MEMORIA FINAL

Implementación de técnicas mejoradas de comunicaciones digitales aplicadas a docencia.

Autor: Lucas Novales Peleato
Director: Javier Matanza Domingo
Co-Director: Carlos García de la Cueva

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

Implementación de técnicas mejoradas de comunicaciones digitales aplicadas a la docencia en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2020/21 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.: Lucas Novales Peleato Fecha: 10/07/2021

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Javier Matanza Domingo Fecha: 10/07/2021

MATANZA DOMINGO JAVIER - 48543978 V

Firmado digitalmente por MATANZA DOMINGO JAVIER - 48543978V
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-48543978V, givenName=JAVIER, sn=MATANZA DOMINGO, cn=MATANZA DOMINGO JAVIER - 48543978V
Fecha: 2021.02.15 13:19:52 +01'00'

Fdo.: Carlos García de la Cueva Fecha: 10/07/2021

MEMORIA FINAL

Implementación de técnicas mejoradas de comunicaciones digitales aplicadas a docencia.

Autor: Lucas Novales Peleato
Director: Javier Matanza Domingo
Co-Director: Carlos García de la Cueva

Agradecimientos

En primer lugar, agradecer a mis directores de proyecto por la gran ayuda que me han brindado a lo largo del grado en general y durante el proyecto en particular.
A mis amigos, por haberme acompañado durante esta bonita etapa y haberme apoyado en mis peores momentos.
A mi familia, por haberme dado la oportunidad de formarme en una gran escuela de ingeniería y haberme ofrecido una educación de calidad.
Por último, agradecer a todos aquellos profesores que me han demostrado su pasión por la docencia. Si no fuese por vosotros, no me habría convertido en la persona que soy.
Gracias de corazón a todos vosotros.

Implementación de técnicas mejoradas de comunicaciones digitales aplicadas a la docencia.

Autor: Novales Peleato, Lucas

Director: Matanza Domingo, Javier

Co-Director: García de la Cueva, Carlos

Índice

Índice de figuras

1 Resumen del proyecto

Durante la formación como ingenieros de telecomunicaciones se realizan una gran cantidad de prácticas en los laboratorios para asimilar la teoría impartida durante el curso académico. Estas prácticas de laboratorio suelen componerse por una parte de simulación y una parte correspondiente a la implementación del diseño simulado. Por este motivo, desde el departamento de Señales y Sistemas, se desea establecer un nuevo dispositivo compuesto por hardware real con el fin de desarrollar nuevas prácticas de laboratorio.

En primer lugar, se ha llevado a cabo la búsqueda de un dispositivo adecuado para programar las nuevas prácticas. Para ello, se ha realizado una comparación de los distintos dispositivos existentes en el mercado, tanto en términos económicos como en versatilidad para su uso en distintas asignaturas de los planes de estudio actuales.

Finalmente, el dispositivo seleccionado para la realización de este proyecto fue la radio ADALM-PLUTO de Analog-Devices. Un dispositivo didáctico de Radio Definida por Software que proporciona un sistema de comunicaciones completamente integrado en el chip “AD9163”.

Al trabajar con esta tarjeta, se le proporcionará al alumno una visión más completa de la realidad en cuanto a un sistema de comunicaciones, así como un conocimiento adicional de la interacción con hardware mediante lenguajes de programación de alto nivel. Además, los alumnos asimilarán conceptos nuevos relacionados con el área de la Radio Definida por Software, un campo en pleno auge en el mundo de las telecomunicaciones. Por otro lado, se fomentará el aprendizaje de las aplicaciones reales de los distintos conceptos estudiados durante el grado en el área de teoría de la señal y comunicaciones.

Una vez elegido el dispositivo hardware, se valoraron los distintos lenguajes de programación que permitían la interacción con el mismo. Finalmente, se eligió Python para la realización del proyecto, con el fin de aprender un nuevo lenguaje de programación que no se ha estudiado a fondo durante el plan de estudios.

Para analizar las posibilidades que brinda nuestro dispositivo, se han llevado a cabo una serie de prácticas de dificultad progresiva. Estas prácticas abordan situaciones tan simples como la transmisión de tonos a través del transmisor de la ADALM-PLUTO y llegan hasta la implementación de algoritmos de sincronismo pertenecientes a un receptor BPSK.

Para ello se han desarrollado distintas subrutinas (códigos escritos en Python o Matlab) donde se muestre la configuración de los distintos elementos HW, así como los algoritmos de procesado de señal aplicados sobre las muestras recibidas en banda base.

El acceso a la configuración de bajo nivel de la radio se realiza a través de las librerías proporcionadas por el fabricante del dispositivo (''Analog Devices''), ofreciendo una capa de abstracción que brinda al alumno la oportunidad de configurar y utilizar cada uno de los componentes de un sistema de comunicaciones real, con un nivel de dificultad moderado.

Gran parte del análisis de las transmisiones realizadas por los alumnos, se pueden realizar utilizando el software IIO Oscilloscope, proporcionado por el fabricante del dispositivo. Utilizando dicho software, el alumno se familiarizará con la visualización en tiempo de real los datos procesados por un sistema de comunicaciones y sus diferentes maneras de ser representados, como las constelaciones, espectros y muestras en el dominio temporal. En la siguiente figura se muestra el espectro proporcionado por IIO Oscilloscope de una señal de audio modulada en amplitud y transmitida por la ADALM-PLUTO, centrada en una frecuencia digitan con valor de 0,25.

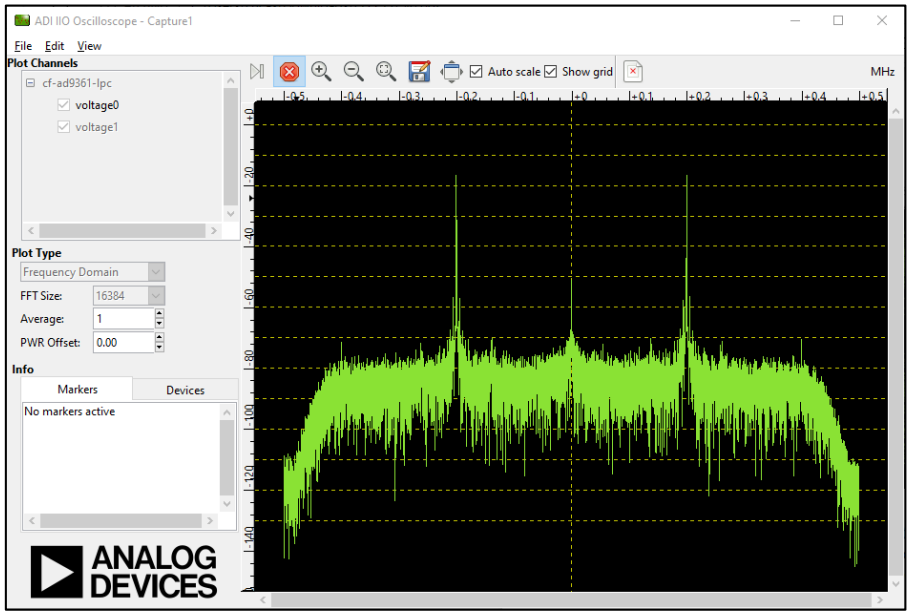


Figura 1. Espectro de una señal modulada en amplitud

Para asimilar los conceptos estudiados en las asignaturas relacionadas con el procesado de la señal y llevar a cabo su correspondiente implementación real, se han desarrollado las siguientes prácticas.

- Transmisión de señales sencillas como exponenciales complejas.
- Implementación de un modulador AM mediante técnicas digitales
- Visualización de los datos transmitidos por la ADALM-PLUTO
- Generación de forma de onda PSK
- Implementación de algoritmos de sincronismo
- Evaluación de los filtros FIR configurables en la ADALM-PLUTO

El desarrollo del proyecto se ha dividido en cuatro bloques principales que se explican a continuación.

En primer lugar, se ha analizado en detalle tanto la cadena de transmisión como la cadena de recepción de la radio ADALM-PLUTO, prestando atención a las funcionalidades y márgenes de operación de cada uno de sus componentes, por ejemplo:

- Rango de frecuencia de muestreo permitida
- Número de bits de los ADC
- Ancho de banda
- Rango de frecuencias para transmitir
- Número de coeficientes máximo de los filtros FIR programables

En segundo lugar, se ha llevado a cabo un aprendizaje del lenguaje de programación Python con dos principales objetivos:

- Familiarizarse con las librerías y elementos Python asociados al procesado digital de la señal
- Analizar las clases Python correspondientes a la librería *"libiio"*
- Aprender a interactuar con hardware real mediante las APIS de acceso propuestas por el fabricante del dispositivo y realizar las primeras prácticas

Una vez aprendidos los conceptos básicos para utilizar la tarjeta, se realizó un análisis de las técnicas digitales necesarias para la implementación de una modulación en amplitud y una modulación digital en fase, con el fin de obtener un modelo de transmisor y receptor para cada uno de los tipos de modulación.

Por último, el reto más complicado del trabajo, y a su vez, la mayor toma de contacto con la realidad de las comunicaciones ha consistido en adentrarse en el campo de los algoritmos de sincronismo, típicos en cualquier receptor digital.

En una modulación digital de fase, los símbolos transmitidos se ven afectados por el nivel de ruido, el retardo de propagación y los desajustes de frecuencia entre transmisor y receptor. Estos factores empeoran la calidad de la señal recibida y generan errores en la recepción de los datos.

Para recomponer la información transmitida, será necesario procesar las muestras recibidas a lo largo de unos bloques destinados al sincronismo de portadora, y el ajuste del instante óptimo de muestreo de los símbolos recibidos.

En la siguiente figura se muestra el diagrama del modem BPSK desarrollado a lo largo del proyecto.

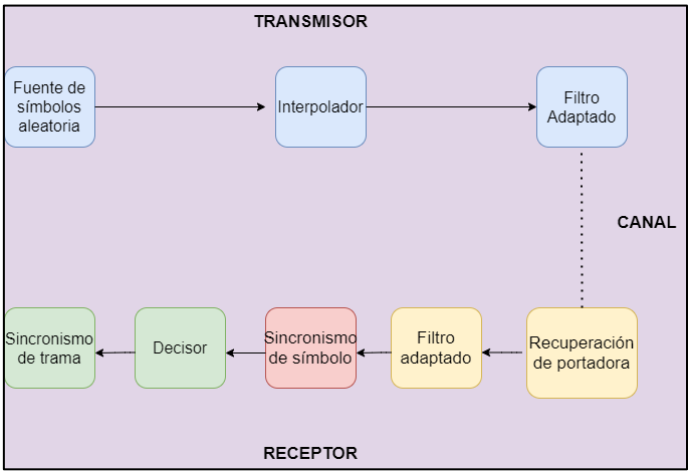


Figura 2. Diagrama de bloques de un modulador MPSK

Además, el chip "AD9163" incorpora un filtro FIR programable de 128 coeficientes tanto en la cadena de transmisión como en la de recepción. Una vez realizadas las primeras prácticas e implementadas las modulaciones AM y BPSK, se ha llevado a cabo un análisis del comportamiento de dichos filtros. Para ello, se han implementado distintos filtros FIR mediante el método de Parks-McClellan.

Con los programables configurados correctamente, se ha llevado a cabo un barrido espectral transmitiendo una exponencial compleja en un proceso iterativo, donde se varia el valor de la frecuencia digital asociada a la exponencial en cada iteración con el fin de cubrir el ancho de banda de interés.

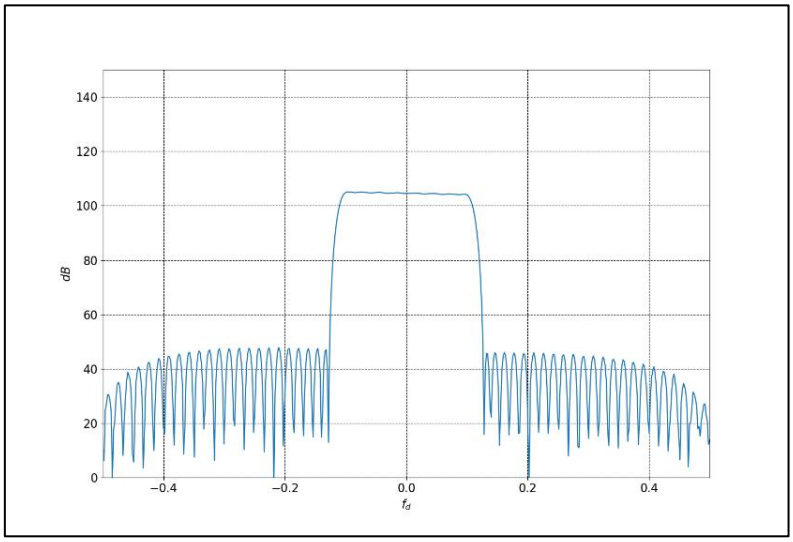


Figura 3. Respuesta en frecuencia de los filtros embebidos

2 Abstract

We have carried out a large number of practical exercises in laboratories to assimilate the theory taught during the academic year during our training as telecommunications engineers. These laboratory practices usually consist of a simulation part, and another part corresponding to the implementation of the simulated design. That is why, from the Signals and Systems Department, we want to establish a new device composed of real hardware in order to develop new laboratory practices.

The first work consisted in finding a suitable device to carry out the new practices. For this purpose, an analysis of the different didactic devices combining interaction with hardware, by means of a programming language with Software Defined Radio, was carried out.

Finally, the device chosen for the realization of this project was the Analog-Devices ADALM-PLUTO radio, a Software Defined Radio educational device that provides a fully integrated communication system in the "AD9363" chip.

Working with this board will provide students with a more complete vision of the reality as a communications system, as well as an additional knowledge of the interaction with hardware through programming languages. In addition, new concepts related to the area of Software Defined Radio, a booming field in the world of telecommunications, may be learned with this device. On the other hand, it will help students to understand the real applications of different concepts studied during the undergraduate in general, and in the field of digital signal processing in particular.

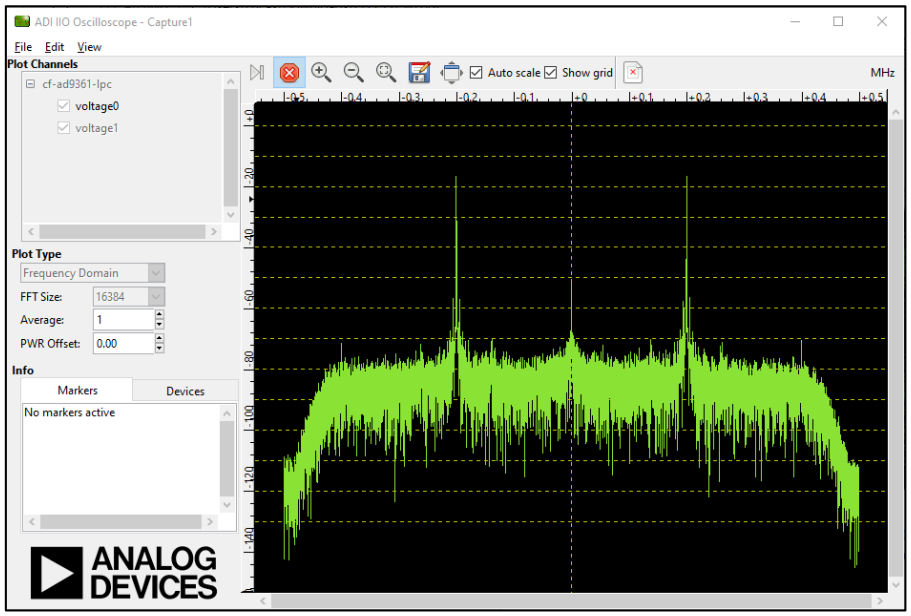
Once the device for the project was chosen, the different programming languages alloweing the interaction with the hardware of the card were studied. Finally, Python was chosen for the realization of the project, in order to learn a new programming language that has not been studied in depth during the undergraduate.

In order to analyze the possibilities offered by our device, a series of practices will be carried out, progressively increasing in difficulty. These practices will address situations such as the transmission of tones through the transmitter of the ADALM-PLUTO and will reach its maximum level with the implementation of synchronization algorithms belonging to a BPSK receiver.

For this purpose, different subroutines (codes written in Python or Matlab) will be developed to show the configuration of the different HW elements, as well as the signal processing algorithms applied on the samples received in baseband.

To access the low level configuration of the radio, the libraries provided by the manufacturer of the device (''Analog Devices'') will be used, offering an abstraction layer that will give the student the opportunity to configure and use each of the components of a real communication system, with a moderate level of difficulty.

Much of the analysis of the transmissions performed by the students will be carried out using the IIO Oscilloscope software provided by the device manufacturer. Using this software, the student will become familiar with the real-time visualization of the data processed by a communication system and its different ways of being represented, such as constellations or spectra in decibel scale. The following figure shows the spectrum provided by IIO Oscilloscope of an amplitude modulated audio signal transmitted by the ADALM-PLUTO.



Spectrum of an amplitude-modulated signal

In order to assimilate the concepts studied in the subjects related to signal processing and to carry out their corresponding real implementation, the following practices have been developed.

- Transmission of simple signals as complex exponential signals.
- Implementation of an AM modulator using digital techniques.
- Visualization of the data transmitted by the ADALM-PLUTO.
- Generation of PSK modulator
- Implementation of synchronization algorithms
- Evaluation of the configurable FIR filters in the ADALM-PLUTO.

The development of all of the above has been divided into four main blocks, which are explained below.Firstly, both the transmission chain and the reception chain of our device were completely analyzed, paying attention to the functionalities and limitations of each of its components, for example:

- Allowed sampling frequency range
- Number of ADC bits
- Bandwidth Range of frequencies to transmit

- Number of maximum coefficients of the programmable FIR filters.

Secondly, a learning of the Python programming language has been carried out with two main objectives:

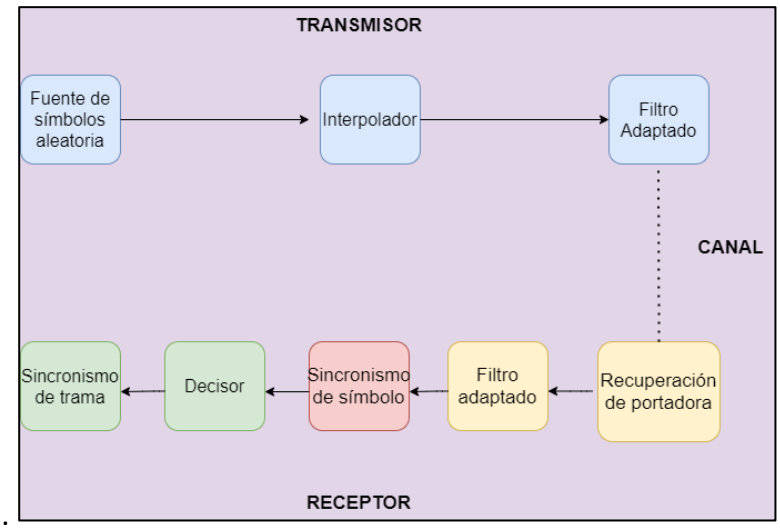
- To become familiar with the Python libraries and elements associated with digital signal processing.
- To analyze the Python classes corresponding to the "libiio" library.
- To learn how to interact with real hardware by means of the access Apis proposed by the device manufacturer and to perform the first practices.

Once the basic concepts for using the card have been learned, an analysis of the digital techniques necessary for the implementation of an amplitude modulation and a phase modulation was carried out, in order to obtain a transmitter and receiver model for each of the modulation types.

Finally, the most complicated challenge of this work, and at the same time, the most important contact with the reality of communications, was touching base with the field of synchronization algorithms and PLL theory.

When performing a phase modulation, the transmitted symbols are affected by the noise level and, in addition, they suffer from frequency deviations and time delays. These factors worsen the quality of the received signal and generate errors in transmission.

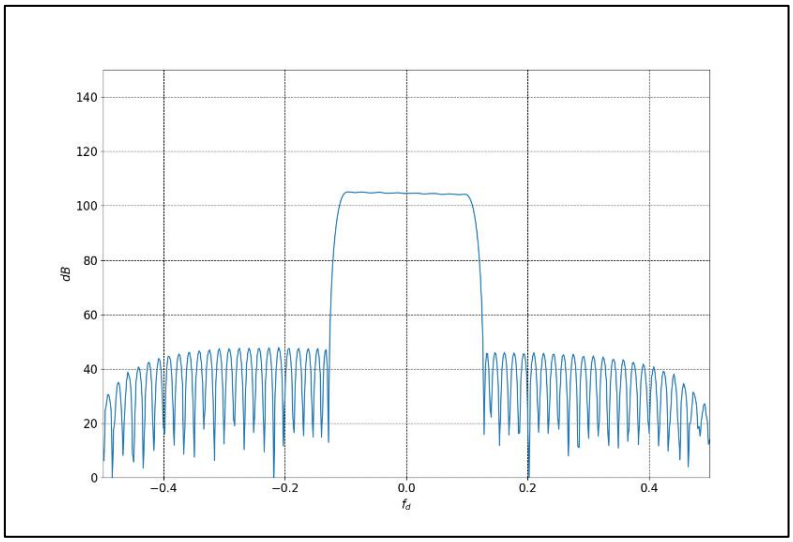
To remedy these phenomena, it will be necessary to process the received signal through synchronization blocks in order to recover the original symbols from the received ones. The following figure shows the diagram of the BPSK modulator developed throughout the project.



Block diagram of an MPSK modulator

In addition, the ADALM-PLUTO incorporates a user-programmable FIR filter in both the transmission and reception chains. Once the first practices have been carried out and the AM and BPSK modulations have been implemented, an analysis of the behavior of these filters has been performed. For this purpose, different FIR filters have been implemented by means of the optimal Remez algorithm and using the Parks-McClellan design formula.

With the ADALM-PLUTO filters correctly configured, a spectral sweep was made by transmitting a complex exponential in an iterative process, where the value of the frequency associated to the exponential was varied in each iteration.



Frequency response of the embedded filters

3 Estado de la cuestión y descripción de las tecnologías

La motivación de este proyecto surge para conocer con mayor profundidad la parte práctica de la rama de señales y sistemas de la ingeniería de telecomunicaciones.

Para ello, fue necesaria la búsqueda de un dispositivo que llevase incorporado un sistema de comunicaciones completo y permitiera al alumno familiarizarse con los distintos parámetros de cada uno de los componentes que lo forman.

En la siguiente tabla se muestra información sobre las distintas alternativas analizadas para realizar el proyecto.

3.1 Estudio de mercado

Modelo	**Redpitaya**	**Hack-RF**	**Ettus B200 Mini**	**Adalm-Pluto**
Precio	350 €	250 €	886 €	125 €
Transceptor	Sin transceptor (Recibe en Banda Base)	1MHz-6GHz	70MHz-6GHz (56 MHz BW máximo)	70MHz-6GHz (56 MHz BW máximo)
Nº ADCs	2	1	2	2
Nº DACs	2	1	2	2
Nº bits del DAC/ADC	14	8	12	12
Diseño HW/SW Open Source	Si	Si	Si	Si
Lenguajes de programación	Matlab, Labview, Python Scilab	C/C++, Python GNU Radio	C/C++, Python, GNU Radio	Matlab, Python, GNU Radio, C/C++
Unidad de Procesamiento	Zynq 7010 (FPGA+2 Cores ARM)	NXP LPC43xx (Procesador)	Xilinx Spartan-6 XC6SLX75 (FPGA)	Zynq 7010 (FPGA+2 Cores ARM)

Tabla 1 Estudio de mercado

Entre todas las alternativas mostradas en la tabla anterior, destacamos la radio ADALM-PLUTO debido a su carácter didáctico y a su sencillez a la hora de configurar el Hardware, ya que el fabricante del dispositivo ofrece una serie de Apis de acceso a los elementos de la tarjeta, ofreciendo una capa de abstracción y permitiendo al alumno configurar los componentes que intervienen durante la comunicación.

A pesar de ser un dispositivo con carácter didáctico, la ADALM-PLUTO presenta un gran potencial y ha sido utilizada en una gran cantidad de proyectos de investigación, por ejemplo:

- *Ilia Iliev and Ivaylo Nachev "An Automatic System for Antenna Radiation Pattern Measurement".[1]*
  Donde se utilizó la ADALM-PLUTO como transmisor con el objetivo de implementar un sistema automático capaz medir de la radiación emitida por las antenas.
- *"Algorithm and Software for Intelligent Analysis of the Frequency Spectrum for Cognitive Radio Systems".*
  En el proyecto de investigación anterior, la ADALM-PLUTO fue utilizada como fuente de datos portátil e independiente con el fin de utilizar un análisis del espectro en sistemas de radio cognitivos. [2]
- *Chirstopher Gravelle, Ruolin Shou. "SDR Demonstration of Signal Classification in Real-Time Using Deep Learning". IEEE Globecom Workshops, 2019.[3*]
  Proyecto de investigación que demuestra la viabilidad de reconocer y clasificar distintos tipos de señales inalámbricas mediante el uso de MathWorks y Pluto SDR.

