

# Laboratorio de Comunicaciones Digitales Radio Definida por Software

Iván Pinar Domínguez  
Juan José Murillo Fuentes  
Dep. Teoría de la Señal y Comunicaciones  
Universidad de Sevilla



Departamento Teoría de  
la Señal y Comunicaciones

Portada:  
Alejandro Mogollo Díez

# **Laboratorio de Comunicaciones Digitales Radio Definida por Software**

Iván Pinar Domínguez, Juan José Murillo Fuentes

1<sup>a</sup> Edición. 2011

Autores: Iván Pinar Domínguez, Juan José Murillo Fuentes

[murillo@us.es](mailto:murillo@us.es)

Copyright © 2011

ISBN:

Nota legal: Cualquier uso deberá ser permitido expresamente por el autor. Si parte o la totalidad del texto va a ser utilizado en la elaboración de otro material, deberá ser citado en el mismo, indicando título y nombre del autor.

### *Agradecimientos*

Agradecemos a Javier Payán, M<sup>a</sup> José Madero y Mitchel Allegue el que hayan compartido con nosotros sus conocimientos sobre diversos sistemas de comunicaciones digitales implementados en este libro. Asimismo, agradecemos al profesor Payán Somet la revisión que del texto realizó.

La Universidad de Sevilla financió este texto a través del proyecto de innovación docente “Plataforma Software-Hardware SDR Multidisciplinar para la Titulación de Ingeniero de Sistemas de Telecomunicación (SISTEM-I)”. Si con esta ayuda y el contrato de la Universidad de Sevilla número 0623/0263 se financió el capital humano, el proyecto de investigación “Distributed Learning Communication and Information Processing” (DEIPRO) financiado por el Ministerio de Ciencia e innovación y fondos FEDER hicieron posible adquirir el hardware necesario.



## Prólogo

---

En la docencia en la rama de ingeniería de telecomunicación se enseñan varias disciplinas que, una vez imbricadas, explican la transmisión de datos vía cable o propagación electromagnética. Así, el tratamiento digital de señal, las comunicaciones digitales, la electrónica de radiofrecuencia, las antenas o la propagación, se introducen en asignaturas diferentes. Esto suele prevenir al alumno de adquirir unas nociones prácticas sobre cómo luego todas estas materias se conjugan en un sistema de telecomunicación.

Por otro lado, en el apartado de investigación de sistemas digitales de transmisión y recepción de señales, el investigador suele plantear soluciones sobre las que ha experimentado mediante simulación, dado que el *hardware* necesario para hacer pruebas es muy caro de adquirir, configurar y mantener, además de ser específico de un determinado sistema radio. Esto es, sólo ha venido sirviendo para trabajar a unas frecuencias, modulaciones, regímenes binarios y protocolos determinados.

En este escenario aparece en la primera década del siglo XXI la radio definida por software, o *software defined radio* (SDR). A partir del cual se pueden diseñar sistemas de radiocomunicaciones con equipamiento hardware mínimo y realizando el procesamiento en un equipo externo, de tal manera que con un mismo equipo se consigue implementar diversos sistemas de radiocomunicación sin más que realizar un rediseño software. Así, el SDR consta de dos partes. Por un lado el equipo SDR propiamente dicho, que permite subir y bajar una señal a y desde radiofrecuencia. Y por otro, un ordenador, que genera las muestras en banda base que se le pasan al SDR para su transmisión, o recibe las muestras de la señal que el SDR ha recibido para luego detectar la información transmitida.

En ambos procesos se materializan todas las áreas que se enseñan y sobre las que se investiga en ingeniería radio. Para generar la señal hay que aplicar conocimientos de comunicaciones digitales. Como hay que trabajar con muestras en la interfaz entre SDR y el ordenador, es necesario utilizar conceptos de tratamiento digital de señales. Si además hay que generar una trama de algún sistema de comunicación, se aplicarán materias de radiocomunicación. Después, hay que conocer las especificaciones del SDR en términos de ganancias en los amplificadores, frecuencias de trabajo, ruido, antenas, etc. De forma que aparecen involucrados aspectos de electrónica de comunicaciones. Es más, la señal se sube y baja en frecuencia interpolando y diezmando, y de nuevo aparecen conceptos de tratamiento digital de señales. La señal se propaga entre antenas, y los conocimientos sobre propagación y radiación son necesarios.

En definitiva, el SDR nos brinda unas grandes y fantásticas oportunidades de cara a la docencia e investigación, y este texto pretende ser una guía para la puesta en marcha de un laboratorio basado en el mismo. Primero se explican los principios del SDR, de forma que el lector pueda concluir qué se puede y qué no se puede hacer con el SDR. En particular se introduce el USRP 1 (*Universal Software Radio Peripheral*) de Ettus, sus ventajas y sus limitaciones. El USRP presenta dos características muy interesantes. Por un lado tiene un precio asequible, con precios en torno a los 1500 USD. Y por otro, tiene una gran cantidad de software libre disponible.

En segundo lugar se aborda la programación del SDR. Básicamente, el fabricante

Ettus proporciona unos drivers que se pueden compilar para diversos sistemas operativos de forma que cualquier programa podría transmitir y recibir desde el USRP 1. En este libro se describen varias soluciones que incorporan ya este driver. Sería posible programar directamente en Python rutinas para generar y recibir señales, y transmitirlas y recibirlas vía USRP 1. Existen librerías libres disponibles para hacerlo. Sin embargo, esto puede ser engoroso. Así que existe un entorno de ventanas y bloques, el *GNU radio companion*, que de forma intuitiva, y a partir de librerías libres ya existentes, permite la transmisión y recepción vía radio con este SDR de Ettus. Por otro lado, para algunos docentes e investigadores programar en Matlab es inmediato. Además, las librerías de funciones están más depuradas. Y existen herramientas que, basadas en el driver, permiten comunicar un *script* de Matlab, e incluso el Simulink de Matlab, con el USRP 1; ambas se analizan en este libro.

Finalmente, se dedica gran parte de la redacción a presentar diversos ejemplos implementados tanto en GNU radio, y por tanto en Python, como en Matlab. Estos ejemplos incluyen primero el uso del USRP 1 como analizador de espectro y osciloscopio, para pasar luego a presentar sistemas de transmisión con modulaciones QPSK y DQPSK, BPSK, GMSK y OFDM. En estos ejemplos se hace énfasis, además, en los aspectos relacionados con el ancho de banda y muestreo, y los problemas que aparecen en los convertidores, interpoladores y diezmadores.

El resultado es un manual, que esperamos sea de interés al lector, de cómo utilizar el SDR USRP para montar un laboratorio de radiocomunicación, con énfasis en aspectos de tratamiento digital de señal, de arquitectura de sistemas de radiocomunicación y de comunicaciones digitales.

# INDICE

---

<b>Prólogo</b>	vii
<b>INDICE</b>	ix
<b>Capítulo 1 INTRODUCCIÓN</b>	11
1.1 <i>Inicios de la SDR</i>	11
1.2 <i>Wireless Innovation Forum</i>	12
1.3 <i>El concepto de SDR</i>	12
1.4 <i>Conceptos previos</i>	13
1.5 <i>Aplicaciones y mercado</i>	14
<b>Capítulo 2 USRP</b>	17
2.1 <i>Tarjeta USRP 1</i>	17
2.1.1 ¿Qué es el USRP 1?	17
2.1.2 Características del USRP 1	18
2.1.3 Tipos de tarjetas secundarias y antenas	23
2.1.4 Instalación del USRP	27
<b>Capítulo 3 GNU RADIO</b>	30
3.1 <i>Instalación de GNU Radio</i>	31
3.1.1 Instalación en Ubuntu	32
3.1.2 Instalación en MAC	32
3.2 <i>Introducción al diseño de módulos en GNU Radio</i>	33
3.2.1 GNU Radio Companion	34
<b>Capítulo 4 DISEÑOS EN GNU RADIO COMPANION</b>	37
4.1 <i>Analizador de espectros</i>	37
4.1 <i>Sistema FM</i>	46
4.1.1 Transmisor FM	46
4.1.1 Receptor FM	51
4.1 <i>Sistema (D)QPSK</i>	58
4.1.1 Transmisor (D)QPSK	60
4.1.2 Receptor (D)QPSK	71
4.2 <i>Sistema digital</i>	78
4.2.1 Transmisor digital	79
4.2.1 Receptor digital	87
4.3 <i>Cálculo de Bit Error Rate</i>	94
4.1 <i>Sistema OFDM</i>	96
4.1.1 Transmisor OFDM	96
4.1.1 Receptor OFDM	101
<b>Capítulo 5 USRP Y MATLAB</b>	105
5.1 <i>Toolbox USRP para Simulink</i>	106
5.1.1 Analizador de espectros con Simulink-USRP	110
5.2 <i>Toolbox SDR4all</i>	111
5.2.1 Introducción a la transmisión de señales utilizando SDR4all	113

5.2.2    Diseño de un sistema de transmisión DQPSK utilizando SDR4all	115
<b>Conclusiones</b>	<b>133</b>
<b>Apéndices</b>	<b>136</b>
<b>BIBLIOGRAFÍA</b>	<b>161</b>

# **Capítulo 1 INTRODUCCIÓN**

---

Un sistema Software Defined Radio (SDR), o radio definida por software, es un sistema de radiocomunicación donde la mayor parte de los componentes necesarios se implementan en software en lugar de en hardware. Al utilizar esta tecnología, se implementa un receptor *Zero-IF* o *low-IF*<sup>1</sup> configurable de tal manera que puede utilizarse para diseñar distintos componentes como mezcladores, filtros, amplificadores, moduladores/demoduladores y detectores entre otros e incluso sistemas completos tales como transmisores, receptores, transceptores, osciloscopios, analizadores de espectros o analizadores vectoriales de redes, siendo sus parámetros configurables dinámicamente y por consiguiente aportando una gran flexibilidad a la hora de realizar un sistema de radiocomunicación.

## **1.1 Inicios de la SDR**

En la evolución de las telecomunicaciones inalámbricas han aparecido diversas tecnologías para el intercambio de información entre dos puntos distantes, con requisitos cada vez más exigentes. Las incompatibilidades entre las diferentes tecnologías han supuesto un problema a la hora de reutilizar equipos o prestar determinados servicios, como por ejemplo los terminales de telefonía móvil. La SDR surge para solucionar estos inconvenientes de compatibilidad e interoperabilidad, definiendo un conjunto de procedimientos y técnicas orientadas a realizar el procesamiento de señales radio por medio de un dispositivo de propósito general, el cual puede ser modificado mediante software logrando así un cambio dinámico, automático y eficiente entre tecnologías sin tener que incurrir en costes, de ahí surge la necesidad de diseñar un dispositivo capaz de entregar en banda base, o *low IF*, la señal radio para su procesamiento digital, incluso si la señal es analógica. El dispositivo debe ser configurable para que sea posible recibir distintos anchos de banda y frecuencias para posteriormente, con un procesado digital, poder realizar la sincronización y la detección de la señal recibida.

La primera implementación importante del concepto SDR fue en el proyecto militar

---

<sup>1</sup> IF, *intermediate frequency* o frecuencia intermedia

estadounidense SpeakEasy, cuyo objetivo era implementar más de 10 tipos de tecnologías de comunicaciones inalámbricas en un equipo programable, operando en la banda de frecuencias de 2MHz a 200MHz. Un objetivo adicional del proyecto era que el prototipo debía tener la posibilidad de actualizar su código para que así se pudieran tener en cuenta los estándares futuros. El proyecto empezó en 1991 y sólo en 1995 fue posible lograr todos los objetivos planteados. El problema fue que únicamente se podía realizar una comunicación a la vez, por lo cual se modificaron sus alcances apareciendo así una segunda fase en la cual se trabajarían aspectos como disminución de peso y costo, incremento en su capacidad de procesamiento, simultaneidad de comunicaciones o diseño basado en arquitecturas de software libre. La nueva fase del proyecto necesitó 15 meses para lograr sus objetivos, obteniendo así importantes resultados que llevaron a la producción del dispositivo diseñado, el cual trabajó en el rango de 4 MHz a 400 MHz.

Desde entonces, se han diseñado diferentes dispositivos SDR que han marcado un importante avance en este campo. Entre estos, el equipo Universal Software Radio Peripheral (USRP) del fabricante Ettus Research ha contribuido especialmente a acercar al usuario esta tecnología.

## 1.2 Wireless Innovation Forum

Actualmente no hay ningún estándar sobre SDR, pero sí existe un grupo de trabajo denominado *Wireless Innovation Forum*, que es un foro dedicado a conducir la innovación tecnológica en el campo de la SDR desarrollando estándares y especificaciones del mismo, logrando así la divulgación de dicha tecnología para soportar necesidades tanto militares y civiles como comerciales. Los miembros de este grupo constituyen una amplia base en el diseño de plataformas Software Defined Radio (SDR), Cognitive Radio (CR) y Dynamic Spectrum Acces (DSA). Cuenta con el apoyo de más de 100 empresas, instituciones y organizaciones como Altera, Xilinx, NASA, VirginiaTech, Toshiba, Samsung, Lockheed, Motorota, QUALCOMM, Hitachi u Ohio Aerospace Institute, entre otras. El *Wireless Innovation Forum* prepara una serie de eventos periódicos en los que se presentan productos, avances y estándares que proponen las empresas inscritas en él con respecto al tema mencionado.

## 1.3 El concepto de SDR

El concepto SDR ha ido evolucionando con los años pero se siguen basando en el esquema básico que se muestra en la Figura 1-1, compuesta por tres bloques funcionales: sección de RF<sup>2</sup>, sección de IF y sección banda base. La parte de RF e IF se implementan en hardware mientras que la sección de banda base en software.

---

<sup>2</sup> RF, *radiofrequency* o radiofrecuencia

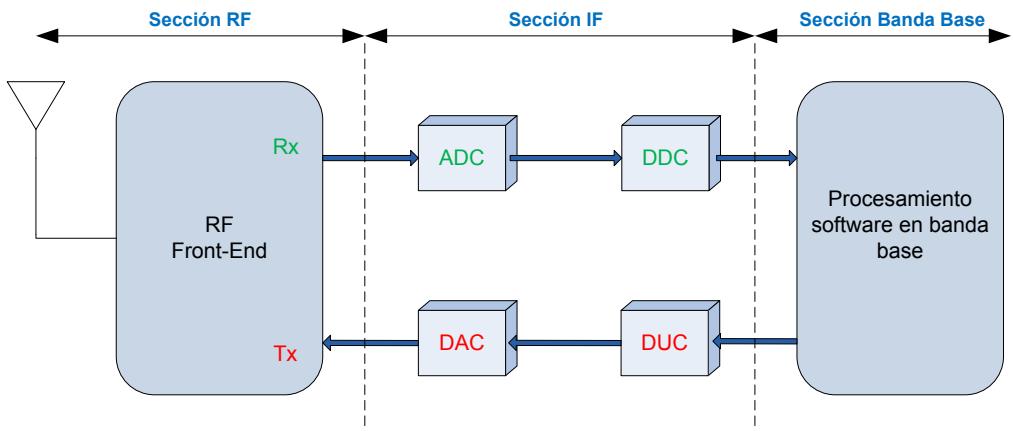


Figura 1-1 Diagrama de bloques funcionales de SDR

La sección de RF, también denominada *RF Front-End* o cabecera de RF, es la encargada de transmitir/recibir las señales de radiofrecuencia para adecuarlas y convertirlas en frecuencia intermedia en recepción o amplificar y modular las señales de IF en el caso de transmisión. La frecuencia intermedia puede ser 0, dando lugar al concepto de *Zero-IF*; el cual es posible gracias a los avances en los componentes hardware.

De igual manera, la sección de IF se encarga de pasar la señal de IF a banda base y digitalizarla en recepción o pasar la señal de banda base a IF y hacer la conversión digital-analógica de la señal en el caso de la transmisión. Las encargadas de la conversión analógico-digital o digital-analógica de la señal son los módulos ADC/DAC. A su vez, se insertan los módulos DDC/DUC para poder bajar/subir, respectivamente, la tasa de muestreo en el sentido de recepción/transmisión, consiguiendo que la tasa de muestras por la interfaz entre IF y banda base sea inferior.

La sección de banda base es la encargada de todo el procesamiento en banda base de la señal como modulación/demodulación, análisis espectral de la señal,.. llevándose a cabo en software.

## 1.4 Conceptos previos

Una vez analizado la estructura de un sistema SDR, a continuación se analizan en profundidad algunos de sus bloques.

**ADCs:** Realizan el paso de señales analógicas a digitales asignando a cada nivel de tensión un número digital para ser utilizado por el sistema de procesamiento. Un convertidor analógico a digital tiene cuatro características principales:

- *Sample Rate (Tasa de Muestreo):* es el número de veces por segundo que el ADC toma una medida de la señal analógica y cuantifica el valor analógico utilizando para ello un conjunto bits. Mientras más bits se utilicen, menor será el error de cuantificación, es decir, el error entre la señal analógica medida y la salida del

ADC<sup>3</sup>.

- *Dynamic Range (Rango Dinámico)*: se refiere a la diferencia entre la señal más pequeña y la más grande que puede convertir el ADC. Hay que tener en cuenta el número de bits que se utilizan, ya que, con un rango dinámico determinado, el error de cuantificación será menor a medida que se aumente el número de bits.
- *Tiempo de conversión*: existe un tiempo de conversión  $T_c$  que necesita el ADC para obtener un número digital a partir de un dato analógico.
- *Número de niveles*: indica la precisión con la que se cuantifica un dato analógico y depende del número de bits del ADC.

**Teoría de Nyquist:** La teoría de Nyquist determina que, para evitar el efecto de aliasing cuando convertimos de analógico a digital, la frecuencia de muestreo del ADC debe ser de al menos dos veces el ancho de banda de la señal de interés, es decir, para poder reconstruir fielmente la señal analógica a partir de la salida del ADC, salvo por el error de cuantificación, la señal analógica debe ser limitada en banda y la frecuencia de muestreo igual o mayor al doble del ancho de banda de la señal, asumiendo que la señal analógica está en banda base.

$$\text{Ancho de banda de interés: } \{0 - f_{\max}\} \Rightarrow \text{Frecuencia de muestreo} \geq 2f_{\max}$$

**DACs:** Un convertidor digital a analógico es un elemento que recibe información de entrada digital, en forma de una palabra de  $n$  bits y la transforma a señal analógica. Para ello, dada una señal digital expresada por un conjunto de valores cuantificados en  $n$  bits asociados a un tiempo de muestreo  $T_s$ , se puede generar su versión analógica obteniendo una delta de valor el indicado por los  $n$  bits y filtrando el resultado por un filtro paso de baja perfecto (rectangular en frecuencia con ancho de banda el de la señal analógica original), denominado filtro reconstructor. Las características de los DACs son similares a los ADCs, siendo capaz de generar señales de frecuencia máxima igual a la frecuencia Nyquist (es decir, la mitad de la de muestreo). En la práctica, como los filtros reconstructores no son perfectos, la frecuencia máxima que puede generar adecuadamente es más baja que la frecuencia Nyquist.

**RF front end:** Un RF front end se encarga de trasladar adecuadamente y amplificar el centro de un rango de frecuencias a otro rango de frecuencias. La frecuencia central del rango de salida es la frecuencia intermedia (IF) y generalmente será 0 (Zero-IF).

## 1.5 Aplicaciones y mercado

Dados los objetivos de diseño del SDR, el conjunto de aplicaciones de esta tecnología es muy amplio, las cuales incluyen nuevos terminales para soportar cualquier tipo de tecnología, infraestructura de comunicaciones, como por ejemplo estaciones base de telefonía móvil, y soporte a tecnologías de acceso radio de banda ancha tanto en entornos privados como públicos.

En la primera etapa (2000-2005) la mayoría de las aplicaciones de SDR eran de tipo militar debido a los altos costes en su implementación, correspondiendo el 76,5% de todas las aplicaciones a este entorno, el 7,1% a infraestructura inalámbrica comercial, y el 16,4% a otro

---

<sup>3</sup> Se asume que la cuantificación es uniforme.

tipo de aplicaciones ([10]). Los avances en microelectrónica y en plataformas de desarrollo de software para SDR así como su inclusión como alternativa para la construcción de la capa física de los estándares de redes modernas han hecho que el mercado de SDR en el 2007 sobrepase los 5,3 billones de dólares, de los cuales 335 millones corresponden a venta de procesadores banda base para SDR, aumentando estos valores exponencialmente en los años siguientes ([10]).

Si nos centramos en la cadena de valor, existen tres entes:

- Los vendedores de dispositivos y componentes base para SDR
- Los vendedores de infraestructura y equipos basados en SDR
- Usuarios

Sin embargo hay que tener en cuenta que la mayoría de las tecnologías están orientadas a ofrecer datos y servicios multimedia, por lo que los desarrolladores de software base y de aplicaciones, así como los desarrolladores de contenido y operadores, tendrán un papel importante en esta cadena de valor y especialmente en los modelos de negocio que se formarán alrededor de la SDR. Una vez llegado este punto, debe aparecer otro actor: el regulador. A fecha actual no existe una regulación clara en torno a la Radio Definida por Software pero claro está que gran parte del éxito de SDR vendrá marcado por la flexibilidad de la regulación y estandarización, donde la SDR juega un papel imprescindible.

Actualmente, existen numerosos proyectos de SDR, entre ellos cabe destacar el ambicioso proyecto *OpenBTS* con el que se implementa una estación base de telefonía GSM en software a partir de la arquitectura GNU Radio y el USRP apoyándose en *Asterisk*<sup>4</sup> para servicio VoIP. A modo de ejemplo se muestra la arquitectura funcional, pero antes conviene introducir el funcionamiento del USRP con la siguiente figura:

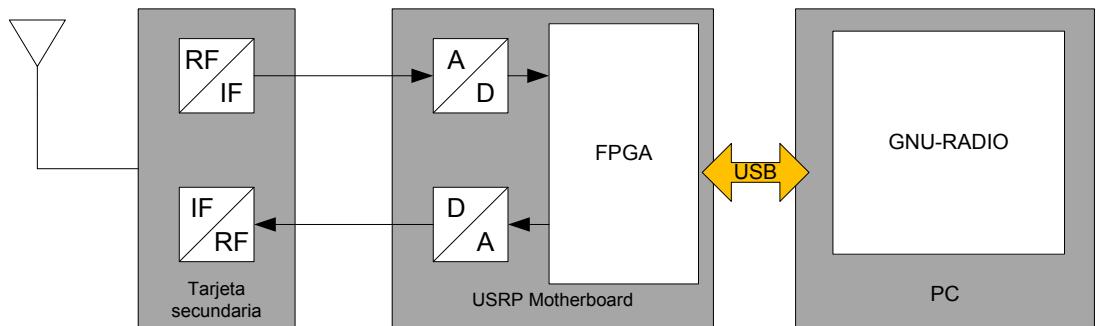


Figura 1-2 Diagrama básico del Universal Software Radio Peripheral

---

<sup>4</sup> **Asterisk** es un programa de software libre (bajo licencia GPL) que proporciona funcionalidades de una central telefónica (PBX). Como cualquier PBX, se puede conectar un número determinado de teléfonos para hacer llamadas entre sí e incluso conectar a un proveedor de VoIP.

La señal radio es captada o transmitida por la antena y pasando por la tarjeta secundaria, o *daughterboard*, que implementa la cabecera de RF. En la tarjeta motherboard, o *motherboard*, se produce la conversión A/D y D/A respectivamente y en la FPGA se realiza un procesado de la señal digital para reducir la tasa de muestras en la interfaz USB. Finalmente se conecta al PC mediante dicha interfaz para realizar el procesamiento software con la arquitectura GNU Radio.

El diagrama funcional del proyecto *OpenBTS* se muestra en la

Figura 1-3.

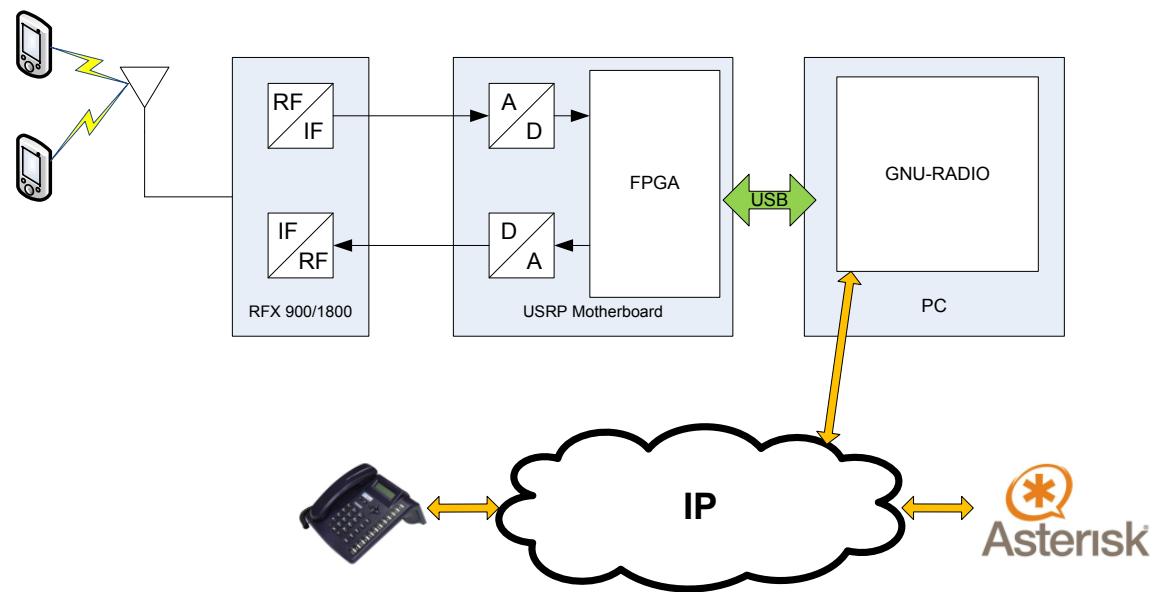


Figura 1-3 Arquitectura OpenBTS

## Capítulo 2 USRP

---

### 2.1 Tarjeta USRP 1

#### 2.1.1 ¿Qué es el USRP 1?

El Universal Software Radio Peripheral es un periférico del fabricante Ettus Research diseñado para trabajar en conjunto con un procesador externo (PC, Workstation, ...) a través de una FPGA y permite la realización de *software radios*. Al comienzo de la preparación de este texto existían dos versiones, la 1 y la 2. En este documento se describirá y utilizará la versión 1. Y en adelante nos referiremos a él simplemente como USRP. En este texto se ha escogido este equipo por dos motivos. En primer lugar por el precio. Por entre 1000 y 2000 USD se puede adquirir un equipo USRP. Y en segundo lugar, hay disponible una gran variedad de software libre para el USRP.



Figura 2-1 Universal Software Radio Peripheral

El USRP realiza las funciones de llevar la señal a banda base de RF a través de la sección de IF y viceversa tal y como se muestra en la Figura 2-2. La interfaz USB permite comunicar al USRP con el ordenador que realizará el procesamiento software. El ordenador puede ser un

ordenador personal, un ordenador portátil, una estación de trabajo, etc. En general un ordenador que soporte sistema operativo Linux, MAC OS o Windows.

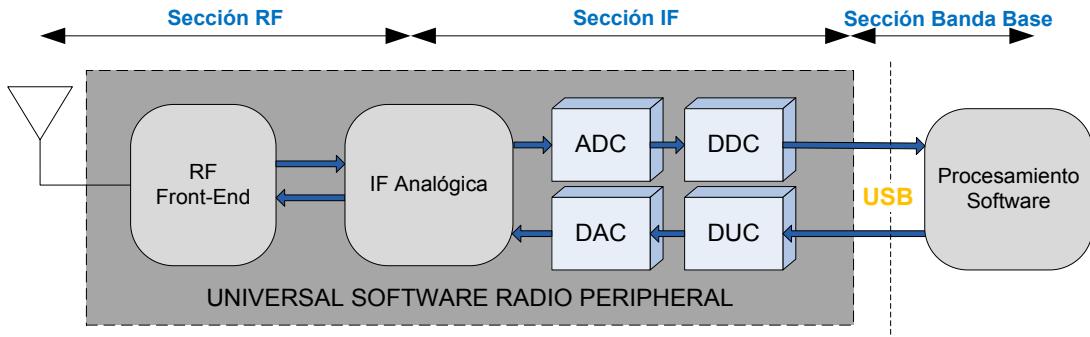


Figura 2-2 Bloques principales en el sistema de comunicaciones de Software Radio

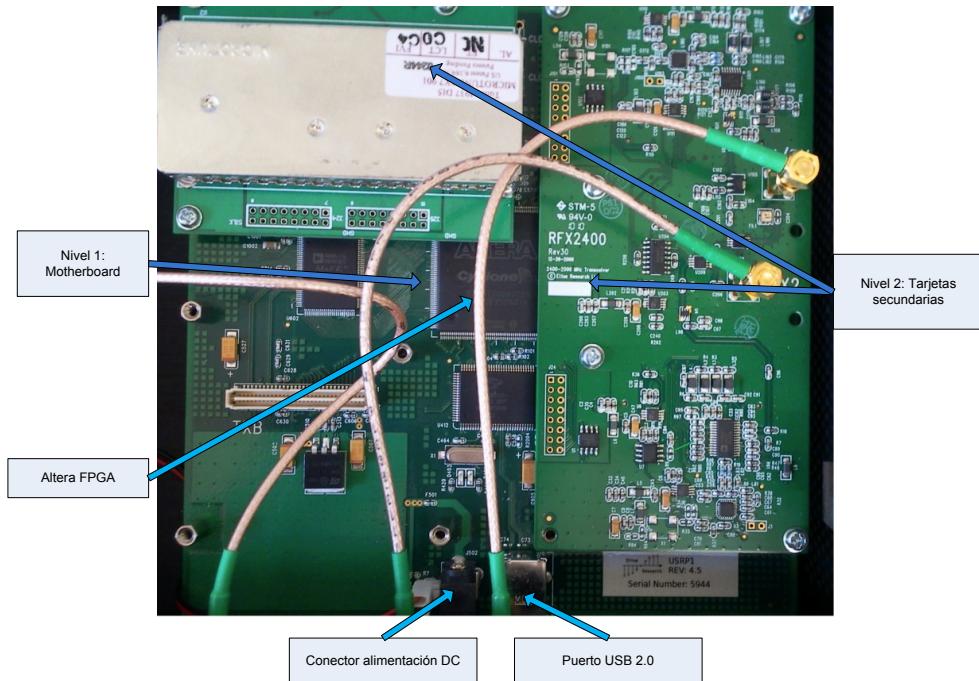


Figura 2-3 Fotografía el interior del USRP con la tarjeta principal al fondo y las secundarias en primer plano.

### 2.1.2 Características del USRP 1

El USRP cuenta con dos niveles de tarjetas como se muestra en la Figura 2-3. El primero es la tarjeta principal, también denominada madre o *motherboard*, en donde se encuentra la

FPGA, los convertidores ADC's y DAC's, la alimentación y la conexión vía USB<sup>5</sup>. El segundo nivel se compone de tarjetas secundarias, hijas o *daughterboards*. Éstas existen para transmisión y/o recepción. Así, el USRP puede trabajar con varias tarjetas secundarias, que pueden funcionar como transmisores, receptores ó transceptores (*transceiver*, TRX), en este último caso pueden transmitir y recibir a la vez, y llevan la señal de banda base o IF hasta la banda de RF deseada o viceversa.

La Figura 2-4 muestra un diagrama de bloques del USRP, donde se puede ver la disposición de la FPGA, convertidores, tarjetas secundarias y la interface USB.

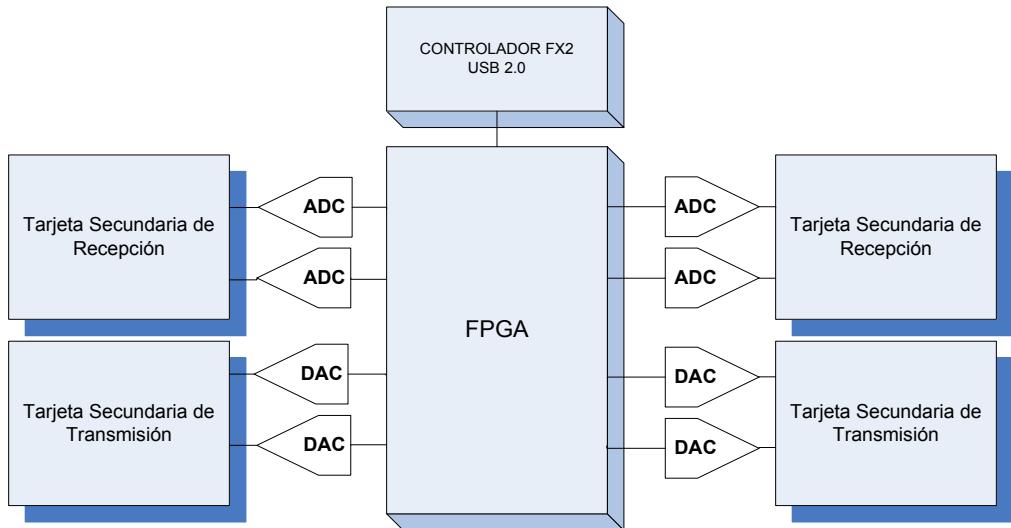


Figura 2-4 Diagrama a bloques del Universal Software Radio Peripheral (USRP)

## Primer nivel: convertidores, procesador e interfaces

En este apartado se mostrará la arquitectura de la tarjeta principal del USRP, es decir, la FPGA junto a los convertidores AD y DA. Para una visión más detallada, se recomienda al lector interesado dirigirse al Apéndice B.

### Convertidores AD

El USRP tiene 4 convertidores ADC de alta velocidad, cada uno a 12 bits por muestra y 64 millones de muestras por segundo, luego teóricamente se puede recibir una señal que tenga un ancho de banda paso de baja menor o igual a 32 MHz.

Existe un *amplificador de potencia programable* (PGA) antes del ADC para amplificar la señal de entrada en el caso de que sea débil, además de la ganancia que introduce la tarjeta secundaria.

---

<sup>5</sup> En la versión USRP 2, la conexión no se realiza por USB sino por Ethernet, permitiendo el envío de una mayor tasa de información.

## Convertidores DA

El USRP tiene 4 convertidores DAC de alta velocidad para transmisión, cada uno a 14 bits por muestra y 128 millones de muestras por segundo (128 MS/s), por lo que la frecuencia de Nyquist es de 64 MHz en teoría.

Contamos también con un PGA conectado después del DAC para aumentar la ganancia 20 dB, aunque la ganancia final puede ser mayor ya que algunas tarjetas secundarias permiten amplificar la señal en RF.

## Procesador e Interfaces

El procesador utilizado por el USRP es una FPGA Altera Cyclone EP1C12 al que están conectados los ADC's y DAC's. Comprender el funcionamiento de la FPGA es importante de cara a utilizar GNU Radio, ya que la tarea de éste es realizar un procesado de las señales en la banda que se precise para reducir las tasas de muestreo de datos en la interfaz USB 2.0, en el USRP 1. A este proceso se le denomina *diezmado* en recepción e *interpolado* en transmisión.

En recepción la señal se convierte hacia una frecuencia menor. La configuración estándar de la FPGA incluye 4 convertidores digitales de bajada (DDC) para disminuir la tasa de muestreo (diezmado). El factor de diezmado no puede tomar cualquier valor (sólo pares entre 8 y 512) y por temas de eficiencia a veces es recomendable que sea potencia de 2 (para algoritmos como FFT<sup>6</sup>). La Figura 2-5 muestra un diagrama de bloques de la trayectoria de recepción desde la antena hasta la interfaz USB.

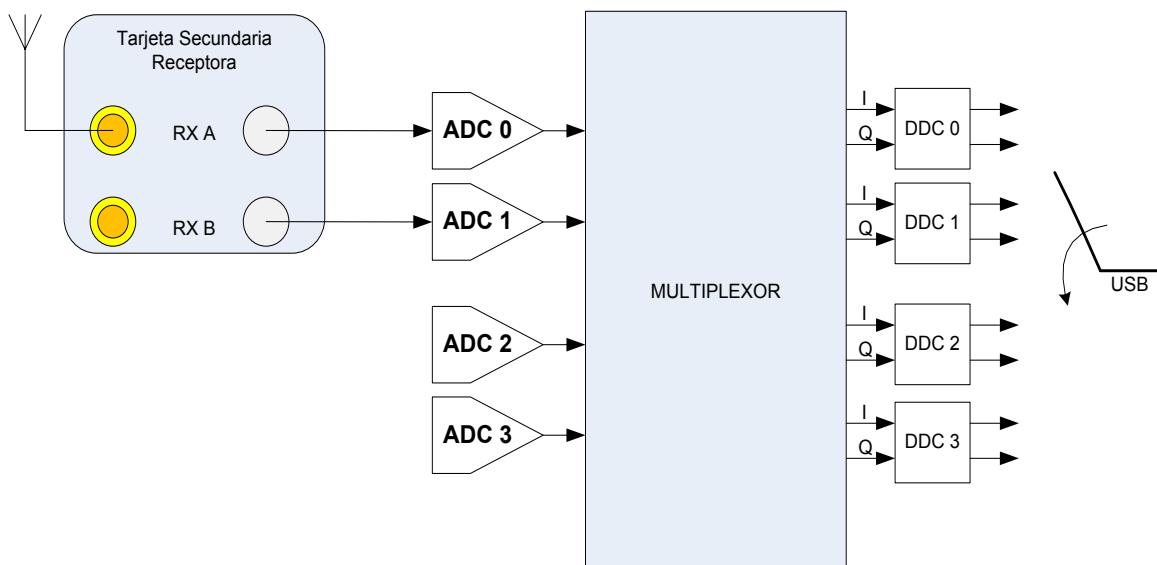


Figura 2-5 Multiplexor en el USRP, donde I es la señal en fase y Q la señal en cuadratura

El multiplexor (MUX) en la Figura 2-5 es un circuito seleccionador, que determina que ADC se conecta a cada DDC.

<sup>6</sup> FFT, *fast Fourier transform*

**Ejemplo:** si se diseñara un receptor de FM, donde el ancho de banda de una estación de FM es de 200 kHz, podemos seleccionar un factor de diezmado de 200, con lo que la tasa de recepción (y ancho de banda) a través del USB será de  $64\text{MS/s} / 200 = 320 \text{kS/s}$  (y por tanto 320 kHz) el cual satisface los 200 kHz de ancho de banda sin perder información.

En general, en comunicaciones, la señal no tendrá un espectro en banda base simétrico y hará falta aportar parte en fase (I) y parte en cuadratura (Q). De forma que, en general, finalmente las señales de I y Q entran al ordenador vía USB y lo siguiente es todo procesamiento software.

Para la transmisión, en la conversión hacia arriba, todo sucede de forma similar, con la diferencia de que el proceso es contrario. Se envía una señal I y Q en banda base al USRP. El convertidor digital de subida (DUC) contenido fuera de la FPGA en el chip AD9862 CODEC lleva a cabo un proceso de interpolación de la señal y la envía a través del DAC. El factor de interpolado será 128 MS/s dividido por la tasa de muestreo de los datos de entrada al USRP.

Por último, se mostrará una figura resumen del proceso de recepción desde la antena hasta la interfaz USB en cuanto a muestreos y ganancias se refiere. Hay que tener en cuenta que normalmente la frecuencia intermedia será 0, por ello la tarjeta secundaria intentará sintonizarse al máximo a la radiofrecuencia deseada. El proceso de transmisión sería simétrico. Para más detalle, consultar el Apéndice B.

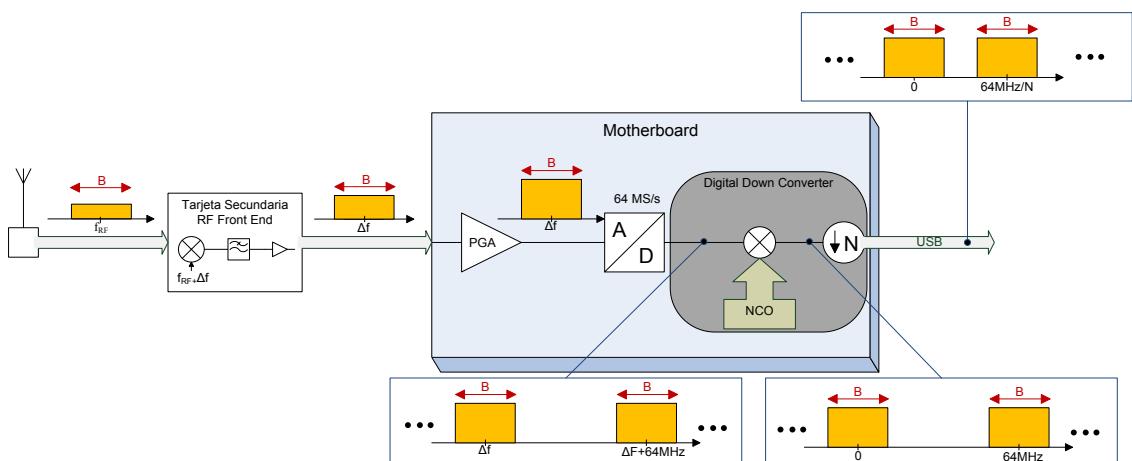


Figura 2-6 Proceso de recepción

### Segundo nivel: tarjetas secundarias y antenas

Una vez analizado el primer nivel, la señal discurre por la segunda etapa (sección de RF) hasta ser transmitida por la antena o viceversa. La sección de RF se implementa en la tarjeta secundaria. La localización de las tarjetas secundarias se mostró en la Figura 3-3 y realizan las

funciones de la sección de RF. Se conectan al primer nivel que cuenta con 4 entradas, 2 para recepción y 2 para transmisión.

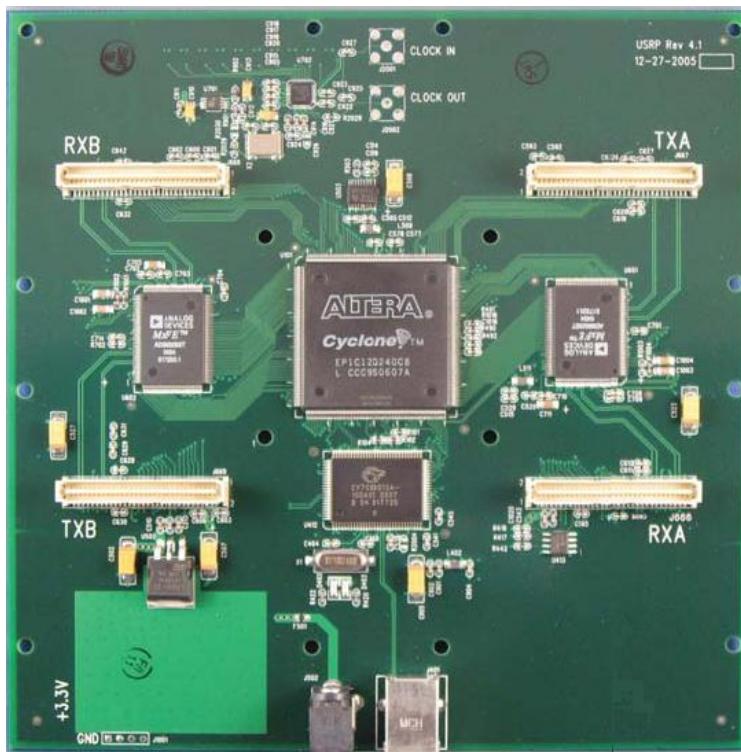


Figura 2-7 Disposición de los elementos en la tarjeta principal o motherboard

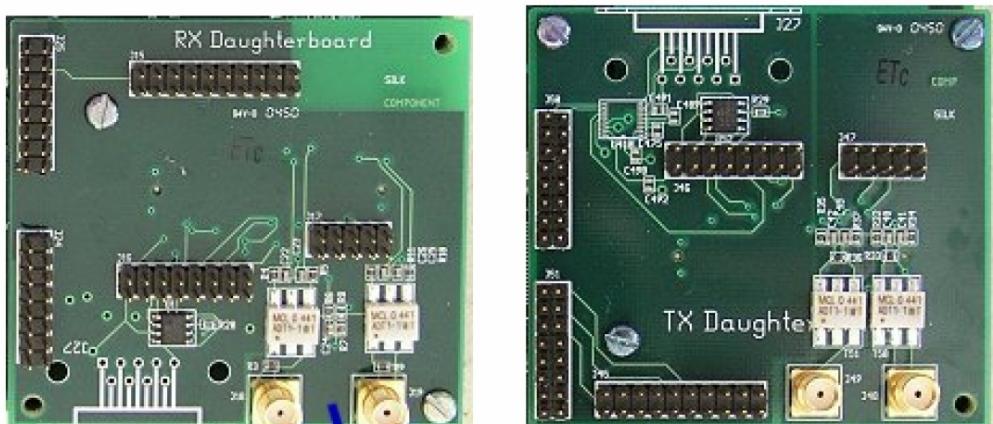


Figura 2-8 Imagen de las tarjetas secundarias Basic RX y Basic TX

En el USRP 1 existen 2 ranuras para tarjetas de transmisión etiquetadas como TXA y TXB y sus dos correspondientes para recepción etiquetadas como RXA y RXB. Cada ranura para tarjeta tiene acceso a 2 de los 4 convertidores de datos de alta velocidad (Salidas de DACs para TX y entradas de ADC para RX). En la Figura 2-7 aparece la disposición de los elementos en la tarjeta principal.

En principio, existen 4 entradas (2 por ranura RX) y 4 salidas (2 por ranura TX) en el USRP si se utilizan señales reales. Sin embargo, hay más flexibilidad y ancho de banda si se hace un muestreo complejo (I-Q), por lo que habrá 2 entradas complejas (1 por ranura) y 2 salidas complejas (1 por ranura). En la Figura 2-8 se incluyen dos tipos tarjetas secundarias.

### 2.1.3 Tipos de tarjetas secundarias y antenas

Podemos ver en la Figura 2-8 que cada tarjeta secundaria básica cuenta con dos conectores de tipo SMA, los cuales se utilizan para acoplar una antena o bien conectar señales de entrada (como un generador de señales) o salida (como un osciloscopio).

Los tipos de tarjetas secundarias más importantes que se puede adquirir para utilizar en el USRP 1 son:

- **Basic daughterboards:** Son las más sencillas. Existen bien para transmisión (Basic Tx) o bien para recepción (Basic Rx) y son utilizadas para conectar sintonizadores o generadores de señales puesto que operan en banda base (no realizan una traslación de frecuencias). El rango frecuencial para estas tarjetas es de hasta 64MHz. Se pueden utilizar para transmisiones del tipo de AM en incluso FM con algunas limitaciones.
- **DBSRX daughterboards:** Éstas son también sólo para recepción, y su rango de frecuencia va desde 800MHz hasta 2.4GHz.

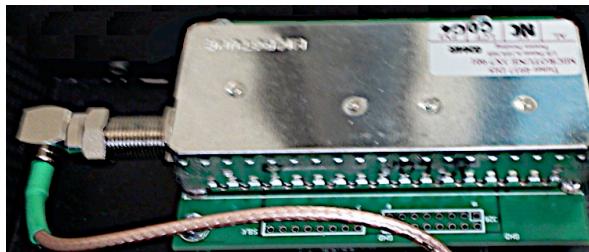


Figura 2-9 Tarjeta TVRX

Tabla 2-1 Características tarjeta TVRX

Rango de frecuencia	50 MHz -860 MHz
Figura de ruido	8 dB
Ancho de banda	6 MHz
Ganancia	0-95 dB

- **TVRX daughterboards:** Son tarjetas únicamente para recepción con un ancho de banda máximo de 6MHz. Se utilizan para detección de señales de FM y TV (NTSC, PAL no es posible porque utiliza un ancho de banda de 8 MHz). En la Figura 2-9 se incluye una imagen de la tarjeta. Las especificaciones son las incluidas en la Tabla 2-1.
- **WBX daughterboards:** Es una tarjeta transceptor con un amplio ancho de banda aportando una gran flexibilidad a la hora de realizar diseños SDR. Las especificaciones son:

Tabla 2-2 Características tarjeta WBX

<b>Rango de frecuencia</b>	50 MHz – 2.2 GHz
<b>Figura de ruido</b>	5-6 dB
<b>Sensibilidad</b>	-130 dBm
<b>IIP2</b>	40-55 dBm
<b>IIP3</b>	5-10 dBm
<b>Ganancia máxima RX</b>	70 dB
<b>Alimentación</b>	6 V – 1.1 A
<b>Antena</b>	TX/RX, RX2
<b>Potencia de transmisión</b>	50-100 mW (< 1 GHz), 30-50 mW (> 1 GHz)

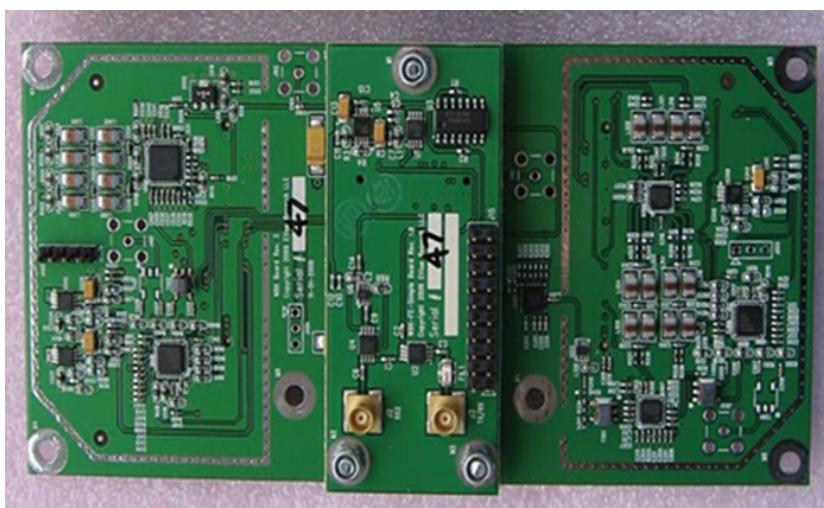


Figura 2-10 Tarjeta WBX

- **RFX 2400 daughterboards:** Es una tarjeta transceptor, es decir, transmisora y receptora simultáneamente, que está pensada en principio para usarse en la banda ICM de 2.4 GHz. La ganancia es controlable en recepción aunque no en transmisión. Puede configurarse como sistema MIMO si se utilizan dos tarjetas de este tipo,

aunque con una sola Daughterboard se podría realizar un sistema SIMO, puesto que cada tarjeta consta de dos conectores, TX/RX y RX2, es decir, un transmisor y dos receptores. Las especificaciones son:

Tabla 2-3 Características tarjeta RFX2400

<b>Rango de frecuencia</b>	2.3 GHz- 2.9 GHz
<b>Figura de ruido</b>	6-10 dB
<b>Ganancia en recepción</b>	0-70 dB
<b>Potencia de transmisión</b>	17 dBm
<b>Antena</b>	TX/RX, RX2

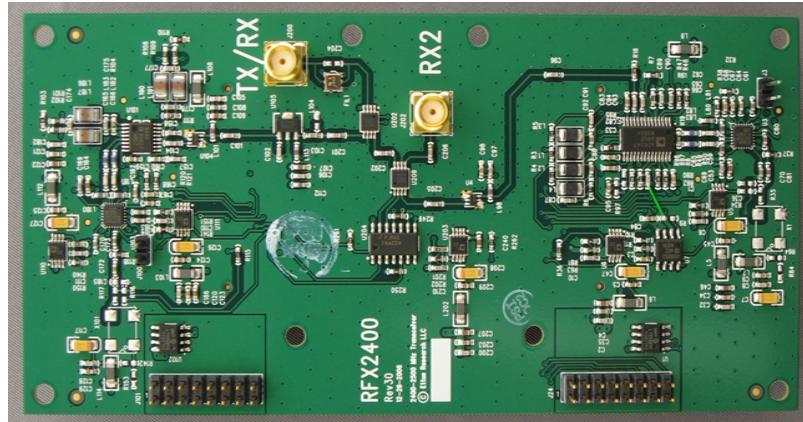


Figura 2-11 Tarjeta RFX2400

En cuanto a los elementos de captación, las antenas más utilizadas son las siguientes:



Figura 2-12 Antenas utilizadas

- **VERT400:** Antena tribanda 144MHz, 400 MHz y 1200 MHz.
- **VERT2400:** Antena de doble banda 2400-2480 MHz y 4.9-5.9 GHz. Ideal para

utilizar con RFX2400 y XCVR2450.

## Conexión mecánica y eléctrica

### Conexión Mecánica

El USRP incluye un equipo completo de pilares y tornillos para realizar una conexión mecánica adecuada. Tenemos 20 pilares (*standoffs*) M3x10mm M-F, de los cuales 4 son utilizados como soporte para el USRP, ubicados en las cuatro esquinas de la tarjeta, los 16 pilares restantes y los tornillos son utilizados para sostener las cuatro tarjetas secundarias. En la

Figura 2-13 se muestra la conexión mecánica de los pilares y tornillos en el USRP:

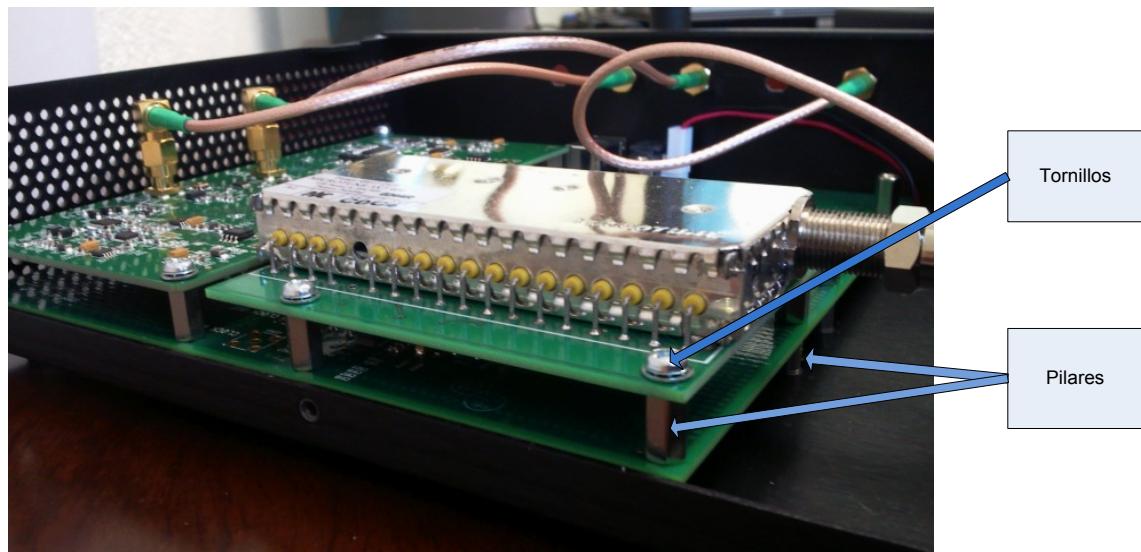


Figura 2-13 Pilares y tornillos. Conexión Mecánica del USRP

La posición en el USRP de los pilares está diseñada para prevenir accidentes o conexiones incorrectas de las tarjetas secundarias.

