.setText('Sintax error')
```

```
        if(self.cutoff!=None and self.width!=None and self.astop!=None):
            if(not(self.cutoff<0 or self.cutoff>500000) and not(self.width<0 or
(self.cutoff+self.width)>500000) and not(self.astop<0 or self.astop>1000)):
                try:
                    self.coeficientes                                          =
list(FilterDesigner.design(self.MY_RATE,self.width,self.astop,self.cutoff))
                    self.tipo_filtrado = 4
                    self.auxi_filtrado = True
                except Exception as e:
                    self.ui.lineEdit.setText('Sintax error')
                    self.ui.lineEdit_2.setText('Sintax error')
                    self.ui.lineEdit_3.setText('Sintax error')


    #CheckBox show original
    def checkBox_Clicked(self):
        if(self.auxi_ori==True):
            self.auxi_ori = False
        else:
            self.auxi_ori = True

    #CheckBox show filtered
    def checkBox_2_Clicked(self):
        if(self.auxi_filtered==True):
            self.auxi_filtered = False
        else:
            self.auxi_filtered = True

    #Enable/Not enable trigger
    def pushButton_3_Clicked(self):
        if(self.auxi_tigger==True):
            self.auxi_tigger = False
            self.ui.pushButton_3.setText('Freeze')
        else:
            self.auxi_tigger = True
            self.ui.pushButton_3.setText('Enabled')

    #Plot coeficients filter
    def pushButton_5_Clicked(self):
        try:
            Parse.parseQStringToList(self.ui.lineEdit_5.text())
```

```
            self.win3 = pg.GraphicsWindow()
            self.win3.resize(1000,600)
            self.win3.setWindowTitle('Filter frequency response window')
            pg.setConfigOptions(antialias=True)
            self.p_plot = self.win3.addPlot(title="Filter frequency response")
            self.curve_plot = self.p_plot.plot(pen='r')
            self.p_plot.setLabel('bottom',                              'Frequency',
units='Hz',unitPrefix=None)
            self.p_plot.setLabel('left', 'Response', units='dB',unitPrefix=None)
            h = Parse.parseQStringToList(self.ui.lineEdit_5.text())
            xf,yf                                                                    =
self.my_plotter.calcularFFTCoeficientes(self.MY_SIZE,self.MY_RATE,h,5)
            hn = self.my_plotter.v_to_dB(yf)
            self.curve_plot.setData(x=xf,y=hn,pen='b')
            self.p_plot.enableAutoRange('xy', True)
            self.win3.show()
        except Exception as e:
            self.ui.lineEdit_5.setText('Sintaxis should be: [b1,b2, ... ,bk]')

    #Plot specifications filter
    def pushButton_4_Clicked(self):
        cutoff = None
        width = None
        astop = None
        try:
            cutoff = Parse.parseQStringToFloat(self.ui.lineEdit.text())
            if(cutoff<0 or cutoff>500000):
                self.ui.lineEdit.setText('Out of range')
        except Exception as e1:
            self.ui.lineEdit.setText('Sintax error')
        try:
            width = Parse.parseQStringToFloat(self.ui.lineEdit_2.text())
            if(width<0 or (cutoff+width)>500000):
                self.ui.lineEdit_2.setText('Out of range')
        except Exception as e2:
            self.ui.lineEdit_2.setText('Sintax error')
        try:
            astop = Parse.parseQStringToFloat(self.ui.lineEdit_3.text())
            if(astop<0 or astop>1000):
                self.ui.lineEdit_3.setText('Out of range')
        except Exception as e3:
```

```
            self.ui.lineEdit_3.setText('Sintax error')

        if(not(cutoff<0     or     cutoff>500000)     and     not(width<0     or
(cutoff+width)>500000) and not(astop<0 or astop>1000)):
            try:
                self.win3 = pg.GraphicsWindow()
                self.win3.resize(1000,600)
                self.win3.setWindowTitle('Filter frequency response window')
                pg.setConfigOptions(antialias=True)
                self.p_plot = self.win3.addPlot(title="Filter frequency response")
                self.curve_plot = self.p_plot.plot(pen='b')
                self.p_plot.setLabel('bottom',                          'Frequency',
units='Hz',unitPrefix=None)
                self.p_plot.setLabel('left',                            'Response',
units='dB',unitPrefix=None)
                h = FilterDesigner.design(self.MY_RATE,width,astop,cutoff)
                xf,yf                                                              =
self.my_plotter.calcularFFTCoeficientes(self.MY_SIZE,self.MY_RATE,h,4)
                hn = self.my_plotter.v_to_dB(yf)
                self.curve_plot.setData(x=xf,y=hn,pen='b')
                self.p_plot.enableAutoRange('xy', True)
                self.win3.show()
            except Exception as e:
                self.ui.lineEdit.setText('Sintax error')
                self.ui.lineEdit_2.setText('Sintax error')
                self.ui.lineEdit_3.setText('Sintax error')


