SISTEMA DE COMUNICACIÓN DIGITAL M-ARIA SOBRE UNA PLATAFORMA DE RADIO DEFINIDO POR SOFTWARE (SDR) BASADO EN GNU RADIO

ANDRÉS FRANCISCO GÓMEZ GARCÍA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
BUCARAMANGA
2015

SISTEMA DE COMUNICACIÓN DIGITAL M-ARIA SOBRE UNA PLATAFORMA DE RADIO DEFINIDO POR SOFTWARE (SDR) BASADO EN GNU RADIO

ANDRÉS FRANCISCO GÓMEZ GARCÍA

Trabajo de grado para optar al título de
Ingeniero electrónico

Director
Óscar Mauricio Reyes Torres
Ingeniero Electrónico, Ph.D

Codirector
Jorge Hernando Ramón Suarez
Ingeniero Electricista, MSc

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
BUCARAMANGA
2015

TABLA DE CONTENIDO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABLAS

Pág.

LISTA DE ANEXOS

ABREVIATURAS Y ACRÓNIMOS

ADC	Convertidor analógico-digital (Analog to Digital Converter)
API	Interfaz de programación de aplicación (Application Programming Interface)
BPSK	Modulación por desplazamiento de fase binaria (Binary Phase Shift Keying)
DAC	Convertidor digital-analógico (Digital to Analog Converter)
dBc	Decibels relative to the carrier
FLL	Lazo de seguimiento de frecuencia (Frequency-locked loop)
GRC	GNU Radio Companion
GRASS GIS	Geographic Resources Analysis Support System - Geographic Information System
GUI	Interfaz gráfica de usuario (*Graphical User Interface*)
ISI	Interferencia intersímbolo (Inter-Symbol Interference)
ISM	Industrial, Scientific and Medical
LO	Oscilador local (Local Oscillator)
NCO	Oscilador controlado numéricamente (Numerically controlled oscillator)
NI	National Instruments
PC	Computador personal (Personal Computer)
PLL	Lazo de seguimiento de fase (Phase-locked loop)
ppm	Partes por millón

PSK	Modulación por desplazamiento de fase (Phase Shift Keying)
QAM	Modulación de amplitud en cuadratura (Quadrature amplitude modulation)
RRC	Root raised cosine
Rx	Reception en telecommunications
R&S	Rohde & Schwarz
SFDR	Spurious-Free Dynamic Range
SDR	Radio definida por software (Software Defined Radio)
Tx	Transmisión en telecomunicaciones
UHD	USRP Hardware Driver
USB	Universal Serial Bus
USRP	Universal Software Radio Peripheral

RESUMEN

TÍTULO: SISTEMA DE COMUNICACIÓN DIGITAL M-ARIA SOBRE UNA PLATAFORMA DE RADIO DEFINIDO POR SOFTWARE (SDR) BASADO EN GNU RADIO*

AUTOR: ANDRÉS FRANCISCO GÓMEZ GARCÍA**

PALABRAS CLAVE: USRP1, RFX2400, Comunicaciones Digitales, Radio Definido por Software (SDR), GNU Radio

DESCRIPCIÓN:

El desarrollo de este trabajo de grado constituye una continuación de la investigación iniciada con el proyecto de grado: "DISEÑO DE MANUAL DE PRACTICAS PARA UN LABORATORIO DE COMUNICACIONES DIGITALES BASADO EN LA TECNICA DE RADIO DEFINIDO POR SOFTWARE". Se busca proporcionar soporte teórico-práctico en este tema y explorar la versatilidad que ofrece la tecnología SDR para ser empleada tanto en docencia como en investigación.

Este trabajo de grado presenta el diseño de un modulador y demodulador general construido usando la herramienta gráfica GRC de la plataforma software GNU Radio. Se propone y explica un sistema de modulación/demodulación M-aria que permite analizar el flujo de los datos y las características de las señales durante todo el proceso. Para esto, se desglosa la estructura de un de bloque de modulación y otro de demodulación presente en las librerías de la plataforma. Con esto, se busca brindar herramientas para entender fácilmente el funcionamiento del software y de la tecnología SDR. Además, diferentes tipos de modulaciones digitales (2, 4, 8 PSK y 16 QAM) pueden ser implementadas modificando sólo algunas variables clave, lo cual redunda en una mayor versatilidad del sistema diseñado.

Estos resultados se complementan con una corta reseña bibliográfica que orienta la lectura de aspectos fundamentales para incursionar en la tecnología SDR. Además se enuncian características, aplicaciones y posibles esquemas de conexión de los dispositivos utilizados (USRP1 y tarjeta secundaria RFX2400). Asimismo, las herramientas y bloques en GRC más relevantes para el desarrollo de este trabajo son descritas. Finalmente, se presenta una síntesis de la construcción del sistema en GRC en forma de video tutorial, como complemento a los resultados de este proyecto.

---

* Proyecto de Grado

** Facultad de Ingenierías Físico-mecánicas. Escuela de Ingeniería Eléctrica, Electrónica y Telecomunicaciones. Director: Dr.-Ing. Óscar Mauricio Reyes Torres. Codirector: MSc. Jorge Hernando Ramón Suarez

SUMMARY

TITLE: M-ARIA DIGITAL COMMUNICATION SYSTEM ON A PLATFORM OF SOFTWARE DEFINED RADIO (SDR) BASED IN GNU RADIO*

AUTHOR: ANDRÉS FRANCISCO GÓMEZ GARCÍA**

KEYWORDS: USRP1, RFX2400, Digitals Communications, Software Defined Radio (SDR), GNU Radio

DESCRIPTION:

The development of this project is a continuation of the research started with the graduation project: "DISEÑO DE MANUAL DE PRACTICAS PARA UN LABORATORIO DE COMUNICACIONES DIGITALES BASADO EN LA TECNICA DE RADIO DEFINIDO POR SOFTWARE". It seeks to provide theoretical and practical support on this issue and explore the versatility of SDR technology to be used in both teaching and research.

This grade work presents to design of a general modulator and demodulator built using the graphical tool GRC of the software platform GNU Radio. It proposes and explains an M-aria system of modulation/demodulation that allows analyze the flow of data and features of signals through the process. For this, the structure of a block of modulation and other of modulation presents in libraries in this platform is broken down. With this, it seeks to provide tools to easily understand the operation of the software and SDR technology. In addition, different types of digital modulations (2, 4, 8 PSK y 16 QAM) can be implemented by modifying just some key variables, which results in a greater versatility of the designed system.

These results are complemented by a short bibliographic review that guides reading of fundamentals aspects for venture in the SDR technology. Moreover, applications and possible connections diagrams of the device used (USRP1 and daughterboard) are enunciated. Also, tools and blocks in GRC most relevant for development of this work are described. Finally, a summary of the construction of the systems in GRC is presented as a tutorial video, complementing to the results of this project.

* Grade Works

** Facultad de Ingenierías Físico-mecánicas. Escuela de Ingeniería Eléctrica, Electrónica y Telecomunicaciones. Director: Dr.-Ing. Óscar Mauricio Reyes Torres. Codirector: MSc. Jorge Hernando Ramón Suarez

INTRODUCCIÓN

Radio definido por software o SDR es una tecnología que surgió como solución a las limitaciones que se presentan cuando se tienen sistemas de transmisión y recepción no compatibles, lo cual implica realizar inversiones adicionales si se desea establecer una comunicación entre dichos sistemas. El SDR integra hardware y software que, en conjunto, brindan la capacidad de recibir, procesar y transmitir diferentes clases de señales de radio, brindando una arquitectura flexible que permite programar los radios mediante software.

En la actualidad, además de las aplicaciones afines que utilizan sistemas basados en la tecnología SDR, ésta es también utilizada en el sector de la educación debido a la facilidad de reconfiguración, lo que posibilita realizar procesado de señal o reproducir diferentes técnicas de modulación sin que esto conlleve un cambio en hardware, permitiendo a los estudiantes interactuar de forma rápida con diversos sistemas de comunicaciones digitales en comparación con los sistemas implementados sólo por hardware.

En este documento se presenta el diseño de un modulador y demodulador general desarrollado empleando la herramienta gráfica GRC de la plataforma software GNU Radio, comenzando por una corta reseña bibliográfica de documentos y sitios web con el fin de orientar la lectura de aspectos fundamentales para incursionar en la tecnología SDR. Se incluye una descripción de características de los dispositivos usados (USRP1 – tarjeta secundaria RFX2400), aplicaciones y esquemas de conexión. Posteriormente se presenta el desarrollo del sistema de comunicación digital propuesto basado en librerías genéricas de GNU Radio junto con aspectos claves como variables y definición de las constelaciones, para así luego mostrar los resultados, tanto simulados como pruebas reales, del diseño propuesto usando los elementos antes mencionados. Se finaliza con las conclusiones pertinentes y recomendaciones para trabajos futuros en SDR en la Universidad Industrial de Santander.

Entre los anexos se encuentra información de algunas herramientas necesarias y a tener en cuenta a la hora de crear flujogramas en GRC, así como información de los bloques empleados en el diseño, algunos de ellos detallados con ejemplos. También se dispone de un instructivo para seleccionar el sub-dispositivo y puerto de antena en la USRP1, y algunas notas de aplicación de Ettus Research con aspectos pertinentes de los dispositivos USRP y tarjetas secundarias.

Como complemento al resultado del desarrollo de este trabajo de grado, se entrega un video tutorial en el cual se muestra una síntesis de lo expuesto en el presente trabajo.

1. OBJETIVOS

OBJETIVO GENERAL

Desarrollar el modelo de un sistema de comunicación digital M-aria basado en Radio Definido por Software (SDR) utilizando como software base GNU Radio.

OBJETIVO ESPECIFICO

- Caracterizar los equipos disponibles para la realización del diseño a proponer y los requerimientos del mismo.

- Diseñar un sistema de modulación/demodulación usando modulaciones 2, 4, 8 PSK y 16 QAM configurable, basado en el software GNU Radio

- Proponer y realizar pruebas de funcionalidad de los sistemas diseñados.

- Elaborar material de apoyo para la transferencia del conocimiento adquirido.

2. RESEÑA BIBLIOGRÁFICA

A continuación en la Tabla 1 se muestra una breve reseña de trabajos realizados usando la tecnología SDR, dispositivos USRP, tarjetas secundarias y diversas plataformas de software. También se encuentran documentos y sitios web que contienen información relevante para el desarrollo del presente proyecto. Esta información se incluye con el propósito de dar a conocer al lector algunas aplicaciones en las que pueden ser utilizados los dispositivos USRP así como proporcionar una guía que oriente la lectura de aspectos fundamentales para incursionar en el uso de la tecnología SDR.

Tabla 1. Tabla de proyectos en SDR y documentación

Documento	Descripción
Introducción a SDR y prácticas de laboratorio [1].	En este documento se presentan los siguientes temas: • Historia y definición de SDR. • Definición del USRP1 y sus características e instalación de componentes. • Resumen de las características de las tarjetas secundarias y antenas distribuidas por Ettus Research disponibles a la fecha. • Instalación y uso de GNU Radio junto con el USRP1. • Uso de la USRP1 junto con la herramienta Simulink de Matlab. • Ejemplos de implementaciones en GNU Radio y en Matlab.
Materiales de Tom Rondeau[1] [2].	En la sección de ejemplos se encuentra una introducción a la tecnología de SDR y a GNU Radio, así como también material en el que se explica con ejemplos temas como el filtrado y modulación digital en GRC.
Archivos Ettus Research [3].	En esta página se encuentran los esquemáticos de los USRP y tarjetas secundarias disponibles en Ettus Resarch, las imágenes necesarias para cargar las aplicaciones, binarios de diferentes versiones de *gnuradio* y *uhd* para la instalación en diferentes distribuciones, tutoriales de Python y GRC, y algunos ejemplos en GRC.

[1] Tom Rondeau es uno de los desarrolladores actuales que trabaja en el proyecto GNU Radio.

Tabla 1. (Continuación)

Documento	Descripción
Tutoriales GNU Radio [4].	Este documento cuenta con cinco laboratorios en los que se hace una introducción a GRC con ejemplos simples (p ej. crear y graficar una onda seno). Se explican detalles importantes de algunos bloques y sus parámetros. Así mismo, se presentan conceptos clave para el uso de SDR en aplicaciones reales (p. ej. transmisor y receptor de radio FM).
Aspectos de seguridad en SDR [5].	Documento en el que se propone un método para mejorar la seguridad de un sistema de comunicación digital mediante variaciones en la codificación de los puntos de mensaje en el diagrama de constelación usando código gray en una modulación rectangular QAM, con el propósito de usar este método en SDR debido a la facilidad de configuración de características tales como la codificación de los puntos en el diagrama de constelación.
Sistemas de comunicaciones usando SDR [6].	En este documento se presenta un estudio de la tecnología SDR exponiendo fundamentos, alcances y componentes físicos que lo conforman, así como pruebas para caracterizar estos componentes. Muestra un estudio del software GNU Radio usado para elaborar los diseños propuestos por el autor.
Repetidor de radio [7].	En este documento se presenta el diseño de un repetidor de radio usando la tecnología SDR. Este repetidor es implementado en el software GNU Radio junto con un dispositivo USRP1, mostrando con esto las ventajas de flexibilidad que brinda la tecnología SDR, además de capacidades y características del diseño realizado.
Pruebas sobre OFDM [8].	En este proyecto se construye un banco de pruebas para evaluar las características de los sistemas basados en OFDM en diferentes condiciones de propagación y diferentes configuraciones del sistema, teniendo como principal estudio el error práctico. Los diseños propuestos son basados en la tecnología SDR y se desarrollan usando la plataforma software GNU Radio y el dispositivo USRP2.

3. CARACTERIZACIÓN DE LOS EQUIPOS DE RADIO DISPONIBLES

En este capítulo se presentan características generales de los dispositivos principales usados en este proyecto (USRP1 – RFX2400 de Ettus Research[2]), así como los principales campos de aplicación en los que son usados.

Finalmente se ilustran diferentes esquemas de conexión que se pueden realizar usando estos dispositivos USRP, en función de la clase de aplicación que se desee implementar.

3.1 USRP1 – ETTUS RESEARCH

La Tabla 2 muestra un listado de las principales características del dispositivo USRP1 (Figura 1), así como especificaciones de voltaje, corriente, DAC, ADC y exactitud de frecuencia.

Figura 1. Dispositivo USRP1

Fuente: Ettus Research, USRP1. Disponible en: <www.ettus.com/product/details/USRPPKG>

[2] Desde el 2010 Ettus Research pasa a ser una compañía filial de National Instruments, agregando compatibilidad a los dispositivos de Ettus Research con los productos de NI.

Tabla 2. Características generales y especificaciones del dispositivo USRP1

Características USRP1	
Arquitectura modular: DC – 6 GHz	
2 ranuras para tarjetas secundarias	
DDC/DUC con 15 mHz de resolución	
FPGA Altera Cyclone	
Características USRP1	
Transmisión de hasta 16 MS/s por USB	
Interfaz USB 2.0 con procesador Host	
Entradas-salidas digitales y análogas auxiliares	
Especificaciones	
Entrada DC	6 [V]
Consumo de corriente	0,7 [A]
Tasa muestreo ADC	64 [MS/s]
Resolución ADC	12 Bits
Ancho de banda SFDR ADC	85 [dBc]
Tasa muestreo DAC	128 [MS/s]
Resolución DAC	14 bits
Ancho de banda SFDR DAC	83 [dBc]
Exactitud de frecuencia	25 ppm

Datos tomados de las especificaciones técnicas del dispositivo USRP1.

A continuación, en la Tabla 3, se presentan algunas propiedades que se desprenden de la característica del dispositivo USRP1 de poseer dos ranuras para tarjetas secundarias. Cabe resaltar que el dispositivo USRP1 es el único dispositivo del fabricante Ettus Research que posee esta característica.

Tabla 3. Propiedades de USRP1

Propiedades
Conectividad por dos canales completos de Tx/Rx
Alto aislamiento entre los canales de Tx/Rx
MIMO 2X2
Operaciones de doble banda

El dispositivo USRP1 utiliza el controlador de hardware USRP (UHD) para conectarse a un equipo de cómputo. UHD es el controlador oficial para todos los dispositivos USRP de Ettus Research, siendo éste soportado por sistemas operativos como Linux, Mac OSX y Windows, así como compatible con software

de terceros, como GNU Radio, LabVIEW, Simulink, OpenBTS, Iris, Redhawk y Amarisoft LTE eNodeB [10].

El USRP1 puede transmitir hasta 16 MS/s; sin embargo, cuando se utilizan las dos tarjetas secundarias, y debido a la interfaz de conexión USB 2.0, se tiene un límite en el rendimiento combinado de los canales, el cual no es mayor a 8 MS/s. Note que estas limitaciones se basan en el rendimiento de los datos proporcionados por la interfaz de conexión entre el USRP y el equipo de cómputo. Es importante tener en cuenta el rendimiento de la plataforma de procesamiento y la intensidad computacional que exige la aplicación, pues las restricciones de la plataforma de procesamiento son independientes de la capacidad total de los USRP de Ettus Research y el UHD [9].

La arquitectura del dispositivo USRP1 incluye una FPGA Altera Cyclone EP1C12 que puede ser personalizada o configurada con el fin de otorgar funcionalidad adicional al USRP, contribuyendo a la disminución del procesamiento de datos en el equipo de cómputo. La FPGA Altera Cyclone puede ser programada con el software Quartus II, el cual cuenta con una versión paga, la versión de prueba de 30 días y edición web gratuita de Quartus II [11].

3.2 RFX2400 (2,3 – 2,9 GHz) Rx/Tx

Esta tarjeta secundaria es un transceptor diseñado para operar en la banda de frecuencia de 2,4 GHz. En la Tabla 4 se encuentran las características de la tarjeta secundaria RFX2400.

Esta tarjeta cuenta con dos LO’s y sintetizadores independientes en recepción y transmisión que permiten una operación full-dúplex en diferentes frecuencias de transmisión y recepción.

El puerto Rx2 tiene acceso a todo el rango de frecuencia disponible en la tarjeta RFX2400. Por otro lado, el puerto Tx/Rx posee un filtro pasa banda alrededor de la banda ISM (2400-2483 MHz) para proporcionar selectividad y rendimiento.

Como se observa en la Tabla 4, el usuario puede configurar la recepción de señal en las antenas Tx/Rx o Rx2. Sin embargo, cuando se hace uso de la tarjeta en

modo de operación full-dúplex, la antena receptora es puesta automáticamente en Rx2 [12].

Tabla 4. Características de la tarjeta secundaria transceptora RFX2400

Características Rfx2400	
Banda de operación	2,4 GHz
Rango de operación	2,3 – 2,9 GHz
Antenas de trasmisión	Tx/Rx
Antena de recepción	Tx/Rx o Rx2
Ancho de banda Rx	40 MHz
Ancho de banda Tx	40 MHz
Potencia de transmisión	50 mW (17 dBm)
Potencia de recepción	-10 dBm
Figura de ruido	8 dB

Datos tomados de las especificaciones técnicas de la tarjeta secundaria RFX2400.

Entre las áreas de aplicación más comunes se encuentran el desarrollo de transceptores en la banda ISM de 2,4 GHz y desarrollo de redes inalámbricas. Su principal desventaja radica en el limitado rango de frecuencias de operación (2,3 – 2,9 GHz), por esto es recomendable usar otras tarjetas secundarias que posean un rango más amplio de manera que se pueda incursionar en aplicaciones en diferentes bandas, como por ejemplo las tarjetas transceptoras WBX, SBX o CBX que poseen un rango de frecuencias de 50 MHz – 2,2 GHz, 400 MHz – 4,4 GHz y 1,2 - 6GHz respectivamente [13].

3.3 DISPOSITIVOS UTILIZADOS EN EL PROYECTO

Conociendo las características generales del dispositivo USRP1 y la tarjeta secundaria RFX2400, se procede a realizar un inventario de los dispositivos utilizados en este proyecto, como se muestra en la Tabla 5.

Tabla 5. Listado de elementos utilizados

Elementos
USRP1 (serial = 4d05be08)
USRP1 (serial = 4cc49b63)
Fuentes de alimentación (6V DC, 3A)
(4) Tarjetas secundarias RFX2400
(2) Antenas VERT2450
Atenuador 30dB SMA-F a SMA-M
Cable SMA-M a SMA-M
Analizador vectorial de redes ZVL6 de Rohde & Schwarz

El serial o información de las propiedades del dispositivo USPR1 conectado a un PC puede ser obtenida mediante línea de comando (ver Anexo D).

La carcasa del USRP1 cuenta con cuatro terminales nombrados desde RF1 a RF4. Estos puertos de antenas pueden ser usados como Tx o Rx dependiendo de la tarjeta secundaria que se conecte al USRP1 y también de la conexión que se haga desde los puertos de las tarjetas secundarias a los puertos de la carcasa del USRP1. La Tabla 6 muestra cuáles de los puertos de las carcasas de los dos dispositivos USRP1 trabajados en este proyecto son Tx o Rx, diferenciando cada uno con su número de serial.

En la Tabla 6 se observa que el dispositivo USRP1 con serial *4cc49b63* no posee identificación como Tx o Rx en dos de sus terminales, debido a que una de sus tarjetas secundarias no se encuentra conectada a la carcasa del dispositivo. Esta conexión es posible realizarla mediante un cable MCX-M a SMA-F.

Tabla 6. Puertos de los USRP1 trabajados

USRP1			
SERIAL		**4d05be08**	
Puerto	Antena	Ranura	Tarjeta secundaria
RF1	Tx/Rx	B	RFX2400
RF2	Rx2		
RF3	Tx/Rx	A	RFX2400
RF4	Rx2		

Tabla 6. (Continuación)

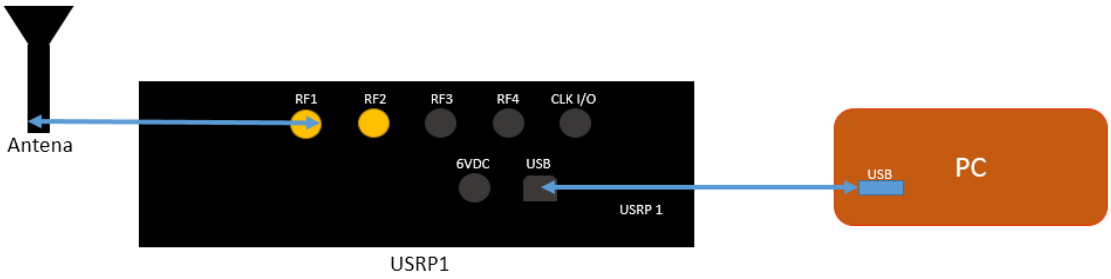
USRP1			
SERIAL		4cc49b63	
Puerto	Antena	Ranura	
RF1	Tx/Rx	B	RFX2400
RF2	Rx2		
RF3	No conectado	A	RFX2400
RF4	No conectado		

3.4 ESQUEMAS DE CONEXIÓN

Los esquemas de conexión presentados a continuación se basan en los elementos disponibles para el desarrollo de este proyecto. Las conexiones de los terminales de antenas del dispositivo USRP corresponden a los expuestos en la sección 3.3.

- **Conexión USRP – PC.** Usado para Tx o Rx según configuración del usuario (Figura 2).

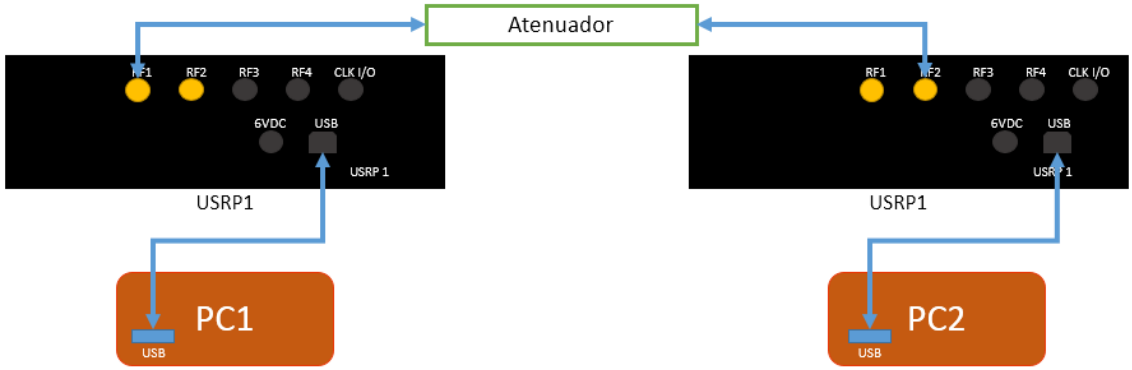
Figura 2. Conexión USRP- PC.



Ejemplo 1. Usando una tarjeta secundaria que posea un rango de operación compatible con la banda de frecuencia de radio FM (p. ej. WBX 50 MHz – 2,2 GHz), se puede utilizar la conexión USRP – PC para crear un radio de recepción FM, configurando el USRP como receptor (ver referencia [4]).

- **Conexión PC – USRP – USRP – PC.** Utilizada para recepción y transmisión por medio de un atenuador (Figura 3). Se debe tener especial cuidado con el atenuador a utilizar, teniendo en cuenta las capacidades máximas de potencia de recepción de las tarjetas secundarias conectadas a los USRP. Este nivel de potencia es de $-10\,\mathrm{dBm}$ para casi todas las tarjetas de Ettus Research exceptuando la BasicRX y LFRX. Se recomienda utilizar un **atenuador de 30 dB** para las conexiones [14].

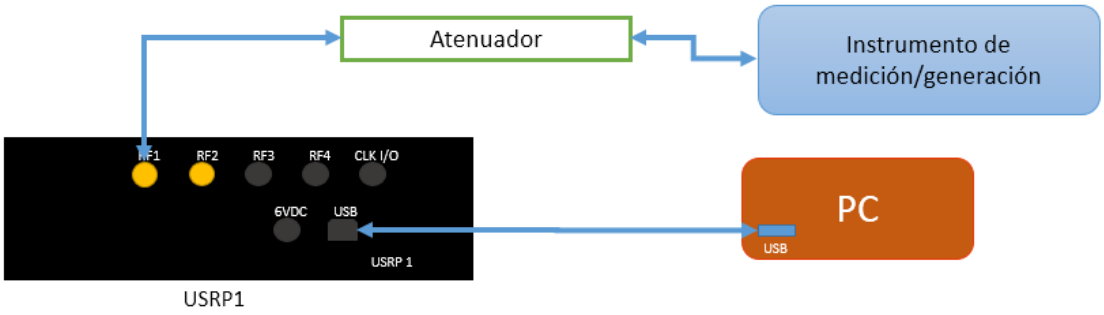
Figura 3. Conexión PC – USRP – USRP – PC



Ejemplo 2. Pruebas de funcionamiento de transmisión y recepción en los sistemas propuestos en este proyecto.

- **Conexión PC – USRP1 – INSTRUMENTO DE MEDICIÓN/GENERACIÓN.** Se puede utilizar para generar señales por medio del PC y transmitirlas a un equipo que pueda visualizarlas, así como también recibir señales transmitidas por un equipo generador de señal (Figura 4). Se debe tener especial cuidado con el atenuador a utilizar, verificando los niveles de potencia máxima permitidos por los equipos utilizados para recibir señales, ya sea el USRP1 o el instrumento de medición.

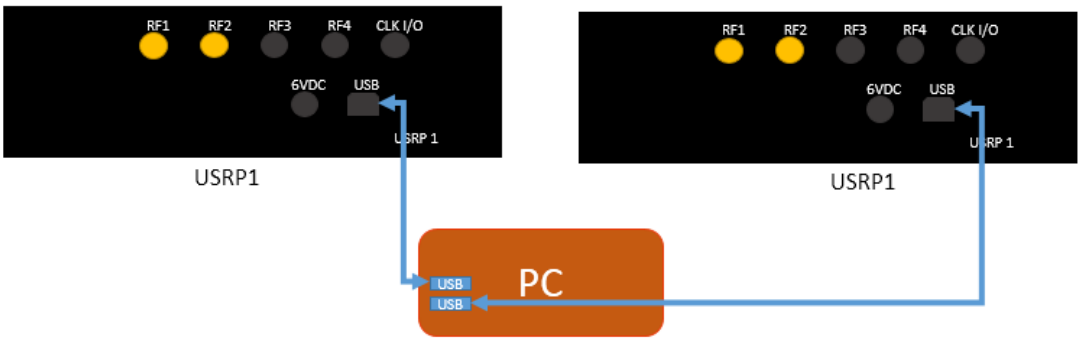
Figura 4. Conexión PC – USRP1 – INSTRUMENTO DE MEDICIÓN / GENERACIÓN



Ejemplo 3. Analizar espectros de frecuencias de una señal proveniente del dispositivo USRP.

- **Conexión de dos dispositivos USRP a un PC.** Es utilizada para realizar pruebas entre los dispositivos USRP (Figura 5). Esta conexión se realiza teniendo los seriales o dirección de cada dispositivo USRP conectado a la PC (ver anexo B).

