



UNIVERSIDAD DE CARABOBO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA DE

TELECOMUNICACIONES

DEPARTAMENTO DE SEÑALES Y SISTEMAS



**DESARROLLO DE UN GENERADOR VECTORIAL DE ONDA
ARBITRARIA BASADO EN RADIO DEFINIDA POR SOFTWARE Y
SOFTWARE LIBRE**

JORGE DE CASTRO
KEVIN HENRIQUEZ

Bárbula, 2 de Julio del 2015



UNIVERSIDAD DE CARABOBO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA DE

TELECOMUNICACIONES



DEPARTAMENTO DE SEÑALES Y SISTEMAS

**DESARROLLO DE UN GENERADOR VECTORIAL DE ONDA
ARBITRARIA BASADO EN RADIO DEFINIDA POR SOFTWARE Y
SOFTWARE LIBRE**

TRABAJO ESPECIAL DE GRADO PRESENTADO ANTE LA ILUSTRE UNIVERSIDAD DE
CARABOBO PARA OPTAR AL TÍTULO DE INGENIERO DE TELECOMUNICACIONES

JORGE DE CASTRO
KEVIN HENRIQUEZ

Bárbula, 2 de Julio del 2015



UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA DE
TELECOMUNICACIONES



DEPARTAMENTO DE SEÑALES Y SISTEMAS

CERTIFICADO DE APROBACIÓN

Los abajo firmantes miembros del jurado asignado para evaluar el trabajo especial de grado titulado «DESARROLLO DE UN GENERADOR VECTORIAL DE ONDA ARBITRARIA BASADO EN RADIO DEFINIDA POR SOFTWARE Y SOFTWARE LIBRE», realizado por los bachilleres JORGE DE CASTRO, cédula de identidad 19.755.372, KEVIN HENRIQUEZ, cédula de identidad 20.696.184, hemos decidido otorgar la máxima calificación y la mención honorífica al presente trabajo, con base a los siguientes motivos:

1) *El trabajo integra de manera estructurada, sistemática y detallada de los aspectos conceptuales y metodológicos usados para desarrollo del prototipo.* 2) *El prototipo desarrollado quedó integrado como una plataforma de amplio uso para las actividades de laboratorio y futuras investigaciones en el área de los Sistemas de Comunicaciones.* 3) *Los autores demostraron un dominio muy amplio del tema tratado.*

Firma

Prof. CARLOS MEJIAS

TUTOR

Firma

Prof. ANTONIO FEDÓN

JURADO

Firma

Prof. AHMAD OSMAN

JURADO

Bárbula, 2 de Julio del 2015

Dedicatoria

*Para Jorge y Suzana,
Mis guías, mi apoyo, mi vida
Para mis abuelos, Adriano y Gastón,
Siempre les deberé ese último adiós*

JORGE DE CASTRO

*Para Lisbeth,
Siempre estarás conmigo*

KEVIN HENRIQUEZ

Agradecimientos

Antes que todo, queremos agradecer a nuestro tutor, el ingeniero Carlos Mejías por su invaluable orientación y atención a lo largo de la realización de este trabajo.

A los profesores de la Escuela de Telecomunicaciones de la Universidad de Carabobo por su labor y dedicación en la formación de nuevos profesionales. También queremos agradecer al profesor Antonio Fedón, por todo su apoyo y ayuda en el desarrollo de este trabajo.

A nuestros padres y familiares, Erwin, Erika, Erwin Gabriel, Anelise, Vicente, Jorge, Suzana, Lilisbeth, Adriano y Fiorella, ya que sin su apoyo constante nada de esto sería posible.

Por último, a nuestros compañeros de estudio, aquellos que estuvieron desde el inicio y a los que conocimos a largo del camino, y que hicieron del mismo, más allá de lo académico, una etapa de crecimiento personal.

Índice general

Índice de Figuras	xii
Índice de Tablas	xv
Acrónimos	xvii
Resumen	xix
I. Introducción	1
1.1. Motivación	1
1.2. Objetivos	4
1.2.1. Objetivo General	4
1.2.2. Objetivos Específicos	4
1.3. Alcance	5
II. Marco conceptual	7
2.1. Generador Vectorial de Radio Frecuencia (GVRF)	7
2.2. Generador Arbitrario de Señales (GAS)	7
2.3. Radio Definida por Software (SDR)	8
2.4. Tarjeta de desarrollo HackRF One	9
2.4.1. Microntrolador LPC43xx ARM Cortex-M4	10
2.4.2. Convertidor Analógico-Digital/Digital Analógico MAX5864. Maxim Integrated	10
2.4.3. Transceptor RF de banda ancha MAX2837. Maxim Integrated	10
2.4.4. Mezclador/Sintetizador RFFC5072 RFMD	11
2.4.5. Amplificador MGA81563. Avago Technologies	11
2.5. GNU Radio	11
2.6. Modulación	11
2.6.1. Modulación en amplitud	12
2.6.1.1. Portadora Suprimida (DSB-SC)	12
2.6.1.2. Gran Portadora (AM)	13
2.6.1.3. Modulación de Amplitud en Cuadratura (QAM)	14
2.6.1.4. Modulación por Desviación de Amplitud (ASK)	16

2.6.2. Modulación de Ángulo	17
2.6.2.1. Modulación en Frecuencia (FM)	18
2.6.2.2. Modulación por Desviación de Frecuencia (FSK)	20
2.6.2.3. Modulación por desviación de Frecuencia Gaussiana (GFSK)	22
2.6.2.4. Modulación por Desviación de Fase (PSK)	24
2.6.2.5. Modulación por Desviación Mínima (MSK)	26
2.6.2.6. Modulación por Desviación Mínima Gaussiana (GMSK)	28
2.7. Multiplexación por División de Frecuencias Ortogonales (OFDM)	30
2.7.1. Multiplexación por División de Frecuencias Ortogonales Codificada (COFDM)	33
2.8. Técnicas de Modulación y Codificación	35
2.8.1. Variable Coding and Modulation (VCM)	35
2.8.1.1. Adaptative Coding and Modulation (ACM)	36
2.9. Estándares Internacionales de Transmisión de Televisión	37
2.9.1. NTSC	37
2.9.2. DVB	39
2.9.2.1. DVB-T	41
2.9.2.2. DVB-S2	42
2.10. MPEG-2 Transport Stream	44
III. Procedimientos de la investigación	47
3.1. Revisión	47
3.2. Pruebas de Adquisición	48
3.3. Elaboración de diagramas de bloques en GNU Radio	49
3.4. Caracterización del dispositivo	50
3.5. Calibración	51
3.5.1. En Frecuencia	51
3.5.2. En Potencia	51
3.5.3. En Amplitud	53
3.6. Diseño de la Aplicación	54
IV. Análisis, interpretación y presentación de los resultados	55
4.1. Estructura del Generador Vectorial de Onda Arbitraria Basado en SDR	55
4.2. Generación de Señales	57
4.2.1. Modulaciones Analógicas	58
4.2.1.1. Portadora Suprimida (DSB-SC)	58
4.2.1.2. Gran Portadora (AM)	61
4.2.1.3. Modulación en Frecuencia (FM)	64
4.2.2. Modulaciones Digitales	67
4.2.2.1. Modulación por Desviación de Amplitud (ASK)	68

4.2.2.2. Modulación por Desviación de Fase (PSK) y Modulación de Amplitud en Cuadratura (QAM)	72
4.2.2.3. Modulación por desviación de Frecuencia Gaussiana (GFSK) y Modulación por Desviación Mínima Gaussiana (GMSK)	75
4.2.3. Multiplexación por División de Frecuencias Ortogonales (OFDM)	79
4.2.4. Estándares Internacionales de Transmisión de Televisión	82
4.2.4.1. NTSC	82
4.2.4.2. DVB-T	84
4.2.4.3. DVB-S2	86
4.3. Calibración	87
4.3.1. En Frecuencia	87
4.3.2. En Potencia	88
4.3.2.1. Ajuste por Esquema	90
4.3.3. En Amplitud	97
4.4. Desarrollo de la Aplicación	100
4.5. Especificaciones Técnicas	103
V. Conclusiones y recomendaciones	109
5.1. Conclusiones	109
5.1.1. Recomendaciones	110
 A. Tabla de Canales del Estándar NTSC	 113
 Referencias Bibliográficas	 119
 Anexos	
 A. Listado de Programas	

Índice de figuras

2.1. Diagrama de Bloques de un Generador Arbitrario de Señales de un Canal [1]	8
2.2. Etapa digital de la HackRF One [2]	9
2.3. Etapa de radio de la HackRF One [2]	10
2.4. Diagrama de bloques y señal QAM temporal [3]	14
2.5. Espectro de la señal QAM — línea punteada. Forma del filtro de co-seno realzado — línea sólida [4]	15
2.6. Diagrama de Bloques para la generación de una señal ASK binaria [5]	16
2.7. Representación temporal de una señal ASK binaria [6]	17
2.8. Espectro de una señal ASK binaria [5]	17
2.9. Diagrama de Bloques de la generación de una señal FSK binaria [7] .	21
2.10. Señal FSK binaria ideal y su descomposición en dos señales ASK [6] .	21
2.11. Espectro de una señal FSK binaria [6]	22
2.12. Respuesta al impulso del filtro Gaussiano para BT 0,3 y 0,5 [6] . . .	22
2.13. Diagrama de bloques para la generación de una señal GFSK	23
2.14. Señal GFSK temporal [8]	23
2.15. Espectro de una señal GFSK [9]	24
2.16. Diagrama de Bloques de un transmisor PSK [10]	24
2.17. Representación Temporal de una señal BPSK [11]	25
2.18. Espectros de señales 2-PSK, 4-PSK y 8PSK [10]	25
2.19. Diagrama de bloques de la generación de una señal MSK [12] . . .	26
2.20. Modulación MSK [13]	27
2.21. Espectro de una señal MSK [14]	27
2.22. Diagrama de bloques de la generación de una señal GMSK [12] . .	28
2.23. Información transmitida antes y después de pasar por el filtro gausiano en GMSK [15]	29
2.24. Espectro de una señal GMSK para distintos valores de BT [12] . . .	29
2.25. Diagrama de Bloques de un transmisor OFDM [16]	31
2.26. Espectro de una señal OFDM [16]	32
2.27. Principios de COFDM [17]	34
2.28. Diagrama de Bloques de un transmisor COFDM [17]	35
2.29. VCM en DSB-S2 [18]	36
2.30. Diagrama de Bloques de la técnica ACM [19]	37

2.31. Espectro de la señal NTSC	38
2.32. Codificador NTSC [20]	38
2.33. Creación de las señales Y, I y Q en NTSC [20]	39
2.34. Señales Correspondientes a los colores primarios (R, G y B). Señales de Crominancia (I y Q). Señal de Luminancia (Y) y Señal de Crominancia Compuesta [21]	40
2.35. Trama MPEG-2 [22]	45
4.1. Diferencia entre un transmisor tradicional y uno basado en SDR	56
4.2. Diagrama de Bloques del Generador Vectorial de Onda Arbitraria	57
4.3. Diagrama en GNU Radio Companion de la Generación de una señal DSB-SC	59
4.4. Diagrama en GNU Radio Companion de la Generación de una señal DSB-SC con fuente de Audio	60
4.5. Diagrama en GNU Radio Companion de la Generación de una señal DSB-SC con fuente de señal arbitraria	60
4.6. Espectro de una señal senoidal con modulación DSB-SC de frecuencia modulante 200 kHz y frecuencia de la señal portadora de 1.2 GHz	61
4.7. Diagrama en GNU Radio Companion de la Generación de una señal AM	62
4.8. Diagrama en GNU Radio Companion de la Generación de una señal DSB-SC con fuente de audio	62
4.9. Diagrama en GNU Radio Companion de la Generación de una señal AM con fuente de señal arbitraria	63
4.10. Modulación AM	63
4.11. Diagrama en GNU Radio Companion para la generación de Señales NBFM	65
4.12. Diagrama en GNU Radio Companion para la generación de Señales WBFM	65
4.13. Diagrama en GNU Radio Companion para la generación de Señales FM con fuente de audio	66
4.14. Diagrama en GNU Radio Companion para la generación de Señales FM con fuente arbitraria	66
4.15. Espectro de Señales NBFM para distintos valores de β	67
4.16. Espectro de Señales WBFM para distintos valores de β	68
4.17. Diagrama en GNU Radio Companion para la generación de Señales 4-ASK con fuente aleatoria	69
4.18. Diagrama en GNU Radio Companion para la generación de Señales OOK con fuente aleatoria	70
4.19. Diagrama en GNU Radio Companion para la generación de Señales ASK con fuente de archivo	70
4.20. Diagrama en GNU Radio Companion para la generación de Señales ASK con fuente de audio	71

4.21. Diagrama en GNU Radio Companion para la generación de Señales ASK con fuente UDP	71
4.22. Espectro de señales ASK con tasa de bit igual a 1 Mbps	72
4.23. Diagrama en GNU Radio Companion para la generación de Señales PSK con fuente de aleatoria	73
4.24. Diagrama en GNU Radio Companion para la generación de Señales ASK con fuente de archivo	73
4.25. Diagrama en GNU Radio Companion para la generación de Señales ASK con fuente de audio	74
4.26. Diagrama en GNU Radio Companion para la generación de Señales ASK con fuente UDP	74
4.27. Espectro de señales PSK con distintos valores de constelación, tasa de bit igual a 1 Mbps y factor de roll-off igual a 0.35	75
4.28. Espectro de señales PSK y QAM con distintos valores de constelación, tasa de bit igual a 500 kbps y factor de roll-off igual a 0.35	75
4.29. Espectro de señales 8-PSK con distintos valores de tasa de bit igual y factor de roll-off igual a 0.35	76
4.30. Diagrama en GNU Radio Companion para la generación de Señales GFSK con fuente aleatoria	77
4.31. Diagrama en GNU Radio Companion para la generación de Señales GMSK con fuente aleatoria	77
4.32. Diagrama en GNU Radio Companion para la generación de Señales GMSK con fuente de archivo	78
4.33. Diagrama en GNU Radio Companion para la generación de Señales GFSK con fuente de audio	78
4.34. Diagrama en GNU Radio Companion para la generación de Señales GMSK con fuente UDP	79
4.35. Espectro de señal GMSK para BT=0.35	79
4.36. Espectro de señal GFSK para BT=0.1	80
4.37. Espectro de señal GMSK con BT=0.9	80
4.38. Diagrama en GNU Radio Companion para la Generación de una señal OFDM	81
4.39. Espectro de señales OFDM para distintos valores de FFT, Tonos Ocupados y Prefijo Cíclico	82
4.40. Diagrama en GNU Radio Companion para la transmisión de una señal NTSC	83
4.41. Espectro de una señal NTSC a 439.25 MHz	84
4.42. Diagrama en GNU Radio Companion para la transmisión de una señal DVB-T	85
4.43. Espectro de un canal DVB-T de 8 MHz	85
4.44. Diagrama en GNU Radio Companion para la transmisión de una señal DVB-S2	86
4.45. Espectro de un canal DVB-S2 de 6 MHz	87

4.46. Desviación de Frecuencia Portadora	87
4.47. Factor de Corrección de Frecuencia Portadora	88
4.48. Potencia de Salida con Ajustes de Ganancia Constante	89
4.49. Ajustes de Ganancia IF para distintos niveles de potencia de salida a lo largo de la potencia	90
4.50. Potencia de Salida de Señal AM con distintos ajustes de ganancia IF	91
4.51. Potencia de Salida de Señal AM-SC con distintos ajustes de ganancia IF	91
4.52. Potencia de Salida de Señales FM con distintos ajustes de ganancia IF	92
4.53. Potencia de Salida de Señales GMSK y GFSK con distintos ajustes de ganancia IF	92
4.54. Potencia de Salida de Señales OOK y 4-ASK con distintos ajustes de ganancia IF	93
4.55. Potencia de Salida de Señales PSK con distintos números de constelación y ajustes de ganancia IF	93
4.56. Potencia de Salida de Señales QAM con distintos números de constelación y ajustes de ganancia IF	94
4.57. Potencia de Salida de Señales OFDM con distintos ajustes de ganancia IF	94
4.58. Potencia de Salida de Señal DVB-T para distintos ajustes de ganancia IF	96
4.59. Potencia de Salida de Señal DVB-S2 para distintos ajustes de ganancia IF	96
4.60. Potencia de Salida de Señal NTSC para distintos ajustes de ganancia IF	97
4.61. Variación de la Amplitud Analógica con Respecto a la Amplitud Digital	99
4.62. Interfaz de Inicio de la Aplicación	101
4.63. Interfaz con parámetros modificables	101
4.64. Otras Opciones en la Aplicación	102
4.65. Control Auxiliar de la Aplicación	102

Indice de tablas

2.1. Factores de cresta para distintas constelaciones de modulación QAM [23]	15
2.2. Especificaciones del Estándar DVB-T [24]	43
2.3. Especificaciones del estándar DVB-S2 [25]	44
4.1. Variación de Potencia de Canal en Señales OFDM según el Porcentaje de Tonos Ocupados y Ajuste de Ganancia IF Correspondiente	95
4.2. Ajustes de Ganancia IF con Respecto a las Señales de Referencia para Distintos Esquemas de Modulación	98
4.3. Amplitud pico-pico de salida de señal portadora con potencia de canal de -5 dBm	98
4.4. Fuentes Disponibles por Esquema	100
4.5. Especificaciones DSB-SC	104
4.6. Especificaciones AM-LC	105
4.7. Especificaciones NBFM	105
4.8. Especificaciones WBFM	105
4.9. Especificaciones GMSK y GFSK	106
4.10. Especificaciones PSK	106
4.11. Especificaciones QAM	106
4.12. Especificaciones ASK	107
4.13. Especificaciones NTSC	107
4.14. Especificaciones DVB-T	107
4.15. Especificaciones DVB-S2	108
4.16. Especificaciones OFDM	108

Acrónimos

ACM	Adaptative Coding Modulation
AM	Amplitude Modulation
ASK	Amplitude Shift Keying
DSB-SC	Double Side Band - Supressed Carrier
DVB	Digital Video Broadcasting
FEC	Forward Error Correction
GAS	Generador Arbitrario de Señales
GFSK	Gaussian Frequency Shift Keying
GMSK	Gaussian Minimun Shift Keying
GRC	GNU Radio Companion
GVRF	Generador Vectorial de Radio Frecuencia
NBFM	Narrow Band Frequency Modulation
NTSC	National Television System Committee
OFDM	Ortogonal Frequency Division Multiplexing
OOK	On Off Keying
PSK	Phase Shift Keying
QAM	Quadrature Amplitude Modulation
SDR	Software Defined Radio
UC	Universiad de Carabobo
VCM	Variable Coding Modulation
WBFM	Wide Band Frequency Modulation

**DESARROLLO DE UN GENERADOR VECTORIAL DE ONDA
ARBITRARIA BASADO EN RADIO DEFINIDA POR SOFTWARE Y
SOFTWARE LIBRE**

por

JORGE DE CASTRO y KEVIN HENRIQUEZ

Presentado en el Departamento de Señales y Sistemas
de la Escuela de Ingeniería en Telecomunicaciones
el 2 de Julio del 2015 para optar al Título de
Ingeniero de Telecomunicaciones

RESUMEN

El uso de generadores de señales es una necesidad primordial para los institutos de educación superior en el área de telecomunicaciones. Por otro lado, el desarrollo de la Radio Definida por Software (SDR), permite sustituir dispositivos típicamente empleados con *hardware* e implementarlos en *software*, lo que constituye una herramienta de bajo costo que permite la apertura de nuevas líneas de investigación para el desarrollo de proyectos en diversas áreas de las telecomunicaciones. Pensando en esto, se llevó a cabo un proyecto dedicado al desarrollo de un Generador Vectorial de Onda Arbitraria utilizando el transceptor de bajo costo HackRF One. Utilizando el *software* GNU Radio, se sintetizaron diferentes tipos de formas

de onda y distintos esquemas de modulación analógicos y digitales como DSB-SC, AM, NBFM, WBFM, ASK, PSK, GMSK, GFSK, QAM, además de señales de televisión analógica (NTSC – imágenes fijas) y digital (DVB-T y DVB-S2). El prototipo admite distintos métodos de adquisición adicionales a las generadas en GNU Radio, como audio desde la tarjeta de sonido del computador, archivos, o mediante *streaming* utilizando *software* adicional. Además admite la importación de archivos de datos generados en *software* matemáticos como Matlab, Octave, entre otros, para sintetizar formas de señales aleatorias. Así mismo, se diseñó una interfaz gráfica que permite el control y operación del dispositivo, donde se definen distintos parámetros de operación. Cabe señalar que a pesar de que el HackRF One puede operar en el rango entre 10MHz—6GHz, el rango de operación del prototipo en frecuencia va desde 50MHz hasta 3GHz debido a que a menores frecuencias no puede ofrecerse un grado de estabilidad confiable en potencia y el rango de los equipos de medición disponibles para la calibración del mismo llegaba hasta los 3 GHz. En cuanto a la potencia, el equipo podrá operar en distintos rangos dependiendo de la frecuencia de interés. Entre 50MHz y 2.74 GHz, el dispositivo podrá operar desde -10dBm a 0dBm. Entre 2.157GHz hasta 2.749GHz el generador puede operar entre -10dBm hasta 5dBm. Finalmente, entre 2.76GHz a 3GHz, el generador está limitado a un rango de operación de -10dBm a -5dBm. Todo esto con precisión de ± 1 dBm.

Palabras Claves: Generador Vectorial de Onda Arbitraria, Radio Definida por Software, Modulación

Tutor: CARLOS MEJIAS

Profesor del Departamento de Señales y Sistemas

Escuela de Telecomunicaciones. Facultad de Ingeniería

Capítulo I

Introducción

1.1. Motivación

El mundo actual se ha visto inundado por el crecimiento cada vez más acelerado de las tecnologías de transmisión inalámbricas, las cuales se han convertido en aspectos comunes y en muchos casos necesarios para la vida cotidiana de las personas. Puesto que existen equipos de bajo costo para la transmisión y recepción inalámbrica, las tecnologías de comunicaciones móviles se han convertido en la alternativa más rentable en cuanto a conectividad y facilidad de implementación.

En virtud de lo anteriormente expuesto, se tiene que el uso de las redes inalámbricas, la telefonía móvil, la radiodifusión comercial, entre otros, requiere la utilización de distintas técnicas de modulación tanto analógicas como digitales que permitan mejorar el aprovechamiento del espectro y la protección de la información ante ruido e interferencia por mencionar algunas necesidades [6]. En este sentido, el uso de generadores de señales es necesario para el estudio de las características de las diferentes técnicas de modulación, el cual forma parte de la formación de ingenieros de Telecomunicaciones.

Actualmente, los laboratorios de Comunicaciones y de Antenas de la Escuela de Ingeniería Eléctrica de la Universidad de Carabobo, sólo cuentan con equipos para la generación de señales moduladas en los formatos analógicos tradicionales

(AM, PM y FM) hasta 1 GHz, lo que limita el estudio de señales moduladas en los formatos digitales modernos (PSK, ASK, QAM, MSK, entre otros). Así mismo, existe una carencia de equipos en los laboratorios de la Escuela de Ingeniería de Telecomunicaciones, entre ellos los generadores de señales, lo que impide la realización de las diferentes prácticas de laboratorio en los espacios destinados dentro de la Escuela.

En consecuencia a lo anteriormente descrito, la realización de las prácticas de laboratorio es limitada ya que no es posible utilizar formatos de modulación modernos, lo que permitiría un mejor entendimiento de los conocimientos teóricos dictados en los diferentes cursos de sistemas de comunicaciones. Así mismo la falta de este equipo ha limitado la realización de trabajos de investigación que requieran emplear generadores de señales para el estudio de formatos de modulación más complejos, entre otras aplicaciones. Cabe señalar que este tipo de equipo de laboratorio es costoso debido a su complejidad, a los rangos de frecuencia que maneja y los componentes necesarios para su fabricación.

Atendiendo a consideraciones como la expuesta anteriormente, surgen nuevas tecnologías como la Radio Definida por Software (SDR), la cual permite sustituir componentes típicamente empleados por hardware e implementarlos por medio de software [26], lo que representa una ventaja económica considerable y abre las puertas a nuevos proyectos en el ámbito de la electrónica y las telecomunicaciones.

En base a esto, el uso de la tecnología de SDR ha permitido el desarrollo de proyectos que representan una alternativa de bajo costo para los generadores de señales arbitrarias presentes en el mercado. En ese sentido, Hsieh, Tsai y Lin en 2003 [27] utilizaron una FPGA para crear un generador de onda arbitraria de n-canales. Posteriormente y basados en dicho proyecto, Trabes, Costa y Sosa Páez [28] mejoraron la implementación basándose en un Sintetizador Digital Directo usando de igual forma una FPGA. Sin embargo, cabe señalar que para ambos casos la frecuencia máxima del generador estaba por debajo de los 100MHz. Dentro de ese marco, Jovanović, Petrović, Pavić y Remenski [29], implementaron un Generador de pruebas RF para receptores basado en SDR utilizando un sintetizador digital controlado por un microprocesador, cuya frecuencia máxima era de 400 MHz y contaba con un

rango dinámico de 130 dB, y los esquemas de modulación de las señales que podía generar eran ASK y FSK.

Por otro lado, para el año 2000, Antonio S. Fedón, Profesor de la Escuela de Ingeniería Eléctrica de la Facultad de Ingeniería de la Universidad de Carabobo, realizó verificaciones experimentales para reducir el factor de cresta de los formatos lineales de modulación digital [30]. Basándose en los resultados de dicho proyecto, Antonio J. Fedón, utilizó distintas piezas de hardware en conjunto con el software GNU-Radio, para la construcción de un prototipo de un generador vectorial de radio frecuencias limitado a una frecuencia portadora entre 800 y 990MHz, el cual permite generar señales moduladas usando los esquemas de modulación QPSK, Pi4QPSK, OQPSK, 16QAM, 64QAM y 256QAM calibradas en amplitud y frecuencia [1].

Por consiguiente, ante la carencia de un equipo de estas características, se desarrolló un prototipo de generador vectorial para un amplio rango de frecuencia, destinado a ser utilizado por los estudiantes y profesores de la Escuela de Ingeniería de Telecomunicaciones para la realización de las diferentes prácticas de laboratorios que actualmente se dictan en los Laboratorios de Comunicaciones y Antenas de la Escuela de Ingeniería Eléctrica. En este sentido, la implementación se llevó a cabo mediante el transceptor comercial de bajo costo, «HackRF One», el cual puede operar en un rango de frecuencia de 10MHz a 6 GHz, y con una potencia de transmisión que depende del rango de frecuencia en el que opere el generador. Por otra parte, en cuanto al software, se utilizó la plataforma de software libre conocida como GNU Radio para el soporte del hardware antes mencionado. El desarrollo de un generador vectorial de amplio rango de frecuencias mediante Radio Definida por Software (SDR) representa una solución factible a la carencia de este tipo de equipo en los laboratorios de la Universidad de Carabobo, específicamente en la Escuela de Ingeniería de Telecomunicaciones debido al alto costo comercial del mismo.

Por consiguiente, con este equipo se puede generar una gran variedad de señales como AM, FM, PSK, ASK, QAM, GFSK, GMSK, OFDM, NTSC, lo que ayudará al entendimiento del contenido de asignaturas como Teoría de las Comunicaciones I, Teoría de las Comunicaciones II, Comunicaciones Inalámbricas y Radiodifusión

Sonora y Televisión. Además, el generador cuenta con distintos métodos de adquisición de datos, bien sea audio, video, imágenes, flujos de datos, incluso pudiendo adquirir varios de estos mediante streaming, la cual es una característica novedosa. Por otro lado, al implementar este dispositivo utilizando Radio Definida por Software, se logró reducir el tamaño de su contraparte implementada de forma convencional porque los componentes de hardware han sido remplazados por el software, y los componentes usados por la tarjeta son pequeños. Por lo tanto, se cuenta con un dispositivo versátil y fácil manejo.

Adicionalmente, la donación del prototipo diseñado a la Escuela de Telecomunicaciones, abre las puertas para el establecimiento de una nueva línea de investigación dedicada a la Radio Definida por Software, la cual es una tecnología novedosa en la Escuela de Telecomunicaciones, por lo que la realización de este proyecto es de relevancia debido a su carácter innovador. Por otra parte, aumentará los conocimientos que se tienen sobre esta área y despertará el interés en esta línea de investigación, además de traer beneficios a futuros estudiantes y profesores.

1.2. Objetivos

1.2.1. Objetivo General

- Implementar un generador vectorial de onda arbitraria utilizando Radio Definida por Software.

1.2.2. Objetivos Específicos

- Describir la estructura y el funcionamiento de un generador vectorial de onda arbitraria basado en SDR.
- Desarrollar los algoritmos necesarios para el control del funcionamiento del generador vectorial de onda arbitraria.
- Caracterizar el rango de operación y la potencia de salida del dispositivo para cada uno de los esquemas de modulación utilizados.

- Sintetizar el prototipo del dispositivo implementando los algoritmos desarrollados para el funcionamiento del generador.

1.3. Alcance

El proyecto estará dedicado al desarrollo y la implementación de un prototipo de generador vectorial de onda arbitraria utilizando el transceptor de bajo costo conocido como HackRF One. Por medio del software GNU Radio, en este dispositivo se podrán sintetizar diferentes tipos de señales (senoidales, triangulares, cuadradas) pudiendo además sintetizar diferentes esquemas de modulación tanto analógicos como digitales. Así mismo, se diseñará una interfaz gráfica que permita el control y la operación del dispositivo, donde se podrá definir distintos parámetros de operación (frecuencia y la potencia de salida).

Por otra parte, las pruebas de rendimiento del dispositivo contempladas en el proyecto quedarán sujetas a la disponibilidad de los instrumentos de medición en el país, considerando las colaboración que pudieran prestar los laboratorios de otras universidades e instituciones públicas y privadas que permitan el uso de dichos instrumentos de medición para la caracterización del prototipo.

Finalmente, el funcionamiento del generador de señales de onda arbitraria deberá estar conectado a un computador que se encargará de ejecutar la aplicación y proveer la energía necesaria al transceptor.

Capítulo II

Marco conceptual

2.1. Generador Vectorial de Radio Frecuencia (GVRF)

Un Generador Vectorial de Radio Frecuencia es un instrumento capaz de generar señales moduladas usando un gran número de esquemas de modulación como QAM, QPSK, FSK, OFDM, entre otros, con frecuencia, amplitud y calibradas [31]. Para ello, está compuesto de los siguientes elementos:

- Un Generador Arbitrario de Señales
- Un Oscilador Local Sintetizado
- Un Modulador en Cuadratura
- Un Amplificador de Ganancia Variable

2.2. Generador Arbitrario de Señales (GAS)

En la generación de señales digitales, los generadores digitales producen una serie de niveles de voltaje discretos, lo que posibilita la creación de un amplio rango de señales con un mismo hardware. Una señal arbitraria puede ser descrita de

forma matemática, almacenada en la memoria y luego reproducida. Este es el origen del término Generador Arbitrario de Señales (GAS). [32]

Un generador Arbitrario de Señales está formado por un convertidor Digital/Analógico, una memoria digital, un filtro pasa bajo, un bloque de control y una interfaz entrada salida como se observa en la Figura 2.1. Por lo tanto, para generar la señal arbitraria, primero se cargan los datos de la forma de onda deseada a la memoria a través de la interfaz E/S. Seguidamente se envían los datos al convertidor D/A de manera secuencial ajustando la velocidad del reloj mediante el bloque de control. Finalmente se pasa la señal por el filtro pasabajas con el fin de eliminar los espectros alias de la señal muestreada. De ser necesario la generación de varias señales simultaneas, se utilizan tantos convertidores D/A como sean necesarios. [1]

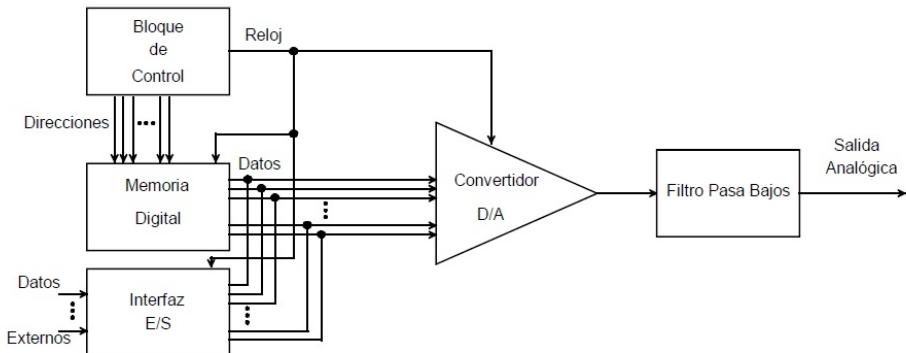


Figura 2.1: Diagrama de Bloques de un Generador Arbitrario de Señales de un Canal [1]

2.3. Radio Definida por Software (SDR)

Según la ITU, se define la SDR como: “Un radio transmisor y/o receptor empleando una tecnología que permite a los parámetros operativos RF incluyendo, pero no limitados a, rango de frecuencia, tipo de modulación, o potencia de salida,

a ser establecidos o modificados mediante software, excluyendo cambios a los parámetros de operación los cuales ocurren durante la operación normal y preinstalada de un radio de acuerdo a una especificación del sistema o estándar.” [33]

2.4. Tarjeta de desarrollo HackRF One

La HackRF One es una plataforma de desarrollo que implementa radio definida por software y funciona como un transceptor para frecuencias de 10MHz a 6GHz, que cuenta con un modulador en cuadratura de 8 bits y un amplificador en el puerto de antena. Es utilizada para experimentos que aplican programas de código abierto, desarrollo de software propio para aplicaciones de radiocomunicaciones y experimentos en la radioafición. Este dispositivo es un hardware de plataforma abierta que puede ser usado como un periférico USB o también puede ser programado para una operación de funcionamiento independiente el cual es compatible con GNU Radio. [2]

En este sentido, la HackRF One puede dividirse en dos etapas, una etapa digital y la otra de banda base y radiofrecuencia. Dentro de estas etapas se encuentran los componentes que permiten la implementación de un generador onda arbitraria.

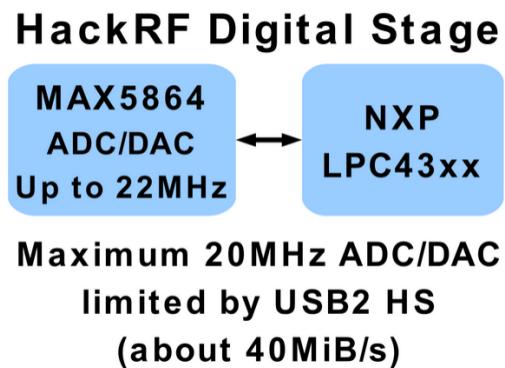


Figura 2.2: Etapa digital de la HackRF One [2]

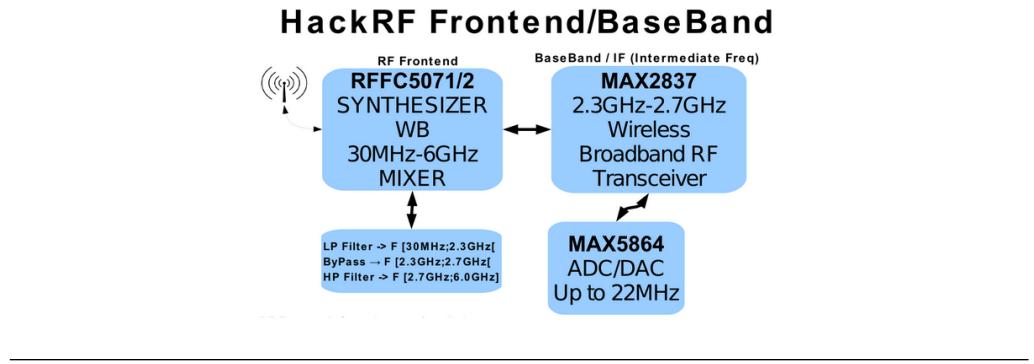


Figura 2.3: Etapa de radio de la HackRF One [2]

2.4.1. Microntrolador LPC43xx ARM Cortex-M4

Es el elemento encargado de procesar la información del equipo bien sea para transmisión o recepción, capaz de operar hasta frecuencias de CPU de 204 MHz. [34]

2.4.2. Convertidor Analógico-Digital/Digital Analógico MAX5864. Maxim Integrated

Cuenta con una frecuencia de reloj de 22 Msps. Resolución de 8 bits de conversión AD (recepción) y 10 bits de resolución DA (transmisión). [35]

2.4.3. Transceptor RF de banda ancha MAX2837. Maxim Integrated

Se encarga de la manipulación de la señal en banda base y en IF. Opera con un modulador en cuadratura y posee uno de los componentes amplificadores que ayudan a controlar la potencia del generador de onda arbitraria, ya que el amplificador permite un control de ganancia de 47 dB en pasos de 1 dB. [36]

2.4.4. Mezclador/Sintetizador RFFC5072 RFMD

Cuenta con un rango de operación de 30 MHz a 6000 MHz. Es el componente que permite trasladar las señales de banda base a RF. [37]

2.4.5. Amplificador MGA81563. Avago Technologies

Es el amplificador ubicado justo a la salida RF de la HackRF One y es el segundo elemento utilizado para controlar la potencia del generador de onda arbitraria. Cuando está encendido, ofrece una ganancia de 14 dB. [38]

2.5. GNU Radio

GNU Radio es un software libre de plataforma abierta compuesto por un conjunto de archivos y aplicaciones que proveen librerías necesarias para el procesamiento digital de señales para la manipulación e implementación de señales de radio. Puede ser usado con hardware de bajo costo para crear radios definidas por software, o sin hardware en un ambiente de simulación. Es ampliamente utilizado tanto en el ámbito académico como comercial para la investigación de sistemas de comunicación inalámbrica. [39]

2.6. Modulación

La transmisión de señales portadoras de información a través de un canal de comunicaciones pasabanda, usualmente requiere un desplazamiento del rango de frecuencias contenidas en la señal a otro rango más apropiado para la transmisión. Este desplazamiento del rango de frecuencias de la señal se conoce como modulación. La modulación se define como el proceso mediante el cual una característica de una señal portadora varía de acuerdo a una señal modulante [40]. La modulación se utiliza para transmitir señales tanto analógicas como digitales [41].

Los tipos básicos de modulación analógica son modulación de onda continua (CW) o modulación de pulso. En la modulación de onda continua, una señal senoidal se utiliza como señal portadora. Por lo tanto, una señal portadora modulada puede ser representada matemáticamente como:

$$X(t) = A(t) \cos(\omega_c t + \theta(t)) \quad (2.1)$$

Donde,

$A(t)$: Amplitud instantánea

$\theta(t)$: Ángulo de fase de la señal portadora

$\omega_c = 2\pi f_c$: Frecuencia de la señal portadora

2.6.1. Modulación en amplitud

La modulación en amplitud consiste en hacer variar la amplitud de una señal con frecuencia y fase fijas, conocida como señal portadora, en proporción a una señal dada, conocida como señal modulante, o como el mensaje o información a transmitir. Esto modifica la señal, ya que se trasladan sus componentes de frecuencia a frecuencias más altas [42][6]. Una señal modulada en amplitud puede describirse por lo general como:

$$\phi(t) = A(t) \cos(\omega_c t) \quad (2.2)$$

Donde,

$A(t)$ está linealmente relacionada con la información o el mensaje a transmitir
 $m(t)$

2.6.1.1. Portadora Suprimida (DSB-SC)

En la modulación de doble banda lateral con portadora suprimida, la amplitud de la señal modulada es directamente proporcional a la amplitud de la señal

modulante, por lo que la señal modulada queda representada como:

$$\phi_{DSB-SC}(t) = m(t) \cos(\omega_c t) \quad (2.3)$$

Este tipo de modulación se llama de portadora suprimida porque la densidad espectral de $\phi(t)$ no presenta una portadora identificable a pesar de estar centrado en ω_c . [6]

2.6.1.2. Gran Portadora (AM)

El uso de señales con portadora suprimida implica que deben emplearse circuitos complejos en los receptores capaces de mantener el sincronismo con transmisores localizados a grandes distancias. Es por esto que surge la alternativa de la modulación AM, la cual permite el uso de receptores más sencillos y menos costosos a expensas de un transmisor menos eficiente. La modulación AM consiste en añadir una portadora a la señal DSB, por lo que este tipo de señales también se conocen como señales de doble banda lateral con gran portadora (DSB-LC). [6]

Las señales AM pueden ser expresadas matemáticamente de la siguiente manera:

$$\phi_{AM}(t) = m(t) \cos(\omega_c t) + A(t) \cos(\omega_c t) = [A + m(t)] \cos(\omega_c t) \quad (2.4)$$

Hay que tomar en cuenta que la amplitud A de la portadora debe ser suficientemente grande de modo que cumpla con la condición:

$$A \geq |\min(m(t))| \quad (2.5)$$

De esta condición surge el parámetro conocido como índice de modulación el cual se define como:

$$\mu = \frac{|\min(m(t))|}{A} \quad (2.6)$$

Donde debe cumplirse que $\mu \leq 1$. En el caso de que $\mu > 1$, se dice que la señal está sobremodulada, resultando en la distorsión de la envolvente.

2.6.1.3. Modulación de Amplitud en Cuadratura (QAM)

La modulación de Amplitud en Cuadratura consiste en enviar dos señales analógicas, o dos flujos de datos binarios paralelos, las cuales modularán a dos portadoras de la misma frecuencia y fase en cuadratura, por lo general senoidales, las cuales tras ser moduladas se suman para formar la señal QAM la cual puede escribirse de manera matemática como:

$$\phi_{QAM}(t) = m_1(t) \cos(\omega_c t) + m_2(t) \sin(\omega_c t) \quad (2.7)$$

Las dos señales modulantes reciben los nombres de I y Q, la señal I modula a la portadora de referencia en fase y la señal Q a la señal portadora de referencia en cuadratura. [6][42]

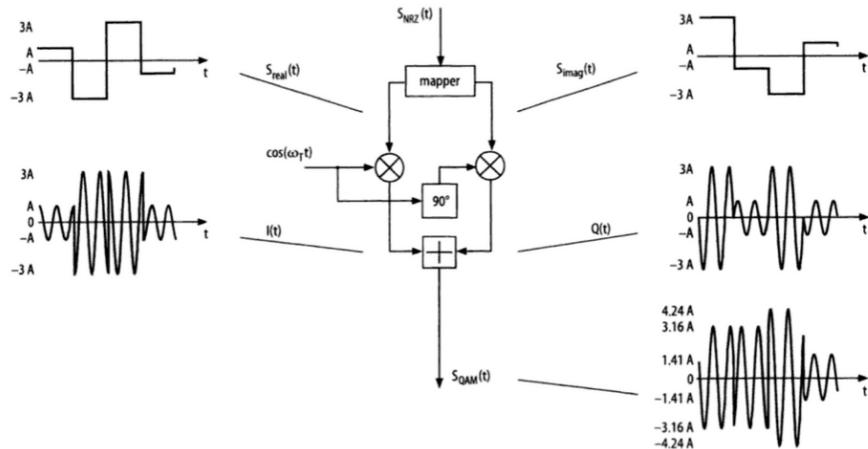


Figura 2.4: Diagrama de bloques y señal QAM temporal [3]

El ancho de banda de la señal depende de la tasa de bit y el número de bits por símbolo según la relación:[10]

$$BW_{QAM} = (1 + r) \frac{R}{N^{\circ} \text{ bits/símbolo}} \quad (2.8)$$

Donde,

r: factor de roll-off del filtro

R: Tasa de bit

Este ancho de banda está relacionado con el filtro de coseno elevado, cuya característica consiste en una magnitud plana o constante a frecuencias bajas y una porción de atenuación que tiene forma senoidal con simetría impar alrededor de la frecuencia de corte.

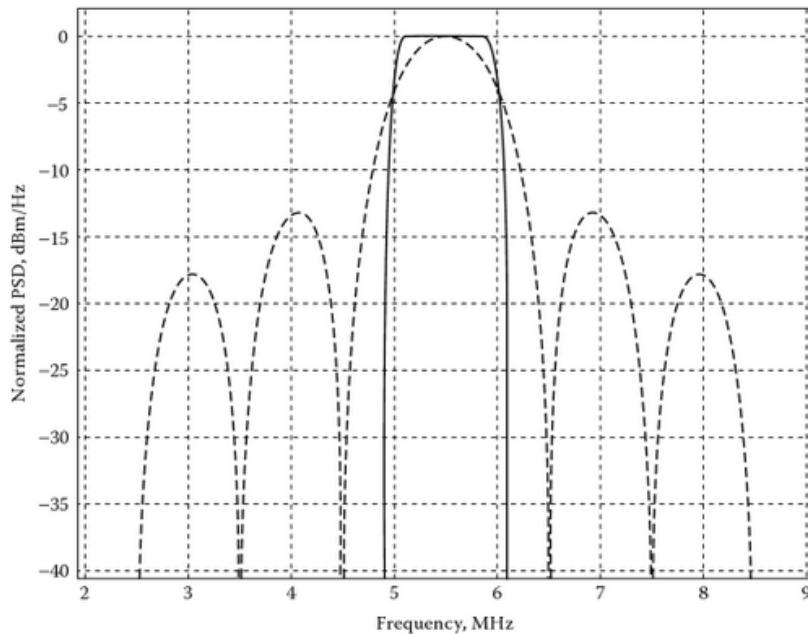


Figura 2.5: Espectro de la señal QAM — línea punteada. Forma del filtro de coseno realzado — línea sólida [4]

El factor de cresta de las señales QAM es el siguiente:

Modulación	QAM	16-QAM	64-QAM	256-QAM
Factor de Cresta	0 dB	2.6 dB	3.7 dB	4.2 dB

Tabla 2.1: Factores de cresta para distintas constelaciones de modulación QAM [23]

2.6.1.4. Modulación por Desviación de Amplitud (ASK)

Es un esquema en el que la información digital está representada en las variaciones de amplitud de una señal portadora. La amplitud de la portadora varía de acuerdo a un flujo de bits que funciona como señal modulante, manteniendo la frecuencia y fase constante. Los distintos valores de amplitud que toma la señal portadora están codificados en patrones de uno o más bits de acuerdo a la cantidad de valores de amplitud que se desee tener. Al igual que la modulación AM, la modulación ASK es susceptible al ruido y a la distorsión. [6]

El caso más simple de la modulación ASK es la conmutación de encendido-apagado (OOK, on-off keying). Representa información digital como la presencia o ausencia de una señal portadora dependiendo de si se trata de un 1 o un 0 binario respectivamente.[43]

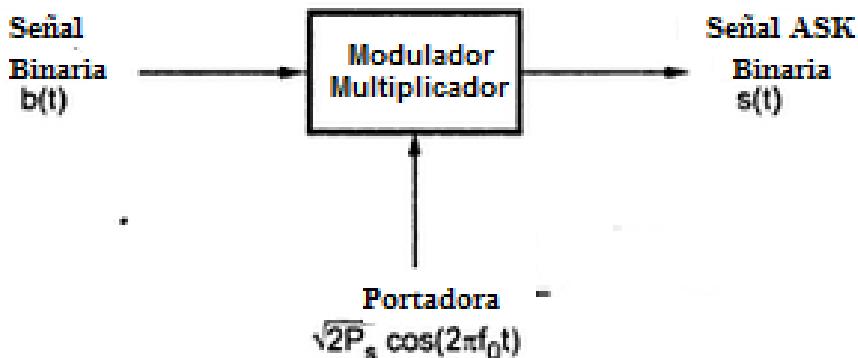


Figura 2.6: Diagrama de Bloques para la generación de una señal ASK binaria [5]

En el caso binario, la expresión de la señal transmitida viene dada por:

$$s(t) = \begin{cases} \sqrt{2P_s} \cos(2\pi f_o t) & 0 < t \leq T \\ 0 & \text{otro} \end{cases} \quad (2.9)$$

El ancho de banda requerido para una señal ASK es [10]:

$$BW_{ASK} = (1 + r) \frac{R}{N^{\circ} \text{ bits/símbolo}} \quad (2.10)$$

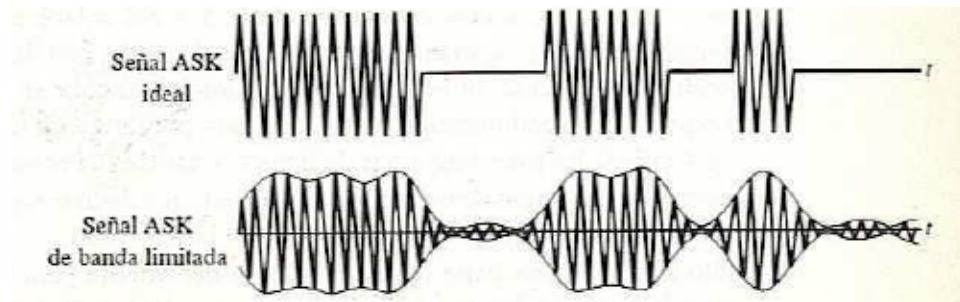


Figura 2.7: Representación temporal de una señal ASK binaria [6]

Donde,

r: factor de roll-off del filtro

R: Tasa de bit

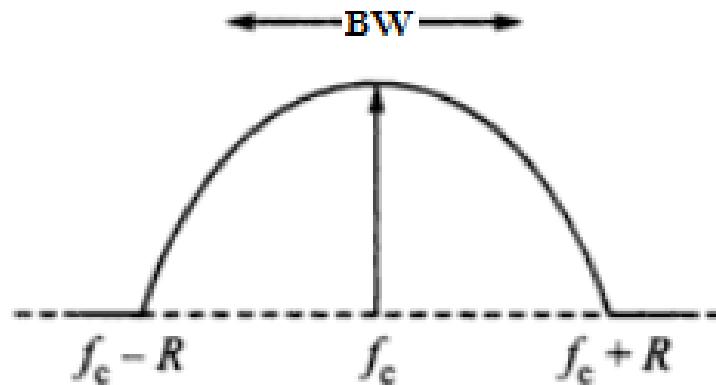


Figura 2.8: Espectro de una señal ASK binaria [5]

2.6.2. Modulación de Ángulo

En este método de modulación, la amplitud de la señal portadora permanece constante, y es el ángulo de dicha señal el que variará en frecuencia o en fase de acuerdo a una señal modulante en bandabase.

Una ventaja que ofrece la modulación de ángulo con respecto a la modulación en amplitud, es que ofrece mejor discriminación ante ruido e interferencia a expensas del ancho de banda de transmisión. [40]

2.6.2.1. Modulación en Frecuencia (FM)

Es una modulación no lineal que viene definida por la siguiente expresión:

$$\phi_{FM}(t) = A \cos \left[2\pi f_c t + 2\pi k_f \int_0^t m(t) dt \right] \quad (2.11)$$

Donde,

k_f : Constante de modulación de frecuencia.

$m(t)$: Señal modulante

Considerando el caso de que la señal modulante viene dada por una señal senoidal como la dada a continuación:

$$m(t) = A \cos(\omega_m t) \quad (2.12)$$

Al tratarse de una señal FM, la frecuencia instantánea, debe ser proporcional a la señal modulante por lo que viene dada por:

$$\omega_i = \omega_c + k_f m(t) = \omega_c + A k_f \cos(\omega_m t) \quad (2.13)$$

Se define entonces una nueva variable llamada desviación de frecuencia pico,

$$\Delta\omega = A k_f \quad (2.14)$$

Por lo que se puede reescribir la ecuación 2.13 como:

$$\omega_i = \omega_c + k_f m(t) = \omega_c + \Delta\omega \cos(\omega_m t) \quad (2.15)$$

Y por lo tanto la señal FM queda definida de la siguiente manera:

$$\phi_{FM}(t) = A \cos \left[\omega_c t + \frac{\Delta\omega}{f_m} \sin(\omega_m t) \right] = A \cos [\omega_c t + \beta \sin(\omega_m t)] \quad (2.16)$$

donde,

$$\beta = \frac{\Delta\omega}{f_m} \quad (2.17)$$

Es una relación adimensional entre la desviación de frecuencia pico y la frecuencia de la señal modulante. El parámetro β se llama índice de modulación de la señal FM y establecerá la condición para saber si se trata de una señal FM de banda angosta o de FM de banda ancha. [6]

■ FM de banda angosta (NBFM)

Por lo general, es común que se tomen valores de $\beta < 0,2$ para satisfacer la condición de banda angosta, de modo que una aproximación a la expresión matemática de una señal FM de banda angosta viene dada de la siguiente forma:

$$\phi_{NBFM}(t) = A \cos(\omega_c t) - \beta A \sin(\omega_m t) \sin(\omega_c t) \quad (2.18)$$

■ FM de banda ancha (WBFM)

En el caso de la señal FM de banda ancha, no pueden hacerse las aproximaciones de banda angosta debido a que $\beta > 1$, por lo que la señal viene dada por:

$$\phi_{FM}(t) = \operatorname{Re} \left\{ A e^{j\omega_c t} e^{j\beta \sin(\omega_m t)} \right\} \quad (2.19)$$

El segundo término exponencial de la ecuación 2.19 es una función periódica del tiempo la cual puede expandirse en una serie de Fourier,

$$e^{j\beta \sin(\omega_m t)} = \sum_{n=-\infty}^{\infty} F_n e^{jn\omega_m t} \quad (2.20)$$

donde,

$$F_n = \frac{1}{T} \int_{-T/2}^{T/2} e^{-jn\omega_m t} e^{j\beta \sin(\omega_m t)} dt \quad (2.21)$$

Haciendo un cambio de variable $\varphi = \omega_m t = (2\pi/T)t$, se tiene

$$F_n = \frac{1}{T} \int_{-\pi}^{\pi} e^{j\beta \sin(\varphi) - jn\varphi} d\varphi \quad (2.22)$$

Esta integral es una función de Bessel de primera clase y puede evaluarse numéricamente en función de los parámetros n y β .

Conociendo esto la ecuación 2.20 puede reescribirse como

$$e^{j\beta \sin(\omega_m t)} = \sum_{n=0}^{\infty} J_n(\beta) e^{jn\omega_m t} \quad (2.23)$$

Por lo que la expresión de la señal FM de banda ancha viene dada por:

$$\phi_{WBFM}(t) = \operatorname{Re} \left\{ A e^{j\omega_c t} \sum_{n=-\infty}^{\infty} J_n(\beta) e^{jn\omega_m t} \right\} = A \sum_{n=-\infty}^{\infty} J_n(\beta) \cos(\omega_c t + n\omega_m t) \quad (2.24)$$

2.6.2.2. Modulación por Desviación de Frecuencia (FSK)

También conocida como conmutación de frecuencia, es un tipo de modulación digital en el que la información es enviada cambiando la frecuencia de la señal portadora. El caso más común es la conmutación de frecuencia binaria BFSK, donde solo se utilizan dos frecuencias. Sin embargo, pueden usarse más de dos frecuencias en dichos casos se utilizan 2 o más bits de información por símbolo.

Se puede considerar a la señal FSK como si estuviera compuesta por señales ASK con frecuencias portadoras distintas. Tiene una tasa de error menor a ASK a expensas de un mayor ancho de banda. [10]

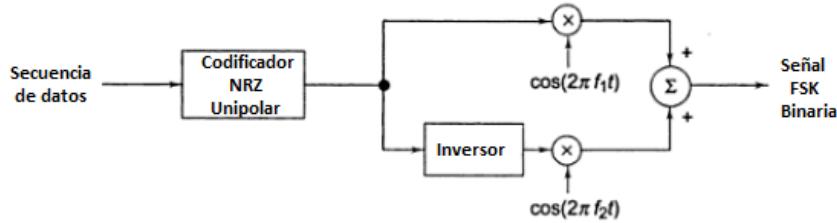


Figura 2.9: Diagrama de Bloques de la generación de una señal FSK binaria [7]

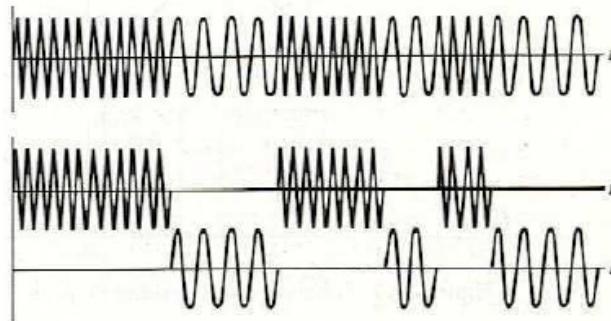


Figura 2.10: Señal FSK binaria ideal y su descomposición en dos señales ASK [6]

La expresión de una señal M-aria FSK es la siguiente[7]:

$$S_{FSK}(t) = A \cos(2\pi f_c t + 2\pi \Delta f a_n t + \phi_o) \quad nT_s \leq t \leq (n+1)T_s \quad (2.25)$$

Donde:

Δf : Espaciamiento entre las frecuencias

a_n : Valor de cualquier de los M símbolos posibles

T_s : Período del símbolo

El ancho de banda de la señal FSK binaria es el siguiente [10]:

$$BW_{FSK} = \Delta f + (1+r)R \quad (2.26)$$

Al ser una señal de envolvente constante, su factor de cresta es de 0 dB. [44][45]

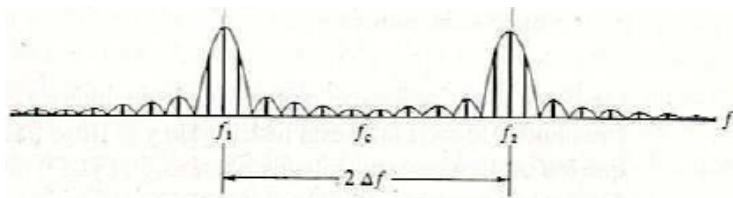


Figura 2.11: Espectro de una señal FSK binaria [6]

2.6.2.3. Modulación por desviación de Frecuencia Gaussiana (GFSK)

Es un esquema de modulación diseñado para disminuir las desventajas de FSK. En la conmutación de frecuencia, cuando se produce un cambio de un 1 binario a un 0, se traduce en una discontinuidad en la salida del modulador y es por esto que una de las grandes desventajas de FSK es su ineficiencia espectral. GFSK utiliza un filtro de respuesta gaussiano para suavizar las transiciones de un símbolo a otro y de esta manera reducir el ancho de banda de la señal; este proceso de hacer pasar la señal a través de un filtro con el fin de adaptarla a las necesidades de un canal de transmisión recibe el nombre de modelación de pulso[46][8][47]. La respuesta del filtro gaussiano es:

$$g(t) = \frac{1}{\sqrt{2\pi}\sigma T} e^{\frac{-t^2}{2\sigma^2 T^2}} \quad (2.27)$$

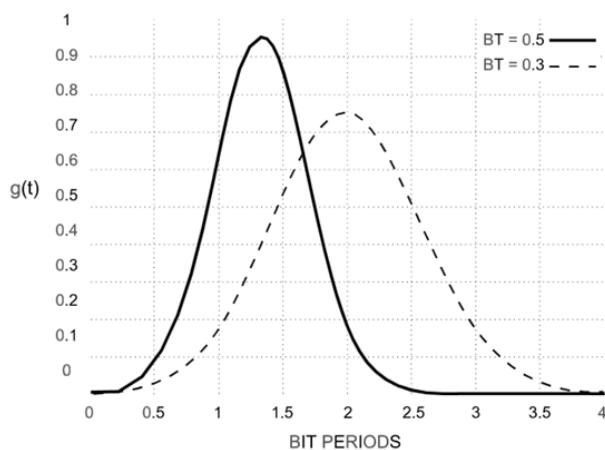


Figura 2.12: Respuesta al impulso del filtro Gaussiano para BT 0,3 y 0,5 [6]

La expresión para señal GFSK se deriva de convolucionar las señales $S_{FSK}(t)$ y $g(t)$:

$$S_{GFSK}(t) = s_{FSK}(t) * g(t) = \int_{-\infty}^{\infty} s(\tau) g(t - \tau) d\tau \quad (2.28)$$

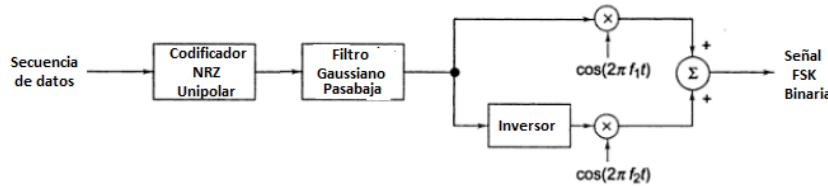


Figura 2.13: Diagrama de bloques para la generación de una señal GFSK

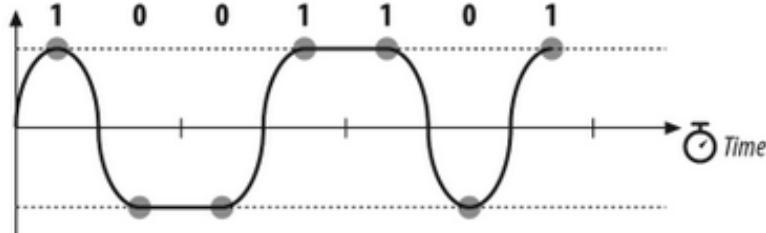


Figura 2.14: Señal GFSK temporal [8]

El factor σ de la respuesta al impulso del filtro gaussiano se relaciona con el ancho de banda del filtro de la forma: [48]

$$\sigma = \frac{\sqrt{\ln 2}}{2\pi B T} \quad (2.29)$$

Siendo B el ancho de banda de -3dB y T el período del símbolo.