---

**NOTA: El USRP no debe ser operado sin los pilares y las tarjetas secundarias no deben ser conectadas o desconectadas mientras el USRP está encendido.**

---

## Conexión Eléctrica

El USRP necesita un convertidor de 6V 3.5A. El convertidor, que se adquiere junto con el USRP, funciona con voltajes de corriente alterna de 100 a 240V, y opera a 50/60Hz. El conector que va al USRP es de 2.1mm/5.5mm *DC power connector*. El USRP por sí solo necesita de 5V y 2A, aunque la fuente de alimentación es de 6V para utilizar algunas tarjetas secundarias específicas (como la WBX por ejemplo).

Para la comunicación con el procesador externo es necesario el cable USB, el cual debe ser conectado a una ranura capaz de soportar USB 2.0, por lo que en ranuras USB 1.1 el USRP no funciona.

Al conectar el USRP, se debe visualizar un LED parpadeando de 3 a 4 veces por segundo, esto indica que el procesador está corriendo y el dispositivo se encuentra en modo de ahorro de energía. Una vez que el soporte lógico (diseño en Python) es cargado a la FPGA, el LED parpadeará a una velocidad menor. En caso de que el LED no muestre ninguna señal, es necesario revisar las conexiones eléctricas, y revisar si hay continuidad en el fusible principal F501, el cual se encuentra cercano al conector de energía como se muestra en la Figura 2-14. Si este fusible requiere ser reemplazado las especificaciones son: tamaño 0603 y corriente 3A.

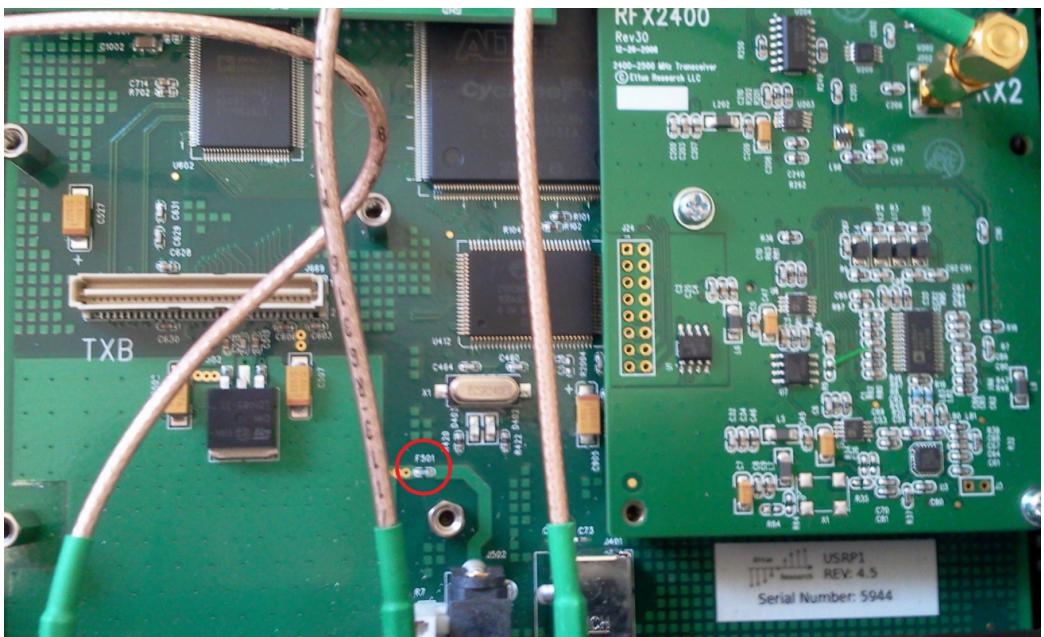


Figura 2-14 Fusible, Conexión eléctrica del USRP

### 2.1.4 Instalación del USRP

La instalación del USRP es bastante sencilla, es necesario haber instalado GNU Radio, ver siguiente capítulo, y hacer la conexión correcta de la tarjeta, por lo que sólo se necesita

conectar el USRP a la corriente y con el cable USB al ordenador. Ahora se abre una terminal en el PC os similar y se ejecuta el siguiente comando:

```
# tail -f /var/log/messages
```

Con este comando se puede ver en la terminal el estado de las conexiones USB, por lo que al conectar el USRP se debe observar un mensaje como el siguiente:

```
new high speed USB device using address 1
```

Ahora al desconectar el cable USB debe aparecer un mensaje correspondiente a haber retirado el conector de la ranura. En caso de que al conectarlo se obtenga un mensaje como el siguiente:

```
new full speed USB device using address 1
```

Significa que la ranura de USB no soporta la versión 2.0 de éste.

Para probar que todo funcione correctamente, en una terminal de Ubuntu habrá que situarse sobre el directorio `gr-build/gnuradio-examples/python/usrp/` si se realizó la instalación manual, o bien en el directorio `/usr/local/gnuradio/examples` si se optó por la instalación automática (con Synaptic). El equivalente en Mac sería `/opt/local/share/gnuradio/examples`. Tras esto, ejecutar el siguiente comando:

```
# ./usrp_oscope.py
```

Si todo ha sido instalado correctamente se debe visualizar una ventana que muestra un osciloscopio similar a la Figura 2-15.

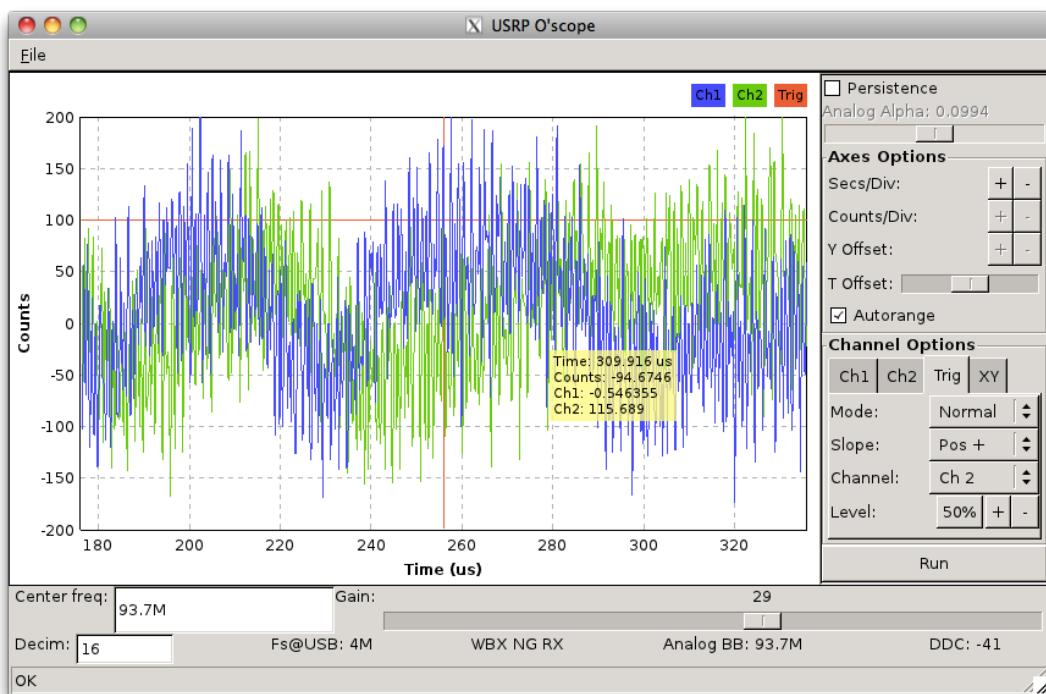


Figura 2-15 Osciloscopio de recepción en GNU Radio

Este programa que se acaba de ejecutar funciona como un osciloscopio y se le pueden conectar señales para analizarlas.

## Capítulo 3 GNU RADIO

---

GNU Radio es un conjunto de archivos y aplicaciones agrupadas en librerías, que permiten manipular señales mediante procesado digital. Mediante estas librerías se puede realizar el diseño de sistemas radio definidos por software si se conecta el ordenador a un SDR. También se puede trabajar con la tarjeta de audio, por ejemplo. GNU Radio corre sobre sistemas operativos (SO) GNU\Linux como Ubuntu. También es posible la instalación en Mac y Windows, algo más complejo en éste último SO utilizando la herramienta Cygwin, cuyo objetivo es portar software de sistemas POSIX<sup>7</sup> a Windows.

Los diseños realizados en GNU Radio se programan en Python. Python es un lenguaje de programación orientado a objetos interpretado, es decir, no se compila sino que el sistema operativo lo ejecuta directamente. Habitualmente se compara a TCL, Perl, Scheme, Java y Ruby. Cabe decir que, además de realizar diseños en Python, es posible utilizar la aplicación GNU Radio Companion. GNU Radio Companion es una herramienta que mediante una interfaz gráfica basada en bloques, similar a Simulink en Matlab, facilita la labor de diseño, creando los ficheros Python a partir del esquemático que se cree, tal y como se verá en apartados posteriores. De esta forma el usuario no tiene por qué saber Python.

Actualmente, Python se desarrolla como un proyecto de código abierto, administrado por la Python Software Foundation. Contiene módulos, clases, tipos de datos de alto nivel y escritura dinámica. Tiene interfaces para diversos sistemas y librerías. También puede utilizarse como un lenguaje de extensión para aplicaciones que necesitan una interfaz programable. Otra ventaja es su portabilidad, funcionando en sistemas Unix y derivados, Windows, Dos, Mac y otros.

Para construir un sistema de radio con GNU Radio se debe crear un grafo, donde los nodos

---

<sup>7</sup> **POSIX** es el acrónimo de Portable Operating System Interface; la X viene de UNIX como seña de identidad de la API. Son una familia de estándares de llamadas al sistema operativo definidos por el IEEE. Persiguen generalizar las interfaces de los sistemas operativos para que una misma aplicación pueda ejecutarse en distintas plataformas.

son bloques de procesado de señal y los enlaces entre nodos representan el flujo de datos entre ellos. Los bloques de procesado de señal son implementados en C++ y portados a Python a partir del programa SWIG, es decir, para implementar los diseños se crearán los módulos en Python que a su vez se apoya en los bloques implementados en C++. Las interacciones entre los diferentes niveles de GNU Radio aparecen en la siguiente figura:

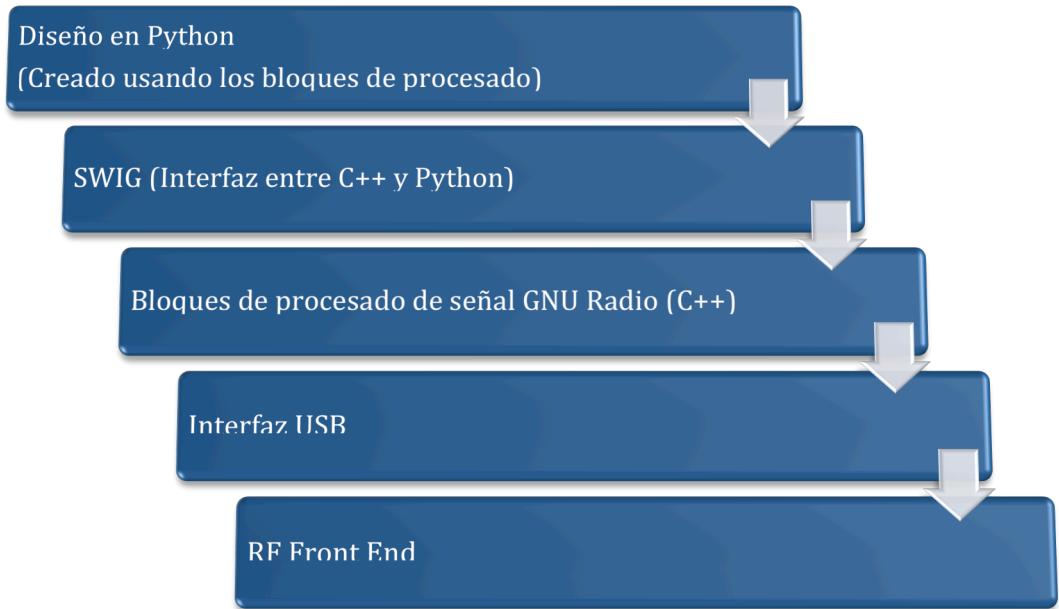


Figura 3-1 Arquitectura GNU Radio

Conceptualmente un bloque procesa señales, de forma continua, desde puertos de entrada hasta puertos de salida. Las partes de un bloque son el número de puerto de entrada, el número del puerto de salida, y el tipo de dato que fluirá de uno al otro. Los tipos de datos más comunes son: “*short*”, “*float*”, “*complex*” y “*byte*”.

Algunos bloques tienen únicamente puertos de salida o puertos de entrada. Existen así fuentes que leen datos de un archivo o de la FPGA a través de la interfaz USB, y señales que escriben datos a un archivo, a la FPGA, a la tarjeta de audio o a un display gráfico.

### 3.1 Instalación de GNU Radio

En los siguientes apartados se mostrará cómo llevar a cabo la instalación de GNU Radio (además de GNU Radio Companion) en Ubuntu y MAC de forma semiautomática. Si se quiere realizar una instalación manual, el proceso se detalla en el Apéndice A.

### 3.1.1 Instalación en Ubuntu

Para la instalación de GNU Radio en Ubuntu lo más inmediato es obtener el paquete *gnuradio* más actualizado desde Synaptic<sup>8</sup> que aúna todo el conjunto de aplicaciones y librerías de GNU Radio y del USRP necesarias. Para buscar el paquete GNU Radio seguimos los siguientes pasos:

1. *System → Administration → Synaptic Package Manager*
2. “*search*” → “*gnuradio 3.x*”
3. *Install*

Una vez realizado estos pasos, se pueden ejecutar los módulos Python desde un terminal. Pruebe a ejecutar el fichero *dial\_tone.py* de la carpeta de ejemplos GNU Radio ubicado en el directorio */usr/local/gnuradio/examples*. Para abrir la aplicación GNU Radio Companion, teclear el comando *grc*.

### 3.1.2 Instalación en MAC

La instalación en Mac es algo diferente pero igualmente sencilla. Para llevarla a cabo habrá que seguir el siguiente proceso:

1. Se requiere XCode Developer Tools para compilar los ejecutables en Mac OS X. Se puede descargar de <http://developer.apple.com/xcode/>. En el momento de la instalación, se puede optar por instalar sólo XCode y ahorrar así espacio en disco, pues toda la instalación ocupa unos 10 GB.
2. Instalar MacPorts. Con esta aplicación se conseguirán los paquetes necesarios para instalar GNU Radio. La podemos instalar de <http://www.macports.org/install.php>.
3. Ejecutar MacPorts para que se descargue GNU Radio. Se puede obtener GNU Radio y todas sus dependencias ejecutando el siguiente comando:

```
sudo port install gnuradio
```

4. Generar los ejecutables para la versión del SO en la máquina donde se instala con XCode.
5. Despues de la instalación, se debe configurar el PYTHONPATH adecuadamente para que al ejecutar módulos en Python busque las librerías necesarias en las carpetas donde se ha instalado GNU Radio. Para ello, abrir el fichero *.profile* que se encuentra en la carpeta de usuario y añadir:

```
export PYTHONPATH=$PYTHONPATH:/opt/local/lib/python2.6/site-packages
```

6. Despues de este paso ya se pueden ejecutar los ejemplos o cualquier diseño en un fichero Python (\*.py). Pruebe a buscar el módulo *dial\_tone.py* en el directorio

---

<sup>8</sup> **Synaptic** es un programa informático para el sistema de gestión de paquetes de Debian GNU/Linux.

*/opt/local/share/gnuradio/examples/audio* y ejecutarlo desde una terminal con el comando *sudo*, es decir:

```
cd /opt/local/share/gnuradio/examples/audio  
sudo dial_tone.py
```

7. Para conseguir que funcione GNU Radio Companion también se deberá añadir lo siguiente:

```
/opt/local/Library/Frameworks/Python.framework/Versions/2.6/lib/python2.6/site-packages/
```

8. El fichero *.profile* debe quedar finalmente con un formato parecido al siguiente:

```
# MacPorts Installer addition on 2011-04-06_at_00:10:12: adding an  
appropriate PATH variable for use with MacPorts.  
export PATH=/opt/local/bin:/opt/local/sbin:$PATH  
# Finished adapting your PATH environment variable for use with  
MacPorts.  
  
export PYTHONPATH=$PYTHONPATH:/opt/local/lib/python2.6/site-packages  
  
export  
PYTHONPATH=$PYTHONPATH:/opt/local/Library/Frameworks/Python.framework/Versions/2.6/lib/python2.6/site-packages/
```

9. Si se quiere ejecutar la aplicación GNU Radio Companion, teclear el comando *gnuradio-companion*. Nótese que el comando no es el mismo que en Ubuntu.

## 3.2 Introducción al diseño de módulos en GNU Radio

La creación de módulos en Python puede llevarse a cabo de dos formas: bien programando directamente en Python o bien utilizando la herramienta GNU Radio Companion. La programación en Python es ardua, mientras que el diseño basado en GNU Radio Companion es sencillo. De ahí que los diseños de este documento se centren en esta herramienta. El lector interesado en la programación en Python encontrará de interés el Apéndice C, donde se introduce a la programación en este lenguaje.

GNU Radio Companion es una herramienta gráfica para realizar un diseño a partir de bloques incluidos en librerías o creados por el usuario [13]. Una vez realizado el diseño, el GNU Radio Companion genera el código en Python asociado, que bien puede ejecutarse desde la propia herramienta o desde la línea de comandos en un terminal. Para aquellos que sean usuarios de Matlab, cabe decir que la filosofía de funcionamiento es similar a la de Simulink. El GNU Radio Companion permite diseñar sistemas que utilicen un SDR como el USRP, pero también permite trabajar con otras señales, como por ejemplo las señales de audio del PC/Workstation.

Para iniciarnos en la programación de Software Radio con GNU Radio Companion es necesario conocer y saber cómo funcionan las estructuras de grafos, bloques y conexiones. Para realizar esta tarea se comenzará con un programa básico en Python, éste programa es *dial\_tone.py*, el cual se analiza detalladamente en el Anexo III. En Ubuntu, este módulo se

encuentra en el siguiente directorio: `gr-build/gnuradio-examples/python/audio` si optamos por la instalación manual, o bien en `/usr/local/gnuradio/examples` si se instaló el paquete completo `gnu-radio 3.X` desde Synaptic. En Mac, el directorio donde se encuentra este fichero Python sería: `/opt/local/share/gnuradio/examples/audio`. Este programa genera dos señales sinusoidales de diferente frecuencia que simulan el tono de llamada en un teléfono local y los envía a la tarjeta de sonido del ordenador, por lo que el USRP no es necesario aquí. En el siguiente apartado se mostrará la construcción del módulo `dial_tone.py` a partir de GNU Radio Companion y se verá lo intuitivo que resulta el diseño de módulos con esta herramienta. Para abrir esta aplicación, basta con teclear en un terminal la sentencia `grc` y se abrirá dicho programa en una distribución LINUX o `gnuradio-companion` si se utiliza MAC.

### 3.2.1 GNU Radio Companion

La construcción de módulos utilizando esta herramienta gráfica se basa en añadir bloques e interconectarlos de manera esquemática<sup>9</sup>. Los bloques utilizados se pueden agrupar de la siguiente manera:

1. **Sources** (fuentes): Estos bloques especifican cualquier tipo de fuente como por ejemplo un archivo de audio wav, un generador de señal de forma arbitraria y con las características que se requieran, una fuente binaria aleatoria, un fichero de cualquier formato, un micrófono, el propio USRP, ...etc.
2. **Bloques de procesado de señal**: A este grupo pertenecen todos aquellos bloques que realizan un tratamiento de la señal de cualquier tipo. Se pueden citar como ejemplos los moduladores, filtros, remuestreadores, multiplicadores, amplificadores en software, etc.
3. **Sinks** (sumideros): La señal tendrá un destino final como bien puede ser un fichero de cualquier formato, la tarjeta de sonido o el USRP. Indicar que a este conjunto también pertenecen los bloques de visualización de señales (*Graphical sinks*) como *FFT sink* (para visualizar la FFT de la señal en un punto), *Constellation sink* u *Oscilloscope sink* (para representar la señal en tiempo en cualquier lugar del diseño) entre otros.

Una vez se incluyan en el esquemático todos los bloques que se consideren oportunos, lo único que queda por hacer es interconectarlos. En el ejemplo que se va a mostrar en este apartado se necesitan dos fuentes, un bloque de adición señales y el sumidero que será la tarjeta de sonido, además de los bloques de visualización que mostrarán el espectro de la señal de audio y la forma de onda que se envía al dispositivo de sonido. La configuración quedaría Figura 3-3.

Como se puede ver en la figura, además de los bloques mencionados se añaden una serie de variables para controlar los parámetros desde la interfaz gráfica (en este caso Variable Slider, aunque hay otras opciones), de tal manera que en el parámetro frecuencia de la primera fuente se le asigna el identificador `f1` y al ejecutar el esquemático se podrá controlar dicha frecuencia. Igual sucede con el resto de parámetros. Una vez se tenga el diseño completado, éste se

---

<sup>9</sup> La programación directa en Python sigue la misma filosofía, crear los objetos de cada bloque de procesado de señal e interconectarlos, aunque desde una interfaz menos amigable.

guardará en un fichero \*.grc que al compilarlo (simplemente con pulsar F6 dentro de la aplicación) genera un fichero \*.py que se puede ejecutar para visualizar la **Error! No se encuentra el origen de la referencia.**

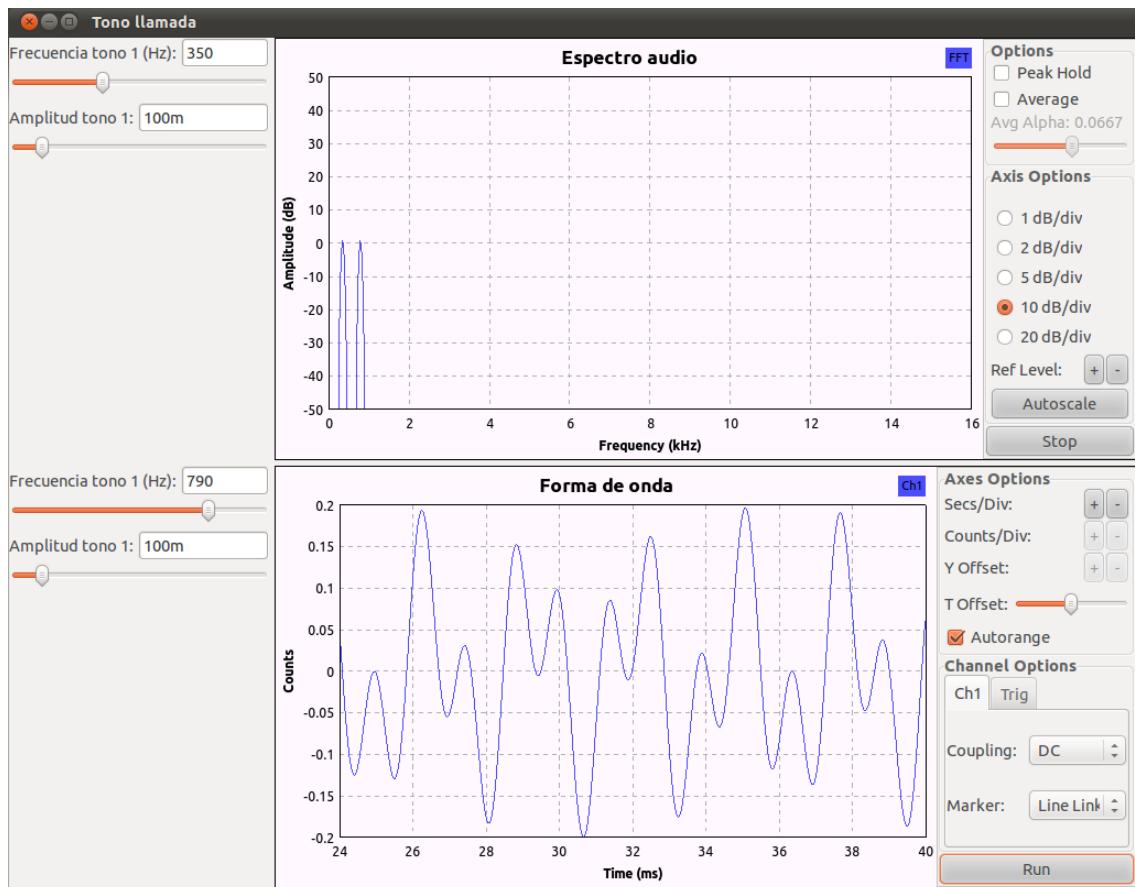


Figura 3-2 Visualización del módulo dial\_tone.py utilizando GNU Radio Companion

Ahora se puede visualizar el espectro y la forma de onda y su modificación al variar los parámetros creados con las *Variables Slidder* a la izquierda.

Siguiendo el mismo proceso se pueden llevar a cabo infinitas aplicaciones teniendo en cuenta que existen bloques para cualquier procesado de señal, e incluso bloques moduladores y demoduladores completos, simplemente añadiendo bloques y configurando todo adecuadamente, a partir de una interfaz gráfica intuitiva que ahorra el trabajo de programar directamente en Python.

Hasta este punto, se ha introducido la SDR, el software GNU Radio y la herramienta GNU Radio Companion y se ha realizado un análisis detallado del USRP. De aquí en adelante, se incluirán diversos ejemplos de diseño de sistemas con estas herramientas.

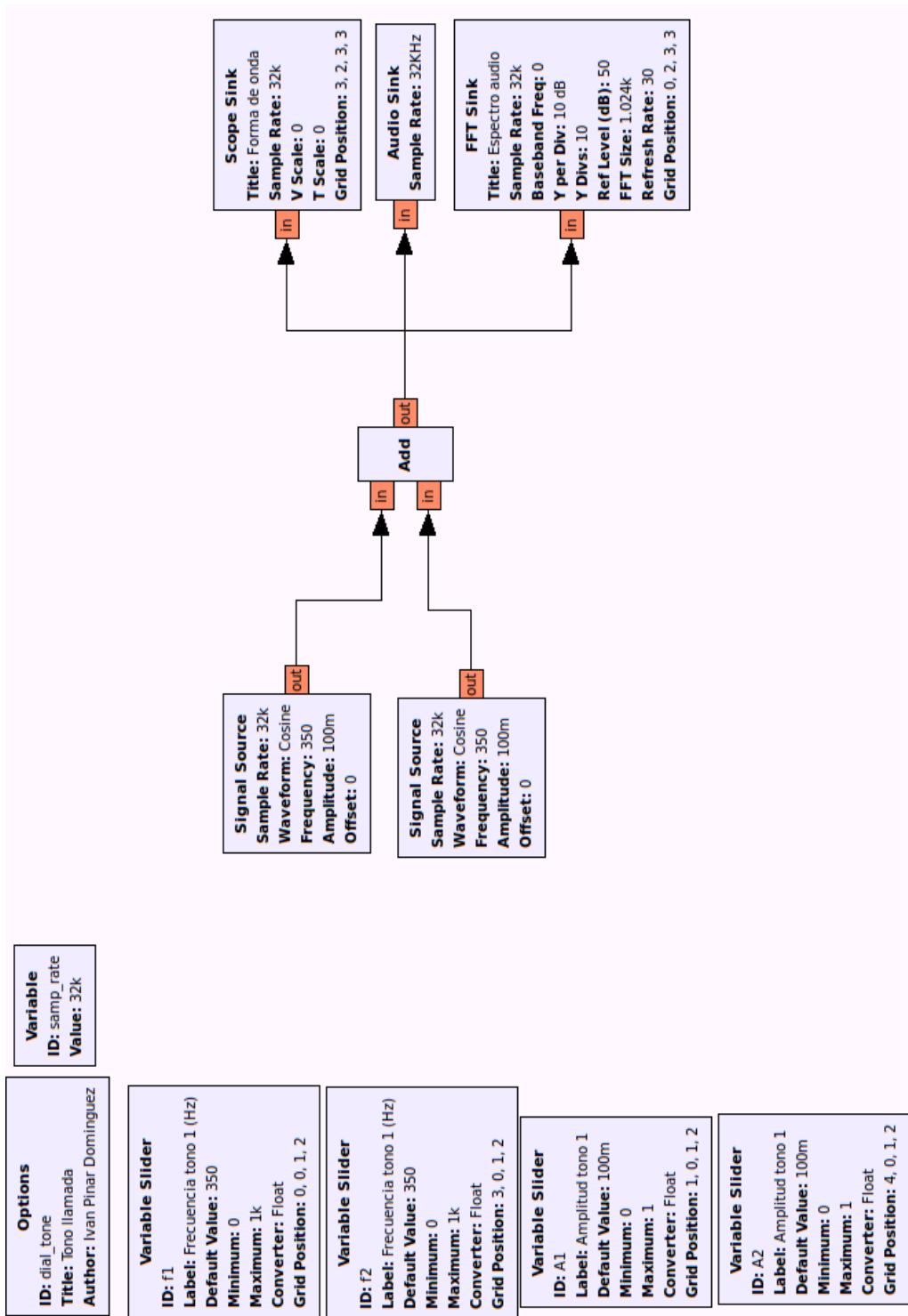


Figura 3-3 Implementación dial\_tone en GNU Radio Companion

## **Capítulo 4 DISEÑOS EN GNU RADIO COMPANION**

---

En este capítulo se mostrarán un conjunto de módulos creados con la herramienta GNU Radio Companion basados en el USRP. En primer lugar se explicará el diseño de un analizador de espectros y seguidamente de un sistema FM tanto transmisor como receptor. Tras esto se pasará a la realización de esquemas de modulación digitales incluyendo entre otras, DQPSK, QPSK, GMSK u OFDM.

Los diseños se han probado en configuraciones que constan de transmisor y receptor, utilizándose los siguientes equipos:

- 2 equipos portátiles con sistema operativo Ubuntu 9.04 en uno y Ubuntu 10.10 en otro<sup>10</sup>.
- 2 USRP en cuyo interior se aloja una tarjeta transceptor RFX-2400 en cada uno para la conexión de los sistemas digitales y una tarjeta WBX en uno y TVRX (que es sólo receptora) en otro para la comunicación vía FM.

La disposición de los equipos durante las pruebas experimentales se puede apreciar en la Figura 4-1, y las daughterboards utilizadas en la Figura 4-2.

### **4.1 Analizador de espectros**

En este primer diseño se llevará a cabo la construcción de un analizador de espectros. La elaboración de este módulo es bien sencilla, basta con añadir al esquemático una fuente USRP y conectarla al sumidero gráfico FFT con la consiguiente configuración que se explicará a continuación.

Se ha realizado el analizador de espectros adecuado a la tarjeta receptora TVRX, cuyo rango de frecuencias es de 50 a 860 MHz. Para realizar el diseño con otra tarjeta simplemente

---

<sup>10</sup> de ahí la pequeña diferencia en la interfaz gráfica de al ejecutar los diseños como se verá posteriormente

se debe ajustar en el esquemático la variable que define la frecuencia de la fuente USRP. El esquemático implementado aparece en la Figura 4-3.



Figura 4-1 Disposición de los equipos durante las pruebas



Figura 4-2 Tarjetas secundarias: WBX y RFX2400 (izquierda) y TVRX y RFX2400 (derecha)

A continuación se irá explicando cada uno de los elementos que aparecen en el diagrama de la Figura 4-3 en una serie de pasos:

**PASO 1) Definición de la fuente:** En este caso, la señal será la que proporcione el USRP a través de la interfaz USB. La configuración de este bloque se muestra en la Figura 4-4.

Como se aprecia, la fuente USRP es de tipo complejo a 16 bits por cada componente compleja (es decir, 4 bytes por muestra). El factor de diezmado se elegirá en función del ancho de banda que se quiera visualizar. Si por ejemplo se requiere un ancho de 2 MHz, se necesitará un factor de diezmado de 32, de esta manera la tasa de muestras es  $64 \text{ MS/s}/32 = 2 \text{ MS/s}$  y la representación, al tratarse de muestras complejas va desde  $-f_s/2$  hasta  $f_s/2$  centrada en la frecuencia sintonizada, donde  $f_s$  es la tasa de muestras, es decir, 2 MHz. A continuación se muestra una tabla con los valores de anchos de banda y diezmados asociados ( $BW = 64 \text{ MS/s} / \text{diezmado}$ ):

Tabla 4-1 Relación entre ancho de banda y factor de diezmado

Ancho de banda	Factor de diezmado
320 kHz	200
640 kHz	100
1 MHz	64
2 MHz	32
4 MHz	16
8 MHz	8

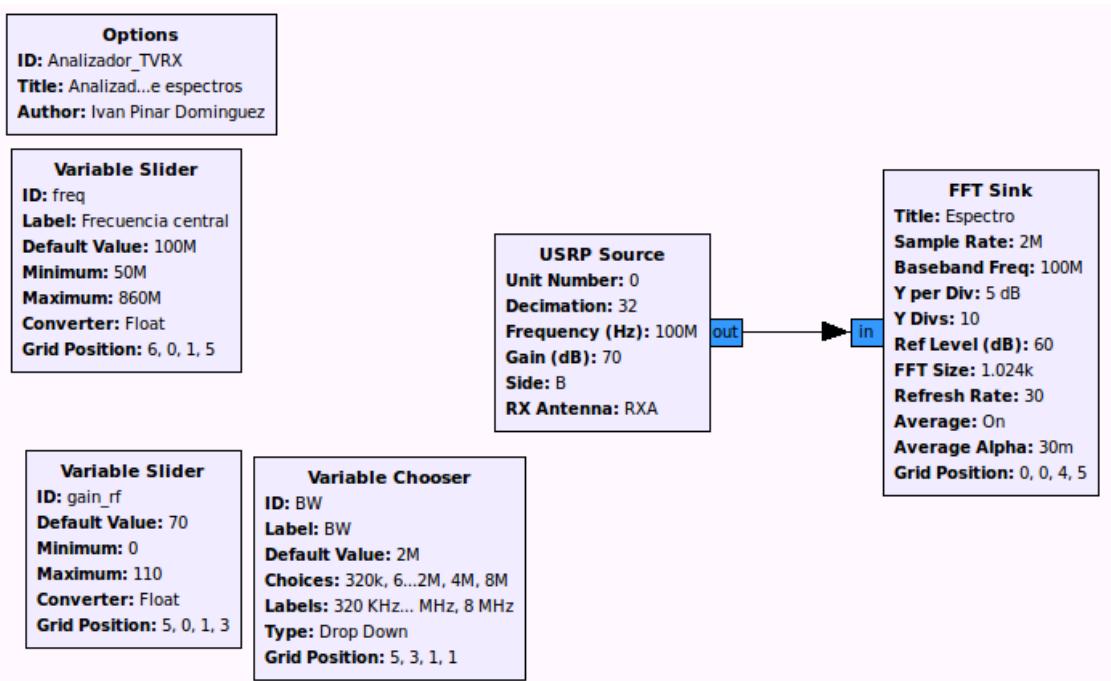


Figura 4-3 Esquemático del analizador de espectros en GNU Radio Companion

Para seleccionar la frecuencia, se indicará el identificador *freq* que corresponde a una *Variable Slider* que se explicará posteriormente. Igualmente ocurre con la ganancia. Por último se elige el lado o *Side* en el que se encuentra la tarjeta secundaria (o daughterboard) en el USRP, en este caso el lado B y receptor de antena A (es indiferente en este diseño porque la tarjeta TVRX sólo tiene una conexión de antena). Se recuerda en este punto que el USRP admite una o dos tarjetas secundarias (lado A y lado B) y cada una de estas tarjetas puede ser transmisor, receptor o transceptor (TRX). En caso de que sea TRX, lo común es que tenga un conector sólo para recepción (RX) y otro para entrada/salida (TX/RX).

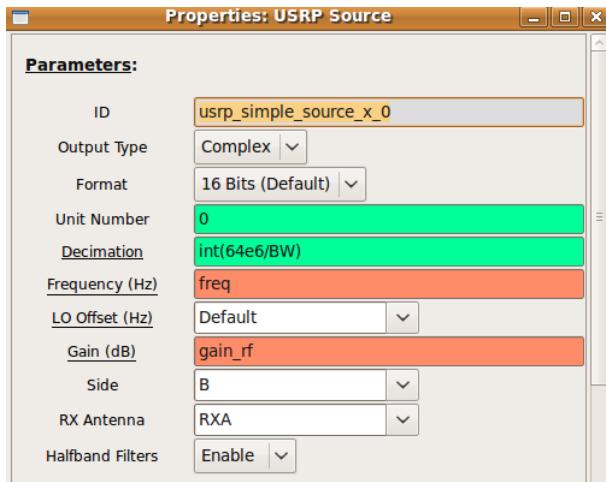


Figura 4-4 Propiedades de la fuente USRP

**PASO 2) Selección del sumidero:** El sumidero del flujo de señal será un sumidero gráfico de tipo FFT\_sink. Su configuración aparece a continuación:

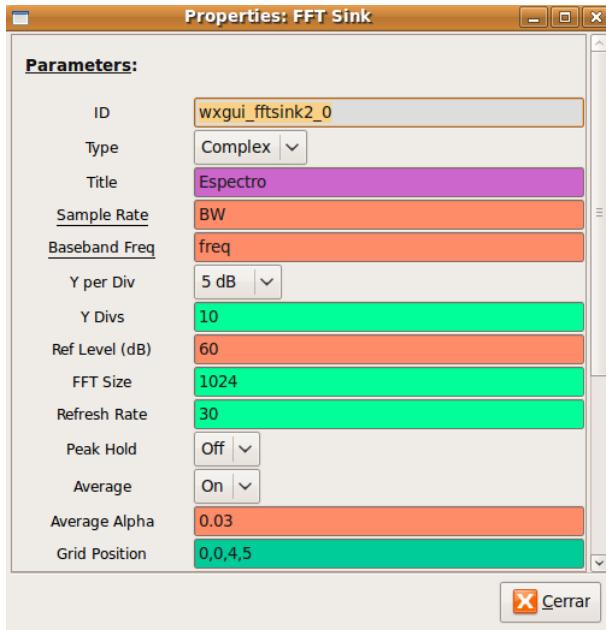


Figura 4-5 Propiedades del sumidero FFT

Este sumidero es de tipo complejo. Obviamente, los extremos de una conexión tienen que ser del mismo tipo. Se especifica el título en la casilla *Title* y se asigna a *Sample Rate* el identificador de la variable *BW* que se creará posteriormente. Para que la representación se centre en la frecuencia sintonizada, se indica el identificador *freq* (el de la *Variable Slider*

que se moverá en la interfaz gráfica). El resto de parámetros están asociados a la visualización: escala, número de puntos de la FFT, tasa de refresco,...

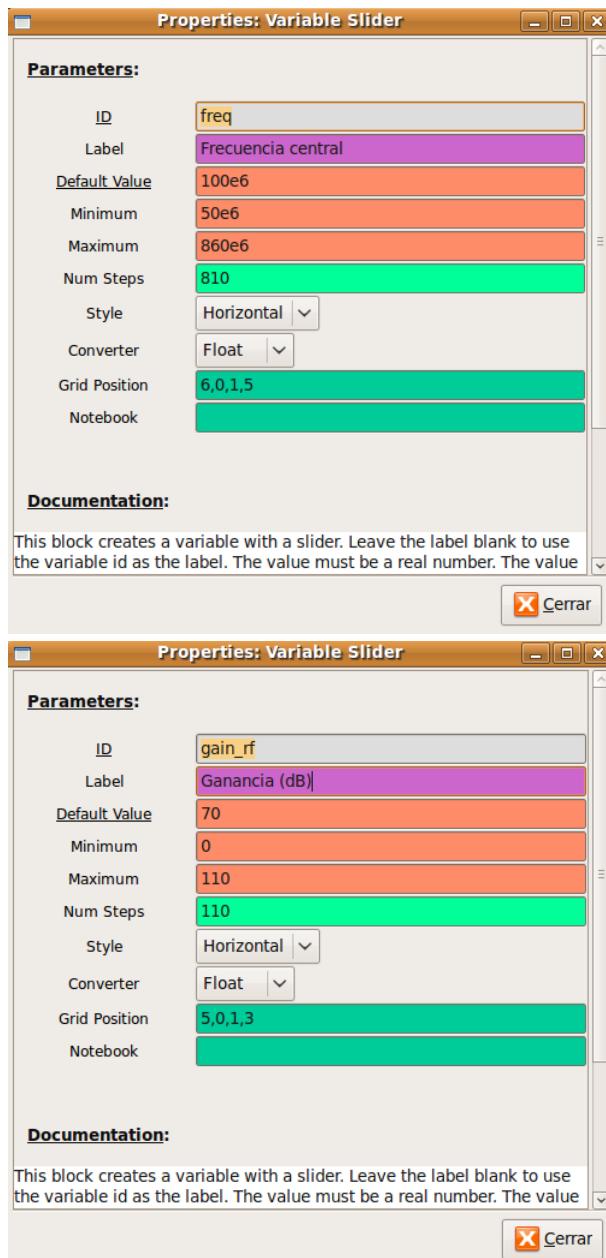


Figura 4-6 Parámetros de las Variables Slidder

**PASO 3) Inserción de variables y opciones:** En este paso se explicará el proceso para crear variables que posteriormente se puedan manejar desde la interfaz gráfica al ejecutar el diseño así como las opciones del gráfico. Empezando por las *Variables Slidder*.

Como se aprecia en la Figura 4-6, lo primero es indicar el identificador o nombre de la variable que se asocia a ellas para identificarlas en el resto del esquemático. Después se selecciona la etiqueta que se mostrará en la interfaz gráfica, el rango de valores posibles y el estilo.

En el caso de la variable frecuencia, en el mínimo y el máximo se seleccionan los extremos del rango de funcionamiento de la tarjeta TVRX así como el número de saltos para que el paso sea de 1 MHz. En la variable ganancia sucede lo mismo, la ganancia máxima será de 110 dB, 90 dB de la tarjeta más 20 dB del PGA. La ganancia especificada primero se traslada al PGA y si se selecciona un valor mayor que 20 dB entonces se aumenta la ganancia de la daughterboard.

La última variable definida en el esquemático es del tipo *Variable Chooser* para ajustar el ancho de banda. Se ha elegido este tipo porque el factor de diezmado no puede tomar cualquier valor (sólo pares entre 8 y 512) y, por temas de eficiencia, potencias de 2 (para algoritmos como FFT). Sus propiedades aparecen en la siguiente figura:

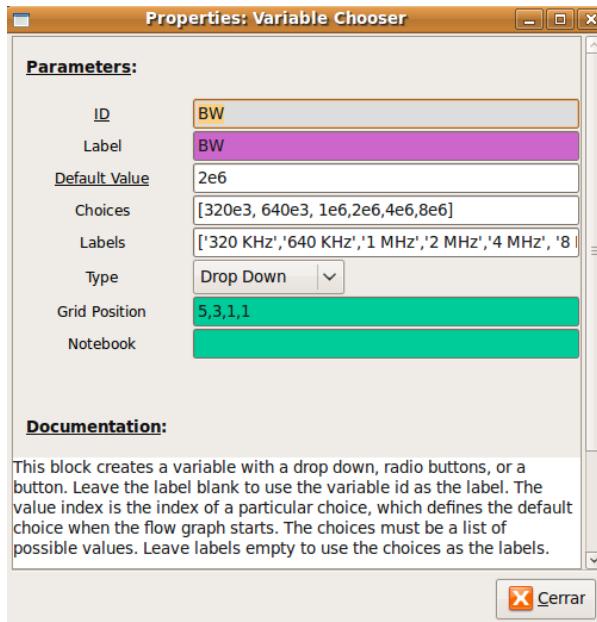


Figura 4-7 Parámetros de la Variable Chooser

Su configuración es similar a las variables *slider* pero en este caso la variable sólo puede tener un conjunto de valores: 320 kHz, 640 kHz, 1 MHz, 2 MHz, 4 MHz y 8 MHz. A estos valores les corresponden las etiquetas seleccionadas en el parámetro *Labels*.

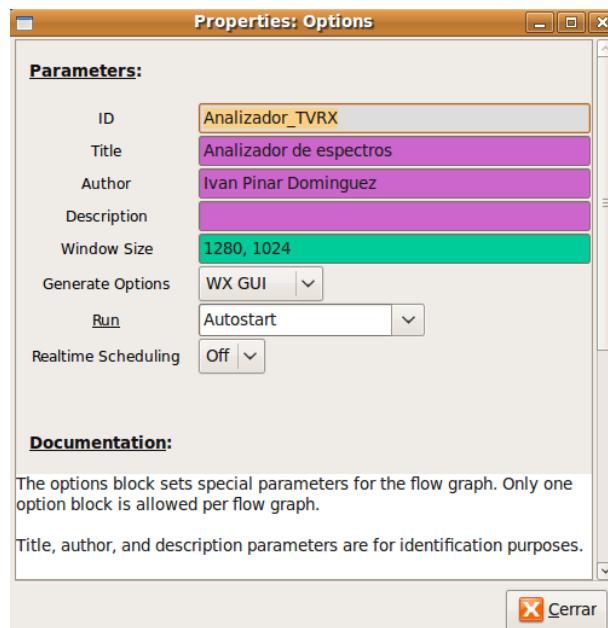


Figura 4-8 Parámetros del bloque Options

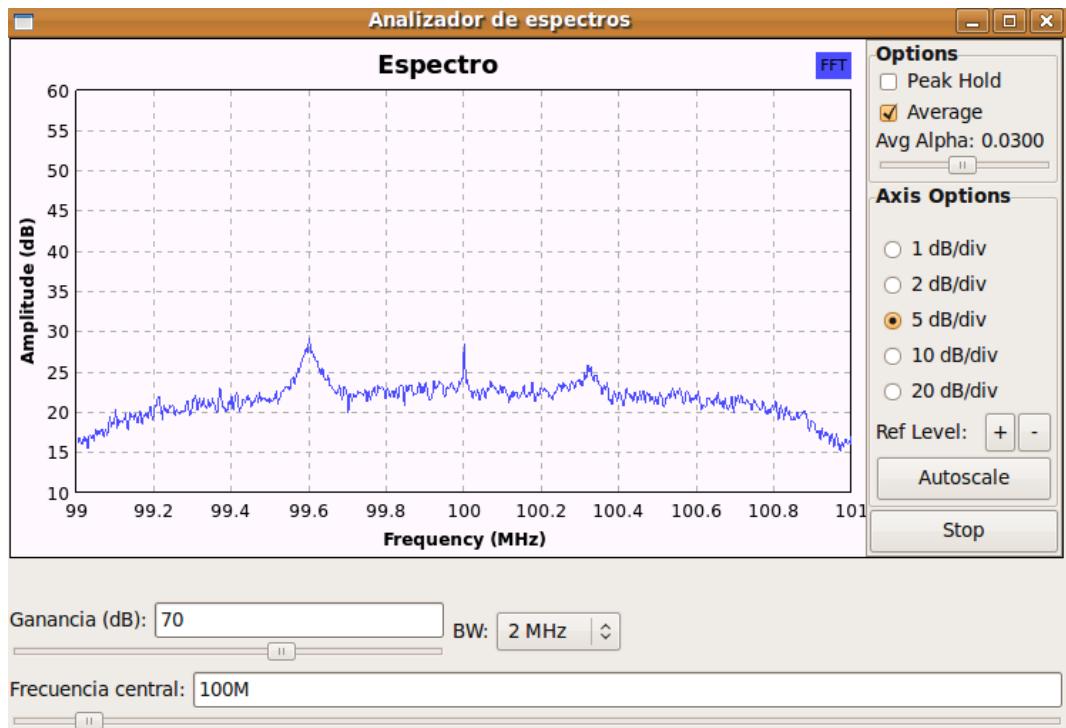


Figura 4-9 Ejecución del analizador de espectros creado

El único bloque que falta por comentar es el de opciones. En éste, se selecciona el nombre que tendrá el fichero *.py* que se genera una vez se compile el esquemático, el título de la interfaz gráfica, el autor y algún otro detalle de visualización, tal como se observa en la Figura 4-8.

Para ajustar todos los elementos creados en la interfaz gráfica en la posición y tamaño que se desee, se utiliza el parámetro *Grid Position*, que se puede apreciar en todos los bloques (menos el de opciones, pues no aparece en la interfaz). La nomenclatura es:

#### Fila, Columna, Alto (en número de filas), Ancho (en número de columnas)

Tras haber realizado el diseño, presionamos la tecla *F5* para compilar y *F6* para ejecutar el fichero Python que se genera automáticamente, o se puede recurrir a los menús del entorno. El resultado aparece en la Figura 4-9.

En este rango frecuencial se observan emisiones de FM a 99.6 MHz y 100.3 MHz. Si se reduce el ancho de banda y se selecciona la frecuencia de sintonía de 97 MHz (recordar que la variable *freq* que sintoniza la frecuencia se definió con un paso de 1 MHz) podemos ver la emisión a 97.1 MHz (40 principales en Sevilla).

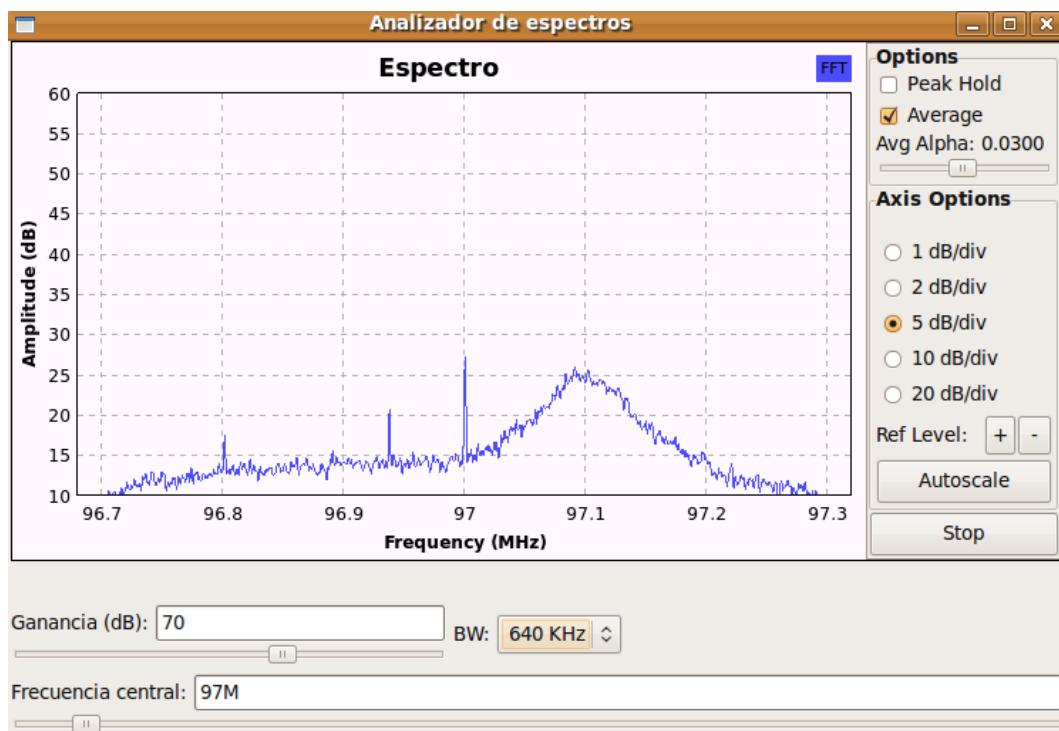


Figura 4-10 Visualización de una emisión FM a 97.1 MHz

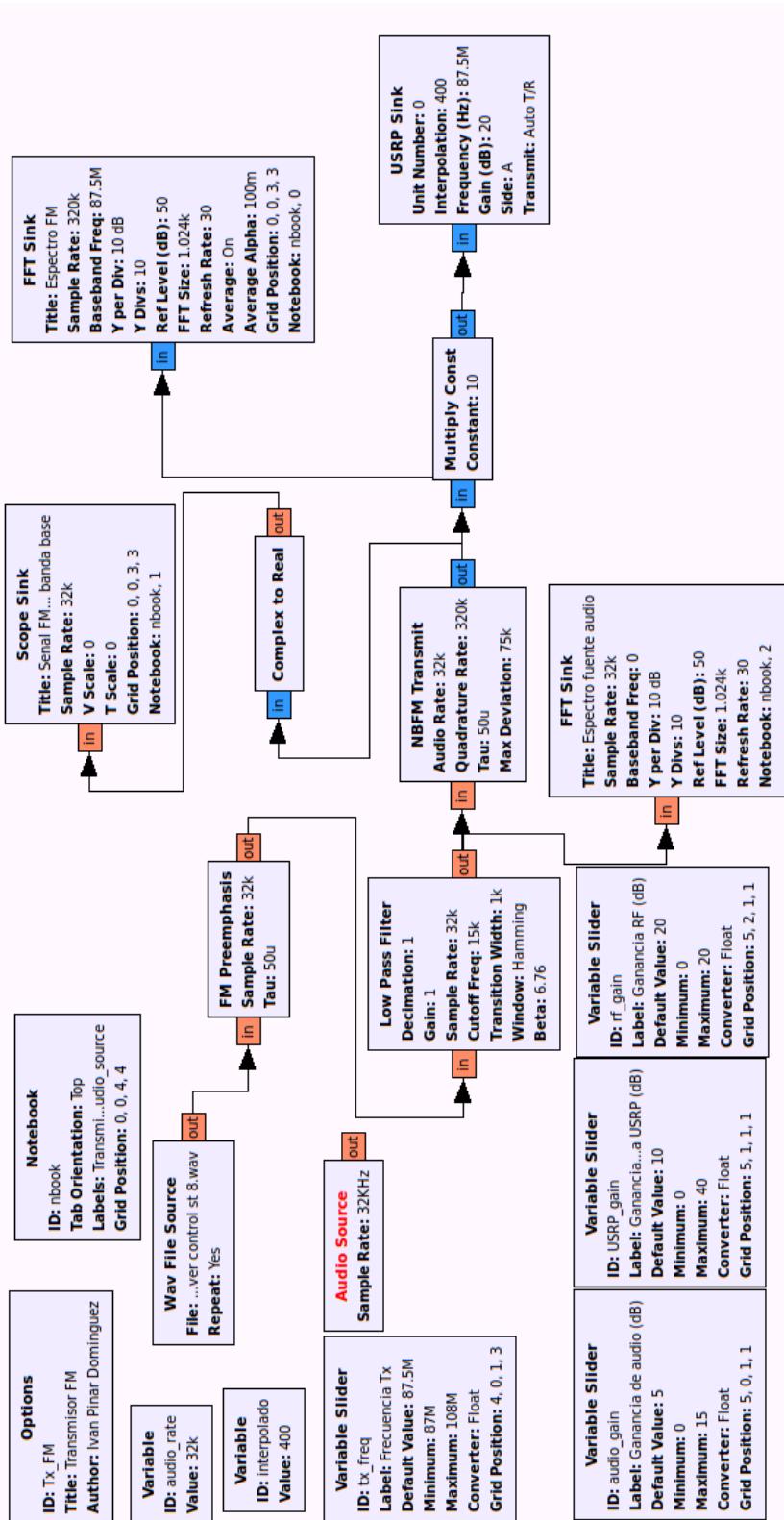


Figura 4-11 Esquemático del transmisor FM

Al igual que se ha implementado este analizador de espectros, se puede realizar el diseño de un osciloscopio sin más que sustituir el bloque *FFT\_sink* del esquemático por un bloque *Scope\_sink*. En los siguientes apartados aparecerá este bloque para visualizar la señal en el dominio temporal, o incluso obtener la constelación activando el modo XY útil para los sistemas de comunicaciones digitales.