Figura 5. Conexión de dos dispositivos USRP a un PC

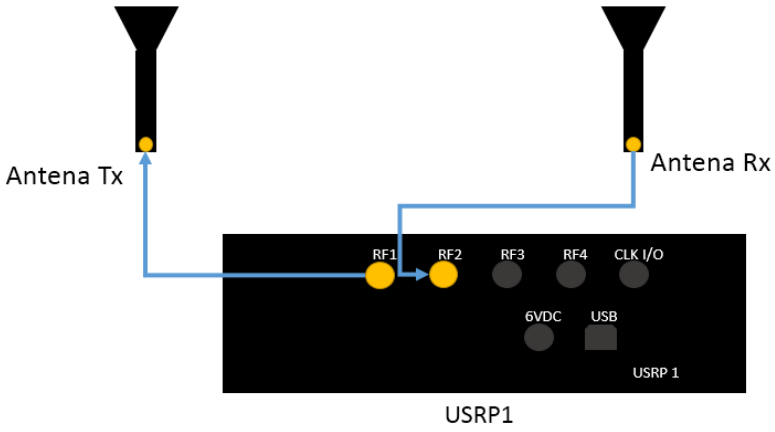


Ejemplo 4. Pruebas de funcionamiento de transmisión y recepción en los sistemas propuestos en este proyecto.

- **Conexión full-dúplex.** Esta conexión se utiliza cuando se requiere que una aplicación desarrollada reciba y transmita al mismo tiempo (Figura 6). El USRP debe contar con una tarjeta secundaria transceptora que tenga la capacidad full-

dúplex. Para más información de la operación full-dúplex de las tarjetas transceptoras véase [12].

Figura 6. Conexión full-dúplex



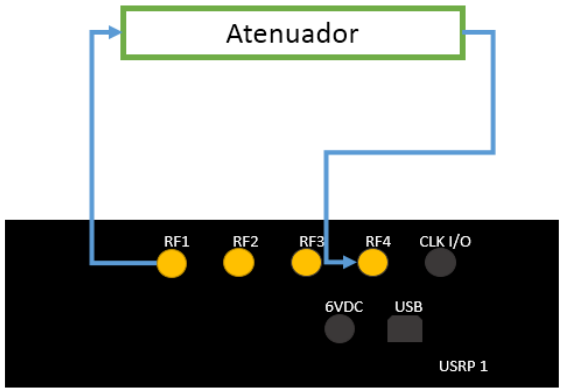
Ejemplo 5. Implementación de un repetidor de radio [7].

- **Conexión de USRP realimentada.** Esta conexión se utiliza cuando se requieren pruebas con un solo dispositivo USRP (Figura 7), siendo esto posible gracias a que el dispositivo USRP1 posee dos ranuras para tarjetas secundarias, con la desventaja que se reduce la tasa de transmisión de datos entre el USRP y el PC por la interfaz USB (ver sección 3.1).

Se debe tener especial cuidado con el atenuador a utilizar, teniendo en cuenta las capacidades máximas de potencia de recepción de las tarjetas secundarias conectadas a los USRP. Este nivel de potencia es de $-10\ dBm$ para casi todas las tarjetas de Ettus Research exceptuando la BasicRX y LFRX. Se recomienda utilizar un **atenuador de 30 dB** para las conexiones [14].

Esta conexión consiste en utilizar las dos ranuras para tarjetas secundarias del USRP1, por lo que también puede usarse en conexión MIMO 2X2 teniendo dos puntos de enlace independientes.

Figura 7. Conexión de USRP realimentada



USRP1

Ejemplo 6. Pruebas de funcionamiento de transmisión y recepción en los sistemas propuestos en este proyecto.

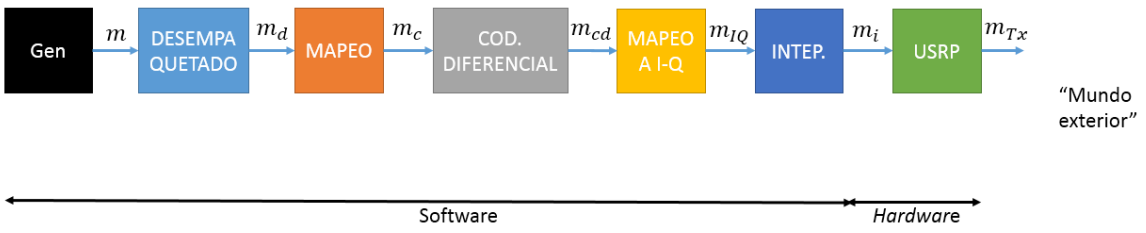
4. DESARROLLO DEL SISTEMA DE COMUNICACIÓN DIGITAL

El diseño del sistema de modulación digital propuesto a continuación está basado en el documento *generic_mod_demod.py*[3], el cual permite construir un modulador y demodulador genérico en lenguaje Python que puede utilizarse para cualquier tipo de modulación, definida por una variable que contiene la información de la distribución de los puntos en la constelación deseada.

Algunos bloques como *constellation_modulator, PSK_Mod, PSK_Demod, QAM_Mod, QAM_Demod, DPSK_Mod, DPSK_Demod* en GRC realizan un llamado de la función *generic_mod_demod.py*. Estos bloques antes mencionados de modulación o demodulación (exceptuando *constellation_modulator*), permiten implementar un modulador completo, en donde cada uno de ellos ya tiene definido un tipo de modulación, es decir, que en su interior ya se ha definido la variable que contiene la información de la constelación. Sin embargo, en el presente proyecto no se utilizan los bloques mencionados anteriormente pues éstos no permiten observar características de los datos mientras se procesan.

La Figura 8 y Figura 9 muestran los esquemas de transmisión y recepción propuestos para ser implementados mediante tecnología SDR junto con la herramienta GRC del software GNU Radio y los dispositivos USRP. La Tabla 7 y la Tabla 8 describen de forma breve cada componente del esquema de transmisión y recepción respectivamente (explicados detalladamente en los anexos A3 y A4) y las señales presentes en cada esquema.

Figura 8. Esquema de transmisión



[3] Este documento hace parte de las librerías de GNU Radio, se encuentra disponible en las carpetas de instalación del mismo.

Tabla 7. Descripción de componentes y señales del esquema de transmisión

Componentes del esquema de transmisión		
Componente	**Descripción**	**Referencia**
GEN	Generador de mensaje.	Anexo A2.1
DESEMPAQUETADO	Convierte de un flujo empaquetado a un flujo desempaquetado.	Anexo A3
MAPEO	Mapea símbolos en banda base a la codificación pre-diferencial-	Anexo A3
COD. DIFERENCIAL	Codificación diferencial de los símbolos.	Anexo A3
MAPEO A I-Q	Convierte símbolos a muestras complejas.	Anexo A3
INTERP.	Realiza la interpolación a muestras/símbolo y conforma el pulso.	Anexo A3
USRP	Medio para exportar las señales al “mundo exterior”.	Anexo A2.3
Señales entre los componentes del esquema de transmisión		
m	Señal mensaje	
m_d	Mensaje desempaquetado	
m_c	Mensaje codificado	
m_{cd}	Mensaje codificado diferencialmente	
m_{IQ}	Mensaje en I-Q	
m_i	Mensaje adecuado al canal	
m_{Tx}	Mensaje transmitido	

Figura 9. Esquema de recepción

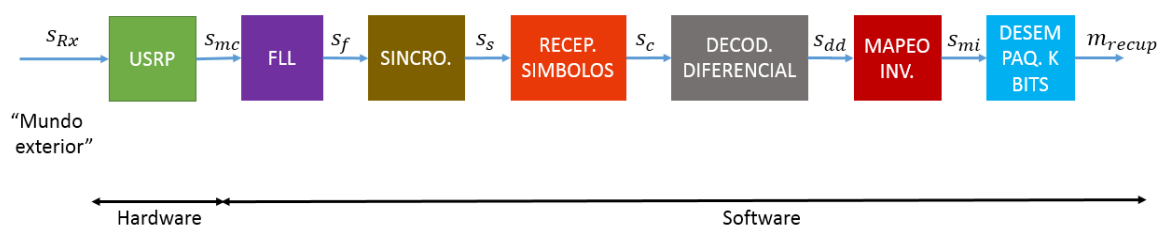


Tabla 8. Descripción de componentes y señales del esquema de transmisión

Componentes del esquema de recepción		
componente	**Descripción**	**Referencia**
USRP	Medio para importar las señales del "mundo exterior".	Anexo A2.3
FLL	Realiza compensación en frecuencia.	Anexo A4
SINCRO	Filtro adaptativo y sincronización.	Anexo A4
RECEP. SÍMBOLOS	Ajuste fino en sincronización de fase y toma de decisiones para recibir las muestras complejas y convertirlas a símbolos.	Anexo A4
DECOD. DIFERENCIAL	Decodificación diferencial.	Anexo A4
MAPEO INVERSO	Mapea a símbolos pre-diferencial.	Anexo A4
DESEMPAQ. k BITS	Desempaqueta k bits/símbolo a un flujo de bits.	Anexo A4
Señales entre los componentes del modulador digital genérico		
s_{Rx}	Señal recibida por el USRP	
s_{mc}	Señal de muestras complejas proporcionadas por el USRP	
s_f	Señal aplicado la compensación en frecuencia	
s_s	Señal sincronizada	
s_c	Señal codificada	
s_{dd}	Señal decodificada diferencialmente	
s_{mi}	Señal mapeada inversamente (decodificada)	
m_{recup}	Mensaje recuperado	

A continuación se presenta una explicación de algunos módulos y funciones necesarias para el posterior desarrollo del sistema de comunicación digital (2-4-8-PSK y 16-QAM). Comienza con una clasificación de las variables que pueden ser modificadas en tiempo de ejecución del flujograma creado en GRC, seguido de una explican los módulos que se utilizan para crear las constelaciones que definen las modulaciones digitales trabajadas en este proyecto, así como también los bloques que hacen parte del modulador y demodulador general. Por último se muestra el diseño realizado del sistema de modulación (modulador y demodulador 16-QAM) en GRC.

4.1 PARÁMETROS USADOS POR UN MODULADOR Y UN DEMODULADOR EN GRC

En la Tabla 9 se presenta una clasificación de las variables que se seleccionaron como *modificables* y *no modificables*. Variables modificables hace referencia a aquellas cuyo valor se puede alterar en tiempo de ejecución. Por otro lado, las variables clasificadas como no modificables son aquellas que no aceptan cambios mientras se está ejecutando el flujograma. Cualquier modificación que se intente hacer en éstas, se verá reflejado sólo cuando el flujograma se vuelva a ejecutar.

Tabla 9. Librerías a importar y variables modificables y no modificables.

Librerías a importar (bloque *import*)		
import math	Librería de herramientas matemáticas.	
from gnuradio import digital	Librería de herramientas de procesado digital de señal.	
Variables no modificables		
Nombre	**Descripción**	**Valor**
tipo_modulacion	En él se encuentran presentes las opciones de las modulaciones PSK o QAM que se pueden realizar. Se puede crear como una variable de valor único, o como una variable de selección entre diferentes opciones (usando un bloque *GUI Chooser*).	Para PSK se utilizan 2, 4 y 8. Para QAM se utiliza 16.
samp_rate	Se crea por defecto al crear un nuevo flujo-grama.	..
bits	Indica el número de bits necesarios para representar los símbolos. El parámetro *bits* debe ser un valor entero, ya que algunos parámetros de variables o bloques que hacen uso de ésta lo indican.	int(math.log(tipo_modulacion,2))
muestras	Indica el número de muestras por símbolo.	bits
simbolos	Indica el número de símbolos usados en la modulación.	pow(2,bits)

Tabla 9. (Continuación)

Variables no modificables		
Nombre	**Descripción**	**Valor**
cod_gray	Es utilizada para generar el código gray según el número de símbolos que tenga la constelación. Es posible usar una codificación diferente, sólo es necesario ingresar un vector con la codificación (lenguaje python), respetando el orden de la constelación.	digital.utils.gray_code.gray_code(simbolos)
cod_gray_inv	Genera un código gray inverso partiendo del código gray usado en el modulador.	digital.mod_codes.invert_code(cod_gray)
constelacion	Crea un objeto de constelación, en el que se definen lista de atributos y funciones útiles para la manipulación de las constelaciones.	Explicación detalla de los módulos utilizados para definir este parámetro en sección 4.2.
nfiltros	Indica el número de filtros a usar en los bloques GRC que utilizan bancos de filtros.	32 (valor sugerido)
rolloff	Valor del factor de *rolloff* (exceso de ancho de banda)	0.35 (valor sugerido)
ntaps	Indica el número de taps a utilizar en el filtro.	11*nfiltros*muestras
taps_PAR	Genera el filtro coseno alzado necesario en el bloque Polyphase Arbitrary Resampler.	firdes.root_raised_cosine(nfiltros,nfiltros,1.0,rolloff,ntaps)
taps_PCS	Genera el filtro coseno alzado necesario en el bloque Polyphase clock sync.	firdes.root_raised_cosine(nfiltros,nfiltros*muestras,1.0,rolloff,ntaps)
Variables modificables		
Variables creadas a partir de parámetros como por ejemplo amplitud, frecuencia, nivel de *offset* en bloques de generación de señal, frecuencia central (bloque de UHD: *usrp sink* o *source*), entre otros.		

4.2 DISEÑO DE LAS CONSTELACIONES

El diseño de las constelaciones es un punto importante para el desarrollo de este proyecto debido a que cada modulación digital está determinada por una constelación particular, es decir, en un mismo flujograma de un modulador digital, se pueden obtener diferentes tipos de modulaciones sólo cambiando la variable que contiene la información de la constelación.

Para trabajar con modulaciones digitales, GNU Radio utiliza clases[4] de objetos de constelación en los que se definen una lista de atributos y funciones para la manipulación de las constelaciones, con la posibilidad de luego utilizar esos atributos por cualquier bloque que requiera la información. Hay una jerarquía de clases para diferentes propósitos y cada una representa una clase especial de constelación [15].

Una de las más importantes funciones utilizadas en las clases de constelación es *gr::digital::constellation::decision_maker*, usada en el proceso de demodulación. Ésta toma un punto en el espacio complejo y devuelve el símbolo mapeado en él, siendo generalmente este proceso el que diferencia una clase de constelación de otra [15].

Los atributos en los objetos de constelación creados se pueden obtener mediante la función `nombre_variable_constelacion.atributo()`. Por ejemplo, para el caso del modulador se utiliza `constelacion.points()` el cual devuelve el conjunto de puntos en la constelación, y `constelacion.arity()` que devuelve un número entero que indica el número de niveles de la modulación (Ver *constellation* en [15]). Esto se aplica a cualquier modulo o función que cree un objeto de constelación.

En el presente proyecto se utilizan tres módulos para crear las constelaciones en los diseños realizados.

4.2.1 **Módulo *digital.psk_constellation.*** Este módulo define una modulación PSK de orden `M`, el cual crea y devuelve un objeto de constelación.

[4] En programación orientada a objetos, una clase es una construcción que define los datos y el comportamiento de un tipo.

```
psk_constellation(M, mod_code, differential)
```

donde:

- `M`: orden de la modulación. Este valor deber ser potencia de 2.
- `mod_code`: define el uso (`mod_codes.GRAY_CODE`) o no (`mod_codes.NO_CODE`) de codificación gray para el mapeado de los símbolos.
- `differential`: permite seleccionar el uso o no de codificación diferencial (*True* o *False* respectivamente).

4.2.2 **Módulo *digital.qam_constellation.*** Este módulo es una de las herramientas proporcionadas por GNU Radio para trabajar con modulaciones QAM digitales. Éste define una modulación QAM de orden `M`, creando con esto un objeto de constelación.

```
qam_constellation(M, differential, mod_code,
     large_ampls_to_corners)
```

donde:

- `M`: indica el orden de la modulación. Este valor debe ser potencia de 4, ya que los puntos de la constelación se distribuyen en forma cuadrada en cada cuadrante, y con simetría respecto a los dos ejes.
- `differential`: permite seleccionar el uso o no de la codificación diferencial (*True* o *False* respectivamente).
- `mod_code`: selecciona si se usa o no codificación gray en la constelación. Se utiliza *none* (valor por defecto) para indicar que no se usa, y *gray* para lo contrario.
- `large_ampls_to_corners`: si es puesto en *True,* cuando la constelación realice el proceso de toma de decisiones, los puntos que estén lejos (por fuera) de la constelación son mapeados al punto de constelación situado en la esquina más cercana en lugar del punto más cercano. *False* viene como valor

por defecto [15]. El funcionamiento de este parámetro se usa en el demodulador.

4.2.3 **Módulo *digital.constellation_calcdist.*** Se puede crear un objeto de constelación normalizado mediante:

```
objeto_constelacion = digital.constellation_calcdist(p,
    pre_diff_code, rotational_symmetry, dimensionality),
```

donde:

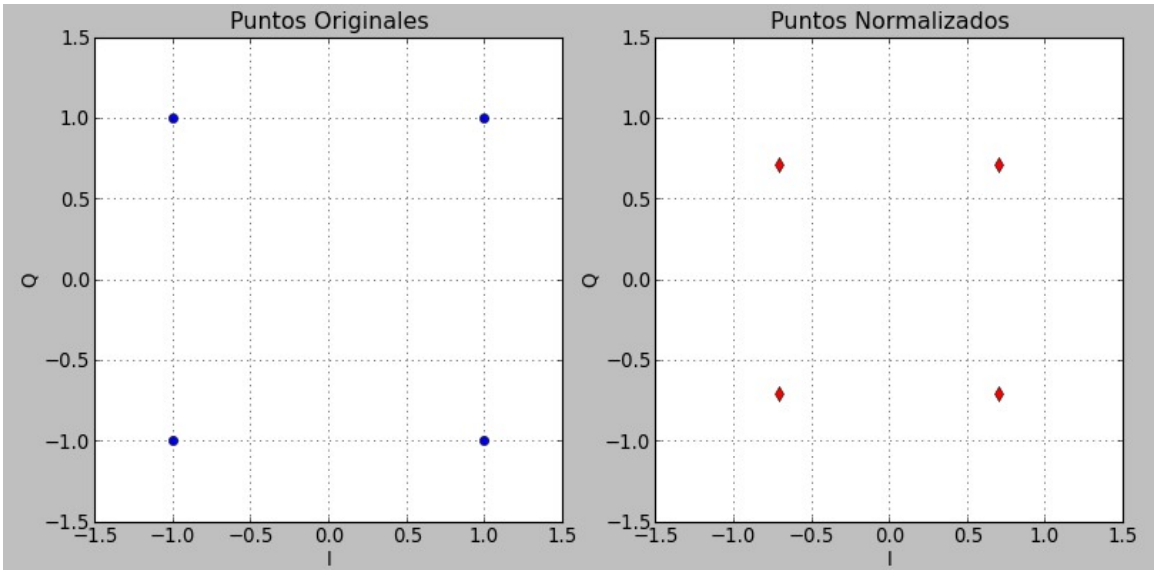
- $\boldsymbol{p} = [p_1, \ldots, p_m]$ es un vector de m elementos complejos, correspondientes a la lista de puntos de la constelación. El orden de esta lista debe coincidir con el orden de la lista de `pre_diff_code`.
- `pre_diff_code`*:* lista del alfabeto de símbolos antes de aplicar codificación diferencial. El orden de esta lista debe coincidir con el orden de la lista de $\boldsymbol{p}$.
- `rotational_symmetry`*:* indica el número de rotaciones a lo largo de la circunferencia unidad que tienen la misma representación.
- `dimensionality`*:* número de dimensiones de la constelación.

Del módulo *digital.constellation.calcdist*, se obtienen los puntos normalizados ($\widehat{\boldsymbol{p}}$) mediante `constelacion.points()`, correspondiendo éstos a:

$$\widehat{\boldsymbol{p}} = \frac{\boldsymbol{p}}{\sum_{i=1}^{m} |p_i|/m}$$

A modo de ejemplo, en la Figura 10 se muestran los diagramas de constelación de elementos contenidos en $\boldsymbol{p} = [(\mathbf{1}+\mathbf{1}\boldsymbol{j}), (-\mathbf{1}+\mathbf{1}\boldsymbol{j}), (-\mathbf{1}-\mathbf{1}\boldsymbol{j}), (\mathbf{1}-\mathbf{1}\boldsymbol{j})]$ (Puntos Originales) y de su normalización en $\widehat{\boldsymbol{p}} = [(\mathbf{1}+\mathbf{1}\boldsymbol{j}), (-\mathbf{1}+\mathbf{1}\boldsymbol{j}), (-\mathbf{1}-\mathbf{1}\boldsymbol{j}), (\mathbf{1}-\mathbf{1}\boldsymbol{j})] * \left(1/\sqrt{2}\right)$ (Puntos Normalizados).

Figura 10. Puntos de constelación ($\boldsymbol{p}$); puntos de constelación normalizados ($\hat{\boldsymbol{p}}$)



En la demodulación, la función `decision_maker(x)` en el módulo *digital.constellation.calcdist* toma una muestra compleja `x` y calcula la distancia euclidiana entre `x` y cada punto ($\hat{\boldsymbol{p}}$) en el mapa del objeto de constelación. El punto que tiene la distancia euclidiana mínima para `x` se selecciona como la mejor coincidencia. De esta forma, `decision_maker(x)` devolve entonces el valor del símbolo que coincide con este punto de la constelación seleccionada [15].

La principal ventaja de este comando radica en que se puede ingresar los puntos de constelación diseñados y la forma de representación de estos, de manera que se pueden crear modulaciones con constelaciones que no están contempladas en las prediseñadas en el software GNU Radio, pero no es recomendable para grandes constelaciones, pues se vuelve un proceso ineficiente.

4.3 DISEÑO MODULADOR – DEMODULADOR

La modulación y demodulación digital son procesos complejos que conllevan el uso de una serie de componentes (bloques de GNU Radio en este caso) para su realización. GNU Radio combina los bloques correspondientes a moduladores y demoduladores genéricos en un archivo denominado *generic_mod_demod.py* de manera que se facilite su acceso y uso [15].

Los moduladores o demoduladores genéricos son bloques que realizan una modulación según los puntos de la constelación y otros parámetros que definen sus atributos. Si bien no se usan los módulos (o bloques en GRC) para crear un modulador o demodulador genérico, a manera de referencia se muestran las sentencias en Python que permiten implementarlos:

- `digital.generic_mod(constellation, differential, samples_per_symbol, pre_diff_code, excess_bw, verbose, log)`

- `digital.generic_demod(constellation, differential, samples_per_symbol, pre_diff_code, excess_bw, freq_bw, timing_bw, phase_bw, verbose, log)`

Si se quisiera reproducir un modulador genérico partiendo de los bloques que lo conforman, éste constaría de:

- `blocks.packed_to_unpacked_bb`*:* convierte de un flujo empaquetado a uno desempaquetado.

- `digital.map_bb`*:* mapea símbolos en banda base a la codificación pre-diferencial.

- `digital.diff_encoder_bb`*:* codificación diferencial de los símbolos.

- `digital.chunks_to_symbols_bc`*:* convierte los símbolos a muestras complejas.

- `filter.pfb_arb_resampler_ccf`*:* realiza la interpolación a muestras/símbolo y conforma el pulso.

El demodulador genérico tiene los siguientes componentes:

- `digital.fll_band_edge_cc`*:* realiza compensación en frecuencia.

- `digital.pfb_clock_sync_ccf`*:* filtro adaptativo y recuperación de tiempo.

- `digital.constellation_receiver_cb`: ajuste fino en la sincronización de fase y “decisiones duras” (*hard decisions*) para recibir los símbolos.

- `digital.diff_decoder_bb`: decodificación diferencial.

- `digital.map_bb`: mapea a símbolos pre-diferencial.

- `blocks.unpack_k_bits_bb`: desempaqueta k bits/símbolo a un flujo de bits

Una desventaja de utilizar los bloques de modulación genéricos es la imposibilidad de visualizar o hacer seguimiento a la señal mientras se procesa; esto es, sólo se puede observar el flujo de datos que entra y sale del bloque. Con el fin de hacer un seguimiento a la señal, se opta por realizar la modulación a partir de los bloques que conforman al modulador y demodulador genéricos.

4.4 DISEÑO EN GNU RADIO COMPANION (GRC)

Anteriormente se ha realizado han descrito los módulos y bloques necesarios para realizar una modulación digital en la que se puede cambiar el tipo de ésta con sólo cambiar la definición de la constelación que se usa. A continuación se presenta un ejemplo de la construcción del modulador y demodulador trabajado en este proyecto (específicamente modulador y demodulador 16-QAM) en la herramienta gráfica GNU Radio Companion (GRC) del software GNU Radio.

4.4.1 **Modulador en GRC.** En la Figura 11 se observa el modulador QAM (en este caso 16-QAM) junto con una fuente de vector y un sumidero virtual. El sumidero virtual en este caso simula el funcionamiento del bloque *UHD: USRP Sink* con el que se estaría enviando la señal al “mundo exterior”.

Las variables necesarias para el modulador 16-QAM se presentan en la Tabla 10, y la descripción de estas variables se encuentra en la Tabla 9 de la sección 4.1.

Figura 11. Modulador 16QAM construido en GRC

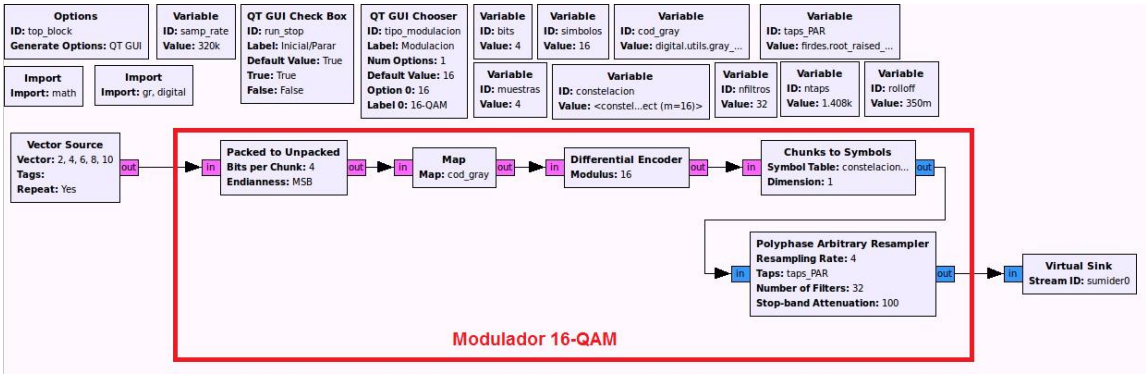


Tabla 10. Variables usadas por el modulador QAM

Variables
sample_rate
tipo_modulacion
bits
muestras
simbolos
cod_gray
constelacion
nfiltros
rolloff
ntaps
taps_PAR

Se usa el módulo *digital.qam_constellation* para definir los objetos de constelación en la variable *constelacion* en el modulador/demodulador QAM. Se construye de la siguiente forma:

```
digital.qam_constellation(simbolos, False)
```

Note que en este caso no se utilizan todos los parámetros utilizados por el módulo *digital.qam_constellation* descritos en la sección 4.2.2. Lo mencionado anteriormente es debido a que el programa (GNU Radio), cuando no se especifican parámetros de una función o bloque, éstos toman valores por defecto detallados en los códigos de programación de esas funciones o bloques.

La variable *constelacion* puede tomar los módulos *digital.psk_constellation* o *digital.constellation_calcdist* para especificar el uso de una modulación PSK o usar una modulación diseñada ingresando el vector de puntos de constelación respectivamente (ver sección 4.2). A continuación, en la Tabla 11 se presenta la manera en la que se escriben éstos módulos.

Tabla 11. Módulos para asignar a la variable *constelación*

Módulo	Parámetros
`digital.psk_constellation`	`(simbolos,no_code,False)`
`digital.constellation_calcdist`	`(puntos,[],1,1)`

El módulo *digital.psk_constellation* crea objetos de constelación que define una modulación del orden del valor numérico de símbolos (valor entero). `no_code` es una variable creada únicamente para el uso de éste asignada con `mod_codes.NO_CODE` (Ver capítulo 5 para ver el uso de éstos en simulación).

Algunos de los valores que toman los parámetros de los bloques en el modulador se basan en los valores por defecto presentes en el archivo g*eneric_mod_demod.py,* éstos son resumidos en la Tabla 12.

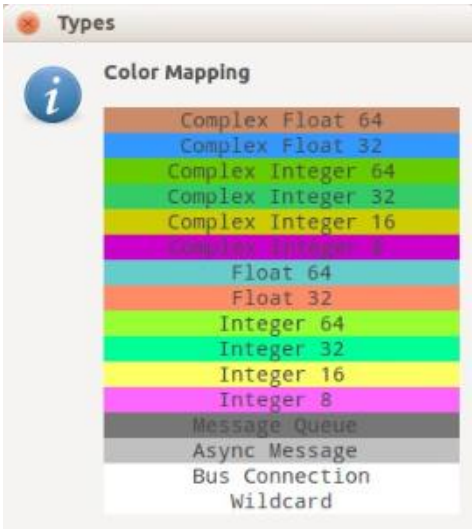
Tabla 12. Bloques en el modulador y sus parámetros

Bloque	Parámetros	Valor
Packed to unpacked	Bits per chunk:	bits
	Endianness:	MSB
Map	Map	cod_gray
Differential encoder	Modulus	Constelacion.arity()
Chunks to symbols	Symbol table	Constelacion.points()
	D	1
Polyphase arbitrary resampler	Rate	muestras
	Taps	Taps_PAR
	Filter size	nfiltros

Tener especial cuidado con el formato de los datos. Al modulador entran datos tipo *byte*, y sigue con éste hasta que se realiza el mapeo de los símbolos complejos (puntos de constelación), en donde cambia a tipo *complejo* (ver Figura 12). Si se utiliza una fuente generadora de ondas u otra similar para generar los datos a

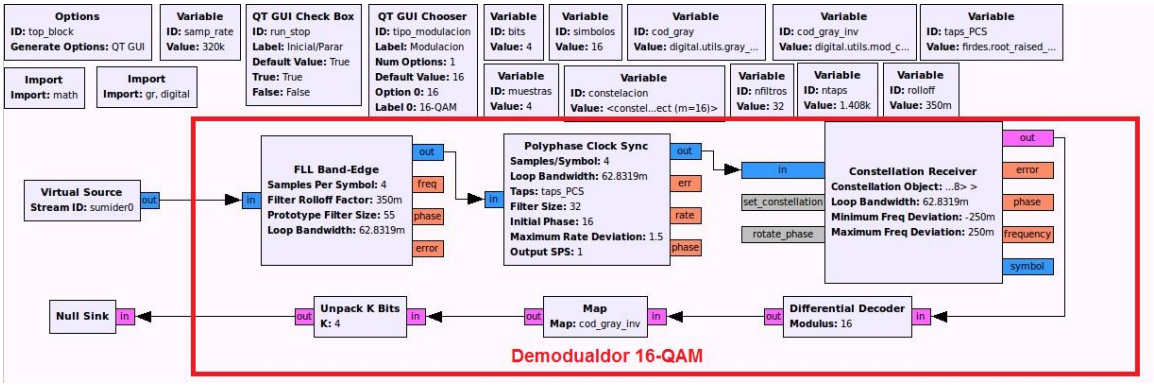
modular y transmitir, usar el bloque *Packet encoder* o cambiar el tipo de dato a byte.