    #Boton rojo (windowClosing)
    def closeEvent(self,event):
        result = QtGui.QMessageBox.question(self,
                      "Confirm Exit",
                      "Are you sure you want to exit?",
                      QtGui.QMessageBox.Yes| QtGui.QMessageBox.No)
        event.ignore()

        if result == QtGui.QMessageBox.Yes:
            event.accept()
            self.my_sampler.close_scope()
            sys.exit()
```

```
if __name__ == '__main__':
    app = QtGui.QApplication(sys.argv)
    windows = Main()
    windows.show()
    sys.exit(app.exec_())
```

windows

```
from PyQt4 import QtCore, QtGui

try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    def _fromUtf8(s):
        return s

try:
    _encoding = QtGui.QApplication.UnicodeUTF8
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig, _encoding)
except AttributeError:
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig)

class VentanaFiltro(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName(_fromUtf8("MainWindow"))
        MainWindow.resize(1011, 660)
        self.centralwidget = QtGui.QWidget(MainWindow)
        self.centralwidget.setObjectName(_fromUtf8("centralwidget"))
        self.groupBox = QtGui.QGroupBox(self.centralwidget)
        self.groupBox.setGeometry(QtCore.QRect(10, 10, 491, 211))
        self.groupBox.setTitle(_fromUtf8(""))
        self.groupBox.setObjectName(_fromUtf8("groupBox"))
        self.label = QtGui.QLabel(self.groupBox)
        self.label.setGeometry(QtCore.QRect(200, 20, 131, 16))
        self.label.setObjectName(_fromUtf8("label"))
```

```
self.checkBox = QtGui.QCheckBox(self.groupBox)
self.checkBox.setGeometry(QtCore.QRect(40, 110, 281, 17))
self.checkBox.setObjectName(_fromUtf8("checkBox"))
self.checkBox_2 = QtGui.QCheckBox(self.groupBox)
self.checkBox_2.setGeometry(QtCore.QRect(40, 150, 281, 17))
self.checkBox_2.setObjectName(_fromUtf8("checkBox_2"))
self.radioButton = QtGui.QRadioButton(self.groupBox)
self.radioButton.setGeometry(QtCore.QRect(180, 20, 82, 17))
self.radioButton.setText(_fromUtf8(""))
self.radioButton.setChecked(True)
self.radioButton.setAutoRepeat(False)
self.radioButton.setObjectName(_fromUtf8("radioButton"))
self.pushButton = QtGui.QPushButton(self.groupBox)
self.pushButton.setGeometry(QtCore.QRect(310, 50, 111, 31))
self.pushButton.setObjectName(_fromUtf8("pushButton"))
self.lineEdit_5 = QtGui.QLineEdit(self.groupBox)
self.lineEdit_5.setGeometry(QtCore.QRect(20, 50, 281, 31))
self.lineEdit_5.setObjectName(_fromUtf8("lineEdit_5"))
self.groupBox_4 = QtGui.QGroupBox(self.groupBox)
self.groupBox_4.setGeometry(QtCore.QRect(240, 110, 251, 101))
self.groupBox_4.setTitle(_fromUtf8(""))
self.groupBox_4.setObjectName(_fromUtf8("groupBox_4"))
self.label_8 = QtGui.QLabel(self.groupBox_4)
self.label_8.setGeometry(QtCore.QRect(100, 10, 49, 17))
self.label_8.setFrameShadow(QtGui.QFrame.Plain)
self.label_8.setObjectName(_fromUtf8("label_8"))
self.doubleSpinBox = QtGui.QDoubleSpinBox(self.groupBox_4)
self.doubleSpinBox.setGeometry(QtCore.QRect(30, 40, 62, 22))
self.doubleSpinBox.setObjectName(_fromUtf8("doubleSpinBox"))
self.pushButton_3 = QtGui.QPushButton(self.groupBox_4)
self.pushButton_3.setGeometry(QtCore.QRect(140, 40, 91, 21))
self.pushButton_3.setObjectName(_fromUtf8("pushButton_3"))
self.label_9 = QtGui.QLabel(self.groupBox_4)
self.label_9.setGeometry(QtCore.QRect(120, 70, 121, 20))
self.label_9.setObjectName(_fromUtf8("label_9"))
self.pushButton_5 = QtGui.QPushButton(self.groupBox)
self.pushButton_5.setGeometry(QtCore.QRect(430, 50, 51, 31))
self.pushButton_5.setObjectName(_fromUtf8("pushButton_5"))
self.groupBox_2 = QtGui.QGroupBox(self.centralwidget)
self.groupBox_2.setEnabled(True)
self.groupBox_2.setGeometry(QtCore.QRect(500, 10, 491, 211))
```

