



IMPLEMENTACIÓN DE UN SISTEMA DE COMUNICACIÓN CQAM SOBRE RADIO DEFINIDA POR SOFTWARE

Un Proyecto de Grado Para Obtener el Título De
Ingeniero Electrónico
Universidad Distrital Francisco José de Caldas

Autores del Proyecto: Ronald Alexander Prieto
Rafael Leonardo Rojas

Director del Proyecto: Iván Ladino

Codirector: Juan Carlos Gómez

Facultad de Ingeniería, Proyecto Curricular de Ingeniería Electrónica
Área de Comunicaciones

Bogotá, 2017

CONTENIDO

Introducción	9
1.3.1. Objetivo Principal	10
1.3.2. Objetivos Específicos	10
Capítulo 2	13
Marco Teórico.....	13
2.1 Representación en Espacio de Señal.....	13
2.2 Modulación por Amplitud de Cuadratura (QAM)	15
2.3 Modulación QAM de tipo Jerárquica (HQAM).....	18
2.3.1 Cálculo del BER para 16-HQAM.....	21
2.4 Conjuntos de Cantor	25
2.4.1 Conjuntos de Cantor generalizados	26
2.5 Modulación CQAM (CANTOR-QAM)	29
2.6 Tecnología SDR	31
2.6.1 Universal Software Radio Peripheral (USRP)	33
2.6.2 USRP Hardware Driver.....	34
2.7 GNU RADIO	35
2.8 Relación Señal a Ruido	40
2.9 Ruido Blanco Gaussiano Aditivo (AWGN)	40
Capítulo 3	43
Diseño e Implementación en GNU Radio.....	43
3.1 Constelación CQAM	43
3.2 Bloques propios del ambiente de GNU Radio	45
3.1.1 Tipos de bloques funcionales.....	45
3.1.2 Bloque de Opciones del Workspace	46
3.2 Fuentes de Información (Bloques: <i>File</i> , <i>Wav File</i> y <i>Signal Source</i>)	47
3.2.1 Bloque limitador (Throttle).....	48
3.3 Bloques del Modulador	48
3.3.1 Empaqueamiento de datos	50
3.3.1.1 Bloque empaquetamiento de datos (Packet Encoder).....	50
3.3.1.2 Bloque asignación de prioridad de datos (Packet to Unpacked).....	51
3.3.2 Codificación	52

3.3.2.1 Bloque de codificación en Gray (Map)	52
3.3.2.2 Bloque codificador en banda base (Chunks to Symbols)	53
3.4 Bloques del Demodulador.....	59
3.4.1 Sincronización de Frecuencia y Fase.....	60
3.4.1.1 Bloque Sincronizador de frecuencia (FFL BAND EDGE).....	60
3.4.1.2 Bloque filtro de Costas Loop	61
3.4.2 Demodulación.....	61
3.4.2.1 Bloque de demodulación por filtro adaptado (Polyphase Clock Sync).....	61
3.4.2.2 Bloque de decodificación	63
3.4.3.1 Bloque definición de la constelación recibida (Constellation Rect Object)	64
3.4.3.2 Bloque decodificador de constelación (Constellation Decoder)	65
3.4.3.3 Bloque decodificación en código Gray (Map)	65
3.4.4 Desempaquetamiento de datos	66
3.4.4.1 Bloque Unpack K bits.....	66
3.4.4.2 Bloque Packet Decoder	67
Capítulo 4	69
Implementación en Hardware SDR.....	69
4.1 USRP B210.....	69
4.2 HackRF	71
4.3 Aspectos Generales	73
4.4 Análisis de Potencia	73
4.4.1 Pruebas de transmisión	73
4.4.2 Distancia mínima entre TX y RX	75
4.4.3 Pruebas de recepción	77
Capítulo 5	79
Resultados Obtenidos.....	79
5.2 Demodulador 16-CQAM	83
5.3 Estimación del BER.....	86
5.4 Medición del BER con USRP	91
Capítulo 6	97
Conclusiones	97
Trabajo Futuro	98
BIBLIOGRAFÍA.....	99

INDICE DE FIGURAS

<i>Figura 1. Constelación de una modulación 16-CQAM. Todos sus estados se encuentran separados a la misma distancia.</i>	17
<i>Figura 2. Esquema de bloques modulador QAM</i>	17
<i>Figura 3. Esquema de bloques demodulador QAM</i>	18
<i>Figura 4. 16-HQAM con $\alpha = 2$,</i>	19
<i>Figura 5. Sistema de modulación y demodulación 16-HQAM.</i>	20
<i>Figura 6. Primer cuadrante de una constelación 16-HQAM</i>	21
<i>Figura 7. BER vs SNR para los bits de HP y LP utilizando 16-HQAM para $\alpha = 2$ a 5 y 16-QAM ($\alpha = 1$).</i>	24
<i>Figura 8. Segmentos generados en la tercera iteración del conjunto de cantor de tercios medios</i>	25
<i>Figura 9. Primeras Etapas de Conjunto de Cantor con varios parámetros f_i</i>	27
<i>Figura 10. Representación binaria del subconjunto C3. Los semicírculos que se encuentran en la mitad de cada intervalo simbolizan los centros P_n</i>	28
<i>Figura 11. Constelación CQAM formada a partir de modulaciones CPAM.</i>	29
<i>Figura 12. Diagrama Bloques Principales de SDR</i>	31
<i>Figura 13. Diagrama Bloques Principales USRP</i>	33
<i>Figura 14. Interconexión en aplicación GNU Radio</i>	35
<i>Figura 15. Arquitectura GNU Radio</i>	37
<i>Figura 16. Funciones desarrolladas en GNU Radio en la cadena del transmisor y receptor</i>	39
<i>Figura 17. Simulación del canal AWGN</i>	41
<i>Figura 18. Diseño constelación CQAM</i>	44
<i>Figura 19. Parámetros de configuración del menú: Generate Options</i>	46
<i>Figura 20. Parámetros bloque File Source, Wav File Source</i>	47
<i>Figura 21. Parámetros bloque Signal Source</i>	48
<i>Figura 22. Parámetros bloque Throttle</i>	48
<i>Figura 23. Diseño modulador 16-CQAM</i>	49
<i>Figura 24. Funcionalidad del bloque Packet Encoder en la modulación convencional</i>	50
<i>Figura 25. Parámetros bloque Packet Encoder</i>	50
<i>Figura 26. Funcionalidad del bloque Packet to Unpacked en la modulación convencional</i>	51
<i>Figura 27. Parámetros bloque Packed to Unpacked</i>	52
<i>Figura 28. Funcionalidad del bloque Map en la modulación convencional</i>	52
<i>Figura 29. Parámetros bloque Map</i>	53
<i>Figura 30. Parámetros variable gcs</i>	53
<i>Figura 31. Funcionalidad del Bloque Chunks to Symbols en la modulación convencional</i>	53
<i>Figura 32. Parámetros bloque Chunks to Symbols</i>	54
<i>Figura 33. Parámetros variable constellation</i>	54
<i>Figura 34. Funcionalidad del Bloque Polyphase Arbitrary Resampler en la modulación convencional</i>	55
<i>Figura 35. Parámetros bloque Polyphase Arbitrary Resampler</i>	56
<i>Figura 36. Parámetros filtro coseno alzado tx</i>	56
<i>Figura 37. Diseño del filtro en GNU Radio Filter Design Tool</i>	57
<i>Figura 38. Diseño del filtro en GNU Radio Filter Design Tool – Respuesta en magnitud</i>	57
<i>Figura 39. Diseño Demodulador 16-CQAM</i>	59
<i>Figura 40. Parámetros bloque FFL Band-Edge</i>	60
<i>Figura 41. Parámetros bloque Costas Loop</i>	61
<i>Figura 42. Funcionalidad del Bloque Polyphase Clock Sync en la modulación convencional</i>	61

<i>Figura 43. Parámetros bloque Polyphase Clock Sync</i>	63
<i>Figura 44. Parámetros filtro coseno alzado rx</i>	63
<i>Figura 45. Parámetros Bloque Constellation Rect. Object</i>	64
<i>Figura 46. Funcionalidad del Bloque Constellation Decoder en la modulación convencional</i>	65
<i>Figura 47. Parámetros Bloque Constellation Decoder</i>	65
<i>Figura 48. Funcionalidad del Bloque de decodificación Gray en la modulación convencional</i>	65
<i>Figura 49. Parámetros Bloque Map – Decodificación Gray</i>	66
<i>Figura 50. Funcionalidad del Bloque Unpack K bits en la modulación convencional</i>	66
<i>Figura 51. Parámetros Bloque Unpack K bits</i>	66
<i>Figura 52. Funcionalidad del Bloque Packet Decoder en la modulación convencional</i>	67
<i>Figura 53. Parámetros Bloque Packet Decoder</i>	67
<i>Figura 54. Parámetros Bloque UHD:USRP Sink</i>	70
<i>Figura 55. Parámetros Bloque osmocom Source</i>	72
<i>Figura 56. Prueba de transmisión USRP B210 @ 897Mhz</i>	74
<i>Figura 57. Regiones de campo de una antena</i>	76
<i>Figura 58. Señal de audio transmitida</i>	79
<i>Figura 59. Señal coseno transmitida</i>	80
<i>Figura 60. Constelación 16-QAM transmitida</i>	80
<i>Figura 61. Constelación 16-CQAM transmitida</i>	81
<i>Figura 62. Señal 16-QAM modulada en amplitud transmitida.</i>	81
<i>Figura 63. Señal 16-CQAM modulada en amplitud transmitida.</i>	82
<i>Figura 64. Espectro transmitido 16-QAM</i>	82
<i>Figura 65. Espectro transmitido 16-CQAM</i>	83
<i>Figura 66. Señal 16-CQAM recibida.</i>	83
<i>Figura 67. Espectro recibido 16-CQAM</i>	84
<i>Figura 68. Constelación recibida 16-CQAM</i>	84
<i>Figura 69. Señal audio recuperada</i>	85
<i>Figura 70. Señal coseno recuperada</i>	85
<i>Figura 71. Parámetros Bloque Error Rate</i>	86
<i>Figura 72. Parámetros Bloque Noise Source</i>	87
<i>Figura 73. Esquema medición del BER</i>	88
<i>Figura 74. Ruido Gaussiano</i>	89
<i>Figura 75. Distribución gaussiana del Bloque Noise Source</i>	89
<i>Figura 76. BER vs SNR para los bits de alta y baja prioridad en los esquemas de modulación 16-QAM ($\alpha = 1$) y 16-CQAM ($\alpha = 2$)</i>	90
<i>Figura 77. Esquema utilizado en la medición del BER</i>	91
<i>Figura 78. Resultado numérico de BER</i>	91
<i>Figura 79. Comportamiento de la constelación recibida en el demodulador al aumentar la distancia entre las antenas TX y RX, así mismo la SNR de 15 a 1 (Figuras a – o)</i>	94
<i>Figura 80. BER vs SNR para los bits de alta y baja prioridad en los esquemas de modulación 16-QAM ($\alpha = 1$) y 16-CQAM ($\alpha = 2$)</i>	95

<i>Anexo A. Puesta en marcha del software necesario:</i>	103
<i>Anexo B. GNU Radio companion-Python-C++</i>	107
<i>Anexo C. Pruebas de transmisión</i>	108
<i>Anexo D. Código Python de la Conversión a código Gray</i>	109
<i>Anexo E. Código Python de la constelación</i>	110
<i>Anexo F. Código Python Bloque cálculo del Error Rate</i>	120

Capítulo 1

Introducción

1.1. Introducción General

Los sistemas de comunicación hoy en día, exigen la transmisión de datos a alta velocidad y baja tasa de error de bit a través de canales imperfectos, con ruido, interferencias y distorsión. Asegurar una tasa o probabilidad de error de bit baja y un uso eficiente del ancho de banda depende directamente de los esquemas de modulación y el tipo de codificación aplicada.

Actualmente se dispone de un amplio espectro de esquemas de modulación desde los sin memoria hasta con los con memoria, lineales y no lineales, sin embargo la mayoría tratan a los bits de baja y alta prioridad de la misma forma, es decir ofrecen una tasa de error de bit uniforme, así que si se tiene información en la cual algunos bits son de mayor prioridad, estos tendrán la misma probabilidad de error que los de más baja prioridad.

A los esquemas de modulación en que a todos los bits se les trata de la misma forma se dice que proporcionan una protección de error igual (EEP) [1], pero también se tienen esquemas de modulación que permiten asignar una protección de error desigual (UEP), por ejemplo la modulación de amplitud de cuadratura jerárquica (HQAM), la cual consiste en realizar una distinción de los bits más significativos (MSB) y los menos significativos (LSB) generando una menor probabilidad de error de bits sobre los MSB en detrimento de los LSB, ello ha demostrado ser una herramienta útil para hacer frente a los problemas de ruido en comunicaciones digitales cuando se transmite información jerarquía de bits [1].

Una expansión de HQAM se llevó a cabo por Fonseca y Capera [2] donde se muestra la aplicación de los conjuntos de cantor generalizados en modulación HQAM generando un nuevo sistema de modulación CQAM¹; el desarrollo planteado por Fonseca y Capera presenta y diseña una novedosa forma de llevar a cabo la discriminación y división de los bits MSB y LSB, presentando resultados simulados que hasta el momento no han sido implementados y validados a nivel de Hardware.

Una de las tecnologías en comunicaciones, para la implementación de sistemas de modulación en hardware, es la tecnología de *hardware* de radio definida por *software* (SDR), que tiene la ventaja de ser un sistema de radiocomunicación donde la mayor parte de los componentes necesarios se implementan en *software* en lugar de *hardware*,

¹ CQAM: Modulación por Amplitud de Pulsos Cantor

traduciendo todos los problemas del *hardware* a problemas de *software* [3] [4], que se plantean en este proyecto en la implementación del sistema de modulación CQAM.

1.2. Planteamiento del Problema

Aunque el desarrollo de las comunicaciones ha llevado a disponer de un amplio conjunto de esquemas de modulación incluyendo los jerárquicos, para los esquemas de modulación con estructuras basadas en los conjuntos de Cantor no se ha implementado en hardware.

Por tanto el problema abordado en este proyecto consiste en que actualmente no se dispone de un modelo funcional y una arquitectura de hardware para la implementación de modulaciones jerárquicas basadas en las estructuras de Cantor.

1.3. Objetivos

1.3.1. Objetivo Principal

Implementar un sistema jerárquico de Modulación por Amplitud de Cuadratura de 16 estados con base en conjuntos de Cantor (CQAM) sobre radio definida por software.

1.3.2. Objetivos Específicos

- Diseñar e implementar el modulador 16-CQAM.
- Diseñar e implementar el demodulador 16-CQAM.
- Evaluar el desempeño del sistema de modulación CQAM implementado y compararlo con el sistema de modulación QAM tradicional en relación a la tasa de error de bits (BER) respecto a la relación señal a ruido (SNR).

1.4. Justificación

En la transmisión de datos sobre un canal, QAM ofrece alta eficiencia de transmisión mediante la utilización de variaciones de amplitud y de fase. Sin embargo, ofrece protección equitativa de bits de alta y baja prioridad generando una gran posibilidad de error por parte del receptor al aumento de información debido a las interferencias en el canal de comunicación.

Por otro lado QAM Jerárquica (HQAM) [5] supera esta desventaja proporcionando una mayor protección a los bits de prioridad alta con la consecuente disminución en la protección de los bits de baja prioridad. Por lo tanto HQAM es una técnica que proporciona transmisión más eficiente para señales como el audio y las imágenes en donde la pérdida de algunos paquetes de información no supone una degradación relevante en la información transmitida [1].

Con el objetivo de brindar una estrategia de separación a cada uno de los datos transmitidos, Görtzen con modulación CPAM [6] y Fonseca y Capera con modulación CQAM [2] propusieron una modulación HQAM basada en conjuntos de cantor, la cual proporciona una estrategia de separación para los bits transmitidos, sin embargo sólo Fonseca y Capera llegaron a proponer y simular un sistema de comunicaciones de modulación HQAM basado en conjuntos de Cantor (CQAM).

Por lo anterior y dado que la simulación no es suficiente para validar el desempeño y la factibilidad de la propuesta se hace necesaria la implementación por medio de hardware de una sistema de comunicaciones que utilice esta nueva modulación CQAM.

Capítulo 2

Marco Teórico

2.1 Representación en Espacio de Señal

Las señales tienen características similares a los vectores y se mostrará una representación vectorial de las formas de onda de las señales [7].

Como en el caso de vectores, se puede definir un tratamiento para una señal definida en el intervalo $[a, b]$. El producto punto de dos señales genéricas $x_1(t)$ y $x_2(t)$ se denota por: $(x_1(t), x_2(t))$ y definido como:

$$(x_1(t), x_2(t)) = \int_a^b x_1(t)x_2^*(t)dt \quad (1)$$

Las señales son ortogonales si su producto interno es cero.

Por otro lado la norma de una señal se define como:

$$\|x(t)\| = \left(\int_a^b |x(t)|^2 dt \right)^{1/2} \quad (2)$$

Un conjunto de m señales son orto-normales si son ortogonales y su norma es la unidad. Un conjunto de señales es linealmente independiente si ninguna de las señales se puede representar por la combinación lineal de las demás señales. Así mismo se aplica la desigualdad triangular como: $\|x_1(t) + x_2(t)\| \leq \|x_1(t)\| + \|x_2(t)\|$ y la desigualdad de Cauchy-Schwartz es:

$$\left| \int_a^b x_1(t)x_2^*(t)dt \right| \leq \left(\int_a^b |x_1(t)|^2 dt \right)^{1/2} \left(\int_a^b |x_2(t)|^2 dt \right)^{1/2} \quad (3)$$

Es igualdad siempre que $x_2(t) = ax_1(t)$, con a siendo un complejo [7]

De acuerdo a [8] y [7] se va a definir una señal usando la aproximación vectorial a continuación

Se define la ortogonalidad de un conjunto de señales $x_1(t), x_2(t), \dots, x_N(t)$ Sobre el intervalo $[t_1, t_2]$ como:

$$\int_{t_1}^{t_2} x_m(t)x_n^*(t)dt \begin{cases} 0 & m \neq n \\ E_n & m = n \end{cases} \quad (4)$$

Si la energía $E_n = 1$ para todos los, entonces el conjunto está normalizado y se denomina conjunto ortonormal.

Un conjunto ortonormal se puede siempre normalizar dividiendo $x_n(t)$ por $\sqrt{E_n}$ para todos los n .

Ahora considere el problema de aproximar una señal $g(t)$ sobre el intervalo $[t_1, t_2]$ por medio de un conjunto de N señales mutualmente ortogonales $x_1(t), x_2(t), \dots, x_N(t)$:

$$g(t) \simeq c_1x_1(t) + c_2x_2(t) + \dots + c_Nx_N(t) \quad (5)$$

$$= \sum_{n=1}^N c_n x_n(t) \quad t_1 \leq t \leq t_2 \quad (6)$$

Se puede mostrar que E_e , la energía de error de señal $e(t)$, en esta aproximación es mínima si se escoge:

$$c_n = \frac{\int_{t_1}^{t_2} g(t)x_n^*(t)dt}{\int_{t_1}^{t_2} x_n^2(t)dt} \quad (7)$$

$$= \frac{1}{E_n} \int_{t_1}^{t_2} g(t)x_n^*(t)dt \quad n = 1, 2, \dots, N \quad (8)$$

Además, si el conjunto ortogonal está completo, la energía de error tiende a cero, y la representación de las ecuaciones 2 y 3 deja de ser una aproximación y se convierte en una igualdad.

2.2 Modulación por Amplitud de Cuadratura (QAM)

La Modulación de Amplitud en Cuadratura o QAM es una modulación digital en la que el mensaje está contenido tanto en la amplitud como en la fase de la señal transmitida. Se basa en la transmisión de dos mensajes independientes por un único camino. Esto se consigue modulando una misma portadora, desfasada 90º entre uno y otro mensaje. Esto supone la formación de dos canales ortogonales en el mismo ancho de banda, con lo cual se mejora en eficiencia de ancho de banda que se consigue con esta modulación respecto a la modulación de Amplitud de Pulso (PAM).

Desde el punto de vista de espacio de señal esta modulación se obtiene mediante la impresión de forma simultánea de dos símbolos de k-bits de la secuencia de información de entrada sobre dos portadoras en cuadratura $\cos 2\pi f_c t$ y $\sin 2\pi f_c t$, Donde A_{mi} y A_{mq} son las amplitudes de señal portadora de información de cuadratura $g(t)$ es la señal de información de pulso y f_c la frecuencia de la portadora, las formas de onda de la señal QAM pueden expresarse como:

$$s_m(t) = \operatorname{Re}[(A_{mi} + jA_{mq})g(t)e^{j2\pi f_c t}] \quad (9)$$

$$s_m(t) = A_{mi}g(t)\cos 2\pi f_c t + A_{mq}g(t)\sin 2\pi f_c t, \quad m = 1, 2, \dots, M \quad (10)$$

Donde A_{mi} y A_{mq} son las amplitudes de la señal portadora de información y $g(t)$ es la señal pulso. Otra descripción del QAM se puede mostrar como en la ecuación (11)

$$s_m(t) = \operatorname{Re}[r_m e^{j\theta_m} e^{j2\pi f_c t}] \quad (11)$$

$$s_m(t) = r_m \cos(2\pi f_c t + \theta_m) \quad (12)$$

Donde $r_m = \sqrt{A_{mi}^2 + A_{mq}^2}$ y $\theta_m = \tan^{-1}(A_{mq}/A_{mi})$. A partir de esta expresión, se puede llegar a entender que las formas de onda de la señal QAM se pueden ver como la combinación de amplitud (r_m) y fase (θ_m)

La dimensión del espacio de señal para QAM es N=2. Usando esta base, podemos obtener:

$$s_m(t) = A_{mi}\sqrt{\frac{\epsilon_g}{2}}\phi_1(t) + A_{mq}\sqrt{\frac{\epsilon_g}{2}}\phi_2(t) \quad (13)$$

Dónde:

$$\phi_1(t) = \sqrt{\frac{2}{\varepsilon_g}} \cos 2\pi f_c t \quad (14)$$

$$\phi_2(t) = \sqrt{\frac{2}{\varepsilon_g}} \sin 2\pi f_c t \quad (15)$$

$\phi_1(t)$ y $\phi_2(t)$ son ortogonales.

Lo que resulta en representación vectorial de la forma

$$s_m = (s_{m1}, s_{m2}) \quad (16)$$

$$s_m = \left(A_{mi} \sqrt{\frac{\varepsilon_g}{2}}, A_{mq} \sqrt{\frac{\varepsilon_g}{2}} \right) \quad (17)$$

Y

$$\varepsilon_m = \|s_m\|^2 = \frac{\varepsilon_g}{2} (A_{mi}^2 + A_{mq}^2) \quad (18)$$

El par ordenado (s_{m1}, s_{m2}) pertenece a las coordenadas de los símbolos de la constelación² QAM, en la figura 1 se muestra un diagrama de constelación para 16 símbolos de modulación en cuadratura (16-QAM), el canal superior es conocido como canal en fase (*I*) y el canal inferior se conoce como canal en cuadratura (*Q*). Generalmente, los espacios entre los símbolos tiene la misma distancia *d* obteniéndose una constelación cuadrada.

En la figura 1 se observa el diagrama de constelación para 16-QAM. Todos sus estados tiene la misma distancia entre ellos, por lo cual esta modulación tiene una protección del error equivalente (EEP) haciendo que todos los bits se transmitan con la misma prioridad. Esto hace que la probabilidad de error sea la misma para los bits más significativos y menos significativos. [2]

² Constelación QAM: Representación cartesiana de $f_1(t)$ como *I* y $f_2(t)$ como *Q*

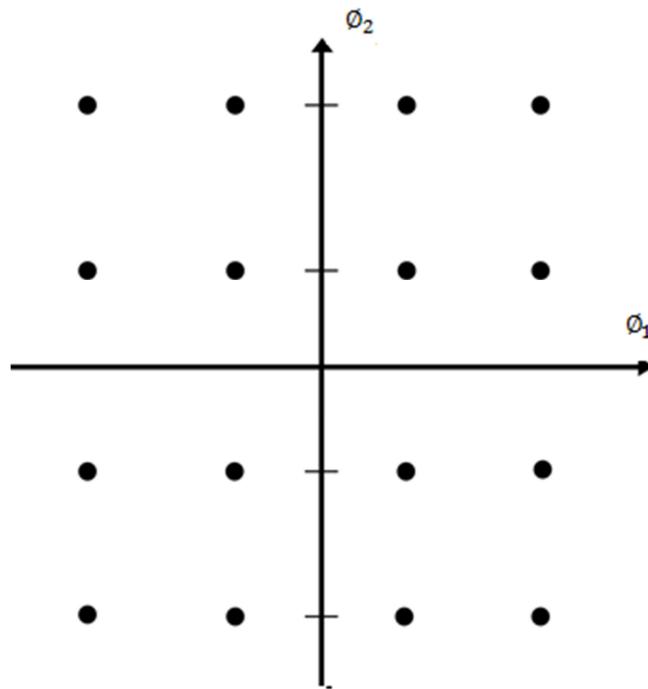


Figura 1. Constelación de una modulación 16-CQAM. Todos sus estados se encuentran separados a la misma distancia.

En cuanto al esquema de modulación QAM, en la figura 2 se muestra el esquema del modulador y en la figura 3 el demodulador.

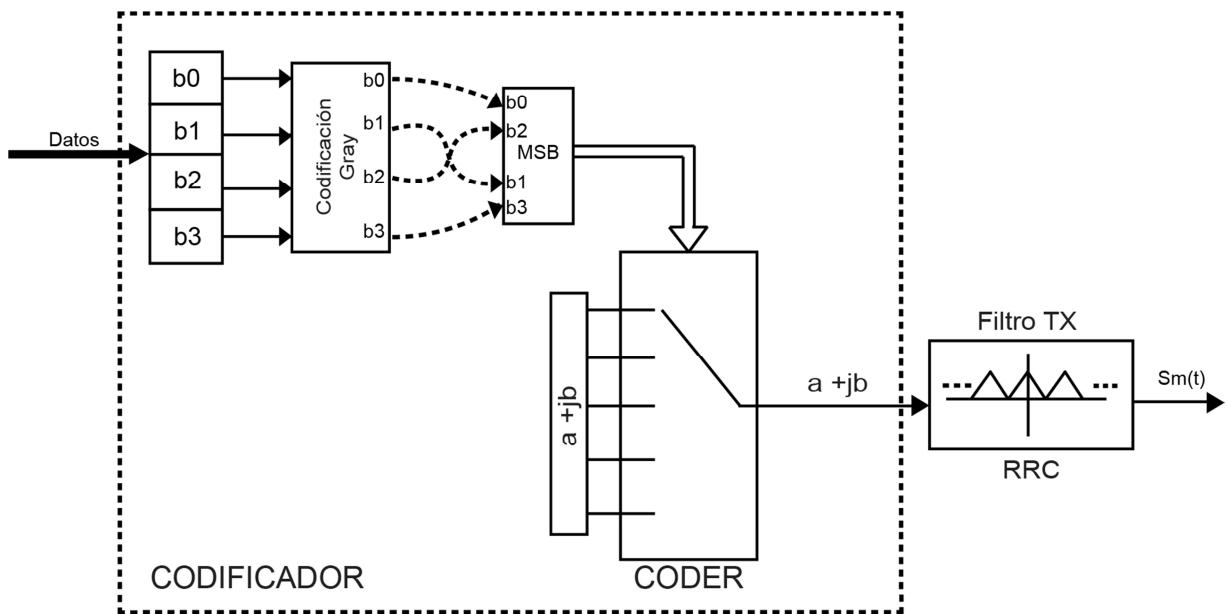


Figura 2. Esquema de bloques modulador QAM

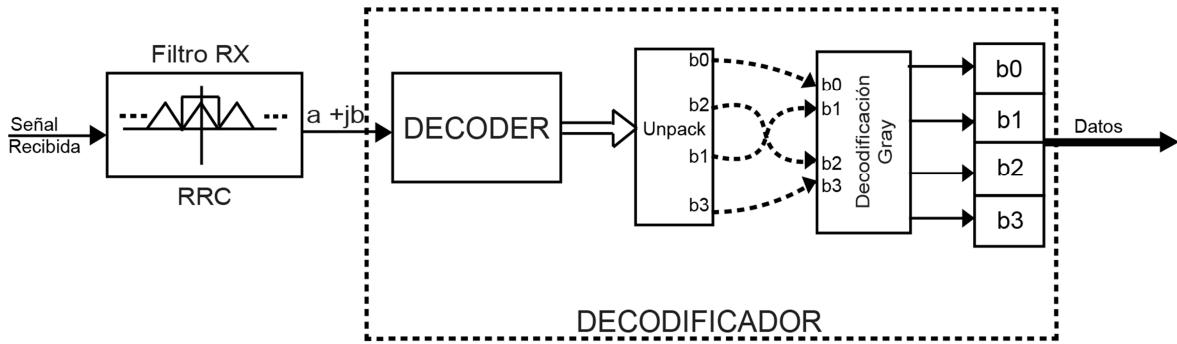


Figura 3. Esquema de bloques demodulador QAM

2.3 Modulación QAM de tipo Jerárquica (HQAM)

La modulación QAM tradicional es más susceptible a la interferencia debido a que los estados están muy cerca entre ellos, donde un pequeño nivel de ruido puede desviar la señal hacia otro punto de decisión. Una alternativa es proporcionar diferente grado de protección a la transmisión de bits de datos, en la que se asignan la alta prioridad (HP) a los bits más significativos (MSB) y la baja prioridad (LP) a los bits menos significativos (LSB) de los puntos de constelación, generando una modulación QAM de tipo jerárquica.

HQAM será, por lo tanto, resultado en una mejor calidad en comparación con QAM especialmente en condiciones de baja SNR del canal, ya que, la gran cantidad de bits de datos sensibles de HP se asignan a los bits más significativos con una baja tasa de error de bit. Este método ha sido aplicado en sistemas digitales en la transmisión de señales como el audio y el video en donde la pérdida de algunos paquetes de información no supone una degradación relevante en la información transmitida como el de difusión de video digital terrestre (DVB-T) [9].

En QAM jerárquica, es posible dar una mayor protección a los datos más importantes (bits significativos) cambiando el valor del parámetro de modulación α , que es la relación de la distancia b entre los cuadrantes a la distancia entre los puntos c dentro de un cuadrante en el diagrama de constelación. La figura 4 muestra el diagrama de constelación para $\alpha = 2$, es decir $b = 2c$.

Por otra parte, el valor de α no debe exceder la raíz cuadrada de la potencia de la portadora, de lo contrario, los puntos de la constelación del mismo cuadrante se solaparán [1]. Cuando $\alpha = 1$, es decir $b = c$ HQAM resulta ser QAM.

Haciendo referencia a la figura 4, los dos primeros bits representan los bits de alta prioridad que tienen un BER menor a los dos últimos bits, siendo estos los bits de baja prioridad. Se puede observar que los cuatro símbolos en cada cuadrante tienen los mismos bits de alta prioridad, pero diferentes bits de baja prioridad; esto también se llama

constelación de superposición y asegurar que los bits de alta prioridad puedan ser transmitidos correctamente [10].

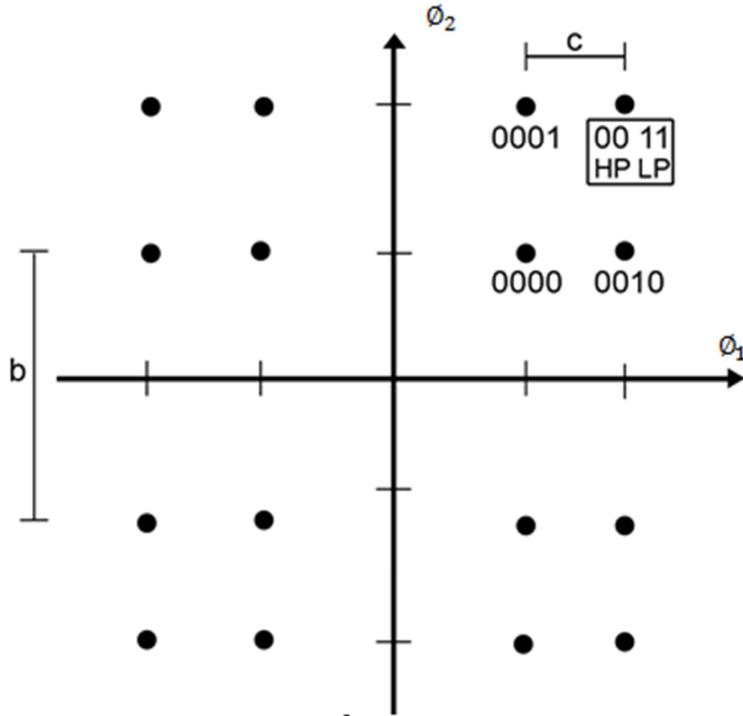


Figura 4. 16-HQAM con $\alpha = 2$,

Cuando aumenta α , la distancia de puntos de la constelación entre los cuadrantes aumenta y dentro de un cuadrante disminuye, además, se puede observar que aumentando el grado de no uniformidad ($b > c$) la mejora del rendimiento de los bits de alta prioridad es significativo haciendo este un método UEP sencillo sin introducir ninguna redundancia a los datos modulados.

En la figura 5 se aprecia el sistema completo de un modulador y demodulador QAM Jerárquico de 16 estados. En la parte del modulador, este empieza recibiendo el flujo de información y se encarga de clasificar y dividir las unidades de información (bytes) en bits de alta y baja prioridad; si se tratara de un flujo de datos que consiste en unidades de información de 4 bits, el sistema divide los primeros dos bits y los envía como bits de alta prioridad (HP), mientras los otros 2 bits restantes serán enviados como bits de baja prioridad (LP).