- *Yacine Adane. "A Smart Digital Software Radio Transceiver Design Concept for UAV and Autonomous Vehicles Application". Advances in Science and Engineering Technology International Conferences, 2019.[4]*
  Realizando una evaluación del concepto de los Smart SDT (Smart Distribution Systems) mediante el uso de la ADALM-PLUTO, debido a la gran similitud de su arquitectura con el concepto de SDT.
- *Ángel E. Ruiz-García et. al. "SDR-Based Channel Emulator for vehicular Communications". IEEE Colombian Conference on Communications and Computing, 2019.[5]*
  Diseñando un emulador de canal combinando el tratamiento de datos mediante Simulink y Matlab con la realización de simulaciones con la ADALM-PLUTO.

Además, el chip incorporado en la ADALM-PLUTO ("AD9163"), presenta un equivalente con varios aspectos en común con el chip("AD9161"), también desarrollado por Analog Devices, calificado para el espacio y ampliamente utilizado en la industria actualmente.

Por todo lo comentado anteriormente, se decidió la elección de la ADALM-PLUTO como dispositivo de aprendizaje.

3.2 Lenguajes de programación

Una vez elegido el dispositivo para desarrollar el proyecto, se realizó un análisis de los distintos lenguajes de programación compatibles con el desarrollo del proyecto.

La radio Pluto SDR ofrece una gran variedad de lenguajes de programación que brindan al alumno la oportunidad de interactuar y configurar los distintos elementos hardware que la componen. Debido a las distintas posibilidades brindadas por el fabricante del dispositivo, se llevó a cabo un estudio de los distintos lenguajes compatibles con la tarjeta, prestando especial atención a las limitaciones y beneficios de cada uno de ellos. Como se ha comentado en el resumen, Analog Devices, presenta una serie de APIS de acceso a los componentes de la tarjeta con el fin de crear una capa de abstracción entre el hardware y el alumno. La utilización de dichas APIS de acceso variará en función del lenguaje de programación utilizado y ha sido uno de los principales factores a tener en cuenta para la elección del lenguaje utilizado.

- Matlab: Este lenguaje de programación es utilizado en un gran número de las asignaturas de nuestro grado por lo que la mayoría de los alumnos presenta nociones suficientes como para poder configurar nuestra radio en este lenguaje. Además, existen varios libros que combinan el uso de Radio Definida por Software con implementaciones sobre la tarjeta.
- Python: Lenguaje menos utilizado durante el grado, ya que solo ha sido cursado en una asignatura, aunque con un gran potencial. Este lenguaje presenta una sintaxis sencilla asociada al procesado digital de señales y nos permite acceder al Hardware de la ADALM-PLUTO a través de la librería IIO desarrollada por el fabricante
- C/C++: Lenguaje menos utilizado que Matlab por parte de los alumnos. Debido a estar fuertemente tipado, es un lenguaje ideal para aprender a programar. No obstante, no es tan utilizado en el área del procesado digital de la señal y la realización del proyecto sería más costosa.

Por todo ello, se ha elegido Python para desarrollar el proyecto, ya que presenta gran similitud con Matlab en lo referido al procesamiento digital de la señal y es un lenguaje que no se ha estudiado en menor profundidad durante el grado.

3.3 Elección de las APIS de acceso.

Tomando Python como lenguaje para desarrollar el proyecto, se examinaron las distintas APIS de acceso ofrecidas por Analog Devices para interactuar con el dispositivo. En la siguiente figura se muestran los distintos niveles a través de los cuales podemos acceder y configurar el hardware de la ADALM-PLUTO.

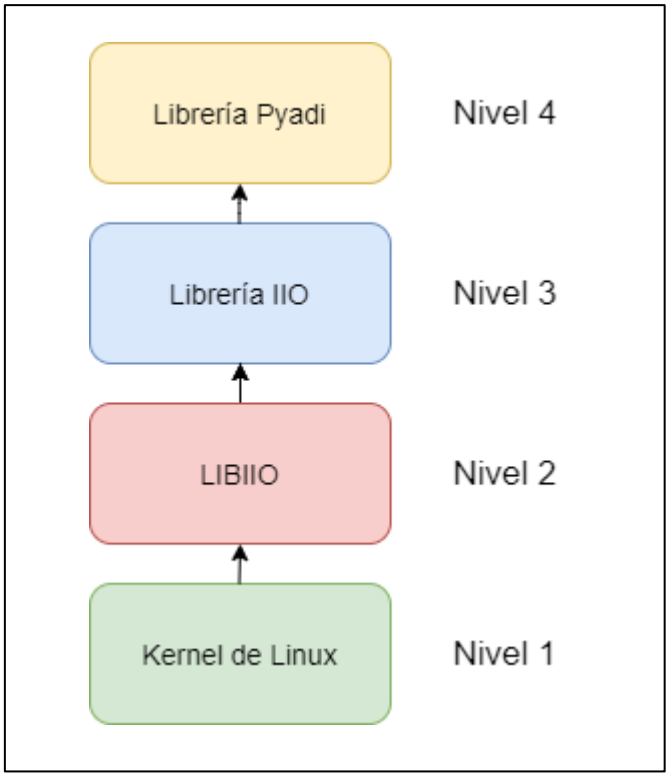


Figura 4. Interacción de las librerías con el Kernel de Linux

En los niveles 1 y 2, encontramos el Kernel de Linux, programado en C mediante un Framework llamado "Industrial Input Output" y la librería *libiio,* también programada en C y montada sobre el Kernel de Linux. A continuación, en los niveles 3 y 4, encontramos las distintas Apis de acceso ofrecidas por Analog Devices para interactuar con el hardware de la tarjeta.

El nivel 3, corresponde a la librería *iio* programada en Python, la cual interactúa con *libiio* accediendo a los componentes de la radio para realizar su configuración. Ofreciendo una capa de abstracción al usuario que facilitará el manejo del dispositivo.

Por último, encontramos el nivel 4, correspondiente a la librería *pyadi*. Esta librería también está programada en Python y ha sido creada con el fin de ofrecer una mayor capa de abstracción entre al usuario y el hardware de la tarjeta.

Una vez analizadas las distintas Apis de acceso ofrecidas por el fabricante, se han realizado distintas pruebas evaluando los beneficios y limitaciones de cada una de ellas.

La librería *pyadi*, utiliza llamadas a la librería *iio* para configurar los elementos de la tarjeta. Es por ello por lo que la capa de abstracción es mayor que si se configurase directamente desde *iio*, ya que el alumno no necesita conocer el funcionamiento de la librería que realmente interactúa con los niveles 1 y 2, provocando así una pérdida de conocimiento y aprendizaje de varios de los conceptos más importantes del proyecto.

Sin embargo, el uso de la librería *iio,* ofrecerá al alumno una visión más completa de la interacción con el hardware de la tarjeta. A pesar de poseer un mayor nivel de dificultad y una sintaxis más compleja, el uso de la librería *iio* brindará al alumno la oportunidad de analizar la interacción de Python con el hardware de la tarjeta, aportando así un conocimiento mucho mayor del dispositivo y una visión más completa de la realidad referida a un sistema de comunicaciones.

Debido a ello, se ha elegido la librería *iio* para realizar el proyecto y aprender manera de interactuar con el hardware de la tarjeta.

3.4 Técnicas de ayuda a la docencia en otras universidades

Por último, se ha realizado una valoración de distintas universidades que utilizan la radio ADALM-PLUTO o dispositivos similares con el fin de ayudar a los alumnos durante su periodo de aprendizaje.

Universidad Politécnica de Madrid: **“**Medida de la SINAD utilizando ADALM-PLUTO”.

Universidad de Granada: “Análisis de Sotware y Hardware del SDR HackRF One”

Universidad de Vigo: **“**Análisis de los parámetros teóricos de las señales de RF con el uso de SDR HackRF One”

Universidad Autónoma Metropolitana (México)**: “**A Software Development Based on Software-Defined Radio Devices for Transmitting Digital Signals”

4 Introducción

4.1 Introducción a ADALM-PLUTO e instalación previa

El sistema ADALM PLUTO incluye un puerto USB a modo de interfaz de comunicación física y alimentación. A través de esta interfaz, se realizará el intercambio de información entre el *"host*" y la placa utilizando el protocolo TCP/IP, encapsulado en la librería de control *"libiio"*

Para establecer esta comunicación a través de un puerto USB perteneciente al *"host",* es necesaria la instalación de una serie de drivers proporcionados por el fabricante, que variarán en función del sistema operativo utilizado por el usuario. Las instrucciones para la instalación de los drivers se encuentran detalladas en el siguiente enlace:

https://wiki.analog.com/university/tools/pluto/users.

Como se ha explicado en el estado de la cuestión, la interacción con los componentes de la ADALM PLUTO se realizan a través de llamadas a la librería *"libiio*", de Analog Devices, la cual está escrita en C/C++.

No obstante, el fabricante del dispositivo ofrece una serie de APis de acceso a libiio mediante otros lenguajes de programación como Python o Matlab. Durante el proyecto utilizaremos Python para interactuar con los componentes de la ADALM PLUTO.

A continuación, se detallan los enlaces para descargar e instalar la librería libiio:

https://wiki.analog.com/resources/tools-software/linux-software/libiio

Además, habrá que descargar la carpeta libiio del github de Analog Devices y guardarla en un directorio conocido:

https://github.com/analogdevicesinc/libiio

Una vez descargada la carpeta, será necesario llevar a cabo la instalación desde la línea de comandos. Deberemos descomprimir la carpeta descargada del github y extraerla en un directorio conocido.

Con la extracción de la carpeta completada, navegaremos desde la línea de comandos hasta el directorio donde se encuentre. Accederemos al directorio "*bindings"* y a continuación al directorio *"Python".*

Una vez en este directorio ejecutaremos el siguiente comando:

"python.setup.py.cmakein install"

En las siguientes imágenes, se muestra la navegación e instalación a través de la línea de comandos.

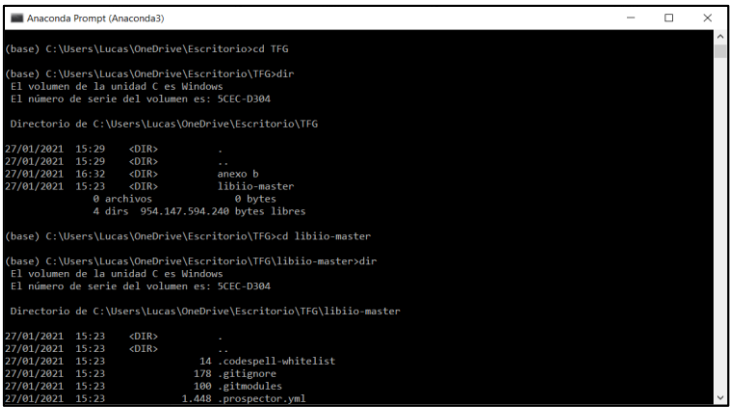


Figura 5. Acceso al directorio libiio-master

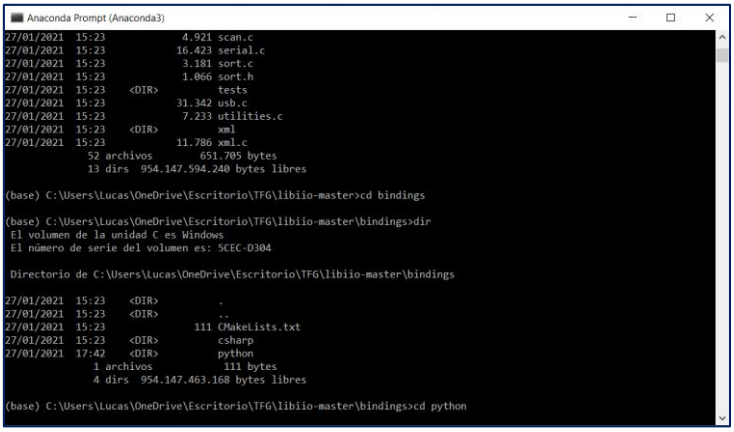


Figura 6. Acceso al directorio python

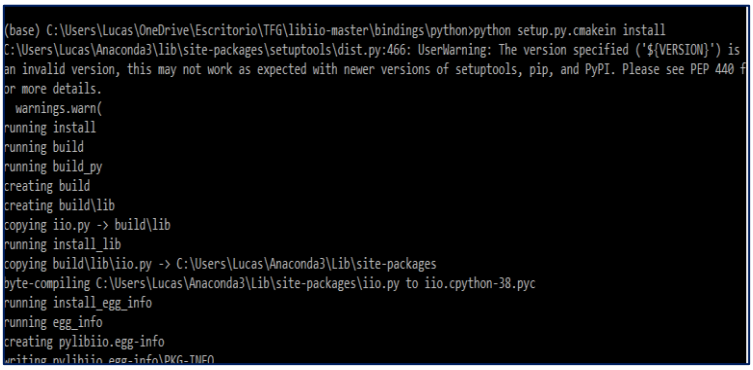


Figura 7. Ejecución del comando de instalación

Una vez terminada la instalación, accedemos a Spyder e introducimos la sentencia "*import iio"* desde la línea de comandos para comprobar que la instalación se ha realizado correctamente.

4.2 Análisis de la cadena de Trasmisión y Recepción de la placa ADALM-PLUTO.

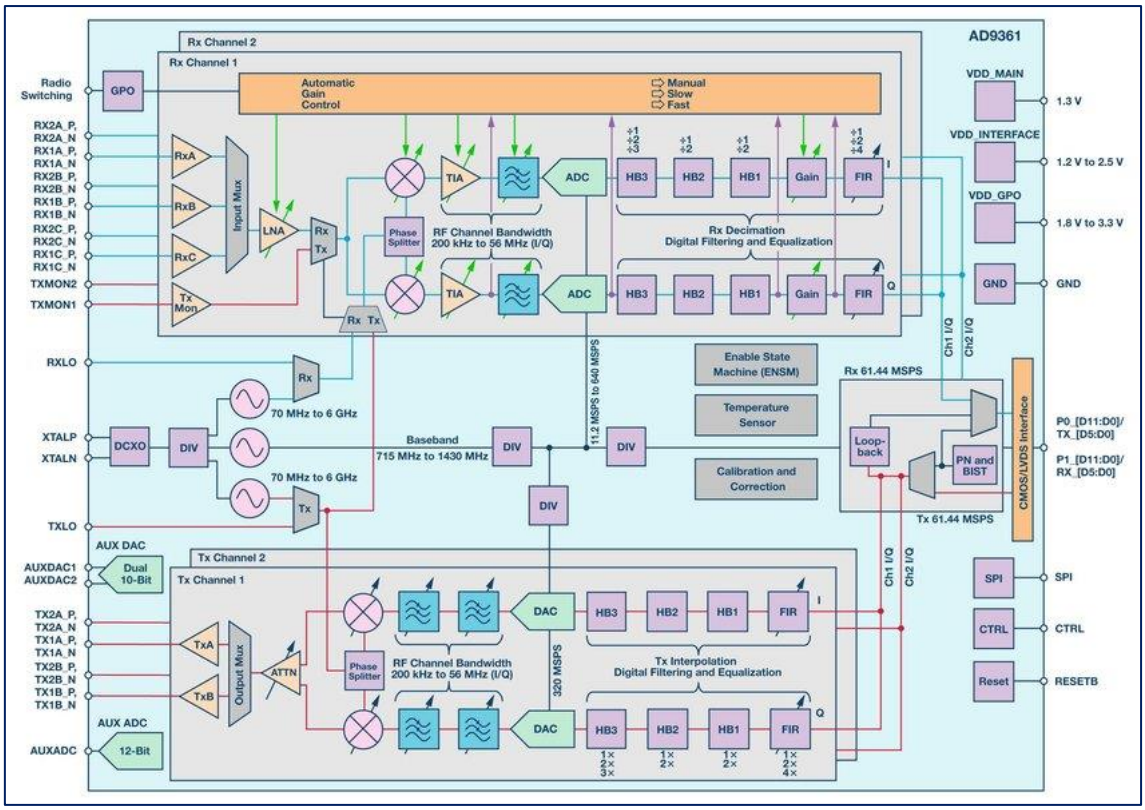


Figura 8. Diagrama de bloques de la ADALM-PLUTO

4.3 Cadena de recepción de la placa ADALM-PLUTO.

Como podemos observar en la figura 8, la señal que recibimos atraviesa un tipo de modulador muy común en este tipo de dispositivos.

Se trata de un modulador de I-Q que tiene como el objetivo de procesar, por un lado, la parte real y, por otro lado, la parte imaginaria de nuestra señal. Este modulador es muy común en este tipo de dispositivos.

El oscilador local posee un margen de variación comprendido entre 325 MHz y 3.8 GHz.

El primer bloque que encontramos en las ramas I-Q, se trata de un filtro paso bajo encargado de filtrar nuestra señal antes de atravesar los ADCs (convertidores analógico-digitales), los cuales están diseñados para ser utilizados con un número fijo de 12 bits y una frecuencia de muestreo comprendida entre 25 MHz y 640 MHz.

A la salida de los ADCs, nuestra señal ha sido convertida desde el dominio analógico al dominio digital para atravesar el resto de los bloques de la ADALM-PLUTO.

Una vez nuestra señal está digitalizada, atravesará una cadena de diezmadores acompañados de su filtro antialiassing, con el objetivo de disminuir la tasa de datos que finalmente será analizada.

A la salida de la cadena de diezmadores, encontramos un control de ganancia, el cual puede ser configurado, así como un filtro FIR configurable con la restricción de que su número de coeficientes ha de ser de 128 como máximo.

Una vez atravesada la cadena de recepción, los datos recogidos son enviados a una FPGA de control, incluida en la propia ADALM-PLUTO, que se encargará de empaquetar y trasladar los datos al usuario.

4.4 Cadena de transmisión de la placa ADALM-PLUTO.

La cadena de trasmisión ha de recorrerse en el sentido inverso al descrito en la cadena de recepción.

Las diferencias entre la cadena de recepción y transmisión de la ADALM-PLUTO son las siguientes:

1) En lugar de encontrar ADCs, encontramos DACs (conversores del dominio digital al dominio analógico)
2) En lugar de encontrar una cadena de diezmadores, encontramos una cadena de interpoladores. El motivo de estos interpoladores es que, a diferencia de la cadena de recepción, cuando estamos transmitiendo nos interesa aumentar la tasa de datos, por lo que nuestra señal en el dominio digital ha de ser interpolada en lugar de diezmada.

5 Introducción al procesado digital de señales con Python.

Como hemos descrito en el punto anterior, el fabricante del dispositivo ofrece una serie de APIS para poder interactuar con el Hardware de la placa mediante distintos lenguajes de programación.

El lenguaje elegido para llevar a cabo el proyecto ha sido Python, por lo que antes de realizar las primeras prácticas con la ADALM-PLUTO será necesario familiarizarse con el procesado digital de señales utilizando este lenguaje.

A lo largo de este apartado se explicarán las librerías, herramientas y recursos necesarios para procesar digitalmente las señales mediante Python.

5.1 Librería Matplotlib

Matplotlib es una librería de Python especializada en la creación de gráficos representados en dos dimensiones, permitiendo utilizar los gráficos más comunes como:

- Diagrama de barras
- Histogramas
- Diagramas de contorno
- Diagramas de áreas
- Diagramas de cajas

Debido a su sencilla sintaxis, esta librería será utilizada para representar gran parte de las figuras incluidas en el contenido de la memoria. En lo referido al procesamiento digital de la señal, principalmente se hará uso para representar las señales procesadas, tanto en el dominio frecuencial como en el temporal.

5.2 Librería NumPy

NumPy es una librería de Python que se caracteriza por los siguientes aspectos:

1) Permite la creación de vectores y matrices de grandes dimensiones.
2) Ofrece herramientas de computación numérica como generar números aleatorios o calcular transformadas de Fourier.
3) Librería del tipo Open Source, desarrollada y mantenida en github.
4) Gran flexibilidad
5) Sintaxis sencilla que permite accesibilidad para programadores de cualquier experiencia

Referido al procesamiento digital de señales, esta librería será principalmente utilizada para la inicializar vectores, crear señales digitales y calcular transformadas de Fourier.

5.3 Creación de una señal digital a partir de un vector de puntos utilizando NumPy.

En el siguiente ejemplo, observamos la creación de un coseno digital a partir de un vector de puntos denominado t. Para ello, utilizamos la función “arange” de la librería NumPy. Esta función recibe como dos primeros argumentos el principio y el final de nuestro vector y como tercer argumento la separación entre dos posiciones consecutivas. En este caso hemos creado un vector desde 0 a 20 con una separación de 0.01. Esta separación nos permitirá modificar la resolución de nuestra señal.

En las siguientes figuras se muestra la representación de una señal coseno a partir de un vector de 2000 y 20 muestras, respectivamente. Como podemos apreciar, cuanto menor sea la separación entre dos posiciones consecutivas de nuestro vector, mayor es la resolución de nuestra señal.

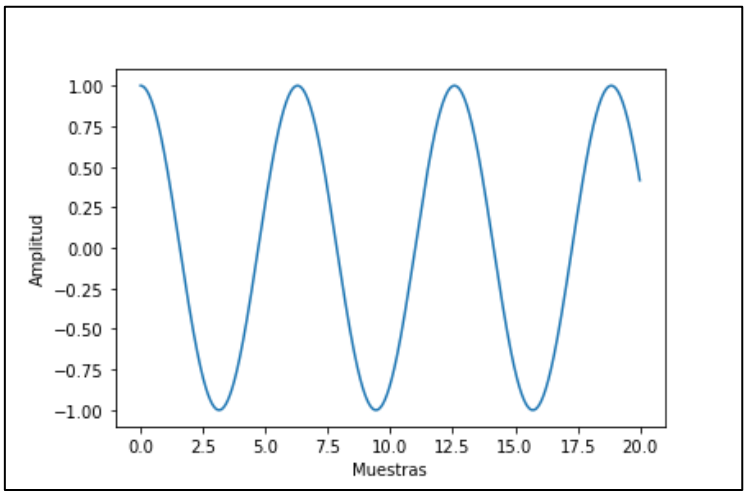


Figura 9. Representación de un coseno con 2000 muestras

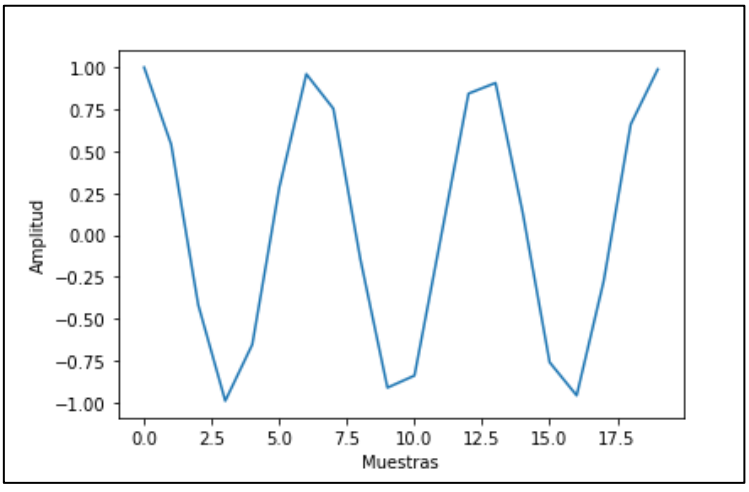


Figura 10. Representación de un coseno con 20 muestras

5.4 Transformada de Fourier utilizando NumPy.

Numpy nos permite realizar el cálculo de la transformada de Fourier de una manera realmente sencilla. En el siguiente código, utilizando una frecuencia de muestreo de 1 KHz, creamos un vector de puntos t para generar una exponencial compleja.

En este caso, hemos generado una exponencial de frecuencia digital de valor igual a 0.3, por lo que su espectro estará posicionado en 0,3. Para realizar el cálculo de nuestra transformada de Fourier, utilizamos la función *fft* de la librería "*NumPy*".