Al ser una señal de envolvente constante, su factor de cresta es de 0 dB. [44][45]

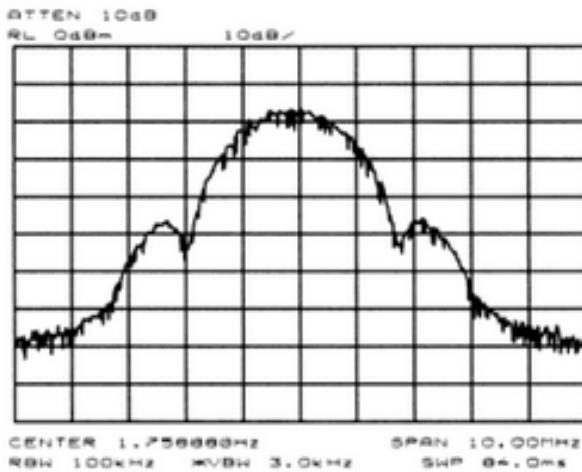


Figura 2.15: Espectro de una señal GFSK [9]

2.6.2.4. Modulación por Desviación de Fase (PSK)

Es un esquema de modulación digital en el que la información modula la fase de una señal de referencia o portadora. PSK utiliza un número finito de fases, cada una asignada a un patrón de dígitos binarios único. Usualmente cada fase codifica un mismo número de bits y cada uno de estos patrones es representado por una fase particular. [43]

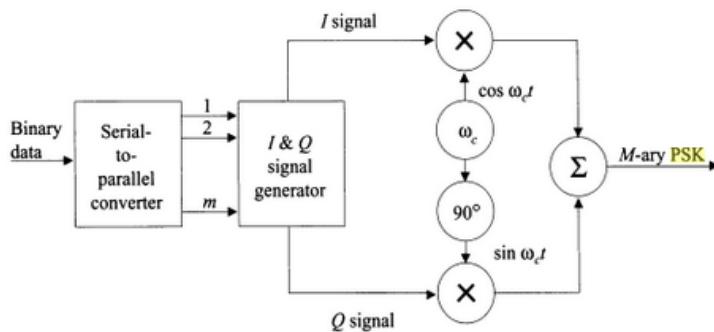


Figura 2.16: Diagrama de Bloques de un transmisor PSK [10]

La señal PSK viene expresada de la forma:

$$x_{\text{PSK}}(t) = A d(t) \cos(\omega_c t) \quad (2.30)$$

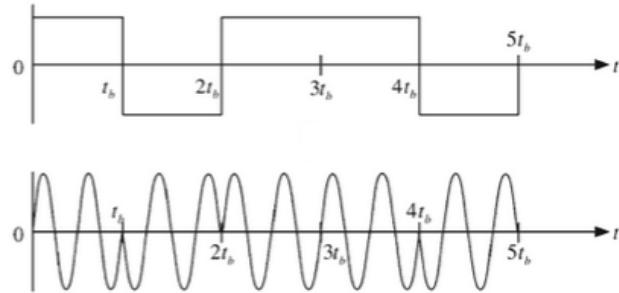


Figura 2.17: Representación Temporal de una señal BPSK [11]

El ancho de banda requerido para una señal PSK es[10]:

$$\text{BW}_{\text{PSK}} = (1 + r) \frac{R}{N^{\circ} \text{ bits/símbolo}} \quad (2.31)$$

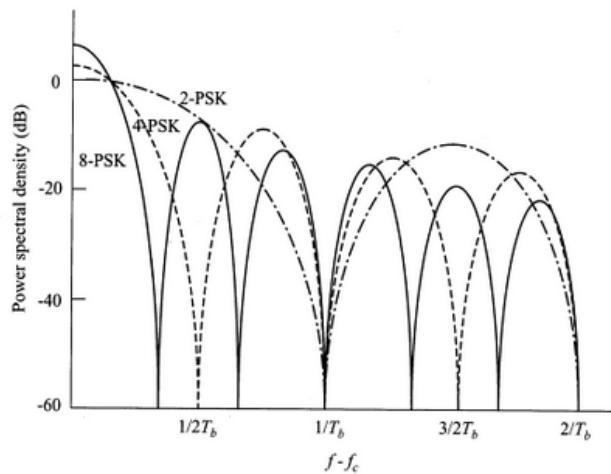


Figura 2.18: Espectros de señales 2-PSK, 4-PSK y 8PSK [10]

El factor de cresta de una señal 8-PSK, típicamente usada en la tecnología EDGE, es de 3.2 dB.[49]

2.6.2.5. Modulación por Desviación Mínima (MSK)

El ancho de banda en las señales FSK depende de la diferencia entre dos frecuencias utilizadas para distintos símbolos. La menor diferencia posible entre dos tonos senoidales para el caso más sencillo de la modulación por desviación de frecuencia, es decir, BFSK, es $1/2T_b$, donde T_b es el tiempo de bit.

En la modulación digital, MSK es un tipo de modulación por desviación de frecuencia de fase continua, la cual utiliza la mínima diferencia entre las frecuencias de la señal, y la codificación se realiza con bits alternando componentes en cuadra-tura. En este sentido, cada bit es codificado como una media senoidal, lo que resulta en una señal de módulo constante, reduciendo así los problemas causados por la no linealidad. [50][13]

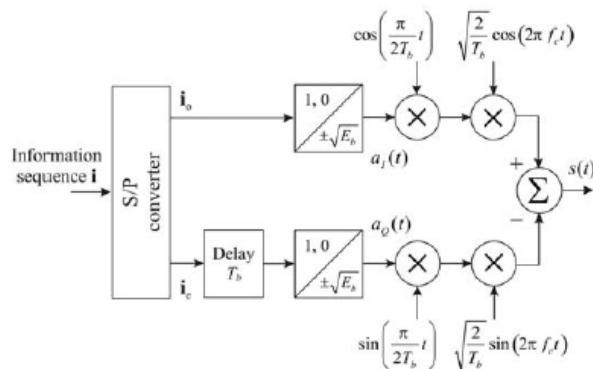


Figura 2.19: Diagrama de bloques de la generación de una señal MSK [12]

La expresión temporal de la señal GMSK es la siguiente[22]:

$$s_{MSK}(t) = A \cos [\omega_c t + \pi b_i t / 2T_b + \varphi_i] \quad (2.32)$$

donde,

$b_i = \pm 1$, representan los dígitos binarios 1 y 0.

$\varphi_i = 0$ ó π como sea necesario para mantener la continuidad de fase en $t = KT_b$

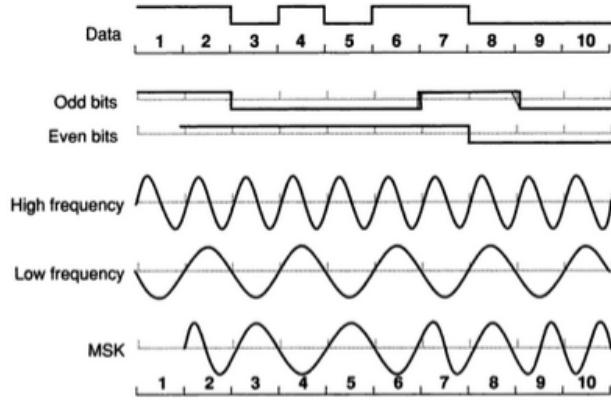


Figura 2.20: Modulación MSK [13]

Debido a que el ancho de banda para una señal FSK depende de Δf , y que para la condición de fase continua de MSK $\Delta f = 1/2T = R/2$, se tiene que el ancho de banda para GMSK es:

$$BW_{MSK} = \frac{R}{2} + (1+r)R \quad (2.33)$$

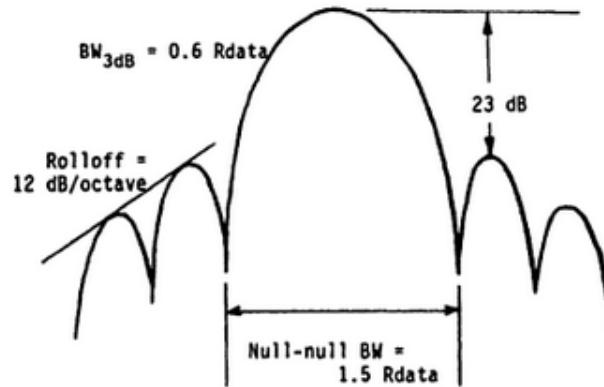


Figura 2.21: Espectro de una señal MSK [14]

El factor de cresta de las señales MSK es de 0 dB debido a que son señales de envolvente constante. [44][45]

2.6.2.6. Modulación por Desviación Mínima Gaussiana (GMSK)

A pesar de que en MSK cerca del 99 % de la energía está contenida en el lóbulo principal del espectro, y que los lóbulos secundarios son pequeños en comparación, estos se extienden resultando en una señal de gran ancho de banda. GMSK consiste en pasar la señal digital modulante a través de un filtro gaussiano con el fin de modelar los pulsos de modo que el resultado la modulación sea una señal con lóbulos laterales reducidos, es decir, con mejor eficiencia espectral, pero con reducción en su rendimiento debido a la interferencia intersímbolo introducida en el proceso de filtrado gaussiano. [12][51]

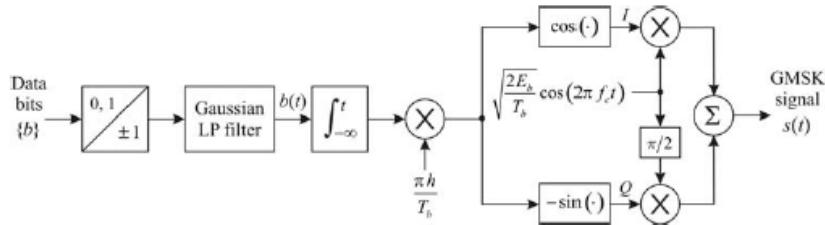


Figura 2.22: Diagrama de bloques de la generación de una señal GMSK [12]

Al igual que con GFSK, la respuesta del filtro gaussiano viene dada por la ecuación 2.27.

La expresión para señal GMSK se deriva de convolucionar las señales de las ecuaciones 2.33 y 2.27: [12]

$$s_{\text{GMSK}}(t) = s_{\text{MSK}}(t) * g(t) = \int_{-\infty}^{\infty} s(\tau) g(t - \tau) d\tau \quad (2.34)$$

$$s_{\text{GMSK}}(t) = \sqrt{\frac{2E_b}{T_b}} \left\{ \cos(\omega_c t) \cos \left[\frac{\pi h}{T_b} \int_{-\infty}^t b(u) du - \sin(\omega_c t) \int_{-\infty}^t b(u) du \right] \right\} \quad (2.35)$$

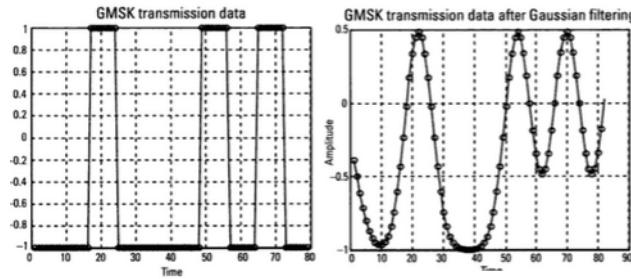


Figura 2.23: Información transmitida antes y después de pasar por el filtro gausiano en GMSK [15]

El ancho de banda de -3dB la señal GMSK se relaciona con la tasa de bit (R) y el factor BT de la forma: [48]

$$BT = \frac{BW_{-3dB}}{R} \quad (2.36)$$

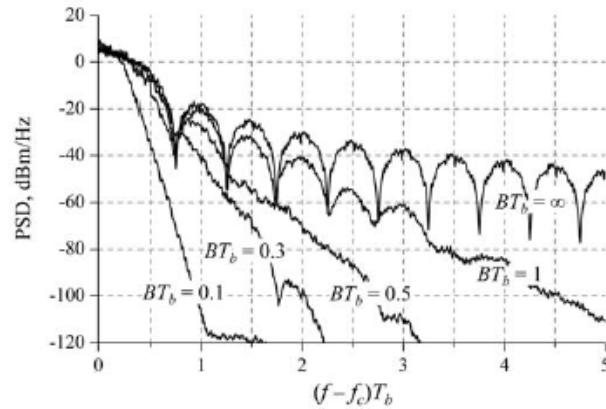


Figura 2.24: Espectro de una señal GMSK para distintos valores de BT [12]

El factor de cresta de una señal GMSK es de 0 dB. [44][45]

2.7. Multiplexación por División de Frecuencias Ortogonales (OFDM)

La idea general de la técnica de transmisión OFDM es dividir el ancho de banda total disponible en distintos sub-canales de banda angosta en frecuencias equidistantes. El espectro de los subcanales se superpondrá los unos con los otros pero las señales sub-portadoras serán ortogonales. Cada sub-canal es modulado de forma individual utilizando modulación digital y serán transmitidos de manera simultánea de forma paralela y superpuesta. Una señal OFDM entonces consiste de N sub-portadoras adyacentes y ortogonales separadas Δf en el eje de la frecuencia. Todas las señales subportadoras son naturalmente ortogonales entre la duración de símbolo de longitud T_S , si la distancia de la subportadora y la duración de símbolo se escogen de manera que $T_S = 1/\Delta f$. La k -ésima señal sub-portadora no modulada es descrita analíticamente por una función exponencial de valor complejo con frecuencia de portadora $k\Delta f$, $\bar{g}_k(t) = k = 0, \dots, N - 1$

$$\bar{g}_k(t) = \begin{cases} e^{j2\pi k \Delta f t} & \forall t \in [0, T_S] \\ 0 & \forall t \notin [0, T_S] \end{cases} \quad (2.37)$$

Debido a que el ancho de banda B es dividido en N sub-canales de banda angosta, la duración de símbolo OFDM T_S es N veces mayor que en el caso de sistemas alternativos de sub-portadoras que cubren el mismo ancho de banda B . Típicamente, para un ancho de banda dado, el número de sub-portadoras se elige de manera que la duración de símbolo T_S sea lo suficientemente grande comparada con el retraso multi-camino del canal de radio.

La duración T_S de la señal sub-portadora es extendida adicionalmente por un prefijo cíclico (llamado también intervalo de guarda), el cual es mayor al máximo retraso multi-canal para evitar la interferencia intersímbolo (ISI) completamente, lo que puede ocurrir en canales multicamino en el intervalo de transición entre dos

símbolos OFDM adyacentes.

$$\bar{g}_k(t) = \begin{cases} e^{j2\pi k \Delta f t} & \forall t \in [-T_G, T_S] \\ 0 & \forall t \notin [T_G, T_S] \end{cases} \quad (2.38)$$

El intervalo de guarda es directamente removido en el receptor luego de un proceso de sincronización. Desde este punto de vista, el intervalo de guarda es puramente una cabecera de sistema. La duración total del símbolo OFDM es por lo tanto $[T = T_G + T_S]$. Es una ventaja importante que la técnica de transmisión OFDM permite evitar completamente la ISI, o que pueda ser reducida considerablemente por una correcta selección de parámetros del sistema. [16]

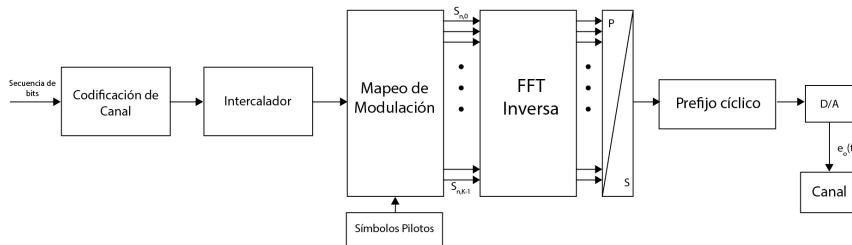


Figura 2.25: Diagrama de Bloques de un transmisor OFDM [16]

En la Figura 2.25 se detalla el diagrama de bloques de un transmisor OFDM, en el que el flujo de datos después de pasar por una codificación de canal y un proceso de intercalación es convertida de serie a paralelo y modulada en N sub-portadoras de acuerdo al tamaño de la FFT. A la entrada del bloque de la IFFT llegan N puntos de constelación $\{S_{n,k}\}$, donde N es el tamaño de la IFFT (n se refiere al intervalo de tiempo y k al índice de la subportadora en símbolo OFDM considerado). Las N muestras a la salida del bloque IFFT forman la señal bandabase transportando la información de símbolos de N sub-portadoras ortogonales. Estas luego pasan por un convertidor paralelo-serie, y de allí se le agrega el intervalo de guarda o prefijo cíclico que como se dijo, funcionará como cabecera para el sistema.

En el transmisor cada señal sub-portadora es modulada de manera independiente por lo que dentro del tiempo de duración del símbolo T_S , la señal en tiempo

continuo del k-ésimo símbolo OFDM es formada por la superposición de todas las N sub-portadoras moduladas simultáneamente: [16]

$$s_n(t) = \sum_{k=0}^{N-1} S_{n,k} g_k(t - nT) \quad (2.39)$$

El espectro típico de OFDM consiste de N funciones de seno cardinal adyacentes las cuales están desplazadas Δf una de la otra, debido que se utiliza una modelación de pulso rectangular para cada sub-portadora. El ancho de banda de la señal es entonces $B = N\Delta f/FFT$, donde N es el número de sub-portadoras. [16]

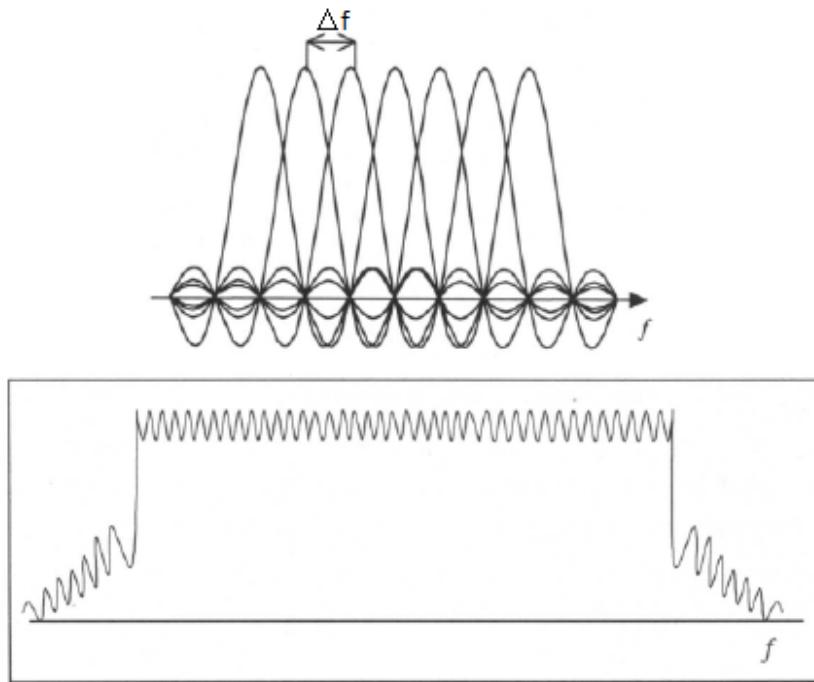


Figura 2.26: Espectro de una señal OFDM [16]

La densidad espectral de potencia de una señal OFDM puede obtenerse a partir de la ecuación 2.39, ya que esta corresponde con la modulación de portadoras ortogonales por señales con pulsos de forma rectangular. Debido a que estos pulsos tienen una duración T_s , la densidad espectral de potencia para una portadora modulada a frecuencia f_o es:

$$S(f) = E\{|s_i|^2\} T_S \operatorname{sinc}^2[(f - f_o) T_S] \quad (2.40)$$

Donde $E\{|s_i|\}$ es la potencia promedio en la secuencia de símbolos $\{s_i\}$. Y debido a que el espaciamiento entre portadoras es $T_S = 1/\Delta f$, entonces la densidad espectral de potencia de la señal OFDM viene dada por: [12]

$$S(f) = E\{|s_i|^2\} T_S \sum_{i=0}^{N-1} \operatorname{sinc}^2[(f - f_o - i/T_S) T_S] \quad (2.41)$$

Las señales OFDM exhiben altos factores de cresta debido a que el que existan fases independientes de las sub-portadoras significa que ocasionalmente éstas se combinarán de forma constructiva. El factor de cresta CF (en dB) para un sistema OFDM con N sub-portadoras no correlacionadas es:

$$CF = 10\log(N) + CF_c \quad (2.42)$$

Donde CF_c es el factor de cresta (en dB) de cada sub-portadora. [52]

2.7.1. Multiplexación por División de Frecuencias Ortogonales Codificada (COFDM)

Es una variante de OFDM especialmente diseñada para canales de transmisión terrestres que presentan múltiples ecos. La información a transmitir cuenta con protección ante errores (codificación), y es distribuida a través de las distintas subportadoras. COFDM produce símbolos de mayor duración debido a que a cada símbolo se le agrega una cabecera para protegerlos ante errores. [53]

La técnica COFDM es particularmente adecuada para proveer una recepción confiable de señales afectadas por fuertes distorsiones. La propagación multcamino es conocida por limitar el rendimiento de esquemas de modulación con altas

tasas de bit. El principio de COFDM consiste en dividir la información a ser transmitida en un gran número de portadoras, de manera que la velocidad de señalización en cada una de ellas sea significativamente menor que el ancho de banda de coherencia asumida. En otras palabras, la señal está condicionada para asegurar que los símbolos modulados serán más largos que la dispersión de retardo de eco. Proveyendo que un intervalo de guarda sea insertado entre símbolos consecutivos, la propagación multicamino no genera interferencia intersímbolo.

Sin embargo, en la presencia de ecos fuertes, algunas portadoras sufrirán desvanecimientos profundos debido a la combinación de varias reflexiones, mientras que otras serán mejoradas por adición constructiva. La relación señal a ruido en el receptor aumenta tan pronto como el poder de la señal es aumentado por ecos separados por al menos un retardo igual al inverso del ancho de banda de la señal. Para beneficiarse de este aumento de potencia, a pesar del hecho de que partes del espectro están profundamente desvanecidas, es necesario incorporar un poderoso esquema de codificación de canal en el diseño del sistema. [17]

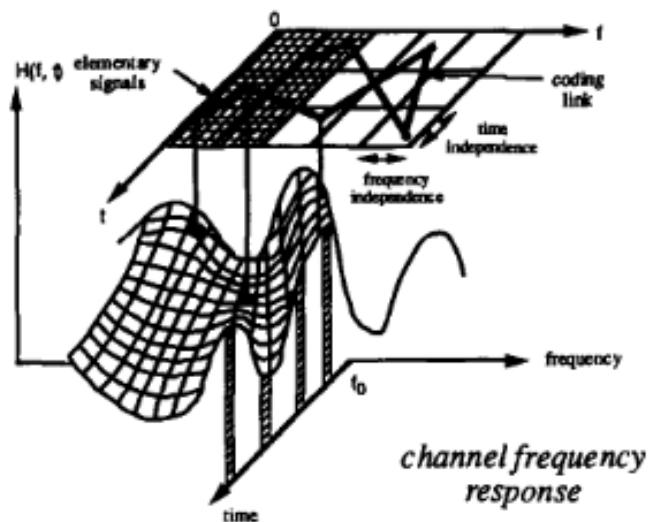


Figura 2.27: Principios de COFDM [17]

El rol de la codificación, en conjunción con la intercalación de frecuencia y tiempo, es proveer un enlace entre los bits transmitidos en portadoras separadas en el espectro de la señal de manera que la información transportada por portadoras

desvanecidas, puedan ser reconstruidas en el receptor gracias a los enlaces de codificación que lo relacionan con la información transportada por portadoras bien recibidas. La codificación e intercalación aplicada a OFDM puede ser vista como una herramienta para promediar desvanecimientos locales a lo largo del ancho de banda de la señal y sobre la profundidad de intercalación de tiempo. La selectividad de frecuencia, que solía ser una desventaja pasa a ser una ventaja llamada diversidad de frecuencia. Esto es la clave permitir la operación de Redes de Frecuencia Única (SNF).[17]

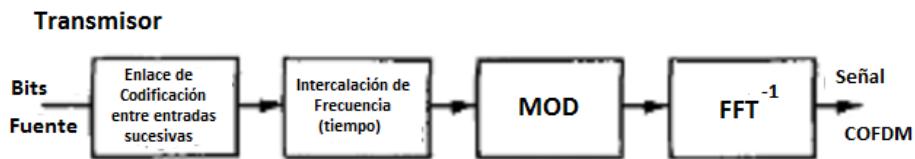


Figura 2.28: Diagrama de Bloques de un transmisor COFDM [17]

2.8. Técnicas de Modulación y Codificación

El desarrollo de la televisión digital hizo necesario la creación de técnicas de compensación ante los cambios en los enlaces satelitales, bien fuese por condiciones climáticas (pérdidas por desvanecimiento), o cambios en el ambiente RF (interferencias, etc). [19]

2.8.1. Variable Coding and Modulation (VCM)

Consiste en una técnica la cual puede ser aplicada para dar niveles de protección distintivos a diferentes servicios (ej: protección robusta para SDTV y menos robusta para HDTV, audio y multimedia). Utilizada en el estándar de televisión digital DVB-S2 el cual soporta la transmisión de diferentes servicios en la misma portadora, cada uno operando con su propio esquema de modulación y tasa de codificación. VCM realiza un tipo de operación de multiplexación en la capa física

para proveer a cada servicio con características distintas de codificación/modulación (CM). Típicamente estas CM varían entre QPSK 4/5 y 16-APSK 2/3. Las distintas tasas de bit varían entre 2.38 y 3.96 Mbps de acuerdo con la CM seleccionada [18] [54]

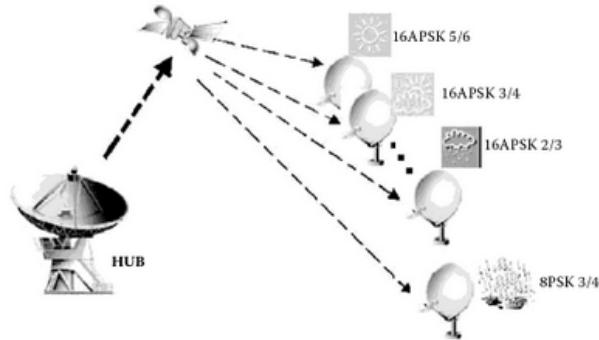


Figura 2.29: VCM en DSB-S2 [18]

2.8.1.1. Adaptive Coding and Modulation (ACM)

Es una técnica que forma parte de un esquema adaptativo donde los parámetros de transmisión, como la modulación, FEC, y potencia, son ajustados basados en la información del estado del canal. Este concepto requiere que el nodo de destino envíe la información de la señal recibida al nodo fuente por medio de un canal de retroalimentación; de esa forma el nodo fuente controla su potencia de transmisión, FEC y/o modulación de acuerdo a la información recibida. Este esquema tiene la capacidad de incrementar significativamente las velocidades de transmisión, la eficiencia espectral y la capacidad del sistema. Cuando la tasa de error en el receptor aumenta debido a señales interferidas o atenuadas, el receptor envía esta información al transmisor a través del canal de retroalimentación, para que el mismo cambie a una técnica AMC más robusta aunque menos eficiente. [19]

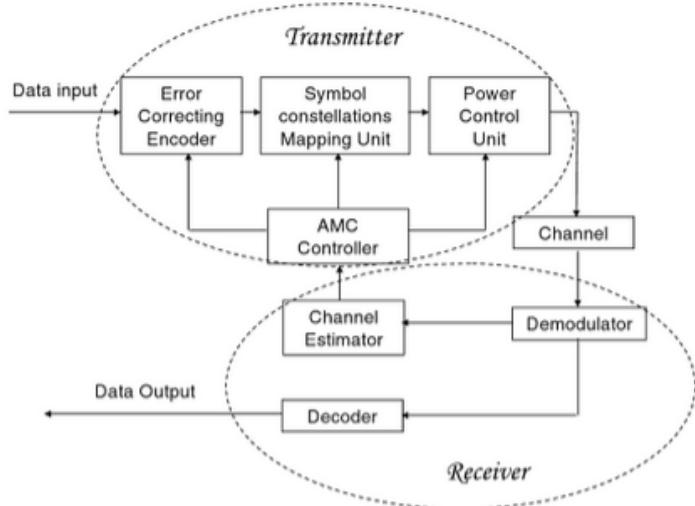


Figura 2.30: Diagrama de Bloques de la técnica ACM [19]

2.9. Estándares Internacionales de Transmisión de Televisión

2.9.1. NTSC

Derivado de las siglas de National Television System Committee, es un estándar de televisión analógico usado en gran parte de América y Japón. El estándar NTSC utiliza un sistema de codificación luminancia-crominancia. La luminancia está representada por la señal de televisión monocromática. Los colores representan la luminancia que es usada como señal de banda base. Las señales de crominancia donde se encuentra codificado el color reciben el nombre de señales I y Q. La señal I tiene un ancho de banda de, aproximadamente, 1.5 MHz y contiene los matices de color en el rango del naranja al cian. La señal Q, a la que corresponden los colores en el rango del verde al magenta, tiene un ancho de banda de alrededor de 0.6 MHz. Estas señales modulan en amplitud con portadora suprimida a una a sub-portadora de 3.58 MHz. El resultado puede verse como senoidal con fase variable respecto a una portadora de referencia y con amplitud variable. La información del matiz corresponde a la fase instantánea mientras que la saturación del color está relacionada con la amplitud instantánea. El ancho de banda de luminancia en NTSC

es de 4.2 MHz y como resultado de esto, la banda lateral superior de la señal I queda recortada ya que $1.5 + 3.58 = 5.08$ MHz, sin embargo, la banda lateral inferior I y ambas bandas laterales Q se transmiten de forma íntegra. Los anchos de banda de las bandas laterales están diseñados de modo que el espectro disponible para el color sea utilizado de la mejor manera para obtener la resolución deseada. La portadora de audio se modula en frecuencia con una desviación máxima de ± 25 kHz con preacentuación de acuerdo con una constante de tiempo de 75 μ s y se sitúa 4,5 MHz por encima de la de video. La potencia efectiva radiada del transmisor de señal de sonido no será menor que el 50 % ni mayor que el 70 % de la potencia de cresta del transmisor de señal de imagen. [20][21]

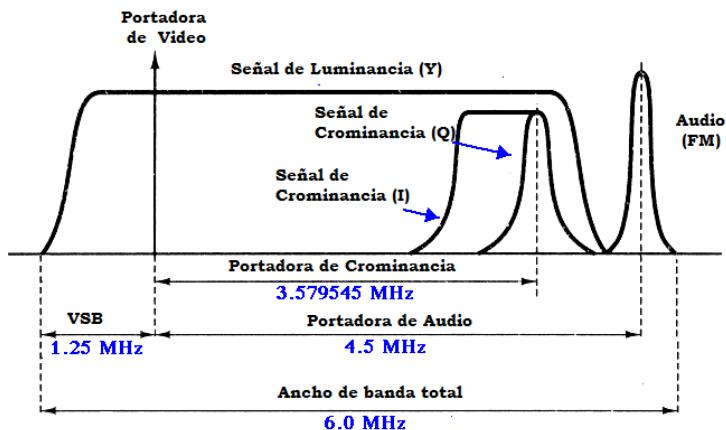


Figura 2.31: Espectro de la señal NTSC

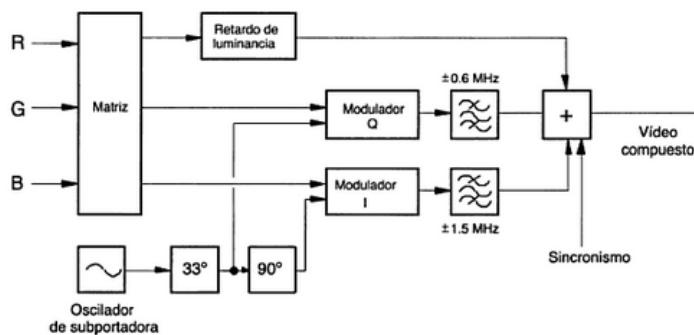


Figura 2.32: Codificador NTSC [20]

Las entradas al codificador NTSC , son las 3 señales correspondientes a los colores primarios R, G y B que se combinan en una matriz con circuitos resistivos y amplificadores inversores (ver figura 2.33) para los necesarios cambios de signo con el fin de generar a la salida la señal de luminancia 'Y' y las dos de crominancia I y Q.

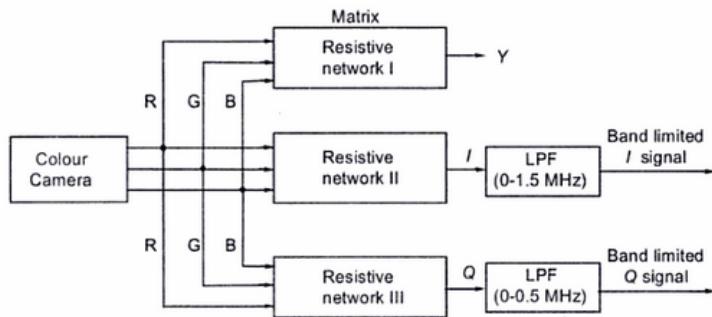


Figura 2.33: Creación de las señales Y, I y Q en NTSC [20]

Las señales I y Q son bipolares y como ya se mencionó son las encargadas de modular en amplitud a la sub-portadora de color mediante dos moduladores balanceados, a los que en el correspondiente a la señal Q se aplica la sub-portadora defasada 33° y al de la señal I se aplica la sub-portadora defasada otros 90° (ver Figura 2.32). Las señales de crominancia se suman para producir una señal de crominancia compuesta, la cual se suma a la señal de luminancia y a las señales de sincronismo y borrado para obtener la señal completa de video compuesto de color.

2.9.2. DVB

De las siglas en inglés 'Digital Video Broadcasting', es una alianza de alrededor de 200 compañías. Su objetivo es acordar las especificaciones para los sistemas de entrega de medios digitales, incluyendo su transmisión. Es una iniciativa abierta del sector privado con una cuota de membresía anual, gobernado por un Memorando de Entendimiento (MoU por sus siglas en inglés) [55]. La televisión digital se refiere a la transmisión de audio y video digitalizado, y de información auxiliar

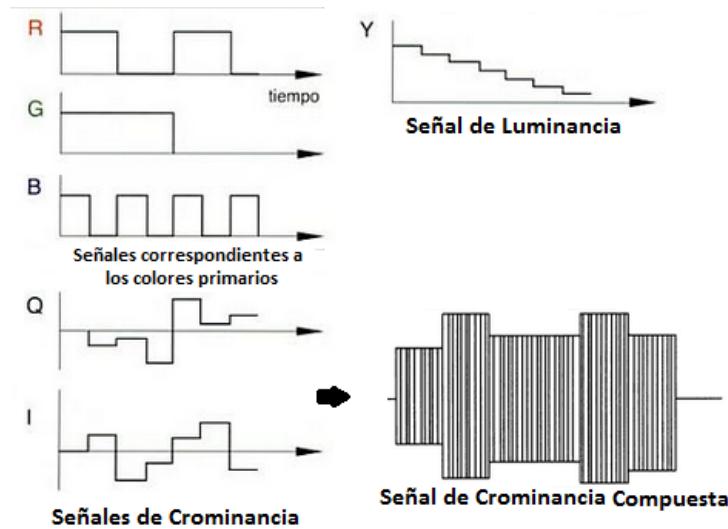


Figura 2.34: Señales Correspondientes a los colores primarios (R, G y B). Señales de Crominancia (I y Q). Señal de Luminancia (Y) y Señal de Crominancia Compuesta [21]

como señales de datos. Sin embargo, como estas señales de datos por lo general deben ser moduladas en ondas portadoras de tiempo continuo para adecuarse a los canales de transmisión, la televisión digital como tal utiliza señales analógicas. [3]

Cuando surgió en 1993 los principios y filosofía del proyecto DVB eran los siguientes: [55]

- Desarrollar una suite completa de tecnologías de transmisión satelital, terrestre y por cable en un ente de 'pre-estandarización'.
- En lugar de tener una correspondencia uno-a-uno entre un canal de entrega y un canal de programa, los sistemas serían 'contenedores' los cuales llevarían cualquier combinación de imagen, audio o multimedia. De esa forma estarían abiertos y listos para SDTV, EDTV, HDTV, sonido surround, o cualquier nuevo tipo de media que surgiera en el tiempo.
- El trabajo resultaría en estándares ETSI (European Telecommunications Standards Institute) para las capas físicas, corrección de errores, y transporte para cada medio de entrega.

- También debía resultar en un reporte ETSI el cual describiría los sistemas ban-dabase que son opciones para el transporte.
- Cuando fuese posible debería de haber una communalidad a través de las dis-tintas plataformas para disminuir costos para usuarios y fabricantes. Solo cuando no hubiese otras opciones habría diferencias entre distintos medios de entrega.
- El Proyecto DVB no debe re-inventar nada, y usaría estándares cuando estos estuviesen disponibles. El transporte para todos los sistemas es el flujo de transporte MPEG2.

Cada estándar DVB inicia su vida en el Módulo Comercial. Basado en las necesi-dades del mercado, el Módulo Comercial y sus grupos de trabajo agrupan un set de requerimientos comerciales los cuales describen los parámetros del mercado como funciones de usuario, escalas de tiempo y rangos de precio. Una vez que se ha lle-gado a un consenso sobre los requerimientos comerciales en el Módulo Comercial, se envían al Módulo Técnico.

Una especificación DVB es desarrollada en el Módulo Técnico y sus grupos de trabajo. Aquí las implicaciones tecnológicas para los requerimientos de usuario son examinadas y las tecnologías disponibles son exploradas. Una vez que el Módulo Técnico alcanza un consenso en la especificación resultante, y el apoyo del Módulo Comercial ha sido asegurado, la especificación se pasa a la Junta Directiva.

La Junta Directiva da la aprobación final para la especificación. Es entonces ofre-cida para estandarización a los entes internacionales relevantes (ETSI o CENELEC). Es así como los estándares son desarrollados. [56]

2.9.2.1. DVB-T

Es el estándar para la transmisión de televisión digital terrestre creado por la DVB. Consiste en la transmisión audio, video y datos a través de un flujo MEPG-2 aplicando la Multiplexación por División de Frecuencia Ortogonal (COFDM). [57]

Para alcanzar la máxima eficiencia espectral cuando se usa en las bandas UHF, la técnica OFDM con dos opciones en número de portadoras, tres esquemas de modulación y diferentes intervalos de guarda permiten la operación de pequeñas y grandes Redes de Frecuencia Simple (SFN). En un rango específico, la recepción de programas idénticos desde un número de transmisores de la misma frecuencia es benéfica.

En cuanto a los requerimientos de ancho de banda, el espaciamiento de canal preferido es de 8 MHz, pero si se desea, también es posible usar espaciamiento de 7 MHz o 6 MHz escalando todos los parámetros del sistema (Ver ITU-R Recommendation BT.1306).

La corrección de errores concatenada puede ser separada en dos bloques: la codificación e intercalación exterior son comunes de las Especificaciones Iniciales de Cable y Satélite, y la codificación interna es común de la Especificación Inicial de Satélite. El uso de intercalación interna es exclusivo de los sistemas DVB-T.

Para acomodar diferentes velocidades de transmisión, en adición a 5 velocidades de código, 3 tipos de esquemas de modulación no-diferencial pueden usarse: QPSK, 16-QAM y 64-QAM. Los esquemas 16-QAM y 64-QAM también pueden ser utilizados en combinación con reglas de mapeo uniformes o no-uniformes y por lo tanto los flujos de datos pueden ser separados en flujos de datos de baja o alta prioridad con diferentes protecciones ante errores para propósitos de transmisión jerárquica. Por razones económicas del receptor, los sistemas DVB-T soportan la transmisión jerárquica pero no la codificación jerárquica. [58]

2.9.2.2. DVB-S2

Estándar para la transmisión de televisión digital satelital, como el nombre implica, corresponde a la segunda generación ya que es el sucesor del estándar DVB-S. Fue desarrollado en el 2003 y adoptado posteriormente por el Instituto de Estándares de Telecomunicaciones Europeos (ETSI). Se ha especificado con el fin de tener un mejor rendimiento, mayor flexibilidad y ofrecer menor complejidad en los receptores. Las mejoras se deben a la implementación de nuevos esquemas de modulación

	DVB-T
FEC	Codificación Convolucional + Reed Solomon $\frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{5}{6}, \frac{7}{8}$
Modos	QPSK, 16QAM, 64QAM
Intervalo de Guarda	$\frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}$
Tamaño de la FFT	2k, 8k
Pilotos dispersas	8 % del total
Pilotos continuas	2 % del total
Ancho de banda	6, 7, 8 MHz
Velocidad de transmisión típica (UK)	24 Mbit/s
Velocidad de transmisión máxima (@20 dB C/N)	31.7 Mbit/s (usando 8 MHz)
Proporción C/N requerida (@24 Mbit/s)	16.7 dB

Tabla 2.2: Especificaciones del Estándar DVB-T [24]

y nuevos modos de operación. Específicamente «Variable Coding and Modulation» (VCM) y «Adaptative Coding and Modulation» (ACM), los cuales optimizan el uso del ancho de banda cambiando dinámicamente los parámetros de transmisión. [18]

El estándar DVB-S2 ha sido especificado en torno a 3 conceptos clave: mejor rendimiento de transmisión, flexibilidad total y complejidad de receptor razonable.

Para alcanzar la mejor compensación rendimiento-complejidad, DVB-S2 se beneficia de los recientes desarrollos en codificación de canal y modulación (QPSK, 8-PSK, 16APSK, y 32APSK), el resultado es típicamente un incremento de 30 % de incremento de capacidad sobre su predecesor DVB-S bajo las mismas condiciones de transmisión. Adicionalmente, para aplicaciones de transmisión, DVB-S2 no está limitado al uso de QPSK y por lo tanto puede entregar velocidades de transmisión más altas sobre satélites de alta potencia, y así incrementa la capacidad de ganancia respecto a DVB-S. La funcionalidad VCM permite distintas modulaciones y niveles de protección de errores a ser usados y cambiados trama-a-trama. Esto puede combinarse con el uso de un canal de retorno para alcanzar ACM de lazo cerrado, y así permitir la transmisión de parámetros para ser optimizados para cada usuario individual, dependiendo de sus propias condiciones de enlace.

DVB-S2	
Modo	Variable Coding and Modulation y Adaptative Coding and Modulation
Esquemas de modulación	QPSK, 8PSK, 16APSK, 32APSK
Code rates	$\frac{1}{4}, \frac{1}{3}, \frac{2}{5}, \frac{1}{2}, \frac{3}{5}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \frac{5}{6}, \frac{8}{9}, \frac{9}{10}$
FEC	Low Density Parity Check (LDPC) + Bpose-Chaudhuri-Hocquenghem (BCH)
Eficiencia spectral	2 bit/s/Hz – 5 bit/s/Hz
Factores de roll-off	0,35 , 0,25 , 0,20

Tabla 2.3: Especificaciones del estándar DVB-S2 [25]

DVB-S2 es tan flexible que puede lidiar con cualquier característica de transpondedor satelital existente, con una gran variedad de eficiencias espectrales y requerimientos C/N asociados. Adicionalmente no está limitado a codificación de video y audio MPEG-2, pero está diseñado para manejar una variedad de formatos de audio-video e información incluyendo formatos que el Proyecto DVB está definiendo para aplicaciones futuras. [59]

2.10. MPEG-2 Transport Stream

También conocido como flujo de transporte, es un flujo de bits que transporta toda la información de programación en los medios de radiodifusión digital. Su principal propósito es facilitar la implementación de un sistema corrección de errores hacia adelante (FEC). [57]

Está conformado por paquetes de longitud constante. Dicha longitud es de 188 bytes, 4 de cabecera y 184 de payload, estos últimos almacenan el video, audio e información. La cabecera está compuesta por un byte de sincronización, espaciado de manera constante 188 bytes en el flujo de transporte, un bit indicador de error de transporte y un identificador de paquete de longitud de 13 bits el cual describirá los contenidos del payload para un determinado paquete. [60]

A los flujos de transporte se les aplica un algoritmo FEC Reed Solomon acoplado con un codificador convolucional de trellis. La data es organizada en paquetes

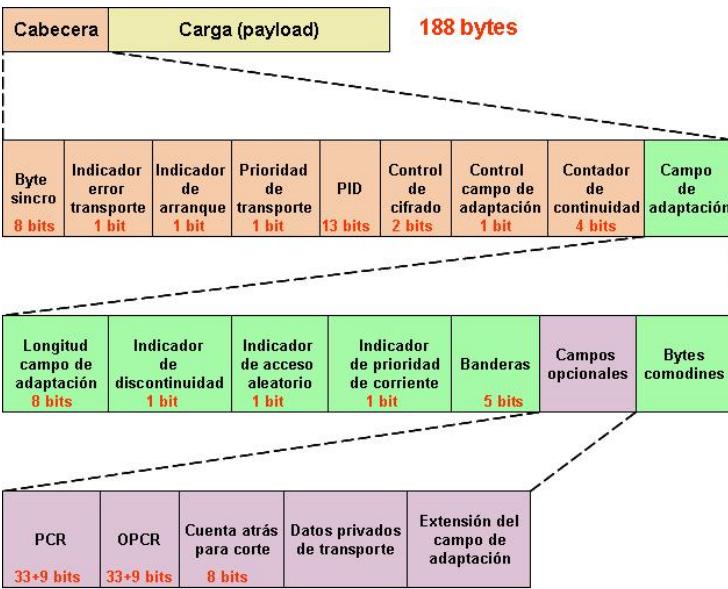


Figura 2.35: Trama MPEG-2 [22]

de cierta longitud y se les asigna una suma de chequeo de cierta longitud. Esta suma de chequeo permite no solo detectar errores, sino también corregir cierto número. El número de errores que pueden ser corregidos es una función directa de la longitud de la suma de chequeo. En Reed Solomon, el número de errores corregibles siempre corresponde a la mitad de los bytes de protección de errores (suma de chequeo). [60]

Como ya fue mencionado, la longitud de un flujo de transporte es de 188 bytes, y al emplearse un Reed Solomon FEC, se crea un paquete de datos de longitud de 204 bytes. Esto es llamado codificación RS (204, 188). En el receptor, hasta 8 errores pueden ser corregidos en este paquete de 204 bytes de longitud. La posición de estos errores no se especifica. Si existen más de 8 errores en un paquete, estos serán detectados pero no será posible corregirlos. Cuando esto ocurre el flujo de transporte es señalado como errado en el indicador de error de la cabecera del flujo, y así el paquete es descartado por el decodificador MPEG-2. [60]

Capítulo III

Procedimientos de la investigación

3.1. Revisión

El primer paso consistió en realizar la investigación teórica del funcionamiento y los componentes de un generador vectorial de onda arbitraria, con el fin de entender los requerimientos iniciales del proyecto. Se consultaron manuales y notas de aplicación de varios modelos existentes en el mercado, además de trabajos de investigación similares. Luego de esto, se procedió a estudiar los componentes que forman internamente al transceptor comercial HackRF One. Se revisaron los esquemáticos del transceptor y los datasheets de los componentes con el fin de conocer las limitaciones del mismo en cuanto a frecuencia, tasa de muestreo y potencia, previo a iniciar la implementación.

Seguidamente, se revisó el software GNU Radio para determinar los esquemas de modulación que podrían ser implementados y se encontró una amplia variedad de bloques relacionados con la modulación análogica y digital de señales. En este sentido, debido a que con esta cantidad de esquemas de modulación a implementar, se consideró innecesaria la creación de nuevos bloques de procesamiento de señales, además de que esta opción le restaría versatilidad al proyecto ya que para poder ejecutar la aplicación con estos esquemas sería necesario instalar los módulos creados en cada equipo. Así mismo, se debe señalar que se trabajó con GNU

Radio Live SDR, el cual es una distribución de Ubuntu Linux con GNU Radio con software pre-instalado para ejecutar la versión 3.7.5 GNU Radio sin necesidad de tener previamente instalada alguna distribución de Linux.

3.2. Pruebas de Adquisición

En esta etapa del proyecto, se procedió a determinar los tipos de entrada de datos que serían posibles de utilizar de acuerdo al esquema de modulación deseado. Para esto se construyeron diagramas de prueba en la aplicación GRC (GNU Radio Companion) que viene instalada con GNU Radio. En este sentido, cada diagrama implementaba un esquema de modulación y contaba con distintos tipos de entrada, los cuales son: Fuente de señal, fuente de audio, fuente aleatoria, fuente de archivo, fuente de vector de datos y fuente UDP (es utilizado para realizar streaming de datos). A partir de esto, se escogieron los tipos de entrada que eran compatibles con los esquemas de modulación deseados. Por lo tanto, la fuente de señal admite el uso de distintas formas de onda como, senoidal, cuadrada, triangular, diente de sierra y constante.

Por otro lado para la fuente de audio, se utilizó la señal proveniente de la entrada de audio de la tarjeta de sonido del computador, la cual se probó reproduciendo archivos de audio desde distintos dispositivos conectados mediante un cable auxiliar a la entrada de micrófono de la computadora. Así mismo, para comprobar que la transmisión fuese correcta, en la mayoría de los casos se utilizó un diagrama dentro de GNU Radio que incluyese modulación y demodulación donde el receptor fuese un sumidero de audio de modo que se escuchase la señal demodulada. Además, cabe señalar que para el caso de FM, se probó utilizando un receptor FM comercial para demodular y escuchar el audio transmitido. Para el caso de transmisión de videos, se debió cumplir con el requerimiento de que los archivos estuviesen en flujo de transporte (archivos .ts), a menos que se transmitan a través de streaming de datos mediante la fuente UDP. En este sentido, la transmisión de señales desde streaming se realizó utilizando el software de reproducción de video

VLC, el cual se configuró como fuente de video desde la opción de streaming (enlace de internet, o archivo del computador), y se asignó un puerto UDP para la salida del video asociada a la dirección IP del computador que actúa como servidor. Luego, en GNU Radio, la fuente UDP se configuró con la dirección IP del servidor y el número de puerto UDP utilizado para obtener el servicio de streaming de video de VLC.

Adicionalmente, se debe señalar que el prototipo tiene método de adquisición imágenes, esencialmente para la transmisión del estándar de televisión analógica (NTSC). En este caso, la imagen a cargar debe ser procesada primeramente a través del programa escrito en Python ‘ntsc-encode’ [61], el cual convierte la imagen en un archivo de datos con la codificación correcta para asociarla a cada línea de la televisión al momento de transmitir con el estándar NTSC.

3.3. Elaboración de diagramas de bloques en GNU Radio

Tras estudiar las librerías disponibles en GNU Radio, se decidió construir diagramas para los siguientes esquemas de modulación: DSB-SC, AM, NBFM, WBFM, ASK, OOK, PSK, GMSK, GFSK, QAM, OFDM y diagramas para transmisores de los esquemas de televisión analógica NTSC, y digitales DVB-T y DVB-S2 ya que GNU Radio contiene los bloques moduladores para estos esquemas en la mayoría de los casos, y en los que no, se pudieron construir utilizando distintos bloques del programa.

Por otro lado, para el caso de las fuentes, se utilizó la fuente de audio de la librería de GNU Radio. No obstante, para el caso de las fuentes de video e imágenes se utilizó la librería de Operadores de Archivos. Mientras que para la generación de señales de tipo senoidal, cuadrada, triangular, diente de sierra, constante, se usó la librería de Generadores de Forma de Onda. Cabe señalar que la fuente de vector utilizada, se encuentra en la librería de Misceláneos. Finalmente, la fuente UDP usada para streaming se encuentra disponible en la librería de Herramientas de Red.

En cuanto a los moduladores, la mayoría puede encontrarse en la librería de Moduladores, incluido el modulador de constelación utilizado para los esquemas de OOK y 4-ASK. Adicionalmente, se utilizaron bloques moduladores de la librería de OFDM. Finalmente, para el estándar de televisión digital satelital DVB-S2 se utilizaron todos los bloques disponibles en la librería «dvbs2», los cuales incluyen codificadores, intercalador, cabecera, modulador y enmarcador de capa física. Esta misma metodología se aplicó para el estándar de televisión DVB-T mediante la librería «dvbt» donde se utilizaron bloques codificación, mapeo, señales de referencia, FFT, prefijo cíclico para OFDM y sincronización.

Por otro lado, para los casos de modulación en amplitud y el formato de televisión analógica NTSC, fue necesario el uso de bloques de la librería de Operadores Matemáticos como multiplicadores.

Finalmente, se debe señalar que todos los diagramas tienen en común el bloque de salida «osmocon sink», disponible en la librería de Sumideros, el cual se configura con los parámetros de frecuencia de transmisión del dispositivo, factor de corrección de frecuencia, y ganancia del mismo, proporcionando la compatibilidad con la HackRF One.

3.4. Caracterización del dispositivo

En esta etapa de desarrollo, se realizó una serie de mediciones para caracterizar a la HackRF One tanto en la frecuencia como en la potencia de salida de la tarjeta. Para ello, primero se realizaron mediciones dejando los parámetros de los amplificadores internos del transceptor fijos, y el ajuste en partes por millón (ppm) de frecuencia en cero en el bloque de salida de GNU Radio Companion. Una vez culminado este procedimiento, esto se procedió a realizar las mediciones desde 10 MHz hasta 3 GHz con pasos de 10 MHz enviando una señal constante de amplitud fija. Cabe señalar que para este proceso se utilizó el equipo Site Master SB331-B de Anritsu, cuyo rango abarca hasta los 3 GHz deseados.

En este sentido, para la rutina de medición se operó el equipo en su modo de Analizador de Espectros. Luego se seleccionó la frecuencia central igual a la de salida establecida en GNU Radio con un Span de 1 MHz, y se procedía a ubicar un marcador en el pico de potencia. De esta manera, se registraron los datos tanto de frecuencia como de potencia y se procedió a repetir el procedimiento para los próximos pasos de 10 MHz hasta cubrir el rango hasta 3 GHz, lo que registró un total de 300 mediciones con el fin de determinar el comportamiento del transceptor en ambos parámetros.

3.5. Calibración

3.5.1. En Frecuencia

Para realizar el procedimiento de calibración de frecuencia, se repitió la rutina de mediciones del apartado 3.4, en el cual se registraron un total de 300 mediciones. No obstante, se realizó esta misma rutina 30 minutos después de encendido el transceptor, y 60 minutos después de encendido el transceptor, con el fin de estudiar la estabilidad del mismo en respuesta al aumento de la temperatura.

Una vez registradas las mediciones en frecuencia, se calculó el factor de corrección en partes por millón que debe aplicarse a cada paso de frecuencia de 10 MHz, el cual se introduce en el bloque de salida hacia la HackRF One de los diagramas en GNU Radio Companion.

3.5.2. En Potencia

Se debe señalar que los pasos de ganancia del amplificador de ganancia IF no son siempre constantes, puesto que en ocasiones se ve reflejado un aumento o disminución de 1 dB correspondiendo así con sus especificaciones técnicas, pero en otras ocasiones un paso de ganancia podía ser menor a 1 dB.

Se decidió segmentar el rango de frecuencia y construir tablas para determinar los factores de ajuste para cada esquema en cuatro valores puntuales dentro del rango de frecuencia, y utilizar la interpolación para determinar el factor de ajuste para los valores intermedios. Los cuatro valores a los que se determinó el factor de ajuste de potencia fueron -10 dBm, -5 dBm, 0 dBm y 5 dBm.

Por lo tanto, el procedimiento de calibración consistió en fijar la potencia a uno de los 4 valores antes mencionados y mantenerla fija a medida que variaba la frecuencia; esto se logró aumentando y disminuyendo cuando fuese necesario la ganancia del amplificador de ganancia IF, teniendo siempre encendido el amplificador de ganancia RF. Para ello, la rutina de medición consistió en utilizar el Site Master SB-331B de Anritsu en el modo de operación de medidor de potencia, y una vez configurado así y hecho el pertinente ajuste en cero al equipo disponible en el menú de dicho modo, se procedió a enviar un pulso constante, y se varió la ganancia del amplificador IF hasta obtener uno de los valores antes mencionados (-10 dBm, -5 dBm, 0 dBm y 5 dBm). Este proceso se aplicó para el rango de frecuencia de 50 MHz hasta 3 GHz en pasos de 50 MHz variando la ganancia del amplificador de ganancia IF con el fin mantener una potencia constante.

Una vez registrados dichos valores, se procedió a repetir la rutina para cada uno de los esquemas de modulación y de esta forma determinar si era necesario un factor de ajuste adicional de cada esquema. Nuevamente, este procedimiento se repitió en 4 ocasiones para los valores de potencia antes mencionados.

Una vez determinados los factores de ajuste para cada esquema se procedió a construir una tabla más precisa, comparando los resultados entre todos los esquemas con su correspondiente ajuste aplicado a fin de determinar si el valor de ganancia seleccionado para señal constante era el más indicado. Seguidamente a las correcciones y ajustes, se realizó una nueva rutina de medición pero esta vez con pasos de frecuencia de 10 MHz con la finalidad de obtener un factor de calibración más preciso a lo largo del rango de operación del prototipo.

3.5.3. En Amplitud

Se debe señalar que para el caso de modulación en amplitud con portadora suprimida fue necesario realizar una calibración de la amplitud de salida de la señal modulada. Para ello, en primer lugar, se midió la amplitud pico-pico de salida de una señal senoidal de frecuencia variable y utilizando los ajustes de ganancia IF correspondientes a una potencia de salida de -5dBm, correspondiente a la señal portadora. Cabe destacar que se escogió este conjunto de ajustes de ganancia IF para que sea posible de obtener el mismo rango de amplitud de salida para todo el rango de frecuencia de operación del dispositivo. En este caso, las mediciones se realizaron en el Laboratorio de Antenas de la Escuela de Ingeniería Eléctrica en la Facultad de Ingeniería de la Universidad de Carabobo, en el rango de frecuencias de 50MHz hasta 400MHz, debido a las limitaciones del equipo disponible en el laboratorio, un osciloscopio digital de tiempo real marca Tektronix modelo TDS380.

Una vez realizada la rutina descrita anteriormente, se determinó la amplitud mínima que se podía generar la señal senoidal en GNU Radio para que pudiese ser transmitida sin presentar distorsión al ser observada en un analizador de espectro. En ese sentido, se utilizó el mismo equipo mencionado en la sección 3.4 y después de haber obtenido este valor se realizó un nuevo conjunto de mediciones en el Laboratorio de Antenas utilizando el mismo osciloscopio mencionado anteriormente. Para ello, la tarea consistió en transmitir una señal senoidal de 300MHz con amplitud variable controlando este parámetro en el diagrama de GNU Radio y medir la amplitud correspondiente en el osciloscopio, utilizando como parámetro de medición el voltaje pico-pico. Estas mediciones se realizaron con la finalidad de obtener los valores mínimos y máximos en amplitud posibles de generar por el dispositivo, además de obtener la calibración necesaria para configurar el parámetro del bloque de señal en GNU Radio al ser especificado un valor de amplitud dentro del rango disponible.

3.6. Diseño de la Aplicación

El desarrollo de la aplicación se realizó utilizando el lenguaje de programación Python ya que es un lenguaje sencillo, multiplataforma y gracias a su elegante sintaxis, su gestión de tipos dinámicos y su naturaleza interpretada hacen de él un buen lenguaje para scripts y desarrollo de aplicaciones. De igual manera, el lenguaje utilizado por la aplicación GNU Radio Companion para realizar la compilación de los diagramas de flujo que fueron construidos en la sección 3.3 es Python [39], por lo que el lenguaje de la aplicación que se encarga de organizar y ejecutar esos diagramas también es Python.

Para el desarrollo de la aplicación de control, primeramente se realizó un bosquejo gráfico para determinar los elementos gráficos con los que la aplicación contendría en su interfaz, es decir, todo lo referente a etiquetas, cuadros de texto, listas y botones. Luego, se procedió a programar la funcionalidad de la aplicación en cuanto a la ejecución de los diferentes esquemas de modulación, los cuales están representados por los diagramas construidos en GNU Radio Companion en la sección 3.3. Estos diagramas poseen un código asociado en lenguaje Python el cual se puede obtener utilizando la función de GNU Radio Companion (GRC) de generar el código del diagrama realizado. Cabe señalar que en este nivel de programación, se le realizaron algunas modificaciones para incluir las calibraciones de potencia y frecuencia que fueron determinadas en la sección 3.5.

Para finalizar, a la interfaz gráfica se agregaron algunos detalles estéticos, como el display de frecuencia y potencia para darle a la aplicación un aspecto más agradable y similar a los equipos físicos que se encuentran en el mercado.

Capítulo IV

Análisis, interpretación y presentación de los resultados

4.1. Estructura del Generador Vectorial de Onda Arbitraria Basado en SDR

Un transmisor SDR difiere de un transmisor tradicional en que el segundo, traslada en frecuencia a la señal modulante al momento de efectuarse la modulación. En el caso de SDR, este proceso consiste, en primera instancia, en efectuar la modulación en *software*, donde el resultado es la señal modulada en banda base, para luego ser trasladada en frecuencia por un hardware determinado. Esta diferencia en los procesos puede apreciarse en la Figura 4.1.