El modulador HQAM agrupa el primero de los bits de alta prioridad con el primer bit de baja prioridad y le asigna un valor del componente, ya sea en fase o cuadratura, previamente calculado según la relación α requerida. Este mismo procedimiento ocurre con el segundo bit de alta prioridad con el segundo de baja prioridad. Una vez se tengan

los valores de los componentes en cuadratura y en fase, se puede transmitir el símbolo en un transmisor QAM normal.

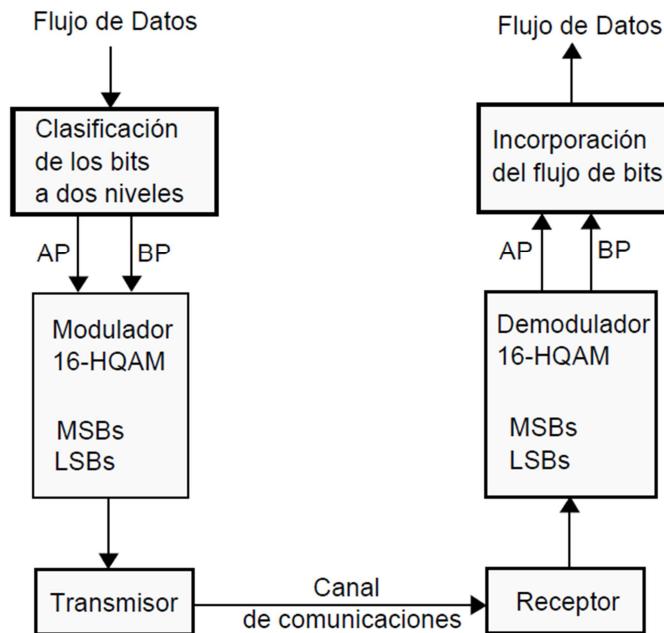


Figura 5. Sistema de modulación y demodulación 16-HQAM.

El demodulador por su parte determina los símbolos que le han sido transmitidos comparándolos con los valores que tiene para realizar la decisión. Posteriormente, de los símbolos recibidos se procede a identificar los bits más significativos e incorporarlos con los bits de baja prioridad.

2.3.1 Estimación del BER para 16-HQAM

El cálculo del BER para una constelación jerárquica de 16 símbolos se puede calcular a partir de uno de los cuadrantes debido a la simetría entre ellos, por ejemplo tomando el primer cuadrante de la figura 6 se puede obtener el BER para los bits de alta prioridad, obteniendo como probabilidad de error las ecuaciones 19 y 22 [17] [18]

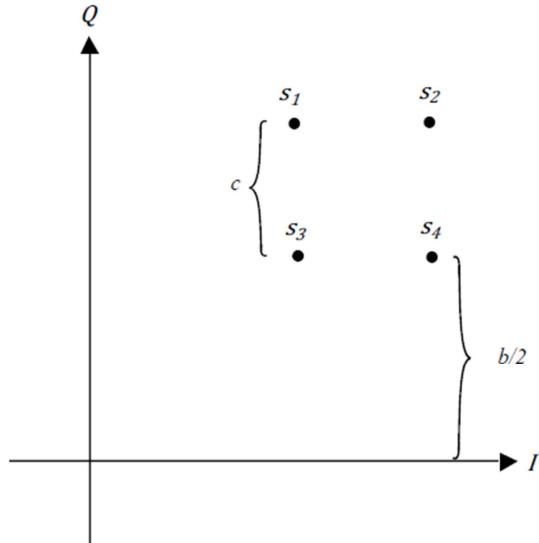


Figura 6. Primer cuadrante de una constelación 16-HQAM

$$P_{HB} = 1 - \left(\frac{1}{2} \left(1 - Q \left(\frac{b}{2\sigma_n} \right) \right) + \frac{1}{2} \left(1 - Q \left(\frac{b+2c}{2\sigma_n} \right) \right) \right)^2 \quad (19)$$

Siendo Q la función erf , si se define:

$$p_{hb} = 1 - Q \left(\frac{b}{2\sigma_n} \right) \quad (20)$$

$$q_{hb} = 1 - Q \left(\frac{b+2c}{2\sigma_n} \right) \quad (21)$$

Se puede definir la probabilidad de error como:

$$P_{HB} = 1 - \left(\frac{1}{2} p_{hb} + \frac{1}{2} q_{hb} \right)^2 \quad (22)$$

Finalmente la energía de cada cuadrante es:

$$E_1 = \left(\frac{b}{2} \right)^2 + \left(\frac{b+2c}{2} \right)^2 = \frac{1}{2} b^2 + bc + c^2 \quad (23)$$

$$E_3 = 2 \left(\frac{b}{2} \right)^2 = \frac{b^2}{2} \quad (24)$$

$$E_2 = 2 \left(\frac{b+2c}{2} \right)^2 = \frac{1}{2} b^2 + 2bc + c^2 \quad (25)$$

$$E_4 = E_1 \quad (26)$$

Así la energía promedio por símbolo es:

$$\bar{E} = \frac{1}{4} [E_1 + E_2 + E_3 + E_4] = \frac{1}{2} b^2 + bc + c^2 \quad (27)$$

Y la energía por bit es:

$$E_b = \frac{\bar{E}}{4} = \frac{1}{8} (b^2 + 2bc + 1c^2) \quad (28)$$

Reemplazando $\alpha = \frac{b}{c}$ obteniendo

$$c = \sqrt{\frac{8E_b}{\alpha^2 + 2\alpha + 2}} \quad (29)$$

La ecuación 29 junto a la relación de ruido espectral $\sigma_n = \sqrt{N_0/2}$ se utilizarán para reemplazar los términos $\frac{b}{2\sigma_n}$ y $\frac{b+2c}{2\sigma_n}$ que se encuentran en p_{hb} y q_{hb} de la ecuación 19, de esta forma:

$$p_{hb} = 1 - Q\left(\alpha \sqrt{\frac{4E_b}{N_0(\alpha^2 + 2\alpha + 2)}}\right) \quad (30)$$

$$q_{hb} = 1 - Q\left((\alpha + 2) \sqrt{\frac{4E_b}{N_0(\alpha^2 + 2\alpha + 2)}}\right) \quad (31)$$

Para los bits de baja prioridad se tiene:

$$\begin{aligned} P_{LB} &= 1 - \left(\frac{1}{2} \left(1 - Q\left((2\alpha + 1) \sqrt{\frac{4E_b}{N_0(\alpha^2 + 2\alpha + 2)}}\right) - Q\left(\sqrt{\frac{4E_b}{N_0(\alpha^2 + 2\alpha + 2)}}\right) \right. \right. \\ &\quad \left. \left. + \left(1 - Q\left(\sqrt{\frac{4E_b}{N_0(\alpha^2 + 2\alpha + 2)}}\right) \right) \right)^2 \right) \end{aligned} \quad (32)$$

Si se define:

$$p_{lb} = 1 - Q\left((2\alpha + 1) \sqrt{\frac{4E_b}{N_0(\alpha^2 + 2\alpha + 2)}}\right) - Q\left(\sqrt{\frac{4E_b}{N_0(\alpha^2 + 2\alpha + 2)}}\right) \quad (33)$$

$$q_{lb} = 1 - Q\left(\sqrt{\frac{4E_b}{N_0(\alpha^2 + 2\alpha + 2)}}\right) \quad (34)$$

Quedando:

$$P_{LB} = 1 - \left(\frac{1}{2}P_{lb} + \frac{1}{2}q_{lb}\right)^2 \quad (35)$$

En la figura 7, se muestra el BER vs SNR (para un canal AWGN) para los bits de HP y LP utilizando una modulación 16-HQAM para $\alpha = 1$ a 5 y una modulación convencional 16-QAM donde $\alpha = 1$ [1].

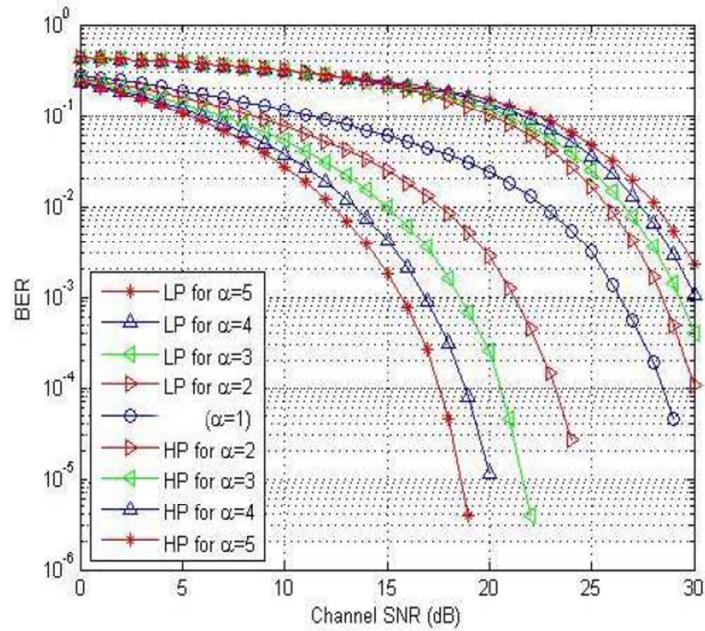


Figura 7. BER vs SNR para los bits de HP y LP utilizando 16-HQAM para $\alpha = 2$ a 5 y 16-QAM ($\alpha = 1$).
Tomada de [1].

Una de las formas que ofrece la posibilidad de obtener estados de constelaciones jerárquicas que poseen diferentes distancias b y c que no se solapan, es mediante los conjuntos de Cantor, que se explican a continuación.

2.4 Conjuntos de Cantor

George Cantor (1845-1918), fue un matemático mejor conocido por su trabajo en la teoría de conjuntos. En 1883 publicó un artículo acerca de los conjuntos que llevan su nombre. Los conjuntos de Cantor juegan un importante rol en muchas ramas de las matemáticas, en especial en la teoría de conjuntos, sistemas dinámicos caóticos y teoría de fractales [13]

El conjunto básico (ternario) de Cantor es un conjunto que se puede expresar a través de su regla de construcción de la figura 8.



Figura 8. Segmentos generados en la tercera iteración del conjunto de cantor de tercios medios

Se define el intervalo $[0,1]$ y se suprime de allí el intervalo $[1/3,2/3]$ los dos conjuntos disyuntos $[0,1/3]$ y $[2/3,1]$ forman el primer conjunto de puntos C_1

$$C_1 = \left[0, \frac{1}{3} \right] \cup \left[\frac{2}{3}, 1 \right] \quad (36)$$

En el segundo paso se remueven los tercios medios de los dos segmentos C_1 lo que es remover $\left(\frac{1}{9}, \frac{2}{9} \right) \cup \left(\frac{7}{9}, \frac{8}{9} \right)$ y quedaría el conjunto $C_2 = \left[0, \frac{1}{9} \right] \cup \left[\frac{2}{9}, \frac{3}{9} \right] \cup \left[\frac{6}{9}, \frac{7}{9} \right] \cup \left[\frac{8}{9}, 1 \right]$; se aplican las mismas reglas para generar C_3 hasta C_n como se muestra en la figura 8. [14], [15].

El conjunto de Cantor se define como la intersección de todos los conjuntos C_i

$$C = \bigcap_{i=0}^{\infty} C_i$$

En la figura 8, se muestra las primeras etapas de la construcción del conjunto de Cantor

2.4.1 Conjuntos de Cantor generalizados

Basándose en el conjunto de Cantor básico ternario, se define a continuación los conjuntos de cantor generalizados, centrando el conjunto en el origen. Por lo tanto se toma el intervalo $[-1, 1]$ en lugar de intervalo unitario. Y el siguiente paso consiste en no sólo permitir remover tercios medios de cada intervalo, sino intervalos arbitrarios. Por lo que se va a introducir el factor de escala f en cada intervalo I , donde la porción media de tamaño $|I|(f - 2)/f$ se remueve.

Claramente al escoger $f = 3$ se obtiene el conjunto ternario de Cantor. Finalmente se permite usar diferentes factores de escala f_i en cada etapa de las iteraciones.

El conjunto de Cantor Generalizado se define como:

$$C_0 = [-1, 1] \quad (37)$$

$$C_i = \frac{1}{f_i} (C_{i-1} - 1(f_i - 1)) \cup \frac{1}{f_i} (C_{i-1} + (f_i - 1)) \quad (38)$$

Para todos los $i \geq 1$, en donde se usa la notación:

$$a \cdot S + b = \{as + b | s \in S\} \quad (39)$$

Siendo S un conjunto y $a, b \in \mathbb{R}$. Claramente, C_n es la unión de 2^n intervalos de longitud $2/\prod_{i=1}^n f_i$. Por lo que el conjunto de cantor generalizado se define como:

$$C = \lim_{n \rightarrow \infty} C_n \quad (40)$$

f_i Es el factor de escala que cambia la porción entre los segmentos de línea en C_i como se puede ver en la figura 9.

Dicha flexibilidad no está presente en el conjunto de Cantor básico ternario donde los segmentos de línea tienen la misma proporción entre si $f_i = 3 \forall i > 0$ en la ecuación (26).

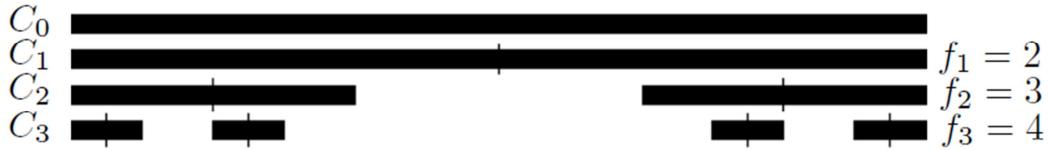


Figura 9. Primeras Etapas de Conjunto de Cantor con varios parámetros f_i

Siguiendo la definición anterior se va a analizar la particularidad de los subconjuntos en la n -etapa finita. Siendo P_n los centros de los intervalos C_n . Claramente $|P_n| = 2^n$, adicionalmente la familia $(P_n)_{n \in \mathbb{N}_0}$ pueden ser representados por un árbol binario, en el cual los nodos de la profundidad n corresponden a P_n . Con la siguiente descripción:

(41)

$$P_0 = \{0\}$$

$$P_i = \left(P_{i-1} - \frac{f_i - 1}{\prod_{j=1}^i f_j} \right) \cup \left(P_{i-1} + \frac{f_i - 1}{\prod_{j=1}^i f_j} \right) \quad (42)$$

Cada nodo x con profundidad $n - 1$ tiene descendencia $x \pm (f_n - 1)/\prod_{i=1}^n f_i$ y todos los niveles del árbol se ordenan de izquierda a derecha. Esta representación de árbol binario permite mapear cada elemento de P_n a una secuencia de bits de longitud n [6].

Cada subconjunto C_i está compuesto por dos intervalos que bien podrían representar los valores alto y bajo del sistema binario. Junto a esto, los centros pueden formar un árbol binario para describir los símbolos que harán parte de la constelación del QAM. La ecuación 25 se modifica para tener una mapeo binario de los centros de los intervalos, que se muestra en la ecuación 26.

$$(b_1 b_2 \dots b_n) \rightarrow \sum_{i=1}^n (-1)^{1-b_i} \frac{f_i - 1}{\prod_{j=1}^i f_j} \quad (43)$$

En la figura 10 se observa un conjunto de cantor en sus primeras etapas junto a los centros en el último subconjunto $C3$ y su representación binaria, introduciendo así, un método de modulación jerárquica basada en conjuntos de cantor CPAM.

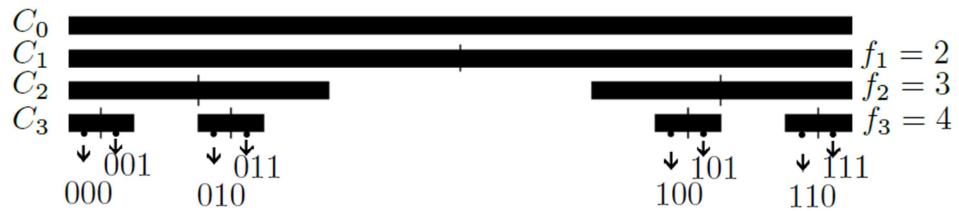


Figura 10. Representación binaria del subconjunto C_3 . Los semicírculos que se encuentran en la mitad de cada intervalo simbolizan los centros P_n

2.5 Modulación CQAM (CANTOR-QAM)

Muchos autores como Wei [5], Seshadri y Sundberg [16] hablaron sobre la propuesta de un modulador QAM con protección de bit desigual o jerárquica, adicionalmente se ha estudiado por Suoto [17] y Pavan [18] la respuesta en cuanto a la mejora del BER para los Bit de mayor prioridad en esquemas de modulación jerárquica y también de cómo se deben calcular y tratar estas tasas de error.

Más adelante en 2011 Görtzen [6] Propone utilizar conjuntos de Cantor en la modulación HQAM creando la CPAM, con esta propuesta Fonseca y Capera [2] proponen y diseñan un modulador y demodulador CQAM y presentan simulaciones orientadas en mostrar las ventajas de este sistema de modulación basado en conjuntos de cantor.

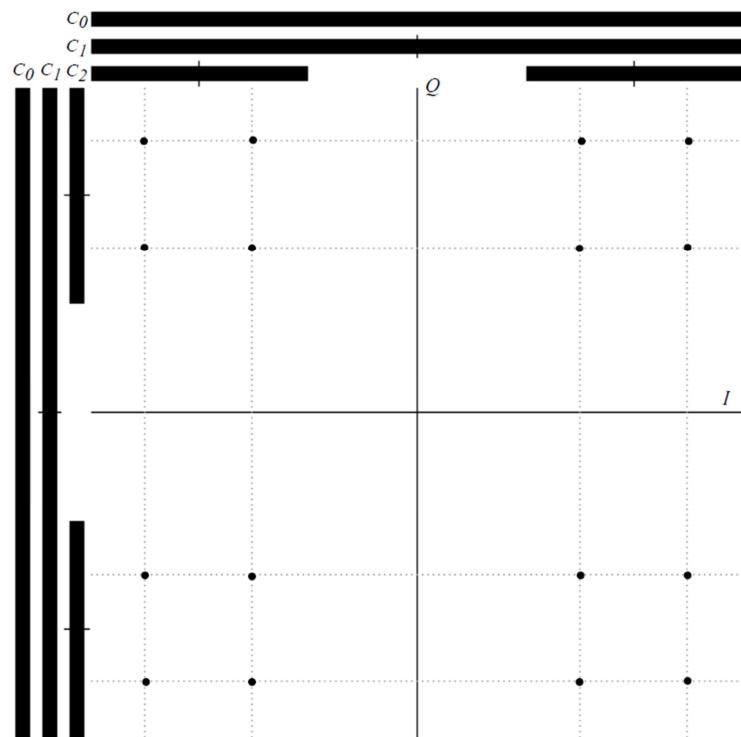


Figura 11. Constelación CQAM formada a partir de modulaciones CPAM.

Al implementar la constelación de un QAM con base en los conjuntos de cantor, este procedimiento ha de ser llevado a la segunda dimensión, esto es, $C_n \times C_n$ donde n hace referencia a la etapa del conjunto de Cantor. Si se usa el subconjunto C_3 por ejemplo, quedaría $C_3 \times C_3$ que da como resultado un 64-CQAM.

En la figura 11 se observa como dos conjuntos de Cantor C_2 con los mismos factores de escala, forman una constelación 16-CQAM [2].

Una de las tecnologías para la implementación en hardware de estos esquemas de modulación o sistemas de comunicación digital, es la tecnología de Software Definida por Radio (SDR), que se explica en la siguiente sección.

2.6 Tecnología SDR

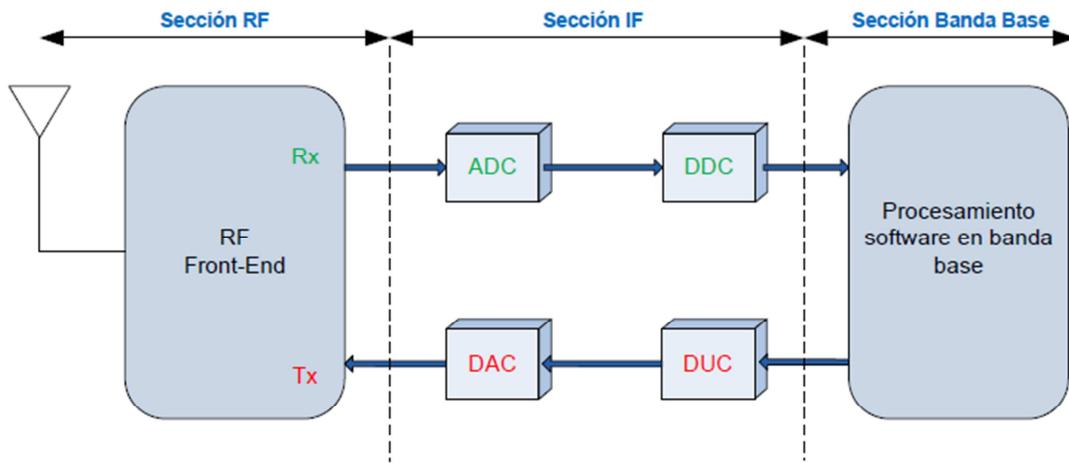


Figura 12. Diagrama Bloques Principales de SDR

Un sistema *Software Defined Radio* (SDR), o radio definida por software, es un sistema de radiocomunicación donde la mayor parte de los componentes necesarios se implementan en software en lugar de en hardware. Al utilizar esta tecnología, se implementa un receptor *Zero-IF* o *low-IF*³ configurable de tal manera que puede utilizarse para diseñar distintos componentes como mezcladores, filtros, amplificadores, moduladores/demoduladores y detectores entre otros e incluso sistemas completos tales como transmisores, receptores, transceptores, osciloscopios, analizadores de espectros o analizadores vectoriales de redes, siendo sus parámetros configurables dinámicamente y por consiguiente aportando una gran flexibilidad a la hora de realizar un sistema de radiocomunicación, en la figura 12 se muestra el diagrama de bloques básico de un sistema SDR [19].

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

³ IF, *intermediate frequency* o frecuencia intermedia

la señal es analógica. El dispositivo debe ser configurable para que sea posible recibir distintos anchos de banda y frecuencias para posteriormente, con un procesado digital, poder realizar la sincronización y la detección de la señal recibida.

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

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

El SDR está compuesto por tres bloques funcionales como se muestra en figura 12: sección de RF⁴, sección de IF y sección banda base. La parte de RF e IF se implementan en hardware mientras que la sección de banda base en software.

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

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

La sección de banda base es la encargada de todo el procesamiento en banda base de la señal como modulación/demodulación, análisis espectral de la señal llevándose a cabo en software [20].

⁴ RF, *radio frequency* o frecuencia intermedia

En la actualidad, hay varios fabricantes encargados de proveer plataformas hardware para la tecnología SDR, de entre los que se pueden destacar los siguientes:

- *Ettus Research*: Es el creador del Universal Software Radio Peripheral.
- *National Instruments*: Es el creador del NI-Universal Software Radio Peripheral
- *Pentek*: Fabricante que permite crear una plataforma SDR personalizada según las necesidades del desarrollador.
- *Datasoft*: Creador del Thunder-SDR
- *FlexRadio Systems*: Es el creador de SDR-1000.
- *Realtek*: Creador del rtl2832, un demodulador DVB-T⁵ COFDM⁶ que puede ser utilizado como un receptor SDR.

2.6.1 Universal Software Radio Peripheral (USRP)

El Universal Software Radio Peripheral es un periférico diseñado para trabajar en conjunto con un procesador externo (PC, Workstation,...) a través de una FPGA y permite la realización de *software radio*. En la figura 13 se muestra el diagrama de bloques principales en la USRP [19].

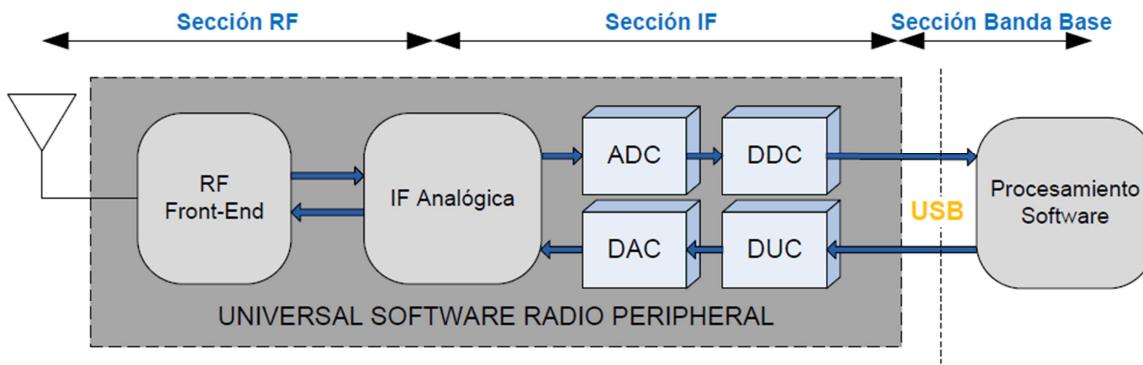


Figura 13. Diagrama Bloques Principales USRP

El USRP realiza las funciones de llevar la señal a banda base de RF a través de la sección de IF y viceversa tal y como se muestra en la figura 13. La interfaz USB permite comunicar al USRP con el ordenador que realizará el procesamiento software. El ordenador puede ser un ordenador personal, un ordenador portátil, una estación de trabajo, etc. En general un ordenador que soporte sistema operativo Linux, MAC OS o Windows

⁵ Digital Video Broadcasting - Terrestrial

⁶ Coded Orthogonal Frequency Division Multiplexing

2.6.2 USRP Hardware Driver

El Driver necesario para trabajar con el USRP es el USRP Hardware Driver (UHD), se trata de una librería escrita en C++. Este driver está pensado para trabajar en las plataformas Linux, Windows y Mac.

El objetivo principal del driver es proveer control sobre los productos de Ettus, el uso de este software puede ser utilizado de manera *stand-alone* o recurriendo a otras aplicaciones como son:

- GNU Radio: Herramienta software, libre y de código abierto de desarrollo que provee la posibilidad de implementar sistemas basados en software radio mediante el uso de bloques de procesado de señal. Es una herramienta de simulación que puede ser utilizado junto a hardware RF (USRP) para implementar físicamente sistemas de software radio.
- LabVIEW: Es una plataforma de desarrollo para diseñar sistemas hardware y software, haciendo uso de un lenguaje de programación gráfico (lenguaje G). Esta plataforma fue creada por National Instruments.
- Simulink: Entorno de programación visual de diagramas de bloques integrado en Matlab. Utilizado para la simulación de sistemas, puede ser utilizado junto a hardware USRP para implementar físicamente sistemas de software radio.
- OpenBTS: Aplicación de Unix que trata de presentar la interfaz GSM mediante el uso de software radio.

En este trabajo se utilizará la herramienta GNU Radio como software de procesado de señal, que se explica en la siguiente sección.

2.7 GNU RADIO

Se trata de una herramienta software libre y de código abierto, constituida por un conjunto de archivos y librerías que proporcionan bloques de procesado de señales, permitiendo así el diseño y la simulación de sistemas basados en software radio.

Esta herramienta software puede ser utilizada con hardware externo adicional (como puede ser el USRP, RTL2832, OsmosSDR...) brindando la posibilidad de implementar físicamente un sistema basado en software radio, o bien, puede ser utilizada en un entorno de simulación [21].

El funcionamiento de GNU radio se puede concebir como un grafo, donde los nodos simbolizan los bloques de procesado de señal, y la interconexión entre ellos determinará el camino que seguirá la señal comenzando en una fuente y terminando en un sumidero.

La siguiente imagen representa el funcionamiento de GNU radio en una aplicación en la que se dispone de una fuente de datos, de un sumidero de datos y de tres bloques que desempeñarán alguna función [19]:

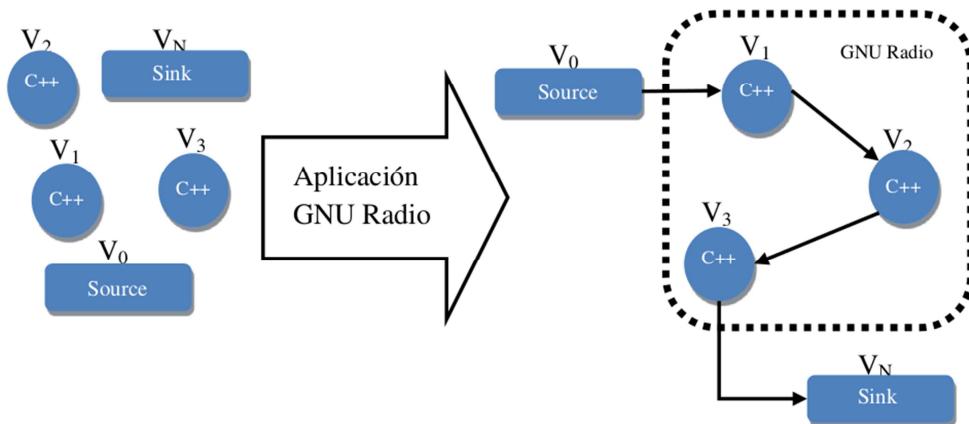


Figura 14. Interconexión en aplicación GNU Radio

Por lo tanto se puede realizar una primera clasificación en los tipos de bloques usados en GNU radio:

- Fuentes: Tales como ficheros, otros programas, hardware radio, micrófono.
- Sumideros: Tales como ficheros, otros programas, hardware radio, altavoces, visualizadores gráficos para poder ver la forma de onda de la señal, la FFT, etc.
- Bloques de procesado de señal: Tales como filtros, amplificadores, operadores lógicos, operadores matemáticos, moduladores, demoduladores...

Los bloques de GNU Radio se caracterizan por procesar los datos de manera continua desde su entrada hacia su salida, idealmente los bloques desempeñan únicamente una función para así hacer GNU Radio más flexible. Estos, son caracterizados por su número de puertos de entrada y de salida así como del tipo de dato que manejan. Las fuentes están caracterizadas por tener solo puerto de salida, mientras que los sumideros tienen solo puertos de entrada.

Los tipos de datos que se manejan son byte (1 byte de datos), short (2 bytes de datos), int (4 bytes de datos), float (4 bytes de datos para números en punto flotante), complex (8 bytes de datos, un float para cada componente). En cualquier aplicación de GNU Radio siempre tendrá que haber algún tipo de fuente y algún tipo de sumidero.

El procesado de señal y en general todo el trabajo a bajo nivel está implementado en C++, mientras que se hace uso de Python para escribir la aplicación, ocupándose de interconectar los bloques a usar. Para que desde un lenguaje de script como es Python se pueda acceder a las funciones implementadas en C++, se utiliza la herramienta software Simplified Wrapper and Interface Generator (SWIG).

La creación de la aplicación puede llevarse a cabo típicamente de dos formas:

La primera de ellas es programando directamente la aplicación en Python, mientras que la segunda opción consiste en diseñarla mediante la herramienta gráfica GNU Radio-companion. La segunda opción surge como necesidad de facilitar en la medida de lo posible la tarea al usuario, minimizando así la programación de la aplicación. Por ello, este documento se centrará en el uso de esta herramienta.

Típicamente los bloques de procesado de señal se implementan en cuatro tipos de archivos:

- Archivos .xml: En ellos se definen los parámetros del bloque como el tipo de dato a utilizar, el número de puertos de entrada y de salida, las librerías etc. (Requisito necesario para que el bloque esté disponible en GNU Radio-companion).
- Archivos .h: Son las bibliotecas de los bloques de procesado de señal
- Archivos .cc: Son los archivos que contienen la función que desempeñará el bloque de procesado de señal, se escriben en C++.
- Archivos .i: Son los archivos encargados de la comunicación entre los bloques de procesado de señal y la interfaz en Python.

La siguiente imagen muestra la arquitectura software de GNU radio [22]:

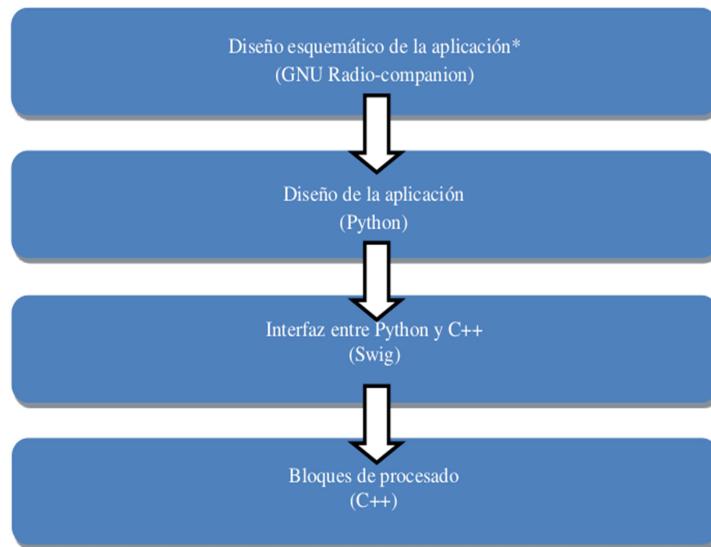


Figura 15. Arquitectura GNU Radio

GNU Radio funciona sobre plataformas Linux, Mac y Windows, en el apéndice A se indican los pasos a seguir para la correcta instalación de dicha herramienta en Linux, así como la correcta configuración de esta para poder utilizar el hardware del que se dispone en este proyecto.

Con motivo de una mejor comprensión en cuanto a los niveles de abstracción utilizados en la arquitectura software utilizada por GNU Radio se presenta el apéndice B, donde se explica el porqué de esta arquitectura.

