Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA

PC y TI

UNIVERSIDAD AUTÓNOMA METROPOLITANA – IZTAPALAPA
DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

DISEÑO DEL NIVEL FÍSICO DE UN RADIO OFDM PARA COMUNICACIÓN DIGITAL CON UNA PLATAFORMA SOFTWARE DEFINED RADIO (SDR)

Idónea Comunicación de Resultados presentada por
Amado Gutiérrez Gómez
Para obtener el grado de
Maestro en Ciencias y Tecnologías de la Información

Asesor:
Dr. Gerardo Abel Laguna Sánchez

Dr. Victor Rangel Licea

Dr. Fausto Casco Sánchez

Dr. Gerardo Abel Laguna Sánchez

Defendida públicamente en la UAM Iztapalapa

Presidente: Dr. Victor Rangel Licea, Universidad Nacional Autónoma de México
Secretario: Dr. Fausto Casco Sánchez, UAM – Iztapalapa, Redes y Telecomunicaciones
Vocal: Dr. Gerardo Abel Laguna Sánchez, UAM – Iztapalapa, Redes y Telecomunicaciones

México, D.F. Enero 2014

Resumen

En las comunicaciones inalámbricas modernas, se emplean esquemas digitales y, para minimizar los efectos de atenuación, el ruido y, al mismo tiempo, maximizar el aprovechamiento del canal, se emplean avanzadas técnicas de procesamiento de señales y comunicaciones digitales, tales como el multiplexado por repartición en frecuencias ortogonales OFDM (siglas del ingl. *Orthogonal Frequency Division Multiplexing*) que ha sido adoptada como técnica de señalización básica para tecnologías como WIMAX (siglas del ingl. *Worldwide Interoperability for Microwave Access*) y LTE (siglas del ingl. *Long Term Evolution*).

El esquema OFDM es de amplia aplicación en las comunicaciones digitales modernas, tanto en esquemas alambrados (por ejemplo, en sistemas ADLS) como en esquemas inalámbricos (por ejemplo, en sistemas de telefonía celular), ya que ofrece ventajas como: Adaptibilidad de la velocidad de transmisión en función de la SNR, eficiencia espectral, robustez ante interferencia entre símbolos (ISI), inmunidad inherente al ruido impulsivo.

En el presente documento se presenta la comunicación idónea de resultados, el diseño, verificación y validación en una plataforma SDR (siglas del ingl. *Software Defined Radio*) de los bloques funcionales de nivel físico de un radio OFDM para la transmisión de datos inalámbrica. La Motivación del presente trabajo se deriva de un, proyecto PLC (siglas del ingl. *Power Line Communication*), debido a la necesidad de probar los conceptos en una plataforma SDR y determinar el alcance del proyecto en un sistema inalámbrico. Los algoritmos propuestos, en este trabajo, se toman del modelo propuesto en [1] y tienen un enfoque práctico a través de la plataforma SDR.

Durante la investigación nos encontramos con muy poca información sobre la plataforma SDR, y descubrimos que no hay una metodología clara que nos ayude a iniciar, con el proceso de migrar nuestros algoritmos a una plataforma SDR. Por esta razón, tradicionalmente el tiempo de ciclo de diseño y desarrollo de sistemas de comunicación digital, ha sido muy largo.

Después de haber realizado una investigación a fondo del estado del arte sobre la platafor-

ma SDR, las ventajas y desventajas del software y hardware SDR, decidimos utilizar la siguiente plataforma: Software SDR (*GNU Radio*) y hardware SDR (*USRP 1*), la cual cumple con nuestros requerimientos. A través de esta plataforma, pusimos en práctica nuestros algoritmos de nivel físico de radio OFDM. El diseño del algoritmo de cada bloque fue verificado por medio de la programación de cada algoritmo en el software SDR GNU Radio. Particularmente, aquí presentamos nuestra propuesta para las siguientes funciones, indispensables en los actuales radios digitales OFDM: Generación de datos, modulación en frecuencia (QAM), modulación OFDM (IFFT), prefijo cíclico y sincronización [2].

Durante la puesta en práctica, validamos nuestros algoritmos con el hardware SDR (USRP1). Por otra parte, reportamos la metodología para verificar y validar nuestros algoritmos en la plataforma SDR. 1) Desde la instalación del software, 2) el funcionamiento del USRP1, 3) la elaboración de nuestros propios módulos usando los lenguajes de programación C++ y Python, 4) la simulación de nuestros módulos en el entorno *GNU Radio Companion*. La implementación del transmisor OFDM[1], se valida utilizando el hardware USRP1 y observando el espectro de la señal con el receptor utilizando como analizador de espectro (proporcionado por GNU Radio). En la parte de la sincronización logramos obtener la sincronización en tiempo a partir de [2], utilizando la estimación ML y por medio de la correlación de prefijo cíclico.

Agradecimientos

La culminación de este trabajo se debe principalmente a mi madre, el apoyo incondicional que me has brindado es un factor muy importante. TE DEDICO CON TODO MI CORAZÓN ESTE TRABAJO MADRECITA MÍA.

Quiero agradecer a mi esposa, que en estos dos años de la maestría ha sido una pieza fundamental para poder concluir este trabajo, tu apoyo, tu comprensión, tu amor han sido la clave de superación, y a mi bebe que al verlo me da más fuerza para seguir adelante. TE DEDICO CON TODO MI CORAZÓN ESTE TRABAJO ESPOSA MÍA.

Quiero agradecer a mi padre, por sus enseñanzas, su fuerza de voluntad, su tenacidad, sus ganas de superarse, han sido la clave para no caér. Es mi ejemplo a seguir y a mi hermana por siempre valorarme y apoyarme incondicionalmente. LES DEDICO CON TODO MI CORAZÓN ESTE TRABAJO PAPA Y HERMANA.

Quiero agradecer al Dr. Alfonso Prieto Guerrero, que me apoyo como tutor, durante el inicio de la Maestría.

Quiero agradecer a mi asesor el Dr. Gerardo Abel Laguna Sánchez por apoyarme con sus conocimientos y todos los consejos recibidos durante la elaboración del presente trabajo de investigación. Por otra parte, la gran enseñanza que me deja es invaluable.

Quiero agradecer a la Universidad Autónoma Metropolitana que me ha formado profesionalmente y por todo el apoyo para la realización de la Maestría.

Quiero agradecer a todos los profesores como el Dr. Fausto Casco, Dr. Luis Rojas, Dr. Miguel López que compartieron sus conocimientos durante la Maestría, los cuales han sido muy importantes para la culminación de este trabajo.

Quiero agradecer a la UNAM y en especial al Dr. Victor Rangel, por el apoyo para la realización de este trabajo de investigación.

Contenido

Lista de Figuras vii

Lista de Tablas ix

1. Introducción **2**
1.1. Planteamiento del problema . . . 3
1.2. Objetivos del proyecto . . . 5
1.3. Alcance . . . 5
1.4. Metodología de investigación . . . 6
1.5. Contribución . . . 7
1.6. Estructura del documento . . . 7

2. Marco Teórico **8**
2.1. Antecedentes . . . 8
2.2. Plataforma SDR . . . 9
2.3. Plataforma de Hardware SDR . . . 10
2.3.1. Enfoque de una computadora de propósito general . . . 10
2.3.2. Enfoque de co-procesador . . . 11
2.3.3. Enfoque de procesador central . . . 12
2.3.4. Enfoque de unidades programables . . . 12
2.4. Hardware USRP1 de Ettus Research . . . 13
2.4.1. Tarjeta madre . . . 14
2.4.2. Tarjeta hija . . . 16
2.5. Software SDR . . . 18
2.5.1. GNU Radio . . . 19
2.5.2. Arquitectura GNU Radio . . . 19
2.6. OFDM . . . 23
2.6.1. Conceptos básicos . . . 23
2.6.2. Reseña histórica de la técnica OFDM . . . 25
2.6.3. Fundamentos teóricos de OFDM . . . 26

2.6.4. Modulación OFDM 26
2.6.5. Señal transmitida con OFDM 27
2.6.6. Intervalo de guarda 31
2.6.7. Símbolo piloto 32
2.7. Proceso de sincronización y estimación de canal 33
2.7.1. Sincronización 33
2.7.2. Estimación de canal 36

3. Programación y evaluación **38**
3.1. Plataforma SDR para el desarrollo del proyecto 39
3.2. Arquitectura del sistema de comunicaciones 40
3.2.1. Sistema de comunicaciones utilizando la arquitectura SDR 41
3.2.2. Diseño de los bloques del nivel físico de un radio OFDM en banda base, utilizando una arquitectura SDR 44
3.2.3. Transmisor en banda base 45
3.2.4. Frecuencia Intermedia Digital(tarjeta madre USRP1) 49
3.2.5. Conversión de Frecuencia Intermedia a Radio-frecuencia (tarjeta hija USRP1) 52
3.2.6. Antena del USRP1 57
3.2.7. Realización de un transmisor OFDM con una plataforma SDR 57
3.2.8. Realización de un receptor OFDM con una plataforma SDR 59

4. Pruebas **63**
4.0.9. Resultados de los bloques individuales en el Software SDR GNU Radio 64
4.0.10. Generación de datos 64
4.0.11. Modulación digital 4 QAM 65
4.0.12. Modulación OFDM 66
4.0.13. Adición del prefijo cíclico 67
4.0.14. Realización del radio OFDM de banda base mediante la interfaz gráfica GNU Radio Companion 68
4.0.15. Radio OFDM en radio frecuencia, utilizando un USRP1 y el bloque de portadora modulada del software GNU Radio 68
4.0.16. Sincronización burda (en tiempo) 73

5. Conclusiones y trabajo Futuro **75**
5.1. Conclusiones 75
5.2. Trabajo futuro 77

Referencias **78**

A. Glosario **81**

B. Instalación del software de GNU Radio **84**

C. Creación de bloques personalizados compatibles con el software de GNU Radio **88**

D. Algoritmos **90**

Lista de Figuras

1.1. Concepto ideal de Software Radio. 3

2.1. Radio digital genérico. 9
2.2. Diagrama a bloques de un sistema de radio. 9
2.3. USRP1 (siglas del ingl. *Universal Software Radio Peripheral*) de Ettus Research. 14
2.4. Diagrama a bloques de la tarjeta madre del USRP1. 15
2.5. Diagrama de la tarjeta hija transmisora RFX 900, TX. 16
2.6. Espectro de una arquitectura Zero IF. 17
2.7. Señal de RF a banda base. 17
2.8. Diagrama de la tarjeta hija receptora RFX 900, RX. 18
2.9. GNU Radio en el contexto de SDR. 19
2.10. Arquitectura GNU Radio. 20
2.11. GNU Radio Companion. 21
2.12. Enlace de los módulos personalizados a través de Python. 22
2.13. Esquema monoportadora (*single carrier*) y esquema multiportadora (*multi-carrier*) utilizando FDM. 23
2.14. Comparación entre ancho de banda requerido para FDM y OFDM. 24
2.15. Interferencia en sistema monoportadora y multiportadora [23]. 24
2.16. Modulación OFDM. 27
2.17. Subportadoras ortogonales y símbolo OFDM en el dominio de la frecuencia. 28
2.18. Subportadoras y símbolo OFDM en el dominio del tiempo. 29
2.19. Inserción de intervalo de guarda. 32
2.20. Prefijo cíclico para evitar interferencia entre símbolos. 33
2.21. Esquema OFDM asistido por piloto [23]. 34
2.22. Correlación $\Psi(\delta_t)$ [23]. 35

3.1. Flujo de datos en el enlace OFDM [1]. 41
3.2. Carpetas generadas por gr_modtool.py. 42
3.3. Flujo de datos en el enlace OFDM con la plataforma SDR. 43

3.4. Diseño propuesto del flujo de datos en el enlace OFDM en una arquitectura SDR. 45
3.5. Formato antipodal. 46
3.6. Constelación 4QAM. 48
3.7. Estructura interna del dominio digital del USRP1 en la parte transmisora. . 50
3.8. Chips AD 9862. 50
3.9. Tarjeta electrónica (motherboard del USRP1) correspondiente a la figura 3.7. 53
3.10. Flujo de datos a través del dominio digital USRP1 (DUC). 54
3.11. Diagrama de la tarjeta hija transmisora RFX 900, TX. 55
3.12. Tarjeta hija transmisora RFX 900, TX. 56
3.13. Antena VERT900 de 900 Mhz. 57
3.14. Transmisor OFDM realizado en una plataforma SDR. 58
3.15. Diagrama a bloques para la sincronización en tiempo [2]. 59

4.1. a) Señal antipodal de la secuencia 1, b) Señal antipodal de la secuencia 2. . . 64
4.2. a) Modulación 4QAM prueba 1, b) Modulación 4 QAM prueba 2. 65
4.3. a) Símbolo OFDM prueba 1, b) Símbolo OFDM prueba 2. 66
4.4. a) Símbolo OFDM con prefijo cíclico prueba 1, b) Símbolo OFDM con prefijo cíclico prueba 2. 67
4.5. Bloque de nivel físico de un radio OFDM en banda base en la plataforma SDR (GNU Radio Companion). 69
4.6. Bloques para la realización de nivel físico de un radio OFDM con portadora modulada en GNU Radio. 70
4.7. a) Espectro OFDM en el analizador de espectro utilizando software y hardware SDR, b) Banco de pruebas para la validación de los bloques de los algoritmos de nivel físico de un radio OFDM para sistema de comunicación digital en un canal de comunicación inalámbrica con una plataforma SDR. 71
4.8. Aplicación para la realización de la sincronización burda en tiempo. 73
4.9. Sincronización burda. 74

Lista de Tablas

2.1. Plataformas USRP. 13

4.1. Parámetros de banda base para el radio OFDM en GNU Radio. 63
4.2. Parámetros del enlace de comunicación inalámbrica a través del hardware USRP1. 68

Acrónimos

SR SOFTWARE RADIO

SDR SOFTWARE DEFINED RADIO

OFDM ORTHOGONAL FREQUENCY DIVISON MULTIPLEXING

SNR SIGNAL TO NOISE RATIO

PLC POWER LINE COMMUNICATION

FPGA FIELD PROGRAMABLE GATE ARRAY

DSP DIGITAL SIGNAL PROCESSOR

USB UNIVERSAL SERIAL BUS

USRP UNIVERSAL SOFTWARE RADIO PERIPHERAL

GNU GENERAL PUBLIC LICENSE RADIO

PCIe PERIPHERAL COMPONENT INTERCONNECT EXPRESS

GPU GRAPHICS PROCESSING UNIT

VHDL VERILOG HARDWARE DESCRIPTION LANGUAGE

LTE LONG TERM EVOLUTION

ARM ADVANCED RISC MACHINE

MAC MEDIA ACCES CONTROL

UMTS UNIVERSAL MOBILE TELECOMMUNICATIONS SYSTEMS

DDC DIGITAL DOWN CONVERTER

CIC CASCADED INTEGRATED COMB

IF INTERMEDIATE FREQUENCY

DUC DIGITAL UPCONVERTER

ADC ANALOGIC DIGITAL CONVERTER

DAC DIGITAL ANALOGIC CONVERTER

RF RADIOFREQUENCY

LO OSCILATOR LLOCAL

LNA LOW NOISE AMPLIFIER

GPL GENERAL PUBLIC LICENSE

SWIG SIMPLIFIED WRAPPER AND INTERFACE GENERATOR

GRC GNU RADIO COMPANION

XML XTENSIBLE MARKUP LANGUAGE

FDM FREQUENCY DIVISION MULTIPLEXING

WIMAX WORLDWIDE INTEROPERABILITY FOR MICROWAVE ACCES

DAB DIGITAL AUDIO BROADCASTING

DVB DIGITAL VIDEO BROADCASTING

DSL DIGITAL SUBSCRIBER LOOP

ICI INTERCARRIER INTERFERENCE

FFT FAST FOURIER TRANSFORM

QAM QUADRATURE MODULATION AMPLITUDE

QPSK QUADRATURE PHASE SHIFT KEYING

IFFT INVERSE FAST FOURIER TRANSFORM

BPSK BINARY QUADRATURE PHASE SHIFT KEYING

IDFT INVERSE DISCRETE FOURIER TRANSFORM

DFT DISCRETE FOURIER TRANSFORM

ISI INTERSYMBOL INTERFERENCE

GI GUARD INTERVAL

LS LEAST SQUARE

ML MAXIMUM LIKELIHOOD

MMSE MINIMUN MEAN SQUARED ERROR

MC MAXIMUM CORRELATION

LLS LINEAR LEAST SQUARE

Capítulo 1
Introducción

De forma tradicional, un radio consistía en una antena, encargada de recibir y enviar información, y un hardware de propósito específico encargado de procesar esa señal de información, filtrarla, modificar su frecuencia, etc. Éste hardware de propósito específico no podía modificar de forma notable su funcionalidad. En la actualidad gran parte de esta funcionalidad se traslada a un dispositivo electrónico, comúnmente llamado FPGA, la utilización de éste, reduce considerablemente la utilización de hardware analógico solo utilizando los dispositivos front-end y los convertidores analógico-digital y digital-analógico como dispositivos electrónicos analógicos.

La gran contribución de la tecnología SDR, aparte de reducir la utilización de los dispositivos electrónicos analógicos, es que la funcionalidad viene definida por el diseño del software, de tal manera que la modificación o actualización es sencilla y no depende de la sustitución de ningún elemento de hardware.

La plataforma SDR hace posible la, transmisión y recepción enlace inalámbrico del radio este concepto va tomando fuerza dentro de la investigación y en la industria de las telecomunicaciones. Muchos investigadores en los campos de redes de banda ancha, no contemplan los efectos reales de la capa física en sus simulaciones, ahora tienen la posibilidad de crear su propia capa física en esta plataforma, para poder validar algoritmos de, capas superiores y proporcionar resultados más cercanos a la realidad.

En centros de investigación, universidades, industrias, sectores gubernamentales están adquiriendo equipos SDR, muchos para hacer investigación, otros para impartir las clases de sistemas de comunicaciones digitales, comunicaciones inalámbricas, etc. Por ejemplo, CFE (Comisión Federal Electricidad) adquirió equipos SDR para utilizarlos en su proyecto de red inteligente.

1.1. Planteamiento del problema

Los equipos de radio están diseñados internamente por circuitos electrónicos especiales; su funcionamiento hace posible la transmisión y recepción de la información a través del enlace inalámbrico de radio, por consiguiente se dice que el diseño de los radios son totalmente dependientes del hardware. A lo largo del tiempo se han utilizado dispositivos electrónicos cada vez más sofisticados, los cuales han mejorado el funcionamiento del radio. Este concepto cambia cuando Joseph Mitola, en 1991 [3], utiliza el nuevo concepto SR (siglas del ingl. *Software Radio*), donde *propone que los sistemas de radio no dependan exclusivamente del hardware.*

Dentro del concepto de SR, los componentes físicos de un sistema de radio, que han sido típicamente realizados en hardware (ejemplo: mezclador, filtro, amplificador, modulador/demodulador, detector, etc), son implementados por medio de un software en una computadora de propósito general como se muestra en la figura 1.1 [3].

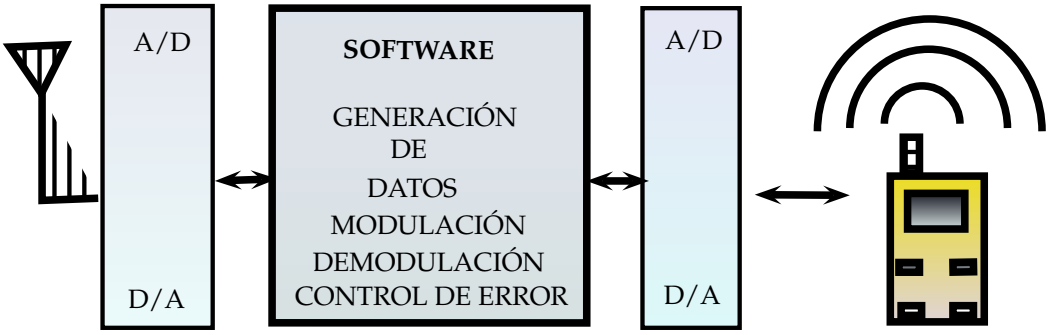


Figura 1.1: Concepto ideal de Software Radio.

Así, el trabajo presentado aquí permite un acercamiento práctico, por medio de la plataforma SDR, al diseño de sistemas de comunicación digital para canales selectivos en frecuencia y variantes en el tiempo, lo cual es de gran utilidad en aplicaciones reales, pues es el caso de los canales inalámbricos, especialmente en la comunicaciones con dispositivos móviles, cuyo comportamiento es precisamente selectivo en frecuencia debido al efecto de las múltiples trayectorias.

En esencia, un esquema OFDM (siglas del ingl. *Orthogonal Frequency Division Multiplexing*, "multiplexado por división de frecuencias ortogonales") es una forma de modulación multi-portadora que divide eficientemente un flujo entrante de datos seriales (de alta veloci-

dad) en un gran número de flujos paralelos de datos (a menor velocidad) y los asigna a un conjunto de subportadoras (subcanales) ortogonales.

Una de las ventajas más prometedora de esta técnica incluye la posibilidad de tener esquemas de modulación adaptivos donde, por ejemplo, se ajusta la velocidad de datos en función de la relación señal a ruido (SNR). Básicamente, para la determinación del esquema de modulación y la asignación de los bits en cada subcanal OFDM, se puede estimar la SNR de cada subcanal de tal manera que un subcanal con mejor SNR conducirá más bits que uno con mala SNR. Así, cada subcanal lleva tantos bits en cada símbolo OFDM como lo requiere el nivel de atenuación y el ruido observado en la frecuencia de su subcanal.

El punto de partida del presente trabajo se encuentra en un proyecto PLC (siglas del ingl. *Power Line Communication*). En este trabajo se reportan los resultados después de adaptar y probar los conceptos en una plataforma SDR y determinar el alcance del proyecto en un sistema inalámbrico.

Por otra parte, en la actualidad, la presión de los avances tecnológicos obligan a los desarrolladores a reducir el tiempo previsto para el ciclo de diseño y desarrollo. Por tal motivo, el proceso de diseño, verificación y validación de los algoritmos para el nivel físico de un radio OFDM pueden ser muy largos si no se sigue la metodología apropiada. Entonces, se requiere de una metodología para el diseño y verificación de los algoritmos de comunicación digital utilizando una plataforma SDR para acelerar la verificación y validación del diseño de los algoritmos del nivel físico OFDM.

1.2. Objetivos del proyecto

Objetivo general

Diseñar los algoritmos básicos de nivel físico de un radio OFDM usando tecnología SDR.

Objetivos particulares

- Desarrollar los bloques de modulación, demodulación y sincronización de un radio OFDM usando una plataforma SDR.
- Verificar y validar los bloques de comunicación digital diseñados en una plataforma SDR.
- Desarrollar la metodología para el diseño y verificación de los algoritmos de comunicación digital usando una plataforma SDR.

1.3. Alcance

En esta tesis, se presenta una plataforma en hardware y software SDR que permita validar y verificar algoritmos básicos de nivel físico de un radio OFDM, aplicado a un sistema de comunicación digital inalámbrico.

La propuesta contempla dos etapas, la etapa de hardware y software.

La primera etapa de hardware contempla los siguientes módulos:

- Módulo de procesamiento.
- Módulo de transmisión y recepción SDR.
- Módulo de radio frecuencia.

La segunda etapa de software contempla los siguientes módulos:

- Módulo de transmisión y recepción SDR
- Módulo de modulación y demodulación SDR.
- Módulo de sincronización burda.

El alcance real del sistema de comunicación en la plataforma SDR esta limitado por el tiempo y financiamiento. La sincronización fina, en un sistema real de comunicación inalámbrica, requiere más tiempo de investigación y práctica y se propone como la continuación de este proyecto de investigación.

El canal inalámbrico y sincronización del transmisor y receptor, influyen en el desempeño de la sincronización fina. Por tal motivo, si la sincronización fina, no funciona adecuadamente, la demodulación estará afectada.

1.4. Metodología de investigación

Para cumplir con los objetivos propuestos anteriormente, se siguió la metodología siguiente.

Estudio del estado del arte

- Estudiar los diferentes tipos de plataforma SDR.
- Selección del software y hardware SDR, que mejor satisface nuestros requerimientos.
- Investigar los algoritmos del nível físico de un radio OFDM.
- Investigar el funcionamiento y manejo adecuado del hardware SDR.
- Investigar los más relevante de la plataforma SDR seleccionada para el desarrollo del proyecto.