Por si resulta de interés, también existe la opción de crear spectrogramas, es decir, visualizaciones tiempo-frecuencia-amplitud aunque suelen tener una elevada carga computacional.

## 4.1 Sistema FM

Este apartado se centrará en la realización de un sistema FM, tanto transmisor como receptor, a partir de las librerías adecuadas de GNU Radio en las que se implementan los bloques modulador y demodulador de FM. Si se quiere observar la implementación de dichos bloques basta con irse al directorio *usr/lib/python2.6/dist-packages/gnuradio/blks2impl* en Ubuntu o bien al directorio */opt/local/share/gnuradio/blks2impl* en Mac y abrir los ficheros de modulador y demodulador FM (*nbfm.py*). Una primera idea para realizar este diseño se obtuvo de [8].

### 4.1.1 Transmisor FM

En primer lugar se mostrará el esquemático creado para el módulo transmisor. Posteriormente se irán comentando los detalles de este diseño que tengan relevancia y se diferencien de los conceptos introducidos en el analizador de espectros. El fichero *grc* creado aparece en la Figura 4-11.

Las variables utilizadas en este diseño son las siguientes:

1. *audio\_rate*: Tasa de muestreo de audio.
2. *tx\_freq*: Frecuencia de transmisión utilizada en el USRP.
3. *interpolado*: Factor de interpolado introducido en el USRP antes del DAC.
4. *audio\_gain*: Ganancia del filtro paso bajo tras la fuente de audio.
5. *USRP\_gain*: Ganancia del amplificador software colocado antes de USRP.
6. *rf\_gain*: Ganancia de RF del USRP considerando el PGA y la tarjeta secundaria.
7. *nbook*: Utilizada para crear pestañas.

Estas variables se irán introduciendo en lo sucesivo. El módulo consta de dos posibles **fuentes** a seleccionar, una de tipo audio que se refiere al micrófono del equipo y otra a un fichero *wav* cuyas características se muestran en la Figura 4-12.

Como se puede apreciar, las especificaciones de esta fuente son la ruta del fichero, la opción de repetición y el número de canales (mono, estéreo o mayor número de canales).

A continuación se introduce un **bloque de preénfasis** de la señal (que en recepción se compensará con un bloque de deénfasis) para disminuir el ruido de alta frecuencia que se genera intrínsecamente en la demodulación en el sistema FM.

Tras esto, se ha añadido un **filtro paso bajo** para limitar el espectro de audio a 15 kHz en caso de que el fichero *wav* tenga mayor ancho y además dar la posibilidad de introducir

ganancia de audio en el fichero a través de la variable *slidder audio\_gain*. Las propiedades de este filtro aparecen en la Figura 4-13.

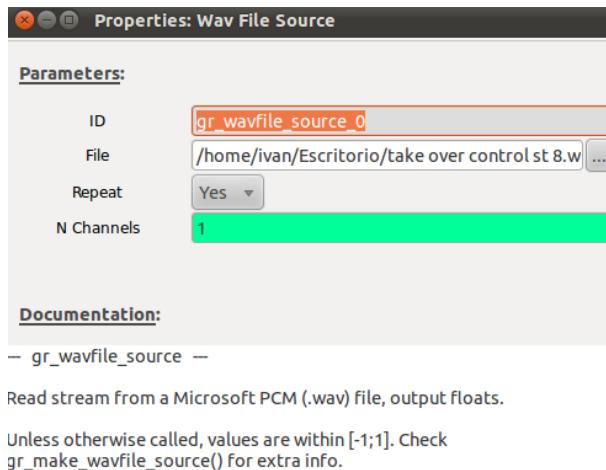


Figura 4-12 Propiedades *wav source*

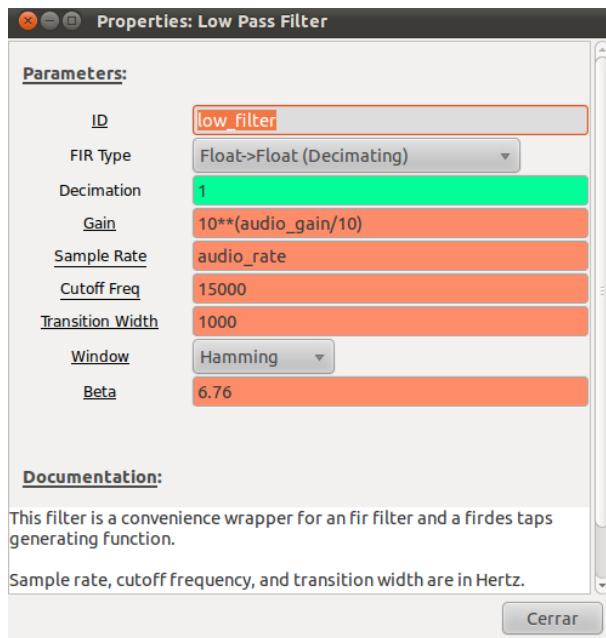


Figura 4-13 Propiedades del filtro paso bajo

El siguiente bloque que aparece en el diseño es el **modulador de FM** de banda estrecha cuyas propiedades son las siguientes:

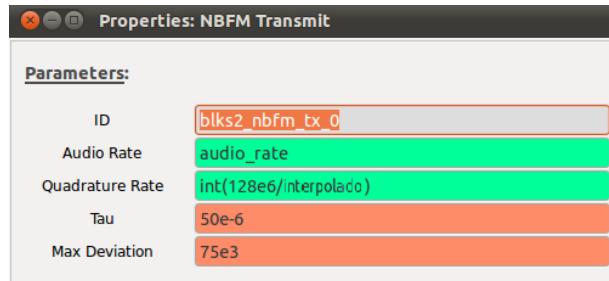


Figura 4-14 Propiedades del bloque modulador FM

Los parámetros son la tasa de audio de entrada (la del fichero *wav*, es decir, 32 kHz), tasa de cuadratura (la tasa de muestras por la interfaz USB), es decir, la tasa a la que van las muestras en la interfaz USB, la constante de tiempo  $\tau$  (50  $\mu$ s en Europa) y la desviación máxima (75 kHz en el estándar europeo).

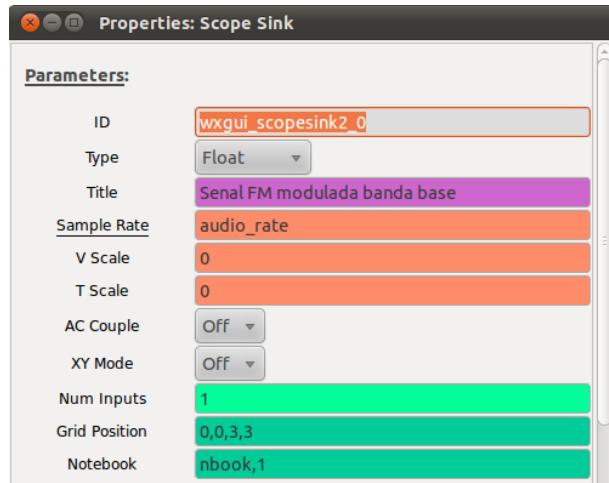


Figura 4-15 Propiedades del osciloscopio

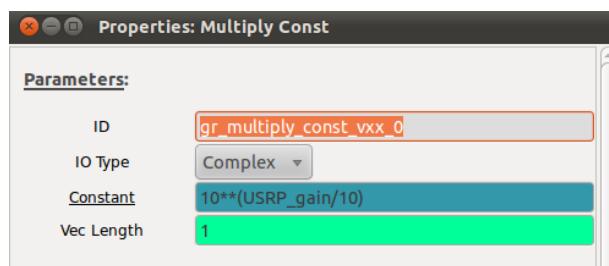


Figura 4-16 Propiedades del amplificador software

En este punto resulta conveniente aclarar la selección del factor de interpolado y diezmado:

- La desviación máxima es 75 kHz, por lo que aplicando la *regla de Carson*:

$$B = 2(\Delta f + f_m) = 2(75\text{kHz} + 15\text{kHz}) = 180\text{kHz} \quad (4-1)$$

En la práctica se escoge una canalización de 200 kHz. El ancho de banda de transmisión y recepción vendrá dado por el factor de interpolado y diezmado respectivamente. Se ha seleccionado un valor de interpolado de 400, por lo que el ancho de banda será de  $128 \text{ MHz} / 400 = 320 \text{ kHz}$  (hay que tener en cuenta que se trata de muestras complejas, por lo que el ancho de banda va desde  $-f_s/2$  hasta  $f_s/2$  que después elevaremos en frecuencia) lo cual es suficiente para la canalización de FM. En recepción el ADC opera a la mitad de tasa por lo que el factor de diezmado será de 200 para obtener el mismo ancho de banda (o tasa de muestras como quiera verse).

A la salida del modulador se conecta un **sumidero gráfico Scope**, es decir, un osciloscopio. Los parámetros de configuración de este bloque se muestran en la Figura 4-15.

La configuración de este bloque es similar al resto pero con las características propias de un osciloscopio como por ejemplo la escala de tiempo y amplitud, que se podría ajustar convenientemente o bien dejarse a 0 para un ajuste automático. Antes de la conexión del osciloscopio se ha añadido un bloque *complex\_to\_real* para ver sólo la componente real de nuestra señal.

Siguiendo en el esquemático, el siguiente bloque es un **multiplicador** (es decir, un amplificador en software) para aumentar la potencia de la señal transmitida a partir de la variable *slider* creada *USRP\_gain*. Sus propiedades se incluyen en la Figura 4-16.

Por último en este módulo, se añade el **sumidero USRP**, por lo que el USRP será donde acabe el flujo, ya que estamos en la parte transmisora. La configuración de este bloque es la siguiente:

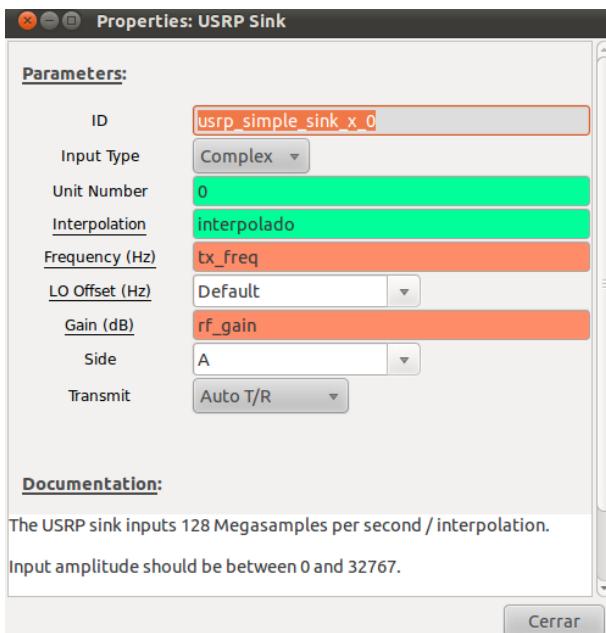


Figura 4-17 Propiedades del sumidero USRP

Las propiedades son similares a las vistas en otros bloques. La única peculiaridad a resaltar es el parámetro *Transmit* que se ha especificado como *Auto T/R*, es decir, se podría transmitir y recibir al mismo tiempo con la misma tarjeta.

Antes de mostrar los resultados se va a explicar cómo crear varias pestañas dentro de una misma ventana para seleccionar cada una de las visualizaciones adecuadamente. Para esto, es necesario haber incluido un objeto o bloque de tipo *Notebook* en el esquemático. Cada uno de los objetos de la interfaz, como visualizaciones o variables, se pueden insertar en una de las pestañas del Notebook y sólo aparecen cuando se selecciona dicha pestaña. Si no se especifica nada en el parámetro *Notebook* de cada objeto, entonces aparecerá independientemente de la pestaña seleccionada. Indicar que si un objeto se inserta dentro de una pestaña del notebook, el *grid position* de ese objeto está referido dentro de esa misma pestaña, no de la visualización global. En el esquemático se puede ver como las visualizaciones de FFT y osciloscopio pertenecen a pestañas diferentes del Notebook (cuyo identificador es *nbook,x* donde *x* es la pestaña). Los parámetros del bloque Notebook aparecen en la Figura 4-18.

Así, para insertar la FFT del espectro modulado en la pestaña Transmisor, se deberá seleccionar *nbook,0* en el parámetro *Notebook* de la visualización *FFT\_sink* correspondiente.

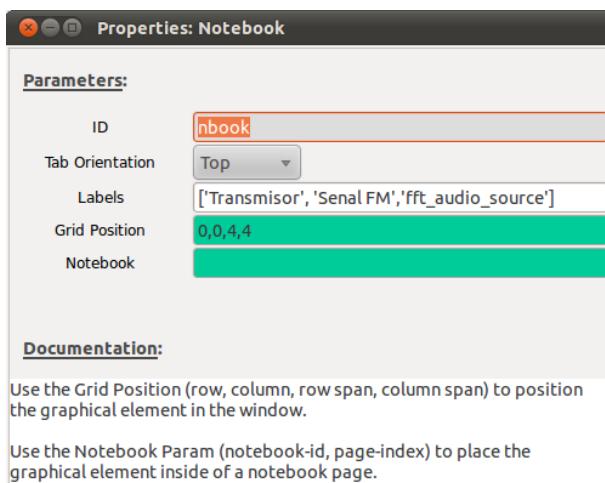


Figura 4-18 Propiedades Notebook

Tras obtener el fichero Python a partir del esquemático y ejecutarlo, la interfaz gráfica es la de la Figura 4-19. Se puede apreciar como la emisión del transmisor FM ocupa un ancho de banda de unos 200 kHz como cabía esperar. También se puede visualizar la señal en el tiempo gracias al bloque osciloscopio añadido previamente, al que se le asignó el parámetro *notebook* a 1 y por eso aparece en la segunda pestaña. El resultado es el incluido en la Figura 4-20.

En la Figura 4-20 se observa como la información viaja en la frecuencia de la señal modulada y no en la amplitud.

La frecuencia seleccionada para el transmisor FM es de 87.5 MHz para no interferir con ninguna otra emisión. Para más información sobre dispositivos de corto alcance, consultar [11]. Indicar también que la ganancia de RF no es seleccionable en la tarjeta WBX (aunque la

ganancia del PGA sí que lo es). Para paliar esta limitación se puede aumentar la potencia de la señal gracias a la ganancia del amplificador software (*Ganancia entrada USRP*) que puede tomar valores entre 0 y 40 dB, aunque también depende del nivel de la señal de audio y la ganancia que se haya introducido a éste (*Ganancia de audio*).

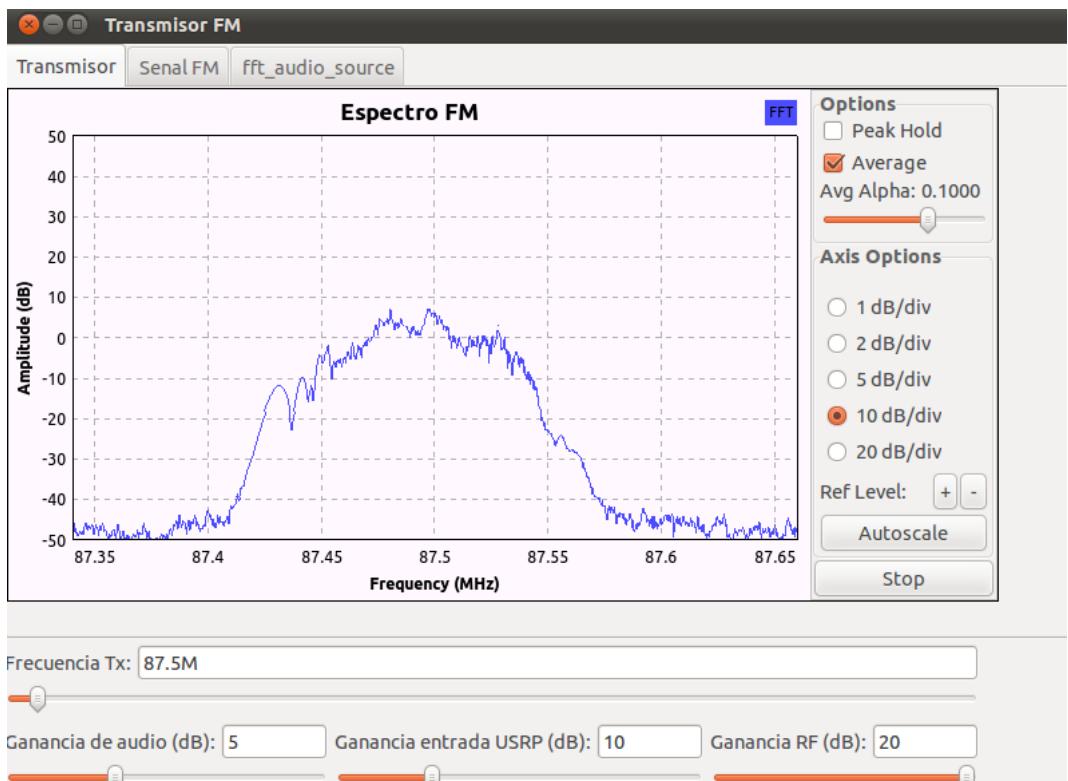


Figura 4-19 Visualización del espectro transmitido

#### 4.1.1 Receptor FM

Una vez analizado en detalle el módulo transmisor, se procederá al análisis del módulo receptor. Este diseño es similar al del transmisor pero en orden inverso, obviamente. El esquemático creado se muestra en la Figura 4-21.

Las variables utilizadas en el esquemático se incluyen a continuación:

- *audio\_rate*: Tasa de muestreo de audio.
- *rx\_freq*: Frecuencia sintonizada en el USRP (es la suma de *usrp\_freq+xlate\_tune*, ajuste grueso y fino respectivamente).
- *usrp\_decim*: Factor de diezmado introducido en el USRP.
- *usrp\_rate*: Tasa de muestras que proporciona el USRP (será 64 MS/s/*usrp\_decim*).

- *af\_gain*: Ganancia de audio antes de enviar a la tarjeta de sonido.
- *rf\_gain*: Ganancia de RF del USRP considerando el PGA y la tarjeta secundaria.
- *nbook*: Utilizada para crear pestañas.
- *Display\_selector*: Variable utilizada para mostrar el espectro en banda base o en radiofrecuencia.
- *sql\_level*: Nivel de squelch para minimización del ruido.

Se comenzará la explicación de este diseño por la **fuente**, es decir, la información que suministra el USRP.

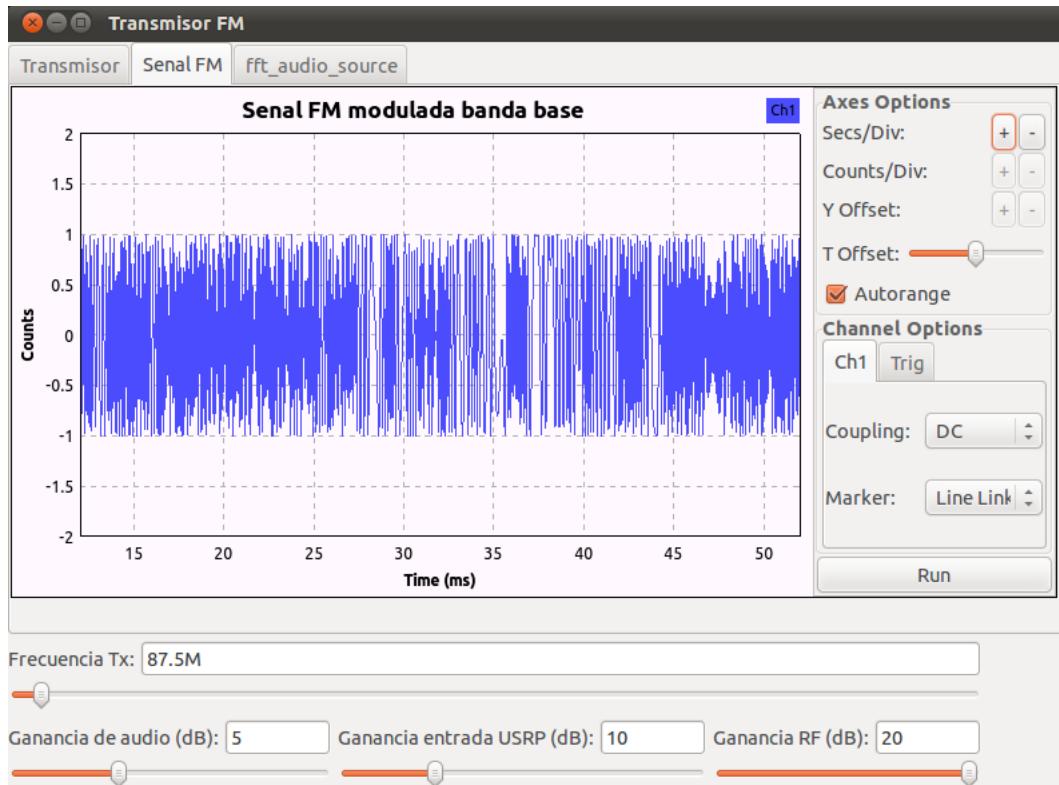


Figura 4-20 Visualización de la señal en el tiempo

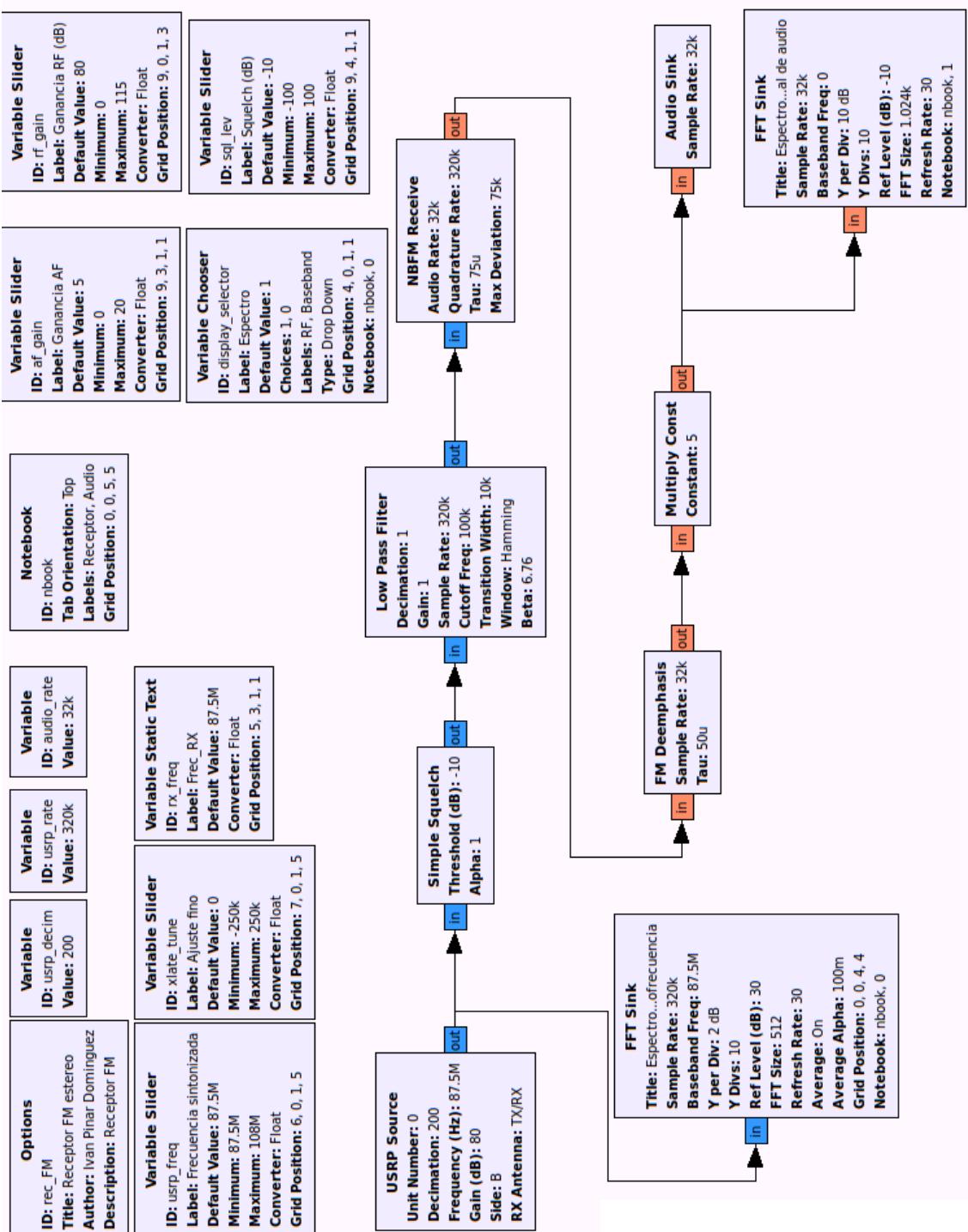


Figura 4-21 Diagrama de bloques del receptor de FM

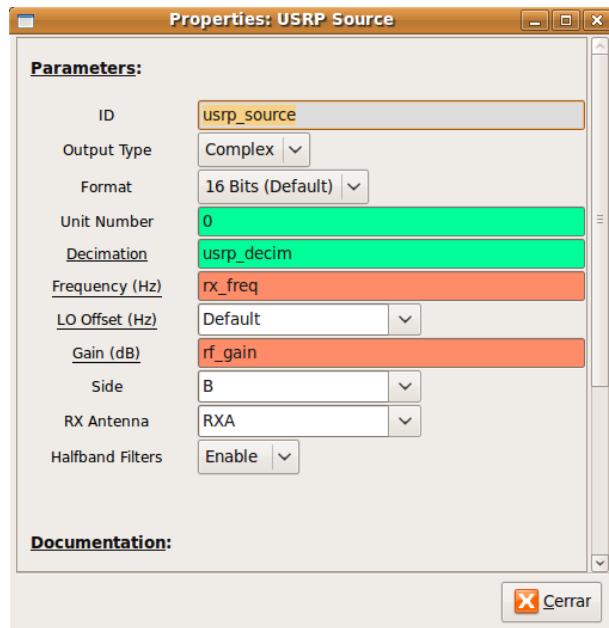


Figura 4-22 Parámetros de la fuente USRP

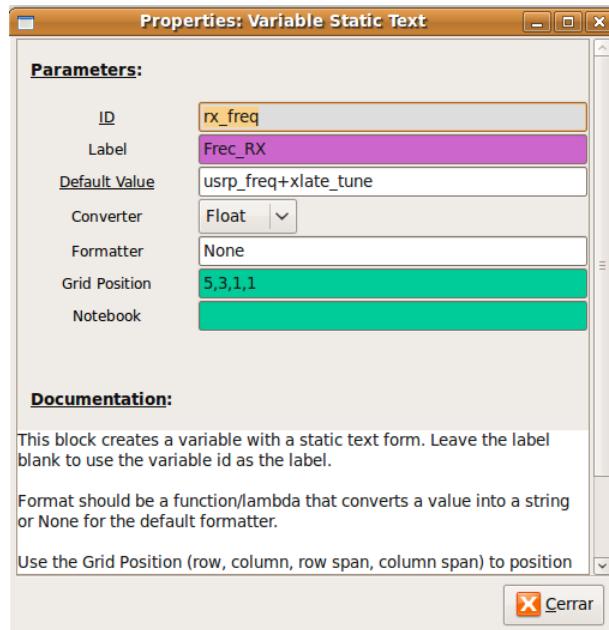


Figura 4-23 Variable de ajuste de frecuencia sintonizada

Como se ha comentado en el Apartado 4.1, para obtener un ancho de banda de 320 kHz ahora es necesario un factor de diezmado de 200. La frecuencia y ganancia de la tarjeta

secundaria (una TVRX ubicada en el lado B) se ajustarán con sendas variables *slider*. Indicar que la variable *rx\_freq* depende de un ajuste grueso y un ajuste fino de la frecuencia tal y como se puede apreciar en la Figura 4-23.

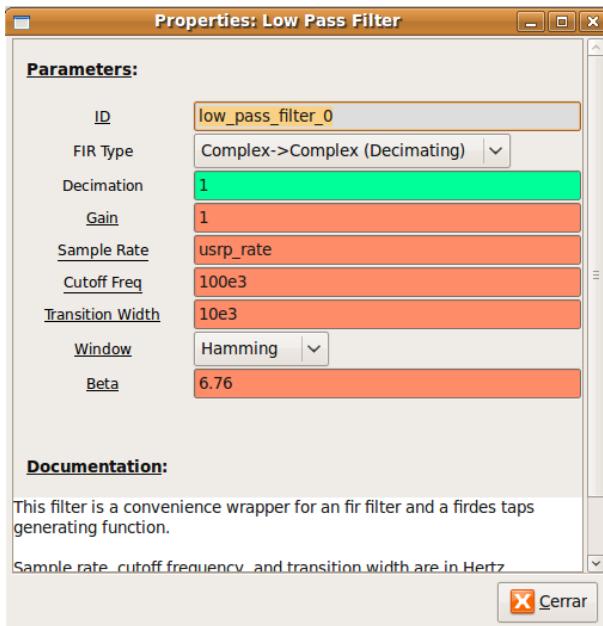


Figura 4-24 Características del filtro de canal

Siguiendo en el diagrama se encuentra el bloque **simple squelch**, utilizado para eliminar toda componente que no supere un determinado nivel y así disminuir el nivel de ruido. El umbral se controla a partir de la variable *slider sql\_lev*. Un valor aconsejable es de -10 dB.

A continuación aparece un **filtro de canal** para seleccionar el ancho de banda de la emisión de FM. También se podría seleccionar un factor de diezmado mayor para obtener el ancho de banda deseado, es decir, con un factor de diezmado 320, la tasa de muestras por la interfaz USB (y por tanto el ancho de banda) sería de 200 MS/s, con lo que se tendrían los 200 kHz, aunque hay que tener en cuenta que la visualización de la FFT muestra un ancho de banda igual a la tasa de muestras que se especifique, obteniendo resultados más vistosos si se selecciona el diezmado 200 como en este caso. Las propiedades del filtro de canal se muestran en la Figura 4-24.

Como la señal de FM tiene un ancho de banda de 200 kHz en banda base, la frecuencia superior estará en 100 kHz y la inferior en -100 kHz y esta será por tanto la frecuencia de corte, asignando un ancho de banda de transición del 10%, es decir, 3 dB de pérdidas en  $\pm 110$  kHz.

Tras sintonizar el receptor, eliminar parte del ruido y seleccionar el ancho de banda de la señal FM, se está en disposición de demodular la señal con el bloque **demodulador de FM** (de banda estrecha en este caso). Los parámetros de este elemento son los mismos que para el

modulador de FM analizado en el apartado anterior:

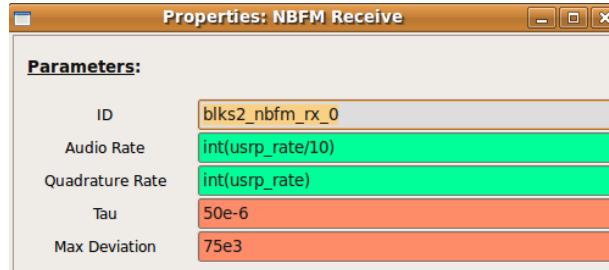


Figura 4-25 Parámetros demodulador FM

Indicar que en el parámetro *Audio Rate* se podría haber seleccionado igualmente la variable *audio\_rate* (32 kHz). El resto de valores ya se explicaron en el apartado previo.

Tras el demodulador se añade el **bloque deénfasis** para compensar el preénfasis utilizado en transmisión y obtener la señal original (con ruido obviamente).

Por si la señal de audio resultara débil se conecta un **bloque amplificador** antes del sumidero final que será la tarjeta de sonido, cuyo único parámetro es la tasa de audio, 32 kHz, es decir la tasa de muestras de audio que se especifique en el sumidero *audio\_sink* y que depende de la tarjeta de sonido utilizada. Otros valores típicos son 44.1 kHz y 48 kHz.

Una vez finalizado el flujo de señal, se añaden las visualizaciones oportunas en diferentes pestañas. Cabe comentar un detalle de implementación en la FFT del espectro de radiofrecuencia, ya que en la interfaz gráfica se ofrece la posibilidad de representar el espectro en banda base o en radiofrecuencia. Para ello se crea una variable *chooser* con identificador *display\_selector* y se configura la visualización *FFT\_sink* como sigue:

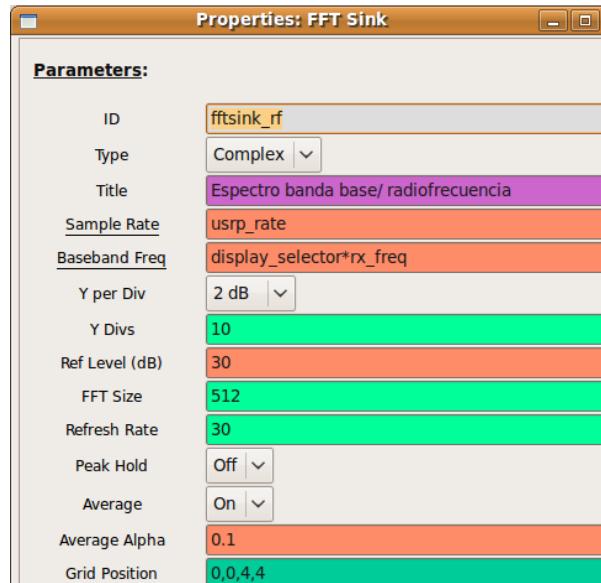


Figura 4-26 Configuración de FFT\_sink

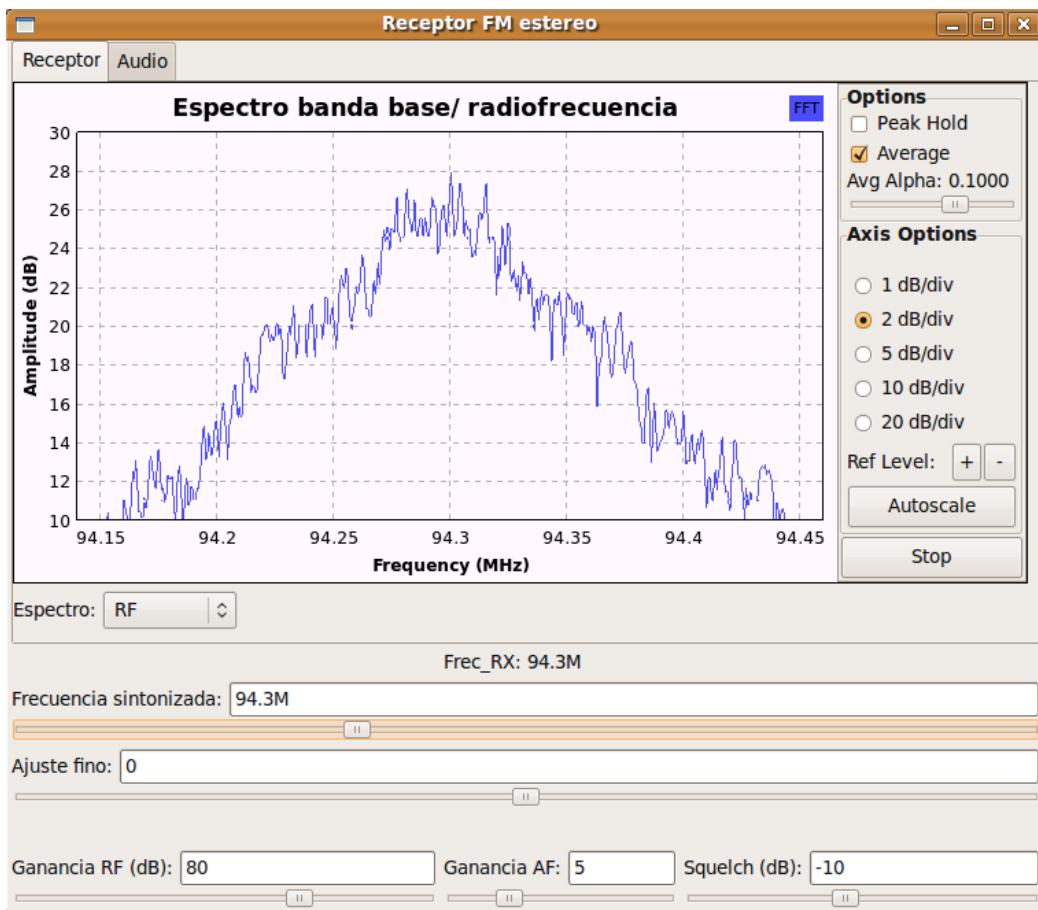


Figura 4-27 Sintonización de la emisión de FM comercial a 94.3 MHz

La variable *display\_selector* puede tomar los valores 0 ó 1, de tal manera que se consigue el cometido buscado (seleccionar banda base o radiofrecuencia).

Tras generar el fichero Python y ejecutarlo, el resultado al sintonizar una emisión FM comercial a una frecuencia de 94.3 MHz se muestra en la Figura 4-27.

Se puede observar como el ancho de banda de la señal de FM es de 200 kHz. No ha sido necesario realizar ajuste fino de frecuencia en ninguna de las pruebas FM realizadas.

Tras comprobar que el receptor funcionaba correctamente, se procederá a sintonizar la emisión generada con el módulo transmisor FM creado anteriormente. El resultado se muestra en la Figura 4-28.

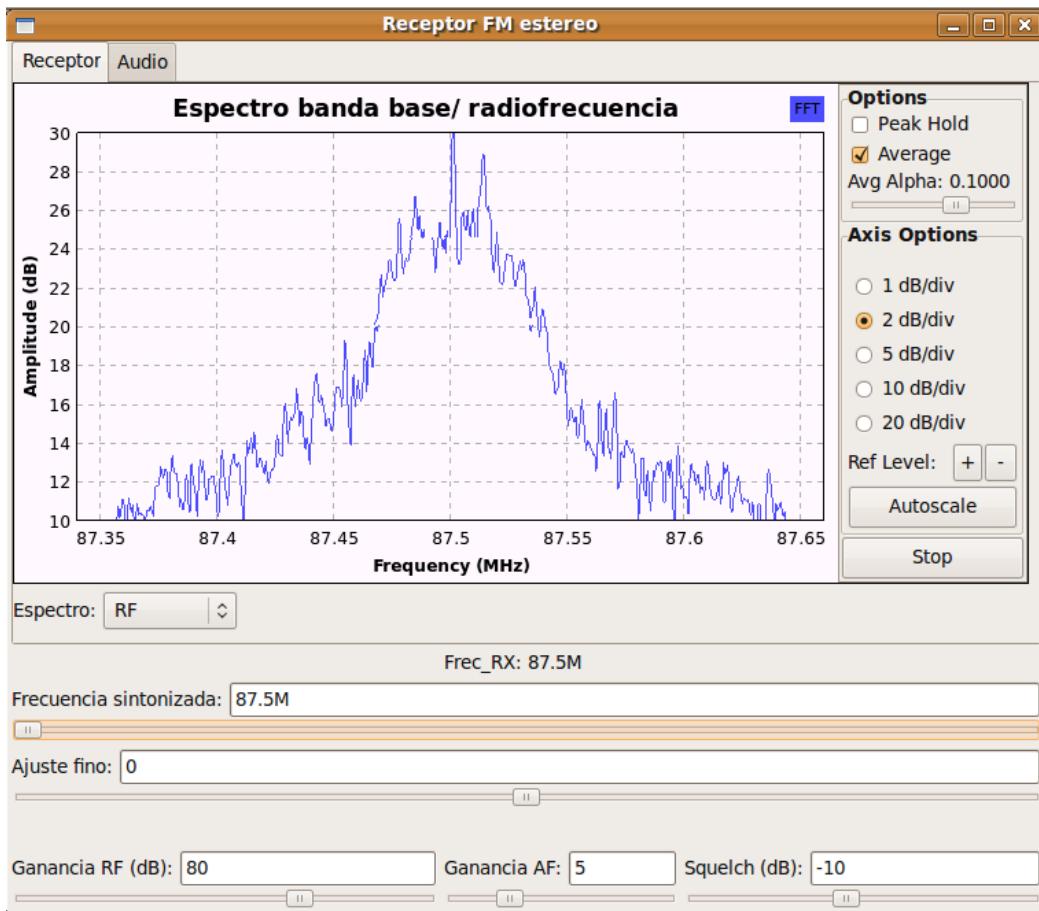


Figura 4-28 Sintonización de la emisión de FM generada con el módulo transmisor FM en otro USRP a 87.5 MHz

La demodulación se realiza correctamente, para corroborarlo se muestra la señal de audio que se envía a la tarjeta de sonido en la Figura 4-29.

La señal de audio se extiende hasta los 3 kHz debido a que el sistema implementado se basa en el modulador y demodulador de banda estrecha (Narrow-Band FM). También se puede llevar a cabo la implementación del mismo sistema FM en banda ancha simplemente con cambiar los bloques NBFM por los bloques WFM también suministrados en la arquitectura GNU Radio, por lo que se plantea como ejercicio, estando la solución disponible en los Apéndices.

## 4.1 Sistema (D)QPSK

En este apartado se introduce el diseño de sistemas de comunicaciones digitales. Antes de profundizar en cada uno de los módulos, se van a aclarar algunos conceptos comunes a tener en cuenta en todos los esquemáticos siguientes:

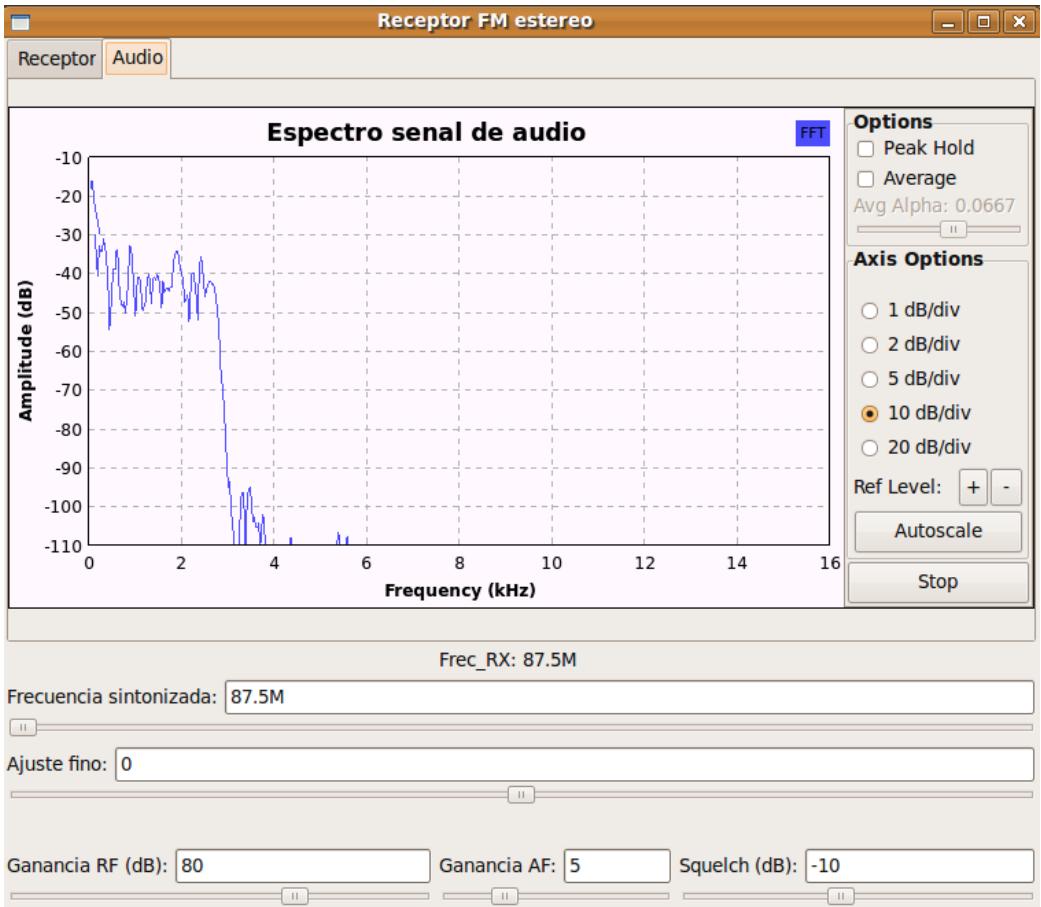


Figura 4-29 Espectro de la señal de audio transmitida a la tarjeta de sonido

- La fuente de información que se pretende transmitir es de tipo *wav source*. La tasa de muestreo para esta fuente será de 32 kHz. Cada una de las 32 kS/s que proporciona esta fuente es de tipo *float*, es decir, 4 bytes. Teniendo en cuenta esto, el régimen binario que en principio se necesitaría sería de 1024 kbps para transmitir correctamente el fichero *wav*.

- Como se ha visto en numerosos apartados, el factor de diezmado e interpolado mínimo a utilizar es de 8 y 16 respectivamente para no sobrepasar la tasa de 32 MB/s en la interfaz USB. Sin embargo, al trabajar con modulaciones digitales se ha comprobado que el límite inferior de funcionamiento correcto es de 64 para el interpolado y 32 para el diezmado, por lo tanto (el diezmado siempre se elegirá la mitad de interpolado para que la tasa en la interfaz USB de transmisión y recepción sea la misma y el sistema opere adecuadamente), es decir, se multiplican por 4 ambos valores. La explicación se encuentra en la respuesta del filtro interpolador CIC (Cascade Integrator Comb) del USRP, que se degrada conforme el factor de interpolado disminuye, distorsionando la forma de onda del coseno alzado [15]. Este hecho limitará la tasa máxima de transmisión o las muestras por símbolo enviadas. Por ejemplo, considerando un factor de interpolado de 64, en la interfaz USB la tasa de muestras será 2

MS/s (128 MS/s / 64), por lo que si se especifica un número de muestras por símbolo de 16 para tener una buena resolución del coseno alzado y por ende de la constelación representada, la tasa de símbolos máxima será de 125 kSimb/s QPSK (2 MS/s / 16).

En los apartados sucesivos se mostrarán las soluciones adoptadas para cada modulación.

#### 4.1.1 Transmisor (D)QPSK

Se pasará a detallar la implementación del transmisor QPSK que aparece en la Figura 4-31. Este diseño se ha realizado bloque a bloque para que resulte más didáctico su análisis, aunque existen bloques en GNU Radio que contienen todos los elementos necesarios para modular la señal de interés. Éstos se mostrarán en apartados sucesivos.

Comenzando el análisis desde el inicio del esquemático, la **fuente** como se ha mencionado es de tipo *wav* seguida de un **bloque remuestreador Rational Resampler** que disminuye la tasa de muestras por 4 en este caso. Sus parámetros de configuración aparecen a continuación:

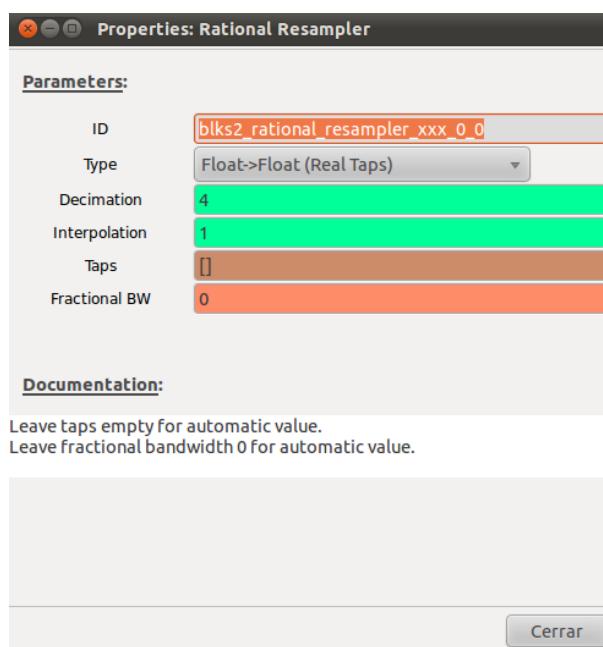


Figura 4-30 Parámetros del bloque Rational Resampler

Seguidamente se encuentra el **codificador** que transforma las muestras de tipo *float* en bytes. Estos bytes estarán formados por 4 símbolos de 2 bits. Nótese que para una QPSK necesitamos 2 bits para codificar un símbolo.

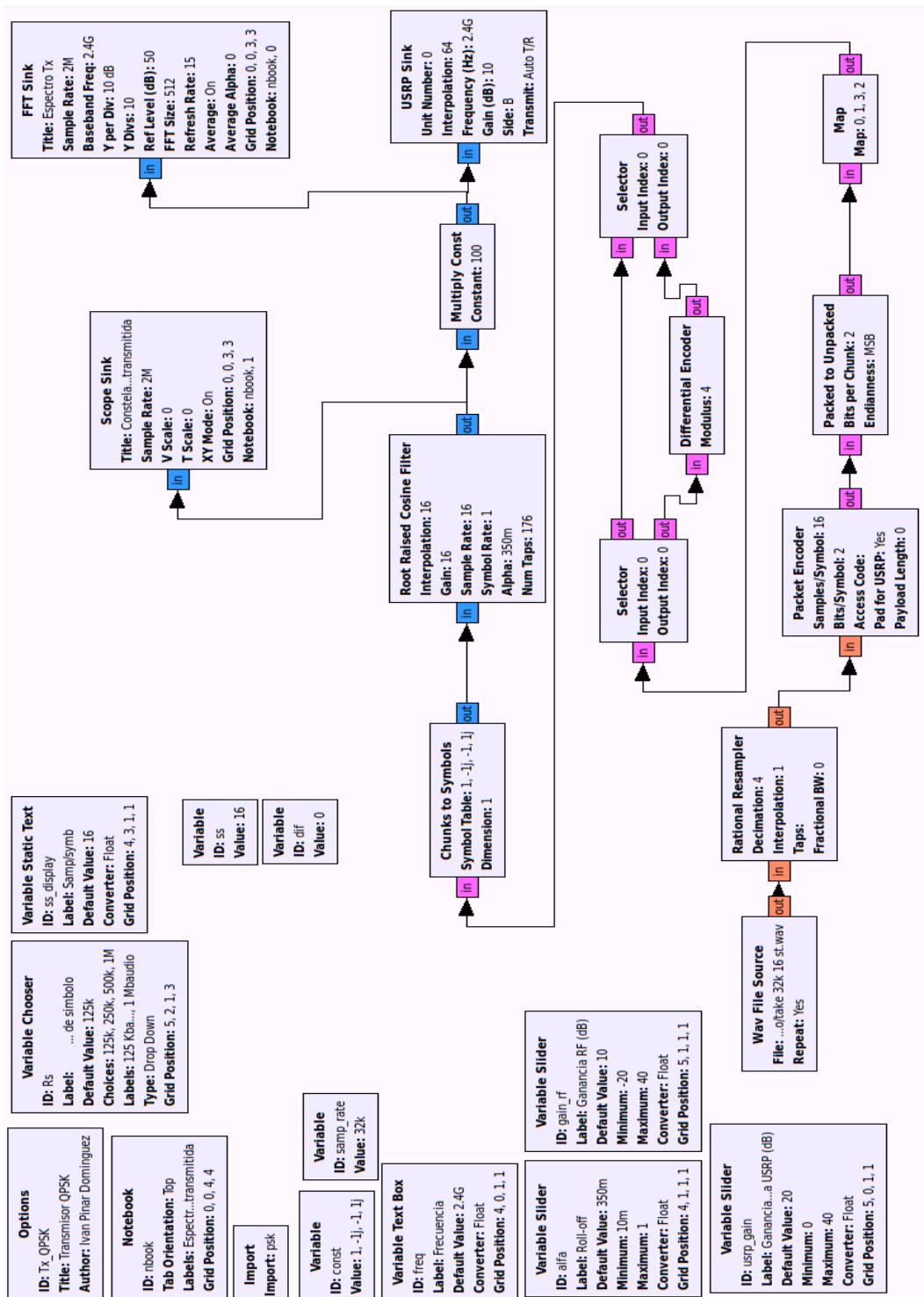


Figura 4-31 Esquemático del transmisor QPSK

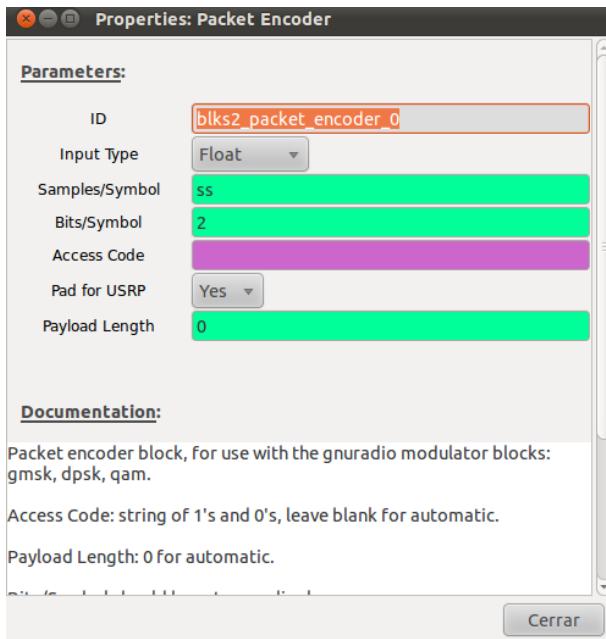


Figura 4-32 Parámetros del bloque Packet Encoder

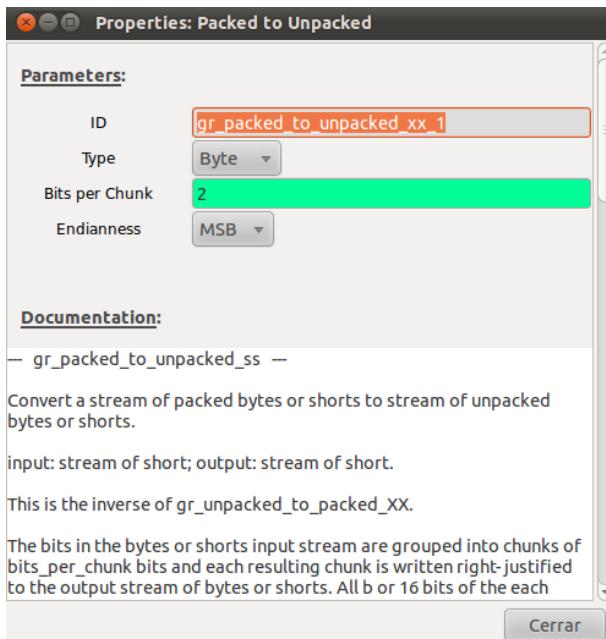


Figura 4-33 Parámetros del bloque Packed to Unpacked

Entre sus parámetros destacar *Samples/Symbol*, que es el número de muestras que se transmiten por cada tiempo de símbolo y que en este caso será 16 (el valor de la variable *ss*) y

*Bits/Symbol*, que al tratarse de una modulación QPSK es 2. El *Access Code* es una contraseña de acceso por si algún agente externo interceptara la información transmitida, que se deja en blanco para que no sea necesaria.