El esquema del prototipo puede verse en la Figura 4.2. La generación de las señales se consiguió a través del software GNU Radio Companion el cual es compatible con el transceptor comercial de radio frecuencia HackRF One mediante la instalación de las librerías y dependencias correspondientes que permitirán al *software* interactuar con el transceptor. Este proceso de generación de señales es realizado por el computador ejecutando distintos diagramas elaborados en el *software* antes mencionado, y luego transmitiendo las tramas de datos al transceptor vía USB; allí la información es procesada por microprocesador LPC43xx del mismo,

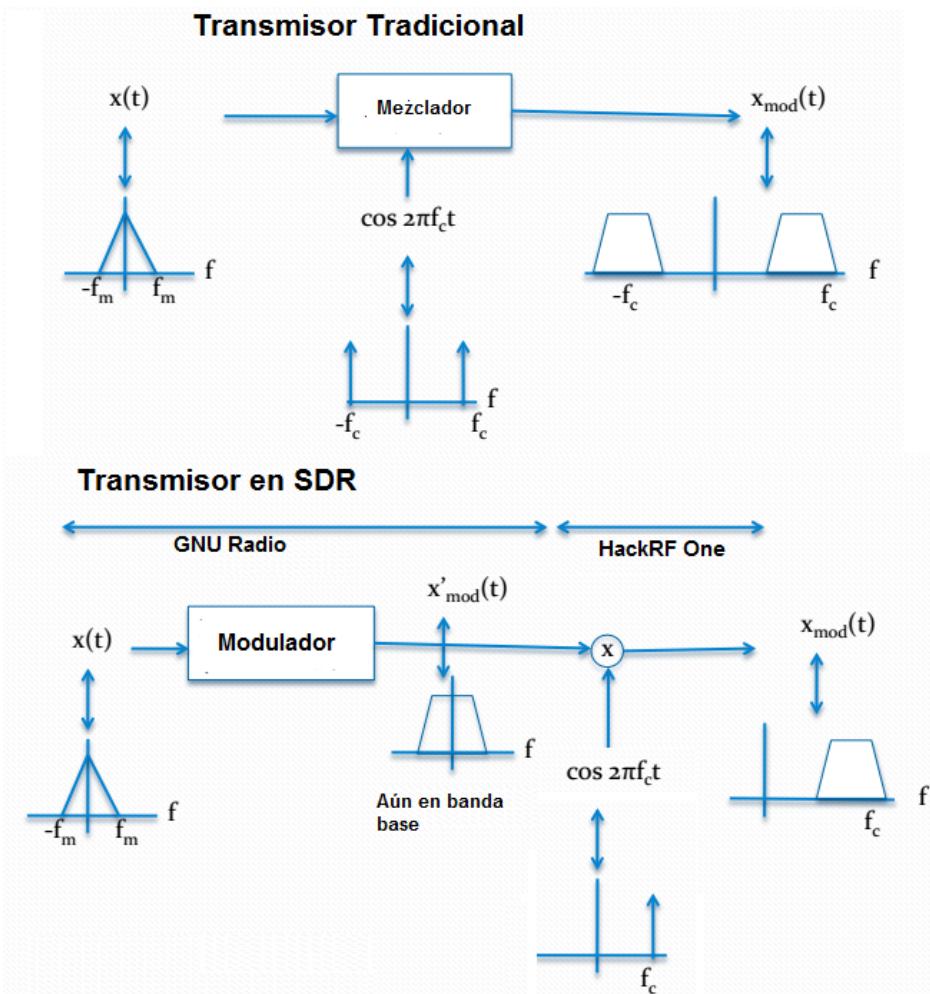


Figura 4.1: Diferencia entre un transmisor tradicional y uno basado en SDR

para luego ser transmitidas al convertidor digital/analógico MAX5864, que a pesar de tener una capacidad de procesamiento de 20 millones de muestras por segundo de acuerdo a las especificaciones del mismo, la máxima capacidad utilizada dependerá del esquema de modulación en el que se esté trabajando.

Luego, las señales I y Q, provenientes del convertidor digital/analógico, se envían al modulador en cuadratura MAX2837, necesario en la estructura de un generador vectorial de onda arbitraria como se expuso en la Sección 2.1. Cabe señalar que en esta etapa se controla también la ganancia IF de la señal a transmitir, pudiendo variarla de 0 hasta 47 dB en pasos de 1 dB, lo que limitará el rango de potencia de

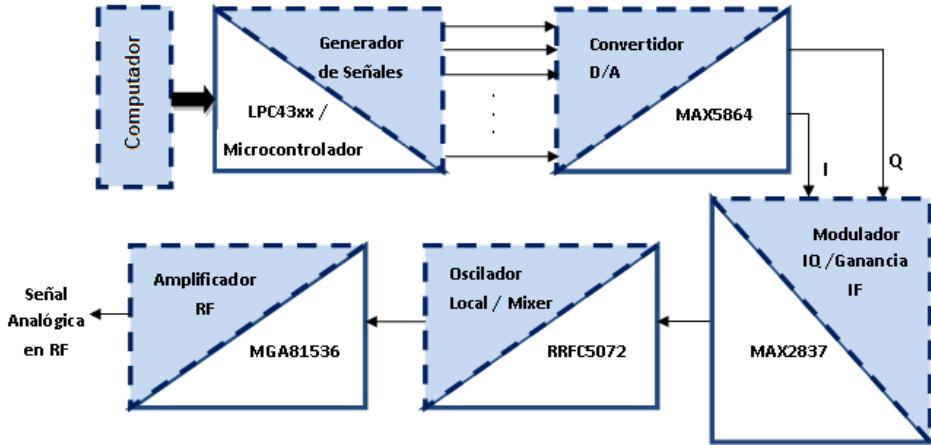


Figura 4.2: Diagrama de Bloques del Generador Vectorial de Onda Arbitraria

operación del generador dependiendo también del tipo de esquema de modulación que se quiera utilizar. Así mismo, se debe agregar que este valor de ganancia IF será el que el usuario manipulará cuando desee cambiar la potencia de transmisión.

Seguidamente la salida del modulador en cuadratura va al Mezclador/Sintetizador RRFC5702 el cual ejecuta la acción de oscilador local permitiendo trasladar la señal modulada a radio frecuencia, específicamente entre 50 MHz y 3 GHz, rango de frecuencia para el cual está calibrado el generador vectorial de señales. Finalmente la señal a la salida del mezclador va al amplificador de RF MGA81563 el cual solo tiene dos estados, encendido y apagado; no obstante, para el caso del prototipo del generador, siempre operará en su estado de encendido para alcanzar un rango de operación en potencia entre -10 dBm y 5 dBm, dependiendo del esquema de modulación en el que se esté trabajando.

4.2. Generación de Señales

Como se mencionó en la sección anterior, la generación de las señales se hace a través del software GNU Radio Companion, empleando una serie de diagramas construidos de acuerdo al esquema de modulación con el que se quiera trabajar, los cuales incluyen una serie de variables que el usuario manipulará a través de una

interfaz gráfica para establecer las características de la señal deseada dentro de las limitaciones que permita la HackRF One y el computador que se emplee.

Para cada esquema de modulación existen varios diagramas dependiendo la fuente con la que el usuario desee trabajar, es decir, en la mayoría de los casos, la estructura del diagrama de bloques en GNU Radio Companion permanece igual para los distintos diagramas del mismo esquema de modulación a excepción del bloque fuente, el cual puede ser una fuente de señal, archivo aleatoria o audio.

Todos estos diagramas generan un archivo Python (.py), los cuales se encargan de crear los bloques y las conexiones necesarias para el funcionamiento de los diagramas, y son dichos archivos los que constituyen la aplicación final que controla el generador.

4.2.1. Modulaciones Analógicas

4.2.1.1. Portadora Suprimida (DSB-SC)

El diagrama básico de la modulación de doble banda lateral con portadora suprimida (Figura 4.3) consiste en enviar la fuente de señal al bloque «*Osmocon Sink*», el cual es el encargado de comunicar al computador con el transceptor HackRF One. Antes de enviarla directamente, se pasa la señal por un bloque que cambia su variable de tipo «float» a complejo, de modo que aparezcan ambas bandas laterales del espectro, cosa que no ocurre si se conectan directamente la fuente de señal y el «*Osmocon Sink*».

En el diagrama se aprecian una serie de variables que son las que le permitirán al usuario manipular las características de la señal a modular. Éstas son `mod_freq`, `carrier_freq` y `amp`, las cuales permiten controlar la frecuencia de la señal modulante, la frecuencia de la portadora y la amplitud de la señal. Existen también otras variables que el usuario manipulará de forma indirecta como lo son el `samp_rate` o frecuencia de muestreo que limita la frecuencia de la señal modulante hasta 1 MHz ya que ésta será igual a 20 veces la frecuencia de la modulante y la variable `mult`

que está asociada a la variable `amp` y se encarga de escalar los valores de amplitud de la señal entre 1 (valor máximo para evitar saturación del conversor D/A) y 0,2 (valor mínimo para evitar distorsión). Adicionalmente se tiene otra variable que el usuario no manipulará las cuales son `corr` que corresponde con el ajuste en ppm para la frecuencia de la portadora. Es importante destacar que el usuario podrá elegir entre potencia o amplitud al momento de ajustar la señal. En este caso de utiliza otro diagrama similar donde se descarta el multiplicador, y la variables `texttpotencia` y `if_gain` sustituyen a las variables `amp` y `mult`; la primera corresponde a la potencia de salida y la segunda, vinculada con la variable `potencia` y es la encargada de variar el valor de la ganancia IF.

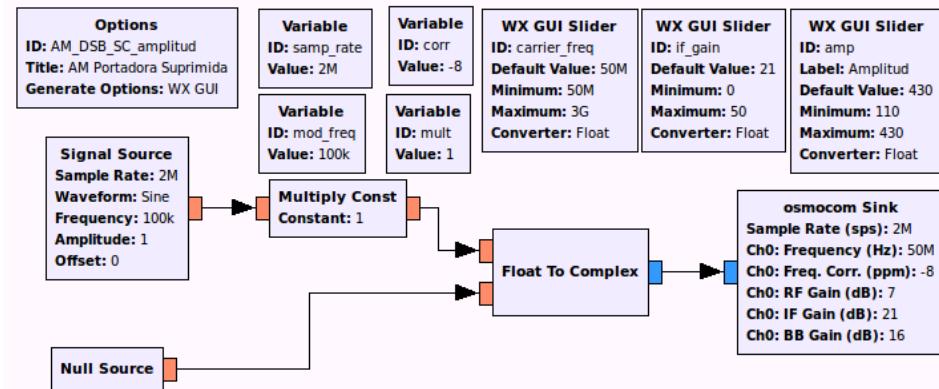


Figura 4.3: Diagrama en GNU Radio Companion de la Generación de una señal DSB-SC

En el caso de que el usuario desee enviar una señal de audio, el diagrama cambia ligeramente al de la Figura 4.4, donde la fuente de señal es sustituida por una fuente de audio seguida por un filtro pasabajas con el fin de eliminar componentes no deseadas provenientes de la tarjeta de sonido del computador.

Si el usuario desea enviar una señal arbitraria, también se utiliza otro diagrama como puede verse en la Figura 4.5, donde la fuente lee los valores de un vector la cual como su nombre indica, es un vector que constituirá un período de la señal a enviar, la cual pasa a través de un primer bloque multiplicador el cual tras identificar el máximo valor absoluto de dicho vector, escala cada uno de sus componentes dividiendo por dicho valor máximo con el fin de garantizar la no

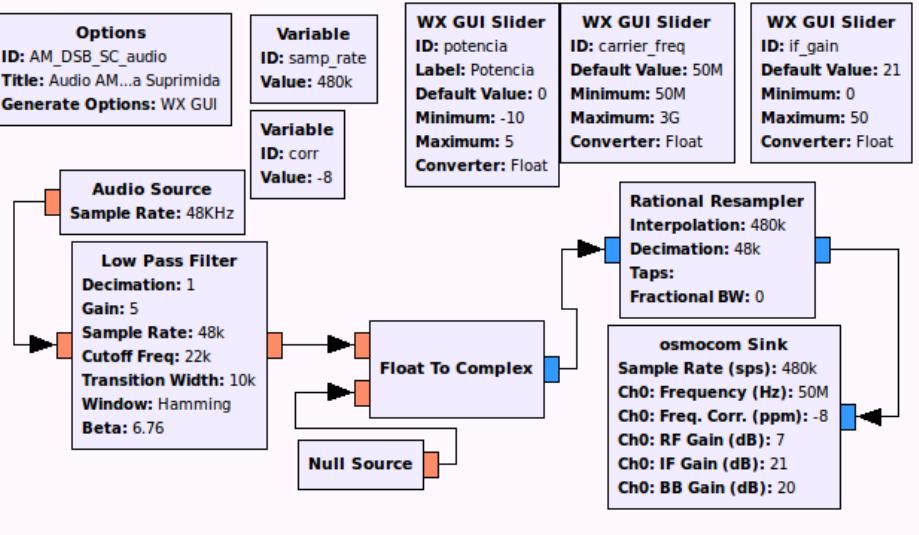


Figura 4.4: Diagrama en GNU Radio Companion de la Generación de una señal DSB-SC con fuente de Audio

saturación del conversor D/A. Luego de la salida de este multiplicador, el resto del diagrama es el mismo al diagrama base de DSB-SC de la Figura 4.3. Para el caso de la señal arbitraria, al fijar el usuario el valor de la frecuencia modulante, la variable samp_rate será igual al producto del número de muestras del archivo leído por la frecuencia de la modulante para corresponda con la frecuencia de la señal deseada.

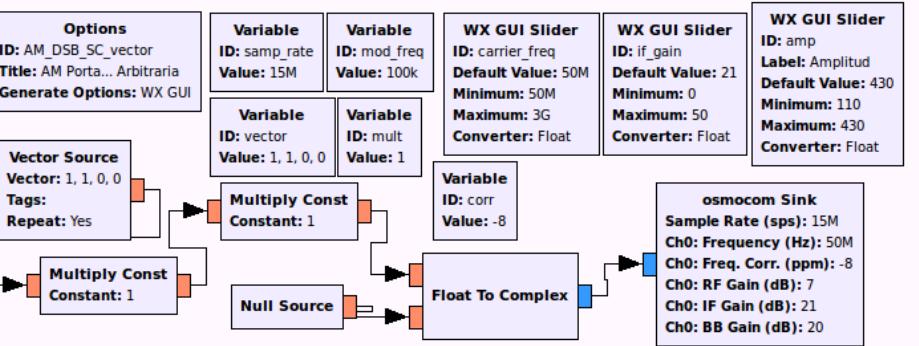


Figura 4.5: Diagrama en GNU Radio Companion de la Generación de una señal DSB-SC con fuente de señal arbitraria

En la Figura 4.6 se observa el espectro de una señal senoidal de frecuencia 200 kHz modulada con una portadora de 1.2 GHz. Es importante destacar que a pesar

de tratarse de una señal de doble banda lateral con portadora suprimida, el transceptor HackRF One induce una pequeña señal portadora propia del dispositivo la cual se aprecia a la frecuencia central de 1.2 GHz, mientras que la banda lateral inferior se encuentra en 1.1998 GHz correspondiente con el valor teórico calculado.

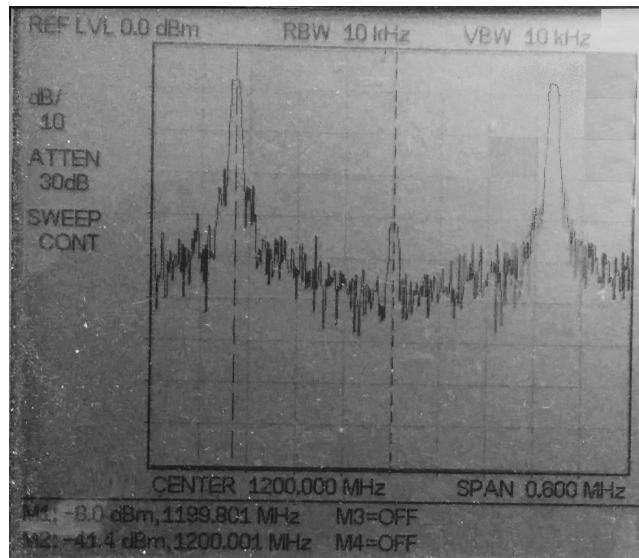


Figura 4.6: Espectro de una señal senoidal con modulación DSB-SC de frecuencia modulante 200 kHz y frecuencia de la señal portadora de 1.2 GHz

4.2.1.2. Gran Portadora (AM)

El diagrama para generar la señal AM-LC o Gran Portadora, consiste en sumar una constante que consiste en la portadora, y la señal modulante. La suma de estas dos señales no debe ser mayor a 1 para evitar la saturación del conversor D/A de la HackRF One, por lo que las amplitudes tanto de la constante como la de la señal, bien sea senoidal, cuadrada, triangular o diente de sierra, serán iguales a 0,5.

Al igual que en el caso de la sección anterior, se cuenta con una serie de variables; `samplerate`, `potencia`, `if_gain`, `mod_freq`, `carrier_freq` y `corr` cumplen con la misma función que en el apartado anterior, pero ahora se cuenta con la variable `m` la cual es una variable que el usuario podrá modificar a través de la interfaz de entrada y que corresponde al índice de modulación.

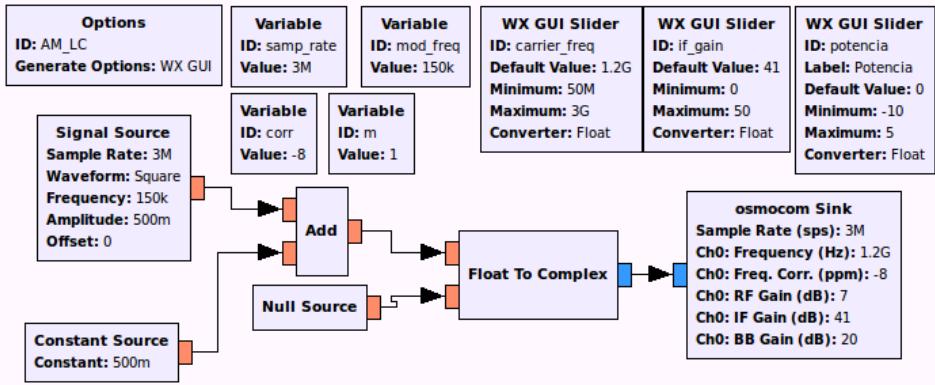


Figura 4.7: Diagrama en GNU Radio Companion de la Generación de una señal AM

Para el caso en que se quiera transmitir audio o una señal arbitraria, el procedimiento es el mismo que para el caso de la modulación DSB-SC, donde la fuente de audio y el filtro sustituyen a la fuente de señal para el caso del audio (Figura 4.8), y la fuente de vector acompañada de la constante de multiplicación encargada de escalar la señal arbitraria dentro de los límites de amplitud correspondientes (Figura 4.9). Al igual que en el caso anterior, luego de que el usuario ajuste la frecuencia de la modulante se calcula la frecuencia de muestreo de acuerdo al número de muestras del vector.

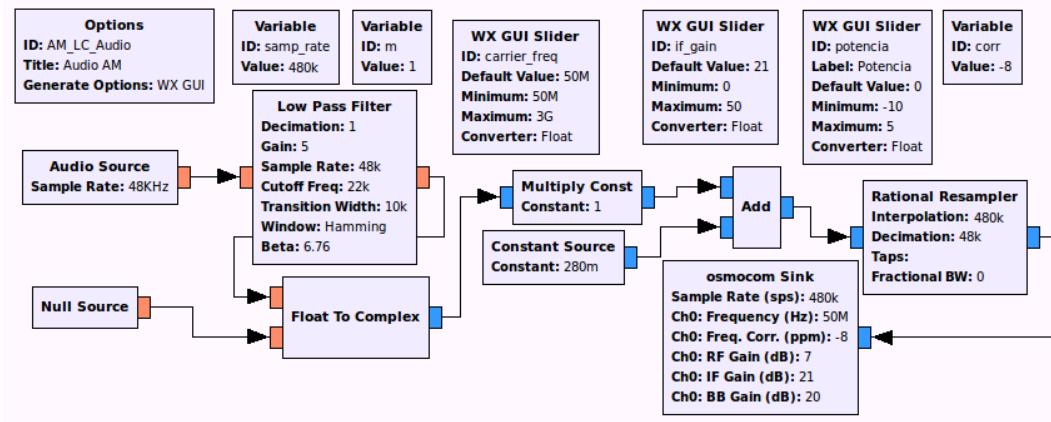


Figura 4.8: Diagrama en GNU Radio Companion de la Generación de una señal DSB-SC con fuente de audio

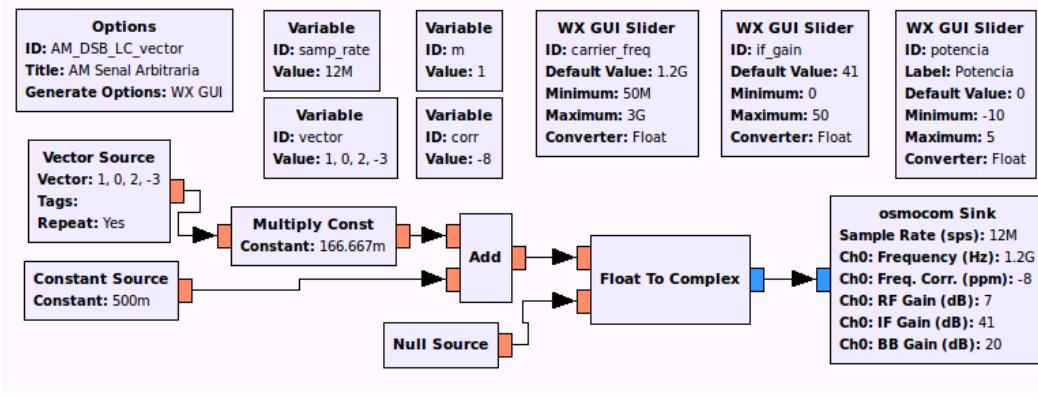


Figura 4.9: Diagrama en GNU Radio Companion de la Generación de una señal AM con fuente de señal arbitraria

En la Figura 4.10 se puede ver el resultado de la modulación AM para dos valores de índice de modulación. Primeramente en la Figura 4.10a se aprecia el espectro de una señal senoidal de frecuencia modulante igual a 200 kHz, frecuencia de señal portadora de 1.2 GHz e índice de modulación igual a 1. Mientras que en la Figura 4.10b se observa la misma señal modulante con la misma frecuencia portadora pero esta vez con índice de modulación igual a 0,5.

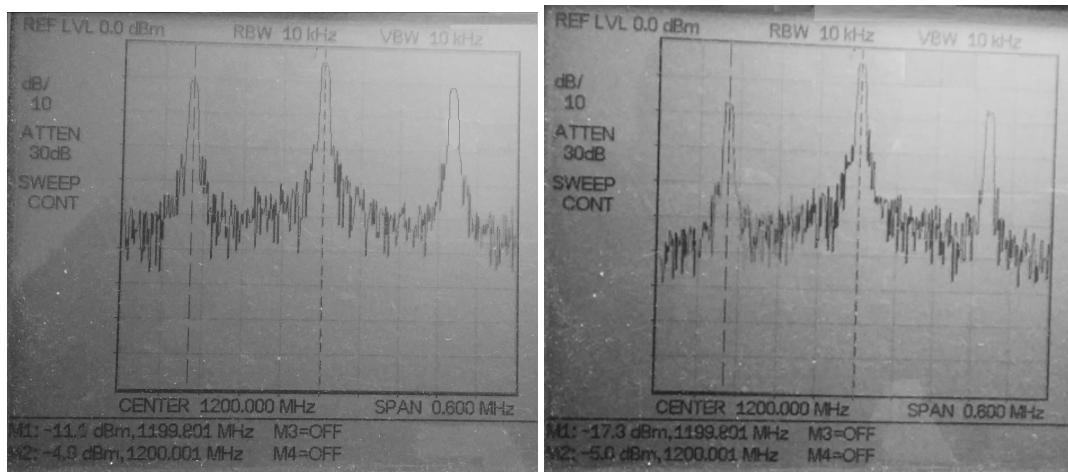


Figura 4.10: Modulación AM

En ambos casos se aprecia que la banda lateral inferior se encuentra 200 kHz por debajo de la frecuencia de 1.2 GHz de la portadora. Además se aprecia una

disminución de amplitud de las bandas laterales de -11.1 dBm a -17.3 dBm cuando se pasa de un índice de modulación de 1 a 0,5 respectivamente lo que corresponde con los valores teóricos.

4.2.1.3. Modulación en Frecuencia (FM)

En el caso de la modulación en frecuencia, se ofrecen las alternativas de la modulación en banda angosta y en banda ancha. Ambos diagramas son similares y varían solo en los bloques de modulación. Además de las variables ya mencionadas en los apartados de las modulaciones de amplitud analógicas que controlan la frecuencia de portadora y potencia, el usuario podrá controlar directamente la frecuencia de la señal modulante mediante la variable `mod_freq`, y la máxima desviación de frecuencia mediante la variable `max_freq_dev`. Como dichos bloques fueron diseñados con la finalidad de trabajar con señales audibles, en los módulos Python `nbfm_tx.py` y `wfm_tx.py` que son los constructores de los bloques moduladores de banda angosta y banda ancha respectivamente, se aprecia el uso de un filtro pasa banda con frecuencia de corte de 4 kHz y transición hasta 7 kHz para banda angosta y frecuencia de corte de 16 kHz con transición hasta 18 kHz para banda ancha. Con el fin de ofrecer un rango mayor de frecuencias modulantes al usuario, se creó un duplicado del archivo con nuevos valores de frecuencia de corte (50 kHz para banda angosta y 150kHz para banda ancha) los cuáles serán incluidos en el diseño de la aplicación y serán estos los que se encargarán de asignar las variables que se pasan a los diagramas correspondientes.

Además, al igual que los casos para la modulación analógica en amplitud, ambos esquemas admiten la transmisión de señales de audio y arbitrarias. En ambos casos, los diagramas de modulación en frecuencia siguen la misma estructura que los usados en la modulación en amplitud; para audio se utiliza la fuente de audio y un filtro pasabajas con ganancia antes de la entrada del modulador, y para la señal arbitraria se utiliza una fuente que leerá la señal arbitraria de un vector de datos seguido de un multiplicador que escalará la señal arbitraria a los valores necesarios para no saturar conversor D/A. Además en el caso de la señal arbitraria, el usuario deberá adaptar el número de muestras de la señal de acuerdo a la frecuencia

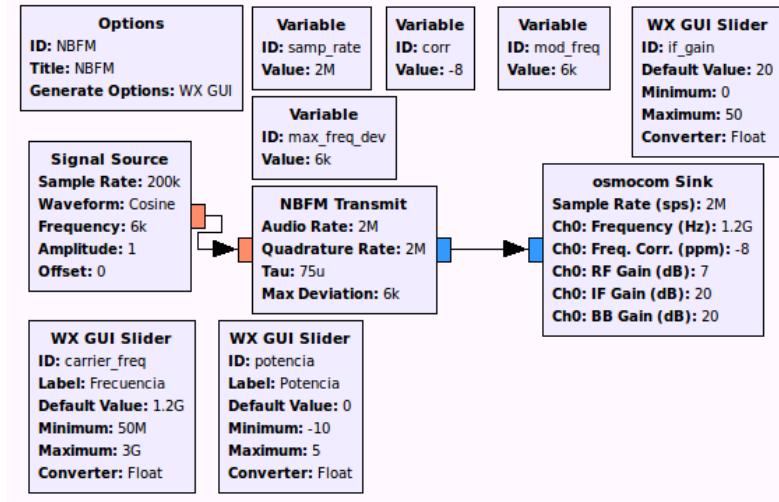


Figura 4.11: Diagrama en GNU Radio Companion para la generación de Señales NBFM

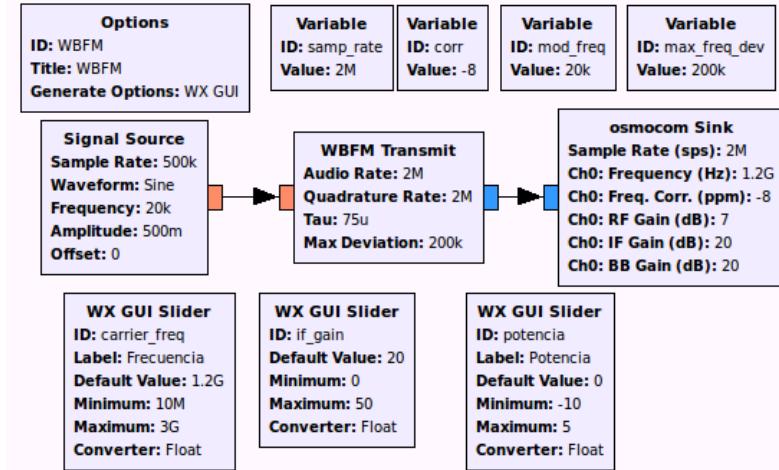


Figura 4.12: Diagrama en GNU Radio Companion para la generación de Señales WBFM

de muestreo para obtener la señal modulante deseada. Los diagramas para dichos tipos de fuente pueden verse en las Figuras 4.13 y 4.14.

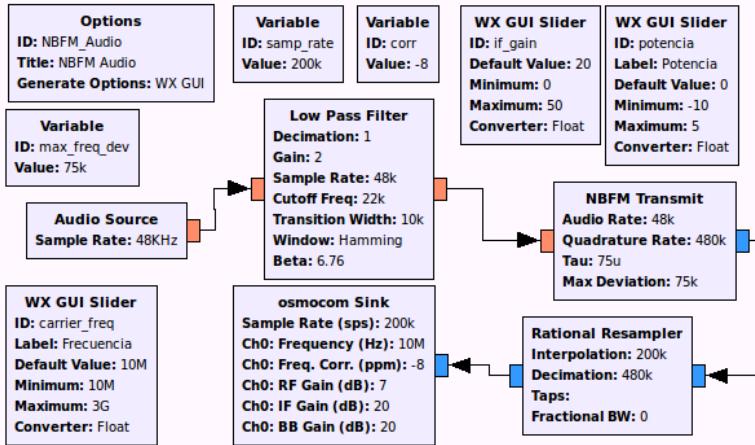


Figura 4.13: Diagrama en GNU Radio Companion para la generación de Señales FM con fuente de audio

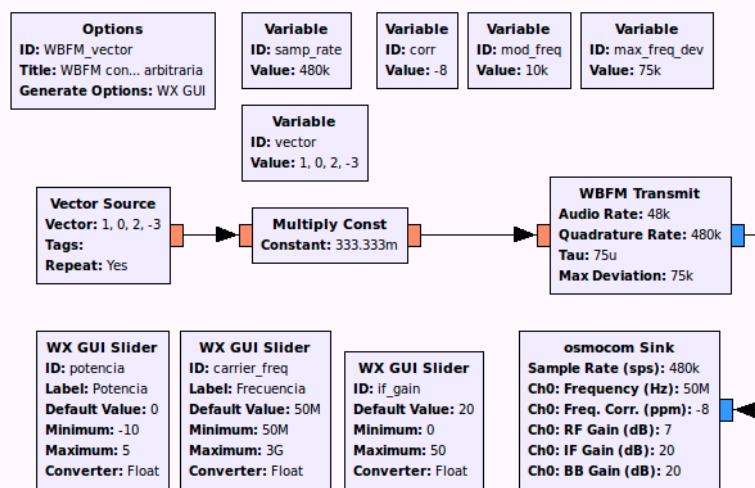


Figura 4.14: Diagrama en GNU Radio Companion para la generación de Señales FM con fuente arbitraria

En la figura 4.15a se aprecia una señal senoidal con modulación en frecuencia de banda angosta con $\beta = 0,8$ (frecuencia modulante = 25kHz y $\Delta f = 20\text{kHz}$), mientras que en la Figura 4.15b se observa otra señal FM de banda angosta con la misma desviación de frecuencia pero con una frecuencia modulante de 50kHz resultando en un $\beta = 0,4$, lo que resulta en un mayor espaciamiento de las bandas laterales y una disminución de amplitud en las mismas, similar al comportamiento

de AM.

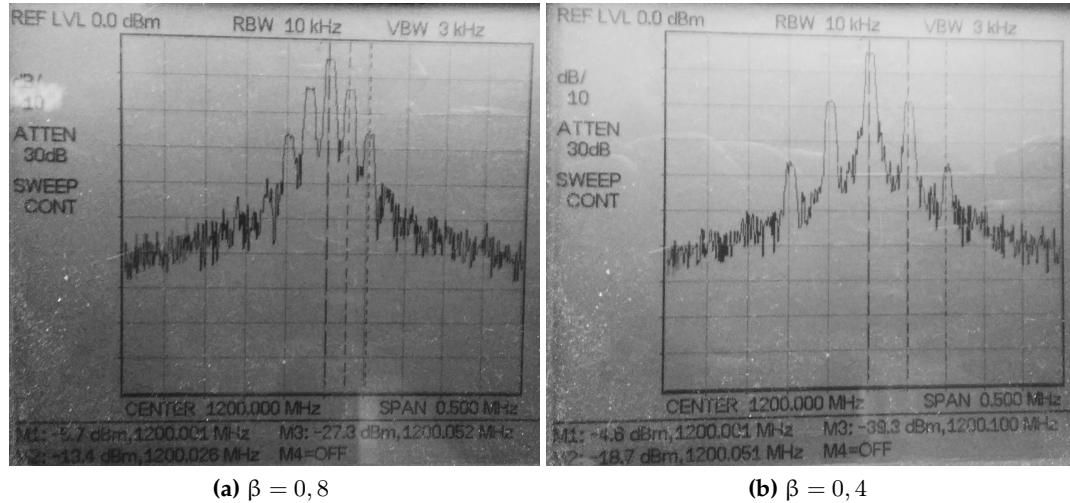


Figura 4.15: Espectro de Señales NBFM para distintos valores de β

Por otro lado en la Figura 4.16a se observa el espectro de una señal senoidal con modulación en frecuencia de banda ancha con frecuencia modulante de 10kHz e índice de modulación $\beta = 10$. Comparándola con la Figura 4.16b donde se mantiene constante la frecuencia modulante pero el índice de modulación aumenta a $\beta = 20$, se aprecia que el ancho de banda de la señal aumenta, siendo para ambos aproximadamente $2\Delta f$ por lo que para la señal de $\beta = 10$ se tiene un ancho de banda aproximado de 200kHz y para la de $\beta = 20$ el ancho de banda es el doble.

4.2.2. Modulaciones Digitales

El ancho de banda teórico de las modulaciones digitales depende principalmente de la tasa de bit y del número de bits por símbolo. En el caso de SDR, esto no se cumple ya que el ancho de banda viene dado por la relación entre la tasa de muestreo y la cantidad de muestras por símbolo, independientemente de la cantidad de bits por símbolo. De acuerdo a esto, se ajustaron los distintos diagramas en GNU Radio Companion para que el usuario pueda seleccionar un valor de tasa de bit e internamente se haga una equivalencia respecto a los valores de los parámetros

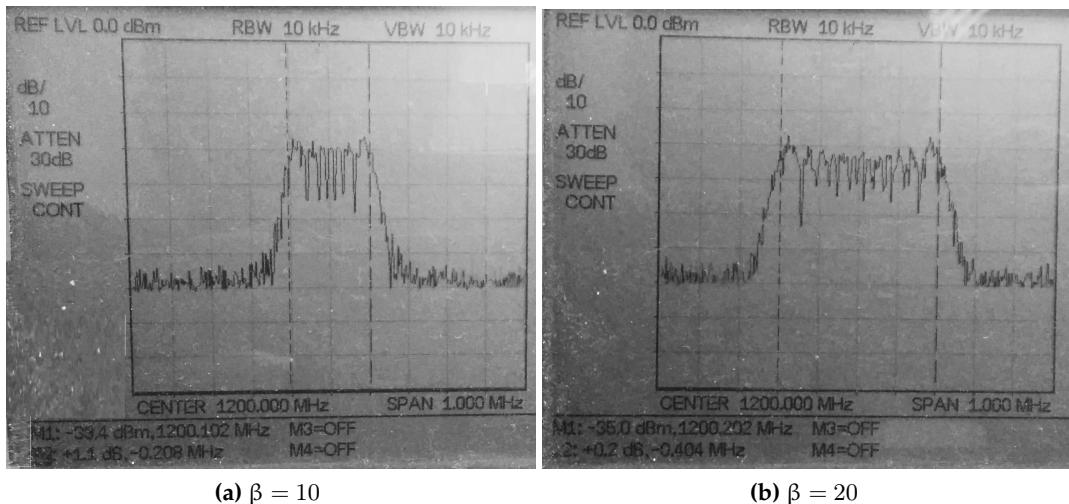


Figura 4.16: Espectro de Señales WBFM para distintos valores de β

de interés en SDR para que exista una concordancia con los resultados que deben obtenerse teóricamente.

La velocidad de procesamiento de los bloques de modulación digital, comprometen la integridad de la información a transmitir ya que ésta se pierde mientras mayor sea la velocidad de transmisión. Por lo que se estableció como tasa de bit mínima 100 kbps, por lo tanto, a medida que este parámetro aumente, el porcentaje de información perdida también aumentará. Al estar la tasa de bit asociada la frecuencia de muestreo, el máximo que pueda alcanzarse estará ligado a la capacidad de procesamiento del computador que ejecute la aplicación.

Del mismo modo, es importante añadir que este tipo de modulaciones le permitirán al usuario a elegir entre distintos tipos de fuente: señal aleatoria, de audio, archivo o via streaming.

4.2.2.1. Modulación por Desviación de Amplitud (ASK)

Se construyeron dos diagramas en GNU Radio Companion, uno para el caso binario (OOK), y uno para la modulación de 4 niveles (4-ASK). Ambos diagramas son similares, variando únicamente en el «*Constellation Rect. Object*», bloque en el

cual se ha fijado las constelaciones respectivas para cada esquema, $[0+0j, 1+0j]$ para OOK y $[-1+0j, -0.33+0k, 0.33+0j, 1+0j]$ para ASK. Los diagramas para ambos esquemas pueden verse en las Figuras 4.17 y 4.18. Ambos diagramas consisten en una fuente aleatoria la cual pasa por un «*Throttle*» que funciona como una especie de embudo para evitar el congestionamiento de muestras y la pérdida de información; a la salida del *throttle* se pasa a un codificador de paquetes donde se ajusta el número de bits por símbolo de acuerdo al esquema de modulación (1 para OOK y 2 para 4-ASK), y seguidamente su salida se conecta a la entrada del modulador de constelación donde se hace referencia al «*Constellation Rect. Object*» y se ajusta el factor de roll-off del filtro. En ambos diagramas se cuenta con las mismas variables para el control de la frecuencia de portadora y potencia de salida que en el caso de las modulaciones analógicas. Además se cuenta con la variable *rolloff* que corresponde al factor de roll-off del filtro, la cual el usuario podrá ajustar en la interfaz gráfica. El usuario también controlará de forma indirecta la variable de *samp_rate* que corresponde a la frecuencia de muestreo; dejando la variable *sps* fija (con valor 2, tanto para OOK como para 4-ASK), la cual corresponde a las muestras por símbolo. Es así como el valor de tasa de bit que el usuario fija, se relaciona con la frecuencia de muestreo para el caso de 4-ASK, y en el caso de la modulación OOK, corresponde con la mitad de la frecuencia de muestreo. De esta manera, la tasa de bit máxima permitida para la modulación OOK es de 4 Mbps mientras que para 4-ASK es 8 Mbps.

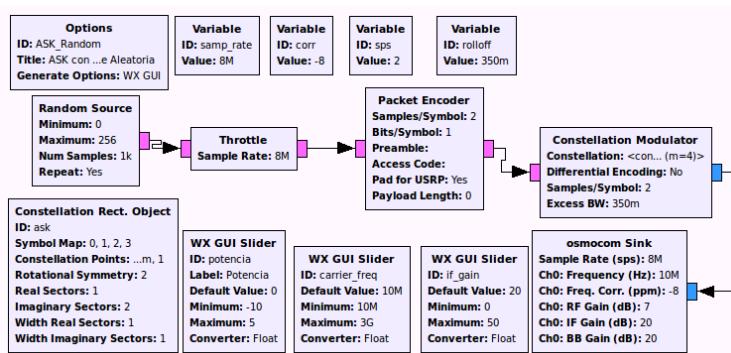


Figura 4.17: Diagrama en GNU Radio Companion para la generación de Señales 4-ASK con fuente aleatoria

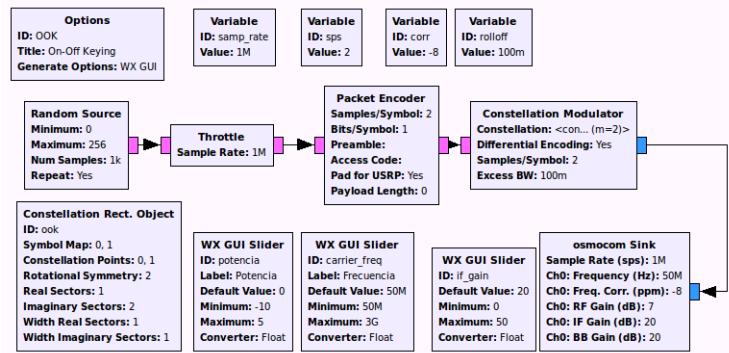


Figura 4.18: Diagrama en GNU Radio Companion para la generación de Señales OOK con fuente aleatoria

Las modulaciones digitales admiten un nuevo tipo de fuente, en caso de que se quiera enviar un archivo, por lo que el diagrama es similar al mostrado anteriormente con excepción de que la fuente aleatoria se sustituye por una fuente de archivo (ver Figura 4.19). No obstante, si el usuario desea desechar una señal de audio, se sustituye a la fuente aleatoria y el *Throttle* por la fuente de audio (ver Figura 4.20). Cabe señalar que es posible incorporar una fuente UDP, la cual permite configurar una transmisión de archivos mediante *streaming* desde un programa que permita configurar una transmisión de datos a través de un puerto UDP; en dicho bloque de fuente se configura el puerto que se desea utilizar además de la dirección IP seleccionada (Figura 4.21).

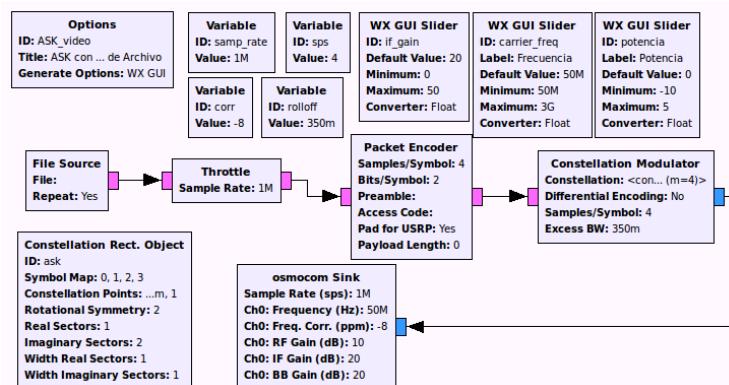


Figura 4.19: Diagrama en GNU Radio Companion para la generación de Señales ASK con fuente de archivo

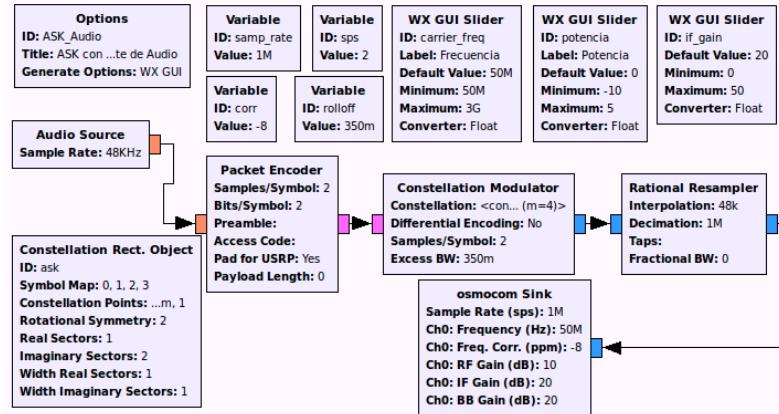


Figura 4.20: Diagrama en GNU Radio Companion para la generación de Señales ASK con fuente de audio

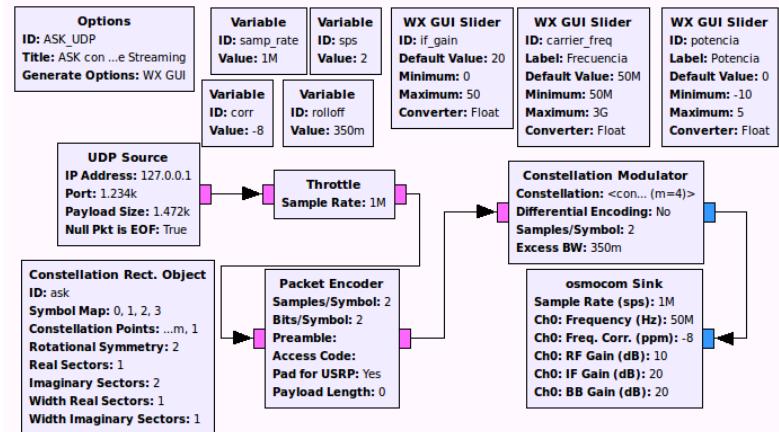


Figura 4.21: Diagrama en GNU Radio Companion para la generación de Señales ASK con fuente UDP

En las Figuras 4.22a y 4.22b se observan los espectros de señales OOK y 4-ASK con una tasa de bit de 1 Mbps. De acuerdo a la ecuación 2.10, el ancho de banda para estas señales es 1.350 MHz y 675 kHz respectivamente, lo que se aproxima mucho a los resultados obtenidos donde se aprecia un ancho de banda de 1.353 MHz y 676 kHz.

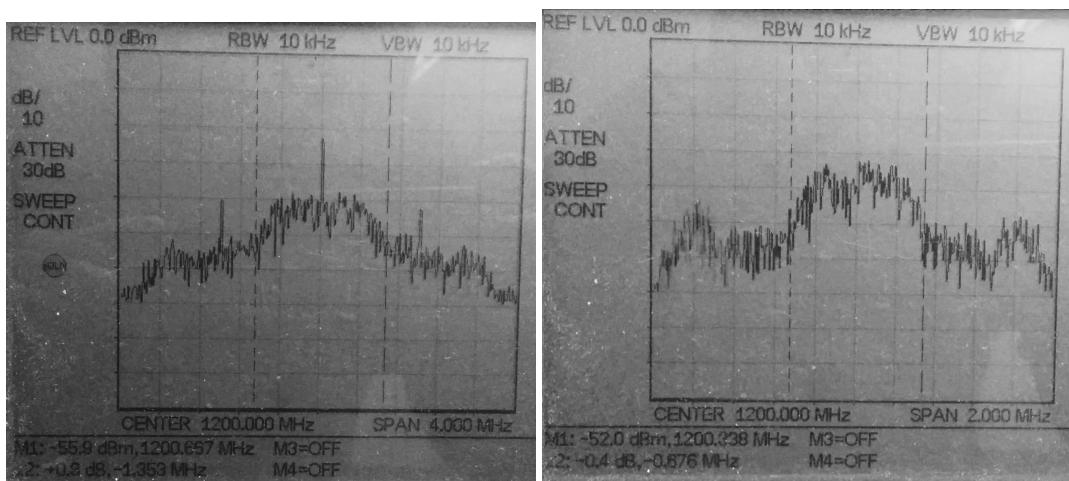


Figura 4.22: Espectro de señales ASK con tasa de bit igual a 1 Mbps.

4.2.2.2. Modulación por Desviación de Fase (PSK) y Modulación de Amplitud en Cuadratura (QAM)

Estos esquemas de modulación, al igual que el resto de los digitales, guardan una relación en cuanto a la estructura de los diagramas, es decir, se basan en una fuente conectada a un codificador de paquetes cuya salida va al bloque modulador para finalmente llegar al bloque que permite enviar la señal al transceptor.

Los diagramas para ambos esquemas de modulación son similares que para el caso de ASK, siendo una de las variaciones el bloque modulador ajustado para esquema por lo que se desecha el «*Constellation Rect. Object*»; otra variante es la presencia de una nueva variable, `number_constellation`, que corresponde al número de constelación la cual el usuario podrá modificar utilizando valores múltiples de 2 (hasta 32) para PSK, y múltiplos de 4 (hasta 64) para QAM. En este sentido, de acuerdo a los valores de esta variable se asignará automáticamente el valor de la variable `sps` de la siguiente manera: 2 para BPSK, 4 para QAM y 4-PSK, 3 para 8-PSK, 4 para 16-QAM y 16-PSK, 5 para 32-PSK y 6 para 64-QAM, de forma que pueda ajustarse el ancho de banda de la señal en concordancia con sus valores teóricos. En base a esto, la tasa de bit seleccionada por el usuario corresponderá con la

frecuencia de muestreo, alcanzando un máximo de tasa de bit de 8 Mbps, a excepción del caso de BPSK, donde la tasa de bit corresponderá con dos veces la tasa de muestreo, logrando un máximo de 4 Mbps.

Como se ha mencionado antes, los esquemas de modulación digital admiten 4 tipos de fuentes distintas y en las Figura 4.23, 4.24, 4.25 y 4.26 se muestran los diagramas para PSK/QAM con sus distintos tipos de fuente.

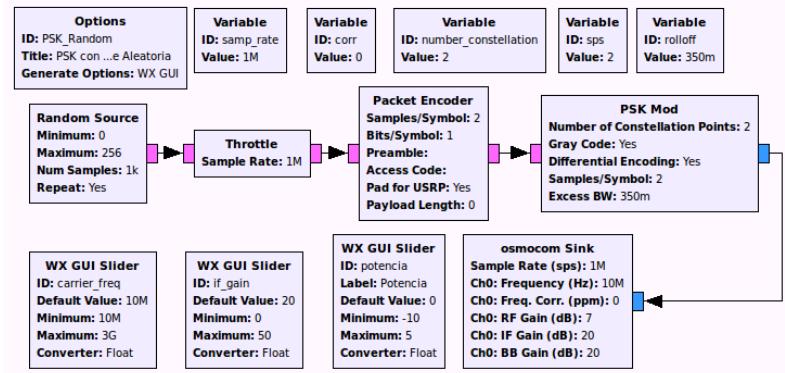


Figura 4.23: Diagrama en GNU Radio Companion para la generación de Señales PSK con fuente de aleatoriedad

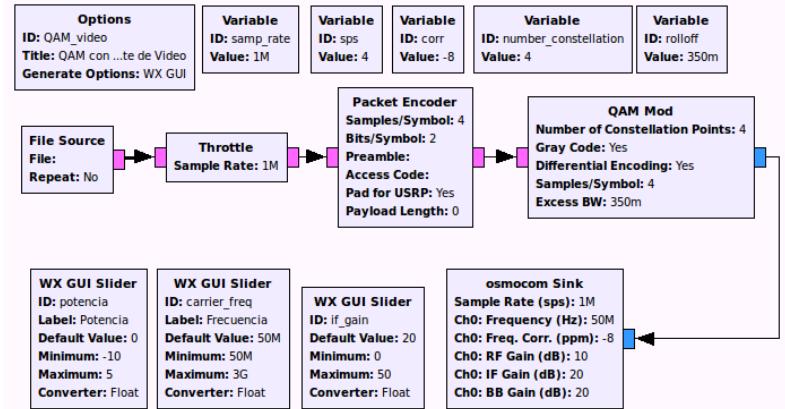


Figura 4.24: Diagrama en GNU Radio Companion para la generación de Señales ASK con fuente de archivo

En la Figura 4.27 se puede ver el espectro de señales 2-PSK y 4-PSK con una tasa de bit 1 Mbps y un factor de roll-off de 0.35, y de acuerdo a la ecuación 2.31, el ancho de banda para señales de estas características debería ser 1.35 MHz y 675 kHz

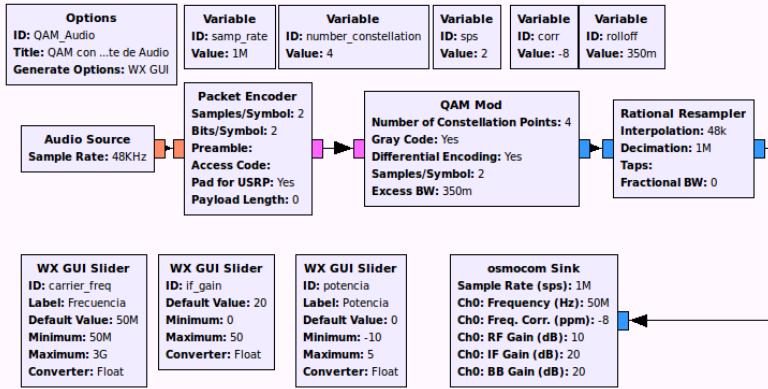


Figura 4.25: Diagrama en GNU Radio Companion para la generación de Señales ASK con fuente de audio

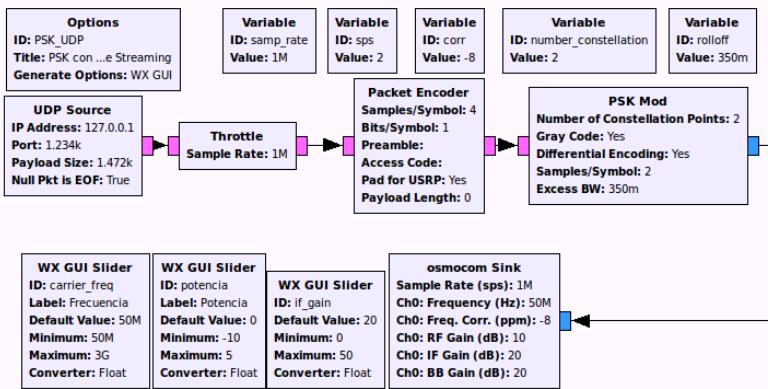


Figura 4.26: Diagrama en GNU Radio Companion para la generación de Señales ASK con fuente UDP

respectivamente lo que coincide con los resultados de las Figuras 4.27a y 4.27b. Lo propio pasa para el caso de las Figura 4.28 donde puede verse el espectro de señales 32-PSK y 64-QAM con una tasa de bit fija de 500 kbps y un factor de roll-off de 0.35 cuyo ancho de banda teórico de acuerdo a las ecuaciones 2.31 y 2.8 es 135 kHz y 112.5 kHz, y nuevamente corresponde con los resultados de las Figuras 4.28a, y 4.28b.

Finalmente se muestra otro ejemplo en la Figura 4.29, donde en este caso se deja el número de constelación constante (8-PSK), y se varía la tasa de bit entre tres valores: 500 kbps, 1 Mbps y 2 Mbps. Para los 3 casos el factor de roll-off es de 0.35, lo que resultaría en anchos de banda teóricos de 225 kHz, 450kHz y 900 kHz.

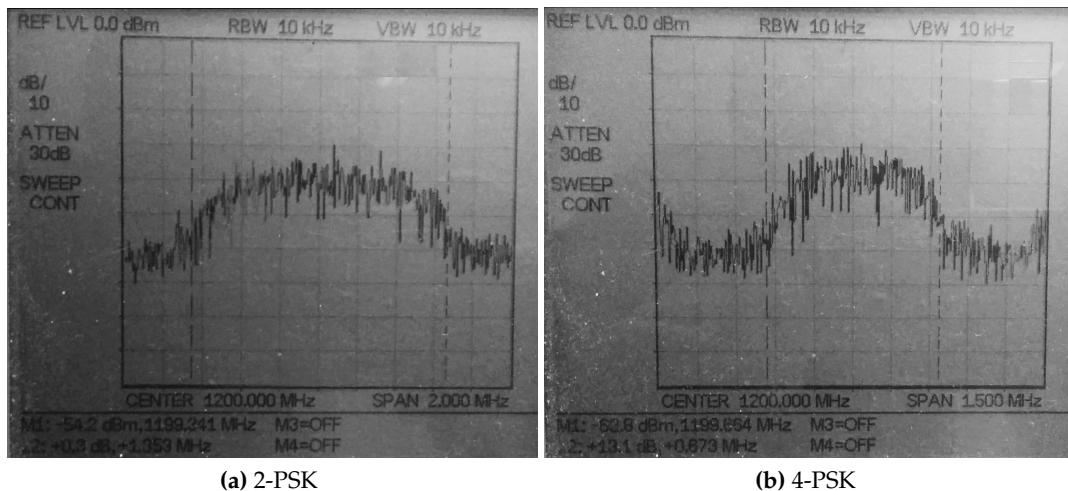


Figura 4.27: Espectro de señales PSK con distintos valores de constelación, tasa de bit igual a 1 Mbps y factor de roll-off igual a 0.35

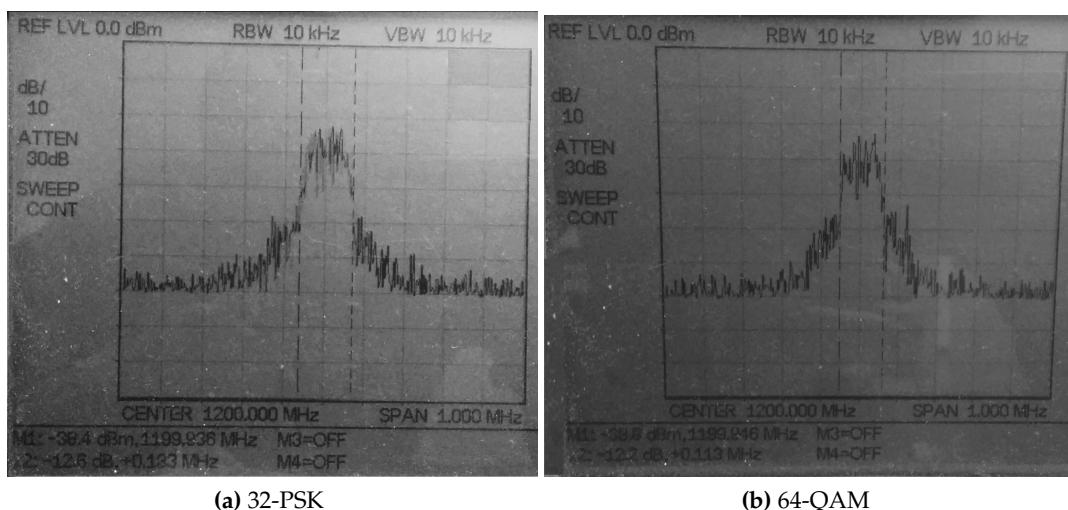


Figura 4.28: Espectro de señales PSK y QAM con distintos valores de constelación, tasa de bit igual a 500 kbps y factor de roll-off igual a 0.35

4.2.2.3. Modulación por desviación de Frecuencia Gaussiana (GFSK) y Modulación por Desviación Mínima Gaussiana (GMSK)

La estructura base de los diagramas para los esquemas de modulación GFSK y GMSK son similares a los demás esquemas de modulación digitales, diferenciándose del resto en los bloques moduladores, además de la sustituir la variable rolloff

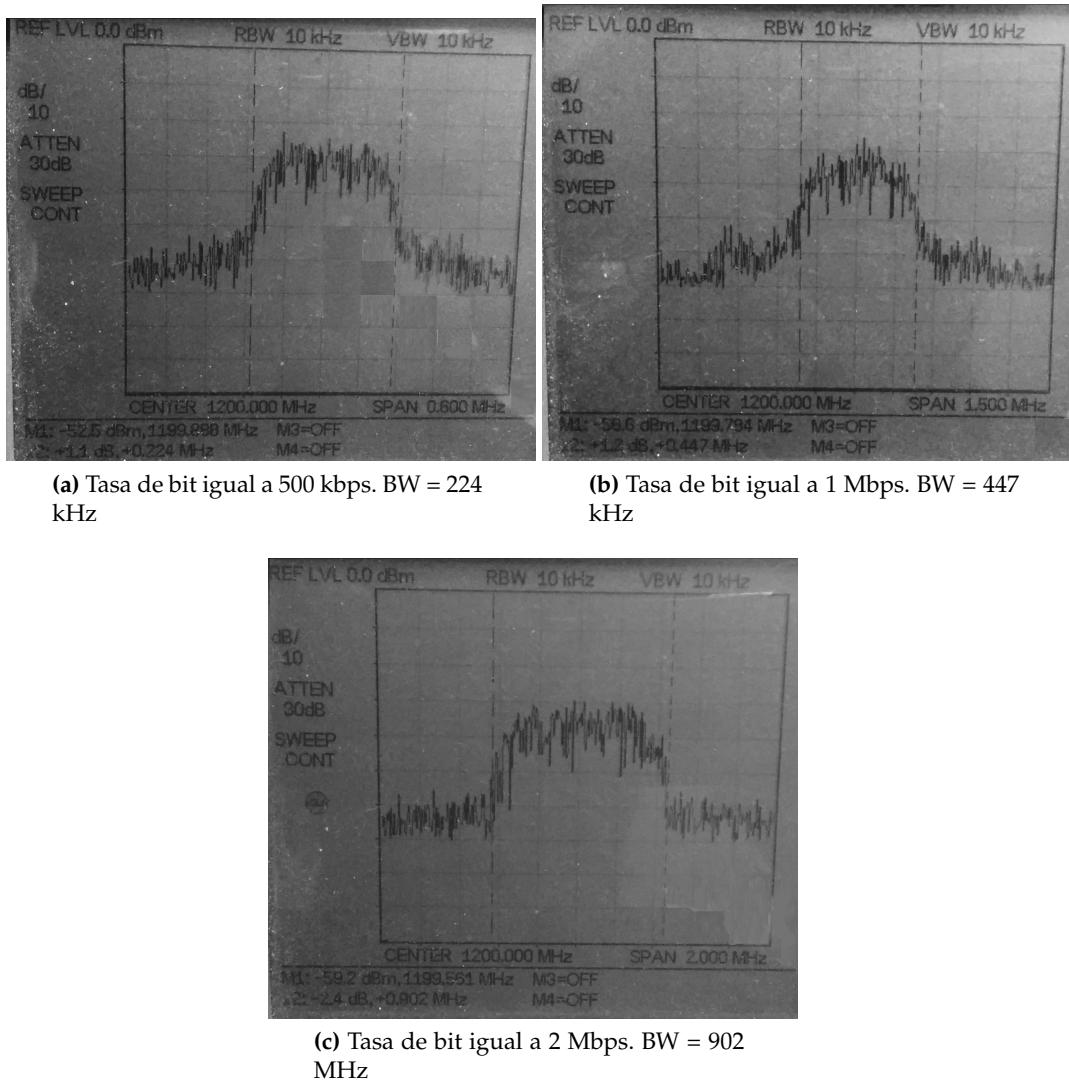


Figura 4.29: Espectro de señales 8-PSK con distintos valores de tasa de bit igual y factor de roll-off igual a 0.35

por la variable bt , que corresponde al factor BT del filtro gaussiano. La tasa de bit para estos diagramas corresponde a dos veces la frecuencia de muestreo, pudiendo alcanzar un máximo de tasa de bit de 8 Mbps, sujeto a la capacidad de procesamiento del computador que se utilice.