Programación de los algoritmos básicos de nivel físico de un radio OFDM

- Diseño del esquema de comunicación OFDM.
- Simulación por computadora del esquema de comunicación OFDM.
- Programación, en una plataforma SDR, de los algoritmos básicos del nivel físico de un radio OFDM.

Verificación y validación de los algoritmos básicos del nivel físico de un radio OFDM en una plataforma SDR.

1.5. Contribución

Después de seguir la metodología descrita anteriormente se tiene los siguientes aportes:

- Un panorama general de la plataforma SDR hasta el año 2013.
- Puesta en práctica de los algoritmos del nivel físico de un radio OFDM.
- La metodología para verificar algoritmos de nivel físico en el software SDR y la validación de estos en hardware SDR.

1.6. Estructura del documento

El presente trabajo se encuentra estructurado de 5 Capítulos. El primer Capítulo contiene la presente introducción.

En el Capítulo 2, se hace una revisión del estado actual de la plataforma SDR y algoritmos básicos que conforman el nivel físico de un radio OFDM.

En el Capítulo 3, se presenta la propuesta de este trabajo, en el cual se propone el software y hardware SDR para el sistema de comunicación digital.

En el Capítulo 4, se presentan los resultados de la verificación de los algoritmos nivel físico así como la validación de estos en la plataforma SDR.

Finalmente, en el Capítulo 5 se presentan las conclusiones del trabajo de investigación y desarrollo de este proyecto, considerando los objetivos propuestos.

Capítulo 2
Marco Teórico

En este capítulo se presenta una introducción a las características y descripción de las plataformas SDR. Por otra parte, se presenta un modelo formal para los algoritmos del nivel físico de un radio OFDM. En particular, nos concentraremos en los bloques de modulación, demodulación y sincronización, para un sistema de comunicación digital.

2.1. Antecedentes

Joseph Mitola, en 1991, introdujo el concepto SR (siglas del ingl. *Software Radio*) afirmando que: Los componentes físicos de un sistema de radio, que han sido típicamente diseñados en hardware (ejemplo: mezclador, filtro, amplificador, modulador/demodulador, detector, etc) pueden ser implementados por medio de un software en una computadora de próposito general [3].

Por otra parte, un sistema SDR es un sistema de radiocomunicación en donde sus componentes de capa física son implementados en una plataforma programable o reconfigurable [4].

Entonces, un sistema de radio digital en el contexto de SDR consiste de tres bloques funcionales, como se muestra en la figura 2.1 [5]. Sección de banda base, sección de IF (frecuencia intermedia) y sección de RF (radio frecuencia).

- Sección de banda base: Esta sección se encarga del procesamiento de la señal útil con relativa baja frecuencia.
- Sección de IF: Esta sección realiza el cambio de bajas frecuencias a altas frecuencias.
- Sección de RF: Esta sección se encarga de realizar la conversión digital-analógica ó analógica-digital de la señal.

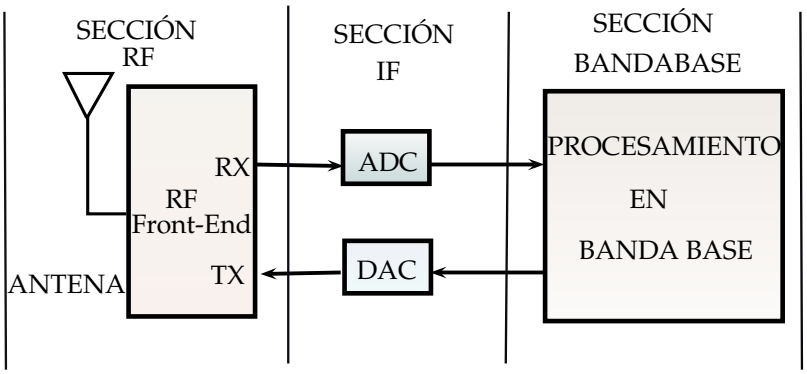


Figura 2.1: Radio digital genérico.

Durante el proceso de investigación, respecto de la frontera del conocimiento, se encontró el estudio reportado en [6], el cual se enfoca en las arquitecturas de las plataformas SDR existentes hasta la actualidad.

2.2. Plataforma SDR

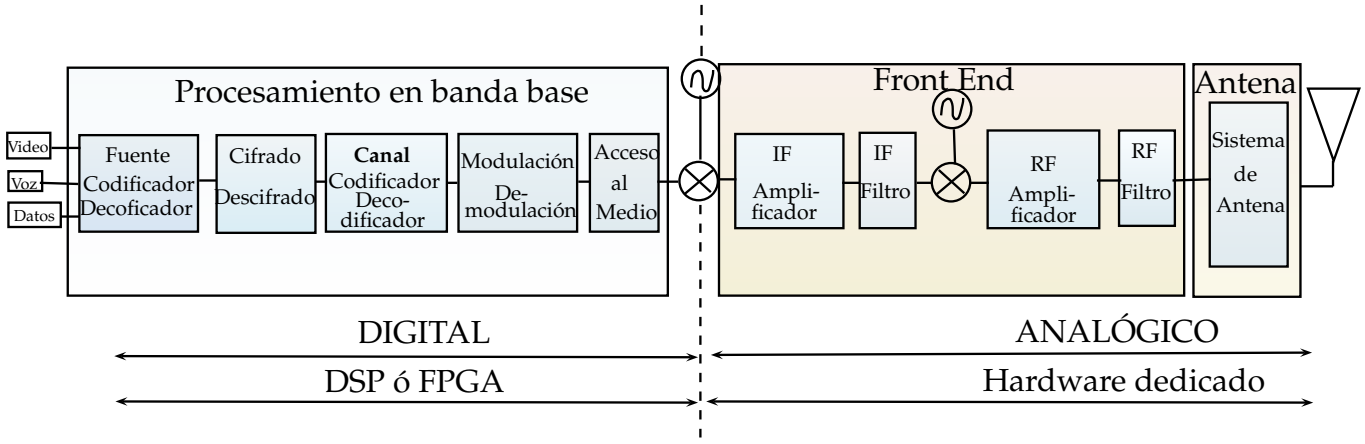


Figura 2.2: Diagrama a bloques de un sistema de radio.

En la figura 2.2 [6] se muestra con más detalle los diferentes componentes que conforman un sistema de radio.

El *bloque de procesamiento en banda base,* figura[2.2][6], es implementado por medio de *software* y los algoritmos de nivel físico de comunicación digital son cargados en un FPGA

(siglas del ingl. *Field Programable Gate Array*) o DSP (siglas del ingl. *Digital Signal Processor*). Los elementos del radio transmisor de un sistema de comunicaciones que conforman este bloque son los siguientes: codificación de la señal, codificación del canal y modulación. El radio receptor esta formado por los siguientes elementos: demodulación, decodificación de canal y decodificación de la señal.

El *bloque front end,* figura[2.2][6], es implementado en hardware analógico. En un SDR típico la parte analógica esta limitada al traslado de la señal de interés a frecuencias intermedias y, por último, a altas frecuencias para mandarlos al canal de comunicaciones inalámbrico.

El *bloque de Antena,* figura[2.2][6], realiza la transformación de los voltajes de la señal en onda electromagnéticas para ser enviadas al medio inalámbrico.

Al seleccionar una plataforma SDR, se considera la elección de una plataforma computacional para la parte digital(i.e. procesamiento en banda base), y una plataforma de hardware para la parte del *front end* (conversión digital-analógica, amplificador y antena).

2.3. Plataforma de Hardware SDR

La plataforma de hardware SDR tiene como función principal, trabajar a nivel de capa física (incluyendo el procesamiento de banda base y conversión de frecuencia intermedia), para la transmisión y recepción de datos [6].

En el artículo [6] se presentan 4 enfoques para la realización de las plataformas SDR:

- Enfoque de una computadora de propósito general
- Enfoque de co-procesador
- Enfoque de procesador central
- Enfoque de unidades configurables

2.3.1. Enfoque de una computadora de propósito general

Este enfoque consiste en utilizar el procesador de una computadora de propósito general, proporcionando un proceso fácil y flexible de programación, pero con un alto consumo de energía. Para ejemplo, tenemos:

- USRP (siglas del ingl. *Universal Software Radio Peripheral*) [7]. Este hardware es un dispositivo con el enfoque de una computadora de propósito general. Se compone de un convertidor digital-analógico y un convertidor analógico-digital de altas frecuencias, que muestrea la señal en frecuencia intermedia, un FPGA (siglas del ingl. *Field Programable Gate Array*) que realiza el procesamiento y almacenamiento de la señal en banda base. Por otra parte, el procesamiento de la señal es delegada a una computadora y la señal procesada es transmitida al FPGA através de un enlace USB (siglas del ingl. *Universal Serial Bus*)[7]. Además, existe otro dispositivo, llamado USRP2, con mayor capacidad en hardware que el USRP1. En este último, la señal procesada se transmite al FPGA através de un enlace ethernet [7]. Estos dispositivos trabajan con software GNU Radio (siglas del ingl. *General Public License Radio*) y son compatibles con productos de National Instruments, LabView y Mathworks Matlab [6][7].

- QUICKSILVER [8]. Esta plataforma es similar al USRP. Sin embargo, sólo es capaz de recibir señales de RF.

- Microsoft SORA [9]. Esta plataforma ha sido desarrollada por Microsoft. El dispositivo SORA es conectado a la computadora por medio de un bus PCIe (siglas del ingl. *Peripheral Component Interconnect Express*). Con esta plataforma se obtiene bajas latencias y un *throughput* alto y permite un amplio uso de las computadoras más modernas para el procesamiento de 802.11b/g en tiempo real. Teniendo en mente la Ley de Moore, uno podría pensar que las computadoras del futuro serían capaces de procesar todos los protocolos en tiempo real. Sin embargo, como se muestra en [10], el incremento del *throughput* de los datos exige un alto costo computacional. Esta plataforma utiliza un grado muy alto de paralelismo.

2.3.2. Enfoque de co-procesador

Este enfoque consiste en acelerar el procesamiento de la señal mediante un co-procesador y se trata de un refinamiento del enfoque de computadora de propósito general. En este enfoque se propone la adición de un co-procesador para la ejecución de grandes cantidades de procesamiento, así mismo, reduciendo consumo de energía y manteniendo una alta capacidad en cuanto a programación y flexibilidad. El trabajo presentado en [11] utiliza un GPU (siglas del ingl. *Graphics Processing Unit*) como co-procesador compatible con el GNU Radio y, como resultado de este trabajo, se obtienen ganancias de un factor de 3 a 4 en velocidad de procesamiento. Como ejemplo, tenemos:

- KUAR[12] (siglas del ingl. *Kansas University Agile Radio*). Esta plataforma utiliza una computadora encapsulada asociada a un FPGA, la implementación es completamente

en VHDL (siglas del ingl. *VHSIC Hardware Description Language*) y el procesador es de implementación propietaria. Otros dispositivos utilizan un DSP como procesador central.

- Imec ADRES (siglas del ingl. *Architecture for Dynamically Reconfigurable Embedded Systems*), desarrollado por Imec, es una arquitectura reconfigurable, que alcanza un alto desempeño a través de diferentes clases de paralelismo. Consiste de un procesador VLIW (siglas de ingl. *Very Long Instruction Word*) y un arreglo reconfigurable. El procesador es programado utilizando un compilador en C y su uso está dirigido al área de telecomunicaciones con punto de referencia en 802.11n, con velocidades de hasta 108 Mbps y LTE (siglas del ingl. *Long Term Evolution*) con velocidades de hasta 18 Mbps y un consumo promedio de 333 mW [6].

Estas arquitecturas ofrecen tareas de paralelismo limitadas con lo que se reduce su eficiencia [6].

2.3.3. Enfoque de procesador central

Este enfoque se apoya en el uso de procesadores dedicados para el procesamiento de señales (DSP). Esta plataforma tiene una alta capacidad de programación, pero la flexibilidad de la plataforma es reducida. El procesador más utilizado es un ARM (siglas del ingl. *Advanced RISC Machine*). Como ejemplo, tenemos:

- NXP EVP16, plataforma presentada en el 2005, esta conformada por varias unidades funcionales, las que se describen a continuación. Un Procesador ARM proporciona el control de la capa de enlace y MAC (siglas del ingl. *Media Acces Control*), mientras que un DSP convencional, procesador vectorial y aceleradores de hardware son utlizados para el procesamiento de la señal. El punto de referencia, es la ejecución de UMTS (siglas del ingl. *Universal Mobile Telecommunications Systems*) [13].

2.3.4. Enfoque de unidades programables

Este enfoque posibilita un bajo consumo de energía. Algunas plataformas sustituyen el DSP por unidades configurables. Para ejemplo, tenemos:

- Fujitsu SDR. Se introdujo en 2005 y utiliza aceleradores de hardware asociados a procesadores reconfigurables. Todos estos componentes están conectados a una línea de red de datos y controlado por un procesador central ARM. Este dispositivo trabaja con el estándar 802.11 a/b con un *throughput* de 43 Mbps [6].

- Imec BEAR[4]. Es un refinamiento de la plataforma ADRES de Imec. Está formado por un procesador ARM para el control y tres ASIP's (siglas del ingl. *Application-specific Instruction-set Processor*), para sincronización fina de los *front ends* son utilizados para procesamiento de banda base con un acelerador Viterbi. Esta plataforma puede ser programada en lenguaje C o Matlab. En términos de energía, BEAR permite tener 2x2 MIMO (siglas del ingl. *Multiple-input Multiple-output*) OFDM a 128 Mbps con tan solo 231 mW [6].
- CEA Magali (siglas del ingl. *Communications Enabled Application*). El chip SDR Magali desarrollado como una plataforma de demostración para telecomunicaciones, permite una recepción LTE 4x2 MIMO con un consumo de 236 mW [14].

2.4. Hardware USRP1 de Ettus Research

Actualmente se tienen disponibles las tarjetas realizadas por Matt Ettus, las cuales son: USRP1, USRP N200 y USRP N210, todas compatibles con GNU Radio o Simulink. En el año del 2010, la empresa NI (siglas del ingl. *National Instruments*) adquiere *Ettus Research*. A partir, de entonces se han realizado 5 nuevas tarjetas, el NI USRP 2920, 2921, 2922, 2930, 2932, todas compatibles con Labview. En la tabla 2.1 se muestra las diferentes plataformas USRP existentes.

Tabla 2.1: Plataformas USRP.

Modelo	Frecuencia	Software	Puerto
USRP1	Tarjeta hija (750-1050 Mhz)	GNU Radio, Simulink	usb
USRP N200	Tarjeta hija (2.4-5.9 Ghz)	GNU Radio, Simulink	ethernet
USRP N210	Tarjeta hija (2.4-5.9 Ghz)	GNU Radio, Simulink	ethernet
NI USRP-2920	50 Mhz-2.2 Ghz	Labview	ethernet
NI USRP-2921	2.4-2.5 y 4.9-5.9 Ghz	Labview	ethernet
NI USRP-2922	400 Mhz-4.4 Ghz	Labview	ethernet
NI USRP-2930	50 Mhz-2.2 Ghz	Labview	ethernet
NI USRP-2932	400 Mhz-4.4 Ghz	Labview	ethernet

Los parámetros que se toman en cuenta para elegir el hardware SDR a utilizar son los siguientes:

- Si no es una limitación el consumo de energía, el enfoque más apropiado es el que se apoya en una computadora de propósito general.

- Si el interes es minimizar el consumo de energía u optimizar el desempeño computacional, el enfoque más apropiado son las plataformas de hardware dedicado.

El trabajo aquí presentado esta enfocado a la verificación y validación de algoritmos de nivel físico. Sólo nos interesa validar los algoritmos en una plataforma SDR y no es nuestro propósito hacer estudios de consumo de energía ni mucho menos evaluar el desempeño computacional. Por otra parte, debido a las limitaciones presupuestales, es importante que nuestra plataforma sea de bajo costo. Por este motivo nos inclinamos por la plataforma USRP1 de Ettus Research. Una vista del equipo se muestra en la figura 2.3 [15].

Figura 2.3: USRP1 (siglas del ingl. *Universal Software Radio Peripheral*) de Ettus Research.

La plataforma USRP1 es una plataforma flexible de bajo costo para el desarrollo de SDR's y se compone de dos tarjetas principales: 1) tarjeta madre y 2) tarjeta hija. A continuación se da una breve descripción de cada una:

2.4.1. Tarjeta madre

En esta parte del hardware se implementa la sección de procesamiento de la señal en banda base como se muestra en la figura 2.2.

Está constituida de un FPGA, donde los esquemas de modulación son almacenados en su memoria o en sus elementos lógicos y la actualización de los bloques funcionales se puede realizar solamente cuando es necesario. La combinación de SDR y FPGA's proporciona a los ingenieros y desarrolladores un método de actualización de los sistemas de comunicaciones de difícil acceso, como por ejemplo, los satélites [16].

A continuación se describe el hardware específico que contiene la tarjeta madre del USRP1.

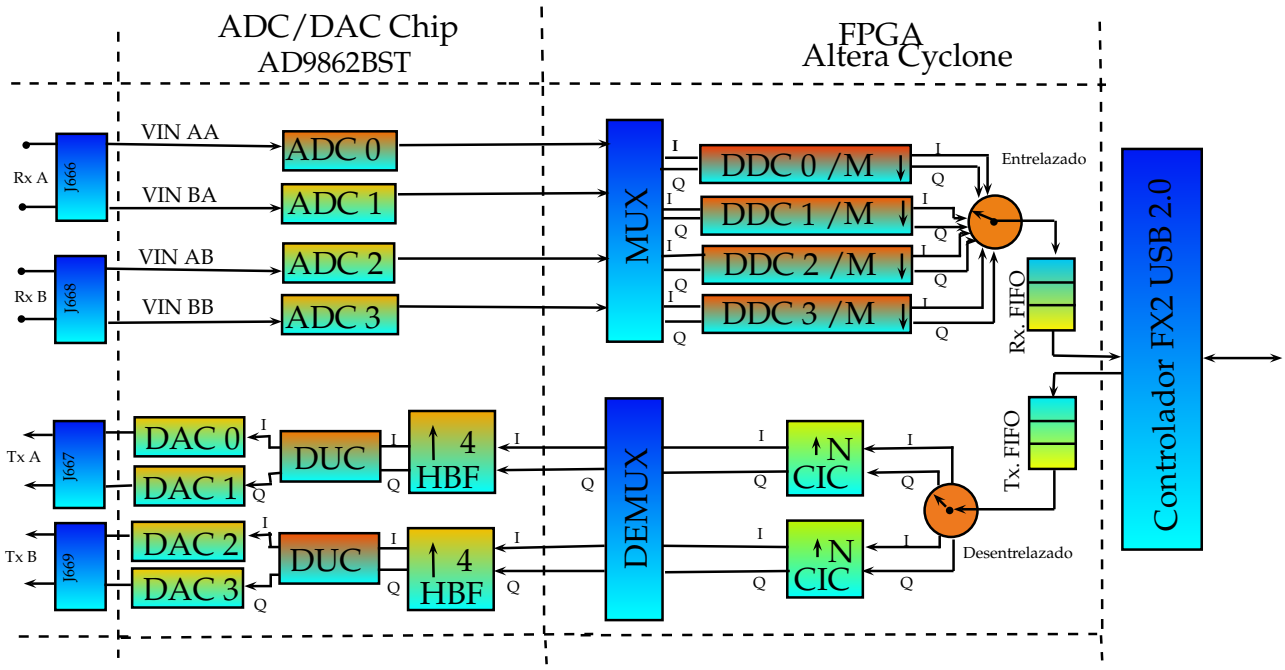


Figura 2.4: Diagrama a bloques de la tarjeta madre del USRP1.

El FPGA Altera Cyclone EP1C12, proporciona 4 DDC (siglas del ingl. *Digital Down Converter*) los que realizan el traslado de la señal de frecuencia intermedia (IF) a banda base. Tipícamente se realizan los deplazamientos de frecuencia utilizando mezcladores (mixer). También contiene 1 multiplexor, 1 demultiplexor y 2 filtros interpoladores CIC (siglas del ingl. *Cascaded Integrated Comb*), los que frecuentemente son utilizados tanto para reducir la velocidad de muestreo (diezmado) como para incrementar la velocidad de muestreo (interpolación), como se muestra en la figura 2.4.

El Controlador USB Cyprees EZ-USB FX2 de alta velocidad (USB2) proporciona la interfaz USB, del USRP1, como se muestra en la figura 2.4.

El AD9862[17] proporciona dos DUC (siglas del ingl. *Digital Upconverter*) los que realizan los traslados de la señal en banda base a frecuencias intermedias (IF) utilizando mezcladores (mixers). También contiene 4 ADC (siglas del ingl. *Analogic Digital Converter*), 4 DAC (siglas del ingl. *Digital Analogic Converter*) y 4 (pines) para la interconexión de la tarjeta madre con las tarjetas hijas de RF, como se muestra en la figura 2.4.

2.4.2. Tarjeta hija

Este elemento consiste de una tarjeta transceptora (*transceiver*) que cuenta con un transmisor y un receptor que comparten la misma circuitería contenida en la misma. En particular, la tarjeta hija elegida para este trabajo es la Ettus Research modelo RFX900, con una antena VERT 900 para operar en la banda de 900 Mhz y una potencia de salida tipíca de 200 mW. A continuación se describen el transmisor y receptor del transceptor.

El Transmisor se muestra en la figura 2.5. El transmisor es de tipo homodino, también llamado Zero IF o de conversión directa. Éste sistema modula directamente la señal banda base (pasabaja) a señal RF (pasabanda) sin realizar ningún tipo de procesamiento en frecuencias intermedias (FI), por lo tanto $f_{LO} = f_{RF}$, como se muestra en la figura 2.6.

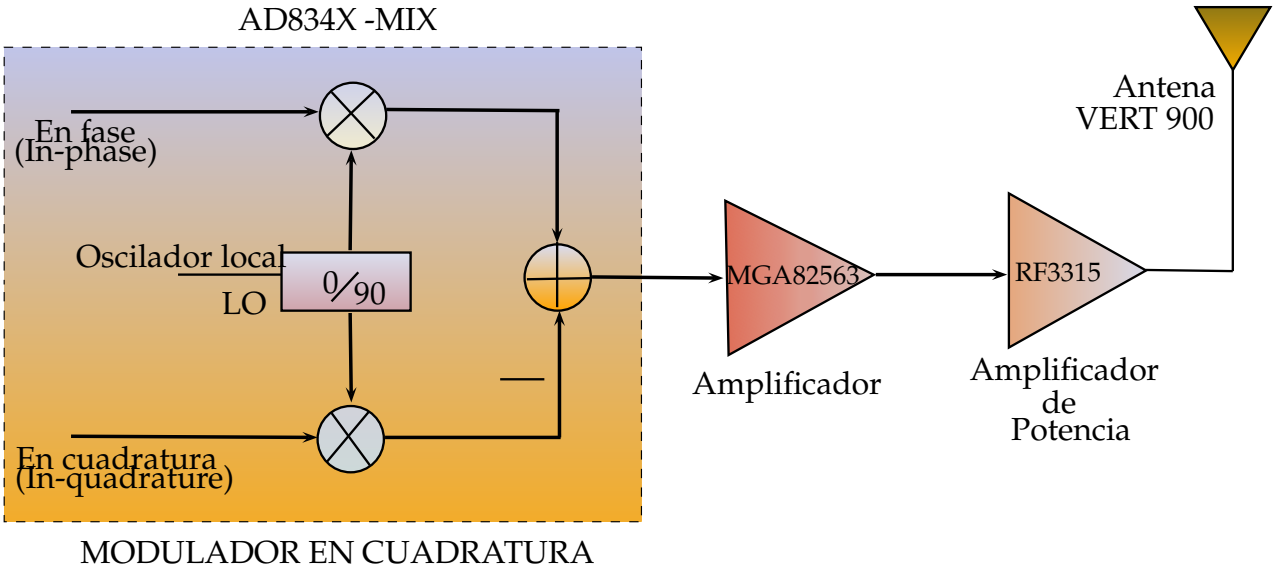


Figura 2.5: Diagrama de la tarjeta hija transmisora RFX 900, TX.