Cabe destacar la creación del vector de frecuencias denominado fp. Este vector, indicará el posicionamiento de nuestra señal al representarla gráficamente y deberá tener una longitud igual a la FFT obtenida. En este caso, al estar trabajando con frecuencia normalizada, nuestro vector estará comprendido entre -0,5 y +0,5 y tendrá una separación igual al inverso de la longitud, en el dominio temporal, de la señal a representar.

```
fs=1000
t=np.arange(0,10,1/fs)
frec=0.3
exponencial=np.exp((2j*np.pi*frec*t))
exponencial= exponencial/max(exponencial)
P=np.fft.fftshift(np.fft.fft(exponencial)/len(exponencial))
fp=np.arange(-0.5,0.5,1/len(exponencial))
fig=plt.figure()
plt.plot(fp,P)
plt.title("Espectro señal exponencial")
plt.xlabel("Frecuencia normalizada, w")
plt.ylabel("Voltios, V")
```

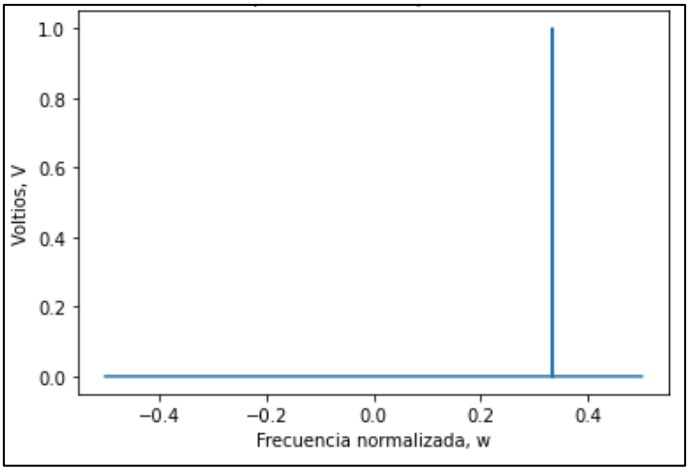


Figura 11. Espectro de una exponencial compleja

5.5 Convolución de dos señales utilizando NumPy.

Además de crear vectores y realizar transformadas de Fourier, la librería NumPy nos permite realizar la convolución de dos señales mediante una sola llamada a la función "convolve". Esta función será muy importante cuando introduzcamos la creación de filtros digitales, ya que la salida de una señal al atravesar un filtro es igual a la convolución de las muestras de la señal con las muestras del filtro.

6 Introducción a filtros digitales.

6.1 Introducción a filtros FIR (Finite Impulse Response)

Los filtros FIR son una subclase de los sistemas LTI discretos. Estos filtros ofrecen una respuesta al impulsa finita, la cual podemos formalizar de la siguiente manera:

$$h[n] = 0 \ \ para \ \ n < A \ y \ n > B$$

Esto quiere decir que podremos implementarlos utilizando una secuencia finita de coeficientes y digitalizarlos en un vector finito de muestras asociando cada coeficiente a una muestra.

De esta forma, podemos expresar la relación entre la entrada $x[n]$ y la salida $y[n]$ de un filtro FIR mediante la siguiente ecuación en diferencias.[6]:

$$y[n] = \sum_{i=0}^{L-1} bi \, . x[n-i]$$

Donde L representa el orden o número de coeficientes del filtro y bi son cada uno de sus coeficientes. A partir de la ecuación en diferencias anterior, podemos obtener la función de transferencia del filtro en el dominio Z. Operando, obtenemos la siguiente expresión.

$$H(Z) = \sum_{k=0}^{L-1} a[k] z^{-k}$$

Una de las grandes diferencias de los filtros FIR con los filtros IIR, que se analizarán en el siguiente punto, es que, en un filtro FIR, la salida dependerá únicamente de los valores de la entrada, sin embargo, en un filtro IIR, dependerá tanto de la entrada como de las muestras retardadas de la salida. Es por ello, por lo que en un filtro IIR, nunca existirá retroalimentación.

Además, en la implementación de un filtro FIR, solo necesitaremos utilizar elementos de retardo, multiplicadores y sumadores, con un número de etapas igual al orden del filtro.

6.2 Introducción a filtros IIR (Infinite Impulse Response)

Un filtro IIR, es aquel que presenta una respuesta al impulso infinita y su principal característica es que la salida es en función tanto de la entrada y como de las salidas anteriores. Su ecuación en diferencias puede expresarse de la siguiente manera.[6]:

$$y[n] = \sum_{k=0}^{L-1} a[k]x[n-k] + \sum_{k=0}^{L-1} b[k]\, y[n-k]$$

Donde $a[k]$ y $b[k]$ son los coeficientes correspondientes a la entrada y a la salida respectivamente. De nuevo, obtenemos la función de transferencia del filtro a partir de su ecuación en diferencias.

$$H(z) = \frac{\sum_{k=0}^{L-1} a[k]\, z^{-k}}{1 - \sum_{k=1}^{L-1} b[k]\, z^{-k}}$$

Si expresamos la función de transferencia del filtro como el cociente entre la salida y la entrada en el dominio Z, podemos realizar esta igualación:

$$\frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{L-1} a[k]\, z^{-k}}{1 - \sum_{k=1}^{L-1} b[k]\, z^{-k}}$$

Si continuamos operando, podemos llegar a la siguiente expresión:

$$Y(z) = X(z) \sum_{k=0}^{L-1} a[k] z^{-k} + Y(Z)(\sum_{k=1}^{L-1} b[k]\, z^{-k})$$

Verificando así que la salida de un filtro IIR, será igual a la suma de la muestra actual y las muestras retardas de la entrada, sumadas a la salida retardada, multiplicadas por sus coeficientes correspondientes. En la siguiente figura encontramos la implementación hardware de un filtro IIR.

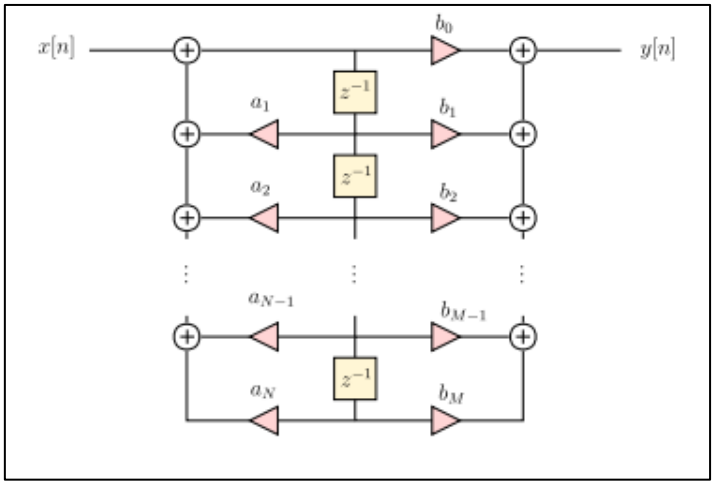


Figura 12. Implementación de un filtro IIR.[7]

6.3 Creación de filtros digitales.

Python proporciona una gran variedad de librerías capaces de crear filtros digitales. Durante el desarrollo del proyecto, se ha utilizado la función “remez” de la librería “signal” para ello.

Esta función es capaz de calcular los coeficientes para un filtro FIR cuya función de transferencia minimiza el error entre la ganancia deseada y la ganancia realizada en las bandas de frecuencia especificadas, mediante el método de diseño de Parks-McClellan y utilizando el algoritmo de intercambio de Remez. A continuación, se muestra la fórmula de diseño citada anteriormente para calcular el número de coeficientes óptimo.[8]

$$N = \frac{-20log_{10}(\partial_1\partial_2) - 13}{14.6\Delta F} + 1$$

Se muestra un código Python utilizado para crear un filtro paso bajo con una frecuencia de corte de 8KHz. En la figura 13 se muestra la respuesta en frecuencia de dicho filtro. Nótese que se ha hecho uso de la escala de dB para representar la respuesta en frecuencia de nuestro filtro.

```
from scipy import signal
import matplotlib.pyplot as plt
import numpy as np
fs = 22050.0
fcorte = 8000.0
tr = 100
numcoeficientes = 400
coeficientes = signal.remez(numcoeficientes, [0, fcorte, fcorte+tr])
w, h = signal.freqz(coeficientes, [1], worN=2000)
```

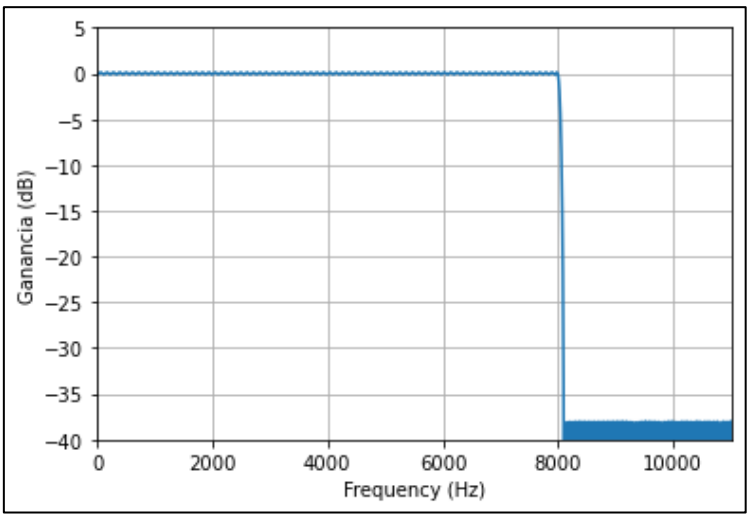


Figura 13. Filtro paso bajo con frecuencia de corte de 8 KHz

6.4 Dispositivos, canales y atributos de ADALM_PLUTO.

Antes de comenzar a transmitir para llevar a cabo las primeras prácticas, es necesario aprender a manipular la libreria iio de Analog Devices para poder interactuar con los distintos elementos Hardware de nuestra tarjeta desde Python. Una vez aprendamos la sintaxis utilizada por dicha librería, podremos configurar y habilitar los dispositivos y canales necesarios para realizar una transmisión adecuada. En la siguiente figura se muestra la relación entre las distintas clases Python que permiten la configuración de los componentes de la ADALM-PLUTO.

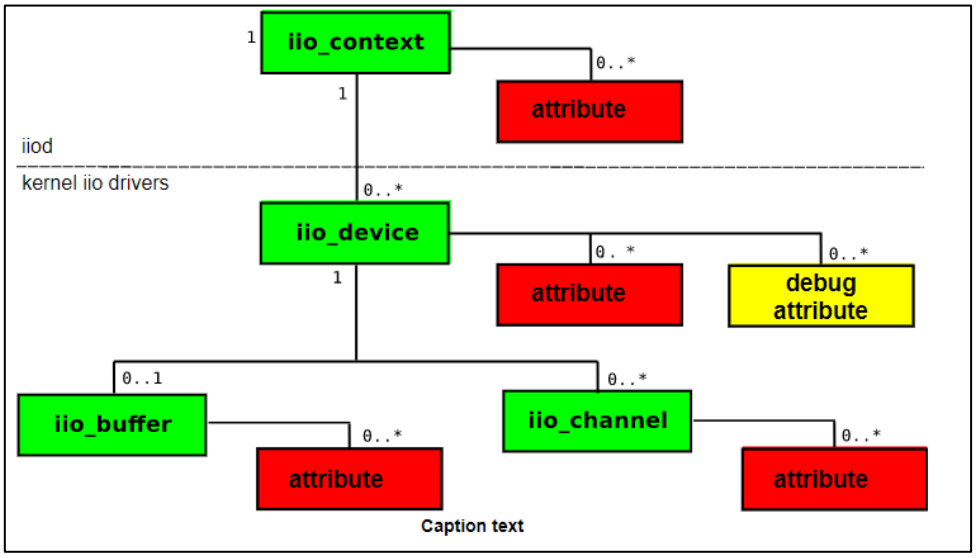


Figura 14. Diagrama de clases de la librería IIO.[13]

- Un objeto *iio_context* representa un contexto local y acceder a los distintos dispositivos de la ADALM-PLUTO. Para configurar un dispositivo es imprescindible haber creado previamente un objeto *iio_context.*
- Un objeto *iio_device* representa los distintos dispositivos presentes en la tarjeta, como por ejemplo el controlador o los ADC.
- Un objeto *iio_chanel* representa los distintos canales a los que se puede acceder desde un dispositivo. Para configurar un canal es necesaria la creación previa de un *iio_device*.
- Un objeto *iio_buffer* nos permitirá crear distintos buffers y asociárselos a cada uno de los iio_device creados.
- Un objeto *iio_context* puede contener cero o más objetos *iio_device,* y un objeto *iio_device* está asociado a un solo *iio_context*
- Un objeto *iio_device* puede contener cero o más objetos *iio_channel/iio_buffer* y un objeto *iio_channel/iio_buffer* está asociado con un solo *iio_device* .
- Un objeto *iio_device* puede estar asociado con un objeto *iio_buffer* y un objeto *iio_buffer* está asociado con un solo *iio_device* .

6.5 Ejemplo de configuración de canales, dispositivos y atributos.

Una vez entendidas las relaciones entre las distintas clases que componen la librería iio, es el momento de realizar la primera configuración sobre la ADALM-PLUTO.

En primer lugar, debemos acceder a un contexto local para poder configurar los dispositivos de la tarjeta. Invocando a la función *"NetworkContext()"* de la librería *iio* y pasando como atributo la IP del dispositivo, crearemos un contexto local que utilizaremos para localizar y configurar los elementos nuestra tarjeta.

La IP por defecto de la ADALM-PLUTO es "*192.168.2.1".*

Una vez creado el contexto, podemos acceder a los distintos dispositivos utilizando la función *"find_device()"* y pasando como argumento la etiqueta asociada al dispositivo que queremos configurar. En este caso, accedemos al controlador mediante la etiqueta *"ad9361-phy".*

```
contexto = iio.NetworkContext('192.168.2.1')
ctrl = contexto.find_device("ad9361-phy")
```

Una vez creado el dispositivo, podemos configurar los atributos de los distintos canales que posea. Para acceder a un canal, es necesario utilizar la función "find_channel()" pasando como argumento la etiqueta del canal a configurar. Una vez se haya accedido al canal, se puede establecer el valor de los distintos atributos configurables.

En el siguiente código se muestra un ejemplo de acceso a distintos canales y a sus atributos.

```
TXLO = 1000000000
TXBW = 5000000
TXFS = 3000000
RXLO = TXLO
RXBW = TXBW
RXFS = TXFS

ctrl.find_channel('RX_LO').attrs["frequency"].value = str(int(RXLO))
ctrl.find_channel('TX_LO').attrs["frequency"].value = str(int(TXLO))
ctrl.find_channel('voltage0').attrs["rf_bandwidth"].value =
str(int(RXBW))
ctrl.find_channel('voltage0',True).attrs["rf_bandwidth"].value =
str(int(TXBW))
ctrl.find_channel('voltage0').attrs["sampling_frequency"].value =
str(int(RXFS))
ctrl.find_channel('voltage0',True).attrs["sampling_frequency"].value =
str(int(TXFS))
ctrl.find_channel('voltage0').attrs['gain_control_mode'].value =
'slow_attack'
ctrl.find_channel('voltage0',True).attrs['hardwaregain'].value = '-
30''TX_LO').attrs["frequency"].value = str(int(TXLO))
```

Donde el argumentos que reciben *"find_chanel"* y *"attrs"* son, respectivamente,la etiqueta del canal y el atributo a configurar. Por último, mediante la instrucción *"value",* indicamos el valor del atributo indicado anteriormente.

En este caso, estamos configurando la frecuencia de recepción y transmisión, el ancho de banda y la frecuencia de muestreo a un valor de 1GHz, 5MHz y 3MHz, respectivamente.

Utilizando la sintaxis anterior, podemos acceder a cualquier canal y atributo configurable de nuestra ADALM_PLUTO. Únicamente, es necesario conocer correctamente, la etiqueta del canal y del atributo que queremos configurar.

Las etiquetas mencionadas anteriormente, no son del todo intuitivas y, en ocasiones, pueden resultar difíciles de encontrar si queremos configurar algún atributo o canal en específico. No obstante, podemos buscar el nombre de dichas etiquetas y atributos mediante la herramienta IIO Oscilloscope.

En la siguiente figura, se muestra la manera de encontrar las etiquetas asociadas tanto a los dispositivos, como a los canales y los atributos.

En las secciones *"Device Selection" e "IIO Device Attributes"* de la figura 15, podemos buscar las etiquetas asociadas a los distintos elementos y parámetros de la ADALM-PLUTO. En este caso, hemos encontrado las etiquetas utilizadas en la configuración anterior.

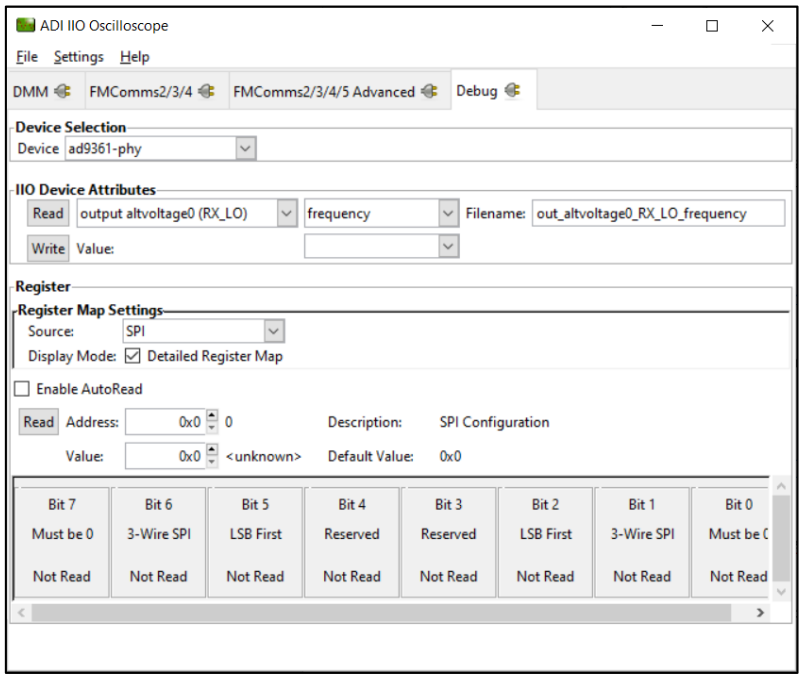


Figura 15.Dispositivos, canales y atributos de la ADALM-PLUTO

Con las siguientes instrucciones, accedemos al DAC de la ADALM-PLUTO, habilitamos tanto los flags de los canales *"voltage0" y "voltage1"* estableciéndolos en *True,* así como las ramas Q e I del modulador IQ, estableciendo su valor a 0.

```
pluto_dac = contexto.find_device('cf-ad9361-dds-core-lpc')
pluto_dac.find_channel("voltage0",True).enabled = True
pluto_dac.find_channel("voltage1",True).enabled = True

pluto_dac.find_channel('TX1_I_F1',True).attrs['raw'].value = str(0)
pluto_dac.find_channel('TX1_Q_F1',True).attrs['raw'].value = str(0)
pluto_dac.find_channel('TX1_I_F2',True).attrs['raw'].value = str(0)
pluto_dac.find_channel('TX1_Q_F2',True).attrs['raw'].value = str(0)
pluto_tx = iio.Buffer(pluto_dac, 2**15, cyclic=True)
```

En la última instrucción, hemos asociado un objeto buffer al DAC del a ADALM-PLUTO y lo hemos configurado en modo cíclico. Este aspecto será analizado posteriormente. Finalmente, configuramos el ADC, habilitamos los canales *"voltage0"*, *"voltage1"* y creamos el buffer de recepción.

```
pluto_adc = contexto.find_device('cf-ad9361-lpc')
pluto_adc.find_channel("voltage0").enabled = True
pluto_adc.find_channel("voltage1").enabled = True
pluto_rx = iio.Buffer(pluto_adc, 2**15)
```

Una vez están configurados todos los canales y atributos, podemos realizar transmisiones de prueba para observar el comportamiento de nuestro dispositivo ante la inyección de distintas señales.

6.6 Primeras prácticas con Adalm Pluto.

Una vez asimilados los conocimientos básicos del procesamiento digital de señales mediante Python y aprendida la sintaxis utilizada por la librería *iio* para configurar la tarjeta, es el momento de realizar las primeras transmisiones reales.

Para ello, utilizaremos la herramienta *IIO Oscilloscope*, desarrollada por el fabricante de nuestro dispositivo, con el objetivo de visualizar los datos que se están transmitiendo por la Pluto SDR.

Para poder utilizar nuestro osciloscopio, es necesario que conectemos previamente la ADALM-PLUTO al ordenador. Una vez conectada, podemos abrir el osciloscopio y visualizar tanto el dominio del tiempo, como en el dominio de la frecuencia, el conjunto de datos transmitidos por nuestro dispositivo.

En la siguiente figura observamos el dominio frecuencial cuando nuestro dispositivo se encuentra en estado de reposo, es decir, sin transmitir ni recibir nada.

En la interfaz podemos seleccionar los canales para observar la señal transmitida. En este caso, resultan ser los canales *voltage0 y voltage1.*

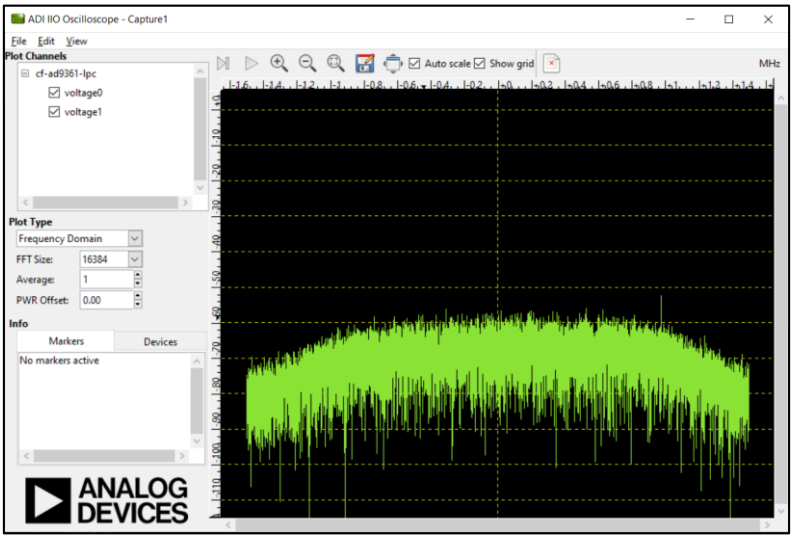


Figura 16.Dominio frecuencial en IIO Oscilloscope

6.7 Transmisión de tonos digitales.

La transmisión de un tono digital en el dominio de la frecuencia es muy sencilla de reconocer. En el siguiente apartado realizaremos la primera transmisión dándole uso a nuestro hardware. Para ello, será necesario aprender la manera de interactuar con el transmisor de la ADALM-PLUTO y de enviar muestras a un modulador IQ.

Primero, creamos la señal a transmitir. En este caso una exponencial compleja a partir de un vector de 2^{15} muestras y de frecuencia digital 0,3.

A continuación, debido a que el transmisor de la ADALM-PLUTO proporciona un modulador del tipo IQ, será necesario intercalar la parte real y compleja de nuestra señal en un solo vector para poder transmitirla. De esta manera, las muestras reales e imaginarias de nuestra señal ocuparán, respectivamente, las posiciones pares e impares del vector de transmisión. Así, la rama real I se ocupará de las posiciones pares y la rama imaginaria Q de las impares.

Nótese que la longitud de el vector de transmisión (2^{16})es el doble que la de la señal a transmitir (2^{15}). Esto se debe a que, por cada muestra de la señal a transmitir, tendremos dos muestras en el vector de transmisión correspondientes a sus partes reales e imaginarias.

```
tx_vector=np.empty(2**16, dtype=np.int16)
tx_vector[0::2]=np.int16((2**14)*np.real(s))
tx_vector[1::2]=np.int16((2**14)*np.imag(s))
```

Una vez adaptada nuestra señal al modulador IQ, es el momento de transmitir el tono. La manera de interactuar con el bloque transmisor es la siguiente:

1) Escribir las muestras a enviar en el buffer de transmisión. Invocamos a la instrucción *"write()"* para escribir el vector de transmisión en el buffer asociado al DAC.
2) Empujar las muestras del buffer al transmisor, utilizando la instrucción *"push()"* de los objetos *Buffer*.

```
pluto_tx.write(bytearray(tx_vector))
pluto_tx.push()
```

Una vez empujadas las muestras desde el Buffer del DAC al bloque transmisor, habremos realizado nuestra primera transmisión real. Podemos observar las muestras transmitidas utilizando la herramienta *IIO Oscilloscope.*

Para poder visualizar la transmisión correctamente, debemos abrir dicho software de manera previa a la ejecución del código Python encargado de transmitir la señal. En caso contrario, es posible que el osciloscopio no sea capaz de conectarse a los canales de la ADALM-PLUTO y no seamos capaces de visualizar la transmisión.

En la Figura 17 se muestra la interfaz del osciloscopio en el dominio frecuencial durante la transmisión del tono digital creado anteriormente.

Recordando que el espectro digital normalizado se representa entre ± 0.5, podemos apreciar un pico de potencia situado en, aproximadamente, un valor de 0.3, valor correspondiente a la frecuencia de nuestra exponencial compleja.