A continuación se realiza un **desempaquetado** de los bytes, colocando un símbolo de 2 bits en cada byte en posición MSB (Most Significant Bit) denominado *chunk*, necesario para operar posteriormente con el bloque *Chunks to Symbols*. Para ello se utiliza el bloque *Packed to Unpacked* que se muestra en la Figura 4-33.

Una vez desempaquetada la información, se realiza un **mapeo de Gray** con el bloque *Map* cuya configuración es la siguiente:

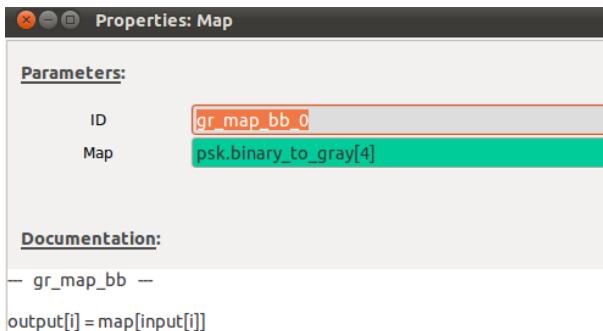


Figura 4-34 Parámetros mapeo de Gray

Se aprecia como para indicar el mapeo se ha utilizado una función del módulo PSK. Para poder utilizar dicha función primero se ha debido importar en el esquemático dicho módulo tal y como puede observarse en la figura Figura 4-35.



Figura 4-35 Import del módulo psk

A continuación aparecen tres bloques para poder convertir la modulación en diferencial. Para ello se añaden dos **bloques selectores** y un **bloque diferencial** cuyo parámetro es *Modulus*, es decir, el número de símbolos de la modulación (en QPSK, 4) tal y como aparece en la Figura 4-36.

El bloque selector inicial y final se controlan con la variable *dif* tal y como se puede apreciar en la Figura 4-37.

Si *dif* toma el valor 0, el flujo de señal transcurre por la rama superior y por tanto no interviene el bloque diferencial al contrario de seleccionarse el valor 1.

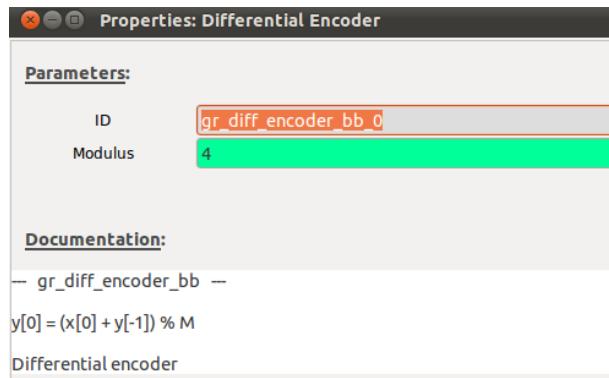


Figura 4-36 Propiedades bloque diferencial

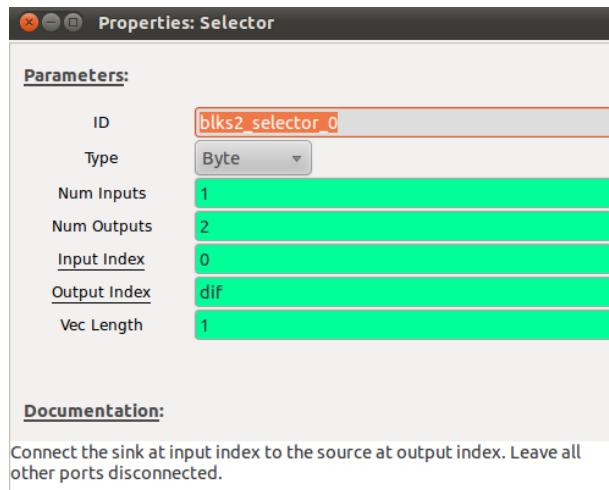


Figura 4-37 Propiedades bloque selector

Tras esto se está en disposición de realizar un **mapeo complejo** para convertir los bytes de información en muestras complejas a través del bloque *Chunks to Symbols* cuyas características son las incluidas en la Figura 4-38.

El parámetro *Dimension* representa el número de dimensiones para representar la constelación en cuestión. Se especifica en 1 ya que se indica el formato de salida *complex* (no tiene que ser 2 como se podría pensar). En el parámetro *Symbol Table* se especifica el mapeo que se va a realizar a los datos, en este caso será un mapeo QPSK especificado por la variable *const* que se muestra en la Figura 4-39.

Tras obtener los símbolos complejos se pasan por un **filtro raíz de coseno alzado** con las especificaciones en la Figura 4-40.

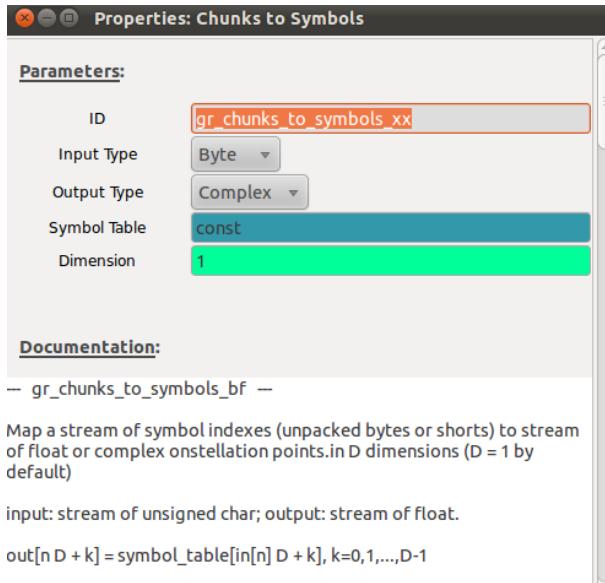


Figura 4-38 Propiedades Chunks to Symbols

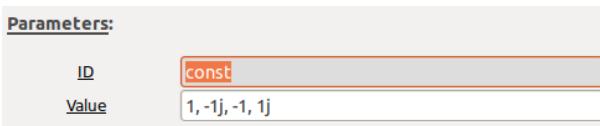


Figura 4-39 Variable const

El filtro será de tipo FIR con factor de interpolación  $ss$  (en nuestro ejemplo 16), es decir, el valor que se especifique en la variable que controla el número de muestras por símbolo QPSK, para que cada símbolo esté representado por 16 muestras. La ganancia también se selecciona según dicha variable (aunque también podría ser 0 dB y amplificar la señal posteriormente) al igual que el parámetro *Sample Rate* (la tasa de símbolos QPSK). Indicar que los valores de *Sample Rate* y *Symbol Rate* tienen significado relativo, es decir, lo que importa es la proporción entre ambos valores y que darán el número de muestras por símbolo:  $\text{Sample Rate}/\text{Symbol Rate} = \text{Samples/Symbol}$ . Por último se especifica el valor de *Roll-off* con la variable *alfa* y el número de *taps* o coeficientes del filtro en función del número de muestras nuevamente, por lo que en este caso el pulso RRC durará 11 símbolos<sup>11</sup>.

Tras pasar por el coseno alzado se aumenta el nivel de señal transmitido gracias al **amplificador software**. La tarjeta RFX2400 utilizada no tiene ganancia variable en transmisión, por lo que la potencia de la señal se puede controlar a partir de este bloque gobernado por la variable *slider usrp\_gain*. Por supuesto, el valor que se especificará a esta variable dependerá del nivel de entrada, es decir, se podría haber multiplicado a los valores de

---

<sup>11</sup> RRC, *root raised cosine*. Pulso de longitud infinita, que en la práctica se acota a un conjunto de muestras o símbolos.

la constelación por una constante igualmente o haber subido la ganancia del filtro RRC.

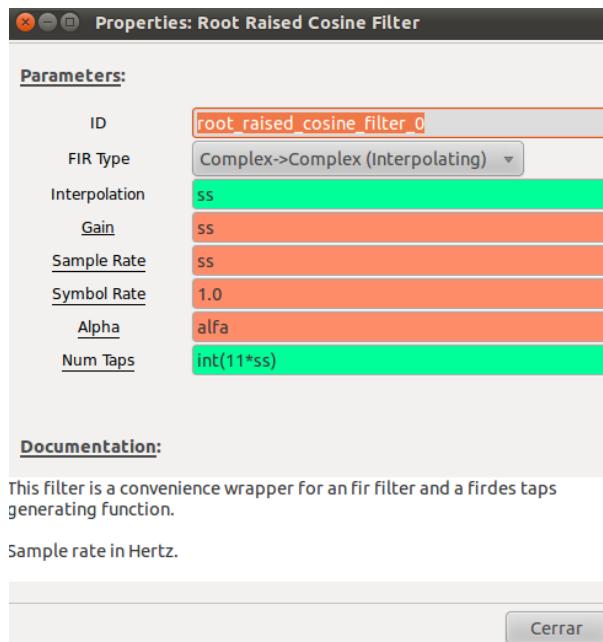


Figura 4-40 Parámetros filtro coseno alzado

El *sumidero* será el USRP, cuyas características aparecen en la siguiente figura:

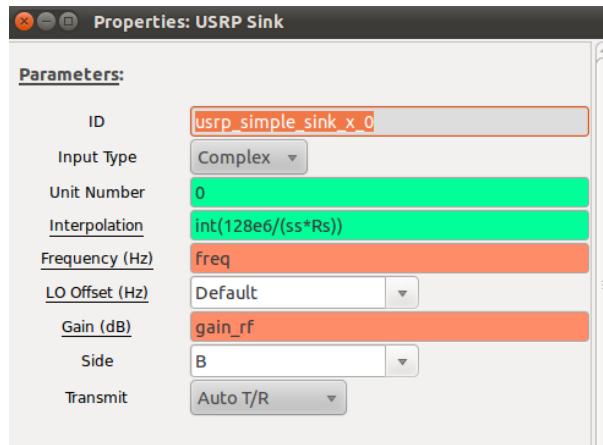


Figura 4-41 Parámetros sumidero USRP

El factor de interpolación se especificará en función de la variable *chooser Rs* que controla la tasa de símbolos a transmitir. Para entenderlo, pensar que la tasa de muestras que irá por la interfaz USB será de  $(128 \text{ MS/s} / \text{interpolación}) = \text{Samples/Symbol}\cdot R_s$ . La frecuencia y la

ganancia (aunque esta última no es posible con esta tarjeta) se controlan a partir de otras dos variables *slidder*.

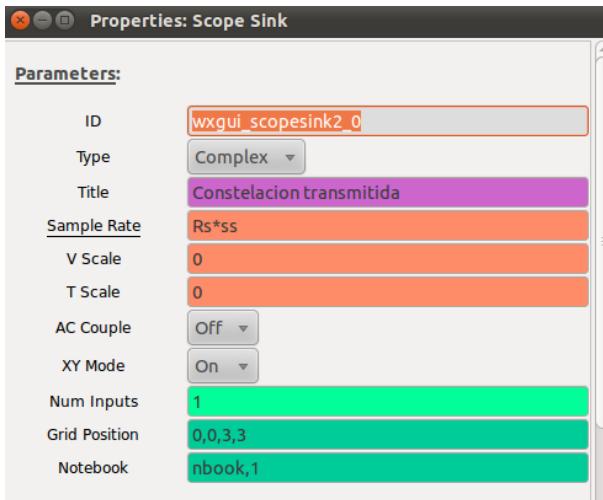


Figura 4-42 Parámetros sumidero USRP

Finalmente se añaden las visualizaciones oportunas de FFT y osciloscopio a partir del cual se puede representar la constelación seleccionando modo XY tal y como aparece en la Figura 4-42.

Tras finalizar el diseño, se pasará a mostrar los resultados. En principio se necesita un flujo binario de 1024 kbps para poder enviar el fichero *wav*, es decir, una tasa de símbolos de 500 kbaudios por tratarse de una modulación QPSK. Esto significa que, como el factor de interpolado no puede sobrepasar el valor 64 para poder operar con señales moduladas digitalmente, el valor máximo del número de muestras por símbolo será 4 debido a la siguiente relación:

$$\text{Interpolado} = 128 \text{ MHz} / (\text{Samp\_symb} \cdot R_s) \quad (4-2)$$

Este valor del número de muestras por símbolo supone una pobre resolución del coseno alzado, obteniendo por consiguiente una mala representación de la constelación tal y como puede apreciarse en la Figura 4-43.

Debido a ello, se aumentará el número de muestras por símbolo a 16, con lo que la tasa máxima de símbolos será 125 kbaudios a partir de la cual no se podría enviar el fichero *wav* original. La solución a esto es el bloque *Rational Resampler* que no se explicó anteriormente. Como la tasa de símbolos máxima es la cuarta parte que la necesaria, reducimos el número de muestras de la fuente en un factor 4, es decir, la tasa de muestras de la fuente es ahora de 8 kHz y la tasa binaria necesaria de 256 kbps (8 kS/s \* 32 bits/muestra) que sí se puede proporcionar (aproximadamente) con una tasa de símbolos QPSK de 125 baud. Las implicaciones de reducir en fuente el número de muestras en las prestaciones del diseño se analizarán en el siguiente apartado. En este momento, cabe destacar que no se ha explorado la posibilidad de enviar un fichero en formato *mp3* ya que, al ser un formato comprimido, su

lectura en tiempo real no es tan inmediata como un fichero *wav*.

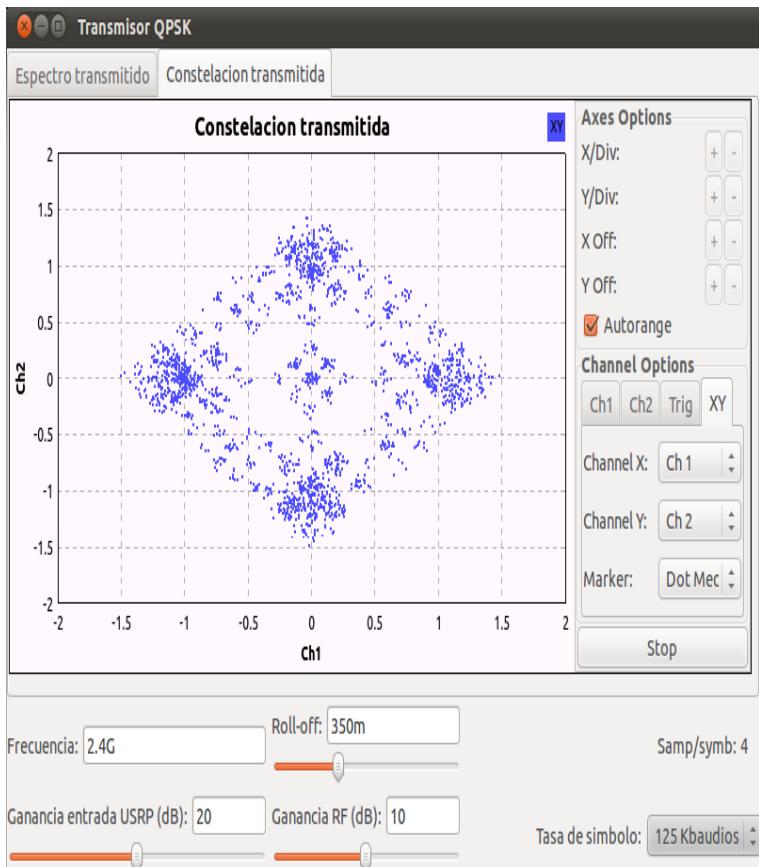


Figura 4-43 Constelación QPSK con 4 muestras por símbolo

Los resultados obtenidos para esta nueva configuración, aparecen en las figuras siguientes. La constelación transmitida se muestra en la Figura 4-44 y el espectro de la señal transmitida en la Figura 4-45.

Se puede apreciar como al aumentar la resolución del coseno alzado mejora la visualización de la constelación QPSK. En la figura se aprecia como el ancho de banda transmitido es de unos 200 kHz, aproximadamente  $(1+\alpha) \cdot R_s$  que es el valor teórico ([5], ecuación 5.7.1). La visualización sin embargo tiene un *span* igual a la tasa de muestras especificada, en este caso, 2 MHz ( $R_s \cdot samp\_symb$ ).

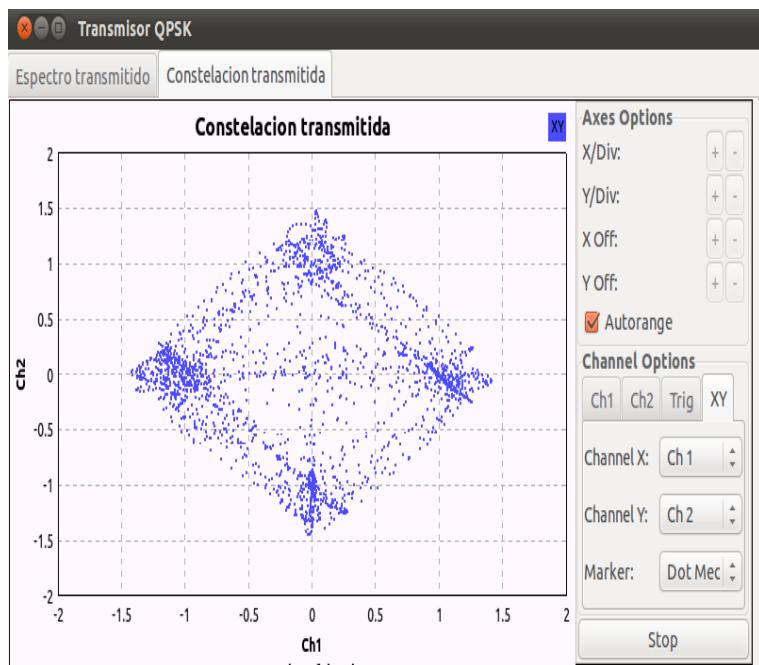


Figura 4-44 Constelación QPSK con 16 muestras por símbolo

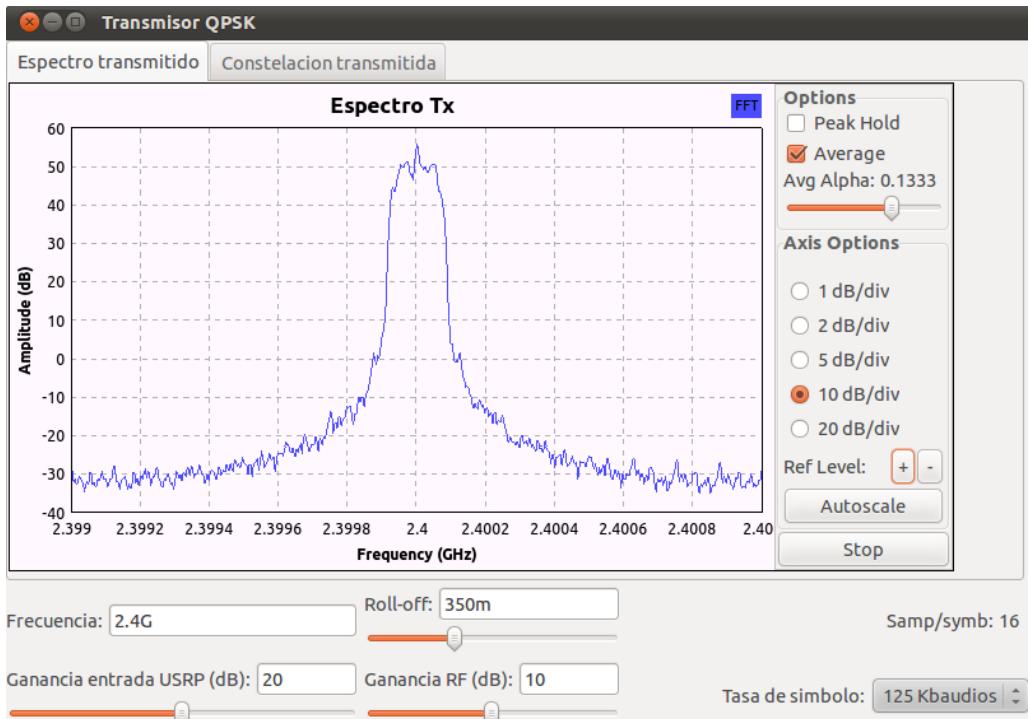


Figura 4-45 Espectro señal transmitida

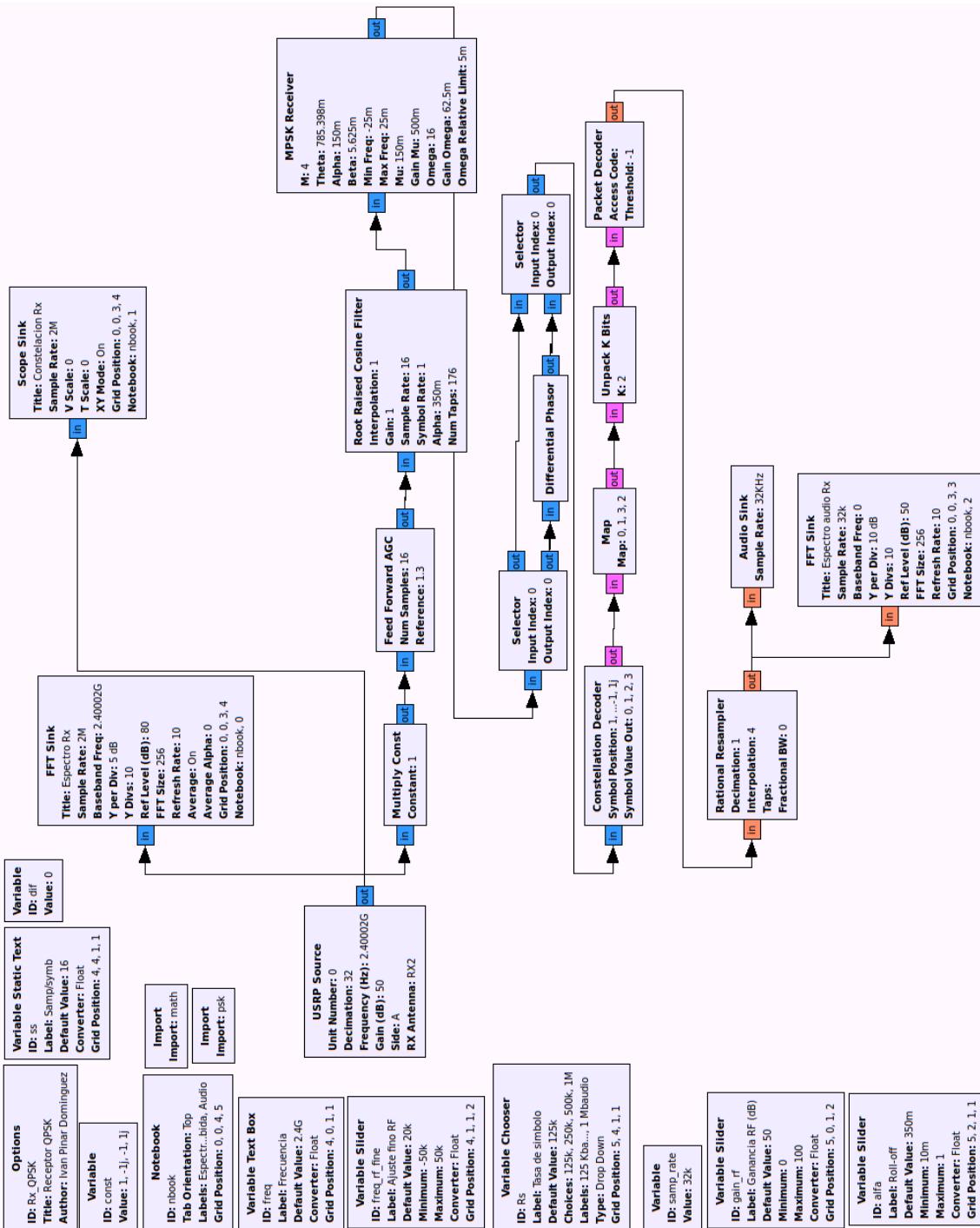


Figura 4-46 Esquemático del receptor QPSK

#### 4.1.2 Receptor (D)QPSK

Una vez explicado en detalle el transmisor (D)QPSK se acometerá el diseño del receptor (D)QPSK que básicamente sigue un esquema inverso al transmisor, aunque es necesario insertar algún bloque extra para compensar el canal de comunicaciones. El esquemático creado aparece en la Figura 4-46.

El análisis de este diseño se iniciará en la **fuente**, es decir, el USRP. La configuración de este bloque aparece en la siguiente figura:

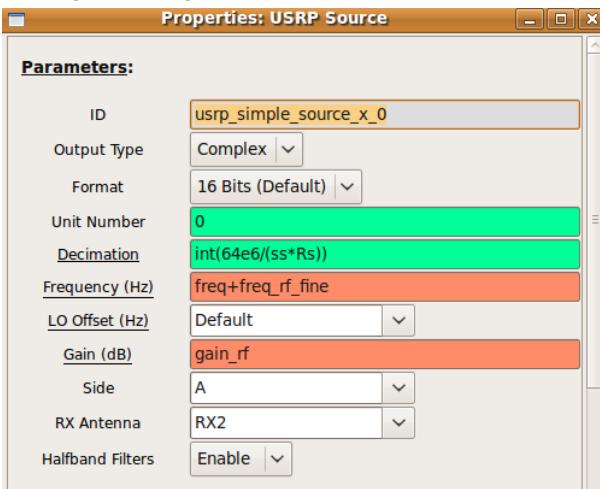


Figura 4-47 Parámetros de la fuente USRP

Los parámetros configurables son los explicados en otras ocasiones. Resaltar que el factor de diezmado sigue el mismo concepto que en el transmisor pero teniendo en cuenta que la frecuencia de muestreo del ADC es de 64 MS/s y no de 128 MS/s como es la del DAC. La frecuencia de sintonización constará de dos variables *slidder*, una de ajuste grueso en frecuencia y otra de ajuste fino debido a la desviación de la frecuencia portadora del equipo transmisor. Todo esto se explicará posteriormente.

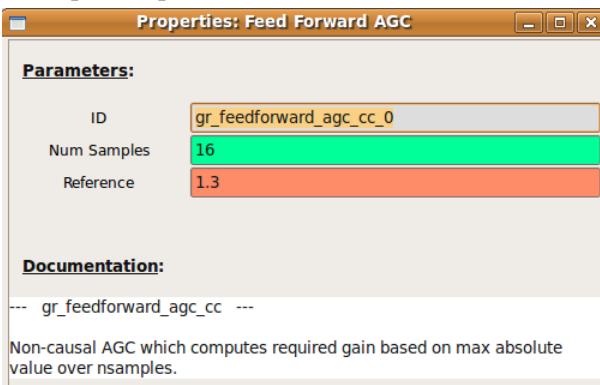


Figura 4-48 Parámetros del AGC

Seguidamente aparece un **amplificador de señal** por si fuera necesario y un **control automático de ganancia** implementado con el bloque *Feed Forward AGC*, cuyos parámetros configurables son el número de muestras sobre el que se calcula cada máximo (*Num Samples*) y el valor de referencia al que se ajusta la señal en módulo (*Reference*), el cual se debe seleccionar entre 1.3 y 1.5 para que se decodifique según la variable *const* (cuyo módulo de cada valor es  $\sqrt{2}$ ) tal y como se muestra en la Figura 4-49.

A continuación aparece el **filtro adaptado**, es decir, el bloque raíz de coseno alzado cuyos parámetros se muestran en la Figura 4-49.

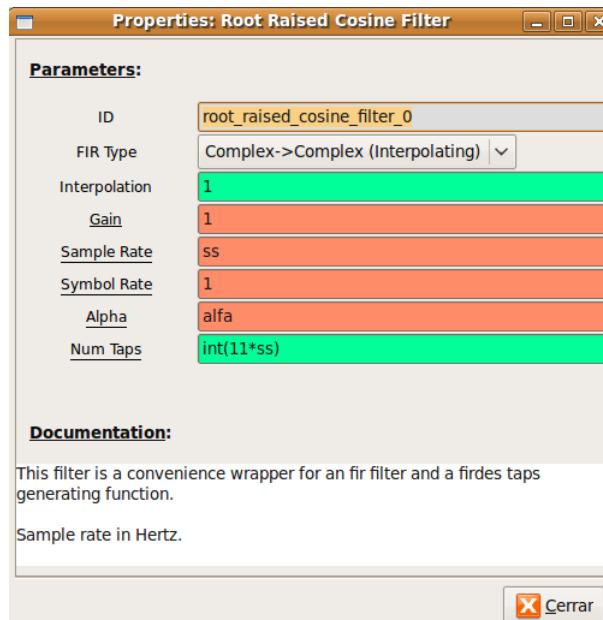


Figura 4-49 Parámetros del filtro raíz de coseno alzado

A diferencia del filtro transmisor aquí obviamente no hay que realizar ningún tipo de interpolación y la ganancia es 1.

Para poder sincronizar al receptor en tiempo, fase y frecuencia se utiliza el siguiente **bloque de sincronización**, *MPSK Receiver*, cuyos parámetros ajustan los valores del NCO (ubicado dentro del DDC, tal y como se explica en el Anexo II) del USRP. Indicar que, para la sincronización en frecuencia, la desviación debe ser mínima, debido a ello se añade en el esquemático un ajuste grueso manual de frecuencia (el ajuste fino se conseguirá con el NCO). Este bloque de sincronización se basa en el algoritmo Mueller & Muller, para más información, consultar [12]. Los valores utilizados para cada parámetro son los valores estándar utilizados en otros módulos de GNU Radio y tienen el siguiente significado:

- **M**: Orden de modulación M-PSK.
- **Theta**: Cualquier rotación de fase constante sobre el eje real de la constelación
- **Alpha**: Parámetro de ganancia para ajustar la fase del bucle de costas.

- **Beta**: Ganancia para ajuste de frecuencia en el bucle de Costas.
- **Fmin**: El mínimo valor de frecuencia normalizada que el bucle puede alcanzar.
- **Fmax**: El máximo valor de frecuencia normalizada que el bucle puede alcanzar.
- **Mu**: Parámetro inicial del interpolador.
- **Gain\_mu**: Parámetro de ganancia del error Mueller & Muller para ajustar mu.
- **Omega**: Valor inicial para el número de muestras por símbolos.
- **Gain\_omega**: Parámetro de ganancia para ajustar omega basado en el error.
- **Omega\_rel**: Selecciona el valor máximo y mínimo de omega, es decir, omega estará en el rango  $[\text{omega}*(1-\text{omega\_rel}), \text{omega}*(1+\text{omega\_rel})]$ .

A continuación aparece la configuración de este bloque:

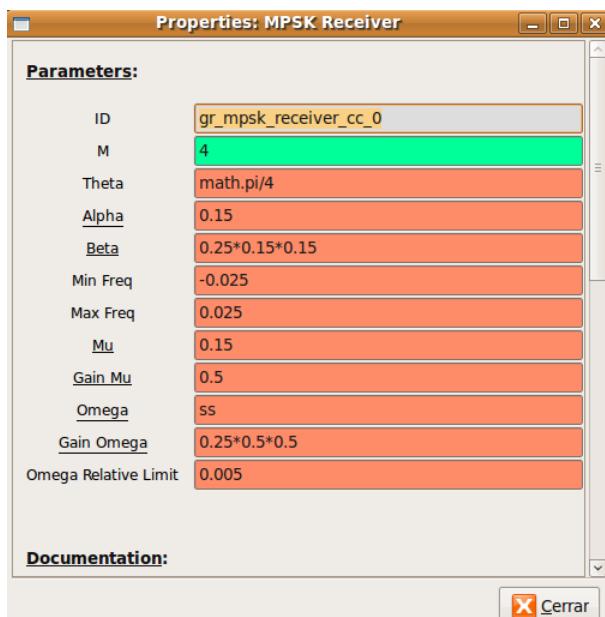


Figura 4-50 Parámetros del bloque de sincronización

Tras realizar la sincronización aparecen los bloques de selección de modulación diferencial cuyas características son las mismas que las explicadas en el transmisor QPSK.

Ahora se convierten los símbolos complejos a bits gracias al **bloque demapper Constellation Decoder** según la tabla especificada en la variable *const*.

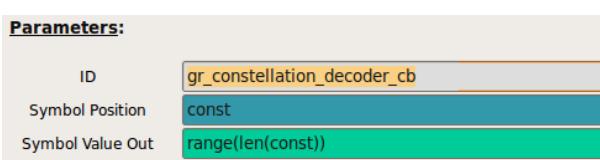


Figura 4-51 Parámetros del bloque Constellation Decoder

Tras decodificar los símbolos complejos, se realiza un **demapeado de Gray** con las siguientes características obteniendo los valores binarios con la función *gray\_to\_binary* del módulo *psk* importado:

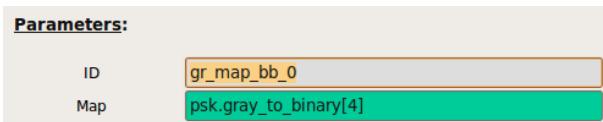


Figura 4-52 Parámetros del bloque Map

En este punto se tendrá la información en bytes y cada byte contiene 2 bits de información, por lo que para trabajar con el bloque *Packet Decoder* se necesita empaquetar los bits, es decir, tener en cada byte 8 bits de información, o lo que es lo mismo 4 símbolos de 2 bits. Para este cometido se introduce el bloque *Unpack K bits* (nombre del bloque desafortunado, ya que empaqueta los bits) y *Packet Decoder*. En definitiva, se realiza el proceso inverso al transmisor.

Como en el módulo del transmisor se disminuyó la tasa de la fuente en un factor 4, ahora en recepción para poder seguir el ritmo del fichero de audio original debemos interpolar por el mismo factor con el bloque *Rational Resampler*, esto implicaba una pérdida de calidad de la señal de audio puesto que al interpolar por 4 el espectro de la señal de audio disminuye por 4 su rango frecuencial. Tras ello se termina el flujo de señal en el **sumidero de audio** Audio Sink con una tasa de 32 kHz.

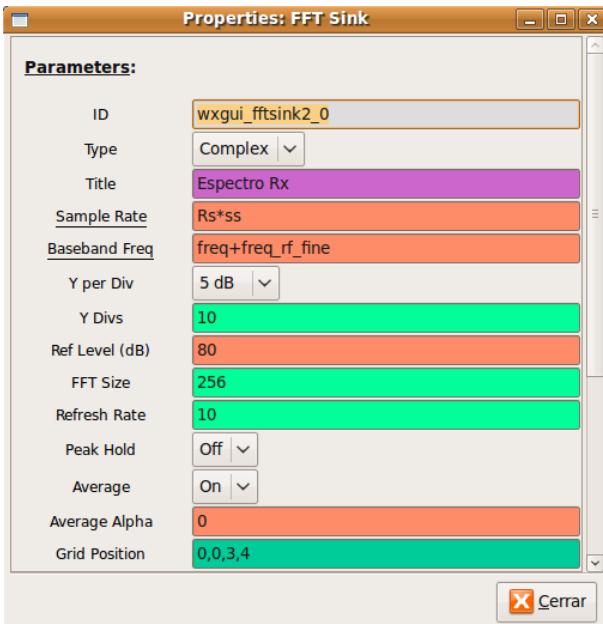


Figura 4-53 Parámetros del sumidero FFT sink

Antes de pasar a los resultados se mostrará, Figura 4-53, la configuración del sumidero gráfico FFT para visualizar el espectro de la señal recibida. Se puede apreciar como la frecuencia a la que centrará la visualización depende de la suma del ajuste grueso y fino de frecuencia. La tasa de muestreo por su parte será  $R_s \cdot ss$ .

En cuanto al sumidero gráfico *Scope* utilizado para visualizar la constelación, indicar que durante las pruebas se ha ido cambiando de posición para representar la constelación en diferentes puntos del receptor.

Finalmente se mostrarán los resultados obtenidos. Comencemos por el espectro de la señal recibida:

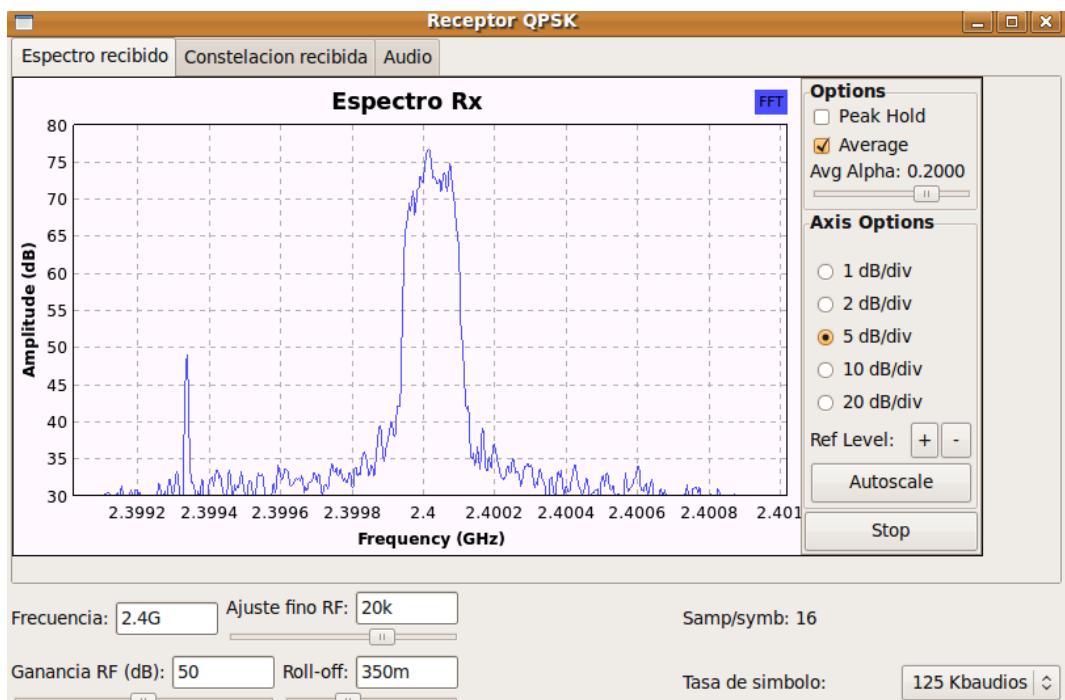


Figura 4-54 Espectro de la señal recibida

En la figura se puede observar como el espectro recibido no está centrado en la frecuencia de 2.4 GHz que se especificó en transmisión, sino que se produce una desviación en frecuencia de unos 20 kHz, es decir, la desviación en frecuencia del transmisor es de 8 ppm aproximadamente.

A continuación se mostrará la constelación en diferentes puntos del diseño. La constelación de la señal recibida está afectada por multirayecto tal y como puede verse en la Figura 4-55.a, además de un pequeño error en frecuencia, a pesar del ajuste fino, que conllevará un desfase variable en el tiempo y que resolverá el bloque *MPSK Receiver*.

Se puede ver como la señal recibida tiene un nivel aceptable, tomando valores de amplitud

en torno a 1500<sup>12</sup>. Tras pasar la señal por el control automático de ganancia, el módulo de ésta se ve limitado a 1.3. Los límites deberían formar una circunferencia en lugar de un polígono, esto se debe a la forma de operar del AGC de la arquitectura GNU Radio. El resultado se puede apreciar en Figura 4-45.b.

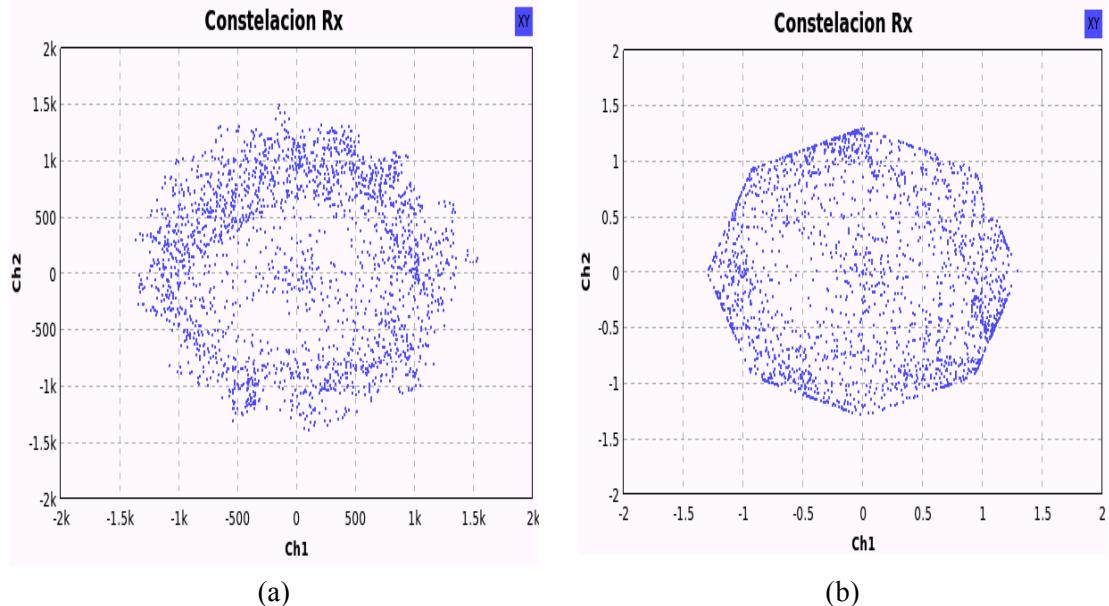


Figura 4-55 Constelación de (a) la señal recibida y (b) de la señal recibida tras el AGC

Tras el filtro adaptado, la constelación queda igualmente enmarañada, puesto que no está sincronizado. Gracias al bloque de sincronización se pueden obtener los símbolos complejos convenientemente como muestra la Figura 4-56.

En esta figura se observa como el espectro de audio está limitado a la cuarta parte del fichero original (cuyo ancho de banda se extendía hasta los 16 kHz) debido al remuestreo, o *resampler* en inglés, que se realiza. Indicar también que la fidelidad de la señal de audio es mayor si se selecciona el bloque diferencial (DQPSK en lugar de QPSK) ya que se comporta mejor cuando hay problemas de recuperación de sincronismo.

Para finalizar este apartado, se va a mostrar la distorsión introducida al aumentar la ganancia de recepción o transmisión (esta última mediante el amplificador software). Si la ganancia de transmisión se aumenta de 20 a 40 dB con los USRPs separados 1.5 metros, el espectro de la señal que proporciona el USRP sufre distorsión por saturación de entrada de la tarjeta secundaria, tal y como se muestra en la Figura 4-58.

---

<sup>12</sup> Los valores de amplitud que se muestran en todos los diseños están dados respecto a un valor de referencia que depende de cada tarjeta secundaria.

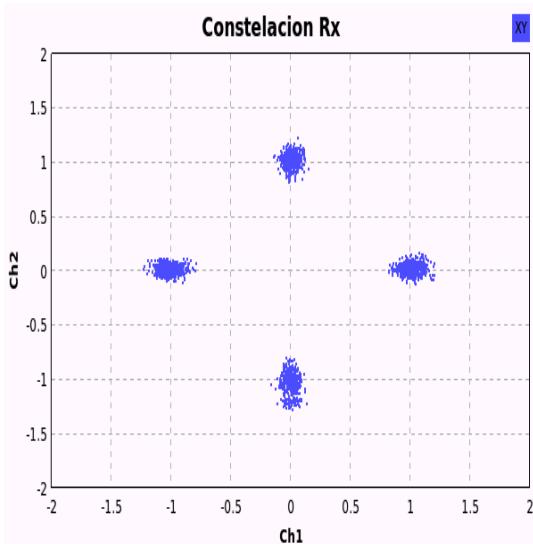


Figura 4-56 Constelación de la señal sincronizada

A partir de esta constelación el bloque *Constellation Decoder* puede trabajar adecuadamente obteniendo la información deseada. La señal de audio decodificada tiene el siguiente espectro:

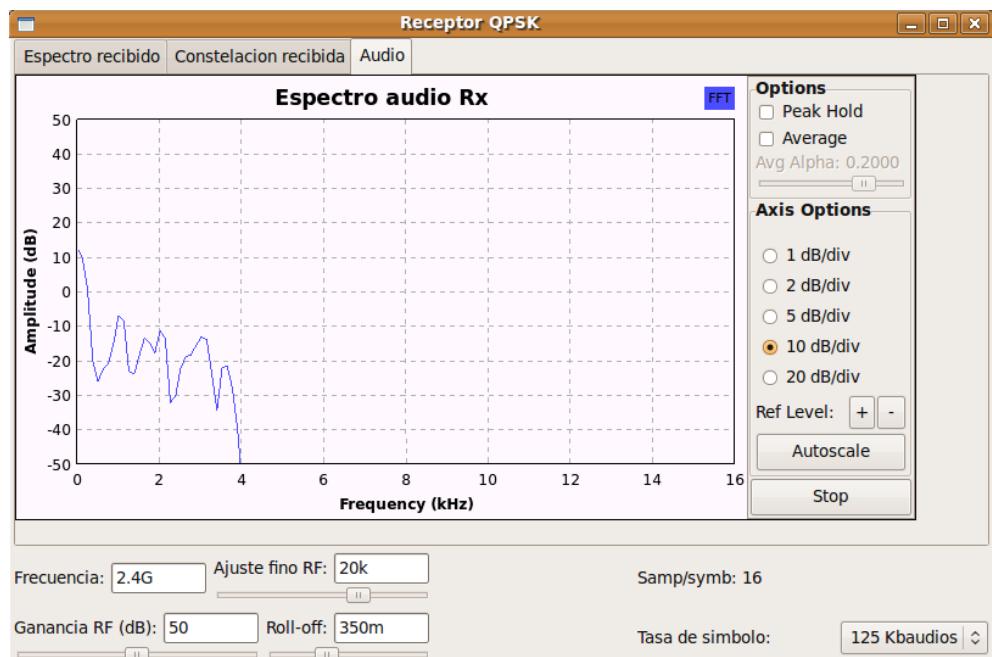


Figura 4-57 Espectro de la señal de audio

Comparado con la Figura 4-54, se tienen unos “hombros” en el espectro, es decir, aumenta la potencia de los lóbulos laterales. Igualmente, si lo que se aumenta es la ganancia de recepción en el USRP (a 80 dB por ejemplo), manteniendo la ganancia de transmisión en 20 dB, se vuelve a apreciar distorsión por intermodulación de la señal como muestra la Figura 4-59. Observándose claramente el incremento de los lóbulos laterales de 3<sup>er</sup> y 5º orden.

## 4.2 Sistema digital

En este apartado se llevará a cabo el diseño de un sistema digital formado por tres tipos de modulación a elegir: DBPSK, DQPSK y GMSK. A diferencia del anterior apartado, para realizar los módulos transmisores y receptores se han utilizado los bloques moduladores y demoduladores proporcionados por la arquitectura GNU Radio. Básicamente, estos módulos engloban la mayor parte de los bloques que se han visto en el apartado anterior, simplificando la tarea de diseño. En cada tipo de modulación se tomarán las medidas oportunas para llevar a cabo la transmisión del fichero de audio.

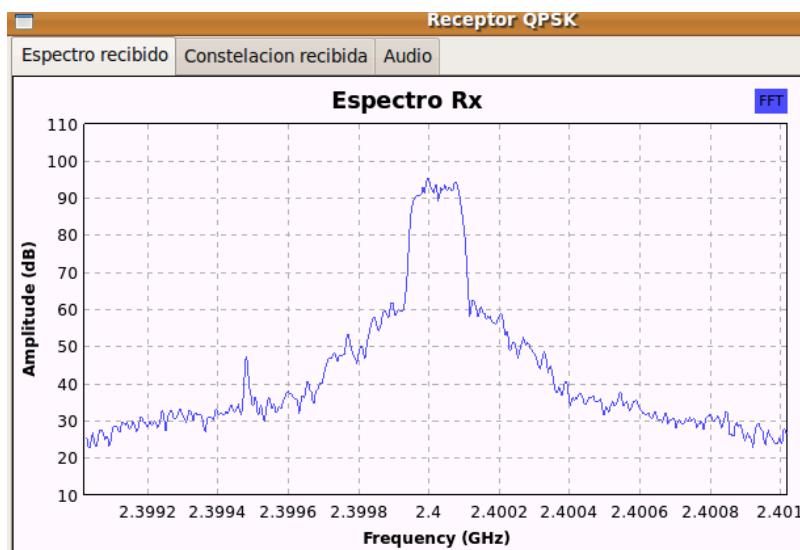


Figura 4-58 Espectro recibido distorsionado por saturación de entrada



Figura 4-59 Espectro recibido distorsionado por ganancia de recepción

#### 4.2.1 Transmisor digital

Se mostrará en primer lugar, como hasta ahora, el módulo transmisor, cuyo esquemático se incluye en la Figura 4-61.

La **fuente**, como se ha comentado, es el fichero de audio *wav* diezmado por 4 al igual que en el apartado anterior, aunque opcionalmente se añaden fuentes de señal diferentes las cuales son un coseno a una tasa de muestreo de 8 kHz (para limitar la tasa binaria necesaria sin necesidad de diezmar) y una fuente de tipo fichero cuyo único parámetro es la ruta del fichero a enviar.

Tras esto, se selecciona la modulación escogida gracias al **bloque selector** que viene a continuación. Su configuración es la de la Figura 4-60.

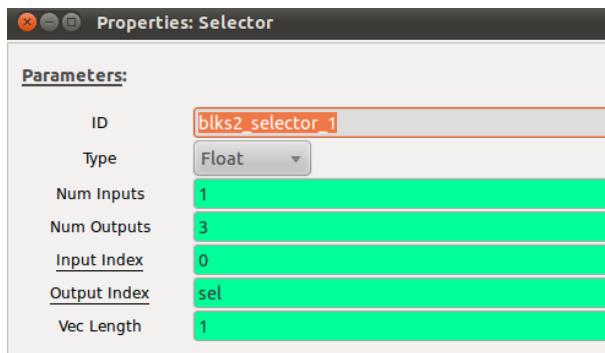


Figura 4-60 Propiedades bloque selector

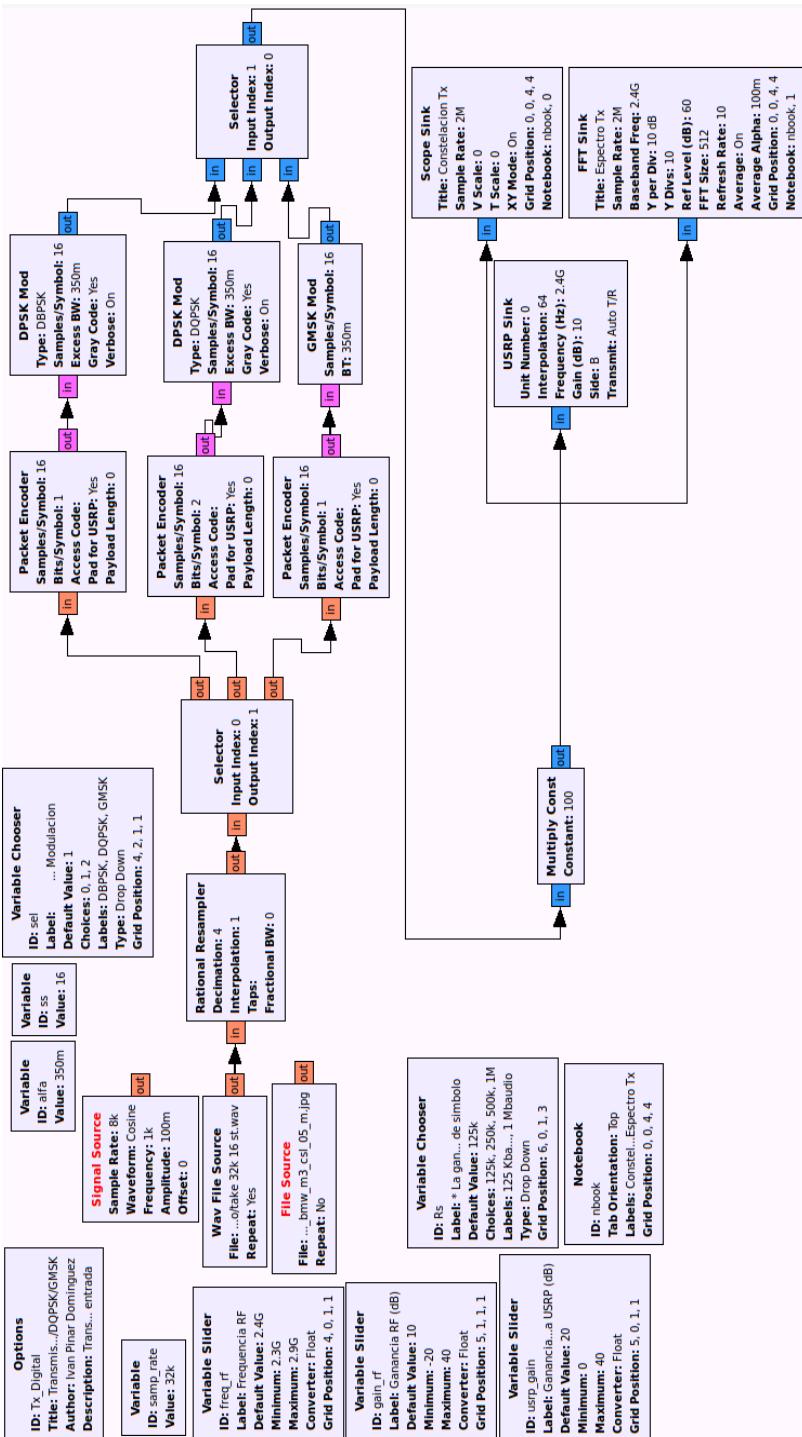


Figura 4-61 Esquemático del transmisor digital

Luego, con la variable *sel* se seleccionará la modulación deseada. Esta variable es de tipo *chooser*, es decir, sólo tiene unos valores predefinidos tal y como se aprecia en la siguiente figura:

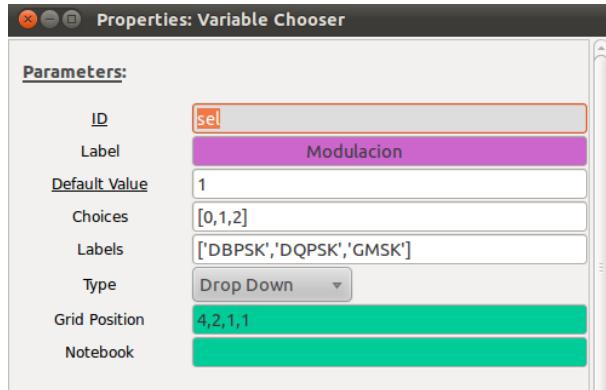


Figura 4-62 Propiedades variable *sel*

En principio el diseño está pensado para poder cambiar de modulación desde la interfaz gráfica pero es conveniente seleccionar la modulación variando el parámetro *Default Value* de la variable *sel* para cada prueba con diferente modulación.

Después aparece el bloque codificador cuyo parámetro principal es el número de bits por símbolo, que será 1 para las modulaciones DBPSK y GMSK y 2 para la modulación DQPSK.