La señal compleja en banda base es interpolada y luego se pasa a través del DAC, como se muestra en la figura 2.4, del que sale una señal analógica. Esta señal compleja analógica entra al modulador de cuadratura (ver la figura 2.5) y la señal modulada es montada sobre la portadora de RF que es precisamente la frecuencia del LO (siglas del ingl. *Local Oscilator*), pasa a través de un filtro pasabanda y un amplificador de potencia para adecuar la señal y, transmitirla por medio de la antena VERT 900.

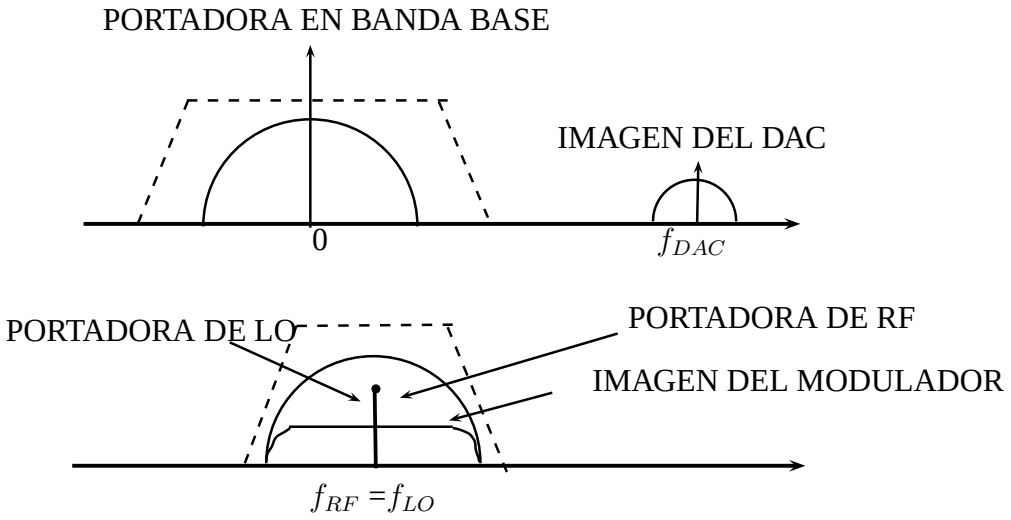


Figura 2.6: Espectro de una arquitectura Zero IF.

Es importante comentar que, en esta clase de configuraciones, las imperfecciones como la asimetría de las entradas *In-phase* y *In-quadrature* originan señales no deseadas en el LO así como imagenes de señal invertida dentro de la señal transmitida lo que origina una degradación del bit error rate de la señal.

El Receptor se muestra en la figura 2.8. El receptor también es de tipo homodino. El oscilador local (LO) se sintoniza a la misma frecuencia que la señal de RF. Luego, la señal de RF y el oscilador local se mezclan dando como resultado la señal en banda base, tal y como se muestra en la figura 2.7 [18].

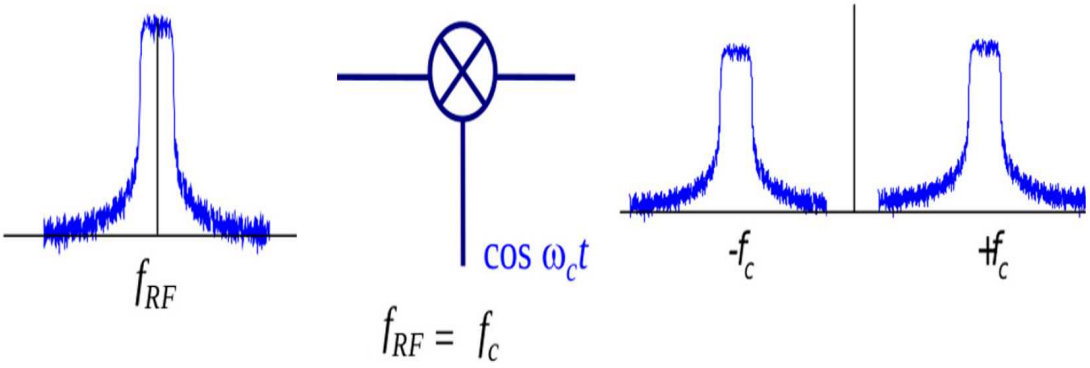


Figura 2.7: Señal de RF a banda base.

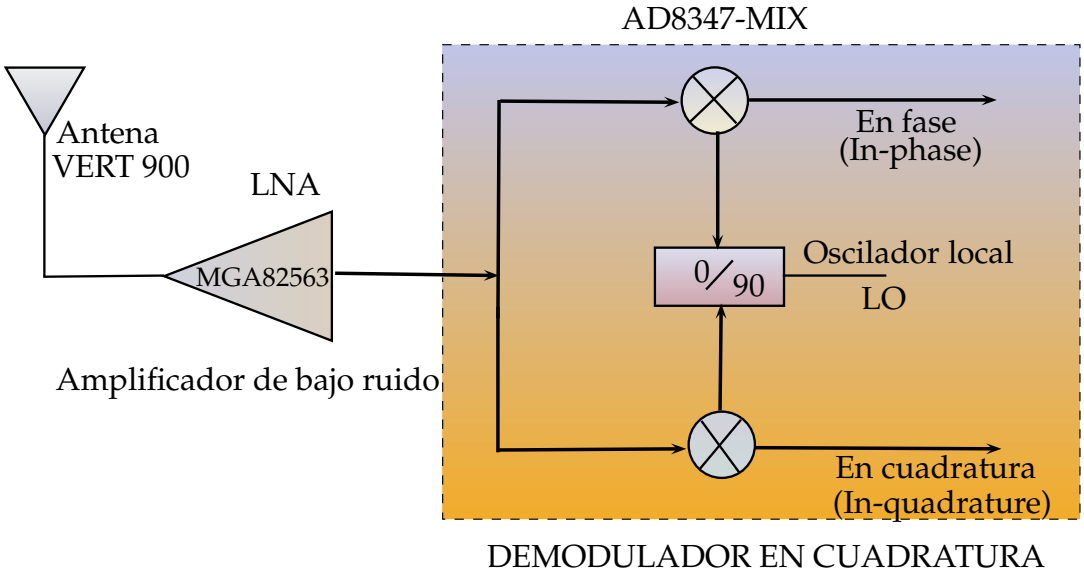


Figura 2.8: Diagrama de la tarjeta hija receptora RFX 900, RX.

La señal analógica proveniente del medio inalámbrico, es amplificada por el LNA (siglas del ingl. *Low Noise Amplifier*) y entra al demodulador de cuadratura (ver figura 2.8) que convierte la señal de RF a banda base. Una vez realizado el procesamiento en la parte analógica, la señal es enviada a la tarjeta madre (ver figura 2.4) para ser digitalizada al pasar por el ADC. Por último se pasa la secuencia resultante por un proceso de diezmado para ser enviada a la computadora de propósito general.
Este sistema tiene un menor costo y no necesita filtro para eliminar la frecuencia imagen pero la desventaja que presenta, es que necesita un ajuste preciso de la frecuencia del oscilador local a la frecuencia de RF recibida.

2.5. Software SDR

La mayor parte del software esta diseñado en capas, con el objetivo de hacerlo modular y adaptable al hardware donde se desea realizar la implementación práctica. Sin embargo, se procura que el lenguaje de programación sea orientado a objetos, con la finalidad de proporcionar servicios de comunicación entre capas con base en interfaces estándar [19].

El hardware seleccionado para la implementación práctica de este trabajo de investigación es el USRP1 de Ettus Research, por consiguiente nos enfocaremos en el Software SDR GNU Radio, el cual es compatible con dicho hardware [7].

2.5.1. GNU Radio

GNU Radio es un software de desarrollo, de código abierto bajo la licencia GPL (siglas del ingl. *General Public License*) que provee bloques, previamente construidos, de procesamiento de señal para implementar un SR (siglas del ingl. *Software Radio*), también puede ser utilizado con hardware adicional de RF para crear un SDR (siglas del ingl. *Software Defined Radio*), como se muestra en la figura 2.9, o solamente como un entorno de simulación sin la necesidad de un hardware adicional [20]. Éste proyecto fue iniciado en el año 2001 por Jhon Gillmore y Eric Blossom.

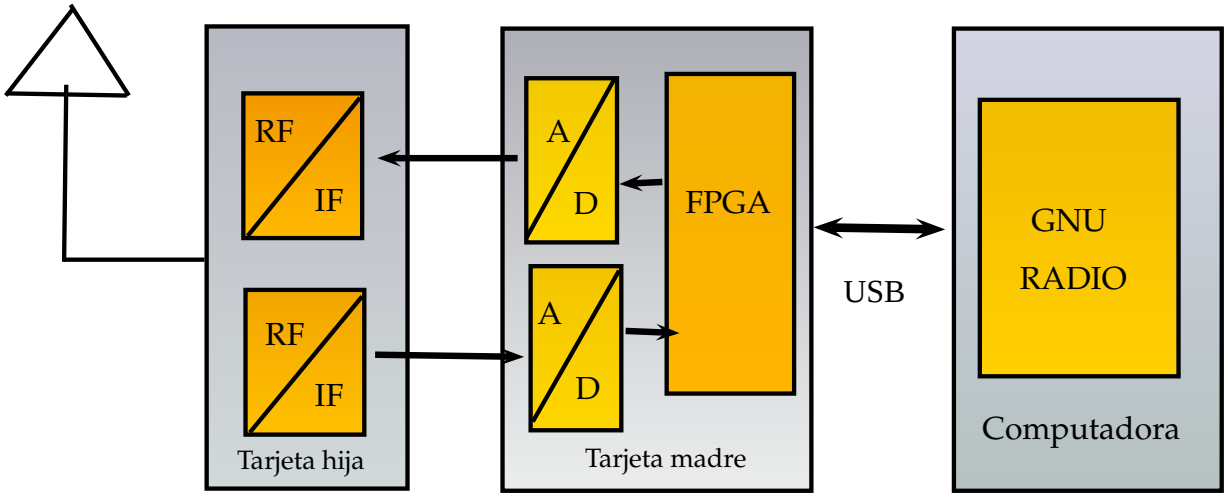


Figura 2.9: GNU Radio en el contexto de SDR.

El software de GNU Radio cuenta con bloques comunmente utilizados, tales como: Filtros, modelos de canales, elementos de sincronización, ecualizadores, demoduladores, etc. Solamente trabaja sobre secuencias de datos digitales, usualmente muestras en banda base. Por otra parte, el software GNU Radio se ejecuta en sistemas operativos como: Linux, MAC y Windows.

2.5.2. Arquitectura GNU Radio

El software GNU Radio esta conformado por un conjunto de bloques de procesamiento de señales, que son programados en lenguaje C++. Python, un lenguaje de programación de alto nivel, es utilizado para conectar los bloques procesamiento de señales en forma tal que se pueda representar al diagrama a bloques del sistema que se construye. SWIG (siglas de ingl. *Simplified Wrapper and Interface Generator*), una herramienta de desarrollo que realiza

la conexión de C++ y Python, como se muestra en la figura 2.10 [21].

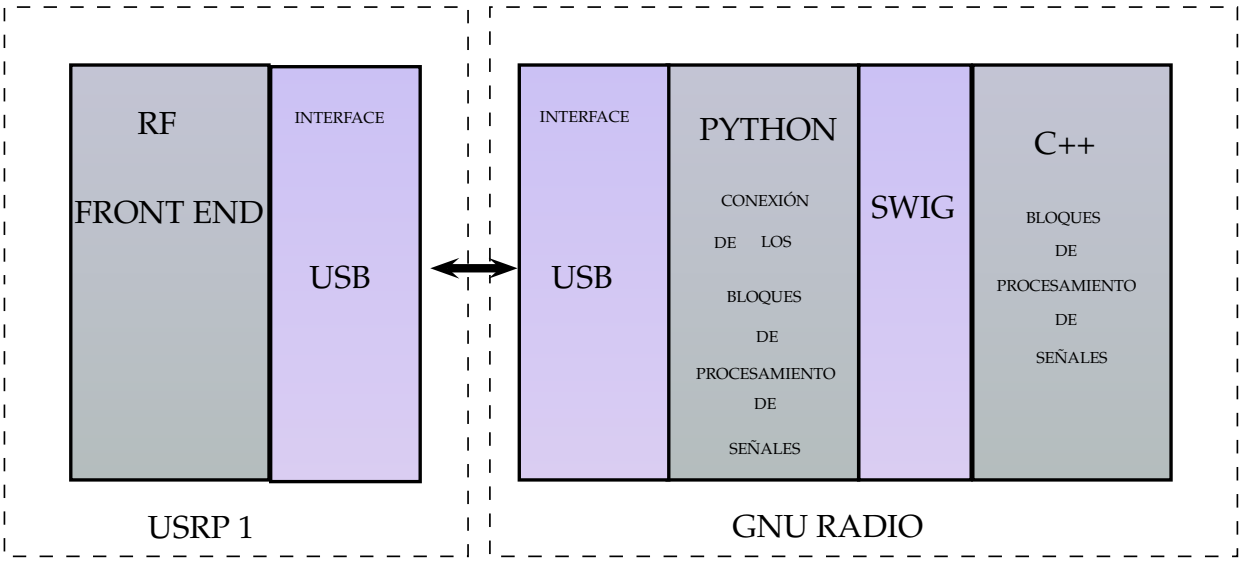


Figura 2.10: Arquitectura GNU Radio.

Para poder visualizar la conexión de los bloques de procesamiento de señales existe una interfaz gráfica llamada GNU Radio Companion, como se muestra en la figura 2.11 [20].

El software GNU Radio Companion (GRC) es una herramienta gráfica para crear los diagramas a bloques (denominados aquí como "grafos de flujo") y fue desarrollada, por Josh Blum. Se trata de una herramienta muy poderosa que incluye herramientas para crear un sistema de comunicación a nivel de banda base, además cuenta con instrumentos virtuales como osciloscopio, analizador de espectros y un gran compendio de útiles herramientas que, en conjunto con python, permiten realizar el análisis de la señal transmitida y recibida.

Las características principales de los bloques que se encuentran en el software GNU Radio Companion son las siguientes:

- Cuenta con puertos de entrada y salida. Estos son muy importantes para realizar bloques personalizados creados por el usuario. Hay dos tipos: flujo de datos y vectores.
- Ofrece distintos tipos de datos como: short, float, complex y byte.
- Hay tres tipos de bloques: 1) fuente, solo tiene una salida; 2) sumidero, solo tiene una entrada y 3) general, tiene entrada y salida, como se muestra en la figura 2.11.

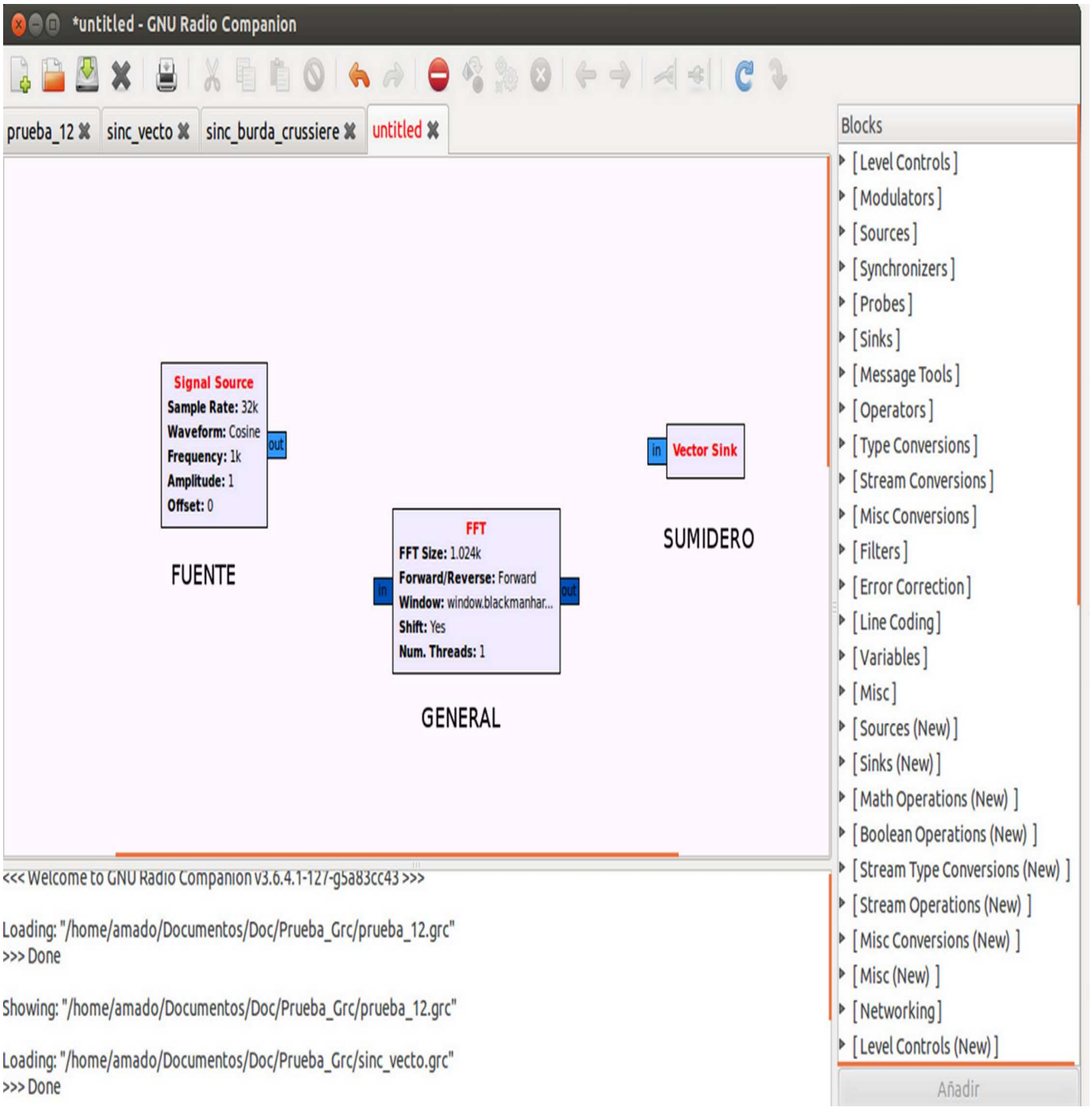


Figura 2.11: GNU Radio Companion.

La plataforma GNU Radio permite crear módulos personalizados por medio del archivo gr_modtool.py.

Los bloques personalizados se implementan mediante 4 archivos y carpetas principales:

- Archivos CC: En esta carpeta se escribe el proceso principal que realizará el bloque durante su funcionamiento y es implementado en lenguaje C++.
- Archivos H: En esta carpeta se encuentran las cabeceras y bibliotecas del bloque programado en lenguaje C++.
- Archivos XML (siglas del Ingl. *eXtensible Markup Language*): En esta carpeta se encuentra la interfaz gráfica del bloque, así como los campos para ser llenado durante la configuración del mismo. La programación en XML hace posible visualizar el bloque en GNU Radio Companion.
- Archivos i: También conocidos como archivos SWIG (siglas del ingl. *Simplified Wrapper and Interface Generator*), permiten obtener los parámetros de comunicación entre los módulos personalizados programados en lenguaje C++ y el enlace (según el grafo de flujo) de los módulos. Estos son programados en Python, como se muestra en la figura 2.12.

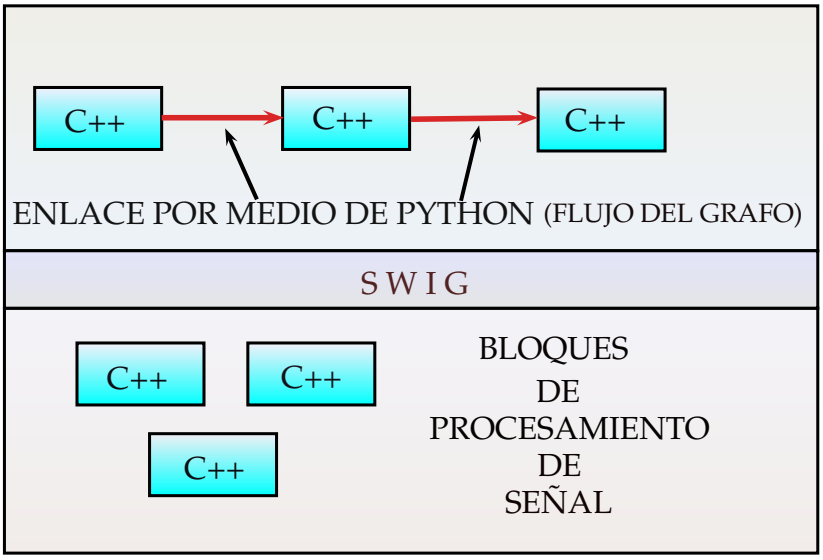


Figura 2.12: Enlace de los módulos personalizados a través de Python.

2.6. OFDM

2.6.1. Conceptos básicos

Dentro de los sistemas de comunicaciones digitales convencionales, se tienen a los sistemas con una única portadora, comúnmente llamados monoportadora, y a los sistemas con varias portadoras, conocidos como multiportadora.

En un sistema de comunicación digital monoportadora cada símbolo se transmite serialmente ocupando todo el ancho de banda disponible en el canal, como se muestra en la figura 2.13.

En un esquema de modulación multiportadora los símbolos son transmitidos paralelamente en múltiples subportadoras adyacentes que se reparten el ancho de banda del canal utilizando FDM (siglas del ingl. *Frequency Division Multiplexing*), de tal manera que exista una separación entre ellas para evitar traslapes, como se muestra en la figura 2.13[22].

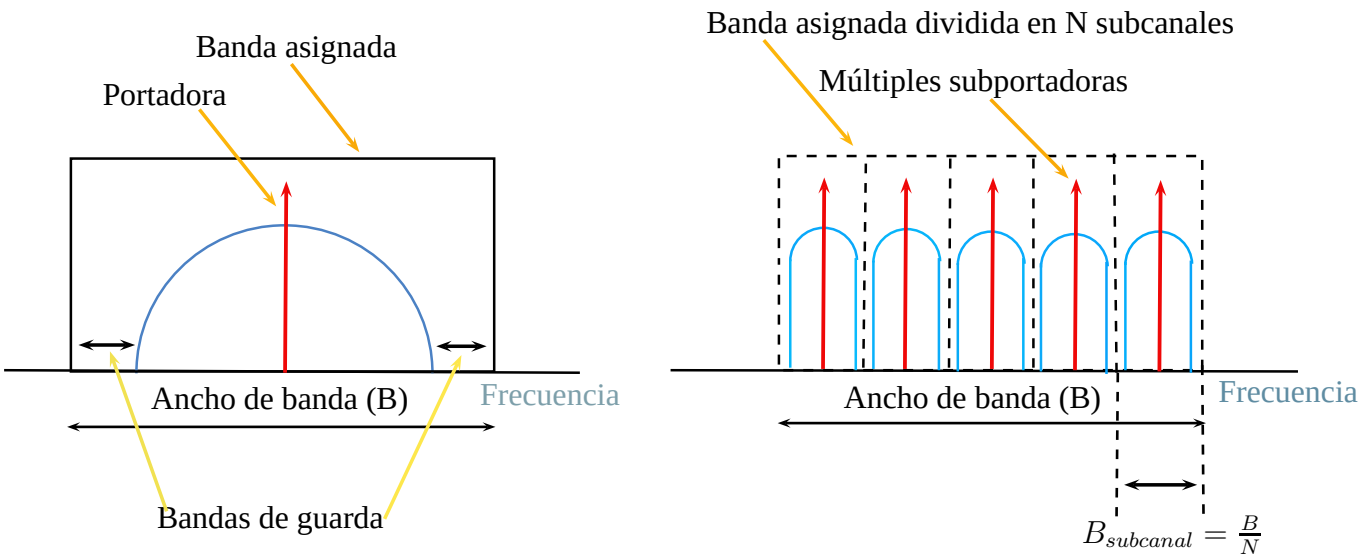


Figura 2.13: Esquema monoportadora (*single carrier*) y esquema multiportadora (*multicarrier*) utilizando FDM.

El uso ineficiente de la banda de frecuencias en FDM dio paso al desarrollo de OFDM (siglas del ingl. *Orthogonal Frequency Division Multiplexing*), que se encuentra dentro de los esquemas de modulación/demodulación multiportadora, donde la transmisión se realiza usando portadoras con frecuencias ortogonales. Las portadoras de OFDM se caracterizan por

la ortogonalidad con las adyacentes, logrando un uso eficiente de ancho de banda, como se muestra en la figura 2.14 en donde se aprecia la diferencia entre el esquema FDM y OFDM [23].