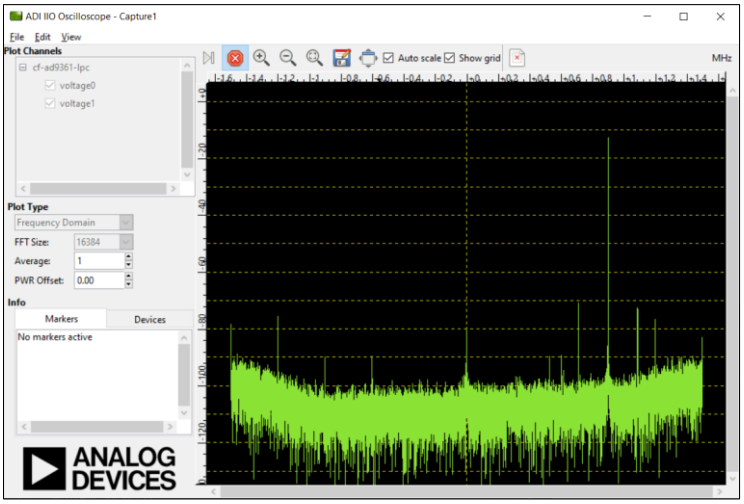


Figura 17. Transmisión de un tono digital de frecuencia 0.3

Otra señal sencilla de reconocer en el dominio de la frecuencia es la señal coseno. Podemos utilizar la identidad de Euler para expresar una señal cosenoidal de la siguiente manera.[9]:

$$\cos x = \frac{e^{jx} + e^{-jx}}{2}$$

De esta manera, expresamos un coseno como $\frac{1}{2}$ de la suma de dos exponenciales complejas con exponentes cambiados de signo. Si recordamos los puntos anteriores, el espectro de una exponencial compleja a una determinada frecuencia, es igual a una delta de Dirac situada en dicha frecuencia. Al expresar el coseno como $\frac{1}{2}$ de la suma de dos exponenciales complejas, podemos confirmar que el espectro de un coseno será iguala dos deltas de Dirac con amplitud igual a la mitad de la amplitud del coseno, situadas en $\pm$ la frecuencia del coseno.

Fijándonos en la Figura 18, observamos dos deltas de Dirac situadas en una frecuencia digital normalizada igual a ± 0.3, correspondientes al espectro de un coseno con frecuencia igual a 0,3.

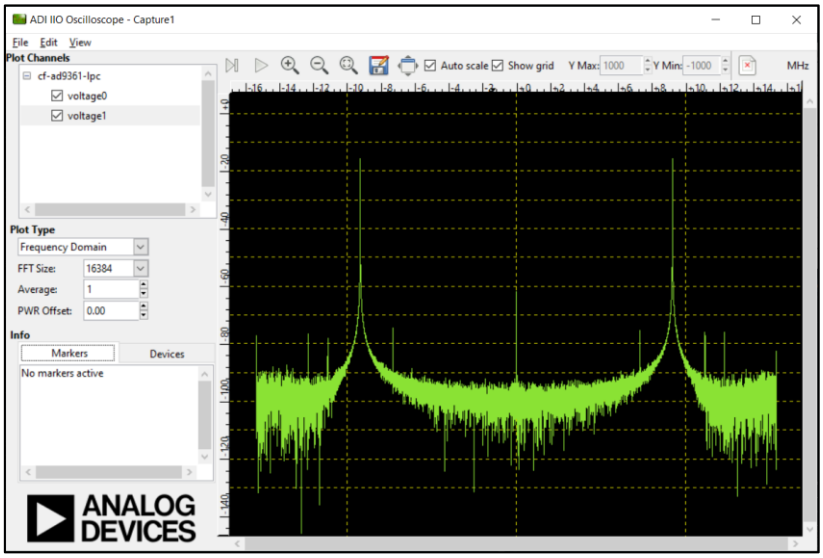


Figura 18.Transmisión de un coseno de frecuencia 0.3

7 Modelo desarrollado

7.1 Implementación de un modulador AM.

Una vez asimiladas las técnicas necesarias para interactuar con nuestra ADALM-PLUTO, damos procedemos a la implementación de una modulación AM con técnicas digitales. Para ello, haremos uso de las librerías mencionadas anteriormente con el objetivo de crear cada uno de los componentes de los bloques transmisor y receptor de un

modulador AM. Una vez implementados ambos bloques se realizará una simulación en Python, así como una modulación real utilizando nuestra tarjeta.

Realizar una modulación AM, consiste en crear una señal modulada como una combinación de una señal moduladora o de información y una señal portadora.

Podemos expresar una señal modulada en amplitud de forma genérica utilizando la siguiente ecuación. [9]:

$$yAM(t) = Ac[1 + m\,.\,xN(t)]\cos(2\pi fot)$$

Donde los parámetros Ac y fo son, respectivamente, la amplitud y la frecuencia de la señal portadora, $yAM(t)$ es la señal modulada en amplitud, $xN(t)$ es la señal moduladora o señal de información y m es el índice de modulación.

Reescribiendo la ecuación anterior, podemos expresarla de la siguiente manera:

$$yAM(t) = Ac.\cos(2\pi fot) + Ac.m\,.\,xN(t)\cos(2\pi fot)$$

Además, si calculamos la transformada de Fourier de la señal anterior expresada de forma genérica, obtenemos:

$$yAM(f) = \frac{Ac}{2}[\delta(f - fo) + \delta(f + fo)] + \frac{Ac.m}{2}[X(f) * (\delta(f + fo) + \delta(f - fo))]$$

Donde $\delta(f \pm fo)$ es la función delta de Dirac desplazada una frecuencia $\pm fo$ y "*" representa el operador convolución.

La salida de una convolución de un espectro cualquiera $X(f)$ con una delta de Dirac desplazada en frecuencia $\delta(f - fo)$, será igual a ese mismo espectro desplazado a la posición donde se encuentre situada la delta.

$$X(f) * \delta(f - fo) = X(f - fo)$$

$$X(f) * \delta(f + fo) = X(f + fo)$$

Utilizando estas propiedades, podemos expresar el espectro de una señal modulada en amplitud de la siguiente forma:

$$yAM(f) = \frac{Ac}{2}[\delta(f - fo) + \delta(f + fo)] + \frac{Ac.m}{2}[X(f - fo) + X(f + fo)]$$

Por lo que el espectro de nuestra señal será desplazado a las posiciones donde se encuentren situadas las frecuencias de la señal portadora.

7.2 Transmisor AM.

Una vez entendida la teoría asociada a una modulación AM, es el momento de llevar a cabo la implementación del bloque transmisor y realizar una simulación para verificar que los bloques desarrollados operan correctamente. En la figura 19 podemos apreciar el diagrama de bloques de un transmisor AM.

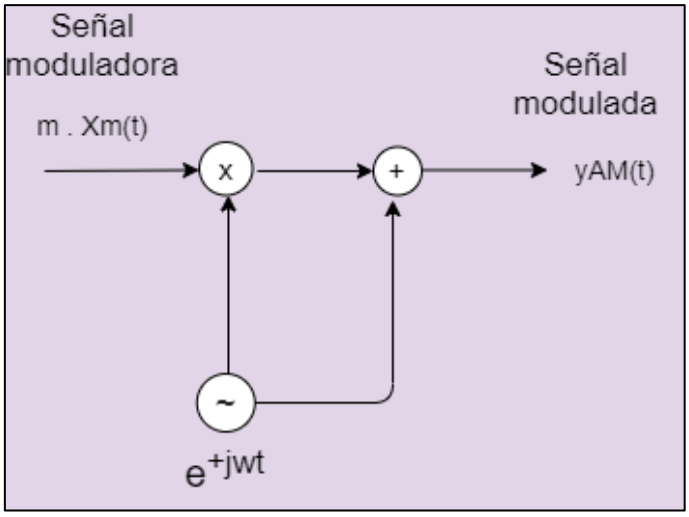


Figura 19. Diagrama de bloques de un transmisor AM

Donde nuestro modelo de transmisor está compuesto por un multiplicador conectado a dos ramas distintas.

La primera de ellas corresponde la señal de información multiplicada por el índice de modulación y la segunda, a la señal portadora. Posteriormente, encontramos un sumador conectado tanto a la salida del multiplicador y a la señal portadora.

De esta manera, podemos obtener la expresión genérica de una señal modulada en amplitud a la salida de nuestro bloque transmisor. En el siguiente código, mostramos la implementación en Python del transmisor AM concorde al diagrama de bloques mostrado en la figura 19.

```
import numpy as np
import matplotlib.pyplot as plt
fmoduladora=100
fportadora=300
fs=900
t=np.arange(0,10,1/fs)
portadora=np.exp((2j*np.pi*fportadora*t))
portadora=portadora/max(portadora)
moduladora=np.cos(2*np.pi*fmoduladora*t)
moduladora=moduladora/max(moduladora)#normalizamos la señal moduladora
Ac=3 #amplitud de la señal portadora
m=0.2 #indice de modulación
modulada=(1+m*moduladora)*Ac*portadora
modulada=modulada/max(modulada)#normalizamos la señal modulada
```

Donde, para realizar la simulación, se ha utilizado un coseno de 300 Hz como señal moduladora, una exponencial compleja de 300Hz como señal portadora, una frecuencia de muestreo de 900 Hz y un índice de modulación de 0,2.

Nótese que, en lugar de utilizar un coseno como señal portadora, se ha utilizado una exponencial compleja. Este aspecto será analizado en un punto posterior.

7.3 Receptor AM

Basándonos en el espectro de la señal modulada, podremos recuperar el espectro original de la señal moduladora si centramos en frecuencia 0 de la señal modulada y eliminamos su componente piloto. El receptor será el encargado de implementar dicha funcionalidad. El diagrama de bloques propuesto para llevar a cabo la recepción de una señal modulada en amplitud se muestra en la figura 20.

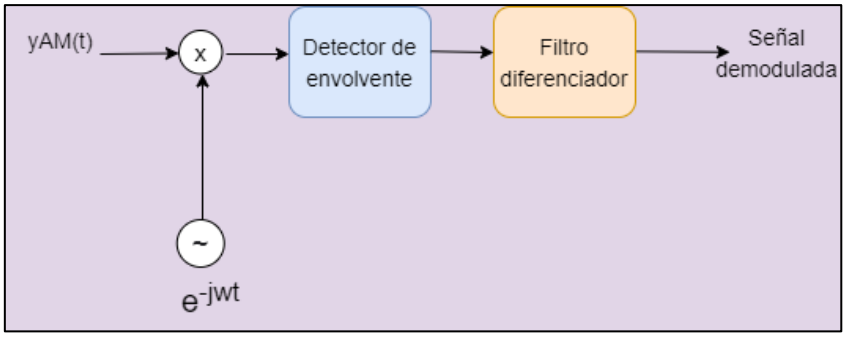


Figura 20. Diagrama de bloques de un receptor AM

Para que las componentes situadas en $\pm fo$ recuperen sus posiciones originales en el eje frecuencial, es necesario centrar en frecuencia 0 el espectro de la señal modulada. Para ello, multiplicamos a la señal modulada por una exponencial de frecuencia $-fo$. En la figura 21 se muestra la representación temporal de una señal modulada de amplitud 1.

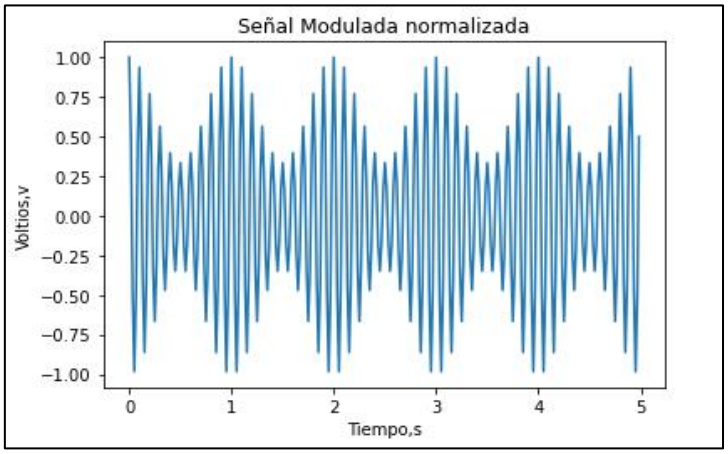


Figura 21. Representación temporal de una señal modulada

Para recuperar la forma de onda de la señal moduladora, implementaremos un detector de envolvente digital. Para ello, será suficiente con obtener el valor absoluto de la señal moduladora. Si nos fijamos en la figura 22, visualizamos el espectro de la señal modulada centrada en frecuencia 0 tras haber obtenido su envolvente. Para recuperar el espectro original de nuestra señal de información, nos bastará con eliminar la componente continua. Para filtrar esta componente, debemos diseñar un filtro cuya respuesta en frecuencia anule la componente continua y deje pasar todas las demás.

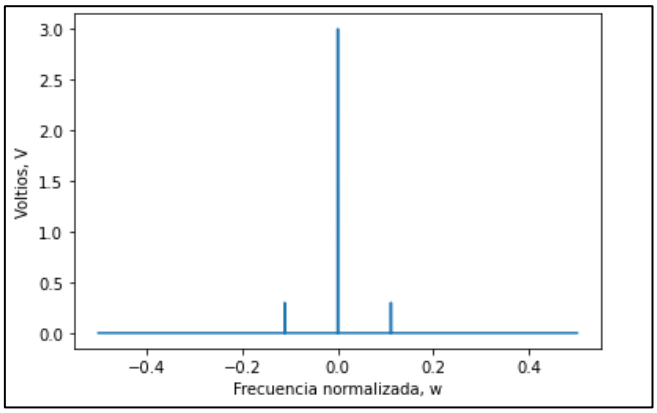


Figura 22. Salida del detector de envolvente

La respuesta en frecuencia de un filtro diferenciador se muestra en la figura 23. Si inyectamos nuestra señal en este filtro, habremos eliminado la componente de frecuencia 0 y recuperaremos las componentes frecuenciales originales de la señal de información.

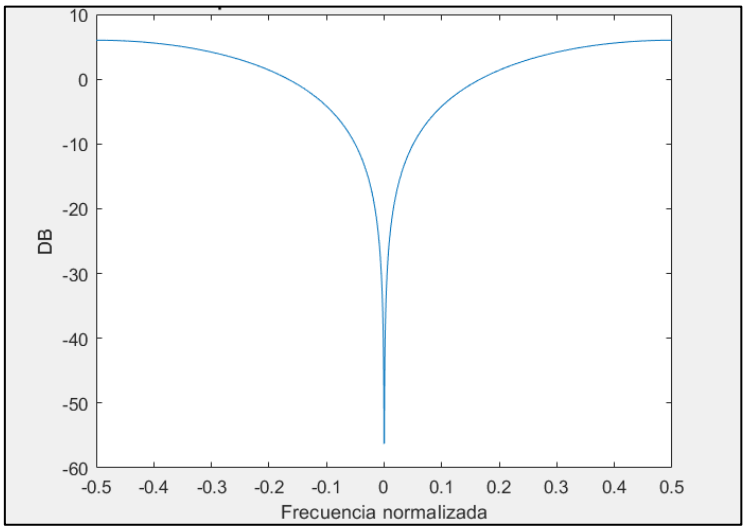


Figura 23. Respuesta en frecuencia del filtro diferenciador

7.4 Exponencial compleja o coseno como señal portadora

Tras haber comprobado el buen funcionamiento de los componentes del transmisor/receptor de nuestro modulador AM, se procederá a realizar una modulación AM real utilizando el Hardware de nuestra tarjeta. En la simulación anterior se ha utilizado una exponencial compleja como señal portadora debido a que, en la práctica, utilizar una señal coseno como portadora no es conveniente.

En la figura 24 podemos apreciar el espectro de una señal modulada utilizando una un coseno con frecuencia portadora $fo = 0{,}3$. De esta manera, el espectro de la señal modulada se desplazará tanto a las frecuencias positivas como negativas, obteniendo así dos componentes piloto.

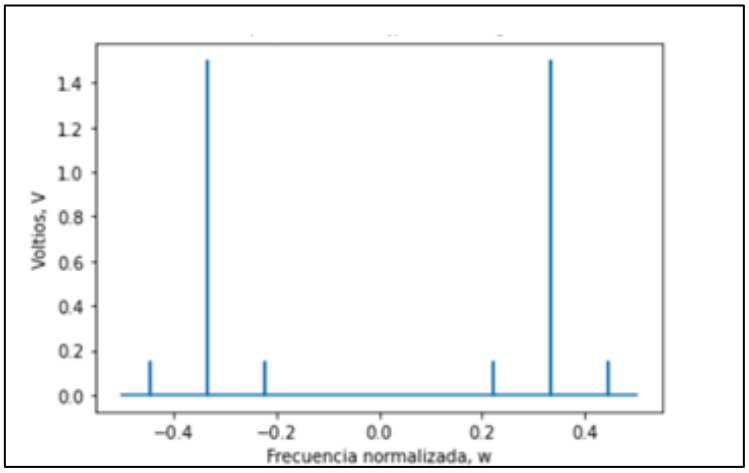


Figura 24. Espectro de una señal modulada por un coseno

Multiplicando la señal de la figura 24 por un coseno de frecuencia igual a fo, podemos centrar en frecuencia 0 el espectro de la señal modulada, obteniendo así el espectro de la figura 25. Donde, para recuperar la información, debemos eliminar las repeticiones y la componente continua.

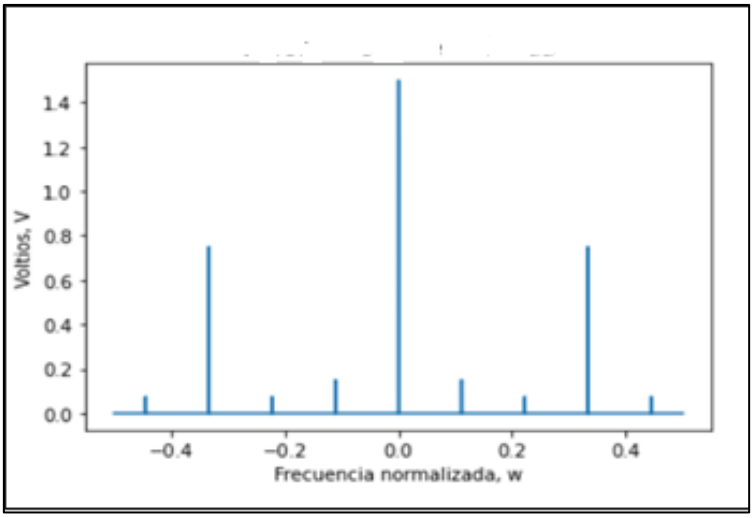


Figura 25. Señal demodulada por un coseno

En la figura 25, aparecen 3 componentes piloto situadas en frecuencias iguales a $\pm 0{,}3\ y\ 0$. Las componentes situadas en frecuencia 0 se obtienen debido al solapamiento de las componentes espectrales de la figura 24. Este solapamiento, en la práctica, no suele ser preciso ya que los osciladores locales del transmisor y del receptor no logran ir completamente sincronizados. Si este solapamiento no es perfecto, nuestra señal recuperada habrá sufrido daños que empeorarán la calidad de la comunicación. Para evitar dicho solapamiento, podemos sustituir nuestra señal portadora por una exponencial compleja, desplazando así el espectro una sola vez y evitando el solapamiento a la hora de la demodulación. Además, no tendremos que eliminar las repeticiones frecuenciales debidas a la multiplicación por un coseno.

7.5 Buffer cíclico y buffer no cíclico

Para realizar transmisiones con la ADALM-PLUTO, es necesario comprender una serie de conceptos relacionados con los buffers. Cuando invocamos al constructor de un objeto buffer, indicamos su longitud y a que dispositivo está asociado. Además, podemos indicar si el buffer ha de funcionar de manera cíclica o no. En el siguiente código Python se muestra la creación de un buffer configurado de manera cíclica.

```
bufferciclico = iio.Buffer(pluto_dac, longitud, cyclic=True)
```

Donde el primer argumento indica a que dispositivo está asociado el objeto Buffer (En este caso se trata del DAC), el segundo la longitud y el tercero la configuración en modo cíclica. Si no especificamos ningún argumento en la tercera posición en el momento de la creación del buffer, la tarjeta lo interpretará por defecto como un buffer no cíclico.

Un buffer cíclico se caracteriza por transmitir de manera cíclica el contenido escrito en su memoria. Sin embargo, un buffer no cíclico lo transmitirá únicamente una vez. Conociendo este comportamiento de los buffers, elegiremos el más adecuado en función del tipo de transmisión a implementar.

Por ejemplo, si queremos realizar una transmisión en tiempo real, no debemos configurar el buffer de forma cíclica, ya que debemos descartar los datos transmitidos para poder cargar nuevos datos a enviar. Si nuestra transmisión no requiere el procesado de los datos en streaming, podemos configurarlo de manera cíclica para que nuestro transmisor esté constantemente transmitiendo. De esta manera, nos aseguramos de que el receptor vaya a capturar los datos transmitidos.

En un proceso de comunicación, el receptor está constantemente escuchando los datos que envía el transmisor. Si el buffer del transmisor está configurado de manera cíclica, puede darse la situación de que el receptor comience a recibir información en un instante de tiempo que no se corresponde con el inicio de los datos, provocando así pérdida de información. Para asegurarnos de la recepción completa de los datos

transmitidos, debemos tener en cuenta la longitud del buffer de transmisión y de recepción.

Si configuramos el buffer del trasmisor para escribir N muestras, debemos configurar el buffer del receptor con una longitud mayor o igual que *2N*, ya que, si ambos buffers poseen la misma longitud, y en ausencia de sincronización, lo más probable es que el receptor no sea capaz de recibir nada. Si configuramos la longitud de los buffers adecuadamente, nos aseguraremos de la recepción completa de los datos transmitidos.

8 Introducción a las comunicaciones digitales.

Un transceptor digital es un sistema compuesto por un conjunto de dispositivos, tanto analógicos como digitales que se combinan para tratar y manipular información binaria. La principal función de estos dispositivos es lograr la transmisión y recepción de datos transmitidos a través de algún medio. En el caso de la ADALM-PLUTO, transmitiremos los datos a través de un cable coaxial.

El objetivo principal de una modulación digital es comprimir la mayor cantidad de datos posibles reduciendo al máximo el espectro a utilizar.Además,en una modulación digital, transmitiremos símbolos compuestos por un número finito de bits en lugar de valores de tensión.

8.1 Modulación en fase PSK.

Una modulación en fase es una modulación digital que convierte símbolos en ondas con distinta fase a partir de una fase de referencia. Cualquier modulación PSK, utiliza un número finito de fases, asociando cada una de ellas a un símbolo binario compuesto por 1 o más bits. El número de símbolos a transmitir dependerá del número de bits utilizado para representar cada uno de los símbolos, cumpliendo siempre la siguiente relación:

$$nº\ simbolos = 2^{nº\ bits\ por\ símbolo}$$

El trabajo del modulador consistirá en asociar un símbolo binario a una señal con una fase determinada, por lo que el número de fases a utilizar será igual al número de símbolos. Por otro lado, el demodulador, se encargará de realizar el proceso inverso, obteniendo un símbolo a partir de una fase.

Matemáticamente, podemos expresar la forma de onda de una señal PSK de la siguiente manera [9]:

$$si(t) = A\cos\left(2\pi fct + (2i-1)\frac{\pi}{m}\right) para\ i = 1, \dots, log2(m)$$

Donde *A* es la amplitud, fc es la frecuencia de la señal portadora, *m* es el número de símbolos y $(2i-1)\frac{\pi}{m}$ representa el cambio de fase para cada símbolo.Existen distintos esquemas de modulación PSK basados en el número de M posibles valores que se pueden asignar a una señal PSK. El caso más sencillo consiste en una modulación BPSK donde utilizamos 1 bit por símbolo y transmitimos "0" ó "1". Por lo general, la regla básica para llevar a cabo una modulación BPSK es la siguiente:

$$1 \rightarrow s1(t) = A\cos(wc.t)$$
$$0 \rightarrow s2(t) = A\cos(wc.t + \pi)$$

Obteniendo así el símbolo A cuando enviemos "1" y el símbolo -A cuando enviemos "0", convirtiendo nuestra secuencia binaria en una señal PAM. En otras palabras, las dos señales BPSK están separadas una fase π. En la siguiente figura, podemos observar la conversión de los símbolos a sus correspondientes señales BPSK, donde el valor de la amplitud es igual a 1.

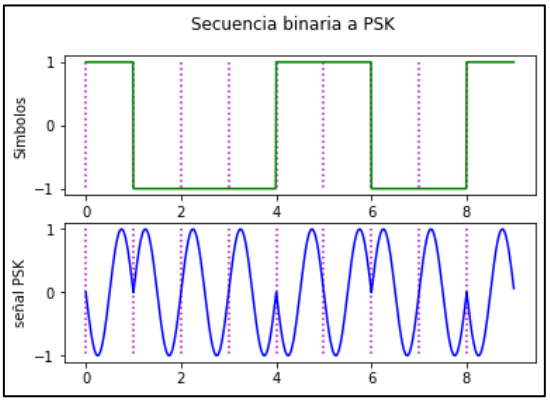


Figura 26. Señal PAM a PSK

Normalmente, los símbolos de una modulación PSK se representan en el plano real e imaginario, formando lo que se conoce como una constelación. En la figura 27 podemos apreciar la representación de los símbolos correspondientes a una modulación BPSK, donde los puntos representan los posibles símbolos a transmitir representados en el eje real y en el eje imaginario.

