Universidad de La Laguna

UNIVERSIDAD DE LA LAGUNA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO:

RADIO DEFINIDA POR SOFTWARE EN DISPOSITIVOS DE BAJO COSTE

Autor:
David Carralero Alonso

Tutor:
Dr. José Gil Marichal Hernández

D. José Gil Marichal Hernández, Doctor en Informática, Profesor Ayudante Doctor del área de Teoría de la Señal y las Comunicaciones, perteneciente al Departamento de Ingeniería Industrial de la Universidad de La Laguna, como Director,

AUTORIZA: a D. David Carralero Alonso, estudiante de Ingeniería Electrónica Industrial y Automática, a presentar el Trabajo de fin de Grado que ha realizado bajo mi dirección.

Lo que firmo, en La Laguna a 9 de Junio de 2016.

Dr. José Gil Marichal Hernández
Dpto. Ingeniería Industrial
Universidad de La Laguna.

Dedico este trabajo a mi familia, a mi padre Eduardo, a mi Madre María Jesús, y a mi hermana Ana.

Agradecimientos

En primer lugar, a mi tutor, José, por todo lo que he aprendido con él, y ser un excelente tutor, estando muchas horas junto a él avanzando en el proyecto.

A todo el profesorado del Ciclo Superior de Sistemas de Regulación y Control Automáticos de Salesianos la Cuesta, en especial a mi tutor, Diego, por enseñarme muchísimo académicamente y moralmente.

A mi familia, en especial, a mis padres, Eduardo y María Jesús, por ayudarme y apoyarme siempre en mi carrera universitaria, dándome esa confianza emocional que siempre he necesitado.

A mi tío y padrino José María, que me ha ayudado todos estos años y siempre lo he tenido presente para cualquier problema.

A todos los compañeros de grado que he tenido estos tres años, siendo vitales en mi día a día como estudiante, y en el aspecto motivacional, siendo mi inspiración para afrontar los cursos académicos con ganas y esfuerzo.

Y por último, a mi equipo de natación, Bentacu Laguna, por ser mi vía de desconexión de mis problemas, y sacarme siempre una sonrisa en los entrenamientos.

David Carralero Alonso

Índice general

Índice de figuras

Resumen

En el presente proyecto, se aborda el conjunto de elementos software que intervienen en la demodulación de una radio FM comercial. A diferencia de una radio física, donde tenemos una serie de elementos hardware que realizan las diferentes funciones, en nuestro caso, todo se hará, paso a paso, por software manejado por ordenador.

Como punto de partida, se usarán dispositivos de bajo coste, que únicamente permiten la recepción de señales. Con bajo coste nos referimos a dispositivos que su precio oscila entre los 10 y 20€.

Explicaremos cómo usar el dispositivo para nuestros fines, ya que en la compra del mismo está enfocado a la recepción de señales de televisión digital, cosa en la que no estamos interesados en el proyecto. Por tanto instalamos los drivers y programas necesarios tanto en Windows como en Ubuntu para poder usar el dispositivo para nuestro propósito, la radio FM.

En este proyecto nos centramos en el aspecto didáctico de las comunicaciones, en primer lugar, explicamos analógicamente y de forma teórica la modulación y demodulación FM, y continuamos con el RDS. Se explicarán también las modulaciones y demodulaciones digitales, viendo los resultados paso a paso utilizando el programa Matlab, para explicar cómo se modula y demodula digitalmente una señal FM, así como el RDS y continuaremos con tratamiendo de la captura de la señal que hace el dispositivo.

Por último, veremos en tiempo real, con una herramienta de la comunidad GNURadio, cómo se realiza paso a paso la demodulación FM estéreo comercial, mediante diagramas de bloque. Asociándolos a lo aprendido anteriormente con el Matlab.

Abstract

In this project, we study the digital FM demodulations, focusing on frequency modulation FM, using a low cost device called RTL2832U.

This project is part of the GNU Radio development, where people share their own projects to the community.

We start in basic analog FM modulation and demodulation, for didactical purposes only. We continue explaining digital FM, modulation and RDS modulation and demodulation, based on BPSK. Then we use our device to capture I and Q samples and demodulate step by step on matlab. We study some filters, but for demodulating we explain IIR, FIR, Parks-McClellan, Chebyshev, and Low-Pass and Band-pass filters. We explain the magnitude and phase response and also the group delay.

The output at first is mono audio, then we demodulate separately both audios to obtain stereo audio, and finally we obtain RDS data.

Additionally, we used a real time program created by the GNU Radio development team, called GNU Radio Companion, and we demodulate commercial FM stations, block by block, and get the audio mono and stereo output, we compare the steps that we use in GRC to the matlab functions that we used.

Capítulo 1

Introducción

Este capítulo tiene como objetivo exponer, de forma general, el objetivo central de este proyecto, realizar un pequeño estudio de cuáles son las tecnologías actuales relacionadas con el mismo y describir los motivos y objetivos del proyecto.

1.1. Objetivos y Motivaciones del Proyecto

La motivación principal del proyecto vino con la asignatura "Sistemas de Comunicación"de Tercero del Grado de Ing. Electrónica Industrial y Automática. Especialmente realizando las prácticas de la asignatura con el entrenador de comunicaciones.

En primer lugar, el objetivo es explicar los procesos de modulación y demodulación FM de una forma teórica, para irnos familiarizando con las técnicas analógicas convencionales, para pasar a explicar cómo obtenemos el RDS, tanto modulación como codificación y decodificación. Se explica en el capítulo 2.

El objetivo de este proyecto será en primer lugar, conseguir el correcto funcionamiento del dispositivo de bajo coste comprado en los sistemas operativos Windows y Ubuntu, como requisito necesario para poder empezar con el proyecto en sí. Podemos ver detalladamente los pasos de la instalación en ambos sistemas operativos en el capítulo 3.

Como continuación, explicaremos con fines didácticos la modulacion y demodulación FM digital. Luego aplicaremos estos conceptos para demodular una captura de señal preprocesada, para pasar en primer lugar a audio mono, y en segundo lugar audio estéreo. Como último paso en la demodulación, se aplicará en una señal real capturada. El espectro de frecuencia de la FM comercial abarca desde los 88MHz hasta los 108MHz. Se usará MatLab/Octave como lenguaje de programación para probar varios esquemas de demodulación sobre señales capturadas con el dispositivo. Además, se obtendrá la señal RDS. Todo esto se explica en el capítulo 4 y 5, en el capítulo 4 se hace con una señal preprocesada, y en el capítulo 5 se captura una señal real con nuestro dispositivo.

Como objetivo final muestreamos en tiempo real la señal de FM con el programa GNU-Radio companion, en el que comenzamos haciendo una introducción a su funcionamiento, y posteriormente hacemos una aplicación para explicar cómo demodular FM con técnicas similares a las usadas en Matlab. Se obtendrán como salidas, dibujos de espectro, señales, audio monofónico y estéreo, y señal RDS. El uso de GRC para la demodulación FM está tratado en el capítulo 6.

1.2. El dispositivo

El dispositivo que se va a utilizar para la realización del Trabajo de Fin de Grado es el RTL2832U + R820T. Se puede adquirir por tan solo 12€, y venía destinado para aplicaciones de recepción de televisión digital, aspecto que no tratamos en el proyecto. En la figura 1.1 podemos ver su aspecto en su embalaje original.

Figura 1.1: Pack de componentes

1.2.1. RTL2832U

El RTL2832U es un demodulador de alto rendimiento con interfaz USB 2.0. Posee un ancho de banda de 8MHz.

Es compatible con los sintonizadores de Frecuencia Media(36.125MHz), baja frecuencia (4.57MHz), o salida de cero si se utiliza un cristal 28.8MHz. Cuenta con alta estabilidad en la recepción portátil. En la figura 1.2 podemos ver externamente el USB donde se encuentra el demodulador.

Figura 1.2: Stick USB RTL2832U + R820T

1.2.2. R820T

El R820T* es un sintonizador de alto rendimiento y baja potencia, de silicio, podemos ver si diagrama de bloques en la figura 1.3. Se basa en un amplificador de bajo ruido, un mezclador, un PLL y un regulador de voltaje y un filtro de seguimiento.

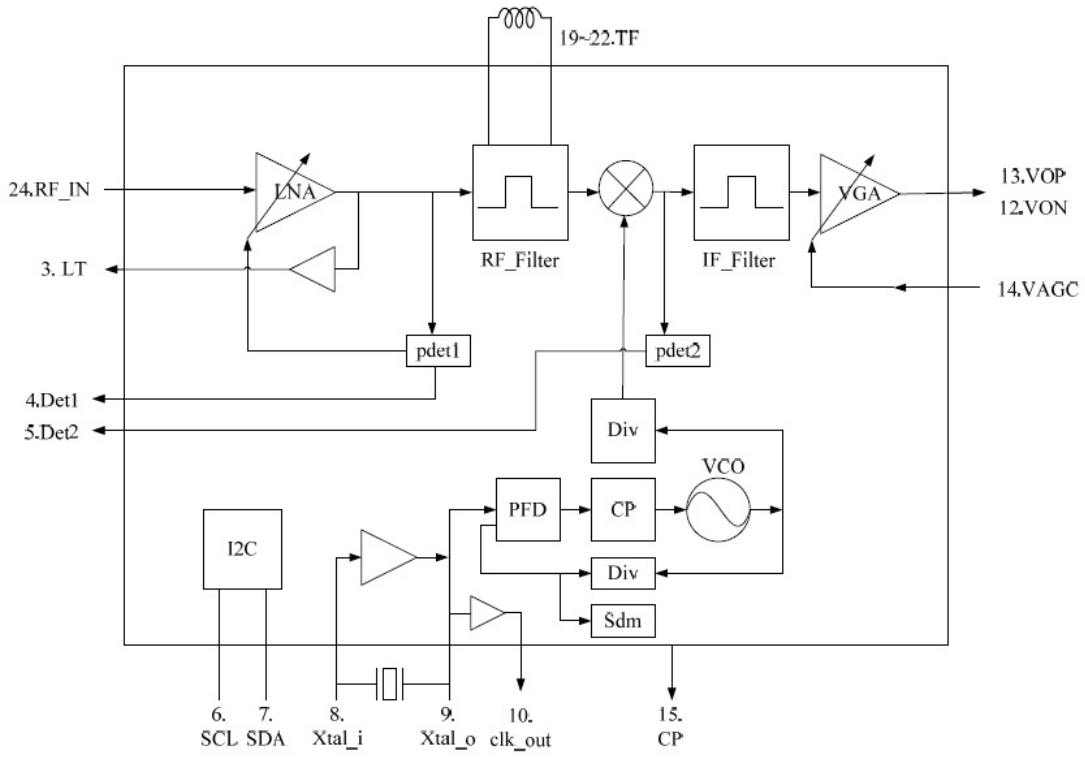

Figura 1.3: Diagrama de bloques del R820T

*Podemos ver el "Data Sheet" al completo de este componente (R820T) en la referencia [1] de la bibliografía.

1.2.3. Características relevantes para nuestro proyecto

- Rango de frecuencias: 42 a 1002 MHz

- Hasta 3.2 MHz de frecuencia de muestreo (sample rate)

- Ruido: 3.5 dB

- Ruido de fase: -98 dBc/Hz a 10kHz

- Consumo de corriente: <178mA, a 3.3V

- Máxima potencia de entrada: +10 dBm

- Muy bajo consumo de potencia, <178mA

- Fuente de alimentación única de 3.3V

1.2.4. Detalles

Los dispositivos con un sintonizador R820T[**] pueden oscilar entre 27 hasta 1760 MHz con una separación de más de 1100 a 1250 MHz en general. Existe una pérdida de rendimiento a más de 1500 MHz. Podemos ver internamente el dispositivo, donde hacemos referencia a los integrados RTL2832U y R820T en la figura 1.4

Los dispositivos R820T utilizan una frecuencia intermedia de 3.57MHz.

El error del sintonizador es de 30±20 PPM (Partes por millón). Todas las entradas de antena son de 75Ω de impedancia. El rango dinámico para la mayoría de dispositivos es de alrededor de 45 dB. La sensibilidad es de alrededor de -110 dBm normalmente. La frecuencia de muestreo máxima segura es 2,56 MS/s (Megamuestras por segundo), pero en algunas situaciones hasta 3.2 MS/s sin dejar caer las muestras (en el RTL2832U caen internamente). La mínima tasa de muestreo permitida es 0.5MS/s.

El dispositivo utilizan muestreo I/Q, debido a ello la frecuencia de muestreo es igual al ancho de banda, en lugar de sólo la mitad de ella. Se requiere por tanto, el modo de transferencia de datos USB 2.0, con versiones anteriores como por ejemplo USB 1.1 no funcionaría.

El R820T utiliza sobre los 300 mA de potencia a 5V USB.

[**]Más información el respecto, variantes en la referencia [2] de la bibliografía.

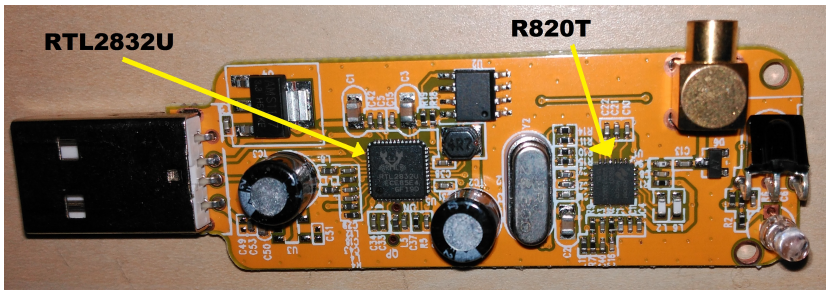

Figura 1.4: Integrados RTL2832 + R820T

Los dispositivos basados en RTLSDR, es decir, en el integrado RTL2832U, utilizan un sintetizador basado en un PLL (Lazo de enganche de fase) para producir el oscilador local requerido por el mezclador en cuadratura. El mezclador en cuadratura ("Quadrature Mixer") produce una salida de banda base compleja, donde la señal se extiende desde -BW/2, hasta BW/2, es decir, ± la mitad del ancho de banda, y este valor es el ancho de banda analógico de las etapas de salida de mezclador.

Estas muestras complejas (I y Q) son muestreadas, por el conversor analógico-digital (ADC). A partir de esto se produce un stream de 28,8 Msps en 8 bits. Eso se puede volver a muestrear dentro del RTL2832U para obtener cualquier tasa de muestreo (sample rate) que deseemos en nuestro PC.

Los dispositivos con un sintonizador R820T, suelen venir con un conector coaxial MCX o micro coaxial, que son conectores coaxiales desarrollados en los años 80 un 30 % más pequeños que los conectores coaxiales SMB (*SubMiniature coaxial conector*). Usan una interfaz de encaje y suelen tener una impedancia de 50 Ω (algunos 75 Ω). Ofrecen capacidad de banda ancha de DC a 6 GHz. En la figura 1.5 podemos ver el patillaje del integrado R820T.

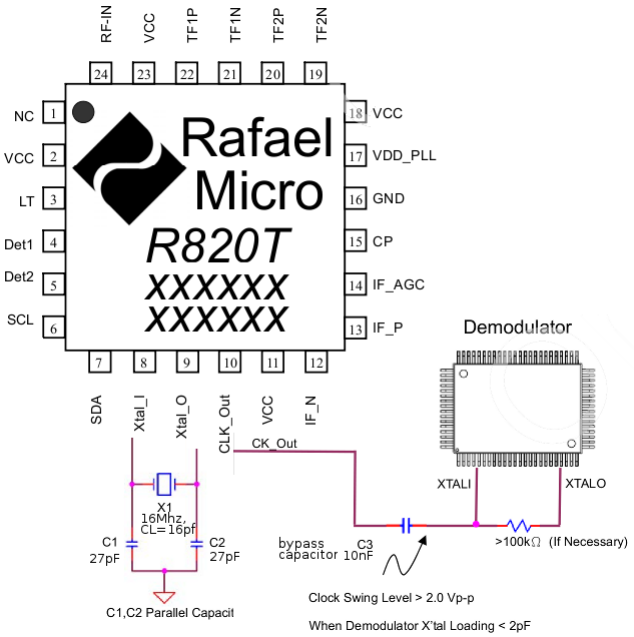

Figura 1.5: Patillaje R820T

Capítulo 2

Marco Teórico

2.1. Software Defined Radio (SDR)

Radio definida por software o SDR consiste en un sistema de comunicaciones por radio donde varios de los componentes típicamente implementados en hardware (mezcladores, filtros, moduladores/demoduladores, detectores, etc) son implementados en software, normalmente utilizando un ordenador (PC).

Un SDR básico está formado por un ordenador equipado con un adaptador de radiofrecuencia(RF), y una tarjeta de sonido u otro conversor analógico a digital (CAD). La gran parte del procesamiento de las señales se realiza en procesadores de propósito general (microprocedador del PC), en lugar de utilizar hardware de propósito específico (sistemas empotrados). Esta configuración permite cambiar los protocolos y formas de onda simplemente cambiando el software.

El descubrimiento del uso para SDR a partir de sintonizadores de DVB-T o vídeo digital, surgió en marzo de 2010 cuando Eric Fly estuvo capturando los paquetes USB del software de Windows en modo FM con el objetivo de sacar un equivalente para Linux. Después, surgió la posible potencialidad SDR del dispositivo, y, a partir de ahí, fue desarrollándose software, o adaptándose el existente, para aprovechar la posibilidad de sintonizar entre 50 Mhz y 2200 MHz, logro fruto de Realtek y Osmocom que trabajaron independientemente, pero en 2012 se unieron para crear RTLSDR.

Los SDR son de gran utilidad tanto en los servicios de telefonía celular como en el ámbito militar, pues en ambos se manejan varios protocolos en tiempo real, que cambian casi constantemente.

A largo plazo, se prevé que los radios definidos por software se conviertan en la tecnología dominante en radiocomunicaciones.

En nuestro caso, la radio definida por software es usada con fines didácticos, dado que éste es un sistema de bajo coste, que puede ayudar a entender el funcionamiento interno de las comunicaciones reales en un laboratorio de comunicaciones.

2.2. Breve historia de la radio definida por software

Hace más de 30 años desde que la radio definida por software fuera introducida, detallaremos los sucesos a lo largo de estas tres décadas de evolución:

1984: E-Systems inventó el término "Software Radio"

Con este término hacía referencia a prototipo digital receptor de banda base equipado con una serie de procesadores que realizan filtrado adaptativo para el rechazo de interferencias y demodulación de señales de banda ancha.

1991: SPEAKeasy

El primer programa militar que usó específicamente una radio para tener sus componentes físicos implementados en software fue SPEAKeasy de DARPA. Su objetivo principal era tener una sola radio que pudiera detectar diez protocolos de radio militares diferentes y operar en cualquier lugar entre 2MHz y 2GHz.

1992: Publicación de Joseph Mitola sobre radio software

Fue el primero en publicar sobre el tema de software, en la conferencia nacional de telesistemas IEEE, con su ponencia *"Software Radio: Survey, Critical Analysis and Future Directions"*. Es mencionado por muchos como el padrino de la radio por software.

1996: Creación del foro SDR

El foro reunía a personas y organizaciones de la industria y el mundo académico,con el objetivo de avanzar en las tecnologías relacionadas con la radio definida por software. Se formaron varios grupos de trabajo y comités para estimular y dirigir la innovación y estándares.

1997: Creación del foro JTRS

JTSR o "Joint Tactical Radio System" fue creado por el Departamento de Defensa de los Estados Unidos para aumentar la portabilidad de forma de onda a través de la definición y estandarización de las capas de abstracción de las interfaces, conocida como la arquitectura de comunicación Software (SCA).

1998: Generación automática de código para SDR

Nutaq se asoció con MathWorks para crear el primer entorno de desarrollo que podía generar ejecutables directamente de un modelo de Simulink para DSP o procesador digital de señales de Texas Instruments y además FPGA de Xilinx.

2001: GNU Radio

Fue fundado por Eric Blossom y financiado por John Gilmore. GNU Radio es un entorno de trabajo de código abierto para el desarrollo de aplicaciones de SDR dentro de un ordenador (PC). Con más de 5000 usuarios anuales a partir de 2012, es la herramienta de desarrollo más popular.

2004: Procesadores PHY o de capa física

Picochip introdujo su PC102, un procesador diseñado específicamente para el procesamiento PHY, que conforman la circuitería necesaria para implementar las funciones de la capa física en el modelo OSI. El PC 102 (y sus versiones posteriores) reducen drásticamente el tamaño, coste, y consumo de energía de los equipos inalámbricos.

2006: TI y Xilinx se unen para facilitar desarrollo de SDR

Texas Instruments y Xilinx, antaño competidores, se unieron junto con Nutaq para crear la primera plataforma de desarrollo independiente de SDR. Estaba equipado con un DSP, una FPGA y un sintonizador desde 200MHz a 1GHz. No era muy grande y podía ser alimentado por batería, lo que abre nuevas posibilidades para aplicaciones y experimentos.

2009: Primer chip comercial RF de radiofrecuencia

Lime Mycrosystems creó el LMS6002, un circuito integrado de radiofrecuencia (RFIC). Los circuitos integrados de radiofrecuencia podían sintonizar en cualquier lugar entre 400MHz y 4GHz, además tenían 28MHz de ancho de banda. Motorola también desarrolló su propio RFIC. Desde entonces, otras compañias han empezado a ofrecer soluciones de radiofrecuencia.

2.3. GNU Radio

GNU Radio* es una herramienta de desarrollo libre y abierta que provee bloques de procesamiento de señal para implementar sistemas de radio definida por software. Puede utilizarse con hardware de radiofrecuencia de bajo costo para crear radios definidas por software, o sin hardware en un ambiente de simulación.

Una radio definida por software es un sistema de radio que lleva a cabo el procesamiento de la señal requerida en software en lugar de utilizar circuitos integrados dedicados (hardware). La ventaja es que el software puede ser fácilmente reemplazado en el sistema de radio. El hardware se puede utilizar para crear muchos tipos de radios para muchos estándares de transmisión diferentes. Por tanto, una radio software se puede utilizar para una variedad de aplicaciones.

Como en nuestro proyecto, es utilizada extensivamente en ambientes académicos, y por aficionados y comerciales para dar soporte a la investigación en comunicaciones inalámbricas y en sistemas de radio en el mundo real.

Las aplicaciones de GNU Radio se construyen mediante un entorno grafico GNU Radio Companion o utilizando el lenguaje de programación Python, mientras que la parte que requiere cierto rendimiento para el procesamiento de señal se implementa en C++. Así, es posible implementar en tiempo real, sistemas de radio de alto rendimiento mediante el uso simple y rápido de su entorno de desarrollo de aplicaciones.

GNU Radio soporta el desarrollo de algoritmos de procesamiento de señal usando datos generados o grabados previamente, evitando la necesidad de utilizar hardware real.

*Podemos ver la página web de la comunidad GNURadio en la referencia [3] de la bibliografía.

2.4. Modulación de Frecuencia (FM)

La modulación de frecuencia o frecuencia modulada nos permite transmitir información a través de una onda portadora variando su frecuencia. La amplitud de la onda modulada es constante e igual que la de la onda portadora.

La frecuencia oscilará en mayor o menor medida, en torno a la de la portadora y según marca la onda moduladora. El grado de esta variación dependerá del volumen con que modulemos la portadora, a lo que denominamos "índice de modulación". En la figura 2.1 podemos ver las dos señales que intervienen en la modulación, y la resultante.

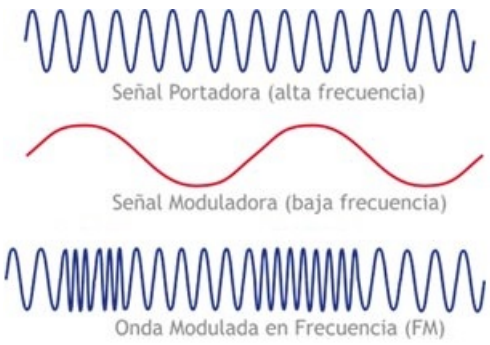

Figura 2.1: Señales onda modulada en FM

La expresión matemática de la señal portadora viene dada por:

$$v_p(t) = V_p \cdot sen(2\pi f_p t)$$

Donde V_p es el valor pico de la señal portadora y f_p es la frecuencia de la señal portadora.

Mientras que la expresión de la señal moduladora, ejemplificada aquí con un tono puro, está dada por:

$$v_m(t) = V_m \cdot sen(2\pi f_m t)$$

Siendo V_m es el valor pico de la señal moduladora y f_m su frecuencia.

La frecuencia f de la señal modulada variará alrededor de la frecuencia de la señal portadora de acuerdo a la siguiente expresión:

$$f = f_p + \Delta f sen(2\pi f_m t)$$

Por último, la expresión matemática de la señal modulada resulta:

$$v_p(t) = V_p sen[2\pi(f_p + \Delta f sen(2\pi f_m t))t]$$

Δf se denomina desviación de frecuencia y es el máximo cambio de frecuencia que puede experimentar la frecuencia de la señal portadora. A la variación total de frecuencia desde la más baja hasta la más alta, se la conoce como oscilación de portadora.

De esta forma, una señal moduladora que tiene picos positivos y negativos, tal como una señal senoidal pura, provocara una oscilación de portadora igual a 2 veces la desviación de frecuencia.

Una señal modulada en frecuencia puede expresarse mediante la siguiente expresión:

$$v(t) = V_p sen(2\pi f_p t + \frac{\Delta f}{f_m} sen(2\pi f_m t))$$

Se denomina índice de modulación a:

$$m_f = \frac{\Delta f}{f_m}$$

Se denomina porcentaje de modulación a la razón entre la desviación de frecuencia efectiva respecto de la desviación de frecuencia máxima permisible.

$$\%modulacion = \frac{\Delta f_{efectiva}}{\Delta f_{maxima}}$$

2.4.1. FM en la Radio

Los receptores emplean un detector de FM y el sintonizador es capaz de recibir la señal más fuerte de las que transmiten en la misma frecuencia.

Las emisoras de FM pueden trabajar en bandas de frecuencias muy altas, en las que las interferencias en AM son importantes. Las estaciones o emisoras comerciales de radio FM tienen frecuencias entre 88 y 108 MHz. El alcance en estas bandas está limitado para que pueda haber emisoras de la misma frecuencia situadas a unos cientos de kilómetros sin que se interfieran entre ellas.

2.4.2. Demodulación FM estéreo

En un sistema de sonido monofónico sólo hay un altavoz, o hay varios altavoces pero todos reproducen la misma señal sonora. En un sistema de sonido estereofónico[**], en cambio, hay dos juegos de altavoces; uno que reproduce sonidos destinados al oído derecho y otros que reproducen sonidos destinados al oído izquierdo. Esto permite realizar efectos como hacer aparentar que un sonido procede de una determinada dirección.

[**] FM estéreo con más detalle en la referencia [4] de la bibliografía.

Cuando quisieron desarrollar un sistema para transmitir sonido estéreo por la radio, decidieron añadir esta capacidad a la radio FM mono ya existente. El objetivo era que una emisora FM pudiese transmitir sonido estéreo por el mismo canal que venía usando para el sonido mono de manera que las radios monofónicas que ya estaban en el mercado pudiesen recibir correctamente esas transmisiones estéreo, aunque (por supuesto) se escuchasen en mono. Para ello, las frecuencias audibles del programa demodulado deben contener una señal monofónica de manera que una radio FM mono pueda tratar el programa estéreo como si fuera un programa mono. En la figura 2.2 podemos ver las diferentes señales que contiene en espectro de FM estéreo.

Sin embargo, no había nada que impide añadir más información en frecuencias superiores a estas frecuencias audibles. Por tanto, se genera una onda con toda la información necesaria para reconstruir la señal estereofónica y desplazarla en frecuencia hasta una frecuencia inaudible, y luego hacer que el receptor la vuelva a trasladar hasta las frecuencias audibles.

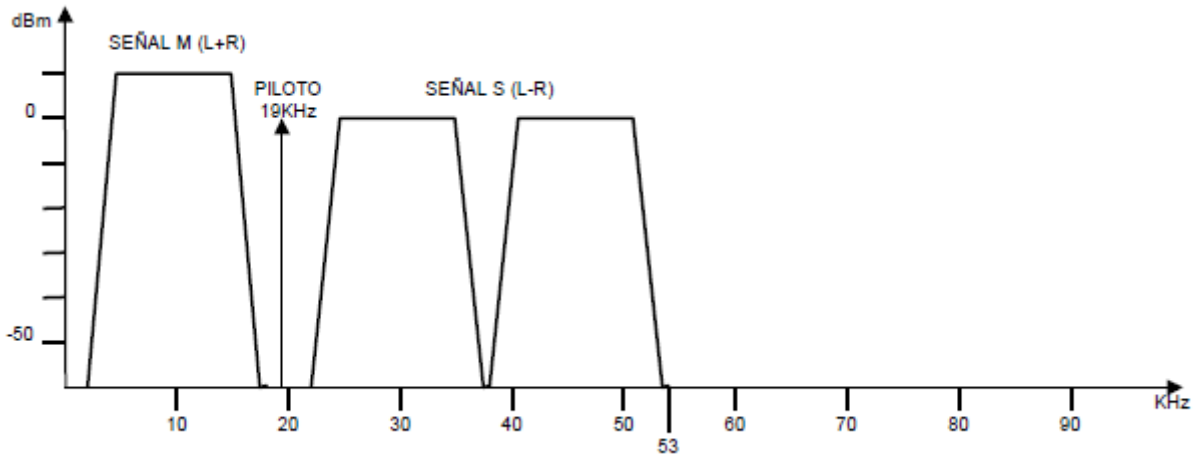

Figura 2.2: Espectro de la señal compuesta en la radio FM comercial

Dentro del programa de una radio FM moderna. Entre 0 y 15 kHz está la señal monofónica, que no es más que la suma de las señales sonoras correspondientes a los oídos derecho e izquierdo. Entre 23 y 53 kHz (centrada en 38 kHz) está la señal "diferencia", que es la diferencia entre la señal del oído izquierdo y la señal del oído derecho.

Veamos cómo se generan estas señales estereofónicas. Si L y R son las ondas correspondientes a los oídos izquierdo y derecho, respectivamente, resulta que la señal mono es M=L+R y la señal diferencia es D=L-R.

Para reconstruir la señal del oído izquierdo, la radio no tiene más que sumar las dos señales, M+D=L+R+L-R=2L y para reconstruir la del oído derecho sólo tiene que restarlas, M-D=L+R-(L-R)=2R.

Para poder sumar y restar las señales mono y diferencia hay que desplazar primero la señal diferencia para que esté centrada en 0 Hz en lugar de 38 kHz. En teoría podríamos hacerlo multiplicándola por la senoidal $cos(2\pi 38000t)$ y aplicándole un filtro paso bajo que sólo deje pasar las frecuencias inferiores a 15 kHz. Este plan tiene dos problemas. El primero es que sería difícil generar una senoidal que tuviese exactamente la misma frecuencia que el oscilador que desplazó la señal diferencia en primer lugar. El segundo problema es que el oscilador del receptor de radio y el de la emisora estarían fuera de fase, con lo que la señal diferencia desplazada podría tener la fase cambiada y el sistema para generar las señales estereofónicas no funcionaría bien.

Para solucionar este problema, el programa incluye un "tono piloto", que es una senoidal de frecuencia 19 kHz. Esta senoidal proporciona una referencia de frecuencia y fase que sirve para generar la senoidal de 38 kHz necesaria para desplazar la señal diferencia. Esto se hace utilizando un componente de las radios FM llamado "bucle enganchado en fase"(PLL). Este componente es capaz de detectar una senoidal de una frecuencia determinada y producir otra senoidal con la misma fase y cuya frecuencia es un múltiplo de la frecuencia original. En las radios FM, este PLL está preparado para detectar y engancharse al tono de 19 kHz y generar una oscilación en fase de 38 kHz, y esta oscilación es la que se usa para desplazar la señal de diferencia, también será de utilidad, como veremos más adelante, el tercer múltiplo de la piloto de 19kHz, pues con ella se generará la subportadora de 57kHz que emplea la señal RDS.

2.5. RDS

El sistema RDS[***] (Radio Data System) o RBDS (Radio Broadcast Data System) es un sistema de transmisión de datos por emisoras de radio FM comerciales en sus canales de emisión regular, sin afectar la calidad del audio normalmente transmitido. Los datos transmitidos proveen de una serie de servicios al público con receptores de radio RDS.