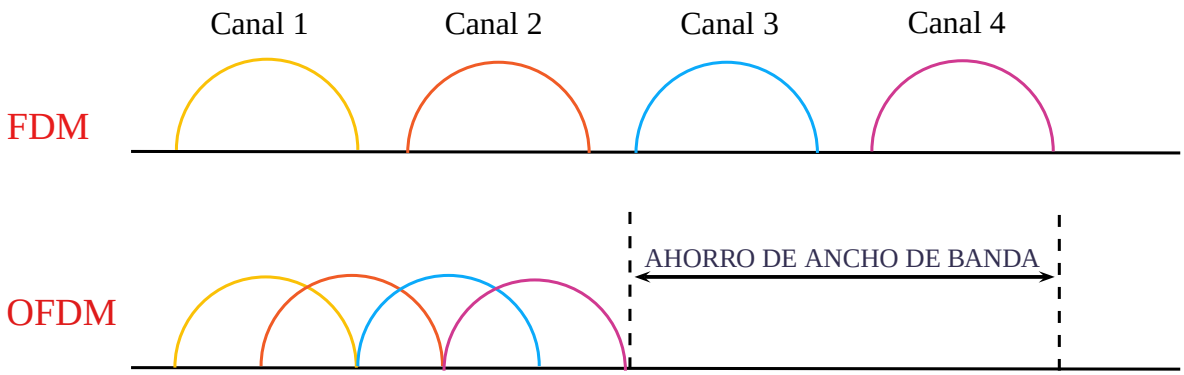
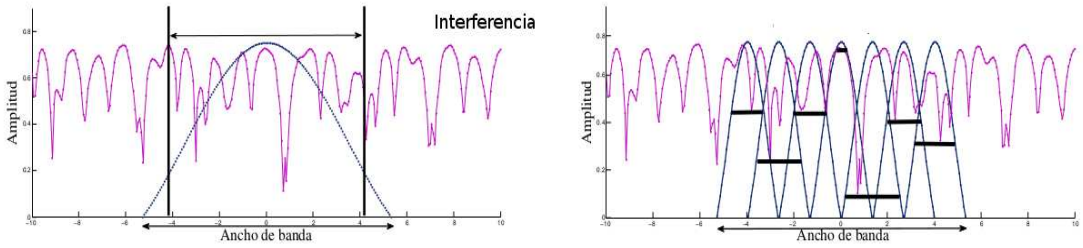


Figura 2.14: Comparación entre ancho de banda requerido para FDM y OFDM.

En un esquema OFDM, como sistema multiportadora, la interferencia afecta sólo a algunas portadoras y, por ello, tiene una gran ventaja sobre un sistema monoportadora donde la interferencia podría causar que el enlace de comunicación se pierda completamente, como se muestra en la figura 2.15[23].

Hoy en día, la técnica OFDM es una conveniente técnica de transmisón en la capa física, especialmente para ambientes propensos a efectos de multi-trayectoria, variantes en el tiempo y selectivos en frecuencia [23]; por ejemplo las redes de banda ancha como LTE y WIMAX.



(a) Esquema de una portadora. (b) Esquema multi-portadora.

Figura 2.15: Interferencia en sistema monoportadora y multiportadora [23].

2.6.2. Reseña histórica de la técnica OFDM

El concepto de la transmisión paralela y del multiplexado por división de frecuencia tiene sus orígenes a mediados de 1960. A continuación se comentan algunas fechas relevantes.

En los primeros sistemas de comunicación electrónica la eficiencia espectral era muy ineficiente. Debido a esto, a lo largo del tiempo se han realizado numerosos estudios para mejorar la eficiencia espectral y, finalmente, en el año de 1966, surge el primer artículo publicado por Robert W. Chang con la introducción de la técnica OFDM para contribuir a la solución de la problemática.

- En 1960 la técnica OFDM fue usada en múltiples sistemas militares a alta frecuencia tales como: KINEPLEX, ANDEFT y KATHARYN.
- En 1970 fue publicada una patente sobre OFDM.
- En 1971 Weinstein y Ebert emplean, por primera vez, la transforma discreta de Fourier (o DFT, siglas del ingl. *Discrete Fourier Transform*), para la modulación en OFDM.
- En 1980 Peled y Ruiz introducen el prefijo cíclico para resolver los problemas de pérdida de la ortogonalidad de las señales. Entonces, OFDM fue incluido en módems de altas velocidades y comunicaciones móviles digitales. Por primera vez, también se utilizó OFDM con modulación QAM (siglas del ingl. *Quadrature Amplitude Modulation*) usando la DFT.
- En 1990 OFDM fue utilizado para comunicaciones en banda ancha en enlaces de radio FM (siglas del ingl. *Frequency Modulation*), en líneas HDSL (siglas de ingl. *Digital Subscriber Line*) de 1.6 Mbps.
- En 2002 se especifica OFDM como una técnica de modulación para el estándar IEEE 802.11g.
- En 2004 OFDM se considera para el estándar IEEE 802.11n, una nueva generación de comunicación inalámbrica.
- En 2009 la transmisión OFDM, en un canal óptico, logra transmitir datos a una velocidad de hasta 160 Gbps.
- En 2010 se elige la técnica OFDM para el estándar 3GPP *Long Term Evolution* (LTE), el más reciente estándar para comunicación de datos de alta velocidad.

Actualmente, la técnica OFDM es ampliamente adoptada en estándares de redes inalámbricas de banda ancha y redes de línea fija tales como 802.11 a/g Wi-Fi, 802.16 WIMAX (siglas del ingl. *Worldwide Interoperability for Microwave Acces*), LTE (siglas del ingl. *Long Term Evolution*), DAB (siglas del ingl. *Digital Audio Broadcasting*), DVB (siglas del ingl. *Digital Video Broadcasting*) y DSL (siglas del ingl. *Digital Subscriber Loop*) [24].

2.6.3. Fundamentos teóricos de OFDM

OFDM es una clase especial del esquema de modulación multiportadora que transmite una trama de datos digital de alta velocidad; dividiendola en mútiples canales ortogonales de datos en paralelo, lo que da origen a lineas datos de baja velocidad, estos canales se les denomina subcanales [25].

El concepto que hace de OFDM una técnica ampliamente usada es el concepto de las subportadoras ortogonales. El uso de frecuencias ortogonales en OFDM permite la superposición de múltiples portadoras en una misma señal. Cada banda se sobrepone a su adyacente sin afectar la recuperación ni generar ICI (siglas del ingl. *Inter Carrier Interference*). Dos señales son ortogonales si el producto punto entre ambas es igual a cero [23].

Aunque OFDM ha existido de manera conceptual durante varias décadas, su aplicación práctica sólo se hace realidad cuando surgen los microprocesadores y los dispositivos de lógica programable DSP (siglas del ingl. *Digital Signal Processor*). En particular, la parte más importante para llevar a cabo esta modulación es la utilización de la Transformada Rápida de Fourier (o FFT, siglas del ingl. *Fast Fourier Transform*).

2.6.4. Modulación OFDM

En un símbolo OFDM los datos de alta velocidad son transformados en datos en paralelo montados sobre N subcanales o subportadoras. A su vez, los datos transmitidos de cada subcanal en paralelo son modulados en QPSK o QAM, como se muestra en la figura 2.16.

Considerando una secuencia de datos modulados en cuadratura para los N subcanales (d_0, d_1, d_2,.....,d_{N-1}) donde cada d_n es un número complejo, $d_n = d_{I_n} + jd_{Q_n}$, resultante de los valores para d_{I_n} y d_{Q_n} conforme a una modulación QPSK (siglas del ingl. *Quadrature Phase Shift Keying*) o para una modulación 16QAM (siglas del ingl. *Quadrature Amplitude Modulation*) que son enviados al bloque de la *Transformada Rápida de Fourier*, como se muestra en la figura 2.16, para así obtener el símbolo OFDM.

En el dominio de la frecuencia, OFDM permite que el espectro de cada suportadora se

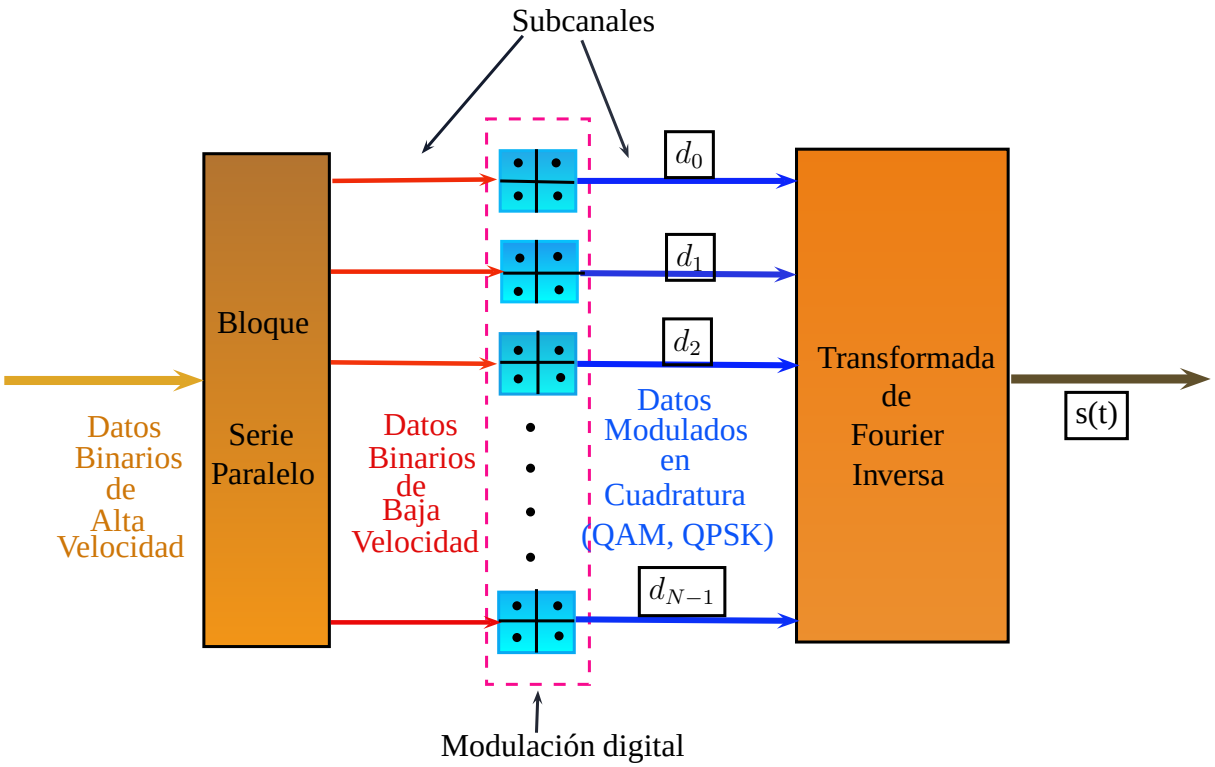


Figura 2.16: Modulación OFDM.

superponga sin traslapes porque conservan la ortogonalidad, esta condición entre múltiples subportadoras se satisface cuando su frecuencia central es espaciada $\frac{n}{T_s}$ donde n es un entero y T_s es la duración del símbolo. En la figura 2.17 se observa que el pico del espectro de la subportadora corresponde a un cruce por cero del espectro de otra subportadora. Así, la ortogonalidad permite un uso eficiente de los recursos del espectro.

En el dominio del tiempo, el símbolo OFDM es la suma de todas las subportadoras, como se muestra en la figura 2.18.

2.6.5. Señal transmitida con OFDM

El modulador OFDM recibe como entrada, $b \times M$ bits en serie, de alta velocidad, que son divididos en M módulos paralelos, de menor velocidad, definiendo así a los símbolos complejos de modulación con la forma $c_n = a_n + jb_n$, con $n = 0, 1, ..., N$ que pertenecen a una constelación QAM o QPSK, para igual número de subcanales espectrales correspondientes a las N subportadoras.

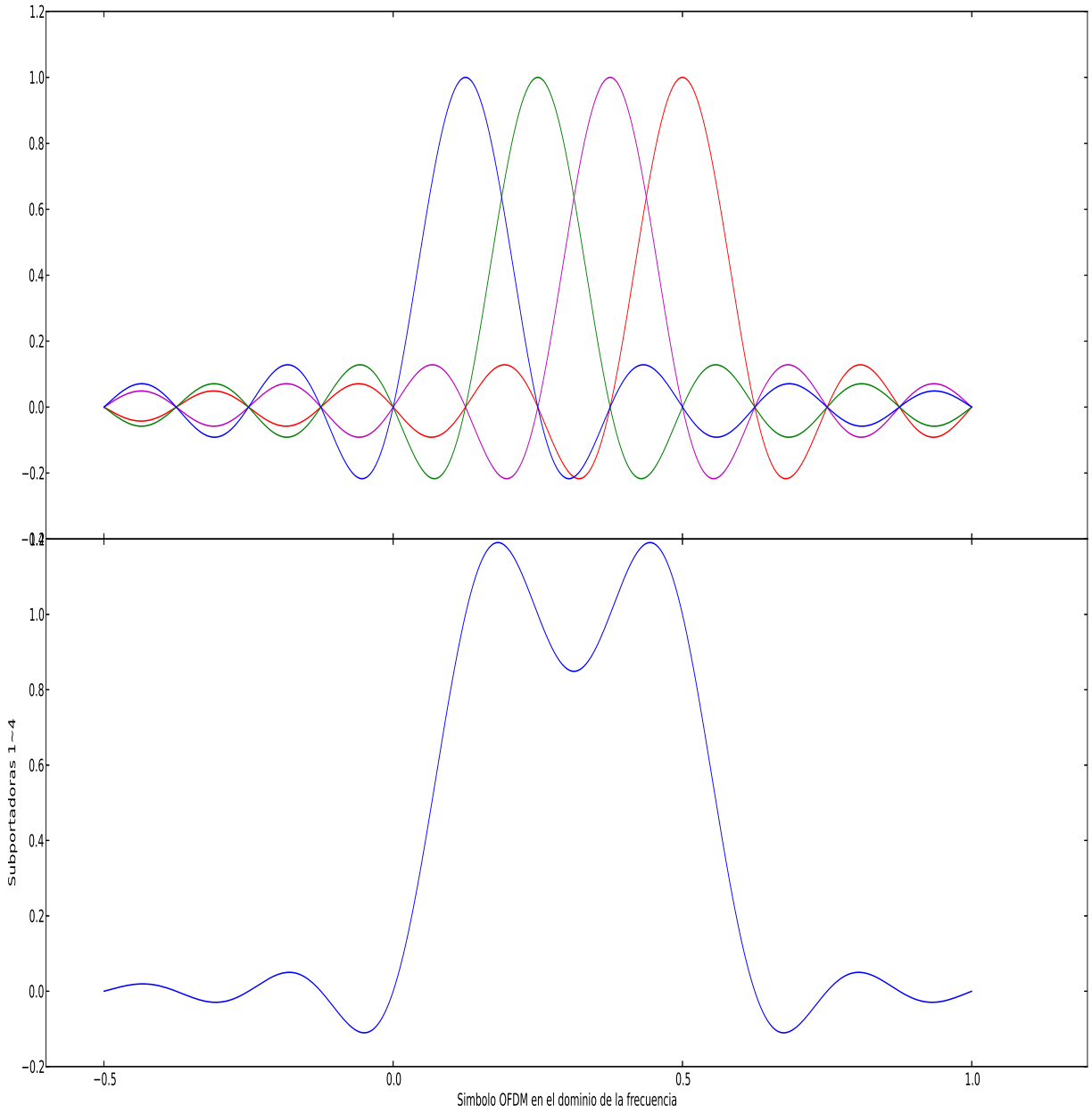


Figura 2.17: Subportadoras ortogonales y símbolo OFDM en el dominio de la frecuencia.

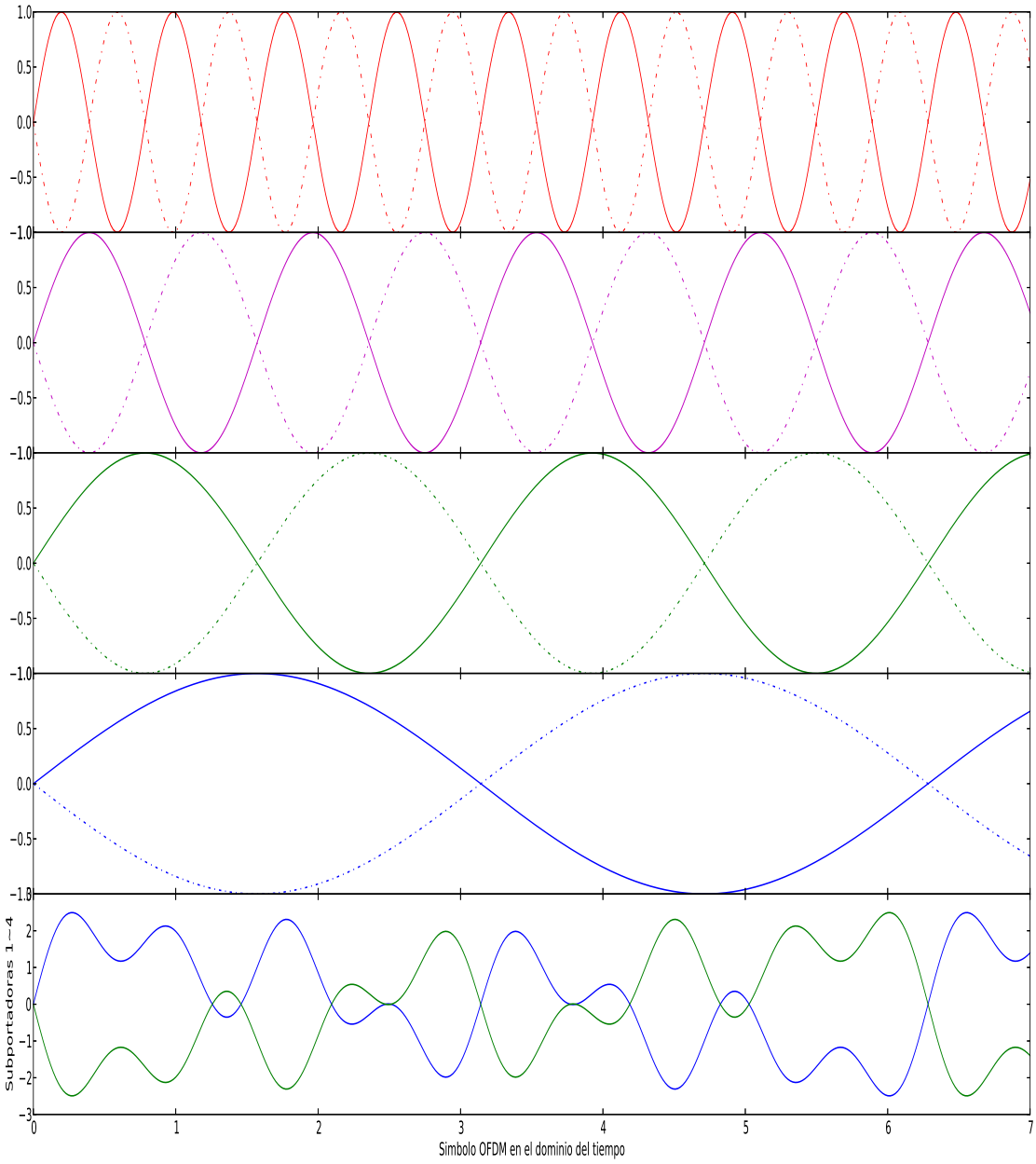


Figura 2.18: Subportadoras y símbolo OFDM en el dominio del tiempo.

La duración de un símbolo OFDM es igual a N veces el periodo de un bit individual (T_{sc}), es decir $T_s = NT_{sc}$. El ancho de banda se obtiene dividiendo el número de subportadoras entre la duración de símbolo OFDM, esto es $B_s = N/T_s$ [23].

Si se aumenta suficientemente el número de subportadoras, la duración del símbolo aumenta significativamente respecto de la respuesta al impulso del canal, con lo que los símbolos OFDM, se ven menos afectados por el efecto de multi-trayectoria y, como consecuencia, disminuye el efecto ISI [1].

Ahora partamos de que la señal OFDM se constituye de N subportadoras de frecuencia $f_k = f_0 + k\Delta_f, k \,\epsilon\, [0 : N-1]$, utilizadas para la transmisión paralela de N símbolos denotados x_k. Los símbolos d_k son elementos complejos que toman su valor de un alfabeto conforme a la modulación digital (QAM, BPSK). Si usamos la función rectangular $\Pi(t)$ como función de formateo de pulso, la cual cumple con los criterios de ortogonalidad, aseguramos que $\Delta_f = \frac{1}{T_s}$. Entonces, la expresión normalizada de la señal OFDM generada durante el intervalo $[0 : T_s]$ está dada por: [1]

$$s(t) = \frac{1}{\sqrt{N}} \sum_{k=1}^{N-1} \Re \left\{ x_k \prod(t) e^{j2\pi(f_0 + \frac{k}{T_s})t} \right\} t \;\epsilon\; [0...T_s] \tag{2.1}$$

Donde $\prod(t)$ representa la función rectángulo que está definida por

$$\prod(t) = \begin{cases} A & |t| < \frac{1}{2}T_s \\ 0 & Otro\ \ caso \end{cases} \tag{2.2}$$

y A es la amplitud de la señal. En la figura 2.17 se presenta el espectro en frecuencia de la función rectángulo. Por otro lado, al definir a la frecuencia central como $f_c = f_0 + N/(2T_s)$ y sustituir en la ecuación (2.1) se tiene [23]

$$s(t) = \Re \left\{ \prod(t) e^{j2\pi f_c t} \sum_{k=0}^{N-1} \frac{x_k}{\sqrt{N}} e^{j2\pi(k - \frac{N}{2})\frac{t}{T_s}} \right\} \tag{2.3}$$

que puede expresarse como:

$$s(t) = \Re\{\tilde{s}(t) \prod(t) e^{j2\pi f_c t}\} \tag{2.4}$$

Donde $\tilde{s}(t)$ es la parte compleja de la señal $s(t)$ antes del ventaneo por la función rectángulo. Estando limitado el espectro al intervalo $[-\frac{N}{2T_s} : \frac{N}{2T_s}]$, la señal $\tilde{s}(t)$ puede ser muestreada a una frecuencia $f_m = \frac{N}{T_s}$ sin que haya superposición espectral. Las muestras obtenidas se expresan

$$s_n = \Re\left\{\prod(n\frac{T_s}{N})e^{j2\pi f_c n\frac{T_s}{N}}\sum_{k=0}^{N-1}\frac{x_k}{\sqrt{N}}e^{j2\pi(k-\frac{N}{2})\frac{n}{N}}\right\}$$

$$= \Re\left\{(-1)^n\prod(n\frac{T_s}{N})e^{j2\pi f_c n\frac{T_s}{N}}\underbrace{\sum_{k=0}^{N-1}\frac{x_k}{\sqrt{N}}e^{j2\pi\frac{kn}{N}}}_{DFT^{-1}}\right\} \tag{2.5}$$

De la ecuación (2.5), se observa que la señal puede ser generada fácilmente utilizando una transformada discreta de Fourier inversa (o IDFT, siglas del ingl. *Discrete Fourier Transform*) [1], mientras que el receptor tiene que realizar una DFT para recuperar los símbolos QAM o PSK, según sea el caso [23,26].

Otra representación de la señal OFDM es en forma matricial, donde se representa como

$$S_k = F^{-1}x_k \tag{2.6}$$

Siendo S_k el vector de salida $[S_0, S_1, ..., S_{(k-1)}]^T$, que representa al símbolo OFDM, F es la matriz de Fourier de orden N x N definida por:

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & e^{\frac{-j2\pi}{N}} & \cdots & e^{\frac{-j2\pi(N-1)}{N}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & e^{\frac{-j2\pi(N-1)}{N}} & \cdots & e^{\frac{-j2\pi(N-1)^2}{N}} \end{bmatrix} \tag{2.7}$$

Obsérvese que esta matriz es unitaria y su inversa es igual a su transpuesta Hermitiana (conjugada) $F^{-1} = F^H$ [23].