Como en el resto de los esquemas de modulación digital utilizados, GMSK y GFSK admiten el uso de fuente aleatoria, de audio, de archivo y UDP.

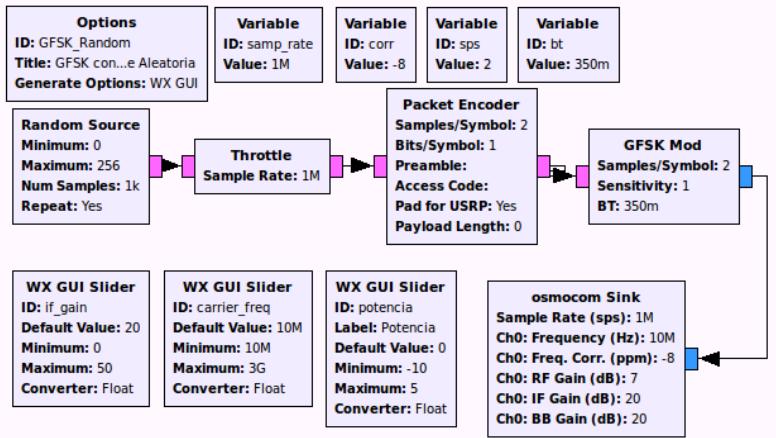


Figura 4.30: Diagrama en GNU Radio Companion para la generación de Señales GFSK con fuente aleatoria

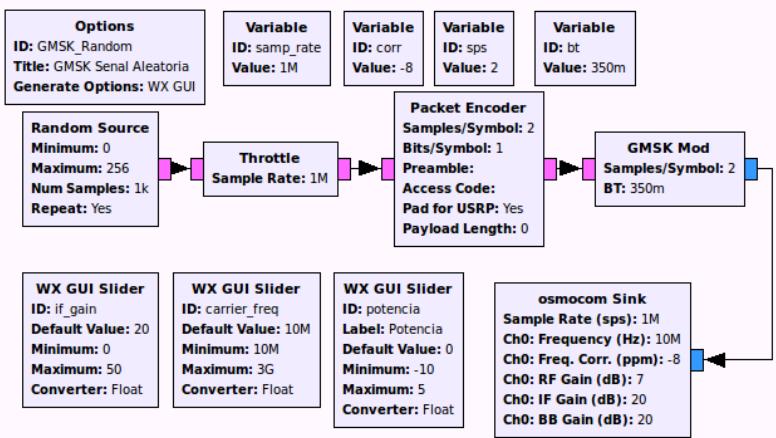


Figura 4.31: Diagrama en GNU Radio Companion para la generación de Señales GMSK con fuente aleatoria

En las Figuras 4.35a y 4.35b se observa el espectro de una señal GMSK con tasa de bit igual a 1 Mbps y producto BT = 0.35, cuyo ancho de banda teórico de -3dB es de 750 kHz y el ancho de banda del lóbulo principal es aproximadamente a 1.5 veces la tasa de bit. Adicionalmente en las Figuras 4.36a y 4.36b, se aprecia una señal GFSK con la misma tasa de bit y producto BT= 0.1 por lo que su ancho de banda de -3dB teórico es de 200kHz.

En el caso de que BT >0.35, el ancho de banda de -3dB deja de aproximarse

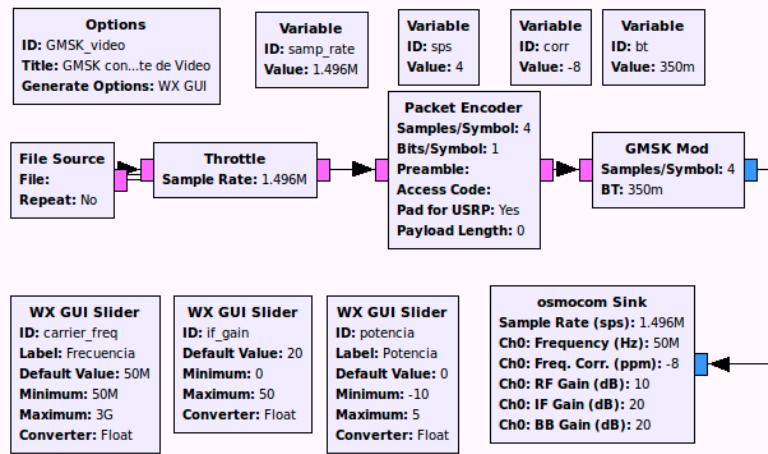


Figura 4.32: Diagrama en GNU Radio Companion para la generación de Señales GMSK con fuente de archivo

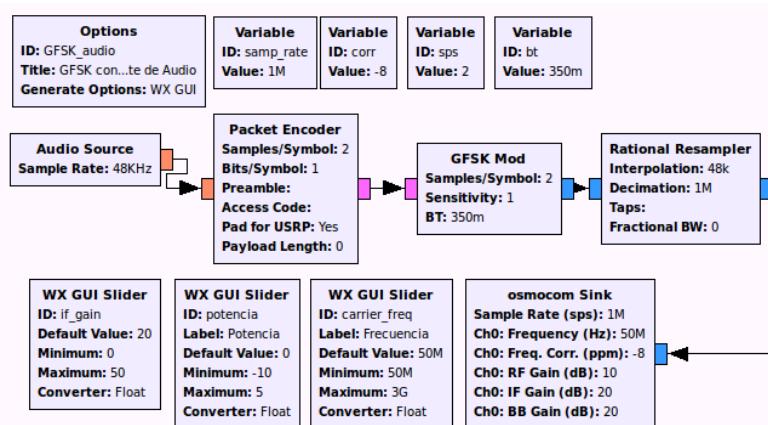


Figura 4.33: Diagrama en GNU Radio Companion para la generación de Señales GFSK con fuente de audio

de acuerdo a la ecuación, mientras que al ancho de banda del lóbulo principal se aproxima más a 1.5 veces la tasa de bit debido a que se aproxima más a una señal MSK. En la Figura 4.37 se aprecia una señal GMSK con producto BT = 0.9, tasa de bit 1 Mbps y ancho de banda de aproximada mente 1.5 MHz.

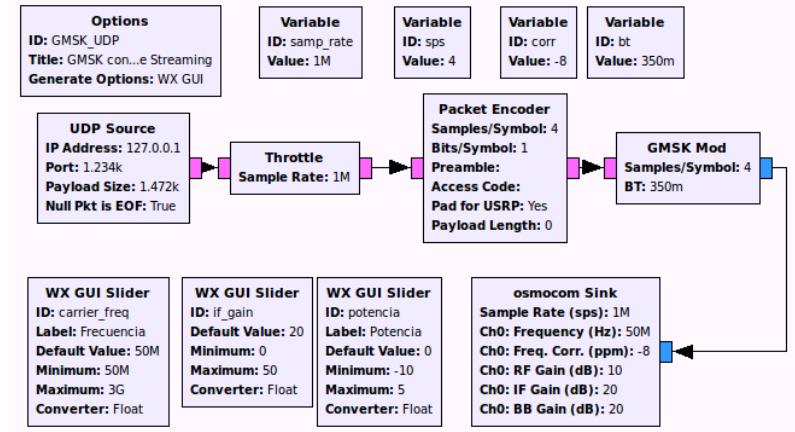


Figura 4.34: Diagrama en GNU Radio Companion para la generación de Señales GMSK con fuente UDP

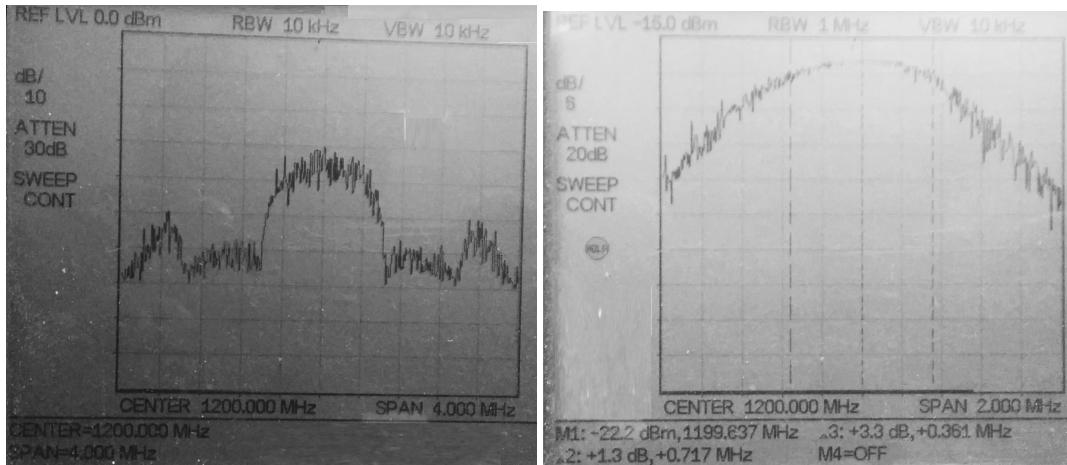
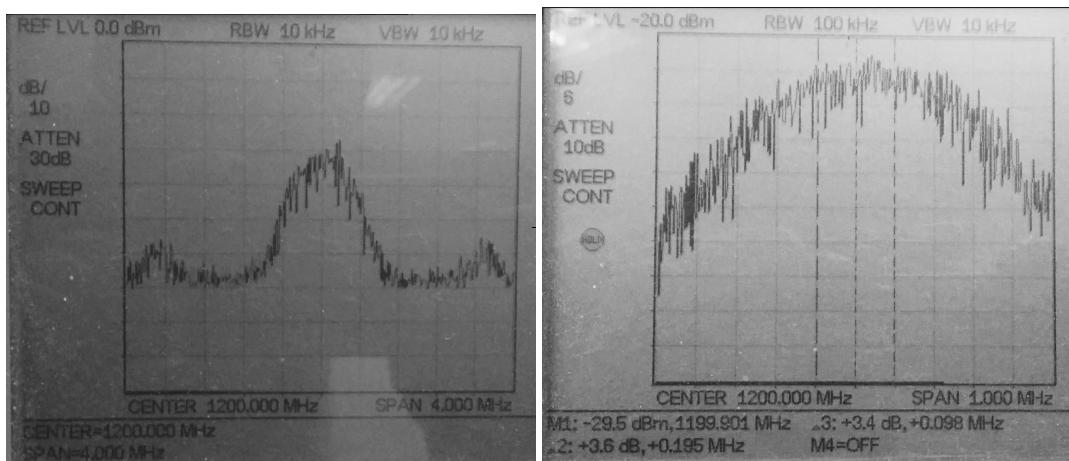


Figura 4.35: Espectro de señal GMSK para BT=0.35

4.2.3. Multiplexación por División de Frecuencias Ortogonales (OFDM)

El diagrama para OFDM es similar al de las modulaciones digitales como se aprecia en la Figura 2.26, consiste en un fuente aleatoria que va a un «Throttle» y de allí se convierte el tipo de dato a «float» ya que este es el tipo de dato que maneja el modulador. Luego, la salida del modulador se conecta finalmente al «Osmocon sink». Además de las variables de control de frecuencia de la portadora y de



(a) Espectro de señal GMSK con BT=0.1

(b) Ancho de banda de -3dB de señal GMSK con BT=0.1

Figura 4.36: Espectro de señal GFSK para BT=0.1

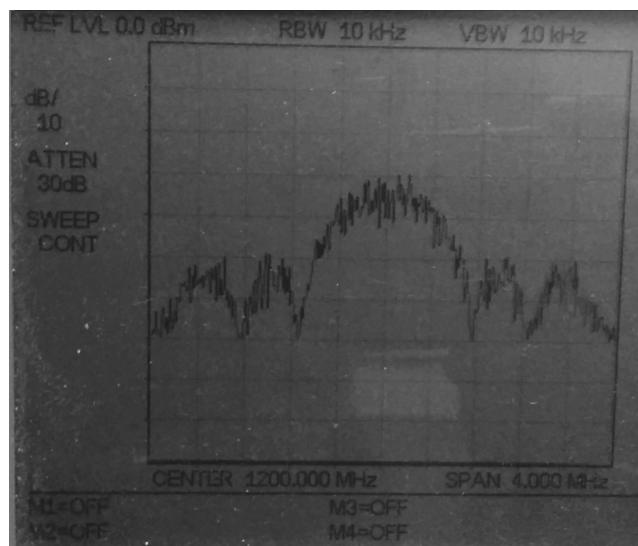


Figura 4.37: Espectro de señal GMSK con BT=0.9

potencia, el usuario cuenta con una serie de variables para manejar los distintos parámetros de la modulación OFDM; dichas variables son: `fftlen`, la cual establece la longitud de la FFT, `canales`, la cual establece el número de tonos o portadoras ocupadas, y `prefijo` que establece la longitud del prefijo cíclico. Adicionalmente el usuario podrá establecer el esquema de modulación deseado, ya que el bloque modulador ofrece los esquemas BPSK, QPSK, 8-PSK, 16-QAM y 64-QAM.

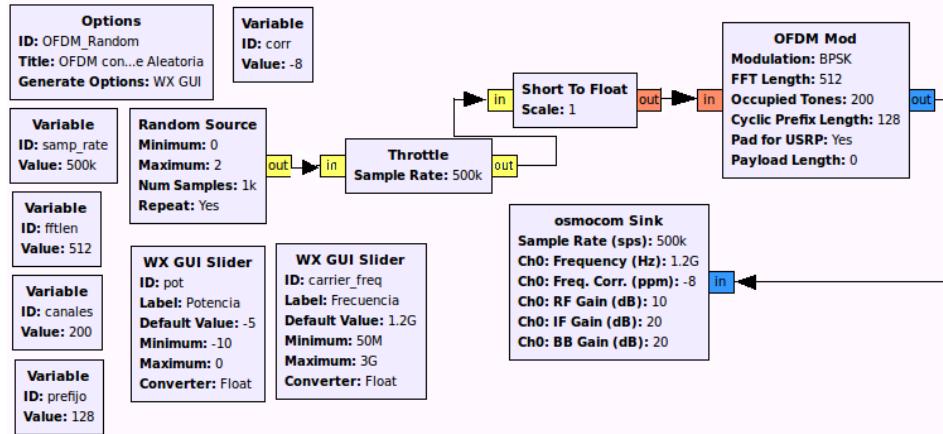


Figura 4.38: Diagrama en GNU Radio Companion para la Generación de una señal OFDM

En SDR, el ancho de banda para las señales OFDM dependerá directamente de la frecuencia de muestreo según la relación [62]:

$$BW = \frac{\text{SampRate} * \text{TonesOcupados}}{\text{FFT}} \quad (4.1)$$

De acuerdo a esto, se fijó un valor de 2Msps para la frecuencia de muestreo y el usuario podrá elegir entre una FFT de 64, 128, 256, 512, 1024, 1536 ó 2048.

En la Figura 4.39, se muestra el espectro de 2 señales OFDM con distinto tamaño de FFT y Tonos ocupados cuyo ancho de banda corresponde al calculado de acuerdo a la Ecuación 4.1. De acuerdo a estos resultados, se puede apreciar como el ancho de banda de una señal OFDM aumentará a medida que aumente el porcentaje de tonos ocupados respecto al tamaño de la FFT.

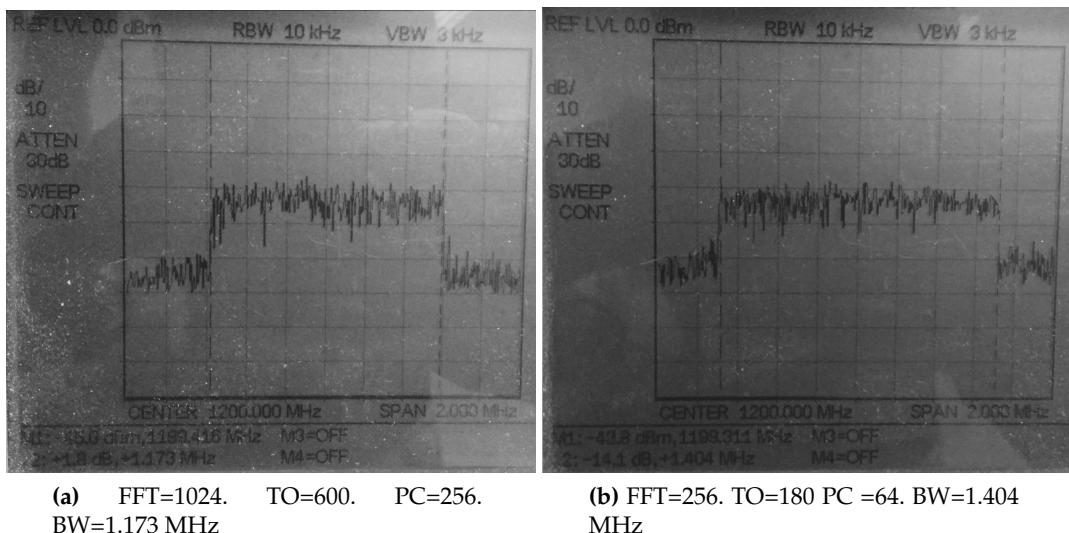


Figura 4.39: Espectro de señales OFDM para distintos valores de FFT, Tonos Ocupados y Prefijo Cíclico

4.2.4. Estándares Internacionales de Transmisión de Televisión

4.2.4.1. NTSC

La base del diagrama para la transmisión NTSC fue desarrollado por Clayton Smith [61] y fue ligeramente modificado para adaptarlo a las necesidades del generador vectorial. La señal transmitida consiste en una imagen la cual al ser seleccionada por el usuario, es codificada a través del programa `ntsc-encode.py`, el cual convierte la imagen en un archivo de datos con la información correspondiente que debe ocupar cada línea en la pantalla de un televisor, en otras palabras, el archivo tendrá la información de las señales de crominancia y luminancia.

Luego, dicho archivo de datos es compilado y leído a través de una fuente de archivo cuya salida se multiplica por una constante con el fin de escalarla y prevenir la saturación del conversor D/A, y de ahí pasa por un filtro que corresponde a la ventana de 6 MHz del canal. Cabe señalar que la salida de este filtro se suma a una señal senoidal que representará la portadora de sonido, ubicada a 4.5 MHz de la portadora de video. Esto solo se hace a modo de apreciar la portadora de sonido

en el espectro del canal, ya que en la realidad, no fue posible transmitir audio debido a que éste debe modularse previamente en FM. Para esto el cociente entre la frecuencia de muestreo de salida del bloque modulador FM (*Quadrature Rate*) y la frecuencia de muestreo de entrada (*Audio Rate*), debe ser un número entero.

Por lo tanto, debido a que la frecuencia de muestreo del diagrama es un valor no racional que depende de las muestras por línea las cuales a su vez están ligadas al programa encargado de codificar la imagen, no existe un valor que pueda ajustarse. Adicionalmente, a pesar de que existen fracciones que provean una relación racional que al multiplicarla por el valor del *Quadrature Rate*, resulta en un valor aproximado al de la frecuencia de muestreo, al no haber una adaptación exacta de estos valores, no se podrá transmitir la señal sin errores, por lo que el uso de un remuestreador racional no es suficiente. Igualmente, tampoco existe la posibilidad de usar un factor de decimación igual a la frecuencia de muestreo a la salida del modulador y un factor de interpolación igual a la frecuencia de muestreo del diagrama ya que la cantidad de recursos del computador necesarios para efectuar operaciones de esa magnitud son demasiado para computadores convencionales.

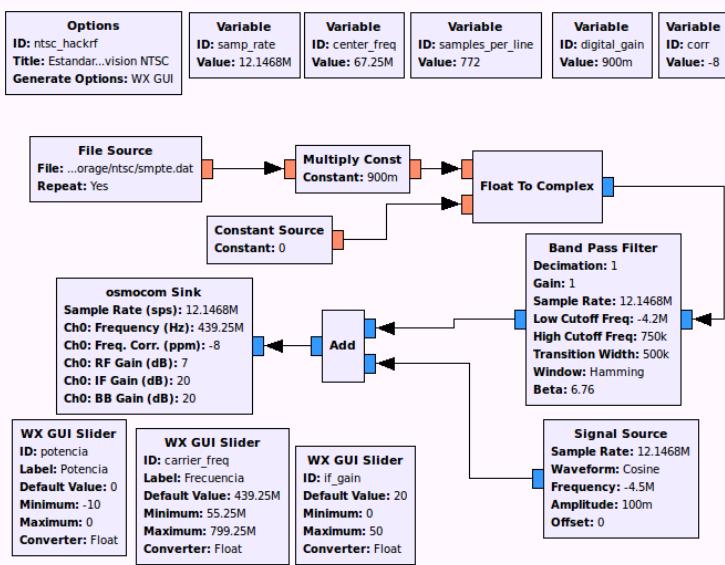


Figura 4.40: Diagrama en GNU Radio Companion para la transmisión de una señal NTSC

Por consiguiente, el usuario puede transmitir la señal NTSC a las distintas frecuencias de los canales comprendidos entre 55.25 MHz y 799.25 MHz. En la Figura 4.41 se observa el espectro de un señal NTSC a 439.25 MHz, comparándola con el espectro de una señal de estas características presentado en la Figura 2.31, se observa que la portadora de sonido efectivamente está a 4.5 MHz de la portadora de video. Adicionalmente a 3.5 MHz de dicha portadora de video se aprecian las señales de crominancia y luminancia, y en general, un ancho de banda de aproximadamente 6 MHz.

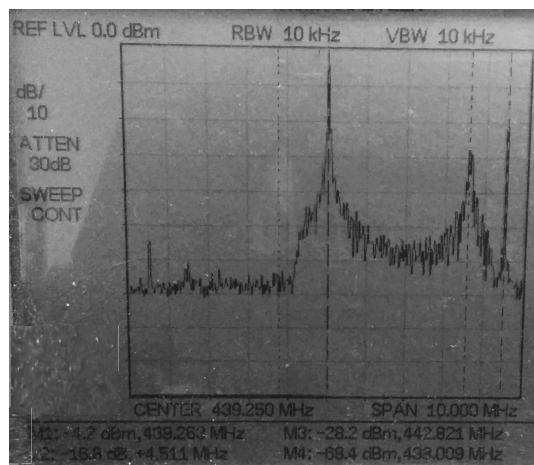


Figura 4.41: Espectro de una señal NTSC a 439.25 MHz

4.2.4.2. DVB-T

El diagrama para la trasnmisión DVB-T se construye a partir de los bloques de transmisión disponibles en la librería *dvbt*, que en esencia, es una modulación OFDM con una codificación previa (COFDM). En la Figura 4.42 puede verse el dia-grama, que inicia con una fuente de archivo, pero es importante destacar que estos archivos deben ser flujos de transporte MPEG-2. Luego, la salida de la fuente va al bloque de «energy_dispersal» el cual se encarga de producir un flujo de datos con la densidad espectral de potencia lo más uniformemente posible. Posteriormente sigue una etapa constituida por el proceso de codificación para protección ante errores, constituido por codificación Reed Solomon y convolucional ademá-s

de intercalación interna, debido a que se trata del estándar DVB-T. Tras la etapa de codificación, se realiza el mapeo y la modulación de las sub-portadoras tomando la IFFT para finalmente añadirle el prefijo cíclico correspondiente.

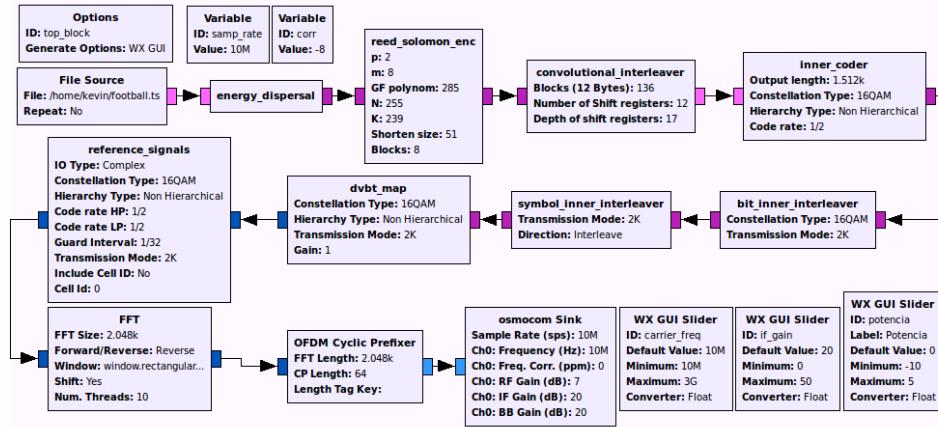


Figura 4.42: Diagrama en GNU Radio Companion para la transmisión de una señal DVB-T

Adicionalmente, el usuario podrá seleccionar el esquema de modulación que desee además de que también podrá variar la tasa de codificación. En la Figura 4.43 se aprecia el espectro de un canal DVB-T de 8 MHz de ancho de banda.

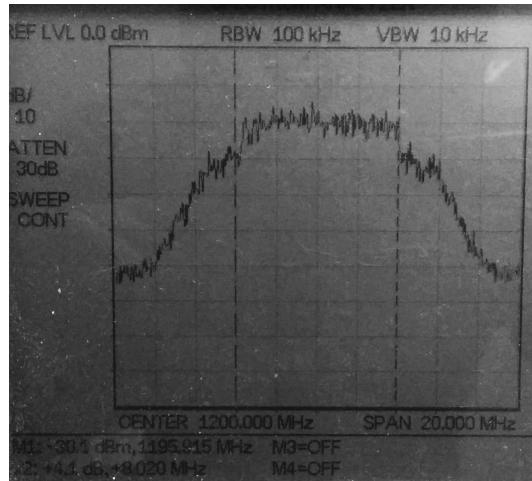


Figura 4.43: Espectro de un canal DVB-T de 8 MHz

4.2.4.3. DVB-S2

La estructura del diagrama para la generación de señales DVB-S2 es similar a la de las señales DVB-T; se construye utilizando todos los bloques de la librería *dvbs2*. En la Figura 4.44 puede verse que la fuente es una fuente de archivo donde el mismo debe ser de flujos de transportes MPEG-2. La etapa de codificación se divide en dos partes, la de codificación BCH y la de codificación LDPC, ambas propias del estándar DVB-S2. Seguidamente, se tiene la etapa de modulación OFDM y finalmente la información pasa por un filtro que establece el ancho de banda del canal de 6 MHz.

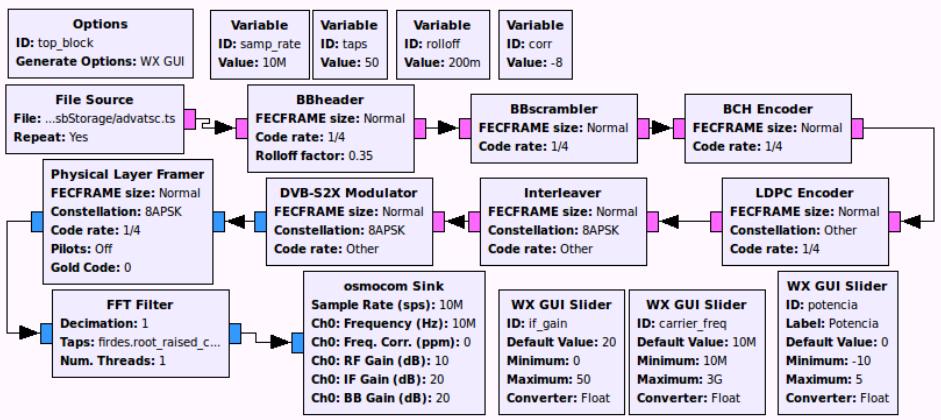


Figura 4.44: Diagrama en GNU Radio Companion para la transmisión de una señal DVB-S2

Al igual que en el caso de DVB-T, el usuario podrá modificar el esquema de modulación (QPSK, 8PSK, 8APSK, 16APSK, 32APSK) y la tasa de codificación (1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 5/6).

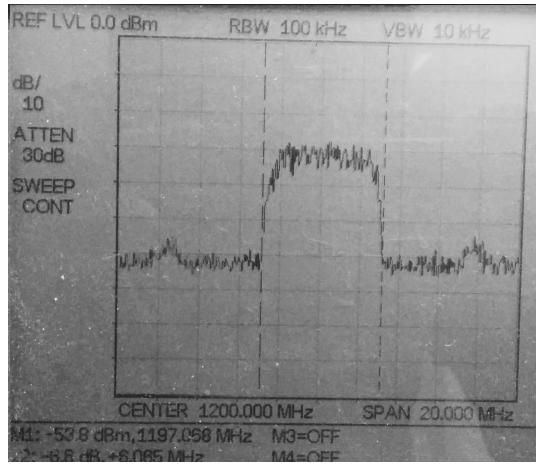


Figura 4.45: Espectro de un canal DVB-S2 de 6 MHz

4.3. Calibración

4.3.1. En Frecuencia

Tras caracterizar el dispositivo en frecuencia, se obtuvo que a medida que aumenta la frecuencia de la portadora la desviación medida con respecto a ésta aumenta de forma progresiva como se aprecia en la Figura 4.46.

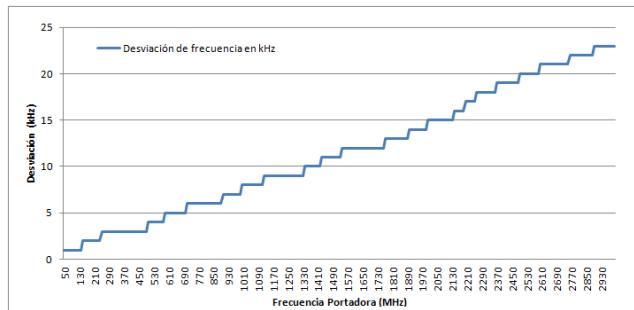


Figura 4.46: Desviación de Frecuencia Portadora

Para corregir esta desviación, se incluyó la variable `corr` en todos los diagramas construidos en GNU Radio Companion, la cual asignará un valor acorde al factor de corrección de frecuencia portadora (en ppm) que existe en el bloque «*Osmocon Sink*». Dicho factor de corrección calculado en ppm sigue un comportamiento como

el de la Figura 4.47. Dicha calibración se hizo en pasos de 10 MHz y para los valores intermedios se utilizó una función de interpolación. Es importante destacar que dicha desviación no es siempre la misma, ya que aumenta con el tiempo de uso debido al incremento de la temperatura del dispositivo; sin embargo, ésta no es predecible, pues puede llegar hasta un máximo de 27 kHz sobre el final del rango de frecuencia de operación.

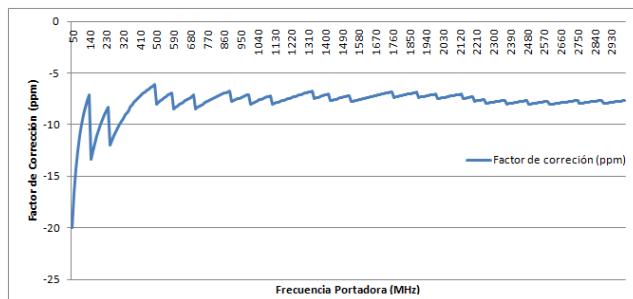


Figura 4.47: Factor de Corrección de Frecuencia Portadora

4.3.2. En Potencia

La caracterización del dispositivo en potencia se hizo enviando un pulso constante y midiendo la potencia de salida en pasos de 10 MHz con los valores de ganancia del transceptor siempre constantes. Los resultados se vueden ver en la Figura 4.48, donde se evidencia un comportamiento irregular en el rango comprendido entre 50 MHz y 3 GHz, lo que se deriva en la necesidad de usar distintos ajustes de ganancia dependiendo de la frecuencia, con el fin de obtener una potencia de salida constante a lo largo del rango. Adicionalmente, se observan cuatro valores en los cuales el transceptor presenta errores de distorsión y atenuación: 920 MHz, 2150 MHz, 2310 MHz y 2750 MHz. Así mismo, se determinó que en los rangos de 916-928 MHz y 2306-2318 MHz, el transceptor presenta una severa distorsión de la señal a la salida por fallas intrínsecas del dispositivo, cuyo origen no puede ser explicado basado en las especificaciones técnicas del mismo, por lo que se infiere que dicho error es una falla de fábrica; por su parte, en los rangos de 2149-2157 MHz y 2749-2760 MHz, el transceptor presenta una fuerte atenuación de la señal de salida; de acuerdo las especificaciones de la HackRF One, a 2150 MHz y 2750 MHz son

valores en los que el transceptor comuta a una nueva etapa de ganancia. Debido a todo esto, se determinó que los 4 rangos mencionados anteriormente, no serán utilizables en el prototipo del generador.

Al igual que en el caso de la frecuencia, el tiempo de operación también influye sobre la potencia de salida, ya que ésta disminuye hasta cierto punto luego de que el transceptor opere por un período de tiempo de aproximadamente 30 minutos, adicionalmente es importante destacar que éste comportamiento en potencia no es enteramente predecible y puede variar hasta 0.3 dB de un uso a otro bajo las mismas condiciones.

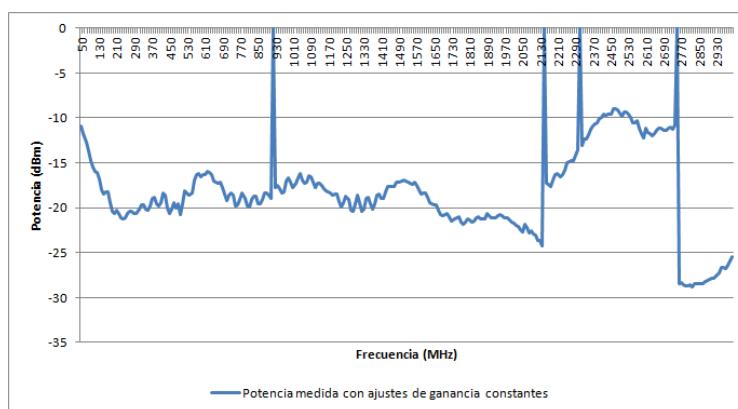


Figura 4.48: Potencia de Salida con Ajustes de Ganancia Constante

Por otro lado, en la Figura 4.48 también se observa que el rango entre 2157 MHz y 2749 MHz presenta la mayor potencia de salida. De la misma manera puede apreciarse que entre 2760 MHz y 3000 MHz, se obtiene la menor potencia de salida. En este sentido, se pudo derivar el rango de frecuencia para el cual el prototipo puede operar en la mayoría de los esquemas. Siendo estos:

- -10 dBm — 0 dBm entre 50MHz y 2749 MHz
- -10 dBm — 5 dBm entre 2157 MHz y 2749 MHz
- -10 dBm — -5 dBm entre 2760 MHz y 3000 MHz

4.3.2.1. Ajuste por Esquema

Manteniendo el amplificador RF siempre en su estado de encendido, se establecieron ajustes de ganancia IF con el fin de lograr un valor de potencia constante a la salida. Esto se hizo para 4 valores: -10 dBm, -5 dBm, 0 dBm y 5 dBm, los resultados de estas señales de referencia pueden verse en la Figura 4.49. En este sentido, se midió la potencia de salida de la señal para cada esquema de modulación utilizando los diferentes valores de ganancia IF establecidos anteriormente, esto se realizó con la finalidad de determinar el factor de ajuste para cada esquema respecto de la referencia medida. Es necesario acotar se cuenta con una presión de ± 1 dB, y además la potencia de salida para la mayoría de los esquemas oscilará en torno a ± 0.3 dB con respecto al valor de potencia de canal que se mida.

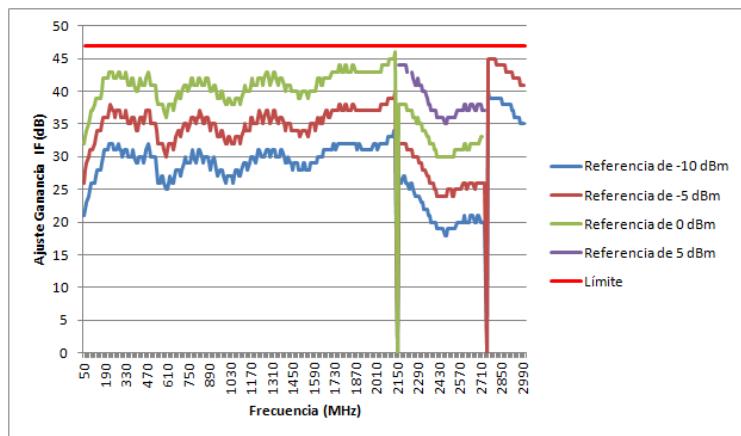


Figura 4.49: Ajustes de Ganancia IF para distintos niveles de potencia de salida a lo largo de la potencia

Por otro lado, en la Figura 4.50 se aprecia que la potencia de portadora de una señal AM está 6dB por debajo de la potencia de salida de señal de referencia para los cuatro niveles tomados. Sin embargo, no es posible alcanzar la potencia de salida establecida en los rangos expuestos al final de la sección 4.3.2, para los niveles de potencia más altos. Esto se debe a que el amplificador de ganancia IF cuenta con pasos de 1 dB hasta 47dB, y en vista a esto, no es posible sumar 6 dB a los valores de ganancia IF de referencia. Por ésta razón, este esquema en particular está limitado hasta -5 dBm de potencia de señal portadora hasta 2.749 GHz, ya que los ajustes

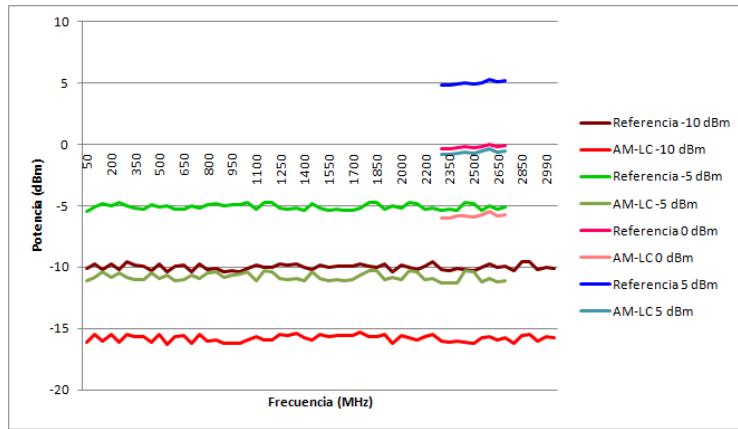


Figura 4.50: Potencia de Salida de Señal AM con distintos ajustes de ganancia IF

de ganancia IF de referencia para el rango comprendido entre 2.76 GHz y 3 GHz exceden los 41 dB, por lo que un ajuste de 6 dB excedería el límite de ganancia de 47 dB del amplificador. Es así como por éste mismo motivo, se pueden alcanzar hasta 5 dBm solo entre 2318 MHz hasta 2749 MHz.

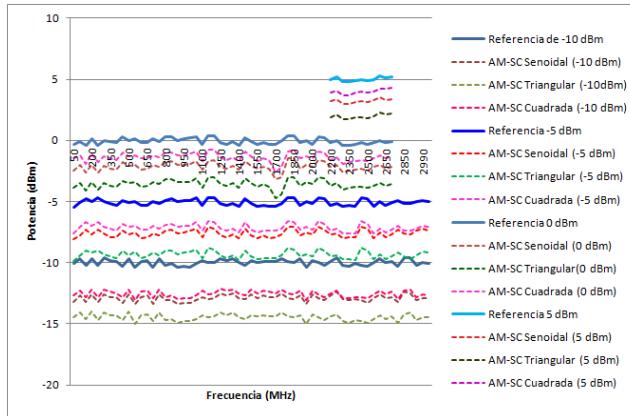


Figura 4.51: Potencia de Salida de Señal AM-SC con distintos ajustes de ganancia IF

Así mismo, en la Figura 4.51 puede observarse la potencia de salida de señales AM-SC con distintas formas de onda de señal modulante para los distintos ajustes de ganancia IF, los cuales dependen del tipo de señal y rango de potencia a utilizar. Al igual que para la modulación AM-LC, el caso de portadora suprimida tiene ciertas limitaciones de potencia dependiendo del tipo de forma de onda que se utilice

debido a las limitantes de ganancia del amplificador IF. En este sentido, cuando la señal modulante tiene una forma de onda triangular o diente de sierra, entre 2760 MHz y 2870 MHz esta no alcanzará el máximo de -5dBm. Por otro lado, entre 1760 MHz y 2140 MHz no se podrá alcanzar el máximo de 0 dBm; no obstante, entre 2160 MHz y 2220 MHz no se alcanzará la potencia máxima de 5 dBm. Así mismo, se debe señalar que si la señal modulante es una senoidal, la potencia estará limitada hasta -6 dBm entre 2760 MHz y 2810 MHz.

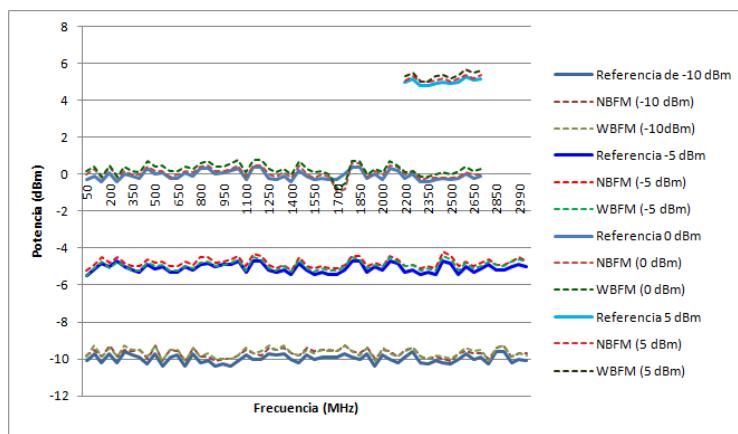


Figura 4.52: Potencia de Salida de Señales FM con distintos ajustes de ganancia IF

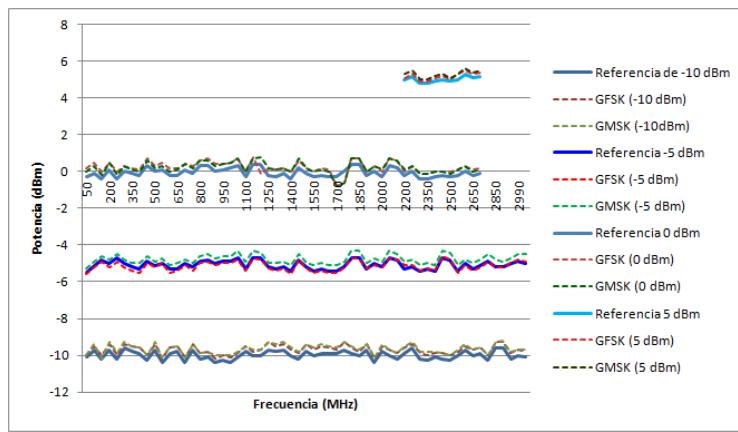


Figura 4.53: Potencia de Salida de Señales GMSK y GFSK con distintos ajustes de ganancia IF

Por otro lado, en el caso de las modulaciones NBFM, WBFM, GMSK y GFSK, no fue necesario un ajuste de ganancia, puesto que la potencia de canal de ambas

señales coincide con la potencia de la señal de referencia como se observa en las Figuras 4.52 y 4.53. No obstante, para los caso de las modulaciones OOK, 4-ASK, QAM y PSK, el ajuste dependerá del número de constelación y el rango de potencia en el que se desee trabajar como se observa en las Figuras 4.54, 4.55 y 4.56.

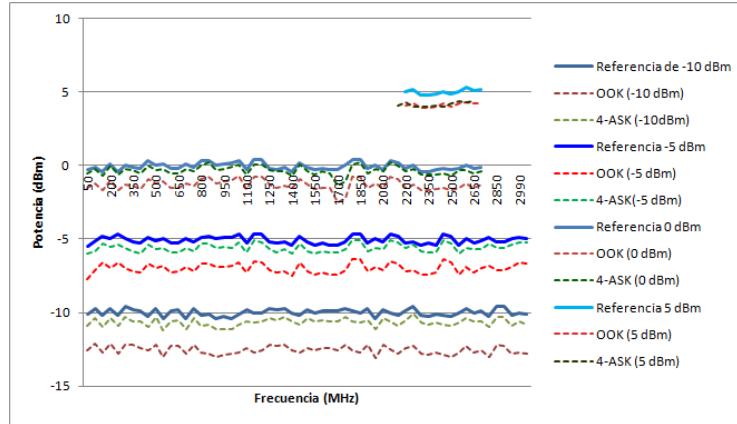


Figura 4.54: Potencia de Salida de Señales OOK y 4-ASK con distintos ajustes de ganancia IF

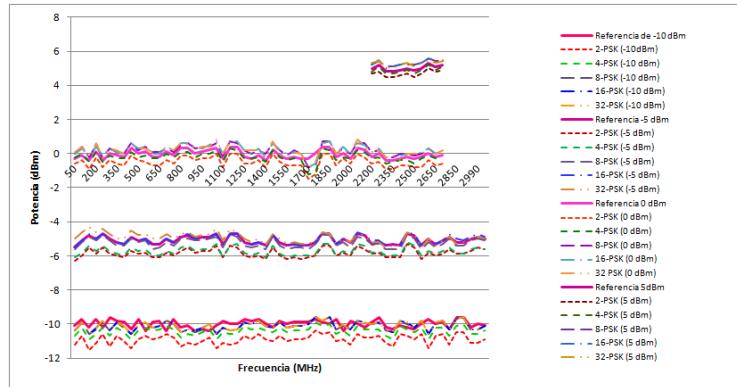


Figura 4.55: Potencia de Salida de Señales PSK con distintos números de constelación y ajustes de ganancia IF

Para la calibración del esquema OFDM, primeramente se siguió el mismo procedimiento que para el resto de los esquemas anteriores. Para esto, se fijó previamente un número de FFT (512) y un número de tonos ocupados (300), lo que equivale a 58,59 % de los tonos disponibles, y así determinó que la variación con respecto a la

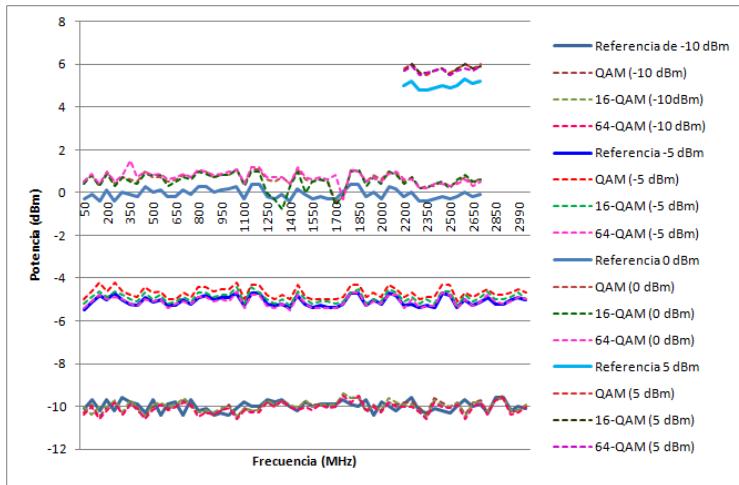


Figura 4.56: Potencia de Salida de Señales QAM con distintos números de constelación y ajustes de ganancia IF

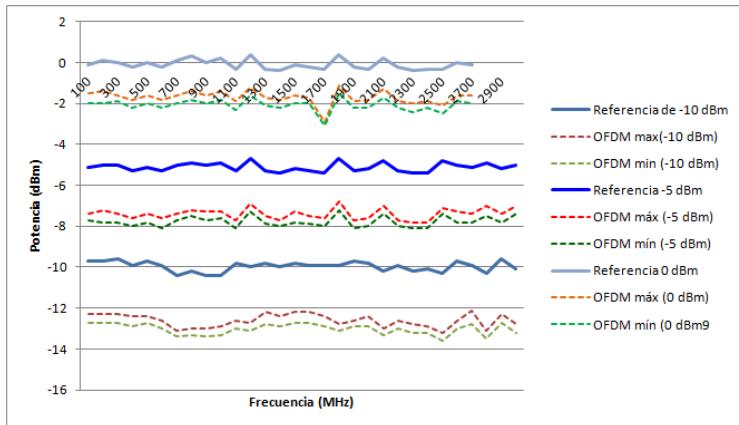


Figura 4.57: Potencia de Salida de Señales OFDM con distintos ajustes de ganancia IF

señal de referencia para cada uno de los niveles es constante como puede verse en la Figura 4.57.

Luego de esto, se seleccionó una frecuencia fija y se determinó la variación de potencia con respecto al porcentaje de tonos ocupados ajustando este valor entre 10 % hasta el 100 %, con el fin de obtener el ajuste definitivo de acuerdo al porcentaje de tonos ocupados como se observa en la tabla 4.1.

Tabla 4.1: Variación de Potencia de Canal en Señales OFDM según el Porcentaje de Tonos Ocupados y Ajuste de Ganancia IF Correspondiente

%TO	-10 dBm			-5 dBm			0 dBm		
	Máx (dBm)	Mín (dBm)	Ajuste (dB)	Máx (dBm)	Mín (dBm)	Ajuste (dB)	Máx (dBm)	Mín (dBm)	Ajuste (dB)
100	-11	-11,6	1	-6,1	-6,6	1	-0,4	-1	1
90	-11	-11,9	1	-6,1	-6,9	1	-0,6	-1,2	1
80	-11,3	-12,1	1	-6,5	-7,1	1	-0,7	-1,5	1
70	-11,9	-12,6	2	-6,9	-7,7	2	-1,2	-2	2
60	-12,2	-12,7	2	-7,3	-7,8	2	-1,6	-2	2
50	-12,8	-13,3	3	-7,9	-8,5	3	-2	-2,7	3
40	-13,6	-14,2	4	-8,7	-9,9	4	-2,9	-3,9	3
30	-14,7	-15,3	5	-9,8	-10,5	5	-3,9	-4,6	4
20	-16,5	-17,3	6	-11,8	-12,6	6	-5,9	-6,5	6
10	-19,7	-20,3	10	-15	-15,9	10	-9,4	-10,1	10

Cabe señalar que debido a las limitaciones de ganancia del amplificador IF de la HackRF One se estableció en 20 % el límite mínimo de número de tonos. Adicionalmente, debido al alto factor de cresta de las señales OFDM, el generador operará por debajo del rango de 0 dBm. De esta manera el usuario podrá enviar señales OFDM con un porcentaje de tonos ocupados del 20 al 100 % y estará limitado a niveles de potencia entre -10 dBm y -4 dBm para el rango de frecuencia de 50MHz a 2749 MHz, y -10 dBm entre 2760 MHz y 3000 MHz.

Para el caso del estándar DVB-T, el ajuste de ganancia IF es negativo ya que la potencia de canal de la señal está por encima de nivel de potencia de la señal de referencia como se ve el Figura 4.58. Esta señal no está disponible para el rango de 5 dBm debido que al ser una señal que utiliza modulación OFDM posee un factor de cresta elevado, y el amplificador IF del transceptor HackRF garantiza un máximo de potencia de canal para señales OFDM de hasta 0 dBm sin saturación.

Ahora bien, para el caso de los estándares DVB-S2 y NTSC, se siguió un procedimiento distinto ya que el rango de frecuencia es diferentes para estos esquemas. En el caso de DVB-S2, se decidió trabajar en el rango de frecuencias de la Banda L comprendido entre 1 GHz y 2 GHz, y se seleccionaron los valores para la ganancia IF directamente sin usar una señal de referencia en pasos de 10 MHz; sin embargo,

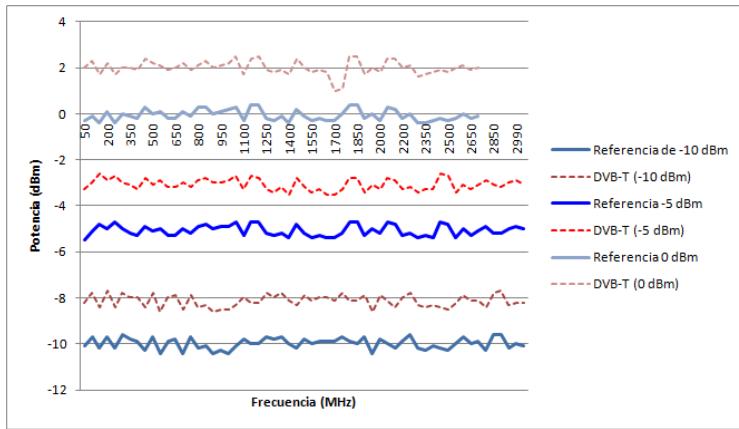


Figura 4.58: Potencia de Salida de Señal DVB-T para distintos ajustes de ganancia IF

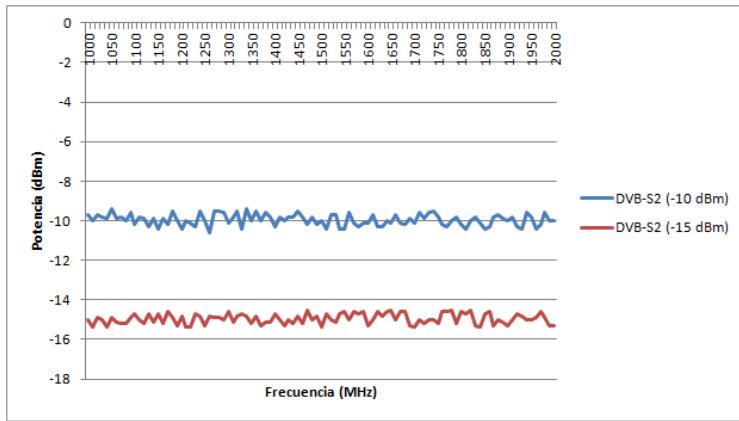


Figura 4.59: Potencia de Salida de Señal DVB-S2 para distintos ajustes de ganancia IF

debido al bajo nivel de potencia que presenta el estándar, se decidió trabajar en el rango entre -15 dBm y -10 dBm como se aprecia en la Figura 4.59.

Finalmente, la calibración para el estándar NTSC, se hizo de la misma forma que para DVB-S2; ; no obstante, el rango de frecuencia seleccionado es de 55.25 MHz hasta 799.25 MHz. Es importante destacar, que la potencia de canal para el estándar, no permanece en un nivel constante, ya que la ésta oscila entre un valor promedio máximo y mínimo, como se aprecia en la Figura 4.60.

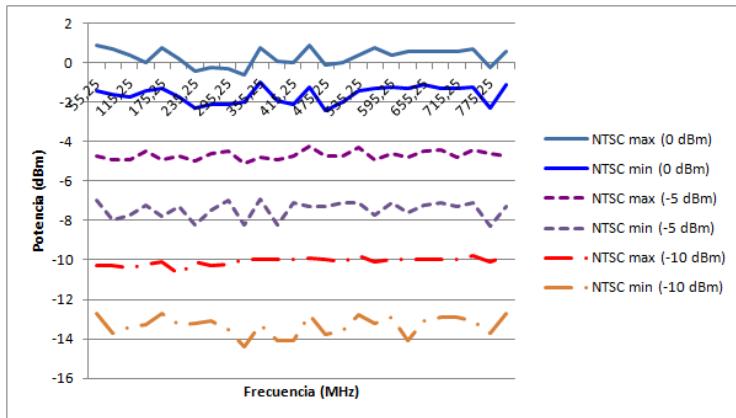


Figura 4.60: Potencia de Salida de Señal NTSC para distintos ajustes de ganancia IF

Los ajustes de ganancia para los estándares, a excepción de DVB-S2 y NTSC que cuentan con su calibración propia, pueden verse en la Tabla 4.2. No obstante, es necesario señalar que a la frecuencia de 2140 MHz, se obtuvo la potencia de salida más baja de acuerdo a lo observado en la Figura 4.49, y a esta frecuencia, el ajuste de ganancia IF para la señal de referencia de 5 dB es de 46 dB. Como se mencionó anteriormente, el máximo valor de ganancia IF es de 47 dB; por consiguiente cualquier señal que requiera un ajuste mayor a 1 dB, no podrá alcanzar los 5 dB de potencia máxima para este valor de frecuencia.

4.3.3. En Amplitud

Para el caso de la modulación con portadora suprimida, el usuario tendrá la opción de ajustar no solo la potencia sino también la amplitud de la señal, por lo que siguiendo el procedimiento de la sección 3.5.3 se realizó la calibración de dicho parámetro. La primera fase se midió la amplitud pico-pico de salida de una señal senoidal de frecuencia variable y potencia de -5dBm, correspondiente a la señal portadora. En la Tabla 4.3 se muestran los resultados de esta medición.

Como se puede observar en esta tabla la amplitud se mantiene dentro de un rango cercano a 430 mV a medida que la frecuencia aumenta, por lo que se escogió

Tabla 4.2: Ajustes de Ganancia IF con Respecto a las Señales de Referencia para Distintos Esquemas de Modulación

Tipo de Señal	Ajuste -10 dBm	Ajuste -5 dBm	Ajuste 0 dBm	Ajuste 5 dBm
Constante (Referencia)	0 dB	0 dB	0 dB	0 dB
GFSK	0 dB	0 dB	0 dB	0 dB
GMSK	0 dB	0 dB	0 dB	0 dB
WBFM	0 dB	0 dB	0 dB	0 dB
AM-LC	6 dB	6 dB	6 dB	6 dB
AM-SC (senoidal)	3 dB	3 dB	2 dB	2 dB
AM-SC (triangular)	4 dB	4 dB	4 dB	3 dB
AM-SC (cuadrada)	3 dB	2 dB	1 dB	1 dB
NBFM	0 dB	0 dB	0 dB	0 dB
OOK	3 dB	2 dB	1 dB	1 dB
4-ASK	1 dB	1 dB	0 dB	1 dB
2-PSK	1 dB	1 dB	0 dB	0 dB
4-PSK	0 dB	1 dB	0 dB	0 dB
8-PSK	0 dB	0 dB	0 dB	0 dB
16-PSk	0 dB	0 dB	0 dB	0 dB
32-PSK	0 dB	0 dB	0 dB	0 dB
QAM	0 dB	0 dB	0 dB	-1 dB
16-QAM	0 dB	0 dB	0 dB	-1 dB
64-QAM	0 dB	0 dB	0 dB	-1 dB
DVB-T	-2 dB	-2 dB	-2 dB	-2 dB

Tabla 4.3: Amplitud pico-pico de salida de señal portadora con potencia de canal de -5 dBm

Frecuencia (MHz)	Amplitud (mVpp)
50	830
100	690
150	540
200	430
250	430
300	420
350	460
400	480

este valor como la amplitud máxima del generador cuando se encuentre operando en este modo. Se debe tomar en cuenta que la amplitud real va a diferir del valor marcado en la aplicación, pero se va a mantener dentro de un rango de $\pm 50\text{mVpp}$.

En la Figura 4.61 se muestra el resultado de la segunda fase de la medición para la calibración de amplitud, donde se mantuvo la señal de salida a una frecuencia constante para determinar la variación en la amplitud de salida al modificar la amplitud de la señal digital generada en GNU Radio, hasta un valor mínimo de 0,2; el cual es el valor mínimo que puede tener la señal sin presentar distorsión al pasar por el convertidor digital analógico de la tarjeta HackRF One.

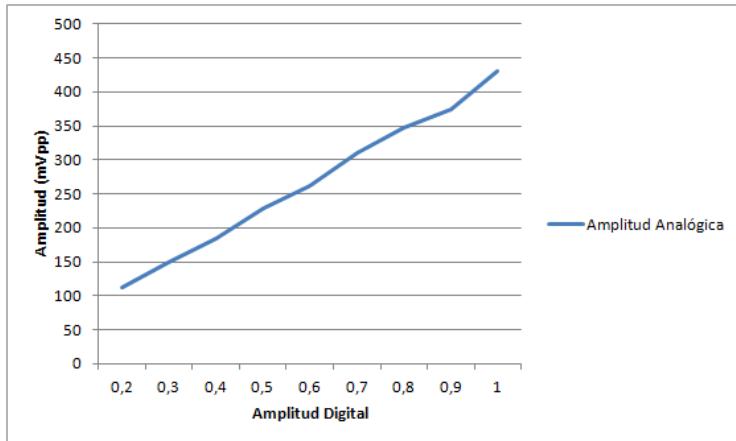


Figura 4.61: Variación de la Amplitud Analógica con Respecto a la Amplitud Digital

En la gráfica de la Figura 4.61, se puede observar que la variación de amplitud de salida del generador vectorial es linealmente proporcional a la variación de amplitud del bloque de señal analógica en el diagrama de GNU Radio, por lo que se puede obtener una relación que permita controlar esta variable por medio de un control deslizable que contenga los valores finales de salida del generador, los cuales se encuentran entre 110 mVpp y 430 mVpp. La relación obtenida es la siguiente:

$$\text{amp} = \frac{\text{Amplitud} - 30}{400} \quad (4.2)$$

Donde “Amplitud” se refiere al valor de la amplitud de salida del generador en mVpp y “amp” se refiere a la amplitud de la señal generada en GNU Radio, por lo que si la amplitud de salida deseada es de 340mVpp, la amplitud de la señal analógica generada por software será configurada a 0,775.