La idea del sistema RDS es enviar datos en forma digital junto con una señal de radio en frecuencia modulada FM. Los datos transmitidos pueden llegar a un gran número de usuarios gracias a la amplia cobertura de la red de emisoras FM y a un costo mínimo por parte de éstas. La información enviada con el sistema de RDS puede ser muy diversa:

- Identificación de la emisora
- Frecuencias alternativas de la misma emisora
- Información sobre los programas emitidos
- Radiotexto

[***]Teoría del RDS en la referencia [5] de la bibliografía.

Dependiendo del equipo receptor utilizado y de la aplicación, el sistema RDS puede prestar múltiples servicios a los usuarios, entre los cuales pueden citarse:

- Presentación del nombre de la emisora en la pantalla del radiorreceptor
- Traducción de música o comentarios en la pantalla del radiorreceptor
- Sintonización automática de emisiones alternativas en el caso de atenuación de señales
- Sincronización horaria del radiorreceptor

Ya con anterioridad se han utilizado los canales de FM comercial para transmitir información adicional. Se destacan:

- El sistema SCA (Subsidiary Communication Authorisation) de música ambiental
- El sistema ARI (Autofahrer Rundfunk Information) de información de tráfico

2.5.1. Modulación RDS

El ancho de banda disponible en una señal a la entrada del modulador de FM es de unos 90 KHz, de los cuales 53 KHz son ocupados por las señales de audio estéreo. Los restantes 33 KHz disponibles en la banda pueden ser aprovechados, siempre y cuando se cumplan ciertas condiciones que aseguren la no interferencia entre canales adyacentes.

En el sistema RDS se utiliza una modulación en subportadora de 57 KHz como se puede observar en la figura 2.3. Esta frecuencia está sincronizada en cuadratura con el tercer armónico de la frecuencia piloto para transmisiones estéreo (19 KHz ± 2 Hz x 3 = 57 KHz ± 6 Hz). Durante las emisiones mono, en las cuales no existe frecuencia piloto, la subportadora de RDS, no está sincronizada pero mantiene su valor de 57 KHz ± 6 Hz.

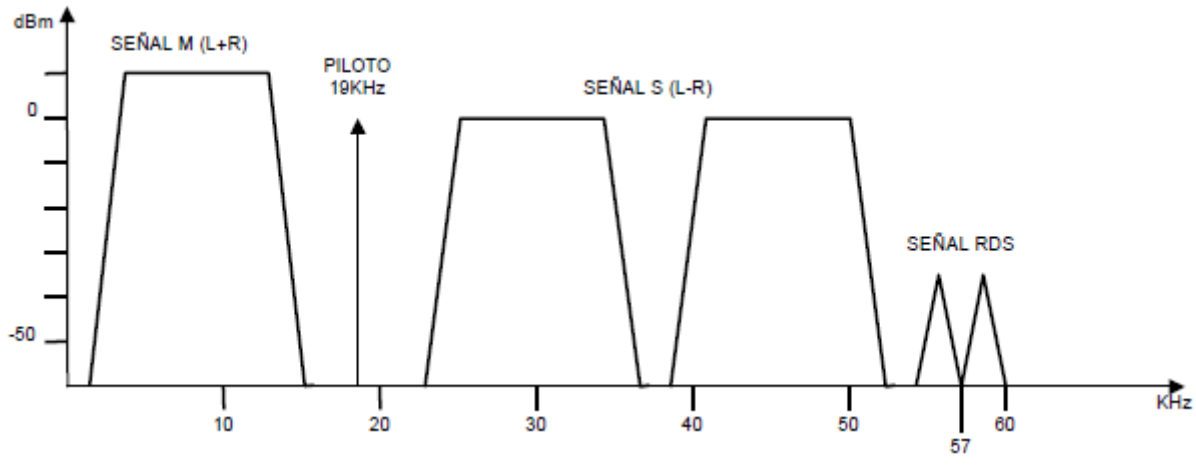

Figura 2.3: Espectro en emisiones FM con RDS

La modulación utilizada es PSK (Phase Shift Keying)o modulación por desplazamiento de fase, con desviación de fase de ±90 grados con una tolerancia de ±10 grados. Este sistema de modulación produce un nulo en la frecuencia de subportadora, con toda la energía alrededor de un par de bandas laterales separadas de la frecuencia central. De esta manera se obtiene una modulación con subportadora suprimida.

Las bandas laterales de la señal RDS tienen una amplitud pico de 3 % de la señal de audio transmitida, y debe ser estrictamente controlada para evitar interferencias que degraden la calidad de audio del programa principal.

La señal RDS es modulada por un codificador binario a un régimen de 1187.5 bps, lo cual corresponde a la subportadora de 57kHz dividida por 48 (57 KHz / 48 = 1187.5). Esta velocidad está condicionada por compatibilidad y robustez, y es suficiente para las aplicaciones actuales y futuras.

2.5.2. Codificación RDS

Los datos del sistema RDS se codifican en dos etapas, primero por un codificador diferencial y luego por un codificador bifase. Tal y como se mencionó con anterioridad, los datos se codifican a una velocidad de 1187.5 bps.

El codificador diferencial tiene como objetivo la recuperación de los ceros y unos en el decodificador aún cuando la señal llegue invertida al receptor. Los datos se codifican según la expresión:

$$Salida = EntradaActual \oplus EntradaAnterior \tag{2.1}$$

Lo podemos ver en la tabla de verdad:

E_{act}	E_{ant}	S
0	0	0
0	1	1
1	0	1
1	1	0

El codificador bifase tiene como objetivo incorporar la información del reloj de sincronismo de datos. En esta codificación se envía un positivo seguido de negativo para indicar un “uno” y un negativo seguido de un positivo para indicar un “cero”. Podemos ver las formas de señal en la figura 2.4. Para esta operación el codificador requiere un reloj de sincronismo de 2 veces el reloj de transmisión, es decir, 2375 Hz.

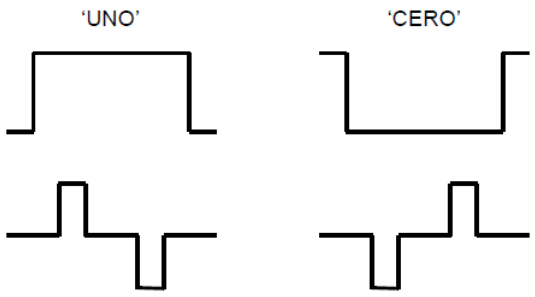

Figura 2.4: Símbolos uno y cero en la codificación bifase

Antes de alimentar el modulador FSK, la salida del codificador bifase se hace pasar por un filtro que conforma la señal. En la figura 2.5 podemos ver las transformaciones que sufre la señal.

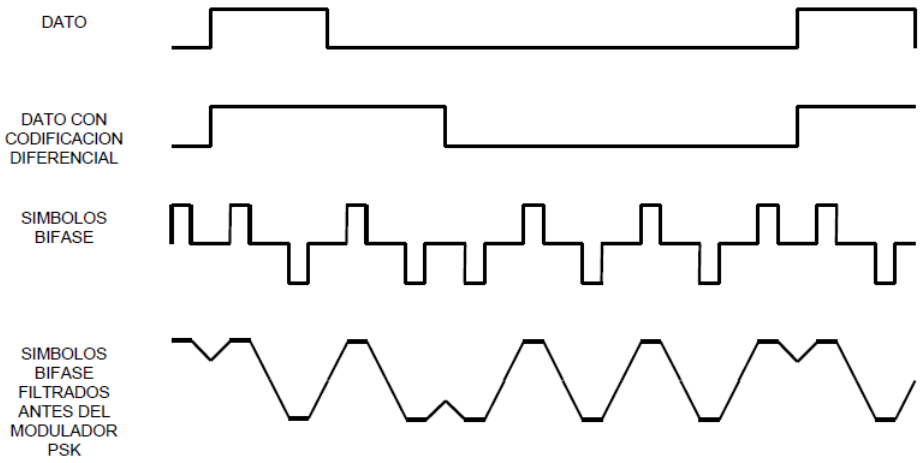

Figura 2.5: Pasos de codificación de datos para el sistema RDS

2.5.3. Decodificador RDS

Este apartado y el siguiente (Formato de datos), no lo atacaremos en el TFG, pero desde el punto de vista teórico es interesante ya que puede investigarse más adelante como una línea abierta de este proyecto.

En un posible diseño de receptor RDS analógico, la señal de entrada al decodificador se toma del demodulador de FM, antes de hacerla pasar por el filtro de de-enfasis. Esta señal es filtrada en banda para separar la señal de 57 KHz RDS y demodularla en forma síncrona. La salida de este demodulador es la señal bifase filtrada que se hace pasar por un inversor un circuito de "integra y salta", por un binarizador y por último pasa al decodificador diferencial que reconstruye los datos del sistema RDS. En la figura 2.6 podemos ver los cambios que sufre la señal.

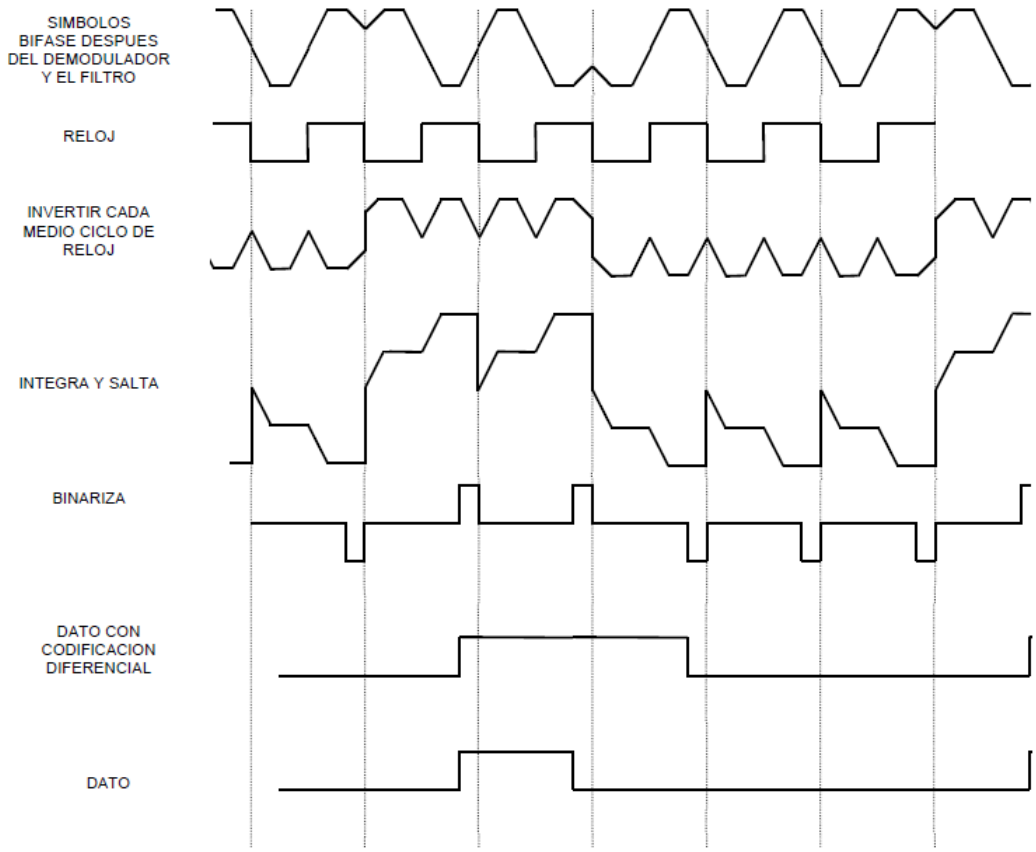

Figura 2.6: Decodificación RDS

La fase del reloj de referencia es muy importante para la decodificación, es por ello que se hace tanto hincapié en la precisión de la sincronizacion de las diferentes frecuencias en el transmisor.

2.5.4. Formato de datos

En el sistema RDS los datos son transmitidos en paquetes de 104 bits, denominados grupos, divididos en 4 bloques de 26 bits. Cada bloque contiene 16 bits de datos y 10 bits de código para corrección de errores y alineación de trama, podemos ver los bloques en la figura 2.7.

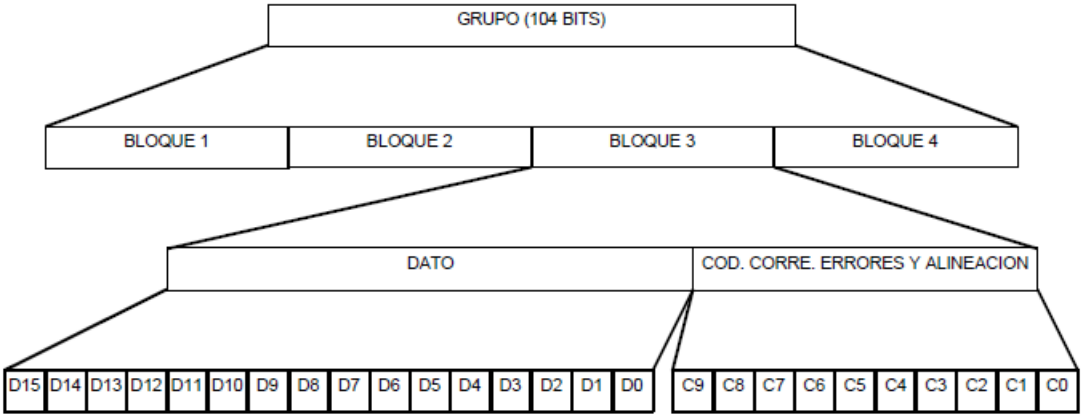

Figura 2.7: Paquete de información del sistema RDS

Para permitir la flexibilidad en la utilización del sistema, los grupos se dividen en dos versiones de 16 tipos cada uno. Cada tipo de grupo está destinado a un tipo de aplicación particular. El tipo y versión del grupo de datos siempre se hallan indicados en el segundo bloque del mismo, que se pueden ver en la figura 2.8.

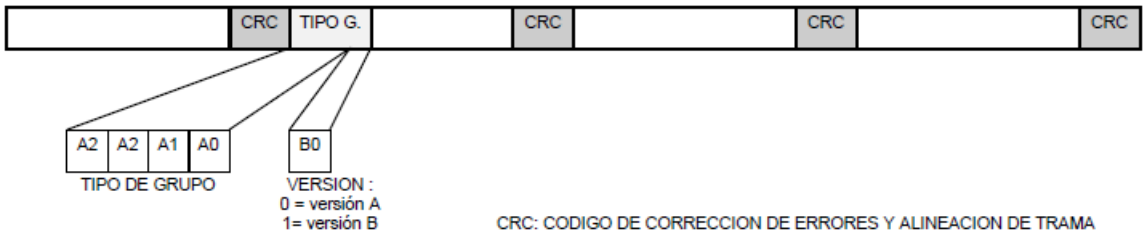

Figura 2.8: Tipo de grupo y versión

Cuatro bits identifican los 16 tipos de grupos y un quinto bit identifica si el grupo es versión A ó B. En la figura 2.9 se muestran las aplicaciones para los distintos grupos definidos en el sistema RDS.

VERSION A					VERSION B				
A3	A2	A1	A0	APLICACION	A3	A2	A1	A0	APLICACION
0	0	0	0	Información básica de sintonía	0	0	0	0	Información básica de sintonía
0	0	0	1	Información del programa	0	0	0	1	Información del programa
0	0	1	0	Radiotexto	0	0	1	0	Radiotexto
0	0	1	1	Información de otras redes	0	0	1	1	Información de otras redes
0	1	0	0	Hora y fecha	0	1	0	0	-
0	1	0	1	Canales transparentes de datos	0	1	0	1	Canales transparentes de datos
0	1	1	0	Aplicaciones de la emisora	0	1	1	0	Aplicaciones de la emisora
0	1	1	1	Buscapersonas	0	1	1	1	-
1	0	0	0	-	1	0	0	0	-
1	0	0	1	-	1	0	0	1	-
1	0	1	0	-	1	0	1	0	-
1	0	1	1	-	1	0	1	1	-
1	1	0	0	-	1	1	0	0	-
1	1	0	1	-	1	1	0	1	-
1	1	1	0	Soporte ampliado de otras redes	1	1	1	0	Soporte ampliado de otras redes
1	1	1	1	-	1	1	1	1	Información de sintonía rápida

Figura 2.9: Tipos de grupos

Además de los que definen el tipo y versión del grupo, hay otros bits fijos que deben transmitirse siempre. Entre ellos están los 16 bits que conforman un bloque con el código PI (Programme identification). Este código, ocupa el primer bloque en los grupos versión A y en los grupos versión B se repite también en el tercer bloque, como podemos ver en la figura 2.10. El código PI consta de: los cuatro primeros bits indican el país de la emisión, los siguientes cuatro la cobertura de emisión (local, internacional, nacional o regional), y los últimos 8 bits son asignados por un comité en cada país para identificar la emisora.

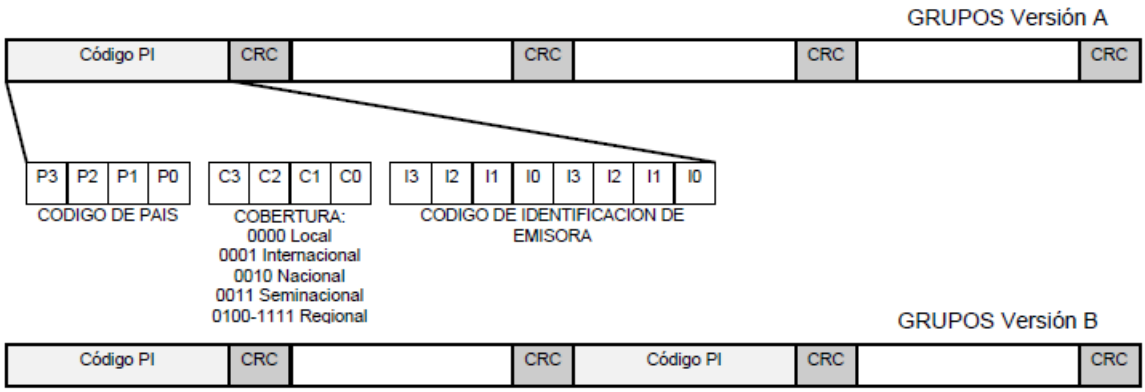

Figura 2.10: Código de identificación de emisora PI

En definitiva, en el sistema RDS cada paquete de información o grupo contiene siempre el código PI, el tipo y versión de grupo, la identificación de programa de tráfico y el tipo de programa. Ellos ocupan 27 bits en los grupos versión A y 43 bits en los grupos versión B, dejando libres 37 y 21 bits, respectivamente, para información que depende del tipo de grupo.

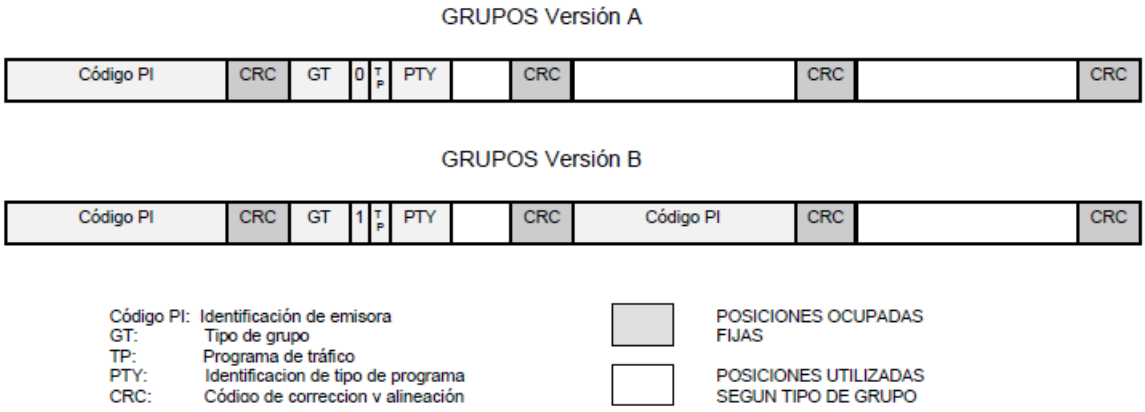

Figura 2.11: Información en los grupos del sistema RDS

Capítulo 3

Software Defined Radio

3.1. Drivers del RTL-SDR en Windows

Para poder poner en funcionamiento el dispositivo en Windows*, en primer lugar no se instalarán los drivers que vienen en la caja junto con el receptor. Se debe descargar de la siguiente página "www.rtl-sdr.com"su software denomidado SDRSharp, una vez descargado se instala en la consola de windows (Ejecutar/cmd), el siguiente archivo "install-rtlsdr.bat":

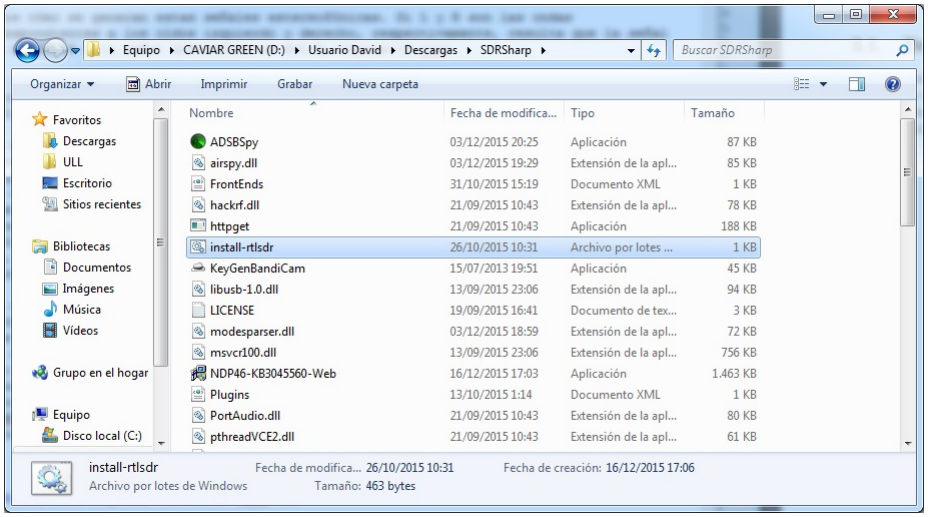

Figura 3.1: Archivos del comprimido sdrsharp.zip

Pasado un tiempo, la consola habrá instalado todo lo necesario y se cerrará automáticamente, ahora tenemos que conectar el dispositivo (RTL2832U + R820T), y ejecutar como administrador el programa "Zadig", en el que haremos lo siguiente:

*Instalación paso a paso en Windows en la cita [6] de la bibliografía, más información en la cita [10]

- Vamos a opciones y seleccionamos la opción “List All Devices”

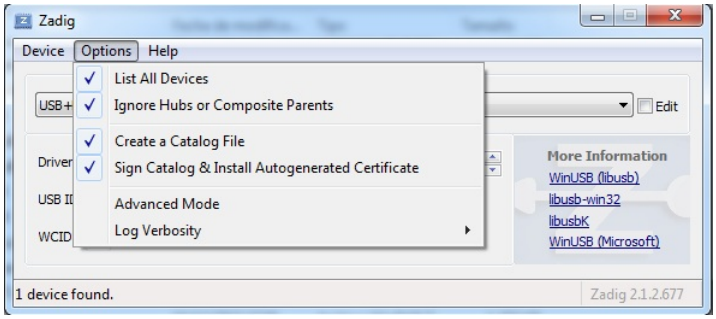

Figura 3.2: Opciones del Zadig

- Seleccionamos la opción “Bulk-In, (Interface 0)”de la lista de puertos, nos debe salir el nombre del aparato o algo parecido a “RTL2832UHIDIR o RTL2832U”

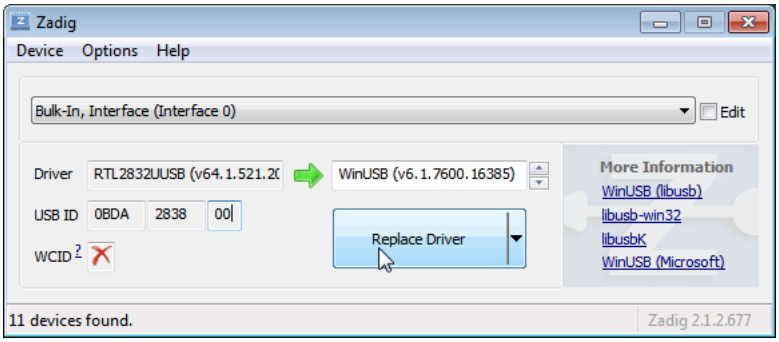

Figura 3.3: Menú del Zadig

Hacemos click en la casilla de sustituir los Drivers “Replace Drivers”, si el firewall de windows bloquea la instalación, seleccionamos la opción de “Instalar los drivers de todos modos”.

- Abrimos el SDRSharp y el selector de dispositivo le damos a “RTL-SDR (USB)”

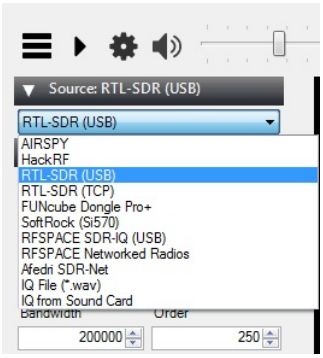

Figura 3.4: ”Sources”del SDRSharp

En los ajustes del programa debemos cambiar la ganancia, que viene por defecto en cero, y sustituirla por una mayor, en buen valor sería por ejemplo 37.2 dB.

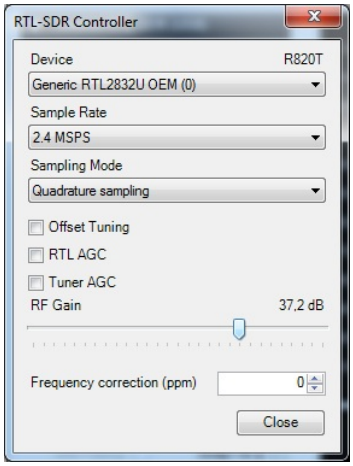

Figura 3.5: Ventana de ajustes del SDRSharp

Tras hacer todos los pasos mencionados anteriormente, deberíamos sintonizar la radio sin problema alguno, debería salir algo como esto:

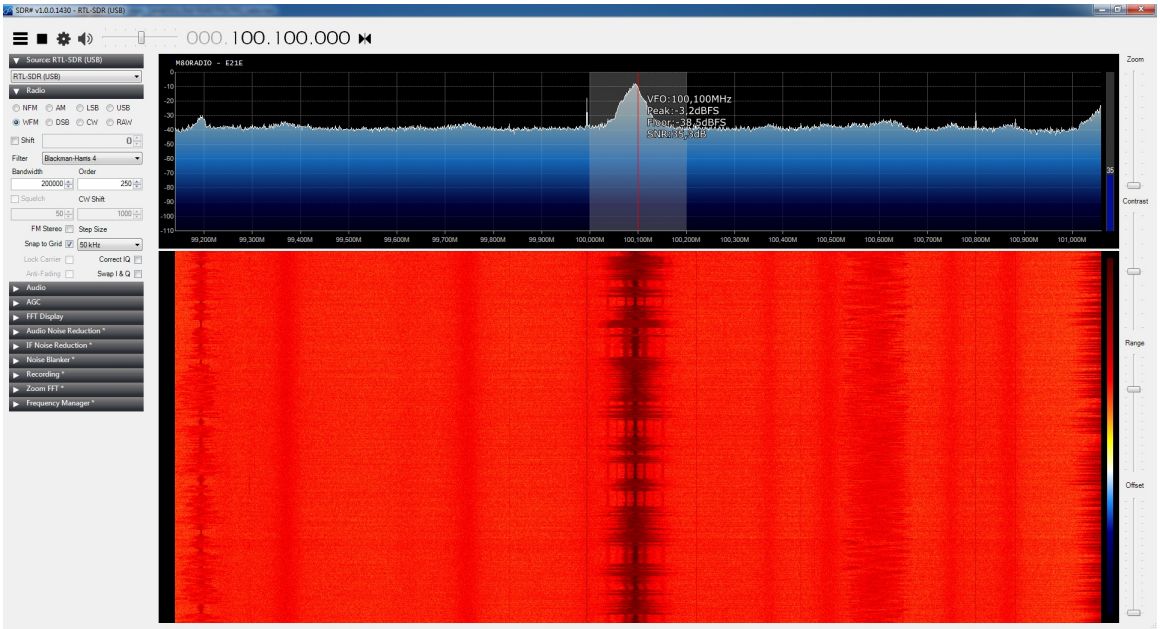

Figura 3.6: Radio sintonizada en SDRSharp

Es necesario tener conectada la antena que viene con el receptor, de lo contrario las señales de FM tendrán mucho ruido y serán prácticamente inaudibles

3.2. CygWin, Instalación de un entorno Linux en Windows

Cygwin es una serie de herramientas para proporcionar un comportamiento similar a los sistemas UNIX en Windows. Su objetivo es portar software que ejecuta en sistemas POSIX a Windows con una recompilación a través de sus fuentes.

Una vez instalado, con todos sus paquetes, al ejecutarlos nos aparece una ventana similar a la consola (konsole) que aparece en Linux.

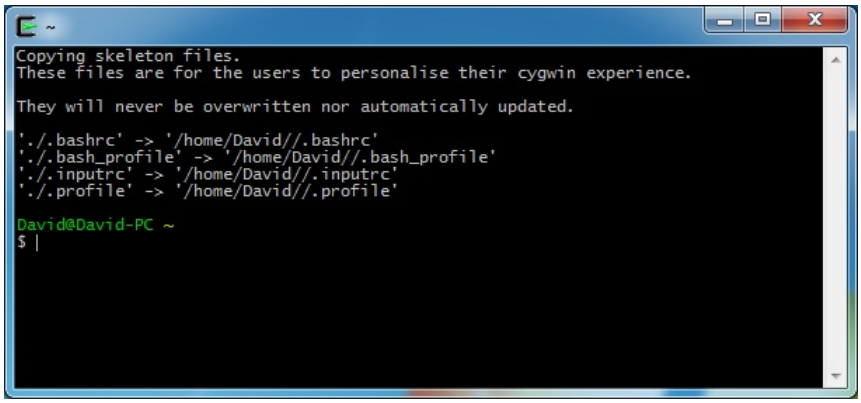

Figura 3.7: Interfaz CygWin

Lo ideal es trabajar directamente en Linux, pero como trabajo provisional empezaremos usando CygWin.

La lista de pasos que se debe seguir para instalar correctamente el dispositivo RTL2832U es la misma que en Ubuntu (Linux) por lo que se describirán todos estos pasos en la sección "Instalación en Linux"

3.3. Instalación en Linux

3.3.1. Pre-requisitos

Necesitamos una serie de paquetes para la compilación de GNU Radio en Ubuntu[**], para instalarlos hacemos uso del comando de la consola.

```
apt-get
install
```

La lista es la siguiente:

· Herramientas de desarrollo

g++
git
make
cmake
sdcc
guile
ccache

· Librerías

python-dev
SWIG
FFTW 3.X (libfftw3-dev)
cppunit (libcppunit-dev)
Boost 1.35 (o posterior, pero no las versiones 1.46, 1.47 o 1.52)
GSL GNU Scientific Library (libgsl0-dev)
libusb y libusb-dev
ALSA (alsa-base, libasound2 y libasound2-dev)

· GNU Radio Companion

python-numpy
python-cheetah
python-lxml

· WX GUI

Componentes WX GUI (python-wxgtk2.8 y python-numpy)

· QT GUI

PyQT4 (python-qt4)
PyQwt5 (python-qwt5-qt4)
QT-OpenGL (libqt4-opengl-dev)
libqwt5-qt4-dev
Fontconfig (libfontconfig1-dev)
Xrender (libxrender-dev)y Xinput (libxi-dev)

[**]Instalación paso a paso en Ubuntu en la cita [7] de la bibliografía.