Una vez se ha presentado el modo en el que funciona la herramienta y su arquitectura software, queda por presentar de qué manera se agrupan las librerías y archivos que conforman GNU Radio. Dichas librerías y archivos son agrupados en módulos dependiendo de la función que desempeñen, los principales módulos que presenta son [19]:

GNU Radio project	
gr	Módulo principal de GNU Radio, esta se necesitará prácticamente en todos los casos puesto que contiene bloques básicos como gran parte de las fuentes, gran parte de los sumideros así como funciones fundamentales como adición, sustracción...
digital	Este módulo contiene las librerías y archivos que se encargan de llevar a cabo las modulaciones y demodulaciones digitales.
audio	Este módulo proporciona control sobre la tarjeta de sonido, permite enviar o recibir señales de audio a través de la tarjeta de sonido.
blocks	Este módulo contiene bloques de procesado comúnmente utilizados en los flow graphs
blks2	Este módulo contiene bloques adicionales escritos en python
trellis	Este módulo provee los archivos necesarios para poder realizar codificaciones convolucionales.
analog	Es en este módulo donde se ubican los archivos relativos a las modulaciones analógicas.
wavelet	Este módulo proporciona bloques de procesado para las trasformadas wavelet.
fft	Este módulo proporciona bloques de procesado para las Fast Fourier Transform (FFT)
window	Este módulo contiene rutinas para el diseño de ventanas.
optfir	Este módulo contiene rutinas para el diseño óptimo de filtros de respuesta al impulso finita (FIR)
filter	Este módulo proporciona bloques de procesado para operaciones de filtrado.
qtgui	Módulo que proporciona sumideros gráficos basados en QT.
wxgui	Módulo que proporciona una GUI basada en Wx.
grc	Módulo necesario para poder utilizar la interfaz gráfica gnuradio-companion.
video_sdl	Este módulo proporciona control para permitir enviar o recibir señales de video.
vocoder	Este módulo incluye varios bloques de procesado los cuales implementan vocoders.
uhd	Este es el módulo que sirve de interfaz a la librería UHD para poder transmitir o recibir datos de los USRP.
How to write a block	Este módulo contiene información para la creación de nuevos módulos e incluirlos al proyecto de gnuradio.
out-of-tree	
osmosdr	Este módulo proporciona el soporte necesario para el uso del hardware osmoSDR.
baz	Este módulo añade nuevas funcionalidades al proyecto GNU Radio, como por ejemplo soporte para el hardware RTL-2832.

Tabla 1. Módulos GNU Radio

Los módulos de GNU Radio son a su vez estructurados en carpetas, las cuales son las encargadas de agrupar las ya mencionadas librerías y archivos, típicamente un módulo en GNU Radio presenta la siguiente estructura [19]:

Módulo genérico de GNU Radio	Carpetas	Descripción:
	Apps	Esta carpeta contiene ejemplos y aplicaciones de prueba del módulo.
	Cmake	Esta carpeta contiene archivos de configuración necesarios para la correcta instalación del módulo.
	GRC	Esta carpeta contiene los diferentes archivos “.xml” de los bloques para poder usarlos en la aplicación GNU Radio-companion.
	Include	Esta carpeta contiene los archivos fuente de las librerías “.h” de los bloques de procesado.
	Lib	Esta carpeta contiene los archivos fuente “.cc” de los bloques de procesado.
	Python	Esta carpeta contiene los diferentes scripts de Python
	Swig	Contiene los archivos swig “.i” con la configuración del intérprete de C++ y Python

Tabla 2. Estructura de un módulo GNU Radio

Para terminar de presentar GNU Radio, faltaría por hablar de que parte del procesado en un sistema tradicional es llevado a cabo en esta herramienta. La siguiente figura muestra las funciones llevadas a cabo por GNU Radio en la cadena de transmisión y recepción [3].

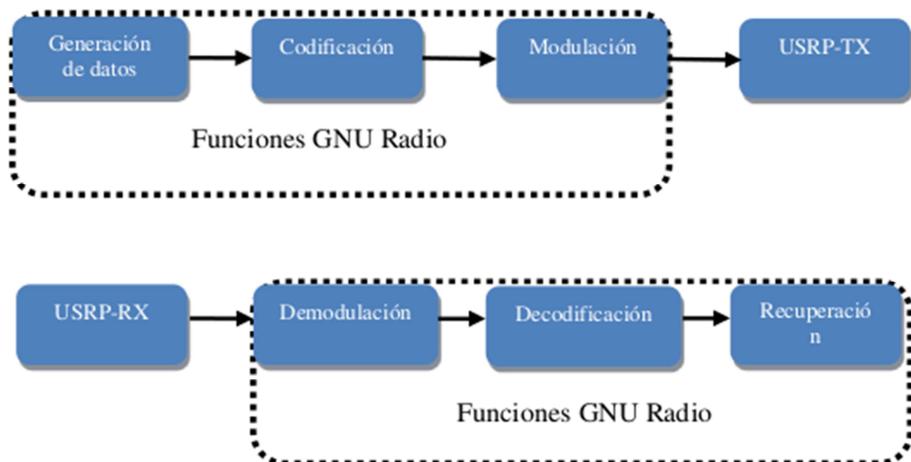


Figura 16. Funciones desarrolladas en GNU Radio en la cadena del transmisor y receptor

2.8 Relación Señal a Ruido

En comunicaciones analógicas es común usar la relación señal a ruido (SNR o S/N) que compara el nivel de la señal transmitida respecto al ruido que se encuentra en el canal de comunicación. En el caso de las comunicaciones digitales, se transmiten símbolos usando una forma de onda dentro de una ventana de tiempo y es necesario comparar un sistema a nivel de bits. Es por esta razón que es necesario usar otro tipo de relación que tenga en cuenta la energía de la señal digital, que es de corta duración, en vez de la potencia de la señal, que es utilizada en la SNR para caracterizar señales analógicas de infinita energía [7].

La relación E_b/N_0 divide el nivel de energía por bit (E_b) entre la densidad espectral de potencia del ruido (N_0), en [23] se muestra la siguiente equivalencia

$$\frac{E_b}{N_0} = \frac{S \cdot T_b}{N/W} = \frac{S / R_b}{N/W} \quad (44)$$

Donde S es la potencia de la señal, T_b es el tiempo del bit que también puede expresarse como $1/R_b$ debido a que R_b es la tasa de transmisión de bits por segundo. Por la parte del ruido, N es la potencia del ruido y W es el ancho de banda.

2.9 Ruido Blanco Gaussiano Aditivo (AWGN)

El ruido blanco gaussiano aditivo (AWGN) es un modelo básico producido por interferencias físicas presentes en la naturaleza como lo es la radiación de cuerpo negro o vibraciones térmicas de los átomos conductores [7]. El ruido blanco tiene la propiedad de que su densidad espectral de potencia es una constante en todas las frecuencias, mientras el ruido gaussiano cumple con la siguiente distribución

$$f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (45)$$

El AWGN muestra una distribución normal en el dominio del tiempo cuyo promedio es cero.

$$f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \quad (46)$$

Un canal modelado con ruido AWGN se expresa como

$$r(t) = s(t) + n(t) \quad (47)$$

En la ecuación 30, $r(t)$ es la señal recibida, $s(t)$ la señal transmitida y $n(t)$ es el ruido blanco gaussiano simulado. En la figura 17 se muestra el diagrama de bloques del modelo de canal.

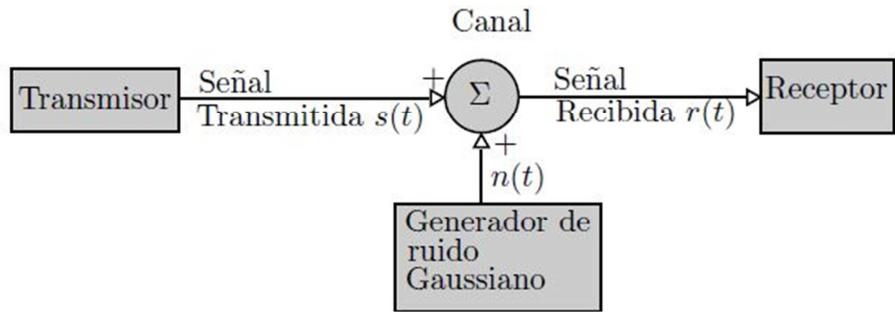


Figura 17. Simulación del canal AWGN

Capítulo 3

Diseño e Implementación en GNU Radio

En este capítulo se mostrará el diseño e implementación del modulador y demodulador 16-CQAM generados con la herramienta GNU Radio companion. Estos diseños se llevarán a cabo teniendo en cuenta las plataformas de hardware disponible.

En primer lugar, se realizara el diseño teórico de la constelación cantor, para luego mostrar por medio de bloques, como una función que históricamente recaía en la parte hardware, ahora es implementado vía software, se abordará los bloques encargados del sincronismo y finalmente, se mostrará el modulador y demodulador 16-CQAM.

A la hora de diseñar sistemas, es importante conocer los bloques de procesado disponibles en la herramienta, para ello GNU Radio ha creado una API [22] donde se podrá consultar información acerca de los módulos construidos en C++.

3.1 Constelación CQAM

En una constelación 16-QAM, el radio α relaciona la distancia de los estados más próximos de diferentes cuadrantes (distancia b), entre la distancia de los estados que se encuentran en un mismo cuadrante (distancia c). A un mayor valor de α , los estados pertenecientes a un determinado cuadrante se aíslan de los demás y por lo tanto, es más fácil reconocer los bits más significativos.

Los factores de escala f_i del conjunto de Cantor Generalizado, ofrecen una infinidad de posibilidades de obtener los estados de constelaciones jerárquicas que posean diferentes distancias b y c. En la figura 18, se puede observar las distancias b y c de una constelación CQAM de cuatro estados o C2. Se analizara una constelación C2 debido a que es la representación de una constelación 16-CQAM en una sola dimensión.

Para calcular las distancias y obtener el mapeo binario de los centros de los intervalos, se utiliza la ecuación 26 y se escoge $f_i = 3$ (para la utilización de conjuntos de cantor ternarios), de la cual da como resultado las coordenadas o puntos de la constelación jerárquica para 16-CQAM como se muestra en la siguiente figura 18.

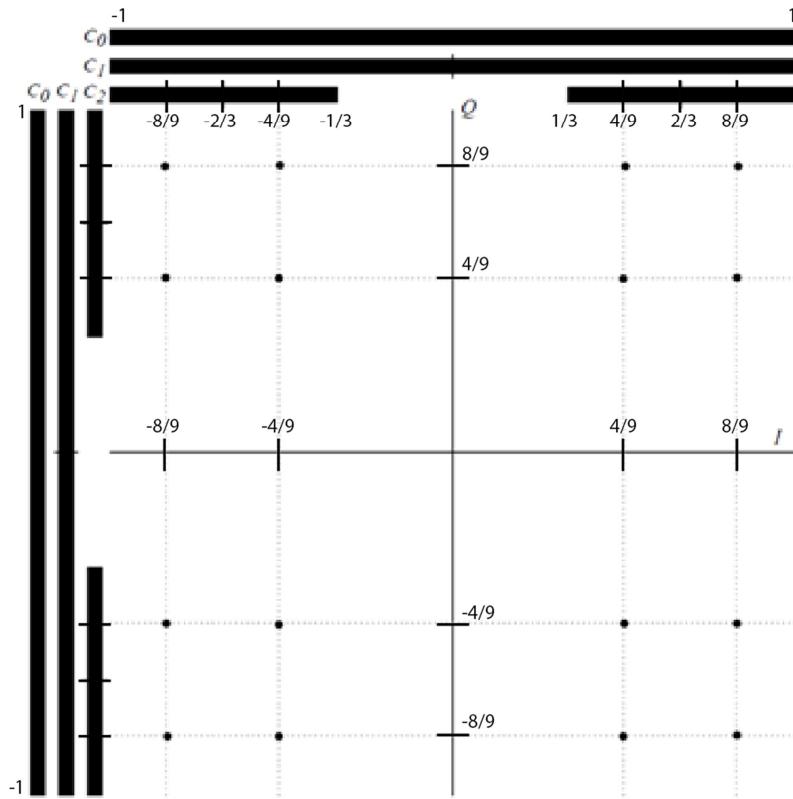


Figura 18. Diseño constelación CQAM

Los puntos de la constelación encontrada y de acuerdo a la figura 18 son: $(-\frac{8}{9} - \frac{8}{9}j, -\frac{8}{9} - \frac{4}{9}j, -\frac{8}{9} + \frac{4}{9}j, -\frac{8}{9} + \frac{8}{9}j, -\frac{4}{9} - \frac{8}{9}j, -\frac{4}{9} - \frac{4}{9}j, -\frac{4}{9} + \frac{4}{9}j, -\frac{4}{9} + \frac{8}{9}j, \frac{4}{9} - \frac{8}{9}j, \frac{4}{9} - \frac{4}{9}j, \frac{4}{9} + \frac{4}{9}j, \frac{4}{9} + \frac{8}{9}j, \frac{8}{9} - \frac{4}{9}j, \frac{8}{9} + \frac{4}{9}j, \frac{8}{9} + \frac{8}{9}j)$

Al llevar el modelo a la implementación en GNU Radio es necesario entender que dentro de este entorno se va realizar un pre-procesamiento para poder ajustar el transmisor y receptor a las técnicas de modulación utilizadas en SDR y en el entorno propio de GNU Radio, en este caso a un modulador y demodulador jerárquico CQAM

3.2 Bloques propios del ambiente de GNU Radio

GNU Radio requiere para su interfaz de trabajo algunos bloques que sirven de parámetros iniciales de configuración y fuentes de datos para el procesamiento, los bloques utilizados en este caso se muestran a continuación, en una sección posterior se presentan los bloques directamente relacionados con parámetros y componentes de una modulación convencional, allí se realizará una comparación y detalle de la función que cumple cada bloque, sin embargo es importante primero conocer algunos bloques funcionales que se puede incluir dentro del entorno de GNU Radio.

3.1.1 Tipos de bloques funcionales

Por otra parte y antes de explicar detalladamente los bloques que componen el modulador y demodulador implementado, es importante mencionar que en el entorno de trabajo de GNU Radio solo existen los siguientes tres tipos de bloques:

- Source: Bloques sin entrada, que tiene fuente de información o de señales periódicas.
- Proceso: Bloques que tienen varias entradas y varias salidas.
- Sink: Bloque que solo tiene entradas.

Cada uno de los bloques debe ser conectado sin dejar puertos libres o se producirán errores. Sin embargo, si se quiere dejar un bloque sin conectar se deben utilizar los bloques *Null Sink* y *Null Source*. En la categoría *Misc* se encuentran la mayoría de los bloques generales.

En la mayoría de estos bloques es necesario especificar la tasa de muestreo que se esté manejando en la señal. Esto porque el programa simplemente interpreta una cantidad de muestras de datos y necesita la información de la representación del tiempo y/o de la frecuencia de la señal que está recibiendo. Por lo que aparecen los conceptos de Interpolación y Decimación.

La Interpolación es el proceso en que se aumenta la frecuencia de muestreo de una señal digital agregando muestras entre las muestras originales de la señal. Por otra parte, la decimación permite disminuir la frecuencia de muestreo quitando muestras intermedias de la señal. Estos elementos serán ampliamente utilizados en el trabajo de señales digitales en GNU Radio. Por ejemplo, una señal de audio que quiere ser escuchada debe estar a una tasa de 48 Kbps para que pueda ser reproducida por los parlantes de un computador. Por otra parte, hay que notar a qué medida que se tenga una mayor cantidad de muestras, se consumirán mayor cantidad de recursos del PC, así que en algunos casos, se preferirá tener una tasa de muestreo más baja.

3.1.2 Bloque de Opciones del Workspace

En primer lugar, GNU Radio cuenta con un bloque de opciones en donde se debe especificar el ID, Titulo, el tamaño de la ventana, y lo más importante el tipo de flujograma (bloques de interfaz gráfica) que se está diseñando (figura 19).

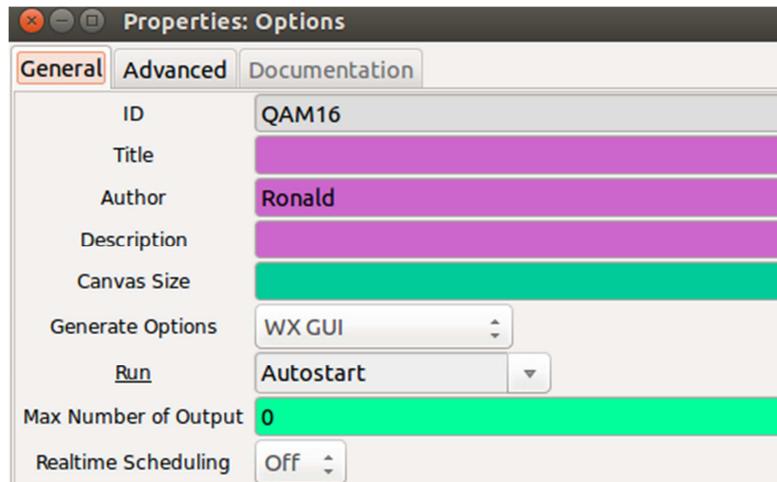


Figura 19. Parámetros de configuración del menú: Generate Options

QT GUI y WX GUI: hacen referencia al tipo de librerías que se utilizaran para la interfaz gráfica, QT consume menos recursos de maquina

NO GUI: es para hacer un flujograma sin interfaz gráfica y así no consumir tantos recursos de máquina

HIER BLOCK: es para crear un bloque jerárquico que será almacenado dentro del programa y podrá ser usado en otros programas.

3.2 Fuentes de Información (Bloques: *File*, *Wav File* y *Signal Source*)

Los bloques File Source y Wav File Source leen los valores de datos en formato binario desde el archivo especificado, para el bloque Wav File Source, el archivo se debe encontrar en formato de audio sin compresión wav⁷, los parámetros de los bloques se muestran a continuación:

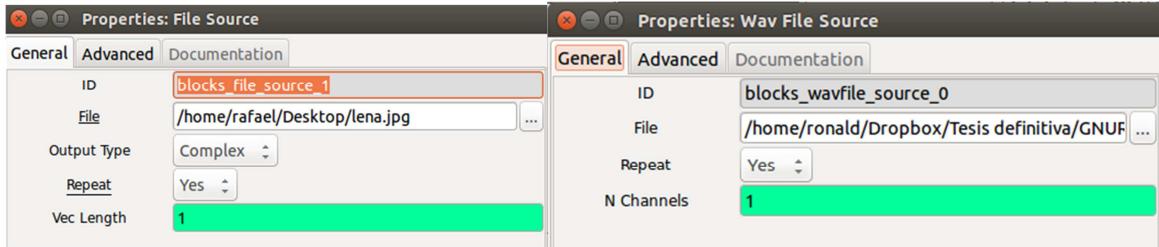


Figura 20. Parámetros bloque File Source, Wav File Source

Un parámetro importante de estos bloques es el de Repeat, este parámetro en caso de que esté habilitado generará la fuente indefinidamente mientras el flujograma este en ejecución.

La fuente de transmisión que se pretende transmitir puede ser de tipo wav source, una imagen en sus diferentes formatos (BMP⁸, JPG⁹ y PNG¹⁰) o una señal cosenoidal. La tasa de muestreo para la fuente de audio será de 22.05 Khz. Cada una de las 22.05 KS/s que proporciona esta fuente es de tipo *float*, es decir, 4 bytes. Teniendo en cuenta eso, el régimen binario que en principio se necesitaría seria de 705.60 Kbps para transmitir correctamente el fichero .wav

El bloque Signal Source genera diferentes tipos de señal, en este caso una señal cosenoidal con tasa de muestreo correspondiente a la señal de audio, además de otros parámetros básicos importantes en la verificación del funcionamiento del modulador y demodulador (figura 21).

⁷ Waveform Audio File Format

⁸ Bitmap

⁹ Graphics Interchange Format

¹⁰ Joint Photographic Group

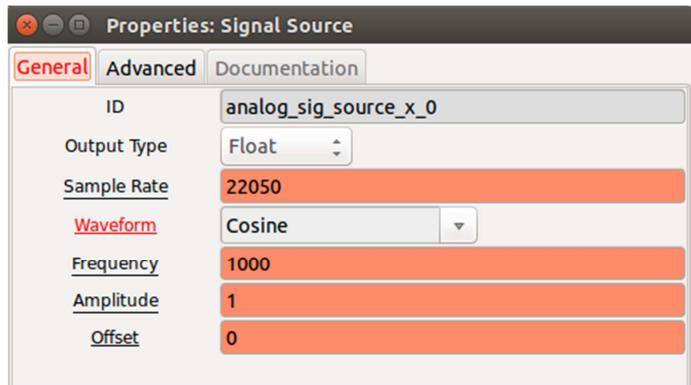


Figura 21. Parámetros bloque Signal Source

3.2.1 Bloque limitador (Throttle)

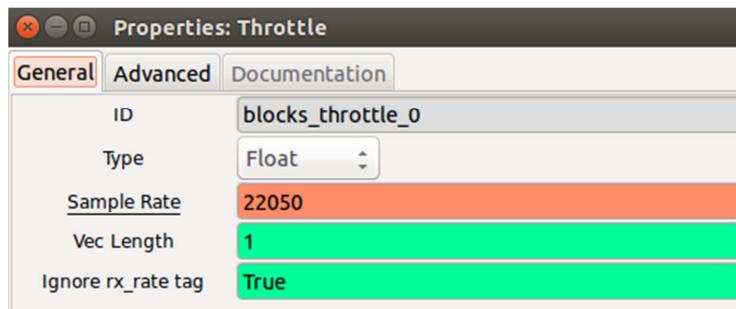


Figura 22. Parámetros bloque Throttle

Este bloque limita la transferencia de datos a la velocidad de muestreo especificado. Esto evita a GNU Radio consumir todos los recursos de la CPU cuando el flujograma no está siendo regulado por hardware externo (USRP o Dispositivo SDR). Los parámetros de este bloque se muestran en la figura 22.

3.3 Bloques del Modulador

En esta sección se entrara en detalle en los bloques del flujograma usados para la implementación del modulador y se realizará una comparación con los bloques equivalentes de un sistema convencional de transmisión con modulación jerárquica CQAM.

La siguiente figura muestra el diseño realizado para un modulador 16-CQAM:

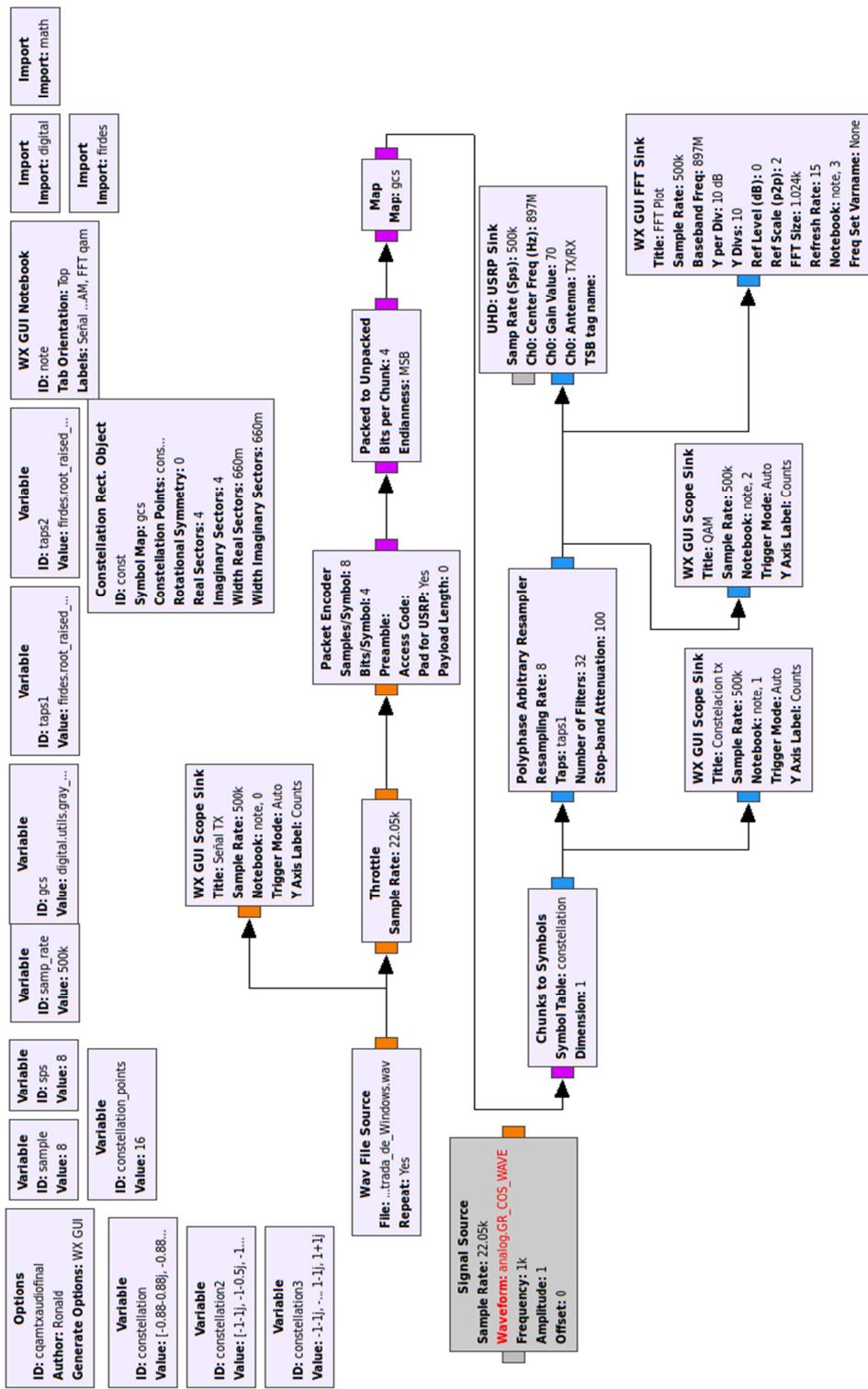


Figura 23. Diseño modulador 16-CQAM

3.3.1 Empaqueamiento de datos

Inicialmente en todo sistema de comunicaciones es importante definir el tamaño del bus de datos en particular para la modulación 16-CQAM que se plantea implementar en este caso, las configuraciones deben permitir un tamaño de paquete de datos de 4 bits.

3.3.1.1 Bloque empaquetamiento de datos (Packet Encoder)

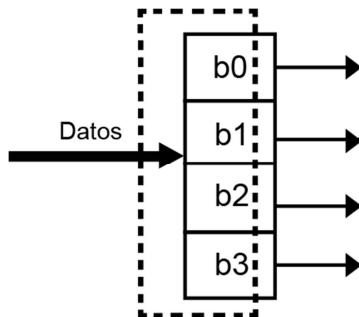


Figura 24. Funcionalidad del bloques Packet Encoder en la modulación convencional

El objetivo principal de este bloques consiste en separar la data en un número de bits que debe corresponder a la tasa de bits por símbolos del tipo de modulación que se va a implementar, para el caso de este documento se separan en grupos de 4 bits [24], la siguiente figura muestra las propiedades de este bloques:

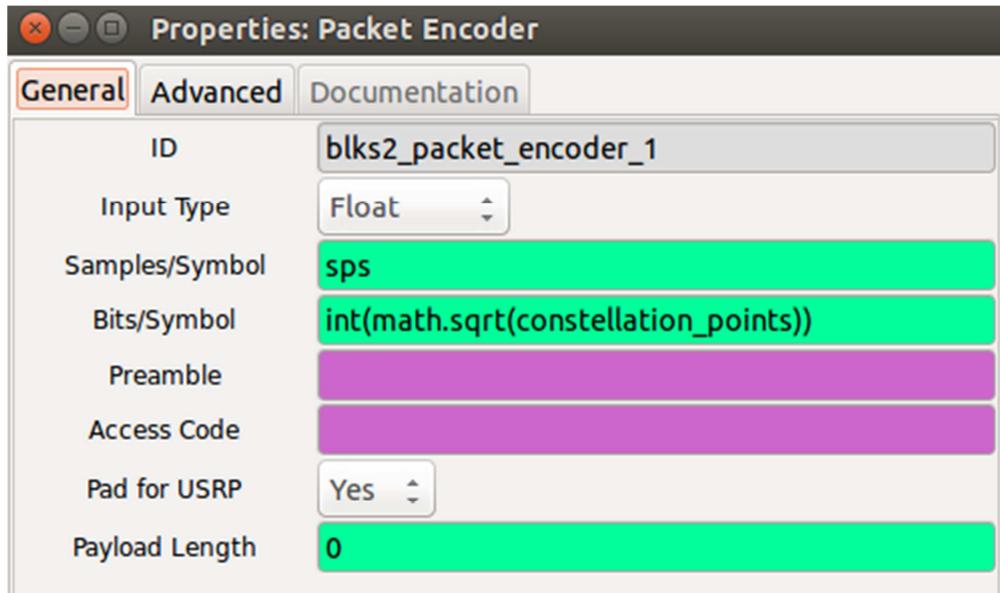


Figura 25. Parámetros bloques Packet Encoder

Se tienen variables importantes: *sample* que define el número de muestras por símbolo (*sample/symbol*) que será la misma variable para todo el flujograma y como es de esperar toma el muestreo de todo el flujograma, y *bits* (*Bits/Symbol*) que en este caso es:

$$Bits/Symbol = \sqrt{M} = \sqrt{16} = 4 \quad (48)$$

Siendo *M* el número de símbolos que se traduce en los puntos de la constelación, en este caso 16. Esto ocasiona que la información del *file source* se divida y transmita al final del bloque en grupos de datos de 4 bits.

Los parámetros de Preamble y Access Code se utilizan en el caso de que se requiera realizar sincronización de paquete mediante la verificación por redundancia cíclica (CRC).

3.3.1.2 Bloque asignación de prioridad de datos (Packet to Unpacked)

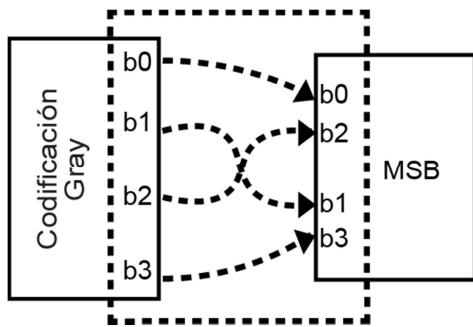


Figura 26. Funcionalidad del bloque Packet to Unpacked en la modulación convencional

Este bloque cumple la funcionalidad de asignar la *Endianess*¹¹ de los datos a transmitir, para realizar este proceso se debe tratar cada bit de forma separada, por lo que es necesario especificar la cantidad de bits que van a ser codificados.

Desempaquetar los bytes significa colocar los bits que se han empleado para definir cada símbolo en un nuevo byte en los bits en este caso más significativos (MSB). Las propiedades de configuración de este bloque se muestran a continuación:

¹¹ Este concepto se refiere al formato de escritura y lectura de los datos.

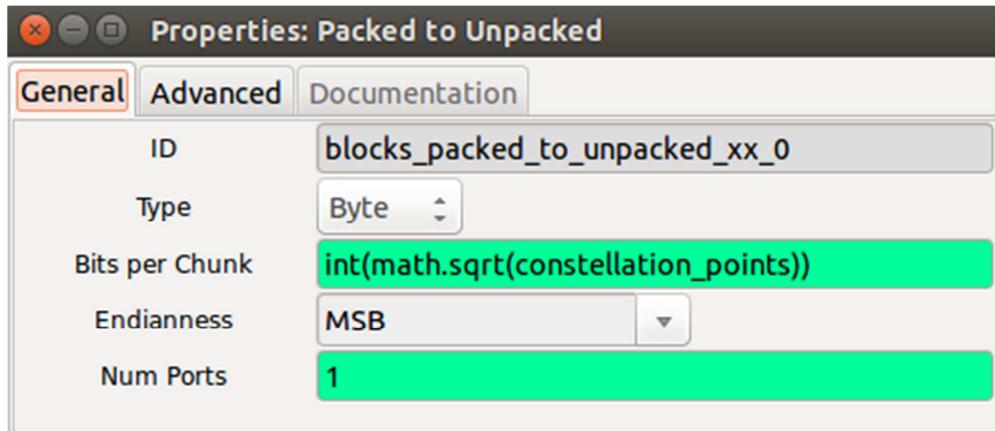


Figura 27. Parámetros bloque Packed to Unpacked

Se sigue utilizando la variable *bits* para definir la cantidad de datos a asignar su *Endianness* y se escoge *Endianness* MSB, en este caso para la modulación y así mismo debe ajustarse para la demodulación.

3.3.2 Codificación

Se realizará a continuación la descripción de los bloques encargados de realizar la codificación de la señal de entrada.