Figura 12. Color asignado a los formatos de datos usados por GNU Radio Companion



4.4.2 **Demodulador en GRC.** En la Figura 13 se presenta el demodulador QAM (en este caso 16QAM) junto con una fuente virtual y un sumidero nulo. La fuente virtual simula el funcionamiento del bloque *UHD: USRP Source* con el que se estaría recibiendo la señal desde el "mundo exterior".

Figura 13. Demodulador 16QAM construido en GRC



Las variables necesarias para el demodulador QAM se presentan en la Tabla 13, y la descripción de estas variables se encuentra en la Tabla 9 de la sección 4.1. En su mayoría las variables utilizadas en el demodulador y el modulador son iguales, exceptuando una variable adicional que contiene el código gray inverso (*cod_gray_inv*) utilizada en la etapa de decodificación de los datos y *taps_PCS* que reemplaza a *taps_PAR*.

Tabla 13. Variables usadas por el demodulador QAM

Variables
sample_rate
tipo_modulacion
bits
muestras
simbolos
cod_gray
cod_gray_inv
constelacion
nfiltros
rolloff
ntaps
taps_PCS

Al igual que en el modulador, algunos de los valores de los parámetros usados por los bloques que hacen parte del modulador están basados en los valores presentes (valores por defecto) en el archivo *generic_mod_demod.py.* En la Tabla 14 se muestran los valores que toman los parámetros de los bloques usados en el demodulador.

Tabla 14. Bloques en el demodulador y sus parámetros

Bloque	Parámetros	Valor
Agc2	Attack rate	$0.6e^{-1}$ (60m)
	Decay rate	$1e^{-3}$ (1m)
	Reference	1
	Gain	1
	Max gain	65536
FLL Band Edge	Samples per symbol	Muestras
	Filter rolloff factor	Rolloff
	Prototype filter size	55
	Loop bandwidth	2*math.pi/100.0

Tabla 14. (Continuación)

Bloque	Parámetros	Valor
Polyphase clock sync	Samples/symbol	Muestras
	Loop bandwidth	2*math.pi/100.0
	Taps	Taps_PCS
	Filter size	Nfiltros
	Initial phase	Nfiltros//2
	Maximum rate	1.5
	Output sps	1
Constellation receiver	Constellation object	constelacion.base()
	Loop bandwidth	2*math.pi/100.0
	Minimum freq. deviation	-0.25
	Maximum freq. deviation	0.25
Differential decoder	Modulus	constelacion.arity()
Map	Map	cod_gray_inv
Unpack k bits	K	bits

5. SIMULACIONES

En este capítulo se presentan las simulaciones correspondientes a las modulaciones digitales 16-QAM (Figura 14), 8-QAM (Figura 15) y BPSK (Figura 16), con el fin de mostrar ejemplos del uso de los tres módulos para crear la constelación (ver sección 4.2) que define la modulación digital a utilizar.

Como se menciona en la sección 4.4.1, para la modulación 16-QAM (Figura 14) se utiliza el módulo *digital.qam_constellation* y en la modulación BPSK (Figura 16) el módulo *digital.psk_constellation* para definir la variable constelación. Una modulación 8-QAM no es posible realizarla con el módulo *digital.qam_constellation* debido a que éste sólo crea constelaciones con un número de niveles que sea potencia de 4, y es por esta razón que se aprovecha la particularidad del módulo *digital.constellation_calcdist* (ver sección 4.2.3) para construir el modulador y demodulador 8-QAM (Figura 15).

De manera general, en los flujogramas (Figura 14, Figura 15 y Figura 16) se muestra un transmisor, un receptor y un bloque de modelado de canal, junto con las variables que en su mayoría son iguales para los tres diagramas. El transmisor cuenta con una fuente tipo vector (generador de mensaje) y el modulador (sección 4.4.1), el receptor está compuesto por el demodulador (sección 4.4.2). Por último, el modelado de canal es realizado por el bloque *Channel model* el cual es agregado con el propósito de introducir ruido a la señal que se “transmite”.

Además de la funcionalidad antes mencionada, el bloque *Channel model* permite modelar desplazamiento en frecuencia y tiempo, y multi-trayectorias [15].

La frecuencia de transmisión de los datos (*bytes* o *binarios*) entre bloques no es controlada debido a que en el software GNU Radio, no se controla el tiempo de transmisión de los datos; sólo es posible controlar el tiempo o frecuencia de transmisión en el bloque *USRP* (sumidero o fuente) cuando se exportan o importan señales desde el PC o el “mundo exterior” respectivamente, para una transmisión o recepción real. Por lo anterior, se observa en Figura 14, Figura 15 y Figura 16 que ninguno de los bloques posee como parámetro la variable *samp_rate*.

Figura 14. Diagrama de flujo modulación digital 16-QAM

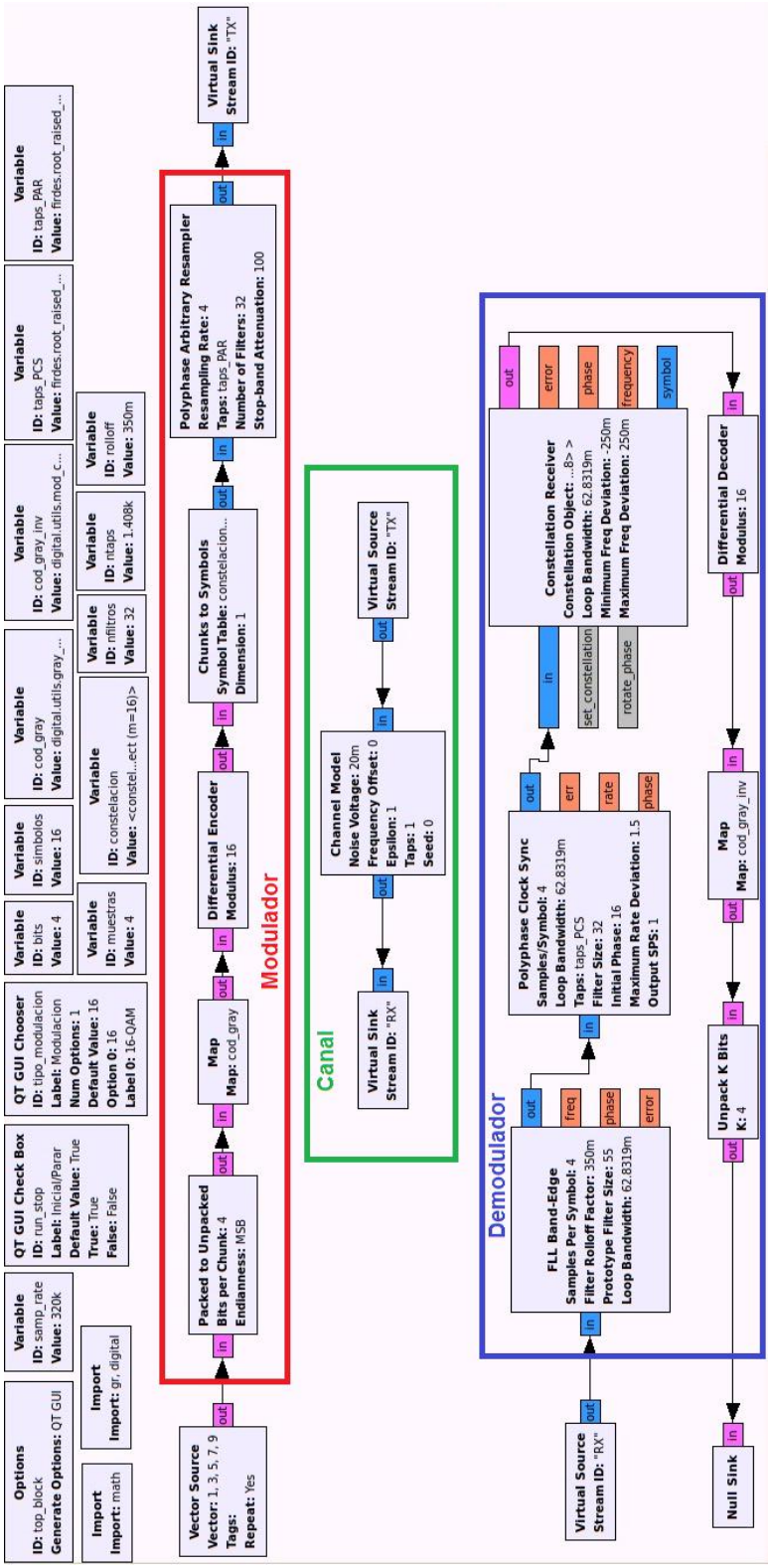


Figura 15. Diagrama de flujo modulación digital 8-QAM

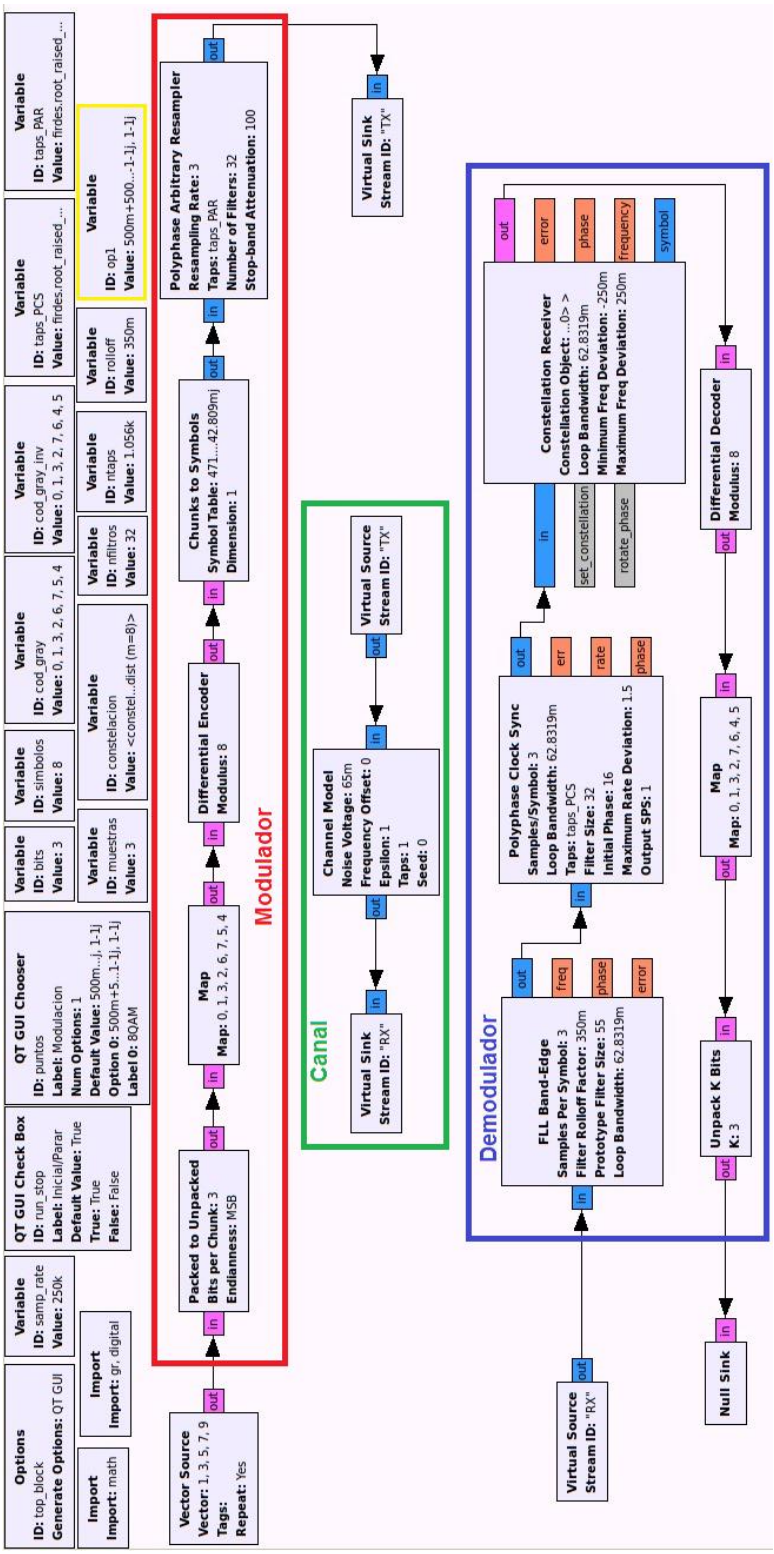
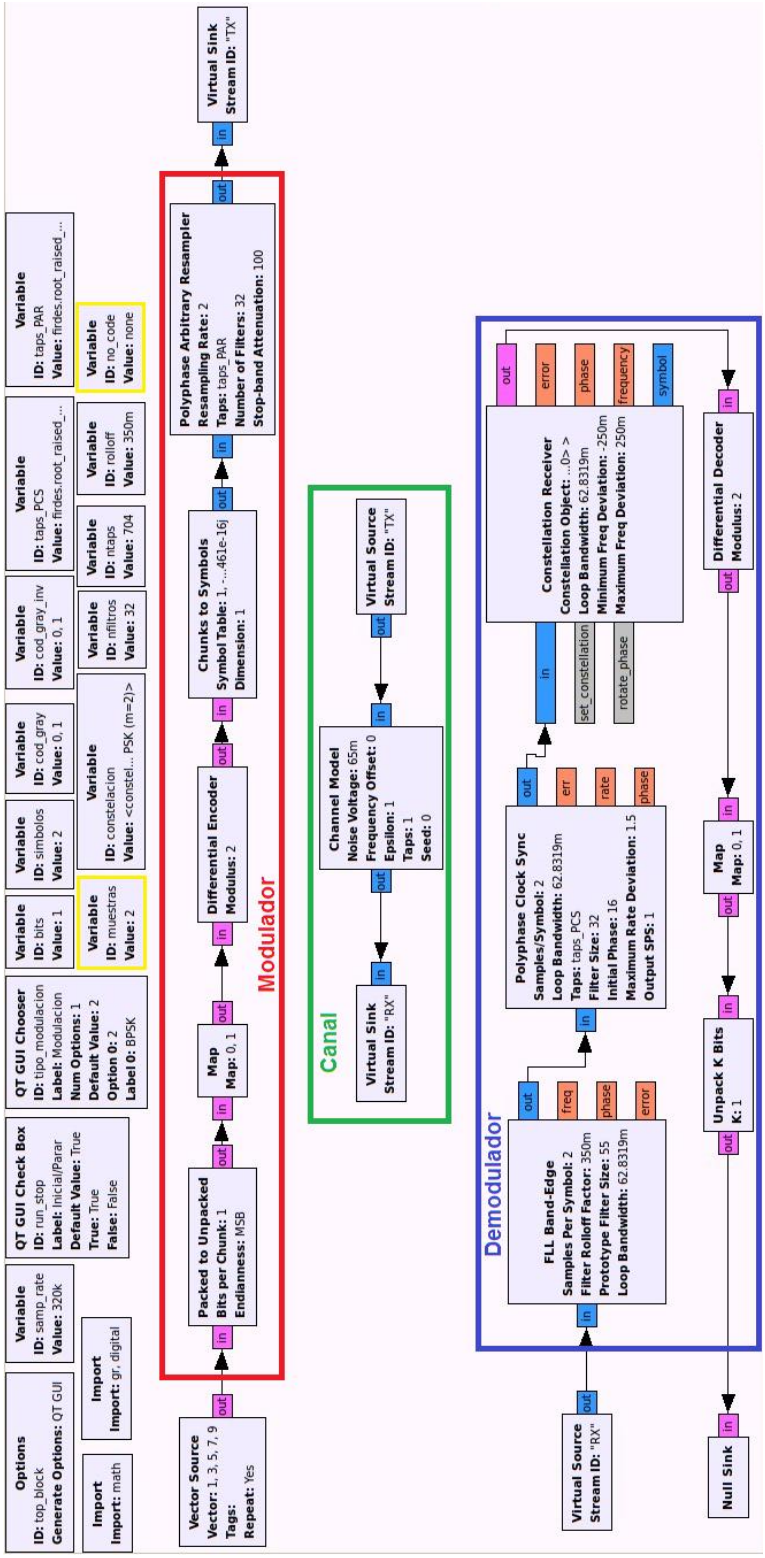
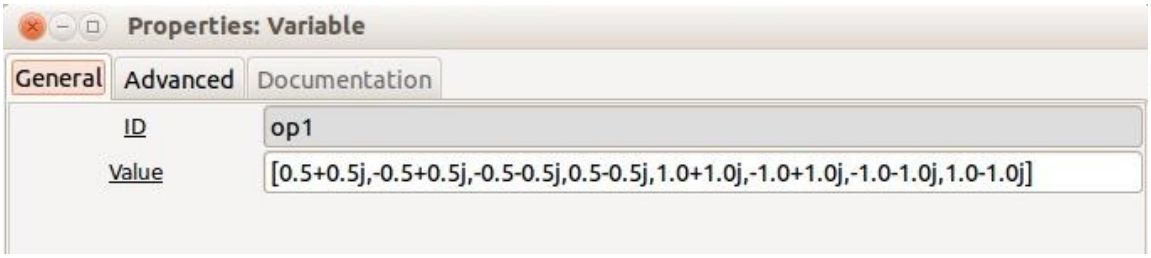


Figura 16. Diagrama de flujo modulación digital BPSK



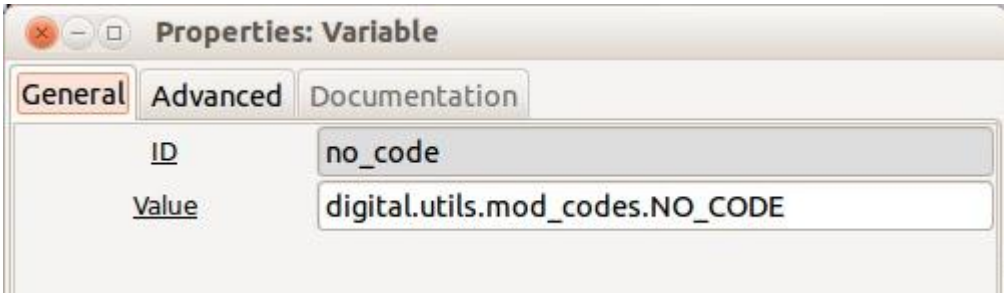
En la sección 4.2.3 se especifica que para utilizar la función *digital.constellation_calcdist* se necesita ingresar el vector de puntos de constelación que va a definir la modulación. La Figura 17 muestra los puntos de constelación que definen la modulación 8-QAM, siendo ingresados en forma de vector, en este caso mediante una lista escrita en lenguaje de programación Python.

Figura 17. Vector de los puntos de constelación ingresados en la variable constelación



La única diferencia entre las variables utilizadas por los tres sistemas de modulación propuestos en Figura 14, Figura 15 y Figura 16, es la variable *op1* de la Figura 17 necesaria para la modulación 8-QAM y la variable *no_code* de la Figura 18 necesaria para la modulación BPSK. El resto de variables son compartidas por los tres sistemas. Estas variables adicionales en las modulaciones BPSK y 8-QAM son incluidas para definir la variable *constelacion* en cada caso, mas no para el proceso que realiza la modulación.

Figura 18. Variable no_code



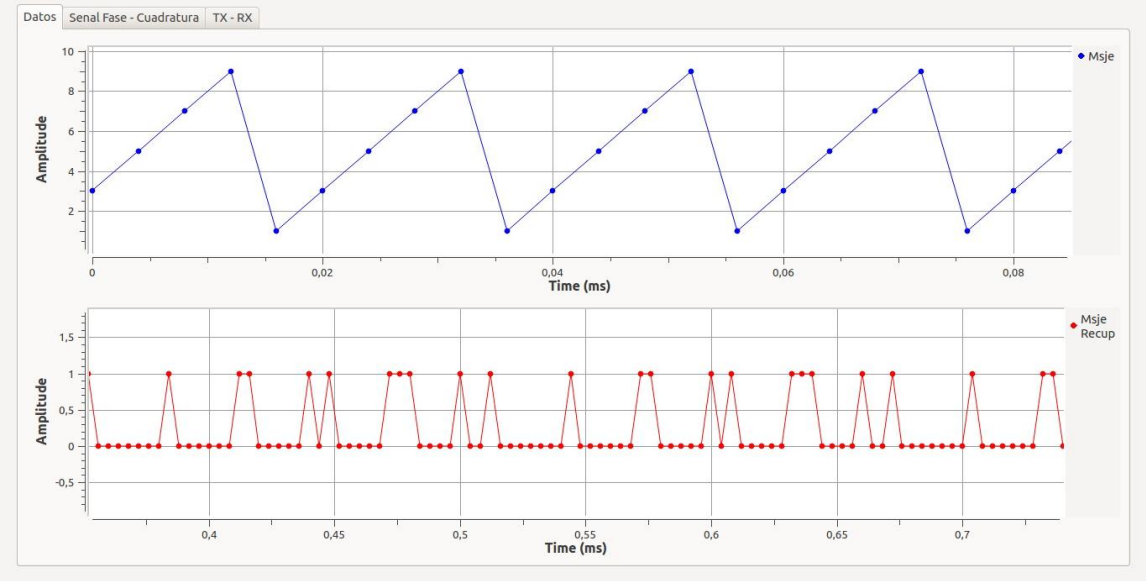
Como se puede interpretar de la Tabla 9 de la sección 4.1, la mayoría de las variables están relacionadas a una principal (*tipo_modulacion*, la cual es definida

por el usuario), de manera que el resto de ellas toman sus valores dependiendo de ésta. Por ejemplo, en general se asigna a la variable *muestras* el valor de la variable *bits* (*muestras* ← *bits*), siendo *bits* ← $log_2\,(tipo\_modulacion)$, siempre y cuando la variable *tipo_modulacion* > 2. Esto debido a que el bloque *Polyphase arbitrary resampler* realiza un ajuste de la frecuencia de muestreo reduciendo el ancho de banda de la señal en función del valor de la variable *muestras.* Por esto, en el caso para la modulación BPSK (*tipo_modulacion* ← 2), se recomienda cambiar manualmente el valor de *muestras* (Figura 19) a dos (2), tanto en el modulador como en el demodulador.

Figura 19. Variable muestras para la modulación BSPK.

La Figura 20 muestra la señal mensaje que se genera con la fuente de vector (gráfica superior) y la señal de datos binarios recuperados (gráfica inferior) de cada uno de los sistemas de modulación propuestos en simulación (8-QAM, 16-QAM y BSPK). Se observa que los números $1, 3, 5, 7\ y\ 9$ son pasados repetitivamente al modulador en formato *byte*. La señal recuperada corresponde a la representación binaria de los bytes enviados. En ella se observa que se repite la secuencia $(00000001\ 00000011\ 00000101\ 00000111\ 00001001)$ que corresponde a la representación binaria de la secuencia enviada en el transmisor.

Figura 20. Fuente de datos (superior) y datos binarios recuperados (inferior) usando modulación BPSK, 8-QAM y 16-QAM.



En los bloques sumideros de tiempo de la *GUI-QT* la base de tiempo con la que se visualizan las señales en la interfaz gráfica depende del parámetro *Sample Rate* del mismo, por tanto si a todos los bloques sumideros utilizados en un flujograma (caso de Figura 14, Figura 15 y Figura 16) se asigna *Sample Rate* ← *samp_rate* la base de tiempo entre muestras de todos los bloques es la misma, resultando así que en algunos casos (p. ej. Figura 20) no exista una correspondencia entre los tiempos de las muestras de las señales mostradas en dichos sumideros de tiempo. El caso particular de la Figura 20 se soluciona asignando *Sample Rate* ← *samp_rate*8* en el bloque de sumidero de tiempo de la señal recuperada en el receptor (gráfica inferior) siempre y cuando en el sumidero de tiempo de la fuente de datos (gráfica superior) *Sample Rate* ← *samp_rate*, pues para una muestra (byte) de la gráfica superior corresponden ocho muestras (bits) de la gráfica inferior.

La gráfica superior (señal m_{IQ}) en Figura 21, Figura 22 y Figura 23, corresponde a las componentes en fase (I) y cuadratura (Q) en el transmisor a la salida del bloque *Chunks to symbol*. La señal de la gráfica inferior (s_c) de las mismas figuras, muestra las componentes en fase (I) y cuadratura (Q) en el receptor a la salida *symbol* del bloque *Constellation receiver.*

Figura 21. Señales en fase (I) y cuadratura (Q) en banda base en el transmisor (arriba) y receptor (abajo) en modulación 16-QAM.

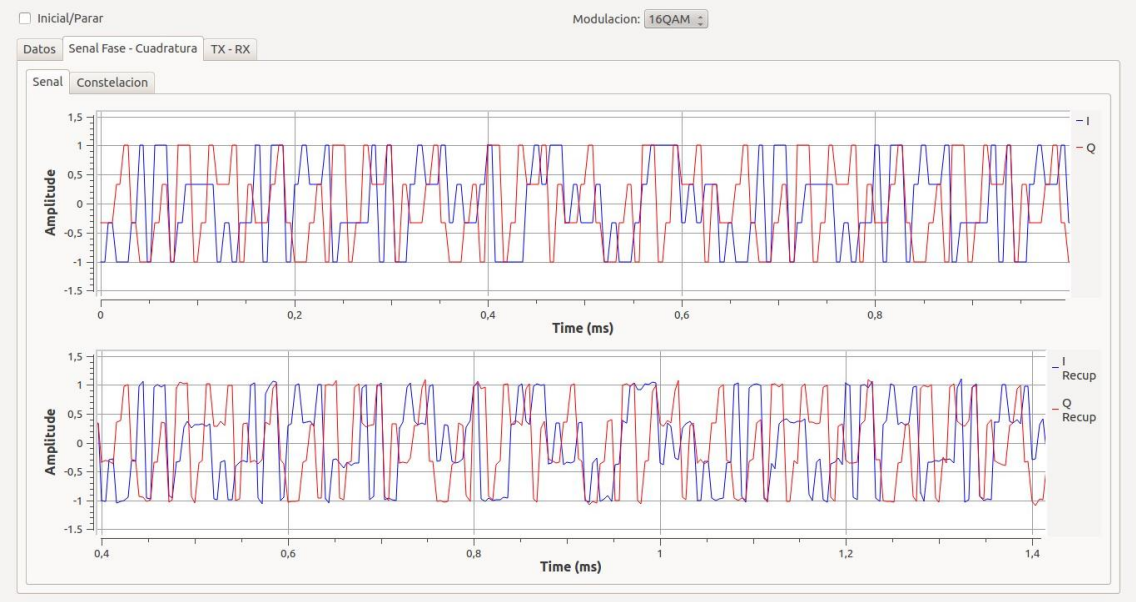


Figura 22. Señales en fase (I) y cuadratura (Q) en banda base en el transmisor (arriba) y receptor (abajo) en modulación 8-QAM.