· **Video-SDL**
Simple DirectMedia Layer (libsdl1.2-dev)

· **Polyphase Filter Bank**
python-scipy
python-matplotlib
python-tk

· **Otros paquetes de interés**
doxigen
octave

3.3.2. Instalación las dependencias

Con los comandos que pondremos a continuación se instalarán todas las dependencias que son necesarias, estas instrucciones pueden variar según la versión de Ubuntu utilizada, en nuestro caso usaremos la versión 12.04 LTS también conocida como "Precise Pangolin". Se recomienda utilizar una versión de ubuntu que sea LTS (Long Time Support), es decir, de "Soporte a Largo Plazo", ya que tendrá soporte y será actualizada durante más tiempo que una versión normal. Además, las versiones LTS suelen ser versiones más estables y probadas que el resto. Las dependencias se instalarán usando la siguiente serie de comandos en la Terminal o consola:

```
sudo apt-get -y install git-core autoconf automake libtool g++
python-dev swig
pkg-config libboost1.48-all-dev libfftw3-dev libcppunit-dev
libgsl0-dev
libusb-dev sdcc libsdl1.2-dev python-wxgtk2.8 python-numpy
python-cheetah python-lxml doxygen python-qt4 python-qwt5-qt4
libxi-dev
libqt4-opengl-dev libqwt5-qt4-dev libfontconfig1-dev libxrender-dev
```

3.3.3. Instalación de GNURadio

Usamos los siguientes comandos:

```
git clone --recursive git://git.gnuradio.org/gnuradio.git
```

Para configurar y hacer el "build"

```
cd gnuradio
mkdir build
cd build
cmake ../
make
```

3.3.4. Instalación de RTL-SDR

Podemos acceder al código a través del siguiente repositorio[***]:

```
git clone git://git.osmocom.org/rtl-sdr.git
```

Para construir el software volvemos a usar el comando cmake:

```
cd rtl-sdr/
mkdir build
cd build
cmake ../
make
sudo make
install
sudo ldconfig
```

3.3.5. Fuente de GNURadio

Necesitaremos para este paso tener instalada una versión reciente de GNURadio (v3.7 o v3.6), así como la última versión de las dependencias, tal y como explicamos en el paso de la instalación de GNURadio.

Podemos acceder al código a través del siguiente repositorio:

```
git clone git://git.osmocom.org/gr-osmosdr
cd gr-osmosdr/
```

En nuestro caso usaremos la versión de GNURadio 3.6, por lo que tenemos que cambiar a la rama gr3.6 con el siguiente comando:

```
git checkout
gr3.6
```

Continuamos haciendo el correspondiente make:

```
mkdir build
cd build/
cmake ../
```

Tras completarse todos estos pasos, cmake debe imprimir un resumen de los componentes habilitados/desabilitados. Para cambiar de uno a otro puedes seguir las pautas mostradas por el cmake. Benemos asegurarnos de que nuestro dispositivo esté en esa lista de componentes, de lo contrario hemos de revisar las dependencias e intentarlo de nuevo.

[***]Instalación de osmosdr en la cita [8] de la bibliografía. Aplicaciones más comunes en la cita [9]

El resumen tiene la siguiente forma:

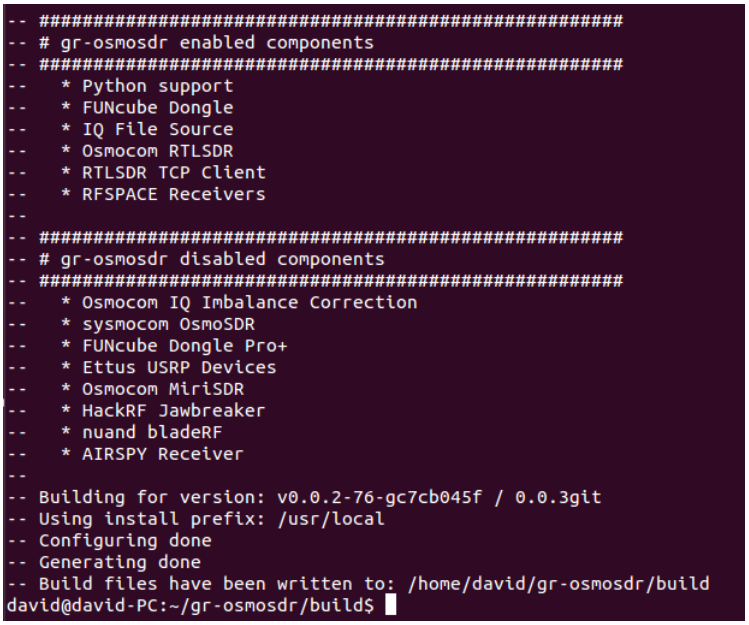

Figura 3.8: Terminal de Ubuntu al terminarse el cmake

Terminamos haciendo el make e instalando:

```
make
sudo make
install
sudo ldconfig
```

3.3.6. Sintonizando la radio comercial

Tras instalar todo el software, conectamos el aparato RTL2832U y se comprueba que el PC detecta que está conectado, tras esto ejecutamos el comando:

```
rtl_fm -f 96.3e6 -M wbfm -s 200000 -r 48000 - | aplay -r 48k
-f S16_LE
```

Si por ejemplo queremos sintonizar m80radio, cuya frecuencia es 100.1MHz, introducimos lo siguiente:

```
rtl_fm -f 100.1e6 -M wbfm -s 200000 -r 48000 - | aplay -r 48k
-f S16_LE
```

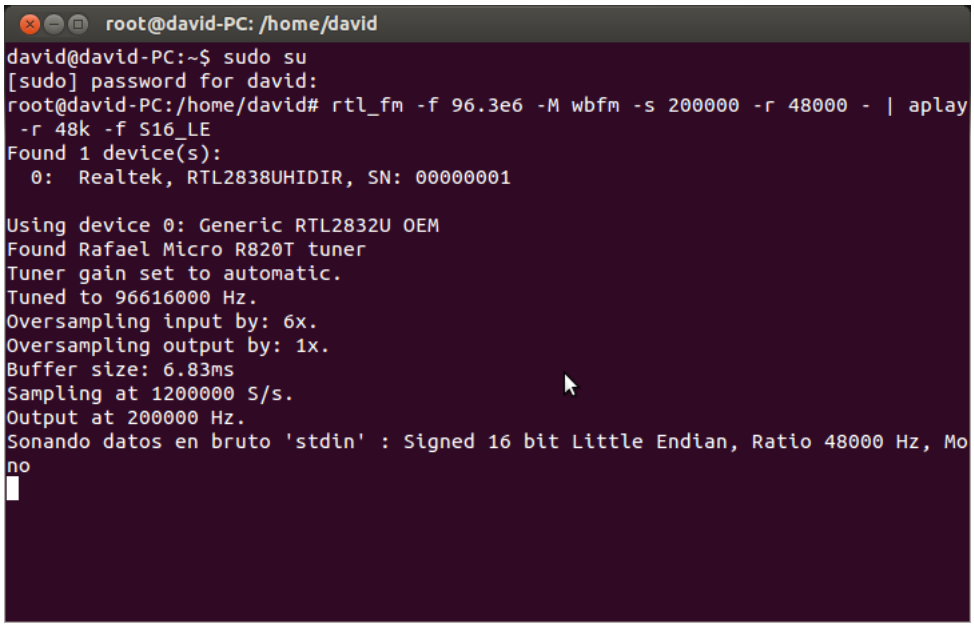

Figura 3.9: Dispositivo RTL2832U sintonizando la radio

Tras comprobar que se escucha correctamente, hemos completado la primera parte o primer bloque del proyecto, que consistía en la correcta instalación de todo el software necesario para el correcto funcionamiento del dispositivo de bajo coste RTL2832U.

A partir de ahora debemos entender cómo funciona la modulación/demodulación FM y crear un programa particular para nuestro dispositivo.

Capítulo 4

Modulación y demodulación FM Digital

4.1. Modulación de FM

El siguiente paso consiste en demodular la FM obtenida de la emisora comercial para poder tratar la señal a nuestro gusto. Comenzamos explicando teóricamente como se modula y demodula el FM digital paso a paso con el Octave/Matlab.

Partimos de un mensaje coseno:

$$m(t) = acos(w_m t)$$

La frecuencia instantánea será:

$$w_i(t) = w_c + ak_k cos(w_m t)$$

Y por lo tanto la fase:

$$\theta(t) = w_c t + \frac{\Delta w}{w_m} sen(w_m t)$$

El análisis espectral de la señal modulada en FM no es sencillo, ya que obtener la transformada de fourier de la expresión $\mathfrak{F}\{Ae^{jw_c t}e^{j\beta sen(w_m t)}\}$ no es sencillo.

No hay propiedades de la transformada de Fourier que expliquen el comportamiento de una exponencial compleja cuyo argumento a su vez es un seno. Sin embargo sí se intuye que el espectro de la señal ya se ha trasladado en frecuencia, la señal modulada tiene su espectro en lo que abarca su $w_i = w_c \pm \Delta w$.

Pasamos a analizar la forma de su espectro, si analizamos la forma de onda de la señal modulada:

$$\phi_{FM}(t) = \mathfrak{Re}\{Ae^{jw_c t}e^{j\beta sen(w_m t)}\} = Acos(w_c t + \beta sen(w_m t))$$

tras aplicar la propiedad trigonométrica "$cos(A+B) = cosA \cdot cosB - senA \cdot senB$", obtenemos lo siguiente:

$$\phi_{FM}(t) = Acos(w_c t)cos(\beta sen(w_m t)) - Asen(w_c t+)sen(\beta sen(w_m t))$$

La técnica IQ, que es la que usamos en Matlab para modular la FM, no es más que aplicar la fórmula trigonométrica de suma de argumentos de la función coseno, como ya vimos anteriormente:

$$cos(A + B) = cosA \cdot cosB - senA \cdot senB$$

```
% Asumiremos que estamos trabajando con una frecuencia de muestreo de
% 20mil hercios, trabajaremos sobre un mensaje que transcurre durante
% dos segundos y consiste simplemente en un coseno de 5Hz

fs = 20000;
t = [0:1/fs:2];
x = cos(2*pi*5*t);
```

El mensaje tiene la siguiente forma:

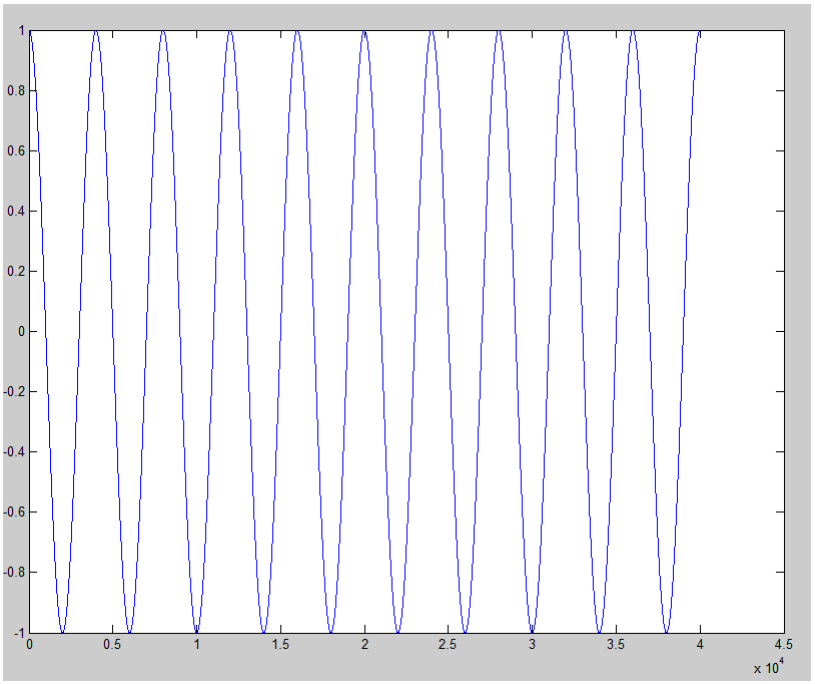

Figura 4.1: Mensaje $cos(10\pi t)$

```
% Generamos primeros las senales I y Q como las proyecciones sobre el
% eje real y el eje imaginario de una sinusoide compleja de amplitud
% constante. Pero cuya fase, phi, varia en funcion de la integral del
% mensaje

A = 1;
k = 0.05;
phi = k*cumsum(x);

I = A*cos(phi); Q = A*sin(phi);
```

Dibujamos las señales I y Q:

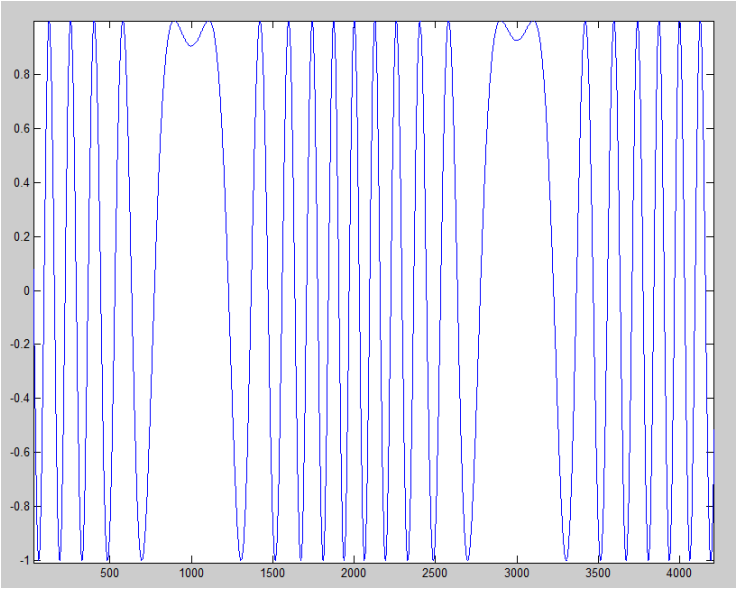

Figura 4.2: Señal I, proyección en el eje real

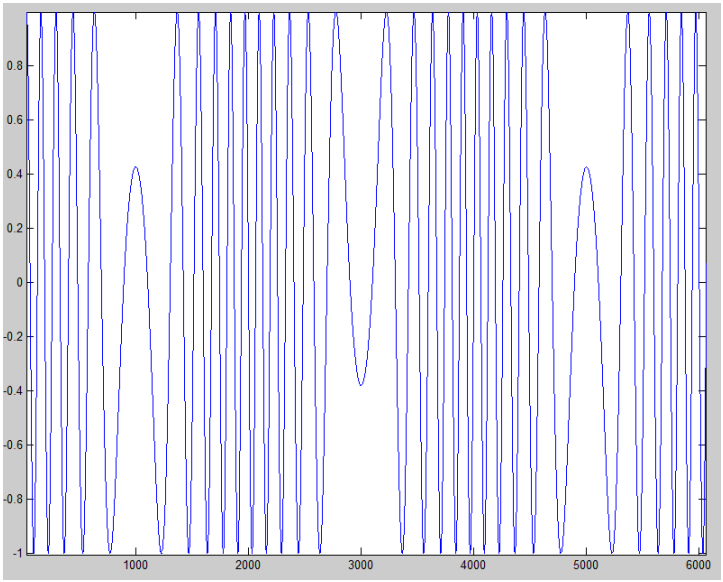

Figura 4.3: Señal I, proyección en el eje imaginario

```
% Posteriormente modulamos simultaneamente ambas senales I y Q sobre
% sendas sinusoides de igual frecuencia, pero una desfasada 90 grados
% de la otra, esto es, una senal se modula con el coseno de la frecuencia
% portadora, y otra con el seno.

wc = 2 * pi * 2000;
IQ = I.*cos(wc*t) - Q.*sin(wc*t);
```

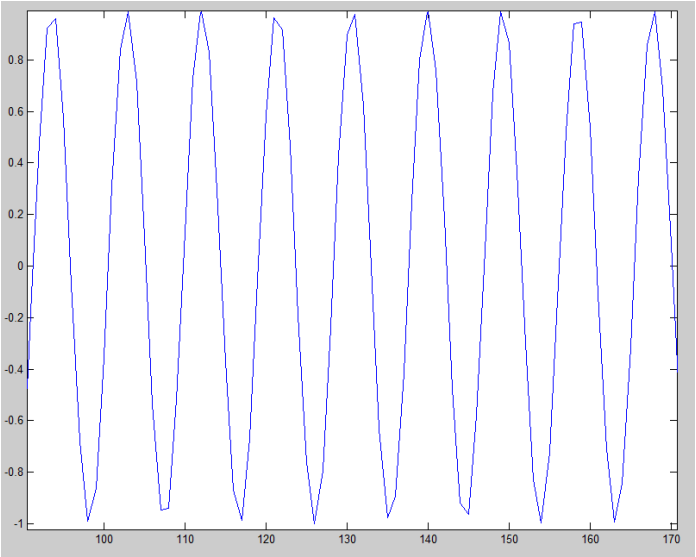

Figura 4.4: Señales I y Q moduladas simultáneamente

```
% Podemos comparar la senal IQ con una senal FM generada con  la
% la siguiente expresion:

FM = A.*cos(wc*t + phi);
```

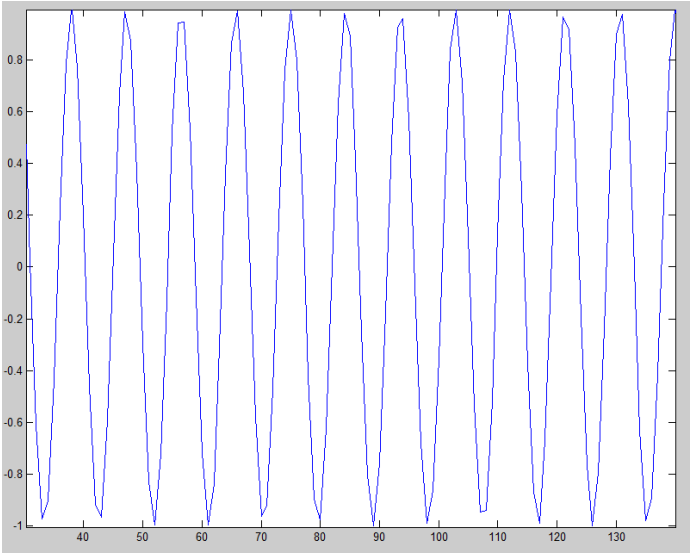

Figura 4.5: Señal FM generada, igual a la IQ

4.2. Demodulación de FM

Pasamos ahora a la parte de demodulación de la señal FM:

```
% Para ello nos proponemos recuperar por separado las senales I y Q,
% pese a que se transmitieron como una unica senal: IQ. Tendremos que
% multiplicar lo recibido porun seno en un caso y por un coseno en el
% otro. Ambos se han de filtrar pasabaja.

th = [-100:100]*1/fs;
lpf = sinc(2*500*th);
I_fm = conv(IQ.*cos(wc*t), lpf);
Q_fm = conv(-IQ.*sin(wc*t), lpf);
```

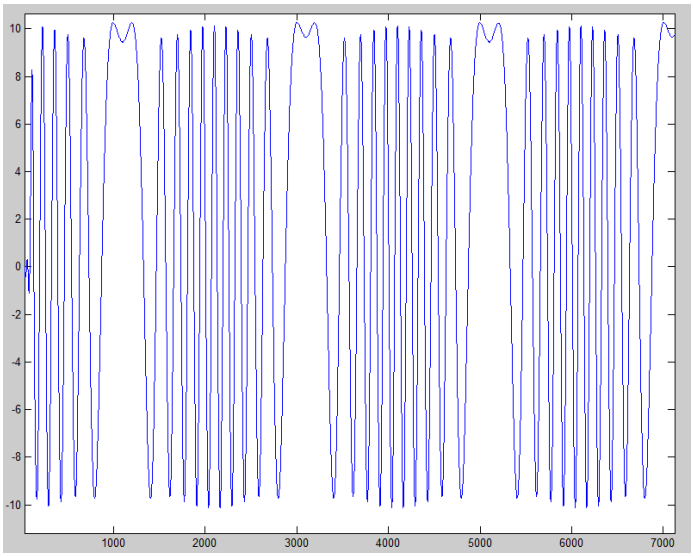

Figura 4.6: Señal I recuperada

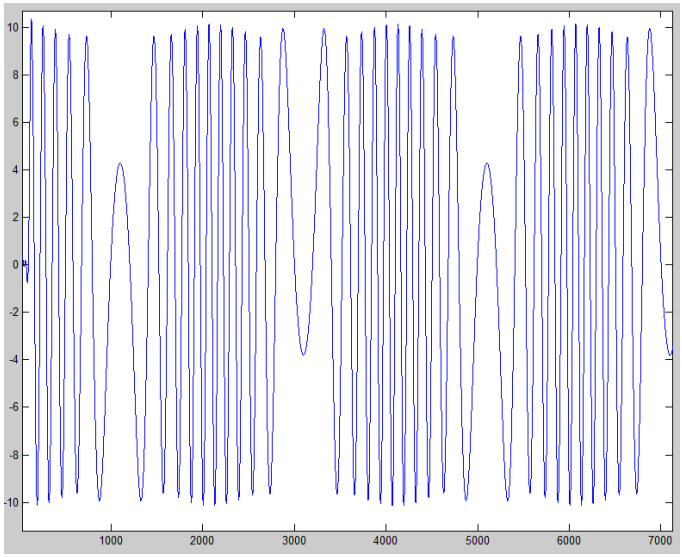

Figura 4.7: Señal Q recuperada

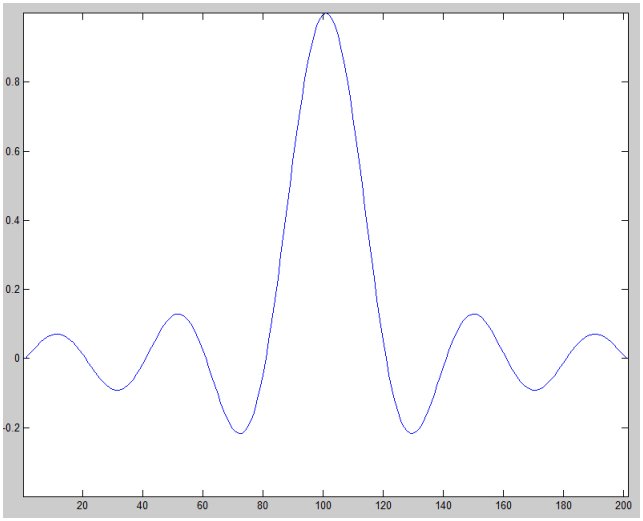

Figura 4.8: Forma del filtro pasa bajas

```
% Como las senales recuperadas se pueden interpretar como parte
% imaginaria y real de un fasor que codifica el mensaje en las
% diferencias de fase entre dos muestras se puede demodular FM
% como sigue:

R = I_fm + i*Q_fm;

dem = angle( R(2:end) .* conj(R(1:end-1)) );
```

Comprobamos visualmente que tenemos un coseno de 5Hz usando el comando plot:

```
plot(dem(500:end-500));
```

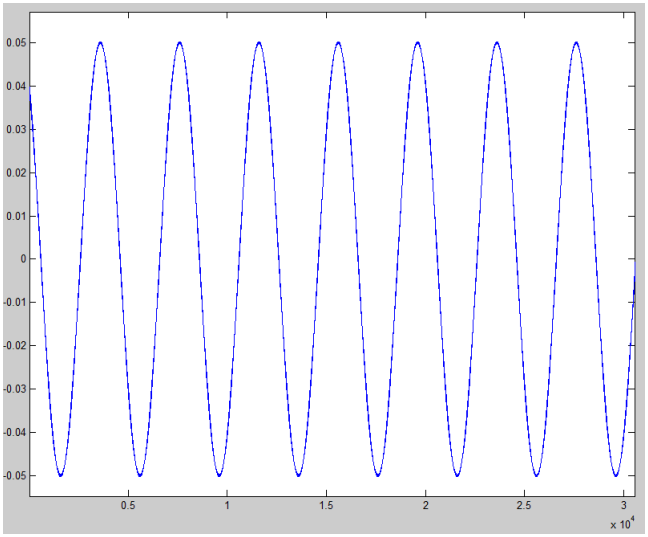

Figura 4.9: Coseno recuperado

```
% El metodo de recuperar las diferencias de fase, conceptualmente
% podria ser mas simple:

ph = angle(R); dph = ph(2:end) - ph(1:end-1); plot(dph);
```

Pero esto falla dado que "angle"devuelve fases que están comprendidas entre π y -π, por lo que cambios en torno a esos valores límites devuelven diferencias de fase erróneas.

```
% Para comprobar la robustez de la FM en recepcion podemos fallar por
% un 10% en la frecuencia de las portadoras locales, 2200 Hz en lugar
% de 2000Hz, y fallar en 90 grados en la fase de las portadoras
% locales, y ver que aun asi se recupera correctamente la forma
% del mensaje.

I_fm2 = conv(IQ.*cos(2*pi*2200*t + pi/2), lpf);
Q_fm2 = conv(-IQ.*sin(2*pi*2200*t + pi/2), lpf);

R2 = I_fm2 + i*Q_fm2;
dem2 = angle( R2(2:end) .* conj(R2(1:end-1)) );
plot(dem(500:end-500), 'b');
hold on;
plot(dem2(500:end-500), 'r');
hold off;
```

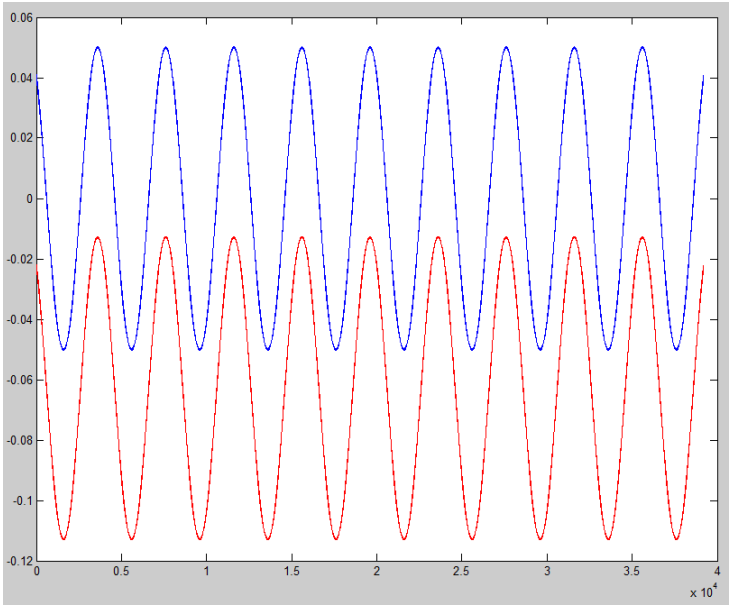

Figura 4.10: Comprobamos que el mensaje se recupera correctamente

4.3. Procesamiento de señal real en Matlab

Para demodular una señal FM procedente de una captura del espectro, en lugar de una simulación, repetiremos los pasos de demodulación sobre los datos contenidos en el fichero en que se aloja dicha señal capturada.

Comenzamos leyendo el fichero con la senal capturada:

```
nombreFichero = 'wfm_classicmusic_320ksps_60sec_cfloat.raw';

fs = 320e3;      %Frecuencia de muestreo a 320kHz
limiteTam = 10e6;    %Limite tamano muestreo 1000000 muestras

fid = fopen(nombreFichero, 'rb');
if limiteTam > 0
  d = fread(fid, limiteTam, 'float32');
else
  d = fread(fid, Inf, 'float32');
end;
fclose(fid);

if (size(d:1)  = 1)
  d = transpose(d);
end;
```

Código 1

Ya vienen demodulados en banda base tanto I como Q. En el fichero dos muestras consecutivas, $2i$ y $2i+1$, se corresponden al dato i-ésimo de las señales I y Q. Las consideramos por separado.

```
I_fm = d(1:2:end);
Q_fm = d(2:2:end);

% Procedemos a extraer las diferencias de fase, los cambios de
% frecuencia, el mensaje; son todos sinonimos, como antes:

R = I_fm+j*Q_fm;
dem = angle(R(1:end-1).*conj(R(2:end)));
clear d I_fm Q_fm R;

%quitamos las primeras muestras, no se habia puesto en marcha el sistema
dem = dem(5000:end);
dem = decimate(dem, 2, 'fir');
fs = fs/2;

anchoFFT = 2^12;
f = ([1:anchoFFT]-anchoFFT/2)/anchoFFT*fs;
plot(f, fftshift(abs(fft(dem(floor(length(dem)/2):
floor(length(dem)/2)+anchoFFT-1)))));
xlim([0 60e3]);
```

Código 2

En la figura 4.11 podemos ver el espectro de la señal capturada, resulta ser una señal donde el audio L+R se encuentra en banda base hasta 15KHz y más arriba aparecen la portadora piloto en 19KHz y el audio L-R modulado en DSB-SC en torno a 38KHz. Más allá la informacion RDBS centrada en 57kHz.

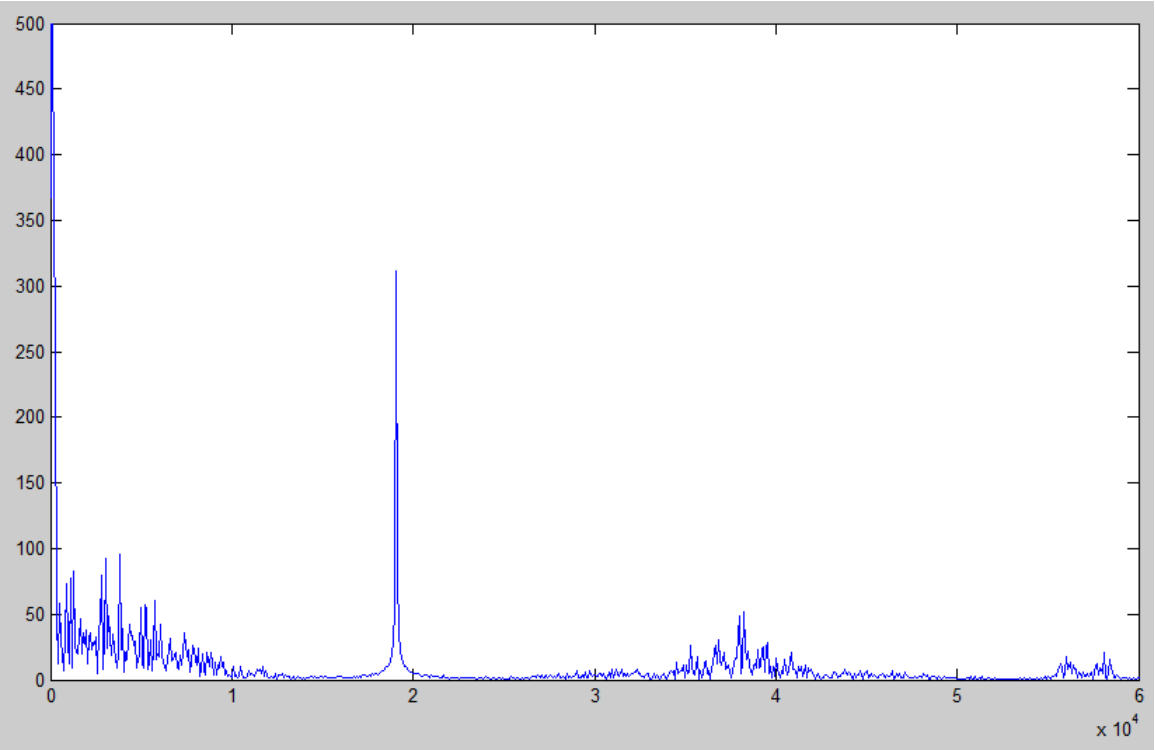

Figura 4.11: Dibujo del espectro

Podemos ver cómo el espectro de la señal coincide con el esquema teórico de una señal FM comercial, visto anteriormente, y que repetimos en la figura 4.12

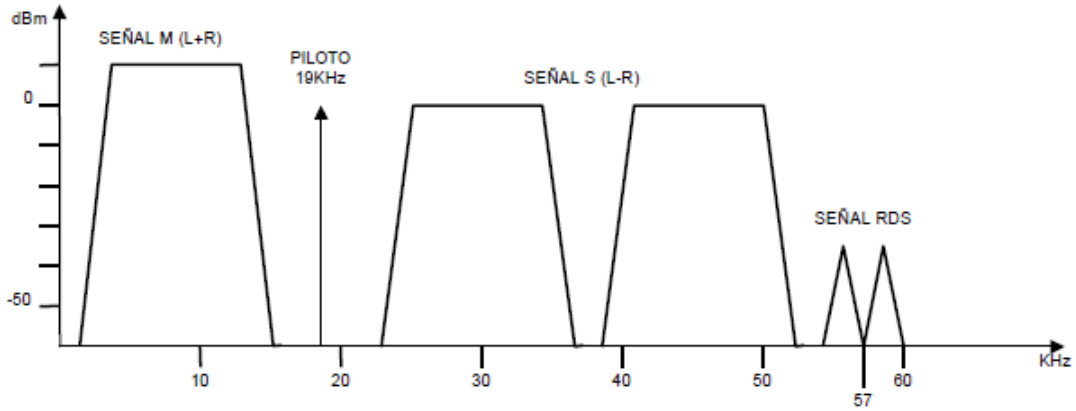

Figura 4.12: Espectro FM Comercial española

4.3.1. Extracción de la portadora piloto

Para extraer la portadora piloto, optamos por un filtro FIR*, en particular un filtro de respuesta impulsiva finita de Parks-McClellan, usando el comando "firpm" de MatLab:

```
wo = 19000/(fs/2);  bw = 1000/(fs/2);
aRF_19k=1; bRF_19k = firpm(380, [0 wo-2*bw  wo-bw/2  wo+bw/2  wo+2*bw 1],
[0 0 1 1 0 0]);
```

Código 3

Si optásemos por un filtro IIR (Infinite Impulse Response), tendríamos menos taps, pero un retardo de muestras o "group delay" menos previsible, además de distorsión de fase.

```
[bRF_19k,aRF_19k] = iirpeak(wo,bw);
```

Podemos ver con detalle el funcionamiento de ambos filtros usando la herramienta "fvtool", Filter Visualization Tool. En la figura 4.13 vemos la respuesta en magnitud y de este filtro.

```
fvtool(bRF_19k,aRF_19k);
```

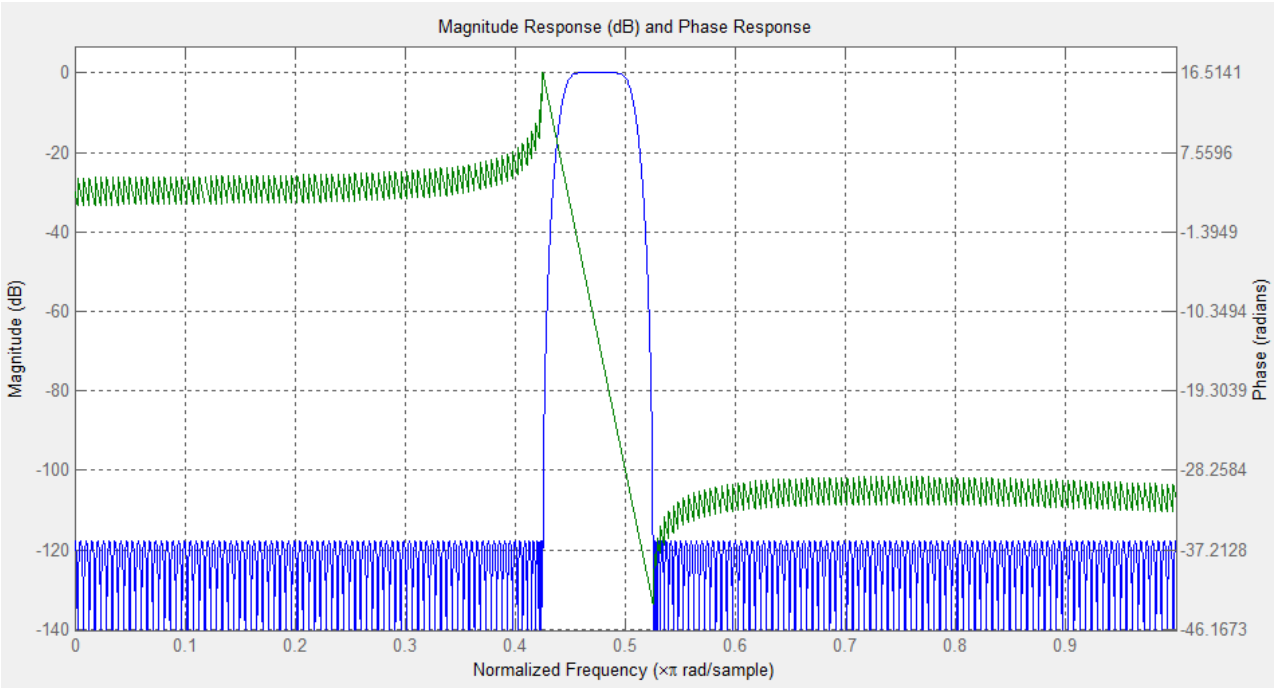

Figura 4.13: Respuesta en magnitud frente a fase filtro FIR

*Más acerca del filtro FIR en la referencia [11] de la bibliografía.