3.3.2.1 Bloque de codificación en Gray (Map)

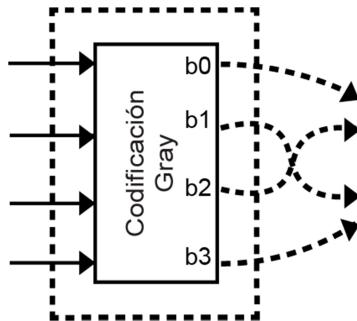


Figura 28. Funcionalidad del bloque Map en la modulación convencional

El siguiente bloque en la cadena del modulador es el encargado de realizar la codificación o mapeo de los bits a transmitir en código Gray, después de asignada la *Endianness* del mismo, esta codificación ha sido utilizada por su sencillez y eficiencia en la codificación de datos para combatir ruido cuando solo cambia uno de los bits transmitidos [7], la configuración del bloque se muestra a continuación:

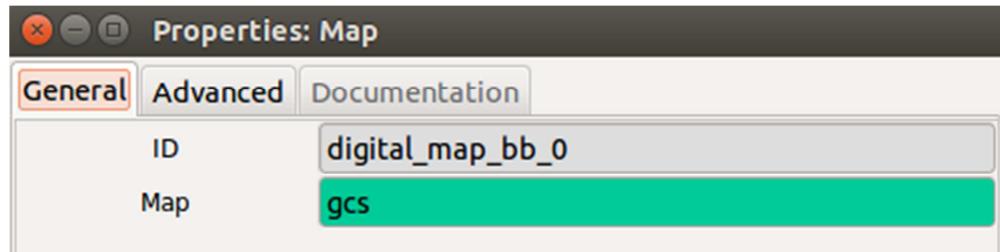


Figura 29. Parámetros bloque Map

Donde el parámetro que se le indica almacenado en la variable previamente definida *gcs* es el código Gray de tanta longitud como símbolos existan, los parámetros de esta variable se muestran a continuación:

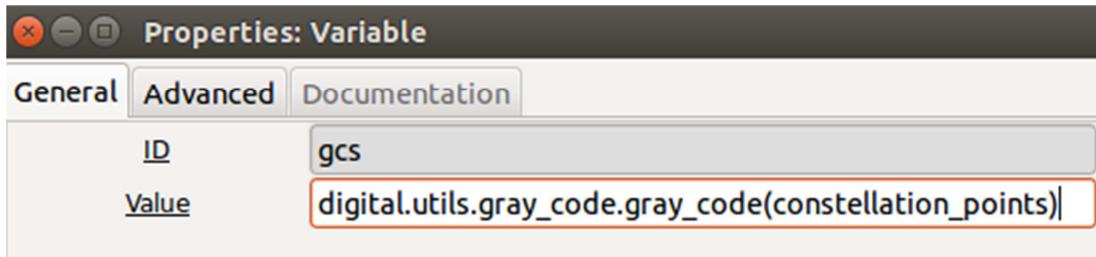


Figura 30. Parámetros variable gcs

El bloque *map* mapea una señal de entrada a un valor en el mapeador (en este caso asigna una posición en código gray al byte ingresado de acuerdo a la función de mapeo *gcs*) y siendo la variable *constellation_points* igual a 16, el algoritmo del método en phyton *digital.utils.gray_code.gray_code(constellation_points)* que a su vez invoca al método en C++ *digital.graycode* generara la codificación gray para los 16 símbolos de 4 bits.

3.3.2.2 Bloque codificador en banda base (Chunks to Symbols)

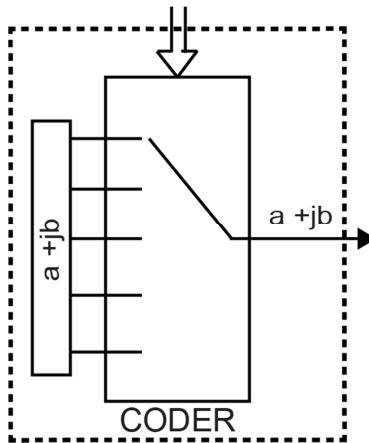


Figura 31. Funcionalidad del Bloque Chunks to Symbols en la modulación convencional

Llegados a este punto, la información disponible sigue en byte, el bloque que se encarga de mapear este flujo de bytes en muestras complejas (fase y cuadratura) es el *Chunks To Symbols*:

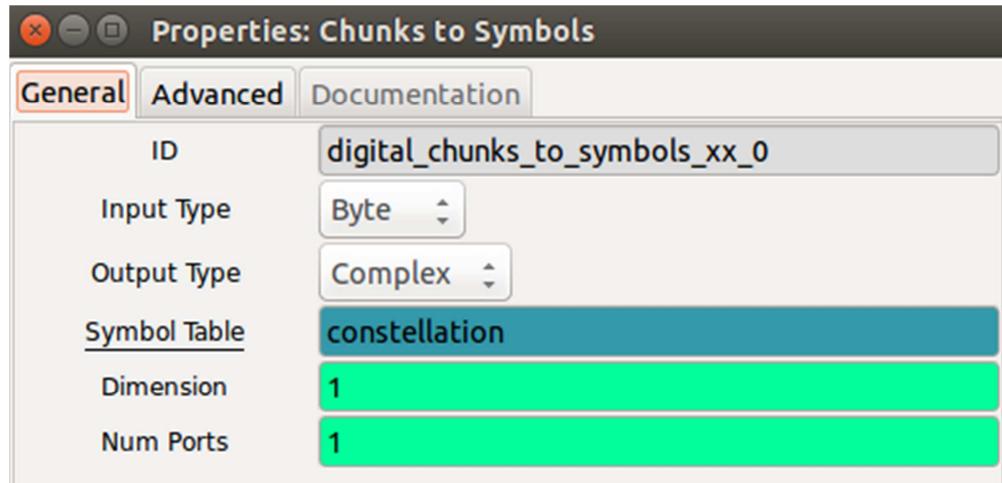


Figura 32. Parámetros bloque Chunks to Symbols

Este bloque, es el que lleva a cabo la codificación QAM (en general muchos de los esquemas de modulación pueden ser creados a partir del bloque *packed_to_unpucke*d seguidas de *chunk_to_symbols*). Este bloque recibe una cadena de bytes desempaquetados (lista de símbolos) y se encarga de proporcionar una cadena de muestras complejas. Para ello se realiza un mapeo 1 a 1 de los símbolos de entrada a puntos de constelación. En este caso al realizarse el mapeo 1 a 1 de los símbolos el parámetro que define la *Dimensión* ha de permanecer fijado a 1, en este caso y por facilidad de cómputo, los valores de la constelación son directamente calculados y operados en el espacio complejo, sin necesidad de tratar por aparte el espacio I y Q (fase y cuadratura), reduciendo la cantidad de procesos computacionales. El bloque de *chunks to symbols* espera los puntos de la constelación, en este caso son los puntos calculados en la sección 3.1 para los esquemas de modulación CQAM y QAM.

La variable *constellation* tendrá los puntos de la constelación deseada que se deberán mapear con la información a su entrada como se muestra en la figura 33.

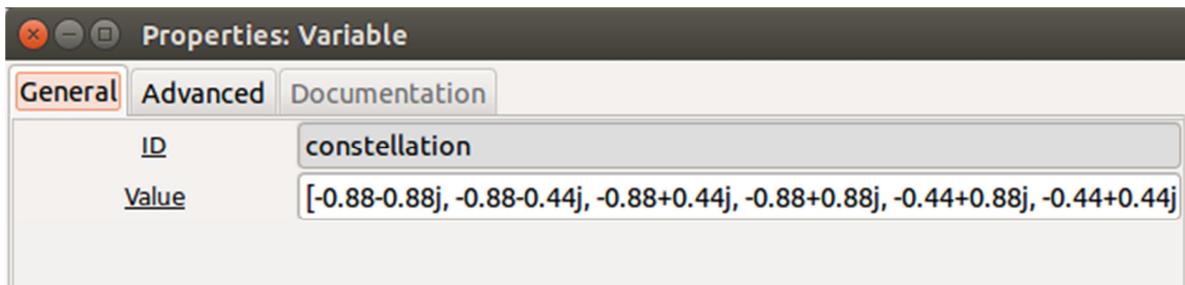


Figura 33. Parámetros variable constellation

En general, el objeto *constellation* puede ser generado a través de distintas clases que derivan de la clase padre o base con la intención de explotar las características geométricas da cada constelación a la hora de llevar a cabo la demodulación. El método invocado en el bloque de la figura 32, invoca a su vez a la clase *digital_constellation_rect* situada en el archivo *digital_constellation.cc* de código C++ encargada de crear el objeto *constellation*. Es necesario que desde *qam.py* en Python se llame a la función *constellation_rect* en C++, para que esto sea posible es necesario el fichero *digital_swig* encargado de la integración entre Python y C++ para los módulos digitales

Para poder crear la constelación desde el archivo *qam.py*, se le ha de indicar el índice de modulación y si se trata de una modulación diferencial así como si se desea codificación Gray, aunque por defecto no se realiza.

Es importante destacar que en este punto es donde se establecen los valores de la constelación que va a ser formada, en GNU Radio es necesario ajustar los valores del bloque con extensión *.xml* para poder generar puntos en el plano normalizado en el intervalo [-1,1].

3.3.1 Modulación

En este momento, ya se dispone de la señal M-QAM en banda base. Por motivos de reducción de la Interferencia entre Símbolos (ISI) se introduce un bloque el cual llevara a cabo la tarea de conformar el pulso mediante un filtro de transmisión en coseno alzado así como de ajustar la tasa de muestreo *sample rate* acorde a la que tiene establecida en el periférico SDR en la implementación en hardware.

3.3.3.1 Bloque de modulación por filtrado (Polyphase Arbitrary Resampler (filtro RRC)]

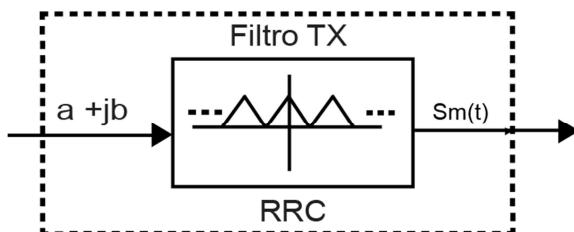


Figura 34. Funcionalidad del Bloque Polyphase Arbitrary Resampler en la modulación convencional

En este caso es donde se realiza el mayor cambio con el método de transmisión QAM convencional, esta forma moderna de transmitir los datos, se hace uso del muestreo en el tiempo que en frecuencia se refleja como una multiplicación del espectro y posteriormente se usa un filtro raíz de coseno alzado, el cual constituye un transmisor efectivo y confiable reduciendo la ISI, y es una de las formas modernas de realizar modulaciones digitales [25]

Este bloque, además de ser el encargado de conformar el pulso, será el que alinee las diferentes tasas del sistema en función de las muestras que se han definido por símbolo. Será el encargado por lo tanto de llevar a cabo la interpolación en diseños que operen con plataformas hardware. Anteriormente, en otras versiones de GNU Radio, este bloque no existía y se debía especificar el factor de diezmado y/o interpolación que se requería para alinear las tasas de muestreo [3].

El número de filtros debe estar alineado con el parámetro sample rate impuesto en el dimensionamiento del filtro RRC.

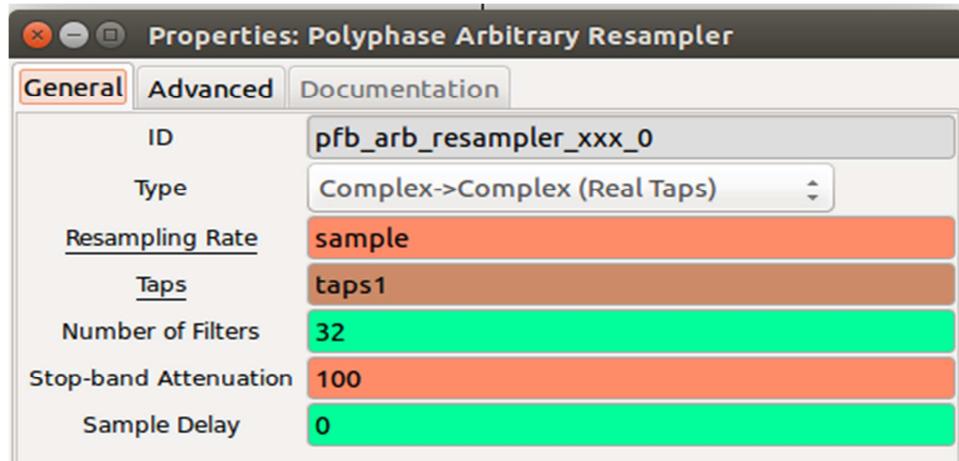


Figura 35. Parámetros bloque Polyphase Arbitrary Resampler

El parámetro taps1 se utiliza para describir el filtro de transmisión conformador de pulso que se va a utilizar, en este caso se trata de un filtro raíz coseno alzado:

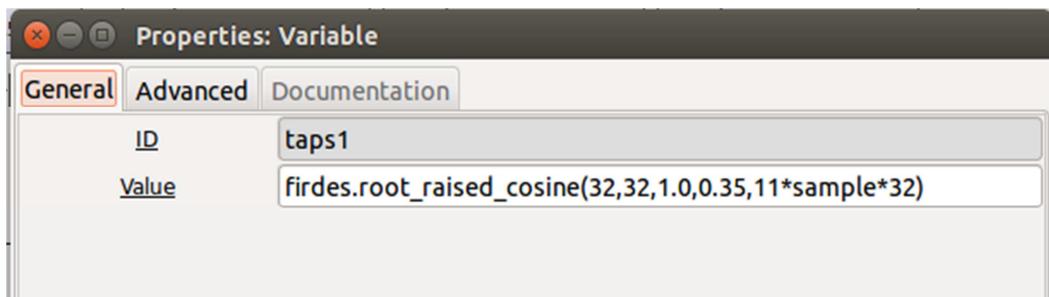


Figura 36. Parámetros filtro coseno alzado tx

Donde el primer parámetro indica la ganancia, el segundo es la *sample_rate* utilizada por el filtro, el tercero es el *symbol rate*, el siguiente es el parámetro *alpha* propio del filtro raíz coseno alzado y el último el número de coeficientes del filtro. Para el diseño del filtro se utilizó la herramienta de GNU Radio Filter Design Tool (figuras 37-38)

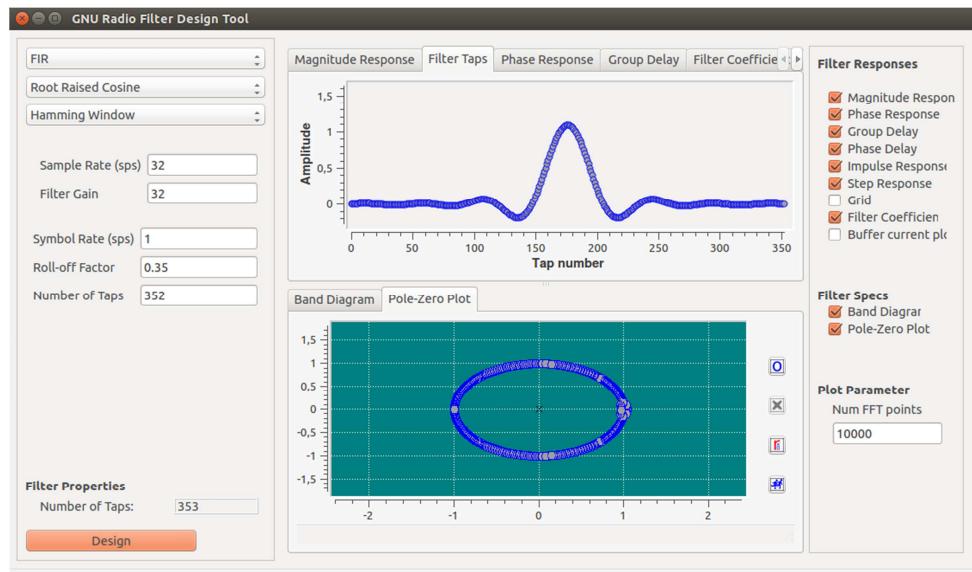


Figura 37. Diseño del filtro en GNU Radio Filter Design Tool

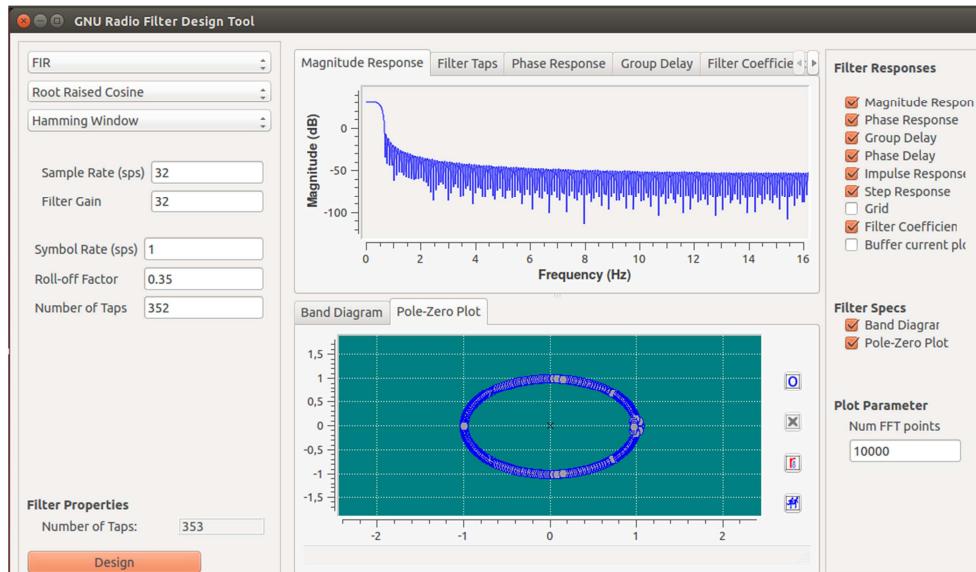


Figura 38. Diseño del filtro en GNU Radio Filter Design Tool – Respuesta en magnitud

Finalmente se sitúa el sumidero en el diseño, en este caso al estar en un entorno de simulación se coloca un sumidero virtual, cuya función es trasladar todos los datos de entrada a una fuente virtual (la cuál simulará ser la fuente de datos del receptor). [3]

En la implementación en hardware se coloca un bloque a la salida del filtro de transmisión llamado USRP Sink encargado de la conexión y comunicación entre el entorno de GNU Radio en un PC y el dispositivo Hardware SDR, los parámetros y funcionalidad se explicaran en el capítulo 4.

3.4 Bloques del Demodulador

En esta sección se entrara en detalle en los bloques del fluajograma usados para la implementación del demodulador y se realizará una comparación con los bloques equivalentes de un sistema convencional de recepción con modulación jerárquica CQAM.

La siguiente figura muestra el diseño realizado para un demodulador 16-CQAM:

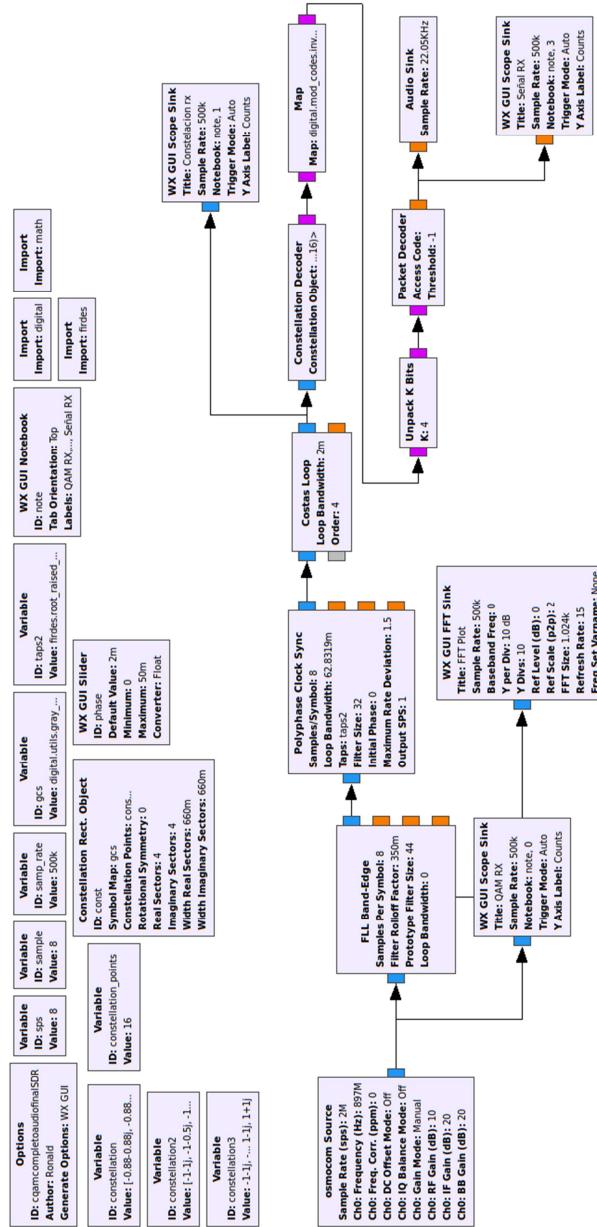


Figura 39. Diseño Demodulador 16-CQAM

3.4.1 Sincronización de Frecuencia y Fase

3.4.1.1 Bloque Sincronizador de frecuencia (FFL BAND EDGE)

Este bloque es de especial consideración para la implementación física (incluyendo hardware SDR), implementa un FLL (Frequency-Locked-Loop) digital, y se encarga de realizar el compensado en frecuencia o de ajustar las diferencias en frecuencia en cuanto a sintonización de portadora en frecuencia entre el transmisor y receptor.

La siguiente figura muestra las propiedades del bloque mencionado:

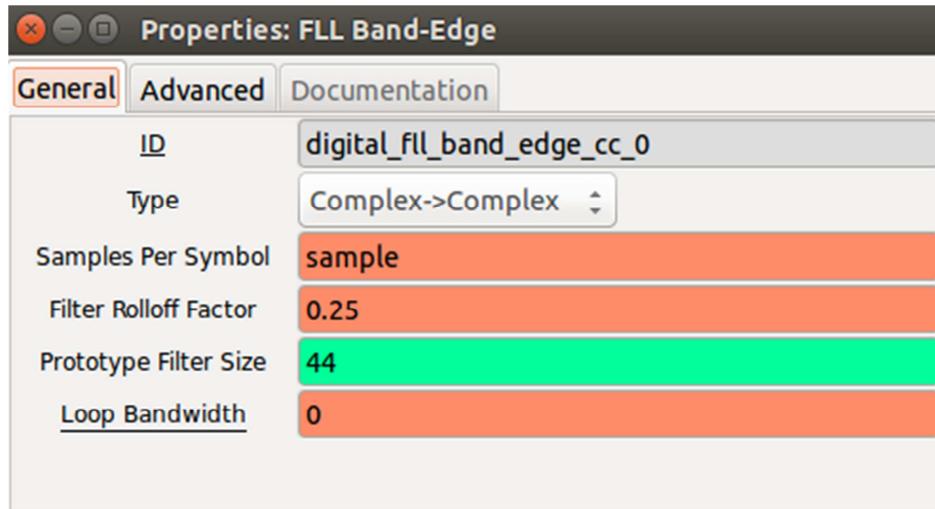


Figura 40. Parámetros bloque FFL Band-Edge

El sistema de control genera una señal de salida cuya frecuencia está relacionada con la frecuencia de entrada, esto lo hace mediante un filtro que cubre los anchos de banda superiores e inferiores de la señal modulada digitalmente. El intervalo de ancho de banda está determinado por el exceso de ancho de banda (por ejemplo, el factor de desplazamiento) de la señal modulada. La colocación en frecuencia de los bordes de banda se determina por la relación de sobremuestreo (número de muestras por símbolo) y el exceso de ancho de banda.

El FLL funciona filtrando los bordes de banda superior e inferior en $x_u(t)$ y $x_l(t)$, respectivamente. La combinación de estos forma la señal $e(t)$, que proporciona una señal de error directamente proporcional a la frecuencia portadora. Entonces se hace un bucle de segundo orden utilizando la señal de error que es el promedio de $e(t)$. Se hace uso de filtros FIR porque los filtros tienen que tener una respuesta de fase plana en toda la gama de frecuencias para permitir que sus comparaciones sean válidas. Es muy importante que los filtros sean derivados del filtro de conformación de impulsos, y que sean de fase lineal. De lo contrario, la varianza del error será muy grande [26] [27].

Los parámetros que se le indican son el número de muestras por símbolo, la longitud del banco de filtros, el ancho de banda del bucle y el rolloff del filtro.

3.4.1.2 Bloque filtro de Costas Loop

Este bloque implementa el algoritmo de Costas Loop con el fin de corregir los efectos que causa los desplazamientos en fase entre el transmisor y el receptor, también importante en la implementación física y en aplicaciones de receptores inalámbricos debido a que se adapta mejor en el seguimiento de la señal de entrada [28], las propiedades de este bloque son:

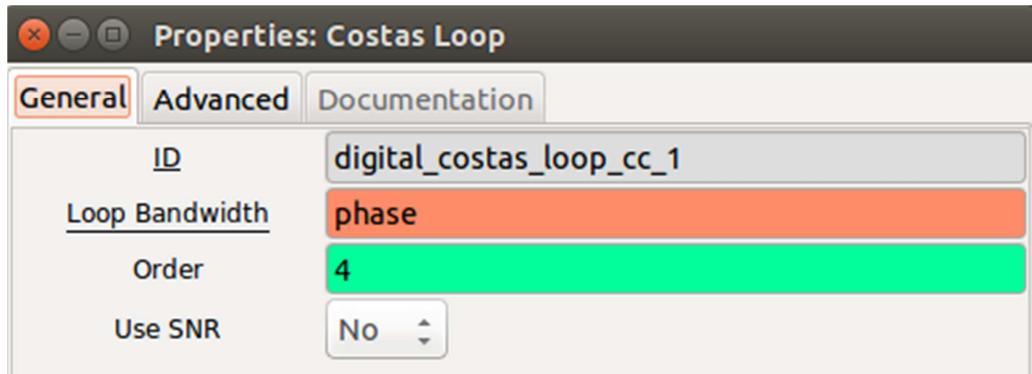


Figura 41. Parámetros bloque Costas Loop

Los parámetros que se le indican son el *order* que varía de 2, 4 u 8 y el ancho de banda del bucle.

3.4.2 Demodulación

Se realizará a continuación la descripción de los bloques encargados de realizar la demodulación de la señal transmitida.

3.4.2.1 Bloque de demodulación por filtro adaptado (Polyphase Clock Sync)

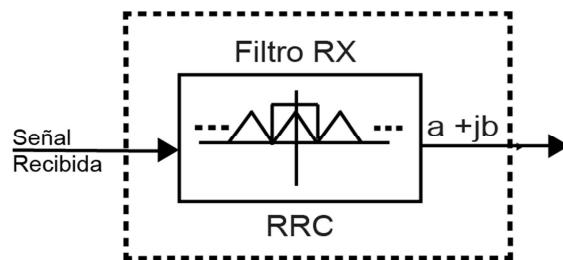


Figura 42. Funcionalidad del Bloque Polyphase Clock Sync en la modulación convencional

Una vez se atraviese el bloque encargado del compensado en frecuencia, se encuentra otro bloque que realiza el alineamiento en los tiempos de muestreo (transmisor y receptor).

Es en este mismo bloque donde aparece el filtro adaptado (*root raised cosine*) encargado de deshacer los efectos del filtro conformador de pulsos del transmisor:

Este bloque realiza sincronización de temporización para señales QAM minimizando la derivada de la señal filtrada, que a su vez maximiza la SNR y minimiza ISI. Este enfoque funciona mediante la creación de dos bancos de filtros; Un banco de filtros contiene el filtro adaptado de conformación de impulsos de la señal (tal como un filtro raíz coseno alzado), donde cada rama del banco de filtros contiene una fase diferente del filtro. El segundo banco de filtros contiene las derivadas de los filtros en el primer banco de filtros. Pensando en esto en el dominio del tiempo, el primer filtro contiene filtros que tienen una forma $\text{sinc}(t)$. Queremos alinear la señal de salida para ser muestreada exactamente en el pico de la forma $\text{sinc}(t)$. La derivada del sinc contiene un cero en el punto máximo del sinc ($\text{sinc}(0) = 1, \text{sinc}'(0) = 0$). Además, la región alrededor del punto cero es relativamente lineal. Hacemos uso de este hecho para generar la señal de error [29].

La señal de error $e[n]$, nos da un valor proporcional de lo lejos del punto cero que estamos en la señal derivada. Queremos llevar este valor a cero, así que establecemos un bucle de segundo orden. Tenemos dos variables para este bucle; D_k es el número de filtro en el banco de filtros en el que estamos y d_{rate} es la velocidad a la que viajamos a través de los filtros en estado estacionario. Es decir, debido a las diferencias de reloj natural entre el transmisor y el receptor, d_{rate} representa esa diferencia y recorrería los trayectos de fase del filtro para mantener el receptor sintonizado. Pensando en esto como un PLL de segundo orden, el d_{rate} es la frecuencia y d_k es la fase. Así que actualizando d_{rate} y d_k usando las ecuaciones de bucle estándar basadas en dos señales de error, d_{alpha} y d_{beta} . Tenemos estos dos valores establecidos basados uno en el otro para un sistema críticamente amortiguado, así que en el constructor de bloques, solo se pide "loop bandwidth", que es d_{alpha} mientras que d_{beta} es igual a $(\text{gain}^2) / 4$. [30]

Además de realizar las tareas ya mencionadas, su salida transforma no en cuanto a tipo (siguen siendo muestras complejas), si no en el número de muestras que definen cada símbolo (*output sample per symbol*). A la entrada de este bloque se emplean el número de muestras definidas por la variable *sample* (8 en este diseño) y la salida de este, solo se empleará una muestra por símbolo. Es por ello por lo que se había introducido el factor *sample* en el parámetro *sample rate* del filtro adaptado [3] [30]. Los parámetros del bloque se muestran en la figura 43.

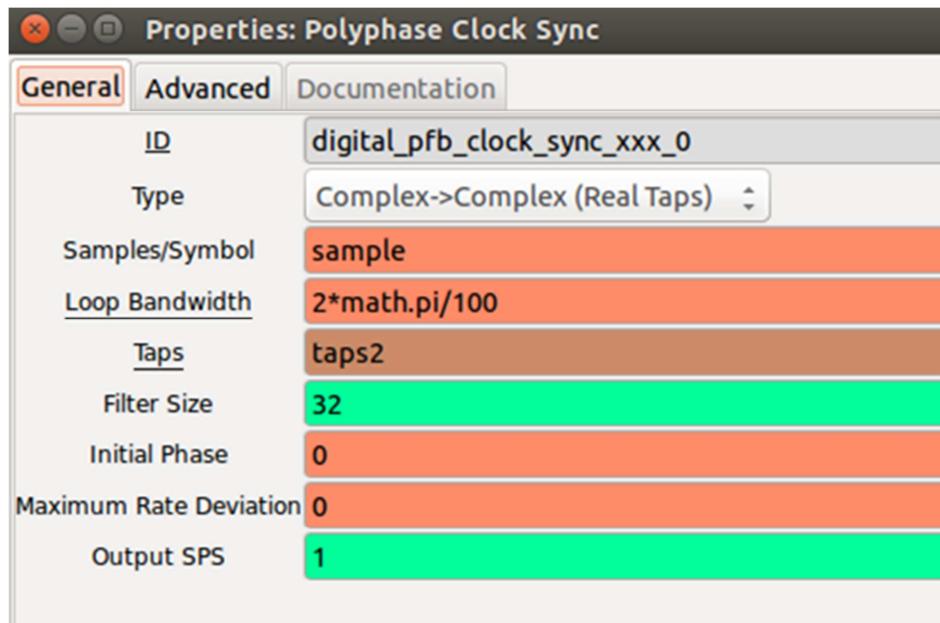


Figura 43. Parámetros bloque Polyphase Clock Sync

El parámetro `taps2` se utiliza para describir el filtro de recepción conformador de pulso que se va a utilizar, en este caso se trata de un filtro en raíz de coseno alzado:



Figura 44. Parámetros filtro coseno alzado rx

Donde el primer parámetro indica la ganancia, el segundo es la `sample_rate` utilizada por el filtro, el tercero es el `symbol rate`, el siguiente es el parámetro `alpha` propio del filtro raíz coseno alzado y el último el número de coeficientes del filtro.

3.4.3 Decodificación

En esta sección presentamos el decoder el cual requiere de dos módulos principales: `constellation decoder` y `constellation Rect Object`, ambos son dependientes y básicamente cumplen la función de devolver la data desde el dominio o representación compleja a su representación binaria.

3.4.3.1 Bloque definición de la constelación recibida (Constellation Rect Object)

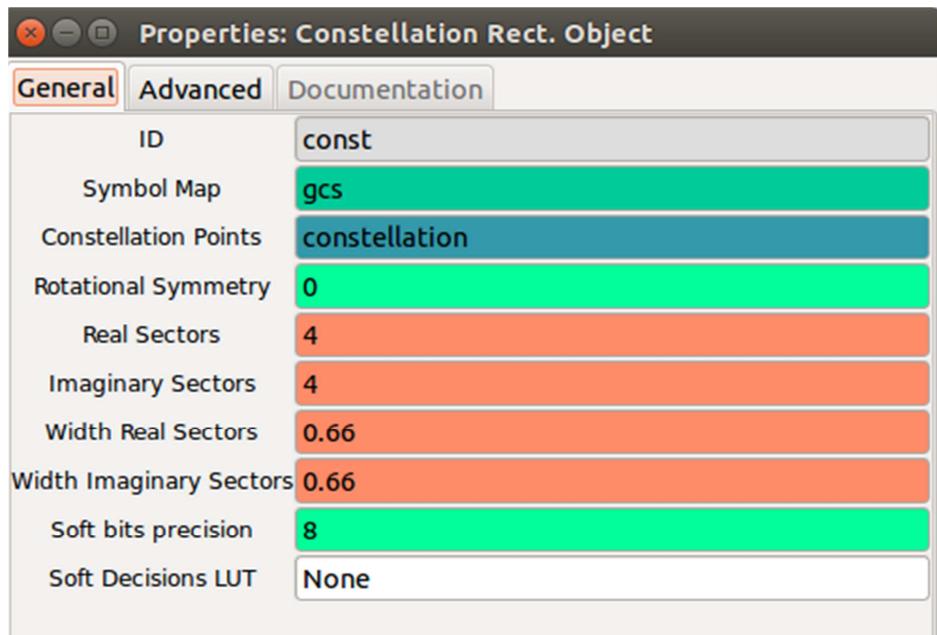


Figura 45. Parámetros Bloque Constellation Rect. Object

Este módulo facilita el acceso y la configuración personalizada de una constelación. El parámetro de Ancho de sectores reales e imaginarios se ha modificado para que pueda tomar valores en coma flotante, en este caso 0.66, que corresponde al ancho de cada sector en la constelación CQAM en el intervalo [-1, 1], para un esquema de modulación de 16 estados el número de sectores reales e imaginarios debe ser igual a cuatro.

Los parámetros *Symbol Map* y *Constellation Points* contienen las variables *gcs* y *constellation* explicadas en la sección 3.3.2

La explicación del porqué de la creación del objeto constelación (Constellation Rect Object) a través de diferentes clases reside principalmente con la intención de reducir el coste computacional en el cálculo de las distancias de las muestras a los puntos definidos en la constelación, cada modulación explota las características geométricas de la propia constelación. En particular, en las modulaciones tipo QAM, el cálculo se lleva a cabo por sectores. En el constructor, el espacio que define a la constelación es dividida en sectores cuadrados y cada uno de ellos es asociado a un punto de la constelación.