A continuación se inserta el **bloque modulador** oportuno que incorpora todos los bloques necesarios para modular la señal, desde el mapeo de Gray hasta la aplicación del filtro raíz de coseno alzado. La configuración de cada uno de estos bloques aparece en las siguientes figuras. En primer lugar se muestra la configuración del bloque DBPSK:

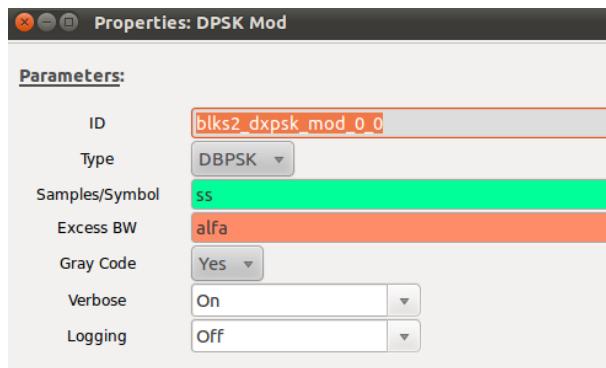


Figura 4-63 Parámetros del modulador DBPSK

Los parámetros configurables, como se puede apreciar, son el número de muestras por

símbolo (*Samples/Symbol*), el factor de *Roll-off* (*Excess BW*) y la opción de elegir codificación de Gray, obviando el diseñador todos los subbloques contenidos dentro del modulador.

El modulador DQPSK es de la misma forma pero variando el campo *tipo* a DQPSK. Por último, el modulador GMSK es similar, aunque ahora el parámetro relacionado con el ancho de banda es el valor de  $B_b \cdot T$ , que también se controla con la variable alfa como se puede apreciar en la siguiente figura:

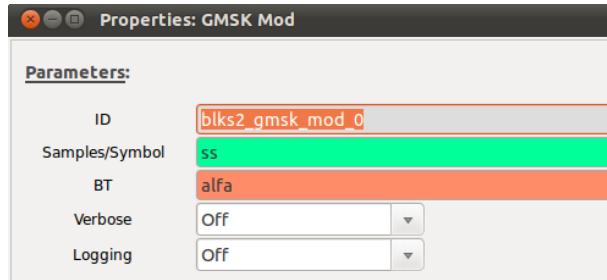


Figura 4-64 Parámetros del modulador GMSK

Finalmente, se añade al esquemático un **amplificador software** y el **sumidero USRP**, cuya configuración es la misma que en el apartado anterior, tal y como se aprecia en la Figura 4-65.

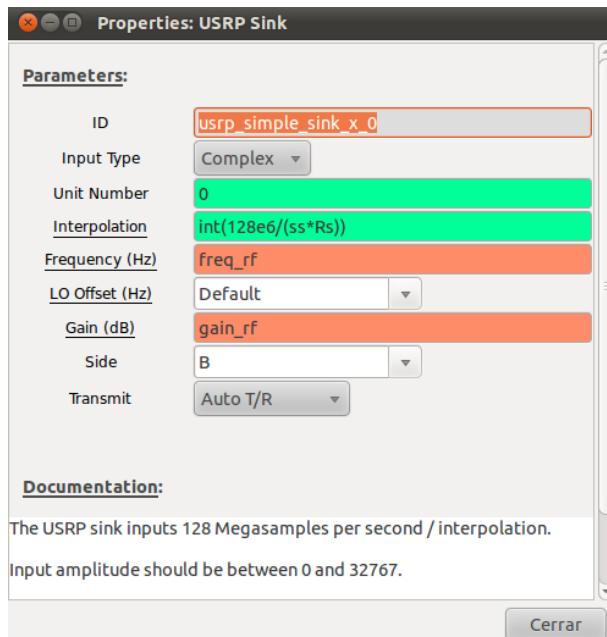


Figura 4-65 Parámetros del sumidero USRP

A continuación aparecerán los resultados y los parámetros seleccionados para cada modulación:

- **DQPSK:** Se comenzará con el sistema DQPSK para compararlo con el diseño implementado en el Apartado 4.1. Para lograr la tasa binaria necesaria para el envío del fichero *wav* submuestreado, es decir, 256 kbps (1024 kbps/4) se necesita una tasa de símbolos de aproximadamente 125 kbaudios, por lo que el número de muestras por símbolo puede ser como máximo de 16 para no incumplir el factor de interpolado mínimo de 64. En esta situación, los resultados de la constelación y el espectro transmitido se incluyen en la Figura 4-67.

Se puede apreciar como los resultados obtenidos son los mismos que en el apartado anterior pero ahorrándose la inclusión de varios bloques.

- **DBPSK:** Ahora se analizará el modulador DBPSK. Como este tipo de modulación es menos eficiente enviándose 1 bit por cada símbolo, para poder lograr la tasa de 250 kbps se necesita una tasa de símbolos de 250 kbaudios, con el consiguiente aumento del ancho de banda. Esto también implica que el número de muestras por símbolo máximo es de 8 para no incumplir el interpolado mínimo. La constelación y el espectro para este tipo de modulación son los de la Figura 4-67 y Figura 4-67, donde se han mantenido la misma configuración anterior. Se puede apreciar como ahora el ancho de banda es de unos 400 kHz, aproximadamente  $(1+\alpha) \cdot R_s$ .

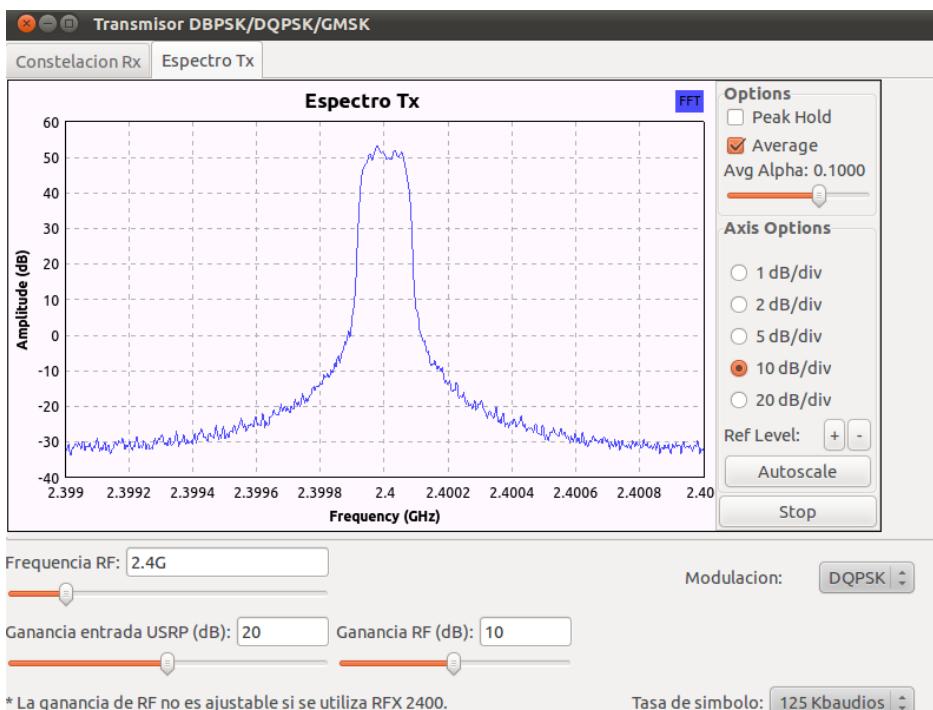


Figura 4-66 Espectro transmitido con modulación DQPSK

- **GMSK:** Por último se analizará la modulación GMSK. En este caso también se envía un bit por símbolo por lo que nuevamente se necesita una tasa de 250 kbaudios. Los resultados, con la misma configuración de la placa secundaria que en ejemplos anteriores, son los de la Figura 4-69.

Teóricamente, el primer nulo de la densidad espectral se debe dar para una frecuencia de  $0.75 \cdot R_b$  respecto de la portadora ([5], figura 7.19), como el flujo binario es 250 kbps, este nulo se debe producir a 187,5 kHz respecto de la portadora, lo cual coincide con lo que se aprecia en la figura, con un nivel 40 dB inferior respecto al máximo. El ancho de banda entre nulos es por tanto de 375 kHz.

Como el producto  $B_b \cdot T$  es 0.35, el ancho de banda a 3 dB es el siguiente:

$$B = 2 \cdot B_b = 2 \left( \frac{0.35}{T} \right) = 2 \left( \frac{0.35}{1/(250 \cdot 10^3)} \right) = 175 \text{ kHz} \quad (4-3)$$

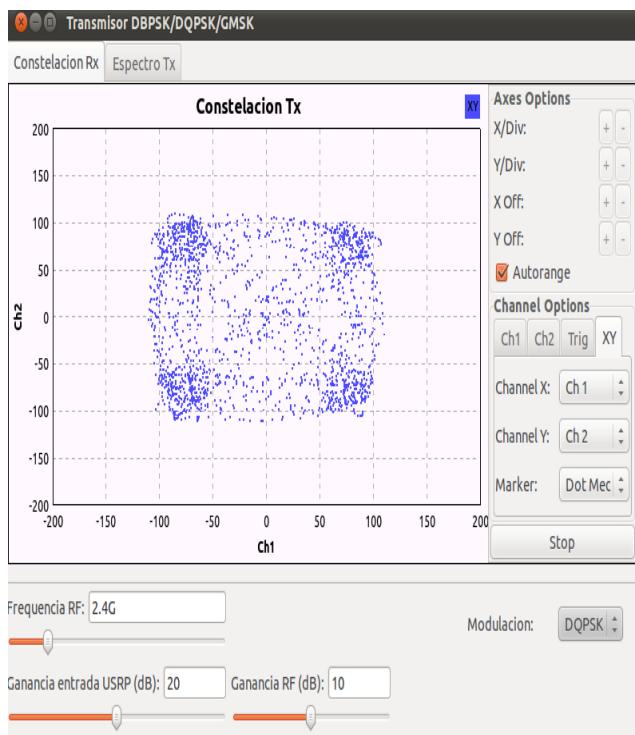
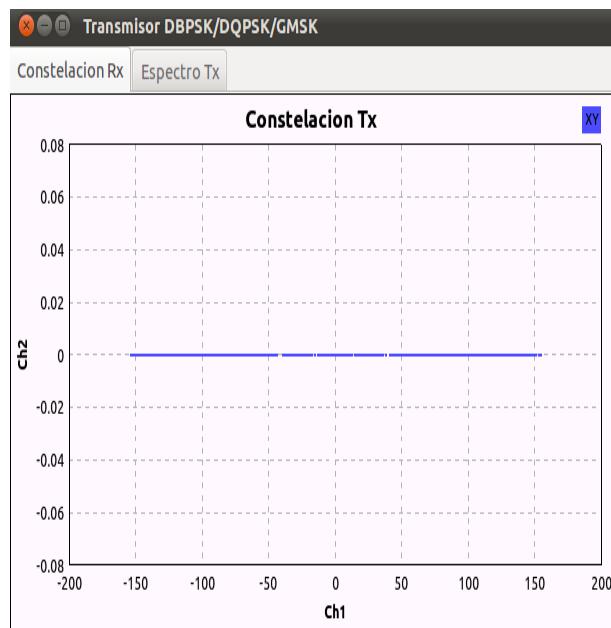
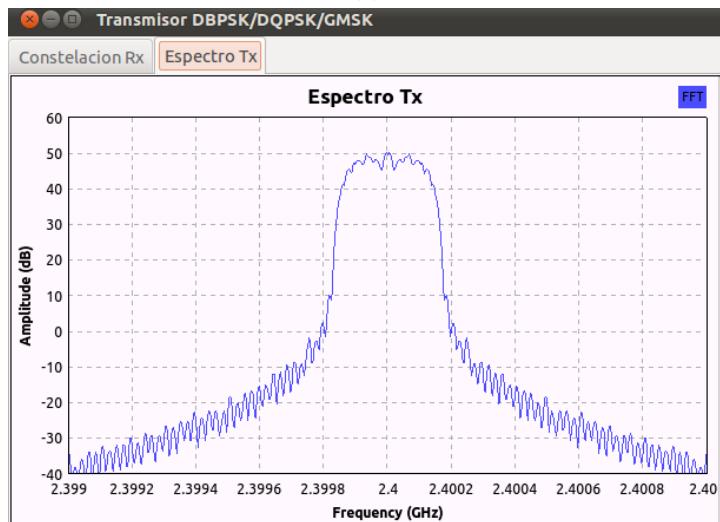


Figura 4-67 Constelación con modulación DQPSK



(a)



(b)

Figura 4-68 (a) constelación y (b) espectro transmitido con modulación DBPSK

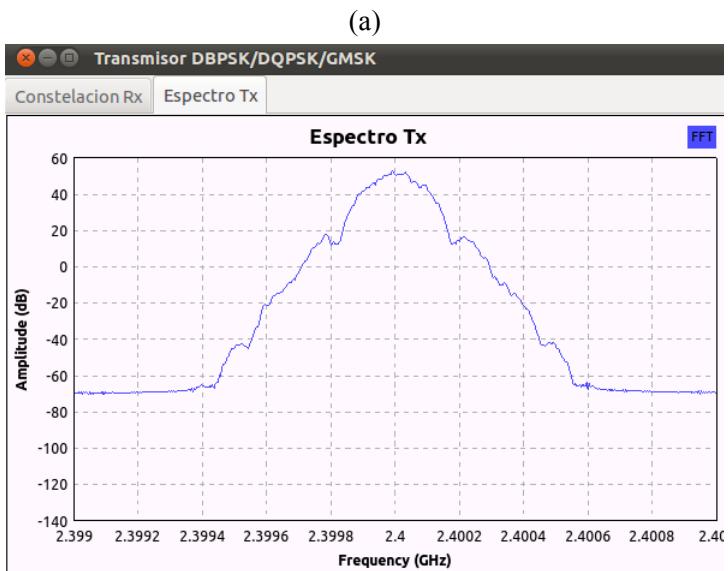
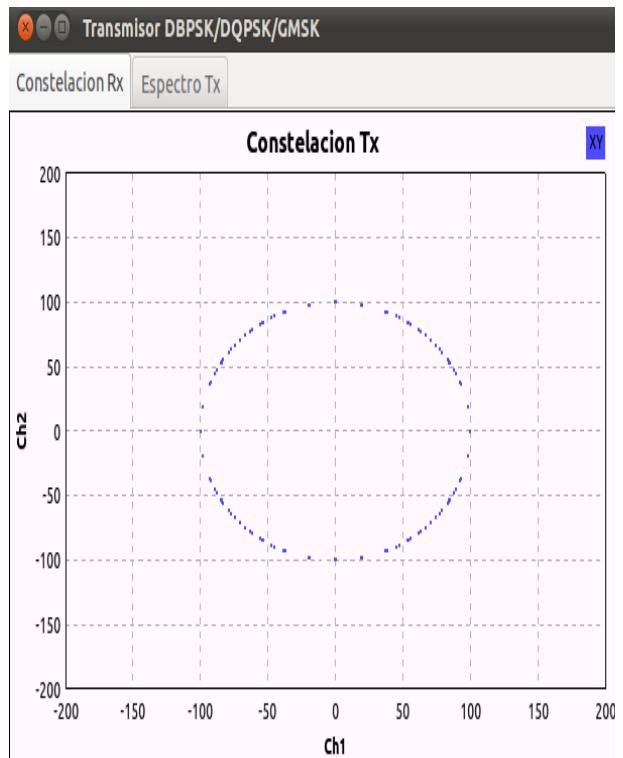


Figura 4-69 (a) constelación y (b) espectro transmitido con modulación GMSK

### 4.2.1 Receptor digital

A continuación se mostrará el módulo implementado para recibir la señal enviada por el transmisor del apartado anterior. El esquemático sigue tiene un proceso similar pero invertido tal y como aparece en la Figura 4-71.

La **fuent**e es la proporcionada por el USRP con la misma configuración que el receptor QPSK, como se puede apreciar en la siguiente figura:

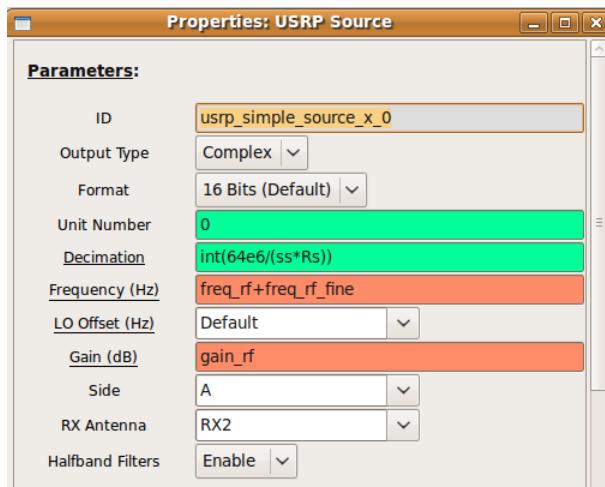


Figura 4-70 Parámetros de la fuente USRP

Tras la fuente aparece el **bloque selector** para seleccionar la modulación deseada. Cada uno de estos **bloques demoduladores** sustituye a varios bloques necesarios para la demodulación, incluyendo también un bloque de sincronización tal y como se vio en el Apartado 4.1.2. A continuación se analizan los parámetros de cada bloque demodulador. Se comenzará por el bloque demodulador DBPSK, en la Figura 4-72.

Se puede apreciar como incluye parámetros generales como el número de muestras por símbolo, el valor de *Roll-off* y parámetros de sincronización necesarios para el bloque *MPSK Receiver* incluido dentro de este bloque que, como se ha comentado, servirán para la sincronización temporal y ajustarán el NCO del USRP para la sincronización en frecuencia.

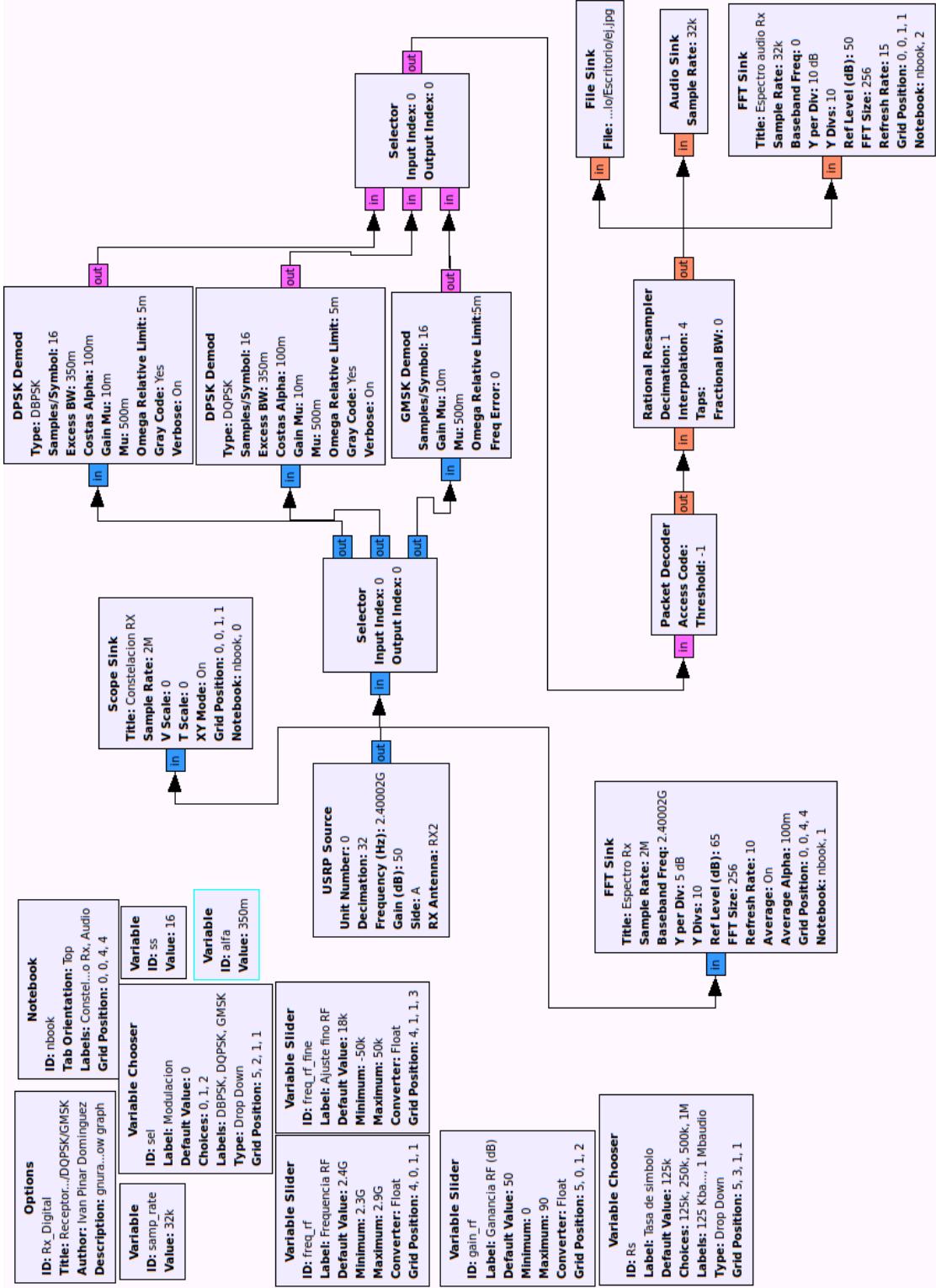


Figura 4-71 Esquemático del receptor digital

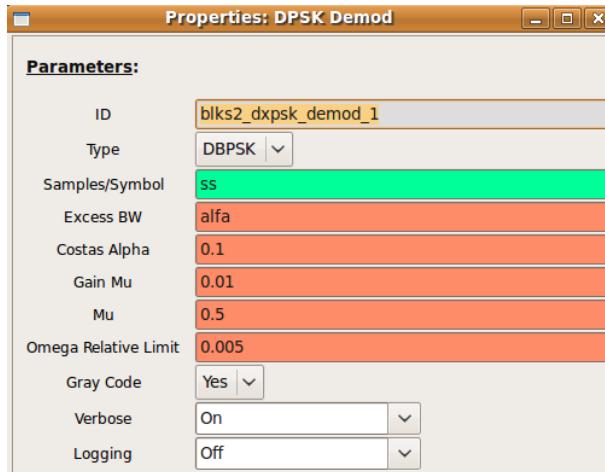


Figura 4-72 Parámetros del demodulador DBPSK

El demodulador DQPSK y el GMSK tienen la misma estructura que el DBPSK, es decir, los mismos parámetros configurables. Para el primero sólo se cambia el campo *tipo* y el segundo pertenece a un bloque distinto pero con las mismas características:

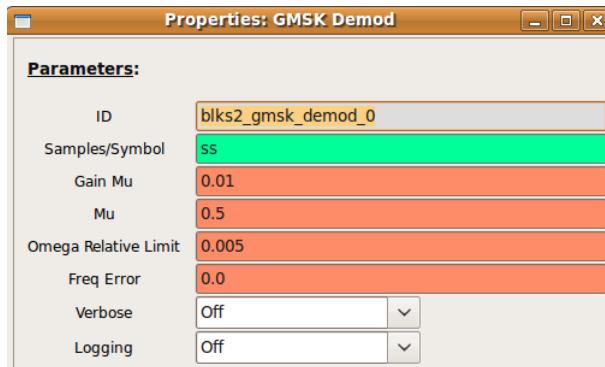
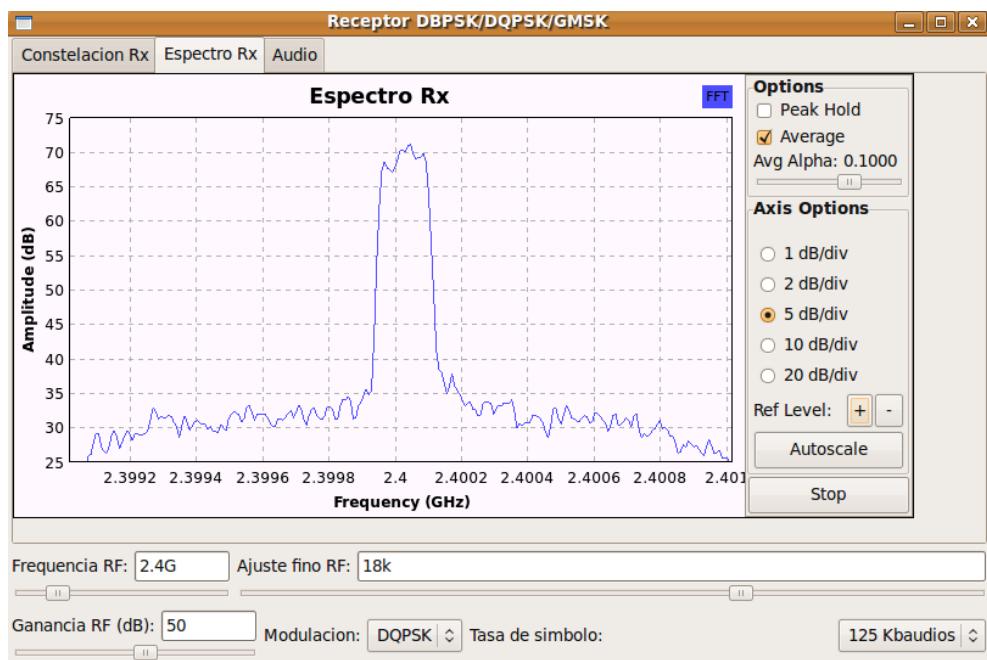


Figura 4-73 Parámetros del demodulador DBPSK

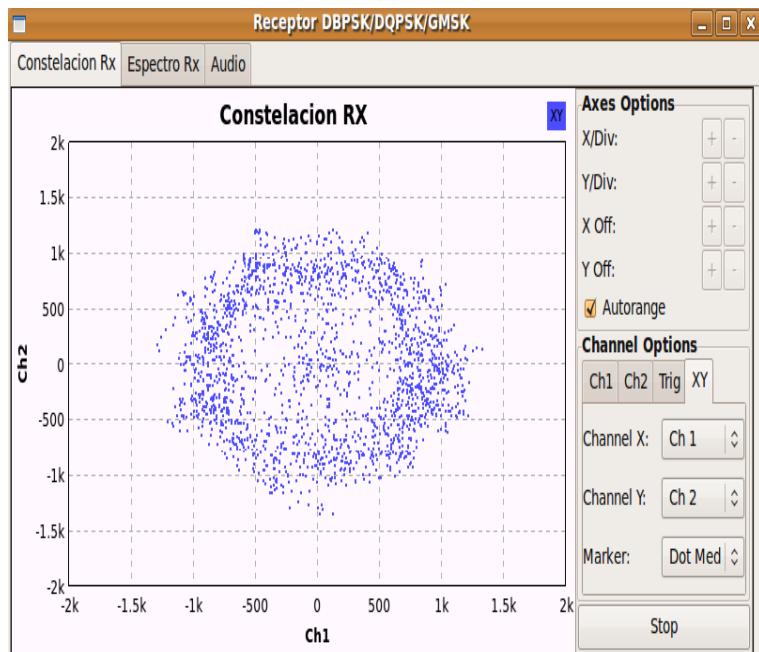
Tal y como se comentó en el análisis del transmisor digital, es recomendable seleccionar el tipo de modulación con el parámetro *Default Value* de la variable *sel* por posibles problemas de carga computacional.

Tras pasar el flujo de señal por el demodulador correspondiente, se decodifica la información y se envía a la **tarjeta de sonido** y a un **sumidero de tipo fichero** en caso de que se desee enviar a un archivo de cualquier formato.

En cuanto a las visualizaciones, como el demodulador consta de un único bloque sólo se podrá visualizar el espectro y la constelación de la señal recibida, antes del demodulador, y no de ningún bloque intermedio de la demodulación.



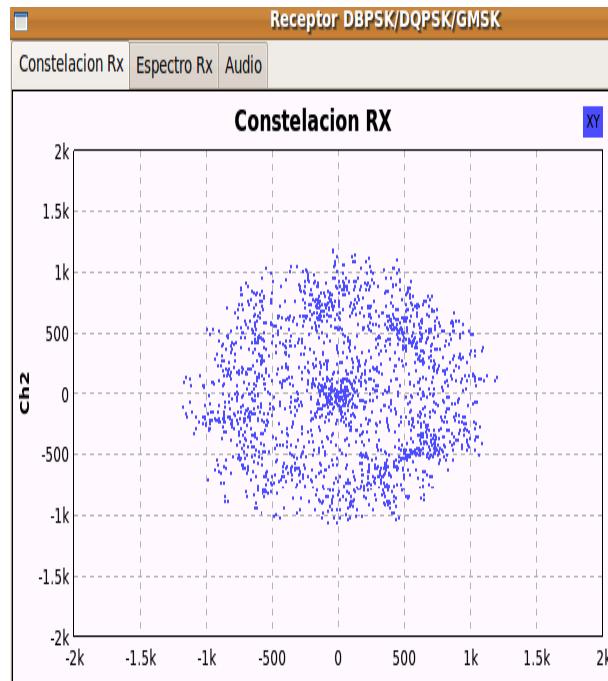
(a)



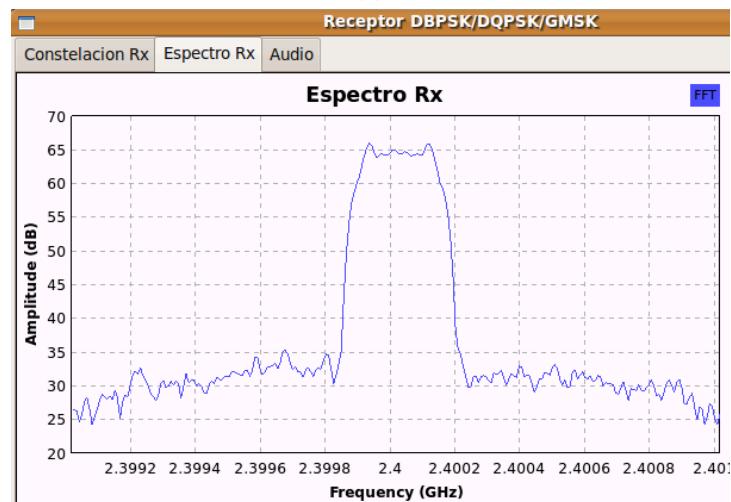
(b)

Figura 4-74 (a) Espectro recibido y (b) constelación con modulación DQPSK

Los resultados obtenidos para cada tipo de modulación se irán mostrando a continuación:

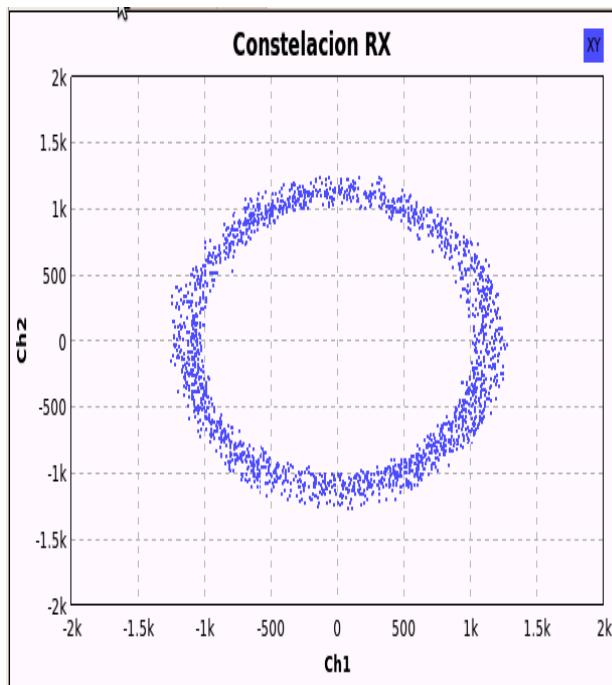


(a)

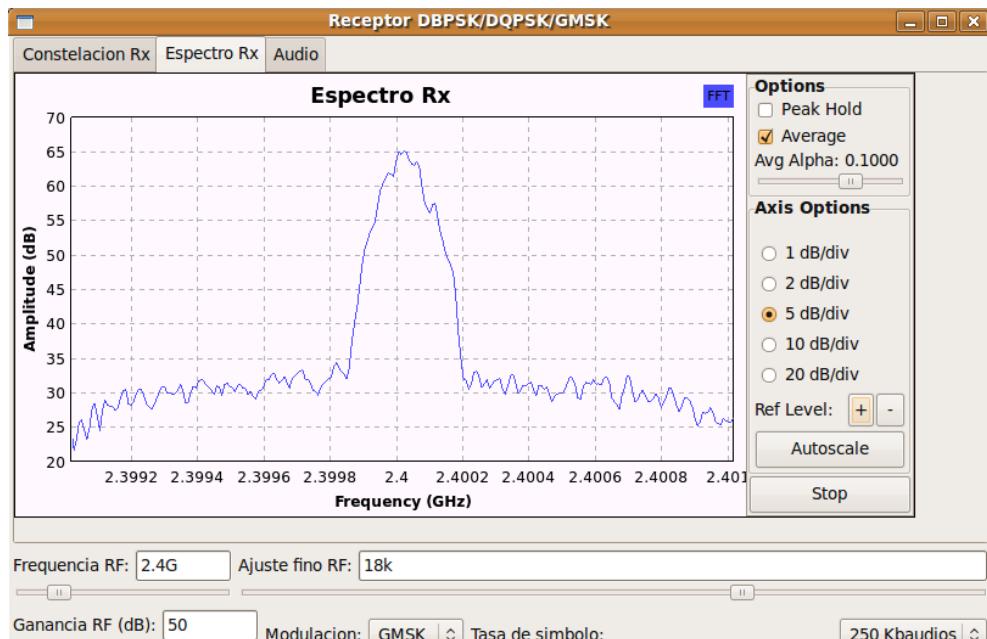


(b)

Figura 4-75 (a) constelación y (b) espectro recibido con modulación DBPSK



(a)



(b)

Figura 4-76 (a) constelación y (b) espectro recibido con modulación GMSK

- **DQPSK:** Como se sabe del análisis del módulo transmisor, la tasa de símbolos será de 125 kbaudios con un número de muestras por símbolo de 16. La constelación y espectro recibido se muestra en la Figura Figura 4-74. En la constelación se puede apreciar que la señal recibida está afectada de multitrayecto y error, deriva, de frecuencia. El espectro recibido tiene un ancho de banda de unos 200 kHz como cabía esperar y la desviación en frecuencia del transmisor está entre 18 y 20 kHz.
- **DBPSK:** Con este tipo de modulación se debe enviar a una tasa de 250 kbaudios, por lo que el número de muestras por símbolo máximo ahora es de 8. La constelación y espectro recibido se incluyen en la Figura 4-75, con igual configuración que en anterior caso. Lo único destacable en estos resultados es el aumento del ancho de banda al doble que en el caso anterior, es decir a unos 400 kHz como se preveía. La constelación igualmente está afectada por multitrayecto y offset en frecuencia, por lo que difiere de lo ideal (una línea entre los dos puntos de la constelación BPSK).
- **GMSK:** Al igual que en el caso anterior, es necesario transmitir a 250 kbaudios con un número de muestras por símbolo de 8. Los resultados se aprecian en la Figura 4-76. Tal y como se observa en la constelación, la señal recibida tiene amplitud constante, por lo que es más robusta frente a no linealidades. En cuanto al espectro recibido, el ancho de banda entre nulos es de 375 kHz como cabía esperar.

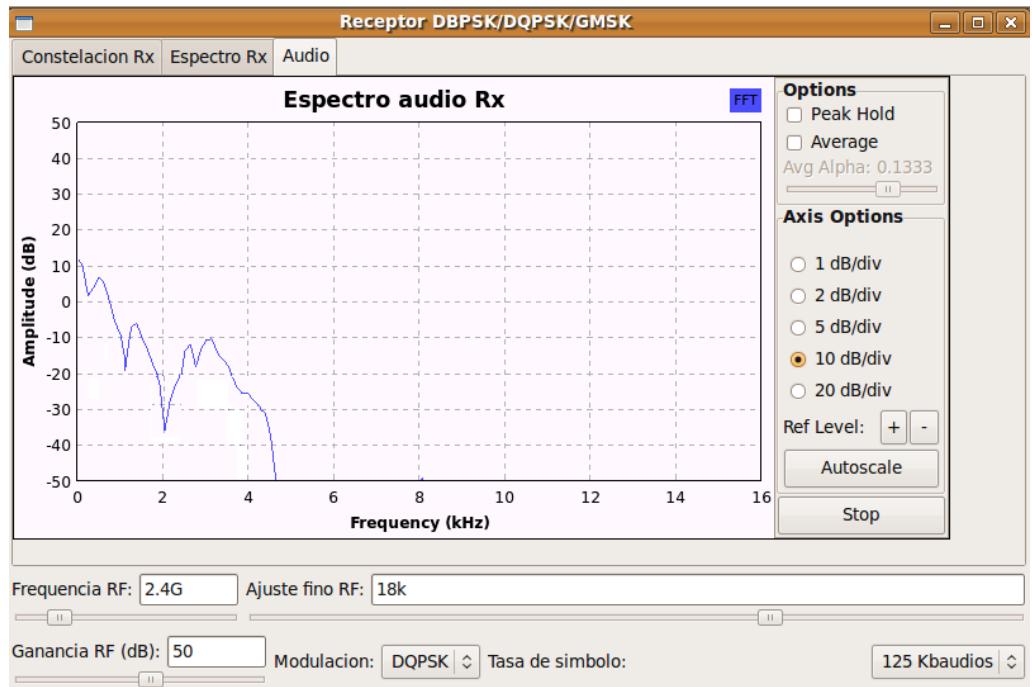


Figura 4-77 Espectro de audio recibido

En cuanto a la fidelidad de la señal de audio percibida, mejora para las modulaciones DQPSK y DBPSK frente a la modulación GMSK, aunque obviamente sigue teniendo un ancho de banda de 4 kHz por el proceso de interpolación en recepción. En la Figura 4-77 se muestra el espectro de audio para el caso DQPSK (aunque es similar en el resto).

### 4.3 Cálculo de Bit Error Rate

Tras explicar varios sistemas de comunicaciones digitales, se va a realizar el envío de un fichero, en particular una imagen en formato *jpg*. Para evitar en la medida de lo posible los errores de transmisión, los módulos transmisor y receptor se intentarán ejecutar simultáneamente. En la siguiente figura aparecen la imagen original y la recibida:



Figura 4-78 Imagen transmitida (izquierda) y recibida (derecha)

Para esta transmisión se ha seleccionado la modulación DQPSK con los mismos parámetros que los utilizados en la transmisión del fichero *wav*. Para que se pueda calcular la tasa de errores en la transmisión, se ha diseñado el esquemático en la Figura 4-80. El bloque que realiza el cálculo de la BER es el bloque *Error Rate*, que toma como referencia la imagen original y como entrada la imagen recibida y calcula la tasa de error cada 1000 muestras. Tras ejecutarlo se obtiene el resultado en la Figura 4-79.



Figura 4-79 Resultados de BER

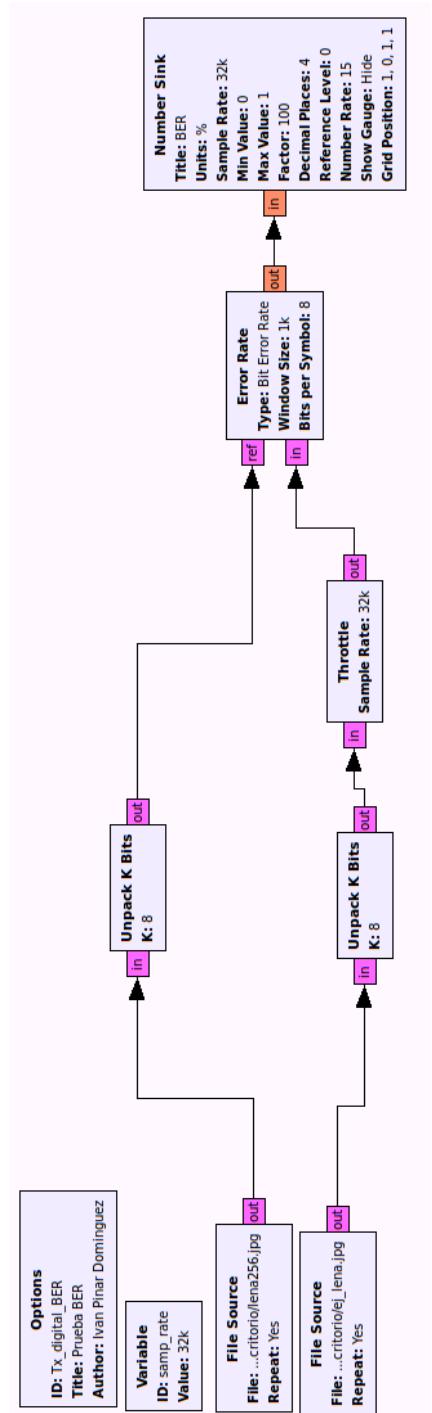


Figura 4-80 Esquemático para evaluación de la BER

## 4.1 Sistema OFDM

En este apartado se implementará un sistema OFDM compuesto por transmisor y receptor al igual que en diseños anteriores. En los apartados sucesivos se analizarán ambos módulos y las medidas adoptadas para transmitir la señal de información deseada.

### 4.1.1 Transmisor OFDM

El esquemático del transmisor OFDM aparece en la Figura 4-81.

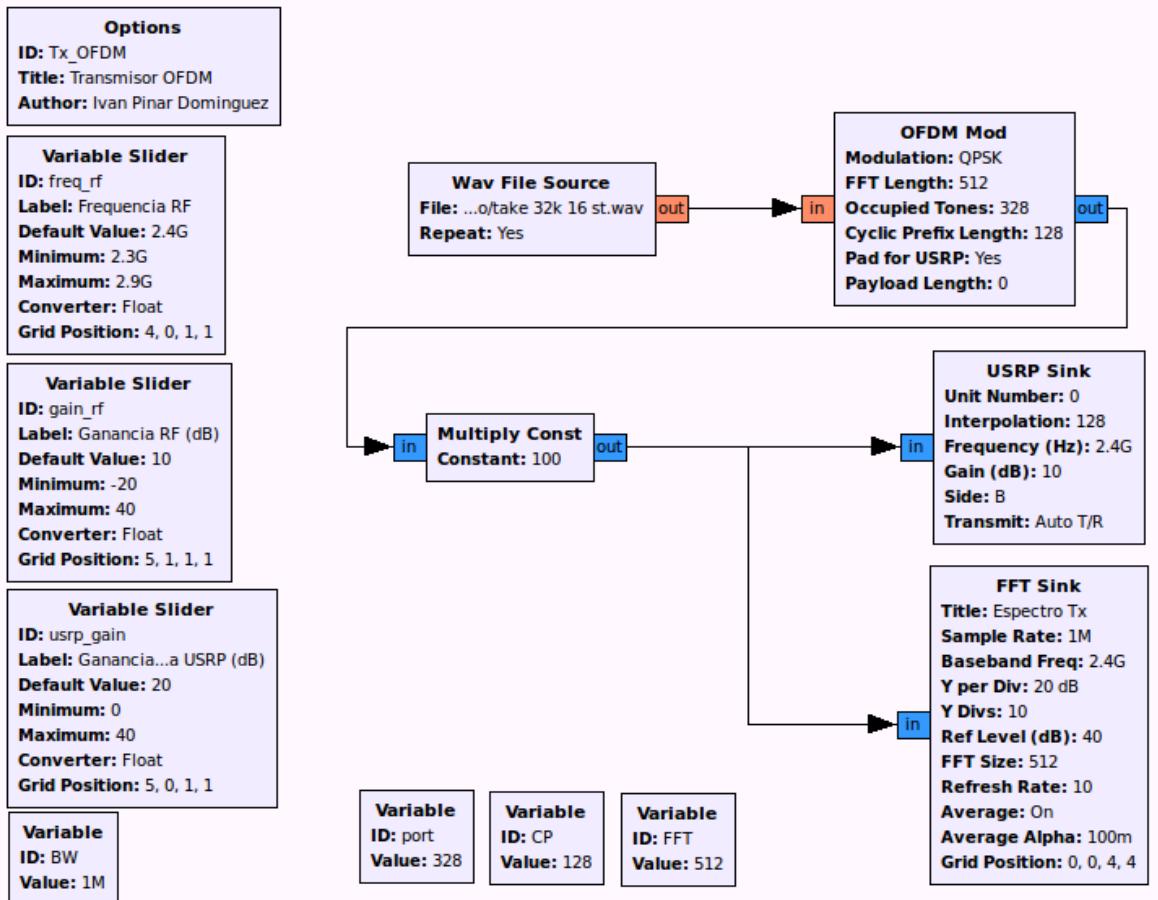


Figura 4-81 Diagrama de bloques del transmisor OFDM

Comenzando por la **fuente**, la información proviene de un fichero *wav* al igual que en otros diseños realizados. La fuente se introduce en el **bloque modulador** de OFDM contenido dentro de la arquitectura GNU Radio y que engloba todos los subbloques necesarios para la modulación. Estos subbloques son los siguientes:



Figura 4-82 Estructura interna del modulador OFDM

La configuración del modulador aparece en la siguiente figura:

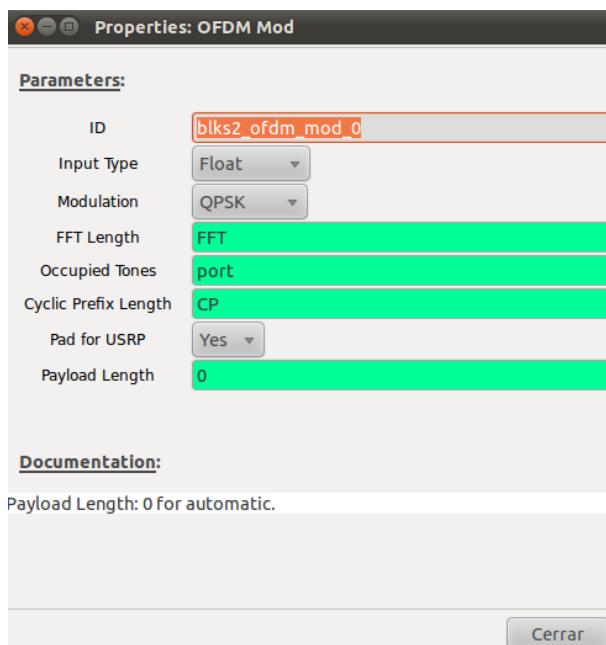


Figura 4-83 Parámetros del modulador OFDM

Como se puede apreciar, los parámetros configurables son el tipo de modulación de cada subportadora (desde BPSK hasta 256 QAM) la longitud de la FFT (IFFT en este caso) el número de portadoras (*Occupied Tones*) y el prefijo cíclico, dado en valor absoluto, no como una fracción de la longitud del símbolo que es lo típico. Estos tres últimos se controlan con las variables *FFT*, *port* y *CP* respectivamente.

Tras el modulador se incluye un **amplificador software** y finalmente el **sumidero**, es decir, el USRP cuya configuración se muestra en la Figura 4-84.

El parámetro destacable en este bloque es el factor de interpolación, que será función del ancho de banda de transmisión deseado controlado con la variable *BW*, que nos daría un tiempo de muestreo de  $1/T_m$ .

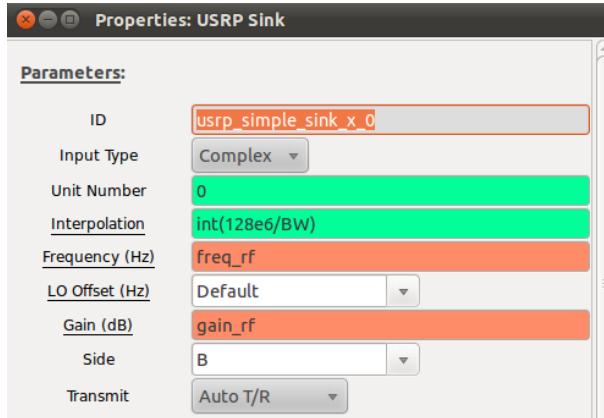


Figura 4-84 Parámetros del sumidero USRP

Antes de mostrar los resultados, se analizarán los valores tomados para las variables. Se necesita una tasa para transmitir el fichero *wav* de 1024 kbps si no se realiza ningún proceso de submuestreo en fuente, en este caso no será necesario porque no se necesita resolución espectral para el coseno alzado como en casos anteriores que era lo que limitaba la tasa de transmisión, ya que el factor de interpolado del USRP no debía ser inferior a 64. Si se escoge un ancho de banda de transmisión de 1 MHz, el factor de interpolado será 128 y no habrá ningún problema en este sentido. El régimen binario para la modulación OFDM se puede calcular conforme a la siguiente expresión ([5], ecuación 8.2.1):

$$R_b = \frac{1}{T_{s_{CP}}} \cdot N_{port} \cdot \log_2(M) \quad (4-4)$$

Donde:

- $T_s$  es el tiempo de símbolo sin incluir prefijo cíclico
- $T_{s_{CP}}$  es el tiempo de símbolo incluido el prefijo cíclico
- $N_{port}$  es el número de portadoras, el tamaño del bloque OFDM
- $M$  es el número de símbolos de la modulación escogida

El ancho de banda “efectivo” de la transmisión estará dado por la siguiente igualdad, que tiene en cuenta la relación entre el número de portadoras y la longitud de la FFT:

$$BW_{ef} = BW \cdot \frac{N_{port}}{FFT\_length} \quad (4-5)$$

Es decir, en realidad no se va a transmitir en el ancho de banda de 1 MHz especificado. Por su parte, la separación entre portadoras  $\Delta f$  es:

$$\Delta f = \frac{1}{T_s} = \frac{BW_{ef}}{N_{port}} \Rightarrow T_s = \frac{N_{port}}{BW_{ef}} = \frac{FFT\_length}{BW} \Rightarrow T_{s_{CP}} = T_s + CP \cdot T_s = (1+CP) \cdot \frac{FFT\_length}{BW} \quad (4-6)$$

Con esto ya se puede calcular el régimen binario a partir de los parámetros configurables del modulador OFDM quedando finalmente:

$$R_b = \frac{1}{(1+CP)} \cdot \frac{BW}{FFT\_length} \cdot N_{port} \cdot \log_2(M) \quad (4-7)$$

Si la modulación utilizada es un esquema QPSK ( $M=4$ ), el ancho de banda total es 1 MHz (no el efectivo), la longitud de la FFT es 512 y el prefijo cíclico es 1/4 para proteger la señal frente al multirayecto, el número de portadoras a utilizar es de 328 portadoras si se requiere un régimen binario de 1024 kbps.

Tras justificar los valores seleccionados, en la siguiente figura se muestra la ejecución del módulo transmisor, cuyo espectro es el siguiente:

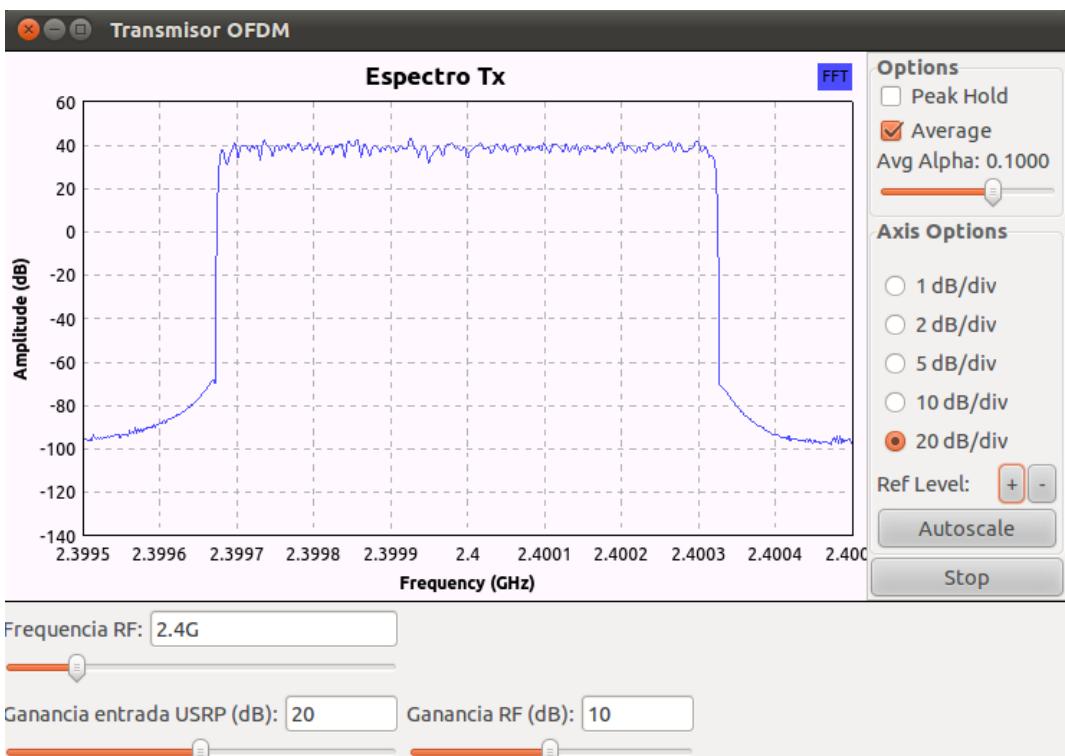


Figura 4-85 Espectro transmitido OFDM

En la figura se puede apreciar como no se transmite en todo el ancho de banda seleccionado en principio (1 MHz), sino que el ancho de banda ocupado por las portadoras es el ancho de banda “efectivo” definido anteriormente, tomando un valor de:

$$BW_{ef} = BW \cdot \frac{N_{port}}{FFT\_length} = 1\text{MHz} \cdot \frac{328}{512} = 640\text{kHz} \quad (4-8)$$

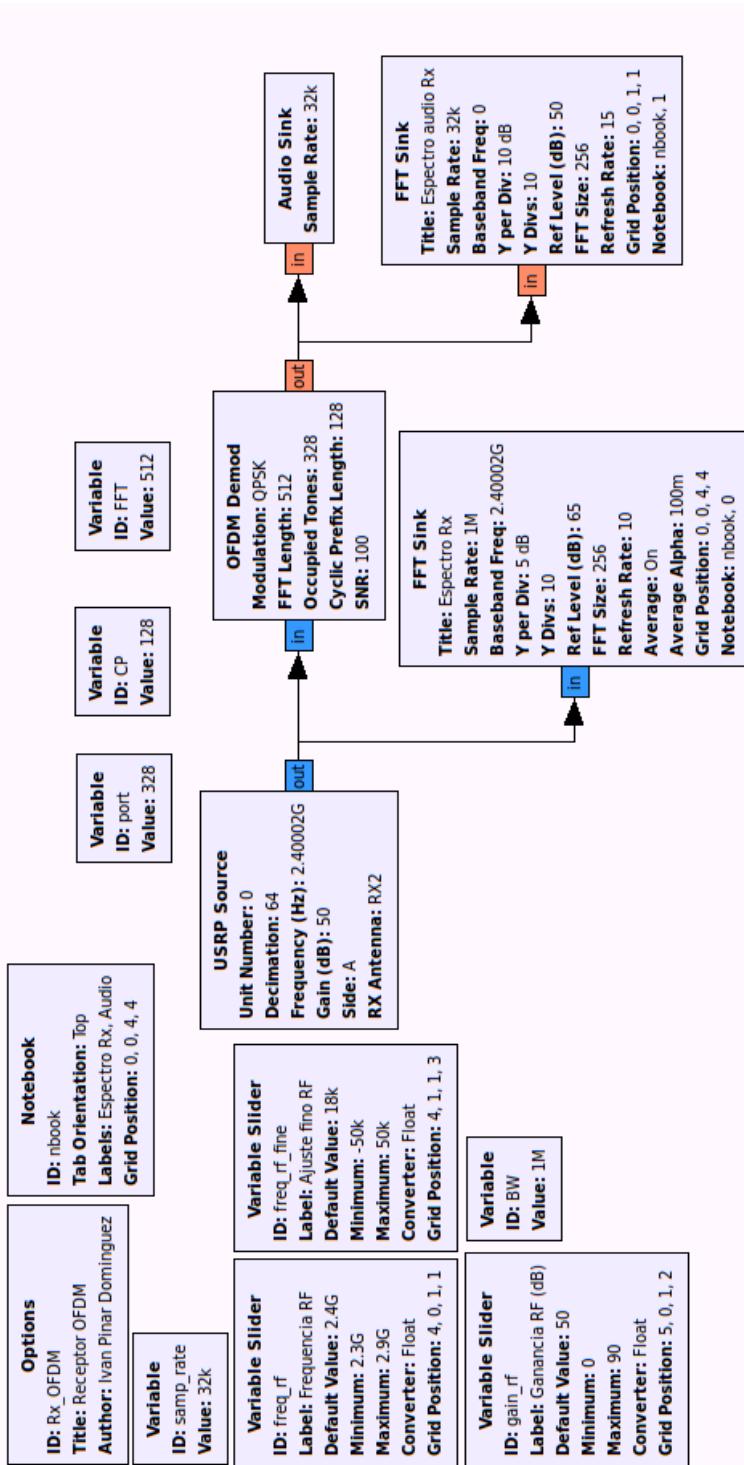


Figura 4-86 Diagrama de bloques del receptor OFDM

### 4.1.1 Receptor OFDM

Tras analizar el transmisor OFDM y justificar los valores seleccionados, se mostrará el diseño del receptor OFDM. El esquemático implementado es el que aparece en la Figura 4-86.