```
self.groupBox_2.setTitle(_fromUtf8(""))
self.groupBox_2.setObjectName(_fromUtf8("groupBox_2"))
self.label_2 = QtGui.QLabel(self.groupBox_2)
self.label_2.setGeometry(QtCore.QRect(200, 20, 131, 16))
self.label_2.setObjectName(_fromUtf8("label_2"))
self.label_3 = QtGui.QLabel(self.groupBox_2)
self.label_3.setGeometry(QtCore.QRect(110, 50, 91, 16))
self.label_3.setObjectName(_fromUtf8("label_3"))
self.label_4 = QtGui.QLabel(self.groupBox_2)
self.label_4.setGeometry(QtCore.QRect(110, 90, 91, 16))
self.label_4.setObjectName(_fromUtf8("label_4"))
self.label_5 = QtGui.QLabel(self.groupBox_2)
self.label_5.setGeometry(QtCore.QRect(110, 130, 91, 16))
self.label_5.setObjectName(_fromUtf8("label_5"))
self.lineEdit = QtGui.QLineEdit(self.groupBox_2)
self.lineEdit.setEnabled(False)
self.lineEdit.setGeometry(QtCore.QRect(205, 50, 216, 20))
self.lineEdit.setObjectName(_fromUtf8("lineEdit"))
self.lineEdit_2 = QtGui.QLineEdit(self.groupBox_2)
self.lineEdit_2.setEnabled(False)
self.lineEdit_2.setGeometry(QtCore.QRect(205, 90, 216, 20))
self.lineEdit_2.setObjectName(_fromUtf8("lineEdit_2"))
self.lineEdit_3 = QtGui.QLineEdit(self.groupBox_2)
self.lineEdit_3.setEnabled(False)
self.lineEdit_3.setGeometry(QtCore.QRect(205, 130, 216, 20))
self.lineEdit_3.setObjectName(_fromUtf8("lineEdit_3"))
self.btnSpecifications = QtGui.QRadioButton(self.groupBox_2)
self.btnSpecifications.setGeometry(QtCore.QRect(180, 20, 82, 17))
self.btnSpecifications.setText(_fromUtf8(""))
self.btnSpecifications.setObjectName(_fromUtf8("btnSpecifications"))
self.pushButton_2 = QtGui.QPushButton(self.groupBox_2)
self.pushButton_2.setEnabled(False)
self.pushButton_2.setGeometry(QtCore.QRect(180, 160, 131, 31))
self.pushButton_2.setObjectName(_fromUtf8("pushButton_2"))
self.pushButton_4 = QtGui.QPushButton(self.groupBox_2)
self.pushButton_4.setEnabled(False)
self.pushButton_4.setGeometry(QtCore.QRect(320, 160, 51, 31))
self.pushButton_4.setObjectName(_fromUtf8("pushButton_4"))
self.groupBox_3 = QtGui.QGroupBox(self.centralwidget)
self.groupBox_3.setGeometry(QtCore.QRect(10, 220, 981, 411))
self.groupBox_3.setTitle(_fromUtf8(""))
```