4.4. Desarrollo de la Aplicación

La aplicación finalizada cuenta con dos grupos de elementos que la conforman. En primer lugar, tenemos la interfaz gráfica, la cual muestra una lista desplegable con los diferentes esquemas de transmisión disponibles, también muestra una lista desplegable que contiene las fuentes de información o señales disponibles, y ésta cambia dependiendo del esquema de transmisión que ha sido previamente seleccionado. En la tabla 4.4 se muestra un resumen de las fuentes disponibles dependiendo del esquema seleccionado.

Tabla 4.4: Fuentes Disponibles por Esquema

Esquema seleccionado	Fuentes disponibles
AM-LC, AM-SC, NBFM, WBFM	Constante, Senoidal, Cuadrada, Triangular, Diente de sierra, Audio, Arbitraria
OOK, ASK, GMSK, GFSK, PSK, QAM	Aleatoria, Audio, Archivo (Video, Imagen, Texto), Streaming
DVB-T, DVB-S2	Video
NTSC	Imagen
OFDM	Aleatoria

Luego de escoger el esquema de transmisión que se va a utilizar y la fuente de información, aparecen los cuadros de texto identificados con los nombres de los parámetros que pueden ser modificados para la combinación seleccionada, entre los cuales se encuentran frecuencia modulante, índice de modulación, tasa de bits, entre otros.

En la parte inferior de la interfaz, se encuentran dos parámetros que se muestran de forma permanentes y se aplican a todos los esquemas de transmisión, éstos son la frecuencia portadora y la potencia de salida.



Figura 4.62: Interfaz de Inicio de la Aplicación



Figura 4.63: Interfaz con parámetros modificables

Por otro lado, un caso particular que aparece en la aplicación es el esquema AM-SC. Como fue mencionado en la sección 4.3.3, si la fuente seleccionada es una señal analógica, se puede escoger entre fijar la potencia de salida o la amplitud de la señal de salida. Se debe tomar en cuenta que cada parámetro tiene un rango diferente como fue mencionado en la sección 4.3.

Luego, los elementos que aparecen en la esquina inferior derecha son los botones «Inicio» y «Detener» que como indican sus nombres permiten el arranque del generador y también detener su ejecución. Se debe tomar en cuenta que después de haber iniciado un esquema de transmisión, si se desea detener la ejecución del

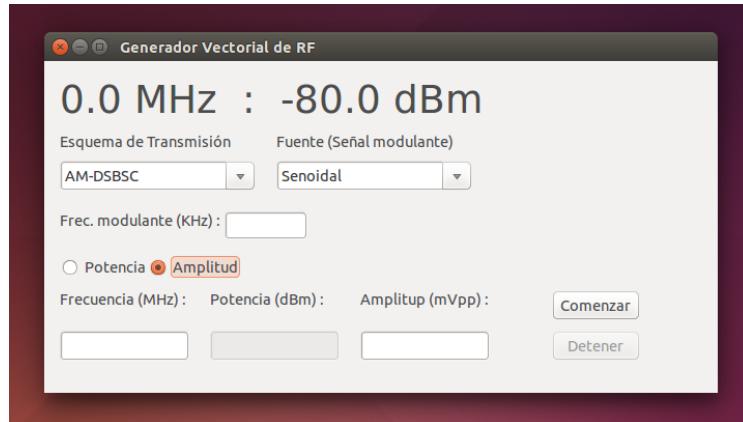


Figura 4.64: Otras Opciones en la Aplicación

transmisor, se debe pulsar el botón «Reset» ubicado en la tarjeta HackRF One después de haber pulsado el botón «Detener» en la aplicación. Finalmente, en la parte superior de la interfaz aparece un *display* de frecuencia y potencia que cambiará de valor cuando esté en funcionamiento el generador. Durante el desarrollo de la aplicación se intentó que la fuente de este *display* se pareciese a la de un *display* LCD como en los generadores comerciales, pero sólo es posible colocar fuentes que estuviesen ya instaladas en el sistema, en este caso, se colocó la fuente por defecto del sistema para evitar errores al ejecutar la aplicación en un equipo donde no se encuentre instalada la fuente deseada. En la Figura 4.62 se pueden observar los elementos que se muestran en la interfaz gráfica al iniciar la aplicación.

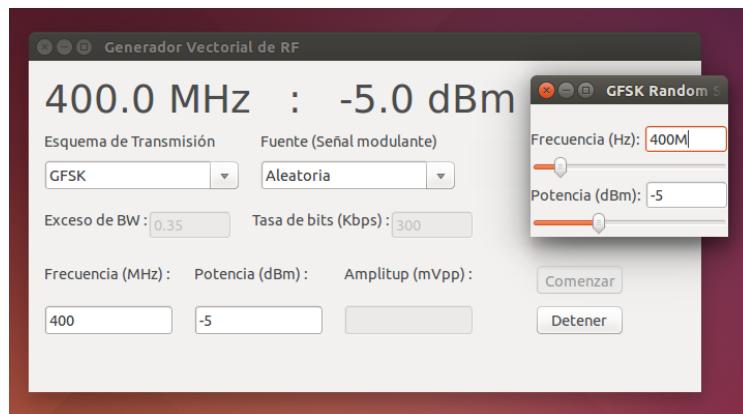


Figura 4.65: Control Auxiliar de la Aplicación

En segundo lugar, se encuentran los módulos que corresponden a cada uno de los diagramas construidos en GNU Radio Companion. Cada módulo posee una clase donde se crean las variables necesarias para el funcionamiento de ese esquema. También se crea un control gráfico auxiliar para el control de la frecuencia portadora y la potencia de salida (o puede también ser amplitud para el caso de AM-SC), los cuales son parámetros que se pueden modificar mientras el generador se encuentra en funcionamiento. De igual forma, estas clases crean los bloques y realizan las conexiones necesarias para el funcionamiento de su diagrama respectivo, y cuentan con métodos para cambiar la frecuencia portadora y la potencia de salida.

Por otra parte, para el caso de que el esquema de transmisión sea una modulación digital (OOK, ASK, GFSK, GMSK, PSK, QAM) con fuente de Streaming, aparecerán dos campos que corresponden a la dirección y el número de puerto por el cual se realizará la transmisión de datos. Hay que tomar en cuenta que se debe hacer uso de una aplicación externa para configurar la transmisión de datos a la dirección y el puerto que el usuario especifique en la aplicación.

Por último, la aplicación cuenta con un módulo adicional que contiene los métodos que implementan la calibración que se muestra en la sección 4.3.3. Este módulo importa un archivo que contiene la lista que contienen todos los valores de corrección de frecuencia, y también varias listas que contienen los ajustes de ganancia IF para los valores de frecuencia especificados en la sección 4.3, además de las listas de los esquemas de caso especial (DVB-S2 y NTSC). Estos métodos adicionales se encuentran en el archivo `Funciones.py` y las listas en el archivo `lista.py`.

4.5. Especificaciones Técnicas

1. **Rango de Frecuencia:** 50MHz — 3GHz.

2. **Rangos no disponibles:**

- 916 — 928 MHz
- 2149 — 2157 MHz

- 2306 — 2318 MHz
- 2749 — 2760 MHz

3. Rangos Generales de Potencia:

- -10 dBm — 0 dBm entre 50 MHz — 2.749 GHz
- -10 dBm — 5 dBm entre 2.157 GHz — 2.749 GHz
- -10 dBm — -5 dBm entre 2.76 GHz — 3 GHz

4. Precisión en Potencia: $\pm 1\text{dB}$ ¹

5. Esquemas de Modulación

- DSB-SC

Tabla 4.5: Especificaciones DSB-SC

Frecuencia Modulante		5 kHz — 1 MHz
Amplitud		110 mVpp — 430 mVpp
Rango de Potencia	Senoidal	-10 dBm — 0 dBm entre 50MHz — 2.749GHz -10dBm — 5dBm entre 2.157 GHz — 2.749GHz -10dBm — -6dBm 2.76GHz — 2.87 GHz -10dBm — -5dBm entre 2.87GHz — 3GHz
	Triangular	-10 dBm — 0dBm entre 50MHz — 2.04GHz -10dBm — -3dBm entre 2.04GHz — 2.157GHz -10dBm — 5dBm entre 2.157 GHz — 2.749GHz -10dBm — -7dBm entre 2.76GHz — 2.88GHz -10dBm — -5dBm entre 2.88GHz — 3GHz
	Cuadrada	-10dBm — 0dBm entre 50MHz — 2.749GHz -10dBm — 5dBm entre 2.57GHz — 2.749GHz -10dBm — -5dBm entre 2.6GHz — 3GHz
Tipo de Fuente		Constante, Senoidal, Cuadrada, Triangular, Diente de sierra, Audio, Arbitraria
Variación Promedio de Potencia		$\pm 0.2 \text{ dBm}$

¹A excepción de OFDM y NTSC que varían entre valores máximos y mínimos especificados.

■ **AM-LC**

Tabla 4.6: Especificaciones AM-LC

Frecuencia Modulante	5 kHz — 1 MHz
Rango de Potencia	-10 dBm — -5dBm entre 50MHz — 2.749 GHz -10 dBm — 5 dBm entre 2.318 GHz — 2.749 GHz -10 dBm entre 2.76 GHz — 3 GHz
Tipos de Fuente	Constante, Senoidal, Cuadrada, Triangular, Diente de sierra, Audio, Arbitraria
Variación Promedio de Potencia	±0.3 dBm

■ **NBFM**

Tabla 4.7: Especificaciones NBFM

Máxima Frecuencia Modulante	50 kHz
Desviación de Frecuencia	100Hz — 25 kHz
Rango de Potencia	-10dBm — 0dBm entre 50MHz — 2.749GHz -10dBm — 5dBm entre 2.57GHz — 2.749GHz -10dBm — -5dBm entre 2.6GHz — 3GHz
Tipo de Fuente	Constante, Senoidal, Cuadrada, Triangular, Diente de sierra, Audio, Arbitraria
Variación Promedio de Potencia	±0.3dBm

■ **WBFM**

Tabla 4.8: Especificaciones WBFM

Máxima Frecuencia Modulante	100 kHz
Desviación de Frecuencia	100Hz — 100 kHz
Rango de Potencia	-10dBm — 0dBm entre 50MHz — 2.749GHz -10dBm — 5dBm entre 2.57GHz — 2.749GHz -10dBm — -5dBm entre 2.6GHz — 3GHz
Tipo de Fuente	Constante, Senoidal, Cuadrada, Triangular, Diente de sierra, Audio, Arbitraria
Variación Promedio de Potencia	±0.2dBm

■ GMSK y GFSK

Tabla 4.9: Especificaciones GMSK y GFSK

Tasa de Bit	100kbps — 8Mbps
Rango de Potencia	-10dBm — 0dBm entre 50MHz — 2.749GHz -10dBm — 5dBm entre 2.57GHz — 2.749GHz 0dBm — -5dBm entre 2.6GHz — 3GHz
Tipo de Fuente	Aleatoria, Audio, Archivo (Video, Imagen, Texto), Streaming
Variación Promedio de Potencia	±0.2 dBm

■ PSK

Tabla 4.10: Especificaciones PSK

Tasa de Bit	100kbps — 8Mbps (hasta 4Mbps para BPSK)
Número de constelación	BPSK, 4-PSK, 8-PSK, 16-PSK, 32-PSK
Rango de Potencia	-10dBm — 0dBm entre 50MHz — 2.749GHz -10dBm — 5dBm entre 2.57GHz — 2.749GHz 0dBm — -5dBm entre 2.6GHz — 3GHz
Tipo de Fuente	Aleatoria, Audio, Archivo (Video, Imagen, Texto), Streaming
Variación Promedio de Potencia	±0.2 dBm

■ QAM

Tabla 4.11: Especificaciones QAM

Tasa de Bit	100kbps — 8Mbps
Número de constelación	QAM, 16-QAM, 64-QAM
Rango de Potencia	-10dBm — 0dBm entre 50MHz — 2.749GHz -10dBm — 5dBm entre 2.57GHz — 2.749GHz -10dBm — -5dBm entre 2.6GHz — 3GHz
Tipo de Fuente	Aleatoria, Audio, Archivo (Video, Imagen, Texto), Streaming
Variación Promedio de Potencia	± 0.3dBm

■ ASK

Tabla 4.12: Especificaciones ASK

Tasa de Bit	100kbps — 8Mbps (hasta 4Mbps para OOK)
Número de constelación	OOK, 4-ASK
Rango de Potencia	-10dBm — 0dBm entre 50MHz — 2.749GHz
	-10dBm — 5dBm entre 2.57GHz — 2.749GHz
	-10dBm — -5dBm entre 2.6GHz — 3GHz
Tipo de Fuente	Aleatoria, Audio, Archivo (Video, Imagen, Texto), Streaming
Variación Promedio de Potencia	±0.2dBm

■ NTSC

Tabla 4.13: Especificaciones NTSC

Rango de Frecuencia	55.25MHz — 799.25 MHz			
Rango de Potencia	-10 dBm — 0 dBm			
Valor de Potencia Promedio(dBm)	Nivel	-10dBm	-5dBm	0dBm
	Máximo	-10	-4.7	0.3
	Mínimo	-13.3	-7.4	-1.6
Tipo de Fuente	Imágenes			

■ DVB-T

Tabla 4.14: Especificaciones DVB-T

Esquema de Modulación	QPSK, 16QAM, 64QAM
Rango de Potencia	-10dBm — 0dBm entre 50MHz — 2.749GHz
	-10dBm — 5dBm entre 2.57GHz — 2.749GHz
	-10dBm — -5dBm entre 2.6GHz — 3GHz
Tasa de Codificación	1/2, 2/3, 3/4, 5/6, 7/8
Tipo de Fuente	Video (Formato MPEG-2)
Variación Promedio de Potencia	±0.1 dB

■ DVB-S2

Tabla 4.15: Especificaciones DVB-S2

Esquema de Modulación	QPSK, 8PSK, 8APSK, 16APSK, 32APSK
Rango de Potencia	-10dBm — -15 dBm
Rango de Frecuencia	1GHz — 2GHz
Tasa de Codificación	1/4, 1/3, 2/5, 1/2, 3/5, 2/3, ¾, 4/5, 5/6
Tipo de Fuente	Video (Formato MPEG-2)
Variación Promedio de Potencia	±0.3 dB

■ OFDM

Tabla 4.16: Especificaciones OFDM

Frecuencia de Muestreo	2Msps					
FFT	64, 128, 256, 512, 1024, 1536, 2048					
Número mínimo de tonos ocupados	20 % de la FFT					
Rango de Potencia	-10dBm — -4dBm entre 50MHz — 2749 MHz -10 dBm entre 2.76 GHz — 3GHz					
Tipo de Fuente	Aleatoria					
Valor de Potencia	Valor de Potencia Promedio(dBm)	%TO	Nivel -10dBm		Nivel -5dBm	
Máx		Mín	Máx	Mín		
100 %		-10	-10,6	-5,1	-5,6	
90 %		-10	-10,9	-5,1	-5,9	
80 %		-10,3	-11,1	-5,3	-6,1	
70 %		-9,9	-10,6	-5,2	-6,2	
60 %		-10,2	-10,7	-5,3	-5,8	
50 %		-9,8	-10,5	-4,9	-5,5	
40 %		-10,6	-11,2	-5,7	-6,9	
30 %		-10,3	-11,3	-4,8	-5,5	
20 %	-10,5	-11,3	-5,8	-6,6		

Capítulo V

Conclusiones y recomendaciones

5.1. Conclusiones

- Las técnicas de Radio Definida por Software permiten implementar dispositivos de comunicaciones móviles haciendo uso de poco hardware lo que permite reducir el costo y el tamaño físico de los equipos. Específicamente, en el caso de los generadores vectoriales de radio frecuencia, su implementación es factible con el uso de un convertidor digital/analógico de alta velocidad de procesamiento, un modulador en cuadratura, un oscilador local y un amplificador de ganancia variable, los cuales procesarán la información o flujo de datos de las señales generadas en un computador. Las especificaciones de cada uno de estos componentes, en conjunto con la capacidad de procesamiento del computador que se utilice, limita la frecuencia de muestreo, el cual a su vez está relacionado con los límites de frecuencia de las señales generadas en banda base.
- El software GNU Radio constituye una herramienta útil en la generación de señales ya que ofrece los elementos para crear diagramas de distintos esquemas de modulación, permitiendo ajustar diversos parámetros propios de cada uno estos. Así mismo, algunos de estos bloques de procesamiento de señal

utilizados, permiten modificaciones en su código fuente para mejorar su rendimiento en cuanto al procesamiento de información, sin tomar en cuenta la capacidad de procesamiento del computador que ejecute el software.

- La diversidad de los bloques de procesamiento de señal existentes en las librerías del software GNU Radio permiten construir los diagramas con numerosos esquemas de modulación diferentes y a su vez permiten distintos tipos de fuente de señal para estos tipos de modulación.
- Los distintos esquemas de modulación, en su mayoría, varían sus niveles de potencia en función de la frecuencia, por lo que es admisible lograr una calibración respecto a una señal de referencia, utilizando un sencillo ajuste de ganancia para cada uno de los esquemas disponibles. No obstante, la precisión de la calibración de frecuencia está limitada de acuerdo a la estabilidad del oscilador de reloj que se emplee.
- La integración de todos los diagramas por medio de una interfaz gráfica permite el fácil acceso a los diversos tipos de modulación disponibles en el dispositivo, logrando de esta manera la síntesis de un dispositivo versátil, de bajo costo y sencillo de manejar y transportar.
- El generador vectorial de radio frecuencia es un prototipo de instrumento de laboratorio con una precisión aceptable para un dispositivo de bajo costo y de utilidad para los laboratorios en el área de las telecomunicaciones, el cual ofrece gran versatilidad en cuanto a la variedad de fuentes, esquemas de modulación, rango de frecuencia, rango de potencia y parámetros disponibles.

5.1.1. Recomendaciones

- Caracterizar el prototipo en frecuencia y potencia entre el rango de 3GHz y 6 GHz.
- Continuar el desarrollo del generador vectorial de señales integrando el uso un computador portátil de bajo costo que soporte el software, una pantalla de

cristal líquido, y/o desarrollar una aplicación móvil la cual permita ajustar los parámetros del generador de forma remota.

- Desarrollar un módulo que permita la extracción y manipulación de las señales en banda base, antes de la etapa de radio frecuencia del transceptor HackRF One.
- Escribir un conjunto de prácticas de laboratorio donde se utilice el generador vectorial de radio frecuencia en los laboratorios de las asignaturas de Teoría de las Comunicaciones
- Generar señales con esquemas de modulación de espectro ensanchado como DSSS, CSS y FHSS, para ser utilizados con el generador vectorial de señales.
- Desarrollar un receptor capaz de demodular los distintos esquemas de modulación disponibles en el generador vectorial de radio frecuencia con el fin de escribir nuevas prácticas de laboratorio no solo en las asignaturas de teoría de las comunicaciones o instrumentación, sino también de antenas.
- Iniciar el desarrollo con tecnología similar a la utilizada, de un prototipo de Analizador Vectorial de Redes. Se debe tomar en cuenta que este tipo de dispositivo no se puede desarrollar utilizando el transceptor HackRF One debido a que éste sólo cuenta solamente con un puerto de salida/entrada.

Apéndice A

Tabla de Canales del Estándar NTSC

CATV	MHz	VIDEO	BROADCAST	MHz	VIDEO
02	54-60	55.25	02	54-60	55.25
03	60-66	61.25	03	60-66	61.25
04	66-72	67.25	04	66-72	67.25
05	76-82	77.25	05	76-82	77.25
06	82-88	83.25	06	82-88	83.25
07	174-180	175.25	07	174-180	175.25
08	180-186	181.25	08	180-186	181.25
09	186-192	187.25	09	186-192	187.25
10	192-198	193.25	10	192-198	193.25
11	198-204	199.25	11	198-204	199.25
12	204-210	205.25	12	204-210	205.25
13	210-216	211.25	13	210-216	211.25
14	120-126	121.25	14	470-476	471.25
15	126-132	127.25	15	476-482	477.25
16	132-138	133.25	16	482-488	483.25
17	138-144	139.25	17	488-494	489.25
18	144-150	145.25	18	494-500	495.25
19	150-156	151.25	19	500-506	501.25
20	156-162	157.25	20	506-512	507.25

CATV	MHz	VIDEO	BROADCAST	MHz	VIDEO
21	162-168	163.25	21	512-518	513.25
22	168-174	169.25	22	518-524	519.25
23	216-222	217.25	23	524-530	525.25
24	222-228	223.25	24	530-536	531.25
25	228-234	229.25	25	536-542	537.25
26	234-240	235.25	26	542-548	543.25
27	240-246	241.25	27	548-554	549.25
28	246-252	247.25	28	554-560	555.25
29	252-258	253.25	29	560-566	561.25
30	258-264	259.25	30	566-572	567.25
31	264-270	265.25	31	572-578	573.25
32	270-276	271.25	32	578-584	579.25
33	276-282	277.25	33	584-590	585.25
34	282-288	283.25	34	590-596	591.25
35	288-294	289.25	35	596-602	597.25
36	294-300	295.25	36	602-608	603.25
37	300-306	301.25	37	608-614	609.25
38	306-312	307.25	38	614-620	615.25
39	312-318	313.25	39	620-626	621.25
40	318-324	319.25	40	626-632	627.25
41	324-330	325.25	41	632-638	633.25
42	330-336	331.25	42	638-644	639.25
43	336-342	337.25	43	644-650	645.25
44	342-348	343.25	44	650-656	651.25
45	348-354	349.25	45	656-662	657.25
46	354-360	355.25	46	662-668	663.25
47	360-366	361.25	47	668-674	669.25
48	366-372	367.25	48	674-680	675.25
49	372-378	373.25	49	680-686	681.25
50	378-384	379.25	50	686-692	687.25
51	384-390	385.25	51	692-698	693.25
52	390-396	391.25	52	698-704	699.25

CATV	MHz	VIDEO	BROADCAST	MHz	VIDEO
53	396-402	397.25	53	704-710	705.25
54	402-408	403.25	54	710-716	711.25
55	408-414	409.25	55	716-722	717.25
56	414-420	415.25	56	722-728	723.25
57	420-426	421.25	57	728-734	729.25
58	426-432	427.25	58	734-740	735.25
59	432-438	433.25	59	740-746	741.25
60	438-444	439.25	60	746-752	747.25
61	444-450	445.25	61	752-758	753.25
62	450-456	451.25	62	758-764	759.25
63	456-462	457.25	63	764-770	765.25
64	462-468	463.25	64	770-776	771.25
65	468-474	469.25	65	776-782	777.25
66	474-480	475.25	66	782-788	783.25
67	480-486	481.25	67	788-794	789.25
68	486-492	487.25	68	794-800	795.25
69	492-498	493.25	69	800-806	801.25

CATV	MHz	VIDEO	CATV	MHz	VIDEO
70	498-504	499.25	82	570-576	571.25
71	504-510	505.25	83	576-582	577.25
72	510-516	511.25	84	582-588	583.25
73	516-522	517.25	85	588-594	589.25
74	522-528	523.25	86	594-600	595.25
75	528-534	529.25	87	600-606	601.25
76	534-540	535.25	88	606-612	607.25
77	540-546	541.25	89	612-618	613.25
78	546-552	547.25	90	618-624	619.25
79	552-558	553.25	91	624-630	625.25
80	558-564	559.25	92	630-636	631.25
81	564-570	565.25	93	636-642	637.25

CATV	MHz	VIDEO	CATV	MHz	VIDEO
94	642-648	643.25	110	708-714	709.25
95	90-96	91.25	111	714-720	715.25
96	96-102	97.25	112	720-726	721.25
97	102-108	103.25	113	726-732	727.25
98	108-114	109.25	114	732-738	733.25
99	114-120	115.25	115	738-744	739.25
100	648-654	649.25	116	744-750	745.25
101	654-660	655.25	117	750-756	751.25
102	660-666	661.25	118	756-762	757.25
103	666-672	667.25	119	762-768	763.25
104	672-678	673.25	120	768-774	769.25
105	678-684	679.25	121	774-780	775.25
106	684-690	685.25	122	780-786	781.25
107	690-696	691.25	123	786-792	787.25
108	696-702	697.25	124	792-798	793.25
109	702-708	703.25	125	798-804	799.25

Referencias Bibliográficas

- [1] Antonio J Fedón. Diseño y construcción de un prototipo de generador vectorial de radio frecuencia de bajo costo sobre plataforma de software libre en el rango de frecuencias de telefonía celular, Proyecto de Pregrado: Universidad de Carabobo, 2006.
- [2] Michael Ossman. Hackrf one: Low cost software radio platform. URL <https://github.com/mossmann/hackrf>.
- [3] Ulrich Reimers. *DVB: the family of international standards for digital video broadcasting*. Springer Science & Business Media, 2005.
- [4] Philip Golden, Hervé Dedieu, and Krista S Jacobsen. *Fundamentals of DSL technology*. CRC Press, 2005.
- [5] JS Chitode. *Analog And Digital Communication Engineering*. Technical Publications, 2009.
- [6] Ferrel G Stremler. *Introducción a los Sistemas de Comunicaciones*. Addison Wesley, 1993.
- [7] Amitabha Bhattacharya. *Digital Communication*. Tata McGraw-Hill, 2005.
- [8] Matthew Gast. *802.11 Wireless Networks: The Definitive Guide*. O'Reilly Media, Inc.", 2005.
- [9] Dan McMahill. *Automated Calibration of Modulated Frequency Synthesizers*, volume 650. Springer Science & Business Media, 2001.

- [10] James Irvine and David Harle. *Data communications and networks: An engineering approach*. John Wiley & Sons, 2002.
- [11] RK Rao Yarlagadda, Radha Krishna Rao Yarlagadda, and Radha Krishna Rao Yarlagadda. *Analog and Digital Signals and Systems*, volume 1. Springer, 2010.
- [12] Dayan Adionel Guimaraes. *Digital Transmission: A Simulation-Aided Introduction with VisSim/Comm*. Springer Science & Business Media, 2010.
- [13] David R Smith. *Digital transmission systems*. Springer Science & Business Media, 2004.
- [14] Robert Dixon. *Radio receiver design*, volume 104. CRC Press, 1998.
- [15] Hiroshi Harada and Ramjee Prasad. *Simulation and Software Radio for Mobile Communications*, volume 1. Artech House, 2002.
- [16] Hermann Rohling. *OFDM: Concepts for Future Communication Systems*. Springer Science & Business Media, 2011.
- [17] Bernard Le Floch, Michel Alard, and Claude Berrou. Coded orthogonal frequency division multiplex [tv broadcasting]. *Proceedings of the IEEE*, 83(6): 982–996, 1995.
- [18] Daniel Minoli. *Satellite Systems Engineering in an IPv6 Environment*. CRC Press, 2009.
- [19] Abdallah Shami, Martin Maier, and Chadi Assi. *Broadband Access Networks*. Springer, 2009.
- [20] RG Gupta. *Television Engineering and Video Systems*. Tata McGraw-Hill Education, 2005.
- [21] Constantino Pérez Vega. *Fundamentos de televisión analógica y digital*. Ed. Universidad de Cantabria, 2003.
- [22] Dennis Kucera. Introduction to mpeg-2 compression and transport streams, November 2002.

- [23] Periklis Chatzimisios, Christos Verikoukis, Ignacio Santamaría, Massimiliano Laddomada, and Oliver Hoffmann. *Mobile Lightweight Wireless Systems: Second International ICST Conference, Mobilight 2010, May 10-12, 2010, Barcelona, Spain, Revised Selected Papers*, volume 45. Springer Science & Business Media, 2010.
- [24] DVB Project Office. 2nd generation terrestrial. the worl's most advanced digital terrestrial tv system, May 2015. URL https://www.dvb.org/resources/public/factsheets/dvb-t2_factsheet.pdf.
- [25] Digital Video Broadcasting (DVB). Second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering and other broadband satellite applications; part 1: Dvb-s2. Technical Report 302 307-1 V1.4.1, ETSI, 2014. URL http://www.etsi.org/deliver/etsi_en/302300_302399/30230701/01.04.01_60/en_30230701v010401p.pdf.
- [26] Markus Dillinger, Kambiz Madani, and Nancy Alonistioti. *Software Defined Radio: Architectures, Systems and Functions*. John Wiley & Sons, 2005.
- [27] Jen-Wei Hsieh, Guo-Ruey Tsai, and Min-Chuan Lin. Using FPGA to implement a n-channel arbitrary waveform generator with various add-on functions. *Proceedings. 2003 IEEE International Conference on Field-Programmable Technology (FPT) (IEEE Cat. No.03EX798)*, 2003. doi: 10.1109/fpt.2003.1275761. URL <http://dx.doi.org/10.1109/FPT.2003.1275761>.
- [28] Emanuel Trabes, Diego Esteban Costa, and Carlos Federico Sosa Páez. Generador de señales con forma de onda arbitraria y ruido usando dds en fpga. In *IV Congreso Microelectrónica Aplicada*, 2003.
- [29] Predrag Jovanovic, Predrag Petrovic, Branislav Pavic, and Ninoslav Remenski. Implementation of RF signal generator for demodulator/receiver testing in SDR design. *2011 19th Telecommunications Forum (TELFOR) Proceedings of Papers*, Nov 2011. doi: 10.1109/telfor.2011.6143545. URL <http://dx.doi.org/10.1109/TELFOR.2011.6143545>.
- [30] A. Fedón. Reduction of peak-to-average power ratio of bandwith efficient linear formats, PhD Dissertation: University of Florida, 2000.

- [31] Agilent Technologies. Signal source basics. URL <http://cp.literature.agilent.com/litweb/pdf/5965-7918E.pdf>.
- [32] Agilent Technologies. The abc's of arbitrary waveform generation, 2005.
- [33] Radiocommunication Sector of ITU (ITU-R). Definitions or software defined radio (sdr) and cognitive radio system (crs). Technical report, International Telecommunication Union (ITU), 2009.
- [34] Lpc4350/30/20/10: 32-bit arm cortex-m4/m0 flashless mcu; up to 264 kb sram; ethernet; two hs usbs; advanced configurable peripherals. URL http://www.nxp.com/products/microcontrollers/product_series/lpc4300/.
- [35] Max5864: Ultra-low-power, high-dynamic-performance, 22msps analog front end, . URL <http://www.maximintegrated.com/en/products/analog/data-converters/analog-front-end-ics/MAX5864.html>.
- [36] Max2837: 2.3ghz to 2.7ghz wireless broadband rf transceiver, . URL <http://www.maximintegrated.com/en/products/comms/wireless-rf/MAX2837.html>.
- [37] Rffc5071/5072: Wideband synthesizer/vco with integrated 6ghz mixer. URL <http://www.rfmd.com/store/rffc5072-1.html>.
- [38] Mga81563: 0.1 - 6 ghz 3v, 14 dbm amplifier. URL <http://www.avagotech.com/docs/AV02-0966EN>.
- [39] Gnu radio: The free & open software ecosystem. URL <http://gnuradio.org/redmine/projects/gnuradio/wiki>.
- [40] Simon Haykin. *Communication systems*. John Wiley & Sons, 2008.
- [41] H Hwei and Dr D Mitra. *Analog and Digital Communications*. EU: Schaums Outline Series, 1993.
- [42] Bhagwandas P Lathi. *Modern Digital and Analog Communication Systems*. Oxford University Press, Inc., 1990.
- [43] Kevin Roebuck. *Software-Defined Radio (SDR): High-Impact Technology - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors*. Emereo Roebuck, 2012.

- [44] John LB Walker. *Handbook of RF and microwave power amplifiers*. Cambridge University Press, 2011.
- [45] Application note1ma98: db or not db? everything you ever wanted to know about decibels but were afraid to ask... URL http://www.rohde-schwarz.de/file_5613/1MA98_4E.pdf.
- [46] Pejman Roshan and Jonathan Leary. *802.11 Wireless LAN Fundamentals*. Cisco Systems, 2010.
- [47] PM Nitaigour. Sensor networks and configuration: Fundamentals, standards, platforms, and applications, 2007.
- [48] Com-1028 fsk/msk/gfsk/gmsk digital modulator, September 2009. URL www.comblock.com/download/com1028.pdf.
- [49] Michael Bernard Steer. *Microwave and RF design: a systems approach*. SciTech Pub., 2010.
- [50] Michel Daoud Yacoub. *Foundations of mobile radio engineering*. CRC press, 1993.
- [51] V Jeyasri Arokiamary. *Mobile Communications*. Technical Publications, 2009.
- [52] Bernhard Kaebs. Application note 7ts02: The crest factor in dvb-t (ofdm) transmitter systems and its influence on the dimensioning of power components, Janurary 2007. URL http://cdn.rohde-schwarz.com/pws/dl_downloads/dl_application/application_notes/7ts02/7TS02_2E.pdf.
- [53] Walter Fischer. *Digital video and audio broadcasting technology: a practical engineering guide*. Springer Science & Business Media, 2008.
- [54] Giovanni Giambene et al. Resource management in satellite networks. *Optimization and cross-layer design*, 2007.
- [55] History of dvb,. URL <https://www.dvb.org/about/history>.
- [56] Development of standards,. URL <https://www.dvb.org/about/process>.
- [57] M Massel. Digital television dvb-t, cofdm and atsc 8-vsb, digitaltvbooks, 2000.

- [58] Digital Video Broadcasting (DVB). Implementation guidelines for dvb terrestrial services;transmission aspects. Technical Report 101 190 V1.3.2, ETSI, 2011. URL http://www.etsi.org/deliver/etsi_tr/101100_101199/101190/01.03.02_60/tr_101190v010302p.pdf.
- [59] Digital Video Broadcasting (DVB). User guidelines for the second generation system for broadcasting, interactive services, news gathering and other broadband satellite applications (dvb-s2). Technical Report 102 376 V1.1.1, ETSI, 2005. URL http://www.etsi.org/deliver/etsi_tr/102300_102399/102376/01.01.01_60/tr_102376v010101p.pdf.
- [60] Walter Fischer. *Digital television: a practical guide for engineers*. Springer Science & Business Media, 2004.
- [61] Clayton Smith. Sdr-examples, 2015. URL <https://github.com/argilo/sdr-examples>.
- [62] Lei Zhang. Implementation of wireless communication based on software defined radio. Master's thesis, Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación. Universidad Politécnica de Madrid. Departamento de Ingeniería Audiovisual y Comunicaciones, 2013.

Anexo A

Listado de Programas

Listado de programa generadorVectorialRF.py

Listado de programa Ventana.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
''' Generador Vectorial RF como Trabajo especial de Grado
    Kevin Henriquez C.I. 20.696.184
    Jorge De Castro C.I. 19.755.372 '''

import os
import wx
import sys
print 'Ensamblando los componentes, espere por favor...'
from Ventana import MainWindow
from gnuradio import gr

class MyApp(wx.App):
    def OnInit(self):
        if gr.version() >= 3.7:
            self.main = MainWindow(None,"Generador Vectorial de RF")
            self.main.Show()
            self.SetTopWindow(self.main)
            return True
        else:
            print 'La versión de gnuradio debe ser mayor o igual a 3.7'
            sys.exit()

    def main():
        application = MyApp(0)
        application.MainLoop()

if __name__ == '__main__':
    main()
```

Listado de programa Ventana.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from AM_DSB_LC import *
from AM_DSB_LC_audio import AM_LC_Audio
```

```
from AM_DSB_LC_vector import AM_DSB_LC_vector
from AM_DSB_SC import AM_DSB_SC
from AM_DSB_SC_amplitud import AM_DSB_SC_amplitud
from AM_DSB_SC_audio import AM_DSB_SC_audio
from AM_DSB_SC_vector import AM_DSB_SC_vector
from NBFM import NBFM
from NBFM_Audio import NBFM_Audio
from NBFM_vector import NBFM_vector
from WBFM import WBFM
from WBFM_Audio import WBFM_Audio
from WBFM_vector import WBFM_vector
from GFSK_Random import GFSK_Random
from GFSK_audio import GFSK_audio
from GFSK_video import GFSK_video
from GFSK_UDP import GFSK_UDP
from GMSK_Random import GMSK_Random
from GMSK_audio import GMSK_audio
from GMSK_video import GMSK_video
from GMSK_UDP import GMSK_UDP
from PSK_Random import PSK_Random
from PSK_Audio import PSK_Audio
from PSK_video import PSK_video
from PSK_UDP import PSK_UDP
from OOK import OOK
from OOK_audio import OOK_audio
from OOK_UDP import OOK_UDP
from OOK_video import OOK_video
from QAM_Audio import QAM_Audio
from QAM_Random import QAM_Random
from QAM_video import QAM_video
from QAM_UDP import QAM_UDP
from OFDM_Randomv2 import OFDM_Random
from ASK_Random import ASK_Random
from ASK_Audio import ASK_Audio
from ASK_video import ASK_video
from ASK_UDP import ASK_UDP
from dvbs2_folder.top_block import top_block_dvbs2
from dvbt_folder.top_block import top_block_dvbt
from ntsc_encode import convertir_imagen
from ntsc_hackrf import ntsc_hackrf
```

```

from wxPython.tools import helpviewer
import wx
import pygtk
import sys
import math
import traceback
import wx.lib.agw.genericmessagedialog as GMD
pygtk.require('2.0')
import gtk

class MainWindow(wx.Frame):
    def __init__(self, parent, title):
        wx.Frame.__init__(self, parent, title=title, pos=(350, 300), size=(650, 320))

        '''Menu Archivo'''
        filemenu = wx.Menu()
        menuExit = filemenu.Append(wx.ID_EXIT, "&Salir", "Salir de la aplicación")
        '''Menu Ayuda'''
        helpmenu = wx.Menu()
        menuAyuda = helpmenu.Append(wx.ID_HELP,
        'Ayuda', 'Ayuda sobre la aplicación')
        menuabout = helpmenu.Append(wx.ID_ABOUT,
        "&Acerca de", "Información sobre este programa")
        '''Creando la barra de menu'''
        menuBar = wx.MenuBar()
        menuBar.Append(filemenu, "&Archivo") # Agregar el menu Archivo
        menuBar.Append(helpmenu, '&Ayuda')
        self.SetMenuBar(menuBar) # Agregar la barra de menu al Frame
        self.panel_uno = MyPanel(self)
        sys.excepthook = MyExceptionHook
        '''Eventos de menu'''
        self.Bind(wx.EVT_MENU, self.OnAbout, menuabout)
        self.Bind(wx.EVT_MENU, self.OnExit, menuExit)
        self.Bind(wx.EVT_MENU, self.OnHelp, menuAyuda)
        self.Bind(wx.EVT_CLOSE, self.OnClose, self)

    def OnAbout(self, e):
        info = """Generador Vectorial de Onda Arbitraria basado
en Radio Definida por Software y software libre.

```

Universidad de Carabobo, Facultad de Ingeniería,
Escuela de Telecomunicaciones.

versión 1.0, Julio de 2015

```
Kevin Henriquez, Jorge De Castro"""
dlg = wx.MessageDialog(self, info, "Acerca de la aplicación", wx.OK)
dlg.ShowModal() # Mostrar
dlg.Destroy() # Cerrar y destruir al finalizar

def OnExit(self, e):
    self.Close(True) # Cerrar la ventana
    sys.exit()

def OnHelp(self, e):
    helpviewer.main(['', 'AyudaGenerador/ayudaGenerador.hhp'])

def OnClose(self, e):
    sys.exit()

class MyPanel(wx.Panel):
    def __init__(self, parent):
        wx.Panel.__init__(self, parent=parent)
        self.flag = 0
        self.mainSizer = wx.BoxSizer(wx.HORIZONTAL)
        self.vectorstr = ''
        vbox = wx.BoxSizer(wx.VERTICAL)
        '''Header'''
        font = wx.Font(28, wx.DEFAULT, wx.NORMAL, wx.NORMAL, False)
        cuadro = wx.StaticText(self, label="0.0 MHz : -80.0 dBm")
        cuadroSetFont(font)
        cuadro.Refresh()
        self.cuadro = cuadro
        vbox.Add(self.cuadro)
        vbox.Add((-1, 10))
        '''ComboBox de esquemas de modulación'''
        esquemas = []
        esquemas.append('AM-LC')
        esquemas.append('AM-DSBSC')
        esquemas.append('NBFM')
```

```

        esquemas.append('WBFM')
        esquemas.append('OOK')
        esquemas.append('4-ASK')
        esquemas.append('PSK')
        esquemas.append('GFSK')
        esquemas.append('GMSK')
        esquemas.append('QAM')
        esquemas.append('NTSC')
        esquemas.append('DVB-S2')
        esquemas.append('DVB-T')
        esquemas.append('OFDM')
        self.esquema = wx.ComboBox(self, value='Esquema',
choices=esquemas, style=wx.CB_READONLY)
        '''ComboBox de Fuente'''
        fuentes = []
        self.fuente = wx.ComboBox(self, value='Fuente',
choices=fuentes, style=wx.CB_READONLY)
        '''ComboBox de Puntos de Constelacion'''
        const_puntos = []
        self.const = wx.ComboBox(self, value='Constelación',
choices=const_puntos, style=wx.CB_READONLY)
        '''ComboBox de OFDM'''
        op_ofdm = ['BPSK', 'QPSK', '8PSK', 'QAM16', 'QAM64']
        self.ofdm = wx.ComboBox(self, value='Opción',
choices=op_ofdm, style=wx.CB_READONLY)
        '''Auxiliares'''
        tvmodlist = []
        tvcoderate = []
        '''Etiquetas iniciales'''
        self.lblmod = wx.StaticText(self, label="Esquema de Transmisión")
        self.lblfuente = wx.StaticText(self, label="Fuente (Señal modulante)")
        self.lblconst = wx.StaticText(self, label="Nro. de puntos de Constelación")
        self.lblofdm = wx.StaticText(self, label="Opción de Modulación")
        grid1 = wx.FlexGridSizer(2, 4, 10, 20)
        grid1.Add(self.lblmod)
        grid1.Add(self.lblfuente)
        grid1.Add(self.lblconst)
        grid1.Add(self.lblofdm)
        grid1.Add(self.esquema)
        grid1.Add(self.fuente)

```

```

grid1.Add(self.const)
grid1.Add(self.ofdm)

self.grid1 = grid1
self.grid1.Hide(self.lblconst)
self.grid1.Hide(self.const)
self.grid1.Hide(self.lblofdm)
self.grid1.Hide(self.ofdm)

vbox.Add(self.grid1)
vbox.Add((-1, 20))

'''Boxes para controles con su etiqueta'''
self.box1 = wx.BoxSizer(wx.HORIZONTAL)
self.box2 = wx.BoxSizer(wx.HORIZONTAL)
self.box3 = wx.BoxSizer(wx.HORIZONTAL)
self.box4 = wx.BoxSizer(wx.HORIZONTAL)
self.box41 = wx.BoxSizer(wx.HORIZONTAL)
self.box5 = wx.BoxSizer(wx.HORIZONTAL)
self.box6 = wx.BoxSizer(wx.HORIZONTAL)
self.box7 = wx.BoxSizer(wx.HORIZONTAL)
self.box8 = wx.BoxSizer(wx.HORIZONTAL)
self.box9 = wx.BoxSizer(wx.HORIZONTAL)
self.boxVector = wx.BoxSizer(wx.HORIZONTAL)
self.boxVideo = wx.BoxSizer(wx.HORIZONTAL)
self.boxUDP = wx.BoxSizer(wx.HORIZONTAL)
self.boxOpciones = wx.BoxSizer(wx.HORIZONTAL)

'''Controles Adicionales'''
self.fmlabel = wx.StaticText(self, label="Frec. modulante (KHz) : ")
self.fm = wx.TextCtrl(self, -1)
self.ind_modlbl = wx.StaticText(self, label="Indice modulación :")
self.ind_mod = wx.TextCtrl(self, -1)
self.desvlabel = wx.StaticText(self, label="Max desviación (KHz):")
self.desv = wx.TextCtrl(self, -1)
self.btlable = wx.StaticText(self, label="Exceso de BW :")
self.bt = wx.TextCtrl(self, -1)
self.bitratelb = wx.StaticText(self, label="Tasa de bits (Kbps) :")
self.bitrate = wx.TextCtrl(self, -1)
self.lblfft = wx.StaticText(self, label="Longitud FFT :")

```

```

        self.fftlen = wx.TextCtrl(self, -1, value="512")
        self.lblcanales = wx.StaticText(self, label="Canales ocupados :")
        self.canales = wx.TextCtrl(self, -1, value="200")
        self.lblprefijo = wx.StaticText(self, label="Prefijo ciclico :")
        self.prefijo = wx.TextCtrl(self, -1, value="173")
        self.lbltvmod = wx.StaticText(self, label="Modulación")
        self.tvmod = wx.ComboBox(self, value='Modulación',
            choices=tvmodlist, style=wx.CB_READONLY)
        self.lbltvcr = wx.StaticText(self, label="Tasa codificación")
        self.coderate = wx.ComboBox(self, value='Tasa',
            choices=tvcoderate, style=wx.CB_READONLY)
        self.vectorlbl = wx.StaticText(self, label="Onda :")
        self.vector = wx.TextCtrl(self, -1)
        self.vector_button = wx.Button(self, label="Cargar")
        self.videolbl = wx.StaticText(self, label='Archivo :')
        self.video_path = wx.TextCtrl(self, -1)
        self.video_button = wx.Button(self, label="Buscar")
        self.dirlbl = wx.StaticText(self, label='Dirección :')
        self.dir = wx.TextCtrl(self, -1, size = (128,-1))
        self.portlbl = wx.StaticText(self, label='Puerto :')
        self.port = wx.TextCtrl(self, -1)
        self.rdpot = wx.RadioButton(self, -1, "Potencia", style=wx.RB_GROUP)
        self.rdamp = wx.RadioButton(self, -1, "Amplitud")

    '''Sizer'''
    sizer = wx.FlexGridSizer(rows=4, cols=3, vgap=10, hgap=20)
    self.box1.Add(self.fmlabel, 1)
    self.box1.Add(self.fm, 0)
    self.box2.Add(self.ind_modlbl, 1)
    self.box2.Add(self.ind_mod, 0)
    self.box3.Add(self.desvlabel, 1)
    self.box3.Add(self.desv, 0)
    self.box4.Add(self.btlabel, 1)
    self.box4.Add(self.bt, 0)
    self.box41.Add(self.bitratelb, 1)
    self.box41.Add(self.bitrate, 0)
    self.box5.Add(self.lblfft, 1)
    self.box5.Add(self.fftlen, 0)
    self.box6.Add(self.lblcanales, 1)
    self.box6.Add(self.canales, 0)

```

```

        self.box7.Add(self.lblprefijo, 1)
        self.box7.Add(self.prefijo, 0)
        self.box8.Add(self.lbltvmod, 1)
        self.box8.Add(self.tvmod, 0)
        self.box9.Add(self.lbltvcr, 1)
        self.box9.Add(self.coderate, 0)

        self.boxVector.Add(self.vectorlbl)
        self.boxVector.Add(self.vector, 0, wx.EXPAND)
        self.boxVector.Add(self.vector_button)
        self.boxVideo.Add(self.videolbl)
        self.boxVideo.Add(self.video_path)
        self.boxVideo.Add(self.video_button)
        self.boxUDP.Add(self.dirlbl)
        self.boxUDP.Add(self.dir)
        self.boxUDP.Add((15, -1))
        self.boxUDP.Add(self.portlbl)
        self.boxUDP.Add(self.port)
        self.boxOpciones.Add(self.rdpot)
        self.boxOpciones.Add(self.rdamp)

        self.bs1 = wx.StaticText(self) #Espacio en blanco para
        self.bs2 = wx.StaticText(self) #el gridsizer

        sizer.Add(self.box1, 1)
        sizer.Add(self.box2, 1)
        sizer.Add(self.box3, 1)
        sizer.Add(self.box4, 1)
        sizer.Add(self.box41, 1)
        sizer.Add(self.bs2, 1)
        sizer.Add(self.box5, 1)
        sizer.Add(self.box6, 1)
        sizer.Add(self.box7, 1)
        sizer.Add(self.box8, 1)
        sizer.Add(self.box9, 1)

        '''Agregando los controles y espacio entre ellos'''
        vbox.Add(sizer)
        vbox.Add((-1, 15))
        vbox.Add(self.boxVector)

```

```

vbox.Hide(self.boxVector)
vbox.Add(self.boxVideo)
vbox.Hide(self.boxVideo)
vbox.Add(self.boxUDP)
vbox.Hide(self.boxUDP)
vbox.Add(self.boxOpciones)
vbox.Hide(self.boxOpciones)
vbox.Add((-1, 10))

'''Botones'''
self.button = wx.Button(self, label="Comenzar")
self.button2 = wx.Button(self, label="Detener")
self.button2.Enable(False)
'''Control frecuencia portadora'''
self.lblfreq = wx.StaticText(self, label="Frecuencia (MHz) :")
self.frecuencia = wx.TextCtrl(self, -1)
'''Control de potencia'''
self.lblpot = wx.StaticText(self, label="Potencia (dBm) :")
self.potencia = wx.TextCtrl(self, -1)
'''Control Amplitud'''
self.lblamp = wx.StaticText(self, label="Amplitud (mVpp) :")
self.amplitud = wx.TextCtrl(self, -1)
'''Colocar los controles principales'''
mainbox = wx.GridSizer(rows=2, cols=4, vgap=10, hgap=20)
mainbox.Add(self.lblfreq)
mainbox.Add(self.lblpot)
mainbox.Add(self.lblamp)
mainbox.Add(self.button, flag=wx.ALIGN_RIGHT)
mainbox.Add(self.frecuencia, 1, wx.EXPAND)
mainbox.Add(self.potencia, 1, wx.EXPAND)
mainbox.Add(self.amplitud, 1, wx.EXPAND)
mainbox.Add(self.button2, flag=wx.ALIGN_RIGHT)

self.amplitud.Enable(False)

vbox.Add(mainbox)

self.mainSizer.Add(vbox, proportion=1, flag=wx.ALL | wx.EXPAND, border=15)
self.SetSizer(self.mainSizer)

```

```

        self.sizer = sizer
        sizer.Hide(self.box1)
        sizer.Hide(self.box2)
        sizer.Hide(self.box3)
        sizer.Hide(self.box4)
        sizer.Hide(self.box41)
        sizer.Hide(self.bs2)
        sizer.Hide(self.box5)
        sizer.Hide(self.box6)
        sizer.Hide(self.box7) # Buscar como poner el sizer con un alto minimo
        sizer.Hide(self.box8)
        sizer.Hide(self.box9)

        self.vbox = vbox

        self.Show(True) # Mostrar el panel

    '''Eventos'''
    self.Bind(wx.EVT_COMBOBOX, self.OnSelectMod, self.esquema)
    self.Bind(wx.EVT_COMBOBOX, self.OnSelectFuente, self.fuente)
    self.Bind(wx.EVT_BUTTON, self.OnClick, self.button)
    self.Bind(wx.EVT_BUTTON, self.OnStop, self.button2)
    self.Bind(wx.EVT_BUTTON, self.OnBrowse, self.video_button)
    self.Bind(wx.EVT_BUTTON, self.OnEscogerVector, self.vector_button)
    self.Bind(wx.EVT_RADIOBUTTON, self.OnRadio, self.rdpot)
    self.Bind(wx.EVT_RADIOBUTTON, self.OnRadio, self.rdamp)

'''Funcion para habilitar o deshabilitar Amplitud'''

def OnRadio(self, event):
    btn = event.GetEventObject()
    label = btn.GetLabel()
    if label == 'Potencia':
        self.potencia.Enable(True)
        self.amplitud.Enable(False)
    elif label == 'Amplitud':
        self.potencia.Enable(False)
        self.amplitud.Enable(True)

'''Funcion para buscar el archivo de onda arbitraria'''
```

```

def OnEscogerVector(self, e):
    if gtk.pygtk_version < (2, 3, 90):
        print "Se requiere PyGtk 2.3.90 para esta aplicación"
        raise SystemExit

    dialog = gtk.FileChooserDialog("Escoger Archivo..",
                                   None,
                                   gtk.FILE_CHOOSER_ACTION_OPEN,
                                   (gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL,
                                    gtk.STOCK_OPEN, gtk.RESPONSE_OK))
    dialog.set_default_response(gtk.RESPONSE_OK)

    filter = gtk.FileFilter()
    filter.set_name("Data Files")
    filter.add_pattern("*.dat")
    dialog.add_filter(filter)
    self.pathvec = ''

    response = dialog.run()
    if response == gtk.RESPONSE_OK:
        print dialog.get_filename(), 'seleccionado'
        self.pathvec = dialog.get_filename()
        self.vectorstr = self.leer_archivo(self.pathvec)
        self.flag = 1
    elif response == gtk.RESPONSE_CANCEL:
        print 'Closed, no files selected'
        self.vector.SetValue(self.pathvec)

    dialog.destroy()

'''Funcion para buscar el archivo de video'''

def OnBrowse(self, e):
    if gtk.pygtk_version < (2, 3, 90):
        print "Se requiere PyGtk 2.3.90 para esta aplicación"
        raise SystemExit

    esquema = self.esquema.GetCurrentSelection()

```

```

dialog = gtk.FileChooserDialog("Escoger Archivo..",
                               None,
                               gtk.FILE_CHOOSER_ACTION_OPEN,
                               (gtk.STOCK_CANCEL, gtk.RESPONSE_CANCEL,
                                gtk.STOCK_OPEN, gtk.RESPONSE_OK))
dialog.set_default_response(gtk.RESPONSE_OK)
if esquema < 10:
    self.filtro_video(dialog)
    self.filtro_imagen(dialog)
    self.filtro_texto(dialog)
elif esquema == 10:
    self.filtro_imagen(dialog)
elif esquema == 11 or esquema == 12:
    self.filtro_video(dialog)

self.path = ''

response = dialog.run()
if response == gtk.RESPONSE_OK:
    print dialog.get_filename(), 'seleccionado'
    self.path = dialog.get_filename()
    self.flag = 1
    if esquema == 10:
        convertir_imagen(self.path)
        pass

elif response == gtk.RESPONSE_CANCEL:
    print 'Closed, no files selected'
    self.video_path.SetValue(self.path)

dialog.destroy()

'''Funciones para controlar el campo de vector y de video'''
def OnSelectFuente(self, e):
    sel = self.fuente.GetValue()
    fuente = self.fuente.GetCurrentSelection()
    mod = self.esquema.GetValue()
    mod_num = self.esquema.GetCurrentSelection()
    if sel == 'Video/Archivo' or sel == 'Video' or sel == 'Imagen':
        self.mostrar_sel_video(True)

```

```

        self.mostrar_sel_vector(False)
        self.mostrar_udp(False)
        self.mostrar_opciones(False)
        self.habilitar_audio(True)

    elif sel == 'Arbitraria':
        self.mostrar_sel_vector(True)
        self.mostrar_sel_video(False)
        self.mostrar_udp(False)
        self.mostrar_opciones(False)
        self.habilitar_audio(False)

    if mod == 'AM-DSBSC':
        self.potencia.Enable(False)
        self.amplitud.Enable(True)
        self.habilitar_audio(True)

    elif mod == 'AM-LC':
        self.habilitar_audio(True)

    else:
        self.potencia.Enable(True)
        self.amplitud.Enable(False)

    elif sel == 'Audio':
        self.habilitar_audio(False)
        self.mostrar_sel_vector(False)
        self.mostrar_sel_video(False)
        self.mostrar_udp(False)
        self.mostrar_opciones(False)

    elif sel == 'Streaming':
        self.habilitar_audio(True)
        self.mostrar_sel_video(False)
        self.mostrar_sel_vector(False)
        self.mostrar_opciones(False)
        self.mostrar_udp(True)

    elif 0 <= fuente <= 4:
        if mod == 'AM-DSBSC':
            self.mostrar_opciones(True)
            if self.rdpot.GetValue():
                self.potencia.Enable(True)
                self.amplitud.Enable(False)
            else:
                self.rdpot.SetValue(True)
                self.potencia.Enable(True)

```

```

        self.amplitud.Enable(False)
    else:
        self.mostrar_opciones(False)
    if sel == 'Constante':
        self.habilitar_audio(False)
    else:
        self.habilitar_audio(True)

    self.mostrar_sel_vector(False)
    self.mostrar_sel_video(False)
    self.mostrar_udp(False)
else:
    self.mostrar_sel_vector(False)
    self.mostrar_sel_video(False)
    self.mostrar_udp(False)
    self.mostrar_opciones(False)
    self.habilitar_audio(True)

def mostrar_sel_video(self, mostrar):
    self.vbox.Show(self.boxVideo, mostrar)
    self.mainSizer.Layout()

def mostrar_sel_vector(self, mostrar):
    self.vbox.Show(self.boxVector, mostrar)
    self.mainSizer.Layout()

def mostrar_udp(self, mostrar):
    self.vbox.Show(self.boxUDP, mostrar)
    self.mainSizer.Layout()

def mostrar_opciones(self, mostrar):
    self.vbox.Show(self.boxOpciones, mostrar)
    self.mainSizer.Layout()

'''Funcion para determinar que esquema fue escogido'''
def OnSelectMod(self, e):
    sel = self.esquema.GetCurrentSelection()
    self.reset_combobox()
    self.reset_cajas()
    self.potencia.Enable(True)

```

```

        self.amplitud.Enable(False)
        self.path_reset(self.video_path)
        self.path_reset(self.vector)
        seleccion = {0: self.AMLC, 1: self.DSBSC, 2: self.FM, 3: self.FM,
                     4: self.ASK, 5: self.ASK, 7: self.ASK,
                     8: self.ASK, 6: self.PSK,
                     9: self.QAM, 13: self.OFDM,
                     10: self.TVN, 11: self.TVS, 12: self.TVT}

        seleccion[sel]() # Para determinar que controles habilitar

'''Funciones que deshabilitan y habilitan controles'''
def AMLC(self):
    self.fuente.Clear()
    self.fuente.Append('Constante')
    self.fuente.Append('Senoidal')
    self.fuente.Append('Cuadrada')
    self.fuente.Append('Triangular')
    self.fuente.Append('Sawtooth')
    self.fuente.Append('Audio')
    self.fuente.Append('Arbitraria')
    self.hide()

    ocultar(self, True, True, False, False, False, False, False)
    self.mostrar_opciones(False)
    self.mostrar_sel_video(False)
    self.mostrar_sel_vector(False)
    self.mostrar_udp(False)

def DSBSC(self):
    self.fuente.Clear()
    self.fuente.Append('Constante')
    self.fuente.Append('Senoidal')
    self.fuente.Append('Cuadrada')
    self.fuente.Append('Triangular')
    self.fuente.Append('Sawtooth')
    self.fuente.Append('Audio')
    self.fuente.Append('Arbitraria')
    self.hide()

```

```

        ocultar(self, True, False, False, False, False, False, False)
        self.mostrar_opciones(False)
        self.mostrar_sel_video(False)
        self.mostrar_sel_vector(False)
        self.mostrar_udp(False)

def FM(self):
    self.fuente.Clear()
    self.fuente.Append('Constante')
    self.fuente.Append('Senoidal')
    self.fuente.Append('Cuadrada')
    self.fuente.Append('Triangular')
    self.fuente.Append('Sawtooth')
    self.fuente.Append('Audio')
    self.fuente.Append('Arbitraria')
    self.hide()

        ocultar(self, True, False, True, False, False, False, False)
        self.mostrar_opciones(False)
        self.mostrar_sel_video(False)
        self.mostrar_sel_vector(False)
        self.mostrar_udp(False)

def ASK(self):
    self.fuente.Clear()
    self.fuente.Append('Aleatoria')
    self.fuente.Append('Audio')
    self.fuente.Append('Video/Archivo')
    self.fuente.Append('Streaming')
    self.hide()

        ocultar(self, False, False, False, True, False, False, False)
        self.mostrar_opciones(False)
        self.mostrar_sel_video(False)
        self.mostrar_sel_vector(False)
        self.mostrar_udp(False)

def GSK(self):
    self.fuente.Clear()
    self.fuente.Append('Aleatoria')

```

```

        self.fuente.Append('Audio')
        self.fuente.Append('Video/Archivo')
        self.fuente.Append('Streaming')
        self.hide()

        ocultar(self, False, False, False, True, False, False, False)
        self.mostrar_opciones(False)
        self.mostrar_sel_video(False)
        self.mostrar_sel_vector(False)
        self.mostrar_udp(False)

    def PSK(self):
        self.fuente.Clear()
        self.fuente.Append('Aleatoria')
        self.fuente.Append('Audio')
        self.fuente.Append('Video/Archivo')
        self.fuente.Append('Streaming')
        self.const.Clear()
        self.const.Append('2')
        self.const.Append('4')
        self.const.Append('8')
        self.const.Append('16')
        self.const.Append('32')
        self.grid1.Show(self.lblconst, True)
        self.grid1.Show(self.const, True)
        self.grid1.Hide(self.lblofdm)
        self.grid1.Hide(self.ofdm)

        ocultar(self, False, False, False, True, False, False, False)
        self.mostrar_opciones(False)
        self.mostrar_sel_video(False)
        self.mostrar_sel_vector(False)
        self.mostrar_udp(False)

    def QAM(self):
        self.fuente.Clear()
        self.fuente.Append('Aleatoria')
        self.fuente.Append('Audio')
        self.fuente.Append('Video/Archivo')
        self.fuente.Append('Streaming')