3.4.3.2 Bloque decodificador de constelación (Constellation Decoder)

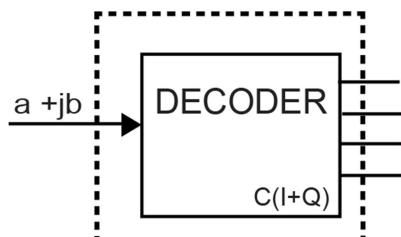


Figura 46. Funcionalidad del Bloque Constellation Decoder en la modulación convencional

Llegados a este punto, los siguientes bloques se encargarán de realizar la función inversa a la realizada en el *chunk to symbols*, por consiguiente el siguiente bloque en la cadena es el *constellation decoder*, al que le llega una cadena de muestras complejas y las mapeará (demodulará) ofreciendo a su salida una cadena de bytes:

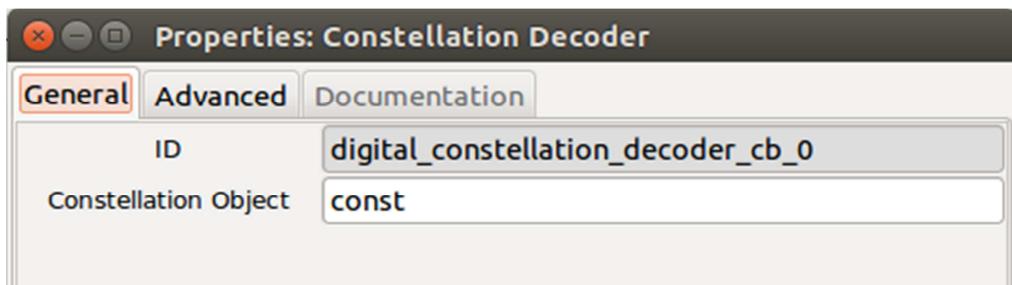


Figura 47. Parámetros Bloque Constellation Decoder

Este bloque es el encargado de realizar el mapeo de muestras complejas a símbolos, para ello, se emplea el método *decision_maker* definido en el objeto de la constelación.

El parámetro *Constellation Object* posee la variable *const* que corresponde al objeto de la constelación (Constellation Rect Object).

3.4.3.3 Bloque decodificación en código Gray (Map)

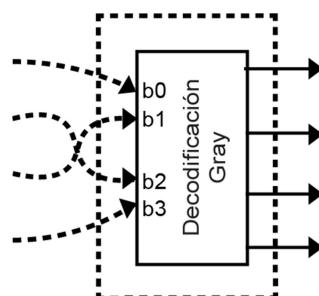


Figura 48. Funcionalidad del Bloque de decodificación Gray en la modulación convencional

Este bloque es el encargado de desmapear el código Gray:

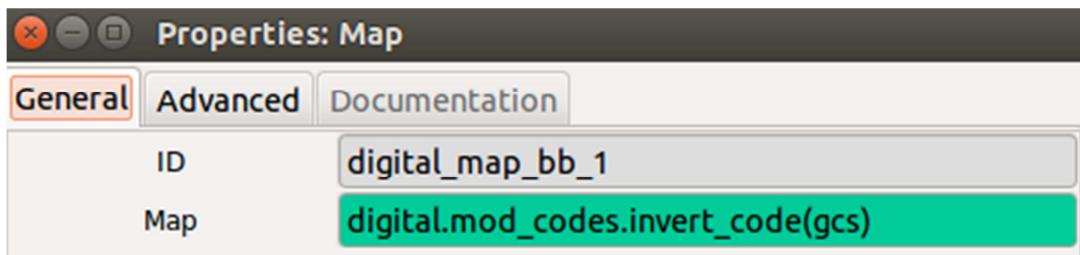


Figura 49. Parámetros Bloque Map – Decodificación Gray

Para llevar a cabo el desmapeo, se ha de indicar el código inverso de Gray empleado en la codificación (digital.mod_codes.invert_code()). Disponiendo así a la salida del bloque, la cadena de bytes correctamente convertidos a binario.

3.4.4 Desempaquetamiento de datos

3.4.4.1 Bloque Unpack K bits

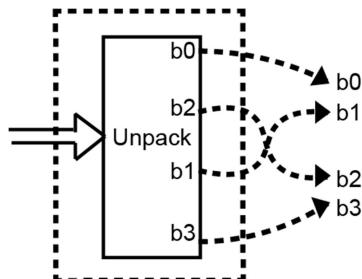


Figura 50. Funcionalidad del Bloque Unpack K bits en la modulación convencional

El bloque que conforma el Desempaquetamiento de datos se llama *unpacked* cuya configuración se muestra a continuación:

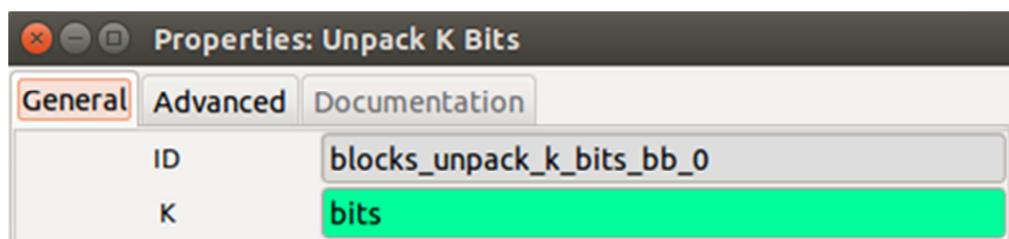


Figura 51. Parámetros Bloque Unpack K bits

La función que realiza este bloque no es más que desempaquetar los bytes. Genera K bytes uno por cada uno de los *chunks* o K bits (los que definen el símbolo) y se coloca en la posición menos significativa (LSB) del byte.

3.4.4.2 Bloque Packet Decoder

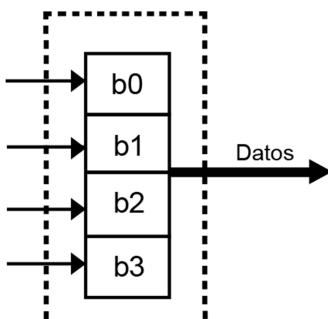


Figura 52. Funcionalidad del Bloque Packet Decoder en la modulación convencional

Una vez se disponen de los bytes desempaquetados el siguiente bloque con el que se encuentra el flujo de bits es el *packet decoder*:

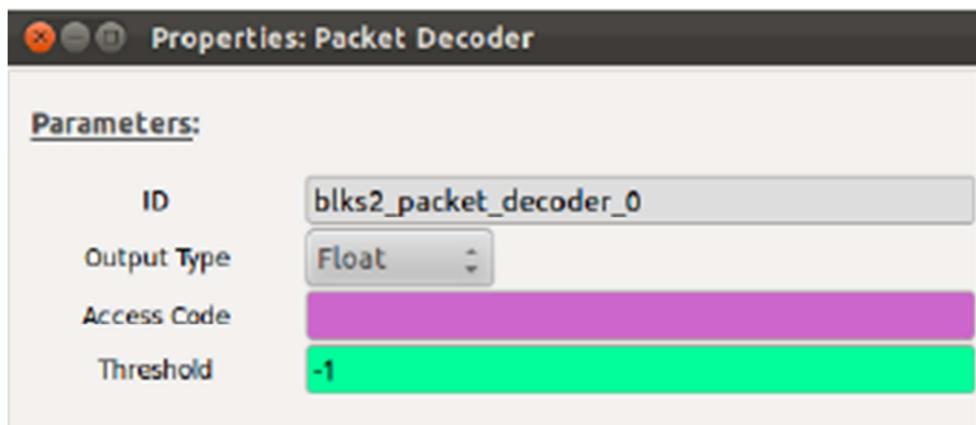


Figura 53. Parámetros Bloque Packet Decoder

Este bloque, parte de una cadena de bytes desempaquetados y se encarga por una parte de lidiar con el sincronismo a nivel paquete y por otra, transformar el tipo de dato de tipo byte a tipo float para poder finalmente entregar los datos demodulados y decodificados al sumidero pertinente.

Capítulo 4

Implementación en Hardware SDR

Este capítulo tiene por objetivo mostrar el proceso de migración del diseño de un sistema de modulación 16-CQAM a uno en donde se utilice plataformas hardware disponible, para ello se han utilizado la USRP B210 para el transmisor siguiendo el esquema de interconexión mostrado en la figura 23 y el Periférico SDR HackRF para el receptor siguiendo el esquema mostrado en la figura 39.

4.1 USRP B210

Proporciona una plataforma hardware USRP completamente integrada, con cobertura de frecuencia continua de 70 MHz a 6 GHz. Diseñado para la experimentación de bajo costo, combina el transceptor de conversión directa RFIC AD9361 que proporciona hasta 56MHz de ancho de banda en tiempo real, un FPGA Xilinx Spartan6 abierto y reprogramable y una conectividad USB 3.0. El soporte completo para el software USRP Hardware Driver (UHD) le permite la integración completa con GNU Radio [31].

Características principales como transmisor:

Antena: TX/RX

Frecuencia de operación: 70 MHz – 6 GHz

Potencia del Transmisor: +20 dBm máx.

Ganancia del Transmisor: 0 – 100

Sample Rate: 62.5 ksps – 61.44 Msps

Master Clock Rate (MCR): 32 Mhz (máx: 61.44e6 Mhz)

Conectividad: USB 3.0

(MCR/sample rate): Debe ser un múltiplo entero divisible por 4 para mejor rendimiento en RF (espectro plano).

La siguiente figura muestra la configuración realizada para el bloque USRP Sink en GNU Radio utilizado como transmisor:

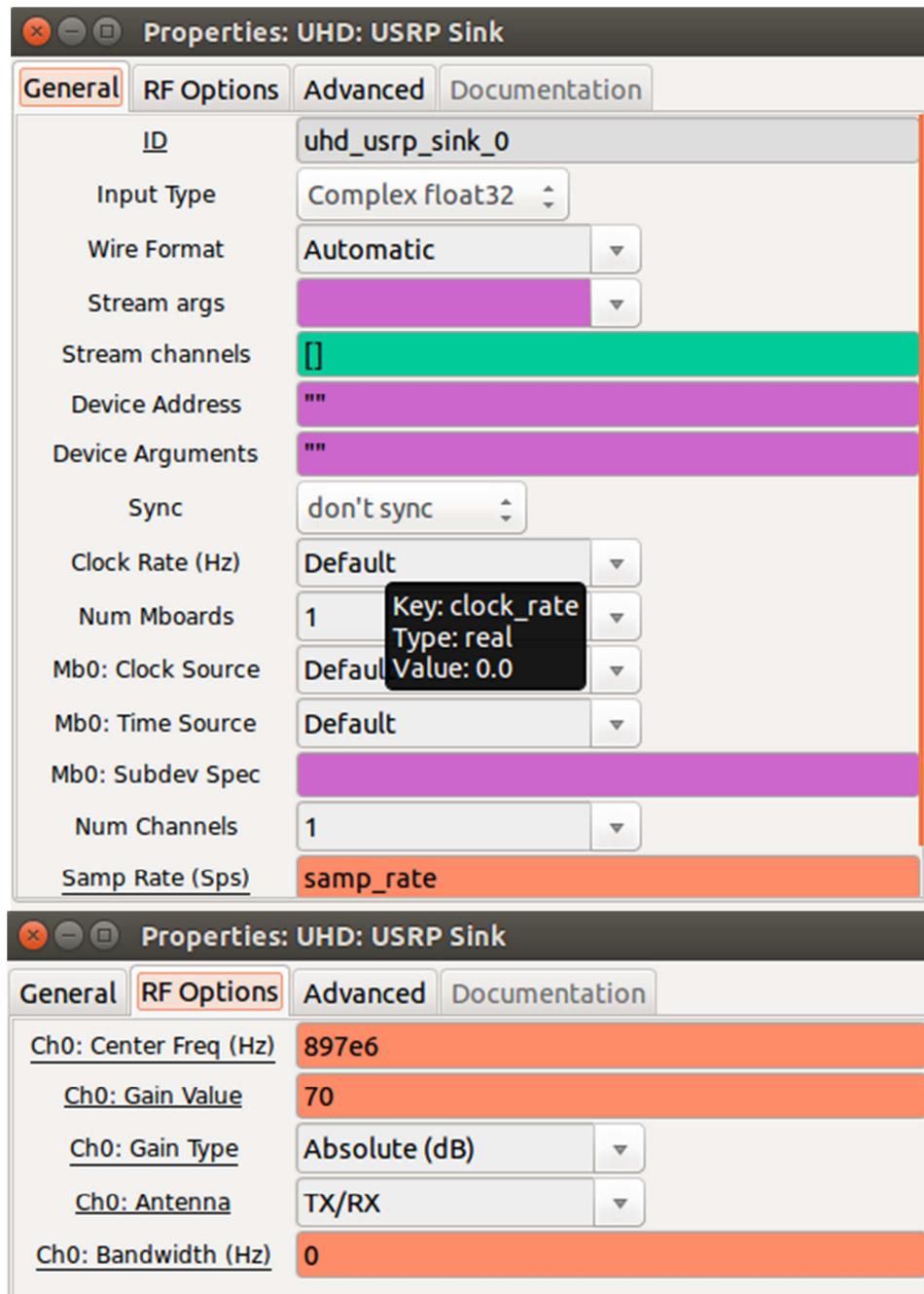


Figura 54. Parámetros Bloque UHD:USRP Sink

La comunicación con el periférico USRP se realiza mediante USB 3.0, usualmente todos los parámetros del bloque USRP Sink se colocan por defecto a excepción de las variables Sample Rate, frecuencia, ganancia y la antena a utilizar. El parámetro *Clock Rate* se deja por defecto (32Mhz) de frecuencia de reloj en el transmisor, *Subdev Spec* sirve para

indicar el camino de la antena al conversor, este parámetro puede seleccionar entre diferentes front ends que posea en periférico.

A este dispositivo, se le indica la *sample rate* deseada, se selecciona una frecuencia de operación de 897Mhz y la antena TX/RX. El parámetro de ganancia es el encargado de establecerla en la cadena transmisora (*daughterboard*). Por último, también se le ha de indicar el ancho de banda de los filtros si los hubiese.

4.2 HackRF

Es un periférico SDR capaz de transmitir o recibir señales de radio de 1 MHz a 6 GHz. Diseñado para permitir la prueba y el desarrollo de tecnologías de radio modernas y de próxima generación, HackRF es una plataforma de hardware de código abierto que puede usarse como un periférico USB o programado para una operación independiente [32].

Características principales como receptor:

Frecuencia de operación: 1 MHz – 6 GHz

Sensibilidad del receptor: $-70 \text{ dBm} < S < -5 \text{ dBm}$.

Ganancia del Receptor: 0 – 100

Sample Rate: 2 Msps – 20 Msps

Master Clock Rate (MCR): 10Mhz

Conectividad: USB 2.0

(MCR/sample rate): Debe ser un múltiplo entero divisible por 4 para mejor rendimiento en RF (espectro plano)

La siguiente figura muestra la configuración del bloque en GNU Radio encargado de realizar la conexión mediante USB con el dispositivo SDR HackRF.

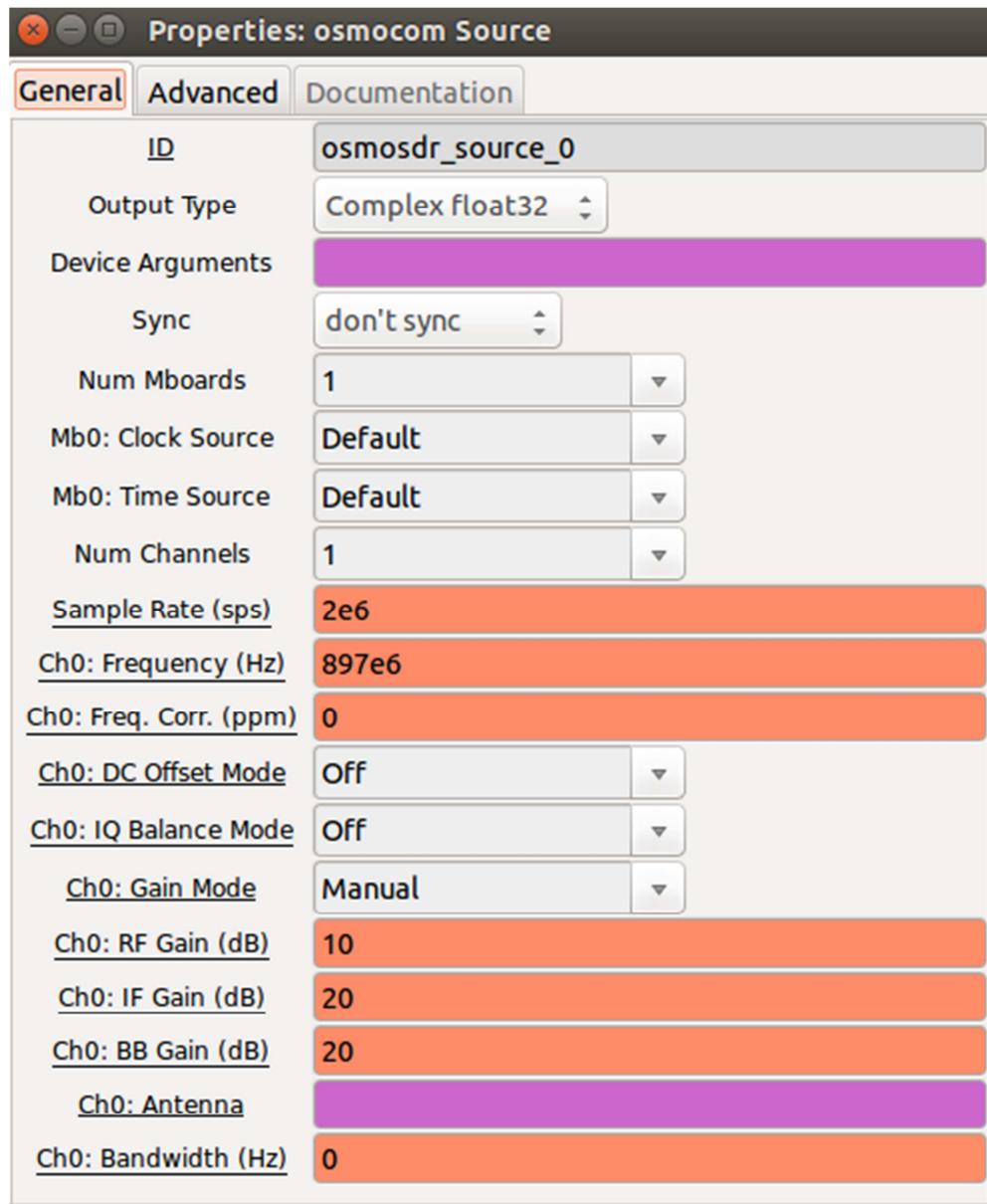


Figura 55. Parámetros Bloque osmocom Source

Entre los parámetros importantes se encuentra la frecuencia de recepción (897Mhz), el *sample rate* utilizado en recepción (debe estar en el rango entre 2Msps – 20Msps por especificaciones propias del dispositivo) y la ganancia en RF que por defecto es 20. La frecuencia del reloj se fija por defecto (10Mhz).

4.3 Aspectos Generales

Debido a las especificaciones propias de los dispositivos utilizados en transmisión y recepción es importante la introducción de los bloques Costas Loop, FLL Band Edge y Polyphase Clock Sync en la sincronización de tiempos de muestreo, frecuencia, fase y recuperación del reloj, se debe tener en cuenta ciertos aspectos como son los valores de las amplitudes y la tasa binaria. Con respecto a la amplitud, los valores de estos impactan directamente en el voltaje de la señal enviada al DAC que presenta un rango desde -1 a +1V y por lo tanto un valor que exceda a esta magnitud (0 dB) saturara el DAC.

Por otra parte, se tendrá que tener en cuenta el factor del *sample rate*. Segundo el fabricante, el máximo *sample rate* soportado para la USRP B210 utilizado como transmisor es de 61.44Msps según el número de bits utilizados para la representación de cada muestra.

En base a esto, habrá una velocidad de transmisión que tendrá que ser soportada por la conexión USB 3.0, para ello se emplea la siguiente fórmula:

$$V(bps) = N_{bits} \left(\frac{\text{bits}}{\text{muestra}} \right) \text{sample rate} \left(\frac{\text{muestras}}{\text{segundo}} \right) < \text{velocidad USB 3.0} \quad (49)$$

Donde N_{bits} son los bits utilizados para representar cada muestra. Como previamente se ha comentado, el *sample rate* fijado en el transmisor es de 500KSps, esto generará una velocidad de transmisión (para el caso de 32 bits por muestra) de:

$$V(bps) = 16 \left(\frac{\text{bits}}{I\text{muestra}} \right) + 16 \left(\frac{\text{bits}}{Q\text{muestra}} \right) (500Ksp) = 16Mbps < 5Gbps \quad (50)$$

4.4 Análisis de Potencia

4.4.1 Pruebas de transmisión

Para la realización de esta prueba se tendrá que hacer el uso del siguiente material:

- USRP B210
- Analizador de Espectro
- Antenas monopolo vertical para el analizador de espectro y la USRP B210

Para llevar a cabo las pruebas de transmisión se ha usado la interfaz uhd_siggen_gui.grc disponible en las librerías de GNU Radio para la caracterización de trasmisores en las que se puede seleccionar diferentes tipos de señal a transmitir (apéndice C). La distancia entre la antena transmisora y receptora es $d = 16cm$

El resultado obtenido es el siguiente:

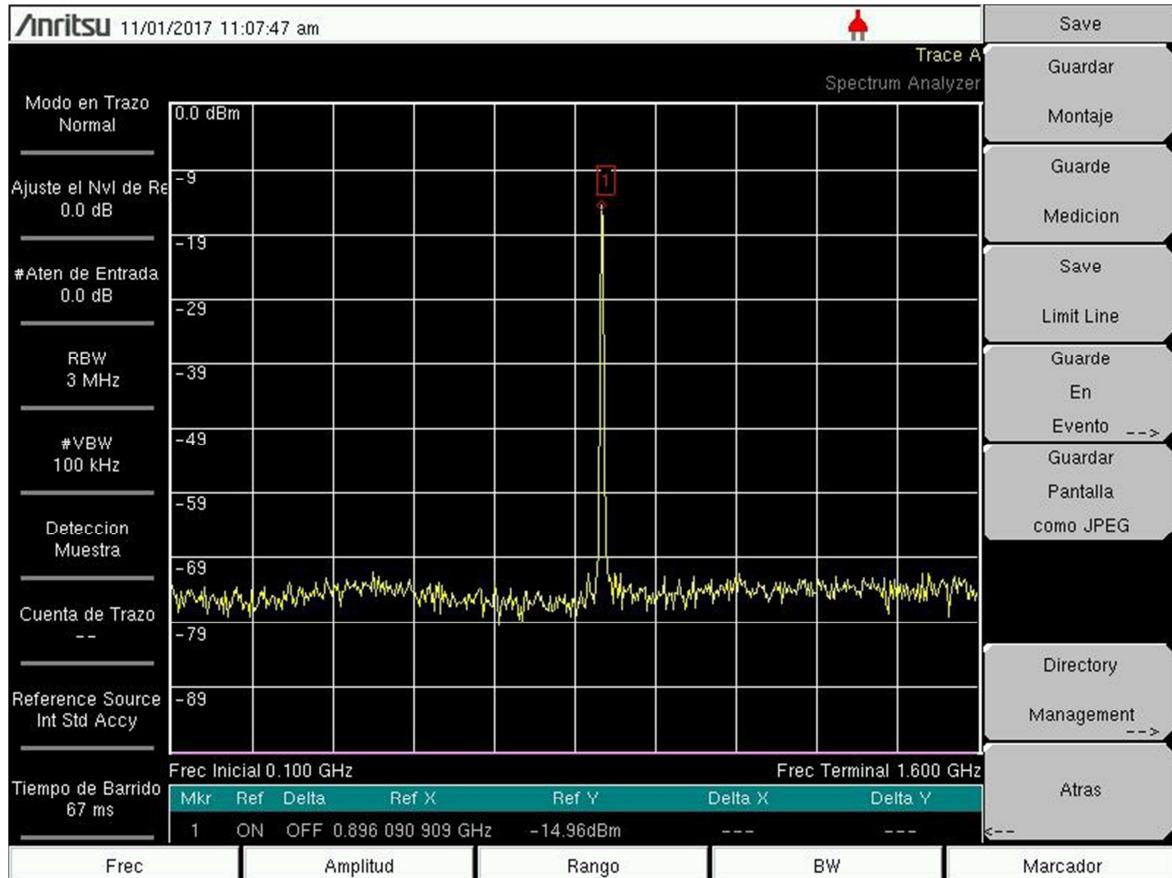


Figura 56. Prueba de transmisión USRP B210 @ 897Mhz

Como se puede apreciar, en la frecuencia de trabajo (987Mhz), la potencia en el analizador de espectro es $P_r = -15dBm$

Con estos valores, es posible calcular la potencia del transmisor mediante la ecuación de Friis (ecuación 34), teniendo en cuenta las perdidas por espacio libre y la ganancia de las antenas transmisora y receptora expresadas en dB [33].

$$P_R(dBm) = P_T(dBm) + G_{Tx}(dB) + G_{Rx}(dB) - L_0 \quad (51)$$

Donde $G_{Tx(dB)}$ y $G_{Rx(dB)}$ son las ganancias de las antenas monopolo vertical $\lambda/4$, que alrededor son de 1.8 dB por antena [34], L_o son las perdidas por espacio libre que se calculan según la ecuación 31.

$$L_0 = 92.4 + 20 \log_{10} f_{GHz} + 20 \log_{10} d_{Km} \quad (52)$$

Entonces;

$$L_0 = 92.4 + 20 \log_{10}(0.897) + 20 \log_{10}(0.00016) = 15.54 \text{ dB} \quad (53)$$

Despejando $P_{T(dBm)}$ en la ecuación 30 y reemplazando los valores correspondientes se encuentra la potencia de transmisión del dispositivo.

$$P_{T(dBm)} = -15dBm - 1.8 \text{ dB} - 1.8 \text{ dB} + 15.54 \text{ dB} = -3 \text{ dBm} \quad (54)$$

4.4.2 Distancia mínima entre TX y RX

Debido a que la comunicación se hace por medio de antenas, es necesario calcular la distancia mínima entre las antenas donde no se presenten otros fenómenos no deseables. Las regiones de campo de una antena suele subdividirse en tres regiones: (a) reactiva de campo cercano, (b) radiación de campo cercano (Fresnel) y (c) campo lejano (Fraunhofer) como se muestra en la figura 57.

La región reactiva de campo cercano se define como la porción de la región de campo cercano que rodea la antena en la que predomina el campo reactivo. Para la mayoría de antenas, el límite exterior de esta región se considera $R < 0,62 \sqrt{D^3 / \lambda}$, donde λ es la longitud de onda y D es la longitud más significativa de la antena. La región radiación de campo cercano (Fresnel) se define como esa región del campo de una antena entre la región de campo cercano reactiva y la región de campo lejano donde la radiación y la distribución del campo angular dependen de la distancia de la antena.

La región de campo lejano (Fraunhofer) es aquella región del campo de una antena donde la distribución del campo angular es esencialmente independiente de la distancia desde la antena. Si la antena tiene una dimensión longitudinal máxima D , la región de campo lejano se considera comúnmente que existe a distancias mayores que $2D^2 / \lambda$ de la antena, siendo λ la longitud de onda y D es la dimensión longitudinal más significativa de la antena, si R es mayor o igual que $2D^2 / \lambda$ se encuentra en zona lejana y la estructura de campo corresponde a la de campo lejano [34].

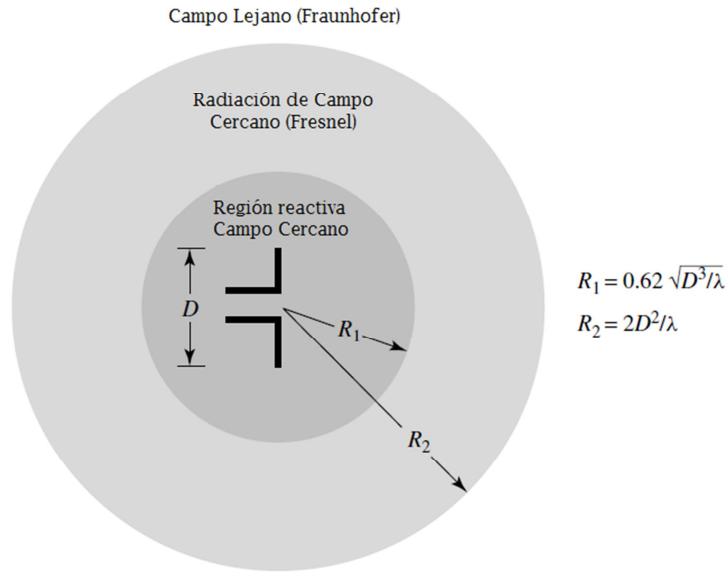


Figura 57. Regiones de campo de una antena

Se calcula longitud de onda en metros teniendo en cuenta la frecuencia de trabajo

$$\lambda_m = \frac{300}{f_{MHz}} = \frac{300}{897} = 33.4 \text{ cm} \quad (55)$$

Considerando ($R = d$) y $D = 15 \text{ cm}$ (longitud significativa de la antena) donde d es la distancia entre las antenas transmisora y receptora, entonces;

$$(R = d) = \frac{2D^2}{\lambda} \approx 13,5 \text{ cm} \quad (56)$$

La distancia mínima inicial para mediciones y pruebas entre las antenas es aproximadamente $13,5 \text{ cm}$.

4.4.3 Pruebas de recepción

Es necesario calcular la potencia de recepción a la distancia mínima inicial ($d = 13,5 \text{ cm}$) y máxima ($d = 1.5 \text{ m}$) y que esta se encuentre dentro del rango de sensibilidad del receptor ($-70 \text{ dBm} < S < -5 \text{ dBm}$).

Con estos valores, es posible calcular la potencia de recepción mediante la ecuación de Friis (ecuación 30), teniendo en cuenta las perdidas por espacio libre y la ganancia de las antenas transmisora y receptora.

Donde $G_{Tx(dB)}$ y $G_{Rx(dB)}$ en este caso, son las ganancias de las antenas log periódicas, que alrededor son de 5 dB por antena según especificaciones de las mismas [34], se utilizaron estas antenas y no las monopolo vertical debido a su mayor ganancia para las mediciones a diferentes distancias, L_o son las perdidas por espacio libre que se calculan según la ecuación 31.

$$L_0 = 92.4 + 20 \log_{10} f_{GHz} + 20 \log_{10} d_{Km} = 14.04 \text{ dB} \quad (57)$$

Entonces;

$$P_{R(dBm)} = -3dBm + 5dB + 5dB - 14.04dB = -7dBm \quad (58)$$

Para la distancia mínima inicial ($d = 13,5 \text{ cm}$) la potencia de recepción $P_{R(dBm)} = -7 \text{ dBm}$, y para la distancia máxima ($d = 1.5 \text{ m}$) la potencia $P_{R(dBm)} = -68 \text{ dBm}$, valores que se encuentran dentro de la sensibilidad del receptor, esto debido al tipo de antenas escogidas y la potencia de transmisión, distancias más pequeñas o grandes pueden causar inconvenientes adicionales en la recepción de la señal [33].