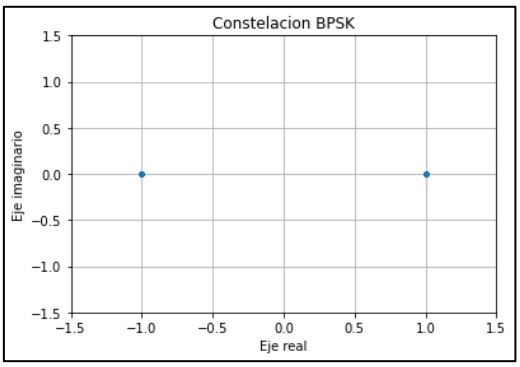


Figura 27. Constelación BPSK

8.2 Implementación de un transmisor BPSK.

Una vez estudiada la teoría asociada a las modulaciones digitales en general y a las modulaciones en fase en particular, procedemos a la implementación de un bloque transmisor sobre la placa ADALM-PLUTO. En la figura 28 encontramos el diagrama de bloques propuesto para nuestro transmisor BPSK.

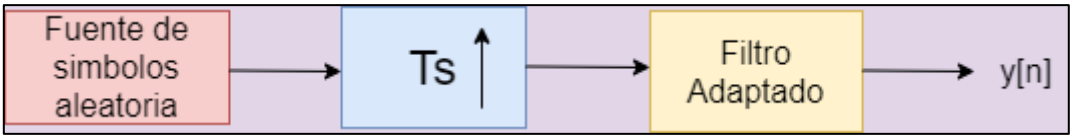


Figura 28. Diagrama de bloques de un transmisor BPSK

En primer bloque que nos encontramos consiste en una fuente aleatoria de símbolos binarios encargada de generar los símbolos a transmitir, cuya salida está conectada a un interpolador con valor igual al periodo del símbolo. El periodo del símbolo será igual al número de muestras utilizadas para representar cada símbolo. Por último, encontramos un filtro adaptado cuya principal función es reducir el ancho de banda de nuestra señal y disminuir la interferencia entre símbolos, también conocida como ISI.

8.2.1 Fuente de símbolos aleatoria

El código Python propuesto para la implementación de este bloque es el siguiente:

```
M=2
num_symbols=100
bits = np.random.randint(0, M, num_symbols)
plt.plot(bits)
```

Donde M es el orden de la modulación y num_symbols el número de símbolos a transmitir.

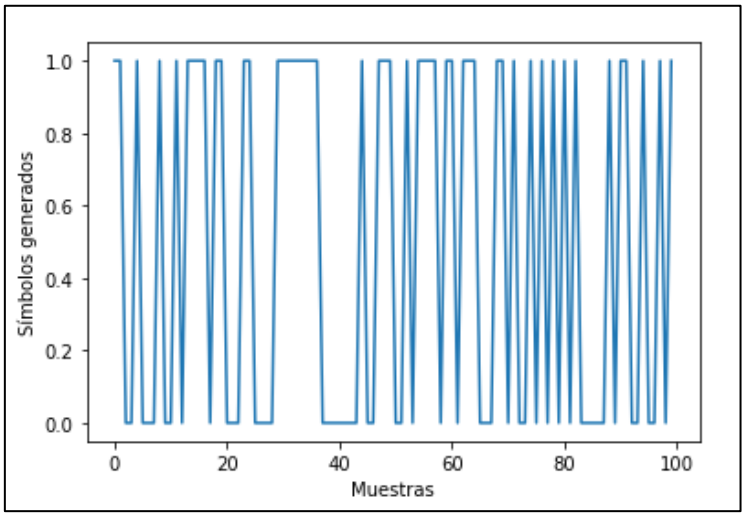


Figura 29. Símbolos generados por una fuente aleatoria

8.2.2 Interpolador.

El bloque interpolador se encargará de introducir muestras entre los distintos símbolos con el fin hacer coincidir el número de muestras que representa cada símbolo con el periodo del símbolo. Además, se encargará de colocar un “-1” en las posiciones donde haya un 0 para construir la señal PSK.

```
for bit in bits:
    pulse = np.zeros(sps)
    pulse[0] = bit*M-1
    pulse_train = np.concatenate((pulse_train, pulse))
```

En la figura 30 mostramos los primeros símbolos de la salida del interpolador. Nótese que los “0” de la señal aleatoria se han convertido en “-1”.

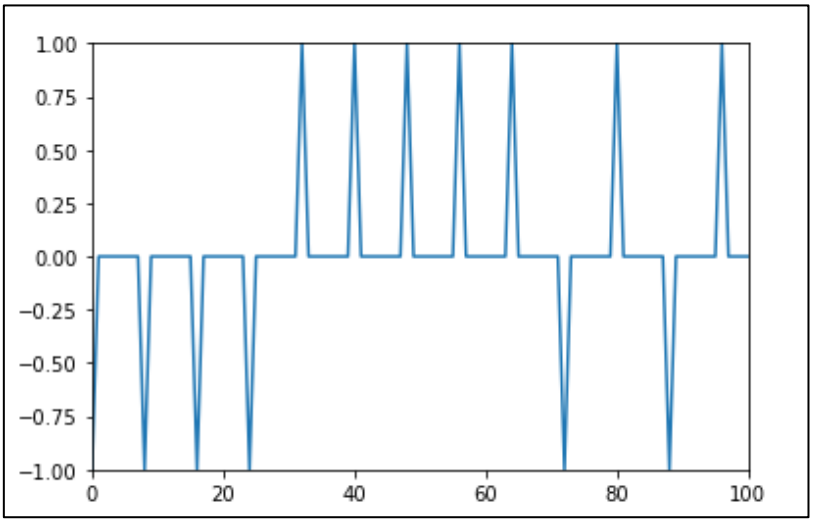


Figura 30. Símbolos a la salida del interpolador

8.2.3 Filtro Adaptado.

El modelado de pulsos es una herramienta imprescindible en las comunicaciones digitales debido a que aporta grandes beneficios a la transmisión. En la figura 31 visualizamos la representación temporal y frecuencial de un tren de pulsos cuadrados.

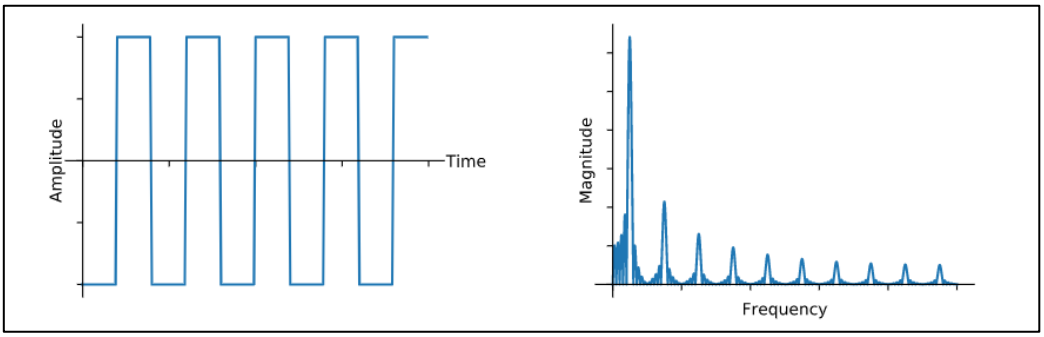


Figura 31. Representación temporal y frecuencial de pulsos cuadrados.[12]

Si nos fijamos en el espectro de la figura 31, observamos que los pulsos cuadrados no son del todo eficientes, hablando en términos de ancho de banda, ya que utilizan una cantidad excesiva de espectro. Éste fenómeno puede provocar interferencias que empeoren la calidad de la comunicación. Variando la forma de onda de nuestros símbolos en el dominio temporal, podemos reducir el ancho de banda consumido en el dominio frecuencial. Para implementar esta mejora, es necesaria la convolución de nuestros símbolos con lo que denominamos un filtro adaptado. El objetivo de este filtro es disminuir la ISI y reducir el ancho de banda ocupado por nuestra señal PSK.

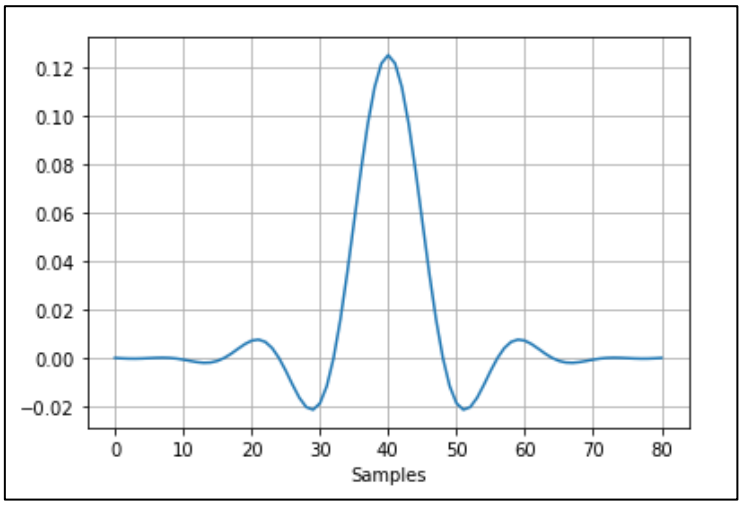


Figura 32. Representación temporal de un filtro coseno Alzado

Si hacemos convolucionar nuestro tren de pulsos con el filtro representado en la figura 32, podremos modelar la forma de los pulsos consiguiendo un uso más eficiente de la banda espectral. En la figura 33 se muestran las muestras resultantes de haber convolucionado la salida del interpolador con el filtro adaptado. A partir de este momento, nuestros símbolos dejan de tener una forma de onda cuadrada consiguiendo así un uso mucho más eficiente del espectro.

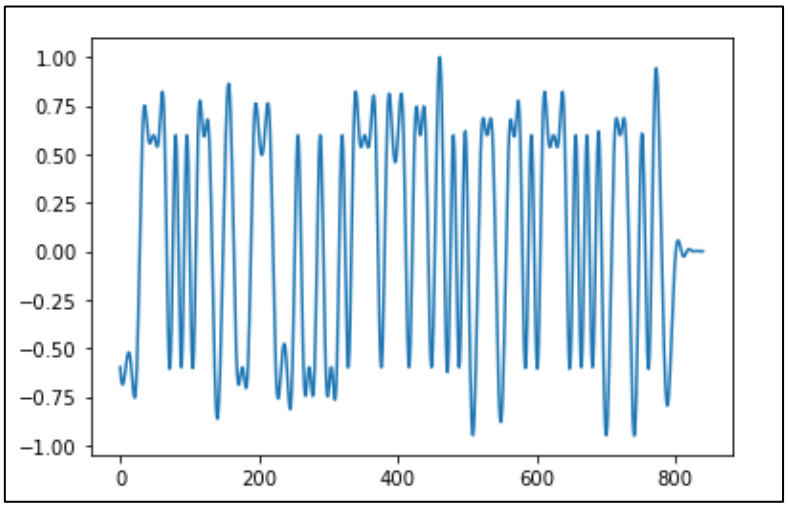


Figura 33. Símbolos a la salida del filtro Adaptado

Por último, en la figura 34 encontramos una comparativa entre la potencia de una misma señal BPSK antes y después de aplicarle un filtro adaptado.

En color azul, encontramos el espectro de los símbolos si haber atravesado el filtro adaptado. Podemos observar que los lóbulos secundarios presentan una potencia considerable comparándola con el pico máximo situado en frecuencia 0. Esta situación provocaría ISI y errores en la transmisión. Por otro lado, en color rojo, apreciamos como la potencia de estos lóbulos secundarios ha disminuido en gran magnitud manteniendo constante la potencia de nuestros símbolos. Al disminuir dicha potencia, habremos eliminado gran parte de la ISI ofreciendo así mejores prestaciones en la comunicación.

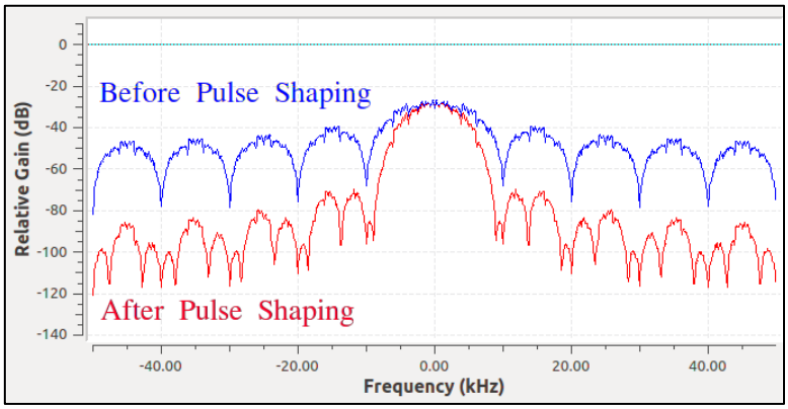


Figura 34. Comparación de potencia antes y después del modelado de pulsos. [12]

8.2.4 Filtro de coseno alzado y filtro raíz de coseno alzado

Normalmente, la respuesta de un filtro de coseno alzado no suele implementarse como un solo filtro en el bloque transmisor, sino que se fracciona entre el transmisor y el receptor. En la figura 35 se muestra la equivalencia entre un filtro de coseno alzado y 2 filtros raíz de coseno alzado.

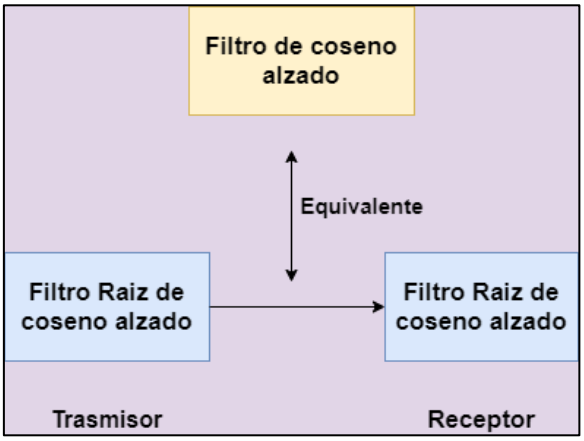


Figura 35. Equivalencia entre filtros

Al conectar dos filtros raíz de coseno alzado en serie, podemos obtener la misma respuesta que si utilizásemos un solo filtro de coseno alzado. De esta forma, podemos dividir el modelado de nuestros pulsos entre el transmisor y el receptor.

Por último, mostramos el código Python del filtro raiz de coseno alzado implementado en nuestro transmisor.

```
def rc_filt(beta, span, sps):

    t = np.arange(0,(span*sps)+1) - (span*sps)/2
    h = np.zeros((span*sps)+1)

    h[t==0] = (1/np.sqrt(sps))*(1-beta+4*beta/np.pi)
    h[abs(t)==(sps/(4*beta))] =
(beta/(np.sqrt(2*sps)))*((1+2/np.pi)*np.sin(np.pi/(4*beta))+(1-
2/np.pi)*np.cos(np.pi/(4*beta)))
    t_aux = t[(t!=0)*(abs(t)!=(sps/(4*beta)))]
    h[(t!=0)*(abs(t)!=(sps/(4*beta)))] =
(1/np.sqrt(sps))*(np.sin(np.pi*t_aux*(1-
beta)/sps)+4*beta*t_aux*np.cos(np.pi*t_aux*(1+beta)/sps)/sps)/(np.pi*t
_aux*(1-(4*beta*t_aux/sps)**2)/sps);
    return h
```

8.3 Implementación de un receptor BPSK.

Durante una transmisión real, la información transmitida sufre una serie de efectos perjudiciales para la comunicación. La principal función de un receptor es recuperar con éxito la información transmitida aproximando al máximo la información recibida a la transmitida. Para poder recuperar la información original de la manera más precisa posible, procesaremos las muestras recibidas a lo largo de los bloques mostrados en la figura 36.

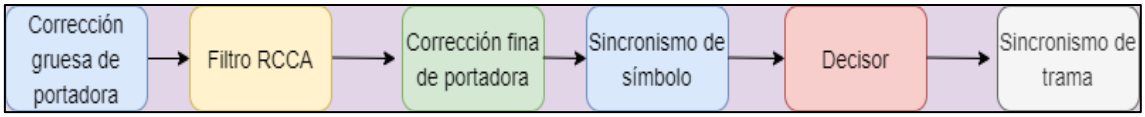


Figura 36. Diagrama de bloques de un receptor BPSK

8.3.1 Corrección gruesa de portadora

Durante una transmisión real, nuestra señal puede sufrir desviaciones frecuenciales que provocarán pérdida de información. Esta desviación frecuencial será corregida mediante la corrección gruesa y fina de portadora.

Para implementar la recuperación gruesa, se ha implementado un algoritmo fundamentado en la FFT. Si nos fijamos en una señal BPSK, podemos preciar que las componentes de la fase están separadas por un ángulo igual a π. Al elevar al cuadrado las muestras de una modulación BPSK, convertiremos la diferencia de fase π, en una diferencia de 2π, solapando así las fases de los símbolos en el mismo punto. En la figura 37 se muestra como las fases de 0 y π se solapan en 0 al elevar las muestras al cuadrado.

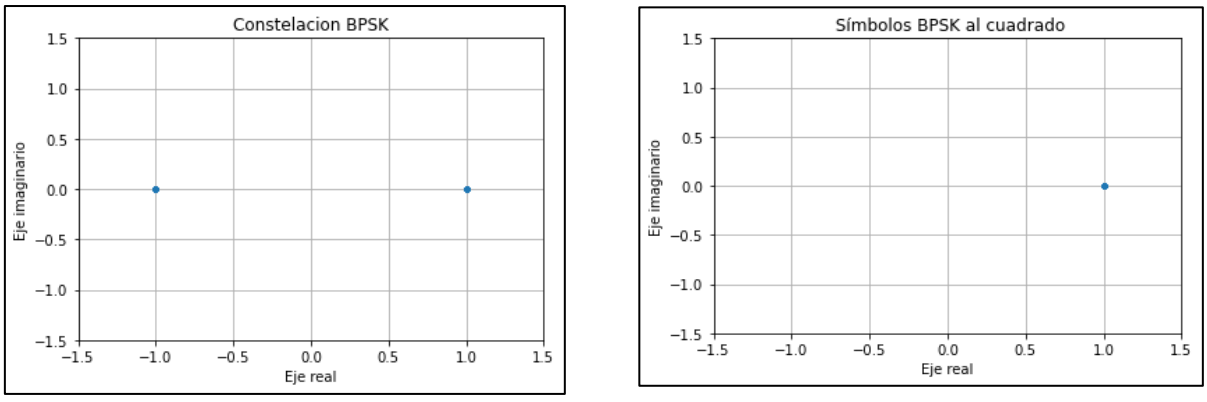


Figura 37. Comparativa entre constelaciones

El proceso explicado anteriormente, puede ser utilizado para cualquier orden M de una modulación PSK, utilizando la siguiente expresión [10]:

$$r^M(k) = S^M(k) * e^{j(2\pi fokT+\Theta)*M}$$

Esto provocará el desplazamiento de la desviación frecuencial a M veces su posición original, provocando que los símbolos sean puramente reales o complejos. Además, obteniendo la FFT de $r^M(t)$, y visualizando su espectro, podremos encontrar un tono situado en M veces la desviación frecuencial original.

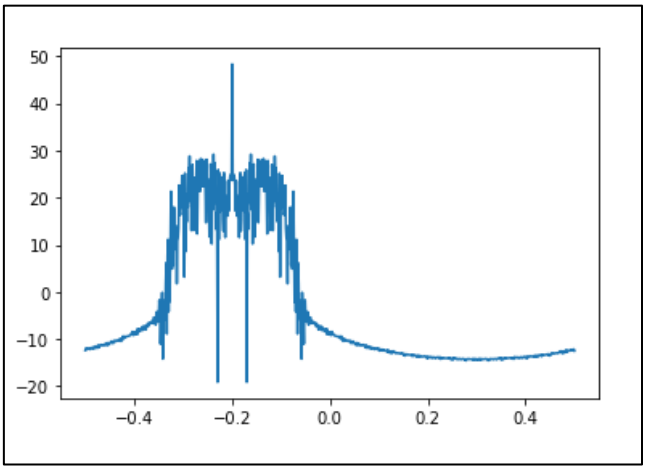


Figura 38. Espectro de los símbolos elevados al orden de la modulación

En la figura 38 se muestra el espectro de los símbolos BPSK elevados al orden de la modulación tras haber incorporado una desviación frecuencial igual a -0,1. Podemos apreciar un pico situado en frecuencia digital igual a -0,2, correspondiente al desplazamiento de 2 veces la desviación frecuencial. Una vez localizado el pico de este espectro, habremos descubierto la desviación de frecuencia gruesa sufrida por nuestra señal y podremos recuperarla en frecuencia 0.

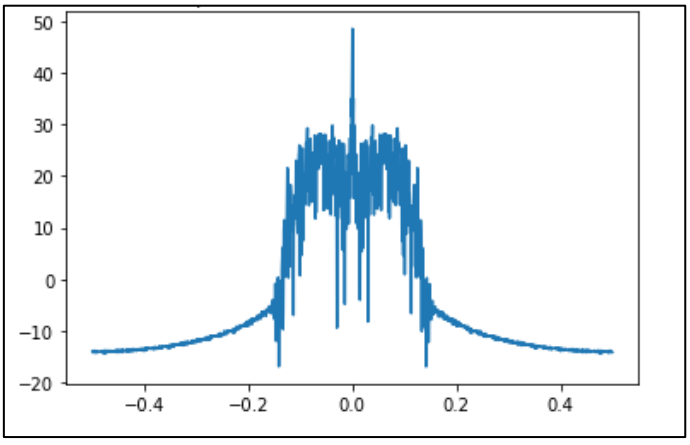


Figura 39. Espectro a la salida de la corrección gruesa

Para desarrollar el código Python del algoritmo descrito anteriormente y en el caso particular de una modulación BPSK, se ha seguido el diagrama de bloques mostrado en la figura 40.

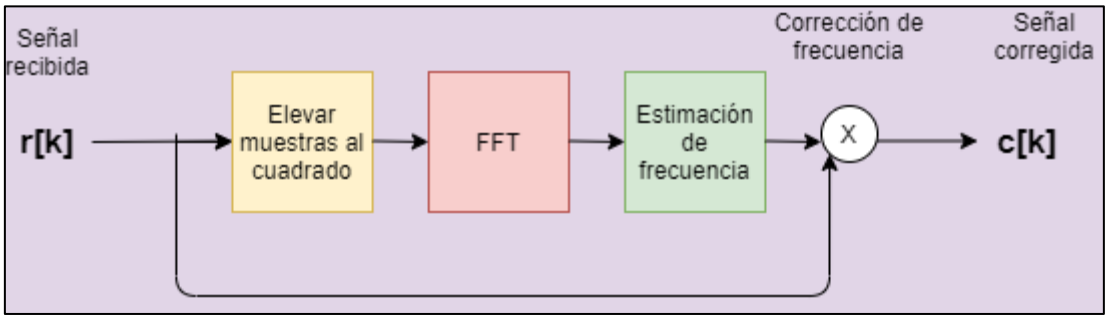


Figura 40. Diagrama de bloques de la recuperación gruesa de frecuencia para BPSK

Donde la estimación de frecuencia es generada por el proceso descrito anteriormente y la corrección se lleva a cabo multiplicando la señal recibida por una exponencial compleja de frecuencia igual a la frecuencia estimada entre el orden de la modulación.

Dividimos la frecuencia estimada entre el orden de la modulación ya que, cuando elevemos las muestras al orden de la modulación, estaremos desplazando M veces la desviación en frecuencia y, por lo tanto, localizaremos la posición de la desviación multiplicada por M.

A continuación, se muestra el código Python responsable de la recuperación gruesa de portadora. Nótese que para adaptarlo a cualquier modulación en fase de distinto orden, es suficiente con elevar las muestras al orden de la modulación en lugar de al cuadrado.

```
M=2
srx=samples
samples=samples**M
psd = np.fft.fftshift(np.abs(np.fft.fft(samples)))
f = np.linspace(-1/2.0, 1/2.0, len(psd))
max_freq = f[np.argmax(psd)]
Ts = 1/fs
t = np.arange(0, len(samples))
srx = srx * np.exp(-2j*np.pi*max_freq*t/M)
```

8.3.2

8.3.3 Filtro RCCA.

Como se ha explicado anteriormente, podemos dividir la respuesta de un filtro coseno alzado en dos filtros raíz de coseno alzado. Modelando así la forma de los símbolos tanto en el transmisor como en el receptor, por lo que en el receptor incorporaremos un nuevo filtro adaptado.