2.6.6. Intervalo de guarda

Una solución para reducir la interferencia intersímbolo es incrementar la duración del símbolos (T_s) de manera ilimitada. Sin embargo, este método podría dificultar la implementación en términos de la estabilidad y ortogonalidad entre las subportadoras y el tamaño de la FFT. Otra manera de limitar la ISI, conservando el principio de ortogonalidad entre

[1] En la práctica se utiliza la transformada rápida de Fourier inversa (de *IFFT*, sigla del ingl. *Inverse Fast Fourier Transform*), la cual tiene una complejidad numérica de $O(NlogN)$ en función a N puntos, por otra parte, la complejidad de la IDFT es $O(N_2)$

portadoras, es agregar un intervalo de guarda (o GI, sigla del ingl. *Guard Interval*). Un GI es un lapso donde no se transmite información útil y para garantizar la ortogonalidad, el símbolo OFDM es precedido por una extensión periódica de la misma señal. A este intervalo se le conoce como prefijo cíclico y, por lo tanto, el símbolo OFDM resultante tiene una duración total de $T_g + T_s$, donde T_g es la duración del intervalo de guarda [23,26].

Por lo tanto, la señal a transmitir se ve afectada por la inserción del GI, resultando $s_n = [g_m s_k]$ donde g_m es una réplica de las últimas muestras de s_k. En la figura 2.19 ilustra un símbolo OFDM con su respectivo prefijo cíclico.

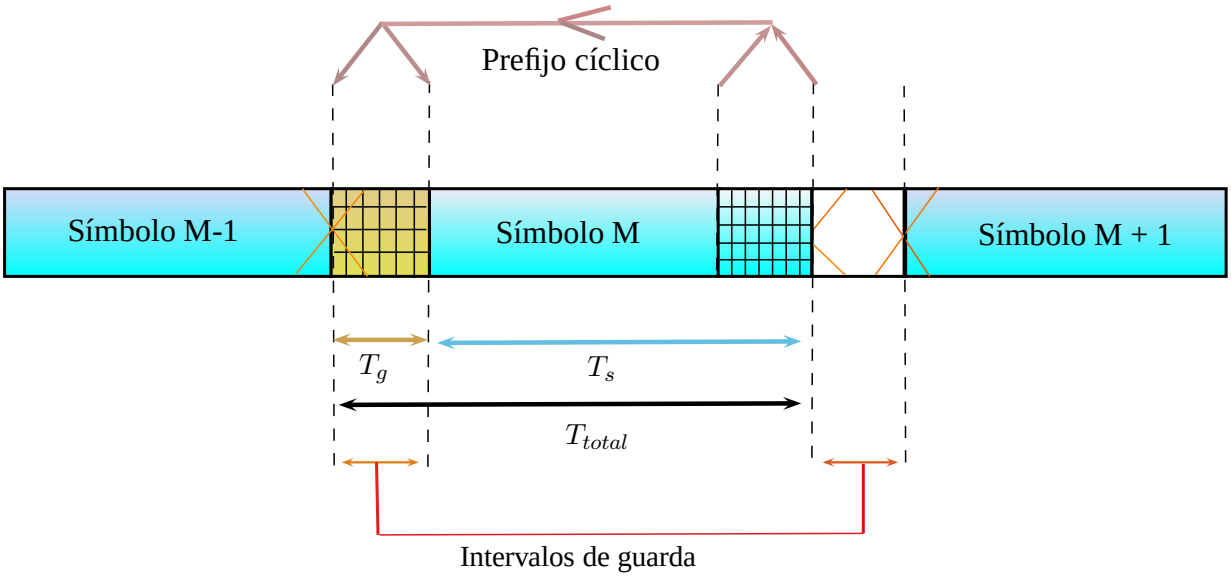


Figura 2.19: Inserción de intervalo de guarda.

Para evitar totalmente la ISI, la respuesta al impulso del canal T_h debe ser menor que T_g del prefijo cíclico. Es decir se debe cumplir que $T_h < T_g$, como se muestra en la figura 2.20.

2.6.7. Símbolo piloto

El uso de un símbolo piloto en la transmisión de OFDM y otros esquemas de comunicación, se conoce como esquema asistido por piloto. La información contenida en el piloto es conocida por el receptor y juega un papel de suma importancia para la sincronización, estimación de canal y estimación de ruido. La inserción de los símbolos pilotos se realiza, como se muestra en la figura [2.21][23], agregándolos al inicio de cada trama de símbolos

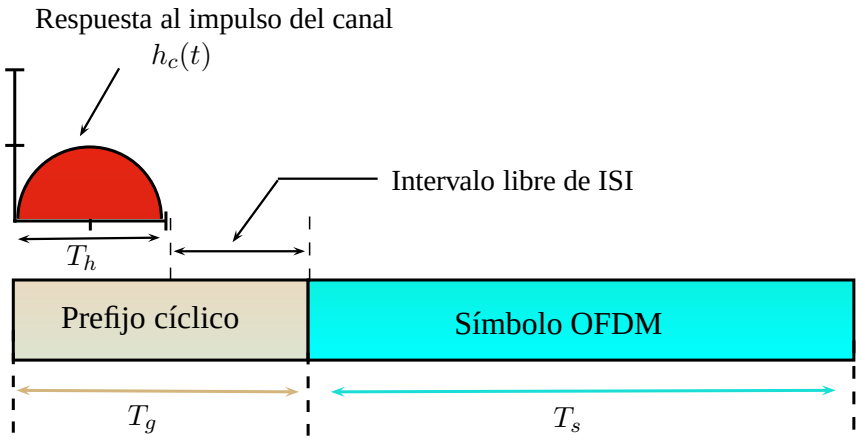


Figura 2.20: Prefijo cíclico para evitar interferencia entre símbolos.

OFDM. Una trama OFDM, es una secuencia de símbolos OFDM que portan los datos útiles.

Los símbolos piloto pueden ser de los siguientes tipos [23]:

- Los símbolos de tipo "A"se insertan al inicio de cada trama OFDM, y se utilizan en proceso de sincronización y estimación de canal.
- Los símbolos de tipo "B"se transmiten sólo eventualmente , y se colocan enseguida de los de tipo "A". Éstos se emplean para la estimación de ruido.

2.7. Proceso de sincronización y estimación de canal

En esta sección se presentan los modelos formales para el proceso de sincronización y de estimación de canal, propuestos por M. Crussière en [27].

2.7.1. Sincronización

Se dice que dos señales con portadora múltiple están sincronizadas si la ortogonalidad entre las subportadoras se preserva una vez recibidas las señales. Lo que implica que en el receptor, la ventana de muestreo inicie en el instante preciso del inicio de los símbolos OFDM. Crussière propone 2 esquemas:

a) Sincronización burda. En la primera etapa, se estima el inicio aproximado de los símbolos, para ello se utiliza la técnica de correlación de prefijo cíclico. Esta etapa garantiza

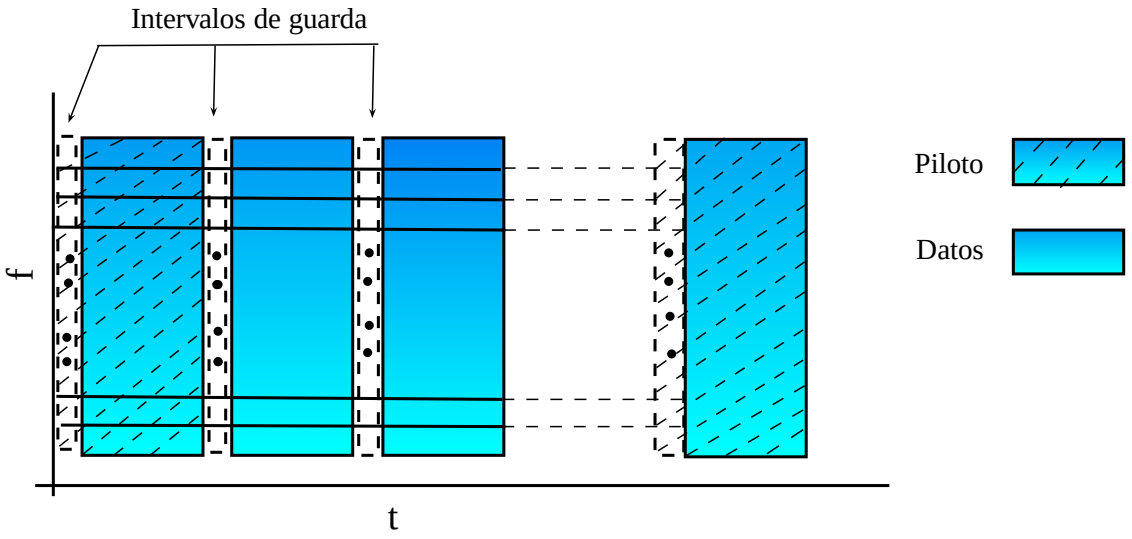


Figura 2.21: Esquema OFDM asistido por piloto [23].

una buena estimación del inicio de cada símbolo OFDM.

b) Sincronización fina. En la segunda etapa, se realiza un ajuste más preciso del inicio del símbolo siguiente. Para ello, se emplea la técnica de mínimos cuadrados (o LS, siglas del ingl. *Least Square*) y la información de fase de los símbolos piloto tipo "A".

Sincronización burda

En la propuesta presentada en [27], para la estimación de la diferencia de tiempo o desfase (δ_t) de las muestras de los símbolos OFDM, el vector de los datos recibidos $\mathbf{r}$ está compuesto en realidad de las muestras obtenidas a una velocidad $2f_m$. La ventana activa selecciona entonces $N' = 2N$ muestras de entre un total de $2N + D$ muestras. D representa las muestras del intervalo de guarda tomadas a una velocidad de $2f_m$. Para el caso de una transmisión por un canal gaussiano, se emplea el principio de máxima verosimilitud (o ML, siglas del ingl. *Maximum Likelihood*) en el dominio temporal.

$$\delta_t^{ML} = \frac{1}{2} arg\_\underset{\delta_t}{max}(\Psi(\delta_t) - \frac{\rho}{2}\Phi(\delta_t)) \tag{2.8}$$

donde $\Psi(\delta_t)$ es la métrica de correlación de la señal recibida sobre D muestras con la misma señal desplazada en 2N (donde N son los puntos de la FFT) posiciones. La métrica $\Phi(\delta_t)$, es la potencia total de D muestras más otras D muestras separadas en N' posiciones. Ambas métricas se definen como sigue:

$$\Psi(\delta_t) = \sum_{n=\delta_t}^{\delta_t+D-1} r_n r_{n+N'} \tag{2.9}$$

$$\Phi(\delta_t) = \sum_{n=\delta_t}^{\delta_t+D-1} r_n^2 + r_{n+N'}^2 \tag{2.10}$$

donde r_n es la n-ésima ventana de muestras recibidas de tamaño $N' = 2N$.

Se define $\rho = \frac{SNR}{SNR+1}$, donde ρ tiene dos posibles limites: $\rho \to 1$ o $\rho \to 0$, esto si los niveles de la relación señal a ruido son altos o bajos, respectivamente. Por lo tanto, se obtienen las siguientes expresiones de estimación sub-óptimas con base al nivel SNR que exista:

$$\delta_t^{MMSE} = \frac{1}{2} arg\_\underset{\delta_t}{max}(\Psi(\delta_t) - \frac{1}{2}\Phi(\delta_t)) \tag{2.11}$$

$$\delta_t^{MC} = \frac{1}{2} arg\_\underset{\delta_t}{max}(\Psi(\delta_t)) \tag{2.12}$$

En la ecuación 2.11, se realiza una estimación del mínimo error cuadrático medio (o MMSE, siglas del ingl. *Minimum Mean Squared Error*) para niveles altos de la relación señal a ruido, mientras que para los niveles bajos se realiza una estimación por máxima correlación (o MC, siglas del ingl. *Maximum Correlation*), tal y como se muestra en la ecuación 2.12 y se observa en la figura 2.22. En la práctica, es más común usar una estimación MC. La correlación completa del GI sólo es calculada una vez cada inicio de la recepción, posteriormente sólo se actualiza la ventana correlación con las muestras entrantes. Esto se realiza de la siguiente manera [23]:

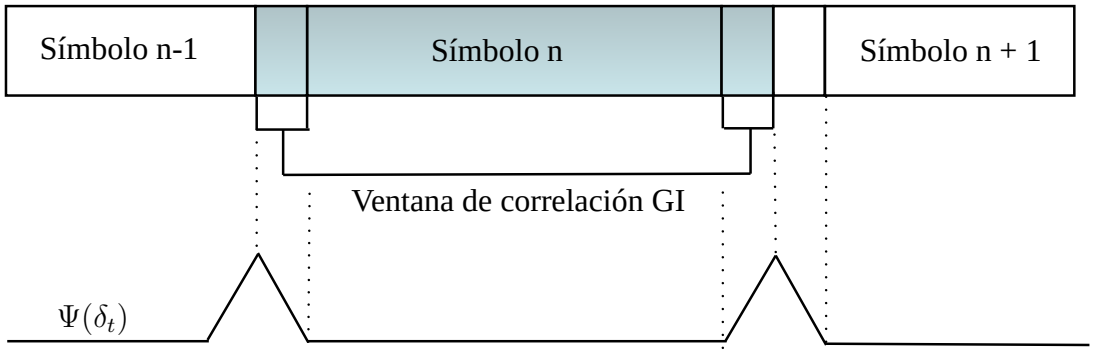


Figura 2.22: Correlación $\Psi(\delta_t)$ [23].

$$\Psi(\delta_t + 1) = \Psi(\delta_t) + r_{\delta_t+1} r_{\delta_t+1+N'} - r_1 r_{1+N'} \tag{2.13}$$

Los efectos de un canal gaussiano sobre la sincronización burda se traducen en una pérdida de pocisión o desfase. Para corregir este desfase se propone la etapa de sincronización fina, que se describe a continuación.

Sincronización fina

El proceso de sincronización fina, permite corregir el apuntador al inicio de la ventana del símbolo OFDM de forma perfecta. Crussière propone un ajuste fino con base en el principio de rotación de fase. Este procesamiento se realiza en el dominio de la frecuencia, es decir, después de aplicar la FFT a una ventana de N muestras recibidas. La estimación fina ($\widehat{\epsilon}$) se calcula con base en el principio de mínimos cuadrados lineales (o LLS, siglas del ingl. *Linear Least Square*) en el dominio de la frecuencia, y se propone como [23]:

$$\widehat{\epsilon}_f^{LLS}(p) = \frac{1}{2\pi} \frac{\sum_{k=0}^{N-1} k\Delta\phi_k(p)}{\sum_{k=0}^{N-1} k^2} \tag{2.14}$$

donde $\Delta\phi_k(p)$ es el resultado de la diferencia de fase entre el vector de datos del p-ésimo símbolo piloto transmitido y el recibido. La representación vectorial de la diferencia de fase se denota $\Delta\phi_k = [\phi_0(p), \phi_1(p), \cdots, \phi_{(N-1)}(p)]^T$, este vector también se expresa como

$$\Delta\phi_k = \angle(X_A^T Y_p) \tag{2.15}$$

donde $X_A^T = diag([x_0^A, x_1^A, \cdots, x_{N-1}^A])$ y Y_p es el vector de muestras recibidas afectadas por el canal. El operador $\angle$ representa el cálculo del ángulo de fase. Se trata de encontrar el ángulo de la pendiente de la recta que mejor ajusta las desviaciones de fase observadas. La corrección de fase la determina el operador angular aplicando lo siguiente regla:

$$\phi(k) = \begin{cases} \phi_k - 2\pi & si \quad (|\phi_k - \phi_{k-1}| < \pi) \quad y(\phi_k < 0), \\ \phi_k + 2\pi & si \quad (|\phi_k - \phi_{k-1}| < \pi) \quad y(\phi_k \geq 0), \\ \phi_k & otro \quad caso \end{cases} \tag{2.16}$$

2.7.2. Estimación de canal

En sistemas que utilizan el esquema OFDM, y que operan en un canal selectivo en frecuencia, si se cuenta con un número suficientemente grande de subcanales, las atenuaciones por cada subcanal pueden considerarse como prácticamente planas. Entonces, la estimación de canal se expresa como,

$$H_{(l,k)} = \frac{C'_{(l,k)}}{C_{(l,k)}} \tag{2.17}$$

Donde $H_{(l,k)}$ es la estimación para el coeficiente complejo de atenuación para la subportadora k del l-ésimo símbolo, $C(l,k)$ representa la secuencia de referencia y $C'_{(l,k)}$ la secuencia afectada por el canal. Dependiendo de la procedencia del valor de referencia, se pueden clasificar en 4 las técnicas de estimación de canal: asistidas por pilotos, regidas por decisión, semi-ciegas y ciegas.

En este trabajo se ha optado por un estimador de canal clásico asistido por pilotos. El uso de estos esquemas representa ventajas para los procesos de sincronización, estimación de ruido y de canal ya que todos ellos pueden emplear el mismo símbolo piloto. Sin embargo, presenta la desventaja de ocupar símbolos OFDM que no transportan datos del usuario, reduciendo con ello el ancho de banda útil [23].

Estimador clásico de canal asistido por piloto

En esta técnica los elementos de la secuencia de referencia $C_{(l,k)}$ constituyen a los símbolos pilotos y, por ello, se conocen tanto por el emisor como por el receptor, facilitando con ello el cálculo de la ecuación 2.17.

Estimación de canal por mínimos cuadrados (LS)

Los coeficientes del canal, obtenidos a través de la estimación LS son:

$$H^{LS}_{(l,k)} = \frac{C'_{(l,k)}}{C_{(l,k)}} \tag{2.18}$$

donde $C'_{(l,k)}$ es la muestra para el l-ésima símbolo OFDM recibido de la k-ésima subportadora $C_{(l,k)}$ es el símbolo original correspondiente.

Estimación por ventaneo rectangular de respuesta al impulso en el dominio temporal (VR)

Esta técnica realiza un filtrado por ventaneo rectangular sobre la respuesta al impulso de los coeficientes de estimación. El filtro rectangular, resulta ser una técnica eficiente y sencilla de implementar para eliminar el ruido en la estimación de canal.

Para llevar a cabo esta técnica, los coeficientes C^{LS} son mapeados al dominio temporal mediante la IFFT, posteriormente se colocan ceros en los valores de las muestras que se encuentran por encima de la máxima dispersión temporal del canal. Finalmente, la secuencia es regresada al dominio de la frecuencia mediante la FFT, obteniendo de forma simple una respuesta en frecuencia suavizada y una mejora en la estimación del canal [23].

Capítulo 3
Programación y evaluación

En esta capítulo se presenta el sistema de comunicaciones implementado en una plataforma SDR, con el propósito de diseñar, verificar y validar los algoritmos de nivel físico de un radio OFDM para comunicación digital. Los modelos formales programados y evaluados en este capítulo (descritos en el capítulo anterior), fueron propuestos M. Crussière [27], existe también una síntesis en español realizada por G. Laguna en [26], quien también desarrolló la simulación por computadora de transmisión sobre PLC [1]. Este simulador se ha empleado en el presente trabajo como sistema de referencia del que se han tomado algunos resultados y propuestas para ser puestos en práctica en una plataforma SDR.

Este trabajo es una extensión del proyecto titulado

"**Diseño y desarrollo de la capa física de un módem OFDM para la transmisión de datos empleando la línea eléctrica como canal de comunicación**"(**con folio UAM-PTC-284, oficio No. PROMEP/103.5/11/4296**).

La principal diferencia es que en este trabajo consideramos que el medio de transmisión es un canal de comunicación inalámbrica y, por ello, empleamos una plataforma SDR.

En esta sección damos un panorama del sistema de comunicaciones realizado en una plataforma SDR. En esencia, usamos una plataforma SDR para verificar y validar los algoritmos de nivel físico de un radio OFDM en condiciones reales. Partiendo del concepto SR que propone J. Mitola en [3] donde los sistemas de radio no dependan exclusivamente de hardware.

Se llegó, finalmente, al concepto más elaborado de SDR, donde los componentes físicos de un sistema de radio, que han sido típicamente diseñados en hardware (ejemplo: mezclador, moduladores, demoduladores, generación de datos, detectores) son implementados por medio de un software.

3.1. Plataforma SDR para el desarrollo del proyecto

La plataforma SDR a utilizar en este proyecto se compone de los siguientes elementos:

- **Hardware USRP1 de ETTUS RESEARCH**.
- **Tarjetas hijas RFX 900**.
- **Antenas VERT 900**.
- **Software GNU Radio**.

USRP1 de Ettus Research

El USRP1 es una plataforma flexible de bajo costo para desarrollo de SDR's. Las dos tarjetas principales, la tarjeta hija (realiza la función de *Front End*) y la tarjeta madre (realiza la función de procesamiento de la señal) fueron elegidas considerando sus especificaciones, a saber:

Tarjeta madre

Que incluye:

- **FPGA Altera Cyclone**.
- **Convertidores Analógico-Digital (ADCs) de 12 bits con una tasa de muestreo de 64 MS/s**.
- **Convertidores Digital-Analógico (DACs) de 14 bits con una tasa de muestreo de 128 MS/s**.
- **DDC(siglas del ingl. *Digital Down Converter*) con resolución de 15 Mhz**.

- **DUC(siglas del ingl. *Digital Up Converter*) con resolución de 15 Mhz.**

- **Interface USB 2.0.**

Tarjeta hija

- **VERT900. Antena vertical omnidireccional con ganancia de 3dBi, con intervalo de frecuencias: 824-960 Mhz y 1710-1990 Mhz.**

- **RFX900. Transceiver con operación en banda de 900 Mhz y potencia de salida típica de 200 mW.**

Software GNU Radio

- **GNU Radio Companion v3.6.4.1-127-g5a83cc43.**

- **UHD_003.005.002-43-gd745186d.**

- **Instalado sobre ubuntu 12.04.**

3.2. Arquitectura del sistema de comunicaciones

En la figura 3.1. se muestra la arquitectura del sistema de comunicaciones propuesto en [1], en donde se observan los módulos que conforman el flujo de datos de un enlace OFDM. Nuestro proyecto se va enfocar en dicho sistema de comunicaciones.

La plataforma SDR permite programar los bloques en banda base de la figura 3.1 utilizando el software SDR GNU Radio. Así, los bloques programados en GNU Radio durante el desarrollo de este trabajo son los siguientes:

- Generación de datos.
- Buffer serie/paralelo.

- Mapeo QAM.
- Modulación OFDM (IFFT).
- Prefijo cíclico.
- Sincronización.

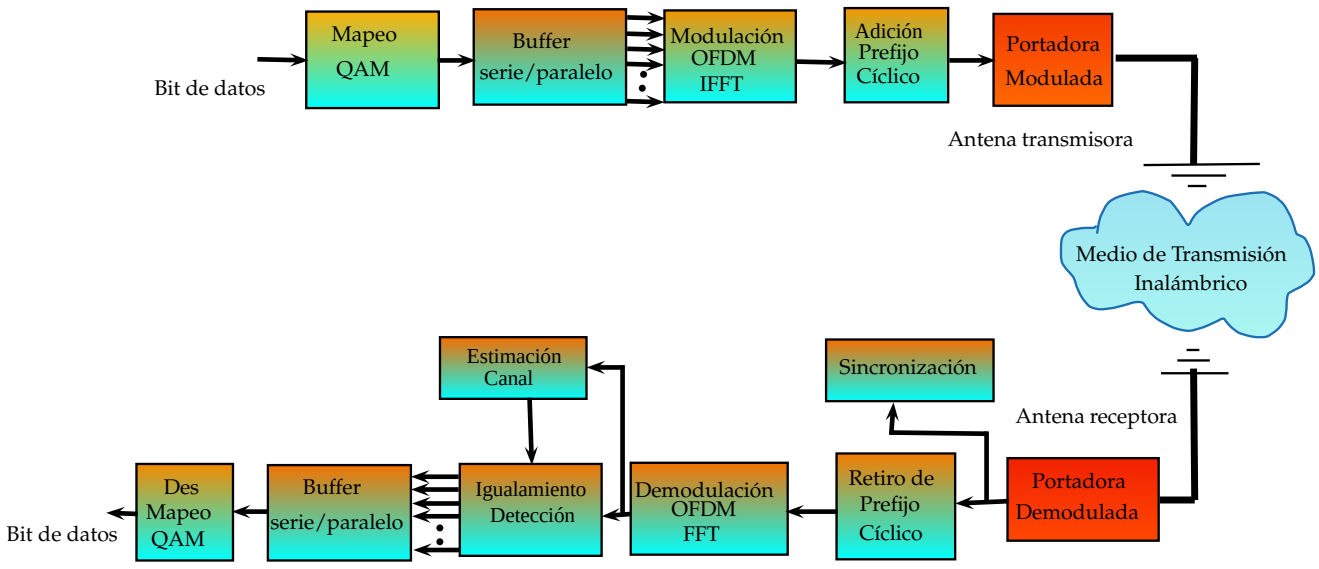


Figura 3.1: Flujo de datos en el enlace OFDM [1].