Capítulo 5

Resultados Obtenidos

Una vez explicado el flujo del diseño e implementación del transmisor y el receptor, así como los bloques de procesado y sus parámetros que conforman tanto el modulador como el demodulador, en este capítulo se mostrara los resultados obtenidos en la ejecución e implementación de este diseño, posteriormente se mostrara un método para la estimación del BER¹² y finalmente se mostrara el desempeño del esquema de modulación respecto a la SNR

5.1 Modulador 16-CQAM

Inicialmente se muestra las señales a transmitir en el tiempo, en primer lugar un archivo de audio con frecuencia de muestreo de 22.05 KHz usando el bloque Wav File Source y una señal Coseno de frecuencia 1Khz

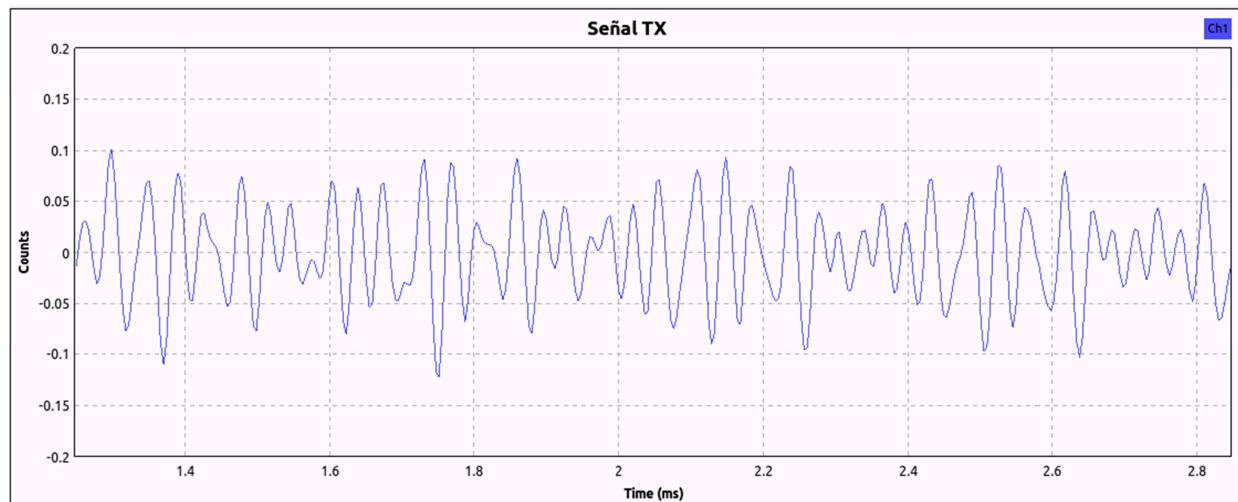


Figura 58. Señal de audio transmitida

¹² Bit Error Rate

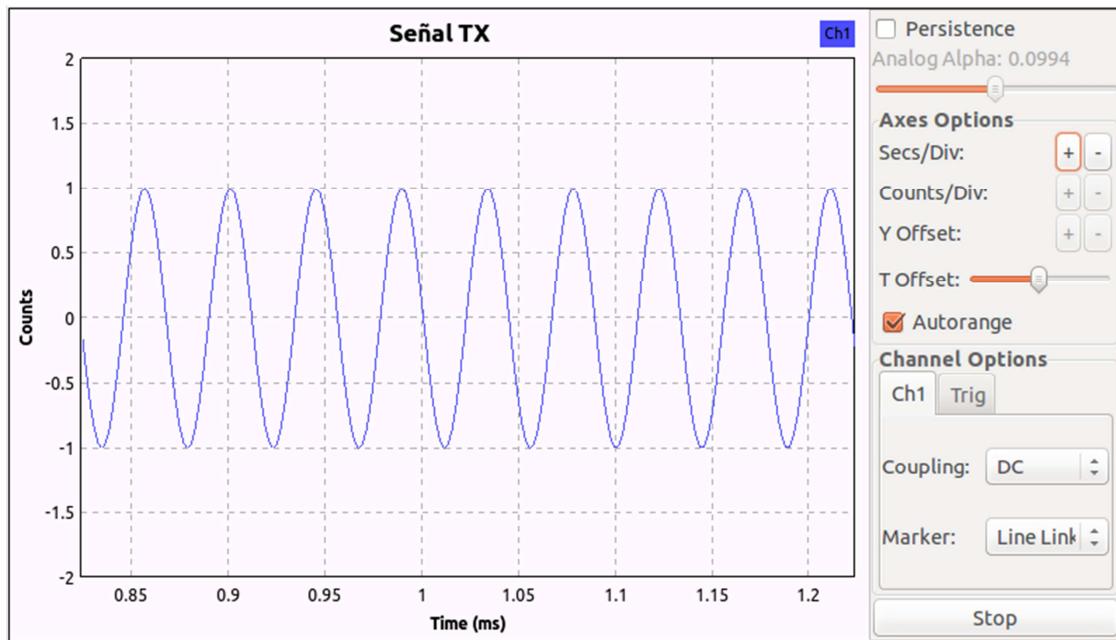


Figura 59. Señal coseno transmitida

Las figuras 60 y 61 muestran la salida del codificador donde se muestra la constelación para los esquemas de modulación 16-QAM y 16-CQAM.

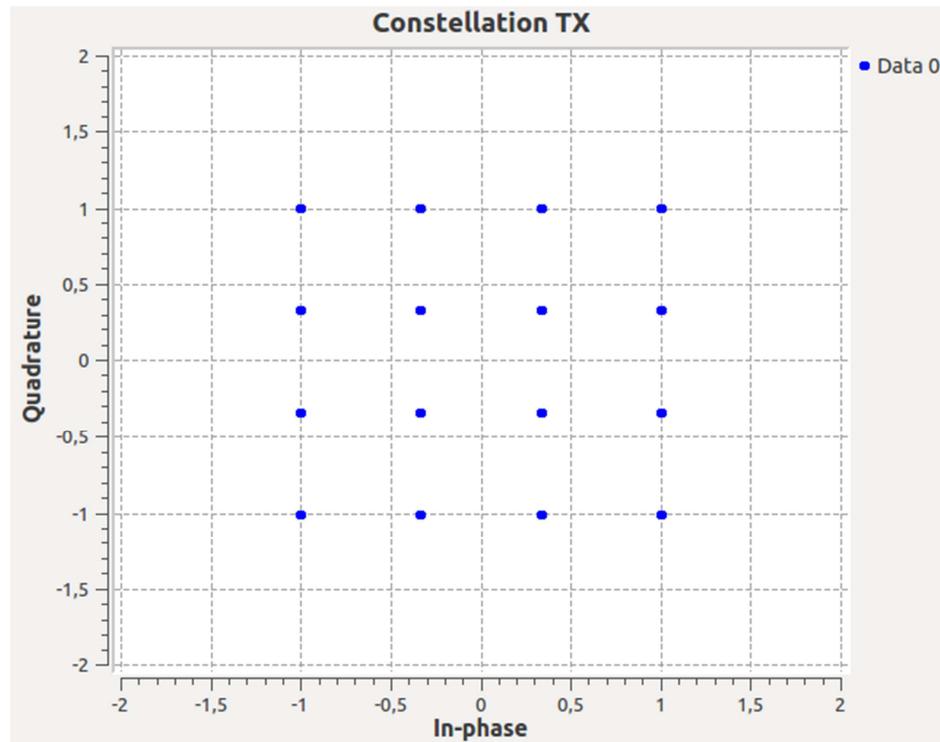


Figura 60. Constelación 16-QAM transmitida

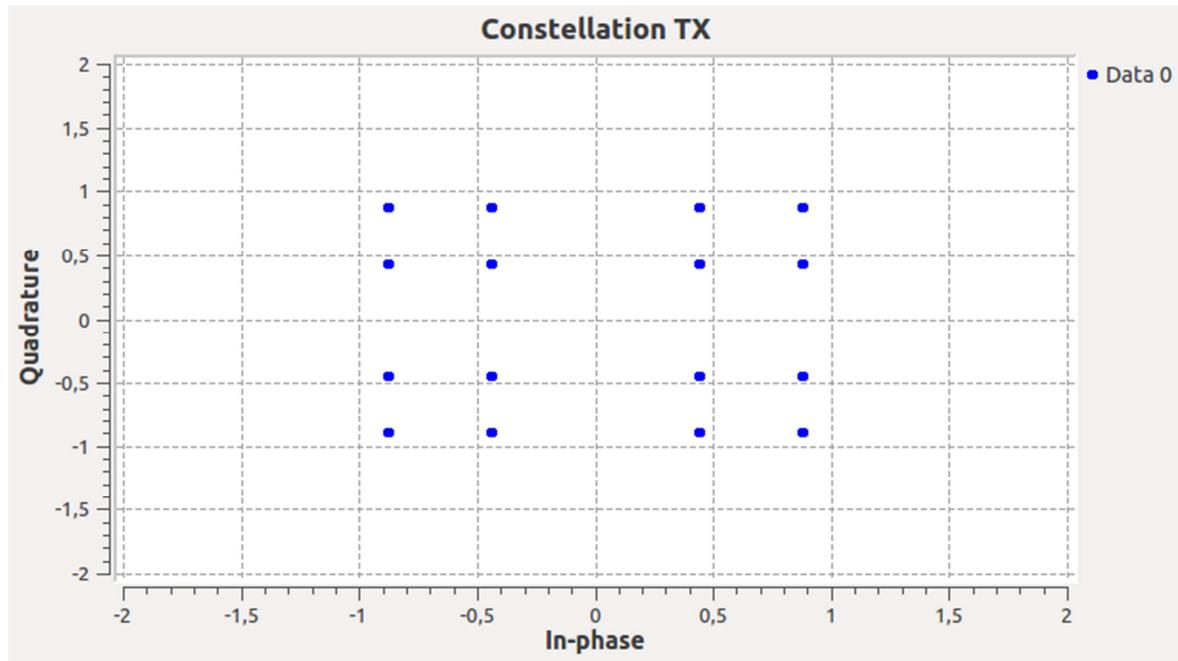


Figura 61. Constelación 16-CQAM transmitida

La constelación transmitida (figura 61) coincide con la figura 18, que se muestra en el apartado 3.1 del diseño.

A continuación se muestran la señal QAM y CQAM en el tiempo y el espectro de la señal a la salida del modulador o filtro de transmisión en banda base.

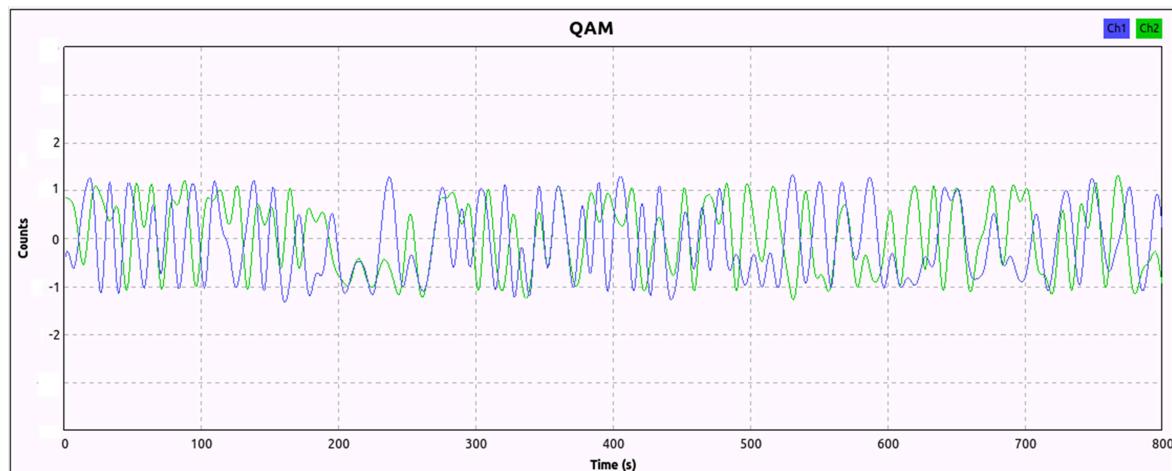


Figura 62. Señal 16-QAM modulada en amplitud transmitida.

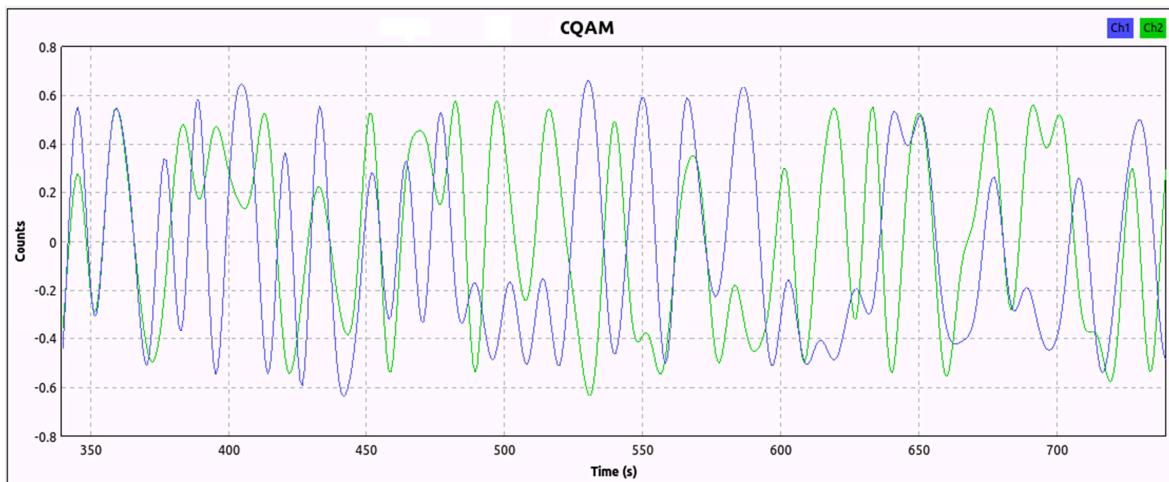


Figura 63. Señal 16-CQAM modulada en amplitud transmitida.

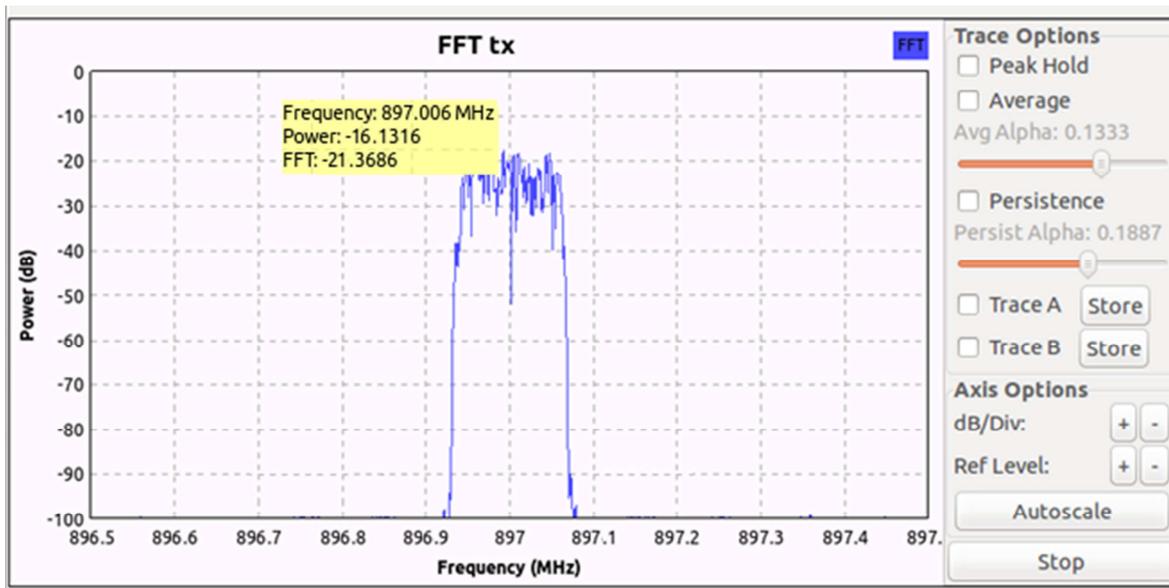


Figura 64. Espectro transmitido 16-QAM

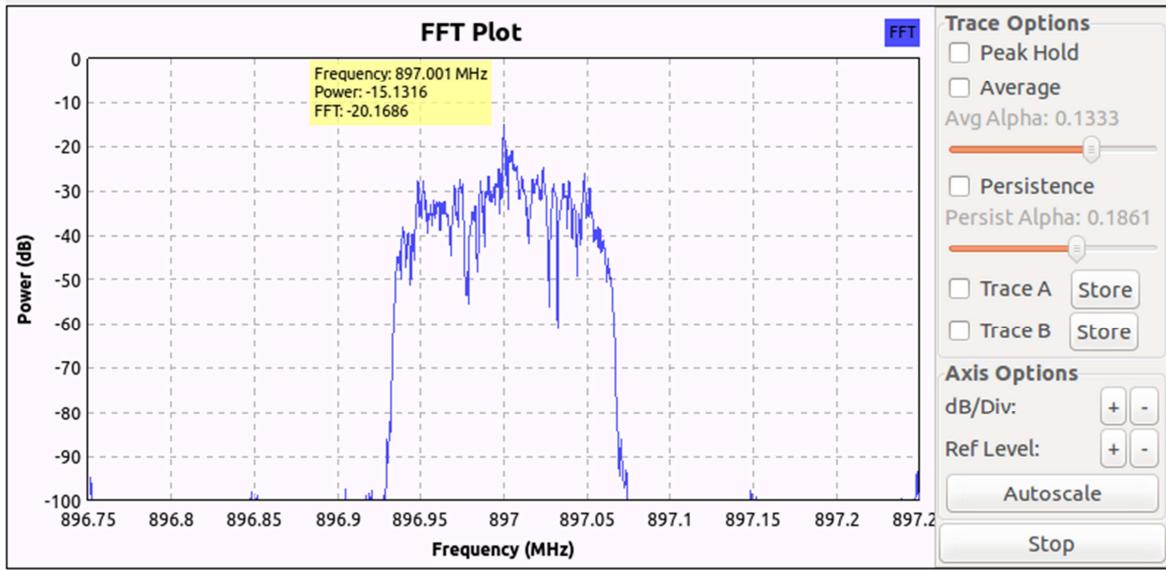


Figura 65. Espectro transmitido 16-CQAM

5.2 Demodulador 16-CQAM

Se muestra la señal de modulación en amplitud en el tiempo recibida por el receptor a una distancia inicial mínima encontrada ($d = 13,5\text{cm}$) en el apartado 4.4.2.

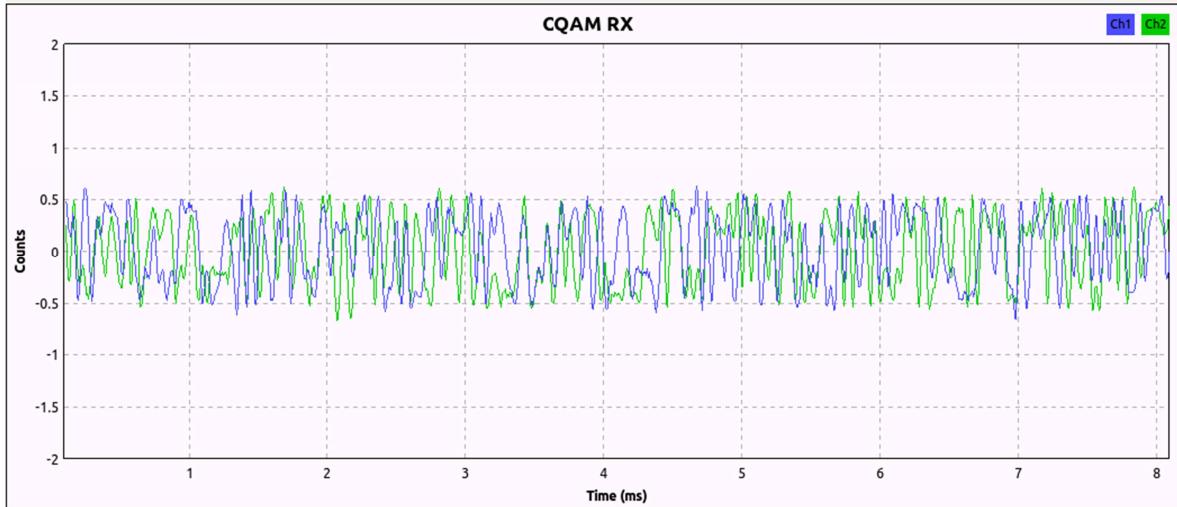


Figura 66. Señal 16-CQAM recibida.

A continuación se muestra el espectro de la señal recibida justo antes de atravesar los bloques del demodulador

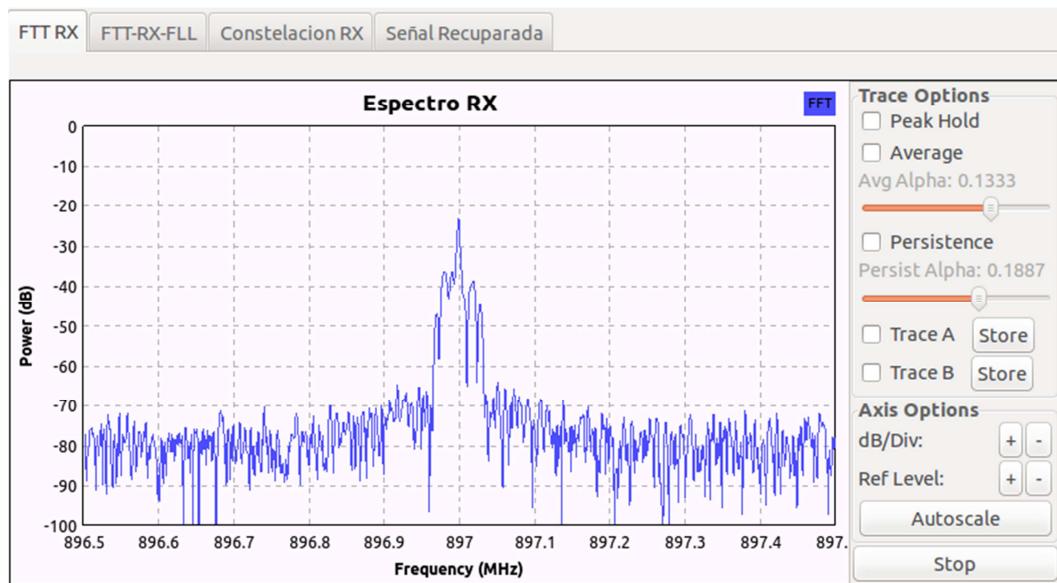


Figura 67. Espectro recibido 16-CQAM

Se puede apreciar que el ruido de fondo que llega al receptor se encuentra alrededor de -70 dBm , para la distancia inicial mínima ($d = 13,5\text{cm}$) entre las antenas de transmisión y recepción.

La siguiente imagen muestra la salida de la cadena de bloques de sincronización de tiempos de muestreo, frecuencia y fase y del filtro de recepción, se puede distinguir la constelación 16-CQAM recuperada.

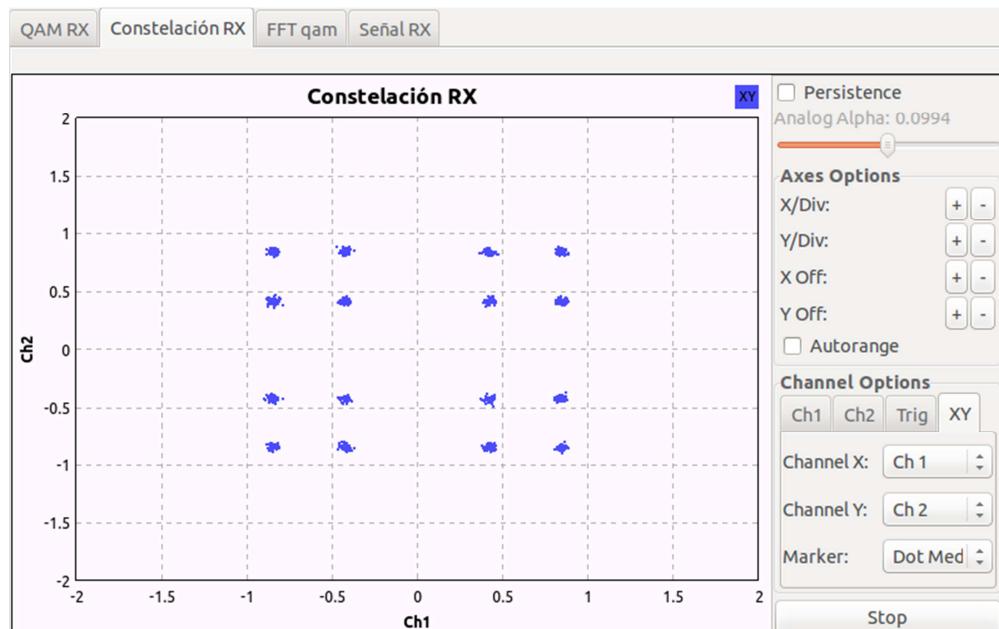


Figura 68. Constelación recibida 16-CQAM

Finalmente, una vez realizado el proceso de decodificación descrito en el apartado 3.4.3 se obtiene la señal de audio o coseno recuperada como se muestra en las siguientes figuras:

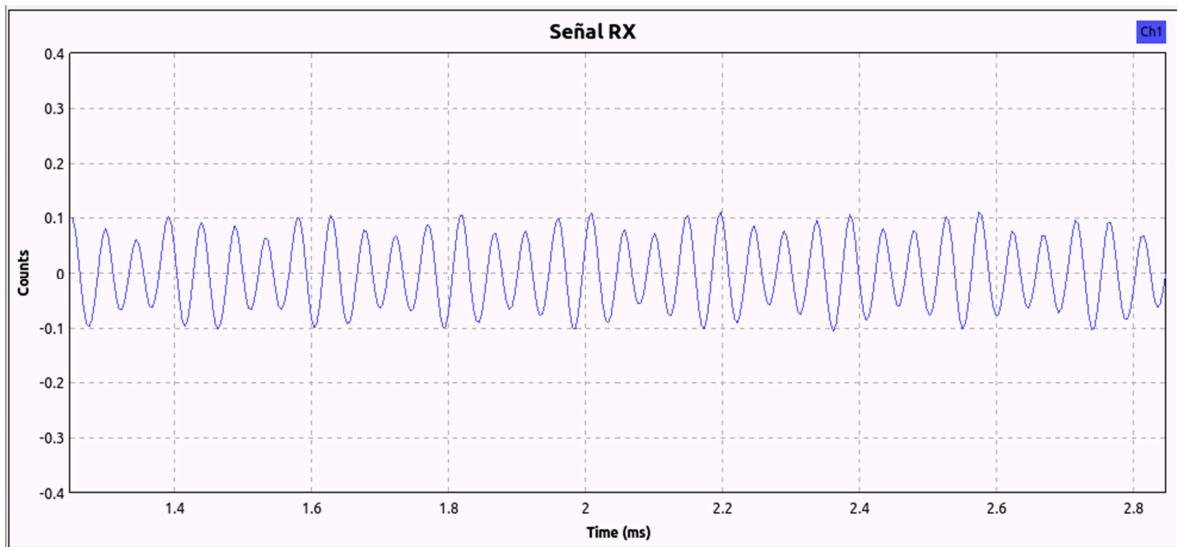


Figura 69. Señal audio recuperada

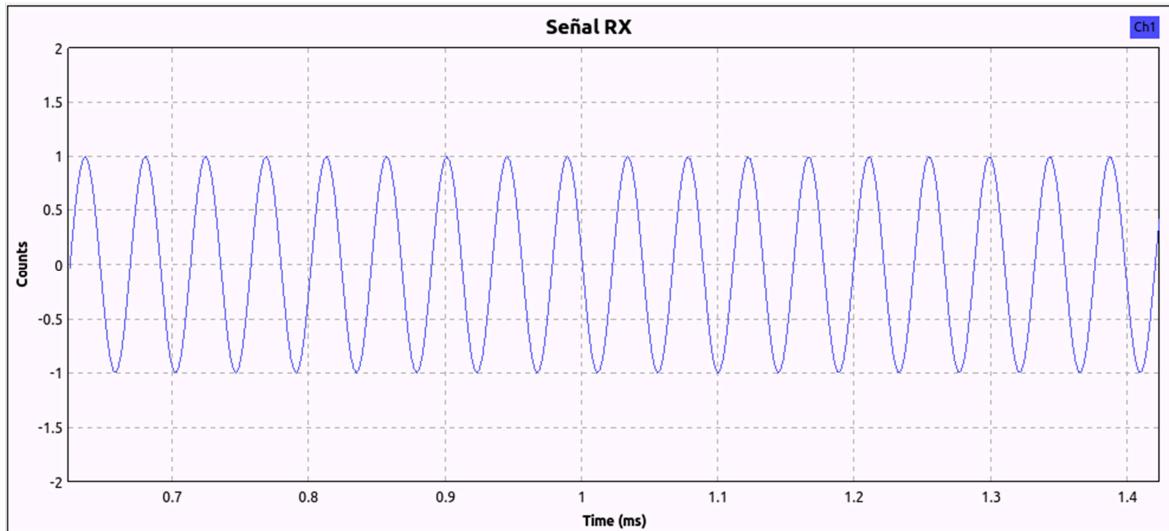


Figura 70. Señal coseno recuperada

Esta es la figura que ilustra que el proceso se ha llevado correctamente, de no haber sido así, si se hubiera producido errores, el bloque *packet decoder* no hubiera devuelto datos y las figuras 69 y 70 hubieran quedado en blanco. Cabe destacar, que si existe alguna diferencia entre las señales de transmisión y recepción en el tiempo para audio y la señal coseno se debe a la variabilidad de la señal y a los diferentes instantes de capturas.

5.3 Estimación del BER

Una vez se ha presentado el sistema de comunicaciones digitales con esquema de modulación CQAM, se ha de caracterizar la calidad de este. Para ello, se utiliza el parámetro conocido como Bit Error Rate, La Medición del BER consiste en fijar un umbral de aceptación el cual se relaciona con un numero de muestras “suficientes” según la cantidad de datos transmitidos por unidad de tiempo, y se transmiten los datos por un canal durante un tiempo que garantice un nivel mínimo de BER, con esto si durante este intervalo de tiempo fijado según el nivel de aceptación, no se ha superado el BER entonces se puede certificar inequívocamente que se cumplen los criterios fijados, en caso negativo se van subiendo los umbrales del BER. Esta medición se hace de esta manera teniendo en cuenta que esto garantiza un comportamiento estadístico predefinido [35].

Para la medición del BER, se va a hacer uso del bloque en GNU Radio: Error Rate, con los siguientes parámetros:

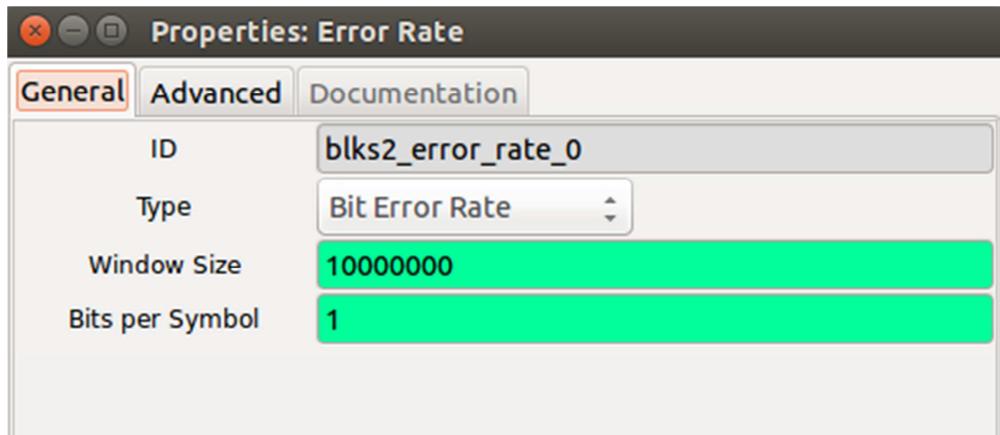


Figura 71. Parámetros Bloque Error Rate

Este bloque calcula el error desde $N = 1$ hasta $N = \text{Ventana}$, la ventana es un parámetro que se debe definir con base a la cantidad de datos transmitidos por unidad de tiempo y durante un intervalo de tiempo, al llenarse la ventana se van desplazando uno a uno los bits del arreglo de entrada para calcular de forma “dinámica” el Bit Error Rate.

Para el caso particular el bloque de Error Rate es una herramienta que cumple con un criterio conveniente y fácil de implementar dado que se sincroniza la señal transmitida junto con los datos transmitidos y esto se compara en el bloque el cual después del tiempo requerido para completar la ventana de medición asigna un valor al BER del canal. Este valor tiende a variar, por lo que se da una medida equivalente a la media de valores indicados por el bloque.

En la figura 71 puede verse el bloque Error Rate y sus parámetros, en general y para realizar las mediciones se ajusta el BER de la siguiente manera:

Type: Bit error Rate

Window Size: 10M

Bits per Symbol: 1

Luego de validar los procedimientos realizados por el bloque se identifica que en su código fuente en Python (apéndice D), realiza una división del vector de error sobre el número de bits por símbolo, lo cual en este momento no es deseable, dado que se busca comprobar la cantidad de bits recuperados para comparar de esta manera los esquemas de modulación QAM y CQAM.

En la figura 73 se muestra el esquema utilizado para la simulación y estimación del BER dentro del mismo entorno GNU Radio sin el uso de hardware SDR, con el fin de verificar el comportamiento de los bloques encargados de la medición, para ello se añade el bloque *Noise source* encargado de la generación de Ruido Blanco Gaussiano (AWGN)

Los parámetros del Bloque *Noise Source* se muestran a continuación:

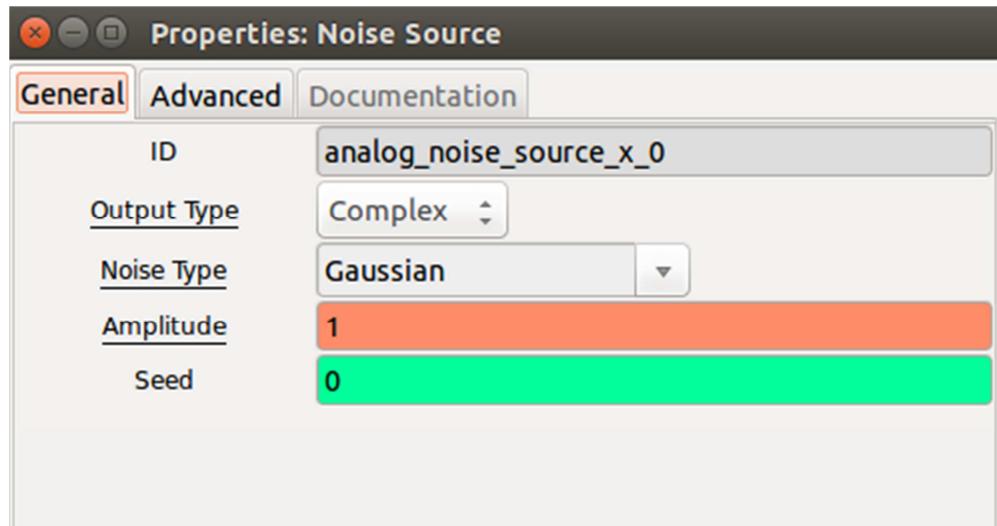


Figura 72. Parámetros Bloque Noise Source

Este bloque es el encargado de la generación de ruido dependiendo de la distribución aleatoria escogida, para las mediciones se escoge un tipo de distribución de ruido de tipo Gaussiano, el parámetro amplitud será modificado aumentando o disminuyendo el ruido con el fin de variar la relación señal a ruido y medir el respectivo BER.