```

```

        self.const.Clear()
        self.const.Append('4')
        self.const.Append('16')
        self.const.Append('64')
        self.grid1.Show(self.lblconst, True)
        self.grid1.Show(self.const, True)
        self.grid1.Hide(self.lblofdm)
        self.grid1.Hide(self.ofdm)

        ocultar(self, False, False, False, True, False, False, False)
        self.mostrar_opciones(False)
        self.mostrar_sel_video(False)
        self.mostrar_sel_vector(False)
        self.mostrar_udp(False)

def OFDM(self):
    self_fuente.Clear()
    self_fuente.Append('Aleatoria')
    self.const.Clear()
    self.const.Append('4')
    self.const.Append('16')
    self.const.Append('64')
    self.grid1.Hide(self.lblconst)
    self.grid1.Hide(self.const)
    self.grid1.Show(self.lblofdm, True)
    self.grid1.Show(self.ofdm, True)

    ocultar(self, False, False, False, False, True, True, True, False)
    self.mostrar_opciones(False)
    self.mostrar_sel_video(False)
    self.mostrar_sel_vector(False)

def TVN(self):
    self_fuente.Clear()
    self_fuente.Append('Imagen')
    self.hide()

    ocultar(self, False, False, False, False, False, False, False, False)
    self.mostrar_opciones(False)
    self.mostrar_sel_vector(False)

```

```

        self.mostrar_udp(False)

    def TVS(self):
        self_fuente.Clear()
        self_fuente.Append('Video')
        self.hide()
        self.append_DVBS2(self.tvmod)
        self.code_rateDVBS2(self.coderate)
        ocultar(self, False, False, False, False, False, False, False, True)
        self.mostrar_opciones(False)
        self.mostrar_sel_vector(False)
        self.mostrar_udp(False)

    def TTV(self):
        self_fuente.Clear()
        self_fuente.Append('Video')
        self.hide()
        self.append_DVBT(self.tvmod)
        self.code_rateDVBT(self.coderate)
        ocultar(self, False, False, False, False, False, False, False, True)
        self.mostrar_opciones(False)
        self.mostrar_sel_vector(False)
        self.mostrar_udp(False)

    '''Funciones auxiliares para los controles'''
    def hide(self):
        self.grid1.Hide(self.lblconst)
        self.grid1.Hide(self.const)
        self.grid1.Hide(self.lblofdm)
        self.grid1.Hide(self.ofdm)

    def append_DVBS2(self, lista):
        lista.Clear()
        lista.Append('QPSK')
        lista.Append('8PSK')
        lista.Append('8APSK')
        lista.Append('16APSK')
        lista.Append('32APSK')

    def append_DVBT(self, lista):

```

```

        lista.Clear()
        lista.Append('QPSK')
        lista.Append('16QAM')
        lista.Append('64QAM')

def code_rateDVBS2(self, lista):
    lista.Clear()
    lista.Append('1/2')
    lista.Append('1/3')
    lista.Append('1/4')
    lista.Append('2/3')
    lista.Append('2/5')
    lista.Append('3/4')
    lista.Append('3/5')
    lista.Append('4/5')
    lista.Append('5/6')

def code_rateDVBT(self, lista):
    lista.Clear()
    lista.Append('1/2')
    lista.Append('2/3')
    lista.Append('3/4')
    lista.Append('5/6')
    lista.Append('7/8')

def OnClick(self, event):
    seleccion = self.esquema.GetCurrentSelection() # Obtener la modulacion
    fuente = self_fuente.GetCurrentSelection() # Obtener la fuente
    freq_txt = self.frecuencia.GetValue() # Obtener la frecuencia
    freq = float(freq_txt) # Convertir para los grc
    self.freq = freq
    if not (50 <= freq <= 3000):
        raise CarrierError
    if self.potencia.IsEnabled():
        pot_txt = self.potencia.GetValue()
        pot = float(pot_txt)
    elif self.amplitud.IsEnabled():
        pot = float(self.amplitud.GetValue())
    key, onda, param1, param2, param3 = self.Set_Key(seleccion, fuente)

```

```
import ctypes
import os

if os.name == 'posix':
    try:
        x11 = ctypes.cdll.LoadLibrary('libX11.so')
        x11.XInitThreads()
    except:
        print "Warning: failed to XInitThreads()"
parser = OptionParser(option_class=eng_option, usage="%prog: [options]")
(options, args) = parser.parse_args()

opciones = {0: AM_DSB_LC,
            1: AM_LC_Audio,
            2: AM_DSB_LC_vector,
            3: AM_DSB_SC,
            301: AM_DSB_SC_amplitud,
            4: AM_DSB_SC_audio,
            5: AM_DSB_SC_vector,
            6: NBFM,
            7: NBFM_Audio,
            8: NBFM_vector,
            9: WBFM,
            10: WBFM_Audio,
            11: WBFM_vector,
            12: GFSK_Random,
            13: GFSK_audio,
            14: GFSK_video,
            51: GFSK_UDP,
            16: GMSK_Random,
            17: GMSK_audio,
            18: GMSK_video,
            91: GMSK_UDP,
            20: PSK_Random,
            21: PSK_Audio,
            22: PSK_video,
            24: PSK_UDP,
            25: OOK,
            26: OOK_audio,
            27: OOK_video,
```

```

29: OOK_UDP,
30: QAM_Random,
31: QAM_Audio,
32: QAM_video,
34: QAM_UDP,
35: OFDM_Random,
39: ASK_Random,
40: ASK_Audio,
41: ASK_video,
43: ASK_UDP,
44: top_block_dvbs2,
45: top_block_dvbt,
46: ntsc_hackrf,
}

self.tb = opciones[key](onda, freq, pot, param1, param2, param3, self)
self.button2.Enable(True)
self.button.Enable(False)
self.tb.Start(True)
self.habilitar_cajas(False)
self.tb.Wait()

def OnStop(self, e):
    print 'Presione el botón "Reset" en la HackRF One'
    try:
        self.tb._frame.Close(True)
    except:
        print 'Ventana cerrada'
    finally:
        self.button2.Enable(False)
        self.button.Enable(True)
        self.habilitar_cajas(True)
        self.OnSelectFuente(e)
        self.cuadro.SetLabel("0.0 MHz : -80.0 dBm")
        #self.reset_cajas()
        #self.reset_combobox()
        self.path_reset(self.video_path)
        self.path_reset(self.vector)

def reset_cajas(self):

```

```

        self.fm.SetValue('')
        self.desv.SetValue('')
        self.ind_mod.SetValue('')
        self.bt.SetValue('')
        self.bitrate.SetValue('')
        self.fftlen.SetValue('')
        self.canales.SetValue('')
        self.prefijo.SetValue('')
        self.frecuencia.SetValue('')
        self.potencia.SetValue('')
        self.amplitud.SetValue('')
        self.dir.SetValue('')
        self.port.SetValue('')

    def reset_combobox(self):
        self_fuente.SetValue('Fuente')
        self_const.SetValue('Constelación')
        self_ofdm.SetValue('Opción')
        self_tvmod.SetValue('Modulación')
        self_coderate.SetValue('Tasa')

    def Set_Key(self, seleccion, fuente):
        global key
        onda = analog.GR_CONST_WAVE
        param1 = 0
        param2 = 0
        param3 = 0
        key = 666

        if seleccion == 0: # AM-LC
            param2 = self.get_indmod()
            if 0 <= fuente <= 4:
                key = 0
                onda = self.Escoger_Fuente(fuente)
                if self.fm.IsEnabled():
                    param1 = self.get_freqmod()
            elif fuente == 5: # Audio
                key = 1
            elif fuente == 6: # Vector
                key = 2
        else:
            if 0 <= fuente <= 4:
                key = 0
                onda = self.Escoger_Fuente(fuente)
                if self.fm.IsEnabled():
                    param1 = self.get_freqmod()
            elif fuente == 5: # Audio
                key = 1
            elif fuente == 6: # Vector
                key = 2

```

```

        onda = self.convertir_vector(self.vectorstr)
        len_vec = self.get_len_vec(onda)
        param1 = self.get_freqmod()
        param3 = len_vec * param1
        if param3 > 20e6:
            raise ArbitrariaError

    elif seleccion == 1: # AM DSB-SC
        if 0 <= fuente <= 4:
            if self.potencia.IsEnabled():
                key = 3
            elif self.amplitud.IsEnabled():
                key = 301
            onda = self.Escoger_Fuente(fuente)
            if self.fm.IsEnabled():
                param1 = self.get_freqmod()
        elif fuente == 5: # Audio
            key = 4
        elif fuente == 6: # Vector
            key = 5
            onda = self.convertir_vector(self.vectorstr)
            len_vec = self.get_len_vec(onda)
            param1 = self.get_freqmod()
            param3 = len_vec * param1
            if param3 > 20e6:
                raise ArbitrariaError

    elif seleccion == 2: # NBFM
        if 0 <= fuente <= 4:
            key = 6
            onda = self.Escoger_Fuente(fuente)
            param1 = self.get_freqFM(seleccion)
        elif fuente == 5: # Audio
            key = 7
        elif fuente == 6: # Vector
            key = 8
            onda = self.convertir_vector(self.vectorstr)

param2 = self.get_maxdesv(seleccion)

```

```

elif seleccion == 3: # WBFM
    if 0 <= fuente <= 4:
        key = 9
        onda = self.Escoger_Fuente(fuente)
        param1 = self.get_freqFM(seleccion)
    elif fuente == 5: # Audio
        key = 10
    elif fuente == 6: # Vector
        key = 11
        onda = self.convertir_vector(self.vectorstr)

param2 = self.get_maxdesv(seleccion)

elif seleccion == 7: # GFSK
    if fuente == 0:
        key = 12 # GFSK Random Source
    elif fuente == 1:
        key = 13 # Audio
    elif fuente == 2:
        key = 14 # Video
        onda = str(self.path)
    elif fuente == 3:
        key = 51 # Streaming
        onda = self.get_direccion()
        param2 = self.get_puerto()
        param1 = self.get_excessbw()
        if self.bitrate.IsEnabled():
            param3 = float(self.bitrate.GetValue())

elif seleccion == 8: # GMSK
    if fuente == 0:
        key = 16 # GMSK Random Source
    elif fuente == 1:
        key = 17 # Audio
    elif fuente == 2:
        key = 18 # Video
        onda = str(self.path)
    elif fuente == 3:
        key = 91 # Streaming
        onda = self.get_direccion()

```

```

        param2 = self.get_puerto()
param1 = self.get_excessbw()
if self.bitrate.IsEnabled():
    param3 = float(self.bitrate.GetValue())

elif seleccion == 6: # PSK
    param2 = {}
    param1 = self.get_excessbw()
    param2[0] = int(self.const.GetValue())
    if self.bitrate.IsEnabled():
        param3 = float(self.bitrate.GetValue())
    if fuente == 0:
        key = 20 # PSK Random Source
    elif fuente == 1:
        key = 21 # Audio
    elif fuente == 2:
        key = 22 # Video
        onda = str(self.path)
    elif fuente == 3:
        key = 24 # Streaming
        onda = self.get_direccion()
        param2[1] = self.get_puerto()

elif seleccion == 9: # QAM
    param2 = {}
    param1 = self.get_excessbw()
    param2[0] = int(self.const.GetValue())
    if self.bitrate.IsEnabled():
        param3 = float(self.bitrate.GetValue())
    if fuente == 0:
        key = 30 # QAM Random Source
    elif fuente == 1:
        key = 31 # Audio
    elif fuente == 2:
        key = 32 # Video
        onda = str(self.path)
    elif fuente == 3:
        key = 34 # Streaming
        onda = self.get_direccion()
        param2[1] = self.get_puerto()

```

```

        elif seleccion == 4: # 00K
            if fuente == 0:
                key = 25 # 00K Random Source
            elif fuente == 1:
                key = 26 # Audio
            elif fuente == 2:
                key = 27 # Video
                onda = str(self.path)
            elif fuente == 3:
                key = 29 # Streaming
                onda = self.get_direccion()
                param2 = self.get Puerto()
                param1 = self.get_excessbw()
                if self.bitrate.IsEnabled():
                    param3 = float(self.bitrate.GetValue())
            elif seleccion == 5: # ASK
                if fuente == 0:
                    key = 39 # ASK Random Source
                elif fuente == 1:
                    key = 40 # Audio
                elif fuente == 2:
                    key = 41 # Video
                    onda = str(self.path)
                elif fuente == 3:
                    key = 43 # Streaming
                    onda = self.get_direccion()
                    param2 = self.get Puerto()
                    param1 = self.get_excessbw()
                    if self.bitrate.IsEnabled():
                        param3 = float(self.bitrate.GetValue())
            elif seleccion == 13: # OFDM
                param1 = self.get_fftlen()
                param2 = self.get_canales(param1)
                param3 = self.get_prefijo(param1)
                self.get_canales_comprobar(param2, param3, param1)
                onda = []
                onda.append(str(self.ofdm.GetValue()).lower())

```

```

        if onda[0] == 'opción':
            raise OpcionOFDMError
        if fuente == 0:
            key = 35 # OFDM Random Source

    elif seleccion == 11: # DVBS2
        key = 44 # Video
        if self.flag == 0:
            raise ValueError
        onda = str(self.path)
        param1 = self.get_tvmod()
        param2 = self.get_tvcoderate()
        if not(1000 <= self.frec <= 2000):
            raise FrecS2Error

    elif seleccion == 10: # NTSC
        key = 46
        if self.flag == 0:
            raise ValueError

    elif seleccion == 12: # DVB-T
        key = 45
        if self.flag == 0:
            raise ValueError
        onda = str(self.path)
        param1 = self.get_tvmod()
        param2 = self.get_tvcoderate()

    if key == 666:
        raise NotKeyError

    return key, onda, param1, param2, param3

def Escoger_Fuente(self, fuente):
    if fuente == 0:
        onda = analog.GR_CONST_WAVE
    elif fuente == 1:
        onda = analog.GR_SIN_WAVE
    elif fuente == 2:
        onda = analog.GR_SQR_WAVE

```

```

        elif fuente == 3:
            onda = analog.GR_TRI_WAVE
        elif fuente == 4:
            onda = analog.GR_SAW_WAVE
        return onda

    def convertir_vector(self, vector):
        onda = vector
        onda = onda.split(',')
        lista = []
        for i in onda:
            x = float(i)
            lista.append(x)
        onda = lista
        return onda

    def get_len_vec(self, onda):
        return len(onda)

    def leer_archivo(self, archivo):
        fileobj = open(archivo)
        vector = fileobj.read()
        fileobj.close()
        return vector

    def habilitar_audio(self, boole):
        self.fm.Enable(boole)
        self.bitrate.Enable(boole)

    def get_freqmod(self):
        valor = float(self.fm.GetValue()) * 1e3
        if not (5000 <= valor <= 1000000):
            raise FreqModError
        return valor

    def get_indmod(self):
        valor = float(self.ind_mod.GetValue())
        if not (0 <= valor <= 1):
            raise IndModError
        return valor

```

```

def get_freqFM(self, sel):
    valor = float(self.fm.GetValue()) * 1e3
    if sel == 2:
        if not (5000 <= valor <= 50000):
            raise NbfmModError
        return valor
    elif sel == 3:
        if not (5000 <= valor <= 100000):
            raise WbfmModError
        return valor

def get_maxdesv(self, sel):
    valor = float(self.desv.GetValue()) * 1e3
    if sel == 2:
        limite = 25000
    elif sel == 3:
        limite = 100000
    if not (100 < valor <= limite):
        raise MaxDesvError
    return valor

def get_excessbw(self):
    valor = float(self.bt.GetValue())
    if not (0 <= valor <= 1):
        raise ExcessBWError
    return valor

def get_bitrate1(self):
    valor = float(self.bitrate.GetValue()) * 1e3
    if not (100e3 <= valor <= 4000000):
        raise BitRate1Error
    return valor

def get_bitrate2(self):
    valor = float(self.bitrate.GetValue()) * 1e3
    if not (100e3 <= valor <= 8000000):
        raise BitRate2Error
    return valor

```

```
def get_direccion(self):
    valor = str(self.dir.GetValue())
    return valor

def get_puerto(self):
    valor = int(self.port.GetValue())
    return valor

def get_tvmod(self):
    valor = self.tvmod.GetValue()
    if valor == 'Modulación':
        raise TVModError
    return valor

def get_tvcodeRate(self):
    valor = self.coderate.GetValue()
    if valor == 'Tasa':
        raise TVCodeRateError
    return valor

def get_fftlen(self):
    valor = float(self.fftlen.GetValue())
    if not (64 <= valor <= 2048):
        raise FlenError
    log = math.log(valor,2)
    if int(log) != log:
        if valor == 1536:
            pass
        else:
            raise FlenError
    return int(valor)

def get_canales(self, fftlen):
    valor = float(self.canales.GetValue())
    if valor != int(valor):
        raise CanalesError
    if not (0 < valor < fftlen):
        raise CanalesError
    return int(valor)
```

```

def get_prefijo(self, fftlen):
    valor = float(self.prefijo.GetValue())
    if valor != int(valor):
        raise CanalesError
    if not (0 < valor < fftlen):
        raise CanalesError
    return int(valor)

def get_canales_comprobar(self,param2,param3,fftlen):
    suma = param2 + param3
    bw = 100 * float(param2) / fftlen
    if suma > fftlen:
        raise OFDMError
    elif bw < 20:
        raise OFDMbwError

def habilitar_cajas(self, bool):
    self.fm.Enable(bool)
    self.ind_mod.Enable(bool)
    self.desv.Enable(bool)
    self.bt.Enable(bool)
    self.bitrate.Enable(bool)
    self.dir.Enable(bool)
    self.port.Enable(bool)
    return

def path_reset(self, path):
    path.SetValue('')
    self.pathvec = ''
    self.flag = 0

def filtro_video(self, dialog):
    filter = gtk.FileFilter()
    filter.set_name("Video")
    filter.add_pattern("*.ts")
    dialog.add_filter(filter)

def filtro_imagen(self, dialog):
    filter = gtk.FileFilter()
    filter.set_name("Imagen")

```

```
filter.add_mime_type("image/png")
filter.add_mime_type("image/jpeg")
filter.add_mime_type("image/gif")
filter.add_pattern("*.png")
filter.add_pattern("*.jpg")
filter.add_pattern("*.gif")
dialog.add_filter(filter)

def filtro_texto(self, dialog):
    filter = gtk.FileFilter()
    filter.set_name("Texto")
    filter.add_pattern("*.txt")
    dialog.add_filter(filter)

class CarrierError(Exception):
    pass

class IndModError(Exception):
    pass

class FreqModError(Exception):
    pass

class ArbitrariaError(Exception):
    pass

class NbfmModError(Exception):
    pass

class WbfmModError(Exception):
    pass

class MaxDesvError(Exception):
    pass

class ExcessBWError(Exception):
    pass

class BitRate1Error(Exception):
    pass
```

```
class BitRate2Error(Exception):
    pass

class NTSError(Exception):
    pass

class TVModError(Exception):
    pass

class TVCodeRateError(Exception):
    pass

class FrecS2Error(Exception):
    pass

class CanalesError(Exception):
    pass

class FlenError(Exception):
    pass

class OFDMError(Exception):
    pass

class OpcionOFDMError(Exception):
    pass

class NotKeyError(Exception):
    pass

class OFDMbwError(Exception):
    pass

class ExceptionDialog(GMD.GenericMessageDialog):
    """
    # -----
    def __init__(self, msg):
        """Constructor"""
        GMD.GenericMessageDialog.__init__(self, None, msg, "Error",
                                         "OK")
    """

    def _onOK(self):
        self.parent().close()

```

```

wx.OK | wx.ICON_ERROR)

# -----
def MyExceptionHook(etype, value, trace):
    """
    Handler for all unhandled exceptions.

    :param 'etype': the exception type ('SyntaxError', 'ZeroDivisionError', etc...);
    :type 'etype': 'Exception'
    :param string 'value': the exception error message;
    :param string 'trace': the traceback header, if any (otherwise, it prints the
    standard Python header: "Traceback (most recent call last)".
    """

    frame = wx.GetApp().GetTopWindow()
    tmp = traceback.format_exception(etype, value, trace)
    exception = "".join(tmp)
    if etype == ValueError:
        exception = "Por favor asegurarse de llenar los campos de forma correcta"
    elif etype == CarrierError:
        exception = "La frecuencia portadora debe estar entre 50 y 3000 MHz"
    elif etype == FreqModError:
        exception = "La frecuencia modulante debe ser un valor positivo entre 5 y 1000 KHz"
    elif etype == IndModError:
        exception = "Indice de modulacion fuera de rango, debe estar comprendido entre 0 y 1"
    elif etype == NbfmModError:
        exception = "La frecuencia modulante para NBFM debe
ser un valor positivo menor o igual a 50 KHz"
    elif etype == WbfmModError:
        exception = "La frecuencia modulante para WBFM debe
ser un valor positivo menor o igual a 100 KHz"
    elif etype == MaxDesvError:
        exception = """El valor 'Máxima desviación de frecuencia' está fuera de rango,
debe ser menor a 25000 para NBFM y menor a 100000 para WBFM"""
    elif etype == ExcessBWError:
        exception = "El exceso de ancho de banda debe ser un valor real entre 0 y 1"
    elif etype == BitRate1Error:
        exception = "La tasa de bits debe ser un valor entre 100 y 4000 kbps"
    elif etype == BitRate2Error:
        exception = "La tasa de bits debe ser un valor entre 100 y 8000 kbps"
    elif etype == NTSCError:

```

```

        exception = """La frecuencia debe corresponder a un canal de TV analógico,
consulte la ayuda para más información"""

    elif etype == TVModError:
        exception = "Por favor seleccione un tipo de modulación para el esquema"

    elif etype == TVCodeRateError:
        exception = "Por favor seleccione una tasa de codificación para el esquema"

    elif etype == FrecS2Error:
        exception = "La frecuencia portadora para DVB-S2 debe estar entre 1000 y 2000 MHz"

    elif etype == FlenError:
        exception = """Error al introducir 'Longitud FFT',
debe ser un valor entero positivo
potencia de 2 comprendido entre 64 y 1024"""

    elif etype == OFDMError:
        exception = """Error al introducir canales ocupados y prefijo ciclico,
la suma debe ser menor a la longitud FFT"""

    elif etype == OpcionOFDMError:
        exception = "Debe colocar una opción de modulación"

    elif etype == CanalesError:
        exception = """La cantidad de canales ocupados y el prefijo cílico deben ser
un valor entero positivo menor a la longitud FFT"""

    elif etype == NotKeyError:
        exception = "Debe escoger una fuente disponible para el esquema seleccionado"

    elif etype == ArbitrariaError:
        exception = """Debe colocar un vector con menos muestras o
una frecuencia modulante más baja,
consulte la ayuda para más información"""

    elif etype == OFDMbwError:
        exception = "El porcentaje de canales ocupados
respecto a la FFT debe ser mayor a 20%"

    elif etype == RuntimeError:
        exception = """Ocurrió un RunTimeError, asegúrese
que la dirección IP corresponda a la
dirección del cliente"""

    dlg = ExceptionDialog(exception)
    dlg.ShowModal()
    dlg.Destroy()

```

Listado de programa AM_DSB_LC_audio.py

`#!/usr/bin/env python`

```

#####
# Gnuradio Python Flow Graph
# Title: Audio AM
# Generated: Sat May  9 15:23:51 2015
#####

from gnuradio import analog
from gnuradio import audio
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import filter
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import wx

class AM_LC_Audio(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self, title="Audio AM")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.samp_rate = samp_rate = 480e3
self.potencia = potencia = float(int(pot))
self.modulation_index = modulation_index = param2
self.if_gain = if_gain = 20
self.carrier_freq = carrier_freq = freq * 1e6
self.corr = corr = -8

```

```

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label='Frecuencia (Hz)',
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
self.rational_resampler_xxx_0 = filter.rational_resampler_ccc(
    interpolation=480000,
    decimation=48000,
    taps=None,
    fractional_bw=None,
)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia (dBm)",
    converter=forms.float_converter(),
)

```

```

        proportion=0,
    )
    self._potencia_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        minimum=-10,
        maximum=5,
        num_steps=15,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_potencia_sizer)
    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(480e3)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(corr, 0)
    self.osmosdr_sink_0.set_gain(7, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)
    self.osmosdr_sink_0.set_bb_gain(20, 0)
    self.osmosdr_sink_0.set_antenna("", 0)
    self.osmosdr_sink_0.set_bandwidth(0, 0)

    self.set_potencia(potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

    self.low_pass_filter_0 = filter.fir_filter_fff(1, firdes.low_pass(
        5, 48000, 22000, 10000, firdes.WIN_HAMMING, 6.76))
    self.blocks_null_source_0 = blocks.null_source(gr.sizeof_float * 1)
    self.blocks_multiply_const_vxx_0 =
    blocks.multiply_const_vcc((modulation_index, ))
    self.blocks_float_to_complex_0 = blocks.float_to_complex(1)
    self.blocks_add_xx_0 = blocks.add_vcc(1)
    self.audio_source_0 = audio.source(48000, "", True)
    self.analog_const_source_x_0 =
    analog.sig_source_c(0, analog.GR_CONST_WAVE, 0, 0, 0.3)

```

```

#####
# Connections
#####
self.connect((self.analog_const_source_x_0, 0), (self.blocks_add_xx_0, 1))
self.connect((self.rational_resampler_xxx_0, 0), (self.osmosdr_sink_0, 0))
self.connect((self.blocks_add_xx_0, 0), (self.rational_resampler_xxx_0, 0))
self.connect((self.audio_source_0, 0),
(self.low_pass_filter_0, 0))
    self.connect((self.low_pass_filter_0, 0),
(self.blocks_float_to_complex_0, 0))
        self.connect((self.blocks_null_source_0, 0),
(self.blocks_float_to_complex_0, 1))
            self.connect((self.blocks_multiply_const_vxx_0, 0),
(self.blocks_add_xx_0, 0))
                self.connect((self.blocks_float_to_complex_0, 0),
(self.blocks_multiply_const_vxx_0, 0))

# QT sink close method reimplementation

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    frecuencia = self.carrier_freq
    if self.potencia < -10:
        self.potencia = -10
    if frecuencia < 2300e6 or frecuencia >= 2940e6:
        if self.potencia >= -5:
            self.potencia = -5
    if frecuencia > 2740e6:
        if self.potencia >= -10:
            self.potencia = -10

```

```

        self._potencia_slider.set_value(self.potencia)
        self._potencia_text_box.set_value(self.potencia)
        self.if_gain = get_ajuste(self)
        self.if_gain += 6
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
        actualizar_label(self, self.parent, self.potencia)

    def get_modulation_index(self):
        return self.modulation_index

    def set_modulation_index(self, modulation_index):
        self.modulation_index = modulation_index
        self.blocks_multiply_const_vxx_0.set_k((self.modulation_index, ))

    def get_if_gain(self):
        return self.if_gain

    def set_if_gain(self, if_gain):
        self.if_gain = if_gain
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

    def get_carrier_freq(self):
        return self.carrier_freq

    def set_carrier_freq(self, carrier_freq):
        self.carrier_freq = carrier_freq
        if self.carrier_freq < 50e6:
            self.carrier_freq = 50e6
        elif self.carrier_freq > 3e9:
            self.carrier_freq = 3e9
        if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):

```

```

        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

```

Listado de programa AM_DSB_LC_vector.py

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-

#####
# Gnuradio Python Flow Graph
# Title: AM Señal Arbitraria
# Generated: Sat May  9 15:28:23 2015
#####

from gnuradio import analog
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio import wxgui
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from gnuradio.wxgui import scopesink2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import wx

class AM_DSB_LC_vector(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):

```

```

grc_wxgui.top_block_gui.__init__(self, title="AM Señal Arbitraria")
    _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
    self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.vector = vector = onda
self.samp_rate = samp_rate = 12e6
self.potencia = potencia = float(int(pot))
self.modulation_index = modulation_index = param2
self.mod_freq = mod_freq = param1
self.if_gain = if_gain = 20
self.corr = corr = -8
self.carrier_freq = carrier_freq = freq * 1e6

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label='Frecuencia',
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
)

```

```

        proportion=1,
    )
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    minimum=-10,
    maximum=5,
    num_steps=15,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_potencia_sizer)
self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
self.osmosdr_sink_0.set_sample_rate(samp_rate)
self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
self.osmosdr_sink_0.set_freq_corr(corr, 0)
self.osmosdr_sink_0.set_gain(7, 0)
self.osmosdr_sink_0.set_if_gain(if_gain, 0)
self.osmosdr_sink_0.set_bb_gain(20, 0)
self.osmosdr_sink_0.set_antenna("", 0)
self.osmosdr_sink_0.set_bandwidth(0, 0)

self.set_potencia(potencia)
set_corr(self)
actualizar_label(self, self.parent, self.potencia)

```

```

        self.blocks_vector_source_x_0 =
blocks.vector_source_f(vector, True, 1, [])
        self.blocks_null_source_0 = blocks.null_source(gr.sizeof_float * 1)
self.blocks_multiply_const_vxx_0 = blocks.multiply_const_vff(
    ((1.0 / max(abs(i) for i in vector)) * 0.5 * modulation_index, ))
self.blocks_float_to_complex_0 = blocks.float_to_complex(1)
self.blocks_add_xx_0 = blocks.add_vff(1)
self.analog_const_source_x_0 =
analog.sig_source_f(0, analog.GR_CONST_WAVE, 0, 0, 0.5)

#####
# Connections
#####
self.connect((self.blocks_float_to_complex_0, 0),
(self.osmosdr_sink_0, 0))
    self.connect((self.blocks_add_xx_0, 0),
(self.blocks_float_to_complex_0, 0))
    self.connect((self.analog_const_source_x_0, 0),
(self.blocks_add_xx_0, 1))
    self.connect((self.blocks_vector_source_x_0, 0),
(self.blocks_multiply_const_vxx_0, 0))
    self.connect((self.blocks_multiply_const_vxx_0, 0),
(self.blocks_add_xx_0, 0))
    self.connect((self.blocks_null_source_0, 0),
(self.blocks_float_to_complex_0, 1))

# QT sink close method reimplementation

def get_vector(self):
    return self.vector

def set_vector(self, vector):
    self.vector = vector
    self.blocks_multiply_const_vxx_0.set_k(
    ((1.0 / max(abs(i) for i in self.vector)) * 0.5 * self.modulation_index, ))

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):

```

```

        self.samp_rate = samp_rate
        self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

    def get_potencia(self):
        return self.potencia

    def set_potencia(self, potencia):
        self.potencia = float(int(potencia))
        frecuencia = self.carrier_freq
        if self.potencia < -10:
            self.potencia = -10
        if frecuencia < 2300e6 or frecuencia >= 2940e6:
            if self.potencia >= -5:
                self.potencia = -5
        if frecuencia > 2740e6:
            if self.potencia >= -10:
                self.potencia = -10
        self._potencia_slider.set_value(self.potencia)
        self._potencia_text_box.set_value(self.potencia)
        self.if_gain = get_ajuste(self)
        self.if_gain += 6
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
        actualizar_label(self, self.parent, self.potencia)

    def get_modulation_index(self):
        return self.modulation_index

    def set_modulation_index(self, modulation_index):
        self.modulation_index = modulation_index
        self.blocks_multiply_const_vxx_0.set_k(
            ((1.0 / max(abs(i) for i in self.vector)) * 0.5 * self.modulation_index,))

    def get_mod_freq(self):
        return self.mod_freq

    def set_mod_freq(self, mod_freq):
        self.mod_freq = mod_freq

    def get_if_gain(self):
        return self.if_gain

```

```

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

```

Listado de programa AM_DSB_LC.py

```
#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: AM DSB-LC
# Generated: Tue Apr 28 10:18:55 2015
#####

from gnuradio import analog
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import wx

class AM_DSB_LC(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self, title="AM DSB-LC")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
        self.parent = parent
        self.samp_rate = samp_rate = param1 * 20
        self.mod_freq_0 = mod_freq_0 = param1
        self.if_gain = if_gain = 20
        self.carrier_freq = carrier_freq = freq * 1e6
        self.potencia = potencia = float(int(pot))
```

```
self.corr = corr = -8

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label="Frecuencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
```

```

        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        minimum=-10,
        maximum=5,
        num_steps=15,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_potencia_sizer)
    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(corr, 0)
    self.osmosdr_sink_0.set_gain(7, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)
    self.osmosdr_sink_0.set_bb_gain(20, 0)
    self.osmosdr_sink_0.set_antenna("", 0)
    self.osmosdr_sink_0.set_bandwidth(0, 0)

    self.set_potencia(potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

    self.blocks_float_to_complex_0 = blocks.float_to_complex(1)
    self.blocks_add_xx_0 = blocks.add_vff(1)
    self.analog_sig_source_x_0 =
analog.sig_source_f(samp_rate, onda, mod_freq_0, 0.5 * param2, 0)
    self.analog_const_source_x_0 =
analog.sig_source_f(0, analog.GR_CONST_WAVE, 0, 0, 0.5)

#####
# Connections
#####
self.connect((self.analog_const_source_x_0, 0),
(self.blocks_add_xx_0, 1))
    self.connect((self.analog_sig_source_x_0, 0),
(self.blocks_add_xx_0, 0))
    self.connect((self.blocks_add_xx_0, 0),

```

```

(self.blocks_float_to_complex_0, 0))
    self.connect((self.blocks_float_to_complex_0, 0),
(self.osmosdr_sink_0, 0))

# QT sink close method reimplementation

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.analog_sig_source_x_0.set_sampling_freq(self.samp_rate)
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    frecuencia = self.carrier_freq
    if self.potencia < -10:
        self.potencia = -10
    if frecuencia < 2300e6 or frecuencia >= 2940e6:
        if self.potencia >= -5:
            self.potencia = -5
    if frecuencia > 2740e6:
        if self.potencia >= -10:
            self.potencia = -10
    #self.potencia = int(self.potencia)
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = get_ajuste(self)
    self.if_gain += 6
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
    actualizar_label(self, self.parent, self.potencia)

def get_mod_freq_0(self):
    return self.mod_freq_0

def set_mod_freq_0(self, mod_freq_0):

```

```
    self.mod_freq_0 = mod_freq_0
    self.analog_sig_source_x_0.set_frequency(self.mod_freq_0)

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = get_ajuste(self) + 6
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)
```

Listado de programa AM_DSB_SC_amplitud.py

```
#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: AM Portadora Suprimida
# Generated: Thu May 21 18:51:16 2015
#####

from gnuradio import analog
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import math
import wx

class AM_DSB_SC_amplitud(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self, title="AM Portadora Suprimida")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.amp = amp = pot # amplitud con el parametro de potencia
self.samp_rate = samp_rate = param1 * 20
self.mult = mult = (amp - 30) / 400
self.mod_freq = mod_freq = param1
self.if_gain = if_gain = 21
```

```

        self.corr = corr = -8
        self.carrier_freq = carrier_freq = freq * 1e6
        self.onda = onda
        self.potencia = -5
        self.pot = self.calcular_potencia(self.mult)

#####
# Blocks
#####

_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label='Frecuencia',
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
self.osmosdr_sink_0.set_sample_rate(samp_rate)
self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
self.osmosdr_sink_0.set_freq_corr(corr, 0)
self.osmosdr_sink_0.set_gain(7, 0)
self.osmosdr_sink_0.set_if_gain(if_gain, 0)
self.osmosdr_sink_0.set_bb_gain(16, 0)
self.osmosdr_sink_0.set_antenna("", 0)

```

```

        self.osmosdr_sink_0.set_bandwidth(0, 0)

        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.pot)

        self.blocks_null_source_0 = blocks.null_source(gr.sizeof_float * 1)
        self.blocks_multiply_const_vxx_1 = blocks.multiply_const_vff((mult, ))
        self.blocks_float_to_complex_0 = blocks.float_to_complex(1)
        self.analog_sig_source_x_0 =
analog.sig_source_f(samp_rate, onda, mod_freq, 1, 0)
        _amp_sizer = wx.BoxSizer(wx.VERTICAL)
        self._amp_text_box = forms.text_box(
            parent=self.GetWin(),
            sizer=_amp_sizer,
            value=self.amp,
            callback=self.set_amp,
            label="Amplitud",
            converter=forms.float_converter(),
            proportion=0,
        )
        self._amp_slider = forms.slider(
            parent=self.GetWin(),
            sizer=_amp_sizer,
            value=self.amp,
            callback=self.set_amp,
            minimum=110,
            maximum=430,
            num_steps=16,
            style=wx.SL_HORIZONTAL,
            cast=float,
            proportion=1,
        )
        self.Add(_amp_sizer)

#####
# Connections
#####
self.connect((self.blocks_float_to_complex_0, 0),
(self.osmosdr_sink_0, 0))

```

```

        self.connect((self.blocks_null_source_0, 0),
(self.blocks_float_to_complex_0, 1))
        self.connect((self.analog_sig_source_x_0, 0),
(self.blocks_multiply_const_vxx_1, 0))
        self.connect((self.blocks_multiply_const_vxx_1, 0),
(self.blocks_float_to_complex_0, 0))

# QT sink close method reimplementation

def get_amp(self):
    return self.amp

def set_amp(self, amp):
    self.amp = amp
    self.set_mult((self.amp - 30) / 400)
    self._amp_slider.set_value(self.amp)
    self._amp_text_box.set_value(self.amp)
    self.pot = self.calcular_potencia(self.mult)
    actualizar_label(self, self.parent, self.pot)

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.analog_sig_source_x_0.set_sampling_freq(self.samp_rate)
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_mult(self):
    return self.mult

def set_mult(self, mult):
    self.mult = mult
    self.blocks_multiply_const_vxx_1.set_k((self.mult, ))

def get_mod_freq(self):
    return self.mod_freq

def set_mod_freq(self, mod_freq):

```

```

        self.mod_freq = mod_freq
        self.analog_sig_source_x_0.set_frequency(self.mod_freq)

    def get_if_gain(self):
        return self.if_gain

    def set_if_gain(self, if_gain):
        self.if_gain = if_gain
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

    def get_corr(self):
        return self.corr

    def set_corr(self, corr):
        self.corr = corr
        self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

    def set_potencia(self, potencia):
        self.potencia = potencia
        self.if_gain = get_ajuste(self)
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

    def get_carrier_freq(self):
        return self.carrier_freq

    def set_carrier_freq(self, carrier_freq):
        self.carrier_freq = carrier_freq
        if self.carrier_freq < 50e6:
            self.carrier_freq = 50e6
        elif self.carrier_freq > 3e9:
            self.carrier_freq = 3e9
        if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)

```

```

        elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        else:
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
            self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
            self.set_potencia(self.potencia)
            set_corr(self)
            actualizar_label(self, self.parent, self.pot)

def calcular_potencia(self, mult):
    self.mult = mult
    cal = 0
    if self.onda == analog.GR_CONST_WAVE:
        cal = 1
    elif self.onda == analog.GR_SIN_WAVE:
        cal = math.sqrt(2)
    elif self.onda == analog.GR_SQR_WAVE:
        cal = math.sqrt(2)
    elif self.onda == analog.GR_TRI_WAVE:
        cal = math.sqrt(3)
    elif self.onda == analog.GR_SAW_WAVE:
        cal = math.sqrt(3)
    rms = mult * 0.1257433 / cal
    print rms
    rms_sqr = rms * rms
    potencia = rms_sqr / 50
    potencia *= 1000
    potencia = 10 * math.log10(potencia)
    potencia = round(potencia, 2)
    print potencia
    return potencia

```

Listado de programa AM_DSB_SC_audio.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: Audio AM Portadora Suprimida

```

```

# Generated: Sat May  9 15:52:21 2015
#####
from gnuradio import audio
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import filter
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import wx

class AM_DSB_SC_audio(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self, title="Audio AM Portadora Suprimida")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.samp_rate = samp_rate = 480e3
self.potencia = potencia = pot
self.if_gain = if_gain = 21
self.corr = corr = -8
self.carrier_freq = carrier_freq = freq * 1e6

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(

```

```

parent=self.GetWin(),
sizer=_carrier_freq_sizer,
value=self.carrier_freq,
callback=self.set_carrier_freq,
label='Frecuencia',
converter=forms.float_converter(),
proportion=0,
)
self._carrier_freq_slider = forms.slider(
parent=self.GetWin(),
sizer=_carrier_freq_sizer,
value=self.carrier_freq,
callback=self.set_carrier_freq,
minimum=50e6,
maximum=3e9,
num_steps=295,
style=wx.SL_HORIZONTAL,
cast=float,
proportion=1,
)
self.Add(_carrier_freq_sizer)
self.rational_resampler_xxx_0 = filter.rational_resampler_ccc(
interpolation=480000,
decimation=48000,
taps=None,
fractional_bw=None,
)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
parent=self.GetWin(),
sizer=_potencia_sizer,
value=self.potencia,
callback=self.set_potencia,
label="Potencia",
converter=forms.float_converter(),
proportion=0,
)
self._potencia_slider = forms.slider(
parent=self.GetWin(),
sizer=_potencia_sizer,

```

```

        value=self.potencia,
        callback=self.set_potencia,
        minimum=-10,
        maximum=5,
        num_steps=15,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_potencia_sizer)
    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(corr, 0)
    self.osmosdr_sink_0.set_gain(7, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)
    self.osmosdr_sink_0.set_bb_gain(20, 0)
    self.osmosdr_sink_0.set_antenna("", 0)
    self.osmosdr_sink_0.set_bandwidth(0, 0)

    self.set_potencia(potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

    self.low_pass_filter_0 = filter.fir_filter_fff(1, firdes.low_pass(
        5, 48000, 22000, 10000, firdes.WIN_HAMMING, 6.76))
    self.blocks_null_source_0 = blocks.null_source(gr.sizeof_float * 1)
    self.blocks_float_to_complex_0 = blocks.float_to_complex(1)
    self.audio_source_0 = audio.source(48000, "", True)

    #####
    # Connections
    #####
    self.connect((self.blocks_float_to_complex_0, 0),
    (self.rational_resampler_xxx_0, 0))
    self.connect((self.low_pass_filter_0, 0),
    (self.blocks_float_to_complex_0, 0))
    self.connect((self.audio_source_0, 0),
    (self.low_pass_filter_0, 0))
    self.connect((self.rational_resampler_xxx_0, 0),

```

```

(self.osmosdr_sink_0, 0))
    self.connect((self.blocks_null_source_0, 0),
(self.blocks_float_to_complex_0, 1))

# QT sink close method reimplementation

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    frecuencia = self.carrier_freq
    if self.potencia < -10:
        self.potencia = -10
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = get_ajuste(self)
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
    actualizar_label(self, self.parent, self.potencia)

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

```

```

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

```

Listado de programa AM_DSB_SC_vector.py

```

#!/usr/bin/env python
#####

```

```

# Gnuradio Python Flow Graph
# Title: AM Portadora suprimida con onda Arbitraria
# Generated: Thu May 21 14:49:24 2015
#####
# Importación de módulos
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import wx
import math

# Definición de la clase AM_DSB_SC_vector
class AM_DSB_SC_vector(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self, title="AM Portadora suprimida con onda Arbitraria")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.amp = amp = pot      #amplitud reemplaza al parametro pot en este caso
self.vector = vector = onda
self.samp_rate = samp_rate = 12e6
self.mult = mult = (amp - 30) / 400
self.potencia = -5
self.if_gain = if_gain = 21
self.corr = corr = -8
self.carrier_freq = carrier_freq = freq * 1e6
self.pot = self.calcular_potencia(self.mult)

```

```

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label='Frecuencia',
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
self.osmosdr_sink_0.set_sample_rate(samp_rate)
self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
self.osmosdr_sink_0.set_freq_corr(corr, 0)
self.osmosdr_sink_0.set_gain(7, 0)
self.osmosdr_sink_0.set_if_gain(if_gain, 0)
self.osmosdr_sink_0.set_bb_gain(20, 0)
self.osmosdr_sink_0.set_antenna("", 0)
self.osmosdr_sink_0.set_bandwidth(0, 0)

self.set_potencia(self.potencia)
set_corr(self)
actualizar_label(self, self.parent, self.pot)

```

```

        self.blocks_vector_source_x_0 =
blocks.vector_source_f(vector, True, 1, [])
        self.blocks_null_source_0 = blocks.null_source(gr.sizeof_float * 1)
        self.blocks_multiply_const_vxx_1 = blocks.multiply_const_vff((mult, ))
        self.blocks_multiply_const_vxx_0 =
blocks.multiply_const_vff(((1.0 / max(abs(i) for i in vector)), ))
        self.blocks_float_to_complex_0 = blocks.float_to_complex(1)
_amp_sizer = wx.BoxSizer(wx.VERTICAL)
self._amp_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_amp_sizer,
    value=self.amp,
    callback=self.set_amp,
    label="Amplitud",
    converter=forms.float_converter(),
    proportion=0,
)
self._amp_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_amp_sizer,
    value=self.amp,
    callback=self.set_amp,
    minimum=110,
    maximum=430,
    num_steps=16,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_amp_sizer)

#####
# Connections
#####
self.connect((self.blocks_float_to_complex_0, 0),
(self.osmosdr_sink_0, 0))
    self.connect((self.blocks_vector_source_x_0, 0),
(self.blocks_multiply_const_vxx_0, 0))
        self.connect((self.blocks_null_source_0, 0),
(self.blocks_float_to_complex_0, 1))

```

```

        self.connect((self.blocks_multiply_const_vxx_0, 0),
(self.blocks_multiply_const_vxx_1, 0))
        self.connect((self.blocks_multiply_const_vxx_1, 0),
(self.blocks_float_to_complex_0, 0))

# QT sink close method reimplementation

def get_amp(self):
    return self.amp

def set_amp(self, amp):
    self.amp = amp
    self.set_mult((self.amp - 30) / 400)
    self._amp_slider.set_value(self.amp)
    self._amp_text_box.set_value(self.amp)
    self.pot = self.calcular_potencia(self.mult)
    actualizar_label(self, self.parent, self.pot)

def get_vector(self):
    return self.vector

def set_vector(self, vector):
    self.vector = vector
    self.blocks_multiply_const_vxx_0.set_k(
((1.0 / max(abs(i) for i in self.vector)), ))

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_mult(self):
    return self.mult

def set_mult(self, mult):
    self.mult = mult
    self.blocks_multiply_const_vxx_1.set_k((self.mult, ))

```

```

def get_mod_freq(self):
    return self.mod_freq

def set_mod_freq(self, mod_freq):
    self.mod_freq = mod_freq

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    self.if_gain = get_ajuste(self)
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)

```

```

        elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        else:
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
            self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
            self.set_potencia(self.potencia)
            set_corr(self)
            actualizar_label(self, self.parent, self.pot)

    def calcular_potencia(self, mult):
        self.mult = mult
        vec = self.vector
        max_num = max(abs(i) for i in vec)
        for i in range(len(vec)):
            vec[i] /= max_num
            vec[i] *= self.mult
        rms = self.calcular_rms(vec)
        print rms
        rms *= 0.1257433
        rms_sqr = rms * rms
        potencia = rms_sqr/50
        potencia *= 1000
        potencia = 10 * math.log10(potencia)
        potencia = round(potencia, 2)
        print potencia
        return potencia

    def calcular_rms(self, onda):
        mag_sum = 0
        for i in range(len(onda)):
            valor = onda[i]
            mag = valor * valor
            mag_sum += mag
        mag_avg = float(mag_sum) / len(onda)
        mag_avg = math.sqrt(mag_avg)

```

```
    return mag_avg
```

Listado de programa AM_DSB_SC.py

```
#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: AM Portadora Suprimida
# Generated: Sat May  9 15:49:14 2015
#####

from gnuradio import analog
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import wx

class AM_DSB_SC(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self, title="AM Portadora Suprimida")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.samp_rate = samp_rate = param1 * 20
self.potencia = potencia = float(int(pot))
self.mod_freq = mod_freq = param1
self.if_gain = if_gain = 21
```

```

        self.corr = corr = -8
        self.carrier_freq = carrier_freq = freq * 1e6
        self.onda = onda

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label='Frecuencia',
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)

```

```

)
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    minimum=-10,
    maximum=5,
    num_steps=15,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_potencia_sizer)
self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
self.osmosdr_sink_0.set_sample_rate(samp_rate)
self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
self.osmosdr_sink_0.set_freq_corr(corr, 0)
self.osmosdr_sink_0.set_gain(7, 0)
self.osmosdr_sink_0.set_if_gain(if_gain, 0)
self.osmosdr_sink_0.set_bb_gain(16, 0)
self.osmosdr_sink_0.set_antenna("", 0)
self.osmosdr_sink_0.set_bandwidth(0, 0)

self.set_potencia(potencia)
set_corr(self)
actualizar_label(self, self.parent, self.potencia)

self.blocks_null_source_0 = blocks.null_source(gr.sizeof_float * 1)
self.blocks_float_to_complex_0 = blocks.float_to_complex(1)
self.analog_sig_source_x_0 =
analog.sig_source_f(samp_rate, onda, mod_freq, 1, 0)

#####
# Connections
#####
self.connect((self.blocks_float_to_complex_0, 0),
(self.osmosdr_sink_0, 0))
self.connect((self.analog_sig_source_x_0, 0),
(self.blocks_float_to_complex_0, 0))

```

```

        self.connect((self.blocks_null_source_0, 0),
(self.blocks_float_to_complex_0, 1))

# QT sink close method reimplementation

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.analog_sig_source_x_0.set_sampling_freq(self.samp_rate)
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    frecuencia = self.carrier_freq
    if self.potencia < -10:
        self.potencia = -10
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = self.ajuste_dsb()
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
    actualizar_label(self, self.parent, self.potencia)

def get_mod_freq(self):
    return self.mod_freq

def set_mod_freq(self, mod_freq):
    self.mod_freq = mod_freq
    self.analog_sig_source_x_0.set_frequency(self.mod_freq)

```

```

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)

```

```

        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

    def ajuste_dsb(self):
        if self.onda == analog.GR_CONST_WAVE:
            self.if_gain = get_ajuste(self, 0, 0, 0, 0)
        elif self.onda == analog.GR_SIN_WAVE:
            self.if_gain = get_ajuste(self, 2, 2, 3, 3)
        elif self.onda == analog.GR_SQR_WAVE:
            self.if_gain = get_ajuste(self, 1, 1, 2, 3)
        elif self.onda == analog.GR_TRI_WAVE:
            self.if_gain = get_ajuste(self, 3, 4, 4, 4)
        elif self.onda == analog.GR_SAW_WAVE:
            self.if_gain = get_ajuste(self, 3, 4, 4, 4)
        print self.if_gain
        return self.if_gain

```

Listado de programa ASK_Audio.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: ASK con fuente de Audio
# Generated: Sat May  9 19:49:10 2015
#####

from gnuradio import audio
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio import filter
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from Funciones import *
from optparse import OptionParser
import osmosdr
import time

```

```

import wx

class ASK_Audio(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self, title="ASK con fuente de Audio")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

        #####
        # Variables
        #####
        self.parent = parent
        self.sps = sps = 2
        self.samp_rate = samp_rate = 1e6
        self.potencia = potencia = float(int(pot))
        self.if_gain = if_gain = 20
        self.corr = corr = -8
        self.carrier_freq = carrier_freq = freq * 1e6
        self.ask = ask = digital.constellation_rect(
           ([-1 + 0j, -0.33 + 0j, 0.33 + 0j, 1 + 0j]),
            ([0, 1, 2, 3]), 2, 1, 2, 1, 1).base()

        #####
        # Blocks
        #####
        _carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
        self._carrier_freq_text_box = forms.text_box(
            parent=self.GetWin(),
            sizer=_carrier_freq_sizer,
            value=self.carrier_freq,
            callback=self.set_carrier_freq,
            label="Frecuencia (Hz)",
            converter=forms.float_converter(),
            proportion=0,
        )
        self._carrier_freq_slider = forms.slider(
            parent=self.GetWin(),
            sizer=_carrier_freq_sizer,
            value=self.carrier_freq,

```

```

        callback=self.set_carrier_freq,
        minimum=50e6,
        maximum=3e9,
        num_steps=295,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia (dBm)",
    converter=forms.float_converter(),
    proportion=0,
)
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    minimum=-10,
    maximum=5,
    num_steps=15,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_potencia_sizer)
self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
self.osmosdr_sink_0.set_sample_rate(samp_rate)
self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
self.osmosdr_sink_0.set_freq_corr(corr, 0)
self.osmosdr_sink_0.set_gain(10, 0)
self.osmosdr_sink_0.set_if_gain(if_gain, 0)
self.osmosdr_sink_0.set_bb_gain(20, 0)
self.osmosdr_sink_0.set_antenna("", 0)

```

```

    self.osmosdr_sink_0.set_bandwidth(0, 0)

    self.set_potencia(potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

    self.digital_constellation_modulator_0_0 = digital.generic_mod(
        constellation=ask,
        differential=False,
        samples_per_symbol=sps,
        pre_diff_code=True,
        excess_bw=param1,
        verbose=False,
        log=False,
    )
    self.blks2_packet_encoder_0 = grc_blks2.packet_mod_f(grc_blks2.packet_encoder(
        samples_per_symbol=sps,
        bits_per_symbol=2,
        preamble="",
        access_code="",
        pad_for_usrp=True,
    ),
        payload_length=0,
    )
    self.audio_source_0 = audio.source(48000, "", True)
    self.rational_resampler_xxx_0 = filter.rational_resampler_fff(
        interpolation=48000,
        decimation=int(samp_rate),
        taps=None,
        fractional_bw=None,
    )

#####
# Connections
#####
self.connect((self.digital_constellation_modulator_0_0, 0), (self.osmosdr_sink_0, 0))
self.connect((self.blks2_packet_encoder_0, 0), (self.digital_constellation_modulator_0_0, 0))
self.connect((self.audio_source_0, 0), (self.rational_resampler_xxx_0, 0))
self.connect((self.rational_resampler_xxx_0, 0), (self.blks2_packet_encoder_0, 0))

```

```

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    frecuencia = self.carrier_freq
    if self.potencia < -10:
        self.potencia = -10
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
        elif frecuencia >= 2750e6:
            if self.potencia >= -5:
                self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = get_ajuste(self, 1, 1, 1, 0)
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

```

```

        self._if_gain_slider.set_value(self.if_gain)
        self._if_gain_text_box.set_value(self.if_gain)

    def get_corr(self):
        return self.corr

    def set_corr(self, corr):
        self.corr = corr
        self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

    def get_carrier_freq(self):
        return self.carrier_freq

    def set_carrier_freq(self, carrier_freq):
        self.carrier_freq = carrier_freq
        if self.carrier_freq < 50e6:
            self.carrier_freq = 50e6
        elif self.carrier_freq > 3e9:
            self.carrier_freq = 3e9
        if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        else:
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
            self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
            self.set_potencia(self.potencia)
            set_corr(self)

    def get_ask(self):
        return self.ask

```

```
def set_ask(self, ask):
    self.ask = ask
```

Listado de programa ASK_Random.py

```
#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: ASK Random
# Generated: Wed May  6 15:59:47 2015
#####

from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import numpy
import osmosdr
import time
import wx

class ASK_Random(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self, title="ASK Random")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
```

```

        self.bitrate = param3
        self.sps = sps = 2
        self.samp_rate = samp_rate = (self.bitrate / 2) * self.sps * 1000
        self.potencia = potencia = float(int(pot))
        self.if_gain = if_gain = 20
        self.carrier_freq = carrier_freq = freq * 1e6
        self.bt = bt = param1
        self.corr = corr = -8
        self.ask = ask = digital.constellation_rect(([ -1 + 0j, -0.33 + 0j, 0.33 + 0j, 1 + 0j]),
([0, 1, 2, 3]), 2, 1, 2, 1, 1).base()

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label="Frecuencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),

```

```

        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        label="Potencia",
        converter=forms.float_converter(),
        proportion=0,
    )
    self._potencia_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        minimum=-10,
        maximum=5,
        num_steps=15,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_potencia_sizer)
    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(-8, 0)
    self.osmosdr_sink_0.set_gain(7, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)
    self.osmosdr_sink_0.set_bb_gain(20, 0)
    self.osmosdr_sink_0.set_antenna("", 0)
    self.osmosdr_sink_0.set_bandwidth(0, 0)

    self.set_potencia(potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

    self.digital_constellation_modulator_0_0 = digital.generic_mod(
        constellation=ask,
        differential=False,
        samples_per_symbol=sps,
        pre_diff_code=True,
        excess_bw=bt,

```

```

        verbose=True,
        log=False,
    )
    self.blocks_throttle_0 = blocks.throttle(gr.sizeof_char * 1, samp_rate)
    self.blks2_packet_encoder_0 = grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
        samples_per_symbol=sps,
        bits_per_symbol=2,
        preamble="",
        access_code="",
        pad_for_usrp=True,
    ),
    payload_length=0,
)
self.analog_random_source_x_0_0 =
blocks.vector_source_b(map(int, numpy.random.randint(0, 4, 1000)), True)

#####
# Connections
#####
self.connect((self.digital_constellation_modulator_0_0, 0),
(self.osmosdr_sink_0, 0))
    self.connect((self.blocks_throttle_0, 0), (self.blks2_packet_encoder_0, 0))
    self.connect((self.analog_random_source_x_0_0, 0),
self.blocks_throttle_0, 0))
    self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_constellation_modulator_0_0, 0))

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):

```

```

        self.samp_rate = samp_rate
        self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

    def get_rrc_taps(self):
        return self_rrc_taps

    def set_rrc_taps(self, rrc_taps):
        self_rrc_taps = rrc_taps

    def get_potencia(self):
        return self.potencia

    def set_potencia(self, potencia):
        self.potencia = float(int(potencia))
        frecuencia = self.carrier_freq
        if self.potencia < -10:
            self.potencia = -10
        if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
            if self.potencia >= 0:
                self.potencia = 0
        elif frecuencia >= 2750e6:
            if self.potencia >= -5:
                self.potencia = -5
        self._potencia_slider.set_value(self.potencia)
        self._potencia_text_box.set_value(self.potencia)
        self.if_gain = get_ajuste(self, 1, 1, 1, 0)
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
        actualizar_label(self, self.parent, self.potencia)

    def get_if_gain(self):
        return self.if_gain

    def set_if_gain(self, if_gain):
        self.if_gain = if_gain
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

    def get_carrier_freq(self):
        return self.carrier_freq

```

```
def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

def get_bt(self):
    return self.bt

def set_bt(self, bt):
    self.bt = bt

def get_ask(self):
    return self.ask

def set_ask(self, ask):
    self.ask = ask
```

Listado de programa ASK_UDP.py

```
#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: ASK con fuente de Streaming
# Generated: Mon May 11 18:43:16 2015
#####

from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import wx

class ASK_UDP(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3,parent):
        grc_wxgui.top_block_gui.__init__(self, title="ASK con fuente de Streaming")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.sps = sps = 2
self.bitrate = param3
self.samp_rate = samp_rate = (self.bitrate / 2) * self.sps * 1000
self.potencia = potencia = float(int(pot))
self.if_gain = if_gain = 20
```

```

        self.corr = corr = -8
        self.carrier_freq = carrier_freq = freq * 1e6
        self.ask = ask = digital.constellation_rect(
([-1 + 0j, -0.33 + 0j, 0.33 + 0j, 1 + 0j]),
([0, 1, 2, 3]), 2, 1, 2, 1, 1).base()

#####
# Blocks
#####

_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label="Frecuencia (Hz)",
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia (dBm)",
    converter=forms.float_converter(),
)

```

```

        proportion=0,
    )
    self._potencia_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        minimum=-10,
        maximum=5,
        num_steps=15,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_potencia_sizer)
    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(corr, 0)
    self.osmosdr_sink_0.set_gain(10, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)
    self.osmosdr_sink_0.set_bb_gain(20, 0)
    self.osmosdr_sink_0.set_antenna("", 0)
    self.osmosdr_sink_0.set_bandwidth(0, 0)

    self.set_potencia(potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

    self.digital_constellation_modulator_0_0 = digital.generic_mod(
        constellation=ask,
        differential=False,
        samples_per_symbol=sps,
        pre_diff_code=True,
        excess_bw=param1,
        verbose=False,
        log=False,
    )
    self.blocks_udp_source_0 = blocks udp_source(gr.sizeof_char * 1, onda, param2, 1472, True)
    self.blocks_throttle_0 = blocks.throttle(gr.sizeof_char*1, samp_rate)

```

```

        self.blks2_packet_encoder_0 = grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
            samples_per_symbol=sps,
            bits_per_symbol=2,
            preamble="",
            access_code="",
            pad_for_usrp=True,
        ),
        payload_length=0,
    )

#####
# Connections
#####
self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_constellation_modulator_0_0, 0))
    self.connect((self.digital_constellation_modulator_0_0, 0),
(self.osmosdr_sink_0, 0))
        self.connect((self.blocks_udp_source_0, 0),
(self.blocks_throttle_0, 0))
            self.connect((self.blocks_throttle_0, 0),
(self.blks2_packet_encoder_0, 0))

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

```

```

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    frecuencia = self.carrier_freq
    if self.potencia < -10:
        self.potencia = -10
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = get_ajuste(self, 1, 1, 1, 0)
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
    actualizar_label(self, self.parent, self.potencia)

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6

```

```

        elif self.carrier_freq > 3e9:
            self.carrier_freq = 3e9
        if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        else:
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
            self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
            self.set_potencia(self.potencia)
            set_corr(self)
            actualizar_label(self, self.parent, self.potencia)

    def get_ask(self):
        return self.ask

    def set_ask(self, ask):
        self.ask = ask

```

Listado de programa ASK_Video.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: ASK con fuente de Video
# Generated: Mon May 11 18:43:19 2015
#####

from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation

```

```

from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import wx

class ASK_video(grc_wxgui.top_block_gui):
    def __init__(self,onda,frec,pot,param1,param2,param3,parent):
        grc_wxgui.top_block_gui.__init__(self, title="ASK con fuente de Video")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

        #####
        # Variables
        #####
        self.parent = parent
        self.sps = sps = 2
        self.bitrate = param3
        self.samp_rate = samp_rate = (self.bitrate / 2) * self.sps * 1000
        self.potencia = potencia = float(int(pot))
        self.if_gain = if_gain = 20
        self.corr = corr = -8
        self.carrier_freq = carrier_freq = frec * 1e6
        self.ask = ask = digital.constellation_rect(([ -1 + 0j, -0.33 + 0j,
0.33 + 0j, 1 + 0j]), ([0, 1, 2, 3]), 2, 1, 2, 1, 1).base()

        #####
        # Blocks
        #####
        _carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
        self._carrier_freq_text_box = forms.text_box(
            parent=self.GetWin(),
            sizer=_carrier_freq_sizer,