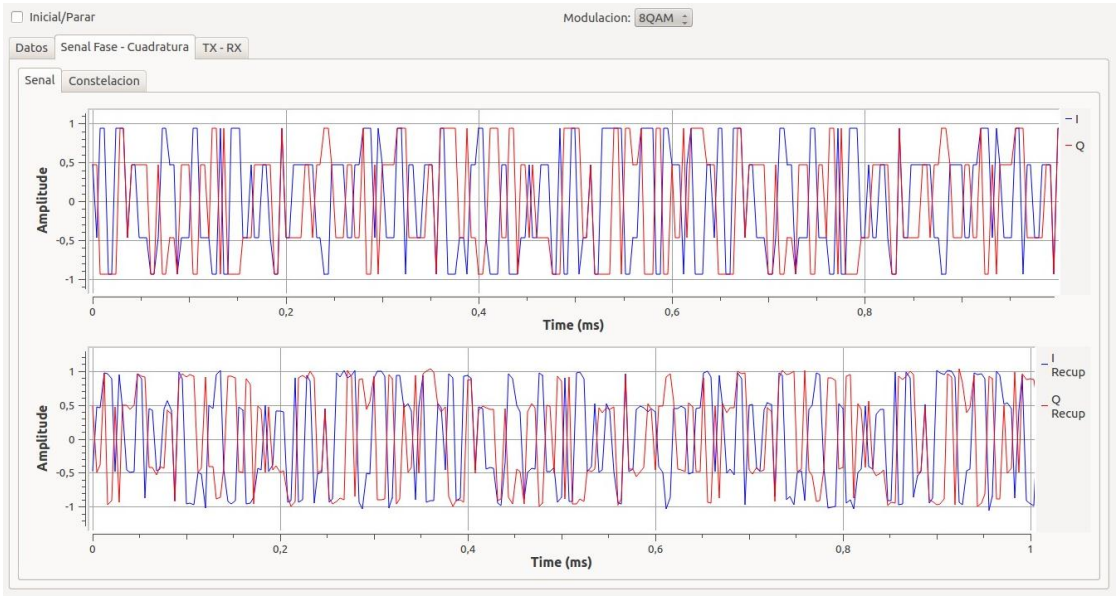
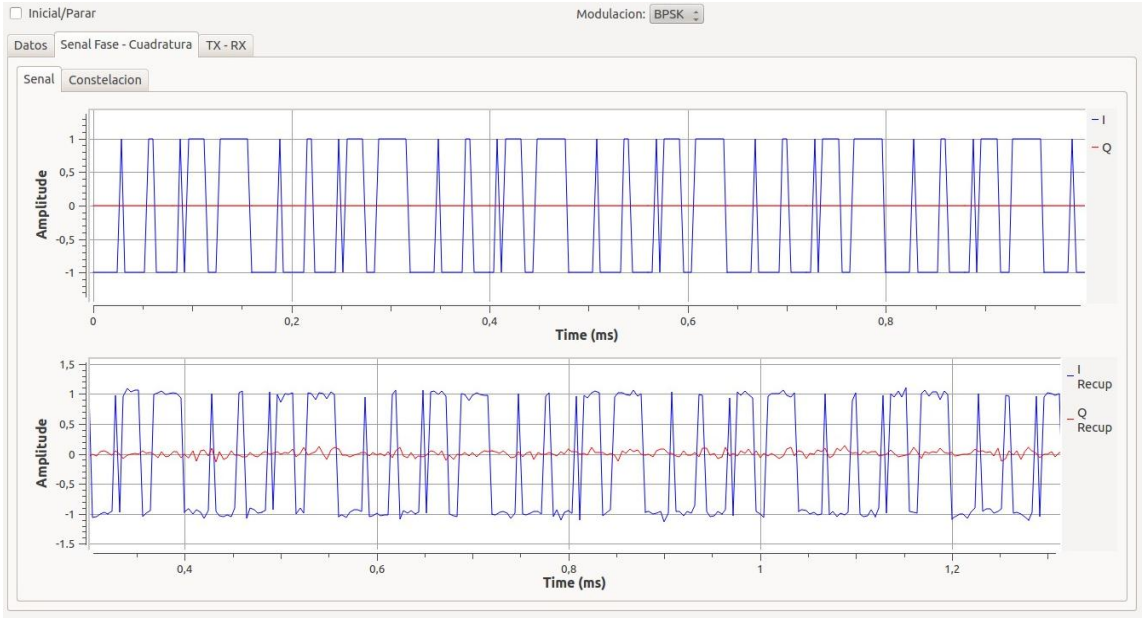


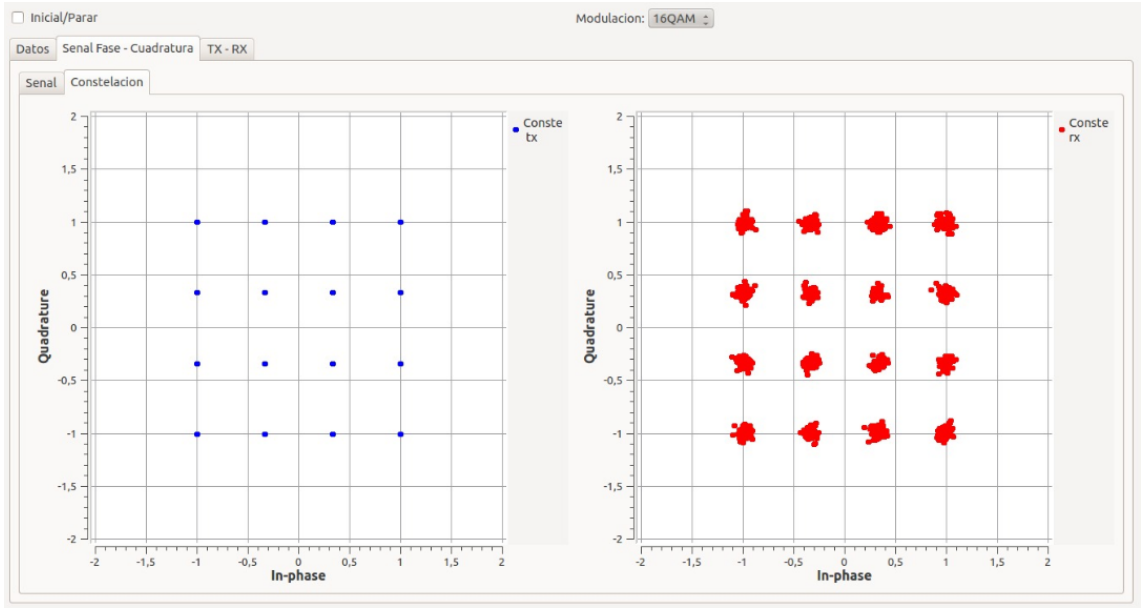
Figura 23. Señales en fase (I) y cuadratura (Q) en banda base en el transmisor (arriba) y receptor (abajo) en modulación BPSK.



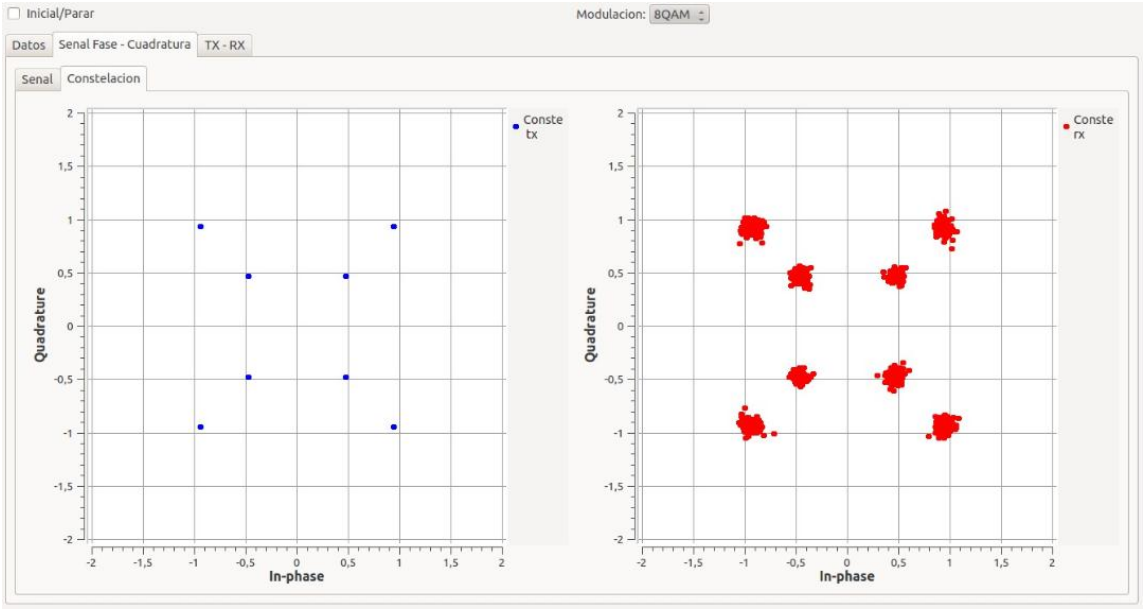
El diagrama de constelación en el receptor se puede ver gracias a que el bloque *constellation_receiver* en los flujogramas en GRC además de su salida principal (*out*), posee otras cuatro salidas adicionales (*error, phase, frequency* y *symbol*) que corresponden a señales provenientes del proceso interno del bloque. Los datos de salida en *symbol* equivalen a los datos de entrada después de realizar el ajuste fino de fase y frecuencia. Tenga en cuenta que los datos en *symbol* son de tipo complejo, en donde estos llevan las componentes en fase (I) y cuadratura (Q) en banda base.

Los diagramas de constelación que resultan de las gráficas mostradas en Figura 21, Figura 22 y Figura 23 se observan en la Figura 24, gráfica (a), (b) y (c) respectivamente. En la gráfica de la izquierda los puntos de constelación o puntos de mensaje se muestran definidos debido a que esta gráfica proviene de las componentes en fase (I) y cuadratura (Q) de la señal en banda base sin ningún ajuste o afectación por ruido. Por otra parte, la gráfica de la derecha exhibe puntos de constelación algo dispersos a causa del ruido agregado a la señal transmitida en el canal y a los problemas en la sincronización entre el transmisor y receptor.

Figura 24. Diagramas de constelación en el transmisor (izquierda) y receptor (derecha) para: (a) modulación 16-QAM, (b) modulación 8-QAM y (c) modulación BPSK.

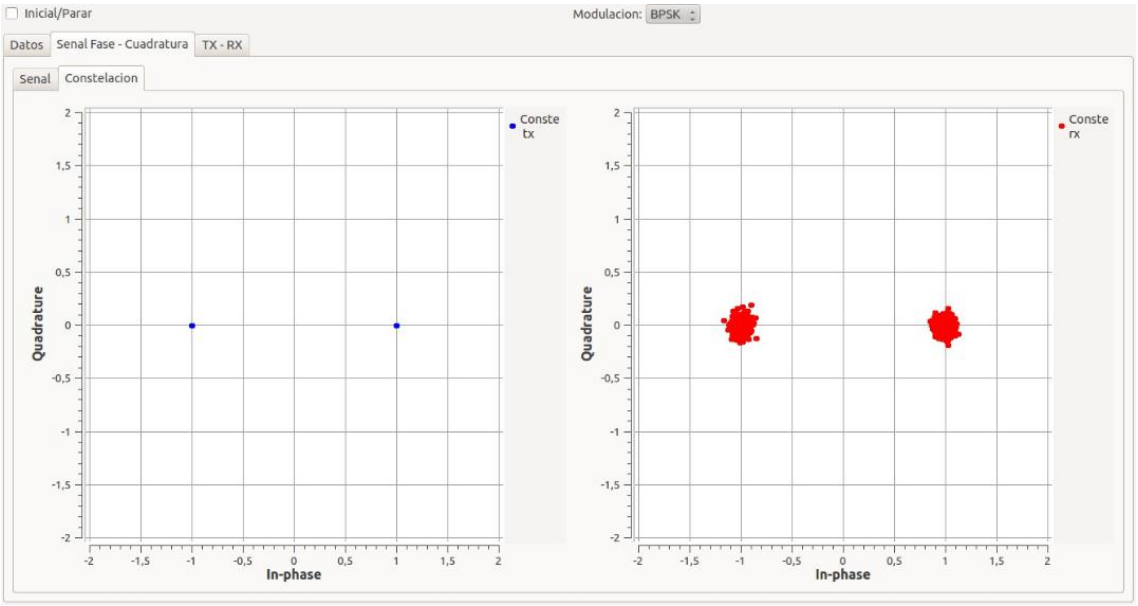


(a)



(b)

Figura 24. (Continuación)



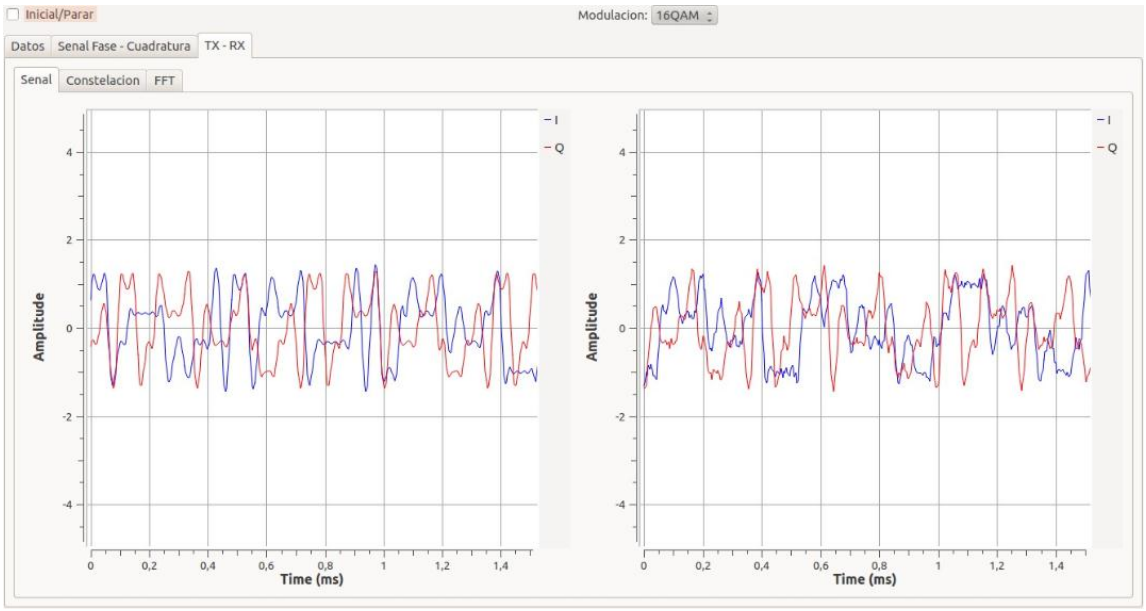
(c)

La Figura 25 muestra las señales en fase (I) y cuadratura (Q) en banda base en el transmisor después del bloque *Polyphase Arbitrary resampler* (después de realizar el ajuste para la transmisión en el canal, gráfica izquierda) y la señal recibida en el receptor (gráfica derecha).

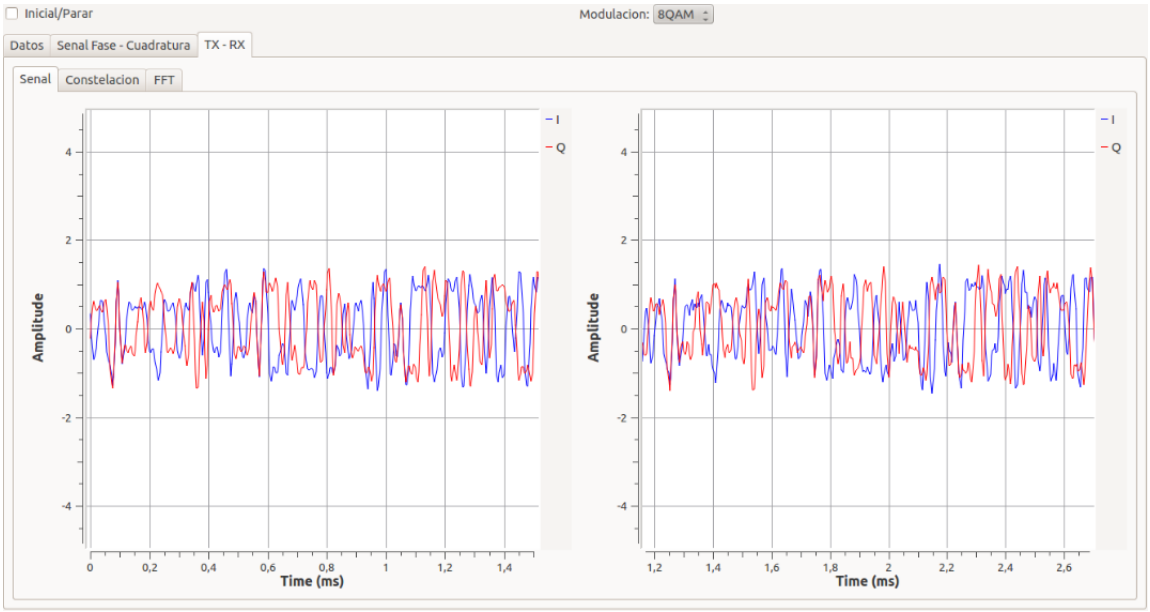
Debido a limitaciones en el ancho de banda del canal en el que se transmite, se hace necesario realizar una adecuación de la señal que se desea transmitir. En este caso se utiliza un filtro RRC junto con un interpolador para limitar el ancho de banda de los símbolos complejos que resultan del mapeo de la señal mensaje. Esto se aprecia comparando la gráfica superior de Figura 21, Figura 22 y Figura 23 con las gráficas izquierdas (a) (b) y (c) de la Figura 25.

Comparando las gráficas de la izquierda y derecha de Figura 25, Figura *26* y Figura 27 se aprecia cómo la incidencia del ruido en el canal distorsiona la señal que es recibida en el receptor, con lo que se justifica la modulación de la señal para transmitirla y recibirla adecuadamente.

Figura 25. Señales en fase (I) y cuadratura (Q) en banda base en el transmisor a la salida del bloque *Polyphase Arbitrary resampler* (izquierda), señales en fase (I) y cuadratura (Q) en banda base de los datos recibidos en el receptor (derecha) para: (a) modulación 16-QAM, (b) modulación 8-QAM y (c) modulación BPSK.

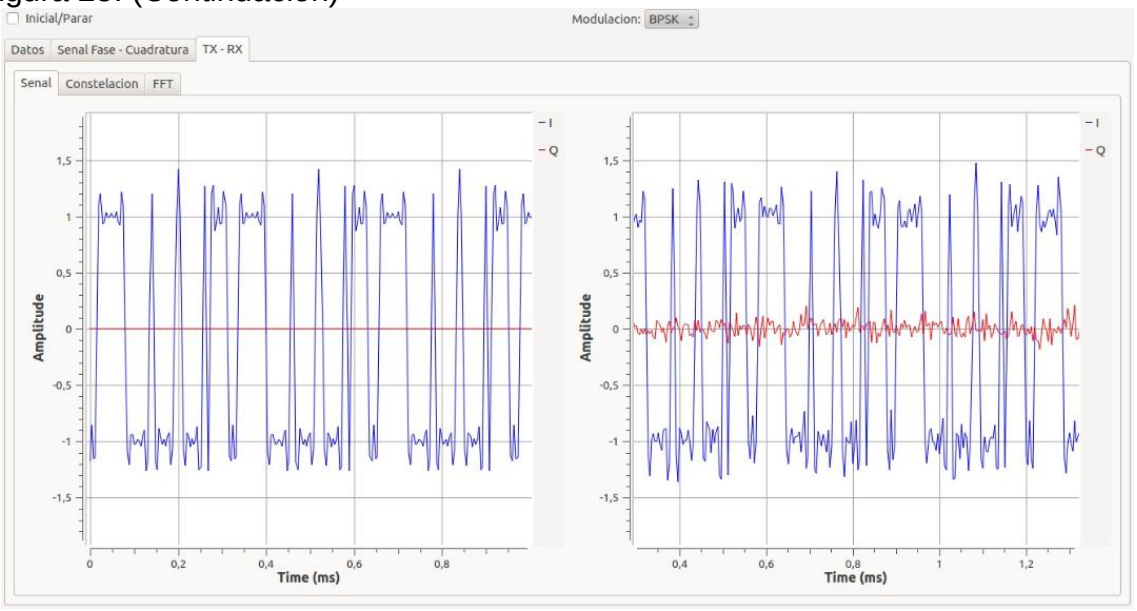


(a)



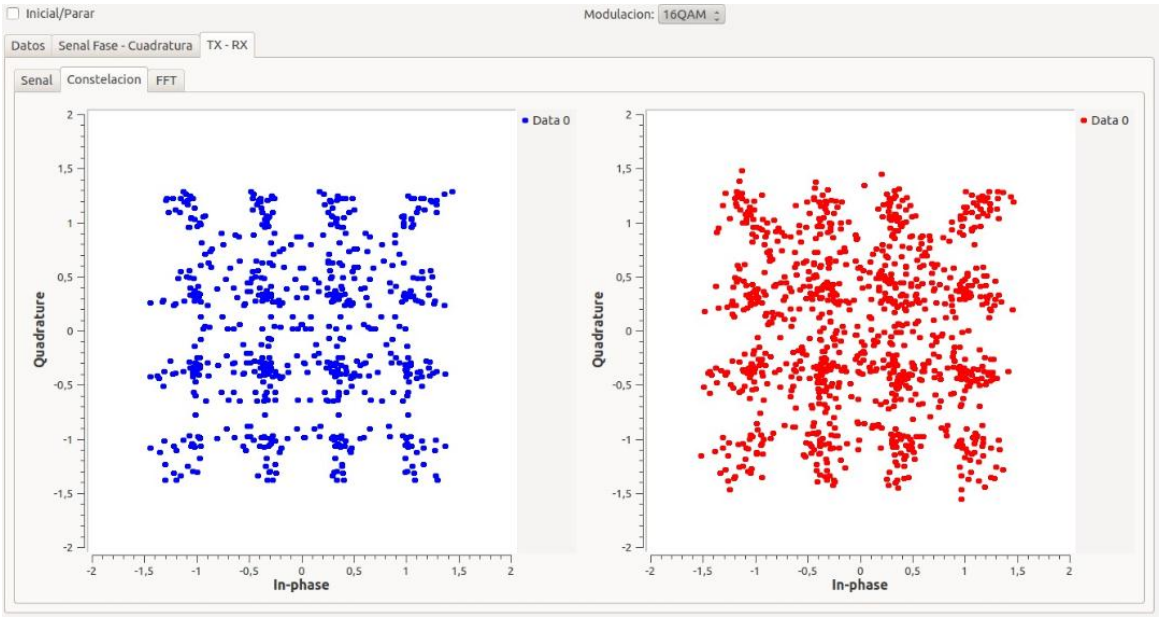
(b)

Figura 25. (Continuación)



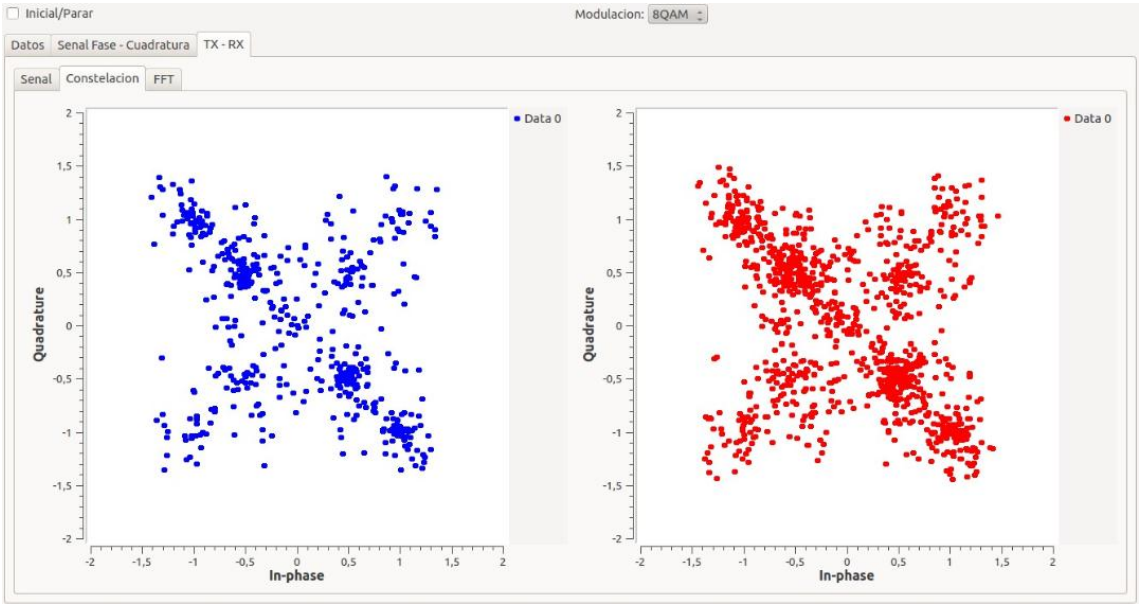
(c)

Figura 26. Diagramas de constelación de las señales en fase (I) y cuadratura (Q) en banda base en el transmisor a la salida del bloque Polyphase Arbitrary resampler (izquierda), diagrama de constelación de las señales en fase (I) y cuadratura (Q) en banda base de los datos recibidos en el receptor (derecha) para: (a) modulación 16-QAM (b) modulación 8-QAM y (c) modulación BPSK.

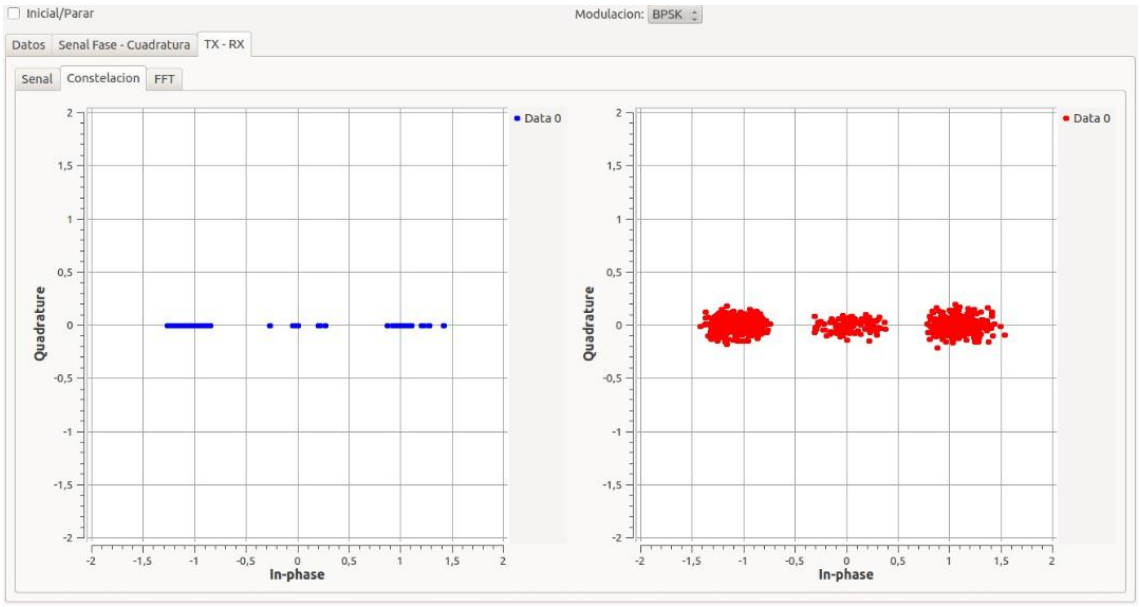


(a)

Figura 26. (Continuación)



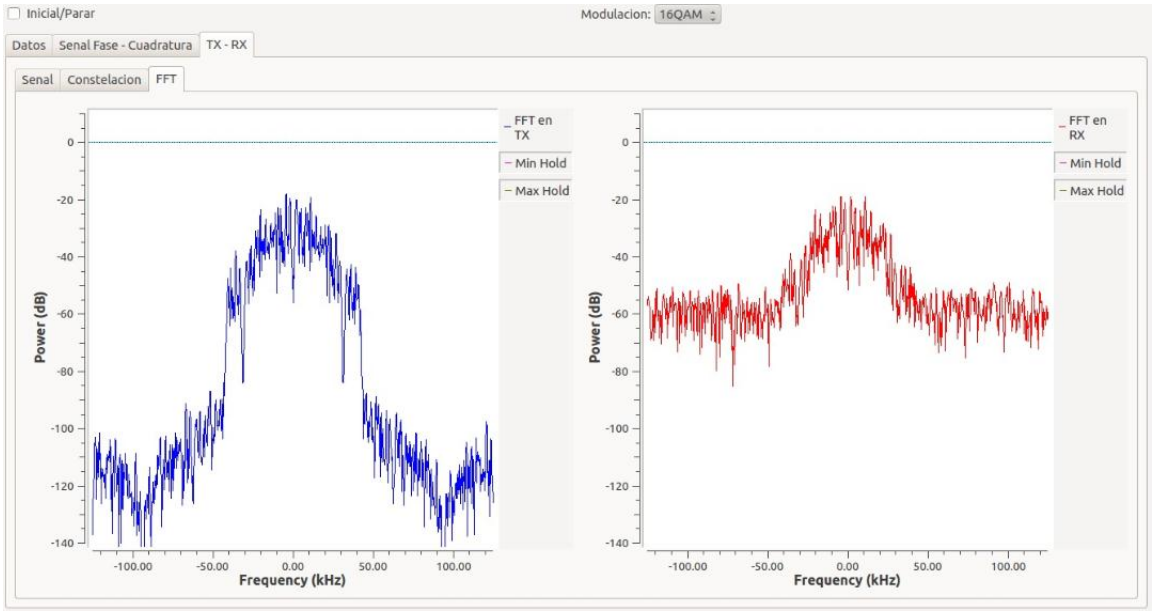
(b)



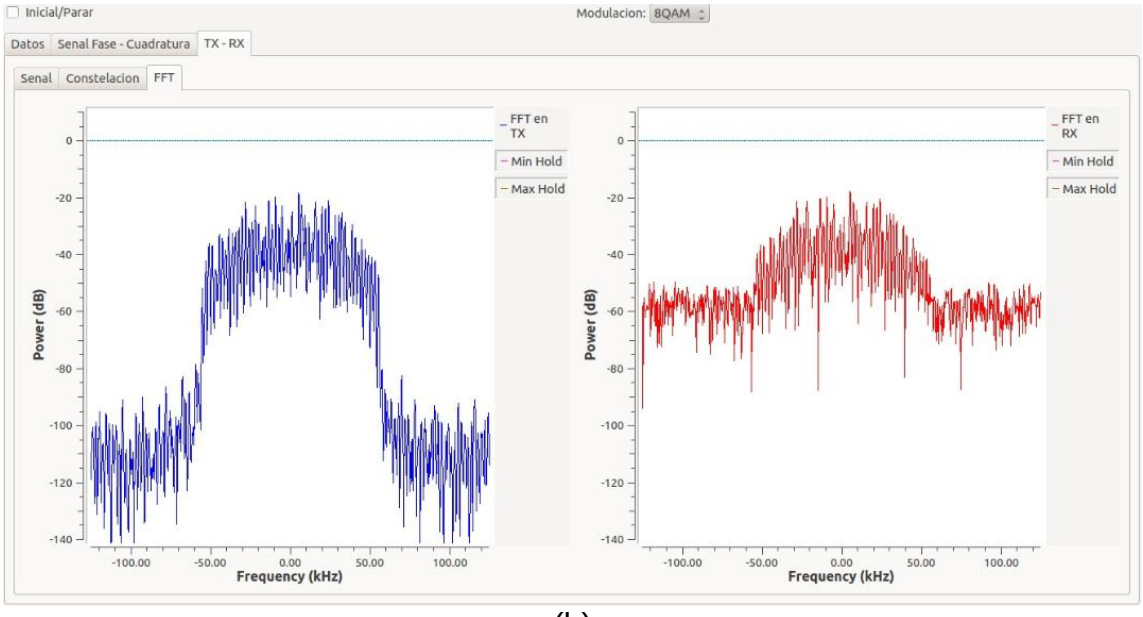
(c)

En la Figura 27 se aprecia que el ancho de banda requerido por cada modulación disminuye a medida que aumenta el número de niveles en éstas, así como también la presencia de un ruido Gaussiano en el receptor (gráfica derecha).