```
        self.groupBox_3.setObjectName(_fromUtf8("groupBox_3"))
        self.label_7 = QtGui.QLabel(self.groupBox_3)
        self.label_7.setGeometry(QtCore.QRect(430, 0, 131, 31))
        self.label_7.setObjectName(_fromUtf8("label_7"))
        self.checkBox_3 = QtGui.QCheckBox(self.groupBox_3)
        self.checkBox_3.setGeometry(QtCore.QRect(390, 380, 181, 17))
        self.checkBox_3.setObjectName(_fromUtf8("checkBox_3"))
        self.checkBox_4 = QtGui.QCheckBox(self.groupBox_3)
        self.checkBox_4.setGeometry(QtCore.QRect(550, 380, 281, 17))
        self.checkBox_4.setObjectName(_fromUtf8("checkBox_4"))
        self.graphicsView = QtGui.QGraphicsView(self.groupBox_3)
        self.graphicsView.setGeometry(QtCore.QRect(20, 30, 941, 171))
        self.graphicsView.setObjectName(_fromUtf8("graphicsView"))
        self.graphicsView_2 = QtGui.QGraphicsView(self.groupBox_3)
        self.graphicsView_2.setGeometry(QtCore.QRect(20, 200, 941, 171))
        self.graphicsView_2.setObjectName(_fromUtf8("graphicsView_2"))
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtGui.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 1011, 21))
        self.menubar.setObjectName(_fromUtf8("menubar"))
        MainWindow.setMenuBar(self.menubar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow", None))
        self.label.setText(_translate("MainWindow", "<html><head/><body><p
align=\"center\"><span style=\" font-size:10pt; font-weight:600;\">Impulse
response</span></p></body></html>", None))
        self.checkBox.setText(_translate("MainWindow", "Show original", None))
        self.checkBox_2.setText(_translate("MainWindow", "Show filtered", None))
        self.pushButton.setText(_translate("MainWindow", "Filter", None))
        self.label_8.setText(_translate("MainWindow",
"<html><head/><body><p><span style=\" font-size:10pt; font-
weight:600;\">Trigger</span></p></body></html>", None))
        self.pushButton_3.setText(_translate("MainWindow", "Freeze", None))
        self.label_9.setText(_translate("MainWindow", "<html><head/><body><p
align=\"center\"><span style=\" font-size:9pt; color:#f80d04;\">Ha salido del
bloqueo</span></p></body></html>", None))
        self.pushButton_5.setText(_translate("MainWindow", "Plot", None))
```

```
        self.label_2.setText(_translate("MainWindow",       "<html><head/><body><p
align=\"center\"><span    style=\"    font-size:10pt;    font-weight:600;\">Filter
Specifications</span></p></body></html>", None))
        self.label_3.setText(_translate("MainWindow", "Cutoff f(Hz):", None))
        self.label_4.setText(_translate("MainWindow", "W trans(Hz) :", None))
        self.label_5.setText(_translate("MainWindow", "A stop(dB)  :", None))
        self.pushButton_2.setText(_translate("MainWindow", "Filter", None))
        self.pushButton_4.setText(_translate("MainWindow", "Plot", None))
        self.label_7.setText(_translate("MainWindow",       "<html><head/><body><p
align=\"center\"><span    style=\"    font-size:12pt;    font-weight:600;\">Signal
Plot</span></p></body></html>", None))
        self.checkBox_3.setText(_translate("MainWindow", "Show time", None))
        self.checkBox_4.setText(_translate("MainWindow", "Show frequency", None))
```

parse

```
class Parse(object):

    @staticmethod
    def parseQStringToList(qstring):
        qstring = str(qstring)
        qstring = qstring[1:-1]
        qstring = qstring.split(",")
        qstring = list(qstring)
        qstring = map(float,qstring)

        return qstring


    @staticmethod
    def parseQStringToFloat(qstring):
        qstring = str(qstring)
        qstring = float(qstring)

        return qstring
```

plotter

```
import numpy as np
from filtro import Filtro
import pyqtgraph as pg
import math
class Plotter(object):

    def __init__(self):
        self.first = True
        self.lastTriggerValue = 0

    #Refrescar grafica
    def
update(self,curves,ps,MY_RATE,MY_SIZE,filtrar,coeficientes,auxi_ori,auxi_filtered
,auxi_tigger,trigger_value,label,tipo_filtrado):
        p6 = ps[0]
        data = self.sampler.getLastSample()
        label.setObjectName('label_t')

        if(self.first):
            self.first = False
            self.lastTriggerValue = trigger_value