En primer lugar aparece la **fuent**e que será el USRP. La configuración es similar a la mostrada en otros casos pero ahora el factor de diezmado depende del ancho de banda de la transmisión, ver Figura 4-87.

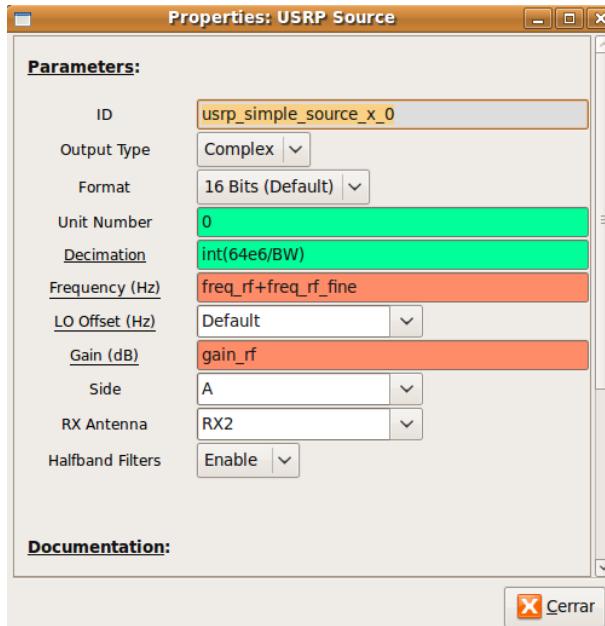


Figura 4-87 Parámetros de la fuente USRP

Tras la fuente se inserta el **bloque demodulador** de OFDM, contenido dentro de la arquitectura GNU Radio y que realiza el proceso inverso al modulador, teniendo el siguiente diagrama de subbloques:



Figura 4-88 Subbloques del demodulador OFDM

La configuración del demodulador aparece en la Figura 4-89. Al igual que antes, los

parámetros se controlarán a partir de variables. Para el parámetro *SNR* se selecciona un valor elevado para que el demodulador no introduzca ruido. Este parámetro es útil, por ejemplo, al realizar una simulación de OFDM que incluya transmisor y receptor en un mismo esquemático (sin necesidad de USRP) y se deseen comprobar las prestaciones del sistema al añadir ruido.

Tras el demodulador, la señal se envía a la **tarjeta de sonido**. Los resultados obtenidos se muestran a continuación. El espectro recibido es mostrado en la Figura 4-90.

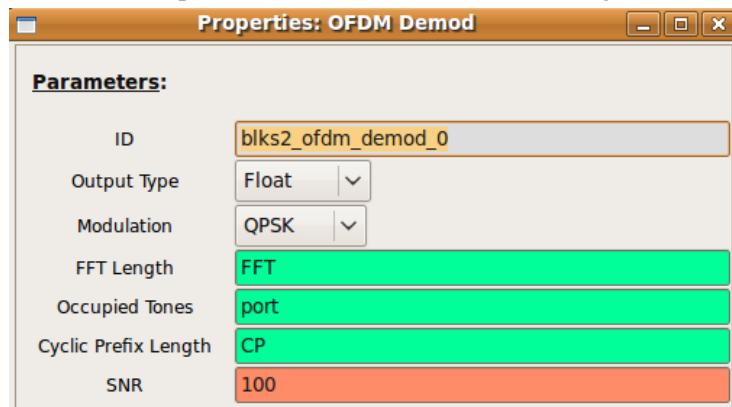


Figura 4-89 Configuración del demodulador OFDM

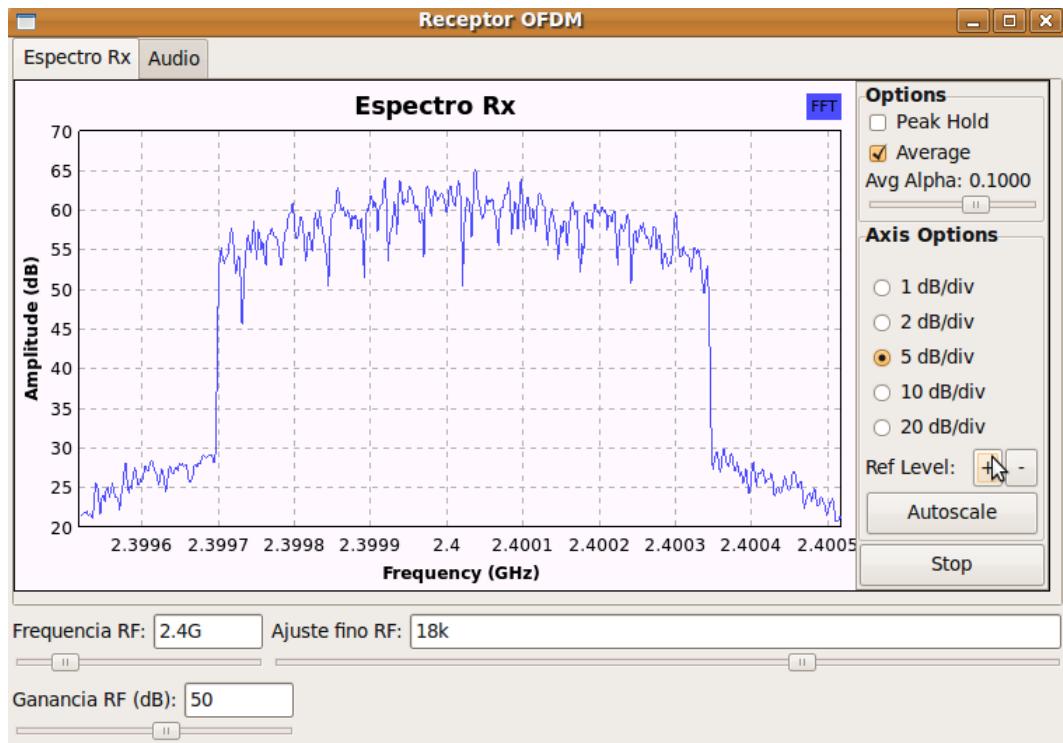


Figura 4-90 Espectro recibido OFDM

El espectro ocupa un ancho de banda de 640 kHz con un offset de frecuencia de unos 18 kHz como en los casos anteriores. En la visualización se puede ver cómo afecta el canal de comunicaciones multitrayecto a la señal transmitida, la cual tiene un espectro prácticamente plano y en la recibida aparece desvanecimiento selectivo. Además, se puede observar que, debido al filtro CIC interpolador, los extremos de la banda se ven atenuados respecto a la parte central.

El audio decodificado y enviado a la tarjeta de sonido se muestra en la siguiente figura:

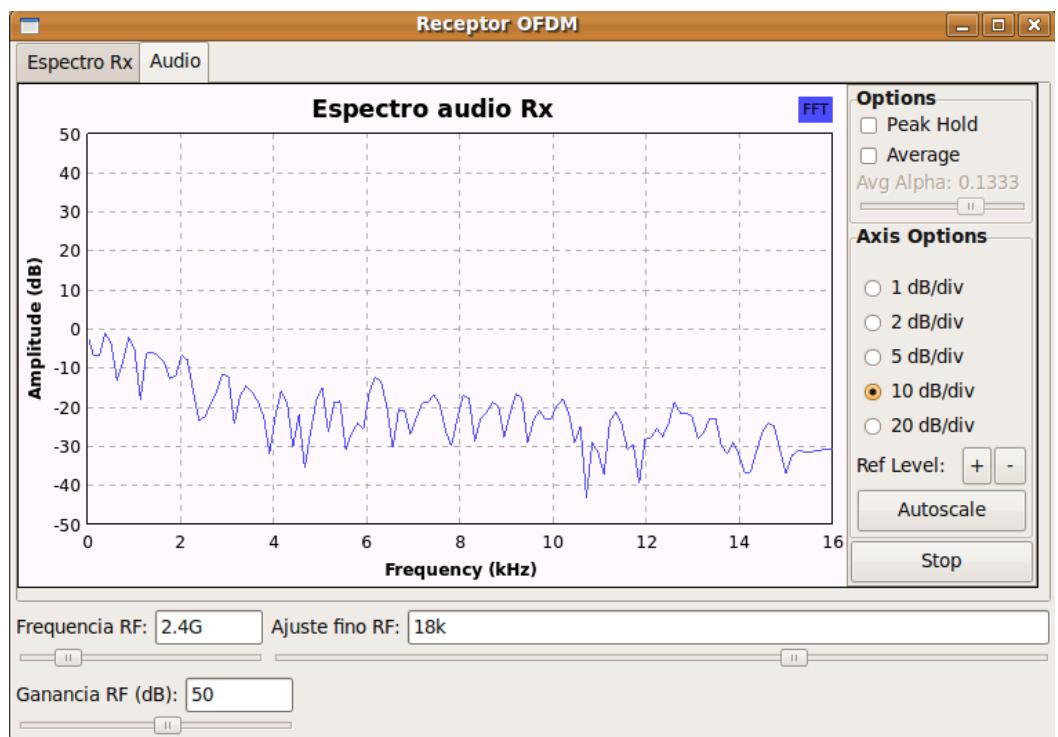


Figura 4-91 Espectro de audio

En este caso no se ha llevado a cabo ningún proceso de interpolado en recepción, por lo que la señal de audio ocupa el mismo ancho de banda que el fichero *wav* original, es decir, 16 kHz.

Por último, se analizara el flujo binario máximo que se podría alcanzar con el USRP y la modulación OFDM. Teóricamente, los valores límites serían los siguientes:

- $BW = 8 \text{ MHz}$  (interpolado = 16)
- $M = 256$
- $CP = 1/32$
- $N_{port}/FFT\_length = 1$

Lo que daría una tasa de:

$$R_b = \frac{1}{(1+CP)} \cdot \frac{BW}{FFT\_length} \cdot N_{port} \cdot \log_2(M) = 62,1 \text{Mbps} \quad (4-9)$$

En la práctica, los valores límites experimentales son los siguientes:

- $BW = 2 \text{ MHz}$
- $M = 16$
- $CP = 1/16$  (para comunicación correcta a 0.5 m de distancia)
- $N_{port}/FFT\_length = 0,82$

Estos valores proporcionan una tasa práctica de:

$$R_b = \frac{1}{(1+CP)} \cdot \frac{BW}{FFT\_length} \cdot N_{port} \cdot \log_2(M) = 6,17 \text{Mbps} \quad (4-10)$$

Es decir, un orden de magnitud inferior al resultado teórico.

## **Capítulo 5 USRP Y MATLAB**

---

Este capítulo trata del funcionamiento del *Universal Software Radio Peripheral* y el entorno *Matlab*. Trabajar con *GNU Radio* o *Matlab* es una decisión que en general depende de muchos aspectos.

*Matlab* es un lenguaje matemático interpretado que permite de forma sencilla simular sistemas de todo tipo. Ya sea mediante su programación en un script o utilizando, de forma parecida al *GNU Radio Companion*, *Simulink*.

Hace una década *Matlab* tenía tiempos de ejecución muy elevados comparados con otros lenguajes de programación como C++. Sin embargo una mayor depuración de *Matlab* unido una mejora sustancial en la carga computacional de los PC o workstations ha hecho que se utilice ampliamente en el diseño de sistemas de comunicaciones. A esto hay que añadir que, además, algunas funciones se pueden programar en C++ para llamarse desde *Matlab*.

Por otro lado, comparado con *GNU Radio Companion*, *Matlab* lleva utilizándose casi tres décadas en simulación de sistemas de comunicaciones. De forma que los *toolboxes* o librerías de funciones especializados en comunicaciones digitales están muy depurados. Hay que destacar que en la versión de *Matlab* 2011 se ha hecho un esfuerzo por mejorar estos *toolboxes*.

Ettus, fabricante de USRP, proporciona un driver que puede utilizarse en los sistemas operativos más importantes y que permiten enlazar programas con el USRP, entre ellos el *Matlab*.

Así, se pueden encontrar herramientas, como el *SDR4all*, que permiten escribir un script en *Matlab* para generar una secuencia de datos que mediante una función sencilla se mandan al USRP para su transmisión. De igual forma, se puede recibir desde el USRP para procesar luego las muestras recibidas por el USB, en el caso del USRP1, a través de una función de *Matlab*.

En *Simulink* también se han diseñado bloques que funcionan de igual forma que las fuentes y sumideros de *GNU Radio Companion*. Con lo que es inmediato simular un sistema a partir de los bloques de *Simulink*.

El mayor inconveniente, por otra parte, es el precio de las licencias de *Matlab*, *Simulink*, y de los *toolboxes*. Para evitar este coste, hay otras alternativas que aquí no se exploran, pero que se basarían en utilizar el driver facilitado por Ettus adaptada a otros programas, como el *SciLab*, un software libre similar a *Matlab*.

En este capítulo se comenzará explicando cómo añadir un *toolbox* del *USRP* para *Simulink* y así aprovechar el entorno de trabajo de éste. Se utilizará software para Microsoft Windows.

Por último se especificará como trabajar con el *USRP* y *Matlab* a partir del *toolbox* *SDR4all*, si bien es necesario para este último utilizar una versión de *Windows* de 32 bits.

## 5.1 Toolbox USRP para Simulink

El proyecto Simulink-USRP es un paquete de software Open Source que permite a los usuarios del *USRP* construir modelos en *Simulink* en tiempo real.

Este paquete de software ha sido desarrollado por el Communication Engineering Lab (CEL) del Karlsruhe Institute of Technology (KIT).

Para instalar este *toolbox* se seguirán los siguientes pasos:

1. **Instalación del driver USRP:** Para instalar el driver del *USRP* lo primero que hay que hacer es descargarse el programa libusb-win32 con el que se pueden instalar dispositivos que no sean *plug-and-play*. Se puede descargar de la siguiente URL: <http://sourceforge.net/apps/trac/libusb-win32/wiki>. Una vez instalado, conectar el *USRP* a un puerto USB y en la carpeta *libusb* del menú inicio pinchar en el icono *Inf-Wizard*, tras lo cual se selecciona *Next* y aparecen los dispositivos USB conectados, similar a como aparece en la siguiente figura:

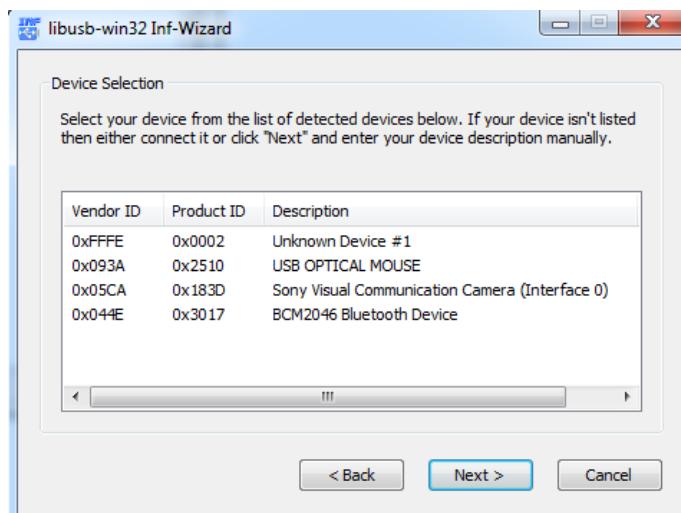


Figura 5-1 Dispositivos USB reconocidos por libusb

Se selecciona *Unknown Device* (será el USRP) y se pincha en *Next*. En el siguiente paso se selecciona la configuración del dispositivo, eligiendo el nombre del mismo.

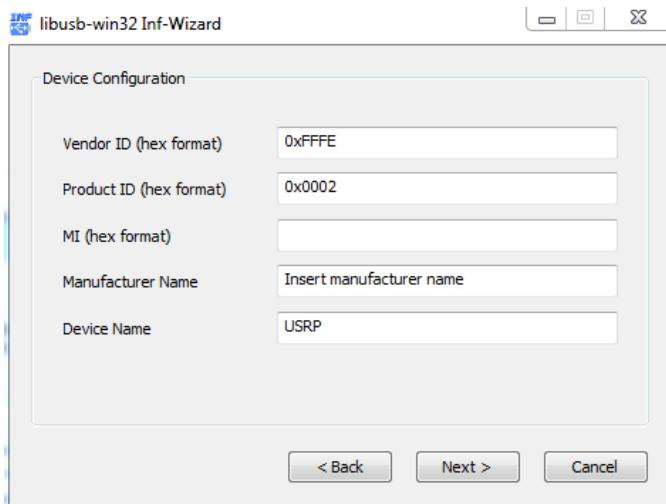


Figura 5-2 Propiedades del dispositivo USRP

Se guarda en cualquier lugar del disco duro y a continuación se selecciona *Install Now* para instalar el driver que se acaba de crear.

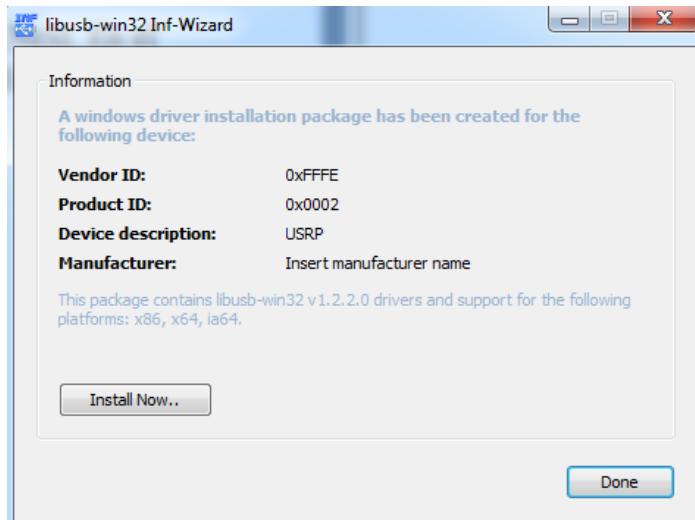


Figura 5-3 Instalación del driver

Tras esto, debe aparecer el driver si se abre el administrador de dispositivos:

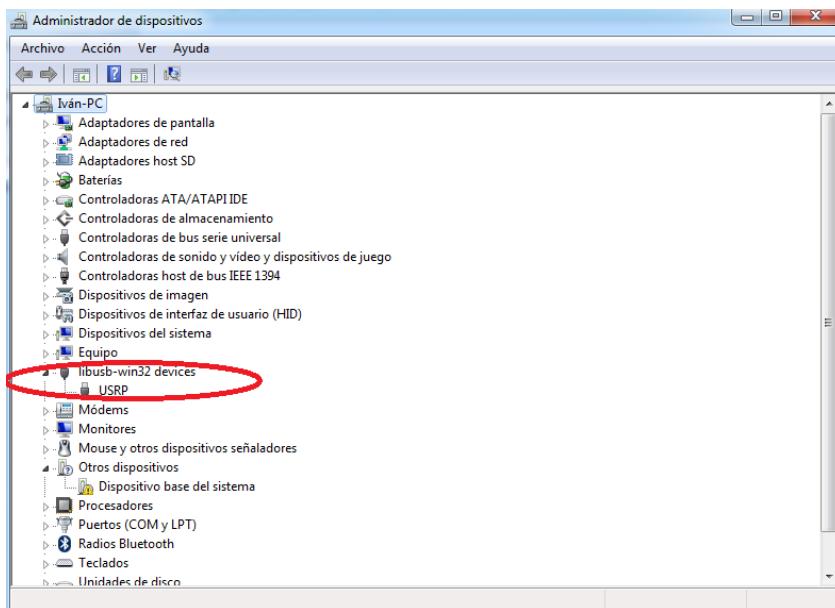


Figura 5-4 Driver instalado del USRP

2. **Seleccionar el compilador Microsoft Visual C++ en Matlab:** Por problemas de licencia, no hay una versión precompilada del proyecto *Simulink-USRP*. Sin embargo, compilarlo es tan fácil como ejecutar un comando desde *Matlab*. Antes de esto, hay que escoger el compilador correcto, para ello en el *prompt* de *Matlab* se escribe *mex -setup* y se selecciona el compilador. Se debería obtener algo similar a lo siguiente:

```
>> mex -setup
Please choose your compiler for building external interface (MEX) files:
Would you like mex to locate installed compilers [y]/n? y
Select a compiler:
[1] Lcc-win32 C 2.4.1 in D:\PROGRA~1\MATLAB\R2008a\sys\lcc\bin
[2] Microsoft Visual C++ 2005 in D:\Programme\Microsoft Visual Studio 8
[0] None
Compiler: 2
Please verify your choices:
Compiler: Microsoft Visual C++ 2005
Location: D:\Programme\Microsoft Visual Studio 8
Are these correct [y]/n? y
Trying to update options file: [...] mexopts.bat
From template:      [...]
Done . .
[...]
>>
```

Figura 5-5 Selección de compilador

Aparte de esto, se debe tener instalado *.NET Framework 3.5*, que se puede bajar desde la web de Microsoft. Si a la hora de compilar hubiera algún problema (posiblemente por

tener instalada una versión de 64 bits) también hay que instalar *Software Development Kit* de Microsoft para *Windows Server 2008* desde la página de Microsoft.

3. **Instalar Simulink-USRP:** Lo último que queda es instalar el toolbox *Simulink-USRP* en *Matlab*, para ello hay que descargarlo de <http://www.cel.kit.edu/download.php> y descomprimirlo en cualquier lugar del disco duro. En *Matlab* habrá que añadir la carpeta donde está el toolbox, para ello se selecciona *File > Set Path*, click en *Add Folder* y seleccionar la carpeta *X:\simulink-usrp\bin\* en primer lugar, donde *X* es la ruta donde se encuentre el toolbox y a continuación añadir *X:\simulink-usrp\blockset\*. Despues se selecciona *Save* y *Close*. Finalmente se compila el toolbox escribiendo en el *prompt* de Matlab ***usrpBuildBinaries***, obteniendo un resultado similar al siguiente:

```
>> usrpBuildBinaries
Running: mex -outdir 'X:\simulink-usrp\bin' -output libusrp LINKER=lib.exe[...]
D:\DOKUME~1\[\...]LOKALE~1\Temp\mex_R45T3d      emplib.x konnte nicht gefunden werden
Running: mex -outdir 'X:\simulink-usrp\bin' -output 'simulink-usrp' LINKER=lib.exe [...]
D:\DOKUME~1\[\...]LOKALE~1\Temp\mex_e5ubei      emplib.x konnte nicht gefunden werden
Running: mex -outdir 'X:\simulink-usrp\bin' [...]usrp_helper.cpp[...]
Running: mex -outdir 'X:\simulink-usrp\bin' [...]usrp_sink.cpp[...]
Running: mex -outdir 'X:\simulink-usrp\bin' [...]usrp_source.cpp[...]
Running: mex -outdir 'X:\simulink-usrp\bin' [...]stdcout2matlab.cpp[...]
>>
```

Figura 5-6 Compilación de Simulink-USRP

Ahora ya se pueden seleccionar los bloques del USRP en la librería Simulink:

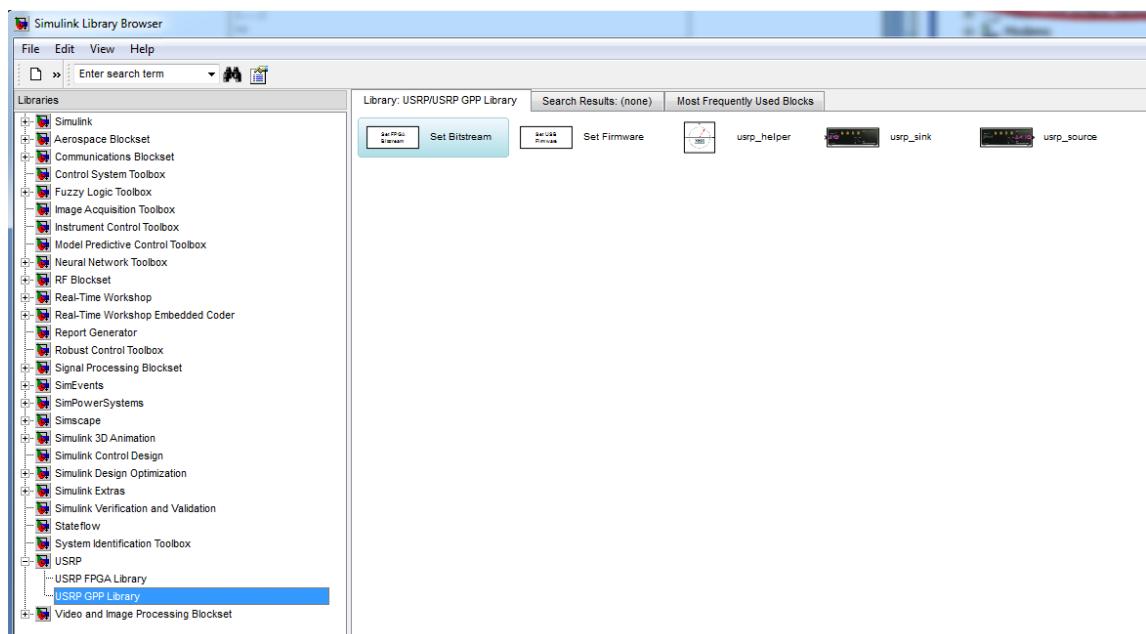


Figura 5-7 Bloques Simulink-USRP

### 5.1.1 Analizador de espectros con Simulink-USRP

A continuación se analizará el diseño del analizador de espectros implementado con el siguiente diagrama de bloques en *Simulink*:

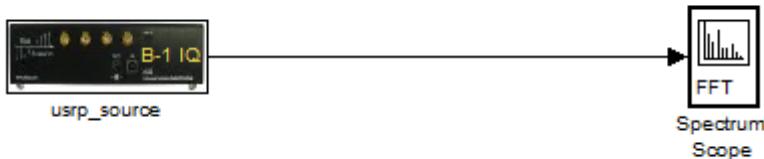


Figura 5-8 Diagrama de bloques del analizador en Simulink

El módulo fuente *usrp\_source* tiene la siguiente configuración:

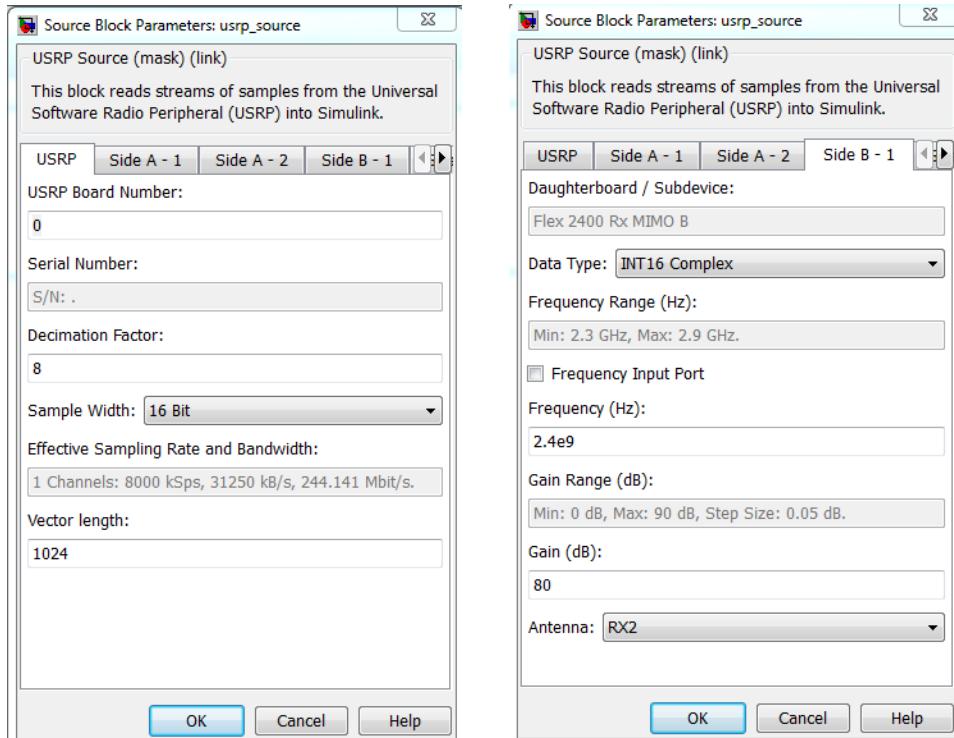


Figura 5-9 Configuración *usrp\_source*

Como se puede apreciar, se selecciona un factor de diezmado de 8, es decir, una tasa de muestras y ancho de banda de 8 MHz (64 MS/s / 8) y cada una de las muestras se cuantifica con 16 bits. En cuanto a la configuración de la tarjeta secundaria (RFX 2400) se selecciona el

tipo de datos complejo de 16 bits cada componente, una frecuencia de 2.4 GHz y una ganancia total de 80 dB.

El otro bloque involucrado en el diseño es el *Spectrum Scope* para visualizar la FFT de la señal recibida. Su configuración aparece en la Figura 5-10.

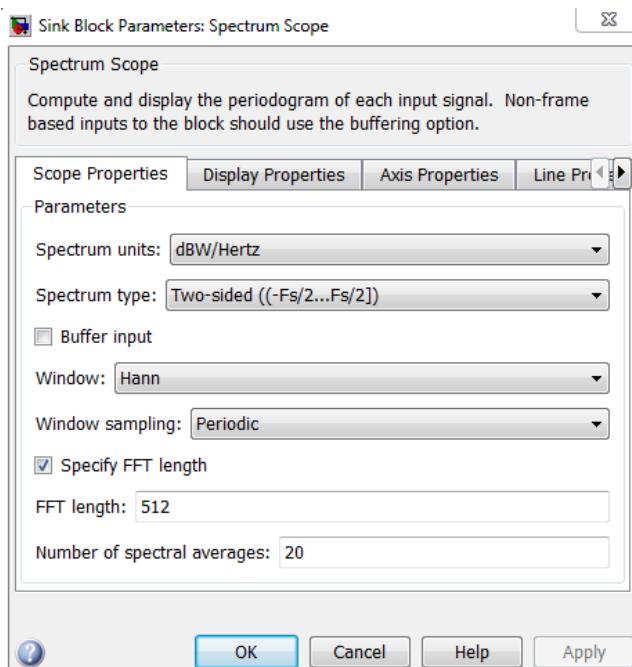


Figura 5-10 Parámetros del Spectrum Scope

Entre los parámetros a elegir se encuentran las unidades del espectro (en este caso se ha seleccionado la densidad espectral en dBW/Hz), el enventanado, la longitud de la FFT, el promediado,...

Tras ejecutar el diseño, el resultado obtenido se aprecia en la Figura 5-11, donde se puede apreciar la señal *WiFi* a 2.4 GHz.

## 5.2 Toolbox SDR4all

En este apartado se describirá el funcionamiento del *toolbox SDR4all* con el que se pueden realizar diseños en *Matlab* y transmitir/recibir con el *USRP*. El *toolbox* se encuentra disponible en la siguiente URL: <http://www.sdr4all.eu/Freeware.html#uiow> y no es necesario la instalación de ningún componente extra como sucedía en el caso anterior.

Para el funcionamiento de esta herramienta se necesita tener instalado en los ordenadores cualquier versión de *Windows* 32 bits.

En los diseños se utilizarán dos ordenadores, ambas con *Matlab*, utilizando una para transmitir las señales a partir del *USRP* y otra para recibir la señal transmitida a partir de otro *USRP*.

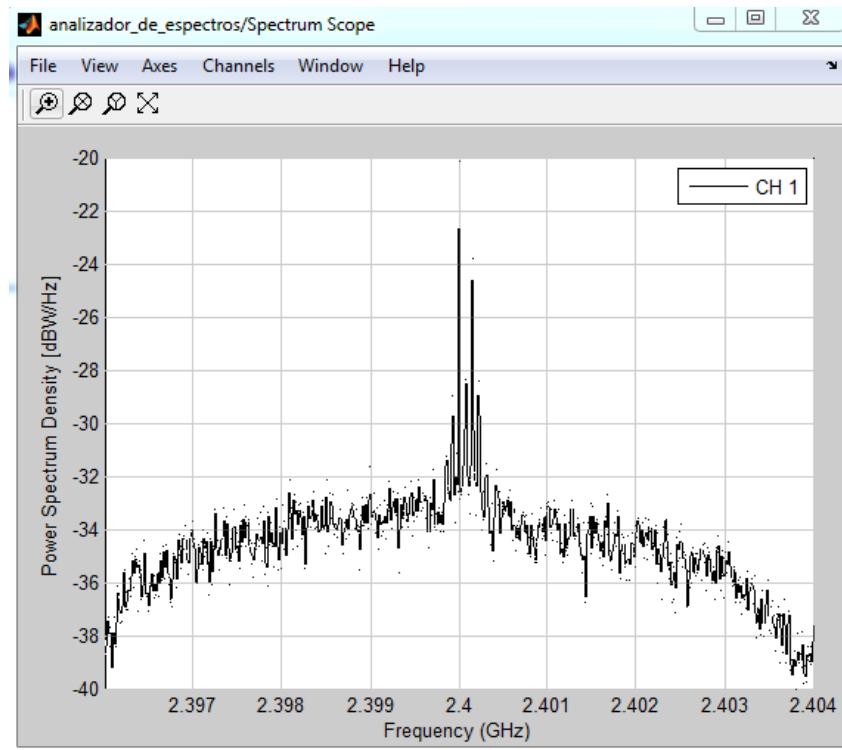


Figura 5-11 Configuración usrp\_source

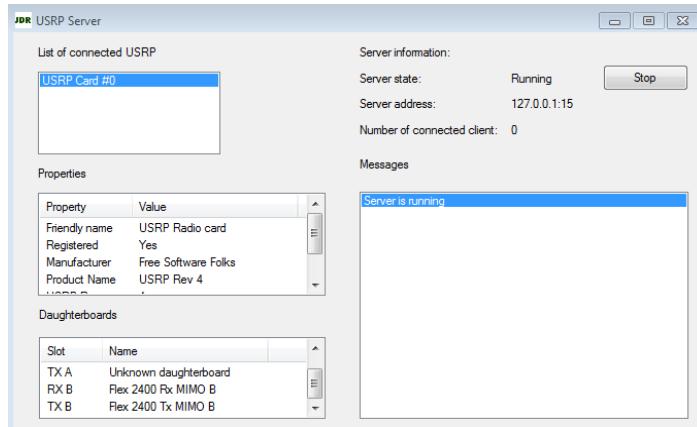


Figura 5-12 Selección USRP con SDR4All server

En la primera sección de este apartado se introducirá a la transmisión de señales utilizando este *toolbox* y en una segunda se acometerá la transmisión de una secuencia de bits utilizando modulación D-QPSK calculando la BER.

Los scripts generados para la realización de diseños en *Matlab* con el *toolbox SDR4all* aparecen en el Anexo IV. No obstante, se irán analizando en los siguientes apartados.

### 5.2.1 Introducción a la transmisión de señales utilizando SDR4all

En esta primera parte se analizará como transmitir señales entre los diferentes equipos utilizando *Matlab* en una serie de pasos:

1. **Selección de USRP.** En primer lugar, se arrancará la aplicación *SDR4All server* (*Inicio/Programas/SDR4All/SDR4All server*) en ambos ordenadores para permitir la comunicación entre *Matlab* y *USRP*, seleccionando el *USRP* detectado, ver Figura 5-12.

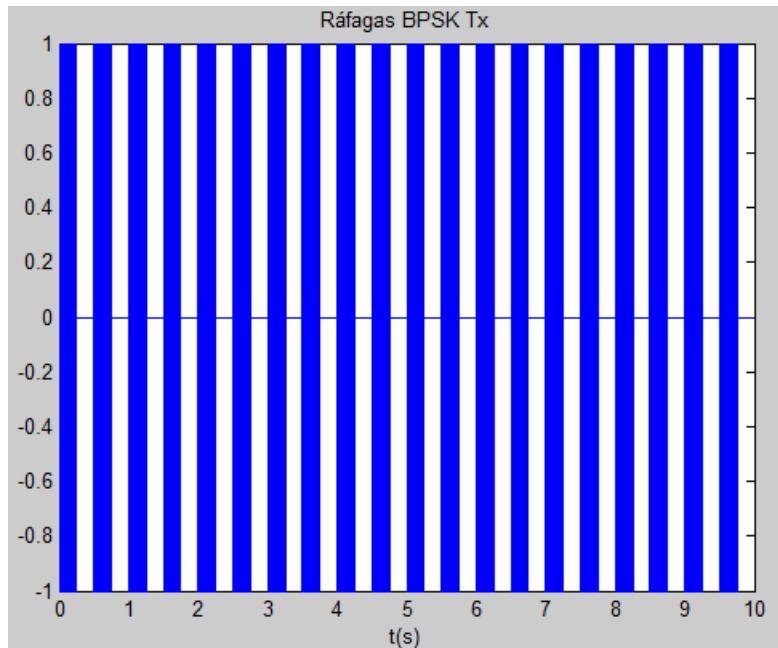


Figura 5-13 Señal transmitida banda base

2. **Configuración de la parte transmisora.** Para ello lo primero es crear un puerto en *Matlab* a partir de la función *SDR4All\_Connect* del *toolbox*, especificando el *USRP* (#0), el lugar de la tarjeta secundaria (*Slot B*) y la función (transmisora en este caso) :

```
sock=SDR4All_Connect(0, 'SlotB', 'TX');
```

Después se seleccionan los parámetros configurables, es decir, la ganancia (el máximo, 20 dB del PGA), frecuencia de transmisión (2,422 GHz) e interpolado (256 para ancho de banda de 500 kHz) a partir de las funciones mostradas a continuación:

```
SDR4All_SetGain(sock, 20);
SDR4All_SetFreq(sock, 2422e6);
SDR4All_SetInterpRate(sock, 256);
```

Tras esto, se define la función en *Matlab* a enviar (en banda base), en este caso se transmitirán una serie de ráfagas con modulación BPSK simplemente para observar la recepción de la señal. La definición de esta señal se puede ver en el fichero mostrado en el Apéndice D.1 y aparece en la Figura 5-13.

Finalmente en este paso, para enviar la señal de ráfagas generada al *USRP* se utilizará el siguiente comando:

```
SDR4All_SendData(sock, Sig);
```

Decir que también se podría haber generado la señal en primer lugar y después configurar los parámetros del *USRP*, es indiferente. De esta manera se hará en el próximo ejemplo.

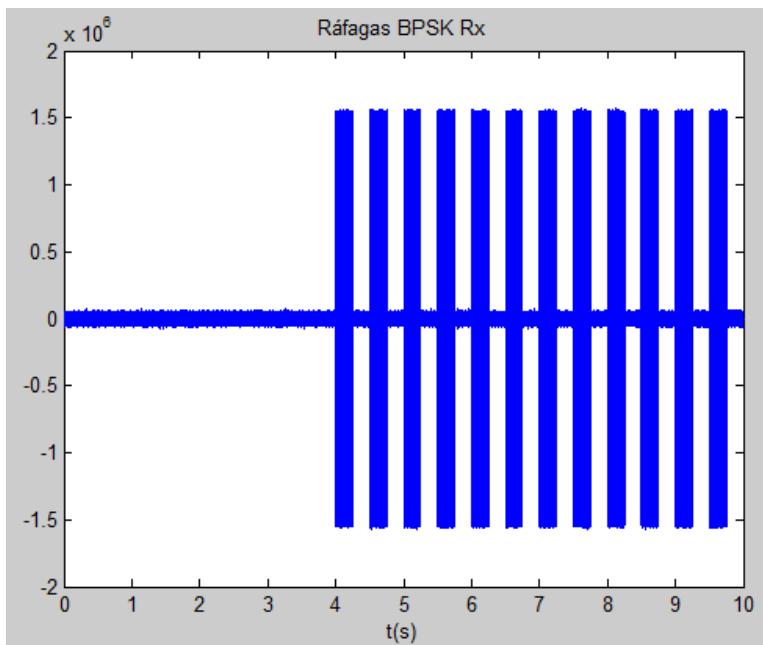


Figura 5-14 Señal recibida banda base

3. **Configuración de la parte receptora.** La configuración de la parte receptora es idéntica a la transmisora, con la salvedad de que la ganancia de recepción se especificará en el punto medio (45 dB), el diezmado será de 128 (ancho de banda 500 kHz) y el lugar de la tarjeta receptora ahora es el A, tal y como puede

verse en el Anexo IV, Apartado 5.2.2D.1. Para recibir la señal durante 5 segundos se ejecuta la siguiente instrucción:

```
[Data] = SDR4All_GetData(sock,5*500e3);
```

La señal recibida está afectada por las adversidades del canal como el multirayecto, el error en frecuencia, interferencias propias de la banda utilizada o ruido. Para obtener la mejor señal posible, se han colocado los USRP a poca distancia, obteniendo la señal en la Figura 5-14.

En el siguiente apartado se intentarán abordar los problemas que supone la transmisión en un canal radioeléctrico. Esto es, se tiene en cuenta la transmisión y recepción de forma conjunta.

Como nota de uso de este *toolbox*, se recomienda que cada vez que se desee realizar una nueva transmisión se reinicie el programa SDR4All Server.

### 5.2.2 Diseño de un sistema de transmisión DQPSK utilizando SDR4all

En este apartado se explicará cómo transmitir una secuencia de bits utilizando modulación DQPSK entre dos USRP mediante enlace radio. El proceso que se ha seguido es el siguiente:

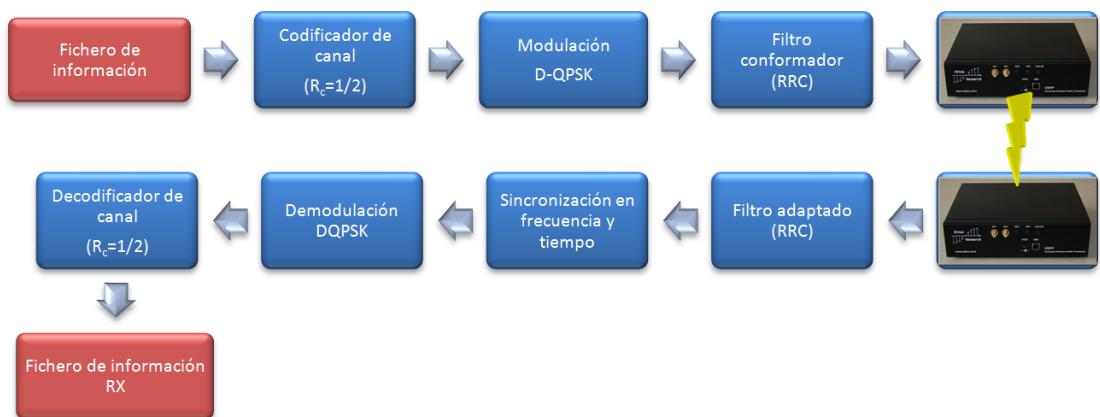


Figura 5-15 Proceso de diseño sistema QPSK

El diseño de este sistema se analizará en una serie de pasos, no obstante, para más detalle véase el Apéndice D.2 donde aparece el código implementado. Se empezará explicando la **parte transmisora**, donde en primer lugar se ha generado la señal y después se han configurado los parámetros de la aplicación USRP Server (al contrario que en el ejemplo anterior).

**PASO1) Información a transmitir:** Los bits de información a transmitir se cargan de una variable *mat* previamente generada (a partir de la función *randn*, asignando 0 ó 1 en función de que el valor sobrepase o no 0.5) tal y como se puede apreciar en el código:

```
load datos_b;
```

Los datos se tienen guardados en una variable para poder comparar posteriormente los datos decodificados en receptor y calcular la BER.

**PASO 2) Codificación de canal:** Para proteger la información frente a errores, se añade un código convolucional a partir de las funciones *poly2trellis* y *convenc* de tasa 1/2, es decir, por cada bit de información se generan dos bits a la salida del codificador convolucional.

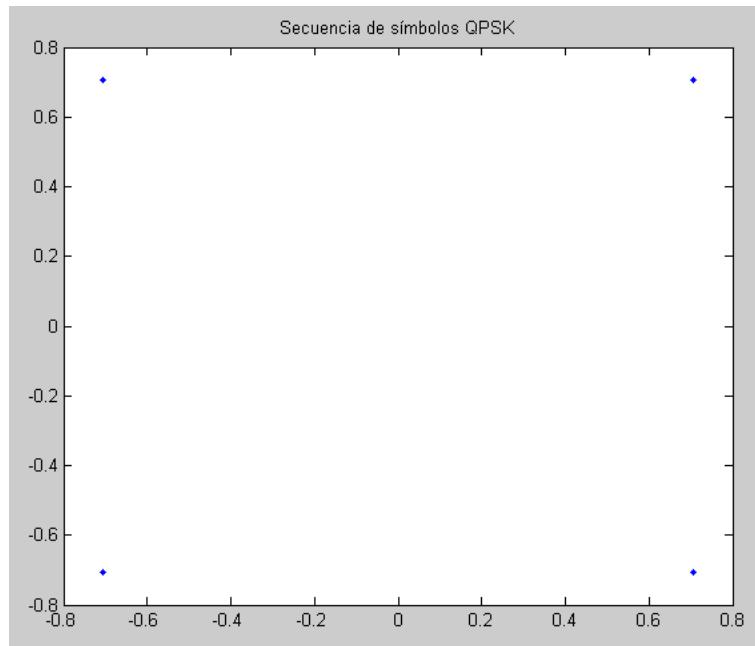


Figura 5-16 Símbolos DQPSK generados

Bits de información	Desfase con respecto al símbolo anterior
<b>00</b>	0
<b>01</b>	$\pi/2$
<b>11</b>	$\pi$
<b>10</b>	$-\pi/2$

Tabla 5-1 Asignación de bits a desfase del símbolo

**PASO 3) Modulación D-QPSK:** A partir de los datos codificados se realiza un mapeo complejo DQPSK agrupando los bits en bloques de longitud 2 y dando un valor complejo correspondiente al estado anterior más un desfase, teniendo en cuenta además que se realiza un mapeado Gray. La tabla de asignación de bits a diferencias de fase sería la Tabla 5-1. El resultado del mapeo DQPSK se puede observar en la Figura 5-16.

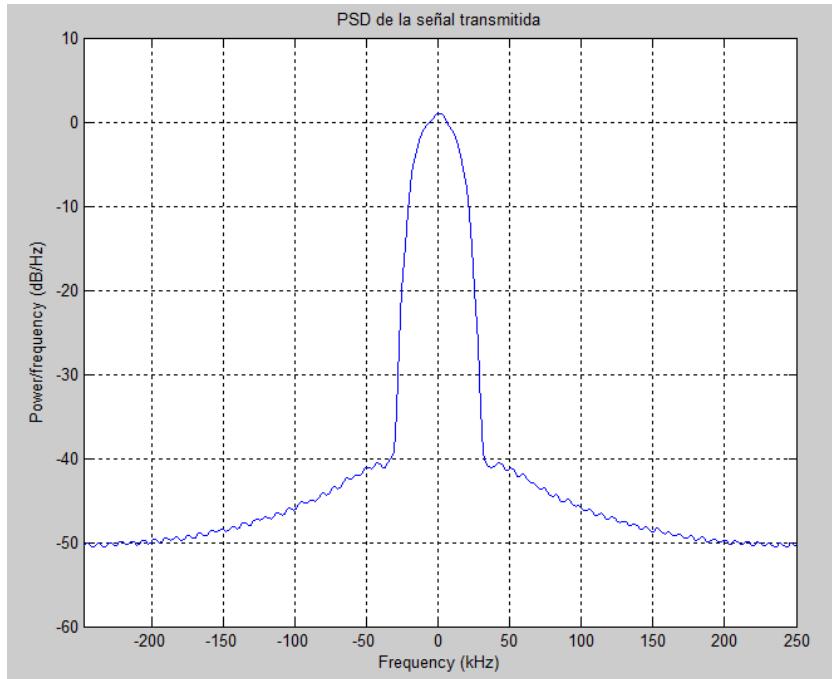


Figura 5-17 Densidad espectral de la señal transmitida

**PASO 4) Filtro conformador:** Tras realizar el mapeo DQPSK, se está en disposición de aplicar el filtro RRC. Como la tasa de muestras se elegirá 500 kS/s y el número de muestras por símbolo 16, el tiempo de símbolo ( $T_s$ ) será:

$$T_s = \frac{\text{sample / symbol}}{f_m} = \frac{16}{500 \cdot 10^3} = 32 \mu\text{s} \Rightarrow$$

$$\Rightarrow R_s = \frac{1}{T_s} = 31,25 \text{ kbaudios} \Rightarrow R_b = 2 \cdot R_s = 62,5 \text{ kbps} \quad (5-1)$$

El factor de *roll-off* por su parte se especificará en 0.22, por tanto, el ancho de banda de la señal a transmitir es:

$$BW_{Tx} = (1 + \alpha) \cdot R_s = 38,125 \text{ kHz} \quad (5-2)$$

En la Figura 5-17 se puede observar la densidad espectral de potencia de la señal transmitida.

Por su parte, el diagrama vectorial y la constelación de la señal DQPSK aparece en la Figura 5-18.

**PASO 5) Señal transmitida por USRP:** Para no tener que realizar una sincronización manual a la hora de poner en funcionamiento el USRP transmisor y receptor, la señal transmitida está formada por la repetición de la misma ráfaga. Para definir la señal transmitida se inserta la siguiente sentencia:

```
sig_Tx=kron(ones(1,5),[200*ones(1,0.2*BW) zeros(1,0.1*BW) sig zeros(1,0.1*BW)]);
```

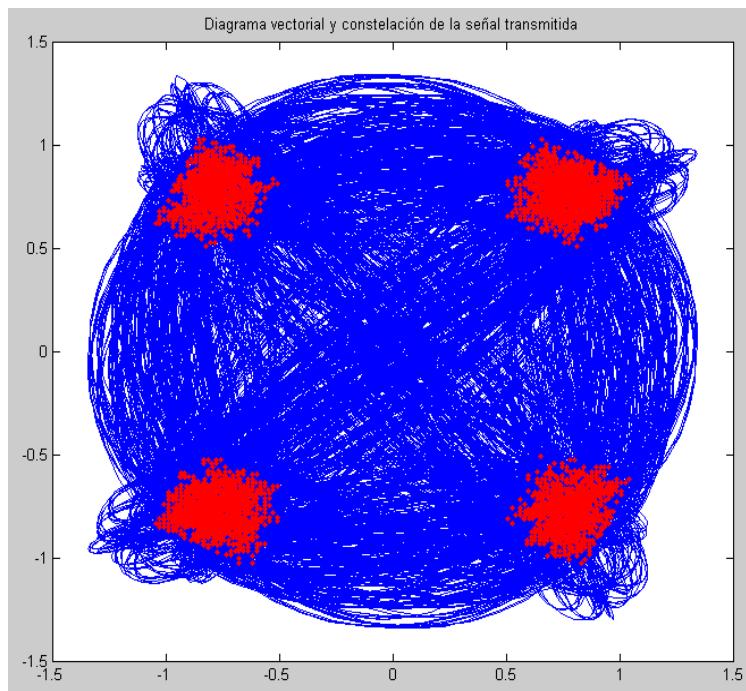


Figura 5-18 Diagrama vectorial y constelación de la señal transmitida

Como se puede apreciar, la señal transmitida a la interfaz aire consta de 5 ráfagas idénticas y cada una de ellas formada por un tramo de continua (0,2 segundos de valor 200), un tramo de ceros para separar la ráfaga de sincronización de frecuencia y la de datos y finalmente la señal de información (variable *sig*). En la Figura 5-19 se ilustra las componentes I/Q de dicha señal.