```

```

        value=self.carrier_freq,
        callback=self.set_carrier_freq,
        label="Frecuencia",
        converter=forms.float_converter(),
        proportion=0,
    )
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    minimum=-10,
    maximum=5,
    num_steps=15,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)

```

```

    )

    self.Add(_potencia_sizer)

    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(corr, 0)
    self.osmosdr_sink_0.set_gain(10, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)
    self.osmosdr_sink_0.set_bb_gain(20, 0)
    self.osmosdr_sink_0.set_antenna("", 0)
    self.osmosdr_sink_0.set_bandwidth(0, 0)

    self.set_potencia(potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

    self.digital_constellation_modulator_0_0 = digital.generic_mod(
        constellation=ask,
        differential=False,
        samples_per_symbol=sps,
        pre_diff_code=True,
        excess_bw=param1,
        verbose=False,
        log=False,
    )

    self.blocks_file_source_0 = blocks.file_source(gr.sizeof_char * 1, onda, True)
    self.blocks_throttle_0 = blocks.throttle(gr.sizeof_char*1, samp_rate)
    self.blks2_packet_encoder_0 = grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
        samples_per_symbol=sps,
        bits_per_symbol=2,
        preamble="",
        access_code="",
        pad_for_usrp=True,
    ),
    payload_length=0,
)

#####
# Connections
#####

```

```

        self.connect((self.digital_constellation_modulator_0_0, 0),
(self.osmosdr_sink_0, 0))
        self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_constellation_modulator_0_0))
        self.connect((self.blocks_throttle_0, 0), (self.blks2_packet_encoder_0, 0))
        self.connect((self.blocks_file_source_0, 0), (self.blocks_throttle_0, 0))

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    frecuencia = self.carrier_freq
    if self.potencia < -10:
        self.potencia = -10
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = get_ajuste(self, 1, 1, 1, 0)

```

```

        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
        actualizar_label(self, self.parent, self.potencia)

    def get_if_gain(self):
        return self.if_gain

    def set_if_gain(self, if_gain):
        self.if_gain = if_gain
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

    def get_corr(self):
        return self.corr

    def set_corr(self, corr):
        self.corr = corr
        self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

    def get_carrier_freq(self):
        return self.carrier_freq

    def set_carrier_freq(self, carrier_freq):
        self.carrier_freq = carrier_freq
        if self.carrier_freq < 50e6:
            self.carrier_freq = 50e6
        elif self.carrier_freq > 3e9:
            self.carrier_freq = 3e9
        if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        else:
            self._carrier_freq_slider.set_value(self.carrier_freq)

```

```

        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

    def get_ask(self):
        return self.ask

    def set_ask(self, ask):
        self.ask = ask

```

Listado de programa Funciones.py

```

#!/usr/bin/env python

from lista import *

def set_corr(self):
    freq = self.carrier_freq
    freq /= 1000000
    freq = (int(freq) / 10) * 10

    frecuencia_actual = float(self.carrier_freq) / 1000000

    corr_freq = self.corr
    corr_freq = ajusteFreq[freq]
    if freq != 3000:
        m = (ajusteFreq[freq + 10] - ajusteFreq[freq]) / 10
    else:
        m = 0
    corr_freq += m * (frecuencia_actual - freq)
    self.osmosdr_sink_0.set_freq_corr(corr_freq, 0)
    return

def get_ajuste(self, aj1=0, aj2=0, aj3=0, aj4=0):
    freq = self.carrier_freq
    freq /= 1000000
    freq = (int(freq) / 10) * 10

```

```

gananciaN = self.if_gain
if self.potencia >= 3:
    gananciaN = ajuste5[freq]
    dif = 5 - self.potencia
    gananciaN += (aj1 - dif)
elif self.potencia >= (-2):
    gananciaN = ajuste0[freq]
    dif = 0 + self.potencia
    gananciaN += dif + aj2
elif self.potencia >= (-7):
    gananciaN = ajuste_5[freq]
    dif = self.potencia + 5
    gananciaN += dif + aj3
else:
    gananciaN = ajuste_10[freq]
    dif = self.potencia + 10
    gananciaN += dif + aj4
if gananciaN > 47:
    gananciaN = 47
return gananciaN

def get_ajuste_dvbs2(self):
    freq = self.carrier_freq
    freq /= 1000000
    freq = (int(freq) / 10)*10
    gananciaN = self.if_gain
    if self.potencia >= -12:
        gananciaN = ajuste_dvbs2_10[freq]
        dif = self.potencia + 10
        gananciaN += dif
    else:
        gananciaN = ajuste_dvbs2_15[freq]
        dif = self.potencia + 15
        gananciaN += dif
    if gananciaN > 47:
        gananciaN = 47
    return gananciaN

def get_ajuste_ntsc(self):
    freq = self.carrier_freq

```

```

freq /= 1000000.0
freq_final = calcular_frecuencia_ntsc(freq)
gananciaN = self.if_gain
if self.potencia >= -2:
    gananciaN = ajuste_ntsc0[freq_final]
    dif = self.potencia
    gananciaN += dif
elif self.potencia >= (-6):
    gananciaN = ajuste_ntsc_5[freq_final]
    dif = self.potencia + 5
    gananciaN += dif
else:
    gananciaN = ajuste_ntsc_10[freq_final]
    dif = self.potencia + 10
    gananciaN += dif
if gananciaN > 47:
    gananciaN = 47
return gananciaN

def calcular_frecuencia_ntsc(freq):
    freq_inicial = 55.25
    i = freq_inicial
    while (freq - i) > 6:
        i += 6
    if abs(freq - i) < abs(freq - (i + 6)):
        freq_final = i
    else:
        freq_final = i + 6
    print freq_final
    return freq_final

def ocultar(self, m1, m2, m3, m4, m5, m6, m7, m8):
    self.sizer.Show(self.box1, m1)
    self.sizer.Show(self.box2, m2)
    self.sizer.Show(self.box3, m3)
    self.sizer.Show(self.box4, m4)
    self.sizer.Show(self.box41, m4)
    self.sizer.Show(self.bs2, False)
    self.sizer.Show(self.box5, m5)
    self.sizer.Show(self.box6, m6)

```

```

        self.sizer.Show(self.box7, m7)
        self.sizer.Show(self.box8, m8)
        self.sizer.Show(self.box9, m8)
        self.mainSizer.Layout()

def actualizar_label(self, panel, potencia):
    freq = str(self.carrier_freq / 1000000)
    pot = str(potencia)
    mensaje = "%s MHz : %s dBm" % (freq,pot)
    panel.cuadro.SetLabel(mensaje)

```

Listado de programa dvbs2_folder/top_block.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: Top Block
# Generated: Wed May 13 16:41:25 2015
#####

from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import filter
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import dvbs2
import osmosdr
import wx

class top_block_dvbs2(grc_wxgui.top_block_gui):
    def __init__(self,onda,freq,pot,param1,param2,param3,parent):
        grc_wxgui.top_block_gui.__init__(self, title="Top Block")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

```

```

#####
# Variables
#####
self.parent = parent
self.taps = taps = 50
self.samp_rate = samp_rate = 10e6
self.rolloff = rolloff = 0.2
self.potencia = potencia = float(int(pot))
self.if_gain = if_gain = 20
self.corr = corr = -8
self.carrier_freq = carrier_freq = freq * 1e6
self.mod = mod = self.get_mod(param1)
self.cr = cr = self.get_cr(param2)

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label='Frecuencia',
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=1e9,
    maximum=2e9,
    num_steps=100,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)

```

```

_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    minimum=-15,
    maximum=-10,
    num_steps=5,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_potencia_sizer)
self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
self.osmosdr_sink_0.set_sample_rate(samp_rate)
self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
self.osmosdr_sink_0.set_freq_corr(corr, 0)
self.osmosdr_sink_0.set_gain(10, 0)
self.osmosdr_sink_0.set_if_gain(if_gain, 0)
self.osmosdr_sink_0.set_bb_gain(20, 0)
self.osmosdr_sink_0.set_antenna("", 0)
self.osmosdr_sink_0.set_bandwidth(0, 0)

self.set_potencia(potencia)
set_corr(self)
actualizar_label(self, self.parent, self.potencia)

self.fft_filter_xxx_0 = filter.fft_filter_ccc(1, (
    firdes.root_raised_cosine(1, samp_rate,
    samp_rate / 2, rolloff, taps)), 1)

```

```

        self.fft_filter_xxx_0.declare_sample_delay(0)
        self.dvbs2_physical_cc_0 = dvbs2.physical_cc(mod, cr, dvbs2.PILOTS_OFF,
                                                       dvbs2.FECFRAME_NORMAL, 0)
        self.dvbs2_modulator_bc_0 = dvbs2.modulator_bc(mod, dvbs2.C_OTHER,
dvbs2.FECFRAME_NORMAL)
        self.dvbs2_ldpc_bb_0 = dvbs2.ldpc_bb(cr, dvbs2.FECFRAME_NORMAL,
dvbs2.MOD_OTHER)
        self.dvbs2_interleaver_bb_0 = dvbs2.interleaver_bb(mod, dvbs2.C_OTHER,
dvbs2.FECFRAME_NORMAL)
        self.dvbs2_bch_bb_0 = dvbs2.bch_bb(cr, dvbs2.FECFRAME_NORMAL)
        self.dvbs2_bb scrambler_bb_0 = dvbs2.bb scrambler_bb(cr,
dvbs2.FECFRAME_NORMAL)
        self.dvbs2_bbheader_bb_0 = dvbs2.bbheader_bb(cr, dvbs2.R0_0_35,
dvbs2.FECFRAME_NORMAL)
        self.blocks_file_source_0 =
blocks.file_source(gr.sizeof_char * 1, onda, True)

#####
# Connections
#####
        self.connect((self.blocks_file_source_0, 0),
(self.dvbs2_bbheader_bb_0, 0))
        self.connect((self.dvbs2_physical_cc_0, 0),
(self.fft_filter_xxx_0, 0))
        self.connect((self.dvbs2_modulator_bc_0, 0),
(self.dvbs2_physical_cc_0, 0))
        self.connect((self.dvbs2_interleaver_bb_0, 0),
(self.dvbs2_modulator_bc_0, 0))
        self.connect((self.dvbs2_ldpc_bb_0, 0),
(self.dvbs2_interleaver_bb_0, 0))
        self.connect((self.dvbs2_bch_bb_0, 0),
(self.dvbs2_ldpc_bb_0, 0))
        self.connect((self.dvbs2_bb scrambler_bb_0, 0),
(self.dvbs2_bch_bb_0, 0))
        self.connect((self.dvbs2_bbheader_bb_0, 0),
(self.dvbs2_bb scrambler_bb_0, 0))
        self.connect((self.fft_filter_xxx_0, 0), (self.osmosdr_sink_0, 0))

def get_taps(self):
    return self.taps

```

```

def set_taps(self, taps):
    self.taps = taps
    self.fft_filter_xxx_0.set_taps(
        (firades.root_raised_cosine(1, self.samp_rate,
        self.samp_rate / 2, self.rolloff, self.taps)))

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.fft_filter_xxx_0.set_taps(
        (firades.root_raised_cosine(1, self.samp_rate,
        self.samp_rate / 2, self.rolloff, self.taps)))
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_rolloff(self):
    return self.rolloff

def set_rolloff(self, rolloff):
    self.rolloff = rolloff
    self.fft_filter_xxx_0.set_taps(
        (firades.root_raised_cosine(1, self.samp_rate,
        self.samp_rate / 2, self.rolloff, self.taps)))

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = potencia
    if self.potencia > -10:
        self.potencia = -10

```

```

        elif self.potencia < -15:
            self.potencia = -15
        self._potencia_slider.set_value(self.potencia)
        self._potencia_text_box.set_value(self.potencia)
        self.if_gain = get_ajuste_dvbs2(self)
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
        actualizar_label(self, self.parent, self.potencia)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    if self.carrier_freq < 1000e6:
        self.carrier_freq = 1000e6
    elif self.carrier_freq > 2e9:
        self.carrier_freq = 2e9
    self.carrier_freq = carrier_freq
    self._carrier_freq_slider.set_value(self.carrier_freq)
    self._carrier_freq_text_box.set_value(self.carrier_freq)
    self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
    self.set_potencia(self.potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

def get_mod(self,mod_str):
    mod = dvbs2.MOD_QPSK
    if mod_str == 'QPSK':
        mod = dvbs2.MOD_QPSK
    elif mod_str == '8PSK':
        mod = dvbs2.MOD_8PSK
    elif mod_str == '8APSK':
        mod = dvbs2.MOD_8APSK
    elif mod_str == '16APSK':
        mod = dvbs2.MOD_16APSK
    elif mod_str == '32APSK':
        mod = dvbs2.MOD_32APSK
    return mod

def get_cr(self,cr_str):
    cr = dvbs2.C1_2

```

```

if cr_str == '1/2':
    cr = dvbs2.C1_2
elif cr_str == '1/3':
    cr = dvbs2.C1_3
elif cr_str == '1/4':
    cr = dvbs2.C1_4
elif cr_str == '2/3':
    cr = dvbs2.C2_3
elif cr_str == '2/5':
    cr = dvbs2.C2_5
elif cr_str == '3/4':
    cr = dvbs2.C3_4
elif cr_str == '3/5':
    cr = dvbs2.C3_5
elif cr_str == '4/5':
    cr = dvbs2.C4_5
elif cr_str == '5/6':
    cr = dvbs2.C5_6
return cr

```

Listado de programa dvbt_folder/top_block.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: Estandar de television DVB-T
# Generated: Wed May 13 16:42:23 2015
#####

from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import fft
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.fft import window
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser

```

```

from Funciones import *
import dvbt
import osmosdr
import wx

class top_block_dvbt(grc_wxgui.top_block_gui):
    def __init__(self,onda,frec,pot,param1,param2,param3,parent):
        grc_wxgui.top_block_gui.__init__(self, title="Estandar de television DVB-T")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

        #####
        # Variables
        #####
        self.parent = parent
        self.samp_rate = samp_rate = 9.142857143e6
        self.potencia = potencia = float(int(pot))
        self.if_gain = if_gain = 20
        self.corr = corr = -8
        self.carrier_freq = carrier_freq = frec * 1e6
        self.mod = mod = self.get_mod(param1)
        self.cr = cr = self.get_cr(param2)

        #####
        # Blocks
        #####
        _carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
        self._carrier_freq_text_box = forms.text_box(
            parent=self.GetWin(),
            sizer=_carrier_freq_sizer,
            value=self.carrier_freq,
            callback=self.set_carrier_freq,
            label="Frecuencia",
            converter=forms.float_converter(),
            proportion=0,
        )
        self._carrier_freq_slider = forms.slider(
            parent=self.GetWin(),
            sizer=_carrier_freq_sizer,

```

```

        value=self.carrier_freq,
        callback=self.set_carrier_freq,
        minimum=50e6,
        maximum=3e9,
        num_steps=295,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    minimum=-10,
    maximum=0,
    num_steps=10,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_potencia_sizer)
self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
self.osmosdr_sink_0.set_sample_rate(samp_rate)
self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
self.osmosdr_sink_0.set_freq_corr(corr, 0)
self.osmosdr_sink_0.set_gain(7, 0)
self.osmosdr_sink_0.set_if_gain(if_gain, 0)
self.osmosdr_sink_0.set_bb_gain(20, 0)

```

```

        self.osmosdr_sink_0.set_antenna("", 0)
        self.osmosdr_sink_0.set_bandwidth(0, 0)

        self.set_potencia(potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

        self.fft_vxx_0 = fft.fft_vcc(2048, False,
(window.rectangular(2048)), True, 10)
        self.dvbt_symbol_inner_interleaver_0 =
dvbt.symbol_inner_interleaver(1512, dvbt.T2k, 1)
        self.dvbt_reference_signals_0 =
dvbt.reference_signals(gr.sizeof_gr_complex, 1512, 2048, mod, dvbt.NH,
cr, cr, dvbt.G1_32, dvbt.T2k, 0, 0)
        self.dvbt_reed_solomon_enc_0 =
dvbt.reed_solomon_enc(2, 8, 0x11d, 255, 239, 8, 51, 8)
        self.dvbt_inner_coder_0 =
dvbt.inner_coder(1, 1512, dvbt.QAM16, dvbt.NH, cr)
        self.dvbt_energy_dispersal_0 = dvbt.energy_dispersal(1)
        self.dvbt_dvbt_map_0 = dvbt.dvbt_map(1512, mod, dvbt.NH, dvbt.T2k, 1)
        self.dvbt_convolutional_interleaver_0 =
dvbt.convolutional_interleaver(136, 12, 17)
        self.dvbt_bit_inner_interleaver_0 =
dvbt.bit_inner_interleaver(1512, mod, dvbt.NH, dvbt.T2k)
        self.digital_ofdm_cyclic_prefixer_0 =
digital.ofdm_cyclic_prefixer(2048, 2048 + 64, 0, "")
        self.blocks_file_source_0 =
blocks.file_source(gr.sizeof_char * 1, onda, True)

#####
# Connections
#####
        self.connect((self.blocks_file_source_0, 0),
(self.dvbt_energy_dispersal_0, 0))
        self.connect((self.dvbt_reed_solomon_enc_0, 0),
(self.dvbt_convolutional_interleaver_0, 0))
        self.connect((self.dvbt_energy_dispersal_0, 0),
(self.dvbt_reed_solomon_enc_0, 0))
        self.connect((self.dvbt_convolutional_interleaver_0, 0),
(self.dvbt_inner_coder_0, 0))

```

```

        self.connect((self.dvbt_inner_coder_0, 0),
(self.dvbt_bit_inner_interleaver_0, 0))
        self.connect((self.dvbt_bit_inner_interleaver_0, 0),
(self.dvbt_symbol_inner_interleaver_0, 0))
        self.connect((self.dvbt_symbol_inner_interleaver_0, 0),
(self.dvbt_dvbt_map_0, 0))
        self.connect((self.dvbt_dvbt_map_0, 0),
(self.dvbt_reference_signals_0, 0))
        self.connect((self.dvbt_reference_signals_0, 0),
(self.fft_vxx_0, 0))
        self.connect((self.fft_vxx_0, 0),
(self.digital_ofdm_cyclic_prefixer_0, 0))
        self.connect((self.digital_ofdm_cyclic_prefixer_0, 0),
(self.osmosdr_sink_0, 0))

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = potencia
    frecuencia = self.carrier_freq
    if (frecuencia >= 50e6) and (frecuencia < 2750e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = get_ajuste(self, -2, -2, -2, -2)
    print self.if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

```

```

    actualizar_label(self, self.parent, self.potencia)

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)

```

```

        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

    def get_mod(self,mod_str):
        mod = dvbt.QPSK
        if mod_str == 'QPSK':
            mod = dvbt.QPSK
        elif mod_str == '16QAM':
            mod = dvbt.QAM16
        elif mod_str == '64QAM':
            mod = dvbt.QAM64
        return mod

    def get_cr(self,cr_str):
        cr = dvbt.C1_2
        if cr_str == '1/2':
            cr = dvbt.C1_2
        elif cr_str == '2/3':
            cr = dvbt.C2_3
        elif cr_str == '3/4':
            cr = dvbt.C3_4
        elif cr_str == '5/6':
            cr = dvbt.C5_6
        elif cr_str == '7/8':
            cr = dvbt.C7_8
        return cr

```

Listado de programa GFSK_Audio.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: GFSK con fuente de Audio
# Generated: Sat May  9 19:37:21 2015
#####

from gnuradio import audio
from gnuradio import digital

```

```

from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import wx

class GFSK_audio(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self,
                                         title="GFSK con fuente de Audio")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

        #####
        # Variables
        #####
        self.parent = parent
        self.sps = sps = 2
        self.samp_rate = samp_rate = 1e6
        self.potencia = potencia = float(int(pot))
        self.if_gain = if_gain = 20
        self.corr = corr = -8
        self.carrier_freq = carrier_freq = freq * 1e6

        #####
        # Blocks
        #####
        _carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
        self._carrier_freq_text_box = forms.text_box(
            parent=self.GetWin(),
            sizer=_carrier_freq_sizer,
            value=self.carrier_freq,

```

```
        callback=self.set_carrier_freq,
        label="Frecuencia",
        converter=forms.float_converter(),
        proportion=0,
    )
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    minimum=-10,
    maximum=5,
    num_steps=15,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
```

```

    self.Add(_potencia_sizer)
    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(corr, 0)
    self.osmosdr_sink_0.set_gain(10, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)
    self.osmosdr_sink_0.set_bb_gain(20, 0)
    self.osmosdr_sink_0.set_antenna("", 0)
    self.osmosdr_sink_0.set_bandwidth(0, 0)

    self.set_potencia(potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

    self.digital_gfsk_mod_0 = digital.gfsk_mod(
        samples_per_symbol=sps,
        sensitivity=1.0,
        bt=param1,
        verbose=False,
        log=False,
    )
    self.blks2_packet_encoder_0 =
    grc_blks2.packet_mod_f(grc_blks2.packet_encoder(
        samples_per_symbol=sps,
        bits_per_symbol=1,
        preamble="",
        access_code="",
        pad_for_usrp=True,
    ),
        payload_length=0,
    )
    self.audio_source_0 = audio.source(48000, "", True)

    #####
    # Connections
    #####
    self.connect((self.digital_gfsk_mod_0, 0),
    (self.osmosdr_sink_0, 0))
    self.connect((self.blks2_packet_encoder_0, 0),

```

```

(self.digital_gfsk_mod_0, 0))
    self.connect((self.audio_source_0, 0),
(self.blks2_packet_encoder_0, 0))

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    frecuencia = self.carrier_freq
    if self.potencia < -10:
        self.potencia = -10
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = get_ajuste(self)
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
    actualizar_label(self, self.parent, self.potencia)

```

```

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)

```

```
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)
```

Listado de programa GFSK_Random.py

```
#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: GFSK Random Source
# Generated: Wed Apr 29 14:37:59 2015
#####

from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from grc_gnuradio import blks2 as grc_blks2
from optparse import OptionParser
import numpy
import osmosdr
import time
import wx
from Funciones import *
# from Ventana import MyPanel

class GFSK_Random(grc_wxgui.top_block_gui):
    def __init__(self, onda, frec, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self, title="GFSK Random Source")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.sps = sps = 2
```

```

self.bitrate = param3
self.samp_rate = samp_rate = (self.bitrate / 1) * self.sps * 1000
self.potencia = potencia = float(int(pot))
self.if_gain = if_gain = 20
self.carrier_freq = carrier_freq = freq * 1e6
self.bt = bt = param1
self.corr = corr = -8

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label="Frecuencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
)

```

```

        label="Potencia",
        converter=forms.float_converter(),
        proportion=0,
    )
    self._potencia_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        minimum=-10,
        maximum=5,
        num_steps=15,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_potencia_sizer)
    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(-8, 0)
    self.osmosdr_sink_0.set_gain(7, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)
    self.osmosdr_sink_0.set_bb_gain(20, 0)
    self.osmosdr_sink_0.set_antenna("", 0)
    self.osmosdr_sink_0.set_bandwidth(0, 0)

    self.set_potencia(potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

    self.digital_gfsk_mod_0 = digital.gfsk_mod(
        samples_per_symbol=sps,
        sensitivity=1.0,
        bt=bt,
        verbose=False,
        log=False,
    )
    self.blocks_throttle_0 =
blocks.throttle(gr.sizeof_char * 1, samp_rate)

```

```

        self.blks2_packet_encoder_0 =
grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
            samples_per_symbol=sps,
            bits_per_symbol=1,
            preamble="",
            access_code="",
            pad_for_usrp=True,
),
            payload_length=0,
)
        self.analog_random_source_x_0 =
blocks.vector_source_b(map(int, numpy.random.randint(0, 256, 1000)), True)

#####
# Connections
#####
self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_gfsk_mod_0, 0))
    self.connect((self.analog_random_source_x_0, 0),
(self.blocks_throttle_0, 0))
        self.connect((self.digital_gfsk_mod_0, 0),
(self.osmosdr_sink_0, 0))
            self.connect((self.blocks_throttle_0, 0),
(self.blks2_packet_encoder_0, 0))

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

```

```

def get_rrc_taps(self):
    return self_rrc_taps

def set_rrc_taps(self, rrc_taps):
    self_rrc_taps = rrc_taps

def get_bt(self):
    return self_bt

def set_bt(self, bt):
    self_bt = bt

def get_potencia(self):
    return self_potencia

def set_potencia(self, potencia):
    self_potencia = float(int(potencia))
    frecuencia = self_carrier_freq
    if self_potencia < -10:
        self_potencia = -10
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self_potencia >= 0:
            self_potencia = 0
    elif frecuencia >= 2750e6:
        if self_potencia >= -5:
            self_potencia = -5
    self_potencia_slider.set_value(self_potencia)
    self_potencia_text_box.set_value(self_potencia)
    self_if_gain = get_ajuste(self)
    self.osmosdr_sink_0.set_if_gain(self_if_gain, 0)
    actualizar_label(self, self.parent, self_potencia)

def get_if_gain(self):
    return self_if_gain

def set_if_gain(self, if_gain):
    self_if_gain = get_ajuste(self)
    self.osmosdr_sink_0.set_if_gain(self_if_gain, 0)

```

```

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

```

Listado de programa GFSK_UDP.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: GFSK con fuente de Streaming
# Generated: Tue May 12 10:02:39 2015
#####

from gnuradio import blocks
from gnuradio import digital

```

```

from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import wx

class GFSK_UDP(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self,
                                         title="GFSK con fuente de Streaming")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

        #####
        # Variables
        #####
        self.parent = parent
        self.sps = sps = 2
        self.bitrate = param3
        self.samp_rate = samp_rate = (self.bitrate / 1) * self.sps * 1000
        self.potencia = potencia = float(int(pot))
        self.if_gain = if_gain = 20
        self.corr = corr = -8
        self.carrier_freq = carrier_freq = freq * 1e6

        #####
        # Blocks
        #####
        _carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
        self._carrier_freq_text_box = forms.text_box(
            parent=self.GetWin(),
            sizer=_carrier_freq_sizer,

```

```

        value=self.carrier_freq,
        callback=self.set_carrier_freq,
        label="Frecuencia",
        converter=forms.float_converter(),
        proportion=0,
    )
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    minimum=-10,
    maximum=5,
    num_steps=15,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)

```

```

    )

    self.Add(_potencia_sizer)

    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(corr, 0)
    self.osmosdr_sink_0.set_gain(10, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)
    self.osmosdr_sink_0.set_bb_gain(20, 0)
    self.osmosdr_sink_0.set_antenna("", 0)
    self.osmosdr_sink_0.set_bandwidth(0, 0)

    self.set_potencia(potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

    self.digital_gfsk_mod_0 = digital.gfsk_mod(
        samples_per_symbol=sps,
        sensitivity=1.0,
        bt=param1,
        verbose=False,
        log=False,
    )
    self.blocks_udp_source_0 =
blocks_udp_source(gr.sizeof_char * 1, onda, param2, 1472, True)
    self.blocks_throttle_0 =
blocks.throttle(gr.sizeof_char * 1, samp_rate)
    self.blks2_packet_encoder_0 =
grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
        samples_per_symbol=sps,
        bits_per_symbol=1,
        preamble="",
        access_code="",
        pad_for_usrp=True,
),
        payload_length=0,
    )

#####
# Connections

```

```

#####
    self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_gfsk_mod_0, 0))
    self.connect((self.digital_gfsk_mod_0, 0),
(self.osmosdr_sink_0, 0))
    self.connect((self.blocks_udp_source_0, 0),
(self.blocks_throttle_0, 0))
    self.connect((self.blocks_throttle_0, 0),
(self.blks2_packet_encoder_0, 0))

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    frecuencia = self.carrier_freq
    if self.potencia < -10:
        self.potencia = -10
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)

```

```

        self._potencia_text_box.set_value(self.potencia)
        self.if_gain = get_ajuste(self)
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
        actualizar_label(self, self.parent, self.potencia)

    def get_if_gain(self):
        return self.if_gain

    def set_if_gain(self, if_gain):
        self.if_gain = if_gain
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

    def get_corr(self):
        return self.corr

    def set_corr(self, corr):
        self.corr = corr
        self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

    def get_carrier_freq(self):
        return self.carrier_freq

    def set_carrier_freq(self, carrier_freq):
        self.carrier_freq = carrier_freq
        if self.carrier_freq < 50e6:
            self.carrier_freq = 50e6
        elif self.carrier_freq > 3e9:
            self.carrier_freq = 3e9
        if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)

```

```

    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

```

Listado de programa GFSK_Video.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: GFSK con fuente de Video
# Generated: Tue May 12 10:02:34 2015
#####

from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import wx

class GFSK_video(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self,
            title="GFSK con fuente de Video")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

```

```

#####
# Variables
#####
self.parent = parent
self.sps = sps = 2
self.bitrate = param3
self.samp_rate = samp_rate = (self.bitrate / 1) * self.sps * 1000
self.potencia = potencia = float(int(pot))
self.if_gain = if_gain = 20
self.corr = corr = -8
self.carrier_freq = carrier_freq = freq * 1e6

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label="Frecuencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(

```

```

parent=self.GetWin(),
sizer=_potencia_sizer,
value=self.potencia,
callback=self.set_potencia,
label="Potencia",
converter=forms.float_converter(),
proportion=0,
)
self._potencia_slider = forms.slider(
parent=self.GetWin(),
sizer=_potencia_sizer,
value=self.potencia,
callback=self.set_potencia,
minimum=-10,
maximum=5,
num_steps=15,
style=wx.SL_HORIZONTAL,
cast=float,
proportion=1,
)
self.Add(_potencia_sizer)
self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
self.osmosdr_sink_0.set_sample_rate(samp_rate)
self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
self.osmosdr_sink_0.set_freq_corr(corr, 0)
self.osmosdr_sink_0.set_gain(10, 0)
self.osmosdr_sink_0.set_if_gain(if_gain, 0)
self.osmosdr_sink_0.set_bb_gain(20, 0)
self.osmosdr_sink_0.set_antenna("", 0)
self.osmosdr_sink_0.set_bandwidth(0, 0)

self.set_potencia(potencia)
set_corr(self)
actualizar_label(self, self.parent, self.potencia)

self.digital_gfsk_mod_0 = digital.gfsk_mod(
samples_per_symbol=sps,
sensitivity=1.0,
bt=param1,
verbose=False,

```

```

        log=False,
    )
    self.blocks_file_source_0 =
blocks.file_source(gr.sizeof_char * 1, onda, True)
    self.blocks_throttle_0 =
blocks.throttle(gr.sizeof_char*1, samp_rate)
    self.blks2_packet_encoder_0 =
grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
        samples_per_symbol=sps,
        bits_per_symbol=1,
        preamble="",
        access_code="",
        pad_for_usrp=True,
),
        payload_length=0,
)

#####
# Connections
#####
self.connect((self.digital_gfsk_mod_0, 0),
(self.osmosdr_sink_0, 0))
    self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_gfsk_mod_0, 0))
    self.connect((self.blocks_file_source_0, 0),
(self.blocks_throttle_0, 0))
    self.connect((self.blocks_throttle_0, 0),
(self.blks2_packet_encoder_0, 0))

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

```

```

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    frecuencia = self.carrier_freq
    if self.potencia < -10:
        self.potencia = -10
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
        elif frecuencia >= 2750e6:
            if self.potencia >= -5:
                self.potencia = -5
            self._potencia_slider.set_value(self.potencia)
            self._potencia_text_box.set_value(self.potencia)
            self.if_gain = get_ajuste(self)
            self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
            actualizar_label(self, self.parent, self.potencia)

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

```

```

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

```

Listado de programa GMSK_Audio.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: GMSK con fuente de Audio
# Generated: Sat May  9 19:37:19 2015
#####

from gnuradio import audio
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr

```

```

from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import wx

class GMSK_audio(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self,
                                         title="GMSK con fuente de Audio")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

        #####
        # Variables
        #####
        self.parent = parent
        self.sps = sps = 4
        self.samp_rate = samp_rate = 1e6
        self.potencia = potencia = float(int(pot))
        self.if_gain = if_gain = 20
        self.corr = corr = -8
        self.carrier_freq = carrier_freq = freq * 1e6

        #####
        # Blocks
        #####
        _carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
        self._carrier_freq_text_box = forms.text_box(
            parent=self.GetWin(),
            sizer=_carrier_freq_sizer,
            value=self.carrier_freq,
            callback=self.set_carrier_freq,
            label="Frecuencia",
        )

```

```

        converter=forms.float_converter(),
        proportion=0,
    )
    self._carrier_freq_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_carrier_freq_sizer,
        value=self.carrier_freq,
        callback=self.set_carrier_freq,
        minimum=50e6,
        maximum=3e9,
        num_steps=295,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_carrier_freq_sizer)
    _potencia_sizer = wx.BoxSizer(wx.VERTICAL)
    self._potencia_text_box = forms.text_box(
        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        label="Potencia",
        converter=forms.float_converter(),
        proportion=0,
    )
    self._potencia_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        minimum=-10,
        maximum=5,
        num_steps=15,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_potencia_sizer)
    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")

```

```

        self.osmosdr_sink_0.set_sample_rate(samp_rate)
        self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
        self.osmosdr_sink_0.set_freq_corr(corr, 0)
        self.osmosdr_sink_0.set_gain(10, 0)
        self.osmosdr_sink_0.set_if_gain(if_gain, 0)
        self.osmosdr_sink_0.set_bb_gain(20, 0)
        self.osmosdr_sink_0.set_antenna("", 0)
        self.osmosdr_sink_0.set_bandwidth(0, 0)

        self.set_potencia(potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

        self.digital_gmsk_mod_0 = digital.gmsk_mod(
            samples_per_symbol=sps,
            bt=param1,
            verbose=False,
            log=False,
        )
        self.blks2_packet_encoder_0 =
grc_blks2.packet_mod_f(grc_blks2.packet_encoder(
            samples_per_symbol=sps,
            bits_per_symbol=1,
            preamble="",
            access_code="",
            pad_for_usrp=True,
),
            payload_length=0,
)
        self.audio_source_0 = audio.source(48000, "", True)

#####
# Connections
#####
        self.connect((self.digital_gmsk_mod_0, 0),
(self.osmosdr_sink_0, 0))
        self.connect((self.audio_source_0, 0),
(self.blks2_packet_encoder_0, 0))
        self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_gmsk_mod_0, 0))

```

```

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    frecuencia = self.carrier_freq
    if self.potencia < -10:
        self.potencia = -10
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = get_ajuste(self)
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
    actualizar_label(self, self.parent, self.potencia)

def get_if_gain(self):
    return self.if_gain

```

```
def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)
```

Listado de programa GMSK_Random.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#####
# Gnuradio Python Flow Graph
# Title: GMSK Senal Aleatoria
# Generated: Sat May 9 17:57:09 2015
#####

from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import numpy
import osmosdr
import time
import wx

class GMSK_Random(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self, title="GMSK Señal Aleatoria")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.sps = sps = 2
self.bitrate = param3
self.samp_rate = samp_rate = (self.bitrate / 1) * self.sps * 1000
```

```

self.potencia = potencia = float(int(pot))
self.if_gain = if_gain = 20
self.corr = corr = -8
self.carrier_freq = carrier_freq = freq * 1e6

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label='Frecuencia',
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)

```

```

        )

        self._potencia_slider = forms.slider(
            parent=self.GetWin(),
            sizer=_potencia_sizer,
            value=self.potencia,
            callback=self.set_potencia,
            minimum=-10,
            maximum=5,
            num_steps=15,
            style=wx.SL_HORIZONTAL,
            cast=float,
            proportion=1,
        )

        self.Add(_potencia_sizer)

        self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
        self.osmosdr_sink_0.set_sample_rate(samp_rate)
        self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
        self.osmosdr_sink_0.set_freq_corr(corr, 0)
        self.osmosdr_sink_0.set_gain(7, 0)
        self.osmosdr_sink_0.set_if_gain(if_gain, 0)
        self.osmosdr_sink_0.set_bb_gain(20, 0)
        self.osmosdr_sink_0.set_antenna("", 0)
        self.osmosdr_sink_0.set_bandwidth(0, 0)

        self.set_potencia(potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

        self.digital_gmsk_mod_0 = digital.gmsk_mod(
            samples_per_symbol=sps,
            bt=param1,
            verbose=False,
            log=False,
        )

        self.blocks_throttle_0 = blocks.throttle(gr.sizeof_char * 1, samp_rate)
        self.blks2_packet_encoder_0 =
        grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
            samples_per_symbol=sps,
            bits_per_symbol=1,
            preamble="",

```

```

        access_code="",
        pad_for_usrp=True,
),
        payload_length=0,
)
self.analog_random_source_x_0 =
blocks.vector_source_b(map(int, numpy.random.randint(0, 256, 1000)), True)

#####
# Connections
#####
self.connect((self.blocks_throttle_0, 0),
(self.blks2_packet_encoder_0, 0))
    self.connect((self.analog_random_source_x_0, 0),
(self.blocks_throttle_0, 0))
    self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_gmsk_mod_0, 0))
    self.connect((self.digital_gmsk_mod_0, 0),
(self.osmosdr_sink_0, 0))

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):

```

```

        self.potencia = float(int(potencia))
        frecuencia = self.carrier_freq
        if self.potencia < -10:
            self.potencia = -10
        if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
            if self.potencia >= 0:
                self.potencia = 0
        elif frecuencia >= 2750e6:
            if self.potencia >= -5:
                self.potencia = -5
        self._potencia_slider.set_value(self.potencia)
        self._potencia_text_box.set_value(self.potencia)
        self.if_gain = get_ajuste(self)
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
        actualizar_label(self, self.parent, self.potencia)

    def get_if_gain(self):
        return self.if_gain

    def set_if_gain(self, if_gain):
        self.if_gain = if_gain
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

    def get_corr(self):
        return self.corr

    def set_corr(self, corr):
        self.corr = corr
        self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

    def get_carrier_freq(self):
        return self.carrier_freq

    def set_carrier_freq(self, carrier_freq):
        self.carrier_freq = carrier_freq
        if self.carrier_freq < 50e6:
            self.carrier_freq = 50e6
        elif self.carrier_freq > 3e9:
            self.carrier_freq = 3e9
        if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):

```

```

        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

```

Listado de programa GMSK_UDP.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: GMSK con fuente de Streaming
# Generated: Tue May 12 10:19:59 2015
#####

from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr

```

```

import time
import wx

class GMSK_UDP(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self,
        title="GMSK con fuente de Streaming")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.sps = sps = 2
self.bitrate = param3
self.samp_rate = samp_rate = (self.bitrate / 1) * self.sps * 1000
self.potencia = potencia = float(int(pot))
self.if_gain = if_gain = 20
self.corr = corr = -8
self.carrier_freq = carrier_freq = freq * 1e6

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label="Frecuencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
)

```

```

        callback=self.set_carrier_freq,
        minimum=50e6,
        maximum=3e9,
        num_steps=295,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_carrier_freq_sizer)
    _potencia_sizer = wx.BoxSizer(wx.VERTICAL)
    self._potencia_text_box = forms.text_box(
        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        label="Potencia",
        converter=forms.float_converter(),
        proportion=0,
    )
    self._potencia_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        minimum=-10,
        maximum=5,
        num_steps=15,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_potencia_sizer)
    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(corr, 0)
    self.osmosdr_sink_0.set_gain(10, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)
    self.osmosdr_sink_0.set_bb_gain(20, 0)
    self.osmosdr_sink_0.set_antenna("", 0)

```

```

        self.osmosdr_sink_0.set_bandwidth(0, 0)

        self.if_gain = get_ajuste(self)
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
        set_corr(self)

        self.digital_gmsk_mod_0 = digital.gmsk_mod(
            samples_per_symbol=sps,
            bt=param1,
            verbose=False,
            log=False,
        )
        self.blocks_udp_source_0 =
blocks.udp_source(gr.sizeof_char * 1, onda, param2, 1472, True)
        self.blocks_throttle_0 = blocks.throttle(gr.sizeof_char*1, samp_rate)
        self.blks2_packet_encoder_0 =
grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
            samples_per_symbol=sps,
            bits_per_symbol=1,
            preamble="",
            access_code="",
            pad_for_usrp=True,
),
            payload_length=0,
)
        #####
# Connections
#####
        self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_gmsk_mod_0, 0))
        self.connect((self.digital_gmsk_mod_0, 0),
(self.osmosdr_sink_0, 0))
        self.connect((self.blocks_throttle_0, 0),
(self.blks2_packet_encoder_0, 0))
        self.connect((self.blocks_udp_source_0, 0),
(self.blocks_throttle_0, 0))

# QT sink close method reimplementation

```

```

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    frecuencia = self.carrier_freq
    if self.potencia < -10:
        self.potencia = -10
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = get_ajuste(self)
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
    actualizar_label(self, self.parent, self.potencia)

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

```

```

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

```

Listado de programa GMSK_Video.py

```
#!/usr/bin/env python
```

```

#####
# Gnuradio Python Flow Graph
# Title: GMSK con fuente de Video
# Generated: Tue May 12 10:20:04 2015
#####

from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import wx

class GMSK_video(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self,
                                         title="GMSK con fuente de Video")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.sps = sps = 2
self.bitrate = param3
self.samp_rate = samp_rate = (self.bitrate / 1) * self.sps * 1000
self.potencia = potencia = float(int(pot))
self.if_gain = if_gain = 20
self.corr = corr = -8
self.carrier_freq = carrier_freq = freq * 1e6

```

```

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label="Frecuencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_potencia_sizer,

```

```

        value=self.potencia,
        callback=self.set_potencia,
        minimum=-10,
        maximum=5,
        num_steps=15,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_potencia_sizer)
    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(corr, 0)
    self.osmosdr_sink_0.set_gain(10, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)
    self.osmosdr_sink_0.set_bb_gain(20, 0)
    self.osmosdr_sink_0.set_antenna("", 0)
    self.osmosdr_sink_0.set_bandwidth(0, 0)

    self.set_potencia(potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

    self.digital_gmsk_mod_0 = digital.gmsk_mod(
        samples_per_symbol=sps,
        bt=param1,
        verbose=False,
        log=False,
    )
    self.blocks_file_source_0 =
blocks.file_source(gr.sizeof_char * 1, onda, True)
    self.blocks_throttle_0 =
blocks.throttle(gr.sizeof_char*1, samp_rate)
    self.blks2_packet_encoder_0 =
grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
        samples_per_symbol=sps,
        bits_per_symbol=1,
        preamble="",
        access_code="",

```

```

        pad_for_usrp=True,
),
        payload_length=0,
)

#####
# Connections
#####
self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_gmsk_mod_0, 0))
    self.connect((self.digital_gmsk_mod_0, 0),
(self.osmosdr_sink_0, 0))
    self.connect((self.blocks_file_source_0, 0),
(self.blocks_throttle_0, 0))
    self.connect((self.blocks_throttle_0, 0),
(self.blks2_packet_encoder_0, 0))

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    frecuencia = self.carrier_freq
    if self.potencia < -10:

```

```

        self.potencia = -10
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = get_ajuste(self)
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
    actualizar_label(self, self.parent, self.potencia)

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
    actualizar_label(self, self.parent, self.potencia)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)

```

```

        elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        else:
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
            self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
            self.set_potencia(self.potencia)
            set_corr(self)
            actualizar_label(self, self.parent, self.potencia)

```

Listado de programa NBFM_Audio.py

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-
#####
# Gnuradio Python Flow Graph
# Title: NBFM Audio
# Generated: Sat May  9 18:49:01 2015
#####

from gnuradio import analog
from gnuradio import audio
from gnuradio import eng_notation
from gnuradio import filter
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time

```

```

import wx

class NBFM_Audio(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self, title="NBFM Audio")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

        #####
        # Variables
        #####
        self.parent = parent
        self.samp_rate = samp_rate = 200e3
        self.potencia = potencia = float(int(pot))
        self.max_freq_dev = max_freq_dev = param2
        self.if_gain = if_gain = 20
        self.corr = corr = -8
        self.carrier_freq = carrier_freq = freq * 1e6

        #####
        # Blocks
        #####
        _carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
        self._carrier_freq_text_box = forms.text_box(
            parent=self.GetWin(),
            sizer=_carrier_freq_sizer,
            value=self.carrier_freq,
            callback=self.set_carrier_freq,
            label="Frecuencia",
            converter=forms.float_converter(),
            proportion=0,
        )
        self._carrier_freq_slider = forms.slider(
            parent=self.GetWin(),
            sizer=_carrier_freq_sizer,
            value=self.carrier_freq,
            callback=self.set_carrier_freq,
            minimum=50e6,
            maximum=3e9,
        )

```

```

        num_steps=295,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_carrier_freq_sizer)
    self.rational_resampler_xxx_0_0 = filter.rational_resampler_ccc(
        interpolation=200000,
        decimation=480000,
        taps=None,
        fractional_bw=None,
    )
    _potencia_sizer = wx.BoxSizer(wx.VERTICAL)
    self._potencia_text_box = forms.text_box(
        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        label="Potencia",
        converter=forms.float_converter(),
        proportion=0,
    )
    self._potencia_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        minimum=-10,
        maximum=5,
        num_steps=15,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_potencia_sizer)
    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(corr, 0)
    self.osmosdr_sink_0.set_gain(7, 0)

```

```

        self.osmosdr_sink_0.set_if_gain(if_gain, 0)
        self.osmosdr_sink_0.set_bb_gain(20, 0)
        self.osmosdr_sink_0.set_antenna("", 0)
        self.osmosdr_sink_0.set_bandwidth(0, 0)

        self.set_potencia(potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

        self.low_pass_filter_0 = filter.fir_filter_fff(1, firdes.low_pass(
            2, 48000, 22000, 10000, firdes.WIN_HAMMING, 6.76))
        self.audio_source_0 = audio.source(48000, "", True)
        self.analog_nbfm_tx_0 = analog.nbfm_tx(
            audio_rate=48000,
            quad_rate=480000,
            tau=75e-6,
            max_dev=max_freq_dev,
        )

#####
# Connections
#####
        self.connect((self.rational_resampler_xxx_0_0, 0),
        (self.osmosdr_sink_0, 0))
        self.connect((self.audio_source_0, 0),
        (self.low_pass_filter_0, 0))
        self.connect((self.low_pass_filter_0, 0),
        (self.analog_nbfm_tx_0, 0))
        self.connect((self.analog_nbfm_tx_0, 0),
        (self.rational_resampler_xxx_0_0, 0))

# QT sink close method reimplementation

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

```

```

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    frecuencia = self.carrier_freq
    if self.potencia < -10:
        self.potencia = -10
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = get_ajuste(self)
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
    actualizar_label(self, self.parent, self.potencia)

def get_max_freq_dev(self):
    return self.max_freq_dev

def set_max_freq_dev(self, max_freq_dev):
    self.max_freq_dev = max_freq_dev

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

```

```

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

```

Listado de programa NBFM_vector.py

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-

#####
# Gnuradio Python Flow Graph
# Title: NBFM con senal arbitraria
# Generated: Tue May 12 19:20:20 2015
#####

```

```

from gnuradio import analog
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
from nbfm_tx import *
import osmosdr
import time
import wx

class NBFM_vector(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self,
        title="NBFM con señal arbitraria")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.vector = vector = onda
self.samp_rate = samp_rate = 2e6
self.potencia = potencia = float(int(pot))
self.mod_freq = mod_freq = param1
self.max_freq_dev = max_freq_dev = param2
self.if_gain = if_gain = 20
self.corr = corr = -8
self.carrier_freq = carrier_freq = freq * 1e6

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)

```

```
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label="Frecuencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    minimum=-10,
    maximum=5,
    num_steps=15,
```

```

        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_potencia_sizer)
    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(corr, 0)
    self.osmosdr_sink_0.set_gain(7, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)
    self.osmosdr_sink_0.set_bb_gain(20, 0)
    self.osmosdr_sink_0.set_antenna("", 0)
    self.osmosdr_sink_0.set_bandwidth(0, 0)

    self.set_potencia(potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

    self.blocks_vector_source_x_0 =
blocks.vector_source_f(vector, True, 1, [])
    self.blocks_multiply_const_vxx_0 = blocks.multiply_const_vff(
((1.0 / max(abs(i) for i in vector)), ))
    self.analog_nbfm_tx_0 = nbfm_tx(
        audio_rate=2000000,
        quad_rate=2000000,
        tau=75e-6,
        max_dev=max_freq_dev,
    )

#####
# Connections
#####
    self.connect((self.analog_nbfm_tx_0, 0),
(self.osmosdr_sink_0, 0))
    self.connect((self.blocks_vector_source_x_0, 0),
(self.blocks_multiply_const_vxx_0, 0))
    self.connect((self.blocks_multiply_const_vxx_0, 0),
(self.analog_nbfm_tx_0, 0))

```

```

# QT sink close method reimplementation

def get_vector(self):
    return self.vector

def set_vector(self, vector):
    self.vector = vector
    self.blocks_multiply_const_vxx_0.set_k(
((1.0 / max(abs(i) for i in self.vector)),))

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    frecuencia = self.carrier_freq
    if self.potencia < -10:
        self.potencia = -10
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = get_ajuste(self)
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
    actualizar_label(self, self.parent, self.potencia)

def get_mod_freq(self):
    return self.mod_freq

```

```

def set_mod_freq(self, mod_freq):
    self.mod_freq = mod_freq

def get_max_freq_dev(self):
    return self.max_freq_dev

def set_max_freq_dev(self, max_freq_dev):
    self.max_freq_dev = max_freq_dev

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):

```

```

        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

```

Listado de programa NBFM.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: NBFM
# Generated: Sat May  9 18:49:03 2015
#####

from gnuradio import analog
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
from nbfm_tx import *
import osmosdr
import time
import wx

class NBFM(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self, title="NBFM")

```

```

_icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.samp_rate = samp_rate = 2e6
self.potencia = potencia = pot
self.mod_freq = mod_freq = param1
self.max_freq_dev = max_freq_dev = int(param2)
self.if_gain = if_gain = 20
self.corr = corr = -8
self.carrier_freq = carrier_freq = freq * 1e6

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label="Frecuencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)

```

```

        self.Add(_carrier_freq_sizer)
        _potencia_sizer = wx.BoxSizer(wx.VERTICAL)
        self._potencia_text_box = forms.text_box(
            parent=self.GetWin(),
            sizer=_potencia_sizer,
            value=self.potencia,
            callback=self.set_potencia,
            label="Potencia",
            converter=forms.float_converter(),
            proportion=0,
        )
        self._potencia_slider = forms.slider(
            parent=self.GetWin(),
            sizer=_potencia_sizer,
            value=self.potencia,
            callback=self.set_potencia,
            minimum=-10,
            maximum=5,
            num_steps=15,
            style=wx.SL_HORIZONTAL,
            cast=float,
            proportion=1,
        )
        self.Add(_potencia_sizer)
        self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
        self.osmosdr_sink_0.set_sample_rate(samp_rate)
        self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
        self.osmosdr_sink_0.set_freq_corr(corr, 0)
        self.osmosdr_sink_0.set_gain(7, 0)
        self.osmosdr_sink_0.set_if_gain(if_gain, 0)
        self.osmosdr_sink_0.set_bb_gain(20, 0)
        self.osmosdr_sink_0.set_antenna("", 0)
        self.osmosdr_sink_0.set_bandwidth(0, 0)

        self.set_potencia(potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

        self.analog_sig_source_x_0 =
analog.sig_source_f(2000000, onda, mod_freq, 1, 0)

```

```

        self.analog_nbfm_tx_0 = nbfm_tx(
            audio_rate=2000000,
            quad_rate=2000000,
            tau=75e-6,
            max_dev=max_freq_dev,
        )

#####
# Connections
#####
self.connect((self.analog_nbfm_tx_0, 0),
(self.osmosdr_sink_0, 0))
    self.connect((self.analog_sig_source_x_0, 0),
(self.analog_nbfm_tx_0, 0))

# QT sink close method reimplementation

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    frecuencia = self.carrier_freq
    if self.potencia < -10:
        self.potencia = -10
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)

```

```

        self._potencia_text_box.set_value(self.potencia)
        self.if_gain = get_ajuste(self)
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
        actualizar_label(self, self.parent, self.potencia)

    def get_mod_freq(self):
        return self.mod_freq

    def set_mod_freq(self, mod_freq):
        self.mod_freq = mod_freq
        self.analog_sig_source_x_0.set_frequency(self.mod_freq)

    def get_max_freq_dev(self):
        return self.max_freq_dev

    def set_max_freq_dev(self, max_freq_dev):
        self.max_freq_dev = max_freq_dev

    def get_if_gain(self):
        return self.if_gain

    def set_if_gain(self, if_gain):
        self.if_gain = if_gain
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

    def get_corr(self):
        return self.corr

    def set_corr(self, corr):
        self.corr = corr
        self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

    def get_carrier_freq(self):
        return self.carrier_freq

    def set_carrier_freq(self, carrier_freq):
        self.carrier_freq = carrier_freq
        if self.carrier_freq < 50e6:
            self.carrier_freq = 50e6
        elif self.carrier_freq > 3e9:

```

```

        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

```

Listado de programa ntsc_encode.py

```

#!/usr/bin/env python
# Copyright 2014 Clayton Smith
#
# This file is part of sdr-examples
#
# sdr-examples is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 3, or (at your option)
# any later version.
#
# sdr-examples is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with sdr-examples; see the file COPYING. If not, write to

```

```

# the Free Software Foundation, Inc., 51 Franklin Street,
# Boston, MA 02110-1301, USA.

from PIL import Image
from array import array
import math

def convertir_imagen(path_imagen):
    global ntsc_signal, COLOR_FREQ, SAMPLES_PER_LINE, SAMP_RATE,
           RADIANS_PER_SAMPLE, SYNCH_LEVEL, BLANKING_LEVEL,\n
           BLACK_LEVEL, WHITE_LEVEL, EQUALIZING_PULSE, SYNCHRONIZING_PULSE,\n
           INTERVALS, EXTRA_HALF_LINE, FRONT_PORCH,\n
           SYNCH_PULSE
    image = Image.open(path_imagen)
    size = (640,480)
    im = image.resize((640, 480), Image.ANTIALIAS)

    pixels = list(im.getdata())

    COLOR_FREQ = 3579545.0
    SAMPLES_PER_LINE = 772
    SAMP_RATE = SAMPLES_PER_LINE * 60 * .999 * 525 / 2
    RADIANS_PER_SAMPLE = 2 * math.pi * COLOR_FREQ / SAMP_RATE

    SYNCH_LEVEL = -40.0
    BLANKING_LEVEL = 0.0
    BLACK_LEVEL = 7.5
    WHITE_LEVEL = 100.0

    EQUALIZING_PULSE = [SYNCH_LEVEL] * 28 + [BLANKING_LEVEL] * 358
    SYNCHRONIZING_PULSE = [SYNCH_LEVEL] * 329 + [BLANKING_LEVEL] * 57
    INTERVALS =
    EQUALIZING_PULSE * 6 + SYNCHRONIZING_PULSE * 6 + EQUALIZING_PULSE * 6
    EXTRA_HALF_LINE = [BLANKING_LEVEL] * 386

    FRONT_PORCH = [BLANKING_LEVEL] * 18
    SYNCH_PULSE = [SYNCH_LEVEL] * 57

def addBackPorch():
    global ntsc_signal

```

```

ntsc_signal += [BLANKING_LEVEL] * 13
l = len(ntsc_signal)
for x in range(l, l+31):
    ntsc_signal +=
[BLANKING_LEVEL + 20 * math.sin(math.pi + RADIANS_PER_SAMPLE * x)]
ntsc_signal += [BLANKING_LEVEL] * 13

def addNonVisibleLine():
    global ntsc_signal
    ntsc_signal += SYNCH_PULSE
    addBackPorch()
    ntsc_signal += [BLANKING_LEVEL] * 658

def addFirstHalfFrame():
    global ntsc_signal
    ntsc_signal += SYNCH_PULSE
    addBackPorch()
    ntsc_signal += [BLACK_LEVEL] * 272

def addSecondHalfFrame():
    global ntsc_signal
    ntsc_signal += SYNCH_PULSE
    addBackPorch()
    ntsc_signal +=
[BLANKING_LEVEL] * 272 + [BLACK_LEVEL] * 368 + FRONT_PORCH

def addPixel(p):
    global ntsc_signal, BLACK_LEVEL, WHITE_LEVEL
    Er = float(p[0]) / 255
    Eg = float(p[1]) / 255
    Eb = float(p[2]) / 255

    Ey = 0.30 * Er + 0.59 * Eg + 0.11 * Eb
    Eq = 0.41 * (Eb - Ey) + 0.48 * (Er - Ey)
    Ei = -0.27 * (Eb - Ey) + 0.74 * (Er - Ey)

    phase =
RADIANS_PER_SAMPLE * len(ntsc_signal) + (33.0 / 180 * math.pi)
    Em = Ey + Eq * math.sin(phase) + Ei * math.cos(phase)

```

```

        ntsc_signal +=
[BLOCK_LEVEL + (WHITE_LEVEL - BLACK_LEVEL) * Em]

ntsc_signal = []

# Generate even field
ntsc_signal += INTERVALS
for x in range(13):
    addNonVisibleLine()
for line in range(0,480,2):
    ntsc_signal += SYNCH_PULSE
    addBackPorch()
    for x in range(line * 640, (line+1) * 640):
        addPixel(pixels[x])
    ntsc_signal += FRONT_PORCH
addFirstHalfFrame()

# Generate odd field
ntsc_signal += INTERVALS + EXTRA_HALF_LINE
for x in range(12):
    addNonVisibleLine()
addSecondHalfFrame()
for line in range(1,481,2):
    ntsc_signal += SYNCH_PULSE
    addBackPorch()
    for x in range(line * 640, (line+1) * 640):
        addPixel(pixels[x])
    ntsc_signal += FRONT_PORCH

ntsc_signal = [0.75 - (0.25/40) * x for x in ntsc_signal]

f = open('imagen_convertida.dat', 'wb')
ntsc_array = array('f', ntsc_signal)
ntsc_array.tofile(f)
f.close()

```

Listado de programa ntsc_hackrf.py

```

#!/usr/bin/env python
#####

```

```

# Gnuradio Python Flow Graph
# Title: Estandar de television NTSC
# Generated: Tue May 12 11:36:25 2015
#####
# Importación de módulos
from gnuradio import analog
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import filter
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import wx

# Definición de la clase ntsc_hackrf
class ntsc_hackrf(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self,
                                        title="Estandar de television NTSC")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.samples_per_line = samples_per_line = 772
self.samp_rate = samp_rate = samples_per_line * 60 * .999 * 525 / 2
self.potencia = potencia = float(int(pot))
self.if_gain = if_gain = 20
self.digital_gain = digital_gain = 0.9
self.corr = corr = -8
self.carrier_freq = carrier_freq = freq * 1e6

#####

```

```

# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label="Frecuencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=55.25e6,
    maximum=799.25e6,
    num_steps=124,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
self.zero = analog.sig_source_f(0, analog.GR_CONST_WAVE, 0, 0, 0)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,

```

```

        callback=self.set_potencia,
        minimum=-10,
        maximum=0,
        num_steps=10,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_potencia_sizer)
self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
self.osmosdr_sink_0.set_sample_rate(samp_rate)
self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
self.osmosdr_sink_0.set_freq_corr(corr, 0)
self.osmosdr_sink_0.set_gain(7, 0)
self.osmosdr_sink_0.set_if_gain(if_gain, 0)
self.osmosdr_sink_0.set_bb_gain(20, 0)
self.osmosdr_sink_0.set_antenna("", 0)
self.osmosdr_sink_0.set_bandwidth(0, 0)

self.set_potencia(potencia)
set_corr(self)
actualizar_label(self, self.parent, self.potencia)

self.blocks_multiply_const_vxx_0 =
blocks.multiply_const_vff((digital_gain, ))
self.blocks_float_to_complex_0 =
blocks.float_to_complex(1)
self.blocks_file_source_0 =
blocks.file_source(gr.sizeof_float * 1, 'imagen_convertida.dat', True)
self.blocks_add_xx_0 = blocks.add_vcc(1)
self.band_pass_filter_0 =
filter.fir_filter_ccc(1, firdes.complex_band_pass(
    1, samp_rate, (2475000 + 1725000) * -1,
(-2475000 + 1725000) * -1, 500000, firdes.WIN_HAMMING, 6.76))
self.analog_sig_source_x_1 =
analog.sig_source_c(samp_rate, analog.GR_COS_WAVE, -4500000, 0.1, 0)

#####
# Connections
#####

```

```

        self.connect((self.zero, 0), (self.blocks_float_to_complex_0, 1))
        self.connect((self.blocks_multiply_const_vxx_0, 0),
(self.blocks_float_to_complex_0, 0))
            self.connect((self.blocks_file_source_0, 0),
(self.blocks_multiply_const_vxx_0, 0))
            self.connect((self.blocks_float_to_complex_0, 0),
(self.band_pass_filter_0, 0))
            self.connect((self.band_pass_filter_0, 0),
(self.blocks_add_xx_0, 1))
            self.connect((self.analog_sig_source_x_1, 0),
(self.blocks_add_xx_0, 0))
            self.connect((self.blocks_add_xx_0, 0), (self.osmosdr_sink_0, 0))

def get_samples_per_line(self):
    return self.samples_per_line

def set_samples_per_line(self, samples_per_line):
    self.samples_per_line = samples_per_line
    self.set_samp_rate(self.samples_per_line * 60 * .999 * 525 / 2)