        if(self.lastTriggerValue!=trigger_value):
            self.sampler.setTiggerValue(trigger_value)
            self.lastTriggerValue = trigger_value

        if(self.lastTriggerValue>max(data)):
            label.setText('Armed')
            label.setStyleSheet('QLabel#label_t {color: red}')
        elif(auxi_tigger):
            label.setText('Not triggered')
            label.setStyleSheet('QLabel#label_t {color: black}')
        else:
            label.setText('Triggered')
            label.setStyleSheet('QLabel#label_t {color: black}')

        curve = curves[0]
        xData = np.arange(MY_SIZE)/float(MY_RATE)

        if(auxi_ori):
            if(auxi_tigger):
```

```
                curve.setData(x=xData,y=None, pen='g')
                p6.enableAutoRange('xy', False)
            else:
                curve.setData(x=xData,y=data, pen='g')
                p6.enableAutoRange('xy', True)
        else:
            curve.setData(x=xData,y=data, pen=pg.mkPen(width=0.001, color='g'))
            p6.enableAutoRange('xy', True)



        curve2 = curves[1]
        p7 = ps[1]
        xf,yf = self.calcularFFT(MY_SIZE,MY_RATE,np.asarray(data))

        if(auxi_ori):
            if(auxi_tigger):
                curve2.setData(x=xf,y=None, pen='g')
                p7.enableAutoRange('xy', False)
            else:
                curve2.setData(x=xf,y=yf, pen='g')
                p7.enableAutoRange('xy', True)
        else:
            curve2.setData(x=xf,y=yf, pen=pg.mkPen(width=0.001, color='g'))
            p7.enableAutoRange('xy', True)

        if(filtrar):
            filtered = Filtro.filtrar(coeficientes,data,tipo_filtrado)
            curve3 = curves[2]
            if(auxi_filtered):
                if(auxi_tigger):
                    curve3.setData(x=xData,y=None,pen='r')
                else:
                    curve3.setData(x=xData,y=filtered,pen='r')
            else:
                curve3.setData(x=xData,y=filtered,pen=pg.mkPen(width=0.001,
color='r'))

            filtered_freq = self.calcularFFT(MY_SIZE,MY_RATE,filtered)
            curve4 = curves[3]

            if(auxi_filtered):
```

```
                if(auxi_tigger):
                    curve4.setData(x=xf,y=None,pen='r')
                else:
                    curve4.setData(x=xf,y=filtered_freq[1],pen='r')
            else:
                curve4.setData(x=xf,y=filtered_freq[1],pen=pg.mkPen(width=0.001,
color='r'))

    #Puntero a UI
    def setUI(self,UIpointer):
        self.UIpointer = UIpointer

    #Puntero hacia sampler
    def setSampler(self,sampler):
        self.sampler = sampler

    #FFT
    def calcularFFT(self,MY_SIZE,MY_RATE,y):
        T = 1/float(MY_RATE)
        yf = np.fft.fft(y,MY_SIZE)
        yf = 1.0/(MY_SIZE)*np.abs(yf[:MY_SIZE//2])
        xf = np.linspace(0.0, 1.0/(2.0*T), MY_SIZE/2) #Llega hasta fs/2 -> mas
seria aliasing

        return xf,yf

    def calcularFFTCoeficientes(self,MY_SIZE,MY_RATE,h,num):
        T = 1/float(MY_RATE)
        yf = np.fft.fft(h,MY_SIZE)
        yf = np.abs(yf[:MY_SIZE//2])
        primero = yf[0]
        if (num==4):
            for n in range(len(yf)):
                yf[n] = yf[n]/primero
        xf = np.linspace(0.0, 1.0/(2.0*T), MY_SIZE/2)

        return xf,yf

    def v_to_dB(self,yf):
        hn = []
        for n in range(len(yf)):
```

```
            hn.append(20*(math.log10(yf[n])))
        hn = np.asarray(hn)
        return hn
```

sampler

```
from bitlib import *
from PyQt4 import QtCore
import numpy as np

class Sampler(object):

    def __init__(self,MY_SIZE):
        self.first = True
        self.ch1_data = np.zeros(MY_SIZE)
        self.MY_RATE = 1000000
        self.MY_SIZE = MY_SIZE
        self.TRIGGER_VALUE = 0

    #Abrir bitscope
    def open_scope(self):

        print "Attempting to open the BitScope"

        if BL_Open("", 1):

            print "Bitscope is opened"