Podemos ver que tenemos un retardo constante de 190 muestras en la figura 4.14

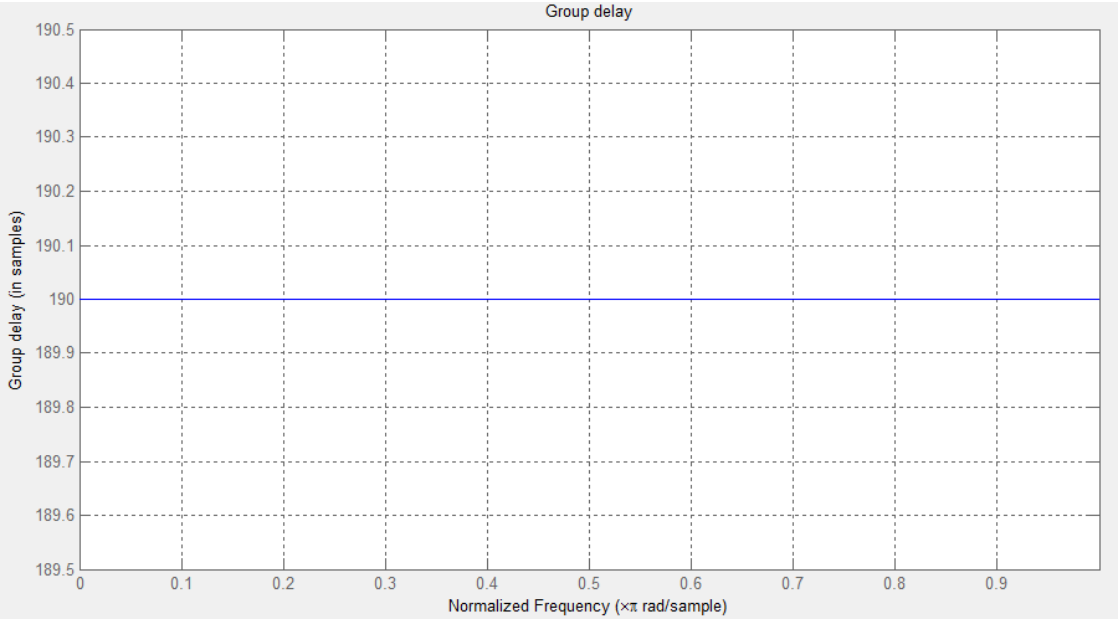

Figura 4.14: Group delay filtro FIR

Además con la visualización de filtros "fvtool", podemos ver la respuesta impulsiva (Figura 4.15), respuesta escalón, la ubicación de polos y ceros del filtro (Figura 4.16), y una serie de coeficientes para poder implementar matemáticamente el filtro.

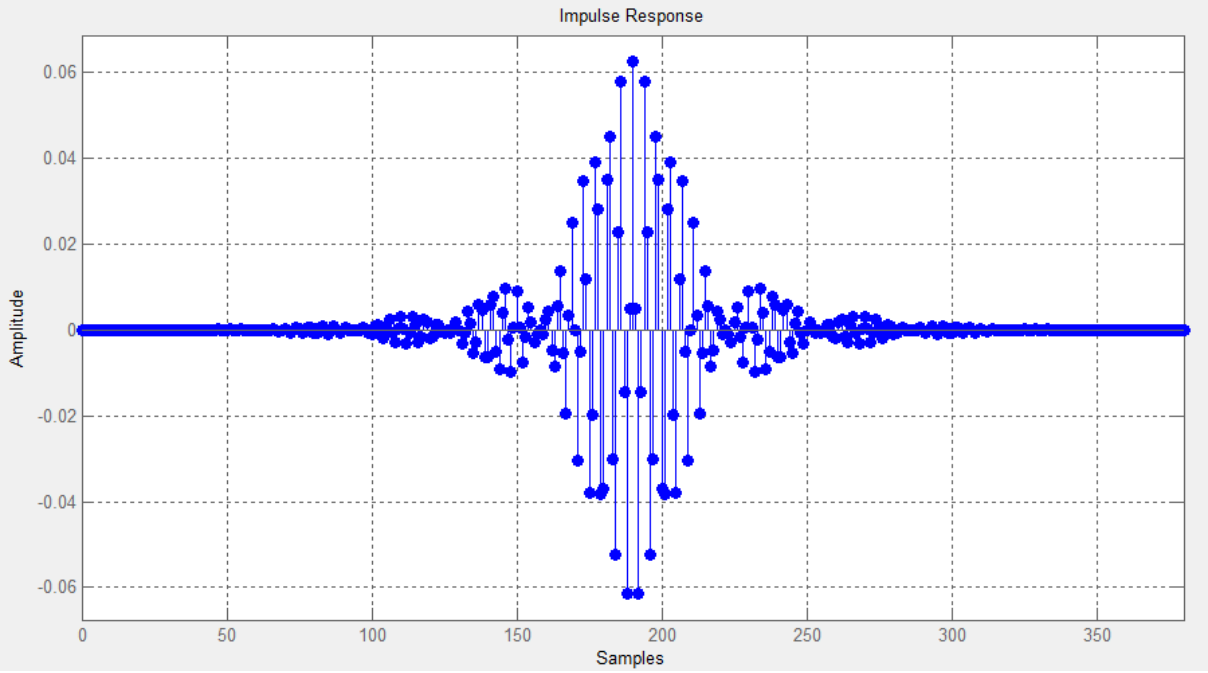

Figura 4.15: Respuesta impulsiva filtro FIR

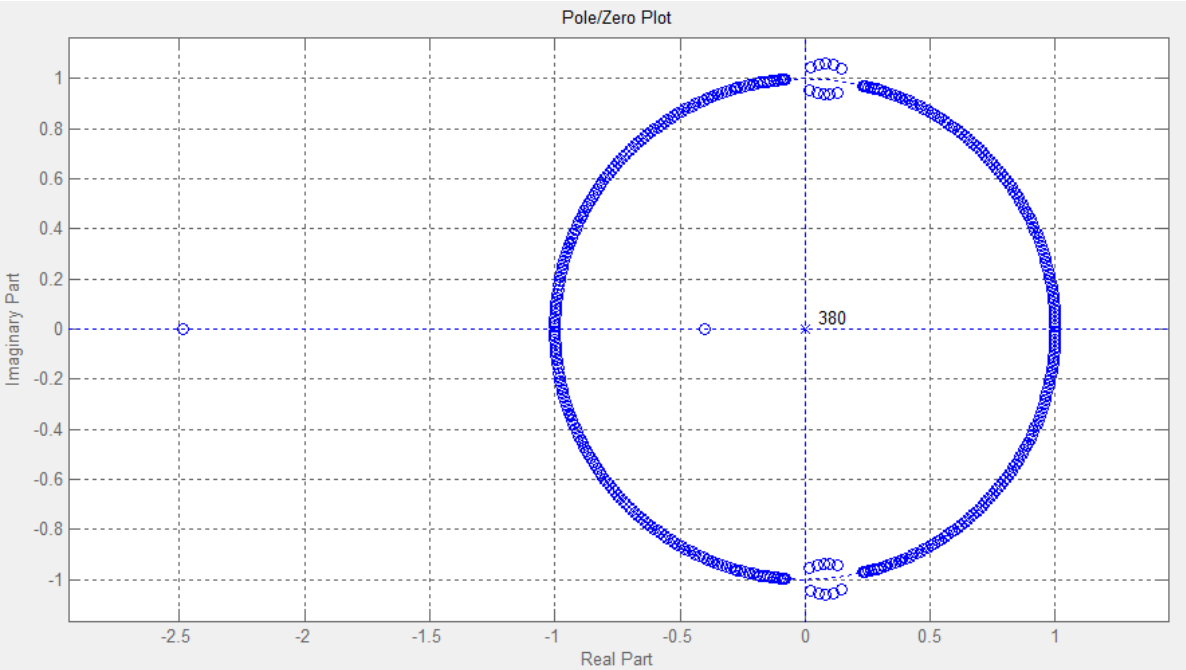

Figura 4.16: Dibujo de ceros y polos filtro FIR

Si utilizamos un filtro de respuesta al impulso infinita, vemos con el "fvtool" las diferencias respecto al FIR, en primer lugar vemos la respuesta en magnitud y en fase (Figura 4.17), a continuación podemos ver que aunque el "group delay" es menos, es muy poco previsible, ya que en el rango de frecuencias de interés, sufre grandes distorsiones en pocos kHz (Figura 4.18).

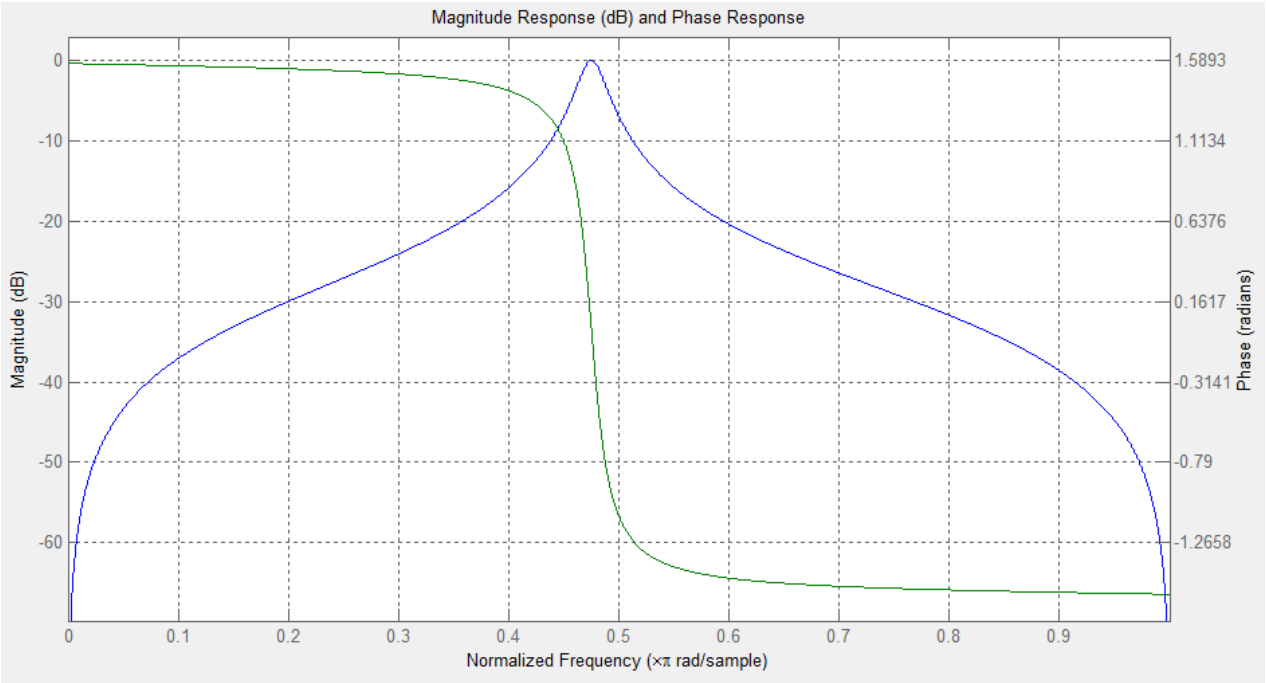

Figura 4.17: Respuesta en magnitud frente a fase filtro IIR

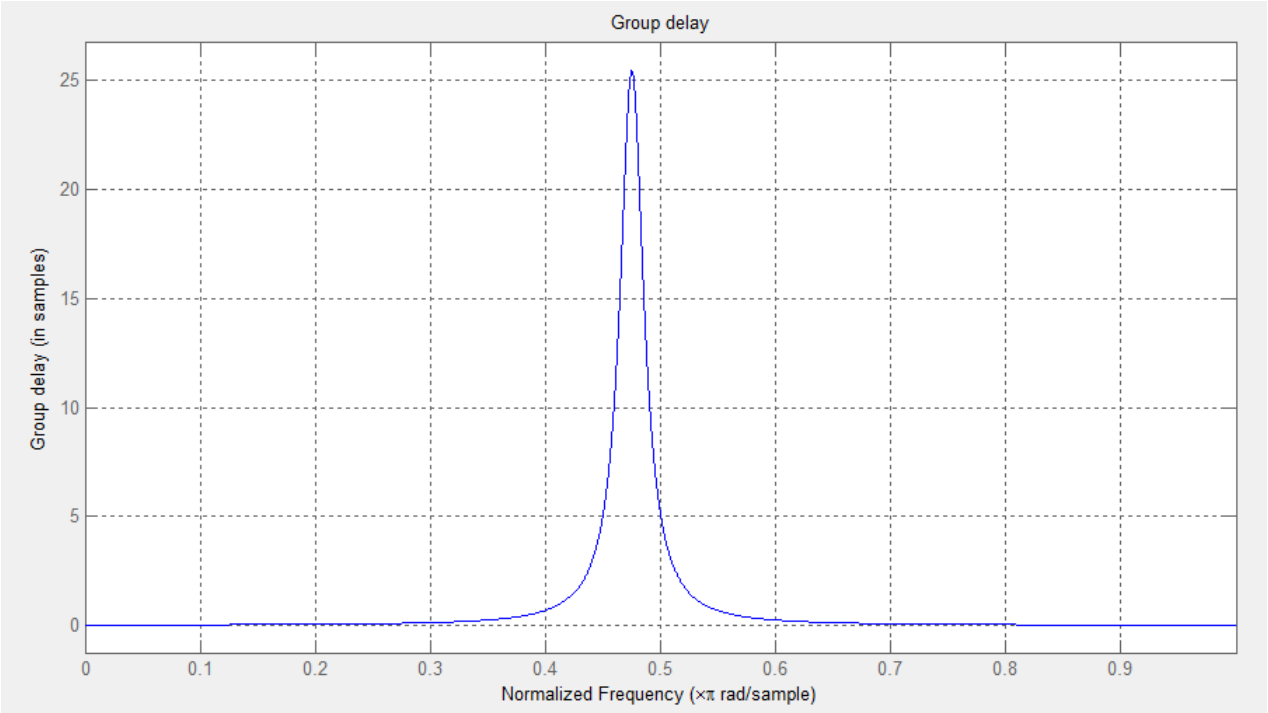

Figura 4.18: Group delay filtro IIR

```
piloto19k = filter(bRF_19k,aRF_19k, 1/0.147*dem);    % compensando cos19k

[gd,w]=grpdelay(bRF_19k,aRF_19k,2000);
[¬,pos19kGD]=min(abs(w-19000/(fs/2)*pi));
grpDelay19k=gd(pos19kGD);
```

En el caso de FIR de mismos taps, todos llevan el mismo “group delay”, quitamos esas muestras del principio de la señal “dem”, y del final de la señal “piloto”

```
dem=dem(round(grpDelay19k+1):end);
piloto19k = piloto19k(1:end-round(grpDelay19k));
```

Dibujamos la portadora para comprobar la eficacia del filtro FIR:

```
 plot(f, fftshift(abs(fft(piloto19k(floor(length(dem)/2):
 floor(length(dem)/2)+anchoFFT-1)))));
 xlim([0 20e3]);
```

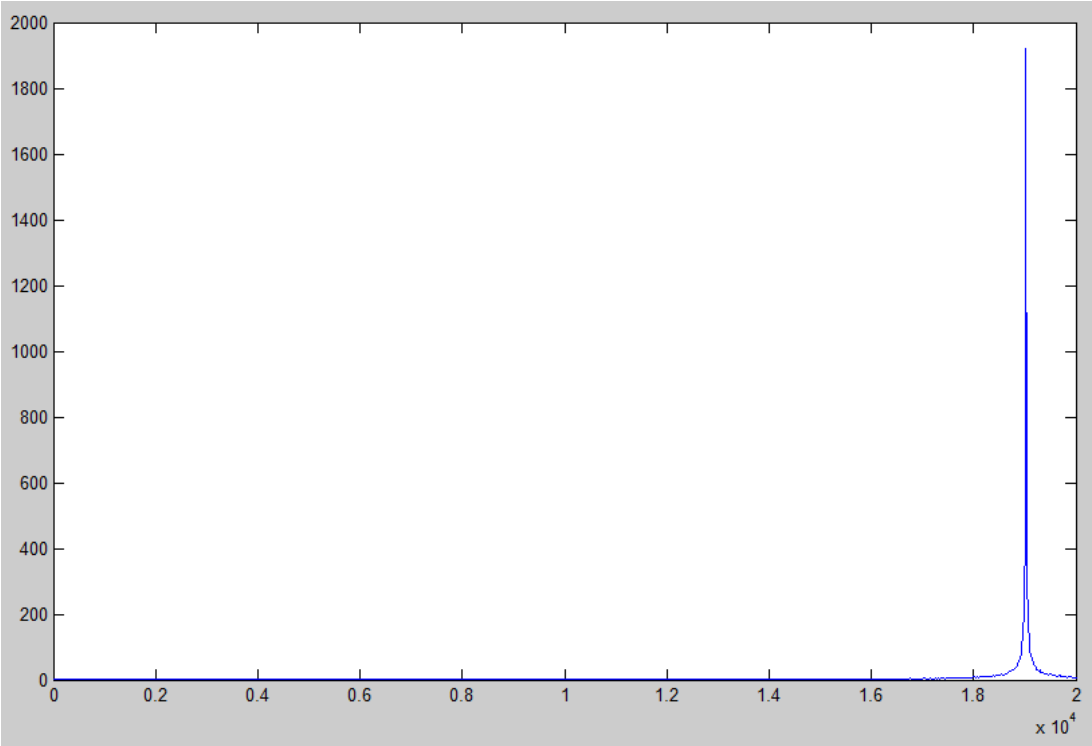

Figura 4.19: Espectro de 0 a 20kHz

Podemos ver que en 19kHz tenemos la portadora, mientras que no tenemos señal en el resto de frecuencias.

A partir de la portadora piloto de 19kHz, podemos obtener las pilotos de 38kHz y de 57kHz, utilizando propiedades matemáticas trigonométricas:

$$\cos 2\alpha = 2\cos^2 \alpha - 1 \tag{4.1}$$

$$\cos 3\alpha = 4\cos^3 \alpha - 3\cos \alpha \tag{4.2}$$

Obtenemos las portadoras múltiplos de la de 19kHz aplicando las fórmulas (4.1) y (4.2)

```
1 cos38 = piloto19k.*piloto19k*2-1;
2 cos57 = piloto19k.*piloto19k.*piloto19k*4-3*piloto19k;
```

Código 4

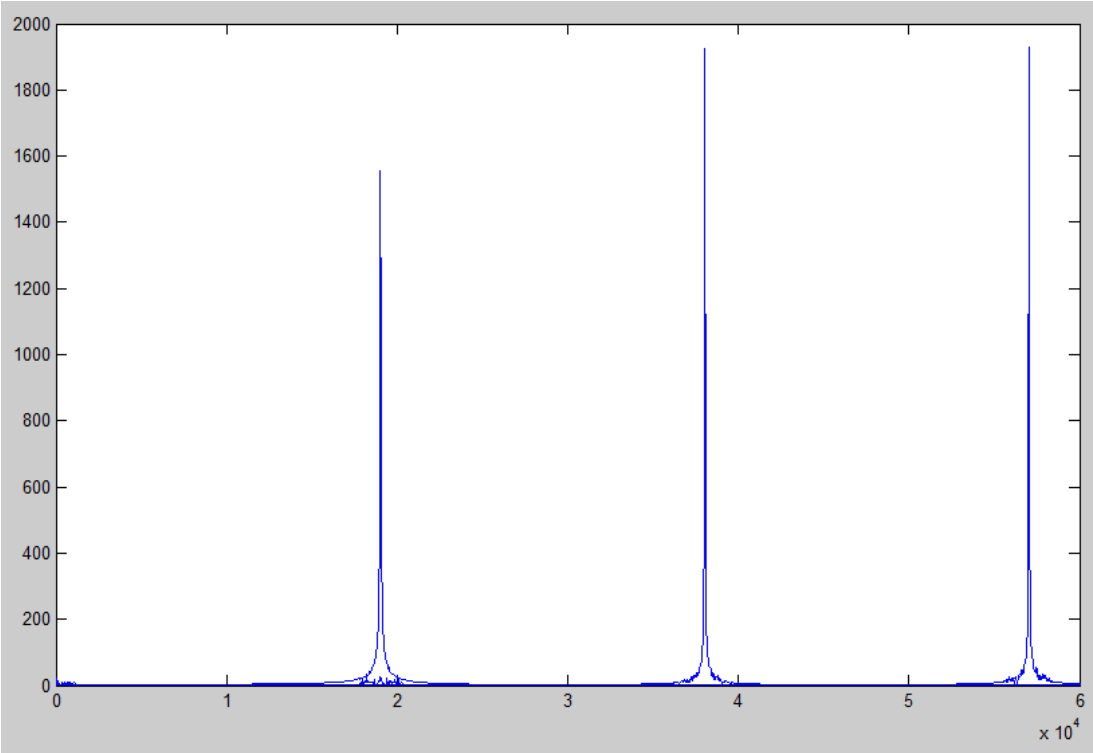

Figura 4.20: Portadoras de 19, 38 y 57KHz

4.3.2. Audio mono

Para filtrar el audio hemos optado por un filtro Chebyshev[**], con el que se consigue una caída de la respuesta en frecuencia más pronunciada en frecuencias bajas debido a que permiten rizado en alguna de sus bandas (paso o rechazo). A diferencia del Filtro de Butterworth donde los polos se distribuyen sobre una circunferencia, los polos del filtro Chebyshev lo hacen sobre una elipse, por lo que sus ceros se encuentran en el eje imaginario.

Tenemos dos tipos de filtro Chebyshev:

- TIPO 1: Son filtros que únicamente tienen polos, presentan un rizado constante en la banda pasante y presentan una caída monótona en la banda de rechazo.

- TIPO 2: Presentan ceros y polos, su rizado es constante en la banda de rechazo y además presentan una caída monotónica en la banda pasante.

Por tanto, nos conviene utilizar un filtro Chebyshev de tipo 2, ya que el rizado lo presenta en la banda de rechazo.

La ganancia en la banda de paso de un filtro de Chebyshev de nivel 2 es:

$$G_n(w, w_0) = \frac{1}{\sqrt{1 + \frac{1}{\varepsilon^2 T_n^2(w_0/w)}}}$$

Donde:

N, es el orden del filtro
ε, es el factor de rizado
w, es la frecuencia angular
w_0, es la frecuencia de corte
T_n, es el polinomio de Chebyshev de orden n

En la banda de rechazo, el polinomio de Chebyshev oscila entre -1 y 1, de modo que la ganancia oscilará entre cero y:

$$\frac{1}{\sqrt{1 + \frac{1}{\varepsilon^2}}}$$

Podemos ver en la figura 4.21 la respuesta en frecuencia del filtro de Chebyshev pasa-bajas de tipo 2, con una ε de 0,01

[**]Más acerca del filtro Chebyshev en la referencia [12] de la bibliografía.

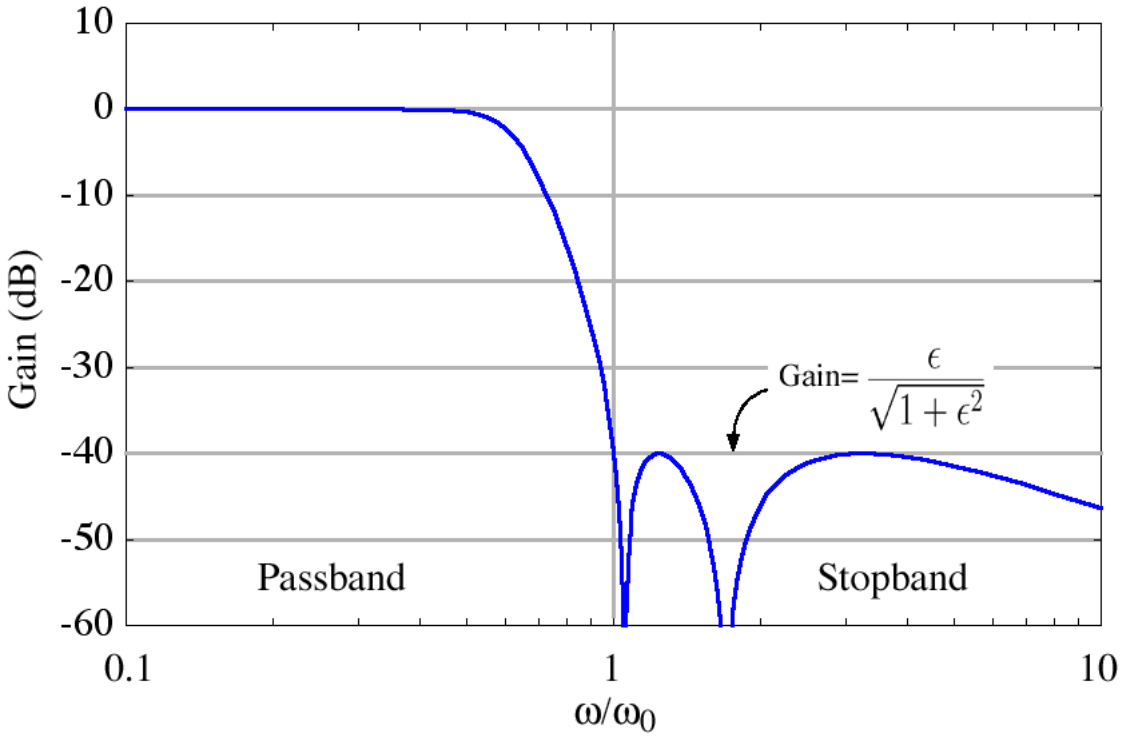

Figura 4.21: Respuesta en magnitud filtro Chebyshev tipo 2

El objetivo de este filtro es aislar los primeros 15kHz de señal mono, asegurándonos de eliminar la frecuencia correspondiente a la portadora de 19kHz, por tanto analizamos el filtro con la herramienta "fvtool":

El filtro en matlab funciona con el comando "cheby2", y se le introducen los siguientes parámetros:

$[B, A] = cheby2(N, R, Wst)$

Donde N es el orden del filtro, R es el rizado en la banda de rechazo y Wst es el límite entre la banda de paso y la de rechazo (-3dB).

El objetivo es que se rechace completamente la frecuencia de 19kHz, por lo que hemos de intentar que un pico negativo del rizado coincida en esa frecuencia, introduciéndole los siguientes parámetros se obtiene un gran resultado:

```
1 [bLPF_15k, aLPF_15k] = cheby2(15, 50, 15000/(fs/2));
2 fvtool(bLPF_15k,aLPF_15k);
```

Código 5

En la figura 4.22 podemos ver la respuesta en magnitud y fase, y cómo en la banda de rechazo se produce el rizado, tal y como explicamos en la teoría del filtro.