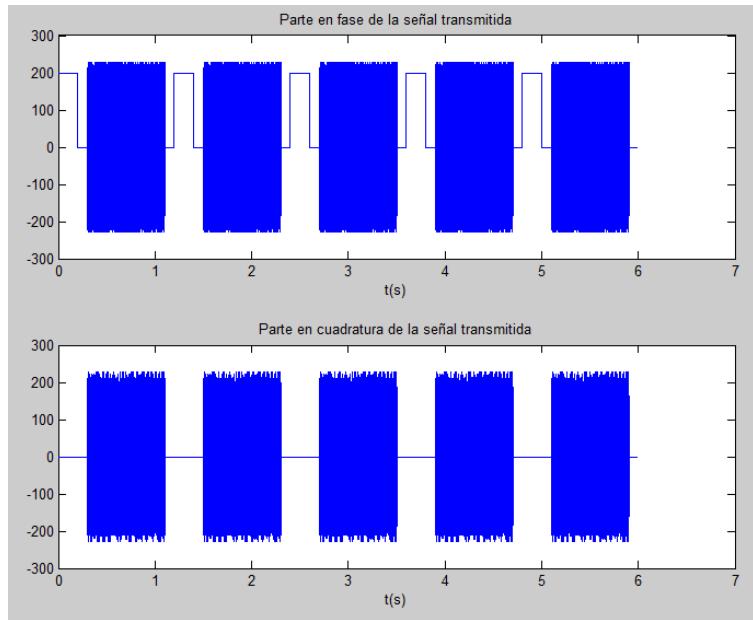


Figura 5-19 Parte en fase y cuadratura de la señal transmitida

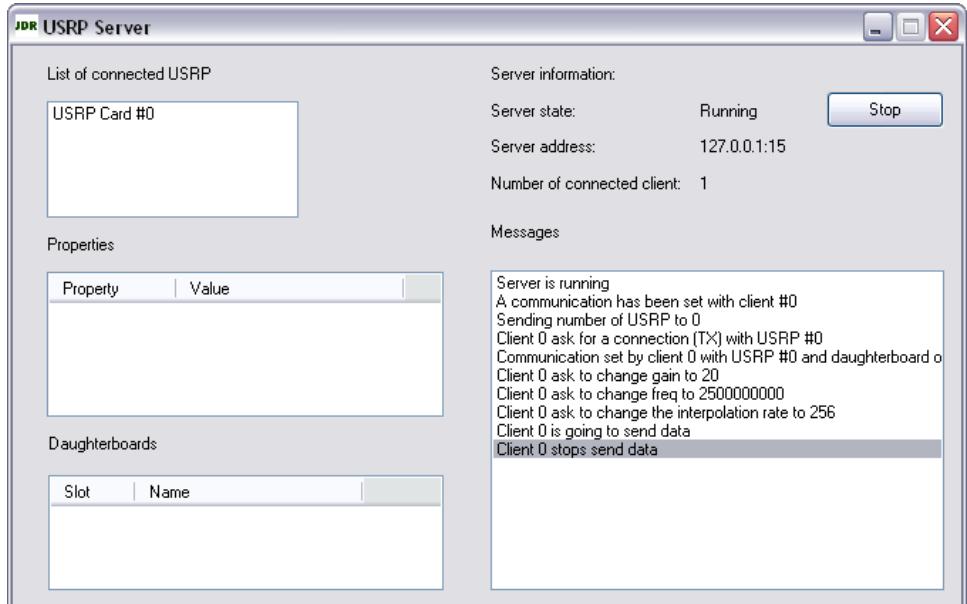


Figura 5-20 Configuración USRP Server transmisor

**PASO 6) Transmisión:** Una vez generada la señal a transmitir, se repiten los pasos en el Apartado 5.2.1. Así, es necesario abrir la aplicación USRP Server, si no se ha hecho antes, contenida dentro del paquete SDR4All (*Inicio/Programas/SDR4All/SDR4All server*) y

seleccionar el USRP que detecte. Al ejecutar el script de *Matlab* incluido en el Apéndice D.2, se van pidiendo por pantalla los parámetros. El script es fácilmente modificable, pudiéndose incluir los parámetros directamente en él. Una vez configurado, aparecerá el mensaje '*Pulsar enter para comenzar la transmisión*', lo cual no debe hacerse hasta que no se configure la parte receptora. La ventana del programa USRP Server debería presentar un aspecto similar a la Figura 5-20.

Como se puede visualizar en la configuración de la figura anterior, derecha abajo, en la prueba se ha seleccionado una ganancia de transmisión de 20 dB, una frecuencia de 2,5 GHz (para disminuir la interferencia) y un interpolado de 256 (por tanto 500 kS/s en la interfaz USB como se ha mencionado). En este punto, el transmisor se quedaría en *Stand-by* a la espera de que se pulse la tecla *Enter* y comenzar la transmisión. Esto se realizará cuando se tenga configurado el receptor.

Tras analizar el transmisor DQPSK implementado, se pasará a la **parte receptora**, lo cual se detalla en una serie de pasos nuevamente,

**PASO 1) Recepción:** Al igual que se realizó en la parte transmisora, se debe configurar los parámetros del equipo receptor. Para ello, lo primero es abrir el programa USRP Server en el ordenador donde esté conectado el USRP que recibirá la señal y a continuación ejecutar el script de la parte receptora, tras lo cual se irán configurando los parámetros desde la pantalla principal de Matlab. Una vez aparece el mensaje '*Pulsar enter para comenzar la recepción*', la ventana del programa USRP Server de la parte receptora debe ser similar a la siguiente figura:

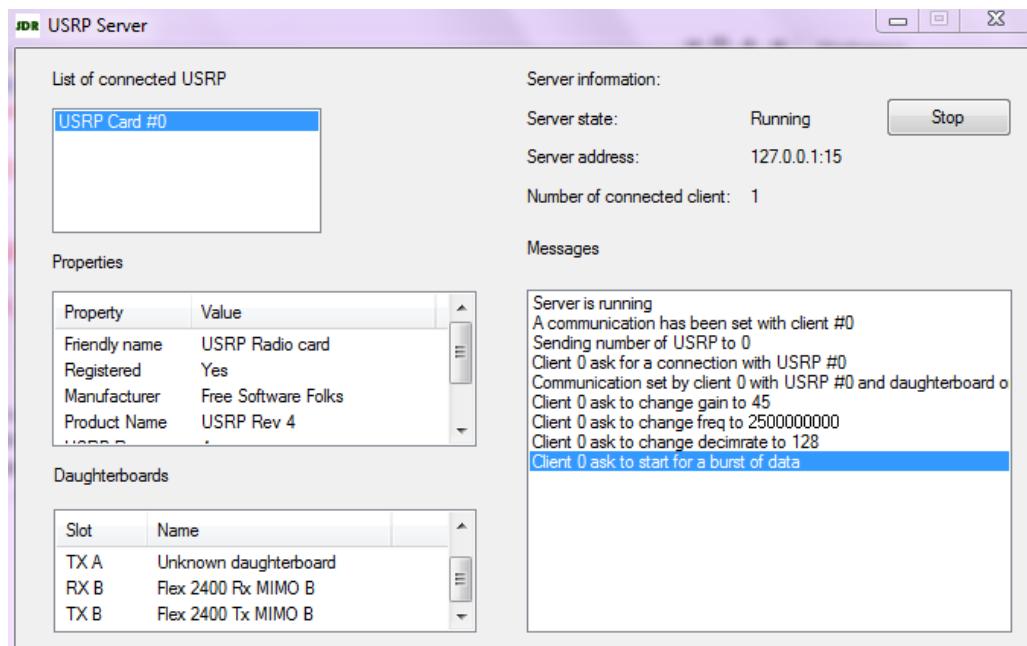


Figura 5-21 Configuración USRP receptor

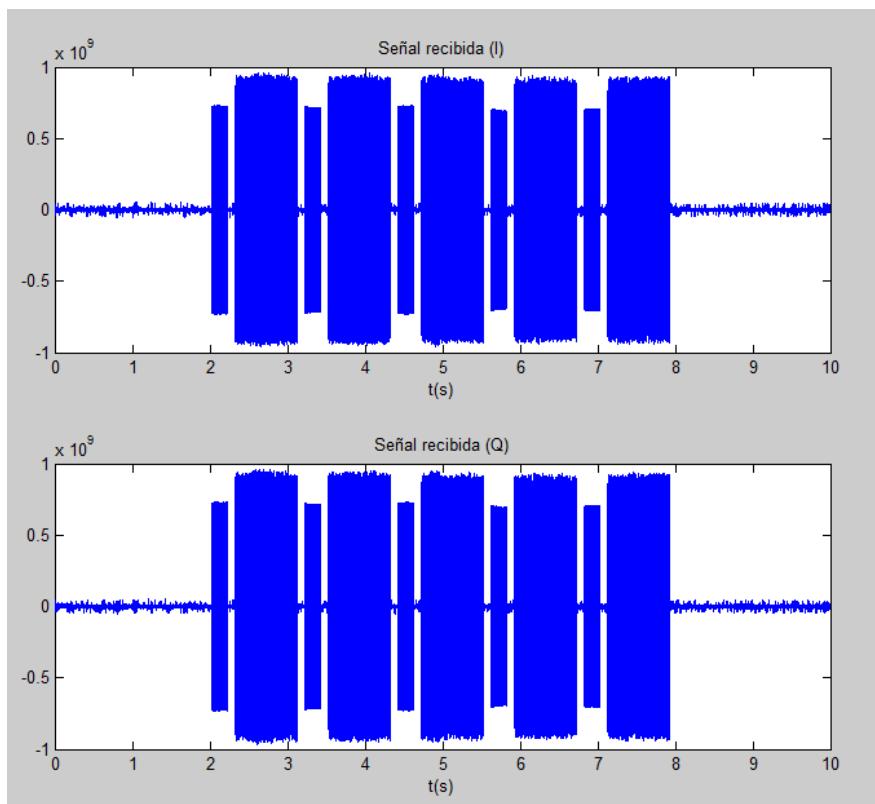


Figura 5-22 Señal recibida, componente I/Q

Tras esto se pulsará la tecla *Enter* en la parte transmisora y receptora simultáneamente, aunque no se precisa una sincronización total ya que la señal transmitida definida consta de varias ráfagas idénticas, por lo que con recibir una de las cinco ráfagas por completo es suficiente.

**PASO 2) Señal recibida del USRP:** La señal captada por el USRP tiene una duración de 10 segundos. Se ha configurado este tiempo de recepción para que sea mayor que el tiempo de la señal transmitida y así poder obtenerla completamente de forma sencilla. El resultado se muestra en la Figura 5-22.

Se puede apreciar como la señal recibida consta de las 5 ráfagas que se transmitieron. Además, la ráfaga de sincronización recibida no tendrá un valor de continua, sino que constará de una exponencial compleja cuya fase será  $(2\pi\Delta f \cdot t)$ , donde  $\Delta f$  es la diferencia entre la frecuencia de transmisión y la de recepción, es decir, la desviación de frecuencia que posteriormente se compensará, debido a los errores y la deriva en osciladores, y no a los 2,5 GHz exactamente tal y como se había especificado. En la siguiente figura se modela la situación comentada de la frecuencia de la ráfaga de sincronización:

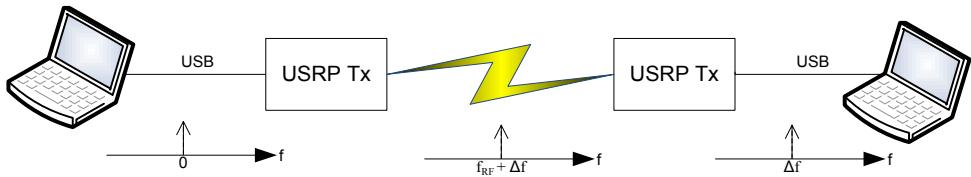


Figura 5-23 Diagrama de la ráfaga de sincronización

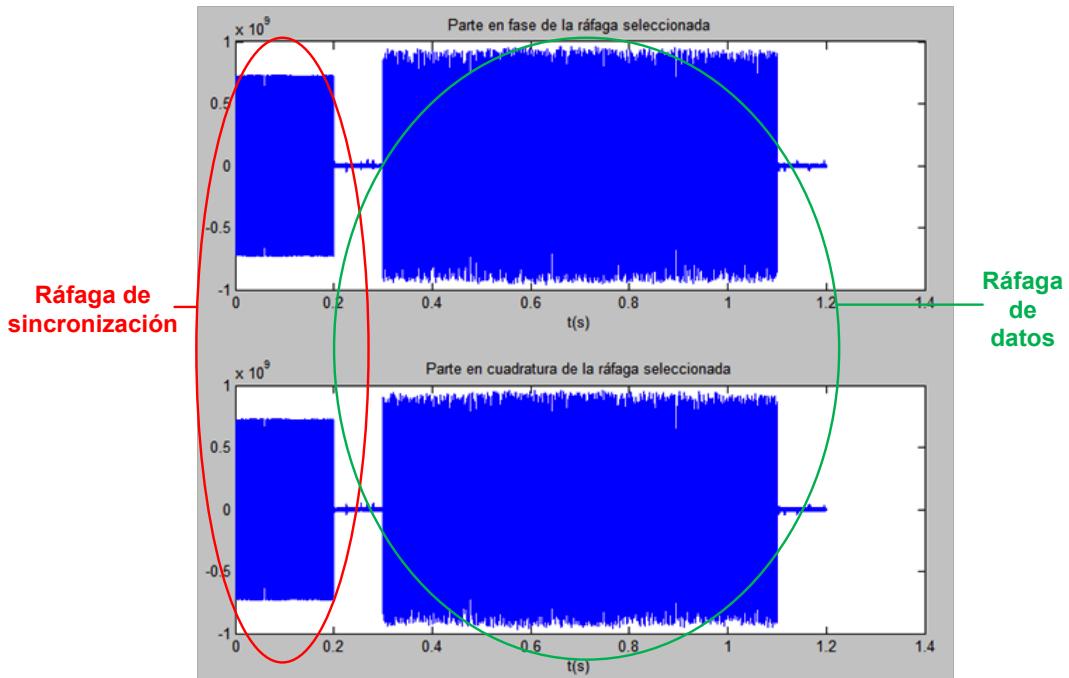


Figura 5-24 Ráfaga seleccionada

**PASO 3) Selección de ráfaga:** Tras obtener la señal enviada, en este sencillo ejemplo, se selecciona una de las ráfagas a partir de la cuál demodular. Para obtenerla, el método seguido se basa en buscar la primera muestra donde se sobrepasa el valor RMS (*root mean square*) de la señal total, obteniendo el resultado en la Figura 5-24.

La ráfaga de datos propiamente dicha no comienza hasta los 0,3 segundos de la ráfaga total, tal y como se especificó en la señal transmitida. A partir de este dato y sabiendo que la ráfaga de datos transmitida dura 0,8 segundos, se selecciona la ráfaga de datos con la que posteriormente se demodulará (se selecciona un intervalo algo mayor, 0,9 s). La Figura 5-25 muestra el resultado.

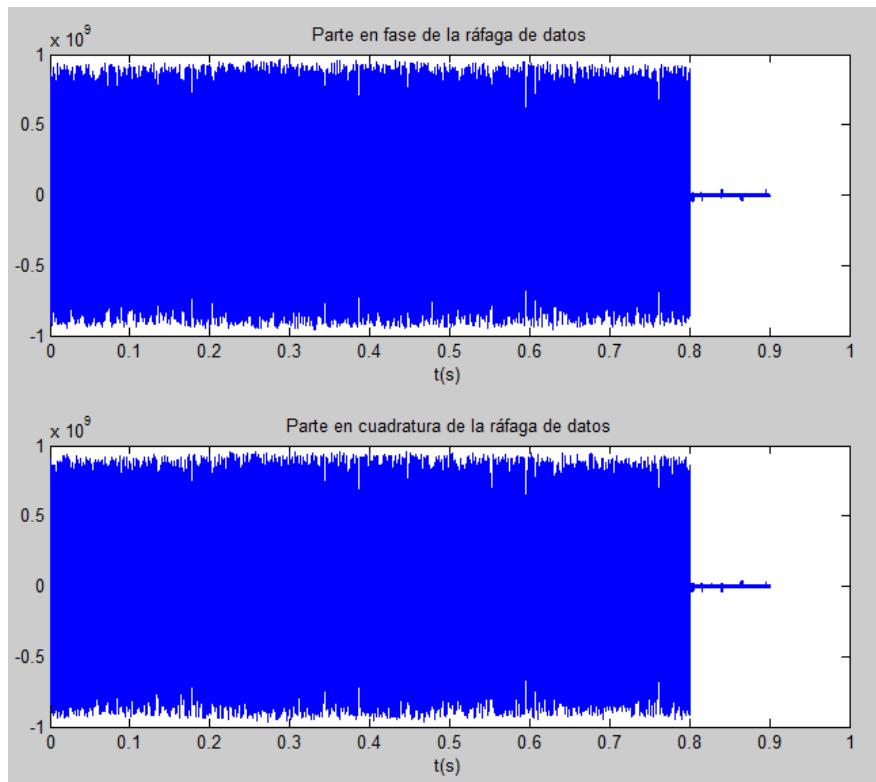


Figura 5-25 Ráfaga de datos

En principio, el espectro de la señal recibida debe estar desplazado respecto a banda base justamente la desviación de frecuencia que se ha comentado. A partir de este desplazamiento se podría obtener dicha desviación de frecuencia, pero la ráfaga de sincronización inicial permite obtener un valor más preciso de este offset. Cabe decir que la desviación de frecuencia además varía con el tiempo, por lo que un cierto offset de frecuencia residual no se corregirá, es por ello que se ha realizado un sistema D-QPSK, ya que cuando se demodule de manera diferencial se podrá soportar estas pequeñas desviaciones de frecuencia residuales. El espectro de la ráfaga de datos aparece en la Figura 5-26.

Se puede apreciar como el espectro aparece desplazado a  $\pm\Delta f$  y además una componente espectral está atenuada frente a la otra debido a una pérdida de potencia. En el siguiente paso se intentará solucionar este error.

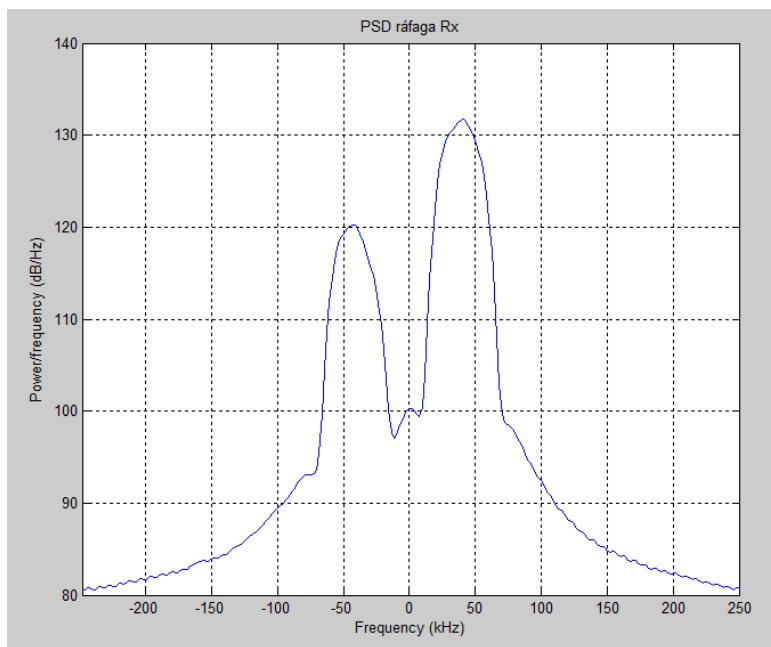


Figura 5-26 Densidad espectral de la ráfaga de datos

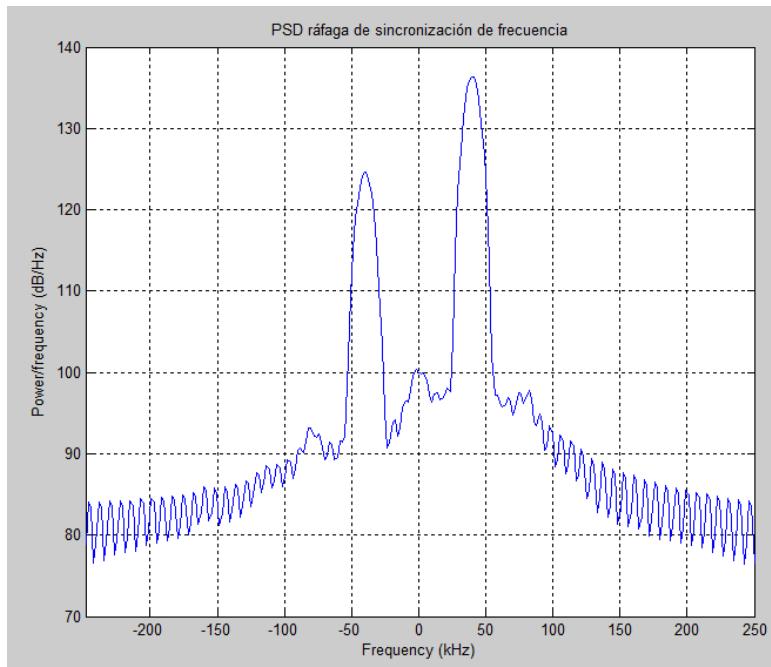


Figura 5-27 Densidad espectral de la ráfaga de sincronización

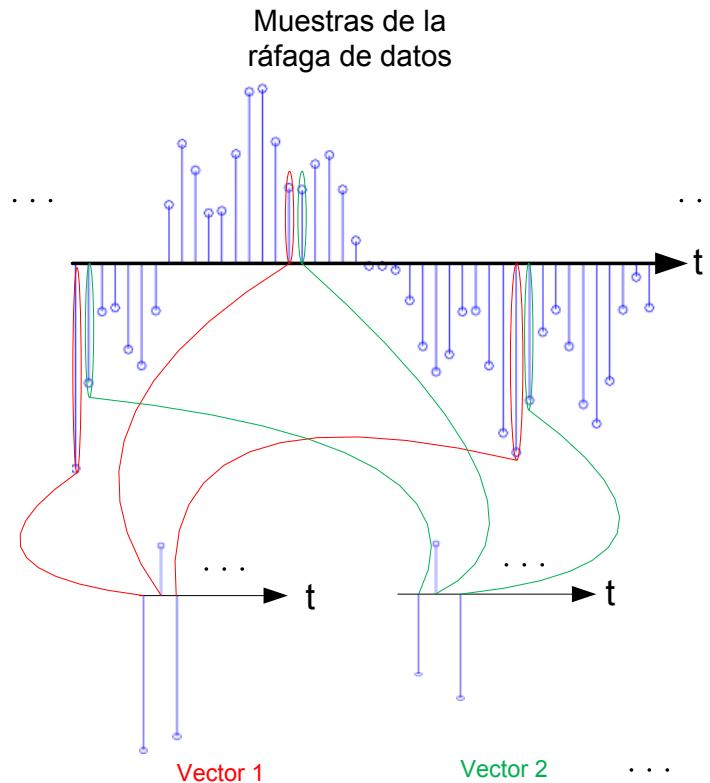


Figura 5-28 Reordenación de la ráfaga de datos para obtener instante de muestreo

**PASO 4) Sincronización:** Sin duda la sincronización es una de las partes más laboriosas a la hora de trabajar con el USRP y Matlab. La arquitectura GNU Radio facilitaba esta labor gracias al bloque MPSK Receiver que realizaba una sincronización en frecuencia, fase y tiempo con el único requisito de especificar un ajuste grueso de la frecuencia. En este diseño, se debe hallar el valor de frecuencia de offset fino además de realizar la sincronización en tiempo, lo cual se ha realizado antes de aplicar el filtro RRC para poder mostrar la constelación en este punto e igualmente se hará tras aplicar el filtrado, ya que la sincronización debe ser más exacta al aumentar la SNR tras ello.

Empezando por la **sincronización en frecuencia**, se ha obtenido el valor de la frecuencia de offset a partir de la ráfaga de sincronización como se ha comentado. La densidad espectral de esta ráfaga se muestra en la Figura 5-27.

En la figura aparecen dos picos espectrales en  $\pm\Delta f$ , por lo que calculando la posición en la que se encuentran estos máximos se obtiene el ajuste fino de frecuencia, resultando un valor de  **$\Delta f=39896 \text{ Hz}$** . Con este valor ya se puede sincronizar en frecuencia la ráfaga de datos,

$$raf_{datos\_f} = raf_{datos} e^{-j \cdot 2\pi \cdot \Delta f} \quad (5-3)$$

Ahora ya se tiene la ráfaga de datos centrada en banda base. Cuando se aplique el filtro RRC adaptado se seleccionará el lóbulo central de esta densidad espectral.

Aunque el valor de desviación variará con el tiempo, la demodulación diferencial paliará el error residual.

En cuanto a **la sincronización en tiempo**, el proceso se ha dividido en dos partes:

1. Obtención del instante óptimo de muestreo: como se tienen 16 muestras por símbolo, en este primer paso se hallará el instante de muestreo óptimo de entre las 16 posibilidades que existen. Para ello, se formarán 16 vectores cada uno de los cuales contiene las muestras en un instante de muestreo diferente. En la Figura 5-28 se aclara este paso.

2.

Una vez se realiza esta reordenación de las muestras, aquel vector cuya varianza de la magnitud sea mínima corresponderá con el instante óptimo de muestreo. La varianza de cada uno de los vectores en este ejemplo se muestra a continuación, donde se puede observar que el instante óptimo es el 6:

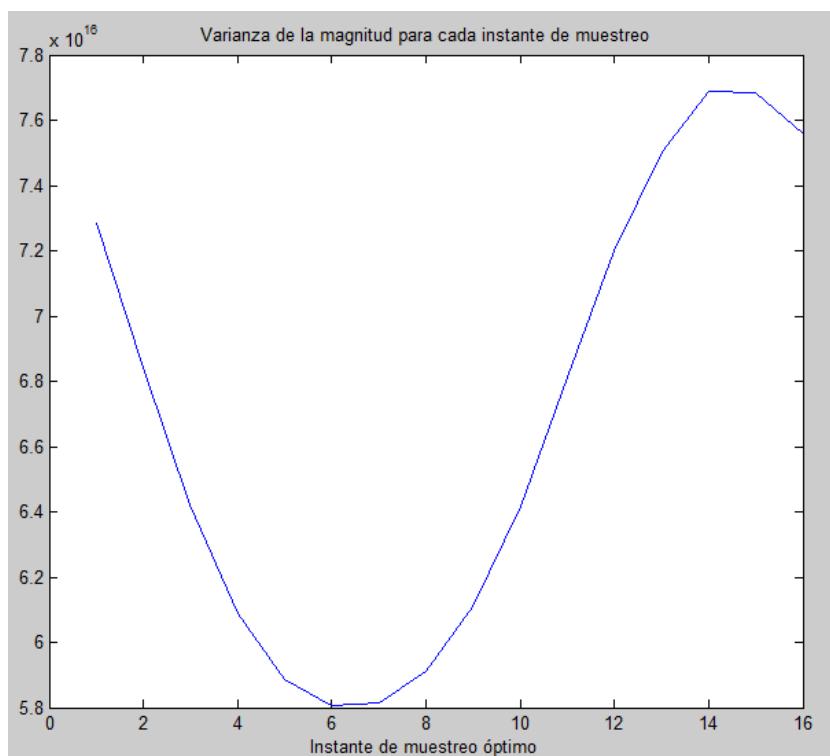


Figura 5-29 Varianza de la magnitud para cada instante de muestreo

3. Para conseguir la sincronización completa en tiempo, se debe conocer el punto inicial de muestreo, es decir, se debe eliminar el transitorio inicial. Se sabe que este

punto inicial de muestreo se encuentra en el vector que contiene las muestras en el instante óptimo de muestreo calculado en el paso anterior, por tanto lo que se hará es una correlación entre la ráfaga de datos y el pulso *Root Raised Cosine*, de tal manera que el máximo de la correlación da una primera aproximación de donde está el punto inicial de muestreo.

Tras realizar el proceso de sincronización, el diagrama vectorial y la constelación de la ráfaga de datos tras ajustar en frecuencia y sincronizar en tiempo se puede observar en la siguiente figura:

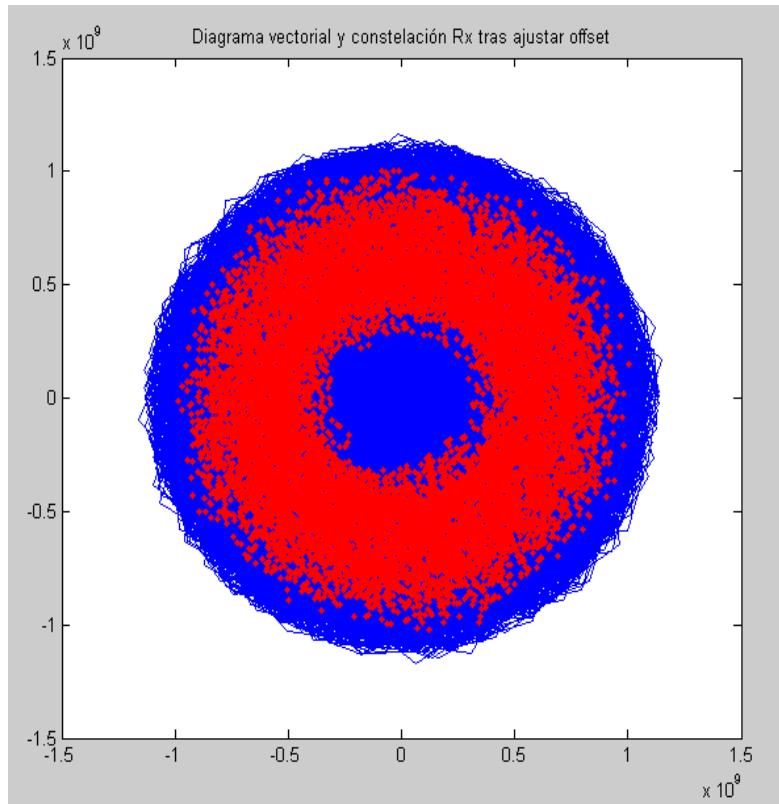


Figura 5-30 Diagrama vectorial (azul) y constelación (rojo) de la ráfaga de datos sincronizada

A pesar de haber realizado la sincronización en frecuencia, se observa como la constelación en lugar de tener 4 puntos define un círculo, debido a la desviación de frecuencia con el tiempo.

**PASO 5) Filtro adaptado RRC:** El siguiente paso en el proceso de recepción será aplicar el filtro adaptado *Root Raised Cosine* con los mismos parámetros que el filtro transmisor. El filtro RRC además se quedará con la banda central de la ráfaga sincronizada mostrada anteriormente. La señal obtenida tras realizar la convolución se muestra a continuación:

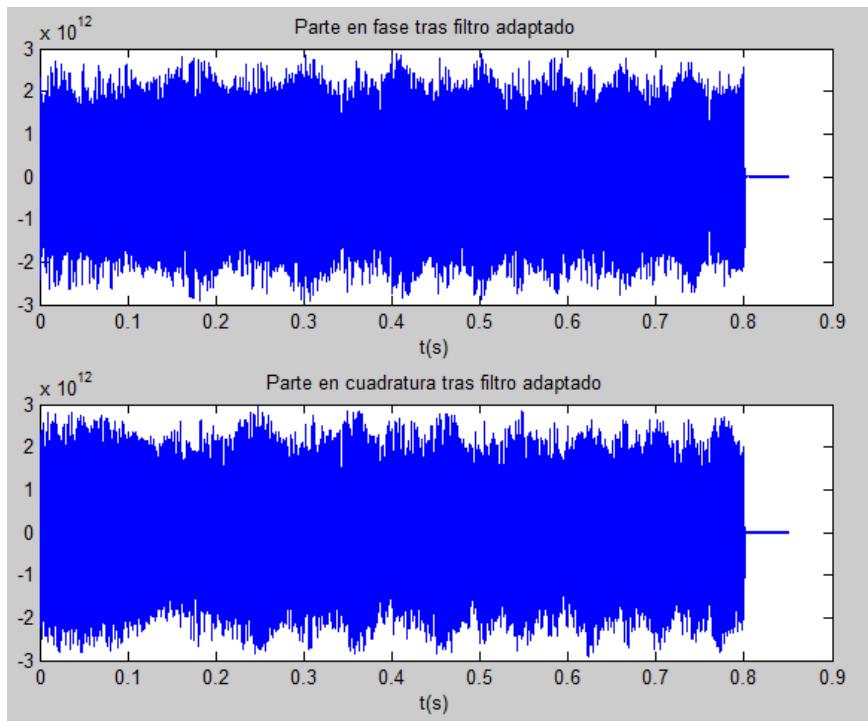


Figura 5-31 Ráfaga de datos sincronizada tras filtro RRC

Para obtener la constelación, se vuelve a realizar el proceso de sincronización de la misma manera que en el apartado anterior, puesto que ahora el resultado debe ser más exacto al haber aumentado la *SNR*. El resultado aparece en la Figura 5-32.

**PASO 6) Demodulador DQPSK:** Llegados a este punto ya se puede demodular la ráfaga de datos (sincronizada y filtrada) a partir de los valores complejos de la misma en los puntos de muestreo, obtenidos a partir del punto de muestreo óptimo inicial (en este ejemplo 166 tras resincronizar) cada samples/symbol (16). El método de demodulación sigue el siguiente proceso:

- 1) Suponiendo el estado inicial con fase  $\pi/4$ , obtener las magnitudes de la diferencia entre el punto complejo actual y el estado anterior multiplicado por cada desfase posible. De esta manera se obtienen 4 valores de magnitud.
- 2) A partir de la magnitud más pequeña se obtiene el desfase que se ha producido en la señal y se seleccionan los bits adecuadamente conforme a la Tabla 5-1.
- 3) Actualizar el valor de la variable que guarda el estado anterior con el valor de la muestra actual y repetir el proceso.

Por ejemplo, si la muestra actual tiene una fase  $3\pi/4$  y el estado anterior es  $\pi/4$ , la magnitud de la diferencia mínima se dará cuando al estado anterior se multiplique por  $e^{j\pi/2}$ , por tanto se seleccionan los bits 01 que corresponden a un desfase  $\pi/2$ .

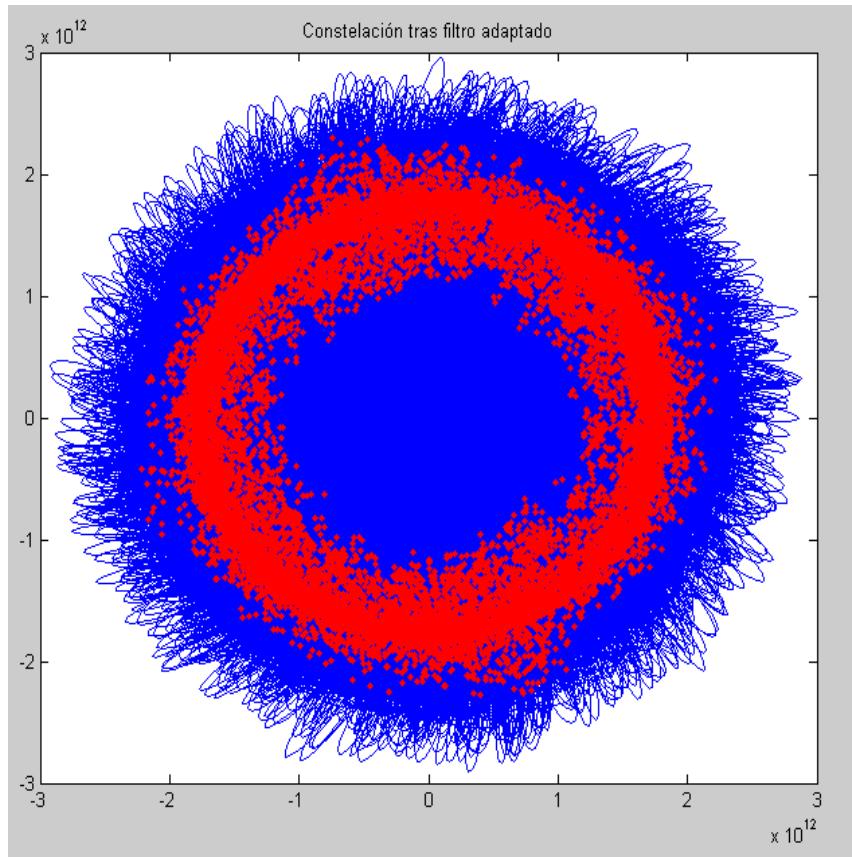


Figura 5-32 Diagrama vectorial y constelación ráfaga de datos sincronizada tras filtro RRC

La densidad espectral de ráfaga tras el filtrado aparece en la Figura 5-33.

**PASO 7) Resultados:** Para verificar que todo el proceso se realizó adecuadamente, lo primero que se debe hacer es cargar las variables que contienen los bits de información originales y los bits codificados de la parte transmisora para realizar la comparación. Los bits que se obtienen tras el demodulador DQPSK están aún codificados y si se comparan con los originales se obtienen los errores en la Figura 5-34.

Como puede observarse, existen muy pocos errores en los bits codificados puesto que la demodulación diferencial soluciona el problema de la variación en el offset de frecuencia y además el canal de propagación no es demasiado adverso. La tasa de error en este punto es de:

$$\text{Code BER} = 7.9 \cdot 10^{-5}$$

Sin embargo, para proporcionar robustez a la señal se introdujo la codificación convolucional en el esquema y por tanto la tasa de errores en los bits de información decodificados debe ser aún menor.

Tras el decodificador de canal no existe ningún error en los bits de información, por tanto:

**BER = 0**

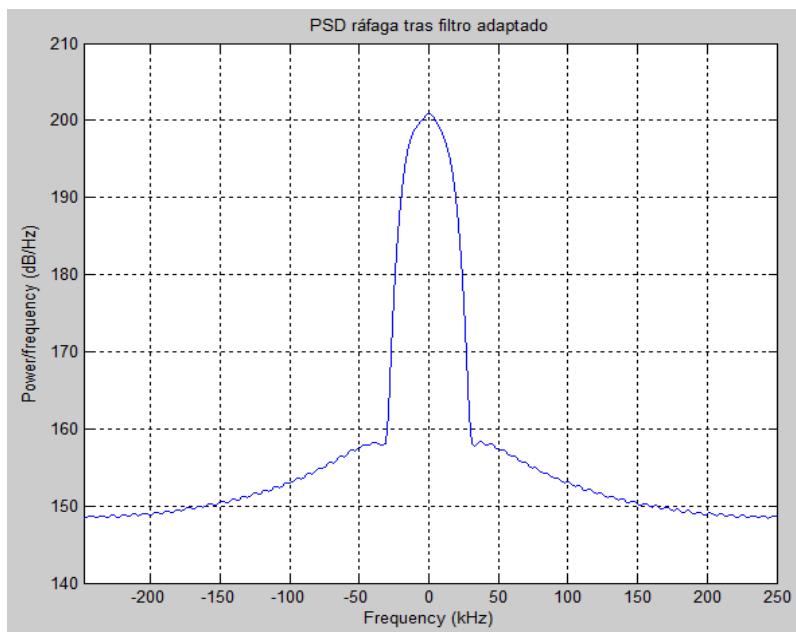


Figura 5-33 Densidad espectral de potencia de la ráfaga tras filtro RRC

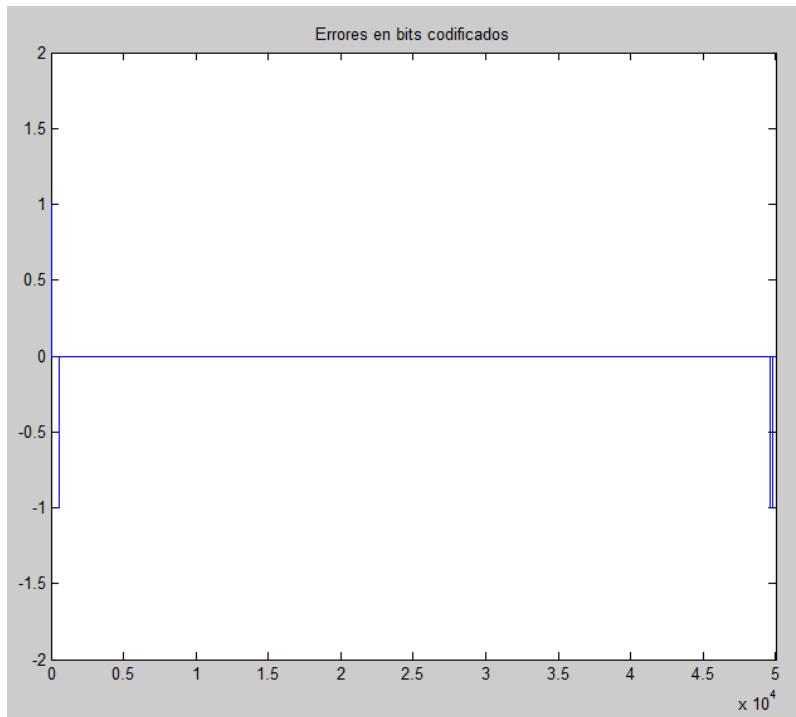


Figura 5-34 Errores en bits codificados

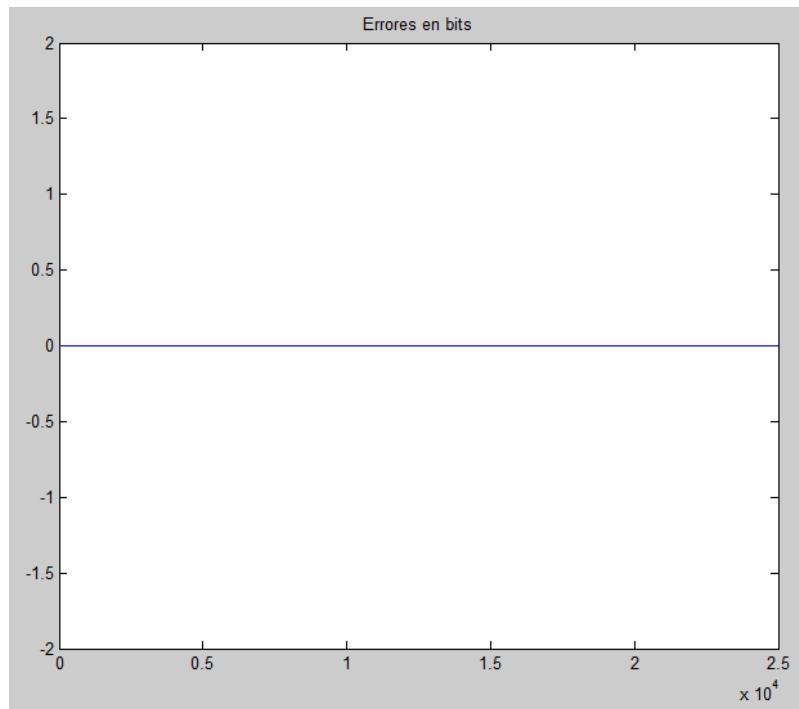


Figura 5-35 Errores en bits decodificados



## Conclusiones

---

En este texto, se plantea el uso del SDR como una plataforma donde realizar experimentos prácticos con gran versatilidad, bajo coste material y reducido tiempo de puesta en marcha. En particular, se propone utilizar el modelo de SDR USRP 1 de Ettus.

Tal como se ha descrito, existen dos partes bien diferenciadas, el SDR propiamente dicho, y un ordenador. La primera realiza el paso de y desde radiofrecuencia a banda base y la segunda prepara la señal en banda base a partir de los datos a transmitir y/o recupera los datos a partir de la señal recibida. Ambos se conectan con un cable USB, para el USRP 1, u otra interfaz para otras SDR. El nexo entre el ordenador y el SDR USRP 1 es el driver que proporciona Ettus. Existen librerías para Python que lo incorporan, para funcionar en Linux y Mac e incluso en Windows. Para evitar programar en Python existe un entorno de ventanas y bloques, el GNU radio companion, que facilita enormemente la tarea. También, y bajo Microsoft Windows, es posible trabajar con *toolboxes* para matlab y simulink, de forma que es inmediato transmitir y recibir con el USRP 1.

El ordenador no tiene por qué ser muy avanzado, un simple ordenador portátil es suficiente. Mediante software se puede diseñar la señal que deseemos para su transmisión, dándole las muestras resultantes al USRP 1, éste las envía y otro las recibe, dándole las muestras obtenidas en banda base a otro ordenador, que las procesa para recuperar los símbolos transmitidos.

El dispositivo USRP 1 está en torno a los 1000-2000 USD, si se tiene en cuenta que hay que comprar la placa hija y la antena. De forma que se puede montar un laboratorio con pocos recursos. Es importante subrayar que, utilizando Matlab, es posible generar una señal, transmitirla y grabarla en recepción para su procesado, también en Matlab. Así, se puede montar un laboratorio docente con un solo par de USRP 1; y los distintos grupos de alumnos sólo los usarían de forma esporádica una vez que tuvieran la señal para transmitir convenientemente preparada, y para grabar la señal en recepción. También, el profesor podría generar una señal transmitida, pasar el script que la generó a los alumnos y pedirles que detecten la señal transmitida a partir de la señal recibida en un USRP 1 receptor, convenientemente almacenada en un fichero.

Respecto a la velocidad de transmisión, la conexión ordenador-USRP 1, mediante USB 2.0, es la que limita la velocidad máxima de transmisión. Otro factor que limita la velocidad es la interpolación de los filtros del USRP 1, que no son muy buenos e introducen distorsión cuando el número de muestras por símbolo no es el suficiente. Estos inconvenientes se pueden solucionar utilizando la versión 2 del USRP.

En definitiva, el USRP 1, o un SDR similar, a tenor de todo lo expuesto en este texto, es una excelente y, en nuestra opinión, indispensable herramienta para enseñar e investigar en ingeniería radio.





## Apéndice A. Instalación manual de GNU Radio

Se recomienda realizar la instalación semiautomática tal y como se indica en el capítulo 2. No obstante en este apartado se mostrará cómo realizar la instalación manual de GNU Radio, que es algo laboriosa. Lo primero que hay que hacer es ubicar los paquetes de GNU Radio los cuales se pueden encontrar en [www.gnu.org/software/gnuradio](http://www.gnu.org/software/gnuradio) (2.x tarball) o utilizando CVS (Concurrent Versions System), lo cual se realizará más adelante. Los paquetes son los siguientes:

- **gnuradio-core**: El núcleo de las librerías.
- **gnuradio-examples**: Ejemplos de GNU Radio.
- **gr-audio-oss**: Soporte para tarjetas de sonido que utilizan (Open Sound System).
- **gr-audio-alsa**: Soporte para tarjetas de sonido que utilizan ALSA (Advanced Linux Sound Architecture).
- **gr-usrp**: Librerías que unen a GNU Radio con el Universal Software Radio Peripheral.
- **gr-wxgui**: wxPython basado en herramientas GNU, incluye un osciloscopio y FFT's.
- **gr-how-to-write-a-block**: Incluye ejemplos de cómo escribir un bloque.
- **usrp**: Soporte para la tarjeta USRP.

Antes de instalar **gnuradio-core** es necesario tener algunos paquetes preinstalados. Estos paquetes son los siguientes:

**FFTW (Fastest Fourier Transform in the West)**: Ésta es una subrutina en C que es capaz de computar transformadas discretas de Fourier en una o más dimensiones, el tamaño de la entrada es arbitrario y maneja datos complejos y reales. FFTW es software libre por lo que se convierte en una librería de FFT para cualquier aplicación.

Se puede obtener de: <http://www.fftw.org/download.html>

**Cppunit**: Es una unidad de prueba para chequear bloques en C++.

Se puede conseguir de: [https://sourceforge.net/project/showfiles.php?group\\_id=11795](https://sourceforge.net/project/showfiles.php?group_id=11795)

**SWIG (Simplified Wrapper and Interface Generator)**: Son herramientas para el desarrollo de software que conectan programas escritos en C o C++ con una gran variedad de lenguajes de programación de alto nivel. SWIG es utilizado con diferentes tipos de lenguaje como Python.

Se puede descargar de: <http://sourceforge.net/projects/swig/>

**Numarray y Numpy**: Son dos módulos de Python necesarios para el funcionamiento de GNU Radio. Éstos son utilizados para computo numérico y están disponibles en: <http://sourceforge.net/projects/numpy>

**wxPython:** Son herramientas para el lenguaje de programación Python, y permite crear programas con funciones e interfaces gráficas de forma simple.

Se puede conseguir de: [https://sourceforge.net/project/showfiles.php?group\\_id=10718](https://sourceforge.net/project/showfiles.php?group_id=10718)

La instalación de los programas antes mencionados es ciertamente complicada ya que éstos también requieren algunos programas preinstalados, lo que vuelve al proceso bastante confuso. Resulta de ayuda el Synaptic Package Manager de Ubuntu, el cual se encuentra en: *System → Administration → Synaptic Package Manager*

Al haber dado de alta, en un principio, los repositorios donde se encuentran las librerías necesarias, ahora simplemente hay que instalarlas; para esto es necesario buscar la librería requerida con el botón de “*search*”, seleccionarla para instalación y aplicar la instalación con el botón “*apply*”.

También se debe instalar las siguientes librerías adicionales:

- **automake1.8**
- **gcc/g++-3.4**
- **g++-3.4**

Después de instalar gcc y g++ se debe crear una liga simbólica con los siguientes comandos ejecutados en una Terminal:

```
# cd /usr/bin  
# sudo ln -s gcc-3.4 gcc  
# sudo ln -s g++-3.4 g++
```

En caso de no ser encontrado wxpython por el Synaptic Package Manager es necesario ejecutar en una Terminal los siguientes comandos:

```
# apt-get update  
# apt-get install wxpython2.5.3
```

Las únicas librerías que no se encuentran en los repositorios son las de SWIG por lo que es necesario instalarlo manualmente, para esto hay que descargarlo de [“http://sourceforge.net/projects/swig/”](http://sourceforge.net/projects/swig/) donde además se puede obtener la última versión.

El archivo descargado se encuentra en formato comprimido **swig-\*.tar.gz** por lo que es necesario descomprimirlo. Para esto es necesario utilizar una Terminal y situarse sobre el directorio donde se encuentra el archivo comprimido. En la Terminal y ya en ese directorio se debe escribir el siguiente comando:

```
# tar zxvf swig-1.3.24.tar.gz
```

Con esto se obtendrá una carpeta con los archivos necesarios de SWIG. En la misma terminal habrá que situarse en esta nueva carpeta y para realizar la instalación es necesario

ahora ejecutar los siguientes comandos esperando a que cada uno termine de ejecutarse antes de correr el siguiente:

```
./configure  
# make  
# make check  
# make install
```

Ahora para obtener los paquetes de GNU Radio se usará CVS, que es una forma de descargar los paquetes en su última versión sin necesidad de hacerlo desde una página de Internet. Para esto se debe ejecutar los siguientes comandos en una Terminal:

```
# export CVS_RSH="ssh"  
# cvs -z3 -d:ext:anoncvs@savannah.gnu.org:/cvsroot/gnuradio co -P gr-build
```

Con estos comandos se creará una carpeta llamada gr-build con los archivos necesarios para instalar a continuación los paquetes de la arquitectura GNU Radio.

Ahora se descargarán los paquetes en esta carpeta que se creó, para esto habrá que situarse sobre esa carpeta y teclear lo siguiente en la línea de comandos de la Terminal:

```
# cvs -z3 -d:ext:anoncvs@savannah.gnu.org:/cvsroot/gnuradio co -P gnuradio-core  
# cvs -z3 -d:ext:anoncvs@savannah.gnu.org:/cvsroot/gnuradio co -P gnuradio-examples  
# cvs -z3 -d:ext:anoncvs@savannah.gnu.org:/cvsroot/gnuradio co -P gr-audio-alsa  
# cvs -z3 -d:ext:anoncvs@savannah.gnu.org:/cvsroot/gnuradio co -P gr-audio-oss  
# cvs -z3 -d:ext:anoncvs@savannah.gnu.org:/cvsroot/gnuradio co -P gr-usrp  
# cvs -z3 -d:ext:anoncvs@savannah.gnu.org:/cvsroot/gnuradio co -P gr-wxgui  
# cvs -z3 -d:ext:anoncvs@savannah.gnu.org:/cvsroot/gnuradio co -P gr-howto-write-a-block  
# cvs -z3 -d:ext:anoncvs@savannah.gnu.org:/cvsroot/gnuradio co -P gr-gsm-fr-vocoder  
# cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/opensdr login  
# cvs -z3 -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/opensdr co -P usrp
```

Para realizar la instalación final se debe ejecutar lo siguiente en la Terminal en el directorio de gr-build:

```
# ./checkout  
# ./for-all-dirs ..//buildit 2>&1 | tee make.log
```

El proceso debe durar algunos minutos y de acuerdo a lo hecho anteriormente no debe haber ningún error, en caso de existir alguno el sistema nos avisara si hace falta algún archivo o librería, de no ser así es necesario revisar el proceso nuevamente.

Ahora se debe obtener las actualizaciones para el USRP, lo único que hay que hacer es descarga este archivo de [www.gnuradio.org/software/gnuradio](http://www.gnuradio.org/software/gnuradio) descomprimirlo como se hizo con SWIG y copiar los archivos que no se encuentren en la carpeta de usrp en gr-build. Ahora para comprobar que todo el proceso se realizó correctamente se ejecutará un programa de ejemplo que simula el tono que da una línea telefónica, para esto, desde una Terminal ubicarse en la carpeta gr-build y teclear lo siguiente:

```
# cd gnuradio-examples/python/audio/  
# ./dial_tone.py
```

Si se escucha un tono como el mencionado anteriormente significa que el ejemplo funciona, y que GNU Radio está instalado correctamente. En caso de no encontrar el archivo `dial_tone.py` se puede crear a través del código que se mostrará en el anexo 7.3.

Como la instalación de GNU Radio se ha realizado directamente desde CVS, hay un archivo (`usrp.fpga.rfb`) que se debe extraer del paquete `usrp-*.*.tar.gz` de la siguiente manera:

```
# mkdir tmp  
# cd tmp (descargar el archivo comprimido a esta carpeta)  
# tar xvfvz usrp-*.*.tar.gz  
# cp usrp-*.*/fpga/rbf/usrp_fpga_rev2.rfb /usr/local/share/usrp/rev2/usrp_fpga.rfb  
# cd ..  
# rm -rf tmp
```

Con esto ya se tienen todos los archivos necesarios en el lugar adecuado y se puede comenzar a ejecutar programas utilizando el USRP y GNU Radio.

## Apéndice B. Análisis detallado de funcionamiento USRP

### Convertidores AD

El USRP tiene 4 convertidores ADC de alta velocidad, cada uno a 12 bits por muestra y 64 millones de muestras por segundo con lo que se puede digitalizar una banda de 32 MHz teóricamente. Si se muestrea una señal de ancho de banda mayor que 32 MHz se introducirá aliasing en el sistema.

El rango completo del ADC es de 2V pico a pico y tiene una entrada diferencial de  $50\Omega$ , lo que significa 13dBm o 20mW de potencia máxima. Antes del ADC se encuentra un amplificador de potencia programable (PGA) para amplificar la señal de entrada y utilizar el rango completo en el caso de que la señal sea débil cuyo rango es de 0 a 20 dB. La tarjeta

secundaria también tendrá una amplificación (programable en general), debido a ello, cuando se especifique la ganancia del USRP, será la suma de ambas contribuciones. Por ejemplo, si se indica una ganancia de 50 dB, el PGA se ajustará a 20 dB y la tarjeta secundaria a 30 dB (al igual ocurre en sentido de transmisión si la ganancia es ajustable). Se verá más claro con la figura resumen al final de esta sección.

## Convertidores DA

El USRP tiene 4 convertidores DAC de alta velocidad para transmisión, cada uno a 14 bits por muestrae y 128 millones de muestras por segundo, por lo que la frecuencia de Nyquist es de 64MHz. Sin embargo, en la práctica la frecuencia de entrada máxima a la que trabaja el DAC es de 44 MHz como se explicará posteriormente.

Los DAC's pueden suministrar 1V pico a pico a una carga diferencial de  $50\Omega$  esto es 10dBm o 10mW. Aparece también un PGA conectado después del DAC para aumentar la ganancia 20 dB (aunque la ganancia puede ser mayor si tenemos en cuenta la amplificación que puede introducir la Daughterboard).

## Procesador e Interfaces

El procesador utilizado por el USRP es una FPGA Altera Cyclone EP1C12. Comprender el funcionamiento de la FPGA es importante para utilizar GNU Radio ya que la tarea de éste es realizar procesos matemáticos de las señales en la banda que se precise y reducir las tasas de muestreo de datos en la interface USB 2.0. Los ADC y DAC's están conectados a la FPGA y éste a su vez se conecta a un chip de interface USB 2.0, el Cipres FX2. En la interfaz USB todas las muestras son de tipo *signed integer* a 32 bits (16 I y 16 Q). La FPGA incluye 4 convertidores digitales de bajada (DDC) como muestra la Figura 5-36 para disminuir la tasa de muestreo (diezmado) permitiendo 1, 2 ó 4 canales independientes de recepción y cada DDC tiene dos entradas, I (en fase) y Q (en cuadratura). Cada ADC puede ser ruteado a cualquiera de las entradas I y Q de los 4 DDC (incluso una salida del ADC puede ser ruteado a más de un DDC).