Basandonos en la arquitectura de GNU Radio, mostrada en la figura 2.10 del capítulo anterior, la programación de los módulos del sistema de comunicaciones descrito se realiza en lenguaje de programación C++.

Para visualizar los bloques del sistema de comunicación de la figura 3.1, en la interfaz gráfica GNU Radio Companion (ver figura 2.11) se implementa el grafo de flujo a través del lenguaje de programación Python, como se muestra en la figura 2.12.

3.2.1. Sistema de comunicaciones utilizando la arquitectura SDR

Para realizar bloques personalizados en GNU Radio se utiliza la herramienta gr_modtool.py que proporciona el software, dicha herramienta forma un esqueleto de carpetas para programar los módulos del sistema de comunicaciones en el software SDR, como se muestra en la

figura 3.2.

Los archivos para la creación de los bloques personalizados se almacenan en carpetas, (ver figura 3.2) de la siguiente manera:

- Los archivos *.CC* (bibliotecas y función principal) se guardan en la carpeta include y lib respectivamente.
- Los archivos *.XML* (para visualizar los bloques en GNU Radio Companion) se guardan en la carpeta grc.
- Los archivos *.PY* (para el grafo de flujo) se guardan en la carpeta python.
- Los archivos *.i* (interface entre C++ y Python) se guardan en la carpeta swig.
- La carpeta build es para la compilación de los programas que se van implementando.

Figura 3.2: Carpetas generadas por gr_modtool.py.

En la figura 3.3 se muestra el sistema de comunicaciones realizado en este proyecto en el contexto SDR (GNU Radio). Los bloques de procesamiento de señal en banda base son programados en lenguaje de programación C++ y el enlace del flujo de datos, entre estos bloques, es programado en lenguaje de programación de python.

A continuación se describe la función de cada bloque que conforma el sistema de comunicación digital.

- **Bits de datos**. Es el módulo inicial del sistema, se encuentra en la parte transmisora en banda base, éste se encarga generar la secuencia binaria de entrada al sistema. La secuencia se genera de forma pseudo-aleatoria para los datos, o bien, se usa alguna secuencia constante para símbolos pilotos [23].

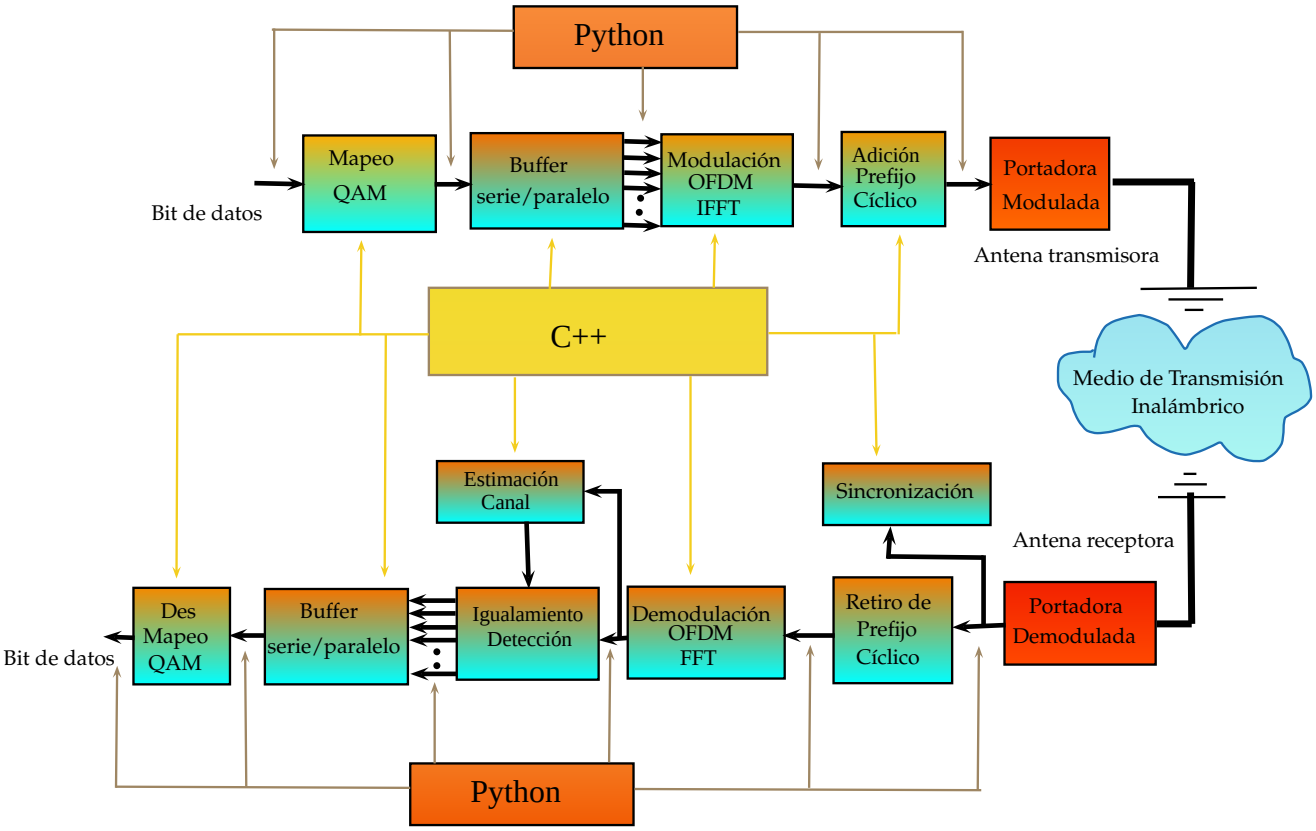


Figura 3.3: Flujo de datos en el enlace OFDM con la plataforma SDR.

- **Mapeo QAM**. Este módulo se encuentra en la parte transmisora en banda base y se encarga de transformar la secuencia binaria (grupo de bits) en un grupo de números complejos, representando a la constelación de símbolos de modulación.

- **IFFT (siglas del ingl. *Inverse Fast Fourier Transform*)**. Este módulo se encuentra en la parte transmisora en banda base y se encarga de la modulación OFDM (analizado en el capítulo anterior).

- **Adición de prefijo cíclico**. Este módulo se encuentra en la parte transmisora en banda base y permite mitigar la interferencia inter-símbolo (analizado en el capítulo anterior) debida a la respuesta al impulso de canal de comunicación.

- **Portadora Modulada**. Módulo que se encuentra en la parte transmisora, éste monta

la señal de datos sobre un portadora de RF a la frecuencia deseada para transmitirlos por el canal de comunicación inalámbrico.

- **Portadora Demodulada**. Módulo que se encuentra en la parte receptora, éste desmonta la señal de datos de la portadora de RF.

- **Sincronización**. Módulo que se encuentra en la parte receptora y permite conocer el instante preciso en el que inicia la secuencia de muestras correspondientes a un símbolo OFDM.

3.2.2. Diseño de los bloques del nivel físico de un radio OFDM en banda base, utilizando una arquitectura SDR

En la figura 3.4 se muestra nuestra propuesta de diseño de los bloques que conforman el sistema de comunicación digital OFDM empleando la plataforma GNU Radio.

En la primera capa (cuadros de color amarillo) se tienen los algoritmos de procesamiento de señal del sistema que son programados en lenguaje C++.

En la segunda capa (cuadros de color blanco) se tiene la programación en XML de los bloques gráficos para visualizarlos en GNU Radio Companion y poder usar los algoritmos de nivel físico del sistema de comunicación digital dentro del entorno gráfico.

En la tercera capa (rectángulo de color naranja) se enlazan los bloques que contienen los algoritmos de nivel físico de OFDM. Con ello, se indica el flujo de datos através de la plataforma GNU Radio, utilizando el lenguaje de programación python.

Descripición del tipo de bloques de acuerdo a los requerimientos de GNU Radio.

- *El bloque de Bits de entrada.* Es un bloque de tipo fuente, con salida vectores flotantes (vf).

- *El bloque serial a paralelo, mapeo QAM.* Es un bloque de tipo general, con entrada de vectores flotantes y salida de vectores complejos (vfc).

- *El bloque de simetría hermitiana (transpuesta conjugada de la señal de entrada).* Es un bloque del sistema PLC, no es funcional en un sistema inalámbrico por cuestiones de hardware.

- *El bloque de modulación OFDM.* Es un bloque de tipo general, con entrada y salida de vectores complejos (vcc).

- *El bloque de prefijo cíclico.* Es un bloque de tipo general, con entrada y salida de vectores complejos (vcc).

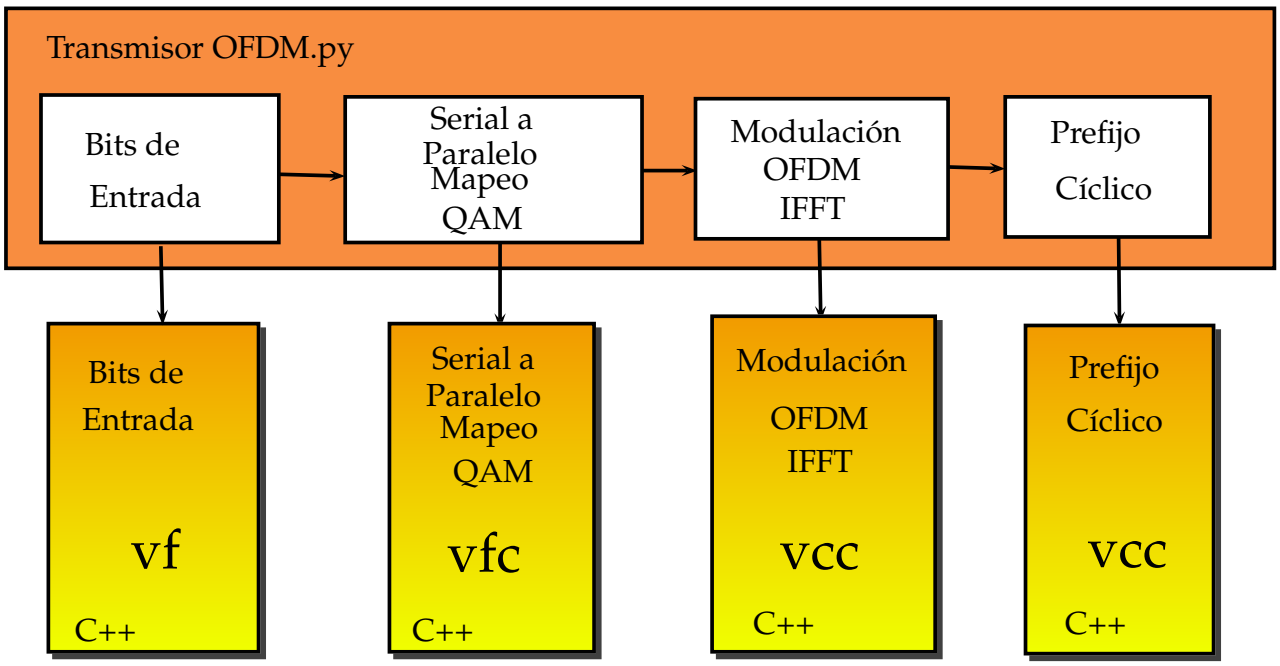


Figura 3.4: Diseño propuesto del flujo de datos en el enlace OFDM en una arquitectura SDR.

3.2.3. Transmisor en banda base

El módulo de transmisión genera una señal OFDM, partiendo de la figura 3.4, y esta conformado por los siguientes bloques: Bits de entrada con señalización antipodal, modulador

QAM y adición del prefijo cíclico. A continuación se describe a detalle cada uno de estos bloques. En el ápendice D se presenta el código de cada bloque.

Bits de entrada

Este bloque es la secuencia entrante, donde se generan los datos con señalización antipodal.

Señalización antipodal

Consiste en mapear las muestras de la secuencia entrante con valor '0' a un valor de '-1' y manteniendo las muestras con valor '1' en su valor, como se muestra en la figura 3.5. La secuencia saliente se expresa como

$$\overrightarrow{\mathbf{a}} = [a'_0, a'_1, ..., a'_{ss \times qamsimbolos-1}]$$

donde ss es el número de bits por símbolo QAM y qamsimbolos es el número de símbolos QAM. Esta convensión facilita un mapeo en cuadratura, con al menos de dos bits por símbolo QAM y un distanciamiento apropiado entre cada punto de la constelación.

El algoritmo 3.1, es el implementado como primer bloque del transmisor.

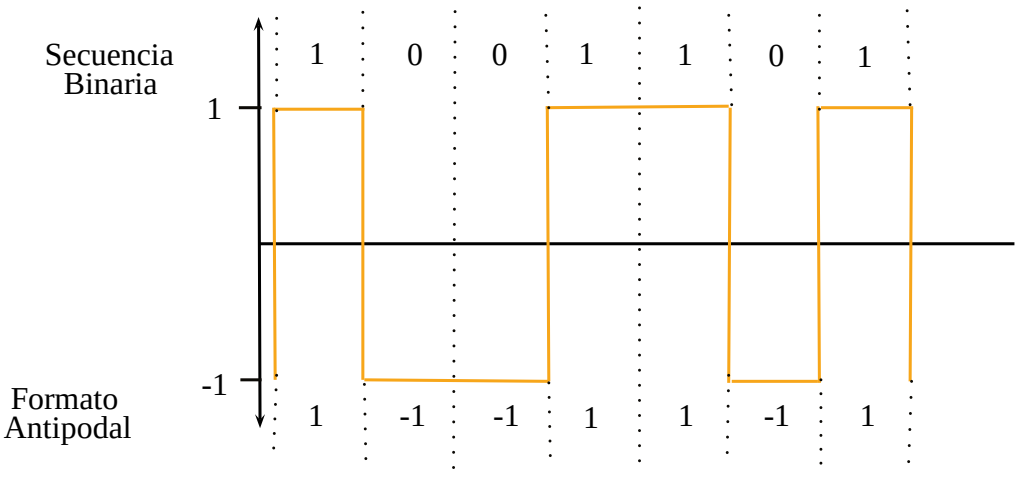


Figura 3.5: Formato antipodal.

Algoritmo 3.1 Secuencia Antipodal
Entrada: Número de bits por símbolos, ss y número de símbolos, qamsimbolos, $nbits = qamsimbolos * ss$
Salida: Vector de secuencia antipodal $\overrightarrow{\mathbf{a}}$.

1: **Para** $i \leftarrow 0$ **hasta** $ss - 1$ **hacer**
2: **Para** $j \leftarrow 0$ **hasta** $nbits - 1$ **hacer**
3 $\overrightarrow{\mathbf{a}} \leftarrow rand()\,\%2$
4: **Si a** $\leftarrow 1$ **Entonces**
5: **a** $\leftarrow 1$
6: **Si no Entonces**
7: **a** $\leftarrow -1$
8: **Fin Para**
9: **Fin Para**
10: **Regresa** $\overrightarrow{\mathbf{a}}$

Mapeo QAM

En este bloque se implementó la modulación QAM. Este bloque recibe la secuencia antipodal, descrita anteriormente, con $qamsimbolos \times ss$ muestras. Por ejemplo, con un valor propuesto de $ss = 2$ cada símbolo QAM esta formado por 2 bits. A la salida de este bloque, se tiene un flujo paralelo de símbolos modulados en *fase (I)* y en *cuadratura (Q)* resultando en una constelación 4QAM.

El vector de salida se construye agrupando en dos columnas, de longitud M, la secuencia antipodal, para formar valores complejos $C_m = a_m + jb_m$ por cada renglon de la doble columna. Entonces, a_m son las primeras M muestras y b_m representan las últimas M muestras de la secuencia entrante. A la salida se tiene un vector complejo, el cual conforma las muestras de la señal analítica. En el algoritmo 3.2 muestra la implementación de este bloque.

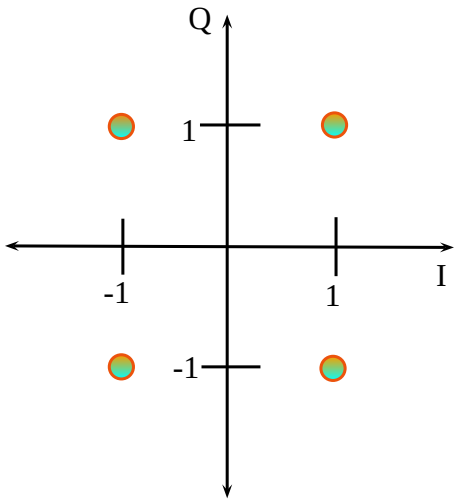


Figura 3.6: Constelación 4QAM.

Algoritmo 3.2 Modulación 4QAM
Entrada: Longitud del vector de símbolos QAM Qamsimbolos, ss y secuencia antipodal, $\overrightarrow{\mathbf{a}}$.
Salida: Señal analitica QAM.(C).

1: **Para** $i \leftarrow 0$ **hasta** $Qamsimbolos - 1$ **hacer**
2: $C_{(i,real)} \leftarrow \overrightarrow{\mathbf{a}}_{antipodal_i}$
3: $C_{(i,imag)} \leftarrow \overrightarrow{\mathbf{a}}_{antipodal_{i+Qamsimbolos}}$
4: **Fin Para**
5:**Regresa** C

IFFT

En este bloque se utiliza la herramienta de FFT que ofrece el software GNU Radio.

Prefijo cíclico

En este bloque se agrega el prefijo cíclico al símbolo OFDM. Sea s_n, la señal de entrada, a la salida se tiene una señal de la forma $\overrightarrow{\mathbf{s'_{n'}}} = [s_{N-D-1}, s_{N-D-2}, ..., S_{N-1}, s_0, s_1, ..., s_{N-1}]$ y una longitud $N + D$. La señal resultante se construye agregando de las últimas D muestras de $\overrightarrow{\mathbf{s_n}}$ al inicio de esta misma. La implementación de este bloque se presenta en el Algoritmo

3.3.

Algoritmo 3.3 Prefijo cíclico
Entrada: Señal de entrada $\overrightarrow{\mathbf{s_n}}$, prefijo D
Salida: Símbolo OFDM con intervalo de guarda $\overrightarrow{\mathbf{s'_{n'}}}$
1: **Para** $n \leftarrow 0$ **hasta** $D-1$ **hacer**
2: $\quad \widehat{s_n} \leftarrow S_{N-D+n}$
3: **Fin Para**
4: **Para** $n \leftarrow 0$ **hasta** $N-1$ **hacer**
5: $\quad \widehat{s}_{n+D} \leftarrow s_n$
6: **Fin Para**
7: **Regresa** s_n

3.2.4. Frecuencia Intermedia Digital(tarjeta madre USRP1)

En la plataforma SDR se incorpora la tecnología Digital IF(siglas del ingl. *Intermediate Frequency*), esta tecnología incorpora soluciones a los problemas que frecuentemente se producen en sistemas heterodinos convencionales, que utilizan componentes analógicos, ocasionando problemas tales como desequilibrio de fase y de ganancia, offset de DC, crosstalk, etc. Por lo tanto, los transceiver con IF digital pueden alcanzar más alto desempeño que los transceiver con IF analógica (que tienen distorsión en la señal por la no linearidad de los dispositivos electrónicos internos).

Esta tecnología puede ser reconfigurable para otros estándares de acceso móvil a través de la correspondiente actualización del firmware en el FPGA. La función principal del transceiver con IF digital incluye la conversión ascendente de frecuencia de una señal de banda base.

Una vez concluido el procesamiento de la señal en banda base, en el FPGA, se realiza el desplazamiento de frecuencia de banda base a una frecuencia intermedia. Cabe mencionar que este proceso es puramente digital, así, la gran importancia y contribuición de la tecnlogía SDR es evitar el uso de hardware analógico para este proceso.

En la figura 3.7, se muestra la estructura interna del hardware digital contenido en un solo chip AD 9862, que realiza el proceso de frecuencia intermedia digital, comúnmente se le conoce como DUC (siglas del ingl. *Digital Upconverter*) y se compone de los siguientes bloques: filtro Hilbert, mezclador en cuadratura fino, interpolador, filtro pasabaja y mezclador en cuadratura grueso y, por último, una etapa de hardware analógico DAC (siglas del ingl. *Digital Analogic Converter*) para la conversión de datos digital-analógico y así enviarlo al *front end*. A continuación describiremos detalladamente cada uno de los bloques, importante mencionar que la tarjeta madre contenida en el USRP1 que se utiliza para la realización de

este proyecto contiene dos chips AD 9862, tal y como se muestra en las figuras 3.7. y 3.8.

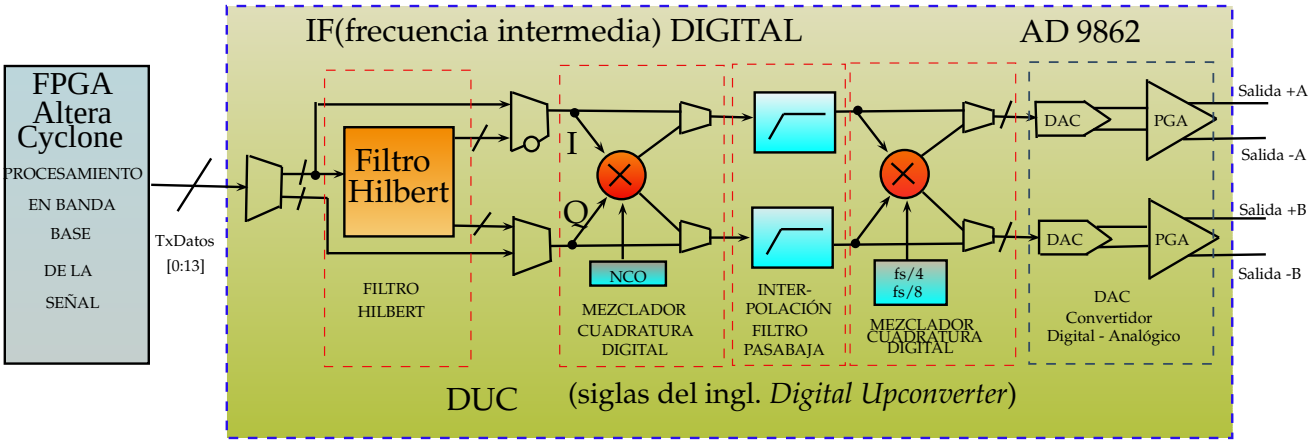


Figura 3.7: Estructura interna del dominio digital del USRP1 en la parte transmisora.

Figura 3.8: Chips AD 9862.

- **Filtro de Hilbert**, forma parte del DUC. Tiene la función de separar las señales, utilizando el crítero de desfase de ±90°. La justificación para utilizar este bloque es la necesidad de separar los datos ya que a la salida del FPGA se obtienen los datos complejos intercalados. Una vez que se realiza el proceso de filtrado Hilbert, los datos son separados y enviados a la entradas de I (In-phase) y Q (In-quadrature), respectivamente, del mezclador.

  El filtro de Hilbert requiere una interpolación 4× (factor 4) y acepta datos a velocidades máximas de 32 Msps.

- **Mezclador en cuadratura fino**, forma parte del DUC. Tiene la función de desplazar la señal de banda base a una mayor frecuencia (frecuencia intermedia). En sus dos entradas en cuadratura (I y Q) se recibe la señal (parte real y compleja) de banda base montada en una frecuencia intermedia. El NCO (siglas del ingl. *Numerically Controlled Oscilator*) es de 24 bits y se usa para utilizar la modulación fina. Se requiere una interpolación de 4 para obtener una mayor tasa de muestreo a partir del sistema de 32 Msps, esto es debido a que la frecuencia del DAC es de 128 MSPS, $\frac{f_{DAC}}{Interpolacion\ \ USRP1} = \frac{128Msps}{4} = 32Msps$.