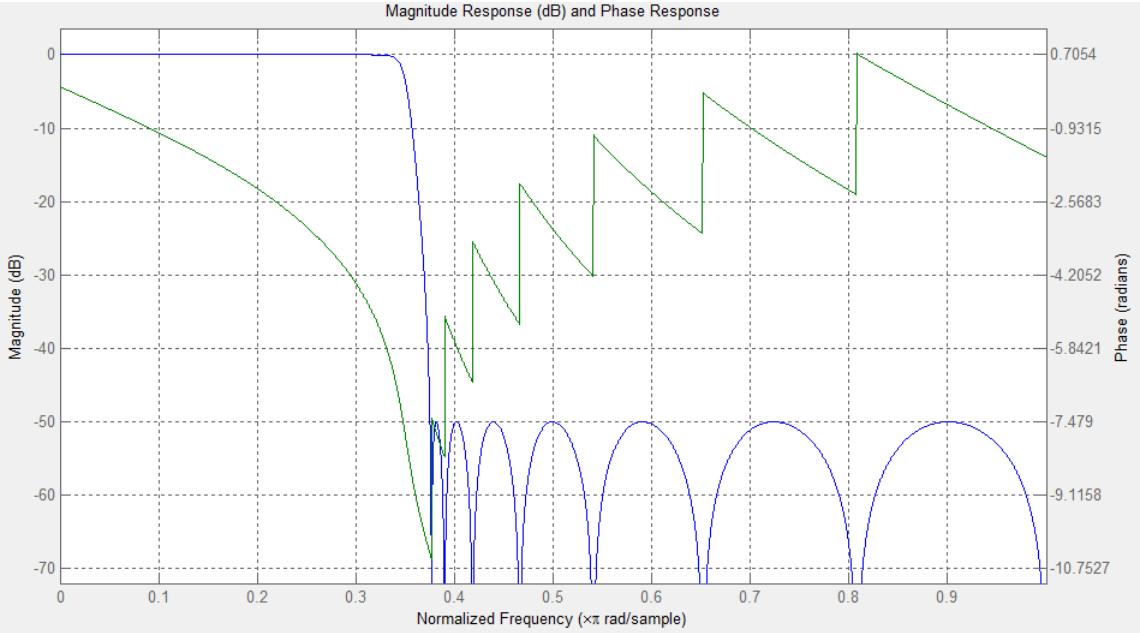

Figura 4.22: Respuesta en magnitud frente a fase filtro Chebyshev tipo 2

```
soundLpR = filter(bLPF_15k, aLPF_15k,dem);
```

Código 6

Como nuestro dispositivo es capaz de muestrear a un amplio rango de tasas de muestreo, hecemos una iteración "if-else" para pasar de nuestra frecuencia de muestreo a una tasa de muestreo audible por una tarjeta de sonido de un PC.

```
% decimacion en cascada para pasar a valores audibles
if fs ==320e3 % para pasar de 320kHz a 16kHz -> downsampling 4 y 5, ...
    factor 20
     soundLpR = decimate(decimate(soundLpR, 4), 5);
     fsAudio = fs/4/5;
elseif fs==160e3 % para pasar de 160kHz a 16kHz -> downsampling 2 y ...
    5, factor 10
     soundLpR = decimate(decimate(soundLpR, 2), 5);
     fsAudio = fs/2/5;
elseif fs==512e3 % para pasar de 512kHz a 16kHz -> downsampling 4 y ...
    8, factor 32
     soundLpR = decimate(decimate(soundLpR, 8), 4);
     fsAudio = fs/4/8;
elseif fs==256e3 % para pasar de 256kHz a 16kHz -> downsampling 4 y ...
    4, factor 16
     soundLpR = decimate(decimate(soundLpR, 4), 4);
     fsAudio = fs/4/4;
else
     disp('Fs no contemplada, anada la decimacion adecuada');
     return;
end;
```

Código 7

Pasamos a dibujar el espectro audio L+R (Figura 4.23) para comprobar que no queda rastro de la portadora:

```
anchoFFTaudio = 2^9;
fAudio = ([1:anchoFFTaudio]-anchoFFTaudio/2)/anchoFFTaudio*fsAudio;
plot(fAudio, fftshift(abs(fft(soundLpR(floor(length(soundLpR)/
2):floor(length(soundLpR)/2)+anchoFFTaudio-1)))));
xlim([0 ,25e3]);
```

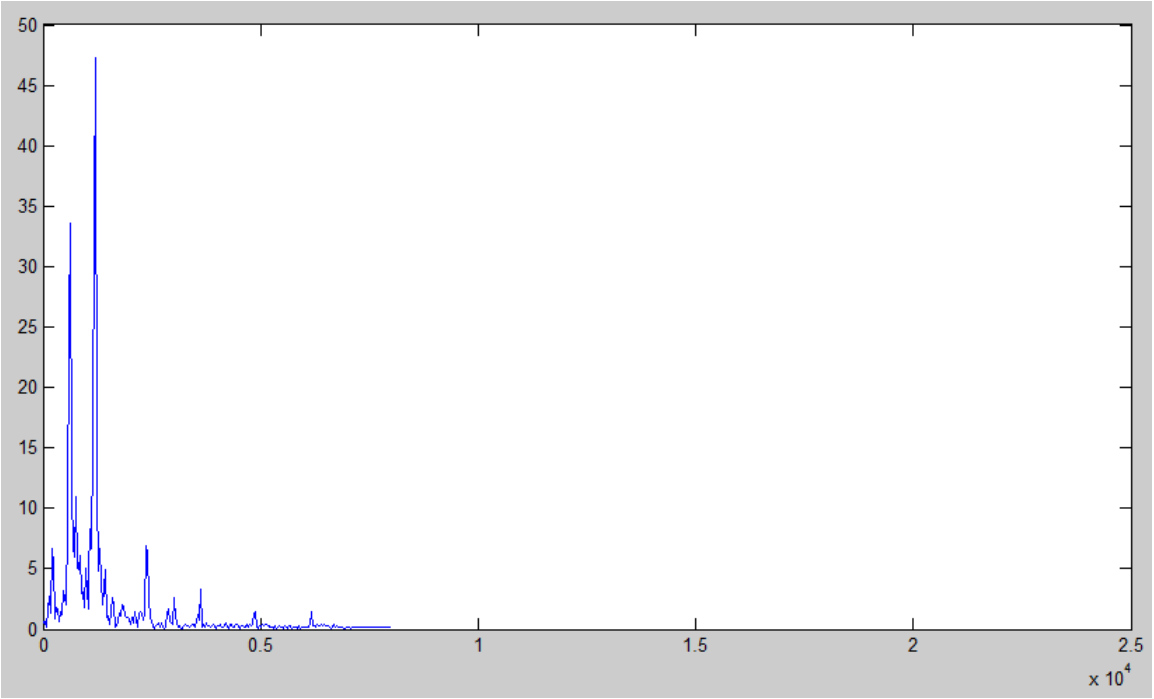

Figura 4.23: Espectro señal L+R

En efecto, la señal esta correctamente filtrada. Pasamos a reproducir el audio, debe salir sin ningún tipo de ruido.

```
r=1:1e4; plot(r,soundLpR(r));
quitarInicio = 4000;
soundLpR = soundLpR(quitarInicio:end);

soundLpR = soundLpR - mean(soundLpR);
maxSoundLpR = max(soundLpR);
soundLpR = soundLpR/maxSoundLpR;
sound(soundLpR, fsAudio);
```

Código 8

4.3.3. Audio estereo

Para obtener el audio L-R y poder reproducir la señal estéreo, creamos la señal "shift38k", que no es más que la sub-portadora de 38kHz generada a partir de la de 19kHz, multiplicada punto a punto con la señal "dem" que es la señal FM demodulada. Con esta multiplicación mandamos la señal a banda base.

La función "filter" de Matlab usada:

$$y = filter(b, a, x)$$

Filtra los datos de entrada, x, utilizando una función de transferencia racional definida por el numerador y denominador, es decir, los coeficientes b y a. La señal es filtrada con los mismos coeficientes a y b usados por el filtro Chebyshev. anterior

```
shift38k = 2*dem.*cos38;
plot(f, fftshift(abs(fft(shift38k(floor(length(shift38k)/
2):floor(length(shift38k)/2)+anchoFFT-1)))));
xlim([0 20e3]);
soundLmR =  filter(bLPF_15k, aLPF_15k, shift38k);
plot(f, fftshift(abs(fft(soundLmR(floor(length(soundLmR)/
2):floor(length(soundLmR)/2)+anchoFFT-1)))));
xlim([0 25e3]);
```

Código 9

Repetimos la iteración "if-else":

```
if fs ==320e3 % para pasar de 320kHz a 16kHz -> downsampling 4 y 5, ...
    factor 20
     soundLmR = decimate(decimate(soundLmR, 4), 5);
     fsAudio = fs/4/5;
elseif fs==160e3 % para pasar de 160kHz a 16kHz -> downsampling 2 y ...
    5, factor 10
     soundLmR = decimate(decimate(soundLmR, 2), 5);
     fsAudio = fs/2/5;
elseif fs==512e3 % para pasar de 512kHz a 16kHz -> downsampling 4 y ...
    8, factor 32
     soundLmR = decimate(decimate(soundLmR, 8), 4);
     fsAudio = fs/4/8;
elseif fs==256e3 % para pasar de 256kHz a 16kHz -> downsampling 4 y ...
    4, factor 16
     soundLmR = decimate(decimate(soundLmR, 4), 4);
     fsAudio = fs/4/4;
else
     disp('Fs no contemplada, anada la decimacion adecuada');
     return;
end;
```

Código 10

Comprobamos una vez más (Figura 4.24) que se ha filtrado las frecuencias no deseadas:

```
plot(fAudio, fftshift(abs(fft(soundLmR(floor(length(soundLpR)/
2):floor(length(soundLpR)/2)+anchoFFTaudio-1)))));

soundLmR = soundLmR(quitarInicio:end);

soundLmR = soundLmR - mean(soundLmR);
soundLmR = soundLmR/maxSoundLpR*.2;
```

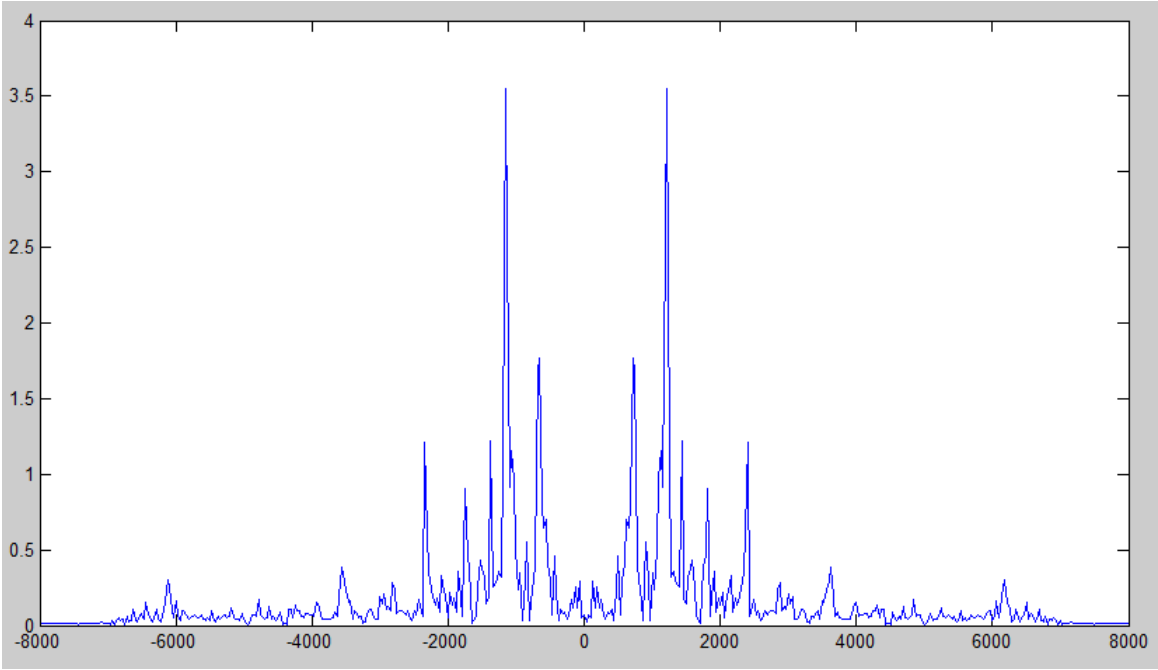

Figura 4.24: Espectro señal L-R

Por último queda sumar y restar las señales para obtener el audio de cada lado, para obtener el sonido del lazo derecho (R) hay que sumar L+R y L-R, y para obtener el sonido del lado izquierdo, restamos:

```
soundR = (soundLpR - soundLmR);
soundL = (soundLpR + soundLmR);
```

Como paso final, reproducimos los audios izquierdo y derecho:

```
sound([soundR; soundL]', fsAudio);
```

Código 11

4.3.4. RBDS

La señal FM demodulada, además de contener el audio, contiene un stream de bits RBDS (Radio Broadcast Data System) conocido como sistema de radiodifusión de datos. Está en una subportadora a 57KHz, es decir, a tres veces la frecuencia de la portadora de 19KHz.

Está modulada en BPSK[***], que significa "Binary Phase Shift Keying" o Modulación Binaria por Desplazamiento de Fase, cada bit enviado se codifica según los símbolos que aparecen en la figura 4.25

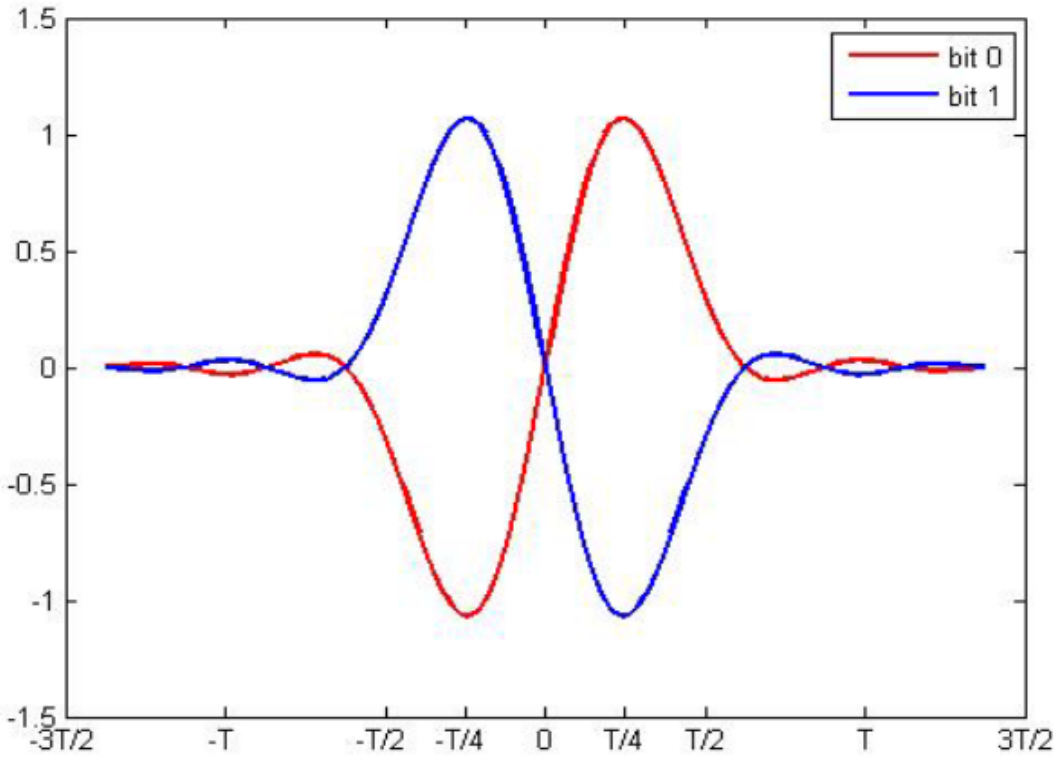

Figura 4.25: Codificación de bits

[***]Información acerca de las modulaciones digitales usadas en la demodulación RBDS en la reserencia [13] de la bibliografía

Explicaremos ahora la modulación y demodulación PSK

La modulación por desplazamiento de fase consiste en hacer variar la fase de la portadora entre un número de valores discretos. Se diferencia de la modulación de fase convencional es que la señal moduladora es una señal digital y por tanto tiene un número de estados limitado. Podemos ver en la figura 4.26 los efectos de la portadora y moduladora sobre la modulada.

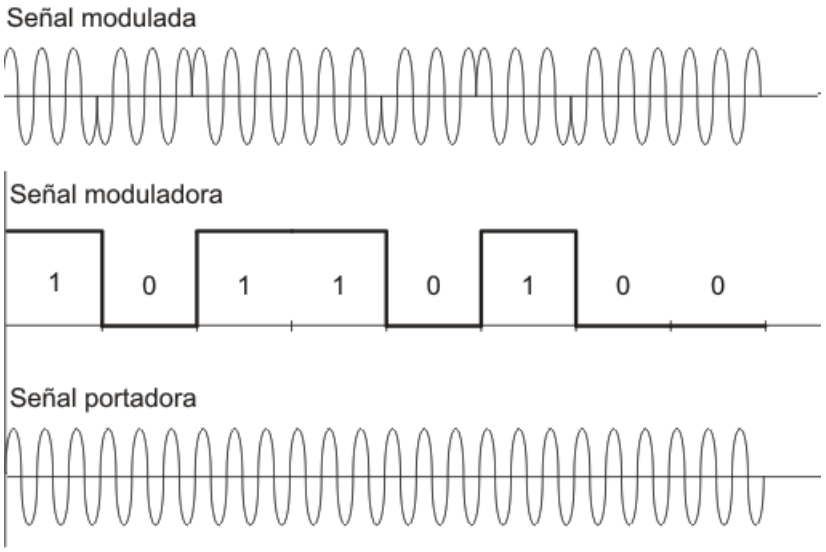

Figura 4.26: Formas de onda en PSK

La modulación BPSK es la más sencilla de todas, puesto que sólo emplea dos símbolos, con 1 bit de información cada uno. Tiene la ventaja de que es el que presenta mayor inmunidad al ruido, puesto que la distancia entre bits es máxima, de 180 grados. Dichos símbolos suelen tener un valor de salto de fase de 0^{o} para el bit 1 y de 180^{o} para el bit 0. Esto lo podemos ver en la constelación para BPSK mostrada en la figura 4.27

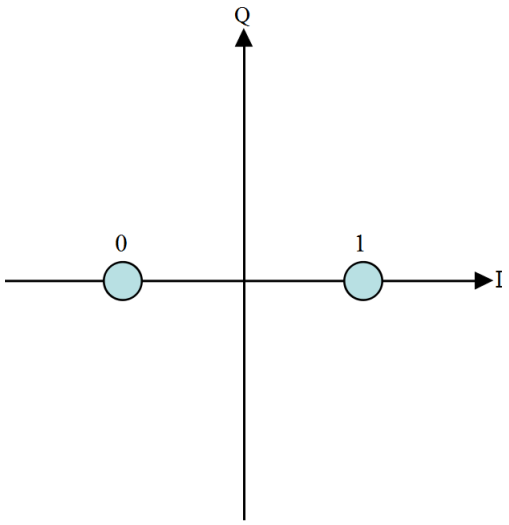

Figura 4.27: Constelación BPSK

Podemos ver la fase de salida contra la relación de tiempo para una forma de onda BPSK en la figura 4.28. El espectro de salida de un modulador de BPSK es, sólo una señal de doble banda lateral con portadora suprimida, donde las frecuencias laterales superiores e inferiores están separadas de la frecuencia de la portadora por un valor igual a la mitad de la razón de bit. En consecuencia, el mínimo ancho de banda (f N) requerido, para permitir el peor caso de la señal de salida del BPSK es igual a la razón de bit de entrada.

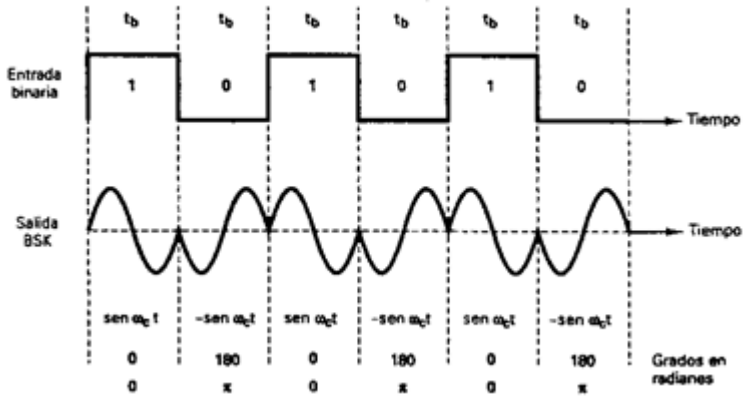

Figura 4.28: Fase de salida BPSK

Pasamos a la obtención de la señal RBDS en Matlab, en primer lugar desplazamos una muestra adicional la señal cos57 antes de multiplicar a dem por ella, la podemos ver en la figura 4.29

```
nDesfase = 1
    desfase = zeros(1,nDesfase);
shift57k = 2*[dem,desfase].*[desfase,cos57];

plot(f, fftshift(abs(fft(shift57k(floor(length(shift57k)/
2):floor(length(shift57k)/2)+anchoFFT-1)))));
```

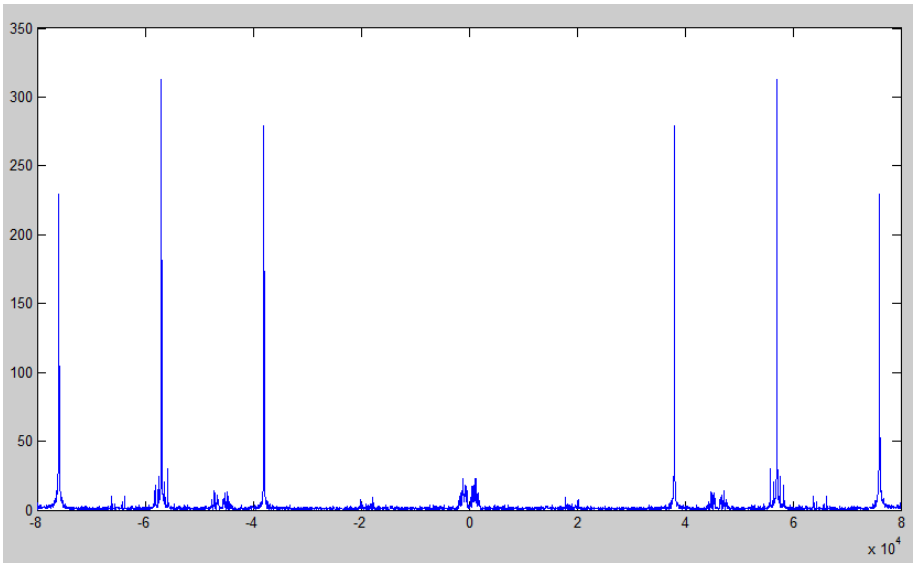

Figura 4.29: Espectro señal coseno(57) desplazada una muestra

Filtramos con un filtro butterworth la señal desplazada de cos57 para obtener la señal RBDS, podemos ver la señal en las figuras 4.30 y 4.31

```
[bPB_1d2k, aPB_1d2k] = butter(5,2300/(fs/2));

RDBS = filter(bPB_1d2k, aPB_1d2k, shift57k);
% plot(f, fftshift(abs(fft(RDBS(floor(length(RDBS)/
%2):floor(length(RDBS)/2)+anchoFFT-1)))));
RDBS = RDBS(5000:end);
r=1e4+[1:1e4];
figure;
plot(RDBS(r),'+-');
title(sprintf('%d',nDesfase));
```

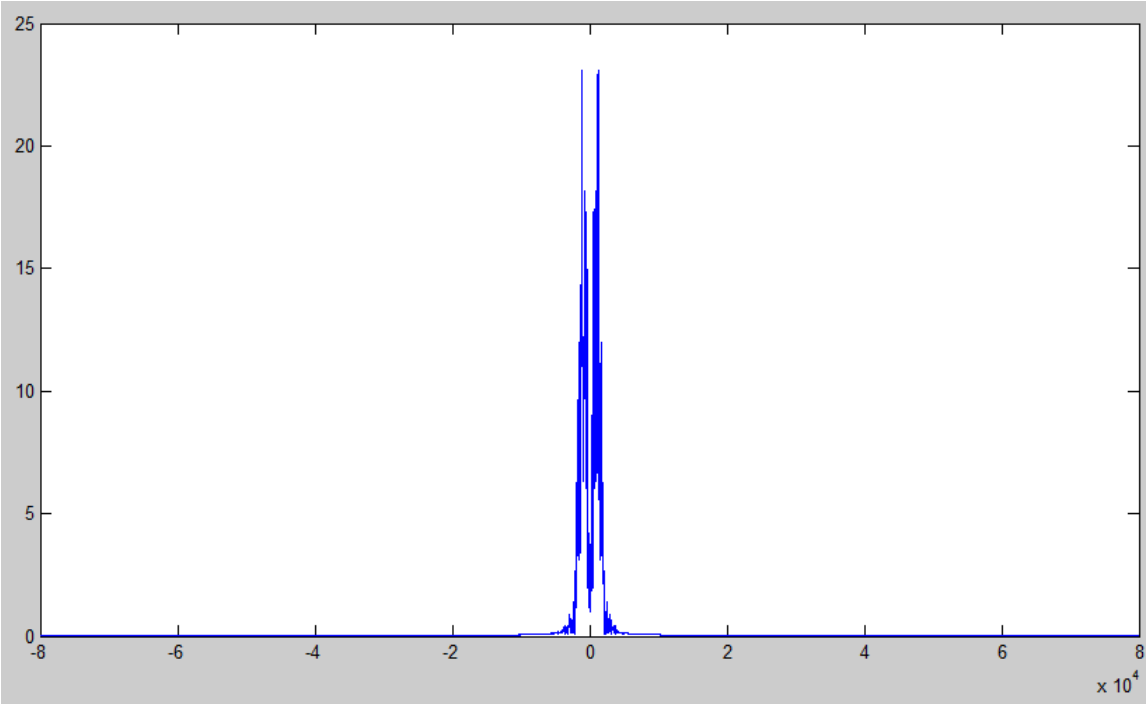

Figura 4.30: Espectro señal RBDS desplazada una muestra

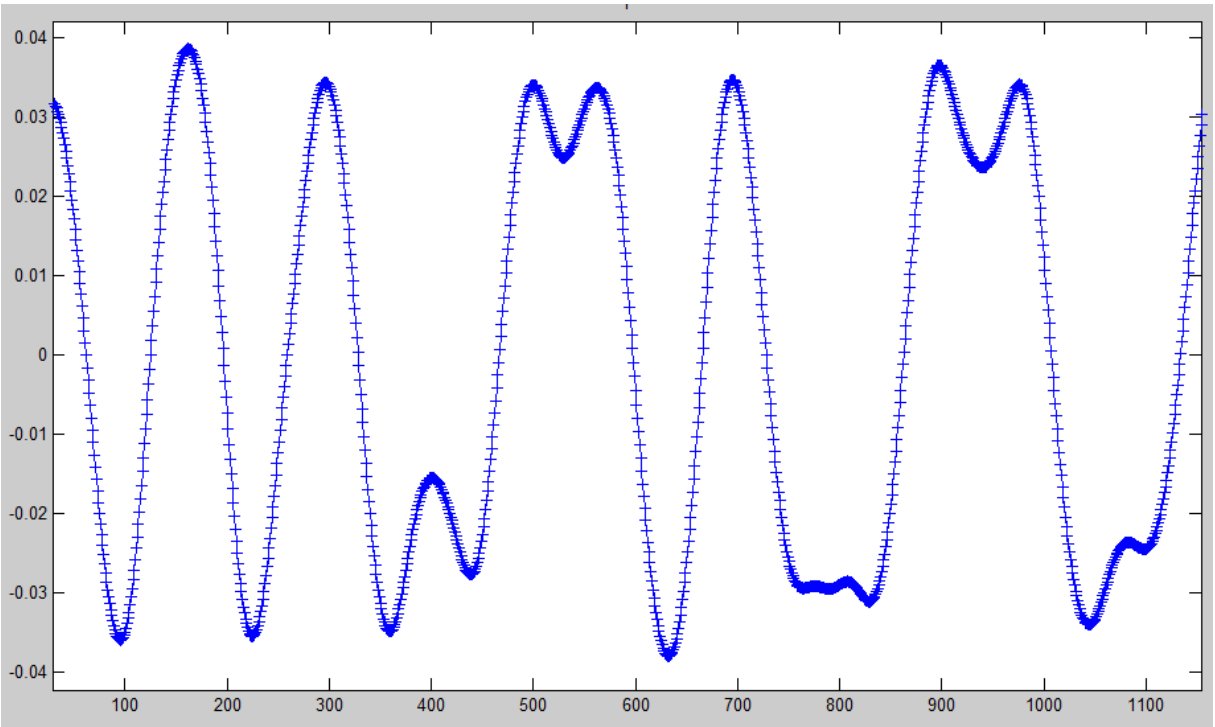

Figura 4.31: Señal RBDS

Al igual que ocurría con el audio mono y estéreo, tenemos que decimar la señal RBDS en un factor en función de la frecuencia de muestreo.

```
if fs ==320e3 % para pasar de 320kHz a 16kHz -> downsampling 4 y 5, ...
    factor 20
     RDBS = decimate(decimate(RDBS, 4), 5);
     fsRDBS = fs/4/5;
elseif fs==160e3 % para pasar de 512kHz a 16kHz -> downsampling 4 y ...
    8, factor 32
     RDBS = decimate(decimate(RDBS, 2), 5);
     fsRDBS = fs/2/5;
elseif fs==512e3 % para pasar de 512kHz a 16kHz -> downsampling 4 y ...
    8, factor 32
     RDBS = decimate(decimate(RDBS, 8), 4);
     fsRDBS = fs/4/8;
elseif fs==256e3 % para pasar de 512kHz a 16kHz -> downsampling 4 y ...
    8, factor 32
     RDBS = decimate(decimate(RDBS, 4), 4);
     fsRDBS = fs/4/4;
else
     disp('Fs no contemplada, anada la decimacion adecuada');
     return;
end;
```

Escribimos en un archivo con extensión ".float32" la señal RBDS normalizada:

```
RDBS = RDBS -mean(RDBS);
RDBS = RDBS/max(RDBS);

fid = fopen([nombreFichero,'.rdbs'], 'wb');
fwrite(fid, RDBS, 'float32');
fclose(fid);
```

4.3.5. Demodulación RBDS

Abrimos el archivo creado en el último paso de la modulación RBDS, fijamos la frecuencia de muestreo en 16kHz, y normalizamos la señal, restándole la media.

Creamos las variables "periodo" y "ejemplo", periodo consiste en redondear el calculo del periodo base. Y ejemplo es una señal en el que tomamos 13 muestras, que es lo que vale el periodo, que servirá para encontrar más adelante los cruces por cero, y cuando la señal se engancha con la RBDS.

```
fid = fopen('wfm_classicmusic_320ksps_60sec_cfloat.raw.rdbs', 'rb');
fs = 16e3;

rdbs = fread(fid, Inf, 'float32');
fclose(fid);
rdbs = transpose(rdbs);

r = rdbs - mean(rdbs);


periodo = round(fs/(57000/48));
ejemplo = cos(2*pi*[1:periodo]/periodo-pi/2);
```

El siguiente paso es crear las señales "pulsos" y "cruces" , que son la convolución de la señal ejemplo con la señal normalizada de la RBDS, además, la señal "ejemplo" ha sido invertida en tiempo con la función de matlab "fliplr" que significa "flip left to right" por lo que pone los valores de un vector al revés. Para obtener la señal "cruces", aplicamos la misma idea, pero la señal ejemplo está además en valor absoluto. Con la orden "same" dentro de la función convolución, devolvemos el tamaño del primer término, es decir, de la señal "r" o la RBDS normalizada.

Con objeto de comprobar el funcionamiento y representar los pasos de la recuperación trabajamos primero con una versión acortada de la RDS, redefinimos los pulsos y los cruces en ese rango de tiempos.

```
pulsos = conv(r, fliplr(ejemplo),'same');
cruces = conv(r, fliplr(abs(ejemplo)),'same');


inicio=15*periodo;
fin = inicio+25*periodo;
rango = [inicio:fin];


rds_corto = r(rango);


pulsos_corto = pulsos(rango)/max(abs(pulsos));
cruces_corto = cruces(rango)/max(abs(cruces));
```

```
clf;
subplot(211);
hold on;
plot(rds_corto);
 % plot(rds_sin_ruido,'r--','linewidth', 2);
subplot(212);
plot(pulsos_corto)
hold on;
plot(cruces_corto);
```

Pasamos a dibujar los pulsos y los cruces, podemos verlo en la figura 4.32. La señal cruces es máxima cuando la RBDS cambia de estado, mientras que la señal pulso es máxima cuando estamos en el centro de los pulsos.