8.3.4 Corrección fina de portadora.

A pesar de haber realizado una corrección gruesa de frecuencia de portadora y haber conseguido eliminar gran parte de la desviación frecuencial, todavía existirá un offset de frecuencia. El mero hecho de que exista una pequeña desviación en frecuencia puede provocar que la constelación se mantenga girando continuamente. El sentido y la velocidad de giro dependerán del signo y del valor de la desviación frecuencial, girando en sentido antihorario si la desviación es positiva y a mayor velocidad cuanto mayor sea esta desviación. En la figura 41 se muestra un ejemplo de una constelación rotada debido al offset de frecuencia.

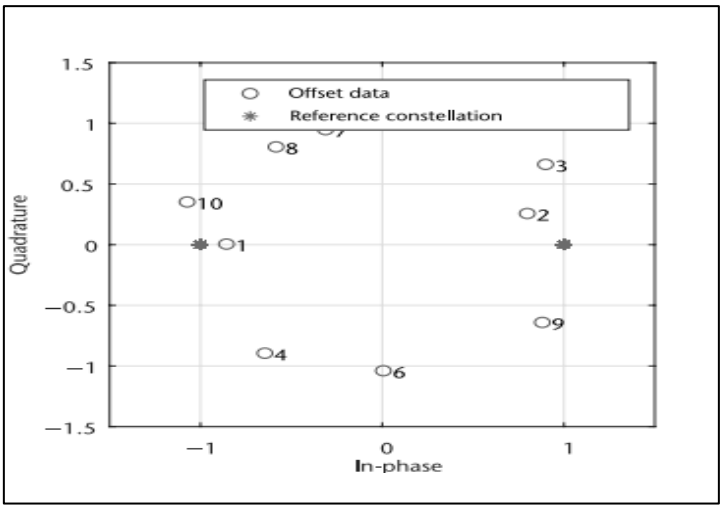


Figura 41. Constelación rotada debido al offset frecuencial. [11]

Para implementar la corrección fina de frecuencia, necesitamos un bucle cerrado que corrija la señal recibida muestra a muestra. Para ello, se ha escogido el bucle de costas, más conocido como Costas-Loop. Esta técnica está basada en un PLL digital y diseñada específicamente para la corrección de desplazamiento de portadora en señales digitales BPSK y QPSK [12]. En la figura 42 se muestra el diagrama de bloques de un bucle de Costas diseñado para una modulación BPSK.

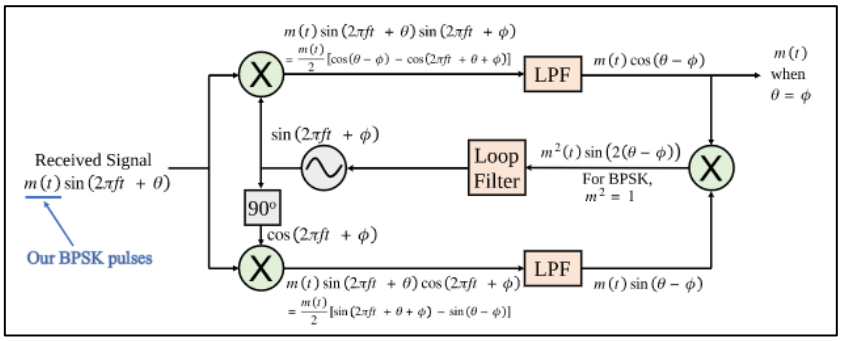


Figura 42. Diagrama de bloques del Costas-Loop. [12]

A continuación, se procede a explicar la función de cada uno de los bloques presentes en el diagrama.

Señal de error.

Los sistemas de bucle cerrado suelen calcular una señal de error en la que fundamentan las correcciones que realizan. El valor de la señal de error varía en función del tiempo y nuestro objetivo es acercarlo lo máximo posible a 0. En el momento en el que la señal de error tome el valor 0, podremos decir que nuestro sistema se ha "enganchado" y que el offset de frecuencia se habrá corregido completamente.

Filtro de lazo (Loop Filter).

El objetivo de este bloque es filtrar la señal de error calculada previamente. Su funcionamiento se describirá en mayor profundidad en el bloque del sincronismo de Símbolo.

Filtros paso bajo (LPF).

Los multiplicadores presentes en la implementación del bucle de Costas, introducirán repeticiones espectrales alejadas de la banda base que deberán ser eliminadas para su correcto funcionamiento. Para ello, se ha implementado un Filtro paso-bajo

A continuación, se muestra el código Python de un bucle de Costas.

```
N = len(recibidas)
phase = 0
freq = 0
alpha = 0.1
beta = 0.009
out = np.zeros(N, dtype=np.complex)
freq_log = []
for i in range(N):
    out[i] = recibidas[i] * np.exp(-1j*phase*2*np.pi)
    error = np.real(out[i]) * np.imag(out[i])
    freq += (beta * error)
    phase += (freq + (alpha * error))
```

8.3.5 Sincronismo de símbolo.

En una comunicación real, existen diversos factores que producen retardos sobre la señal transmitida (desfase entre los relojes del transmisor y el receptor, efectos del canal como Doppler…). Estos retardos temporales generan incertidumbre sobre cuál es el instante óptimo para realizar el muestreo de la señal. Muestrear en instantes equivocados provoca errores en la transmisión y no nos permitirá reconstruir la señal original de manera precisa. En las siguientes figuras se muestra una comparativa entre muestrear en el instante óptimo y en el instante equivocado la parte real de la señal recibida.

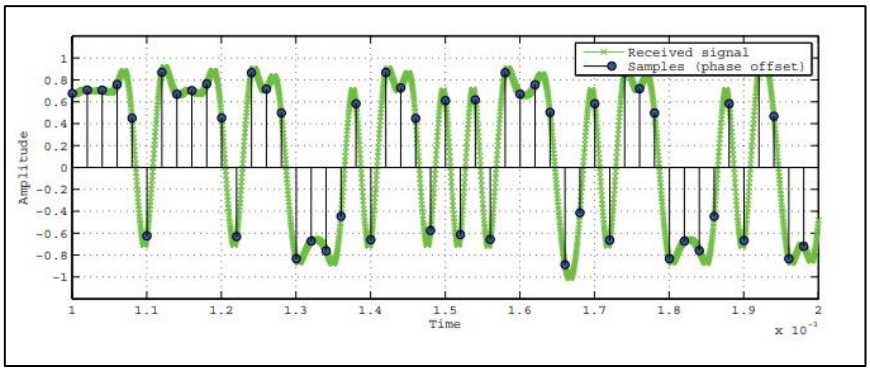


Figura 43. Muestreo de la señal recibida en instantes erróneos.[11]

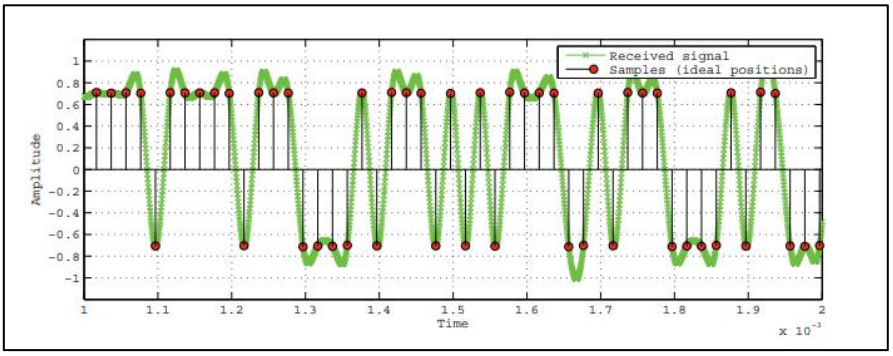


Figura 44. Muestreo de la señal recibida en instantes ideales.[11]

Comparando ambas figuras, apreciamos que, si muestreamos en el instante óptimo, la amplitud de todas las muestras será igual a ± 0.6, sin embargo, muestreando en instante equivocados, la amplitud variará entre distintos valores posibles, provocando que la consecución de un valor de cero ISI sea imposible, ya que los símbolos consecutivos no se muestrean en su cruce por 0.

Existen diversas soluciones para poder llevar a cabo el sincronismo de símbolo y conseguir que la señal recibida sea muestreada en los instantes óptimos.

Algunos protocolos de bajo nivel envían la señal de reloj del transmisor junto a los datos transmitidos para conseguir así una sincronización entre transmisor y receptor. La solución anterior presenta varios inconvenientes. Desde el punto de vista del transmisor, será necesario el uso de mayor potencia y ancho de banda, además de una lógica adicional en el receptor que se encargue de localizar las muestras correspondientes a la señal de reloj y sincronizar la señal de reloj del receptor, por lo que no es una solución del todo eficiente.

La corrección de tiempo que procedemos a implementar está basada en un PLL completamente digital. Un PLL, del inglés Phase Locked Loop, es un sistema realimentado que busca minimizar la diferencia entre la fase de la señal de entrada y la señal de salida utilizando una señal de error.

Los bloques necesarios para implementar la solución anterior son los siguientes:

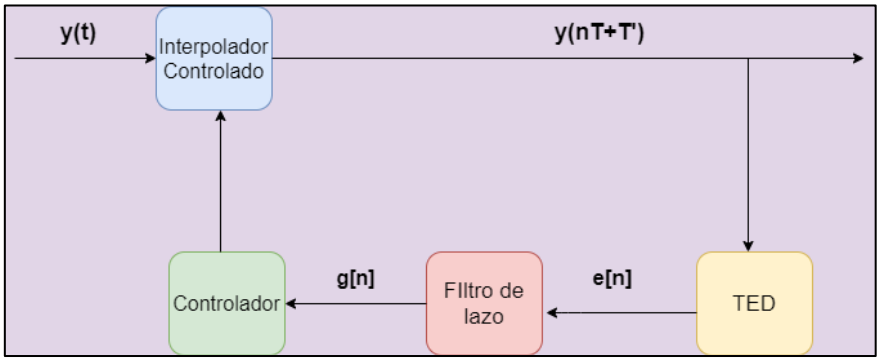


Figura 45. Diagrama de bloques del sincronismo de símbolo

A continuación, se procede a explicar el funcionamiento y la implementación de cada uno de los bloques.

8.3.5.1 Interpolador controlado

La diferencia entre un interpolador controlado y un interpolador corriente reside en que el factor de interpolación variará a lo largo del tiempo con el objetivo de obtener la muestra del instante óptimo del muestreo. Esta variación nos permitirá acercarnos de manera iterativa al instante óptimo del muestreo y será controlada por un factor μ[n] generado por el controlador. El factor μ[n] nos indicará la diferencia entre el instante de muestreo actual y el instante de muestreo ideal.

Para desarrollar interpolador controlado, se ha utilizado un filtro FIR paso bajo, conocido en inglés como Piecewise Polinomial Filter (PPF), implementando una interpolación polinómica. La salida genérica de este interpolador es la siguiente [10]:

$$y(kTs + \mu(k)Ts) = \sum_{n=1}^{2} hk(n)y((k-n)Ts))$$

Con hk igual a los coeficientes del filtro interpolador en el instante k, los cuales vienen determinados por la siguiente expresión [10]:

$$h = [\alpha\mu(k)(\mu(k)-1), -\alpha\mu(k)^2 - (1-\alpha)\mu(k) + 1,$$

$$-\alpha\mu(k)^2(1+\alpha)\mu(k), \alpha\mu(k)(\mu(k)-1)]$$

Donde $\alpha = 0{,}5$ y $\mu(k)$ representa el retardo temporal, generado por el controlador del interpolador.

8.3.5.2 Timing Error Detector (TED)

Bloque encargado de calcular la señal de error utilizando la salida del interpolador controlado. La señal de error puede obtenerse utilizando distintos algoritmos que variarán su calidad en función de la estructura de los datos transmitidos. El algoritmo utilizado en nuestra implementación es el conocido como *"Zero Crossing"*. Este algoritmo busca los cruces por 0 en el diagrama de ojos para ajustar el factor de interpolación y obtener el instante óptimo del muestreo, obteniendo la siguiente señal de error [10]:

$$e[n] = Im\left(y\left(\left(n-\tfrac{1}{2}\right)Ts + \text{ţ}\right)\right) * [sgn\left\{Re\left(y\left(\left(n-\tfrac{1}{2}\right)Ts + \text{ţ}\right)\right)\right\} -$$

$$sgn\{Re(y(nTs + \text{ţ}))\}] + Im\left(y\left(\left(n-\tfrac{1}{2}\right)Ts + \text{ţ}\right)\right) * [sgn\left\{Im\left(y\left(\left(n-\tfrac{1}{2}\right)Ts + \text{ţ}\right)\right)\right\} - sgn\{Im(y(nTs + \text{ţ}))\}]$$

Donde y es la señal de entrada al TED, Im y Re obtienen la parte imaginaria y real de la muestra y la función sgn obtiene el signo de la parte real/imaginaria correspondiente a la muestra procesada. Existen otras señales de error alternativas generadas por distintos algoritmos como Gardner, Müller and Mueller o Early-Late[10].

8.3.5.3 Filtro de lazo

La señal de error generada por el TED atraviesa un filtro de lazo que hemos implementado con un filtro proporcional más un integrador, cuyo diagrama de bloques es el siguiente.

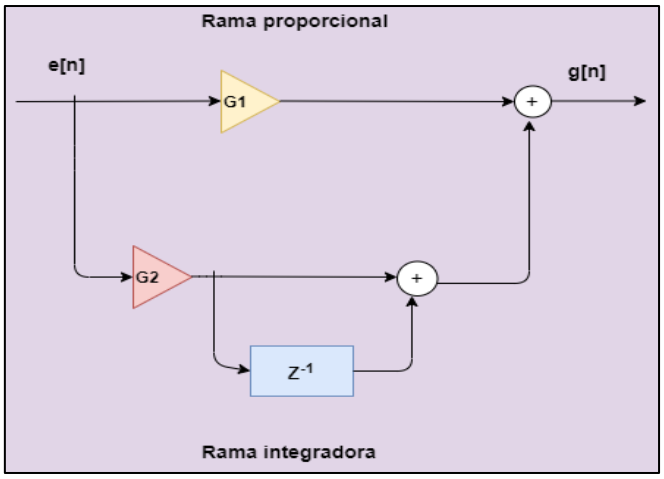


Figura 46. Diagrama de bloques del filtro de lazo

Además, si declaramos variables intermedias en distintos puntos del diagrama de bloques y conseguimos expresar la salida del sistema en función de la entrada, podemos obtener la función de transferencia $H(z)$ realizando el cociente de la salida entre la entrada. Operando, obtenemos:

$$H(z) = G1 + G2 * \frac{1}{1 - z^{-1}}$$

Donde $G1$ es la ganancia proporcional y $G2$ la ganancia integradora. Para elegir correctamente el valor de estas ganancias es necesario tener en cuenta los parámetros del filtro, ancho de banda del ruido y el factor de amortiguamiento utilizando las siguientes expresiones.

$$\theta = \frac{B_{Loop}}{M(\zeta + 0.25/\zeta)} \qquad \Delta = 1 + 2\zeta\theta + \theta^2$$

$$G_1 = \frac{4\zeta\theta/\Delta}{M} \qquad G_2 = \frac{4\theta^2/\Delta}{M}$$

Ganancias proporcional e integradora.[10]

Donde M es el número de muestras por símbolo, $BLoop$ es el ancho de banda de ruido, y ζ es el factor de amortiguamiento.

8.3.5.4 Controlador

El controlador es el bloque encargado de generar el factor μ[n] que controlará el factor de interpolación del interpolador controlado. Su implementación se ha realizado utilizando el siguiente diagrama de bloques:

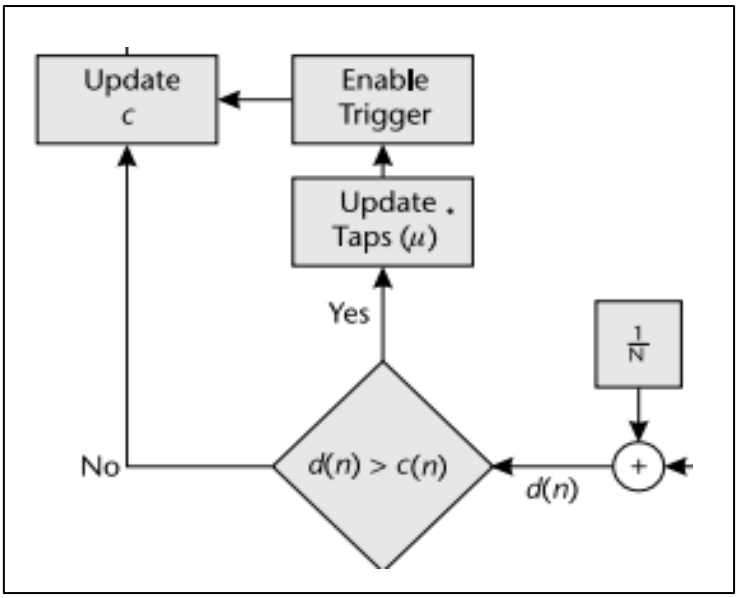


Figura 47. Diagrama de bloques del controlador.[10]

Este diagrama representa un contador de módulo 1 que actualiza el valor de μ cuando se cumple la condición de disparo. Este contador se implementa incrementando un factor de $\frac{1}{N}$ al recibir cada una de las muestras y siendo N el periodo del símbolo. En la situación donde el error toma el valor 0, tras N muestras se alcanza el máximo, se lanza la señal de disparo o de strobe y se reinicia el valor del contador al valor 0.

Por otro lado, en el caso en el que exista error muestreo, se aplicará un factor de corrección que acelere o ralentice el incremento del contador, obteniendo así un paso $d[n] = g[n] + \frac{1}{N}$, donde $g[n]$ es la salida del filtro de lazo. En la siguiente figura se muestra el funcionamiento del contador de módulo 1.

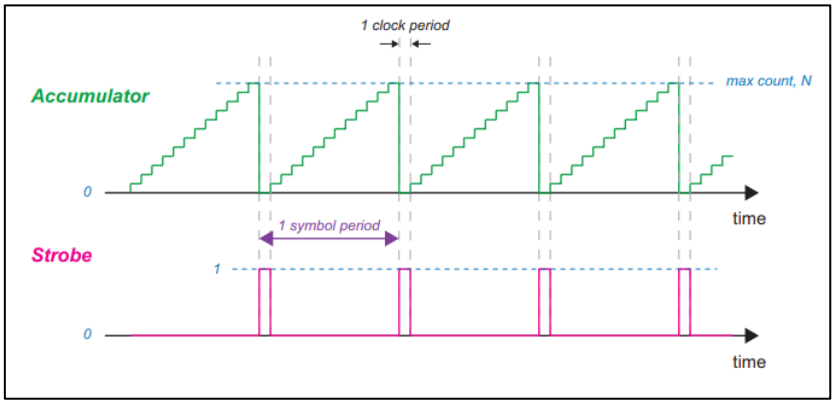


Figura 48. Funcionamiento del contador de módulo 1.[11]

Finalmente, cuando se cumple la condición de disparo, el factor μ[n] se actualiza como el cociente entre el paso de la actualización y el valor del contador.

A continuación, se muestra el código Python encargado del sincronismo de símbolo.

```
for i in range(len(entrada)):
        #Interpolador controlado
        yseq=np.concatenate((entrada[i].reshape(1,1),ifstate))
        x=np.dot(coeficientes,yseq)
        salidafiltro=sum(x*vectormu)
        salidassimb[i]=salidafiltro
        tedbuffer.append(salidafiltro)
        ifstate=yseq[0:-1]

        if(trigger==1):

            end=len(tedbuffer)-1
            posicion1=int((end/2)+1-N%2)
            posicion2=int((end/2)+1)
            t1=tedbuffer[posicion1-1]
            t2=tedbuffer[posicion2-1]
            midsample=(t1+t2)/2
            preal=np.real(midsample)*(np.sign(np.real(tedbuffer[0]))-
            np.sign(np.real(salidafiltro)))
            pimag=np.imag(midsample)*(np.sign(np.imag(tedbuffer[0]))-
            np.sign(np.imag(salidafiltro)))
            error=preal+pimag
        else:
            error=0
        #Filtro de lazo
        w=error*g2+want
        want=w
        g=w+error*g1
        d=g+1/N
        #Controlador
        if(counter<d):
            trigger=1
            mu=counter/d
            listamu.append(mu)
        else:
            trigger=0
        counter=(counter-d)%1
    sincsimb=salidassimb
```

8.3.6 Decisor.

Tras el bloque anterior, debemos recuperar las posiciones originales de los símbolos en la constelación inicial. Debido a la presencia de distintos factores como el ruido, la posición de los símbolos en la constelación puede moverse de la circunferencia unidad y no tomar los valores de ± 1. Para que los símbolos recuperen su valor original, es necesario implementar una lógica adicional encargada de identificar qué símbolos recibidos corresponden a un $+1$ y qué símbolos corresponden a un -1.

Al tratarse de una modulación BPSK, el decisor tiene una fácil implementación y solo necesita conocer el signo de la parte real de los símbolos. De esta manera, si un

símbolo presenta parte real positiva o negativa, corresponderá a un 1 o un -1 respectivamente. El decisor BPSK ha sido implementado mediante el siguiente código Python.

```
for i in range(len(muestras)):
    if np.real(muestras[i])>0:
        sal[i]=1
    else:
        sal[i]=-1
```

8.3.7 Sincronismo de trama

El último bloque referido a la sincronización consiste en el sincronismo de trama. La estructura de la trama de numerosos protocolos de comunicaciones suele contener un preámbulo que facilite al receptor la detección del comienzo de la trama.

La eficiencia del sincronismo de trama variará en función del preámbulo diseñado. Para que un preámbulo sea fácilmente detectable, ha de presentar propiedades de correlación favorables [10].

De esta manera, podremos localizar el inicio de la trama calculando la función de autocorrelación de la trama recibida con el preámbulo conocido, ya que la función de autocorrelación presenta un máximo en la posición donde se alinean los preámbulos.

En la figura 49 se muestra la detección del preámbulo conocido mediante la función de autocorrelación. Podemos apreciar que se produce un máximo situado en el punto de alineamiento.

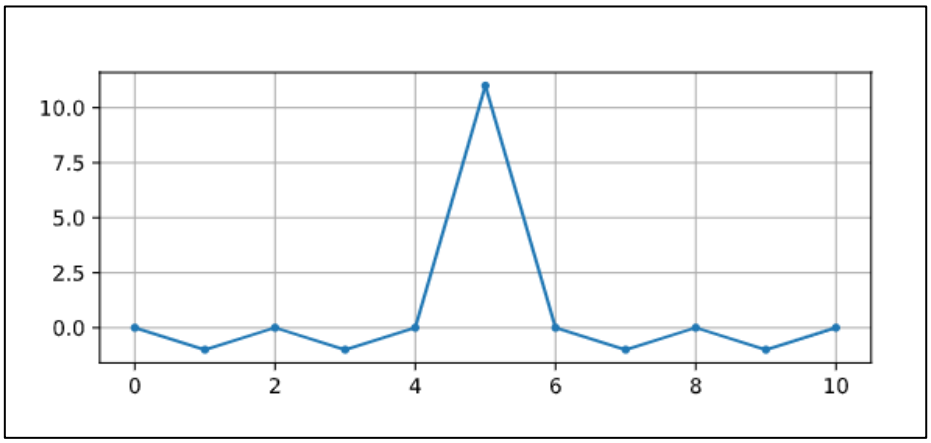


Figura 49. Detección del preámbulo conocido.[12]

9 Análisis de los resultados

9.1 Modulación AM con la ADALM-PLUTO.

Para realizar la modulación AM con la ADALM-PLUTO utilizaremos un archivo de audio con extensión *".waw"* como señal moduladora y una exponencial compleja como señal portadora. Para convertir a un vector de muestras una archivo con extensión *".wav",* utilizamos la función *"read()"* de la librería *"scipy.io.wavfile".* Esta función, nos devuelve un vector de muestras y la frecuencia a la que ha sido muestreado nuestro archivo de audio.

Por lo general, un archivo de música suele muestrearse a 44.1 kHz con el objetivo de evitar el solapamiento de las repeticiones espectrales resultantes del muestreo. No obstante, la señal elegida para realizar la modulación ha sido muestreada a 48 kHz.