Figura 27. Espectro de frecuencias de las señales en fase (I) y cuadratura (Q) en banda base en el transmisor a la salida del bloque Polyphase Arbitrary resampler (izquierda), espectro de frecuencias de las señales en fase (I) y cuadratura (Q) en banda base de los datos recibidos en el receptor (derecha) para: (a) modulación 16-QAM (b) modulación 8-QAM y (c) modulación BPSK

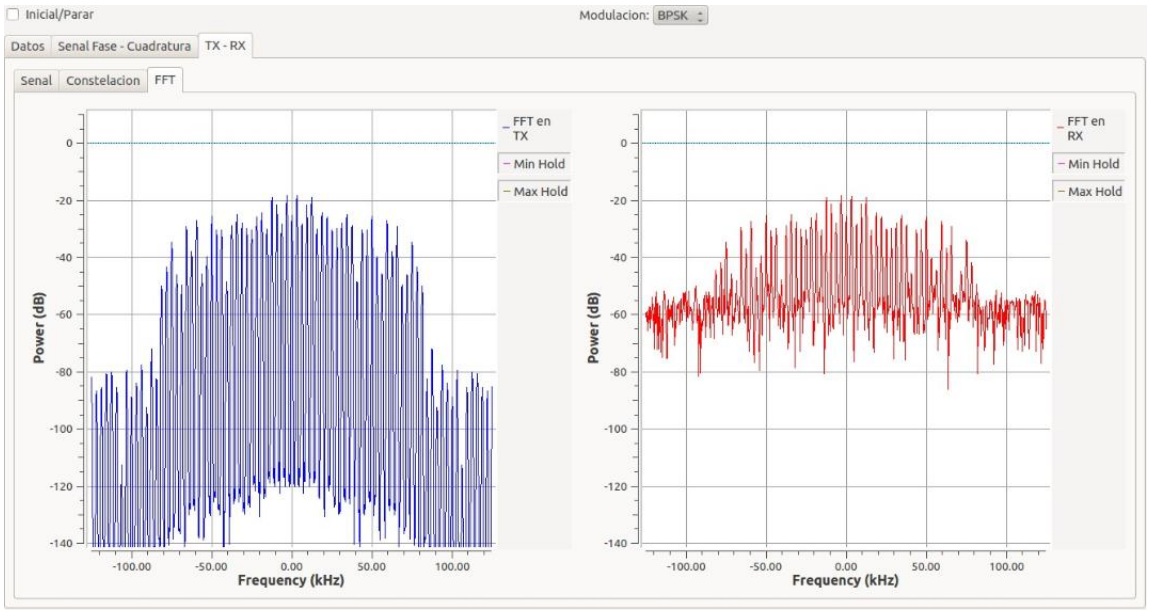


(a)



(b)

Figura 27. (Continuación)



(c)

6. RESULTADOS DE PRUEBAS DE FUNCIONAMIENTO

En esta sección se presentan algunas pruebas que verifican el funcionamiento del diseño propuesto. En primer lugar se presentan los resultados de una prueba de transmisión-recepción usando dos dispositivos USRP. Luego se analiza una prueba de transmisión del USRP usando el *Analizador Vectorial de Redes ZVL6*[5] como receptor. Finalmente, se realiza una prueba para determinar la tasa de error de bits del sistema propuesto con diferentes modulaciones digitales.

6.1 Prueba 1. TRANSMISIÓN-RECEPCIÓN USRP-USRP

Consiste en transmitir una señal conocida en un PC y recibirla en otro usando modulación digital 8-QAM. Para esta prueba se utiliza la conexión *PC – USRP – USRP – PC* que se muestra en la sección 3.4, empleando un atenuador SMA (M/F) de 30 dB para proteger la entrada de la tarjeta secundaria (RFX2400) del USRP1 que trabaja como receptor. Se usa un cable SMA (M/M) para realizar la conexión física entre los dispositivos USRP1.

La Figura 28 muestra el flujograma correspondiente al transmisor que usa modulación 8-QAM, según se describe en el capítulo 5 y el anexo A3. Además, se muestran los bloques sumideros para visualizar en la GUI las gráficas en tiempo, frecuencia y diagrama de constelaciones de las señales tratadas y el bloque *UHD: USRP Sink* con el que se lleva la señal desde el PC hacia el medio de transmisión. Observe que la señal a transmitir se atenúa en software por un factor de 0,15 con el fin de que los datos digitales entrantes al USRP no excedan los máximos numéricos que éste puede registrar para realizar su posterior conversión de señal digital-análoga.

Por su parte, el receptor 8-QAM (Figura 29) cuenta con el bloque *UHD: USRP Source*, y con los bloques mencionados en el anexo A4, incluyendo el bloque que realiza el control automático de ganancia (*agc2*) no utilizado hasta el momento. También se presentan los bloques utilizados para visualizar gráficas en la GUI.

[5] Instrumento analizador de redes de Rohde & Schwarz. Ver: http://www.rohde-schwarz.com.co/product/zvl6.html (último acceso: 26 de febrero 2015)

Figura 28. Diagrama de flujo transmisor modulación 8-QAM en GRC.

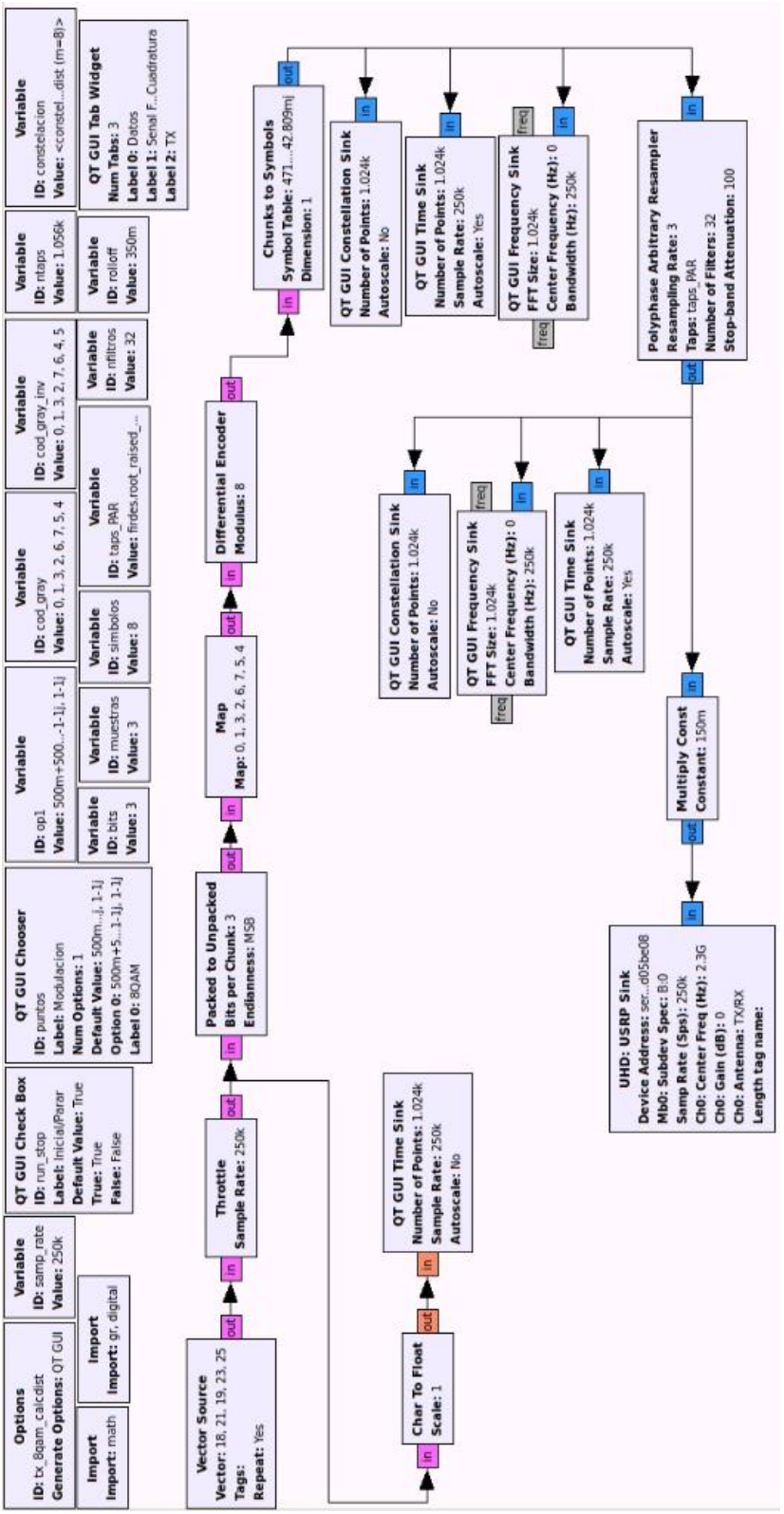
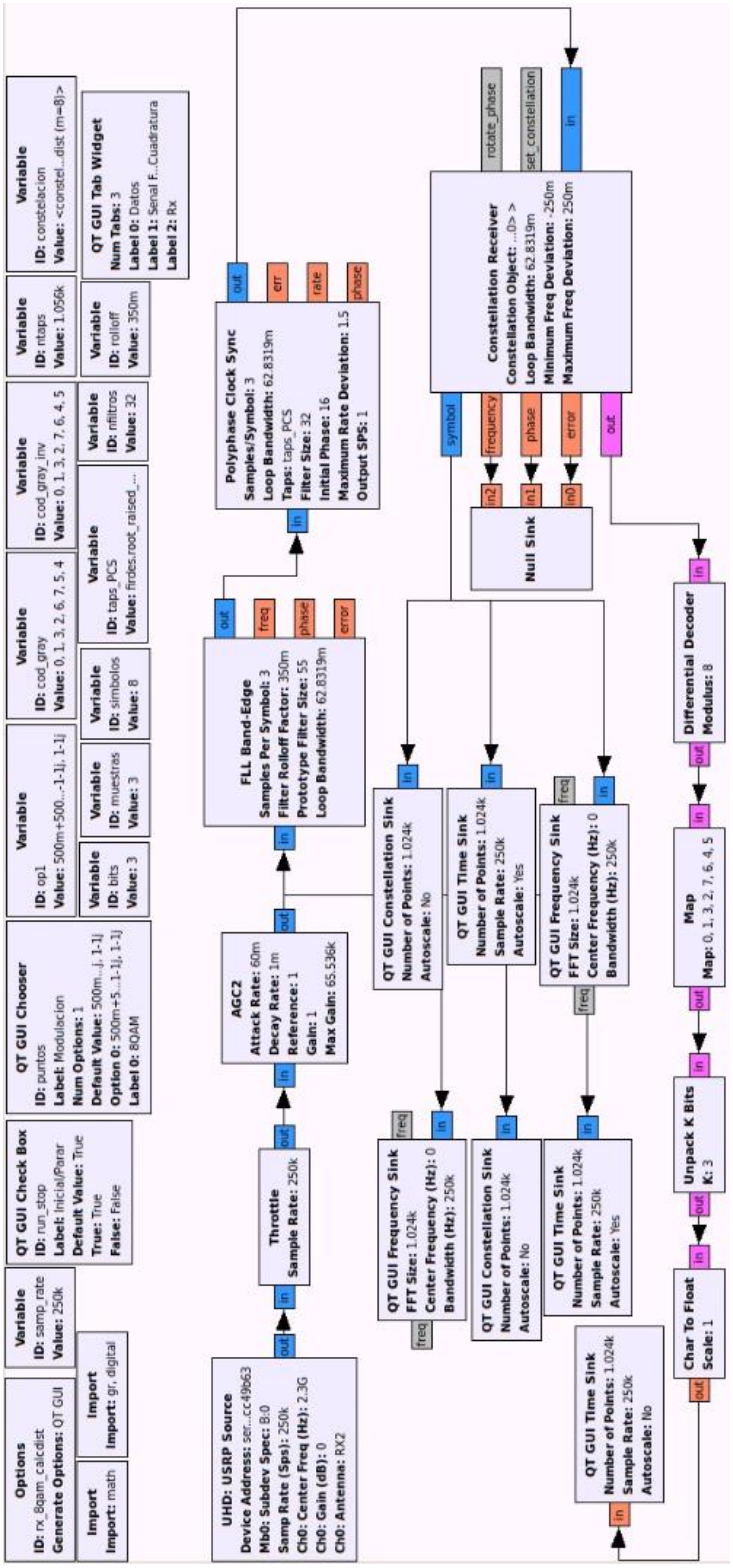


Figura 29. Diagrama de flujo receptor modulación 8-QAM en GRC.



Se utiliza una fuente tipo vector para generar los datos, siendo éstos $18, 21, 19, 23, 25$ (Figura 30) en su representación decimal o $0x12, 0x15, 0x13, 0x17, 0x19$ en su representación hexadecimal. Los datos recuperados en el receptor mostrados en la Figura 31, corresponden a la representación binaria de los bytes enviados $(00010010\ 00010101\ 00010011\ 00010111\ 00011001)$, observándose así que se obtiene lo esperado y que el proceso de demodulación realizado por el módulo *digital.constellation_calcdist* en este caso es funcional.

Figura 30. Vector de datos en formato *byte* procedente de la fuente generadora de mensaje en el transmisor 8-QAM.

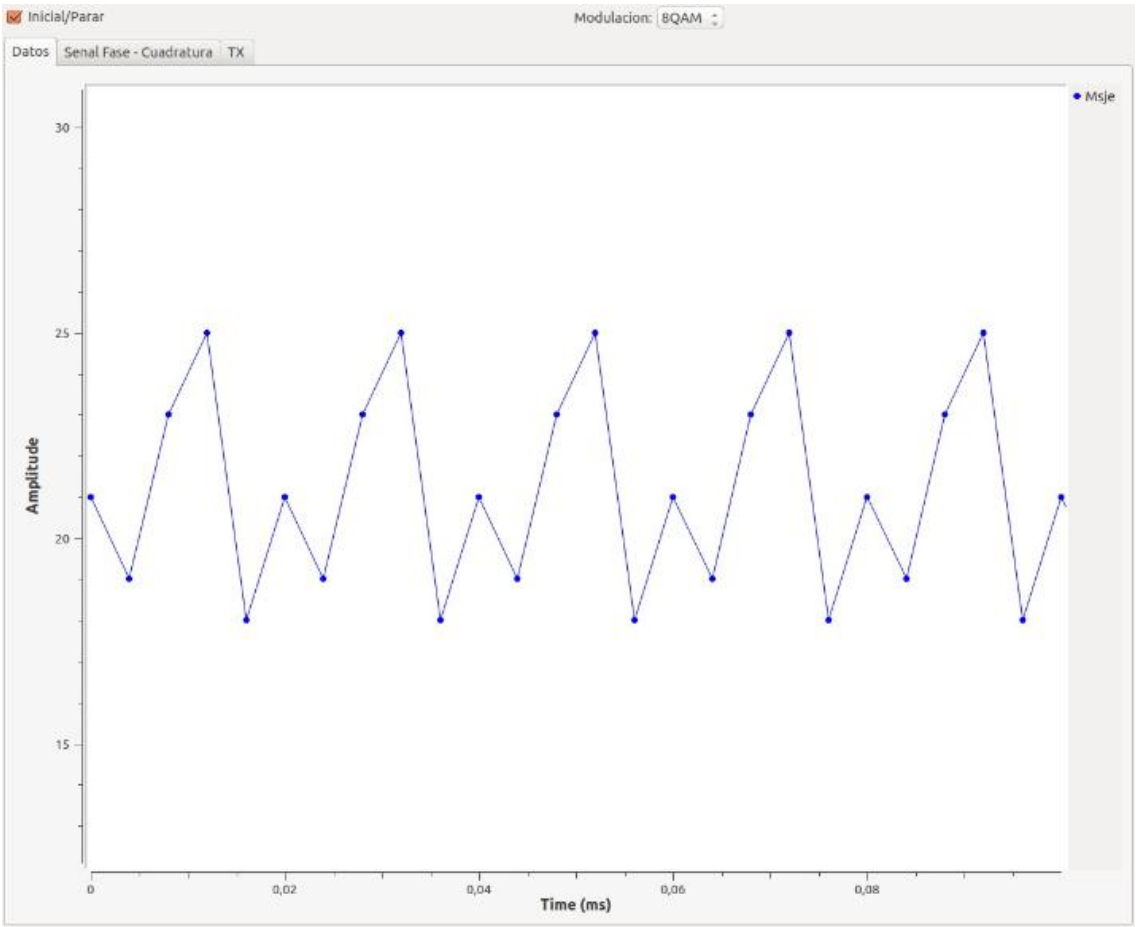
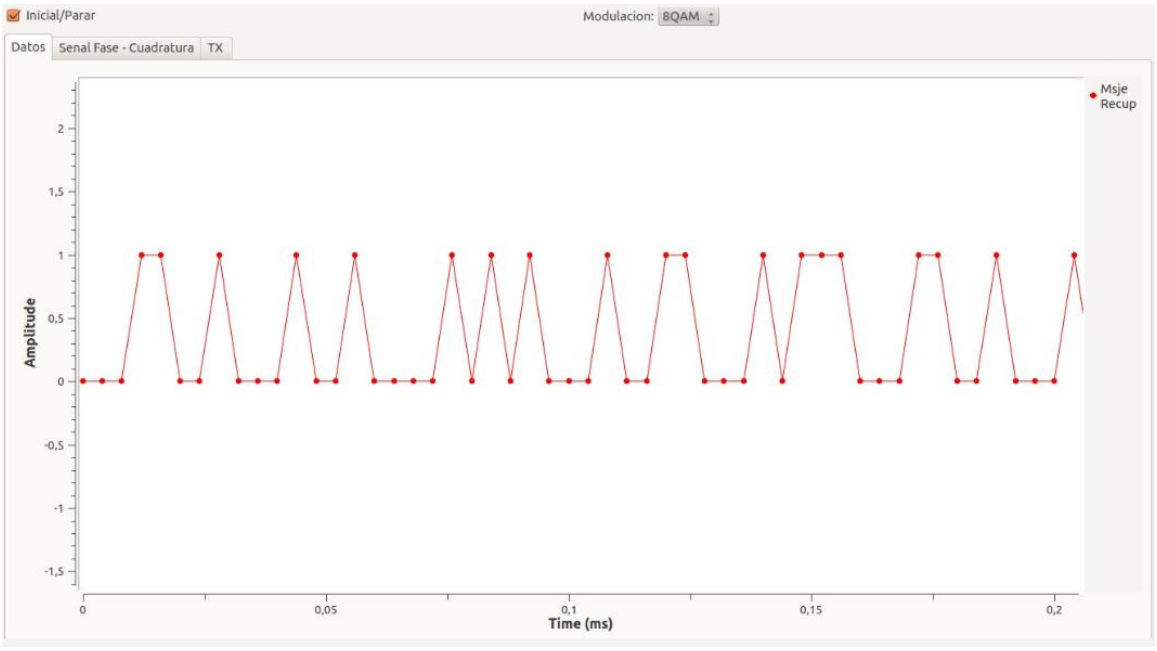


Figura 31. Vector de datos recuperados correspondiente a la representación binaria de los *bytes* enviados en el transmisor, usando modulación 8-QAM.



Comparando las gráficas de las señales en tiempo de Figura 32 y Figura 33 se observa que en el receptor se recuperan las formas de onda correspondientes a las señales en fase (I) y cuadratura (Q) del transmisor con un tiempo de retraso siendo esto normal en los enlaces reales. También se observa que en el diagrama de constelación en el receptor se encuentran los puntos dispersos debido al ruido agregado a la señal en el canal, que en este caso es el ruido agregado por el dispositivo USRP1 cuando se realiza las conversiones de señal digital-análoga en el transmisor y análoga-digital en el receptor.

Viendo que la señal enviada se recupera satisfactoriamente (Figura 31) se concluye que a pesar de los factores adversos que corrompen la señal original transmitida, el bloque *constellation receiver* (*RECEP. SÍMBOLOS*) en el receptor realiza una conversión adecuada de los símbolos complejos a símbolos codificados, resultando en una recuperación aceptable de la señal mensaje.

Figura 32. Señales en fase (I) y cuadratura (Q) en banda base en el transmisor 8-QAM a la salida del bloque *chunks to symbols*: señal en tiempo (grafica superior), diagrama de constelación (inferior izquierda) y espectro de frecuencias (inferior derecha).

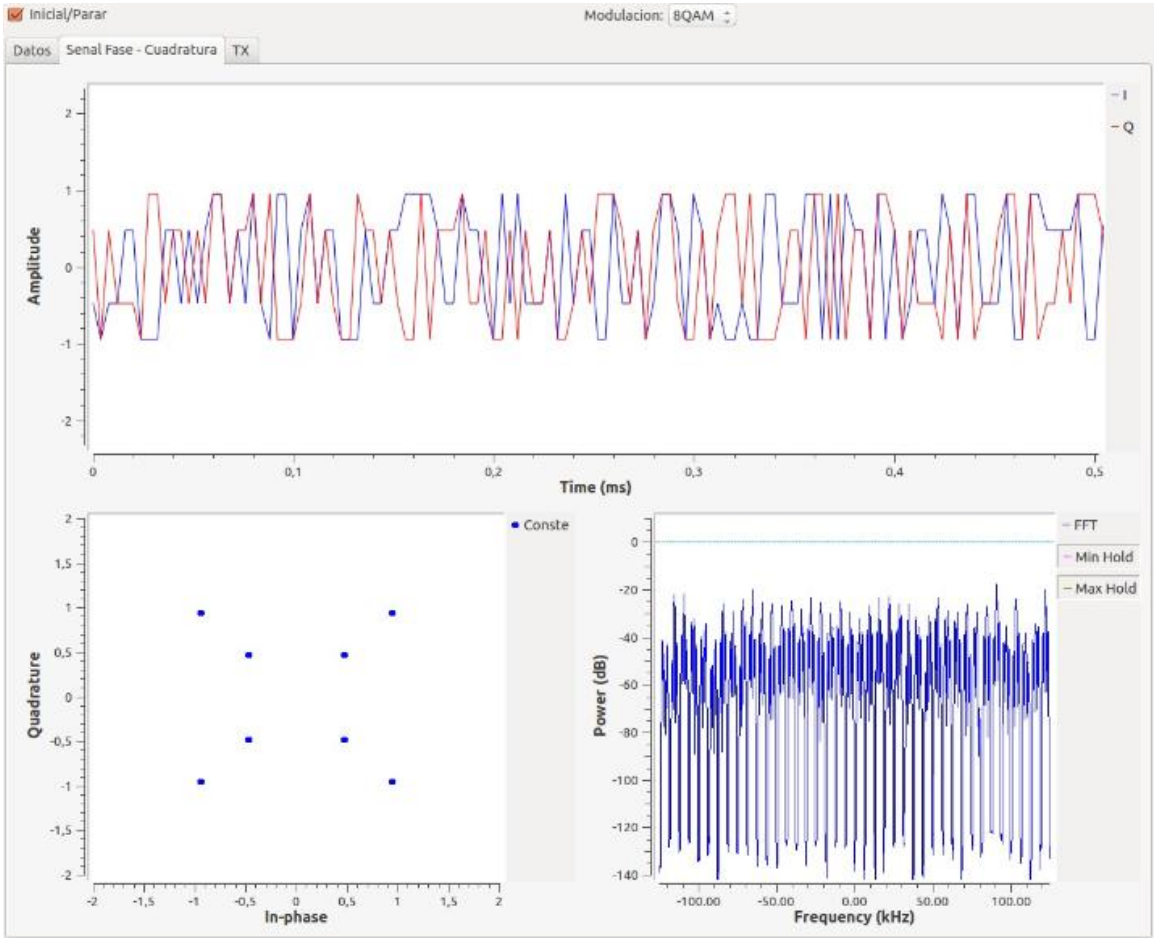
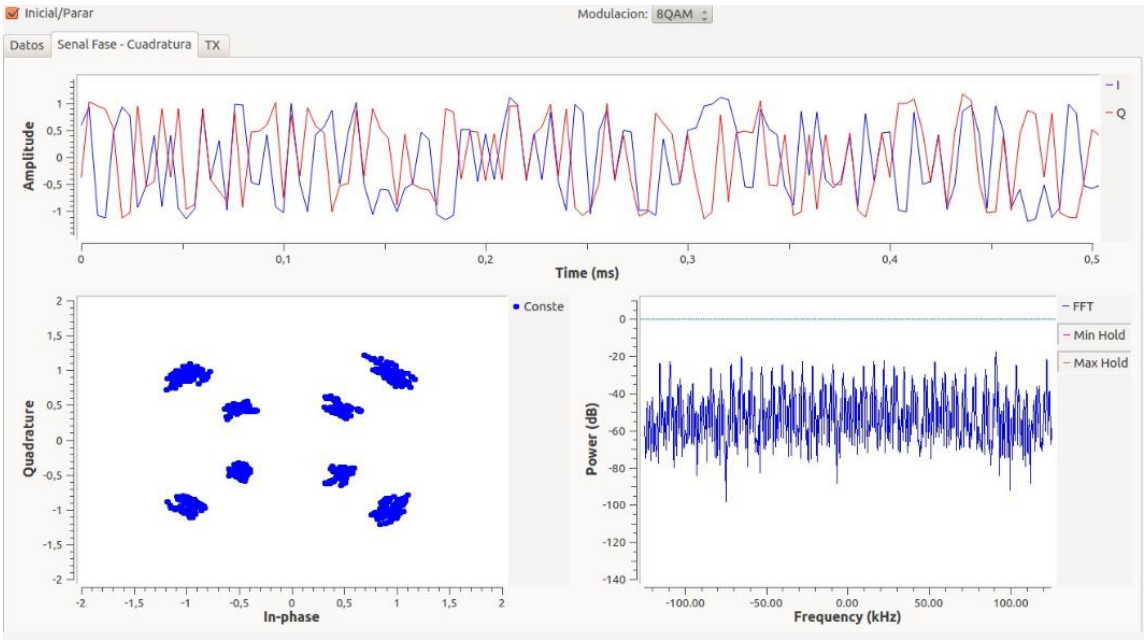
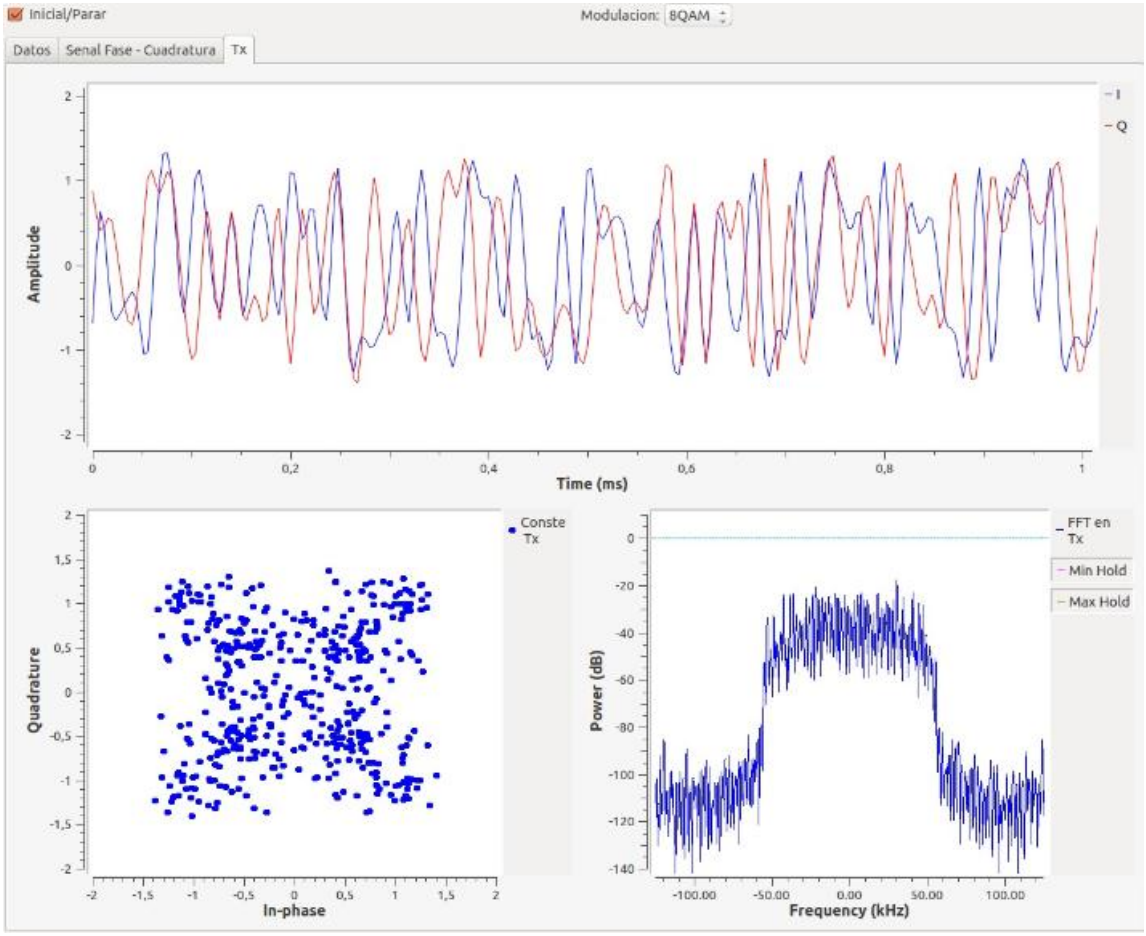


Figura 33. Señales en fase (I) y cuadratura (Q) en banda base recuperadas en el receptor 8-QAM: señal en tiempo (grafica superior), diagrama de constelación (inferior izquierda) y espectro de frecuencias (inferior derecha).