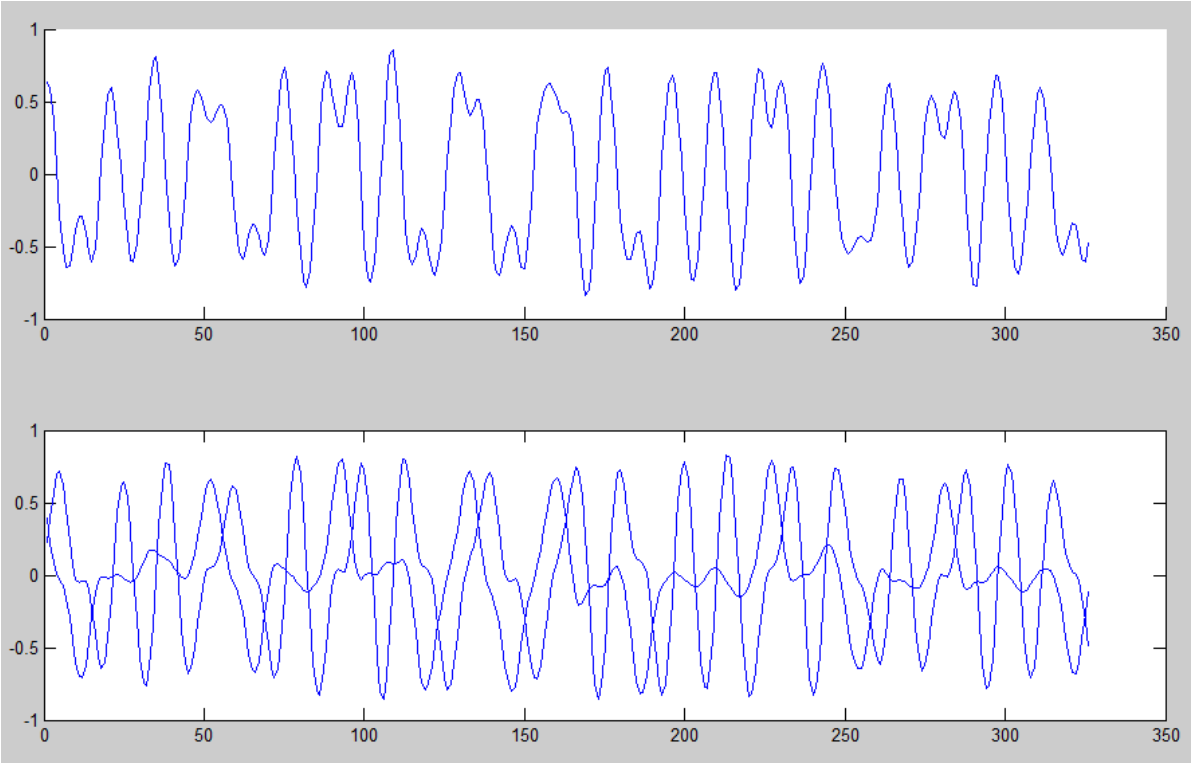

Figura 4.32: Señales pulsos y cruces obtenidas a partir del RBDS

Pasamos a identificar los máximos y mínimos locales de las señales "pulsos" y "cruces", para ello usamos la función "findpeaks" de Matlab, que en función de la altura mínima encuentra máximos y mínimos locales. En la figura 4.33 podemos ver el resultado de aplicar esta función sobre las dos señales. Con la función "stem" lo dibujamos, ya que con esta función podemos dibujar una secuencia discreta de datos, le fijamos la altura de pulsos y cruces a 1.

```
[¬, pulsos_mas] = findpeaks(pulsos_corto);
[¬, pulsos_menos] =findpeaks(-pulsos_corto);
[¬, cruces_mas] =findpeaks(cruces_corto, 'MinPeakHeight',.3);
[¬, cruces_menos] =findpeaks(-cruces_corto, 'MinPeakHeight',.3);

stem(pulsos_mas, ones(1,length(pulsos_mas)));
stem(pulsos_menos, -ones(1,length(pulsos_menos)));
stem(cruces_mas, ones(1,length(cruces_mas)));
stem(cruces_menos, -ones(1,length(cruces_menos)));
```

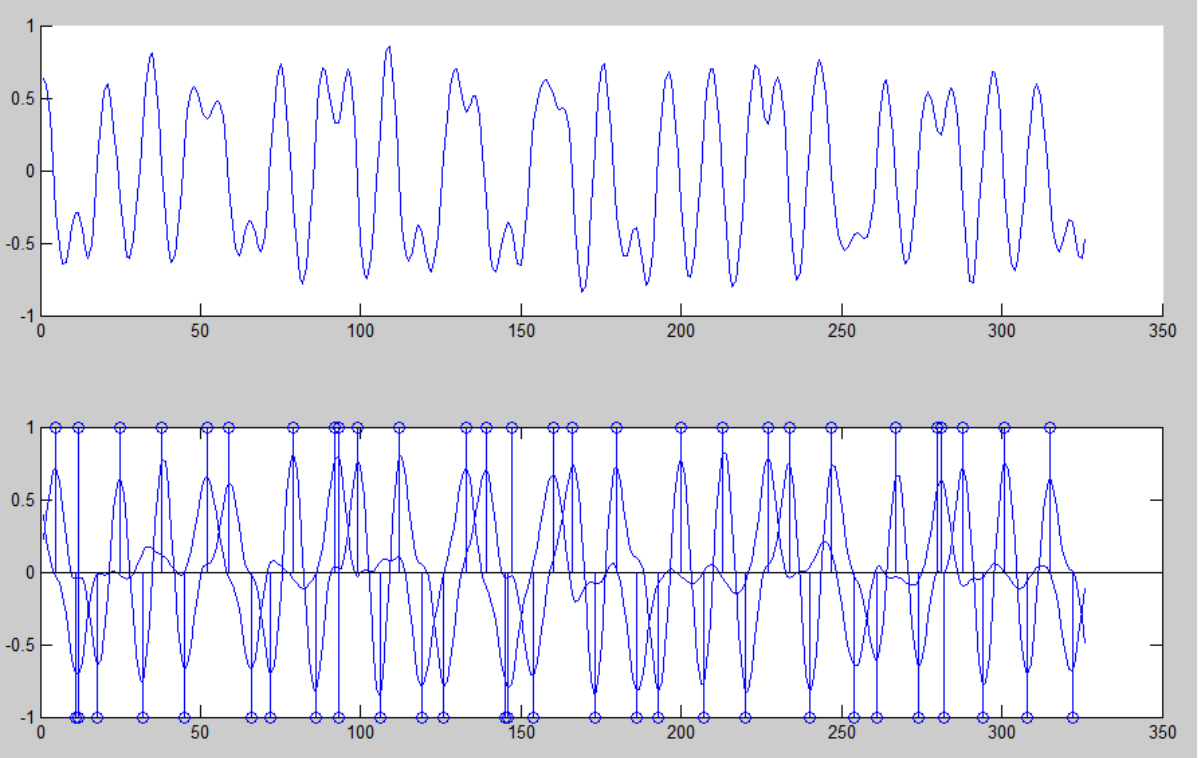

Figura 4.33: Representación de pulsos y cruces con la función “stem”

Por último, ordenamos con la función “sort”. A continuación, creamos la variable “status”, esta variable corresponde con el estado de la línea.

Se comienza evaluando el primer cruce, y obtenemos los estados desde ese primer cruce hasta que no queden más cruces, para cada uno se pone en el stream de salida, denominado “sal”, el número de pulsos presentes entre el cruce actual y el próximo, que serán “unos” o “-unos” según el valor de la variable “status”, a continuación se cambia el estado y se procesa el siguiente cruce. Podemos visualizarlo en la figura 4.34.

```
cruces = sort([cruces_mas, cruces_menos]);

status = (cruces(1) == cruces_mas(1)) -(cruces(1) == cruces_menos(1));

sal = [];
for i = 1:length(cruces)-1
    if (status == 1)
        sal = [sal, status*pulsos_mas(find(pulsos_mas>cruces(i)
         & pulsos_mas<cruces(i+1)))];
    else
        sal = [sal, status*pulsos_menos(find(pulsos_menos>cruces(i)
         & pulsos_menos<cruces(i+1)))];
    end;
    status = status*-1;
end;
```

```
subplot(211);
hold on;

stem(abs(sal), abs(sal)./sal);
stem(cruces_mas, 2*ones(1,length(cruces_mas)));
stem(cruces_menos, -2*ones(1,length(cruces_menos)));
```

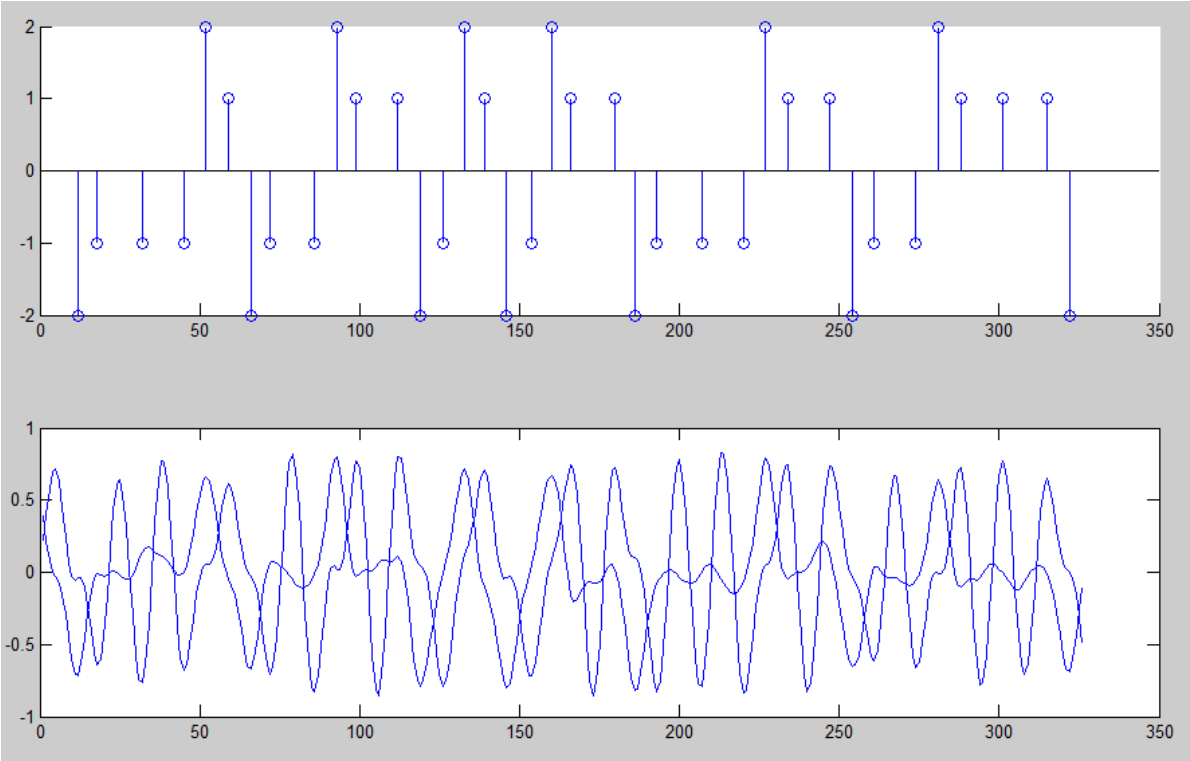

Figura 4.34: Stream de salida RBDS

Para obtener como salida un stream de bits, para una posterior decodificación y poder visualizar el texto, hacemos lo siguiente:

```
status = (cruces(1) == cruces_mas(1));

streamBits = [];
for i=1:length(nPulsos)
    streamBits = [streamBits, status*ones(1, nPulsos(i))];
    status=rem(status+1, 2);
end;

streamBits = double(xor(streamBits(1:end-1), streamBits(2:end))');
```

Creamos un archivo de texto con el comando "dlmwrite" en formato .txt, donde se verán los ceros y unos que representan los estados.

```
dlmwrite('salidaRDBS.txt',streamBits(1:end))
```

Capítulo 5

Captura de FM y tratamiento de señal capturada

5.1. Comprobación del dispositivo

Comenzamos haciendo un test en la consola de comandos para verificar que el dispositivo es reconocido correctamente por el PC:

```
rtl_test -t
```

Nos debe salir algo parecido a:

```
Found 1 device(s):
0: ezcap USB 2.0 DVB-T/DAB/FM dongle
Using device 0: ezcap USB 2.0 DVB-T/DAB/FM dongle
Found Elonics E4000 tuner
Supported gain values (18): -1.0 1.5 4.0 6.5 9.0 11.5 14.0 16.5 19.0
21.5 24.0 29.0 34.0 42.0 43.0 45.0 47.0 49.0
Benchmarking E4000 PLL...
(E4K) PLL not locked for 51000000 Hz!
(E4K) PLL not locked for 2227000000 Hz!
(E4K) PLL not locked for 1114000000 Hz!
(E4K) PLL not locked for 1241000000 Hz!
E4K range: 52 to 2226 MHz E4K L-band gap: 1114 to 1241 MHz
```

5.2. Captura de audio

Hay varias formas de capturar audio para su posterior manipulación con el octave/-matlab*, nosotros optaremos por usar el ejecutable "rtl_sdr"para capturar datos y guardarlos en un archivo, y luego leer ese archivo con el octave/matlab.

Para sintonizar una emisora en particular (M80Radio), con una tasa de muestreo de 2.5MHz, usamos el siguiente comando:

```
rtl_sdr -n 25000000 -s 2500000 -f 100100000 capture.bin
```

Debemos tener en cuenta que el dispositivo tiene una tasa de muestreo mínima de 900 Ksps (miles de muestras por segundo) y una máxima de 3200 Ksps (3.2 Msps).

Para detener la captura en cualquier momento, tenemos que aplicar CRTL+C, con una tasa de muestreo tan elevada, nuestro archivo "capturaFM.bin" crecerá muy rápidamente de tamaño, así que debemos capturar tan solo unos pocos segundos, o bien bajarle la tasa de muestreo.

Para capturar el tiempo deseado, basta con dividir el número de muestras totales entre la tasa de muestreo.

5.3. Tratamiento de señal FM capturada en Matlab

Para cargar el archivo en Matlab, en primer lugar tenemos que crear un "script" con la siguiente función:

```
1  captura = 'M80prueba5seg.dat';
2
3  limiteTam = 10e6;
4
5  fid = fopen(captura,'rb');
6
7  if limiteTam > 0
8    d = fread(fid, limiteTam, 'uint8=>double');
9  else
10   d = fread(fid, Inf, 'uint8=>double');
11 end;
12 fclose(fid);
13
14 if (size(d:1) ≠ 1)
15   d = transpose(d);
16 end;
```

*Información de como capturar muestras IQ de nuestro dispositivo y procesado en Matlab en la referencia [14] de la bibliografía.

Podemos comprobar la duración del audio dividiendo el número de muestras capturadas entre la tasa de muestreo:

```
size(d)

ans =

     10000000           1
```

Se corresponde con el límite de tamaño fijado por nuestra variable "limiteTam"

5.3.1. Funciones de interés

El Matlab nos ofrece las opciones de poder visualizar el diagrama de Bode (Figura 5.1) de la señal y el espectrograma (Figura 5.2), usando las funciones** "freqz" y "spectrogram".

```
y=loadFile('M80prueba5seg.dat');
freqz(y(1:5000),1,[-4E6:.01E6:4E6],2.5E6);
set(gcf,'color','white');
```

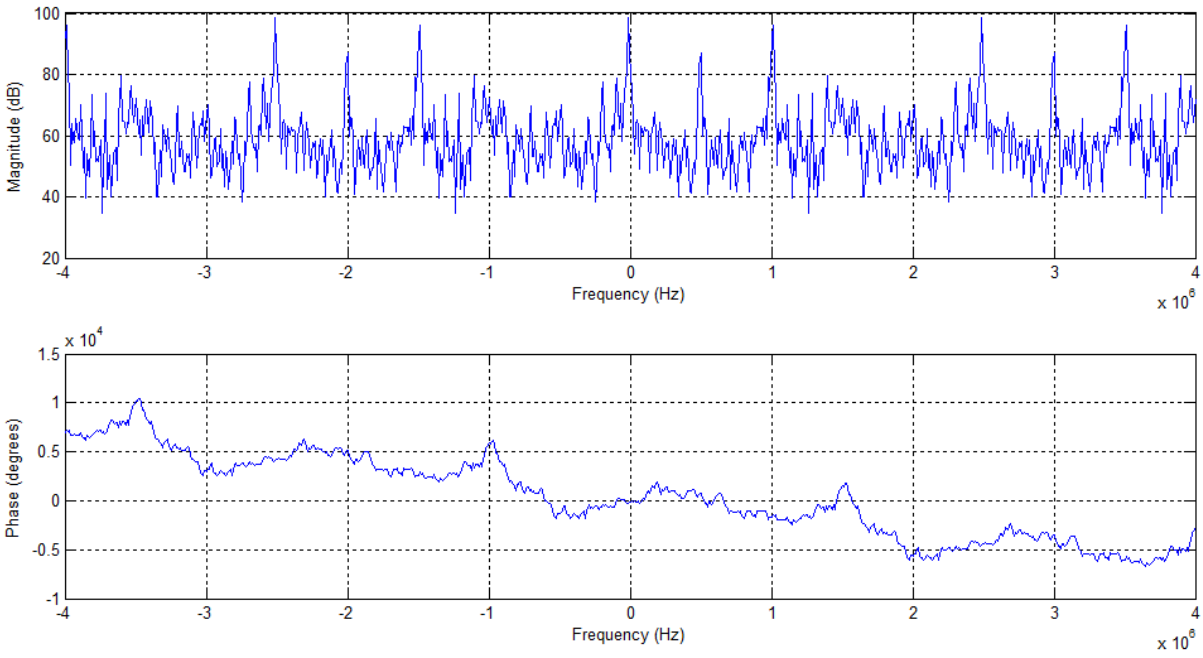

Figura 5.1: Diagrama de Bode captura FM

```
spectrogram(y,200000,1500,[-1.25E6:.02E6:1.25E6],2.5E6,'yaxis');
set(gcf,'color','white');
```

** Ayuda en funciones de Matlab con el comando "help + función" o en la página de ayuda de Mathworks en la cita [15] de la bibliografía

Al tener una tasa de muestreo de 2.5MHz, podemos ver desde nuestra frecuencia central (100.1MHz) hasta 1.25MHz por encima y por debajo. Podemos apreciar que en

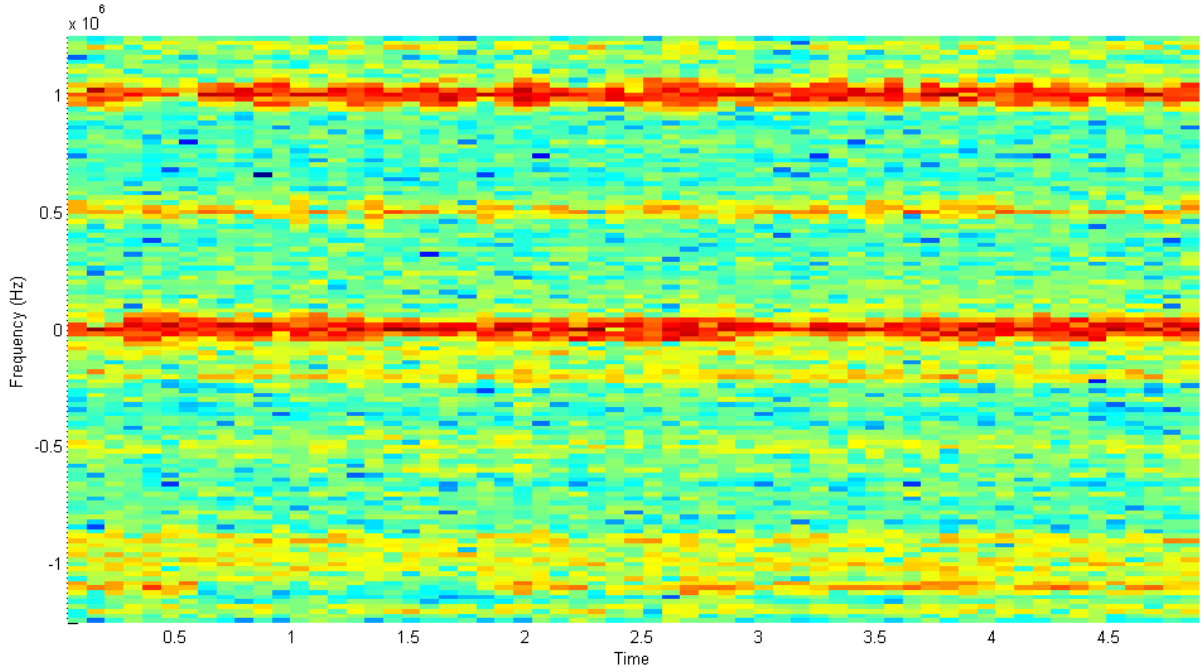

Figura 5.2: Espectrograma centrado en 100.1MHz

la frecuencia central 100.1 MHz se encuentra la emisora correspondiente con M80Radio, y podemos visualizar también la emisora 101.1 correspondiente a la CadenaSer, así como la 100.6 con una señal más débil.

5.3.2. Demodulación FM

Para demodular la señal comenzamos por decimar la señal, dado que comenzamos con tasas de megamuestras por segundo cuando cada emisora FM solo requerirá frecuencias de muestreo de unos 200kHz, usamos el comando "decimate", que nos permite hacer lo siguiente:

- Reduce la velocidad de muestreo original de una captura a una tasa más baja.

- Hace un filtrado paso-bajo la señal de entrada para proteger contra aliasing y luego muestrea el resultado.

En nuestro caso usaremos un filtro FIR que es un filtro de respuesta de impulso finito (Finite Impulse Response). Reducimos la tasa de muestreo en un factor 8, por lo que la nueva tasa de muestreo pasa a ser de 312.5KHz. En la figura 5.4 podemos ver el diagrama de Bode de la señal tras el decimado.

```
1 d = decimate(y,8,'fir');
2 freqz(d(1:5000),1,[-4E6:.01E6:4E6],2.5E6);
```

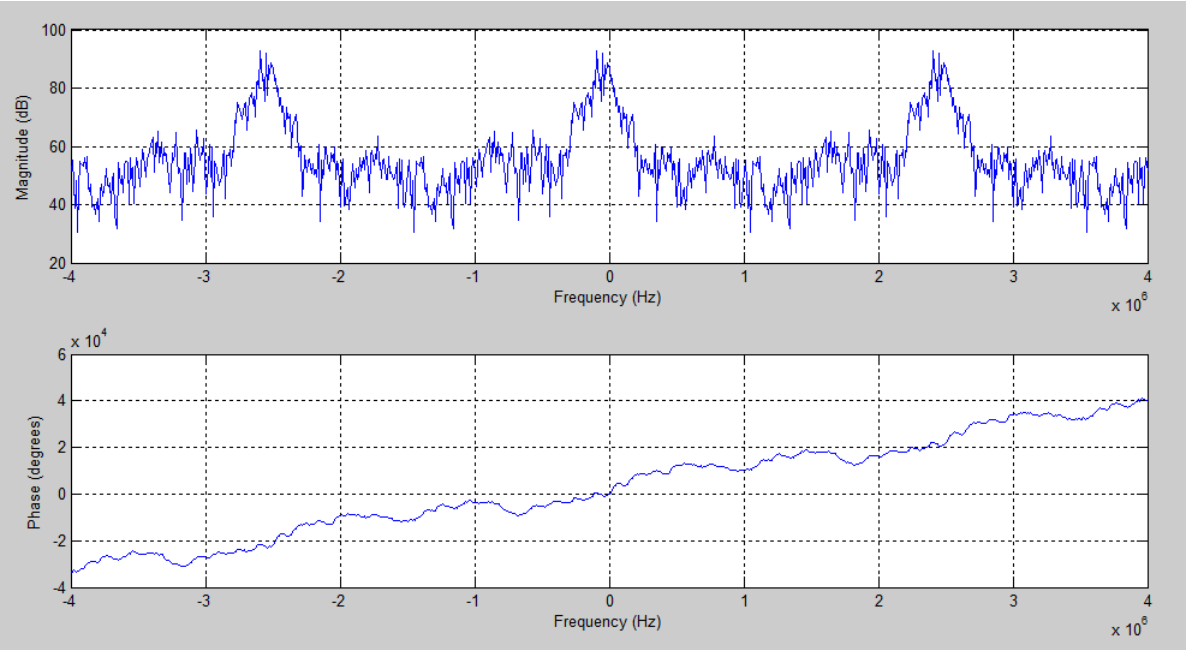

Figura 5.3: Bode de la señal con Tasa de Muestreo de 312.5KHz

Obtenememos las señales I y Q, y las sumamos para obtener la fase:

```
I_fm = d(1:2:end);
Q_fm = d(2:2:end);
R = I_fm+j*Q_fm;
dem = angle(R(1:end-1).*conj(R(2:end)));
clear d I_fm Q_fm R;
```

Usamos la función "FFT" de Matlab que nos hace la Transformada Rápida de Fourier (Fast Fourier Transform) para visualizar el espectro de la señal demodulada, que se puede ver en la figura 5.4

```
%quitamos las primeras muestras, no se habia puesto en marcha el sistema
dem = dem(5000:end);
fs = fs/2;

anchoFFT = 2^12;
f = ([1:anchoFFT]-anchoFFT/2)/anchoFFT*fs;
plot(f, fftshift(abs(fft(dem(floor(length(dem)/2):
floor(length(dem)/2)+anchoFFT-1)))));
axis([0 60e3 0 500]);
```

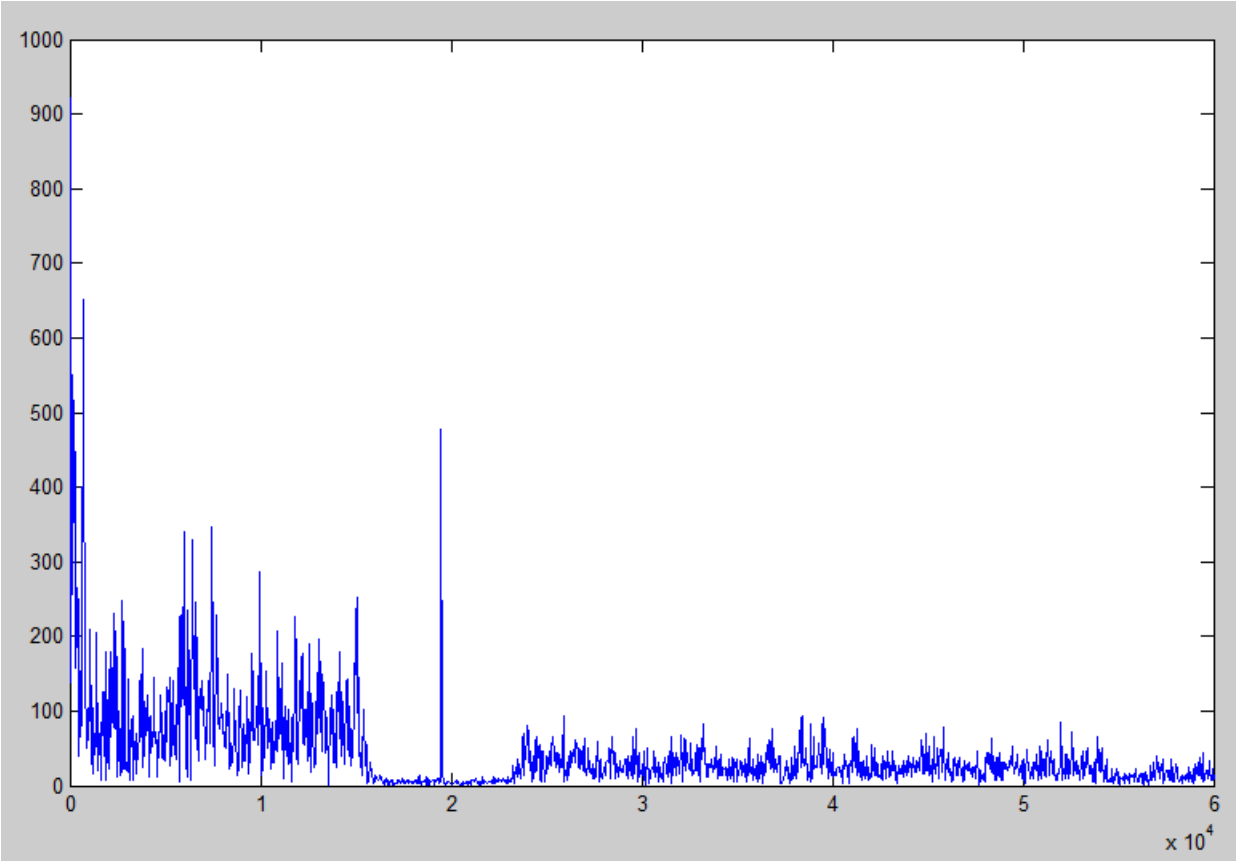

Figura 5.4: Espectro señal demodulada

De aquí en adelante, todos los pasos son los mismos que en el "Capítulo 4: Modulación y demodulación FM Digital" hasta llegar al audio mono en primer lugar, luego el audio estéreo, y para terminar el RBDS. Utilizamos las mismas técnicas de demodulación planteadas tanto para los audios como para los datos RBDS.

Capítulo 6

GNU Radio Companion

6.1. Introducción a GRC

GNU Radio Companion* (GRC) es una interfaz gráfica de usuario que nos permite construir diagramas de flujo en nuestra GNU Radio.

Es un lenguaje de programación visual de código libre para el procesamiento de señales utilizando las librerías de GNU Radio. Su entorno facilita la construcción de aplicaciones sin saber programar en Python o C++ dado que GRC produce justamente el código Python correspondiente a la aplicación construida gráficamente.

6.1.1. Características

- GRC está incluído con las fuentes de GNURadio. Si todas las dependencias se cumplen, GRC se instalará con GNURadio.

- Integración con el escritorio, se puede integrar completamente en un entorno de escritorio.

- Generación de código. GRC utiliza plantillas Cheetah para generar el código fuente de Phyton para el diagrama de flujo. También puede generar código fuente para gráficos WX GUI y que no sean GUI, así como bloques jerárquicos.

- Documentación. GRC puede extraer documentación para los bloques de GNURadio directamente de los archivos XML generados por doxygen.

- Variables. En esta versión tenemos bloques de variables, que se muestran en el diagrama de flujo y actúan como cualquier otro bloque, con excepción de que no tienen puertos de Entrada/Salida. Al bloque variable se le asigna un identificador único(Nombre de Variable) a un valor en particular. Además, tiene varios bloques gráficos de variables que permiten la creación de grafos de flujo WX GUI utilizando controles deslizantes, cuadros de texto, botones, menús desplegables y botones de radio.

*Página oficial de GNU Radio Companion, donde detallan las características de su programa, en la referencia [16] de la bibliografía

- Definiciones de Bloques. Cada bloque de GRC tiene su correspondiente archivo XML que contiene los parámetros, puertos de E/S, y una plantilla para la generación de código. El ID o clave de identificación de cada archivo XML coincide exactamente con el nombre del bloque de GNURadio para asegurar una futura portabilidad. GRC valida todas las definiciones de bloques después de la ejecución, y terminará con error si fallan alguna definición de la validación.

- Formato de archivo. Como las variables y definiciones de bloque han cambiado, la estructura interna de los archivos de grafo de flujo guardados también cambian. GRC puede convertir automáticamente los archivos guardados gráfico de flujo mayores para el nuevo formato.

- Manipulación de bloques. Los bloques tienen una opción de Activado/Desactivado. Por defecto, un bloque está activado. Cuando lo desactivamos, ese bloque se pone de color gris en el grafo de flujo y será ignorado por el validador y por el generador de código fuente. Además, los bloques se pueden cortar, copiar y pegar dentro de un mismo gráfico e insertarlos en otros gráficos.

- Bloques jerárquicos. GRC puede crear bloques jerárquicos con los bloques que vienen incorporados.

6.1.2. Requisitos de GRC

Todos los requisitos para poder ejecutar "GNU Radio Companion" están en el administrador de paquetes de nuestra versión de linux:

- **Python 2.5** (o superior)

```
http://www.python.org/download
```

- **Python-LXML 2.0** (o superior)

```
http://codespeak.net/lxml/installation.html
```

- **Cheetah Template Engine 2.0** (o superior)

```
http://www.cheetahtemplate.org/download.html
```

- **Python-GTK 2.10** (o superior))

```
http://www.pygtk.org/downloads.html
```

6.2. Aplicación simple: Analizador de espectro

Una de las aplicaciones[**] más simples, para familiarizarnos con el entorno de "GNU Radio Companion" es un analizador de espectro.

Explicaremos paso a paso el proceso:

Ejecutamos el programa GRC desde la consola

```
gnuradio-companion
```

La interfaz del programa se reparte en tres paneles:

- El área de desarrollo (Área principal): aquí crearemos el grafo de flujo

- Panel de mensajes (Panel de abajo): proporciona mensajes de registro y depuración

- Bloques (panel derecho): enumera los diferentes bloques de desarrollo que podemos usar el gráfico y aplicación

Lo primero que definimos en nuestro programa es la fuente para nuestra aplicación, como tenemos un RTL-SDR, seleccionamos esa opción. En la figura 6.1 se puede ver el aspecto que presenta la interfaz de trabajo de GRC.

Vamos a la pestaña ubicada en bloques llamada "Sources" y seleccionamos "RTL2832 Source" y la arrastramos al área de trabajo.