En el siguiente código Python mostramos la lectura del archivo".*wav*" correspondiente a nuestra señal moduladora.

```
import scipy.io.wavfile as waves
fs, moduladora = waves.read('Ctangana.wav','False')
moduladora=moduladora/max(moduladora)
```

La señal moduladora se trata de una señal de audio con una duración de 30 segundos. Para garantizar la recepción completa de la señal de audio, debemos configurar el buffer de trasmisión en modo cíclico, con una longitud igual a la longitud de la señal modulada, y el buffer de recepción con una longitud igual a 2 veces la señal modulada. En la siguiente figura se muestra el espectro de la señal moduladora representado en frecuencia digital normalizada.

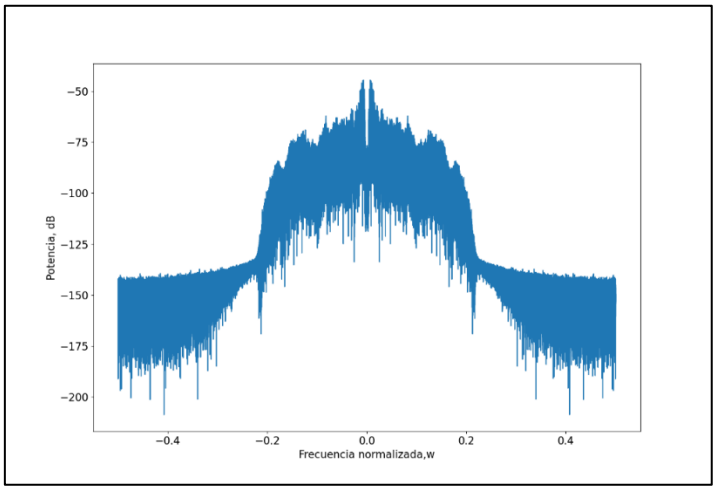


Figura 50. Espectro de la señal moduladora normalizada

Cuando la señal modulada ha sido correctamente procesada, es el momento de realizar la modulación en amplitud para obtener la señal modulada. Inyectando la señal moduladora en el transmisor AM descrito en la figura 16, obtenemos a la salida una señal modulada de Amplitud por una exponencial compleja como portadora. Para realizar la transmisión se ha utilizado una frecuencia de portadora con un valor de 0,25.

En la siguiente figura se muestra el espectro de la señal modulada en amplitud.

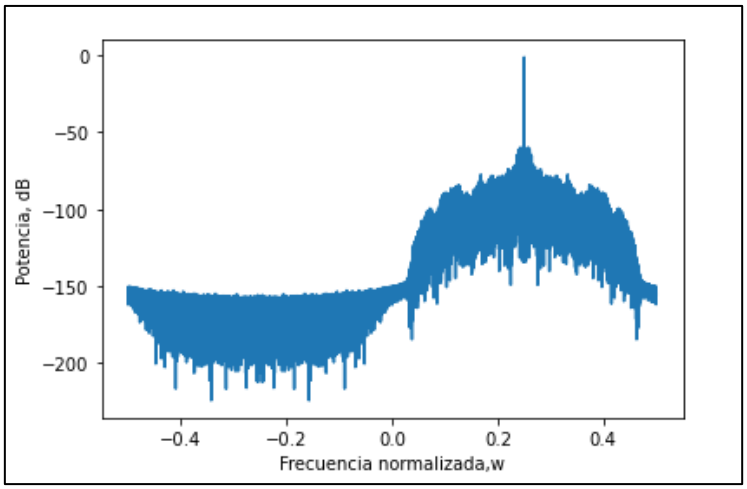


Figura 51. Espectro de la señal modulada con una exponencial compleja

Una vez atravesado el transmisor, procedemos a escribir en el buffer de recepción la señal modulada para llevar a cabo su transmisión con la ADALM_PLUTO.

La transmisión ha sido realizada en las siguientes condiciones:

- Frecuencia de transmisión: 1 GHz
- Ancho de banda configurado en el transmisor y en el receptor: 5 MHz
- Frecuencia de configurada en el transmisor y en el receptor: 3 MHz

Una vez transmitida la señal modulada, procedemos a leer del buffer de recepción las muestras recibidas. Debido a la configuración cíclica de nuestro buffer de transmisión, el buffer de recepción tendrá una longitud mayor al de transmisión. La configuración descrita anteriormente garantizará la recepción completa de la señal transmitida, pero provocará que el vector recibido contenga intervalos de ruido que no corresponden a la señal modulada. Este fenómeno se debe a que el receptor escuchará ruido antes de empezar a recibir información del transmisor.

En la siguiente figura encontramos la representación de la señal recibida normalizada en amplitud.

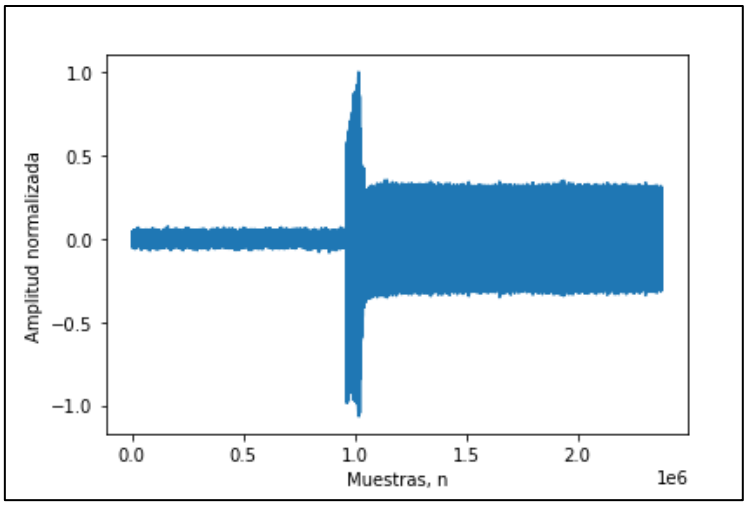


Figura 52. Muestras captadas por el receptor de la ADALM-PLUTO

La señal recibida está compuesta por dos intervalos claramente diferentes. El primero de ellos, formado por muestras de amplitud reducida, corresponde a la captura de ruido previa a recibir la señal modulada. El momento en el que se produce un aumento de la amplitud, corresponde al instante de tiempo a partir del cual el receptor comienza a recibir la señal modulada en amplitud.

Este efecto podría eliminarse implementando una transmisión en tiempo real, ya que el transmisor y el receptor actuarían de forma completamente sincronizada, por lo que el receptor escucharía únicamente durante los periodos de transmisión. Esto garantizaría que el receptor no captase ruido antes de recibir las muestras correspondientes a la señal transmitida. Una vez recibidas las muestras, calculamos su Transformada de Fourier para analizar sus componentes frecuenciales. En la siguiente figura se muestra la representación espectral en frecuencia normalizada de las muestras captadas por el receptor de la ADALM-PLUTO.

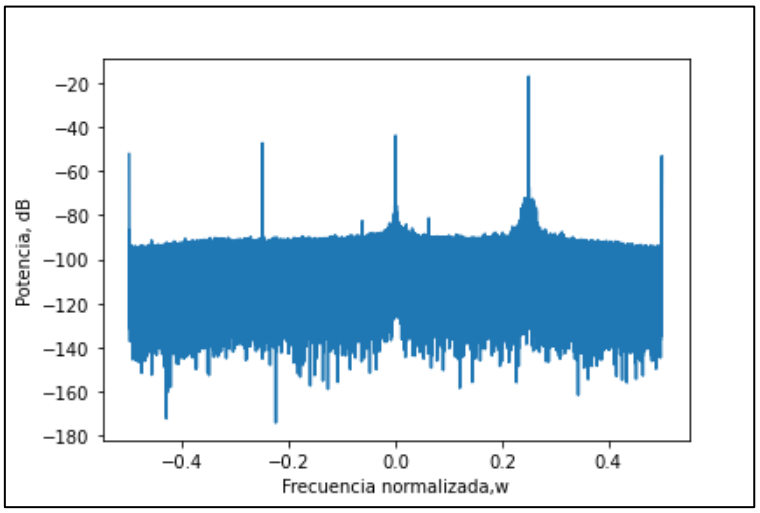


Figura 53. Espectro de las muestras captadas por la ADALM-PLUTO

Donde apreciamos un tono de potencia superior al resto de componentes frecuenciales, situada en una frecuencia igual a la frecuencia de portadora y correspondiente a la componente piloto de la señal modulada.

Además, observamos varios picos de potencia que no corresponden a las componentes frecuenciales de la señal modulada. Estos espurios se deben a diversos factores, entre ellos a la transmisión cíclica que se ha implementado.

A continuación, demodulamos la señal recibida y obtenemos su envolvente. En la siguiente figura se muestra la señal recibida centrada en frecuencia 0.

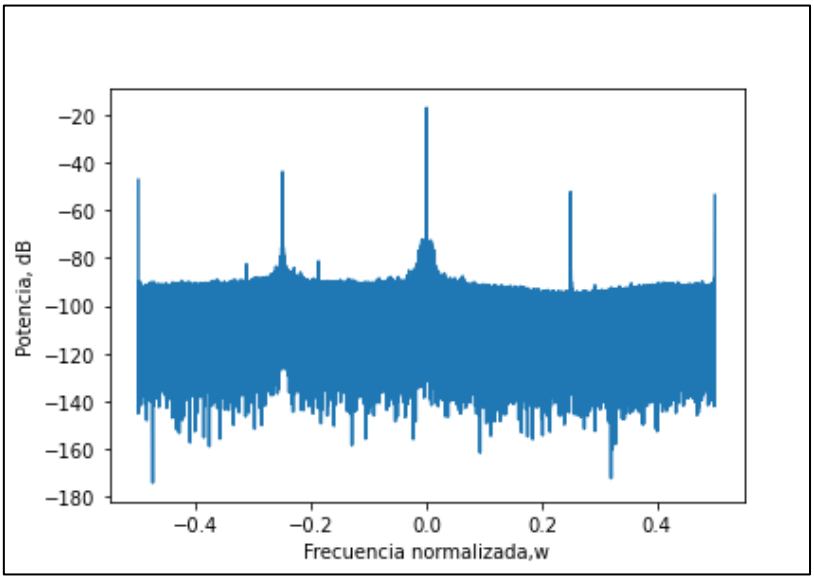


Figura 54. Espectro de la señal recibida en frecuencia 0

9.2 Caracterización de los filtros

Como se comentó en los primeros puntos del proyecto, la ADALM-PLUTO incorpora un filtro FIR programable, tanto en la cadena de recepción como en la de transmisión. Este filtro FIR está limitado en longitud ya que ha de estar compuesto por un máximo de 128 coeficientes y es completamente configurable por el usuario. En el siguiente apartado se realizará la configuración de ambos filtros y se analizará su respuesta en frecuencia. Al igual que ocurría con el filtro de coseno alzado y su equivalencia con dos filtros RCCA, podemos expresar la respuesta en frecuencia de estos filtros configurables como la multiplicación de sus respuestas en frecuencia por individual.

Para llevar a cabo el análisis, se implementará un filtro FIR utilizando el algoritmo de intercambio de Remez y mediante la fórmula de diseño de Parks-McClellan.

A continuación, para evaluar la respuesta en frecuencia de los filtros, se realizará un proceso iterativo donde se transmitirá una exponencial compleja que variará su frecuencia en cada una de las iteraciones.

De esta forma, se llevará a cabo un barrido a lo largo de todo el espectro digital con el fin de evaluar la señal recibida y obtener la respuesta en frecuencia de los filtros.

En la siguiente figura se muestra la representación del tono utilizado para medir la respuesta en frecuencia de los filtros.

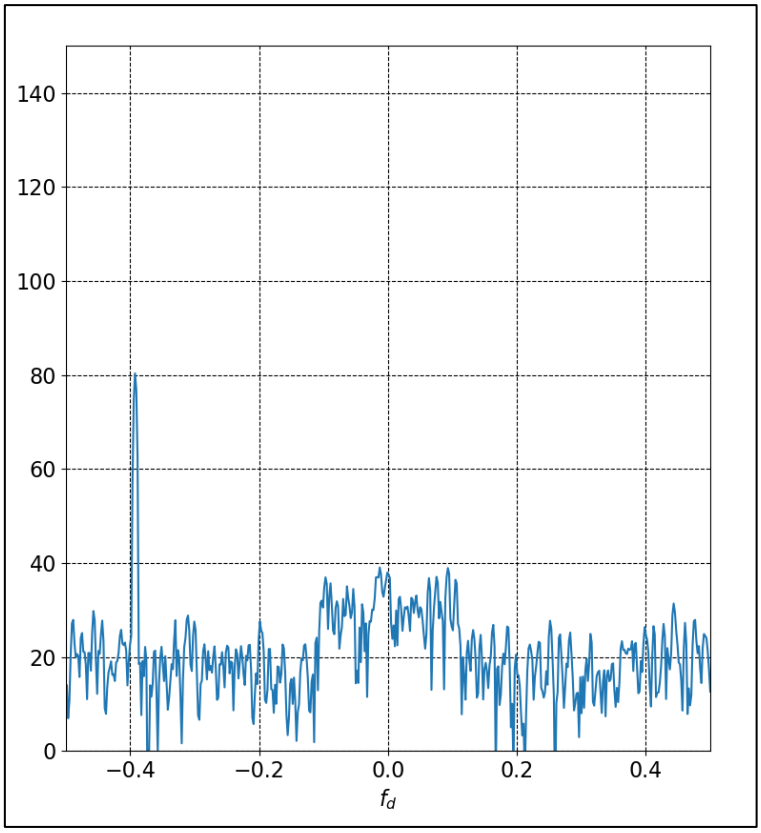


Figura 55. Representación del barrido frecuencial.

En primer lugar, se implementará un filtro FIR paso bajo sobre el cual se realizarán modificaciones para observar las diferencias. El filtro FIR inicial cuenta con las siguientes características:

- 64 coeficientes
- Frecuencia de corte digital de 0.1
- Rizado en la banda de paso de 0.05 dB
- Atenuación de 30 dB

Una vez creado el filtro FIR, lo incorporamos tanto a la cadena de recepción como a la cadena de transmisión. En la siguiente figura se muestra la respuesta en frecuencia obtenida tras realizar este ejemplo.

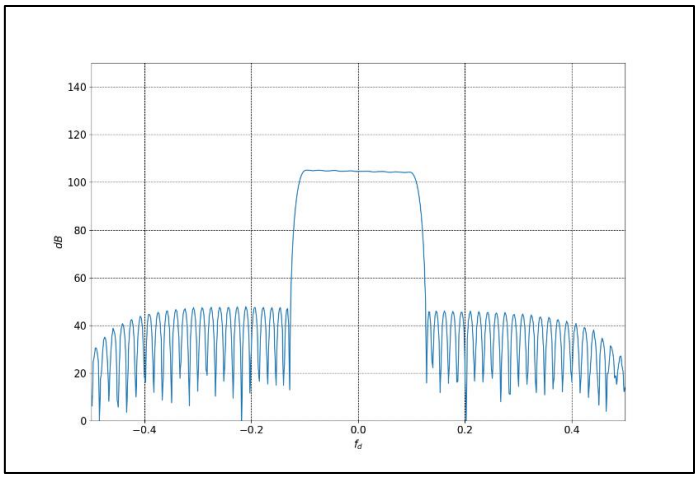


Figura 56. Respuesta en frecuencia con 60 dB de atenuación

Nótese que la respuesta en frecuencia obtenida en la figura anterior presenta una atenuación de unos 60 dB respecto a la banda de paso. Si apreciamos la implementación inicial del filtro, se especifica una atenuación de 30 dB por cada uno de los filtros. Al estar el transmisor y el receptor conectados mediante un cable SMA-SMA, las atenuaciones de los filtros se suman ofreciendo así un valor de 60 dB.

En la siguiente figura, se muestra la realización del mismo experimento descrito en el punto anterior modificando el valor de la atenuación. En este caso, la atenuación que se ha configurado para cada uno de los filtros tiene un valor igual a 10 dB, por lo que la atenuación total que esperamos respecto a la banda de paso será de aproximadamente 20 dB.

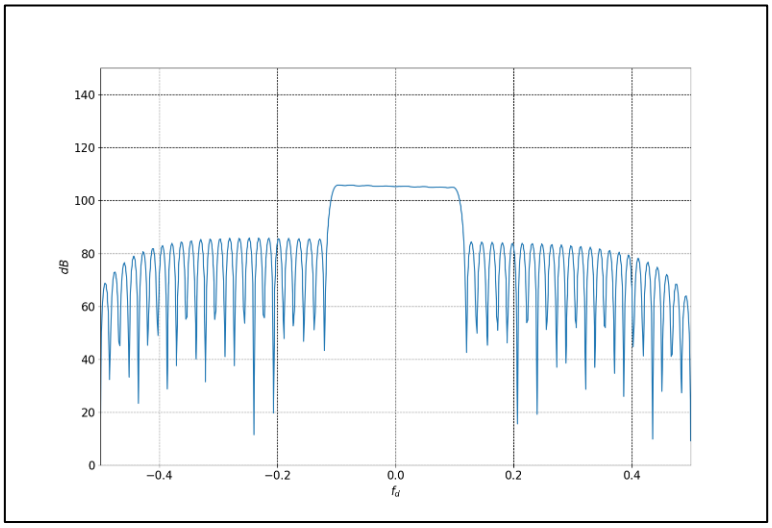


Figura 57. Respuesta en frecuencia con 20 dB de atenuación

Por último, se realiza un experimento especificando un nivel nulo de atenuación. En la siguiente figura se muestra la respuesta en frecuencia correspondiente, donde apenas hay cambio entre la potencia de la banda de paso y el resto de las frecuencias.

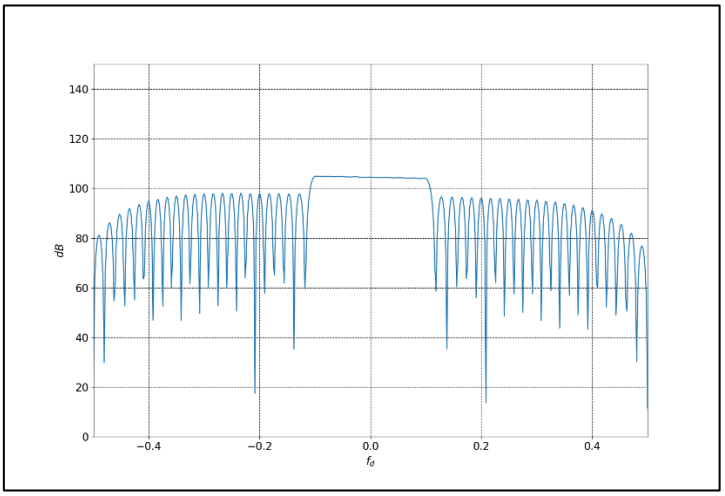


Figura 58. Respuesta en frecuencia sin atenuación

A continuación, se procede a realizar un nuevo experimento. Configuraremos únicamente el filtro correspondiente a la cadena de recepción, utilizando las mismas características descritas en el primer ejemplo.

Si configuramos el filtro del transmisor a partir de un vector de muestras con valor 1 en la primera posición y 0 en el resto de las posiciones, conseguiremos desactivar su interacción con la señal transmitida. En la siguiente figura encontramos la respuesta en frecuencia correspondiente a este nuevo experimento.

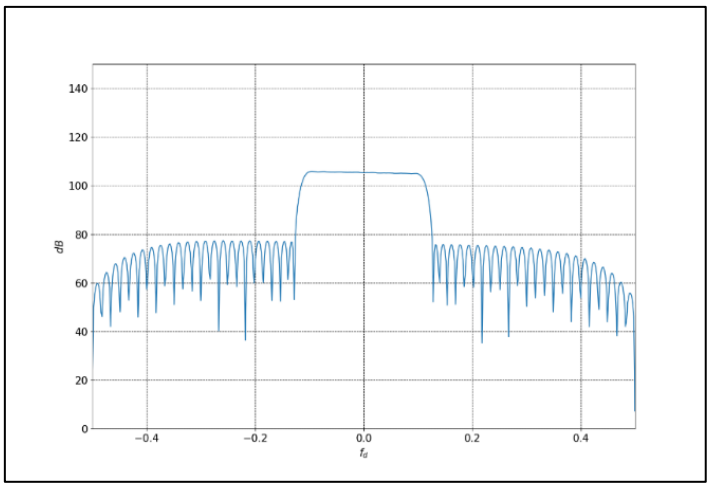


Figura 59. Respuesta en frecuencia

del filtro de recepción

Si comparamos el nivel de potencia en la banda de paso con el del resto de componentes frecuenciales, apreciaremos una caída de unos 30 dB aproximadamente.

Al desactivar el filtro correspondiente al transmisor, la atenuación respecto a la banda de paso estará compuesta únicamente por la atenuación que sea capaz de incorporar el filtro del receptor, ya que la señal transmitida habrá atravesado el transmisor sin atenuación alguna.

El siguiente experimento consiste en variar la longitud del filtro manteniendo el resto el valor del resto de parámetros. El algoritmo Remez utiliza la fórmula de diseño de Parks-McClellan para calcular los coeficientes del filtro óptimo. En la siguiente expresión se muestra implementación en Python de dicha fórmula de diseño donde se calcula la banda de transición del filtro a implementar:

```
Df = (-10*np.log10(rp*att)-13)/(14.6*(N-1))
```

Donde `Df` es la banda de transición del filtro, `rp` es el rizado de la banda de paso, `att` es la atenuación y `N` es el número de coeficientes del filtro. Fijándonos en la fórmula de diseño, observamos que el denominador corresponde a un factor proporcional a la longitud del filtro.

Si disminuimos el valor de dicho parámetro manteniendo el resto constantes, provocaremos una disminución del denominador y por consiguiente, un aumento del cociente y de la banda de paso. En la siguiente figura se muestra la respuesta de los filtros utilizando los parámetros del primer ejemplo y una longitud del filtro de 16 coeficientes, donde observamos una mayor banda de paso.

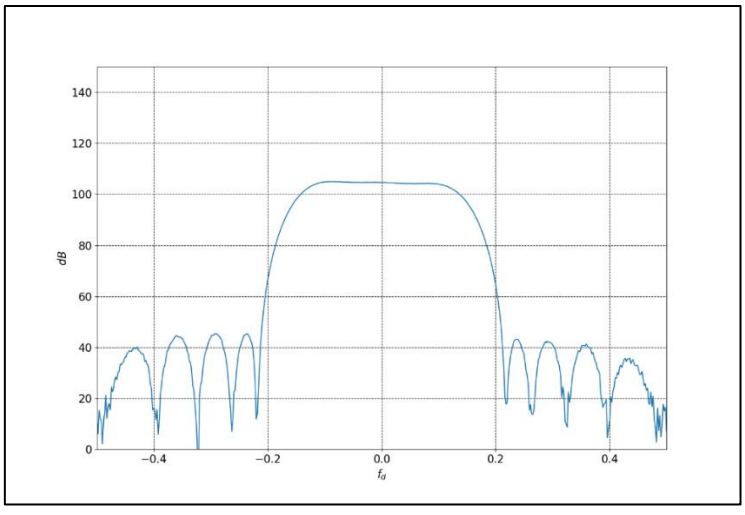


Figura 60. Respuesta en frecuencia usando 16 coeficientes

Por último, se muestra una figura donde se aprecia un segundo pico de potencia, correspondiente a la aparición de la banda imagen.

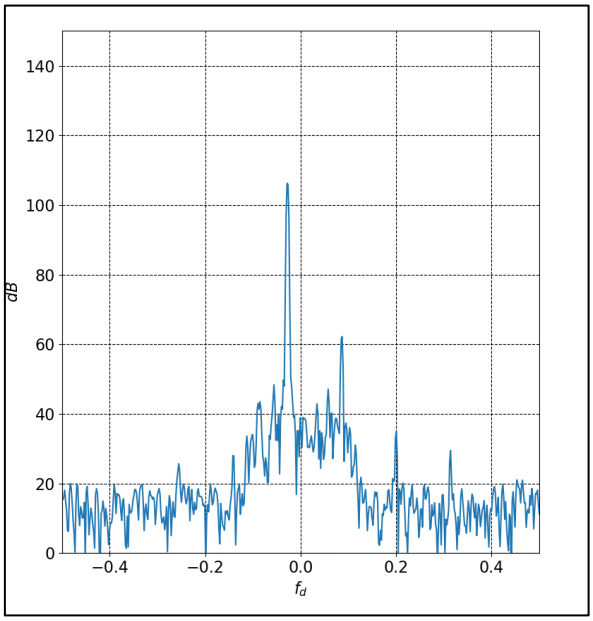


Figura 61. Aparición de la banda imagen

al atravesar frecuencias bajas

Debido a la propiedad de la periodicidad del espectro digital, la transmisión de una señal en una frecuencia cercana a la banda base, puede provocar que las repeticiones espectrales del espectro trasmitido aparezcan, perjudicando así la calidad de la recepción.

Es por ello, por lo que, para realizar una comunicación de calidad, se desplaza el espectro de la señal transmitida a una frecuencia de portadora, alejándolo así de la banda base y evitando las apariciones de la banda imagen.