- **Interpolación y filtro pasabaja**, forma parte del DUC. Tiene la función de incrementar la frecuencia de muestreo por un factor entero L. El objetivo es preservar el contenido espectral de la señal. Añade $L-1$ ceros entre muestras sucesivas de la señal en el dominio del tiempo. Este procedimiento produce una repetición del espectro original dentro de la banda de interés en el dominio de la frecuencia; para evitar este efecto se aplica un filtro pasabaja de frecuencia adecuada para eliminar los duplicados del espectro original. Esta etapa cuenta con 2 interpoladores 2× (factor 2), para construir un interpolador de 4× (factor 4), por tal motivo la interpolación del USRP1 esta en el rango de 4 a 512, en múltiplos de 4.

  El primer filtro interpolador 2× (factor 2) esta constituido de un filtro de 39 taps. Éste suprime señales fuera de banda en 60 dB. La máxima velocidad de datos en la entrada es 64 Msps por canal cuando es utilizado este interpolador. El segundo filtro interpolador 2× (factor 2) es un filtro de 15 taps que también suprime las señales fuera de banda en 60 dB o más. La combinación de ambos interpoladores dan un total de 4× (factor 4) y la máxima velocidad de datos en la entrada por canal es de 32 Msps.

- **Modulador en cuadratura burdo**, forma parte del DUC. Tiene la función de desplazar la señal a altas frecuencias, modificando el espectro de la señal que ingresa a este bloque, donde el cambio de frecuencia esta dado por $\pm\frac{f_{DAC}}{4}$ o $\pm\frac{f_{DAC}}{8}$, siempre que la señal de entrada se componga de datos complejos.

  El modulador en cuadratura burdo puede ser configurado para ejecutar una modulación compleja sólo si la señal de entrada es compleja. Cuando la señal es puramente real, se realiza una modulación real donde el cambio de frecuencia esta dado por $\frac{f_{DAC}}{4}$ o $\frac{f_{DAC}}{8}$.

- **Convertidor Digital-Analógico (DAC)**, es un dispositivo electrónico analógico tiene la función de convertir las formas de onda procesadas digitalmente al dominio

analógico. Se cuenta con dos DAC's duales de 12-14 bits, los cuales soportan velocidades de hasta 128 Msps.

- **PGA (siglas del ingl. *Programmable Gain Amplifier*)**. Tiene la función de proporcionar un intervalo de ganacia de hasta 20 dB para ambos DAC's. La ganacia se controla cambiando las principales corrientes de polarización del DAC.

En la figura 3.9 se muestra la tarjeta madre que conforma al USRP1 de Ettus Research la que, a su vez, se integra por el FPGA y los dos chips AD9862, como se muestra en la estructura interna de la figura 3.7.

En la figura 3.10 se muestra el flujo de datos a través del DUC (dispositivo completamente digital), el DAC (dispositivo totalment analógico) y el PGA (dispositivo completamente analógico). El límite de la frecuencia de muestreo lo proporciona el DAC y a partir de éste se configuran lo parámetros del DUC. El objetivo de la etapa en frecuencia intermedia es realizar la conversión de frecuencia del espectro en banda base, de los datos de entrada, a la frecuencia portadora deseada, a la vez que se asegura que la velocidad de muestreo de los datos de entrada es igual a la velocidad muestreo de la señal portadora. Sin embargo, hay que hacer notar que el límite de la velocidad de muestreo lo proporciona el DAC, por tal motivo, la señal de datos de entrada debe pasar por un proceso adicional de modulación en cuadratura (dentro del front end) y por la amplificación necesaria para poder montar la señal de datos de entrada sobre la frecuencia de la portadora deseada.

3.2.5. Conversión de Frecuencia Intermedia a Radio-frecuencia (tarjeta hija USRP1)

Para realizar este proceso es necesario utilizar dispositivos electrónicos analógicos que permitan montar la señal de datos en la portadora deseada de radiofrecuencia.

La frecuencia portadora a la que vamos a transmitir inalámbricamente es de 900 Mhz. En el capítulo 2.4 (2.4.2 Tarjeta hija), se hace una introducción a la tarjeta hija (daughteboard) RFX900 de Ettus Research compatible con el equipo USRP1.

Esta etapa conocida como transmisor de conversión directa esta conformada por: modulador en cuadratura, amplificador, y amplificador de potencia, como se muestra en la figura 3.11. A continuación detallaremos cada uno de estos bloques, que hacen posible la realización de este proyecto.

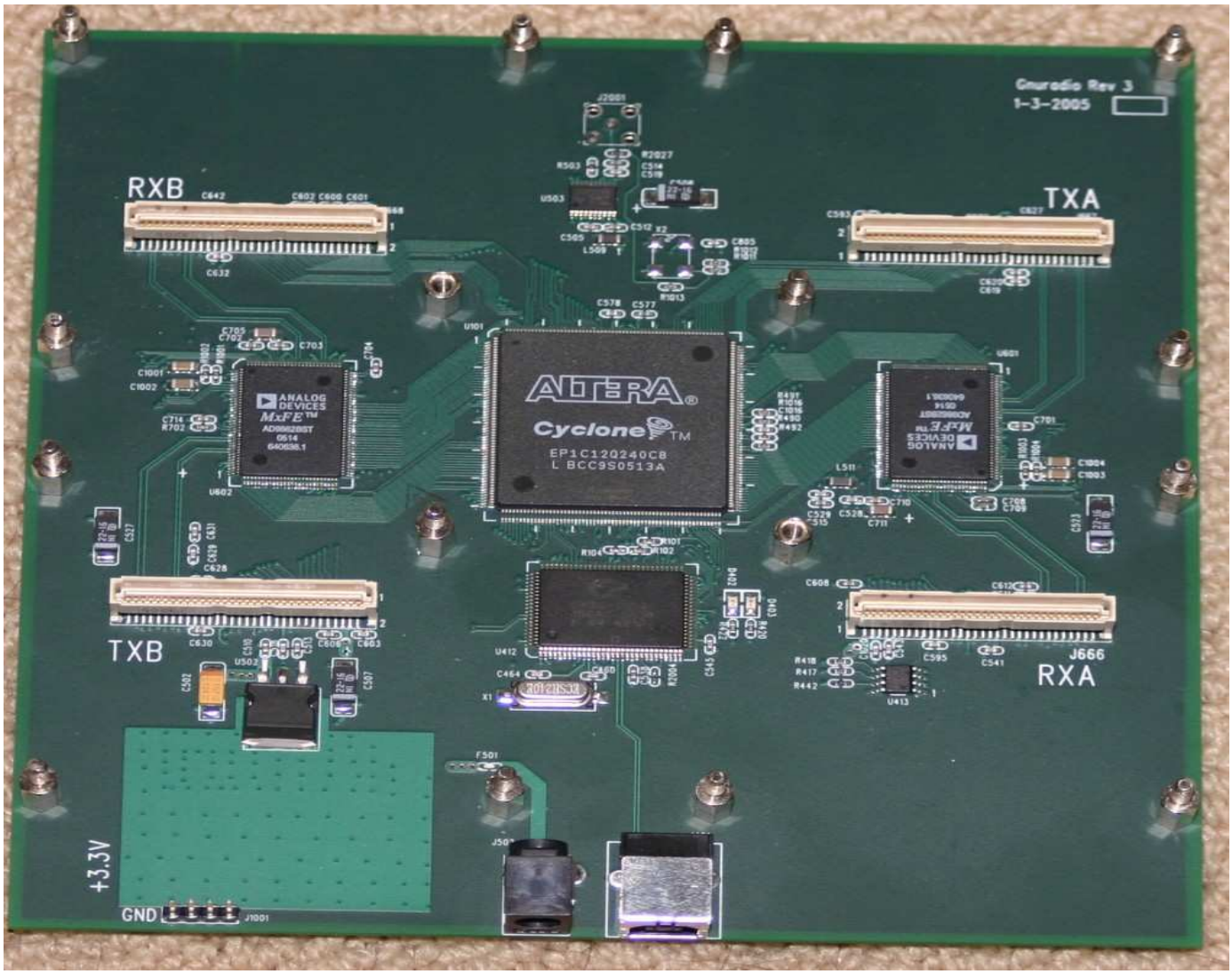


Figura 3.9: Tarjeta electrónica (motherboard del USRP1) correspondiente a la figura 3.7.

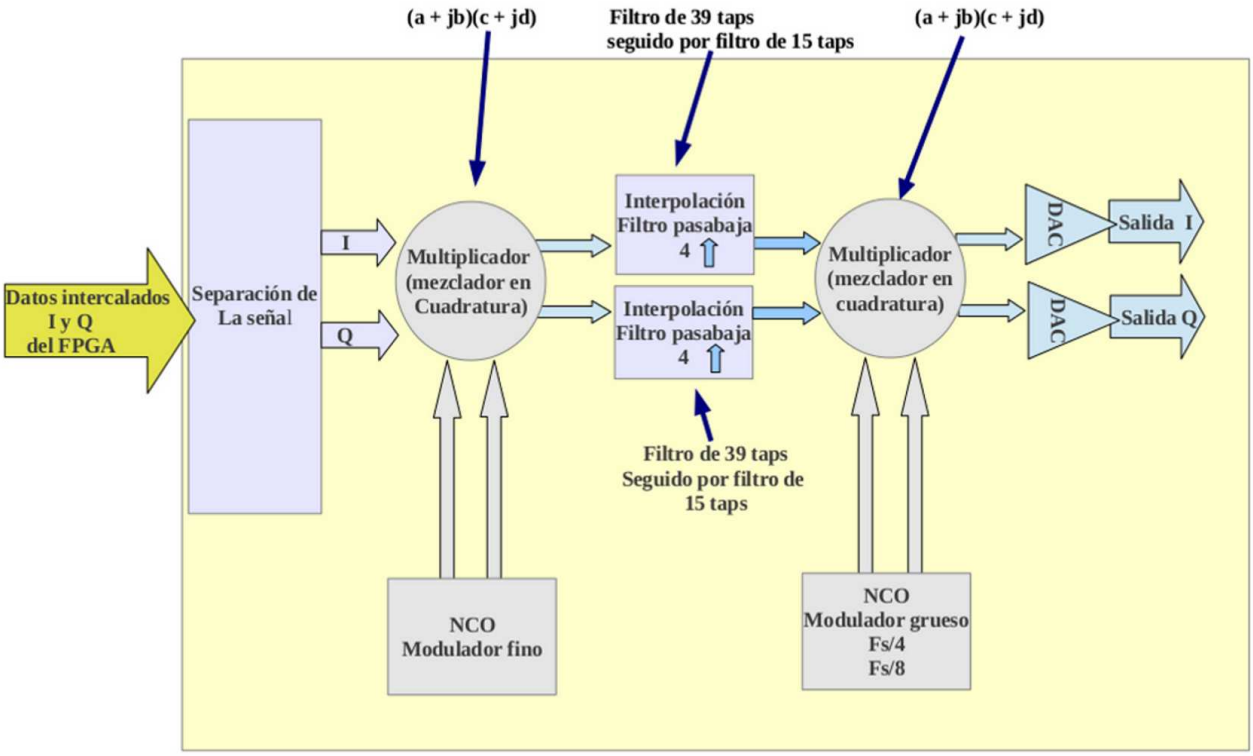


Figura 3.10: Flujo de datos a través del dominio digital USRP1 (DUC).

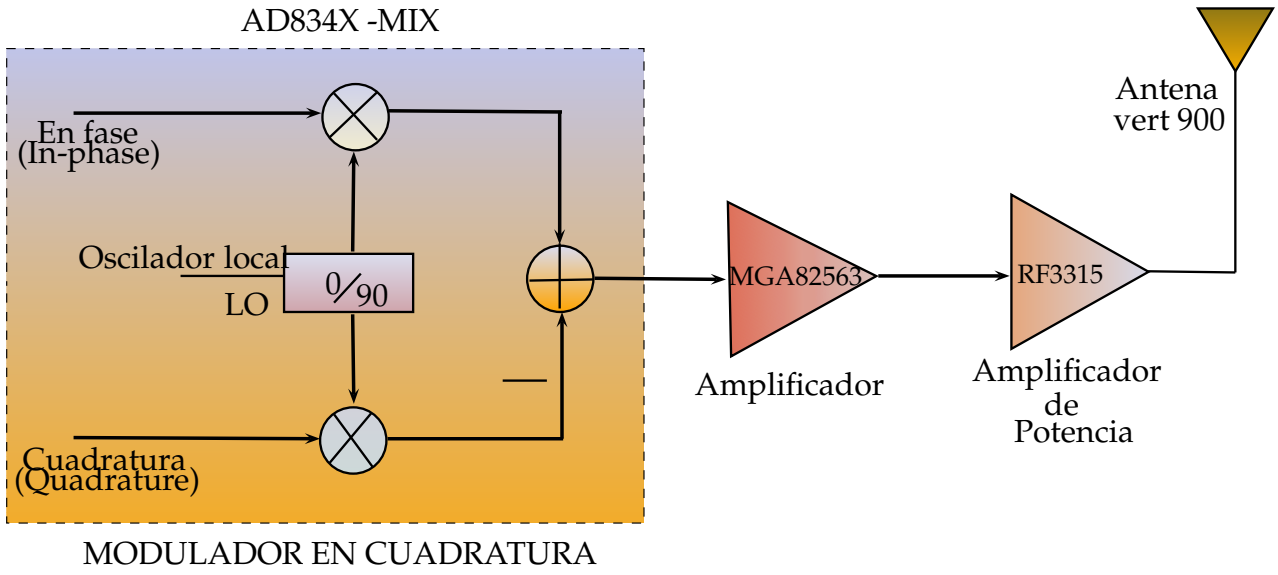


Figura 3.11: Diagrama de la tarjeta hija transmisora RFX 900, TX.

- **Modulador en cuadratura**. Tiene la función de modular con la señal de banda base (pasabaja) a la señal RF (pasabanda). El chip incluido en el USRP1 es el AD8349, que permite modular la señal de datos a 900 Mhz.

- **Amplificador MGA82563**. Tiene la función de amplificar la señal entregada por el modulador de cuadratura como, se muestra en la figura 3.11.

  Este amplificador proporciona una potencia de salida 17.3 dBm (53mw), P_{1dB}, a 2 Ghz (donde el amplificador presenta efectos no lineales mínimos o nulos) y una potencia de salida de 20 dBm (100mw), P_{sat}, a 2 Ghz.

- **Amplificador de potencia RF3315 (Amplificador de alta linearidad de banda ancha)**. Tiene la función de amplificar la señal de banda ancha (señal modulada en RF).

  Este amplificador proporciona una ganancia de 18 dB a 900 Mhz (18 veces la potencia de la señal) y una potencia de salida de 23 dBm (200mw), P_{1dB}, a 900 Mhz.

La tarjeta hija RFX900, mostrada en la figura 3.12, tiene una potencia de salida de 23 dBm (200mw) para el intervalo de frecuencia de 750 a 1050 Mhz.

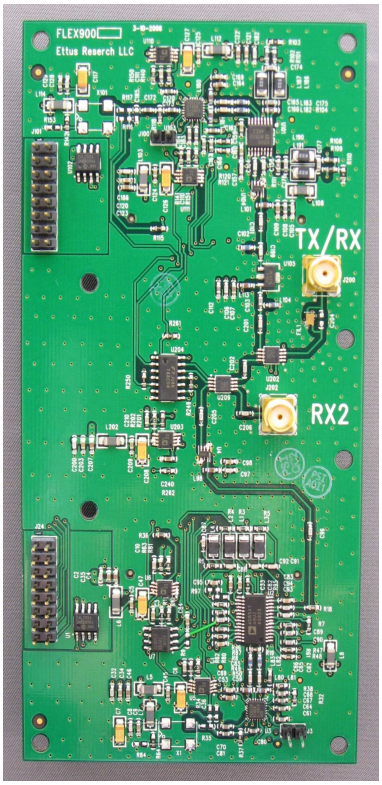


Figura 3.12: Tarjeta hija transmisora RFX 900, TX.

3.2.6. Antena del USRP1

En la figura 3.13 se muestra la antena VERT900 utilizada con el USRP1 durante la realización de este proyecto de investigación. Tiene las siguientes características:
Es una antena ominidireccional con ganancia de 3dBi, con operación en la banda de frecuencia de 824-960 Mhz. Es la antena adecuada para transmitir la señal de datos al medio inalámbrico a una frecuencia de 900 Mhz.

Figura 3.13: Antena VERT900 de 900 Mhz.

3.2.7. Realización de un transmisor OFDM con una plataforma SDR

En la figura 3.14 se muestra la plataforma SDR que es utilizada para la implementación de los algoritmos del nivel físico de un radio OFDM verificandolos y validándolos en la misma.

- **Sección en banda base**. En esta sección se programan los algoritmos de nivel físico OFDM mediante el software GNU Radio, para ser cargados en el FPGA, y realizar el procesamiento de la señal en banda base.

- **Sección IF (Frecuencia Intermedia)**. En esta sección se realiza el procesamiento de la señal que traslada la frecuencia de la señal de banda base a una señal de frecuencia

intermedia (bajas frecuencias). Lo anterior se lleva a cabo mediante un bloque puramente digital (DUC) y otro en hardware (DAC).

- **Sección RF (Radiofrecuencia)**. En esta sección se realiza la conversión de frecuencia intermedia a radiofrecuencia, utilizando hardware analógico (modulador en cuadratura y amplificadores).

- **Antena**. Convierte los voltajes de la señal transmitida a ondas electromagnéticas para transmitir la señal por un medio inalámbrico.

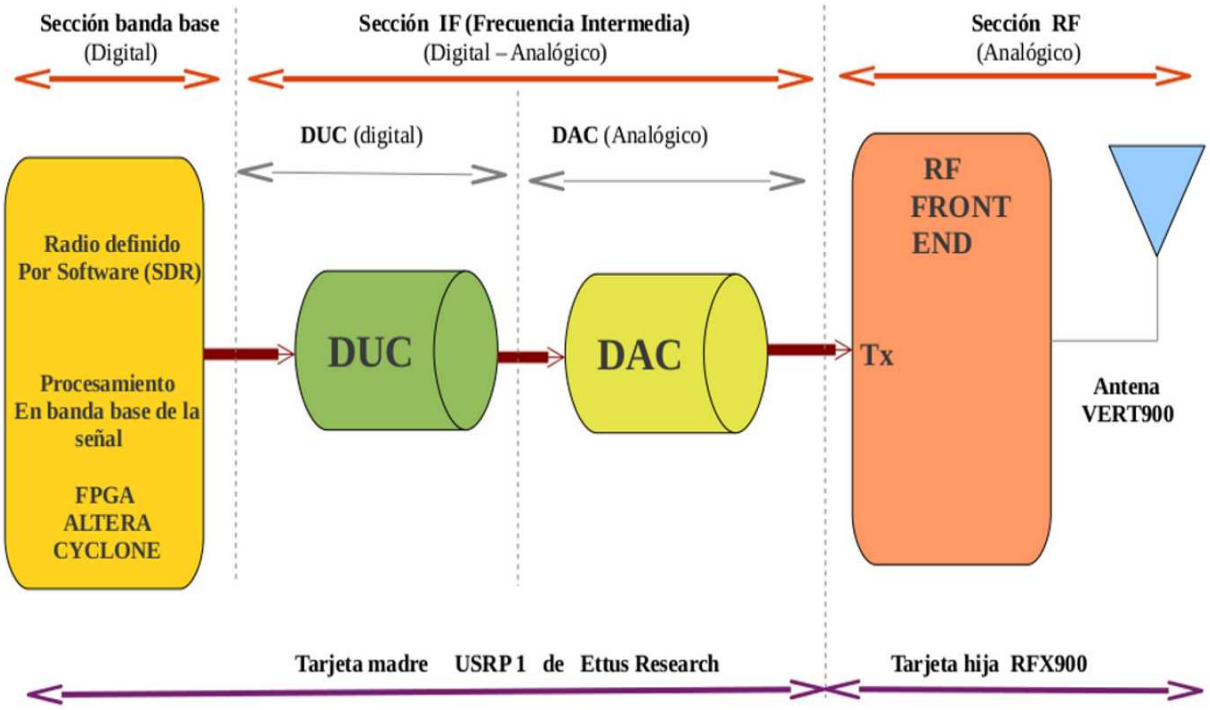


Figura 3.14: Transmisor OFDM realizado en una plataforma SDR.

3.2.8. Realización de un receptor OFDM con una plataforma SDR

En esta etapa del sistema de comunicación digital el papel del módulo de sincronización es fundamental. Debido a ello, en este trabajo presentaremos especial atención a este módulo.

Sincronización burda (sincronización en tiempo)

Consiste en la localización del inicio de cada símbolo OFDM de la trama de datos recibidos. En este trabajo se tomó como punto de partida el método de correlación de prefijo cíclico propuesto en los artículos [2] y [26]. La idea principal se muestra en un diagrama a bloques como el de la figura 3.15.

La figura 3.15 muestra el diagrama a bloques, correspondiente a la ecuación de estimación ML (siglas del ingl. *Maximum likelihood*) [2], dado por $\lambda(\theta)$.

$$\lambda(\theta) = 2\left|\sum_{k=\theta}^{\theta+L-1} r(k)r^*(k+N)\right| - \rho\sum_{k=\theta}^{\theta+L-1}\left(|r(k)|^2 + |r(k+N)|^2\right) \tag{3.1}$$

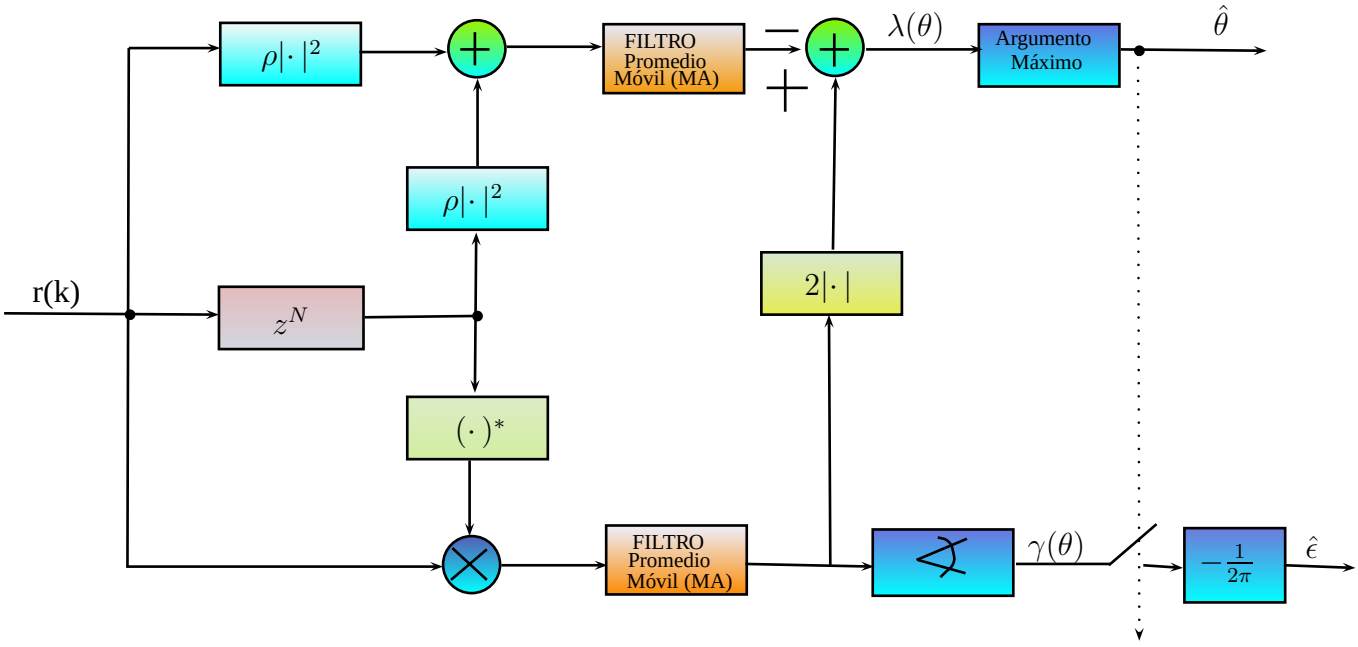


Figura 3.15: Diagrama a bloques para la sincronización en tiempo [2].

Donde $r(k)$ son los datos recibidos.