            MY_DEVICE = 0 # one open device only
            MY_CHANNEL = 0 # channel to capture and display
            MY_PROBE_FILE = "" # default probe file if unspecified
            MY_MODE = BL_MODE_FAST # preferred trace mode
            TRUE = 1

            MODES = ("FAST","DUAL","MIXED","LOGIC","STREAM")
            SOURCES = ("POD","BNC","X10","X20","X50","ALT","GND")

            print " Library: %s (%s)" % (
            BL_Version(BL_VERSION_LIBRARY),
```

```
            BL_Version(BL_VERSION_BINDING))

            BL_Select(BL_SELECT_DEVICE,MY_DEVICE)

            print "    Link: %s" % BL_Name(0)
            print "BitScope: %s (%s)" % (BL_Version(BL_VERSION_DEVICE),BL_ID())
            print "Channels: %d (%d analog + %d logic)" % (
            BL_Count(BL_COUNT_ANALOG)+BL_Count(BL_COUNT_LOGIC),
            BL_Count(BL_COUNT_ANALOG),BL_Count(BL_COUNT_LOGIC))

            print "   Modes:" + "".join(["%s" % (
                (" " + MODES[i]) if i == BL_Mode(i) else "") for i in
range(len(MODES))])

            BL_Mode(BL_MODE_LOGIC) == BL_MODE_LOGIC or BL_Mode(BL_MODE_FAST)

            BL_Range(BL_Count(BL_COUNT_RANGE));
            if BL_Offset(-1000) != BL_Offset(1000):
                print "   Offset: %+.4gV to %+.4gV" % (      BL_Offset(1000),
BL_Offset(-1000))

            for i in range(len(SOURCES)):
                if i == BL_Select(2,i):
                    print "        %s: " % SOURCES[i] + " ".join(["%5.2fV" %
BL_Range(n) for n in range(BL_Count(3)-1,-1,-1)])

            BL_Mode(MY_MODE) # prefered trace mode
            BL_Intro(BL_ZERO); # optional, default BL_ZERO
            BL_Delay(BL_ZERO); # optional, default BL_ZERO
            BL_Rate(self.MY_RATE); # optional, default BL_MAX_RATE
            BL_Size(self.MY_SIZE); # optional default BL_MAX_SIZE
            BL_Select(BL_SELECT_CHANNEL,MY_CHANNEL); # choose the channel
            BL_Trigger(self.TRIGGER_VALUE,BL_TRIG_RISE);    #    optional    when
untriggered */
            BL_Select(BL_SELECT_SOURCE,BL_SOURCE_POD); # use the POD input */
            BL_Range(BL_Count(BL_COUNT_RANGE)); # maximum range
            BL_Offset(BL_ZERO); # optional, default 0
            BL_Enable(TRUE); # at least one channel must be initialised
            BL_Trace()

            print "Complete: trace and acquisition complete. Dump the log...\n"
```

```
        return


    #Cerrar bitscope
    def close_scope(self):

        BL_Close()
        print "Bitscope is closed"

        return



    #Coger muestras
    def scope_acquire(self):

        BL_Trigger(self.TRIGGER_VALUE,BL_TRIG_RISE);
        BL_Trace(0.01, False)

        self.ch1_data = BL_Acquire()



    #Coger array de las ultimas muestras guardadas
    def getLastSample(self):
        return np.asarray(self.ch1_data)

    #Puntero a UI
    def setUI(self,UIpointer):
        self.UIpointer = UIpointer

    #Inicializar timer en hilo secundario -> coger muestras
    def startAcquire(self):
        self.timer = QtCore.QTimer()
        self.timer.timeout.connect(lambda: self.scope_acquire())
        self.timer.start(50)

    #Cambiar numero de puntos
    def setSize(self,SIZE):
        self.MY_SIZE = SIZE
```

```
#Cambiar frecuencia de muestreo
def setRate(self,rate):
    self.MY_RATE = rate

#Devuelve la frecuencia de muestreo
def getRate(self):
    return self.MY_RATE

#Cambiar nivel del trigger
def setTiggerValue(self, triggerValue):
    self.TRIGGER_VALUE = float(triggerValue)

#Devuelve el nivel del trigger
def getTiggerValue(self):
    return self.TRIGGER_VALUE
```