9.3 Simulación de una modulación BPSK mediante Python.

Una vez implementados los componentes del transmisor y del receptor BPSK, realizamos una simulación para comprobar el comportamiento de cada uno de los bloques. Una vez realizada la simulación, se procederá a realizar una modulación BPSK real utilizando el dispositivo *ADALM-PLUTO,* donde se incorporará una desviación frecuencial, un retardo temporal y un ruido complejo. Para realizar la simulación se han utilizado os siguientes parámetros:

Se han generado 100 símbolos, utilizando un periodo de símbolo de 8 muestras y una frecuencia de muestreo de 1 MHz. Durante la simulación se visualizarán las constelaciones a la salida de cada uno de los componentes para verificar su buen funcionamiento. En primer lugar, generamos los símbolos BPSK utilizando una fuente

aleatoria. La diferencia entre la fase de los símbolos toma un valor de π al tratarse de una modulación BPSK. En este momento todos los símbolos toman el valor de ± 1 y su representación en el plano real e imaginario viene representado en la Figura 62.

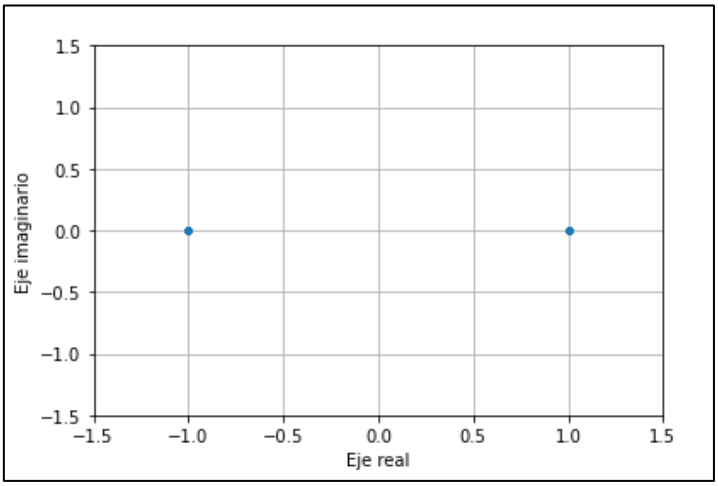


Figura 62. Constelación a la salida de la fuente aleatoria

Una vez generados los símbolos, debemos interpolarlos y posteriormente modelar su forma de onda con el filtro RRC, dando así por finalizado el bloque transmisor. En la figura 63 encontramos la constelación a la salida del transmisor.

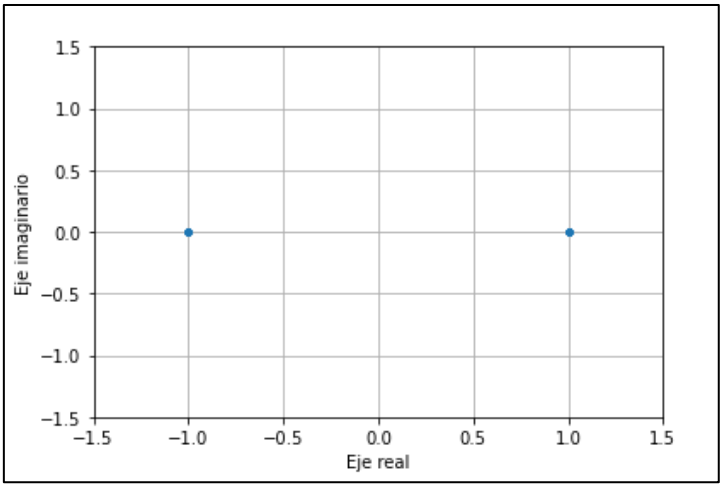


Figura 63.Constelación a la salida del transmisor

Una vez terminada la etapa del transmisor, simularemos una desviación frecuencial, un retardo temporal y una adición de ruido.

Para recuperar la constelación de la desviación frecuencial, utilizaremos los bloques de recuperación gruesa y fina de portadora. Tras la recuperación gruesa, el espectro de nuestros símbolos vuelve a estar centrado en frecuencia 0 y habremos recuperado parte del offset de frecuencia. Se muestra la constelación a la salida de este bloque en la figura 64.

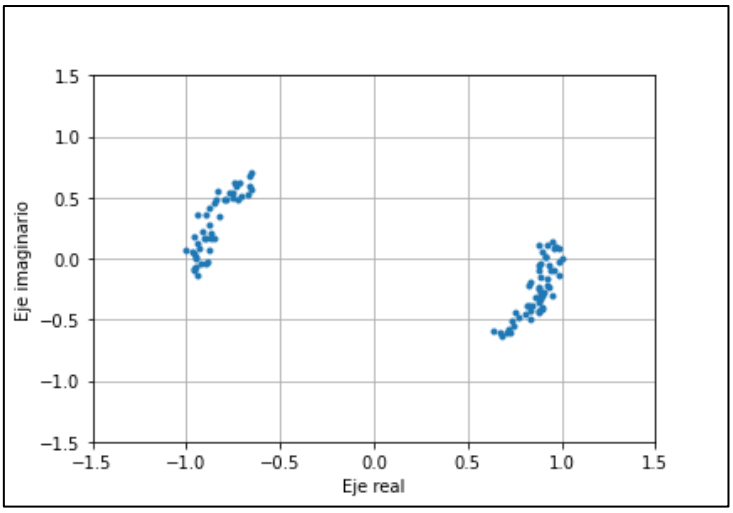


Figura 64.Constelación a la salida de la recuperación gruesa

Los puntos correspondientes a la constelación tras la desviación frecuencial han corregido sus posiciones y se han reagrupado en dos nuevas nubes de puntos situadas más cerca de los símbolos originales. Además, se observa una pequeña rotación de la costelación en el sentido horario, debido al pequeño offset de frecuencia que será corregido en el bloque de la recuperación fina. A continuación, inyectaremos la salida de la recuperación gruesa de frecuencia al bloque correspondiente a la recuperación fina. La recuperación fina se encargará de eliminar el offset de frecuencia provocando así que la constelación vuelva a sus posiciones originales. En la siguiente figura se muestra la constelación a la salida del bucle de costas.

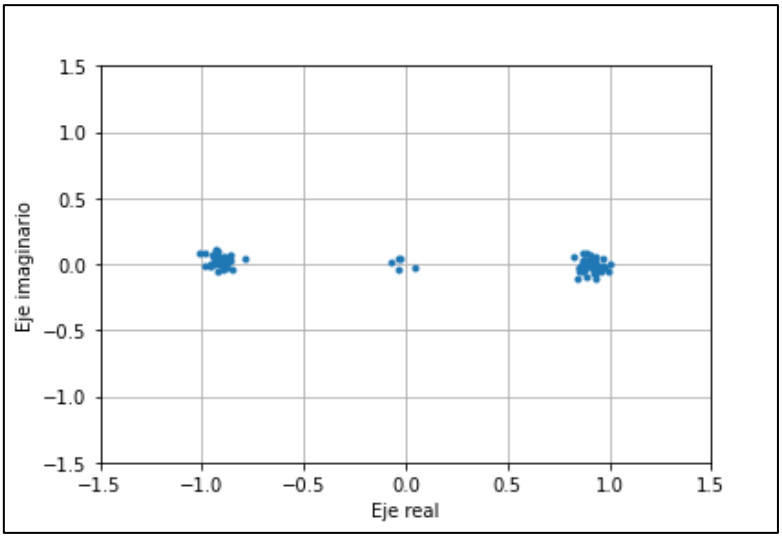


Figura 65. Costelación a la salida de la recuperación fina

Los símbolos han vuelto a reagruparse en dos nubes de puntos situados muy cerca de ± 1 y hemos conseguido recuperar por completo la frecuencia de portadora. A la salida de este bloque, la constelación es muy parecida a la inicial y solo necesitamos eliminar el retardo temporal. Además, encontramos algunos símbolos situados cerca de 0 que

corresponden a las primeras muestras procesadas por el bucle de costas, ya que necesita un número finito de muestras para “engancharse” y realizar la recuperación fina.

Por último, las muestras atraviesan el sincronismo de símbolo con el fin de eliminar el retardo temporal. Las siguientes figuras muestran la constelación y la parte real de la salida de este bloque.

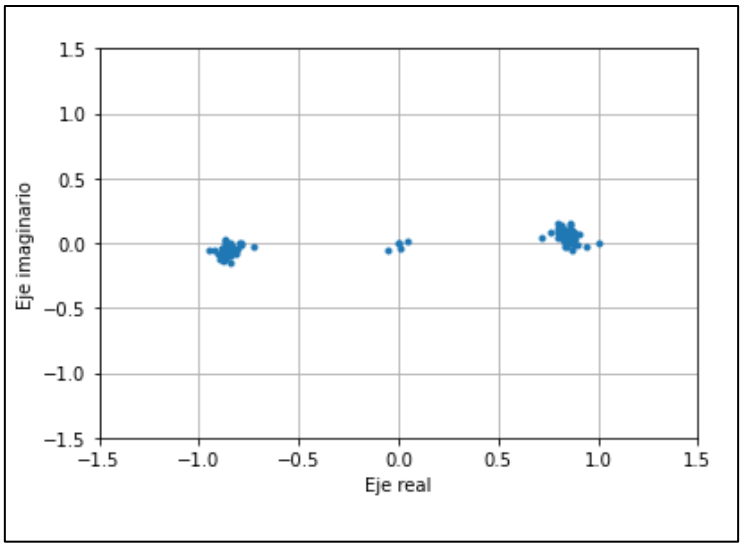


Figura 66. Constelación a la salida del sincronismo de símbolo

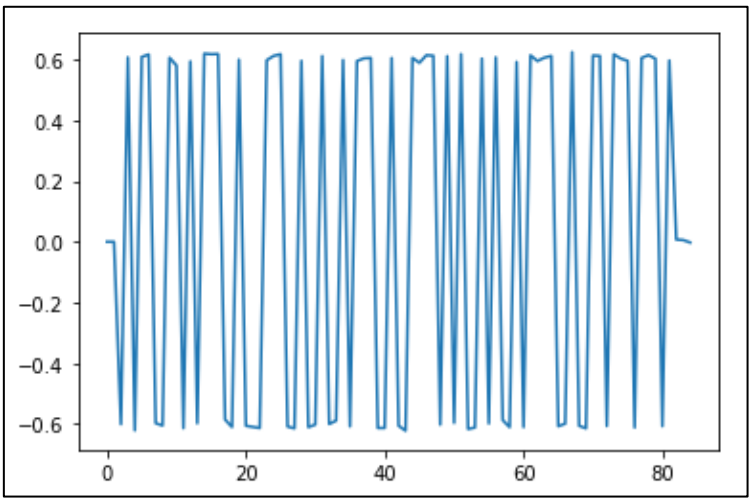


Figura 67. Símbolos tras el sincronismo de símbolo en presencia de ruido

Al igual que el ruido genera sobre la constelación nubes de puntos sobre las posiciones originales, en el dominio temporal podemos apreciar un pequeño error en la amplitud de cada uno de los símbolos. Para recuperar las amplitudes/posiciones originales, los símbolos atravesarán el bloque decisor.

En las siguientes figuras observamos la constelación y los símbolos a la salida del decisor. Podemos apreciar que las posiciones de los símbolos en la constelación y las amplitudes de los símbolos vuelven a ser las originales.

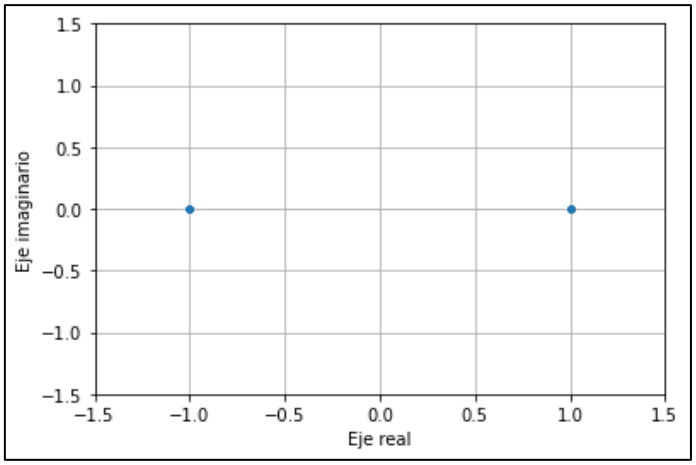


Figura 68. Constelación a la salida del decisor

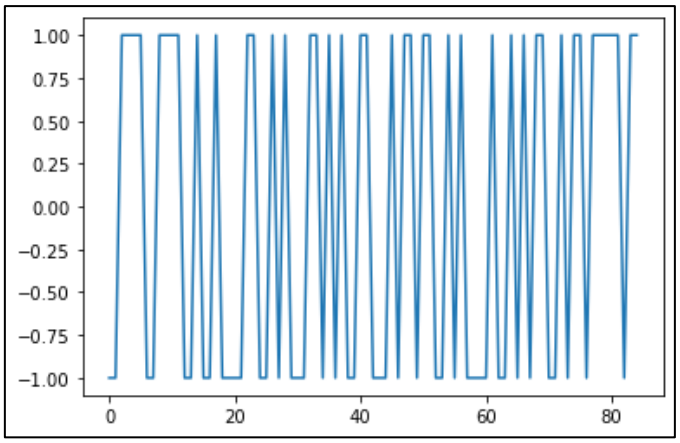


Figura 69. Símbolos a la salida del decisor

9.4 Pruebas BPSK con la ADALM-PLUTO

Una vez implementado el modulador BPSK y realizada la simulación en Python, se procede a realizar varias pruebas utilizando la ADALM-PLUTO.

En primer lugar, probamos a transmitir la señal BPSK compuesta por 1000 símbolos, con un periodo de símbolo igual a 16 muestras por símbolo y con una frecuencia de portadora igual a 0,2.

Además, para realizar una correcta transmisión y recepción de la información, se han configurado las ganancias del transmisor y del receptor en -60 dB y 50 dB respectivamente, con el fin de atenuar la señal al transmitirla y aumentar su potencia al recibirla. En la siguiente figura se muestra el espectro captado por el receptor de la tarjeta. Nótese que al haber utilizado mayor periodo de símbolo, el espectro se comprime, consumiendo así menor ancho de banda.

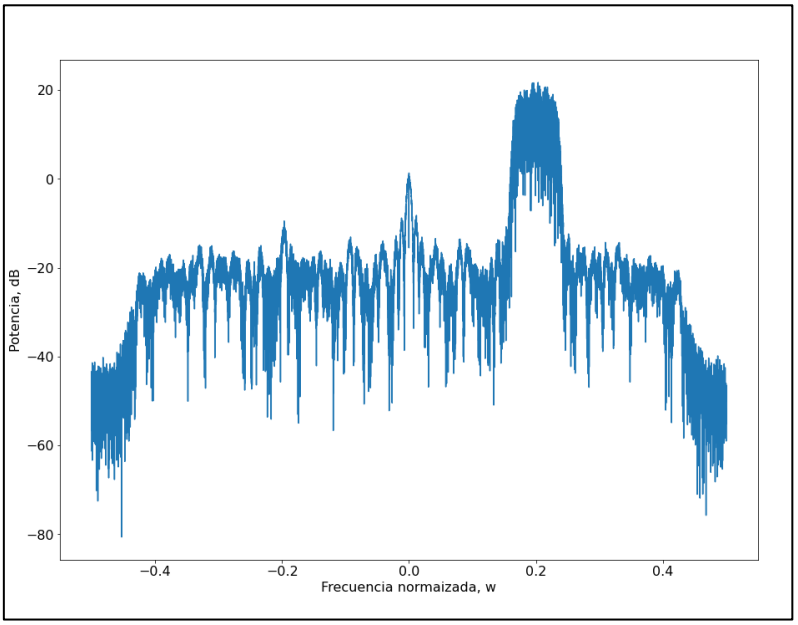


Figura 70. Espectro recibido por la ADALM-PLUTO en una modulación BPSK

En la figura 70 aparece la señal BPSK centrada en frecuencia 0.2, lo que indica que la recepción se ha realizado correctamente. Además, encontramos algo de ruido en el resto del espectro digital. Como se ha analizado en el punto de la caracterización de los filtros, si se transmite una señal cerca de la banda base, pueden aparecer repeticiones espectrales debidas a la periodicidad del espectro digital. Por ello, hemos aplicado una portadora a la señal BPSK, alejándola de frecuencia 0 y garantizando que no aparezca la banda imagen.

Al haber incorporado una portadora, habremos añadido a la señal BPSK una componente en fase que variará en función del tiempo. Este fenómeno puede apreciarse con gran detalle si utilizamos la representación de la constelación transmitida utilizando IIO Oscilloscope.

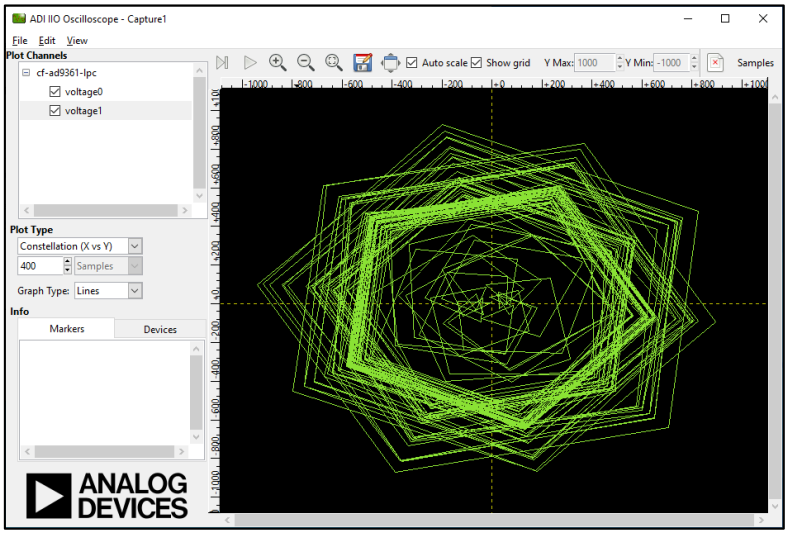


Figura 71. Constelación de los símbolos transmitidos con frecuencia portadora igual a 0.2

Para visualizar de constelación correcta los símbolos transmitidos mediante el IIO Oscilloscope, debemos transmitir la señal BPSK en frecuencia 0 para no incorporar una componente en fase. En la siguiente figura se muestra la constelación de los símbolos transmitidos sin utilizar portadora.

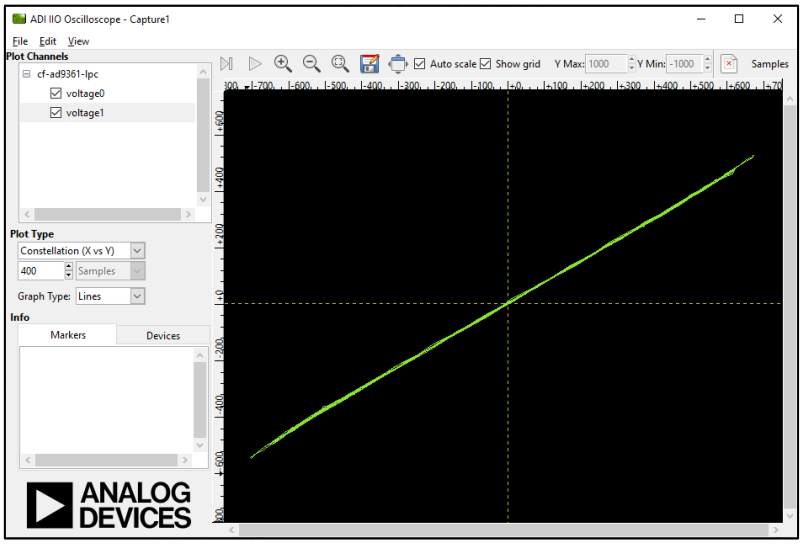


Figura 72. Constelación BPSK transmitiendo sin portadora

Se visualiza una constelación BPSK con una ligera rotación en el sentido antihorario, debido al offset frecuencial.

Debido a la transmisión cíclica, el buffer de recepción está configurado con una longitud mayor a la del buffer de transmisión. Por ello, el receptor captará ruido antes de recibir la información y parte de la señal transmitida estará repetida. Para averiguar el comienzo y el final de la trama, calculamos la función de correlación de las muestras recibidas con el preámbulo conocido que se ha incorporado en el transmisor. En la figura 73 se muestra dicha función de autocorrelación normalizada.

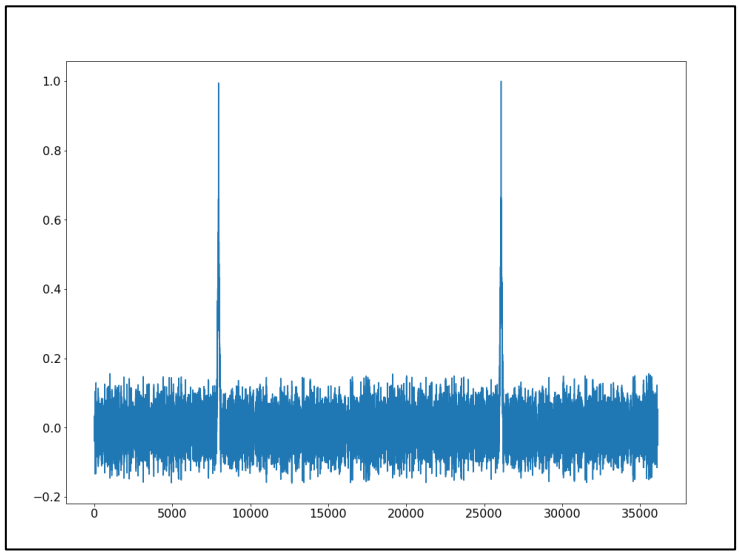


Figura 73. Función de autocorrelación de la señal recibida con el preámbulo conocido

Donde podemos observar dos máximos claros, correspondientes a los inicios de la trama enviada. Para conservar la integridad de la señal transmitida, debemos elegir el fragmento situado entre ambos máximos. En caso contrario, estaríamos obteniendo únicamente parte de la trama enviada.

10 Alineación del proyecto con los ODS

En los últimos años se ha conseguido un gran progreso en la meta relativa a la educación primaria universal debido a la lucha constante por una educación de calidad, alcanzando una tasa de matrícula del 91 % en el año 2015. No obstante, el progreso de la educación en las regiones de desarrollo ha resultado ser más complicado, debido a distintos factores como los conflictos armados o los altos niveles de pobreza.

El objetivo de conseguir una educación adecuada y de calidad para todos, se fundamenta en la convicción de que la educación es uno de los motores más importantes para garantizar el desarrollo sostenible. Mediante una educación adecuada, los alumnos serán capaces de empatizar con la naturaleza y adoptar actitudes que promuevan el desarrollo sostenible.

Por todo ello, desde el departamento de Señales y Sistemas, se ha optado por la incorporación de una nueva herramienta didáctica de radio definida por software, con el fin de mejorar la calidad del aprendizaje y acercar este tipo de iniciativas a otros sectores de la población más desfavorecidos.

11 Bibliografía

[1]. *Ilia Iliev and Ivaylo Nachev "An Automatic System for Antenna Radiation Pattern Measurement"*

[2].*"Algorithm and Software for Intelligent Analysis of the Frequency Spectrum for Cognitive Radio Systems".*

[3].*Chirstopher Gravelle, Ruolin Shou. "SDR Demonstration of Signal Classification in Real-Time Using Deep Learning". IEEE Globecom Workshops, 2019.*

[4].*Yacine Adane. "A Smart Digital Software Radio Transceiver Design Concept for UAV and Autonomous Vehicles Application". Advances in Science and Engineering Technology International Conferences, 2019.*

[5]. *Ángel E. Ruiz-García et. al. "SDR-Based Channel Emulator for vehicular Communications". IEEE Colombian Conference on Communications and Computing, 2019.*

[6].*"http://bibing.us.es/proyectos/abreproy/11375/fichero/MEMORIA%252FFundamentos+teoricos.pdf "*

[7]. *"Alan V. Oppenheim, Ronald W. Schafer. "Discrete Time Signal Processing". Prentice-Hall. Third Edition"*

[8]. *Lawrence R. Rabiner, James H. McClellan, Thomas W. Parks. "FIR Digital Filter Design Techniques Using Weighted Chebyshev Approximation". Proceedings of the IEEE, 1975.*

[9].*" www.tsc.uc3m.es/~antonio/libro_comunicaciones."*

[10]. "*Software-Defined Radio for Engineers Travis F. Collins Robin Getz Di Pu Alexander M. Wyglinsk"*

[11]. "*Software Defined Radio using MATLAB & Simulink and the RTL-SDR"*

[12]" https://pysdr.org/content/filters.html"

[13] . "*https://analogdevicesinc.github.io/libiio/master/libiio/index.html"*