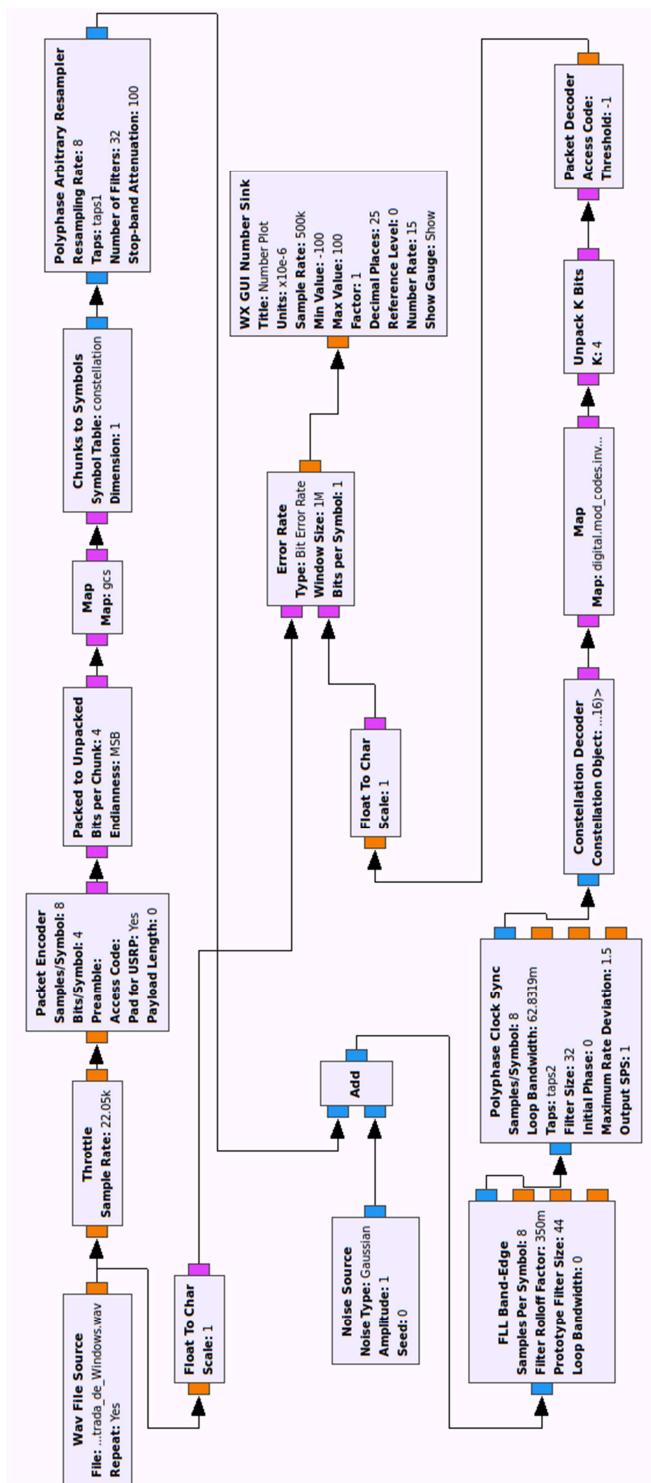


Figura 73. Esquema medición del BER

La señal de ruido normalizada en el tiempo se muestra a continuación:

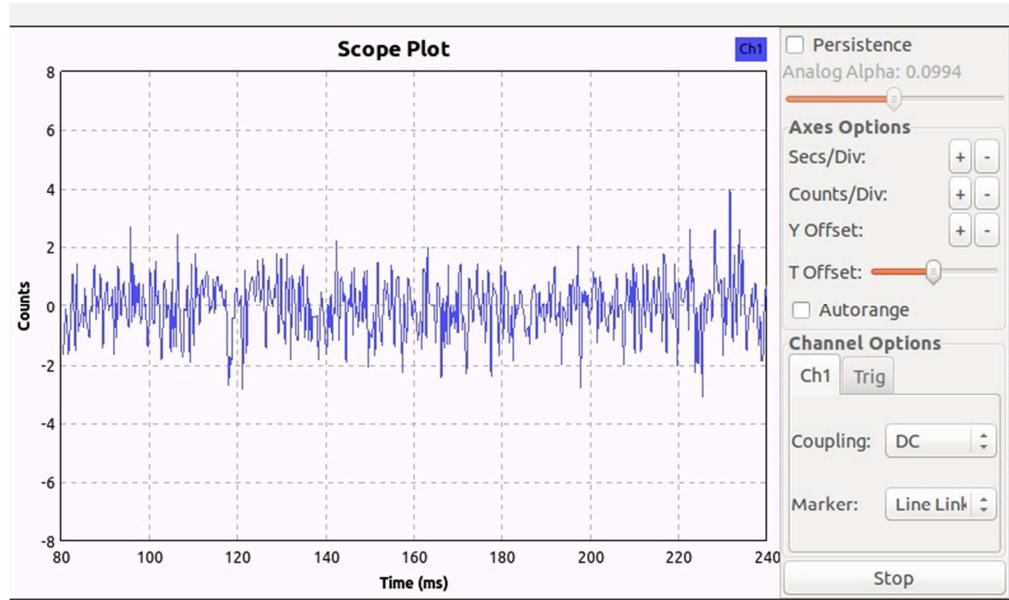


Figura 74. Ruido Gaussiano

Se escoge la distribución gaussiana normalizada como se muestra en la siguiente figura:

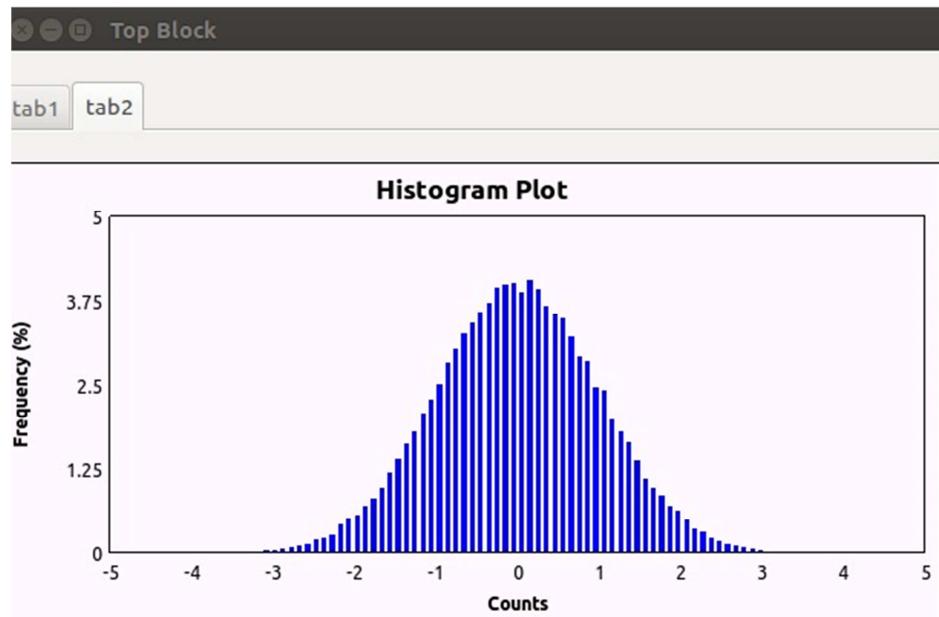


Figura 75. Distribución gaussiana del Bloque Noise Source

Las mediciones se realizan para los esquemas de modulación convencional 16-QAM y el esquema de modulación jerárquico con conjuntos de cantor 16-CQAM, obteniendo los valores de BER en el entorno de simulación en la herramienta de GNU Radio para luego proceder a graficarlos.

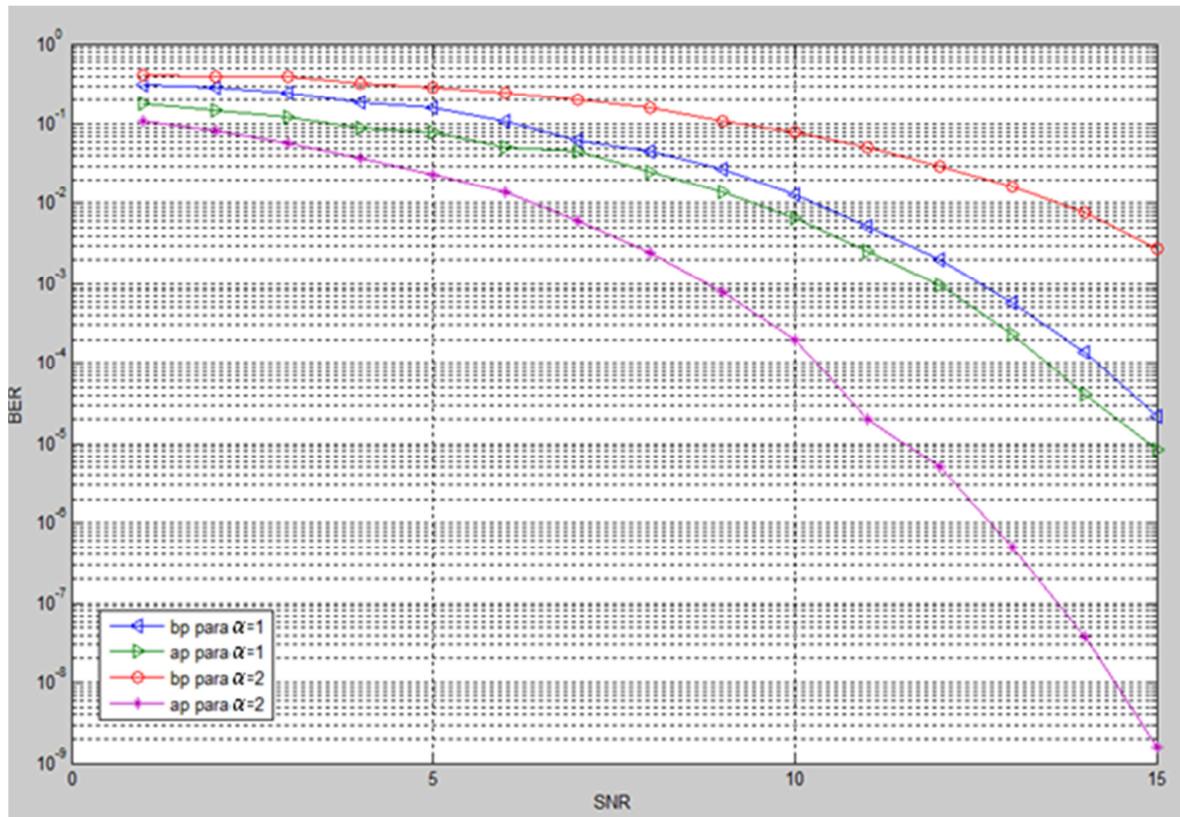


Figura 76. BER vs SNR para los bits de alta y baja prioridad en los esquemas de modulación 16-QAM ($\alpha = 1$) y 16-CQAM ($\alpha = 2$).

Los resultados se ilustran en la figura 76, donde se puede ver el BER de los bits de alta y baja prioridad respecto a la relación señal a ruido para el esquema de modulación convencional 16-QAM ($\alpha = 1$) como para el esquema de modulación cantor 16-CQAM ($\alpha = 2$) y se evidencia claramente cómo se mejora el comportamiento del BER para los bits de alta prioridad, pero empeorando los bits de baja prioridad.

5.4 Medición del BER con USRP

A continuación se muestra el esquema utilizado la estimación del BER haciendo uso de hardware SDR tanto para el transmisor como para el receptor, se envia un archivo de audio como fuente a tranmistir, variando la distancia entre TX y RX se recupera un archivo de audio que llega al receptor y se compara con el archivo de audio original, generando asi, los diferentes valores de BER

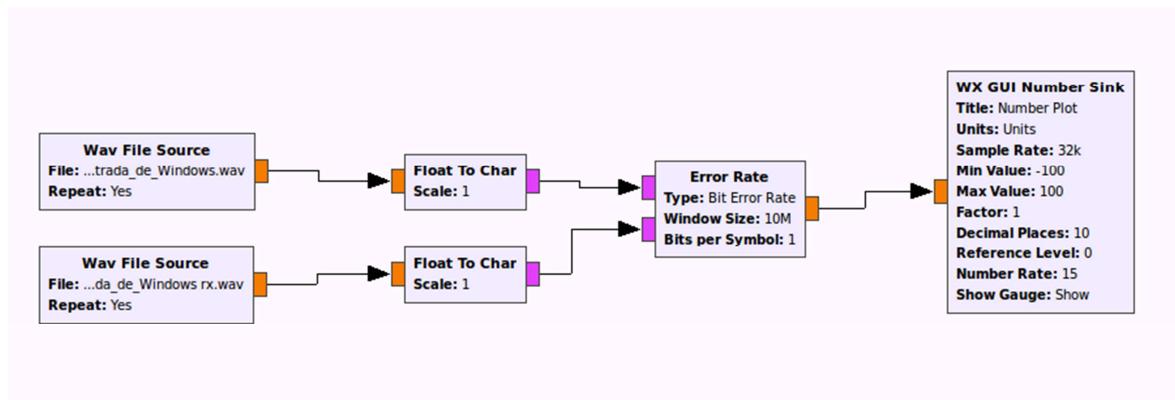


Figura 77. Esquema utilizado en la medición del BER

Un ejemplo de la salida al realizar las mediciones de BER del bloque *Error Rate* se muestra en la siguiente figura

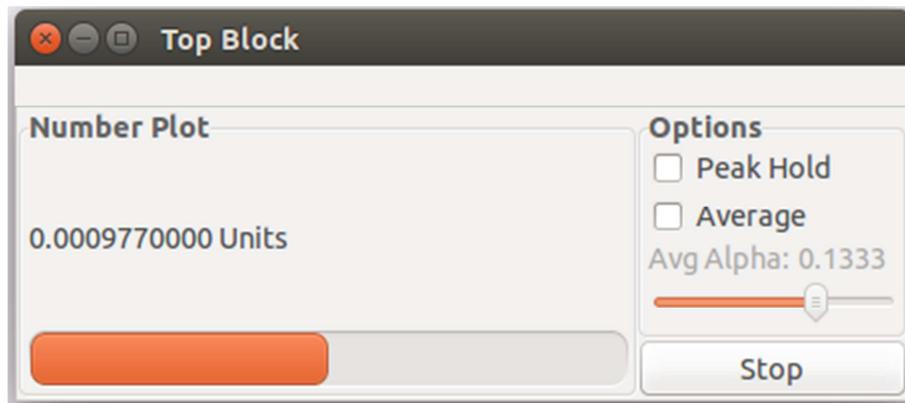
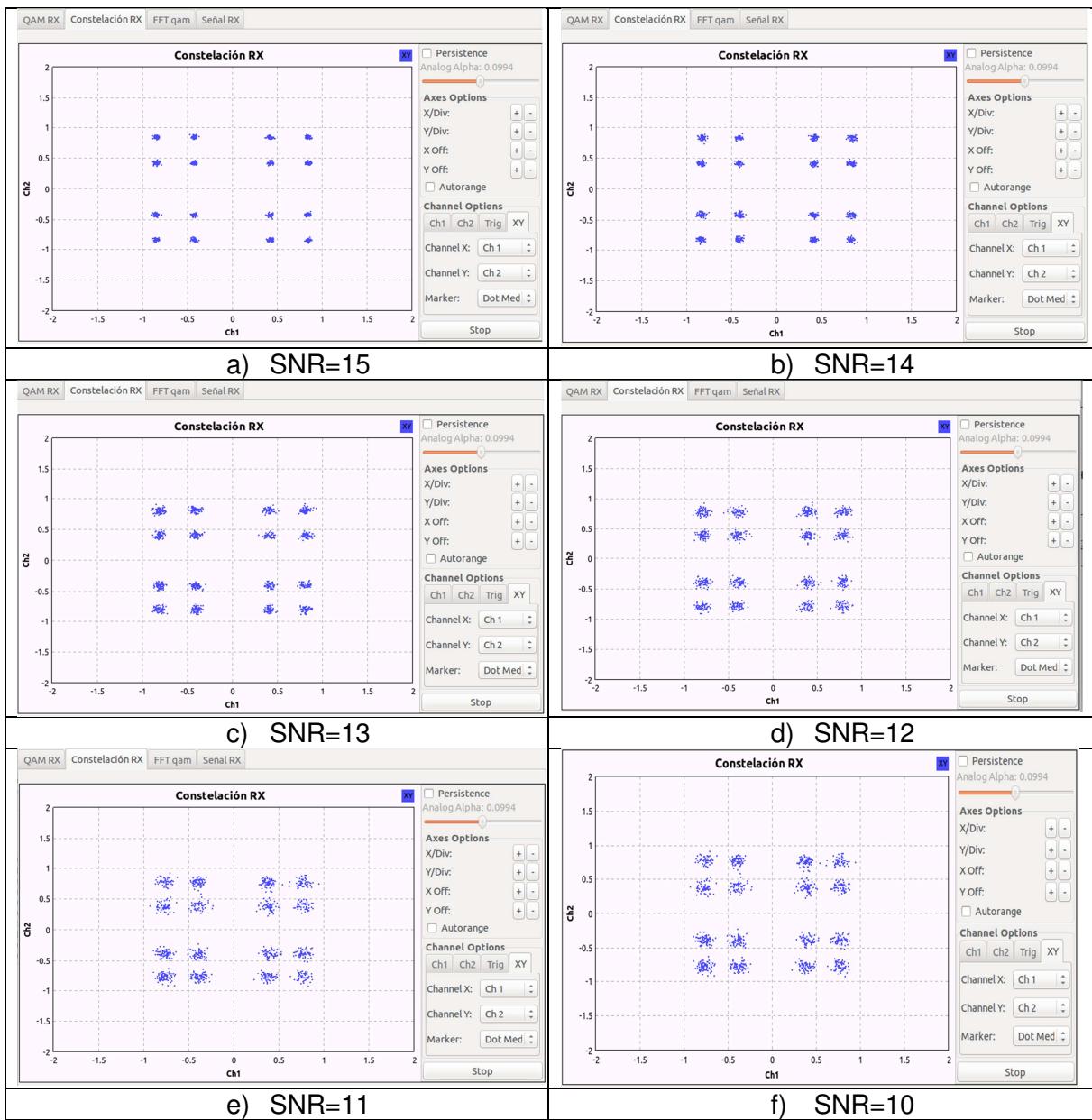
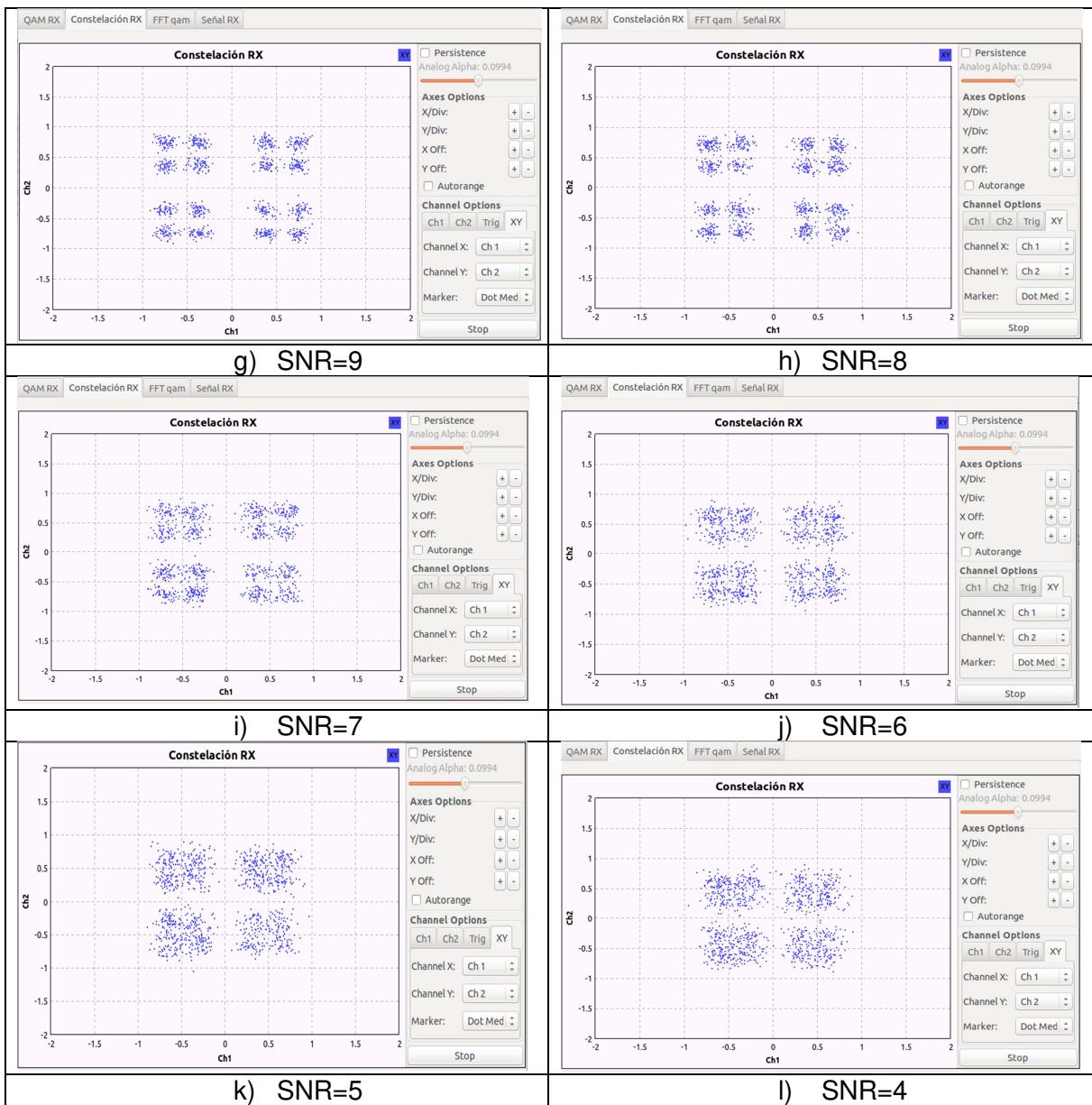


Figura 78. Resultado numérico de BER

Al realizar cambios en la distancia entre el trasmisor y receptor la potencia de recepción disminuye y el ruido aumenta, desde la distancia inicial de 13,5 cm calculada en el apartado 4.4.2 hasta una distancia final de 1,5m, en las siguientes figuras se muestra el aumento de ruido en la constelación CQAM conforme aumenta la distancia entre TX y RX.





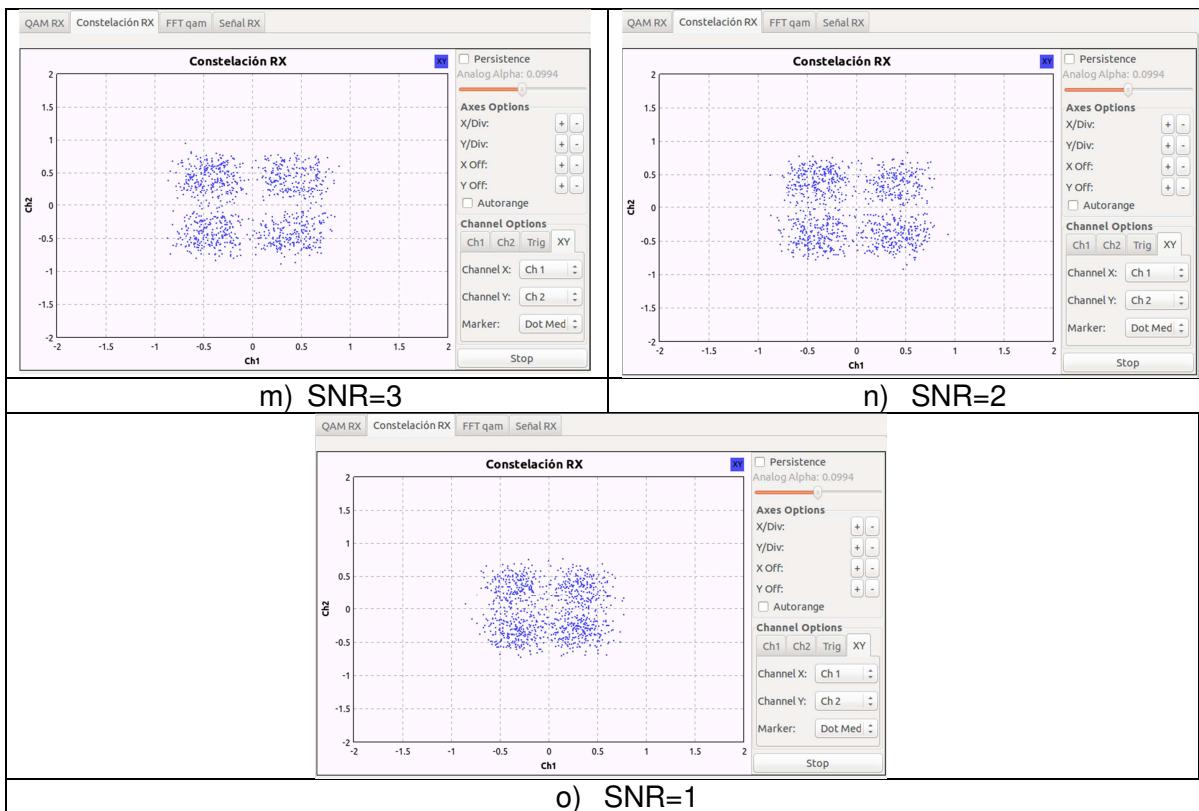


Figura 79. Comportamiento de la constelación recibida en el demodulador al aumentar la distancia entre las antenas TX y RX, así mismo la SNR de 15 a 1 (Figuras a – o)

Las mediciones se realizan para los esquemas de modulación convencional 16-QAM y el esquema de modulación jerárquico con conjuntos de cantor 16-CQAM utilizando hardware SDR (USRP B210 como transmisor y Hackrf-one como receptor), obteniendo los valores de BER en el entorno de GNU Radio

Al graficar los datos obtenidos de BER, se puede observar en la figura 80 el BER de los bits de alta y baja prioridad respecto a la relación señal a ruido para el esquema de modulación convencional 16-QAM ($\alpha = 1$) como para el esquema de modulación cantor 16-CQAM ($\alpha = 2$) usando hardware SDR; así se evidencia un comportamiento similar al que se observó en las simulaciones (figura 74) en donde también los bits de alta prioridad presentan una mayor protección al ruido, esto penalizando los bits de baja prioridad, como aquí se muestra este comportamiento para 16-QAM y 16-CQAM es evidente como mejora la protección de los bits de alta prioridad, confirmando los resultados obtenidos en la simulación.

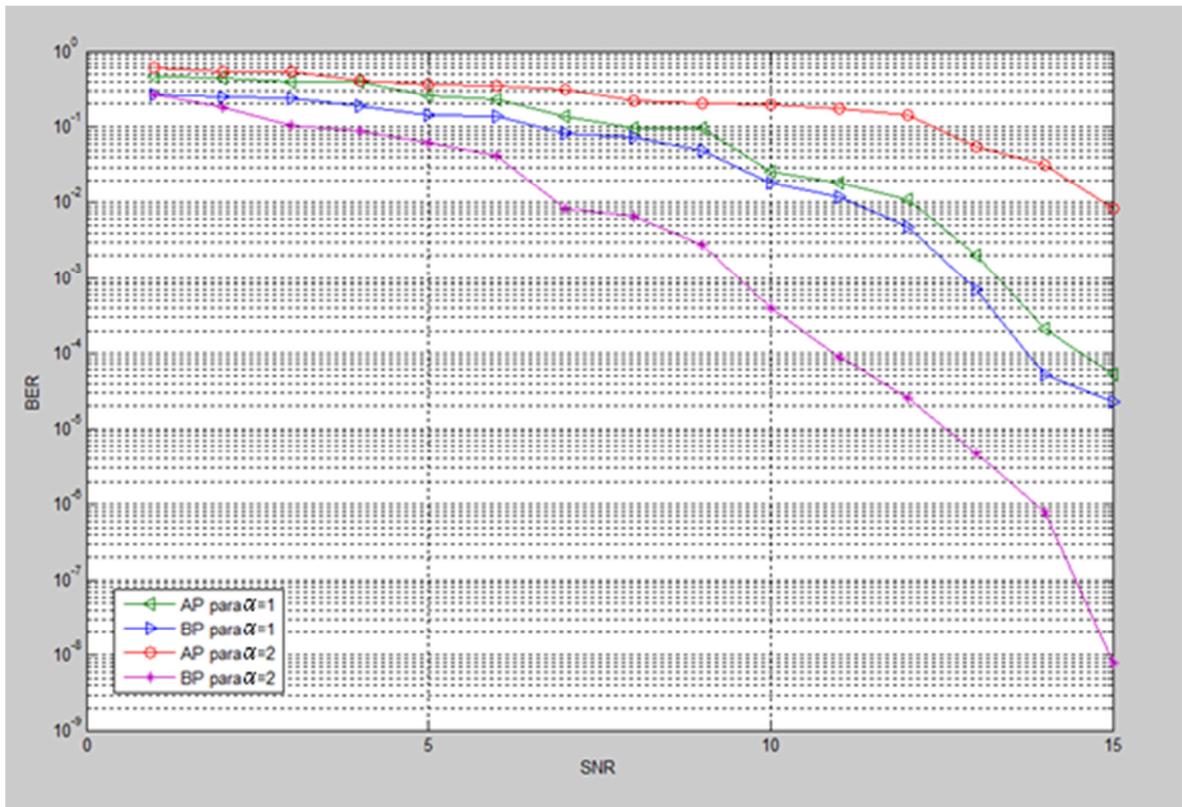


Figura 80. BER vs SNR para los bits de alta y baja prioridad en los esquemas de modulación 16-QAM ($\alpha = 1$) y 16-CQAM ($\alpha = 2$).

SNR	Implementación 1	Simulación 1	Implementación 2	Simulación 2	Implementación 3	Simulación 3	Implementación 4	Simulación 4
	BP $\alpha=1$	BP $\alpha=1$	AP $\alpha=1$	AP $\alpha=1$	BP $\alpha=2$	BP $\alpha=2$	AP $\alpha=2$	AP $\alpha=2$
1	0,465	0,31		0,27	0,18	0,615	0,41	0,265
5	0,26	0,16		0,14	0,079	0,36	0,29	0,063
10	0,026	0,013		0,018	0,0065	0,198	0,078	0,0004
15	0,000051	0,000022		0,0000225	0,0000085	0,0084	0,0028	0,00000008
								1,40E-09

Tabla 3. Valores de SNR Vs BER para algunos valores representativos de la relación señal a ruido, tanto para simulación como para la implementación.

En la Tabla 3, para valores 1, 5, 10 y 15 de la relación señal a ruido se observa que en general los valores tienden a dar valores cercanos a la simulación, y también se observa que en general y para la mayoría de los datos los valores del BER son siempre más bajos para la simulación como es de esperarse.

Capítulo 6

Conclusiones

Se logró implementar el modelo matemático del sistema 16-CQAM a través de Hardware SDR, de gran utilidad para realizar la implementación de prototipos y ambientes de prueba, sin embargo, requieren una capacitación previa para poder utilizarlos de manera adecuada, por lo que después de superar la curva de aprendizaje de estos sistemas y lenguajes de programación (Python y C++) se puede realizar la implementación de prototipos de una forma versátil y eficiente, dado que el hardware es adaptable.

Es posible realizar gran variedad de modelados e implementaciones sobre la plataforma de GNU Radio, sin embargo la documentación de muchos de estos bloques es reducida y en ocasiones es necesario validar todo el código fuente de los bloques para poder tener una idea del funcionamiento de los mismos, lo que no es transparente en muchas etapas y requiere gran cantidad de pruebas y análisis para poder utilizar el ambiente de una forma adecuada.

Se realizó la caracterización en cuanto a sus características de transmisión en la frecuencia de trabajo de los dispositivos USRP B210 y Hack RF. Para ello, se ha utilizado dispositivos de medición externos tal que verifiquen que se controla la transmisión y recepción de señales RF, una vez quedan caracterizados los dispositivos como transmisor y receptor de RF, el siguiente paso es controlar la herramienta software con la que se diseñan las aplicaciones, para ello el paso más sensato es operar en un entorno de simulación con la herramienta software (GNU Radio) tal que se diseñe el sistema a implementar sin recurrir al uso de ningún hardware adicional.

Las mediciones de BER en relación a la SNR permitieron confirmar como se muestra en la tabla 3 que el esquema CQAM brinda una protección adicional sobre los bits de alta prioridad con respecto a un esquema QAM esto con la penalidad de aumentar la exposición de los bits de baja prioridad ante el ruido, aumentando así el BER para estos bits de baja prioridad y comprobando de manera experimental la tesis planteada por Fonseca y Capera [2] para una relación de $\alpha = 2$ en comparación a un esquema convencional QAM ($\alpha = 1$).

Trabajo Futuro

Como trabajo futuro, con el aporte de los conocimientos necesarios para poder implementar sistemas de modulación, se podría proseguir el trabajo estudiando esquemas de modulación CQAM con mayor cantidad de estados (64-CQAM, 128-CQAM, etc.), diferentes valores de α y otros esquemas de modulación (FSK, PSK, etc.), que han quedado fuera del alcance de este proyecto apoyándose en el trabajo ya realizado.

BIBLIOGRAFÍA

- [1] M. A. K. Farid Ghanim, Hierarchical Quadrature Amplitude Modulation For Image Transmision Over Erroneous Wireless Channels, vol. 11, International Journal of Video & Image, 2011.
- [2] Fonseca Rafael, Capera Einer, Diseño y Simulación de un modulador y demodulador QAM Jerarquico con base en los conjuntos de Cantor, 2015.
- [3] J. P. M. Hidalgo, *Implementación de un sistema de comunicacion basado en Software Radio*, Madrid: Universidad Autonoma de Madrid, 2014.
- [4] A. M. M. M. B. Sruthi, «Low cost digital transceiver design for Software Defined Radio using RTL-SDR,» *Automation, Computing, Communication, Control and Compressed Sensing (iMac4s), 2013 International Multi-Conference on*, pp. 852-855, 2013.
- [5] L.-F. Wei, Coded Modulation with unequal Error Protection, vol. 41. No 10., IEEE Transactions on Communications, 1993.
- [6] L. S. S. Simon Görtzen, «Hierarchical Generalized Cantor Set Modulation,» *8th International Symposium on Wireless Communication System , Aachen*, pp. 56-60, 2011.
- [7] M. S. Jhon G proakis, Digital Communications, New York: McGraw Hill Higer Education, 2001.
- [8] B. Lathi, Modern Digital And Analog Communication Systems, Oxford: Oxford University Press, 1998.
- [9] R. P. d. T. y. M.- U. d. M. Sotelo, «Modulación Digital Aplicación a la Televisión Digital en DVB,» MEMORIAS, 2008.
- [10] W. W. T. K. L.Hanzo, Basic QAM Techniques" in Single and multi carrier quadrature amplitude modulation: principles and applications for personal communications, WLANs and broadcasting,, John W. & S., 2000.
- [11] G. Cantor, «On the Power of Perfect Sets of Points (De la puissance des ensembles parfait de points),» *Acta Mathematica* 4, pp. 381 -392, 1884.
- [12] S. I. Khan, «An exploration of the Generalized Cantor Set,» *INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH VOLUME 2, ISSUE 7*, pp. 50-54, 2013.