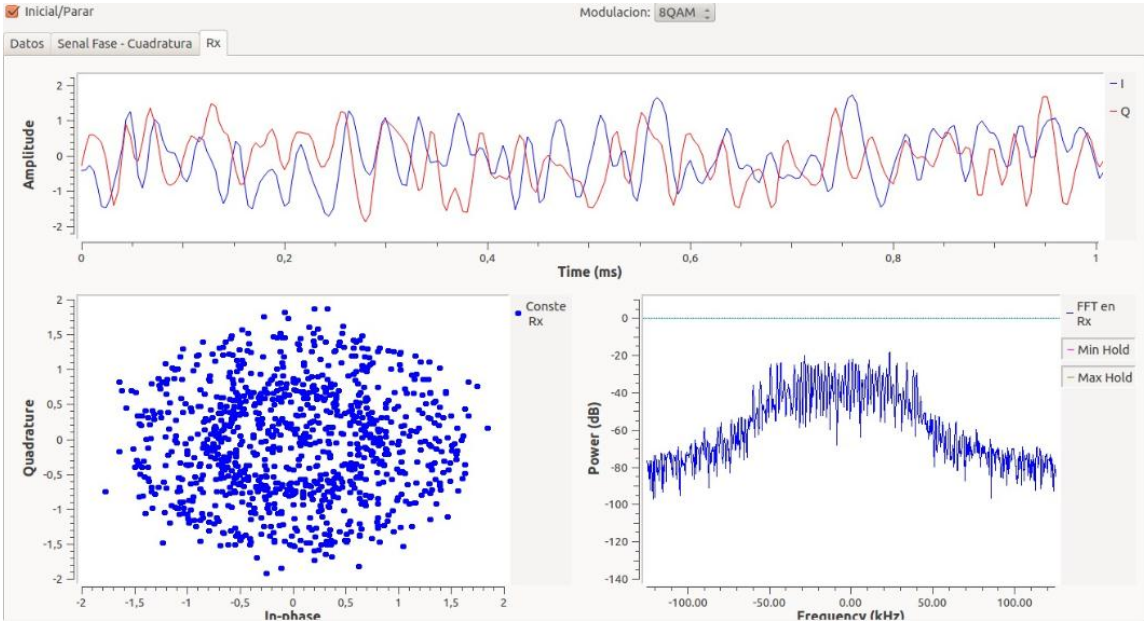
Comparando las señales de espectro de frecuencias de Figura 32 y Figura 34 se evidencia que el espectro de la señal es reducido gracias al bloque *Polyphase arbitrary resampler* (*INTERP.*) de la Figura 28, asegurando que la señal sea adecuada para transmitir hacia el dispositivo USRP1 y posteriormente al canal de comunicación.

Figura 34. Señales en fase (I) y cuadratura (Q) en banda base antes del bloque USRP en el transmisor 8-QAM: señal en tiempo (grafica superior), diagrama de constelación (inferior izquierda) y espectro de frecuencias (inferior derecha).



En la Figura 35 se muestra la incidencia del ruido agregado mientras se transmite la señal desde el dispositivo USRP1 utilizado para transmisión hasta el usado para la recepción. Recuerde que este enlace de comunicación utiliza un cable *SMA* y un atenuador para conectar los dispositivos USRP1, por tanto el ruido presente es el agregado por estos dispositivos.

Figura 35. Señales en fase (I) y cuadratura (Q) en banda base recibidas por el bloque USRP en el receptor 8-QAM: señal en tiempo (grafica superior), diagrama de constelación (inferior izquierda) y espectro de frecuencias (inferior derecha).



6.2 Prueba 2. TRANSMISIÓN USRP – RECEPCIÓN ANALIZADOR VECTORIAL DE REDES ZVL6 (ANALIZADOR DE ESPECTROS)

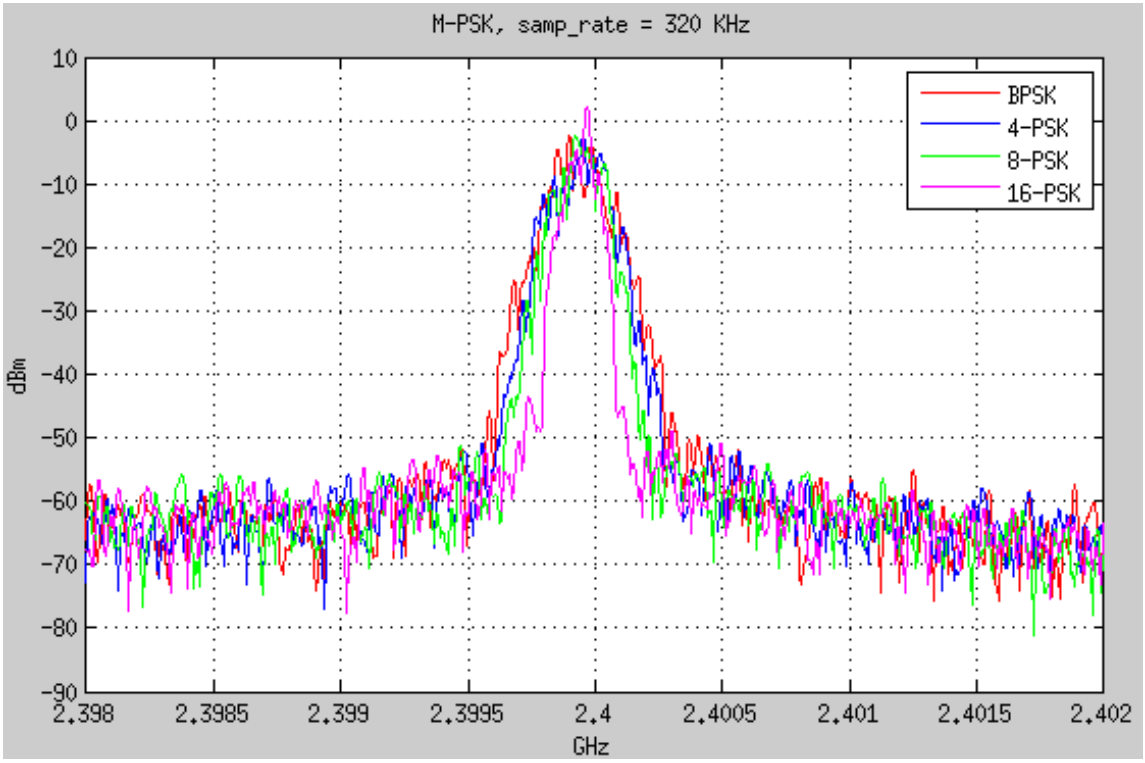
Esta prueba consiste en transmitir una señal modulada digitalmente usando los diseños propuestos para recibirla con *Analizador vectorial de redes ZVL6* en modo analizador de espectros. Se usa la conexión *PC – USRP1 – INSTRUMENTO DE MEDICIÓN/GENERACIÓN* que se muestra en la sección 3.4 con la diferencia que en este caso no se usa el atenuador de señal debido a que el puerto de entrada del *Analizador vectorial de redes ZVL6* recibe hasta 27 dBm y la máxima potencia de transmisión de la tarjeta RFX2400 en el USRP1 es de 17dBm, además que en el flujograma se usa un factor de 0.5 para atenuar la señal antes de pasar al bloque *USRP Sink* (antes de que la señal pase de digital a análoga).

Se guardaron los datos registrados por el dispositivo *Analizador vectorial de redes ZVL6* en archivos .dat para luego ser graficados usando el software MATLAB.

La Figura 36 muestra los espectros de frecuencias de las modulaciones BPSK, 4-PSK, 8-PSK y 16-PSK, observándose allí que a medida que el número de niveles

en la modulación aumenta se disminuye el ancho de banda usado por la modulación (exceptuando entre modulaciones BPSK y 4-PSK), pues para un mismo valor de tasa de muestreo se varia el valor de la variable muestras cambiando el tipo de modulación utilizada.

Figura 36. Espectros de frecuencias de las modulaciones BPSK, 4-PSK, 8-PSK y 16-PSK.

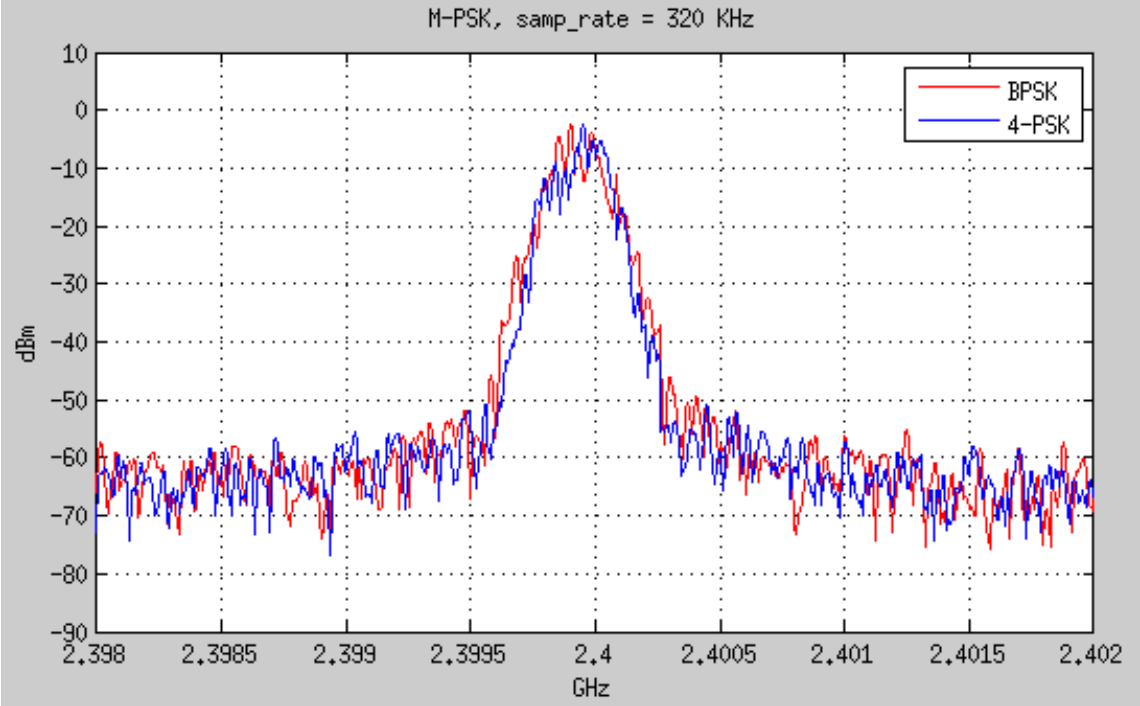


Como se observa en la Figura 37, los espectros de frecuencias de las modulaciones BPSK y 4-PSK propuestos se presentan de manera similar (mismo ancho de banda), pues usan la misma frecuencia de muestreo y un mismo factor de interpolación, cambiando solo que en el sistema BPSK se transmite 1 bit/muestra mientras que en el 4-PSK se transmite 2 bits/muestra. Esto se traduce en una velocidad de transmisión del doble para la modulación 4-PSK en comparación a la modulación BPSK.

El factor de interpolación se encuentra presente en el bloque *Polyphase arbitrary resampler*, específicamente el parámetro *rate*, el cual especifica la tasa de re-muestreo a usar (ver Polyphase arbitrary resampler en la sección A3). En el diseño propuesto se asigna *rate* ← *muestras,* resultando en el uso de un mismo

factor de interpolación debido a que en la modulación 4-PSK la variable *muestras* es igual a dos (2) pues *muestras* ← $log_2(4)$, la cual posee el mismo valor que se obtiene en la modulación BPSK ya que esta variable en ese caso es *muestras* ← $log_2(2) + 1$ (ver capítulo 5).

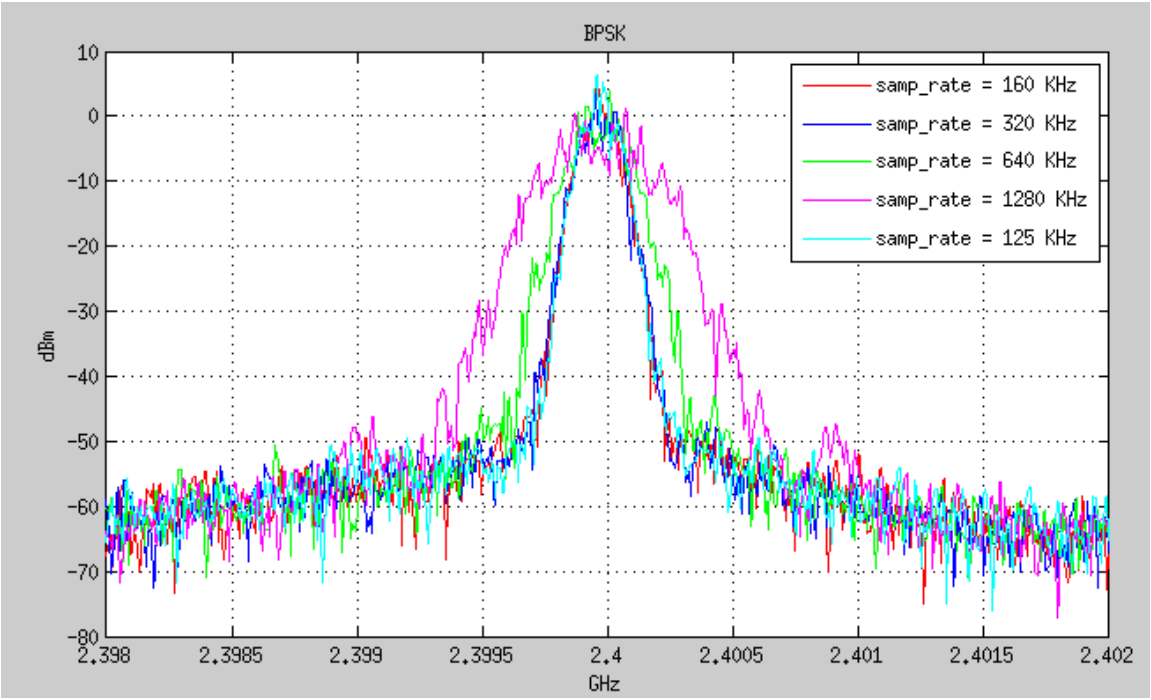
Figura 37. Comparación entre los espectros de frecuencias de las modulaciones BPSK y 4-PSK.



Como se muestra en la Figura 38, la tasa de muestras utilizada (variable *samp_rate*) en el flujograma determina el ancho de banda (a su vez la velocidad de transmisión) de la señal paso banda transmitida.

Existen límites en cuanto a la tasa de muestreo que puede ser utilizada en los flujogramas cuando se usan dispositivos USRP1. Los valores que son asignados a la variable *samp_rate* deben ser factores enteros de las frecuencias de operación del ADC Y DAC (64 MS/s y 128 MS/s respectivamente), esto es *fs de ADC o DAC = tasa de muestras * factor_de interpolado o diezmado*. Como se explica en el apéndice B de [1], el factor de interpolado o diezmado (en transmisión y recepción respectivamente) toma un valor entre 8 y 512.

Figura 38. Espectros de frecuencias para una modulación BPSK con diferentes tasas de muestreo.



6.3 Prueba 3. PROBABILIDAD DE ERROR DE BITS (SIMULACIÓN)

Esta prueba consiste en realizar pruebas simuladas de las modulaciones digitales propuestas en los objetivos variando la relación Eb/N0 y obteniendo una probabilidad de error de transmisión de bits de cada una de ellas, con el fin de presentar una probabilidad de error de bits vs Eb/N0.

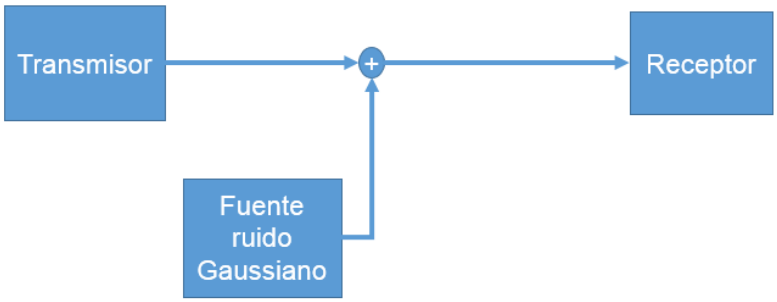
Las simulaciones fueron realizadas usando las librerías del software GNU Radio empleando programación en Python.

Hasta el momento los sistemas propuestos en este proyecto han sido construidos en la herramienta gráfica GRC del software GNU Radio, por la facilidad que proporciona dicha interfaz gráfica. Sin embargo, para estas pruebas se requiere variar la relación Eb/N0 y no es practico ejecutar la aplicación en GRC y anotar los resultados cada vez que se varía este parámetro. Es por esta razón que se opta realizar las pruebas programando directamente en Python y no en la herramienta grafica de GNU Radio (GRC) pues allí es posible programarlas de manera que se

obtengan las probabilidades de error de bits a diferentes valores de Eb/N0 con ejecutar solo una vez un archivo .py.

Una parte del código consiste en los esquemas presentados en Figura 8 y Figura 9 del capítulo 4 junto con un sumador y una fuente de ruido Gaussiano (clase denominada *tx_rx*) como se muestra en la Figura 39, siendo éste el esquema principal que realiza la modulación y demodulación de la señal. En general, el esquema de la Figura 39 es el usado en los flujogramas de Figura 14, Figura 15 y Figura 16 de las simulaciones del capítulo 5, en donde los bloques *transmisor* y *receptor* constan de los componentes expuestos en los anexos A3 y A4 usados de la misma forma que en la sección 4.4.1 y 4.4.2 respectivamente. De la clase *tx_rx* en el código se extraen los vectores que contienen los datos de la fuente del mensaje y los datos recuperados en el receptor para luego ser comparados por la función *biterror*.

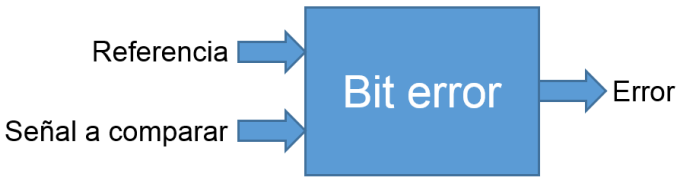
Figura 39. Esquema tx_rx utilizado en simulación



Una segunda parte del código (función denominada *biterror*) es la que realiza la comparación de las señales binarias para hallar el error entre la señal referencia y la señal recuperada en el receptor (Figura 40).

Debido a los retrasos que introducen algunos bloques en la clase *tx_rx*, es necesario encontrar el retraso y truncar adecuadamente las señales a comparar antes de aplicar la función *biterror*.

Figura 40. Representación de la función *biterror*



El bloque *fuente ruido gaussiano* en la Figura 39 es implementado mediante el bloque de GNU Radio *noise_source,* en el cual se selecciona el tipo de ruido, la amplitud y la semilla. Se construye de la siguiente forma:

```
analog.noise_source_c(tipo, amplitud, semilla)
```

en donde

- `tipo`: indica la distribución aleatoria a utilizar. Se selecciona entre ruido uniforme, gaussiano, Laplaciano o impulsivo.
- `amplitud`: representa un factor de escala para la salida; para la fuente gaussiana, este es la desviación estándar.
- `semilla`: semilla para generadores aleatorios. Note que para las distribuciones uniforme y gaussiana, este número puede ser negativo.

El parámetro `amplitud` (σ) del bloque *noise_source* es una aproximación hallada dependiendo de la relación Eb/N0 estipulada para la simulación y la energía de símbolo (E_s) resultante de los puntos de constelación usados en cada modulación:

$$\sigma^2 = \frac{E_s}{2 * (\log_2 M) * \ Eb/N0}$$

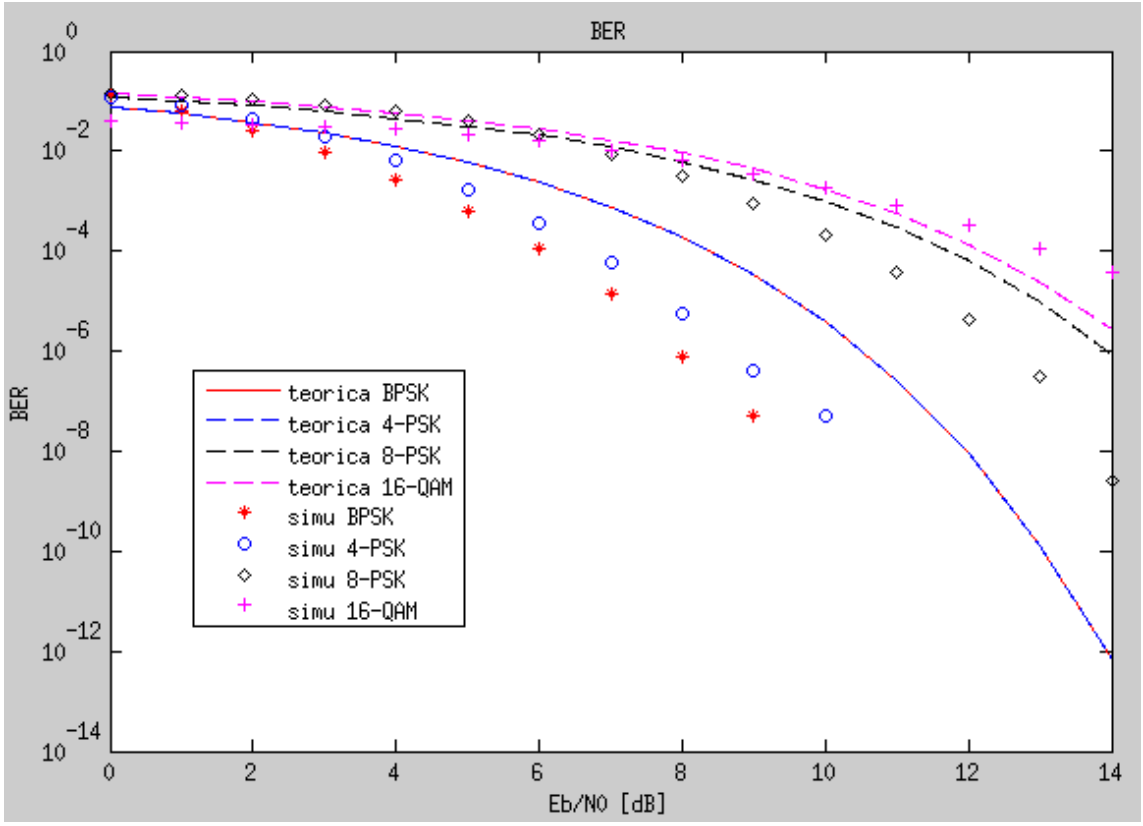
donde:

- E_s: aproximación de energía de símbolo; $E_s = \sum_{i=1}^{M} \frac{|\boldsymbol{p_i}|^2}{M}$
- M: número de niveles en la modulación.
- Eb/N0: energía por bit por densidad espectral de potencia de ruido

Teniendo en cuenta los posibles errores en la aproximación a la hora de encontrar un valor para el parámetro *amplitud* del bloque *noise_source* partiendo de las variaciones de la relación Eb/N0, en la Figura 41 se observan las curvas de BER de las simulaciones de modulaciones digitales trabajadas en el presente proyecto junto con sus respectivas contrapartes teóricas. Estas curvas fueron tomadas con un promedio de veinte pruebas con una transmisión de 10^7 datos binarios.

La curva de BER de la modulación 16-QAM se ajusta según la relación entre la energía de símbolo de la constelación utilizada en la simulación y la energía de símbolo de la constelación teórica debido a que la ecuación para estimar la probabilidad de error de bit teórica utilizada para contrastar teórico-simulación corresponde a una constelación 16-QAM cuya energía de símbolo es mayor a la utilizada en simulación, razón por la cual presentaba un mejor rendimiento del sistema teórico.

Figura 41. Curvas BER para modulaciones BPSK, 4-PSK, 8-PSK y 16-QAM en GNU Radio



7. CONCLUSIONES

El desarrollo de este trabajo de grado constituye una continuación de la investigación iniciada con el trabajo de grado [16], pues, a pesar de los años que se lleva avanzando e implementando la tecnología SDR a nivel mundial, en la Universidad Industrial de Santander es poca la investigación que se ha realizado en este tema. El informe resultado de este proyecto constituye un material de transferencia de las experiencias adquiridas en el desarrollo del mismo, proporcionando herramientas para posteriores trabajos de investigación en la tecnología SDR basada en GNU Radio, en un área de permanente desarrollo como lo son las tecnologías de comunicaciones digitales.

La programación en GNU Radio (GRC) de sistemas de comunicaciones digitales (p. ej. Modulaciones M-PSK, M-QAM, DPSK, etc.) resulta sencilla, ya que el software cuenta con librerías que contienen bloques de procesamiento de señal que integran las funciones necesarias para llevar a cabo estos procesos haciendo uso de un entorno gráfico. Una ventaja de estos bloques es la facilidad que brinda su uso. Sin embargo, para una persona interesada en profundizar en el tema, trabajar sólo a nivel de un entorno gráfico como este, esta facilidad también se puede ver como una limitante ya que los bloques se ven como cajas negras a las que entran datos y salen otros procesados de él. Así, se conoce a grandes rasgos lo que se realiza, pero no cómo se procesan esos datos al interior del bloque. El estudio de la estructura interna de estos bloques ayuda a entender más a fondo el funcionamiento del software GNU Radio en general, y lo que significa trabajar con una tecnología tan innovadora como SDR, siendo posible llegar a desarrollar o crear nuevos bloques de procesamiento compatibles con los existentes y de alguna manera contribuir en el desarrollo de esta tecnología.

Este proyecto se basó mayormente en el desglose de bloques que realizan una modulación y demodulación determinada (específicamente modulaciones M-PSK y M-QAM) permitiendo así el análisis del flujo de datos, características de las señales durante todo el proceso, y en general, el funcionamiento del bloque.

El diseño propuesto en este proyecto permite cambiar el tipo de modulación a implementar en el mismo sistema (flujograma), modificando sólo una variable (*constelacion*) que contiene la información necesaria para definir lo puntos de constelación de la modulación y el método para pasar de muestras complejas a muestras codificadas (en la demodulación) entre otras características. Usando el módulo *digital.constellation_calcdist*, el sistema permite construir cualquier otra topología de constelación diseñada con anterioridad (siendo necesario sólo la

distribución de los puntos de constelación) que no se encuentre entre las opciones prediseñadas en GNU Radio. Estas características son muy útiles pues es uno de los puntos clave a la hora de utilizar la tecnológica SDR (facilidad en la reconfiguración de los sistemas), y, para pasar de simulaciones (software) a transmisiones reales (software + hardware) sólo se requiere contar con el hardware necesario (USRP + tarjeta secundaria).

La variable *constelacion,* en donde se crea el objeto de constelación correspondiente a una modulación específica. Además de contener los puntos de constelación (*tabla de símbolos*), los bits por símbolo, el número de niveles y el método usado para la toma de decisiones en la demodulación, también puede codificar (con código gray y/o codificación diferencial) los datos, de manera que no es necesario en algunos casos (según la clase de objeto de constelación que se utilice) una codificación gray o diferencial externa.

El sistema funciona de tal forma que los bloques en sí no tienen que saber cuántos bits trabajan, esos límites son establecidos por la variable *bits*, las dimensiones y el contenido del *vector de mapeo* y la *tabla de símbolos* (puntos de constelación). Aumentar el valor de la variable *bits* a uno que no corresponda afecta de tal manera que no se recuperan los datos en el receptor por errores en la decodificación de éstos. Por otra parte, si se disminuye el valor de *bits* y no se usa codificación diferencial, no se usaría el total de ancho de banda disponible al no utilizar la totalidad de los puntos de constelación.