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.analog_sig_source_x_1.set_sampling_freq(self.samp_rate)
    self.band_pass_filter_0.set_taps(
        firdes.complex_band_pass(1, self.samp_rate,
(2475000 + 1725000) * -1, (-2475000 + 1725000) * -1,
500000, firdes.WIN_HAMMING, 6.76))
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    frecuencia = self.carrier_freq
    if self.potencia < -10:

```

```

        self.potencia = -10
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = get_ajuste_ntsc(self)
    #print self.if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
    actualizar_label(self, self.parent, self.potencia)

def get_digital_gain(self):
    return self.digital_gain

def set_digital_gain(self, digital_gain):
    self.digital_gain = digital_gain
    self.blocks_multiply_const_vxx_0.set_k((self.digital_gain, ))

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_center_freq(self):
    return self.center_freq

def set_center_freq(self, center_freq):
    self.center_freq = center_freq

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 55.25e6:

```

```

        self.carrier_freq = 55.25e6
    elif self.carrier_freq > 799.25e6:
        self.carrier_freq = 799.25e6
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

```

Listado de programa OFDM_Randomv2.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: OFDM con fuente Aleatoria
# Generated: Wed Jun  3 17:27:23 2015
#####

from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2

```

```

from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import numpy
import osmosdr
import time
import wx

class OFDM_Random(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self,
        title="OFDM con fuente Aleatoria")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.samp_rate = samp_rate = 2e6
self.prefijo = prefijo = int(param3)
self.potencia = potencia = float(int(pot))
self.fftlen = fftlen = int(param1)
self.corr = corr = -8
self.carrier_freq = carrier_freq = freq * 1e6
self.canales = canales = int(param2)
self.onda = onda[0]
self.if_gain = 20
self.ajustes = {20: 6, 30: 5, 40: 4, 50: 3,
60: 2, 70: 2, 80: 1, 90: 1, 100: 1,}

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,

```

```
        callback=self.set_carrier_freq,
        label="Frecuencia",
        converter=forms.float_converter(),
        proportion=0,
    )
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    minimum=-10,
    maximum=0,
    num_steps=10,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
```

```

        self.Add(_potencia_sizer)
        self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
        self.osmosdr_sink_0.set_sample_rate(samp_rate)
        self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
        self.osmosdr_sink_0.set_freq_corr(corr, 0)
        self.osmosdr_sink_0.set_gain(10, 0)
        self.osmosdr_sink_0.set_if_gain(20, 0)
        self.osmosdr_sink_0.set_bb_gain(20, 0)
        self.osmosdr_sink_0.set_antenna("", 0)
        self.osmosdr_sink_0.set_bandwidth(0, 0)

        self.set_potencia(potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

        self.digital_ofdm_mod_0 = grc_blk2.packet_mod_f(digital.ofdm_mod(
            options=grc_blk2.options(
                modulation=onda[0],
                fft_length=ffflen,
                occupied_tones=canales,
                cp_length=prefijo,
                pad_for_usrp=True,
                log=None,
                verbose=None,
            ),
            ),
            payload_length=0,
        )
        self.blocks_throttle_0 =
blocks.throttle(gr.sizeof_gr_complex * 1, samp_rate)
        self.blocks_short_to_float_0 = blocks.short_to_float(1, 1)
        self.analog_random_source_x_0 =
blocks.vector_source_s(map(int, numpy.random.randint(0, 2, 1000)), True)

#####
# Connections
#####
self.connect((self.analog_random_source_x_0, 0),
(self.blocks_short_to_float_0, 0))
self.connect((self.blocks_short_to_float_0, 0),

```

```

(self.digital_ofdm_mod_0, 0))
    self.connect((self.digital_ofdm_mod_0, 0),
(self.blocks_throttle_0, 0))
    self.connect((self.blocks_throttle_0, 0),
(self.osmosdr_sink_0, 0))

# QT sink close method reimplementation

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.blocks_throttle_0.set_sample_rate(self.samp_rate)
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_prefijo(self):
    return self.prefijo

def set_prefijo(self, prefijo):
    self.prefijo = prefijo

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    frecuencia = self.carrier_freq
    if self.potencia < -10:
        self.potencia = -10
    if (frecuencia >= 50e6) and (frecuencia <= 2749e6):
        if self.potencia >= -4:
            self.potencia = -4
    elif frecuencia > 2749e6:
        if self.potencia >= -10:
            self.potencia = -10
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = self.ajuste_ofdm()

```

```

        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
        actualizar_label(self, self.parent, self.potencia)

    def get_fftlen(self):
        return self.fftlen

    def set_fftlen(self, fftlen):
        self.fftlen = fftlen

    def get_corr(self):
        return self.corr

    def set_corr(self, corr):
        self.corr = corr
        self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

    def get_carrier_freq(self):
        return self.carrier_freq

    def set_carrier_freq(self, carrier_freq):
        self.carrier_freq = carrier_freq
        if self.carrier_freq < 50e6:
            self.carrier_freq = 50e6
        elif self.carrier_freq > 3e9:
            self.carrier_freq = 3e9
        if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        else:
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)

```

```

        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

def get_canales(self):
    return self.canales

def set_canales(self, canales):
    self.canales = canales

def ajuste_ofdm(self):
    porcentanje_BW = (float(self.canales) / self.fftlen) * 100
    if 95 < porcentanje_BW <= 100:
        ajuste = 1
    elif 85 < porcentanje_BW <= 95:
        ajuste = 1
    elif 75 < porcentanje_BW <= 85:
        ajuste = 1
    elif 65 < porcentanje_BW <= 75:
        ajuste = 2
    elif 55 < porcentanje_BW <= 65:
        ajuste = 2
    elif 45 < porcentanje_BW <= 55:
        ajuste = 3
    elif 35 < porcentanje_BW <= 45:
        ajuste = 4
    elif 25 < porcentanje_BW <= 35:
        ajuste = 5
    elif 20 <= porcentanje_BW <= 25:
        ajuste = 6
    self.if_gain = get_ajuste(self, ajuste, ajuste)
    if self.onda == 'qpsk':
        self.if_gain = get_ajuste(self, 0, 0, 2, 2)
    else:
        self.if_gain = get_ajuste(self, 0, 0, 1, 2)
    return self.if_gain

```

Listado de programa OOK_Audio.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: On-Off Keying con fuente de Audio
# Generated: Sat May  9 19:13:14 2015
#####

from gnuradio import audio
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import wx

class OOK_audio(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self,
        title="On-Off Keying con fuente de Audio")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.sps = sps = 2
self.samp_rate = samp_rate = 1e6
self.potencia = potencia = float(int(pot))
self.ook = ook = digital.constellation_rect(
([0 + 0j, 1 + 0j]), ([0, 1]), 2, 1, 2, 1, 1).base()
self.if_gain = if_gain = 20

```

```
self.corr = corr = -8
self.carrier_freq = carrier_freq = freq * 1e6

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label="Frecuencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._potencia_slider = forms.slider(
```

```

parent=self.GetWin(),
sizer=_potencia_sizer,
value=self.potencia,
callback=self.set_potencia,
minimum=-10,
maximum=5,
num_steps=15,
style=wx.SL_HORIZONTAL,
cast=float,
proportion=1,
)
self.Add(_potencia_sizer)
self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
self.osmosdr_sink_0.set_sample_rate(samp_rate)
self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
self.osmosdr_sink_0.set_freq_corr(corr, 0)
self.osmosdr_sink_0.set_gain(10, 0)
self.osmosdr_sink_0.set_if_gain(if_gain, 0)
self.osmosdr_sink_0.set_bb_gain(20, 0)
self.osmosdr_sink_0.set_antenna("", 0)
self.osmosdr_sink_0.set_bandwidth(0, 0)

self.set_potencia(potencia)
set_corr(self)
actualizar_label(self, self.parent, self.potencia)

self.digital_constellation_modulator_0 = digital.generic_mod(
    constellation=ook,
    differential=True,
    samples_per_symbol=sps,
    pre_diff_code=True,
    excess_bw=param1,
    verbose=False,
    log=False,
)
self.blks2_packet_encoder_0 =
grc_blks2.packet_mod_f(grc_blks2.packet_encoder(
    samples_per_symbol=sps,
    bits_per_symbol=1,
    preamble="",

```

```

        access_code="",
        pad_for_usrp=True,
),
        payload_length=0,
)
self.audio_source_0 = audio.source(48000, "", True)

#####
# Connections
#####
self.connect((self.digital_constellation_modulator_0, 0),
(self.osmosdr_sink_0, 0))
    self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_constellation_modulator_0, 0))
    self.connect((self.audio_source_0, 0),
(self.blks2_packet_encoder_0, 0))

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    frecuencia = self.carrier_freq
    if self.potencia < -10:

```

```

        self.potencia = -10
if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
    if self.potencia >= 0:
        self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = get_ajuste(self, 1, 1, 2, 3)
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
    actualizar_label(self, self.parent, self.potencia)

def get_ook(self):
    return self.ook

def set_ook(self, ook):
    self.ook = ook

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6

```

```

        elif self.carrier_freq > 3e9:
            self.carrier_freq = 3e9
        if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        else:
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
            self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
            self.set_potencia(self.potencia)
            set_corr(self)
            actualizar_label(self, self.parent, self.potencia)

```

Listado de programa OOK.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: On-Off Keying
# Generated: Sat May  9 19:12:08 2015
#####

from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui

```

```

from optparse import OptionParser
from Funciones import *
import numpy
import osmosdr
import time
import wx

class OOK(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self, title="On-Off Keying")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

        #####
        # Variables
        #####
        self.parent = parent
        self.sps = sps = 2
        self.bitrate = param3
        self.samp_rate = samp_rate = (self.bitrate / 1) * self.sps * 1000
        self.potencia = potencia = float(int(pot))
        self.ook = ook = digital.constellation_rect(
            ([0 + 0j, 1 + 0j]), ([0, 1]), 2, 1, 2, 1, 1).base()
        self.if_gain = if_gain = 20
        self.corr = corr = -8
        self.carrier_freq = carrier_freq = freq * 1e6

        #####
        # Blocks
        #####
        _carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
        self._carrier_freq_text_box = forms.text_box(
            parent=self.GetWin(),
            sizer=_carrier_freq_sizer,
            value=self.carrier_freq,
            callback=self.set_carrier_freq,
            label="Frecuencia",
            converter=forms.float_converter(),
            proportion=0,

```

```

)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    minimum=-10,
    maximum=5,
    num_steps=15,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_potencia_sizer)
self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
self.osmosdr_sink_0.set_sample_rate(samp_rate)
self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)

```

```

        self.osmosdr_sink_0.set_freq_corr(corr, 0)
        self.osmosdr_sink_0.set_gain(7, 0)
        self.osmosdr_sink_0.set_if_gain(if_gain, 0)
        self.osmosdr_sink_0.set_bb_gain(20, 0)
        self.osmosdr_sink_0.set_antenna("", 0)
        self.osmosdr_sink_0.set_bandwidth(0, 0)

        self.set_potencia(potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

        self.digital_constellation_modulator_0 = digital.generic_mod(
            constellation=ook,
            differential=True,
            samples_per_symbol=sps,
            pre_diff_code=True,
            excess_bw=param1,
            verbose=False,
            log=False,
        )
        self.blocks_throttle_0 =
blocks.throttle(gr.sizeof_char * 1, samp_rate)
        self.blks2_packet_encoder_0 =
grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
            samples_per_symbol=sps,
            bits_per_symbol=1,
            preamble="",
            access_code="",
            pad_for_usrp=True,
),
            payload_length=0,
)
        self.analog_random_source_x_0 =
blocks.vector_source_b(map(int,numpy.random.randint(0,256,1000)),True)

#####
# Connections
#####
self.connect((self.digital_constellation_modulator_0, 0),
(self.osmosdr_sink_0, 0))

```

```

        self.connect((self.blocks_throttle_0, 0),
(self.blks2_packet_encoder_0, 0))
        self.connect((self.analog_random_source_x_0, 0),
(self.blocks_throttle_0, 0))
        self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_constellation_modulator_0, 0))

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    frecuencia = self.carrier_freq
    if self.potencia < -10:
        self.potencia = -10
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = get_ajuste(self, 1, 1, 2, 3)
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

```

```

actualizar_label(self, self.parent, self.potencia)

def get_ook(self):
    return self.ook

def set_ook(self, ook):
    self.ook = ook

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)

```

```

        elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        else:
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
            self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
            self.set_potencia(self.potencia)
            set_corr(self)
            actualizar_label(self, self.parent, self.potencia)

```

Listado de programa OOK_UDP.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: OOK con fuente de Streaming
# Generated: Mon May 11 11:47:46 2015
#####

from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import wx

class OOK_UDP(grc_wxgui.top_block_gui):
    def __init__(self,onda,frec,pot,param1,param2,param3,parent):
        grc_wxgui.top_block_gui.__init__(self,
title="OOK con fuente de Streaming")

```

```

_icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.sps = sps = 2
self.bitrate = param3
self.samp_rate = samp_rate = (self.bitrate / 1) * self.sps * 1000
self.potencia = potencia = float(int(pot))
self.ook = ook = digital.constellation_rect(
([0 + 0j, 1 + 0j]), ([0, 1]), 2, 1, 2, 1, 1).base()
self.if_gain = if_gain = 20
self.corr = corr = -8
self.carrier_freq = carrier_freq = freq * 1e6

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label="Frecuencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
)

```

```

        proportion=1,
    )
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    minimum=-10,
    maximum=5,
    num_steps=15,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_potencia_sizer)
self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
self.osmosdr_sink_0.set_sample_rate(samp_rate)
self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
self.osmosdr_sink_0.set_freq_corr(corr, 0)
self.osmosdr_sink_0.set_gain(10, 0)
self.osmosdr_sink_0.set_if_gain(if_gain, 0)
self.osmosdr_sink_0.set_bb_gain(20, 0)
self.osmosdr_sink_0.set_antenna("", 0)
self.osmosdr_sink_0.set_bandwidth(0, 0)

self.set_potencia(potencia)
set_corr(self)
actualizar_label(self, self.parent, self.potencia)

```

```

        self.digital_constellation_modulator_0 = digital.generic_mod(
            constellation=ook,
            differential=True,
            samples_per_symbol=sps,
            pre_diff_code=True,
            excess_bw=param1,
            verbose=False,
            log=False,
        )
        self.blocks_udp_source_0 =
blocks.udp_source(gr.sizeof_char * 1, onda, param2, 1472, True)
        self.blocks_throttle_0 =
blocks.throttle(gr.sizeof_char*1, samp_rate)
        self.blks2_packet_encoder_0 =
grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
            samples_per_symbol=sps,
            bits_per_symbol=1,
            preamble="",
            access_code="",
            pad_for_usrp=True,
),
            payload_length=0,
        )

#####
# Connections
#####
self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_constellation_modulator_0, 0))
    self.connect((self.blocks_udp_source_0, 0),
(self.blocks_throttle_0, 0))
    self.connect((self.blocks_throttle_0, 0),
(self.blks2_packet_encoder_0, 0))
    self.connect((self.digital_constellation_modulator_0, 0),
(self.osmosdr_sink_0, 0))

#
# QT sink close method reimplementation
def get_sps(self):

```

```

        return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    if self.potencia < -10:
        self.potencia = -10
    frecuencia = self.carrier_freq
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = get_ajuste(self, 1, 1, 2, 3)
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
    actualizar_label(self, self.parent, self.potencia)

def get_ook(self):
    return self.ook

def set_ook(self, ook):
    self.ook = ook

def get_if_gain(self):
    return self.if_gain

```

```

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self._if_gain_slider.set_value(self.if_gain)
    self._if_gain_text_box.set_value(self.if_gain)
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)

```

```
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)
```

Listado de programa OOK_Video.py

```
#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: OOK con fuente de Video
# Generated: Mon May 11 11:47:48 2015
#####

from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import wx

class OOK_video(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self,
        title="OOK con fuente de Video")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.sps = sps = 2
```

```

        self.bitrate = param3
        self.samp_rate = samp_rate =
        (self.bitrate / 1) * self.sps * 1000
        self.potencia = potencia = float(int(pot))
        self.ook = ook = digital.constellation_rect(
        ([0 + 0j, 1 + 0j]), ([0, 1]), 2, 1, 2, 1, 1).base()
        self.if_gain = if_gain = 20
        self.corr = corr = -8
        self.carrier_freq = carrier_freq = freq * 1e6

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label="Frecuencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,

```

```

        value=self.potencia,
        callback=self.set_potencia,
        label="Potencia",
        converter=forms.float_converter(),
        proportion=0,
    )
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    minimum=-10,
    maximum=5,
    num_steps=15,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_potencia_sizer)
self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
self.osmosdr_sink_0.set_sample_rate(samp_rate)
self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
self.osmosdr_sink_0.set_freq_corr(corr, 0)
self.osmosdr_sink_0.set_gain(10, 0)
self.osmosdr_sink_0.set_if_gain(if_gain, 0)
self.osmosdr_sink_0.set_bb_gain(20, 0)
self.osmosdr_sink_0.set_antenna("", 0)
self.osmosdr_sink_0.set_bandwidth(0, 0)

self.set_potencia(potencia)
set_corr(self)
actualizar_label(self, self.parent, self.potencia)

self.digital_constellation_modulator_0 = digital.generic_mod(
    constellation=ook,
    differential=True,
    samples_per_symbol=sps,
    pre_diff_code=True,
    excess_bw=param1,
    verbose=False,
)

```

```

        log=False,
    )
    self.blocks_throttle_0 =
blocks.throttle(gr.sizeof_char*1, samp_rate)
    self.blocks_file_source_0 =
blocks.file_source(gr.sizeof_char * 1, onda, True)
    self.blks2_packet_encoder_0 =
grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
        samples_per_symbol=sps,
        bits_per_symbol=1,
        preamble="",
        access_code="",
        pad_for_usrp=True,
),
        payload_length=0,
)

#####
# Connections
#####
self.connect((self.digital_constellation_modulator_0, 0),
(self.osmosdr_sink_0, 0))
    self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_constellation_modulator_0, 0))
    self.connect((self.blocks_file_source_0, 0),
(self.blocks_throttle_0, 0))
    self.connect((self.blocks_throttle_0, 0),
(self.blks2_packet_encoder_0, 0))

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

```

```

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    if self.potencia < -10:
        self.potencia = -10
    frecuencia = self.carrier_freq
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
        elif frecuencia >= 2750e6:
            if self.potencia >= -5:
                self.potencia = -5
            self._potencia_slider.set_value(self.potencia)
            self._potencia_text_box.set_value(self.potencia)
            self.if_gain = get_ajuste(self, 1, 1, 2, 3)
            self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
            actualizar_label(self, self.parent, self.potencia)

def get_ook(self):
    return self.ook

def set_ook(self, ook):
    self.ook = ook

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

```

```

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

```

Listado de programa PSK_Audio.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: PSK con fuente de Audio

```

```

# Generated: Sat May  9 21:03:56 2015
#####
from gnuradio import audio
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import math
import time
import wx

class PSK_Audio(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self,
                                         title="PSK con fuente de Audio")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.sps = sps = int(math.log(param2, 2))
self.samp_rate = samp_rate = 1e6
self.potencia = potencia = float(int(pot))
self.const = const = param2
self.if_gain = if_gain = 20
self.corr = corr = -8
self.carrier_freq = carrier_freq = freq * 1e6

#####

```

```

# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label="Frecuencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
)

```

```

        minimum=-10,
        maximum=5,
        num_steps=15,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_potencia_sizer)
    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(corr, 0)
    self.osmosdr_sink_0.set_gain(10, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)
    self.osmosdr_sink_0.set_bb_gain(20, 0)
    self.osmosdr_sink_0.set_antenna("", 0)
    self.osmosdr_sink_0.set_bandwidth(0, 0)

    self.set_potencia(potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

    self.digital_psk_mod_0 = digital.psk.psk_mod(
        constellation_points=const,
        mod_code="gray",
        differential=True,
        samples_per_symbol=sps,
        excess_bw=param1,
        verbose=False,
        log=False,
    )
    self.blks2_packet_encoder_0 =
grc_blks2.packet_mod_f(grc_blks2.packet_encoder(
        samples_per_symbol=sps,
        bits_per_symbol=2,
        preamble="",
        access_code="",
        pad_for_usrp=True,
),
    payload_length=0,

```

```

        )
self.audio_source_0 = audio.source(48000, "", True)

#####
# Connections
#####
self.connect((self.digital_psk_mod_0, 0),
(self.osmosdr_sink_0, 0))
    self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_psk_mod_0, 0))
    self.connect((self.audio_source_0, 0),
(self.blks2_packet_encoder_0, 0))

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    if self.potencia < -10:
        self.potencia = -10
    frecuencia = self.carrier_freq
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0

```

```

        elif frecuencia >= 2750e6:
            if self.potencia >= -5:
                self.potencia = -5
            self._potencia_slider.set_value(self.potencia)
            self._potencia_text_box.set_value(self.potencia)
            self.if_gain = self.ajuste_dsb()
            self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
            actualizar_label(self, self.parent, self.potencia)

    def get_number_constellation(self):
        return self.number_constellation

    def set_number_constellation(self, number_constellation):
        self.number_constellation = number_constellation

    def get_if_gain(self):
        return self.if_gain

    def set_if_gain(self, if_gain):
        self.if_gain = if_gain
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

    def get_corr(self):
        return self.corr

    def set_corr(self, corr):
        self.corr = corr
        self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

    def get_carrier_freq(self):
        return self.carrier_freq

    def set_carrier_freq(self, carrier_freq):
        self.carrier_freq = carrier_freq
        if self.carrier_freq < 50e6:
            self.carrier_freq = 50e6
        elif self.carrier_freq > 3e9:
            self.carrier_freq = 3e9
        if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)

```

```

        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

def ajuste_dsb(self):
    if self.const == 2:
        self.if_gain = get_ajuste(self, 0, 0, 1, 1)
    elif self.const == 4:
        self.if_gain = get_ajuste(self, 0, 0, 1, 0)
    else:
        self.if_gain = get_ajuste(self, 0, 0, 0, 0)
    return self.if_gain

```

Listado de programa PSK_Random.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: PSK Random Source
# Generated: Mon May  4 14:30:25 2015
#####

from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr

```

```

from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from grc_gnuradio import blks2 as grc_blks2
from optparse import OptionParser
import numpy
import osmosdr
import math
import time
import wx

from Funciones import *

class PSK_Random(grc_wxgui.top_block_gui):
    def __init__(self, onda, frec, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self, title="PSK Random Source")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

        #####
        # Variables
        #####
        self.parent = parent
        self.bitrate = param3
        self.sps = sps = int(math.log(param2, 2))
        self.samp_rate = samp_rate = self.bitrate * 1000
        self.potencia = potencia = float(int(pot))
        self.if_gain = if_gain = 20
        self.const = const = param2
        self.carrier_freq = carrier_freq = frec * 1e6
        self.bt = bt = param1
        self.corr = corr = -8

        #####
        # Blocks
        #####
        _carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)

```

```
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label="Frecuencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    minimum=-10,
    maximum=5,
    num_steps=15,
```

```

        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_potencia_sizer)
    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(corr, 0)
    self.osmosdr_sink_0.set_gain(7, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)
    self.osmosdr_sink_0.set_bb_gain(20, 0)
    self.osmosdr_sink_0.set_antenna("", 0)
    self.osmosdr_sink_0.set_bandwidth(0, 0)

    self.set_potencia(potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

    self.digital_psk_mod_0 = digital.psk.psk_mod(
        constellation_points=const,
        mod_code="gray",
        differential=True,
        samples_per_symbol=sps,
        excess_bw=bt,
        verbose=False,
        log=False,
    )
    self.blocks_throttle_0 = blocks.throttle(gr.sizeof_char * 1, samp_rate)
    self.blks2_packet_encoder_0 =
        grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
            samples_per_symbol=sps,
            bits_per_symbol=sps,
            preamble="",
            access_code="",
            pad_for_usrp=True,
        ),
        payload_length=0,
    )
    self.analog_random_source_x_0 =

```

```

blocks.vector_source_b(map(int, numpy.random.randint(0, 256, 1000)), True)

#####
# Connections
#####
self.connect((self.digital_psk_mod_0, 0),
(self.osmosdr_sink_0, 0))
    self.connect((self.blocks_throttle_0, 0),
(self.blks2_packet_encoder_0, 0))
        self.connect((self.analog_random_source_x_0, 0),
(self.blocks_throttle_0, 0))
            self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_psk_mod_0, 0))

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_rrc_taps(self):
    return self_rrc_taps

def set_rrc_taps(self, rrc_taps):
    self_rrc_taps = rrc_taps

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))

```

```

        if self.potencia < -10:
            self.potencia = -10
        frecuencia = self.carrier_freq
        if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
            if self.potencia >= 0:
                self.potencia = 0
        elif frecuencia >= 2750e6:
            if self.potencia >= -5:
                self.potencia = -5
        self._potencia_slider.set_value(self.potencia)
        self._potencia_text_box.set_value(self.potencia)
        self.if_gain = self.ajuste_dsb()
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
        actualizar_label(self, self.parent, self.potencia)

    def get_if_gain(self):
        return self.if_gain

    def set_if_gain(self, if_gain):
        self.if_gain = if_gain
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

    def get_const(self):
        return self.const

    def set_const(self, const):
        self.const = const

    def get_carrier_freq(self):
        return self.carrier_freq

    def set_carrier_freq(self, carrier_freq):
        self.carrier_freq = carrier_freq
        if self.carrier_freq < 50e6:
            self.carrier_freq = 50e6
        elif self.carrier_freq > 3e9:
            self.carrier_freq = 3e9
        if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)

```

```

        elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        else:
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
            self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
            self.set_potencia(self.potencia)
            set_corr(self)
            actualizar_label(self, self.parent, self.potencia)

    def ajuste_dsb(self):
        if self.const == 2:
            self.if_gain = get_ajuste(self, 0, 0, 1, 1)
        elif self.const == 4:
            self.if_gain = get_ajuste(self, 0, 0, 1, 0)
        else:
            self.if_gain = get_ajuste(self, 0, 0, 0, 0)
        return self.if_gain

    def get_bt(self):
        return self.bt

    def set_bt(self, bt):
        self.bt = bt

```

Listado de programa PSK_UDP.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: PSK con fuente de Streaming
# Generated: Tue May 12 09:37:10 2015
#####

```

```

from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import math
import time
import wx

class PSK_UDP(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self,
                                         title="PSK con fuente de Streaming")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

        #####
        # Variables
        #####
        self.parent = parent
        self.bitrate = param3
        self.sps = sps = int(math.log(param2, 2))
        self.samp_rate = samp_rate = self.bitrate * 1000
        self.potencia = potencia = float(int(pot))
        self.const = const = param2
        self.if_gain = if_gain = 20
        self.corr = corr = -8
        self.carrier_freq = carrier_freq = freq * 1e6

        #####
        # Blocks

```

```

#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label="Frecuencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    minimum=-10,
)

```

```

        maximum=5,
        num_steps=15,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_potencia_sizer)
    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(corr, 0)
    self.osmosdr_sink_0.set_gain(10, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)
    self.osmosdr_sink_0.set_bb_gain(20, 0)
    self.osmosdr_sink_0.set_antenna("", 0)
    self.osmosdr_sink_0.set_bandwidth(0, 0)

    self.set_potencia(potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

    self.digital_psk_mod_0 = digital.psk.psk_mod(
        constellation_points=const,
        mod_code="gray",
        differential=True,
        samples_per_symbol=sps,
        excess_bw=param1,
        verbose=False,
        log=False,
    )
    self.blocks_udp_source_0 =
blocks.udp_source(gr.sizeof_char * 1, onda, param2, 1472, True)
    self.blocks_throttle_0 =
blocks.throttle(gr.sizeof_char * 1, samp_rate)
    self.blks2_packet_encoder_0 =
grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
        samples_per_symbol=sps,
        bits_per_symbol=sps,
        preamble="",
        access_code="",

```

```

        pad_for_usrp=True,
),
        payload_length=0,
)

#####
# Connections
#####
self.connect((self.digital_psk_mod_0, 0),
(self.osmosdr_sink_0, 0))
    self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_psk_mod_0, 0))
    self.connect((self.blocks_throttle_0, 0),
(self.blks2_packet_encoder_0, 0))
    self.connect((self.blocks_udp_source_0, 0),
(self.blocks_throttle_0, 0))

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    if self.potencia < -10:
        self.potencia = -10

```

```

frecuencia = self.carrier_freq
if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
    if self.potencia >= 0:
        self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = self.ajuste_dsb()
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
    actualizar_label(self, self.parent, self.potencia)

def get_number_constellation(self):
    return self.number_constellation

def set_number_constellation(self, number_constellation):
    self.number_constellation = number_constellation

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6

```

```

        elif self.carrier_freq > 3e9:
            self.carrier_freq = 3e9
        if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        else:
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
            self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
            self.set_potencia(self.potencia)
            set_corr(self)
            actualizar_label(self, self.parent, self.potencia)

    def ajuste_dsb(self):
        if self.const == 2:
            self.if_gain = get_ajuste(self, 0, 0, 1, 1)
        elif self.const == 4:
            self.if_gain = get_ajuste(self, 0, 0, 1, 0)
        else:
            self.if_gain = get_ajuste(self, 0, 0, 0, 0)
        return self.if_gain

```

Listado de programa PSK_Video.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: PSK con fuente de Video
# Generated: Tue May 12 09:37:17 2015
#####

```

```

from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import math
import time
import wx

class PSK_video(grc_wxgui.top_block_gui):
    def __init__(self, onda, frec, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self, title="PSK con fuente de Video")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

        #####
        # Variables
        #####
        self.parent = parent
        self.bitrate = param3
        self.sps = sps = int(math.log(param2, 2))
        self.samp_rate = samp_rate = self.bitrate * 1000
        self.potencia = potencia = float(int(pot))
        self.const = const = param2
        self.if_gain = if_gain = 20
        self.corr = corr = -8
        self.carrier_freq = carrier_freq = frec * 1e6

        #####
        # Blocks
        #####
        _carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)

```

```
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label="Frecuencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    minimum=-10,
    maximum=5,
    num_steps=15,
```

```

        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_potencia_sizer)
    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(corr, 0)
    self.osmosdr_sink_0.set_gain(10, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)
    self.osmosdr_sink_0.set_bb_gain(20, 0)
    self.osmosdr_sink_0.set_antenna("", 0)
    self.osmosdr_sink_0.set_bandwidth(0, 0)

    self.set_potencia(potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

    self.digital_psk_mod_0 = digital.psk.psk_mod(
        constellation_points=const,
        mod_code="gray",
        differential=True,
        samples_per_symbol=sps,
        excess_bw=param1,
        verbose=False,
        log=False,
    )
    self.blocks_file_source_0 =
blocks.file_source(gr.sizeof_char * 1, onda, True)
    self.blocks_throttle_0 =
blocks.throttle(gr.sizeof_char*1, samp_rate)
    self.blks2_packet_encoder_0 =
grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
        samples_per_symbol=sps,
        bits_per_symbol=sps,
        preamble="",
        access_code="",
        pad_for_usrp=True,
),

```

```

        payload_length=0,
    )

#####
# Connections
#####
self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_psk_mod_0, 0))
    self.connect((self.digital_psk_mod_0, 0),
(self.osmosdr_sink_0, 0))
        self.connect((self.blocks_file_source_0, 0),
(self.blocks_throttle_0, 0))
            self.connect((self.blocks_throttle_0, 0),
(self.blks2_packet_encoder_0, 0))

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    if self.potencia < -10:
        self.potencia = -10
frecuencia = self.carrier_freq
if (frecuencia >= 50e6) and (frecuencia <= 2150e6):

```

```

        if self.potencia >= 0:
            self.potencia = 0
        elif frecuencia >= 2750e6:
            if self.potencia >= -5:
                self.potencia = -5
            self._potencia_slider.set_value(self.potencia)
            self._potencia_text_box.set_value(self.potencia)
            self.if_gain = self.ajuste_dsb()
            self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
            actualizar_label(self, self.parent, self.potencia)

def get_number_constellation(self):
    return self.number_constellation

def set_number_constellation(self, number_constellation):
    self.number_constellation = number_constellation

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9

```

```

        if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        else:
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
            self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
            self.set_potencia(self.potencia)
            set_corr(self)
            actualizar_label(self, self.parent, self.potencia)

    def ajuste_dsb(self):
        if self.const == 2:
            self.if_gain = get_ajuste(self, 0, 0, 1, 1)
        elif self.const == 4:
            self.if_gain = get_ajuste(self, 0, 0, 1, 0)
        else:
            self.if_gain = get_ajuste(self, 0, 0, 0, 0)
        return self.if_gain

```

Listado de programa QAM_Audio.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: QAM con fuente de Audio
# Generated: Sat May  9 20:08:19 2015
#####

from gnuradio import audio
from gnuradio import digital

```

```

from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import wx
import math

class QAM_Audio(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self,
                                         title="QAM con fuente de Audio")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

        #####
        # Variables
        #####
        self.parent = parent
        self.sps = sps = int(math.log(param2, 2))
        self.samp_rate = samp_rate = 1e6
        self.potencia = potencia = float(int(pot))
        self.number_constellation = number_constellation = param2
        self.if_gain = if_gain = 20
        self.corr = corr = -8
        self.carrier_freq = carrier_freq = freq * 1e6

        #####
        # Blocks
        #####
        _carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
        self._carrier_freq_text_box = forms.text_box(
            parent=self.GetWin(),
            sizer=_carrier_freq_sizer,

```

```
        value=self.carrier_freq,
        callback=self.set_carrier_freq,
        label="Frecuencia",
        converter=forms.float_converter(),
        proportion=0,
    )
    self._carrier_freq_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_carrier_freq_sizer,
        value=self.carrier_freq,
        callback=self.set_carrier_freq,
        minimum=50e6,
        maximum=3e9,
        num_steps=295,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_carrier_freq_sizer)
    _potencia_sizer = wx.BoxSizer(wx.VERTICAL)
    self._potencia_text_box = forms.text_box(
        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        label="Potencia",
        converter=forms.float_converter(),
        proportion=0,
    )
    self._potencia_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        minimum=-10,
        maximum=5,
        num_steps=15,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
```

```

)
self.Add(_potencia_sizer)

self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
self.osmosdr_sink_0.set_sample_rate(samp_rate)
self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
self.osmosdr_sink_0.set_freq_corr(corr, 0)
self.osmosdr_sink_0.set_gain(10, 0)
self.osmosdr_sink_0.set_if_gain(if_gain, 0)
self.osmosdr_sink_0.set_bb_gain(20, 0)
self.osmosdr_sink_0.set_antenna("", 0)
self.osmosdr_sink_0.set_bandwidth(0, 0)

self.set_potencia(potencia)
set_corr(self)
actualizar_label(self, self.parent, self.potencia)

self.digital_qam_mod_0 = digital.qam.qam_mod(
    constellation_points=number_constellation,
    mod_code="gray",
    differential=True,
    samples_per_symbol=sps,
    excess_bw=param1,
    verbose=False,
    log=False,
)
self.blks2_packet_encoder_0 =
grc_blks2.packet_mod_f(grc_blks2.packet_encoder(
    samples_per_symbol=sps,
    bits_per_symbol=math.log(param2, 2),
    preamble="",
    access_code="",
    pad_for_usrp=True,
),
    payload_length=0,
)
self.audio_source_0 = audio.source(48000, "", True)

#####
# Connections
#####

```

```

        self.connect((self.audio_source_0, 0),
(self.blks2_packet_encoder_0, 0))
        self.connect((self.digital_qam_mod_0, 0),
(self.osmosdr_sink_0, 0))
        self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_qam_mod_0, 0))

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    if self.potencia < -10:
        self.potencia = -10
    frecuencia = self.carrier_freq
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = get_ajuste(self, 1, 0, 0, 0)

```

```

        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
        actualizar_label(self, self.parent, self.potencia)

def get_number_constellation(self):
    return self.number_constellation

def set_number_constellation(self, number_constellation):
    self.number_constellation = number_constellation

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)

```

```

        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

```

Listado de programa QAM_Random.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: QAM fuente Aleatoria
# Generated: Sat May  9 20:08:17 2015
#####

from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from grc_gnuradio import blks2 as grc_blks2
from optparse import OptionParser
from Funciones import *
import numpy
import osmosdr
import math
import time
import wx

class QAM_Random(grc_wxgui.top_block_gui):

```

```

def __init__(self, onda, frec, pot, param1, param2, param3, parent):
    grc_wxgui.top_block_gui.__init__(self, title="QAM fuente Aleatoria")
    _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
    self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.sps = sps = int(math.log(param2, 2))
self.bitrate = param3
self.samp_rate = samp_rate = self.bitrate * 1000
self.potencia = potencia = float(int(pot))
self.number_constellation = number_constellation = param2
self.if_gain = if_gain = 20
self.corr = corr = -8
self.carrier_freq = carrier_freq = frec * 1e6

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label="Frecuencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
)

```

```

        cast=float,
        proportion=1,
    )
    self.Add(_carrier_freq_sizer)
    _potencia_sizer = wx.BoxSizer(wx.VERTICAL)
    self._potencia_text_box = forms.text_box(
        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        label="Potencia",
        converter=forms.float_converter(),
        proportion=0,
    )
    self._potencia_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        minimum=-10,
        maximum=5,
        num_steps=15,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_potencia_sizer)
    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(corr, 0)
    self.osmosdr_sink_0.set_gain(10, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)
    self.osmosdr_sink_0.set_bb_gain(20, 0)
    self.osmosdr_sink_0.set_antenna("", 0)
    self.osmosdr_sink_0.set_bandwidth(0, 0)

    self.set_potencia(potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

```

```

        self.digital_qam_mod_0 = digital.qam.qam_mod(
            constellation_points=number_constellation,
            mod_code="gray",
            differential=True,
            samples_per_symbol=sps,
            excess_bw=param1,
            verbose=False,
            log=False,
        )
        self.blocks_throttle_0 =
blocks.throttle(gr.sizeof_char * 1, samp_rate)
        self.blks2_packet_encoder_0 =
grc.blks2.packet_mod_b(grc.blks2.packet_encoder(
            samples_per_symbol=sps,
            bits_per_symbol=sps,
            preamble="",
            access_code="",
            pad_for_usrp=True,
),
            payload_length=0,
)
        self.analog_random_source_x_0 =
blocks.vector_source_b(map(int, numpy.random.randint(0, 256, 1000)), True)

#####
# Connections
#####
        self.connect((self.blocks_throttle_0, 0),
(self.blks2_packet_encoder_0, 0))
        self.connect((self.analog_random_source_x_0, 0),
(self.blocks_throttle_0, 0))
        self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_qam_mod_0, 0))
        self.connect((self.digital_qam_mod_0, 0),
(self.osmosdr_sink_0, 0))

# QT sink close method reimplementation

```

```

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    if self.potencia < -10:
        self.potencia = -10
    frecuencia = self.carrier_freq
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = get_ajuste(self, 1, 0, 0, 0)
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
    actualizar_label(self, self.parent, self.potencia)

def get_number_constellation(self):
    return self.number_constellation

def set_number_constellation(self, number_constellation):
    self.number_constellation = number_constellation

def get_if_gain(self):

```

```

        return self.if_gain

    def set_if_gain(self, if_gain):
        self.if_gain = if_gain
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

    def get_corr(self):
        return self.corr

    def set_corr(self, corr):
        self.corr = corr
        self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

    def get_carrier_freq(self):
        return self.carrier_freq

    def set_carrier_freq(self, carrier_freq):
        self.carrier_freq = carrier_freq
        if self.carrier_freq < 50e6:
            self.carrier_freq = 50e6
        elif self.carrier_freq > 3e9:
            self.carrier_freq = 3e9
        if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
        else:
            self._carrier_freq_slider.set_value(self.carrier_freq)
            self._carrier_freq_text_box.set_value(self.carrier_freq)
            self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
            self.set_potencia(self.potencia)
            set_corr(self)

```

```
actualizar_label(self, self.parent, self.potencia)
```

Listado de programa QAM_UDP.py

```
#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: QAM con fuente de Streaming
# Generated: Tue May 12 10:53:18 2015
#####

from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import wx
import math

class QAM_UDP(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self,
title="QAM con fuente de Streaming")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

#####
# Variables
#####
self.parent = parent
self.bitrate = param3
```

```

self.sps = sps = int(math.log(param2, 2))
self.samp_rate = samp_rate = self.bitrate * 1000
self.potencia = potencia = float(int(pot))
self.number_constellation = number_constellation = param2
self.if_gain = if_gain = 20
self.corr = corr = -8
self.carrier_freq = carrier_freq = freq * 1e6

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label="Frecuencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
)

```

```

        label="Potencia",
        converter=forms.float_converter(),
        proportion=0,
    )
    self._potencia_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        minimum=-10,
        maximum=5,
        num_steps=15,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_potencia_sizer)
    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(corr, 0)
    self.osmosdr_sink_0.set_gain(10, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)
    self.osmosdr_sink_0.set_bb_gain(20, 0)
    self.osmosdr_sink_0.set_antenna("", 0)
    self.osmosdr_sink_0.set_bandwidth(0, 0)

    self.set_potencia(potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

    self.digital_qam_mod_0 = digital.qam.qam_mod(
        constellation_points=number_constellation,
        mod_code="gray",
        differential=True,
        samples_per_symbol=sps,
        excess_bw=param1,
        verbose=False,
        log=False,
    )

```

```

        self.blocks_udp_source_0 =
blocks.udp_source(gr.sizeof_char * 1, onda, param2, 1472, True)
        self.blocks_throttle_0 = blocks.throttle(gr.sizeof_char*1, samp_rate)
        self.blks2_packet_encoder_0 =
grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
            samples_per_symbol=sps,
            bits_per_symbol=math.log(param2, 2),
            preamble="",
            access_code="",
            pad_for_usrp=True,
),
            payload_length=0,
)

#####
# Connections
#####
self.connect((self.blks2_packet_encoder_0, 0), (self.digital_qam_mod_0, 0))
self.connect((self.digital_qam_mod_0, 0), (self.osmosdr_sink_0, 0))
self.connect((self.blocks_udp_source_0, 0), (self.blocks_throttle_0, 0))
self.connect((self.blocks_throttle_0, 0), (self.blks2_packet_encoder_0, 0))

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

```

```

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    if self.potencia < -10:
        self.potencia = -10
    frecuencia = self.carrier_freq
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:
            self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = get_ajuste(self, 1, 0, 0, 0)
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
    actualizar_label(self, self.parent, self.potencia)

def get_number_constellation(self):
    return self.number_constellation

def set_number_constellation(self, number_constellation):
    self.number_constellation = number_constellation

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

```

```

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

```

Listado de programa QAM_Video.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: QAM con fuente de Video
# Generated: Tue May 12 10:53:20 2015
#####

from gnuradio import blocks
from gnuradio import digital
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option

```

```

from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import blks2 as grc_blks2
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import wx
import math


class QAM_video(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self,
                                        title="QAM con fuente de Video")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

        #####
        # Variables
        #####
        self.parent = parent
        self.bitrate = param3
        self.sps = sps = int(math.log(param2, 2))
        self.samp_rate = samp_rate = self.bitrate * 1000
        self.potencia = potencia = float(int(pot))
        self.number_constellation = number_constellation = param2
        self.if_gain = if_gain = 20
        self.corr = corr = -8
        self.carrier_freq = carrier_freq = freq * 1e6

        #####
        # Blocks
        #####
        _carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
        self._carrier_freq_text_box = forms.text_box(
            parent=self.GetWin(),
            sizer=_carrier_freq_sizer,
            value=self.carrier_freq,

```

```
        callback=self.set_carrier_freq,
        label="Frecuencia",
        converter=forms.float_converter(),
        proportion=0,
    )
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    label="Potencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._potencia_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_potencia_sizer,
    value=self.potencia,
    callback=self.set_potencia,
    minimum=-10,
    maximum=5,
    num_steps=15,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
```

```

        self.Add(_potencia_sizer)
        self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
        self.osmosdr_sink_0.set_sample_rate(samp_rate)
        self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
        self.osmosdr_sink_0.set_freq_corr(corr, 0)
        self.osmosdr_sink_0.set_gain(10, 0)
        self.osmosdr_sink_0.set_if_gain(if_gain, 0)
        self.osmosdr_sink_0.set_bb_gain(20, 0)
        self.osmosdr_sink_0.set_antenna("", 0)
        self.osmosdr_sink_0.set_bandwidth(0, 0)

        self.set_potencia(potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

        self.digital_qam_mod_0 = digital.qam.qam_mod(
            constellation_points=number_constellation,
            mod_code="gray",
            differential=True,
            samples_per_symbol=sps,
            excess_bw=param1,
            verbose=False,
            log=False,
        )
        self.blocks_file_source_0 =
blocks.file_source(gr.sizeof_char * 1, onda, True)
        self.blocks_throttle_0 =
blocks.throttle(gr.sizeof_char*1, samp_rate)
        self.blks2_packet_encoder_0 =
grc_blks2.packet_mod_b(grc_blks2.packet_encoder(
            samples_per_symbol=sps,
            bits_per_symbol=math.log(param2, 2),
            preamble="",
            access_code="",
            pad_for_usrp=True,
),
            payload_length=0,
)

```

#####

```

# Connections
#####
self.connect((self.digital_qam_mod_0, 0),
(self.osmosdr_sink_0, 0))
    self.connect((self.blks2_packet_encoder_0, 0),
(self.digital_qam_mod_0, 0))
    self.connect((self.blocks_file_source_0, 0),
(self.blocks_throttle_0, 0))
        self.connect((self.blocks_throttle_0, 0),
(self.blks2_packet_encoder_0, 0))

# QT sink close method reimplementation

def get_sps(self):
    return self.sps

def set_sps(self, sps):
    self.sps = sps

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    if self.potencia < -10:
        self.potencia = -10
    frecuencia = self.carrier_freq
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:

```

```

        self.potencia = -5
self._potencia_slider.set_value(self.potencia)
self._potencia_text_box.set_value(self.potencia)
self.if_gain = get_ajuste(self, 1, 0, 0, 0)
self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
actualizar_label(self, self.parent, self.potencia)

def get_number_constellation(self):
    return self.number_constellation

def set_number_constellation(self, number_constellation):
    self.number_constellation = number_constellation

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):

```

```

        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

```

Listado de programa WBFM_Audio.py

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-
#####
# Gnuradio Python Flow Graph
# Title: WBFM Audio
# Generated: Sat May  9 18:16:51 2015
#####

from gnuradio import analog
from gnuradio import audio
from gnuradio import eng_notation
from gnuradio import filter
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
import osmosdr
import time
import wx

```

```

class WBFM_Audio(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self, title="WBFM Audio")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

        #####
        # Variables
        #####
        self.parent = parent
        self.samp_rate = samp_rate = 200e3
        self.max_freq_dev = max_freq_dev = param2
        self.if_gain = if_gain = 21
        self.corr = corr = -8
        self.carrier_freq = carrier_freq = freq * 1e6
        self.potencia = potencia = pot

        #####
        # Blocks
        #####
        _carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
        self._carrier_freq_text_box = forms.text_box(
            parent=self.GetWin(),
            sizer=_carrier_freq_sizer,
            value=self.carrier_freq,
            callback=self.set_carrier_freq,
            label="Frecuencia",
            converter=forms.float_converter(),
            proportion=0,
        )
        self._carrier_freq_slider = forms.slider(
            parent=self.GetWin(),
            sizer=_carrier_freq_sizer,
            value=self.carrier_freq,
            callback=self.set_carrier_freq,
            minimum=50e6,
            maximum=3e9,
            num_steps=295,
        )

```

```

        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_carrier_freq_sizer)
    _potencia_sizer = wx.BoxSizer(wx.VERTICAL)
    self._potencia_text_box = forms.text_box(
        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        label="Potencia",
        converter=forms.float_converter(),
        proportion=0,
    )
    self._potencia_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        minimum=-10,
        maximum=5,
        num_steps=15,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_potencia_sizer)
    self.rational_resampler_xxx_0_0 = filter.rational_resampler_ccc(
        interpolation=200000,
        decimation=480000,
        taps=None,
        fractional_bw=None,
    )
    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(corr, 0)
    self.osmosdr_sink_0.set_gain(7, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)

```

```

        self.osmosdr_sink_0.set_bb_gain(20, 0)
        self.osmosdr_sink_0.set_antenna("", 0)
        self.osmosdr_sink_0.set_bandwidth(0, 0)

        self.set_potencia(potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

        self.low_pass_filter_0 = filter.fir_filter_fff(1, firdes.low_pass(
            5, 48000, 22000, 10000, firdes.WIN_HAMMING, 6.76))
        self.audio_source_0 = audio.source(48000, "", True)
        self.analog_wfm_tx_0 = analog.wfm_tx(
            audio_rate=48000,
            quad_rate=480000,
            tau=75e-6,
            max_dev=max_freq_dev,
        )

#####
# Connections
#####
self.connect((self.analog_wfm_tx_0, 0),
(self.rational_resampler_xxx_0_0, 0))
    self.connect((self.rational_resampler_xxx_0_0, 0),
(self.osmosdr_sink_0, 0))
        self.connect((self.low_pass_filter_0, 0),
(self.analog_wfm_tx_0, 0))
        self.connect((self.audio_source_0, 0),
(self.low_pass_filter_0, 0))

# QT sink close method reimplementation

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_max_freq_dev(self):

```

```

        return self.max_freq_dev

    def set_max_freq_dev(self, max_freq_dev):
        self.max_freq_dev = max_freq_dev

    def get_if_gain(self):
        return self.if_gain

    def set_if_gain(self, if_gain):
        self.if_gain = if_gain
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

    def get_corr(self):
        return self.corr

    def set_corr(self, corr):
        self.corr = corr
        self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

    def get_potencia(self):
        return self.potencia

    def set_potencia(self, potencia):
        self.potencia = float(int(potencia))
        if self.potencia < -10:
            self.potencia = -10
        frecuencia = self.carrier_freq
        if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
            if self.potencia >= 0:
                self.potencia = 0
        elif frecuencia >= 2750e6:
            if self.potencia >= -5:
                self.potencia = -5
        self._potencia_slider.set_value(self.potencia)
        self._potencia_text_box.set_value(self.potencia)
        self.if_gain = get_ajuste(self)
        self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

    def get_carrier_freq(self):
        return self.carrier_freq

```

```

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)

```

Listado de programa WBFM_vector.py

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-

#####
# Gnuradio Python Flow Graph
# Title: WBFM con senal arbitraria
# Generated: Tue May 12 19:20:23 2015
#####

from gnuradio import analog
from gnuradio import blocks
from gnuradio import eng_notation

```

```

from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
from wfm_tx import *
import osmosdr
import time
import wx

class WBFM_vector(grc_wxgui.top_block_gui):
    def __init__(self,onda,frec,pot,param1,param2,param3, parent):
        grc_wxgui.top_block_gui.__init__(self,
        title="WBFM con señal arbitraria")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

        #####
        # Variables
        #####
        self.parent = parent
        self.vector = vector = onda
        self.samp_rate = samp_rate = 2e6
        self.potencia = potencia = pot
        self.mod_freq = mod_freq = param1
        self.max_freq_dev = max_freq_dev = param2
        self.if_gain = if_gain = 20
        self.corr = corr = -8
        self.carrier_freq = carrier_freq = frec * 1e6

        #####
        # Blocks
        #####
        _carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
        self._carrier_freq_text_box = forms.text_box(
            parent=self.GetWin(),
            sizer=_carrier_freq_sizer,

```

```

        value=self.carrier_freq,
        callback=self.set_carrier_freq,
        label="Frecuencia",
        converter=forms.float_converter(),
        proportion=0,
    )
    self._carrier_freq_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_carrier_freq_sizer,
        value=self.carrier_freq,
        callback=self.set_carrier_freq,
        minimum=50e6,
        maximum=3e9,
        num_steps=295,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_carrier_freq_sizer)
    _potencia_sizer = wx.BoxSizer(wx.VERTICAL)
    self._potencia_text_box = forms.text_box(
        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        label="Potencia",
        converter=forms.float_converter(),
        proportion=0,
    )
    self._potencia_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        minimum=-10,
        maximum=5,
        num_steps=15,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,

```

```

        )

    self.Add(_potencia_sizer)

    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(corr, 0)
    self.osmosdr_sink_0.set_gain(7, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)
    self.osmosdr_sink_0.set_bb_gain(20, 0)
    self.osmosdr_sink_0.set_antenna("", 0)
    self.osmosdr_sink_0.set_bandwidth(0, 0)

    self.set_potencia(potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

    self.blocks_vector_source_x_0 =
blocks.vector_source_f(vector, True, 1, [])
    self.blocks_multiply_const_vxx_0 =
blocks.multiply_const_vff((1.0 / max(abs(i) for i in vector)), )
    self.analog_wfm_tx_0 = wfm_tx(
        audio_rate=2000000,
        quad_rate=2000000,
        tau=75e-6,
        max_dev=max_freq_dev,
    )

#####
# Connections
#####
    self.connect((self.blocks_vector_source_x_0, 0),
(self.blocks_multiply_const_vxx_0, 0))
    self.connect((self.blocks_multiply_const_vxx_0, 0),
(self.analog_wfm_tx_0, 0))
    self.connect((self.analog_wfm_tx_0, 0),
(self.osmosdr_sink_0, 0))

# QT sink close method reimplementation

```

```

def get_vector(self):
    return self.vector

def set_vector(self, vector):
    self.vector = vector
    self.blocks_multiply_const_vxx_0.set_k(
((1.0 / max(abs(i) for i in self.vector)),))

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    if self.potencia < -10:
        self.potencia = -10
    frecuencia = self.carrier_freq
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
        elif frecuencia >= 2750e6:
            if self.potencia >= -5:
                self.potencia = -5
    self._potencia_slider.set_value(self.potencia)
    self._potencia_text_box.set_value(self.potencia)
    self.if_gain = get_ajuste(self)
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)
    actualizar_label(self, self.parent, self.potencia)

def get_mod_freq(self):
    return self.mod_freq

def set_mod_freq(self, mod_freq):
    self.mod_freq = mod_freq

```

```

def get_max_freq_dev(self):
    return self.max_freq_dev

def set_max_freq_dev(self, max_freq_dev):
    self.max_freq_dev = max_freq_dev

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6
    elif self.carrier_freq > 3e9:
        self.carrier_freq = 3e9
    if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):

```

```

        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
    else:
        self._carrier_freq_slider.set_value(self.carrier_freq)
        self._carrier_freq_text_box.set_value(self.carrier_freq)
        self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)
        self.set_potencia(self.potencia)
        set_corr(self)
        actualizar_label(self, self.parent, self.potencia)

```

Listado de programa WBFM.py

```

#!/usr/bin/env python
#####
# Gnuradio Python Flow Graph
# Title: WBFM
# Generated: Sat May  9 18:16:49 2015
#####

from gnuradio import analog
from gnuradio import eng_notation
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.filter import firdes
from gnuradio.wxgui import forms
from grc_gnuradio import wxgui as grc_wxgui
from optparse import OptionParser
from Funciones import *
from wfm_tx import *
import osmosdr
import time
import wx

class WBFM(grc_wxgui.top_block_gui):
    def __init__(self, onda, freq, pot, param1, param2, param3, parent):
        grc_wxgui.top_block_gui.__init__(self, title="WBFM")
        _icon_path = "/usr/share/icons/hicolor/32x32/apps/gnuradio-grc.png"
        self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))

```

```

#####
# Variables
#####
self.parent = parent
self.samp_rate = samp_rate = int(2e6)
self.potencia = potencia = float(int(pot))
self.mod_freq = mod_freq = param1
self.max_freq_dev = max_freq_dev = param2
self.if_gain = if_gain = 20
self.corr = corr = -8
self.carrier_freq = carrier_freq = freq * 1e6

#####
# Blocks
#####
_carrier_freq_sizer = wx.BoxSizer(wx.VERTICAL)
self._carrier_freq_text_box = forms.text_box(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    label="Frecuencia",
    converter=forms.float_converter(),
    proportion=0,
)
self._carrier_freq_slider = forms.slider(
    parent=self.GetWin(),
    sizer=_carrier_freq_sizer,
    value=self.carrier_freq,
    callback=self.set_carrier_freq,
    minimum=50e6,
    maximum=3e9,
    num_steps=295,
    style=wx.SL_HORIZONTAL,
    cast=float,
    proportion=1,
)
self.Add(_carrier_freq_sizer)
_potencia_sizer = wx.BoxSizer(wx.VERTICAL)
self._potencia_text_box = forms.text_box(

```

```

        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        label="Potencia",
        converter=forms.float_converter(),
        proportion=0,
    )
    self._potencia_slider = forms.slider(
        parent=self.GetWin(),
        sizer=_potencia_sizer,
        value=self.potencia,
        callback=self.set_potencia,
        minimum=-10,
        maximum=5,
        num_steps=15,
        style=wx.SL_HORIZONTAL,
        cast=float,
        proportion=1,
    )
    self.Add(_potencia_sizer)
    self.osmosdr_sink_0 = osmosdr.sink(args="numchan=" + str(1) + " " + "")
    self.osmosdr_sink_0.set_sample_rate(samp_rate)
    self.osmosdr_sink_0.set_center_freq(carrier_freq, 0)
    self.osmosdr_sink_0.set_freq_corr(corr, 0)
    self.osmosdr_sink_0.set_gain(7, 0)
    self.osmosdr_sink_0.set_if_gain(if_gain, 0)
    self.osmosdr_sink_0.set_bb_gain(20, 0)
    self.osmosdr_sink_0.set_antenna("", 0)
    self.osmosdr_sink_0.set_bandwidth(0, 0)

    self.set_potencia(potencia)
    set_corr(self)
    actualizar_label(self, self.parent, self.potencia)

'''Original
self.analog_wfm_tx_0 = analog.wfm_tx(
    audio_rate=int(500e3),
    quad_rate=samp_rate,
    tau=75e-6,
```

```

        max_dev=max_freq_dev,
    )'''
    self.analog_wfm_tx_0 = wfm_tx(
        audio_rate=int(2000e3),
        quad_rate=samp_rate,
        tau=75e-6,
        max_dev=max_freq_dev,
    )
    self.analog_sig_source_x_0 = analog.sig_source_f(2000e3, onda, mod_freq, 1, 0)

#####
# Connections
#####
self.connect((self.analog_sig_source_x_0, 0),
(self.analog_wfm_tx_0, 0))
    self.connect((self.analog_wfm_tx_0, 0),
(self.osmosdr_sink_0, 0))

# QT sink close method reimplementation

def get_samp_rate(self):
    return self.samp_rate

def set_samp_rate(self, samp_rate):
    self.samp_rate = samp_rate
    self.osmosdr_sink_0.set_sample_rate(self.samp_rate)

def get_potencia(self):
    return self.potencia

def set_potencia(self, potencia):
    self.potencia = float(int(potencia))
    if self.potencia < -10:
        self.potencia = -10
    frecuencia = self.carrier_freq
    if (frecuencia >= 50e6) and (frecuencia <= 2150e6):
        if self.potencia >= 0:
            self.potencia = 0
    elif frecuencia >= 2750e6:
        if self.potencia >= -5:

```

```

        self.potencia = -5
self._potencia_slider.set_value(self.potencia)
self._potencia_text_box.set_value(self.potencia)
self.if_gain = get_ajuste(self)
self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_mod_freq(self):
    return self.mod_freq

def set_mod_freq(self, mod_freq):
    self.mod_freq = mod_freq
    self.analog_sig_source_x_0.set_frequency(self.mod_freq)

def get_max_freq_dev(self):
    return self.max_freq_dev

def set_max_freq_dev(self, max_freq_dev):
    self.max_freq_dev = max_freq_dev

def get_if_gain(self):
    return self.if_gain

def set_if_gain(self, if_gain):
    self.if_gain = if_gain
    self.osmosdr_sink_0.set_if_gain(self.if_gain, 0)

def get_corr(self):
    return self.corr

def set_corr(self, corr):
    self.corr = corr
    self.osmosdr_sink_0.set_freq_corr(self.corr, 0)

def get_carrier_freq(self):
    return self.carrier_freq

def set_carrier_freq(self, carrier_freq):
    self.carrier_freq = carrier_freq
    if self.carrier_freq < 50e6:
        self.carrier_freq = 50e6

```

```
elif self.carrier_freq > 3e9:  
    self.carrier_freq = 3e9  
if (self.carrier_freq >= 2149e6) and (self.carrier_freq <= 2157e6):  
    self._carrier_freq_slider.set_value(self.carrier_freq)  
    self._carrier_freq_text_box.set_value(self.carrier_freq)  
elif (self.carrier_freq >= 2749e6) and (self.carrier_freq <= 2760e6):  
    self._carrier_freq_slider.set_value(self.carrier_freq)  
    self._carrier_freq_text_box.set_value(self.carrier_freq)  
elif (self.carrier_freq >= 916e6) and (self.carrier_freq <= 928e6):  
    self._carrier_freq_slider.set_value(self.carrier_freq)  
    self._carrier_freq_text_box.set_value(self.carrier_freq)  
elif (self.carrier_freq >= 2306e6) and (self.carrier_freq <= 2318e6):  
    self._carrier_freq_slider.set_value(self.carrier_freq)  
    self._carrier_freq_text_box.set_value(self.carrier_freq)  
else:  
    self._carrier_freq_slider.set_value(self.carrier_freq)  
    self._carrier_freq_text_box.set_value(self.carrier_freq)  
    self.osmosdr_sink_0.set_center_freq(self.carrier_freq, 0)  
    self.set_potencia(self.potencia)  
    set_corr(self)  
    actualizar_label(self, self.parent, self.potencia)
```