[**] Cómo crear programa básicos y utilización de los bloques más comunes en la cita [17] y [18] de la bibliografía

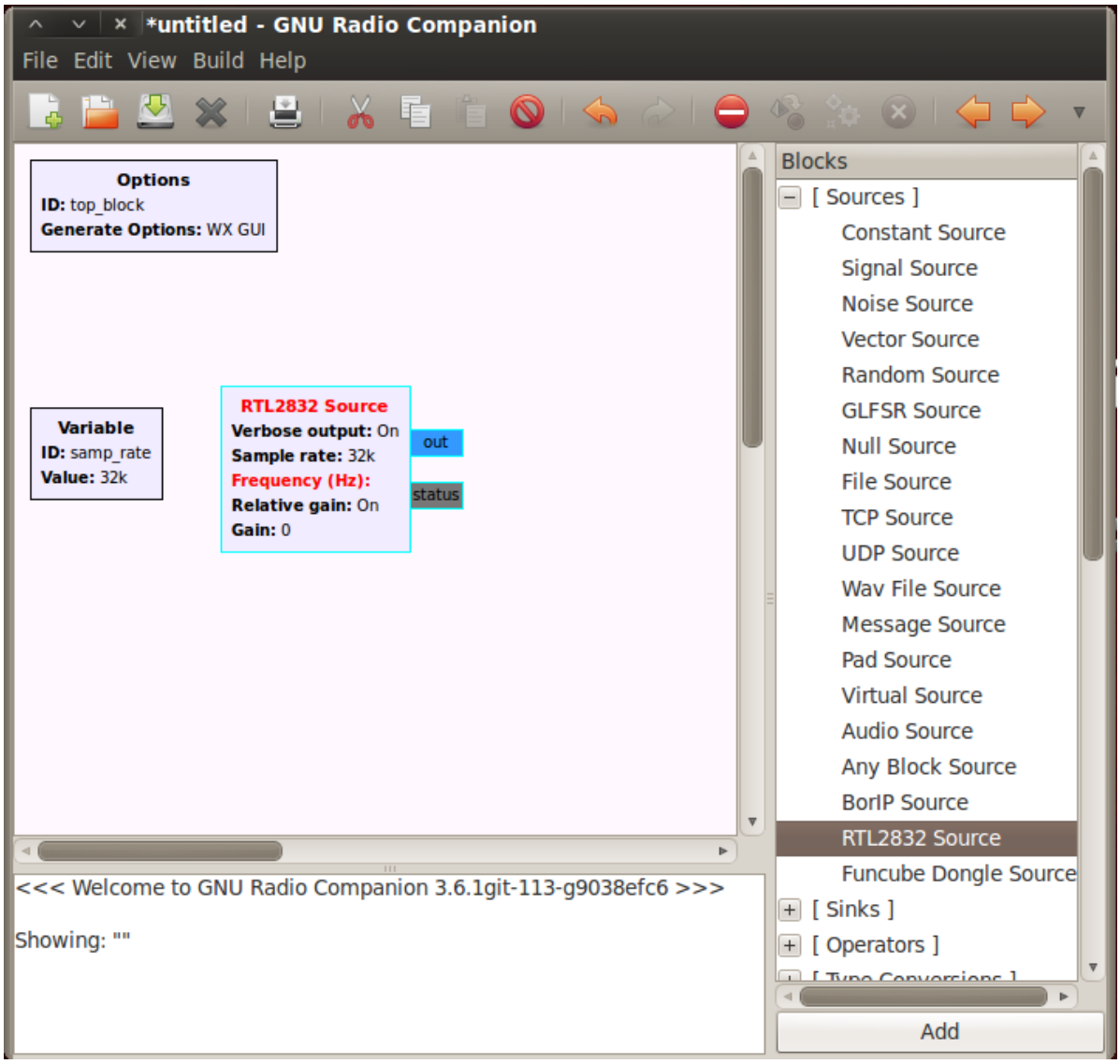

Figura 6.1: Interfaz de trabajo GRC

A continuación, tenemos que definir lo que vamos a hacer con nuestra "Source", es decir, con nuestro RTL-SDR. Como queremos crear un analizador de espectro, lo que necesitamos es el bloque "FFT sink", ya que nos muestra el espectro en tiempo real.

En la pestaña "WX GUI Widgets" seleccionamos la opción "WX GUI FFT Sink" y lo arrastramos hasta el panel principal. En la Figura 6.2 podemos ver de dónde se obtiene el analizador de espectro, así como su aspecto en el área de trabajo. Además, aparecen todos los parámetros que pueden ser modificados en su uso como por ejemplo la tasa de muestreo, su tamaño, o la resolución de los ejes x e y entre otras.

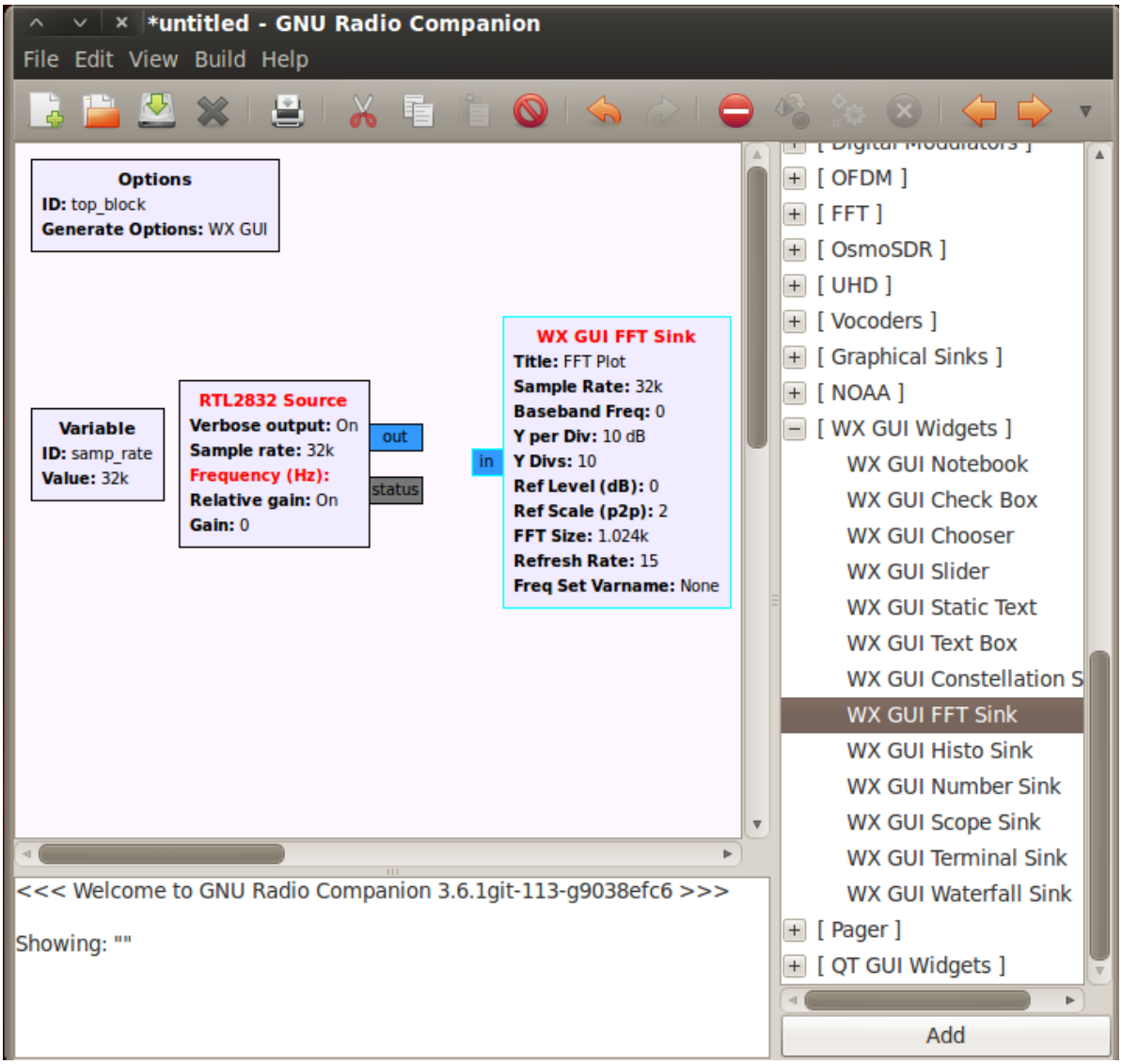

Figura 6.2: Herramienta de analizador de espectro

Para conectar nuestro dispositivo al analizador de espectro, debemos hacer click en “Out” en el dispositivo, y luego en “In” en el analizador de espectro. Con esto conseguiremos que se unan, el aspecto final de la unión se puede ver en la figura 6.3. Se pueden unir varias salidas a diferentes entradas, por ejemplo podemos unir la salida de nuestro RTL-SDR a un “Scope Sink” para ver el dibujo de la señal, o a un “File Sink”, para obtener en fichero las muestras I y Q capturadas por nuestro dispositivo.

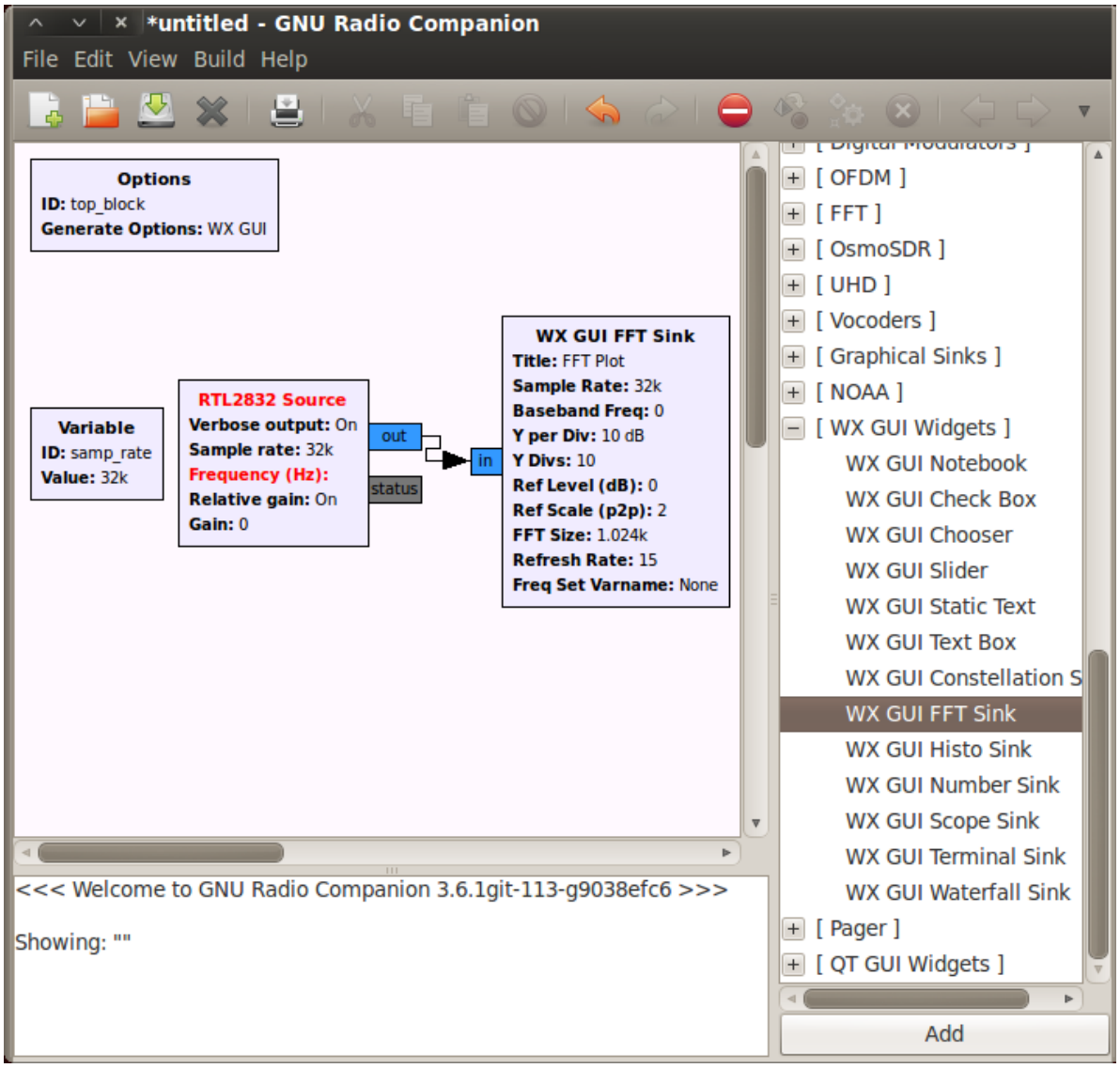

Figura 6.3: Unión entre bloques

Para poder compilar y/o ejecutar un programa de GNU Radio Companion, no debe haber ningún bloque marcado de color rojo. Si nos fijamos, aparece un atributo del bloque "RTL2832 Source" aparece en rojo, correspondiente a la frecuencia. Esto quiere decir que existe un error, en nuestro caso se debe a que la frecuencia está indefinida. Estos errores suceden porque no están definidos los parámetros, o bien porque son parámetros que nuestro dispositivo no puede soportar, por ejemplo una tasa de muestreo muy elevada. Además, debemos tener cuidado con el tipo de variable que usamos y no mezclar bloques que usen enteros con bloques que usen variables decimales (float) o complejas (complex).

Para solucionar nuestro error en particular, hacemos doble "click" en el bloque y establecemos una frecuencia central, en nuestro caso hemos fijado la frecuencia 100.3MHz, con notación para el bloque 1003e5, en la Figura 6.4 podemos ver la ventana emergente que aparece cuando pulsamos con el ratón encima de un bloque, en nuestro caso el "RTL2832 Source".

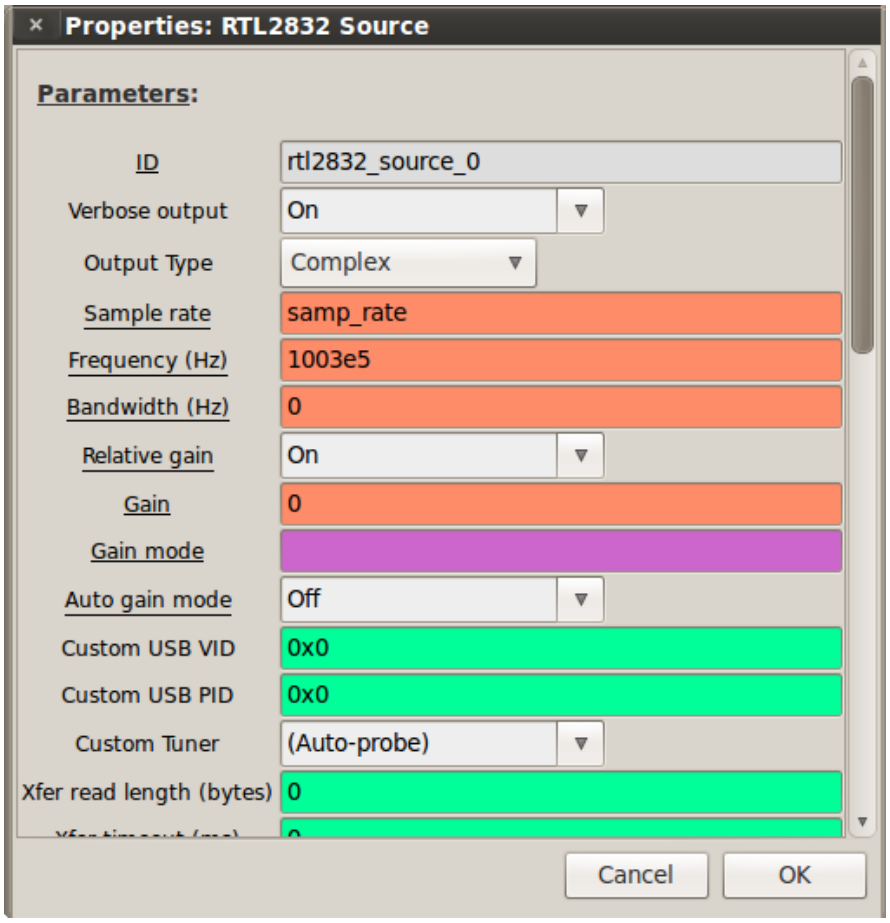

Figura 6.4: Pestaña de propiedades RTL2832U

Y con este paso terminamos este sencillo ejemplo, para generar todo lo necesario debemos ir a "Build" y luego seleccionar la opción "Generate". Se nos preguntará si queremos guardar el archivo, lo podemos ver en la figura 6.5.

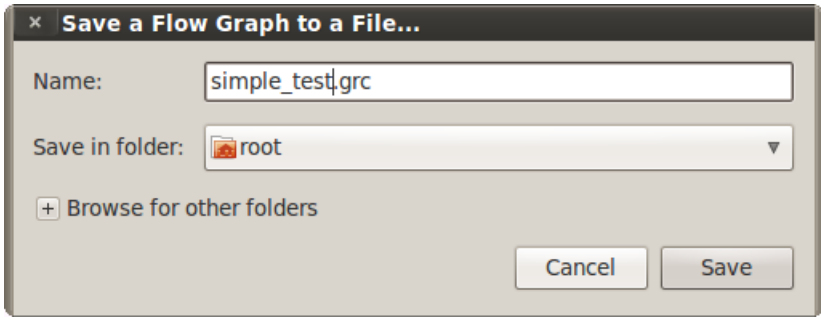

Figura 6.5: Guardar grafo de flujo

Para visualizar la señal en tiempo real, tenemos que ir a "Build" y luego seleccionar "Execute", se nos debe abrir una pestaña con la señal, como se muestra en la figura 6.6

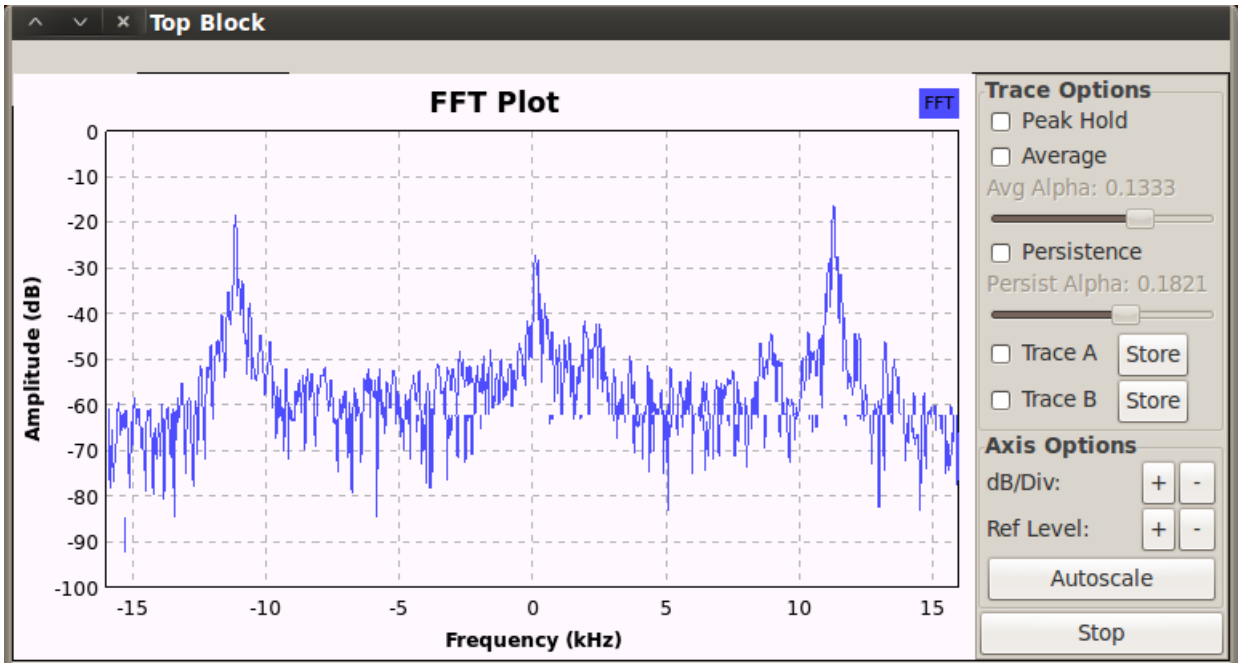

Figura 6.6: Visualización del espectro

Si nuestro equipo no es lo suficientemente potente, puede que se la señal no se refresque correctamente, o que la ventana no responda correctamente. Las opciones del trazo de la señal disponibles son las siguientes:

"Average", donde la señal se ve más limpia(Figura 6.7)

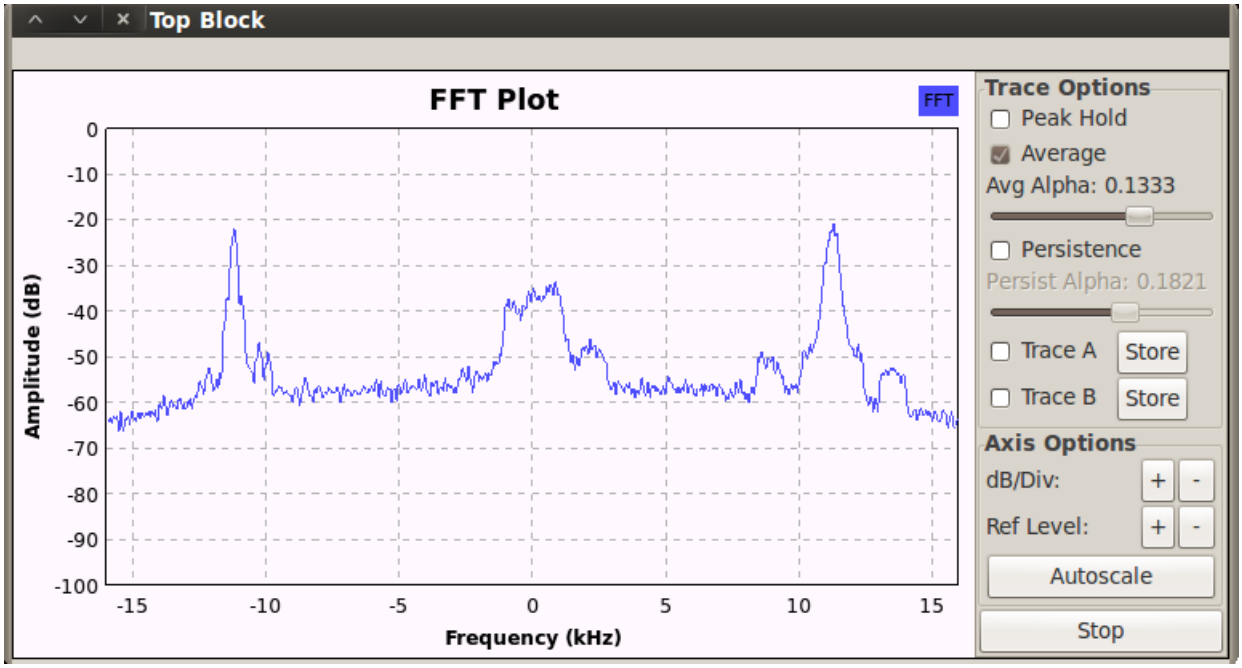

Figura 6.7: Visualización del espectro en su promedio (Average)

"Peak Hold" (Figura 6.8), que sirve para mantener los picos de energía del espectro de la señal, puede ser útil si la señal cambia rápidamente, o si queremos saber en qué frecuencias se está transmitiendo.

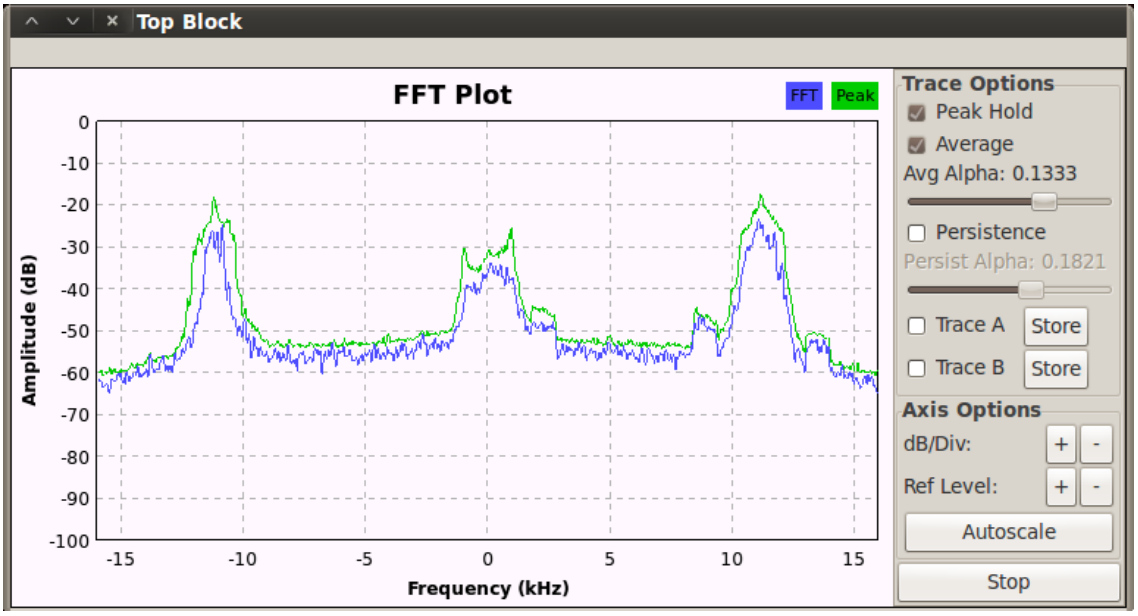

Figura 6.8: Visualización del espectro promedio, manteniendo los máximos de energía

Para mantener estas opciones siempre activadas, vamos a las propiedades del analizador de espectro (Figura 6.9) y las ponemos en "On"

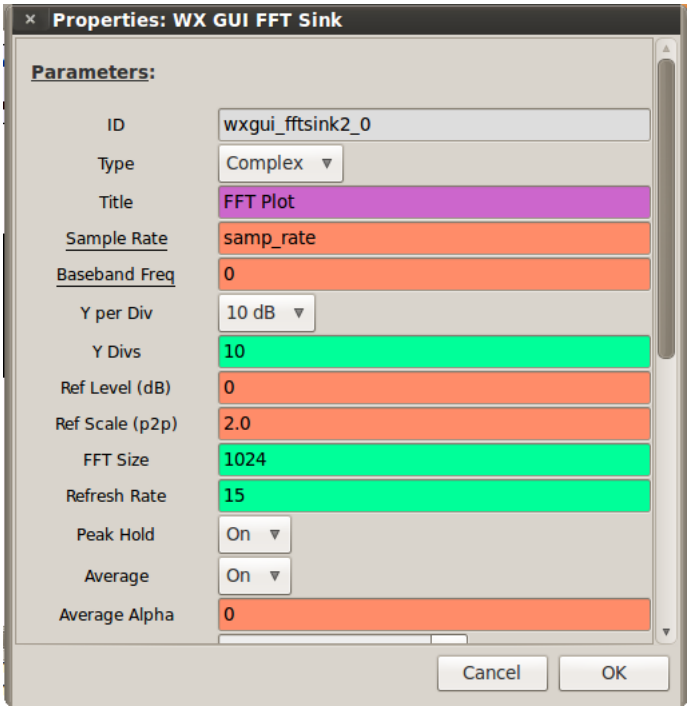

Figura 6.9: Propiedades analizador espectro

Capítulo 7

Aplicaciones Finales

7.1. Demodulador FM monofónico

El primer proyecto que vamos a explicar, consiste en el demodulador digital monofónico, hemos tratado de seguir el mismo orden que en Matlab para indicar qué pasos del proceso en tiempo real corresponden con su equivalente en Matlab, referenciándolos en cada momento y detallando bloque a bloque el funcionamiento. En la figura 7.1 podemos ver el diagrama de bloques al completo de nuestro demodulador monofónico.

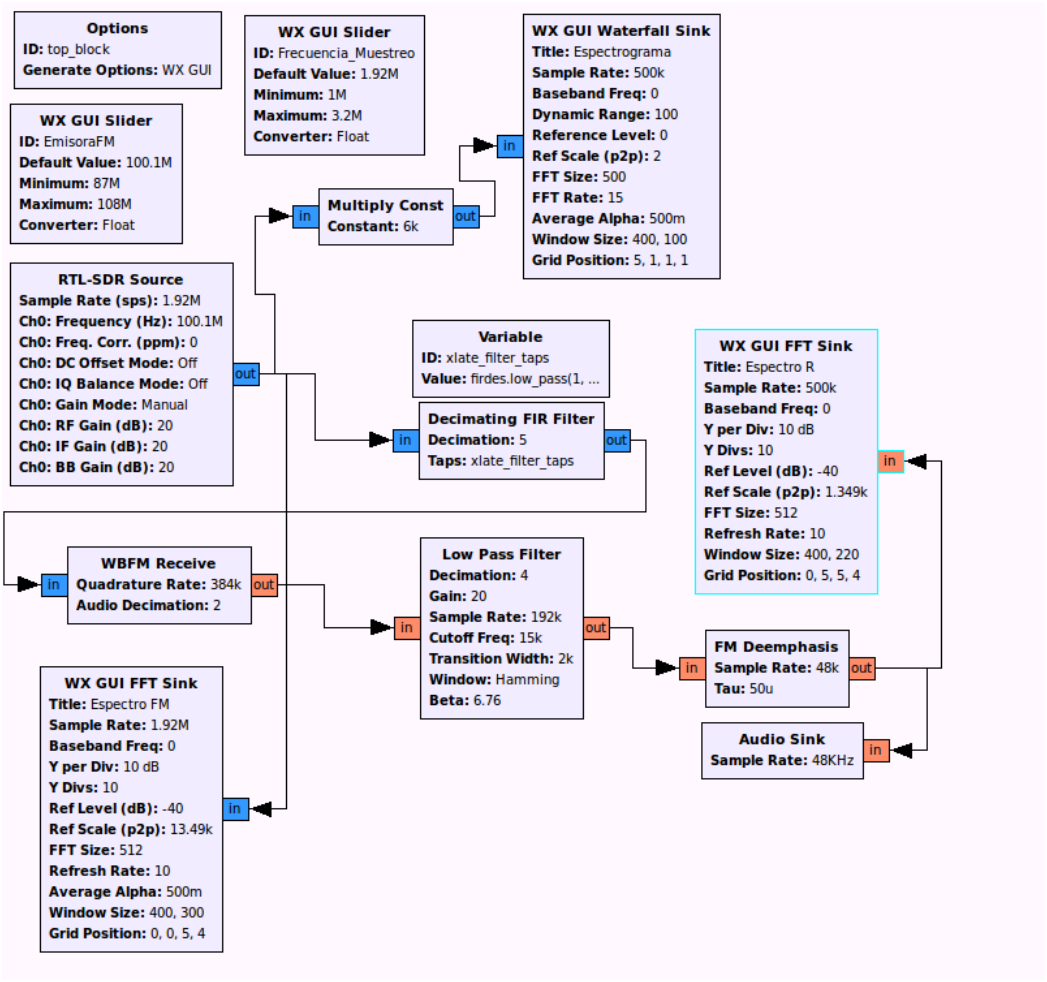

Figura 7.1: Diagrama de bloques demodulador FM mono

Variables

En este programa usaremos dos variables, con ellas podemos cambiarles el valor y automáticamente el programa se actualizará dependiendo de su valor, usamos las siguientes:

- FrecuenciaMuestreo: 1.92MHz
- EmisoraFM: 100.1MHz

Fuente RTL-SDR

Nos proporciona nuestra señal de entrada, establecemos su frecuencia de muestreo en 1.92MHz ya que es un valor bastante bueno para dividirla y obtener 48kHz de frecuencia de audio, por defecto aparece una variable llamada "samp_rate", que se asocia con la frecuencia de muestreo, le asignamos nuestra variable "FrecuenciaMuestreo" y le asignamos valor (1.92e6). Necesitamos establecer además la frecuencia de la emisora que deseamos recibir, le asignamos la variable EmisoraFM. Como recomendación es conveniente añadir todos las variables que vayamos a cambiar en un futuro como variables, ya que tenemos la posibilidad de editarlas durante la ejecución del proyecto.