El bloque *chunk to symbols*, responsable de la conversión de símbolos codificados a símbolos complejos, espera que el valor numérico de los datos de entrada no excedan un máximo, de no cumplirse esta condición anterior, no es posible ejecutar el archivo grc. Este máximo es establecido por el máximo de la longitud de la tabla de símbolos (número de puntos en la constelación) menos la unidad (*logitud_tabla_simbolos - 1*).

Quitar el bloque de codificación diferencial dejaría al sistema usando datos codificados únicamente con codificación gray, resultando así en un menor rendimiento en el sistema. Esto es debido a que la codificación diferencial proporciona un método más robusto al codificar pues representa los datos por los cambios que se producen (usa símbolos actuales y anteriores para la codificación) mas no por los niveles que establecen. Si M corresponde a la longitud de la tabla de símbolos (*constelacion.arity()*[6]), este bloque asegura que se cumpla la

[6] Función que devuelve el número de niveles en la variable *constelacion*.

condición presente en el bloque *chunk to symbols* pues el módulo (M) en la ecuación $y[0] = (x[0] + y[-1])\%M$[7] garantiza que los valores de los datos de salida no excedan un máximo de M – 1. En caso de que M sea menor a la longitud de la tabla de símbolos, el sistema sigue funcionando, solo que no utilizaría la capacidad total de ancho de banda al no usar algunas representaciones de los puntos de constelación, en caso contrario, si M es mayor se presenta un error en el flujograma.

Por otra parte, para un sistema real (software + hardware), en el que se tienen limitaciones de ancho de banda y velocidades de transmisión, es indispensable agregar una manera con controlar estos parámetros de transmisión. Esta función la cumple el bloque *polyphase arbitrary resampler*, responsable de la adecuación de la señal a transmitir a las condiciones del canal (que en este caso son las condiciones de la USRP utilizada). Debido a la adición del bloque *polyphase arbitrary resampler* en el transmisor, y a la posible interferencia intersímbolica (ISI por sus siglas en inglés) se anexa el bloque *polyphase clock sync* en el receptor para mitigar estos efectos, encargándose de la sincronización en tiempo entre el transmisor y receptor, deshaciendo los efectos del filtro *RRC* y encontrando el mejor tiempo para muestrear las señales entrantes [17].

Un aspecto importante es que el valor numérico de 1.0 de una señal entrante en el bloque *UHD: USRP Sink* satura completamente el convertidor digital-analógico. Cualquier valor más grande que la unidad causa un truncamiento en la señal, es decir, que la amplitud de la señal es recortada. Por lo tanto, se debe atenuar la señal en el software antes del bloque *UHD: USRP Sink* para no incurrir en esta limitante.

En el cálculo de las curvas de BER se tienen restricciones en cuanto al número de datos con los que se pueden realizar las pruebas debido al alto consumo de recursos que requiere del PC, y las limitaciones en este sentido del equipo en el que se realizaron las pruebas. Esto resulta en una limitante para los resultados de las pruebas pues en este caso no es posible obtener un error menor a 10^{-7} si éstas se realizan con solo 10^7 datos en una sola prueba. El tiempo de simulación o el número de datos a transmitir es un factor que afecta los resultados de las curvas de BER, pues puede que en un lapso de tiempo de ejecución se presente un error y mucho tiempo después fuera del número de datos estipulados para la simulación se presente otro.

[7] Ecuación utilizada por el bloque *differential encoder* para realizar la codificación diferencial.

Analizando la Figura 41 de la prueba 3 en el capítulo 5, se concluye que el sistema se comporta de manera esperada, pues sigue en gran parte los patrones que se observan en las gráficas teóricas. De igual manera se observa que la posibilidad de error en la transmisión en las simulaciones aumenta a medida que la densidad de puntos en la constelación aumenta. Estas diferencias entre las gráficas teóricas y las simuladas presentes en la Figura 41 pueden originarse en la manera en la que se realiza la demodulación, es decir, la manera en la que se toman las decisiones para pasar de los símbolos complejos a los codificados, o al hecho que las pruebas están limitadas a un número de datos y tiempo de ejecución.

Entre la curva de BER de la modulación 16-QAM simulada y teórica se exhibió una diferencia notable, presentándose una probabilidad de error mayor en la gráfica simulada. Esta diferencia es debida a que la ecuación para estimar la probabilidad de error de bit teórica utilizada para contrastar teórico-simulación corresponde a una constelación 16-QAM cuya energía de símbolo es mayor (posee menor densidad en la distribución de puntos al estar más espaciados) a la utilizada en simulación, razón por la cual se presentaba un mejor rendimiento en el grafica de la curva teórica. La grafica simulada es ajustada según la relación entre la energía de símbolo de la constelación utilizada en la simulación y la energía de símbolo de la constelación teórica. Se observa que, una vez ajustada la gráfica BER de la modulación 16-QAM en la Figura 41 se presenta un comportamiento aceptable.

8. RECOMENDACIONES Y TRABAJOS FUTURO

Con el trabajo desarrollado hasta este punto, se propone continuar o complementar haciendo pruebas para medir rendimiento de relación de señal a ruido en transmisiones reales entre dispositivos USRP y ensayos de multiacceso. Pruebas de medición de rangos de propagación con los equipos disponibles en laboratorio (USRP1, USRP N210, WBX, RFX2400, antenas) para así determinar la cobertura de transmisión e identificar posibles factores externos que afecten el rendimiento de las transmisiones.

El desarrollo de sistemas en GRC es intuitivo y relativamente sencillo pues por medio de una interfaz gráfica se realizan las conexiones pertinentes, terminando en un diagrama de flujo de los datos. Para realizar aplicaciones en GRC, se debe tener al menos idea del funcionamiento de los bloques que se requieren en la aplicación, y posteriormente se procede a realizar las conexiones en GRC. Los bloques de GNU Radio son creados usando programación en C++ pero la conexión de cada uno de los bloques se realiza en lenguaje de programación Python, que resulta más amigable para este propósito. Al ejecutar la aplicación GRC del diseño propuesto, se genera un archivo .py que recopila información de variables y conexiones de bloques así como la GUI a utilizar para presentar los datos según como se dispuso en la aplicación GRC. Se recomienda ahondar en la creación de sistemas desarrollados directamente sobre el lenguaje de programación Python pues éste resulta más versátil que su contraparte gráfica ya que es posible complementar la conexión de los bloques con programación adicional que en GRC resulta complicada o imposible de realizar.

Se propone el desarrollo de una plataforma en *Moodle* con temas básicos en la utilización de GNU Radio y la tecnología SDR, con el fin de ser utilizada por estudiantes en el curso de comunicaciones digitales de manera que se pueda compartir el conocimiento existente en la UIS acerca de este tema, así como también una forma de evaluar competencias en torno a ellos.

Con el transcurso del tiempo, desarrolladores crean nuevos bloques de procesamiento de señal en GNU Radio actualizándose así a las nuevas tecnologías de comunicación existentes en hardware. Se sugiere indagar en la utilización de estas nuevas librerías que surgen para actualizar los conocimientos sobre las aplicaciones que pueden ser realizadas en torno a la tecnología SDR y GNU Radio junto con los nuevos dispositivos adquiridos (USRP-2920 producto de NI, o USRP N219 y tarjeta secundaria WBX en Ettus Reseach) por la Universidad Industrial de Santander.

Adicional se propone la implementación de un sistema de capas que permita conectar un sistema desarrollado o construido usando tecnología SDR (transmisor) junto con un sistema de comunicación real (receptor), haciendo especial énfasis en el acople de los protocolos y estándares usados en el receptor.

Se aconseja a los nuevos usuarios que desean realizar la instalación de GNU Radio en Ubuntu o Fedora, utilizar la opción (6) expuesta en el anexo D. Adicional a lo anterior, se recomienda que al instalar GNU Radio con la opción mencionada anteriormente no se haya intentado instalar GNU Radio con otra de las opciones posibles, pues esto puede crear conflicto en los pasos que realiza el script *build-gnuradio* resultando en una instalación fallida de GNU Radio + UHD.

Debido a que desde el 2010 Ettus Research pasó a ser una compañía filial de National Instruments, se propone realizar un estudio del uso de las herramientas proporcionadas por NI en el software LabVIEW para trabajar junto con las USRP en la tecnología SDR.

CITAS BIBLIOGRÁFICAS

[1] J. Murillo y I. Pinar, Laboratorio de comunicaciones digitales radio definida por software, Universidad de Sevilla: Departamento teoria de señal y comunicaciones, 2011.

[2] T. Rondeau, «UCLA SDR Materials,» 31 Agosto 2014. [En línea]. Disponible en: http://www.trondeau.com/examples/. [Último acceso: 14 Diciembre 2014].

[3] «Ettus Research,» [En línea]. Disponible en: http://files.ettus.com/. [Último acceso: 5 Enero 2015].

[4] B. Seeber, GNU Radio Tutorials Labs 1 – 5, Ettus Research, 18 Abril 2014.

[5] F. Ajmal, M. Ali, S. Ali, S. Islam, U. Nasir y A. Rashdi, «Enhancing Software Defined Radio (SDR) security using variations in Gray Coded Mapping Scheme in Rectangular QAM,» *IEEE,* pp. 1-4, 2009.

[6] J. P. Montero Hidalgo, «Implementación de un sistema de comunicaciones basado en Software Radio,» Universidad Atónoma de Madrid - Escuela Politécnica Superior, 2014.

[7] S. Curtis, D. Nowak, J. Pirog y E. Thompson, «SDR Repeater for the Amateur Radio Service,» Tutor: McNair Bruce, 2012.

[8] D. T. Nguyen, «Implementation of OFDM systems using GNU Radio and USRP,» Universidad de Wollongong, 2013.

[9] Ettus Research, «USRP1,» [En línea]. Disponible en: www.ettus.com/product/details/USRPPKG. [Último acceso: 24 Enero 2015].

[10] Ettus Research, «USRP Hardware Driver™ software (UHD™),» [En línea]. Disponible en: code.ettus.com/redmine/ettus/projects/uhd/wiki. [Último acceso: 24 Enero 2015].

[11] ALTERA, «Low-Cost Cyclone FPGAs,» [En línea]. Disponible en: www.altera.com/devices/fpga/cyclone/cyc-index.jsp. [Último acceso: 15 Diciembre 2014].

[12] Ettus Research, «Application Note Frontends, Sub-Device Specifications, and Antenna Port Selection».

[13] Ettus Research, «RFX2400 2,3-2,9 GHz Rx/Tx,» [En línea]. Disponible en: www.ettus.com/product/details/RFX2400. [Último acceso: 14 Diciembre 2014].

[14] Ettus Research, «Application Note Examples Provided with the USRP Hardware Driver».

[15] GNU Radio, The free & open software radio ecosystem, «GNU Radio Manual and C++ API Reference 3.7.6,» [En línea]. Disponible en: gnuradio.org/doc/doxygen/. [Último acceso: 26 Enero 2015].

[16] J. Suárez y V. Triana, «Diseño del manual de prácticas para un laboratorio de comunicaciones digitales basado en la técnica de radio definido por software,» Universidad Industrial de Santander, Facultad de ingenierías fisicomecánicas, Escuela de Ingeniería eléctrica, electrónica y telecomunicaciones, 2012.

[17] GNU Radio, The free & open software radio ecosystem, «Tutorial: PSK Symbol Recovery,» [En línea]. Disponible en: gnuradio.org/redmine/projects/gnuradio/wiki/Guided_Tutorial_PSK_Demodulation. [Último acceso: 27 Febrero 2015].

[18] QT Project, «Hardware Acceleration and Embedded Platforms,» [En línea]. Disponible en: qt-project.org/doc/qt-4.8/hwacc-rendering.html. [Último acceso: 25 Noviembre 2014].

[19] J. Knows, «GNU Radio WXGUI Signal Analysis Tools,» [En línea]. Disponible en: www.joshknows.com/wxgui. [Último acceso: 25 Noviembre 2014].

[20] GNU Radio, The free & open software radio ecosystem, «GNU Radio Companion,» [En línea]. Disponible en: gnuradio.org/redmine/projects/gnuradio/wiki/GNURadioCompanion. [Último acceso: 14 Noviembre 2014].

[21] GNU Radio, The free & open software radio ecosystem, «GNU Radio 3.7.6 documentation,» [En línea]. Disponible en: gnuradio.org/doc/sphinx/. [Último acceso: 26 Enero 2015].

[22] discuss-gnuradio, «Re: [Discuss-gnuradio] What do Packet Encoder and Packet Decoder do?,» [En línea]. Disponible en: lists.gnu.org/archive/html/discuss-gnuradio/2014-07/msg00537.html. [Último acceso: 25 Enero 2015].

[23] R. González , «Python para todos».

[24] discuss-gnuradio, «Re: [Discuss-gnuradio] FSK modulation,» [En línea]. Disponible en: lists.gnu.org/archive/html/discuss-gnuradio/2012-09/msg00121.html. [Último acceso: 25 Enero 2015].

[25] D. Manolakis y J. Proakis, Tratamiento digital de señales, Principios, algoritmos y aplicaciones, Terceda ed., Madrid: Prentice Hall, 1998. ISBN: 84-8322-000-8.

[26] F. Harris, Multirate Signal Processing for Communication Systems, Primera ed., 2004.

[27] Ruby-Forum, «PFB Arbitrary Resampler,» [En línea]. Disponible en: www.ruby-forum.com/topic/4417690. [Último acceso: 25 Enero 2015].

[28] discuss-gnuradio, «Re: [Discuss-gnuradio] AWAL root raised cosine filter in the GRC trunk?,» [En línea]. Disponible en: lists.gnu.org/archive/html/discuss-gnuradio/2008-12/msg00195.html. [Último acceso: 23 Enero 2015].

[29] GNU Radio, The free & open software radio ecosystem, «Control Loop Gain Values,» [En línea]. Disponible en: gnuradio.squarespace.com/blog/2011/8/13/control-loop-gain-values.html. [Último acceso: 25 Enero 2015].

[30] F. Harris y M. Rice, «Multirate Digital Filters for Symbol Timing Synchronization in Software Defined Radios,» IEEE Selected Areas in Communications, Vol. 19, 2001.

[31] J. Feigin, «Practical Costas loop design: Designing a simple and inexpensive BPSK Costas loop carrier recovery circuit,» *RF signal processing,* pp. 20-36, 2002.

[32] E. Hagemann, «Thes costas loop-an introduction,» 2001.

BIBLIOGRAFIA

Ettus Research, Application Note Examples Provided with the USRP Hardware Driver [En línea]. [Citado 15 Diciembre 2014].

Ettus Research, RFX2400 2,3-2,9 GHz Rx/Tx [En línea]. [Citado 14 de Diciembre de 2014]. Disponible desde Intenet: <www.ettus.com/product/details/RFX2400>.

Ettus Research USRP Hardware Driver™ software (UHD™) [En línea]. [Citado 24 de Enero de 2015]. Disponible desde Internet: <code.ettus.com/redmine/ettus/projects/uhd/wiki>.

Ettus Research, USRP1 [En línea]. [Citado 24 de Enero de 2015]. Disponible desde Internet: <www.ettus.com/product/details/USRPPKG>.

GNU Radio, The free & open software radio ecosystem. GNU Radio 3.7.6 documentation [En línea]. [Citado: 26 Enero 2015]. Disponible desde Internet: <gnuradio.org/doc/sphinx/>.

GNU Radio, The free & open software radio ecosystem. GNU Radio Companion [En línea]. [Citado 14 de Noviembre de 2014]. Disponible desde Internet: <gnuradio.org/redmine/projects/gnuradio/wiki/GNURadioCompanion>.

GNU Radio, The free & open software radio ecosystem. GNU Radio Manual and C++ API Reference 3.7.6 [En línea]. [Citado 26 de Enero de 2015]. Disponible desde internet: <gnuradio.org/doc/doxygen/>.

MANOLAKIS, Dimitris y PROAKIS, John. Tratamiento digital de señales, Principios, algoritmos y aplicaciones, 3ed. Madrid : Prentice Hall, 1998.

MURILLO, Juan y PINAR, Ivan. Laboratorio de comunicaciones digitales radio definida por software. España: Universidad de Sevilla. Departamento teoria de señal y comunicaciones, 2011. 166 p.

RONDEAU, Tom. UCLA SDR Materials [En línea]. [Citado 31 de Agosto de 2014]. 14 de Diciembre de 2014. Disponible desde Internet: <http://www.trondeau.com/examples/>.

SEEBER, Balint. GNU Radio Tutorials Labs 1 – 5, Ettus Research [En línea]. [Citado 18 Abril 2014].

SUÁREZ, Javier y TRIANA, Vivian. Diseño del manual de prácticas para un laboratorio de comunicaciones digitales basado en la técnica de radio definido por software. Trabajo de grado. Santander, Colombia: Universidad Industrial de Santander. Facultad de ingenierías fisicomecánicas. Escuela de Ingeniería eléctrica, electrónica y telecomunicaciones, 2012.

ANEXOS

ANEXO A

DOCUMENTACIÓN HERRAMIENTAS Y BLOQUES DE GNU RADIO

A1. INTERFACES GRAFICAS DE USUARIO DE GNU RADIO COMPANION

GNU Radio Companion cuenta con dos interfaces gráficas de usuario: *gr-qtgui* y *gr-wxgui,* que se describen en la Tabla 15. Estas interfaces permiten adicionar sumideros, variables (de rango, cajas de texto, botones y menús) y herramientas para organizar la GUI.

En el caso de WX GUI, se identificó que con el uso de sumideros de frecuencia, esta clase de interfaz presenta problemas en tiempo de ejecución. Por lo anterior, y debido a que sólo es posible usar una clase de interfaz en un mismo flujograma, se optó por emplear QT GUI, con la cual no se presenta ese problema. Una posible razón por la que la interfaz QT responde mejor que WX puede encontrarse en el hecho que la primera usa aceleración de hardware [18], mientras que WXGUI no.

Tabla 15. Descripción de QT-GUI y WX-GUI

GUI	Descripción
QT-GUI	Este módulo provee clases para aplicaciones graficas escritas con QT. QT es un marco de desarrollo multiplataforma con herramientas diseñadas con el propósito de facilitar la creación de aplicaciones nativas e interfaces de usuario para escritorio, plataformas móviles y más. Usa lenguajes como C++, QML y CSS & JavaScript entre otros.
wx-GUI	wxGUI es una interfaz gráfica de usuario creada por GRASS GIS escrita en Pyhton que usa la librería *wxPython. wxPython* es un conjunto de herramientas multiplataforma de GUI que utiliza la librería *wxWidgets* (escrita en C++) para el lenguaje de programación en Pyhton. Se implementa como un módulo de extensión de Python (código nativo), permitiendo a desarrolladores crear programas con GUI de manera simple y fácil. Los sumideros *OpenGL* reemplazan en GRC a los sumideros originales *wxGUI* los cuales no cuentan con aceleración de hardware [19]. Para usar la versión de OpenGL de los sumideros gráficos ver el documento README.gl

Para seleccionar el tipo de interfaz a utilizar se ingresa a las propiedades del bloque *Options* en GRC (Figura 42) y se selecciona la opción *Generate Options*. Allí se encuentran las opciones que se presentan Tabla 16.

Figura 42. Propiedades del bloque *options* en GRC

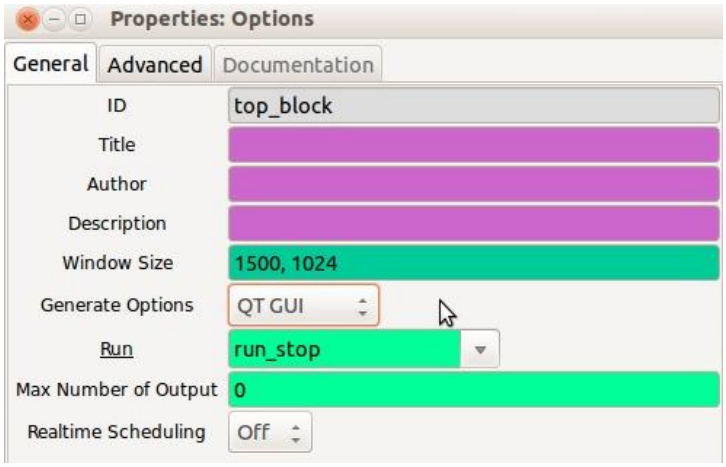


Tabla 16. Opciones para generar datos

"Generate options"	**Descripción**
XW GUI	Aplicación GUI que utiliza el conjunto de herramientas WX
QT GUI	Aplicación GUI que utiliza el conjunto de herramientas QT
No GUI	Aplicación que utiliza línea de comando en una consola sin un GUI
Hier Block (Hierarchical Blocks)	Crea bloques jerárquicos fuera de los bloques incorporados en un flujograma, pudiéndose instanciar éstos dentro de otros. Se utilizan cuatro bloques para la creación de un bloque jerarquico: el *"Options block", "pad source"* y *"pad sink".* Más información en sección "Hierarchical Blocks" en [20]

A1.1 ORGANIZACIÓN DE SUMIDEROS Y WIDGETS EN QT

Como se muestra en las figuras del capítulo 5, en el desarrollo del flujograma del presente trabajo se utilizan varios sumideros de señal como el de tiempo, frecuencia y constelación, y debido a esto se requiere organizar éstos para presentar la información. A continuación se hace una breve explicación de cómo se organizan los sumideros de señal o controles de variables (Widgets) en QT.

Todas las herramientas (sumideros de señal y Widgets) en la interfaz gráfica de QT en sus propiedades poseen una opción llamada *GUI Hint*, la cual se usa para organizar o posicionar los sumideros o Widgets dentro de la aplicación, siendo estas opcionales. Se escribe:

id_pestaña@indice_pestaña:fila,columna,duracion_fila,duracion_columna,

en donde:

id_pestaña@indice_pestaña es la especificación de la pestaña tal que *id_pestaña* es un identificador de texto escogido por el usuario, e *indice_pestaña* es un identificador numérico entre 0 y 4.

fila y columna indican la posición en la cuadrícula.

duracion_fila y duracion_columna indican cuantas posiciones ocupa un elemento en la cuadrícula.

El bloque *QT GUI Tab Widget* se utiliza para crear las diferentes pestañas en las que se van a situar el conjunto de herramientas QT GUI utilizadas en el flujograma. Específicamente el ID[8] de este bloque se puede usar en el *GUI Hint* de otros *QT GUI Tab Widget* para crear sub-pestañas, con el fin de organizar gráficos pertenecientes a una misma señal (p. ej. gráficos de tiempo y frecuencia) dentro de una pestaña.

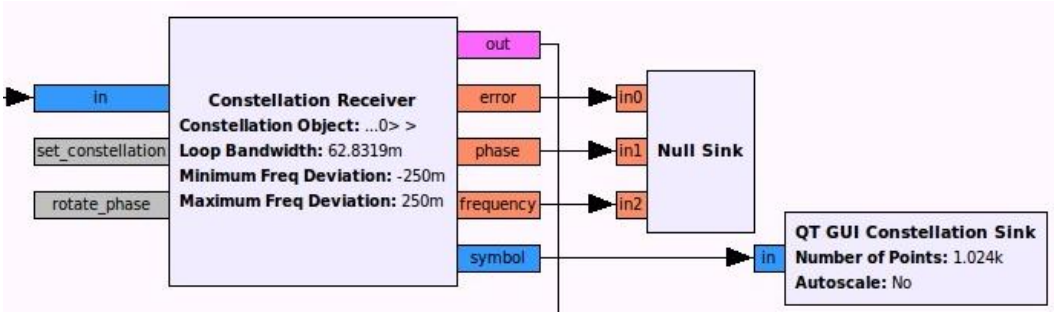
[8] El ID es un nombre único que identifica un bloque específico dentro del flujograma.

A1.2 SUMIDERO NULO

El bloque de sumidero nulo (*null_sink*) es utilizado cuando se tiene un punto terminal (salida de un bloque) en el que no se requiere realizar alguna acción con los datos en ese punto.

Un ejemplo del empleo del bloque *null_sink* se presenta en el uso de un bloque (p. ej. el bloque *constellation_receiver*, Figura 43) que posee salidas adicionales a su salida principal (marcadas como *out*). Ya que la utilización de estas salidas adicionales es opcional debido a que provienen de puntos intermedios en el proceso que se realiza internamente en el bloque, no requieren conexión de ningún tipo. Sin embargo, cuando se utiliza una de esas salidas adicionales hacia otros procesos o simplemente para observación mediante sumideros gráficos, es necesario que a las otras se les conecte un sumidero nulo para que no presente error al ejecutar el flujograma. De manera general, cualquier bloque que posea salidas adicionales a la principal del bloque y se utiliza al menos una de esas salidas adicionales, al resto de ellas se debe poner un sumidero de señal (nulo, de tiempo, de frecuencia, etc...) para que no presente error.

Figura 43. Imagen de ejemplo para el bloque *constellation_receiver*



A2. FUENTES DE SEÑAL

A continuación se presenta una breve explicación de los bloques fuentes de señal que pueden utilizarse como fuente para generar el mensaje a transmitir, haciendo énfasis en la fuente tipo vector, la cual es utilizada en este proyecto para realizar las pruebas simuladas y prácticas pues con esta fuente se pueden conocer los datos que entran directamente al modulador y los que se espera recuperar.

También se agrega algo de información de los bloques para transmitir y recibir datos desde y hacia los USRP.

A2.1 FUENTE TIPO VECTOR

Produce un flujo de muestras basadas en un vector de entrada, que puede ser una *lista* (p. ej. [a,b,c,d]) o una *tupla* (p. ej. (a,b,c,d)) creada haciendo uso del lenguaje de programación Python. Si bien por defecto los datos del vector de entrada se ejecutan una vez, es posible también enviarlos en forma periódica hasta que el flujograma sea detenido por algún evento [15]. El número de elementos en el vector de entrada para este bloque debe estar entre 2 y 255.

De los posibles tipos de datos de salida (*complejo, flotante, entero, short* o *byte*) se selecciona el tipo *byte*, que corresponde al tipo de dato de entrada del modulador.

Este bloque se puede usar como una fuente de señal controlada en la que es posible ingresar los datos a modular en el vector de entrada del bloque. Para esto se recomienda tener en cuenta los bits necesarios para la representación de los símbolos en el modulador/demodulador, debido a que conociendo el número de bits se puede tener control sobre los datos que se van a enviar.

> **Ejemplo 7**. Se desea transmitir de forma periódica el símbolo “01” mediante un modulador 4-PSK empleando la fuente tipo vector.
>
> Si se envía en forma periódica el número uno empleando la fuente tipo vector, el flujo de bits será: 00000001,00000001,... Dado que en un sistema 4-PSK el número de bits por símbolo es 2, el modulador agrupa cada 2 bits del flujo de entrada, interpretándolo como: 00-00-00-01, 00-00-00-01,… de manera que se transmitirían dos símbolos diferentes (00 y 01) en lugar de uno solo.
> Por lo tanto, para transmitir el símbolo “01” de forma periódica, el flujo en formato de byte debe ser: 01-01-01-01, -01-01-01-01 y por lo tanto el número que se debe repetir en la fuente tipo vector es (0x85, 0x85) que corresponde en binario a (01010101, 01010101).

A2.2 GENERADORES DE ONDA, FUENTES DE AUDIO Y DE ARCHIVOS

GNU Radio cuenta con generadores de onda de diferentes formas, fuentes de diversos tipos de ruido, fuentes de audio (provenientes de un micrófono o de un archivo .wav y otros) y fuentes de señal cargadas desde archivos (véase el anexo G de [16] o [15], [21]. Cuando se usan los generadores de señal mencionados anteriormente, es recomendable utilizar los bloques *Packet encoder* y *Packet decoder* junto con el modulador o demodulador respectivamente para poder recibir la señal esperada.

El bloque *Packet encoder* realiza un encapsulamiento de los datos, de manera que usando el bloque *Packet decoder* en el receptor se pueda interpretar correctamente la información que éste recibe. Según T. Rondeau [22], el bloque *Packet encoder* recibe los datos y los envuelve en un paquete de una determinada longitud de datos útiles junto con un encabezado, código de acceso y un preámbulo. El *Packet decoder* revisa el código de acceso teniendo en cuenta un umbral de bit erróneos, de modo que cuando el código de acceso es encontrado, se lee el encabezado para obtener la longitud de los datos útiles, los extrae y por último los pasa al siguiente bloque.

Más información de los parámetros de los bloques nombrados en el párrafo anterior, véase el anexo G en [16] o el documento packet.py.

A2.3 UHD: USRP

Esta biblioteca es utilizada para transmitir y recibir datos desde y hacia los USRP respectivamente. Los parámetros que se utilizan para configurar los bloques UHD son similares, pero uno funciona para transmitir y el otro para recibir.

Se propone y recomienda revisar la información del anexo G de [16] y [4] en los que se encuentra información detallada del funcionamiento y parámetros que utilizan los bloques UHD.

- **UHD: USRP Source.** El bloque de fuente USRP produce muestras en banda base mediante el muestreo de RF en una antena seleccionada en una particular frecuencia, tasa de muestreo y ganancia.

- **UHD: USRP Sink.** El sumidero USRP transmitirá la señal en banda base.

Una descripción de la especificación del sub-dispositivo y selección de puerto de antenas en los dispositivos USRP mediante los bloques de sumidero y fuente USRP se encuentra en el Anexo B.

A3. COMPONENTES DEL MODULADOR

A continuación, se presenta una explicación de cada uno de los bloques con los que mínimamente se construye un modulador.