- [13] R. Sher, «CONCERNING WILD CANTOR SETS IN E3,» *Proc. Amer. Math Soc.*, pp. 1195-1200, 1968.
- [14] C. S. N.Seshadri, «Coded modulation with time diversity, unequal error protection and low delay for Rayleigh fading channel,» *Universal Personal Communications*, pp. 283-287, 1992.
- [15] F. C. D. Nuno Suoto, «On the BER Performance of Hierarchical M-QAM Constellations with Diversity and Imperfect Channel Estimation,» *IEEE Transaction on communications*, Vol 55 NO 10, pp. 1852-1856, 2007.
- [16] M. S. A. Pavan K. Vitthaladevuni, «A recursive Algorithm for Exact BER Computation of Generalized Hierarchical QAM Constellations,» *IEEE Transactions ON INFORMATION THEORY*, VOL 49 NO. 1, pp. 297-307, 2003.
- [17] J. J. M. F. Iván Pinar Domínguez, Laboratorio de Comunicaciones Digitales Radio Definida por Software, vol. 1, Sevilla: Universidad de Sevilla, 2011.
- [18] «GNU Radio Companion,» DOCKBook, 2017. [En línea]. Available: http://www.ece.uvic.ca/~elec350/grc_doc/ar01s03s07.html. [Último acceso: 17 11 2017].
- [19] «GNURadio,» [En línea]. Available: <http://www.gnuradio.org>.
- [20] «API GNU Radio Companion,» [En línea]. Available: <http://gnuradio.org/doc/doxygen/>.
- [21] W. Wolfgang Damm, «Signal-to-Noise, Carrier-to-Noise, EbNo,» Noisecom, 2010.
- [22] «GNU Radio Companion - Packet Encoder/Decoder,» [En línea]. Available: http://aaronsscher.com/GNU_Radio_Companion_Collection/Packet_encode_decode.html. [Último acceso: 18 11 2017].
- [23] J. Chennamangalam, «The Polyphase Filter Bank Technique,» 10 2016. [En línea]. Available: https://casper.berkeley.edu/wiki/The_Polyphase_Filter_Bank_Technique. [Último acceso: 18 11 2016].
- [24] 2016. [En línea]. Available: http://static1.1.sqspcdn.com/static/f/679473/25385817/1409511797677/rondeau-03-digital_demodulation.pdf?token=VLdIIIZq8xQmopwXjPf2Q5AgzkY%3D. [Último acceso: 18 11 2016].

- [25] «semtech,» 2016. [En línea]. Available: http://www.semtech.com/images/datasheet/xe8000_tn_09_locked_loop.pdf. [Último acceso: 18 11 2016].
- [26] J. Feigin, «Desinging a simple and inexpensive BPSK Costas Loop Carrier recovery circuit,» de *Practical Costas Loop Design*, RF Signal Processing, 2002, pp. 20-36.
- [27] F. J. Harris, Multirate Signal Processing for Communications Systems, United states: Pearson, 2004.
- [28] J. h. a. M. Rice, «Multirate Digital Filters for Symbol Timing Synchronization in Software Defined Radios,» *IEEE Selected Areas in Communications*, Vol. 19, No. 12, 2001.
- [29] E. Research, «Especificaciones Tecnicas USRP B210,» 2004. [En línea]. Available: https://www.ettus.com/content/files/kb/b200-b210_spec_sheet.pdf.
- [30] «HackRF,» 2009. [En línea]. Available: <https://greatscottgadgets.com/hackrf/>.
- [31] J. C. G. Paredes, Sistemas de Telecomunicaciones Planeación y Calculo de Enlaces.
- [32] C. A. Balanis, Antenna Theory Analysis And Design, New Jersy: Wiley Interscience, 2006.
- [33] K. Technologies, «How do I measure Bit Error Rate (BER) to a given confidence level,» [En línea]. Available: <http://www.keysight.com/main/editorial.jspx?ckey=1481106&id=1481106&nid=-11143.0.00&lc=eng&cc=CO>.
- [34] B. Sklar, Digital Fundamentals and Applications, 2nd ed., Prentice Hall, 1990.
- [35] D. U. R.I. Lackey, «SpeakEasy: The Military Software Radio,» *IEEE Communications Magazine*, Vol 33, pp. 5-6-61, 1995.
- [36] D. G. M. Edward A. Lee, Digital Communication, Second ed., vol. 16, 1994.
- [37] M. K. Mirabbasi S, «Hierarchical QAM: a spectrally efficient dc- free modulation scheme,» *Communications Magazine, IEEE*, vol. 38, pp. 140-146, Noviembre 2000.
- [38] «GNU Radio Companion,» DockBook, 17 11 2016. [En línea]. Available: http://www.ece.uvic.ca/~elec350/grc_doc/ar01s07s03.html. [Último acceso: 17 11 2016].

- [39] «Ruby Forum : Forum: GNU Radio,» Ruby Forum, [En línea]. Available: <https://www.ruby-forum.com/topic/4402902>. [Último acceso: 18 11 2017].
- [40] «Multirate Digital,» *IEEE Selected Areas in Communications*, vol. 19, nº 12, Dec, 2001.
- [41] «The joint Tactical Radio System and the Army's Future Combat System: Issues for Congress,» 24 04 2016. [En línea]. Available: http://digital.library.unt.edu/ark:/67531/metacrs7941/m1/1/high_res_d/RL33161_2005_N.

Anexo A.: Puesta en marcha del software necesario:

En este proyecto se indicará los pasos a seguir para el correcto uso de las herramientas software necesarias para poder desarrollar un sistema de comunicaciones basado en plataformas software, en particular las herramientas que se instalarán son GNU Radio + UHD para Ubuntu

Instalación y configuración de GNU Radio + UHD

Las herramientas software GNU Radio y UHD pueden ser instaladas en plataformas como LINUX, Mac y Windows.

Para instalar dichas herramientas hay dos posibilidades:

- Instalar el software mediante paquetes pre-compilados.
- Compilarlo manualmente.

La primera de ellas es más cómoda para el usuario, sin embargo, presenta claros inconvenientes entre ellos la alta probabilidad de instalar versiones obsoletas de GNU Radio, o la dificultad que implica modificar parte del proyecto GNU Radio en este método de instalación, por el contrario la otra opción presenta una mayor flexibilidad aunque la dificultad en el proceso de instalación aumenta considerablemente.

En este proyecto se detallará minuciosamente los pasos a seguir para la correcta instalación y configuración de las herramientas software en Ubuntu

Instalación y configuración de GNU Radio y UHD en Ubuntu:

Con motivo de simplificar el proceso de instalar (compilando manualmente) la herramienta software, Marcus Leech desarrolló un script de instalación, el cual se encarga de descargar e instalar todo el software necesario. En este proyecto se decidió emplear este método, siendo éste el método recomendado para la mayoría de los usuarios.

Los pasos a seguir para la puesta a punto del software necesario son los siguientes:

- Abrir la terminal y escribir:

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

- Para utilizar la tarjeta USRP B210 se recomienda instalar y/o actualizar previamente los UHD, para esto:

```
>>$ sudo apt-get install libuhd-dev libuhd003 uhd-host
```

Para comprobar que se reconozca adecuadamente la tarjeta, se debe escribir el siguiente comando. Si la respuesta es contiene los parámetros de direccionamiento del dispositivo, la tarjeta habrá sido correctamente identificada.

```
>>$ uhd_find_devices
```

```
Linux; GNU C++ version 4.8.4; Boost_105400; UHD_003.009.git-186-g4082c748
-- Loading firmware image: /usr/local/share/uhd/images/usrp_b200_fw.hex... done
-----
-- UHD Device 0
-----
Device Address:
  type: b200
  name:
  serial: 3094DBA
  product: B210
```

Para verificar las características de la tarjeta de desarrollo se debe ejecutar el siguiente comando en terminal:

```
>>$ uhd_usrp_probe
```

```
linux; GNU C++ version 4.8.4; Boost_105400; UHD_003.009.git-186-g4082c748
-- Loading firmware image: /usr/local/share/uhd/images/usrp_b200_fw.hex... done
-- Loading FPGA image: /usr/local/share/uhd/images/usrp_b210_fpga.bin... done
-- Operating over USB 3.
-- Detecting internal GPSDO.... No GPSDO found
-- Initialize CODEC control...
-- Initialize Radio control...
-- Performing register loopback test... pass
-- Performing register loopback test... pass
-- Performing CODEC loopback test... pass
-- Performing CODEC loopback test... pass
-- Asking for clock rate 32.000000 MHz...
-- Actually got clock rate 32.000000 MHz.
-- Performing timer loopback test... pass
-- Performing timer loopback test... pass
-- Setting master clock rate selection to 'automatic'.
```

```
| Device: B-Series Device
```

```
| | Mboard: B210
| | revision: 4
| | product: 2
| | serial: 3094DBA
| | FW Version: 7.0
```

| | | FPGA Version: 7.0
| | | Time sources: none, internal, external, gpsdo
| | | Clock sources: internal, external, gpsdo
| | | Sensors: ref_locked

| | | RX DSP: 0
| | | Freq range: -16.000 to 16.000 MHz

| | | RX DSP: 1
| | | Freq range: -16.000 to 16.000 MHz

| | | RX Dboard: A

| | | | RX Frontend: A
| | | | Name: FE-RX2
| | | | Antennas: TX/RX, RX2
| | | | Sensors: lo_locked, temp, rssI
| | | | Freq range: 50.000 to 6000.000 MHz
| | | | Gain range PGA: 0.0 to 76.0 step 1.0 dB
| | | | Bandwidth range: 200000.0 to 56000000.0 step 0.0 Hz
| | | | Connection Type: IQ
| | | | Uses LO offset: No

| | | | RX Frontend: B
| | | | Name: FE-RX1
| | | | Antennas: TX/RX, RX2
| | | | Sensors: lo_locked, temp, rssI
| | | | Freq range: 50.000 to 6000.000 MHz
| | | | Gain range PGA: 0.0 to 76.0 step 1.0 dB
| | | | Bandwidth range: 200000.0 to 56000000.0 step 0.0 Hz
| | | | Connection Type: IQ
| | | | Uses LO offset: No

| | | | RX Codec: A
| | | | Name: B210 RX dual ADC
| | | | Gain Elements: None

| | | TX DSP: 0
| | | Freq range: -16.000 to 16.000 MHz

| | | TX DSP: 1
| | | Freq range: -16.000 to 16.000 MHz

| | | TX Dboard: A

| | | | TX Frontend: A

		Name: FE-TX2
		Antennas: TX/RX
		Sensors: lo_locked, temp
		Freq range: 50.000 to 6000.000 MHz
		Gain range PGA: 0.0 to 89.8 step 0.2 dB
		Bandwidth range: 200000.0 to 56000000.0 step 0.0 Hz
		Connection Type: IQ
		Uses LO offset: No
<hr/>		
		TX Frontend: B
		Name: FE-TX1
		Antennas: TX/RX
		Sensors: lo_locked, temp
		Freq range: 50.000 to 6000.000 MHz
		Gain range PGA: 0.0 to 89.8 step 0.2 dB
		Bandwidth range: 200000.0 to 56000000.0 step 0.0 Hz
		Connection Type: IQ
		Uses LO offset: No
<hr/>		
		TX Codec: A
		Name: B210 TX dual DAC
		Gain Elements: None

Después de esto, la tarjeta USRP B210 estará lista para ser utilizada en GNU Radio para recibir y transmitir señales en RF

Anexo B: GNU Radio companion-Python-C++

Esta sección tiene por objetivo explicar el porqué del uso de diferentes lenguajes en una aplicación GNU Radio.

Como previamente ya se ha mencionado GNU Radio es una herramienta basado en tres capas o niveles de abstracción: GNU Radio companion, Python y C++. Los descriptores xml es el precio que se tiene que pagar por disponer de los bloques en la interfaz gráfica y Swig el precio de embeber C++ en Python.

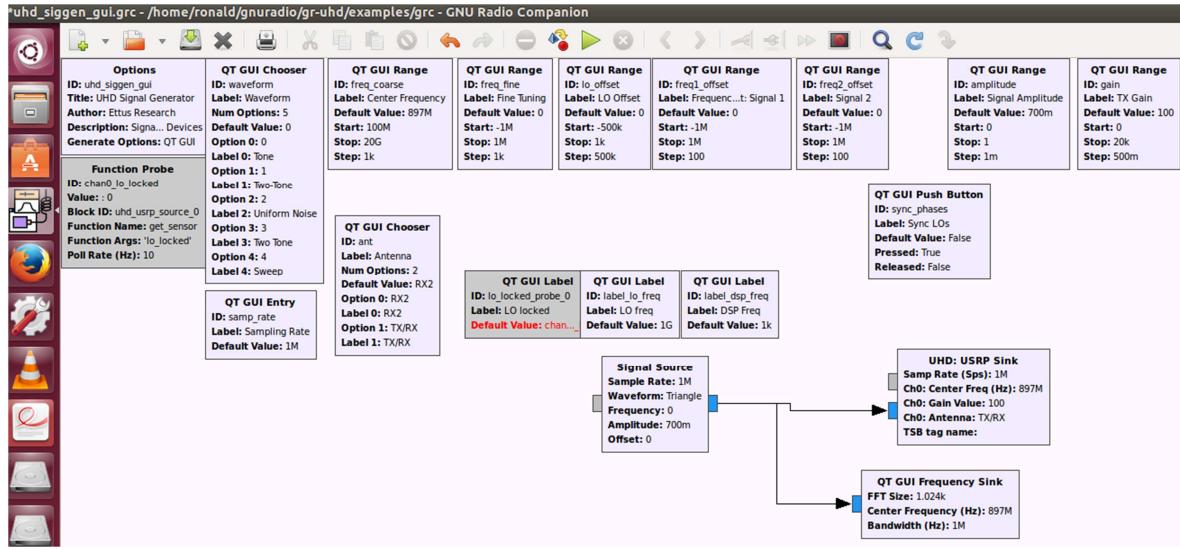
Tal y como se ha mencionado, el núcleo de GNU Radio, los bloques de procesado, están desarrollados en C++. La pregunta a formular en este punto es ¿por qué utilizar C++ como lenguaje *core*? La respuesta reside en las ventajas que aporta este lenguaje, se trata de un lenguaje de nivel intermedio ya que comprende características tanto de alto nivel como de bajo presentando un buen rendimiento, además de permitir la programación orientada a objetos.

Siendo así, la siguiente pregunta que surge es ¿por qué utilizar Python? Para contestar a esta pregunta, primero se ha de mencionar los aspectos negativos que presenta C++. Se trata de un lenguaje que no presenta buenas prestaciones a la hora de realizar interfaces para interactuar con el usuario. Además, no es buen lenguaje para la integración. Python es un lenguaje de alto nivel de *scripting* y por tanto no se necesita compilado, que aporta grandes ventajas como es la integración (*glue*) y la interacción con el usuario.

Por último, la última pregunta que hay que formular es ¿Por qué utilizar GNU Radio companion? GNU Radio companion (GRC) es la interfaz gráfica de la que el desarrollador puede beneficiarse para escribir una aplicación GNU Radio completa sin escribir código, se trata por lo tanto de una herramienta gráfica opcional para simplificar el nivel de complejidad y de conocimientos en lenguajes de programación.

Anexo C. Pruebas de transmisión

Para las pruebas de transmisión se ha usado la interfaz uhd_siggen_gui.grc disponible en las librerías de GNU Radio, el diagrama de bloques se muestra a continuación:



Donde en este caso, se envía un tipo de señal (triangular) a una frecuencia configurable, al periférico (USRP B210), transmitiéndola a una frecuencia de 897 Mhz a máxima potencia (*gain* = 100)

Anexo D . Código Python de la Conversión a código Gray

El código de Python para la conversión de datos en código Gray es el siguiente

```
class GrayCodeGenerator(object):
    """
    Generates and caches gray codes.
    """

    def __init__(self):
        self.gcs = [0, 1]
        # The last power of two passed through.
        self.lp2 = 2
        # The next power of two that will be passed through.
        self.np2 = 4
        # Current index
        self.i = 2

    def get_gray_code(self, length):
        """
        Returns a list of gray code of given length.
        """
        if len(self.gcs) < length:
            self.generate_new_gray_code(length)
        return self.gcs[:length]

    def generate_new_gray_code(self, length):
        """
        Generates new gray code and places into cache.
        """
        while len(self.gcs) < length:
            if self.i == self.lp2:
                # if i is a power of two then gray number is of form 110000...
                result = self.i + self.i/2
            else:
                # if not we take advantage of the symmetry of all but the last bit
                # around a power of two.
                result = self.gcs[2*self.lp2-1-self.i] + self.lp2
            self.gcs.append(result)
            self.i += 1
            if self.i == self.np2:
                self.lp2 = self.i
                self.np2 = self.i*2

_gray_code_generator = GrayCodeGenerator()
gray_code = _gray_code_generator.get_gray_code
```

Anexo E. Código Python de la constelación

El siguiente código define la constelación que se va a utilizar, en este caso se ha generado partiendo del código existente en el fichero qam.py:

Se parte del archivo de Python qam.py en el que se importan todas las librerías necesarias y luego se definen variables y parámetros necesarios dependientes de los valores de entrada desde la interfaz de GNU Radio, el método qam_constellation dentro del fichero qam.py se muestra a continuación:

```
QAM modulation and demodulation.

"""

from math import pi, sqrt, log

from gnuradio import gr
from generic_mod_demod import generic_mod, generic_demod
from generic_mod_demod import shared_mod_args, shared_demod_args
from utils.gray_code import gray_code
from utils import mod_codes
import modulation_utils
import digital_swig as digital

# Default number of points in constellation.
def constellation_points = 16
# Whether the quadrant bits are coded differentially.
def differential = True
# Whether gray coding is used. If differential is True then gray
# coding is used within but not between each quadrant.
def mod_code = mod_codes.NO_CODE

def is_power_of_four(x):
    v = log(x)/log(4)
    return int(v) == v

def get_bit(x, n):
    """ Get the n'th bit of integer x (from little end)."""
    return (x&(0x01 << n)) >> n

def get_bits(x, n, k):
    """ Get the k bits of integer x starting at bit n(from little end)."""
    # Remove the n smallest bits
    v = x >> n
    # Remove all bits bigger than n+k-1
    return v % pow(2, k)
```

```

# ///////////////////////////////////////////////////////////////////
# // QAM constellation
# ///////////////////////////////////////////////////////////////////

def qam_constellation(constellation_points=_def_constellation_points,
                      differential=_def_differential,
                      mod_code=_def_mod_code,
                      large_ampls_to_corners=False):
    """
    Creates a QAM constellation object.

    If large_ampls_to_corners=True then sectors that are probably
    occupied due to a phase offset, are not mapped to the closest
    constellation point. Rather we take into account the fact that a
    phase offset is probably the problem and map them to the closest
    corner point. It's a bit hackish but it seems to improve
    frequency locking.
    """

    if mod_code == mod_codes.GRAY_CODE:
        gray_coded = True
    elif mod_code == mod_codes.NO_CODE:
        gray_coded = False
    else:
        raise ValueError("Mod code is not implemented for QAM")
    if differential:
        points = make_differential_constellation(constellation_points, gray_coded=False)
    else:
        points = make_non_differential_constellation(constellation_points, gray_coded)
    side = int(sqrt(constellation_points))
    width = 2.0/(side-1)

    # No pre-diff code
    # Should add one so that we can gray-code the quadrant bits too.
    pre_diff_code = []
    if not large_ampls_to_corners:
        constellation = digital.constellation_rect(points, pre_diff_code, 4,
                                                    side, side, width, width)
    else:
        sector_values = large_ampls_to_corners_mapping(side, points, width)
        constellation = digital.constellation_expl_rect(
            points, pre_diff_code, 4, side, side, width, width, sector_values)

    return constellation

```

En los esquemas de modulación QAM, HQAM y CQAM propuestos no se hará uso de codificación diferencial entonces, `points = make_non_differential_constellation(constellation_points, gray_coded)` y se guardaran los puntos de la constelación

```

def make_non_differential_constellation(m, gray_coded):
    side = int(pow(m, 0.5))
    if (not isinstance(m, int) or m < 4 or not is_power_of_four(m)):
        raise ValueError("m must be a power of 4 integer.")
    # Each symbol holds k bits.
    k = int(log(m) / log(2.0))
    if gray_coded:
        # Number rows and columns using gray codes.
        gcs = gray_code(side)
        # Get inverse gray codes.
        i_gcs = mod_codes.invert_code(gcs)
    else:
        i_gcs = range(0, side)
    # The distance between points is found.
    step = 2.0/(side-1)

    gc_to_x = [-1 + i_gcs[gc]*step for gc in range(0, side)]
    # First k/2 bits determine x position.
    # Following k/2 bits determine y position.
    const_map = []
    for i in range(m):
        y = gc_to_x[get_bits(i, 0, k/2)]
        x = gc_to_x[get_bits(i, k/2, k/2)]
        const_map.append(complex(x,y))
    return const_map

```

En general, el objeto constelación puede ser generado a través de distintas clases que derivan de la clase padre o base con la intención de explotar las características geométricas da cada constelación a la hora de llevar a cabo la demodulación. El método invocado en la figura, invoca a su vez a la clase digital_constellation_rect situada en el archivo digital_constellation.cc de código C++ encargada de crear el objeto constellation.

Para poder crear la constelación desde el archivo qam.py, se le ha de indicar el índice de modulación y si se trata de una modulación diferencial así como si se desea codificación Gray, aunque por defecto no se realiza.

Creación del objeto constellation mediante la clase digital_constellation_rect situada en el archivo digital_constellation.cc

```

*****  

constellation_rect::sptr
constellation_rect::make(std::vector<gr_complex> constell,
                        std::vector<int> pre_diff_code,
                        unsigned int rotational_symmetry,
                        unsigned int real_sectors,
                        unsigned int imag_sectors,
                        float width_real_sectors,
                        float width_imag_sectors)
{
    return constellation_rect::sptr(new constellation_rect
        (constell, pre_diff_code,
         rotational_symmetry,
         real_sectors, imag_sectors,
         width_real_sectors,
         width_imag_sectors));
}

constellation_rect::constellation_rect(
    std::vector<gr_complex> constell,
    std::vector<int> pre_diff_code,
    unsigned int rotational_symmetry,
    unsigned int real_sectors, unsigned int imag_sectors,
    float width_real_sectors, float width_imag_sectors) :
    constellation_sector(constell, pre_diff_code, rotational_symmetry,
                          1, real_sectors * imag_sectors),
    n_real_sectors(real_sectors), n_imag_sectors(imag_sectors),
    d_width_real_sectors(width_real_sectors),
    d_width_imag_sectors(width_imag_sectors)
{
    d_width_real_sectors *= d_scalefactor;
    d_width_imag_sectors *= d_scalefactor;
    find_sector_values();
}

```

Es necesario que desde qam.py en Python se llame a la función constellation_rect en C++, para que esto sea posible es necesario el fichero digital_swig encargado de la integración entre Python y C++ para los módulos digitales

```

class constellation_rect(constellation_sector):
    """
    Rectangular digital constellation

    Only implemented for 1-(complex)dimensional constellation.

    Constellation space is divided into rectangular sectors. Each sector is associated with the nearest constellation point.

    Works well for square QAM.

    Works for any generic constellation provided sectors are not too large.

    Constructor Specific Documentation:

    Make a rectangular constellation object.

    Args:
        constell : List of constellation points (order of list matches pre_diff_code)
        pre_diff_code : List of alphabet symbols (before applying any differential coding) (order of list matches constell)
        rotational_symmetry : Number of rotations around unit circle that have the same representation.
        real_sectors : Number of sectors the real axis is split in to.
        imag_sectors : Number of sectors the imag axis is split in to.
        width_real_sectors : width of each real sector to calculate decision boundaries.
        width_imag_sectors : width of each imag sector to calculate decision boundaries.
    """
    thisown = __swig_property(lambda x: x.this.own(), lambda x, v: x.this.own(v), doc='The membership flag')
    def __init__(self, *args, **kwargs): raise AttributeError("No constructor defined")
    __repr__ = __swig_repr__
    def make(*args, **kwargs):
        """
        make(pmt::vector<float> constell, std::vector<int, std::allocator<int>> pre_diff_code,
              unsigned int rotational_symmetry, unsigned int real_sectors, unsigned int imag_sectors,
              float width_real_sectors, float width_imag_sectors) -> constellation_rect_sptr

        Rectangular digital constellation

        Only implemented for 1-(complex)dimensional constellation.
        """

```

Adicional utiliza los siguientes métodos en C++ para el mapeo y cálculo de distancias en el plano complejo

```

constellation::~constellation()
{
}

/// Returns the constellation points for a symbol value
void
constellation::map_to_points(unsigned int value, gr_complex *points)
{
    for(unsigned int i=0; i<d_dimensionality; i++)
    | points[i] = d_constellation[value*d_dimensionality + i];
}

std::vector<gr_complex>
constellation::map_to_points_v(unsigned int value)
{
    std::vector<gr_complex> points_v;
    points_v.resize(d_dimensionality);
    map_to_points(value, &(points_v[0]));
    return points_v;
}

float
constellation::get_distance(unsigned int index, const gr_complex *sample)
{
    float dist = 0;
    for(unsigned int i=0; i<d_dimensionality; i++) {
        dist += norm(sample[i] - d_constellation[index*d_dimensionality + i]);
    }
    return dist;
}

unsigned int
constellation::get_closest_point(const gr_complex *sample)
{
    unsigned int min_index = 0;
    float min_euclid_dist;
    float euclid_dist;

    min_euclid_dist = get_distance(0, sample);
    min_index = 0;
    for(unsigned int j = 1; j < d_arity; j++){
        euclid_dist = get_distance(j, sample);
}

```

Para la creación de esquemas de modulación convencionales se utiliza la clase de constellation base

```

// Base Constellation Class
constellation::constellation(std::vector<gr_complex> constell,
                             std::vector<int> pre_diff_code,
                             unsigned int rotational_symmetry,
                             unsigned int dimensionality
)
:
d_constellation(constell),
d_pre_diff_code(pre_diff_code),
d_rotational_symmetry(rotational_symmetry),
d_dimensionality(dimensionality),
d_re_min(1e20),
d_re_max(1e20),
d_im_min(1e20),
d_im_max(1e20),
d_lut_precision(0),
d_lut_scale(0)
{
    // Scale constellation points so that average magnitude is 1.
    float summed_mag = 0;
    unsigned int constsize = d_constellation.size();
    for (unsigned int i=0; i<constsize; i++) {
        gr_complex c = d_constellation[i];
        summed_mag += sqrt(c.real()*c.real() + c.imag()*c.imag());
    }
    d_scalefactor = constsize/summed_mag;
    for (unsigned int i=0; i<constsize; i++) {
        d_constellation[i] = d_constellation[i]*d_scalefactor;
    }
    if(pre_diff_code.size() == 0)
        d_apply_pre_diff_code = false;
    else if(pre_diff_code.size() != constsize)
        throw std::runtime_error(
            "The constellation and pre-diff code must be of the same length.");
    else
        d_apply_pre_diff_code = true;
    calc_arity();
}

```

Mapeo constelación convencional 16QAM

```
constellation_16qam::sptr
constellation_16qam::make()
{
    return constellation_16qam::sptr(new constellation_16qam());
}

constellation_16qam::constellation_16qam()
{
    const float level = sqrt(float(0.1));
    d_constellation.resize(16);
    // The mapping used in 16qam set partition
    d_constellation[0] = gr_complex(1*level,-1*level);
    d_constellation[1] = gr_complex(-1*level,-1*level);
    d_constellation[2] = gr_complex(3*level,-3*level);
    d_constellation[3] = gr_complex(-3*level,-3*level);
    d_constellation[4] = gr_complex(-3*level,-1*level);
    d_constellation[5] = gr_complex(3*level,-1*level);
    d_constellation[6] = gr_complex(-1*level,-3*level);
    d_constellation[7] = gr_complex(1*level,-3*level);
    d_constellation[8] = gr_complex(-3*level,3*level);
    d_constellation[9] = gr_complex(3*level,3*level);
    d_constellation[10] = gr_complex(-1*level,1*level);
    d_constellation[11] = gr_complex(1*level,1*level);
    d_constellation[12] = gr_complex(1*level,3*level);
    d_constellation[13] = gr_complex(-1*level,3*level);
    d_constellation[14] = gr_complex(3*level,1*level);
    d_constellation[15] = gr_complex(-3*level,1*level);
    d_rotational_symmetry = 4;
    d_dimensionality = 1;
    calc_arity();
}

constellation_16qam::~constellation_16qam()
{
}
```

Mapeo 16QAM en constellation.cc

```
constellation_16qam::decision_maker(const gr_complex *sample)
{
    unsigned int ret = 0;
    const float level = sqrt(float(0.1));
    float re = sample->real();
    float im = sample->imag();

    if(im <= 0 && im >= -2*level && re >= 0 && re <= 2*level)
        ret = 0;
    else if(im <= 0 && im >= -2*level && re <= 0 && re >= -2*level)
        ret = 1;
    else if(im <= -2*level && re >= 2*level)
        ret = 2;
    else if(im <= -2*level && re <= -2*level)
        ret = 3;
    else if(im <= 0 && im >= -2*level && re <= -2*level)
        ret = 4;
    else if(im <= 0 && im >= -2*level && re >= 2*level)
        ret = 5;
    else if(im <= -2*level && re <= 0 && re >= -2*level)
        ret = 6;
    else if(im <= -2*level && re >= 0 && re <= 2*level)
        ret = 7;
    else if(im >= 2*level && re <= -2*level)
        ret = 8;
    else if(im >= 2*level && re >= 2*level)
        ret = 9;
    else if(im >= 0 && im <= 2*level && re <= 0 && re <= -2*level)
        ret = 10;
    else if(im >= 0 && im <= 2*level && re >= 0 && re <= 2*level)
        ret = 11;
    else if(im >= 2*level && re >= 0 && re <= 2*level)
        ret = 12;
    else if(im >= 2*level && re <= 0 && re >= -2*level)
        ret = 13;
    else if(im >= 0 && im <= 2*level && re >= 2*level)
        ret = 14;
    else if(im >= 0 && im <= 2*level && re <= -2*level)
        ret = 15;

    return ret;
}
```

Con los parámetros impuestos en el objeto de la constelación, la clase constellation_rect tendrá en cuenta estos parámetros para la demodulación dentro del método decision_maker

```

std::vector<float>
constellation::soft_decision_maker(gr_complex sample)
{
    if(has_soft_dec_lut()) {
        // Clip to just below 1 --> at 1, we can overflow the index
        // that will put us in the next row of the 2D LUT.
        float xre = branchless_clip(sample.real(), 0.99);
        float xim = branchless_clip(sample.imag(), 0.99);

        // We normalize the constellation in the ctor, so we know that
        // the maximum dimensions go from -1 to +1. We can infer the x
        // and y scale directly.
        float scale = d_lut_scale / (2.0f);

        // Convert the clipped x and y samples to nearest index offset
        xre = floorf((1.0f + xre) * scale);
        xim = floorf((1.0f + xim) * scale);
        int index = static_cast<int>(d_lut_scale*xim + xre);

        int max_index = d_lut_scale*d_lut_scale;

        // Make sure we are in bounds of the index
        while(index >= max_index) {
            index -= d_lut_scale;
        }
        while(index < 0) {
            index += d_lut_scale;
        }

        return d_soft_dec_lut[index];
    }
    else {
        return calc_soft_dec(sample);
    }
}

```

Anexo F. Código Python Bloque cálculo del Error Rate

Código del bloque Error Rate en Python para la medición del BER

```
self._num_errs = 0
self._err_index = 0
self._num_samps = 0
self._err_array = numpy.zeros(self._max_samples, numpy.int8)
if type == 'BER':
    input_watcher(msgq_sink, self._handler_ber)
elif type == 'SER':
    input_watcher(msgq_sink, self._handler_ser)
#connect
self.connect(msg_source, self)
self.connect((self, 0), (inter, 0))
self.connect((self, 1), (inter, 1))
self.connect(inter, msg_sink)

def _handler_ber(self, samples):
    num = len(samples)/2
    arr = numpy.zeros(num, numpy.float32)
    for i in range(num):
        old_err = self._err_array[self._err_index]
        #record error
        self._err_array[self._err_index] = _1s_counts[samples[i*2] ^ samples[i*2 + 1]]
        self._num_errs = self._num_errs + self._err_array[self._err_index] - old_err
        #increment index
        self._err_index = (self._err_index + 1)%self._max_samples
        self._num_samps = min(self._num_samps + 1, self._max_samples)
        #write sample
        arr[i] = float(self._num_errs)/float(self._num_samps*self._bits_per_symbol)
    #write message
    msg = gr.message_from_string(arr.tostring(), 0, gr.sizeof_float, num)
    self._msgq_source.insert_tail(msg)

def _handler_ser(self, samples):
    num = len(samples)/2
    arr = numpy.zeros(num, numpy.float32)
    for i in range(num):
```