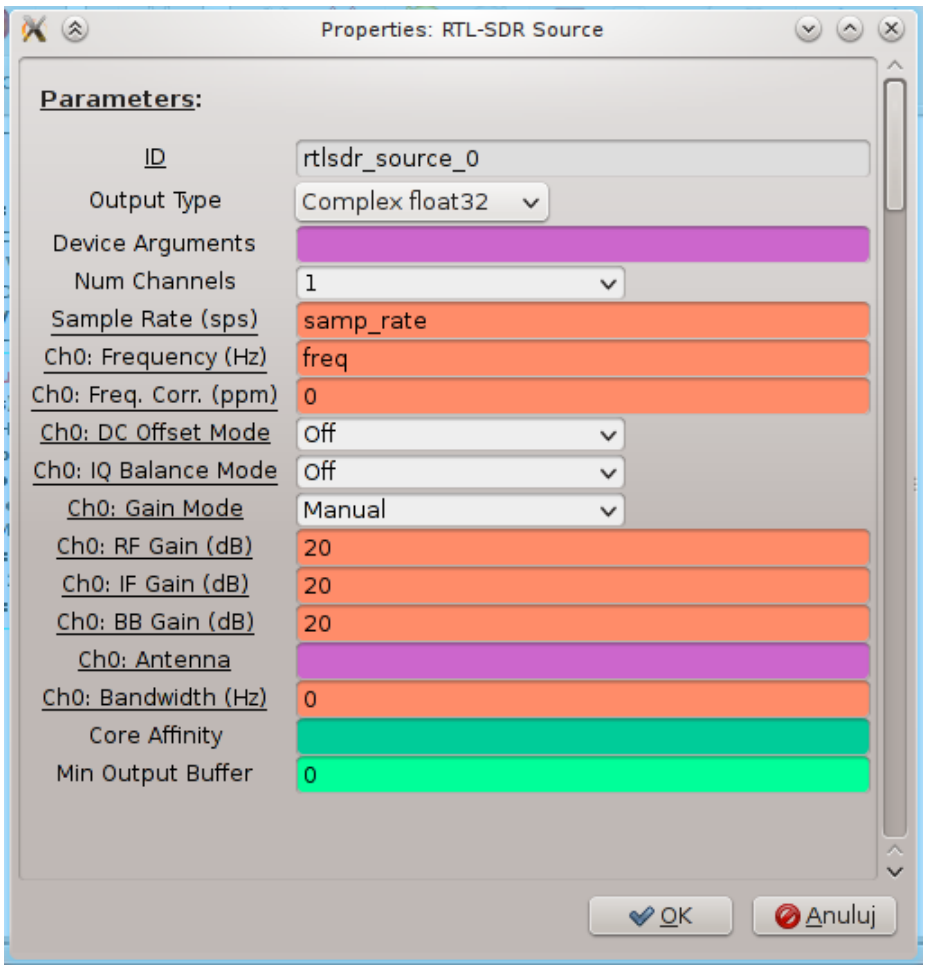

Figura 7.2: Propiedades señal capturada RTL-SDR

Decimating FIR Filter

En este bloque tenemos un filtro de respuesta al impulso finita, que además realiza el decimado. Con este filtro nos quedamos con la frecuencia deseada del FM, que llega hasta los 57kHz, además, reducimos la tasa de muestreo en un factor 5, por lo que pasa a ser de 384kHz.

Podemos ver cómo este bloque se asemeja en Matlab al comando "decimate", cuyos resultados podemos ver en la figura 4.11 su correspondiente código está en el "Código 2" de la página 43.

WBFM Receive

Llegamos al demodulador FM, usamos como parámetro de cuadratura, la tasa de muestreo de salida del filtro anterior, y el parámetro de decimado necesario, el cual hemos usado un factor 2 para obtener una tasa de muestreo de 192kHz. Además, este bloque hace la conversión de muestras enteras(int) a decimales (float).

Este bloque de código tiene su semejanza en Matlab con el "Código 1", donde pasamos las muestras obtenidas a "float32" está en la página 43.

Filtro Pasa Bajas

Usamos un filtro pasa bajas con una frecuencia de corte de 15kHz para obtener la señal monofónica. Además el mismo bloque permite la decimación, la cual hemos usado un factor 4 para dejar la tasa de muestreo final en 48kHz.

Podemos ver este mismo proceso en MatLab realizado en los "Códigos 5 y 6", en las páginas 52 y 53 respectivamente, en el que utilizamos un filtro Chebyshev para eliminar la portadora de 19kHz de la señal monofónica.

Además, como este filtro puede hacer la decimación en un mismo paso, también hace el equivalente del "Código 7" de Matlab, en la página 53

FM Deemphasis

El de-enfasis se utiliza en los receptores para eliminar el ruido en las altas frecuencias, explicamos este paso con más detalle en la siguiente sección, donde explicamos el demodulador FM estereofónico.

Salida de audio

La salida de audio "audio sink", permite obtener el audio de una señal, en nuestro caso sólo tendrá una entrada correspondiente a la señal monofónica (L+R), la tasa de muestreo final es de 48kHz.

Equivalente en Matlab en el "Código 8" en la página 54.

7.2. Demodulador FM estereofónico

En este programa, como ampliación del primer programa basado en el demodulador monofónico, y partiendo de las muestras IQ de nuestro dispositivo de bajo coste RTL2832U, obtendremos el audio estéreo, pasando por una serie de bloques, que explicaremos con más detalle a continuación. Mediante las salidas o "sinks" y el posicionamiento de los mismos, obtendremos por pantalla una serie de parámetros importantes tales como el espectro de la FM, un espectrograma y señales de audios L y R entre otras. El diagrama de bloques completo se puede ver en la figura 7.3

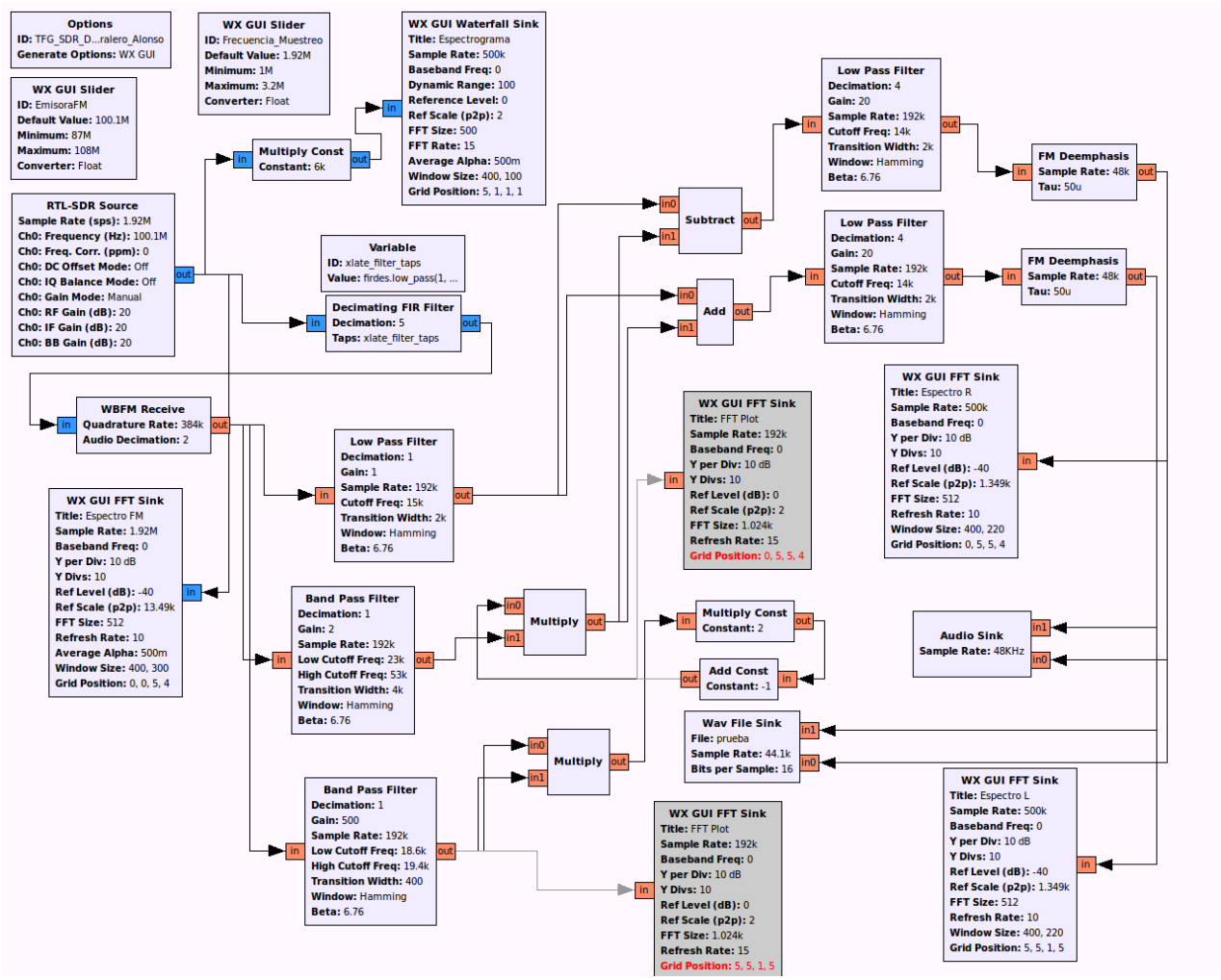

Figura 7.3: Diagrama de bloques de nuestro programa

Sliders

Los "Sliders" son variables* a las que le pasamos un parámetro mínimo, otro máximo, y uno por defecto, y cuando ejecutemos el programa podemos irlos variando para comprobar el funcionamiento ante el cambio de ese valor.

Hemos puesto dos variables con "sliders", la frecuencia de la emisora, muy importante, ya que queremos sintonizar cualquier frecuencia comercial, y lógicamente parámetros mínimos de 88MHz y máximo de 108MHz, y un parámetro por defecto 100.1MHz correspondiente con la emisora M80Radio.

La segunda variable corresponde con la tasa de muestreo, que realmente no debería poder cambiar, pero por razones didácticas se considera útil ponerla variable, para ver cómo el sonido de salida no es audible si no encaja con la frecuencia de la tarjeta de sonido. Lo ajustamos con la tasa de muestreo mínima y máxima de nuestro dispositivo "RTL2832U" y ponemos un mínimo de 1MSPS y máximo 3.2MSPS, con un valor por defecto de 1.92MSPS, cuyo valor es múltiplo de la frecuencia de salida de 44.1kHz.

Frequency Xlating FIR Filter/Decimating FIR Filter

La denominación "Xlating" quiere decir "Translating", por lo que tenemos un filtro de respuesta al impulso finita con traslación de frecuencia, además realiza selección de canales y decimado en un solo paso. Con este filtro nos quedamos con la frecuencia deseada del FM, que llega hasta los 57kHz con el RBDS, es decir, la frecuencia de la portadora de 19kHz multiplicada por tres. Para ello creamos la variable "xlate_filter_taps", en el que le pasamos la frecuencia de corte para que elimine toda la frecuencia superior.

Además, le hemos puesto un decimado de factor 5 para que reduzca la tasa de muestreo de 1.92MSPS a 384KSPS.

Como alternativa al filtro de traslación, podemos usar el filtro que usamos en el demodulador monofónico, el "Decimating FIR Filter" o filtro de respuesta al impulso finita con decimación, que funciona muy bien para filtrar las frecuencias no deseadas a partir de 60kHz y además hacer el decimado de la señal para bajarle el mustreo.

WBFM Receive

Llegamos al demodulador FM, en el cual le pasamos como parámetro de cuadratura la tasa de muestreo de salida del filtro anterior, y el parámetro de decimado necesario.

Las siglas "WBFM" significan "Wide Band Frequency Modulation" que quiere decir Modulación de Frecuencia de Banda Ancha, la usada en la FM comercial, aunque haya más demoduladores en GRC, debemos usar este ya que es el indicado para tratar con este tipo de señales.

*Variables y bloques de GRC explicados con detalle en la referencia [21] de la bibliografía

Filtros Pasa Bajas y Pasa Banda

El primer filtro pasa bajas, con frecuencia de corte de 15kHz, sirve para obtener la señal monofónica, es decir, la suma de los audios L y R, por lo que tenemos la señal L+R.

El filtro pasabanda, centrado en 38kHz, y con frecuencias de corte de 23kHz y 53kHz, sirve para obtener la señal diferencia de los audios L y R, la señal L-R.

Por último, el segundo filtro pasabanda sirve para recuperar la portadora de 19kHz, por lo que tenemos un filtro pasa bandas con sólo 800Hz de banda de paso, estando las frecuencias de corte en 18.6kHz y en 19.4kHz.

Multiplicador

Utilizamos el multiplicador para eliminar la portadora de la señal diferencia (L-R). Esto se consigue utilizando el tono piloto cuya frecuencia es de 19kHz, así el receptor puede sincronizarse con la señal resta, que está justo al doble de la frecuencia de la piloto (centrada en 38kHz) multiplicando el tono piloto por dos: $19 \cdot 2 = 38$.

La ecuación para obtener la señal de 38kHz a partir de la de 19kHz (tono piloto) es la misma que vimos en este mismo proceso en MatLab:

$$\cos 2\alpha = 2\cos^2 \alpha - 1 \tag{7.1}$$

Por tanto introducimos en el multiplicador dos veces la salida del pasabandas de la portadora, y una vez el pasabandas de la señal diferencia.

En el tratamiento de señal de Matlab este paso lo podemos ver en "Código 9" donde se obtiene la subportadora de 38kHz, está en la página 55.

Suma y Diferencia

Para obtener las señales de audio estéreo izquierda y derecha (L y R), simplemente basta con sumar y restar las dos señales obtenidas de los bloques anteriores.

Para el audio izquierdo, L, tenemos que sumar $(L + R) + (L - R) = 2L$. Para ello usamos el bloque suma llamado "Add"

Para el audio derecho, R, tenemos que restar $(L+R)-(L-R) = 2R$. Esto lo hacemos con el bloque llamado "Subtract"

El equivalente en Matlab del multiplicador y la suma/diferencia se puede ver en el "Código 4" de la página 49, donde se aplican las relaciones trigonométricas necesarias para conseguir las sub-portadoras.

Segundo Filtro

Utilizamos este filtro para decimar la señal en un factor 4, y así pasar finalmente de 192KSPS a la tasa de muestreo final de 48kHz, utilizada por nuestra tarjeta de sonido.

En el "Código 10" de Matlab podemos ver el equivalente de este bloque, realizando el decimado necesario para obtener una tasa de muestreo aceptada por nuestra tarjeta de sonido.

FM Deemphasis

En telecomunicaciones el énfasis[**] es la alteración intencionada de las características de amplitud con respecto a la frecuencia de la señal para reducir los efectos adversos de ruido en un sistema de comunicación.

El ruido aleatorio tiene una distribución espectral triangular en un sistema de FM, con el efecto de que el ruido se produce predominantemente en las frecuencias de audio más altas dentro de la banda base. Esto se puede compensar, en una medida limitada, al aumentar las frecuencias altas antes de la transmisión y la reducción de ellos en una cantidad correspondiente en el receptor.

La reducción de las altas frecuencias de audio en el receptor también reduce el ruido de alta frecuencia. Estos procesos de impulsar y luego reducir ciertas frecuencias son conocidos como pre-énfasis y de-énfasis, respectivamente.

La cantidad de pre-énfasis y de-énfasis utilizado se define por la constante de tiempo de un circuito simple filtro RC. En la mayor parte del mundo se utiliza 50 μs de constante de tiempo. En Corea del Sur y América, se utiliza 75 μs. Esto se aplica tanto en transmisiones mono como estéreo. Para estéreo, pre-énfasis se aplica a los canales izquierdo y derecho antes de multiplexación.

Fractional Interpolator

Para ajustar la tasa de muestreo en el caso de que tengamos una tarjeta de sonido de 44.1kHz, debemos utilizar este bloque para pasar de 48kHz a 44.1kHz, por tanto la operación que hay que usar es: $f_{audio} = \frac{48000}{44100} = 1{,}08844$, este proceso también se puede hacer para ajustar la frecuencia de entrada de 48kHz con cualquier otra que sea capaz de reproducir nuestra tarjeta de audio, basta con establecer la relación entre ellas tal y como hemos hecho.

[**]Énfasis explicado con detalle, así como el pre-énfasis y de-énfasis en las citas [22] y [23] de la bibliografía.

Salida de audio

Finalmente configuramos la salida de audio con dos entradas, correspondiente al audio estéreo, y fijamos la tasa de muestreo en 48kHz, en nuestro PC las tasas de muestreo válidas de salida son sólamente 48 y 44.1kHz.

Realizamos lo mismo con una salida de archivo en formato .wav para poder grabar el audio.

El equivalente a el "Audio Sink" de GRC en Matlab es el comando "sound", que podemos ver cómo funciona en el "Código 11" en la página 56.

Display

Como parte final del programa, se ha propuesto la siguiente disposición, en primer lugar tenemos el espectro de la señal capturada por el dispositivo de entrada (RTL2832U) así como su espectrograma (Waterfall Plot). Además, los audios izquierdo y derecho se han decidido dibujar también con la misma herramienta que usamos que con la señal capturada, el bloque "FFT Sink".

Estos "plots" presentan tres opciones de visualización, la cantidad de espectro promedio, la persistente, y la "peak hold", que registra en otro color el valor de pico del espectro. Además, podemos fijar la resolución del dibujo como si se tratase de un osciloscopio con la opción dB/div, así como ajustar el nivel de referencia, en nuestro caso está puesto en -40dB, así como el tamaño del eje que hemos puesto 100dB.

El ancho de banda del espectro de la señal capturada es de 2MHz, y el de las señales L y R es de 250kHz.

Con la opción "Grid Position", podemos ajustar cada gráfico en la zona deseada, en nuestro caso hemos puesto a la izquierda del display el espectro de la señal capturada, así como el espectrograma, y en el lado derecho de la pantalla, las gráficas de los audios L y R. En la parte inferior hemos puesto los "sliders" de la tasa de muestreo y de la frecuencia central de la emisora FM comercial.

En la imagen que mostramos a continuación podemos ver la disposición final de nuestro programa final.

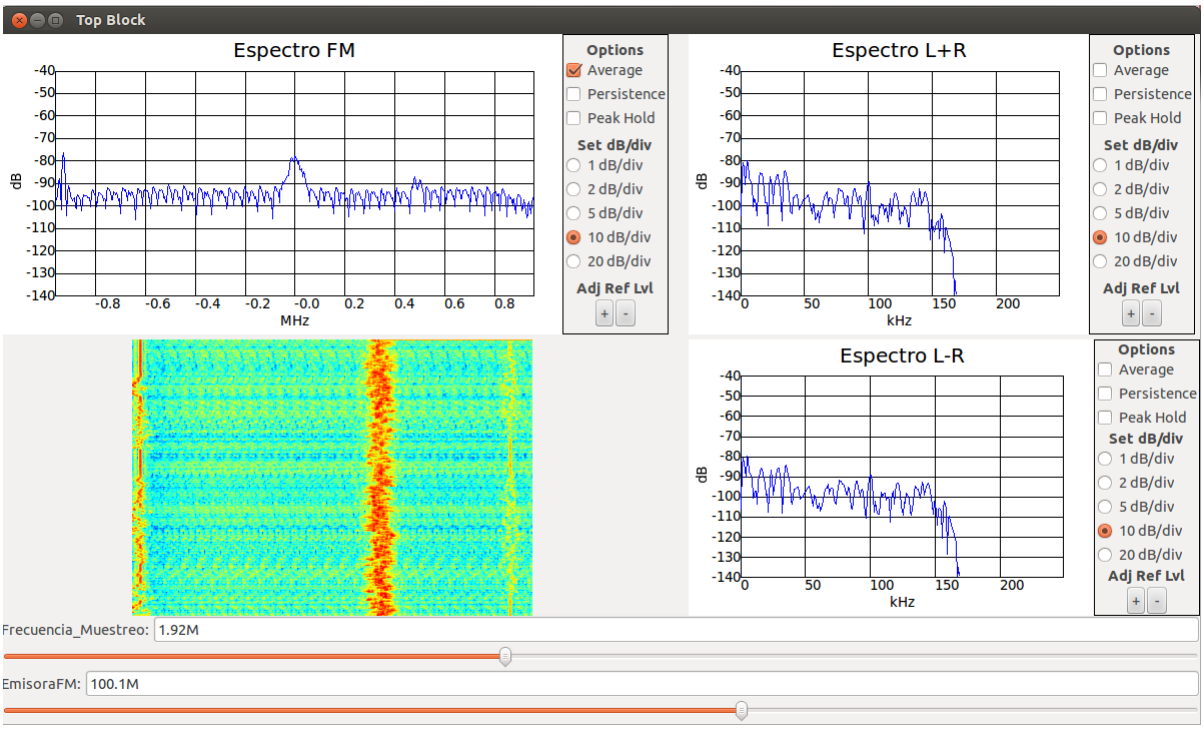

Figura 7.4: Señales que mostramos por pantalla

7.3. RBDS

Usando la el tono piloto de 19kHz, podemos extraer la señal RBDS, para ello en este caso usamos un filtro pasa-banda cuyas frecuencias de corte son de 54kHz y 60kHz, y está centrado en 57kHz, es decir, tres veces la frecuencia de la portadora. Podemos ver el diagrama de bloques al completo en la siguiente figura:

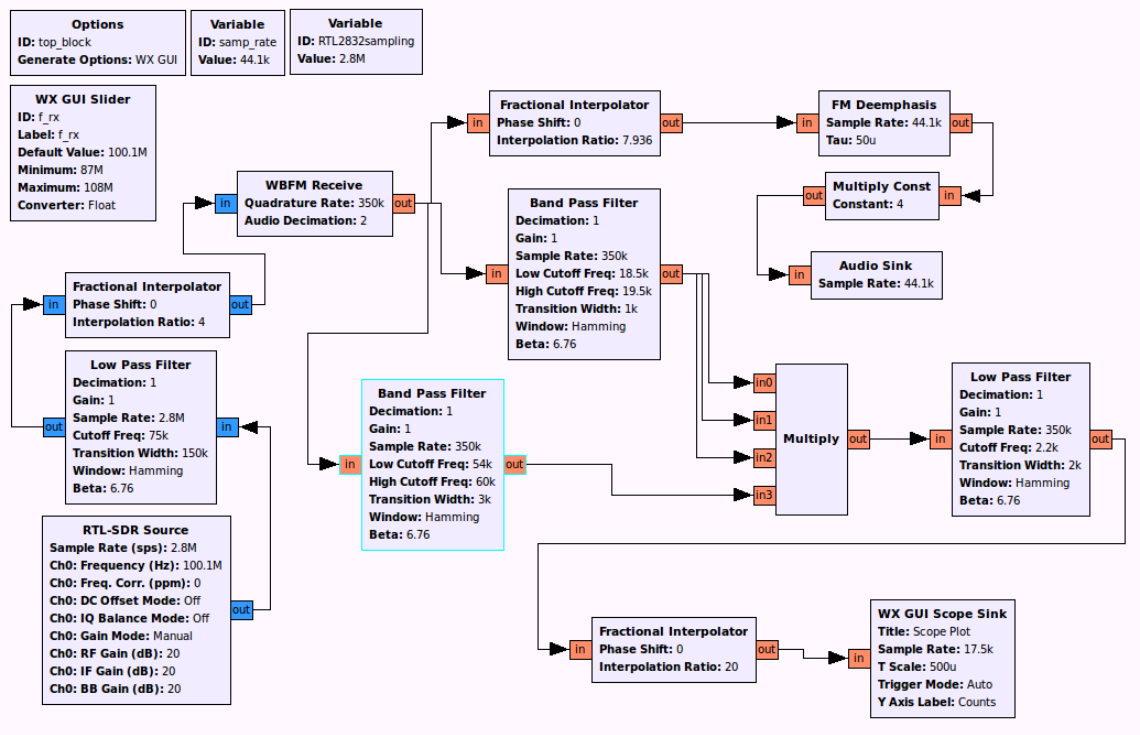

Figura 7.5: Bloques RBDS

Al igual que hacíamos con la señal L-R para eliminar la portadora de la señal, con el RBDS*** hacemos lo mismo, es decir, utilizamos un bloque "Multiply" o multiplicador, en el que introducimos tres veces el tono piloto, y en la cuarta entrada ponemos la señal filtrada donde se encuentra el RBDS.

Le pasamos un pasa bajas y le aplicamos un interpolador fraccionario para reducirle la tasa de muestreo hasta los 17.5kSPS, por tanto para bajar de 350kSPS a 17.5kSPS, el factor es de 20.

***Más información sobre RDS en GNU Radio Companion en la referencia [24] de la bibliografía

Para finalizar tenemos un “Scope Sink” en el que dibuja la señal RBDS, la cual podemos ver en la siguiente imagen:

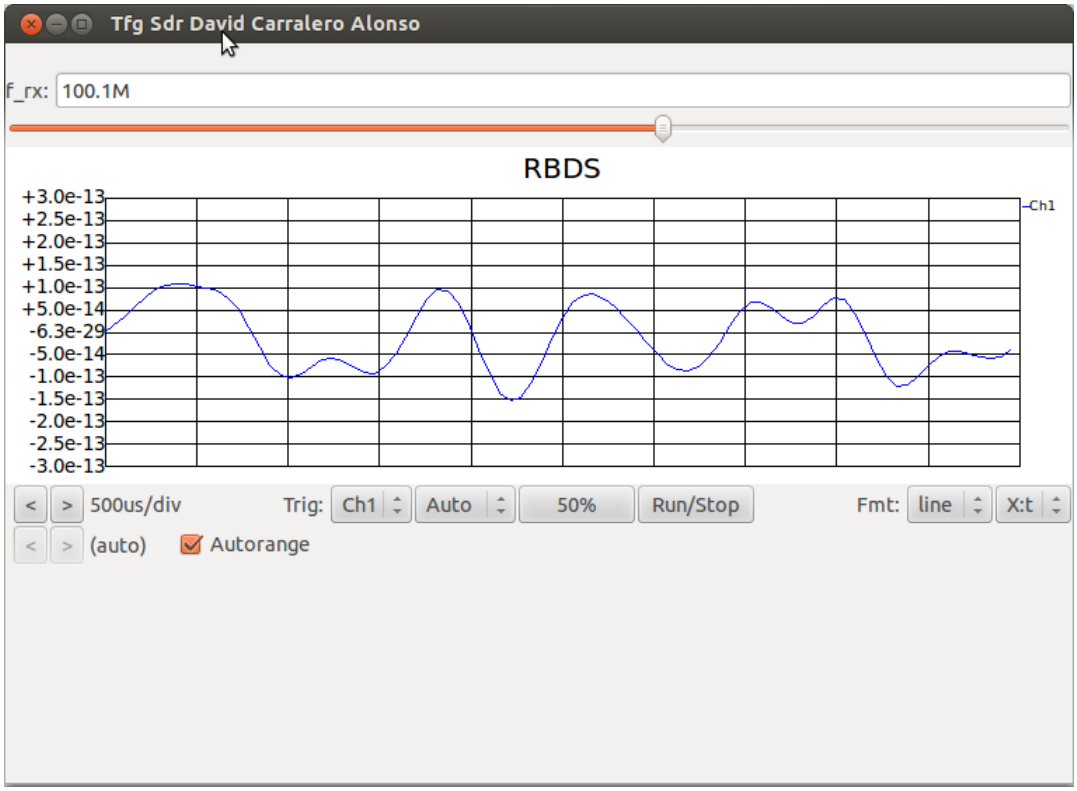

Figura 7.6: Dibujo señal RBDS

Capítulo 8

Conclusiones y Líneas Abiertas

En este capítulo se exponen las conclusiones a las que se llegaron a partir de la realización del proyecto. Además se explicarán qué posibles ampliaciones se pueden llevar a cabo en trabajos posteriores.

8.1. Conclusiones

Las Radios Definidas por Software son una excelente forma de explicar paso a paso el proceso de recepción de la señal hasta el paso final de recepción de audio, tanto mono como estéreo, y señal de datos RDS. Supone un gran cambio didáctico para la asignatura "Sistemas de comunicaciones", así como las asignaturas basadas en las comunicaciones. Todo ello usando dispositivos muy baratos que no pasan de los 15€, lo cual es un gran ahorro económico comparado con el instrumental actual, como los "entrenadores de comunicaciones", los osciloscopios, etc. Con un PC y con las herramientas necesarias, el aprendizaje será de una forma mucho más didáctica, ya que en un primer paso se puede presentar los conceptos teóricos con Matlab, y como continuación se puede ver y experimentar en tiempo real con GRC posible prototipos de demoduladores. Todo ello usando procesamiento digital de señales, por lo que podríamos aprender tanto modulaciones/demodulaciones analógicas, como digitales.

El potencial del proyecto está limitado al coste del dispositivo, a pesar de que por muy poco precio estamos consiguiendo grandes resultados en cuanto a recepción de señales, obteniendo material con mejores prestaciones nos permite ampliar conocimientos y profundizar más en los diferentes métodos de modulaciones de señales digitales.

8.2. Líneas Abiertas

Enfatizar más en la decodificación RBDS, obteniendo por pantalla el stream de datos generados por Matlab, así como su obtención en tiempo real en GRC, ya que con objetivo didáctico resulta interesante.También se puede experimentar con múltiples prototipos de demoduladores, usando una amplia gama de filtros y funciones disponibles tanto en Matlab como en GRC, y explotando todo el potencial del programa, y enfocarlo en futuras prácticas de comunicaciones.

Se puede además enfocar el proyecto desde otros entornos de programación como C++ o Phyton, utilizando la aplicación RTL_SDR_API, almacenando paquetes en tiempo real en "buffers", estos lenguajes de programación están muy optimizados y su utilización con fines didácticos resulta más que interesante.

Por otro lado, existe la posibilidad de llegar comprar dispositivos más caros para hacer funciones adicionales, así como llegar a emitir emitir en FM, lo que aumentaría muchísimo nuestros prototipos de moduladores de FM.

Bibliografía

[1] R820T Data Sheet *http://superkuh.com/gnuradio/R820T_datasheet-Non_R-20111130_unlocked.pdf*

[2] «RTL-SDR and GNU Radio with Realtek RTL2832U [Elonics E4000/Raphael Micro R820T] software defined radio receivers.» *http://superkuh.com/rtlsdr.html*

[3] GNU Radio *http://gnuradio.org/redmine/projects/gnuradio/wiki*

[4] FM Estéreo «Wikipedia: FM estéreo» *https://es.wikipedia.org/wiki/FM_estereo*

[5] Aitzol Zuloaga Izaguirre, Euskal Herriko Unibersitatea Universidad del País Vasco. RDS «Radio Data System: RDS» Julio, 1996

[6] RTL-SDR.com «RTL2832U Quick Start Guide» *http://www.rtl-sdr.com/rtl-sdr-quick-start-guide/*

[7] GNU Radio «Ubuntu Install» *http://gnuradio.org/redmine/projects/gnuradio/wiki/UbuntuInstall*

[8] OsmocomSDR «rtl-sdr» *http://sdr.osmocom.org/trac/wiki/rtl-sdr*

[9] OsmocomSDR «osmocom Gnu Radio Blocks» *http://sdr.osmocom.org/trac/wiki/GrOsmoSDR#KnownApps*

[10] Digital Signal Processing «RTL-SDR: Inexpensive Software Defined Radio» *https://inst.eecs.berkeley.edu/ee123/fa12/rtl_sdr.html*

[11] Finite impulse response «Wikipedia: FIR in signal processing» *https://en.wikipedia.org/wiki/Finite_impulse_response*

[12] Chebyshev Filter «Wikipedia: Chebyshev Electronic Filter» *https://en.wikipedia.org/wiki/Chebyshev_filter*

[13] Electrónica Fácil «Modulación Digital :FSK–PSK-QAM» *http://www.electronicafacil.net/tutoriales/MODULACION-DIGITAL-FSK-PSK-QAM.php*

[14] Dr. Aaron Scher, Oregon Institute of Technology How to capture raw IQ data from a RTL-SDR dongle and FM demodulate with MATLAB *http://www.aaronscher.com/wireless_com_SDR/RTL_SDR_AM_spectrum_demod.html*

[15] Mathworks «Ayuda funciones» *http://es.mathworks.com/help/*

[16] GNU Radio «GNU Radio Companion» *http://gnuradio.org/redmine/projects/ gnuradio/wiki/GNURadioCompanion*

[17] Open Security Research «Getting Started with GNU Radio and RTL-SDRs» *http://blog.opensecurityresearch.com/2012/06/getting-started-with-gnu-radio-and-rtl.html*

[18] Josh Blum «The Gnuradio Companion (GRC)» *http://www.joshknows.com/download/grc_old/grc_gnuradio_hackfest_2009_09_06.pdf*

[19] v3l0c1r4pt0r «Using GNU Radio Companion – simple FM radio tutorial» *http://v3l0c1r4pt0r.tk/2013/11/01/using-gnu-radio-companion-simple-fm-radio-tutorial/*. November 1, 2013

[20] Alexandru Csete «GRC Examples» *http://www.oz9aec.net/index.php/grc-examples*

[21] University of Victoria «GNU Radio Companion, Block Documentation» *http://www.ece.uvic.ca/elec350/grc_doc/index.html*. 08 Mar 2013

[22] Pre-emphasis and de-emphasis «Wikipedia: FM Broadcasting» *https://en.wikipedia.org/wiki/FM_broadcasting#Pre-emphasis_and_de-emphasis*

[23] Emphasis «Wikipedia: Emphasis in telecommunications» *https://en.wikipedia.org/wiki/Emphasis_ %28telecommunications %29*

[24] Nick Anotherurl «RDS reception using SDR» *http://www.anotherurl.com/library/sdr/sdrrds.htm*. 12th May 2016