- **Packed to unpacked.** Los bits en los *bytes* o *shorts* del flujo de entrada son agrupados en pedazos o trozos de bits (especificado por el parámetro *bits_per_chunk*), en donde cada trozo resultante es escrito justificado a la derecha en un nuevo flujo de bytes de salida [15], [21].

Se escribe:

```
blocks.packed_to_unpacked_bb(Bits_per_chunk, Endianness_t)
```

donde:

- `Bits_per_chunk`*:* número de bits por trozo. Está limitado a un valor máximo de 8 o 16 según el tipo de dato que se trabaje, byte o short respectivamente.
- `Endianness_t`*:* define el formato en el que se almacenan los datos. Se selecciona entre `gr.GR_MSB_FIRST` o `gr.GR_LSB_FIRST`.

Los 8 o 16 bits de cada *byte* o *short* del flujo de datos de entrada son procesados, de manera que cuando *bits_per_chunk* no es potencia de dos se completa los *bits_per_chunk* con los bits del siguiente byte o short de entrada.

> **Ejemplo 8.** Se desempaqueta un flujo de datos de entrada usando el bloque *packed to unpacked* cuando *bits_per_chunk* es 2 y el tipo de datos del flujo de entrada y salida es byte.

```
Entrada = (0x33) = (00-11-00-11)

Salida  = 00000000 - 00000011 - 00000000 - 00000011
```

Ejemplo 9. Se desempaqueta un flujo de datos de entrada usando el bloque *packed to unpacked* cuando *bits_per_chunk* es 6 y el tipo de datos del flujo de entrada y salida es byte.

```
Entrada = (0x33, 0x22, 0x11) = (001100-11, 0010-0010,
00-010001)

Salida   = (00001100, 00110010, 00001000, 00010001)
```

- **Map.** Realiza un mapeo de la señal entrante a los valores del vector de mapeo. Este bloque espera que la señal de entrada posea un valor máximo de longitud del vector de mapeo menos la unidad [15], [21]. Se resume en:

$$salida[i] = vector_{mapa}\left[entrada[i]\right]$$

Se escribe:

```
digital.map_bb(vector_mapa)
```

donde:

`vector_mapa`*:* vector de números a los que se mapea.

Ejemplo 10. Se mapea mediante el bloque *Map* un vector de entrada

```
Entrada = [2, 1, 3, 0, 2, 5]
```

usando un vector de mapeo escrito en código gray

```
Vector_mapa = [0, 1, 3, 2].
```

La salida corresponde al vector

```
Salida  = [3, 1, 2, 0, 3, 5].
```

Note que cuando uno de los números en el vector de entrada (‘*5’* en este ejemplo) no cumple con el máximo esperado por el bloque, la salida mapeada corresponde a ese mismo número (‘*5’*).

- **Differential encoder.** Realiza un codificación diferencial haciendo uso de los símbolos actuales y anteriores junto con el modulo del alfabeto [15], [21].

$$y[0] = (x[0] + y[-1])\%M$$

Donde el operador % devuelve el resto de la división entre los dos operandos [23].

Se escribe:

```
digital.diff_encoder_bb(Modulus)
```

donde:

`Modulus`: modulo del alfabeto de codificación.

Ejemplo 11. Se realiza una codificación diferencial a un flujo de entrada mediante el bloque *differential encoder* con un módulo de 4.

Se toma como condición inicial igual a cero.

```
x = [0, 2, 0, 3, 0, 1, 0, 0, 0, 2, 0, 3, 0, 1, 0, 0]

y = [0, 2, 2, 1, 1, 2, 2, 2, 2, 0, 0, 3, 3, 0, 0, 0]
```

- **Chunks to symbols.** Realiza un mapeo del flujo de símbolos desempaquetados indexados a un flujo de puntos de constelación de formato flotante o complejo [15], [21].

Este bloque lleva en su argumento una tabla de símbolos *(sym_table*) compuesta por un vector de números complejos, la cual es usada para mapear el flujo de bytes de entrada a los símbolos complejos. La tabla de símbolos debe ser tan grande como se espera que sea el valor de entrada [24].

$$salida[i] = sym_{table}[entrada[i]]$$

Se escribe:

```
digital.chunks_to_symbols_bc(Symbol_table, D)
```

donde:

- `Symbol_table`*:* lista que contiene una tabla de símbolos.
- `D`*:* dimensión de la tabla.

Ejemplo 12. Se mapea a símbolos complejos mediante el bloque *Chunks to symbols* los datos contenidos en el vector de entrada

```
Entrada = [0, 2, 0, 3, 0, 1, 0, 0]
```

usando la tabla de símbolos

```
Sym_table = [0.5, 0.5j, -0.5, -0.5j].
```

La salida corresponde a los vectores

```
Salida:

Real           = [0.5, -0.5, 0.5,      0, 0.5,    0,
0.5, 0.5]

Imaginario     = [   0,     0,    0, -0.5,    0, 0.5,
0,   0]
```

En la práctica la tabla de símbolos (*symbol_table*) es un vector definido por los puntos de constelación en el objeto de constelación (variable *constelacion*) creado según la modulación.

Tenga en cuenta que la entrada de este bloque es de tipo *byte* y la salida es *compleja*. La salida compleja lleva información de la parte real e imaginaria de la señal mapeada, que en este caso representaría las componentes en fase (I) y cuadratura (Q).

- **Polyphase[9] arbitrary resampler.** Este bloque se adiciona al modulador para reducir los efectos del ISI mediante el llamado de la función *gr::filter::kernel::pfb_arb_resampler_ccf*, la cual realiza un re-muestreo arbitrario en la señal [15].

La función *gr::filter::kernel::pfb_arb_resampler_ccf* toma un flujo de señal y realiza un re-muestreo arbitrario (véase capítulo 10 en [25]), en donde la razón de la nueva frecuencia de muestreo puede ser cualquier número real r. El re-muestreo es realizado por la construcción de un determinado números de filtros (N), siendo este la razón de interpolación [15].

Al usar un interpolador lineal (véase capítulo 10 [25]) para realizar la aproximación de la razón de muestreo especificada, se presenta un error equivalente al error de cuantificación entre los dos filtros usados para los puntos de la interpolación. Tomando en cuenta lo antes descrito, es posible realizar un diseño para un nivel de ruido determinado. Más grande el valor de N, menor es la incidencia del ruido [15].

El truco del diseño de este filtro se encuentra en la forma en la que se especifican los *taps* del filtro prototipo. Los *taps* son especificados usando la razón del filtro interpolador, que en este caso es la tasa de muestreo de entrada multiplicada por el número de filtros en el banco de filtros[10] [15].

Más información del funcionamiento de este bloque se encuentra en capítulo 7.5 de [26].

Se escribe:

```
filter.pfb_arb_resampler_ccf(rate, taps, filter_size)
```

donde:

- `rate`: especifica la tasa de re-muestreo a usar.

[9] Indica que los filtros aplicados por este bloque son realizados mediante una descomposición polifásica de estos. Véase [oppenheim] para más información.

[10] Los bancos de filtros son utilizados para descomponer el espectro de frecuencias de una señal en diferentes rangos de frecuencias más estrechas (etapa de análisis) o reconstruir una señal partiendo de los rangos de frecuencias descompuestos en la etapa de análisis (etapa de síntesis). [oppenheim, proakis manolakis]

- `taps`: el filtro prototipo para llenar el banco de filtros.

- `filter_size`: el número de filtros en el banco de filtros. Este es directamente relacionado con el ruido de cuantificación introducido durante el re-muestreo.

En este caso, en el parámetro *taps* se introduce un filtro coseno alzado (RRC) para conformar el pulso. Se utiliza `firdes.root_raised_cosine(gain, sampling_freq, symbol_rate, alpha, ntaps)` para generar el filtro, y como se muestra en la sección 4.1, de manera general se configura de la siguiente manera:

```
firdes.root_raised_cosine(nfiltros, nfiltros, 1.0,
     rolloff, ntaps)
```

La ganancia del filtro es puesto en *nfiltros* para compensar el aumento de la tasa de muestreo, y la razón de muestreo en si también es puesta en *nfiltros* suponiendo que la señal de entrada se encuentra a una frecuencia de muestreo de 1,0. El valor de *nfiltros* es basado en el parámetro *filter_size* [15].

El tamaño del filtro (*filter_size*) generalmente es de 32, ya que según [15], este número brinda un equilibrio apropiado entre precisión, rendimiento y memoria, proporcionando un error más o menos equivalente al error de cuantificación usando una representación de punto fijo de 16 bits. Incrementar el tamaño de filtro otorga beneficios de precisión, pero requiere de un aumento en las demandas computacionales.

El parámetro *ntaps* en el RRC es puesto en *nfiltros*muestras*11*, definiendo el número de taps que puede procesar el filtro. Éste parámetro crea un gran número de taps, que se distribuyen en los *n* números de filtros (*nfiltros*), de manera que cada filtro tendrá *muestras*11 taps* en él [27]. *Muestras*11* es basado en cuantos números de símbolos por periodo se quiere aplicar el filtro RRC [28], y el número 11 brinda un buen rendimiento en el trabajo de este bloque [27].

A4. COMPONENTES DEL DEMODULADOR

• **FLL band-Edge.** Este bloque se encarga de realizar una corrección en frecuencia. Crea un lazo de seguimiento de frecuencia haciendo uso de filtros

limitadores de banda. Este filtro limitador de banda cubre los anchos de banda superior e inferior de una señal modulada digitalmente, cuyo rango de ancho de banda es determinado por el exceso de ancho de banda (factor de *rolloff*) de la señal modulada. La ubicación en frecuencia de los límites de banda es determina por el número de muestras por símbolo y el factor de *rolloff* [15].

Se escribe

```
digital.fll_band_edge_cc(samp_per_sym, rolloff,
     filter_size, bandwidth)
```

donde:

- `samp_per_sym`: número de muestras por símbolo.
- `rolloff`: factor de *rolloff* (exceso de ancho de banda) de la señal filtrada.
- `filter_size`: Indica el número de taps en el filtro limitador de banda. Se recomienda que este valor sea el mismo número de taps usados en el filtro conformador del transmisor (usualmente valores entre 30 y 70). Un número muy grande resulta en un gran retraso entre la entrada y la estimación de frecuencia, obteniéndose un resultado no preciso [15].
- `bandwidth`: ancho de banda del lazo.

• **Polyphase clock sync.** Este bloque realiza una sincronización de tiempo usando un banco de filtro polifásico. Trabaja mediante la creación de dos bancos de filtros; uno de ellos contiene el filtro adaptativo de la señal *(*en inglés *pulse shaping matched filter)* (como un coseno alzado), donde cada una de las ramas posee una fase diferente del filtro, pensando en tiempo, este primer banco de filtros contiene filtros que tienen la forma de *sinc*. El segundo banco de filtros contiene las derivadas de los filtros en el primer banco. La idea es alinear la señal de salida de manera que esta sea muestreada exactamente en el pico de forma *sinc* [15].

Se escribe

```
digital.pfb_clock_sync_ccf(samples_per_symbol, loop_bw, taps,
     filter_size, init_phase, max_rate_deviation, osps)
```

donde:

- `samples_per_symbol`: número de muestras por símbolo de la señal de entrada.
- `loop_bw`: indica el ancho de banda del lazo de control. Es usado para establecer la ganancia del lazo de control interno [29]. Se sugiere un valor alrededor 2pi/100.
- `taps`: taps del filtro.
- `filter_size`: números de filtros.
- `init_phase`: fase inicial del filtro, viene por defecto en 0.
- `max_rate_deviation`: valor por defecto 1.5
- `osps`: número de muestras por símbolo a la salida.

Para más información del funcionamiento de este bloque véase [30].

Uno de los beneficios de este algoritmo es que se puede poner un filtro adaptativo en los taps, de manera que en él se tenga tanto el filtro adaptativo como la muestra de corrección de tiempo. Al igual que en el bloque de *Polyphase arbitrary resampler* se utiliza un filtro coseno alzado para realizar la sincronización, con la diferencia que en el parámetro *sampling_freq* de la función *firdes.root_raised_cosine* se ingresa el valor *nfiltros*muestras*.

```
firdes.root_raised_cosine(nfiltros, nfiltros*muestras,
     1.0, rolloff, ntaps)
```

- **Constellation receiver.** Construye un receptor de constelación que sincroniza y decodifica los puntos de constelación (traduce las muestras complejas a símbolos) especificados por un objeto de constelación. La sincronización es basada en un lazo de *Costas*, el cual encuentra el error del punto de la señal entrante en comparación con su punto de constelación más cercano. La frecuencia y fase del NCO en el lazo de realimentación son actualizadas según este error [15].

Un lazo de *Costas* es un sistema que se usa para recuperar la portadora de una señal modulada basado en el principio de funcionamiento de un PLL [31], [32].

Se escribe

```
digital.constellation_receiver_cb(constellation,
    looCp_bw , fmin, fmax)
```

donde:

- `constellation`: puntos de constelación para una modulación genérica.
- `loop_bw`*:* ancho de banda del lazo del Costas Loop (~ 2pi/100).
- `fmin`*:* valor mínimo de frecuencia normalizada que el lazo puede alcanzar.
- `fmax`*:* valor máximo de frecuencia normalizada que el lazo puede alcanzar.

• **Differential decoder.** Usa símbolos actuales y anteriores, y el módulo del alfabeto para realizar la decodificación diferencial. Se toma como condición inicial igual a cero para empezar el cálculo [15], [21].

$$y[0] = (x[0] - x[-1])\%M$$

Donde el operador % devuelve el residuo de la división entre los operandos [23].

Se escribe

```
digital.diff_decoder_bb(modulus)
```

Parámetro:

`modulus`*:* módulo del alfabeto de codificación.

Ejemplo 13. Se realiza una decodificación diferencial (proceso inverso al realizado en el **Ejemplo 11**) a un flujo de entrada mediante el bloque *differential encoder* con un módulo de 4.

```
x = [0, 2, 2, 1, 1, 2, 2, 2, 2, 0, 0, 3, 3, 0, 0, 0]

y = [0, 2, 0, 3, 0, 1, 0, 0, 0, 2, 0, 3, 0, 1, 0, 0]
```

- **Map.** Este bloque funciona de la misma manera que el bloque *Map* en el modulador.

En el parámetro *vector_mapa* de este bloque se ingresa el vector de mapeo, en este caso *digital.mod_codes.invert_code(cod_gray),* el cual genera un código gray inverso partiendo del código gray usado en el modulador.

Ejemplo 14. Se desea realizar un mapeo inverso de un flujo de datos mapeados anteriormente con codificación gray (módulo de 16) usando el bloque *Map*.

En la Tabla 17 se muestra la codificación gray y su inverso para llevar a cabo el ejemplo. Para generar el código gray y su inverso se utilizan las variables cod_gray y cod_gray_inv respectivamente, descritas en la sección 4.1. La Figura 44 muestra el diagrama de flujo de los datos.

```
Cod_gray = digital.utils.gray_code.gray_code(16)
                Cod_gray_inv =
    digital.mod_codes.invert_code(cod_gray)
```

Figura 44. Flujograma del presente ejemplo.

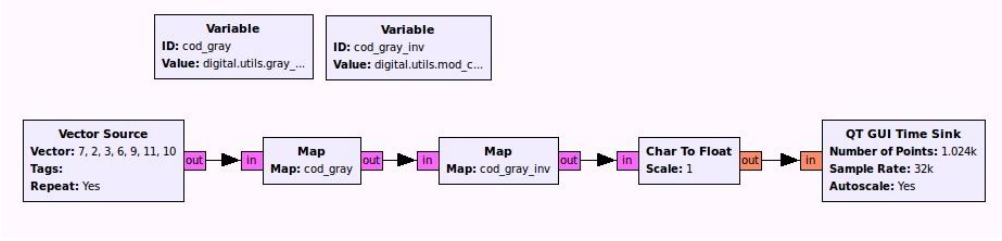


Tabla 17. Codificación y decodificación gray con representación de 4 bits.

Posición elemento	Código gray	Código gray inverso
0	0	0
1	1	1
2	3	3
3	2	2
4	6	7
5	7	6
6	5	4
7	4	5
8	12	15
9	13	14
10	15	12
11	14	13
12	10	8
13	11	9
14	9	11
15	8	10

```
Vector de entrada   = [7, 2, 3, 6,  9, 11, 10]

Vector codificado   = [4, 3, 2, 5, 13, 14, 15]

Vector decodificado = [7, 2, 3, 6,  9, 11, 10]
```

Siguiendo el ejemplo de los vectores escritos anteriormente, el primer elemento del vector de entrada es un 7, lo cual al codificarlo en el vector de codificación (código gray, Tabla 17) resulta un 4 (posición 7 del código gray). Al decodificar este 4 del vector codificado en el vector de decodificación (código gray inverso, Tabla 17) produce un 7 (posición 4 del código gray inverso) obteniéndose así el número original del primer elemento del vector de entrada.

- **Unpack k bits.** Este bloque selecciona los `K` bits menos significativos de un *byte* y los expande en `K` bytes de 0 o 1 [15], [21].

Se escribe

```
blocks.unpack_k_bits_bb(K)
```

donde:

`K`: número de bits a desempaquetar.

> **Ejemplo 15.** Se desea desempaquetar un número determinado de bits (*K* = 4) de un flujo de datos tipo *byte*, usando el bloque *unpack k bits*.
>
> ```
> Entrada = [0xf5, 0x08] = [11110101,00001000]
>
> Salida = [0, 1, 0, 1, 1, 0, 0, 0]
> ```
>
> Cada elemento del vector de salida es tipo *byte*.

- **Agc2.** Adicional a los bloques descritos anteriormente que hacen parte del demodulador, como se muestra en el archivo *generic_mod_demod.py* en la sección del demodulador genérico, es necesario incluir un control automático de ganancia debido a las fluctuaciones de amplitud de la señal recibida en la práctica (recibiendo señal desde un dispositivo USRP).

Este bloque que realiza un control automático de ganancia de alto rendimiento, cuya función es ajustar la ganancia de modo que la amplitud de la señal de salida se mantenga constante (en un valor de referencia), sin importar las variaciones de amplitud de la señal de entrada.

Se escribe:

```
gr.agc2_cc(attack_rate, decay_rate, reference, gain,
     max_gain)
```

donde [15]:

- `attack_rate`*:* tasa de actualización del lazo cuando se encuentra en modo ataque.

- `decay_rate`*:* tasa de actualización del lazo cuando se encuentra en modo de decaimiento.

- `reference`*:* valor de referencia de la potencia de señal a la cual se va a ajustar la salida.

- `gain`*:* valor inicial de ganancia.

- `max_gain`*:* valor máximo de ganancia. Se usa 0 para indicar que no existe límite.

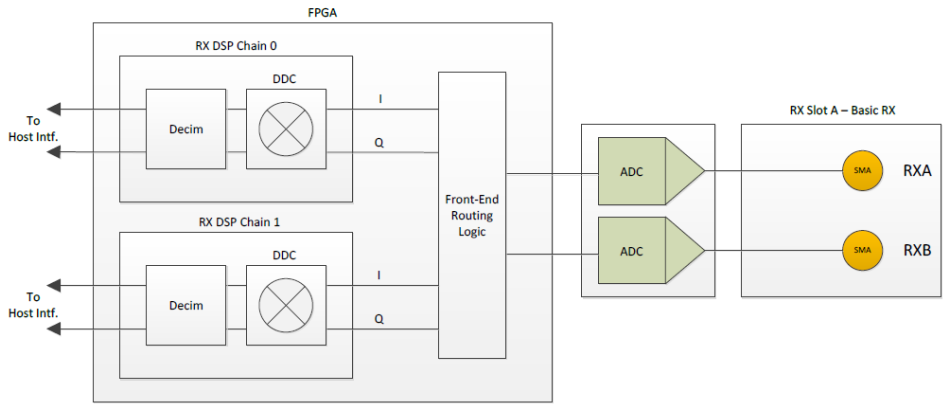
ANEXO B

FRONTEND, ESPECIFICACIÓN DE SUB-DISPOSITIVO, Y SELECCIÓN DE PUERTO DE ANTENA

Las tarjetas secundarias de Ettus Research son elementos que proveen la circuitería necesaria entre los ADCs, DACs y el mundo de salida, con la posibilidad de integrar más de una de ellas en los dispositivos USRP. Cada tarjeta secundaria incluye uno o más canales llamados *frontend*. Mediante un enrutamiento lógico dentro de la FPGA en el dispositivo USRP, se multiplexan los datos de acuerdo a la selección de un *frontend* usando una especificación de sub-dispositivo [12].

A continuación en la Figura 45 se muestra un ejemplo de la arquitectura en un canal de recepción.

Figura 45. Arquitectura de la interfaz Frontend mostrando como ejemplo Basic RX



Fuente: Application Note: Frontends, Sub-Device Specifications, and Antenna Port Selection aportado por Ettus Research

La especificación de sub-dispositivo presenta el siguiente formato “X:Y”. En donde X indica la ranura de la tarjeta secundaria. En los casos donde los dispositivos USRP que poseen solo una ranura para tarjetas secundarias, esta toma el nombre de “ranura A”. El USRP1 incluye dos ranuras, “A” Y “B”. La segunda parte de la especificación de sub-dispositivo, “Y”, selecciona un único *frontend* de la tarjeta secundaria especificada por la ranura designada [12].

Tabla 18. Tarjetas secundarias, sub-dispositivos, y conectores.

Tarjeta secundaria	Dirección	Front-end	Selección de antena válida	Conector asociado
BasicRX	RX	A	n/a	RXA
		B	n/a	RXB
		AB	n/a	RXA & RXB(Quad Intf)
		BA	n/a	RXB & RXA(Quad Intf)
BasicTX	TX	A	n/a	RXA
		B	n/a	RXB
		AB	n/a	RXA & RXB(Quad Intf)
		BA	n/a	RXB & RXA(Quad Intf)
LFRX	RX	A	n/a	RXA
		B	n/a	RXB
		AB	n/a	RXA & RXB(Quad Intf)
		BA	n/a	RXB & RXA(Quad Intf)
LFTX	TX	A	n/a	RXA
		B	n/a	RXB
		AB	n/a	RXA & RXB(Quad Intf)
		BA	n/a	RXB & RXA(Quad Intf)
TVRX2	RX	RX1	n/a	RX1
		RX2	n/a	RX2
DBSRX2	RX	0	n/a	J3
Serie RFX	TX	0	TX/RX	TX/RX
	RX	0	RX2	RX2
			TX/RX	TX/RX
WBX	TX	0	TX/RX	TX/RX
	RX	0	RX2	RX2
			TX/RX	TX/RX
SBX	TX	0	TX/RX	TX/RX
	RX	0	RX2	RX2
			TX/RX	TX/RX
XCVR2450	TX	0	J1	J1
			J2	J2
	RX	0	J1	J1
			J2	J2

Fuente: Application Note: Frontends, Sub-Device Specifications, and Antenna Port Selection aportado por Ettus Research

Específicamente en el dispositivo USRP1 que posee dos ranuras para tarjetas secundarias, las ranuras A o B se seleccionan usando A:0 para seleccionar ranura A y B:0 para seleccionar la ranura B [12].

Para más información refiérase al documento Application Note: Frontends, Sub-Device Specifications, and Antenna Port Selection aportado por Ettus Research

ANEXO C

NOTAS DE APLICACIÓN DE ETTUS RESEARCH

C1. NOTA DE APLICACIÓN: SELECCIONANDO UN DISPOSITIVO USRP

Documento en el que se especifica características generales de algunos de los dispositivos USRP en el mercado ofrecidos por Ettus Reseach, de igual manera se presenta un listado de las aplicaciones comunes en los que son utilizados, con el fin de orientar en la selección de un USRP según las necesidades requeridas.

C2.NOTA DE APLICACIÓN: SELECCIONANDO UNA TARJETA SECUNDARIA DE RF

Documento en el que se especifica características generales de algunas de las tarjetas secundarias en el mercado ofrecidas por Ettus Research, así como también las áreas de aplicación en las que se puede utilizar, con el fin de ayudar en la selección de una tarjeta secundaria que cumpla con los requerimientos necesarios para una aplicación determinada.

C3. NOTA DE APLICACIÓN: EJEMPLOS PROPORCIONADOS CON UHD

Muestra una breve explicación de algunos códigos .py de ejemplo con los que se pueden realizar pruebas de funcionamiento y revisar algunas características del USRP a utilizar, además de algunos esquemas de conexión y listado de errores comunes en las pruebas de transmisión y recepción usando los códigos ejemplos mencionado anteriormente.

C4. NOTA DE APLICACIÓN: FRONTENDS, ESPECIFICACIÓN DE SUB-DISPOSITIVOS Y SELECCIÓN DE LOS PUERTOS DE ANTENAS

Documento en el que se presenta una pequeña explicación de la definición de tarjeta secundaria, fronteds, especificación de sub-dispositivos y la selección de antenas para su correcto funcionamiento.

ANEXO D

GUÍA DE INSTALACIÓN GNU RADIO, VERIFICACIÓN DE CONEXIÓN DE DISPOSITIVO USRP, UBICACIÓN ARCHIVOS

INSTALACIÓN GNU RADIO

A continuación se presenta un listado de algunas formas en las que se puede instalar GNU Radio y GNU Radio + UHD.

1) La manera más sencilla de instalar GNU Radio es a partir del paquete de gnuradio desde los repositorios estándar. En la Tabla 19 se presentan las líneas de código para instalar GNU Radio desde los repositorios. Con este método queda instalado solo GNU Radio sin la posibilidad de utilizarlo junto a los dispositivos USRP.

Tabla 19. Líneas de código para instalación de GNU Radio

Distribución	Línea de código en terminal
Ubuntu, Devian	apt-get install gnuradio
Fedora	yum install gnuradio

2) Otra posibilidad de instalación de GNU Radio es desde los binarios proporcionados por Ettus Research disponible en files.ettus.com/binaries, en donde se encuentra para diferentes sistemas operativos. Con este método queda instalado solo GNU Radio sin la posibilidad de utilizarlo junto a los dispositivos USRP.

3) Se puede obtener un DVD de arranque con GNU Radio preinstalado. Actualmente éste CD viene con Ubuntu 14.04 y GNU Radio + UHD funcional (junto con los dispositivos USRP). Para más información visite gnuradio.org/redmine/projects/gnuradio/wiki/GNURadioLiveDVD.

4) Para la instalar GNU Radio desde el código fuente se puede seguir los pasos mencionados en web gnuradio.org/redmine/projects/gnuradio/wiki/UbuntuInstall#Install-Dependencies, o

los pasos resumidos en [16]. Al finalizar si se realizan todos los pasos de la guía de instalación correctamente, se puede tener GNU Radio + UHD, y por tanto con los dispositivos USRP.

5) La instalación de GNU Radio de forma manual en Microsoft Windows se encuentra en gnuradio.org/redmine/projects/gnuradio/wiki/WindowsInstall. En ella se encuentra algunos link de páginas web donde se presentan los pasos necesarios para instalar GNU Radio + UHD, y por tanto con los dispositivos USRP.

6) En gnuradio.org/redmine/projects/gnuradio/wiki/UbuntuInstall#Install-Dependencies recomiendan utilizar el script *build-gnuradio* debido a que este presenta un buen trabajo para la compilación e instalación de GNU Radio + UHD en Ubuntu y Fedora. Este script descarga e instala todas las dependencias necesarias, descarga la última versión disponible de los códigos fuente de UHD y GNU Radio desde *Git*, ejecuta los procesos de *make*, y por último, compila e instala en el sistema. De manera que, terminada la ejecución del script se tiene GNU Radio instalado junto con UHD para trabajar con los dispositivos USRP compatibles con UHD.

La siguiente línea de comando se utiliza para obtener el archivo *build-gnuradio* y hacerlo ejecutable.

```
wget http://www.sbrac.org/files/build-gnuradio && chmod a+x
            ./build-gnuradio && ./build-gnuradio
```

Antes de ejecutar el script build-gnuradio se debe verificar que el PC tenga conexión a internet y que tenga instalado el software *Git,* en caso de no tenerlo instalado se escribe la siguiente línea de comando en la terminal para instalarlo:

```
sudo apt-get install git
```

VERIFICAR CONEXIÓN O PROPIEDADES DEL DISPOSITIVO USRP CONECTADO AL PC.

- Descubrir dispositivo en línea de comando. El programa *uhd_find_devices* busca en el sistema dispositivos USRP conectados, e imprime una lista

enumerada con los dispositivos encontrados junto con sus seriales y/o direcciones.

- Propiedades del dispositivo. El programa *uhd_usrp_probe* proporciona las propiedades de los dispositivos USRP conectados al sistema, dando información tales como tarjetas secundarias detectadas, rangos de frecuencias, ganancias, rangos, etc.

TABLA DE UBICACIONES DE ARCHIVOS

En la Tabla 20 se presentan las ubicaciones de los archivos que se mencionan en el trabajo de grado y que son pertenecientes a las librerías de GNU Radio

Tabla 20. Tabla de ubicaciones de documentos en GNU Radio

Archivo	Ubicación
packet.py	gnuradio/grc/grc_gnuradio/blks2/packet.py
psk_constellation.py	Gnuradio/gr-digital/python/digital/psk_constellations.py
qam_constellations.py	gnuradio/gr-digital/python/digital/qam_constellations.py
constellation.h	gnuradio/digital/constellation.h
generic_mod_demod.py	gnuradio/gr-digital/python/generic_mod_demod.py
README.gl	gnuradio/gr-wxgui/ README.gl