Para la transmisión se tienen convertidores digitales de subida (DUC) como aparece en la Figura 5-38, que llevan a cabo un proceso de interpolación contenidos fuera de la FPGA en el chip AD9862 CODEC, de esta manera se aprovecha la tasa de muestreo de 128 MHz del DAC en lugar de estar limitado por los 64 MHz de la frecuencia de reloj de la FPGA. En teoría, se podría transmitir señales de hasta 64 MHz de frecuencia máxima, aunque en la práctica el límite está en 44 MHz debido al filtro paso banda del AD9862. La tarjeta secundaria posteriormente elevará la señal generada a la frecuencia deseada.

Los canales múltiples para la recepción deben todos tener la misma tasa de transmisión de datos, al igual que los canales de transmisión que deben tener el mismo régimen binario el cual debe ser distinto del de recepción. Un detalle a tener en cuenta es que, internamente, la FPGA diezma por un factor de al menos 4, que será el mínimo valor de diezmado. Este factor lo limita la tasa de datos por el USB (32 MB/s), ya que si se usan 8 bits para la componente I y otro 8 bits para la componente Q, un factor de diezmado por 4 produce una tasa de 16 MS/s (64 MS/s, la máxima tasa de muestreo de entrada dividida por 4) por lo que la tasa de datos por la interfaz USB será de 32 MB/s (16 MS/s \* 2 Bytes). En el caso de utilizar 16 bits por

componente que será lo habitual para aprovechar los 14/12 bits a los que trabaja el DAC/ADC, el factor de diezmado mínimo será 8 siguiendo el mismo razonamiento, esto es, como la velocidad máxima en el USB es de 32 MB/s, entonces el diezmado tiene que duplicarse si pasamos de 8 a 16 bits por componente. El factor de diezmado máximo admisible es 512 y por tanto el rango de valores posibles está entre 8 y 512 y sólo valores pares para que la tasa de muestras por la interfaz USB sea un número entero, además se recomienda que el diezmado sea una potencia de 2 por eficiencia de algunos algoritmos como la FFT. Igualmente, el factor de interpolado se debe encontrar entre 8 y 512.

En la Figura 5-36 se muestra un diagrama a bloques de la trayectoria de recepción desde la antena hasta el DDC.

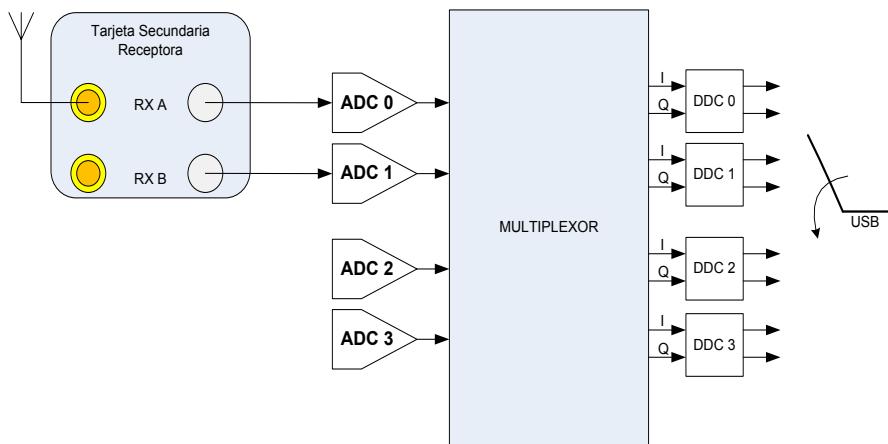


Figura 5-36 Multiplexor en el USRP, donde I es la señal en fase y Q la señal en cuadratura

El multiplexor de la Figura 5-36 es un circuito seleccionador, que determina qué ADC se conecta a cada DDC. Se puede controlar este MUX utilizando el método en Python llamado `usrp.set_mux()`, funcionalidad que se pierde al utilizar *GNU Radio Companion* o el entorno *Matlab*.

Como se ha comentado en el apartado 2.1.2, si se supone el diseño de un receptor de FM en donde el ancho de banda de una emisión es de 200 kHz, se puede seleccionar un factor de diezmado de 200 con lo que la tasa de recepción (y ancho de banda) a través del USB será de  $64\text{MS/s} / 200 = 320 \text{ kS/s}$  (y por tanto 320 kHz) el cual satisface los 200 kHz de ancho de banda sin perder información.

Es posible seleccionar la frecuencia IF del DDC utilizando el método en Python de `usrp.set_rx_freq()`, y para seleccionar el factor de diezmado `usrp.set_decim_rate()`, el cual puede ir de 4 a 512. Hay que tener en cuenta que normalmente se utilizará una frecuencia IF de 0, es decir, banda base. Al seleccionar la frecuencia a la que se desea operar con el método `usrp.tune()`, la tarjeta secundaria intentará ajustarse al máximo a esa frecuencia aunque es difícilmente será capaz de ajustarse exactamente, para solucionarlo en el DDC antes del diezmado se centra el espectro entre  $-f_s/2$  y  $f_s/2$ . En la siguiente figura se aprecia la implementación:

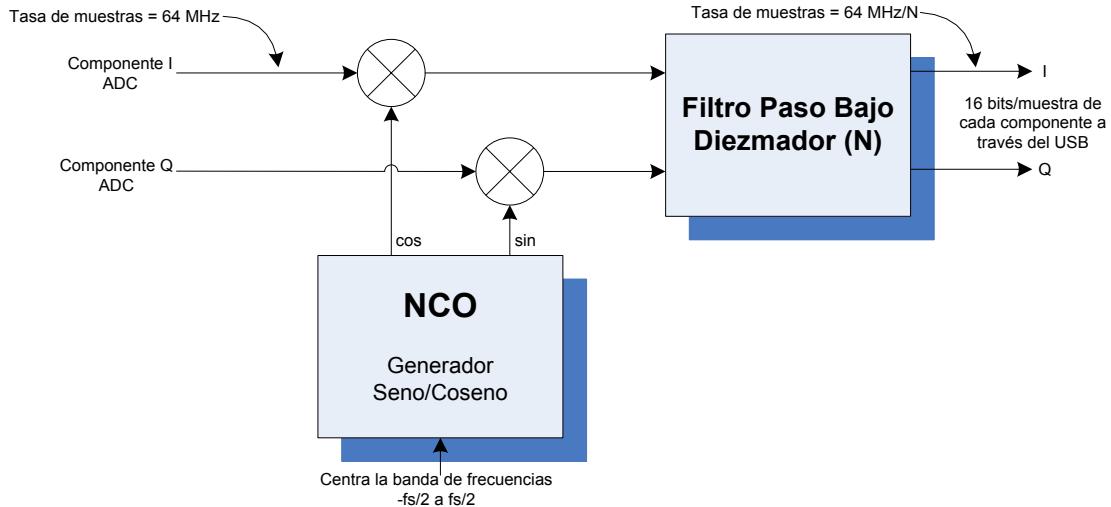


Figura 5-37 Diagrama interno del DDC

El NCO (Numerically Controlled Oscillator) será controlado por los bloques encargados de la sincronización de la arquitectura GNU Radio, cosa que no puede realizarse en *Matlab*, de ahí los problemas de sincronización que se han comentado en el Apartado 5.2.

En cuanto a las muestras por la interfaz USB, cuando se tienen múltiples canales (más de cuatro) los canales son multiplexados. Por ejemplo, con cuatro canales, la secuencia enviada a través del USB será I0 Q0 I1 Q1 I2 Q2 I3 Q3 I0 Q0 I1 Q1, etc.

Finalmente las señales I y Q entran al ordenador vía USB y se procesa la señal banda base en software.

Para la transmisión todo sucede de forma similar, con la diferencia de que el proceso es contrario. Se envía una señal I y Q en banda base del ordenador al USRP. El convertidor digital de subida (DUC) interpola la señal y la envía a través del DAC. El factor de interpolado será 128 MHz dividido por la tasa de muestreo de los datos de entrada al USRP.

Una vez vistos todos los conceptos del USRP, se muestra en la Figura 5-38 un diagrama de bloques general de la tarjeta madre más detallado que el presentado en apartados anteriores.

Por último, se muestra en la Figura 5-39 un resumen del proceso del proceso de recepción desde la antena hasta la interfaz USB en cuanto a muestreos y ganancias se refiere. Hay que tener en cuenta que normalmente la frecuencia intermedia será 0, por ello la tarjeta secundaria intentará sintonizarse al máximo a la radiofrecuencia deseada. El proceso de transmisión es simétrico.

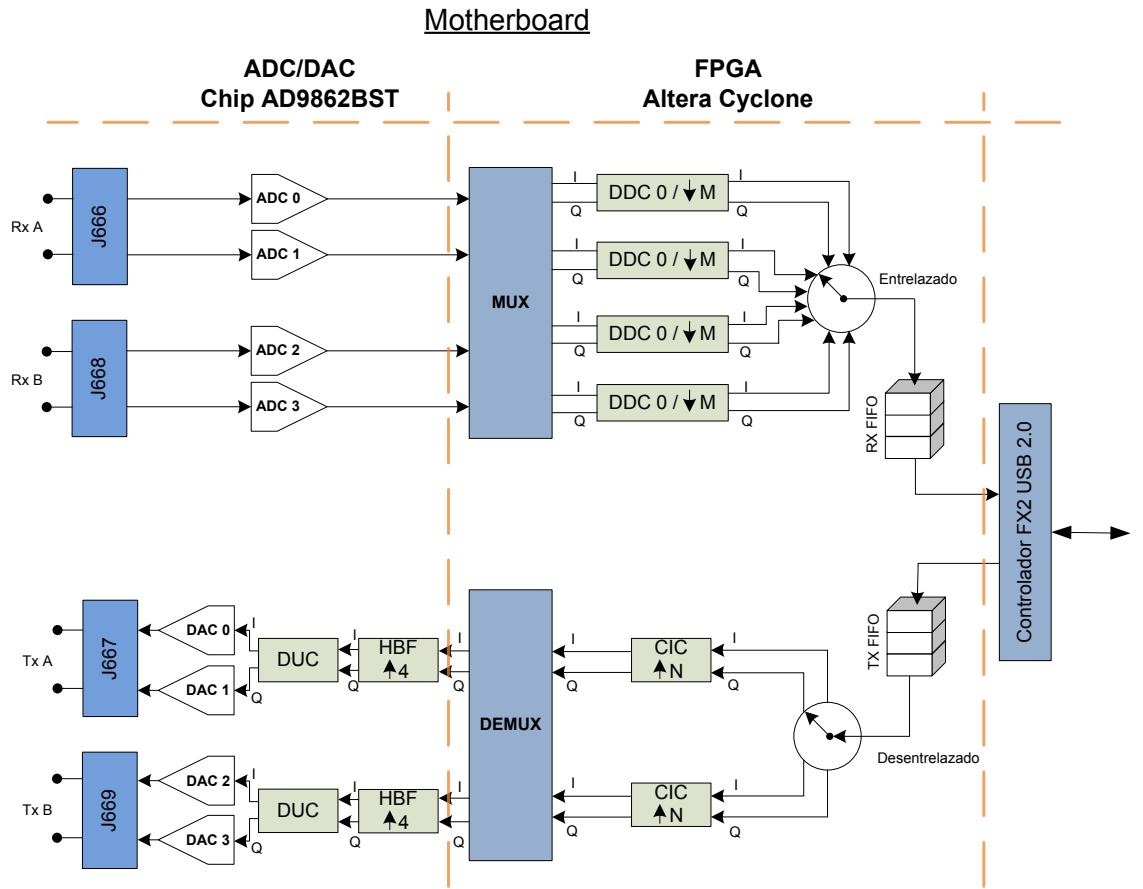


Figura 5-38 Diagrama de bloques detallado del USRP

Como puede verse, la señal de radiofrecuencia con ancho de banda B es captada por la antena y llevada a la tarjeta secundaria, donde se amplifica y se intenta llevar la señal a banda base aunque habrá una pequeña desviación. Después la banda deseada entra en la Motherboard donde se vuelve a amplificar, se digitaliza a 64 MS/s (obteniendo el espectro repetido cada 64 MHz) y gracias al NCO se elimina la desviación de frecuencia obteniendo la señal en banda base. Finalmente se realiza un diezmado por N, por lo que la tasa de muestras que se enviará por la interfaz USB será de 64 MS/s / N.

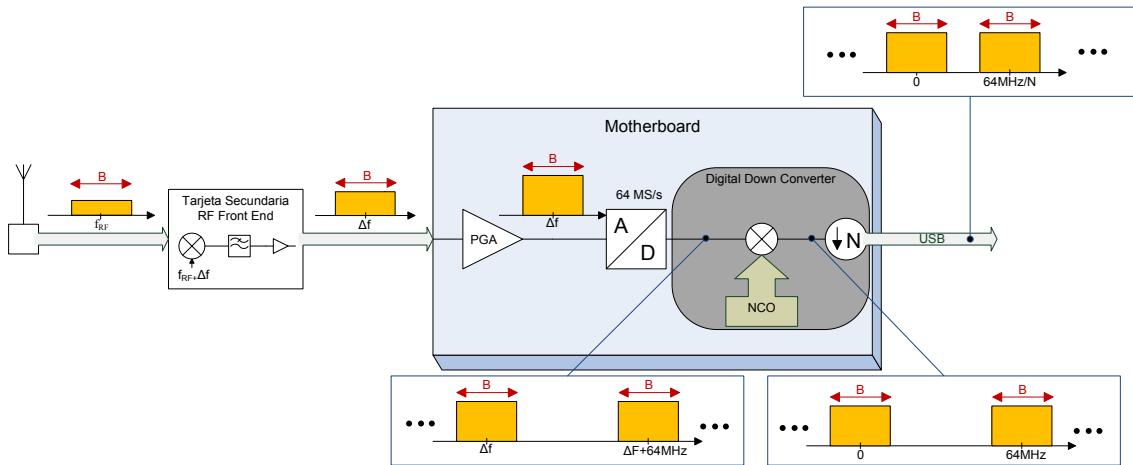


Figura 5-39 Proceso de recepción

## Apéndice C. Programación directa en Python

En este anexo se analizará línea por línea el código del módulo *dial\_tone.py* que se muestra a continuación:

```
from gnuradio import gr
from gnuradio import audio
from gnuradio.eng_option import eng_option
from optparse import OptionParser

class ejemplo_tono(gr.top_block):
    def __init__(self):
        gr.top_block.__init__(self)
        parser = OptionParser(option_class=eng_option)
        parser.add_option("-O", "--audio-output", type="string", default="",
                          help="pcm output device name. E.g., hw:0,0 or /dev/dsp")
        parser.add_option("-r", "--sample-rate", type="eng_float", default=48000,
                          help="set sample rate to RATE (48000)")
        (options, args) = parser.parse_args()

        if len(args) != 0:
            parser.print_help()
            raise SystemExit, 1

        sample_rate = int(options.sample_rate)
        ampl = 0.1
```

```

src0 = gr.sig_source_f (sample_rate, gr.GR_SIN_WAVE, 350, ampl)
src1 = gr.sig_source_f (sample_rate, gr.GR_SIN_WAVE, 440, ampl)
dst = audio.sink (sample_rate, options.audio_output)
self.connect (src0, (dst, 0))
self.connect (src1, (dst, 1))
if __name__ == '__main__':
    try:
        ejemplo_tono().run()
    except KeyboardInterrupt:
        pass

```

Si las primeras líneas, se inician con #, son únicamente comentarios y no afectan a la lógica del programa. Es necesario, para comprender las siguientes líneas, hablar sobre los módulos y paquetes en lenguaje Python.

Un **módulo** es un programa realizado en Python con un sufijo “.py” y contiene un programa con alguna función específica la cual está disponible como una variable global “`__name__`”, esta función puede ser importada dentro de otro modulo en cualquier nivel. Un **paquete** es un conjunto de módulos que realizan funciones similares. Para que Python maneje estas carpetas como paquetes es necesaria la función “`__init__.py`”, un paquete puede contener módulos y sub-paquetes. La estructura dentro del programa para módulos y paquetes es del tipo x.y, donde y es un módulo contenido en el paquete x, por lo que si en el programa tenemos “`gr.sig_source_f`” significa que el módulo `sig_source_f.py` se encuentra en el paquete `gr`. Estos paquetes y módulos se encuentran en: “`/usr/lib/python2.x/dist-packages`”.

Se pasará a comentar el ejemplo. Al inicio del programa se importan los siguientes módulos:

```

from gnuradio import gr
from gnuradio import audio
from gnuradio.eng_option import eng_option
from optparse import OptionParser

```

Con lo que se importa el paquete `gr` y el módulo `audio.py` de la carpeta de `gnuradio`, como se muestra en la Figura 5-40 además de los otros dos módulos.

A continuación se define la clase `ejemplo_tono` la cual hereda los métodos de la clase `gr.top_block` a partir de la siguiente sentencia:

```
class ejemplo_tono(gr.top_block):
```

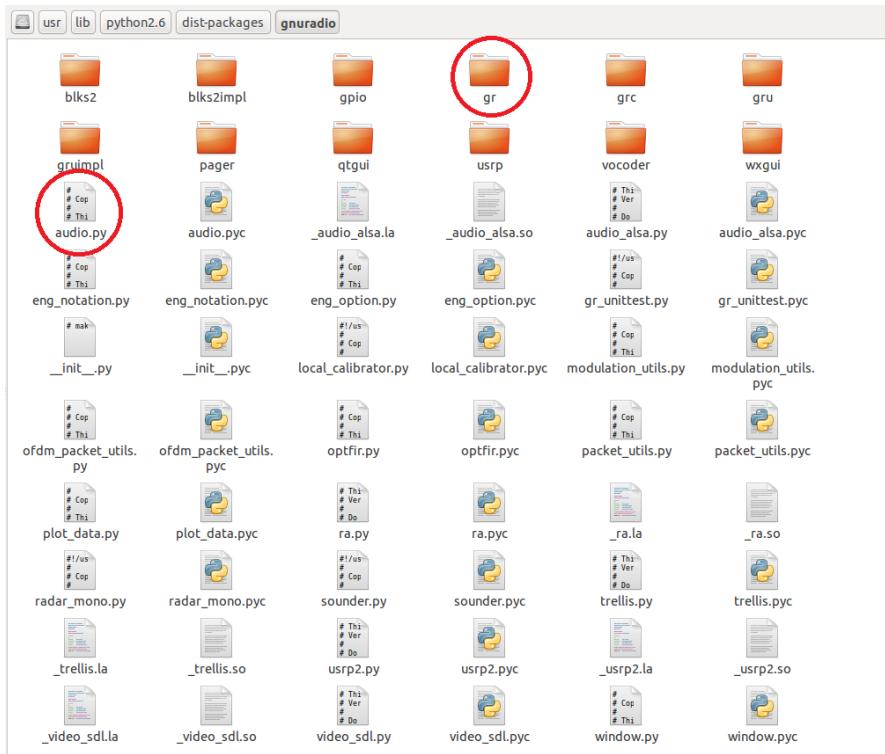


Figura 5-40 Paquete *gr* y módulo *audio.py* en la carpeta *gnuradio*

Ahora se especifica la primera función que se ejecutará al crear un objeto de la clase *ejemplo\_tono* con el argumento *self*, por lo que servirá para cualquier proceso de inicialización. Por su parte, el argumento *self* se utilizará para asignar atributos al objeto creado y poder acceder posteriormente a esos atributos. Si por ejemplo tenemos una función con un argumento *potencia*, para asignar el atributo al objeto escribiríamos *self.potencia=potencia*, por lo que *self* referencia al objeto creado con los atributos correspondientes tal y como se aprecia a continuación:

```
def __init__(self):
    gr.top_block.__init__(self)
```

Con *gr.top\_block.\_\_init\_\_(self)* se llama a la función inicial del módulo *top\_block* del paquete *gr* y se pasa como argumento el objeto creado.

El siguiente bloque de código se refiere a las opciones que se proponen al usuario para seleccionar la salida de audio y la tasa de muestreo:

```
parser = OptionParser(option_class=eng_option)
```

```

parser.add_option("-O", "--audio-output", ...)

parser.add_option("-r", "--sample-rate", ...)
(options, args) = parser.parse_args()
if len(args) != 0:
    parser.print_help()
    raise SystemExit, 1

```

En primer lugar, se crea el objeto *parser* de la clase *OptionParser* al que se aplica el método *add\_option* (definido en la clase *OptionParser* obviamente) para poder seleccionar la salida de audio y la tasa de muestreo (si no se definen tomarán los valores por defecto). Si al ejecutar el fichero se le añade “-O” o bien “-r” se guardará en las variables *options* y *args*, en cuyo caso se imprimirá la ayuda si el número de argumentos no es igual a 0.

Las siguientes instrucciones definen la tasa de muestreo (obtenida de *options*) y la amplitud:

```

sample_rate = int(options.sample_rate)
ampl = 0.1

```

Ahora se crean dos fuentes de señal con el método *sig\_source\_f* al que se pasa como argumentos la tasa de muestreo, la forma de onda, la frecuencia y la amplitud con *sample\_rate*, *gr.GR\_SIN\_WAVE*, 350/440 y *ampl* respectivamente. Además se define el destino a partir de la función *sink* del módulo *audio.py* pasando como argumentos la tasa de muestreo y la salida de audio obtenida de la variable *options*:

```

src0 = gr.sig_source_f (sample_rate, gr.GR_SIN_WAVE, 350, ampl)
src1 = gr.sig_source_f (sample_rate, gr.GR_SIN_WAVE, 440, ampl)
dst = audio.sink (sample_rate, options.audio_output)

```

La fuente pertenece a la categoría de bloques. Los bloques están en una interface entre Python y C++. El bloque utilizado en esta parte es *gr.sig\_source\_f* el cual se encuentra en “*/usr/lib/python2.x/dist-packages/gnuradio/gr*”.

El módulo *audio.py* se encuentra en “*/usr/lib/python2.x/dist-packages/gnuradio/audio.py*” el cual es importado desde el inicio. Este módulo envía las señales a la tarjeta de audio para poder reproducirlas.

Ahora se debe realizar la conexión entre los bloques que se han definido anteriormente para completar el grafo:

```

self.connect (src0, (dst, 0))
self.connect (src1, (dst, 1))

```

Con esto se crea la conexión lógica del grafo uniendo los puertos de entrada y salida (*src*-fuente y *dst*-destino).

La última parte del programa consiste en hacer correr el flujo de datos a través del grafo:

```
if __name__ == '__main__':
    try:
        ejemplo_tono().run()
    except KeyboardInterrupt:
        pass
```

La primera línea sirve para ejecutar el programa directamente, es decir, a través de una Terminal con el comando `./dial_tone.py` ya que en ese caso la variable `_name_` y `_main_` serán iguales y se ejecutará el contenido del bucle. En caso de que el módulo que se ha creado sea llamado por otro módulo, la parte que se encuentra debajo de esta línea será ignorada y solamente se importará el contenido anterior a ésta. Es decir, esta línea sirve para hacer del código un módulo ejecutable o un sub-módulo. La siguiente línea es la que llama a la clase `ejemplo_tono` y hace que se procese con el método `run()`, para ello se realiza el intento con `try` y permanece a menos que se genere la excepción `KeyboardInterrupt` (Control+C), en cuyo caso se ejecuta la orden `pass` que no hace nada pero se utiliza en caso de que sea necesaria una sentencia sintácticamente hablando pero que el programa no haga nada.

El archivo `dial_tone.py` es un módulo básico para trabajar con GNU Radio, de ahí que es importante comenzar a programar con éste, ya que además de dar una clara introducción de cómo se realizan los grafos, bloques y conexiones, introduce a la programación de señales de audio.

No obstante, recordar que para realizar los diseños existe la herramienta GNU Radio Companion que evita al programador tener que trabajar directamente sobre el código aportando una interfaz de interconexión de bloques de procesado de señal.

## Apéndice D. Códigos Matlab con SDR4All

### D.1 Ficheros de introducción a SDR4All

#### Transmisor

```
%%Prueba SDR4All Transmisor
sock=SDR4All_Connect(0,'SlotB','TX'); % USRP #0, slot B
SDR4All_SetGain(sock,20); % Ganancia de transmisión máxima
SDR4All_SetFreq(sock,2422e6);
SDR4All_SetInterpRate(sock,256); % Tasa de muestras y ancho de banda de 500 kHz (128MHz/256)
%Definición de señal a transmitir
Basic      = kron(ones(6250,1),[ones(10,1);-ones(10,1)]); %125000 muestras 1,-1 alternativamente
Vide = zeros(125000,1);
```

```

Base = [Basic;Vide];
Sig = kron(ones(20,1),Base); % 20 ráfagas, Sig contiene 5e6 muestras
Te = (1:length(Sig))/(500e3); % 10 segundos de señal
plot(Te,Sig);
SDR4All_SendData(sock,Sig);

```

### **Receptor**

```

%%Prueba SDR4All Receptor
sock=SDR4All_Connect(0,'SlotA','RX'); % USRP #0 Slot A
[gain_min,gain_max,gain_step] = SDR4All_GetGain(sock);
[freq_min,freq_max] = SDR4All_GetFreq(sock);
SDR4All_SetGain(sock,(gain_max+gain_min)/2);
SDR4All_SetDecimRate(sock,128); % Tasa de muestras y ancho de banda de 500 kHz
SDR4All_SetFreq(sock,2422e6);
[Data] = SDR4All_GetData(sock,5*500e3); % Se reciben 5 segundos
Te = (1:length(Data))/(500e3);
plot(Te,real(Data));

```

## D.2 Ficheros diseño D-QPSK con SDR4All

### **Función pulso.m (Root Raised Cosine Filter)**

```

function [pt,t]=pulso(Ts,alpha)

t=-5*Ts:Ts/16:5*Ts;

%%Definición del pulso
pt = cos((1+alpha)*pi*t/Ts)+Ts*sin((1-alpha)*pi*t/Ts)./(4*alpha*t);
pt = pt./(1-(4*alpha*t/Ts).^2);
pt = pt.*4*alpha/(pi * sqrt(Ts));

%%Indices de los puntos singulares
pos_ceros_den_num=find(t==0);
pos_ceros_den1=find(t==Ts/(4*alpha));
pos_ceros_den2=find(t==-(Ts/(4*alpha)));

%Cálculo singularidades

```

```

if length(pos_ceros_den_num)>0
    pt(pos_ceros_den_num)=4*alpha/(pi*sqrt(Ts))*(1+(1-
alpha)*pi/(4*alpha));
end

t1=t(pos_ceros_den1);
if length(pos_ceros_den1)>0
    pt(pos_ceros_den1)=4*alpha/(pi * sqrt(Ts));
    pt(pos_ceros_den1)=pt.*(-
sin((1+alpha)*pi*t1/Ts)*(1+alpha)*pi/Ts + Ts/4/alpha * (t1*cos((1-
alpha)*pi*t1/Ts)*(1-alpha)*pi/Ts-sin((1-alpha)*pi*t1/Ts))/(t1^2));
    pt(pos_ceros_den1)=pt./(-2*(4*alpha*t1/Ts)*(4*alpha/Ts));
end

pt(pos_ceros_den2)=pt(pos_ceros_den1);

```

### Transmisor

```

%%-----Transmisor DQPSK-----
%%Obtención de los bits del fichero a enviar
clear all;
load bits_info;%Se cargan los datos previamente guardados
%-----%
% CODIFICACIÓN DE CANAL
%-----%
t = poly2trellis(3,[3 5]);
k = log2(t.numInputSymbols);
tblen=3;
[datos_c stateA]=convenc([bits_info zeros(1,tblen*k)],t); %Se añaden
ceros de flushing

%-----%
% MODULACIÓN DQPSK
%-----%
u=0:3;
QPSK_Symb = exp(2*i*pi*((2*u+1)/8));

```

```

est_inicial=QPSK_Symb(1);%estado inicial del que partimos
est_ant=est_inicial;
n=1;
for i=1:2:length(datos_c)
    symb=[datos_c(i) datos_c(i+1)];
    if symb==[0 0]
        symb_QPSK(n)=est_ant;
    elseif symb==[0 1]
        symb_QPSK(n)=est_ant*exp(j*pi/2);
    elseif symb==[1 1]
        symb_QPSK(n)=est_ant*exp(j*pi);
    elseif symb==[1 0]
        symb_QPSK(n)=est_ant*exp(-j*pi/2);
    end
    est_ant=symb_QPSK(n);%Actualizamos el valor actual de la fase
    n=n+1;
end
plot(real(symb_QPSK),imag(symb_QPSK),'b.'),title('Secuencia de
símbolos QPSK');
pause;
close all;

%-----%
%%FILTRO RRC
%-----%
ss=16;%muestras por símbolo
BW=500e3;
Ts=(1/BW)*ss;
alpha=0.22;
[pt,tp]=pulso(Ts,alpha,ss);
symb_QPSK_rem=kron(symb_QPSK,[1 zeros(1,(ss-1))]);
sig=conv(symb_QPSK_rem,pt);

h=spectrum.welch;
psd(h,sig,'CenterDC',true,'Fs',BW),title('PSD de la señal
transmitida');
pause,close all;
sig_n=sig*length(sig)/sum(abs(sig));%Se normaliza la señal a módulo 1
p_inicial=round(length(pt)/2);%punto inicial para representar la
constelación

```

```

plot(sig_n),hold on,plot(real(sig_n(p_inicial:ss:(end-
2*p_inicial))),imag(sig_n(p_inicial:ss:(end-
2*p_inicial))),'r.'),title('Diagrama vectorial y constelación de la
señal transmitida');

%-----%%SEÑAL TRANSMITIDA-----
%
sig_Tx=kron(ones(1,5),[200*ones(1,0.2*BW) zeros(1,0.1*BW)
zeros(1,0.1*BW)]);%Se transmiten 5 ráfagas con los mismos datos de
0,8 seg con 0,3 segundo entre ráfagas
Te = (1:length(sig_Tx))/BW;
figure,subplot(211),plot(Te,real(sig_Tx)),title('Parte en fase de la
señal transmitida'),xlabel('t(s)');
subplot(212),plot(Te,imag(sig_Tx)),title('Parte en cuadratura de la
señal transmitida'),xlabel('t(s)');
pause
close all

%-----%%TRANSMISIÓN-----
%
input('Pulsar enter para configurar parámetros de transmisión');
sock=SDR4All_Connect(0,'SlotA','TX'); % USRP #0, slot A
G=input('Seleccionar ganancia (dB) [0-20]: ');
SDR4All_SetGain(sock,G); % Ganancia de transmisión máxima
F=input('Seleccionar frecuencia (MHz) [2400-2700]: ');
SDR4All_SetFreq(sock,F*1e6);
input('Ancho de banda'),BW
SDR4All_SetInterpRate(sock,128e6/(BW)); % Tasa de muestras y ancho de
banda de 500 kHz (128MHz/256)
input('Pulsar enter para comenzar la transmisión');
SDR4All_SendData(sock,sig_Tx);
%------

```

### **Receptor**

```

%-----Receptor DQPSK-----
%-----%%RECEPCIÓN-----
%
```

```

clear all;
input('Pulsar enter para configurar parámetros de recepción');
sock=SDR4All_Connect(0,'SlotB','RX'); % USRP #0, slot B
G=input('Seleccionar ganancia (dB) [0-90]: ');
SDR4All_SetGain(sock,G); % Ganancia de transmisión máxima
F=input('Seleccionar frecuencia (MHz) [2400-2700]: ');
SDR4All_SetFreq(sock,F*1e6);
BW=500e3;
input('Ancho de banda'),BW
SDR4All_SetDecimRate(sock,64e6/(BW)); % Tasa de muestras y ancho de
banda de 500 kHz (128MHz/256)
input('Pulsar enter para comenzar la recepción');
[sig_Rx] = SDR4All_GetData(sock,10*BW);%10 segundos de señal
plot(real(sig_Rx));
pause;
%load('sigRX250070dB.mat');%Después de hacer la transmisión
sig_Rx=sig_Rx(1000:end);%Elimina posible transitorio inicial

ss=16;%muestras por símbolo
Te = (1:length(sig_Rx))/BW;
subplot(211),plot(Te,real(sig_Rx)),title('Señal recibida
(I)'), xlabel('t(s)');
subplot(212),plot(Te,imag(sig_Rx)),title('Señal recibida
(Q)'), xlabel('t(s)');
pause
close all;

%-----%
%%SELECCIÓN DE RÁFAGA
%-----%
v_rms=norm(sig_Rx)/sqrt(length(sig_Rx));
int_seg=v_rms*0.0;%intervalo para soportar un pico de interferencia
ind_mod=find(abs(sig_Rx)>(v_rms+int_seg));
ind_raf=ind_mod(1);
raf_Rx=sig_Rx(ind_raf:ind_raf+1.2*BW); %%se queda con 1.2 s (la
ráfaga de datos está en 0.8s finales)
Tr=(1:length(raf_Rx))/BW;
figure, subplot(211), plot(Tr,real(raf_Rx)), title('Parte en fase de la
ráfaga seleccionada'), xlabel('t(s)');
subplot(212), plot(Tr,imag(raf_Rx)), title('Parte en cuadratura de la
ráfaga seleccionada'), xlabel('t(s)');

```

```

pause,close all;

raf_Rx_datos=raf_Rx(0.3*BW:end);
Trd=(1:length(raf_Rx_datos))/BW;
figure,subplot(211),plot(Trd,real(raf_Rx_datos)),title('Parte en fase
de la ráfaga de datos'),xlabel('t(s)');
subplot(212),plot(Trd,imag(raf_Rx_datos)),title('Parte en cuadratura
de la ráfaga de datos'),xlabel('t(s)');
pause,close all;
h=spectrum.welch;
psd(h,raf_Rx_datos(1:(end-
0.2*BW)), 'CenterDC',true,'Fs',BW),title('PSD ráfaga Rx');
pause,close all;

%-----%%SINCRONIZACIÓN-----%
%%%Corrección de frecuencia-----
N=5*BW;
f=0:BW/N:(BW-BW/N);
psd(h,raf_Rx(1:(0.19*BW)), 'CenterDC',true,'Fs',BW),title('PSD ráfaga
de sincronización de frecuencia');
pause,close all;
[amp,f_off]=max(abs(fft(raf_Rx(1:0.19*BW),N)));%f_off será el valor
del eje x en número de muestra
f_off=f_off*BW/N;%se calcula f_off a partir del número de muestra del
máximo teniendo en cuenta el sobremuestreo
raf_Rx_f=raf_Rx_datos.*exp(-j*2*pi*f_off*Trd.');
psd(h,raf_Rx_f(1:(end-0.2*BW)), 'CenterDC',true,'Fs',BW),title('PSD
ráfaga sin offset de frecuencia');
pause,close all;

%%%Sincronización en tiempo-----
-- Selección del instante de muestreo óptimo en base a la varianza de
la magnitud para cada instante (16 posibilidades)
fin_raf=mod(length(raf_Rx_f),16);%Para eliminar las muestras finales
y reshape opere adecuadamente
raf_rec=reshape(raf_Rx_f(1:end-fin_raf),16,((length(raf_Rx_f)-
fin_raf)/16));

```

```

v=raf_rec.';%reshape va ordenando por columnas y se pretende tener en
cada fila 16 muestras
m=abs(v);
sigma=var(m);%calcula la varianza de cada una de las 16 columnas
plot(sigma),title('Varianza de la magnitud para cada instante de
muestreo'),xlabel('Instante de muestreo óptimo');
pause,close all;
[Y,inst_opt]=min(sigma);%el mínimo será el instante de muestreo
óptimo de los 16 posibles

%Se define ya el filtro RRC para calcular la correlación y el
instante
%óptimo de muestreo
Ts=(1/BW)*ss;
alpha=0.22;
[pt,tp]=pulso(Ts,alpha,ss);
num_desp=100;
[correlacion,lags]=xcorr(real(raf_Rx_f(1:num_desp)),pt);
plot(lags,abs(correlacion)),title('Correlación ráfaga con pulso
coseno alzado'),xlabel('desplazamiento (número de
muestras)'),pause,close all;
[max_corr,x_max]=max(abs(xcorr(real(raf_Rx_f(1:num_desp)),pt)));
desp_max=(x_max-length(pt))+length(pt)/2;%xcorr tiene desplazamientos
negativos y positivos
punto_muestreo=inst_opt+floor(desp_max/ss);%el punto de muestreo
inicial será el el instante de muestreo óptimo más cercano al máximo
de la correlación
plot(raf_Rx_f),hold
on,plot(real(raf_Rx_f(punto_muestreo:ss:(BW/2))),imag(raf_Rx_f((punto
_muestreo):ss:(BW/2))),'r.'),title('Diagrama vectorial y constelación
Rx tras ajustar offset');
pause
close all;

%-----%
%%FILTRO ADAPTADO RRC
%-----%
raf_Rx_fil=conv(pt,raf_Rx_f(1:0.85*BW));
subplot(211),plot(Tr(1:length(raf_Rx_fil)),real(raf_Rx_fil)),title('P
arte en fase tras filtro adaptado'),xlabel('t(s)');
subplot(212),plot(Tr(1:length(raf_Rx_fil)),imag(raf_Rx_fil)),title('P
arte en cuadratura tras filtro adaptado'),xlabel('t(s)');

```

```

pause,close all;
%punto_muestreo_fil=punto_muestreo+floor(length(pt)/2);% Será el
anterior
%más la mitad de la longitud del pulso coseno alzado---1ª
APROXIMACIÓN

%%%%%%%%%%%%% RESINCRONIZACIÓN EN TIEMPO
fin_raf=mod(length(raf_Rx_fil),16);%Para eliminar las muestras
finales y reshape opere adecuadamente
raf_rec=reshape(raf_Rx_fil(1:end-fin_raf),16,((length(raf_Rx_fil)-
fin_raf)/16));
v=raf_rec.';%reshape va ordenando por columnas y se pretende tener en
cada fila 16 muestras
m=abs(v);
sigma=var(m);%calcula la varianza de cada una de las 16 columnas
plot(sigma),title('Varianza de la magnitud para cada instante de
muestreo'),xlabel('Instante de muestreo óptimo');
pause,close all;
[Y,inst_optRRC]=min(sigma);%el mínimo será el instante de muestreo
óptimo de los 16 posibles
num_despRRC=200;
[correlacionRRC,lags]=xcorr(real(raf_Rx_fil(1:num_despRRC)),pt);
plot(lags,abs(correlacionRRC)),title('Correlación ráfaga tras filtro
con pulso coseno alzado'),xlabel('desplazamiento (número de
muestras)'),pause,close all;
[max_corr,x_maxRRC]=max(abs(xcorr(real(raf_Rx_fil(1:num_despRRC)),pt)
));
desp_maxRRC=(x_maxRRC-num_despRRC)+length(pt)/2;%xcorr tiene
desplazamientos negativos y positivos
punto_muestreo_fil=inst_optRRC+floor(desp_maxRRC/ss)*ss;%el punto de
muestreo inicial será el el instante de muestreo óptimo más cercano
al máximo de la correlación
%%%%%%%%%%

figure, plot(raf_Rx_fil),hold on,
plot(real(raf_Rx_fil((punto_muestreo_fil):ss:(BW/2))),imag(raf_Rx_fil
((punto_muestreo_fil):ss:(BW/2))),'r.'),title('Constelación tras
filtro adaptado');%El punto inicial será el seleccionado para las
constelaciones anteriores más 81 (length(pt)/2)
pause
close all;

```

```

psd(h,raf_Rx_fil(1:(end-0.2*BW)), 'CenterDC',true,'Fs',BW),title('PSD
ráfaga tras filtro adaptado');

pause,close all;

%-----%
%%DEMODULADOR DQPSK
%-----%
r=raf_Rx_fil(punto_muestreo_fil:ss:length(raf_Rx_fil));
est_anterior=abs(real(r(1)))+j*abs(imag(r(1)));
n=1;
for h=1:length(r)
    dif1=abs(r(h)-est_anterior);
    dif2=abs(r(h)-est_anterior*exp(j*pi/2));
    dif3=abs(r(h)-est_anterior*exp(j*pi));
    dif4=abs(r(h)-est_anterior*exp(-j*pi/2));

    vec_dif=[dif1 dif2 dif3 dif4];
    [min_dif,pos_min_dif]=min(vec_dif);

    if pos_min_dif==1
        datos_dem(n:n+1)=[0 0];
    else
        if pos_min_dif==2
            datos_dem(n:n+1)=[0 1];
        else
            if pos_min_dif==3
                datos_dem(n:n+1)=[1 1];
            else
                datos_dem(n:n+1)=[1 0];
            end
        end
    end
    n=n+2;
    est_anterior=r(h);
end

%-----%

```

```

%%Tasa de error bits codificados
%-----
load datos_c;%carga la variable datos_c
Pc=sum(abs(datos_dem(1:length(datos_c))-datos_c))/length(datos_c)%Tasa de error de bits codificados
plot(datos_dem(1:length(datos_c))-datos_c),title('Errores en bits codificados'),axis([0 50006 -2 2]),pause,close all;

%-----
%%DECODIFICADOR DE CANAL
%-----
t = poly2trellis(3,[3 5]);
k = log2(t.numInputSymbols);
tblen = 3;
[d m p in] = vitdec(datos_c,t,tblen,'cont','hard');
datos_dec=d(tblen*k+1:end);%Se elimina el transitorio inicial

%-----
%%BER
%-----
load datos_b;
BER=sum(abs(datos_dec(1:length(datos_b))-datos_b))/length(datos_b)%Tasa de error de bits codificados
plot(datos_dec(1:length(datos_b))-datos_b),title('Errores en bits'),axis([0 25000 -2 2]),pause,close all;

```

## Apéndice E. Implementaciones realizadas

En este Anexo se listarán y describirán todos los diseños que se han llevado a cabo con el equipo Universal Software Radio Peripheral, tanto para la plataforma GNU Radio como para el entorno Matlab.

### E.1 Diseños GNU Radio

Los siguientes diseños se realizaron directamente en código Python.

- **tonoprueba.py**: Fichero que envía la señal compuesta por un tono a la tarjeta de sonido.
- **mod\_FM.py**: Transmisor FM de banda estrecha.
- **dem\_FM.py**: Receptor FM de banda estrecha.

Las implementaciones con la herramienta GNU Radio Companion se listan a continuación, cada fichero Python ha sido obtenido a partir del fichero grc (esquemático) correspondiente:

- **analizador.py**: Analizador de espectros. Disponible para tarjeta TVRX y RFX2400.
- **Tx\_FM.py**: Transmisor FM de banda estrecha.
- **Rec\_FM.py**: Receptor FM de banda estrecha.
- **Tx\_WFM.py**: Transmisor FM de banda ancha.
- **Rec\_WFM.py**: Receptor FM de banda ancha.
- **Tx\_QPSK.py**: Transmisor (D)QPSK creado a partir de todos los bloques necesarios en lugar de utilizar el bloque modulador QPSK de la arquitectura GNU Radio que engloba a la mayoría de los mismos.
- **Rx\_QPSK.py**: Receptor (D)QPSK creado a partir de todos los bloques necesarios en lugar de utilizar el bloque demodulador QPSK de la arquitectura GNU Radio que engloba a la mayoría de los mismos.
- **Tx\_Digital.py**: Transmisor digital donde se puede seleccionar diferentes tipos de modulaciones: DQPSK, DBPSK ó GMSK utilizando los bloques que engloban la mayoría de los subbloques necesarios.
- **Rx\_Digital.py**: Receptor digital donde el tipo de modulación se puede seleccionar entre DQPSK, DBPSK ó GMSK utilizando los demoduladores que engloban la mayoría de los subbloques necesarios.
- **Tx\_OFDM.py**: Transmisor que utiliza la modulación OFDM.
- **Rx\_OFDM.py**: Receptor para la modulación OFDM.
- **Ecuación\_audio.py**: Diseño para realizar el ecualizado de la señal de audio y eliminar frecuencia de resonancia del sistema bajo estudio a partir de un banco de filtros.
- **qpsk\_sim.py**: Simulación de la modulación QPSK sin necesidad de USRP.
- **param.py**: Implementación para una primera toma de contacto con los parámetros de la arquitectura GNU Radio y USRP.

## E.2 Diseños entorno Matlab

- **analizador\_de\_espectro.mdl**: Analizador de espectro para la tarjeta secundaria RFX2400 utilizando el entorno Simulink.
- **Diseño Hello World**: Creación de ficheros *Transmisor.m* y *Receptor.m* para el envío de una serie de ráfagas de prueba entre 2 USRPs.
- **Diseño QPSK**: Creación de ficheros *Transmisor\_QPSK.m* y *Receptor\_QPSK.m* para el envío de bits de información utilizando modulación QPSK.
- **Diseño DQPSK**: Implementación de ficheros *Transmisor\_DQPSK.m* y *Receptor\_QPSK.m* para el envío de información utilizando modulación DQPSK.



## BIBLIOGRAFÍA

---

- [1] Página web GNU Radio, <http://gnuradio.org/>.
- [2] Página Web Josh Blum, <http://www.joshknows.com/gnuradio>.
- [3] Tutorial de programación Python, <http://docs.python.org/tutorial/>.
- [4] Información sobre Modulación por División en Frecuencia Ortogonal, [http://en.wikipedia.org/wiki/Orthogonal\\_frequency-division\\_multiplexing](http://en.wikipedia.org/wiki/Orthogonal_frequency-division_multiplexing).
- [5] José María Hernando Rábano. “Transmisión por radio”, 6<sup>a</sup> Edición. Editorial Ramón Areces.
- [6] Página Web de instalación del toolbox USRP-SIMULINK, <http://www.cel.kit.edu/usrp.php>.
- [7] Página Web con varios diseños GNU Radio, <http://www.swigerco.com/gnuradio>.
- [8] Página Web de Alexandru Csete con ejemplos de GNU Radio Companion, <http://www.oz9aec.net/index.php/gnu-radio/grc-examples>.
- [9] Página Web del toolbox SDR4all, <http://www.tools4sdr.com/wiki/Tools4SDR>.
- [10] W. Tuttlebee. Software Defined Radio: Origins, drivers and international perspectives. John Wiley & Sons, Inc. Nueva York. 2002.
- [11] Recomendación ERC/REC 70-03 relativa al uso de dispositivos de corto alcance.
- [12] Información detallada del bloque de sincronización MPSK Receiver, [http://gnuradio.org/doc/doxygen/classgr\\_mpsk\\_receiver\\_cc.html#details](http://gnuradio.org/doc/doxygen/classgr_mpsk_receiver_cc.html#details).
- [13] Enlace a repositorio GNU Radio para escribir un bloque de señal en GNU Radio: <http://www.gnu.org/software/gnuradio/doc/howto-write-a-block.html>.
- [14] Página web del fabricante Ettus Research, creador del Universal Software Radio Peripheral, <http://www.ettus.com/>.
- [15] Enlace a características del filtro interpolador CIC de GNU Radio: <http://gnuradio.org/redmine/projects/gnuradio/wiki/UsrpFAQDDC>.



## Índice de Palabras

- Antena  
    VERT2400, 25  
    VERT400, 25
- BER, 94, 96, 113, 116, 129, 130, 158
- BPSK, 93, 97, 114
- Chunks, 63, 64, 65
- Codificación  
    canal, 116  
    Gray, 63, 74, 81, 82, 117
- Constelación, 68, 69, 76, 77, 156
- Convertidor  
    Analógico a digital, 13, 14, 19, 20, 23, 49, 71, 139, 140, 141  
    Digital a analógico, 13, 14, 19, 20, 21, 23, 46, 71, 140, 141, 142
- DQPSK, 37, 76, 78, 81, 82, 83, 84, 89, 93, 94, 115, 116, 117, 118, 120, 128, 129, 150, 152, 157, 159
- Espectro, 59, 69, 75, 77, 78, 79, 83, 90, 93, 99, 103
- Ettus, vii, viii, 12, 17, 105, 106, 133, 161
- USRP, viii, 15, 17, 18, 19, 20, 21, 23, 26, 27, 28, 32, 33, 34, 35, 37, 38, 39, 40, 46, 49, 51, 52, 54, 58, 59, 66, 67, 71, 72, 76, 78, 82, 87, 97, 98, 101, 102, 103, 105, 106, 107, 108, 109, 111, 112, 113, 114, 115, 118, 119, 120, 121, 125, 133, 136, 139, 140, 141, 142, 143, 148, 149, 152, 153, 159
- Filtro  
    Adaptado, 127  
    CIC, 59, 103, 161  
    FIR, 65
- FM, 21, 23, 24, 37, 44, 45, 46, 47, 48, 49, 50, 51, 53, 55, 56, 57, 58, 141, 158, 159
- FPGA, 16, 17, 19, 20, 21, 27, 31, 140
- Frecuencia Intermedia, 11, 12, 13, 14, 17, 19, 141
- GMSK, 37, 78, 81, 82, 84, 86, 89, 92, 93, 94, 159
- GNU  
    Radio, viii, 133  
    radio companion, viii, 133
- Instalación, 27, 31, 32, 106, 107, 136
- Linux, 18, 30, 32, 133, 136  
    Ubuntu, 28, 30, 31, 32, 33, 37, 46, 137
- Matlab, viii, 30, 33, 105, 106, 108, 109, 111, 112, 113, 114, 120, 125, 141, 142, 148, 158, 159
- NCO, 72, 87, 142, 143
- Nyquist, 14, 20, 140
- OFDM, 37, 96, 97, 98, 99, 100, 101, 102, 103, 159
- Placa  
    Principal, 143  
    RFX2400, 25, 26, 38, 65, 159  
    Secundaria, 25, 140  
    TVRX, 23, 24, 37, 38, 39, 42, 55, 159  
    WBX, 24, 27, 37, 38, 50
- Python, viii, 27, 30, 31, 32, 33, 34, 35, 44, 50, 57, 133, 136, 137, 141, 144, 145, 147, 158, 161
- QPSK, 37, 58, 60, 61, 63, 64, 65, 67, 68, 69, 70, 71, 73, 76, 87, 99, 115, 150, 151, 159
- Radiofrecuencia, 12, 13, 14, 16, 17, 19, 20, 21, 46, 50, 52
- Raíz de coseno alzado, 65, 66, 117, 125, 126, 127, 128, 129, 130, 151, 155
- SDR, iii, vii, viii, 11, 12, 13, 14, 15, 24, 30, 33, 35, 133
- SDR4All, 112, 113, 114, 115, 119, 148, 149, 152, 153
- Simulink, viii, 30, 33, 105, 106, 109, 110, 159
- Sincronización  
    tiempo, 154

Slidder, 34, 35, 39, 40, 41

Universal Software Radio Peripheral, vii, 12, 15, 17, 19, 105, 136, 158, 161

USRP, viii, 15, 17, 18, 19, 20, 21, 23, 26, 27, 28, 32, 33, 34, 35, 37, 38, 39, 40, 46, 49, 51, 52, 54, 58, 59, 66, 67, 71, 72, 76, 78, 82, 87, 97, 98, 101, 102, 103, 105, 106, 107, 108, 109, 111, 112, 113, 114, 115, 118, 119, 120, 121, 125, 133, 136, 139, 140, 141, 142, 143, 148, 149, 152, 153, 159

USB, Univesal serial peripheral, 16, 17, 19, 20, 21, 27, 28, 31, 38, 48, 55, 59, 66, 105, 106, 120, 133, 140, 141, 142, 143

Windows, 18, 30, 106, 109, 111, 133

Zero-IF, 13, 14

## Glosario

BER (Bit Error Rate): tasa que indica el número de bits erróneos frente al total de bits enviados.

BB (Band Base): banda base, esto es, la señal no ha sido modulada con una portadora. En comunicaciones digitales se suele montar o preparar la señal en banda base y luego se sube en frecuencia con una portadora para transmitirla. En el receptor, se estiman los datos transmitidos a partir de lo que se ha recibido en antena una vez que se ha bajado a banda base.

Buffer: Un buffer de datos es una ubicación de la memoria en un ordenador o en un instrumento digital reservada para el almacenamiento temporal de información digital

BPSK (Binary Phase Shift Keying): [modulación](#) digital angular que consiste en hacer variar la fase de la [portadora](#) entre dos valores.

Codificación: conversión de un sistema de datos de origen a otro sistema de datos de destino. De ello se desprende como corolario que la información contenida en esos datos resultantes deberá ser equivalente a la información de origen.

DBPSK/DQPSK (Differential BPSK/QPSK): modulación digital que se basa en hacer variar la fase de la portadora en función de la diferencia entre los valores binarios de la señal de información.

GMSK (Gaussian Minimum Shift Keying): modulación que deriva de la MSK utilizando un filtro gaussiano para reducir los lóbulos laterales del espectro.

GNU Radio: conjunto de archivos y aplicaciones agrupadas en librerías, que permiten manipular señales mediante procesado digital.

IF (Intermediate Frequency): frecuencia intermedia a la que se lleva la señal modulada antes de pasarla a radiofrecuencia en el transmisor o a banda base en el receptor.

Linux: sistema operativo basado en un núcleo de código libre. Todo su [código fuente](#) puede ser utilizado, modificado y redistribuido libremente por cualquiera bajo los términos de la GPL.

LNA (Low Noise Amplifier): amplificador de bajo ruido, normalmente el primer amplificador que encuentra la señal recibida en antena. Este amplificador es la parte central del *front-end* o cabecera.

Matlab: software matemático que ofrece un [entorno de desarrollo integrado](#) (IDE) con un lenguaje de programación propio. Entre sus prestaciones básicas se hallan: la manipulación de [matrices](#), la representación de datos y funciones, la implementación de [algoritmos](#), la creación de interfaces de usuario ([GUI](#)) y la comunicación con programas en otros [lenguajes](#) y con otros dispositivos [hardware](#).

NCO (Numerically Controlled Oscillator): generador de señal digital capaz de crear una representación síncrona en tiempo discreto de una forma de onda, usualmente sinusoidal.

QPSK (Quadrature Phase Shift Keying): [modulación](#) digital angular que consiste en hacer variar la fase de la [portadora](#) entre cuatro valores.

OFDM (Orthogonal Frequency Division Multiplexing): modulación ortogonal por división de frecuencias.

PA (Power Amplifier): amplificación de potencia, normalmente el amplificador de radiofrecuencia en el transmisor.

RF (radiofrecuencia): frecuencias a las que se envía la señal por antena. Generalmente la frecuencia de radiofrecuencia se denota por la frecuencia de portadora.

SDR (Software Defined Radio): sistema de radiocomunicación donde la mayor parte de los componentes necesarios se implementan en software en lugar de en hardware.

SDR4All: Toolbox utilizado para transmitir y recibir con el USRP a través de Matlab.

Simulink: Entorno de [programación visual](#), o por bloques, que funciona sobre el entorno de programación [Matlab](#).

USRP (Universal Software Radio Peripheral): equipo del fabricante Ettus Research utilizado para la construcción de sistemas SDR diseñado para trabajar en conjunto con un procesador externo.

USB Universal Serial Bus (bus universal en serie): estándar que permite la existencia de un puerto en el ordenador y otros dispositivos que sirve para conectar periféricos.