Operador +, primera rama

El bloque $\rho|\cdot|^2$, corresponde $\rho|r(k)|^2$.
El bloque z^N y $\rho|\cdot|^2$, corresponde $\rho|r(k+N)|^2$.
El operador +, realiza la suma de los términos anteriores, obteniendo $\rho|r(k)|^2 + \rho|r(k+N)|^2$, entra al bloque de filtro MA (siglas del ingl. *Moving Average*) para reducir el ruido aleatorio en el dominio del tiempo y, se obtiene.

$$\rho \sum_{k=\theta}^{\theta+L-1} (|r(k)|^2 + |r(k+N)|^2) \tag{3.2}$$

Operador $\times$, primera rama
El bloque z^N y $(\cdot)^*$, corresponde $r^*(k+N)$.
La salida del operador $\times$ entra al filtro MA y, pasa por el bloque $2|\cdot|$ y, se obtiene.

$$2|\sum_{k=\theta}^{\theta+L-1} r(k)r^*(k+N)| \tag{3.3}$$

Operador +, segunda rama
Se suman los términos obtenidos.

$$2|\sum_{k=\theta}^{\theta+L-1} r(k)r^*(k+N)| - \rho \sum_{k=\theta}^{\theta+L-1} (|r(k)|^2 + |r(k+N)|^2) \tag{3.4}$$

La ecuación 3.4, es el resultado del diagrama de bloques de la figura 3.15, que corresponde a la ecuación 3.1[2]. El diagrama se implementó en el software GNU radio para obtener el inicio del símbolo OFDM.

Sincronización fina (sincronización en frecuencia)

Es la etapa más crítica de un sistema de radio inalámbrico. Los errores en frecuencia se originan por diferencia entre los osciladores del transmisor y receptor, efecto Doppler y ruido de fase introducido por el canal.

Efectos destructivos causados por un offset en la portadora en sistemas OFDM.

- Reducción de la amplitud de la señal. Las funciones sinc tiene un desplazamiento y no vuelven a ser muestreadas en su máximo lo que implica un retraso o adelanto.
- Introducción de ICI en las portadoras.

- La componente de señal útil se rota y atenúa.

Puntos a tomar en cuenta en la sincronización fina

Sincronización de reloj de muestreo. En sistemas prácticos la frecuencia del reloj de muestreo del receptor es ligeramente diferente al transmisor.
Sincronización en frecuencia. Error en frecuencia del oscilador local (LO) del modulador en cuadratura.

La sincronización fina (frecuencia), tiene como objetivo la restauración de ortogonalidad para compensar cualquier offset de frecuencia por osciladores imprecisos.

CFO (siglas del ingl. *Carrier Frequency Offset*)

La forma de onda entrante al receptor $r_{RF}(t)$ es filtrada y puesta en banda base mediante dos sinusoides en cuadratura por medio del oscilador local (LO). La señal en banda base pasa por el ADC (convertidor analógico-digital) y es muestreada con una frecuencia $f_s = \frac{1}{T_s}$.

Debido a la inestabilidad del oscilador y efecto Dopler, la frecuencia f_{LO} de la FFT no es igual a la frecuencia de portadora de la señal recibida f_c. La diferencia da origen al CFO $f_{CFO} = f_c - f_{LO}$. lo cual origina un corrimento en fase.

Parámetros que deben controlarse para compensar el CFO

- Ajustar la posición de la ventana FFT.
- Rastrear el reloj de muestro del ADC.

La sincronización en tiempo detecta el inicio del símbolo OFDM recibido y es utilizado para encontrar la posición correcta de la ventana de FFT.

La estimacion de la frecuencia de muestreo es a través de símbolos pilotos, después de la FFT, alimentar el VCO del DAC para ajustar la frecuencia de muestreo y así compensar el offset de frecuencia.

En un ambiente práctico y real, la estimación de CFO depende del hardware, modulador en cuadratura y ADC. Sin embargo, la teoría muchas veces no contempla los efectos reales del canal y las características físicas del hardware, parámetros importantes, para obtener un buen desempeño en la sincronización fina. La solución propuesta en la práctica, es el método empírico, prueba y error y, una buena observación del espectro de la señal recibida para estimar el CFO.

Ecualización

Solución al desvanecimiento selectivo en el dominio de la frecuencia. La ecualización, permite escalar las portadoras según la atenuación introducida por el canal.

Estimación de canal

Para escalar las subportadoras, conocer las atenuaciones a través de la estimación de canal.

Capítulo 4
Pruebas

En este capítulo se presentan los resultados de las pruebas realizadas al sistema de comunicación digital descrito en el capítulo anterior. En esencia se trata de corroborar el funcionamiento del sistema.

Los resultados obtenidos son comparados con las simulaciones realizadas en matlab y con la teoría del estado del arte referente al proyecto de investigación.

Este capítulo se compone de la siguiente manera:

- **En la primera parte, se presentan los resultados de la verificación de cada uno de los bloques que conforman los algoritmos de nivel físico de un radio OFDM usando la plataforma SDR y el software GNU Radio.**
- **En la segunda parte, se presentan los resultados de los bloques del sistema de comunicación digital presentado en el capítulo anterior, desarrollados para una arquitectura SDR mediante la herramienta gráfica denominado como GNU Radio Companion.**
- **En la tercera parte, se presentan los resultados de la validación de los algoritmos del nivel físico de un radio OFDM empleando todos los recursos disponibles, es decir, software SDR GNU Radio y el Hardware SDR USRP1 con el módulo RFX900. Todo lo anterior con el fin de probar la propuesta en un canal de comunicación inalámbrico.**

Tabla 4.1: Parámetros de banda base para el radio OFDM en GNU Radio.

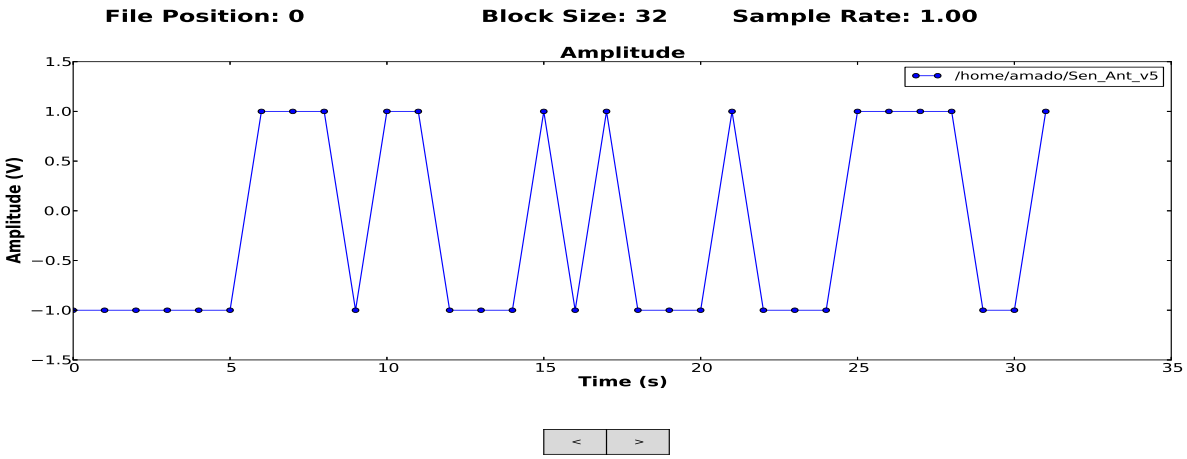
Subportadoras	16
Longitud de la FFT	16
Prefijo cíclico	1/4
Modulación	4-QAM

Se toma los datos de la tabla 4.1 para realizar las pruebas de cada bloque.

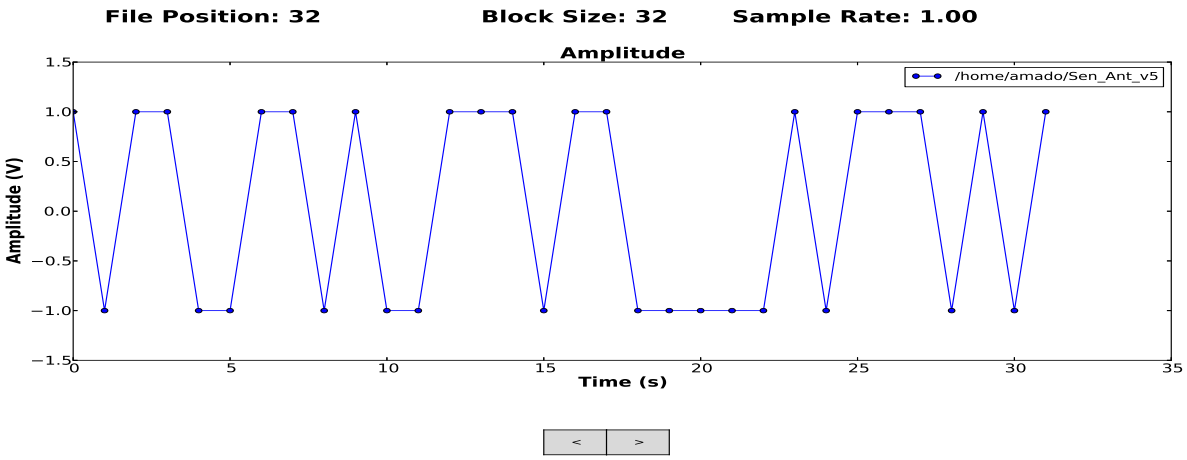
4.0.9. Resultados de los bloques individuales en el Software SDR GNU Radio

4.0.10. Generación de datos

Para ejemplo, aquí presentamos 2 secuencias del primer bloque del sistema de comunicación digital de la señal antipodal aleatoria programada en C++ (la primera capa del software SDR). En la figura 4.1 se muestran las 2 secuencias aleatorias con distribuición uniforme.



(a)

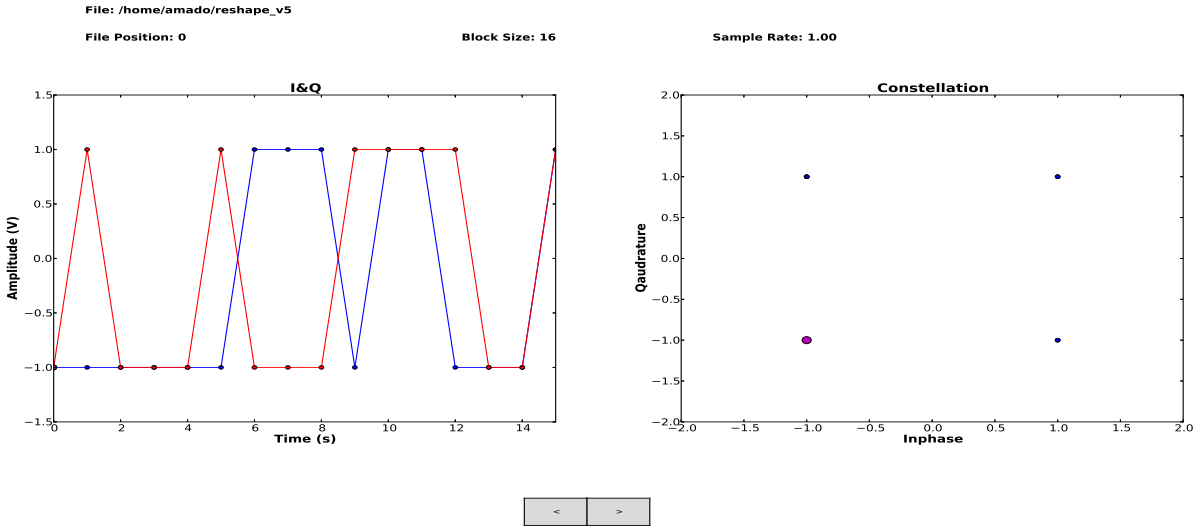


(b)

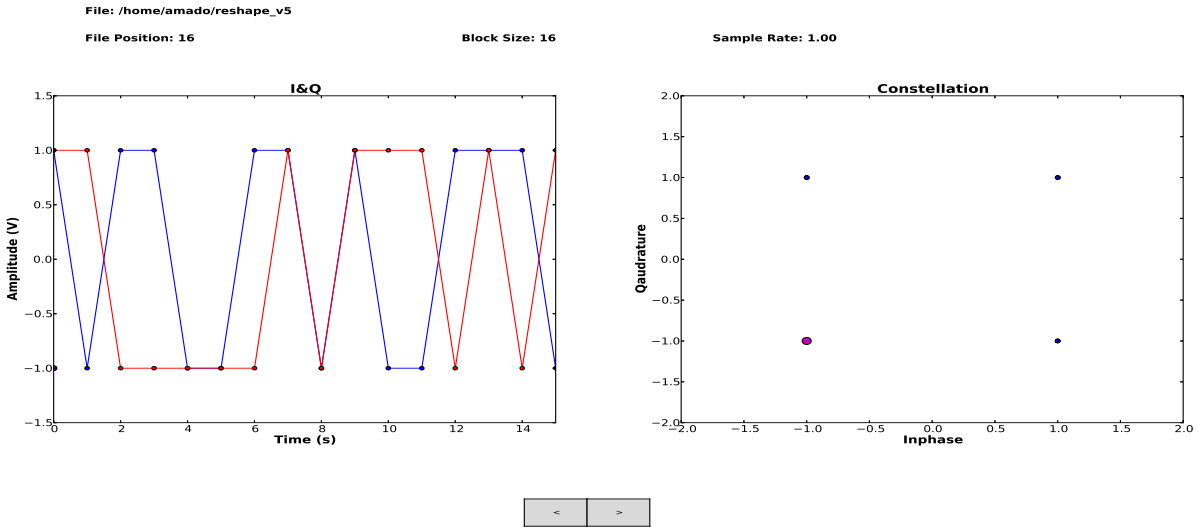
Figura 4.1: a) Señal antipodal de la secuencia 1, b) Señal antipodal de la secuencia 2.

4.0.11. Modulación digital 4 QAM

Aquí se muestran 2 pruebas del segundo bloque del sistema de comunicación digital. En la figura 4.2 se muestra la modulación 4 QAM con sus respectivas constelaciones. Se demuestra que realmente se obtienen las 4 constelaciones correspondientes a este tipo de modulación. A la salida de este bloque se tiene una secuencia compleja que es la representación discreta de la señal analítica.



(a)

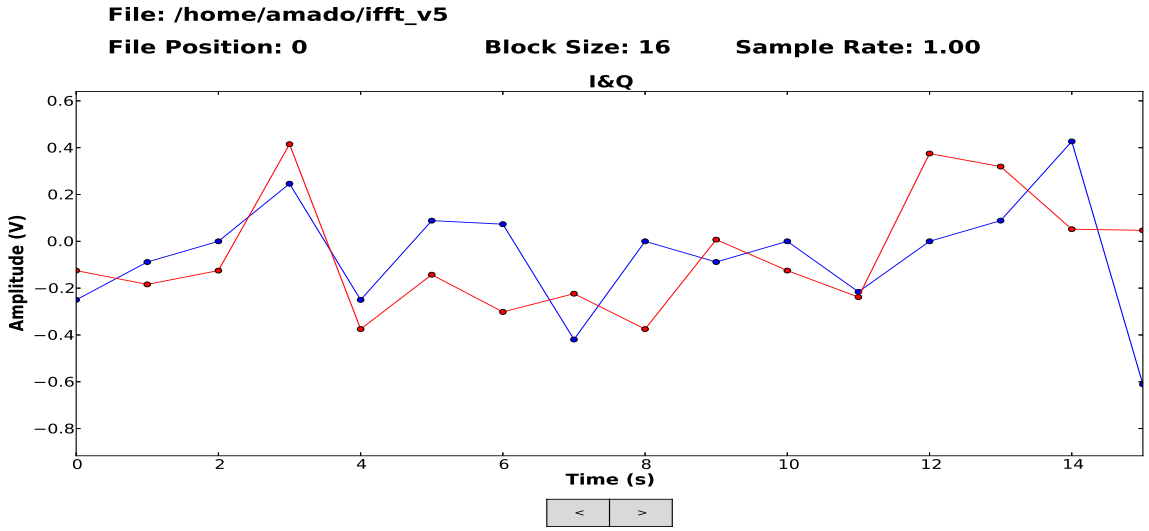


(b)

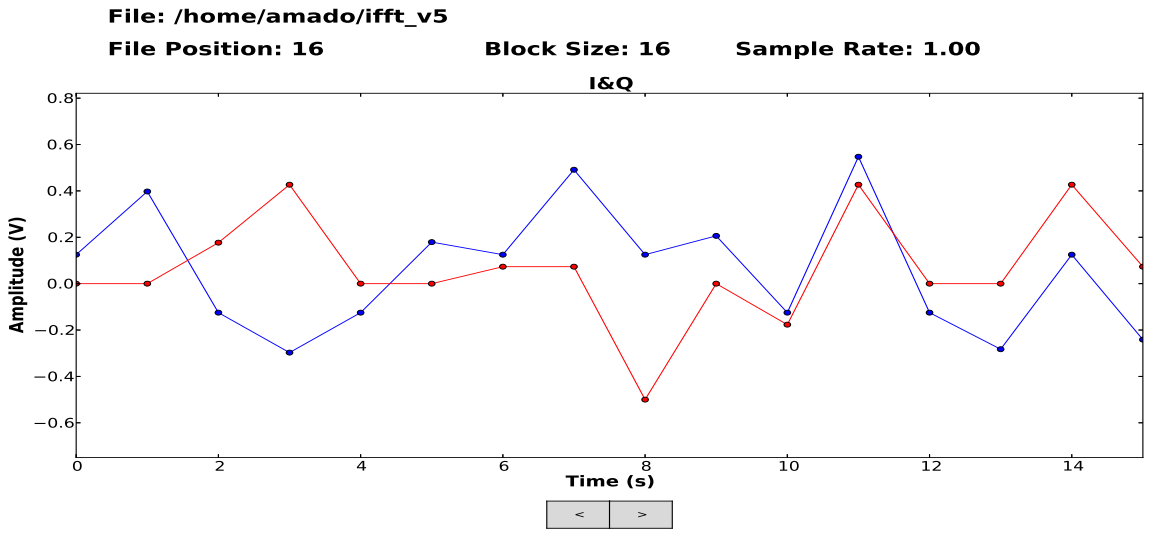
Figura 4.2: a) Modulación 4QAM prueba 1, b) Modulación 4 QAM prueba 2.

4.0.12. Modulación OFDM

Para ejemplo, aquí se muestran 2 secuencias de salida del tercer bloque del sistema de comunicación digital. En la figura 4.3 se muestra el resultado de la Transformada Inversa de Fourier de una señal analítica. Obteniendo así un símbolo OFDM.



(a)

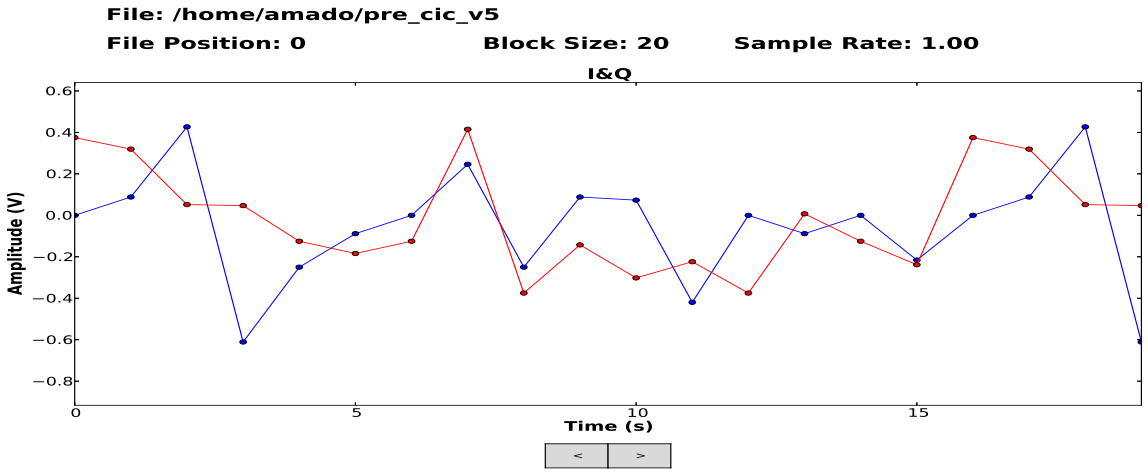


(b)

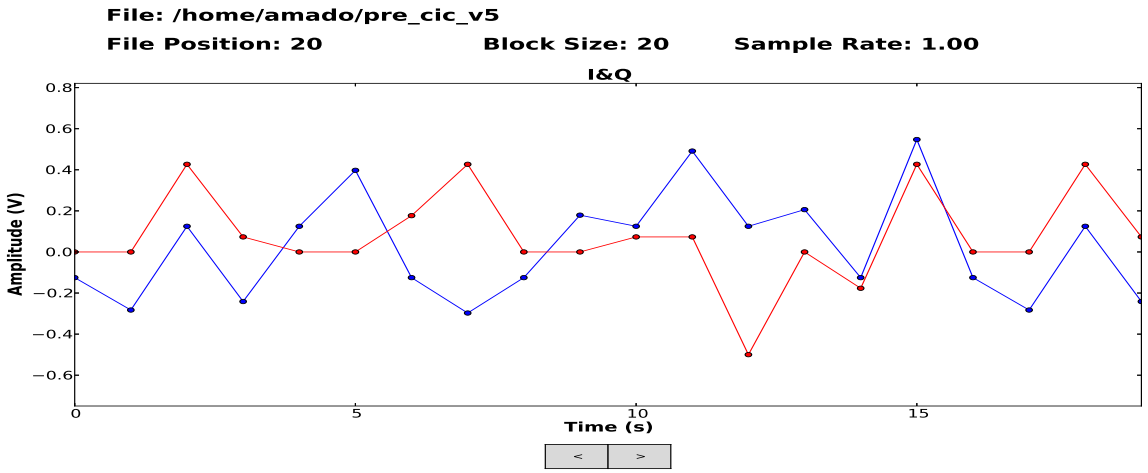
Figura 4.3: a) Símbolo OFDM prueba 1, b) Símbolo OFDM prueba 2.

4.0.13. Adición del prefijo cíclico

Aquí se muestran 2 pruebas del cuarto bloque del sistema de comunicación digital. En la figura 4.4 se muestra la inserción del prefijo cíclico, donde el símbolo OFDM original es precedido por una extensión periódica de la misma señal (las muestras finales del mismo símbolo). En este caso el prefijo cíclico se compone por 1/4 de la secuencia final del símbolo original, es decir, en este ejemplo, equivale a 4 muestras. Así, la duración final del símbolo en muestras queda, para este ejemplo, como $T_s = T_u + T_g = 16 + 4 = 20$.



(a)



(b)

Figura 4.4: a) Símbolo OFDM con prefijo cíclico prueba 1, b) Símbolo OFDM con prefijo cíclico prueba 2.

4.0.14. Realización del radio OFDM de banda base mediante la interfaz gráfica GNU Radio Companion

En esta etapa se muestra el resultado de la programación de los algoritmos de un nivel físico de un radio OFDM para comunicación digital en una plataforma SDR mediante el software GNU Radio Companion (interfaz gráfica).

Recordemos que el desarrollo de aplicaciones con la arquitectura de GNU Radio involucran tres lenguajes de programación: lenguaje de programación C++, lenguaje de programación XML y lenguaje de programación PYTHON la poca y dispersa información existente en la literatura fue un reto importante para la realización de este proyecto de investigación. En la figura 4.5, se muestra el diseño de los algoritmos de nivel físico de un radio OFDM de comunicación digital propuesto en el capítulo anterior utilizando la plataforma SDR.

Es importante mencionar que se utiliza el módulo de FFT que proporciona GNU Radio Companion que con la instrucción **Reverse** ofrece también la IFFT.

4.0.15. Radio OFDM en radio frecuencia, utilizando un USRP1 y el bloque de portadora modulada del software GNU Radio

Los parámetros utilizados en el enlace comunicación inalámbrico en la plataforma SDR, se muestra en la tabla 4.2

Tabla 4.2: Parámetros del enlace de comunicación inalámbrica a través del hardware USRP1.

Subportadoras	512
Longitud de la FFT	512
Prefijo cíclico	1/4
Modulación	4-QAM
Frecuencia portadora	900 Mhz
Sample rate	250 kbps
Interporlación	512
Número de serie, USRP1 Transmisor	2R2BV7U1
Número de serie, USRP1 Receptor	3R2BVCU1
Tarjeta hija transmisora	RFX900
Tarjeta hija receptora	RFX900
Antena	VERT900
GNU Radio software	V3.6.2-165-g315237fd

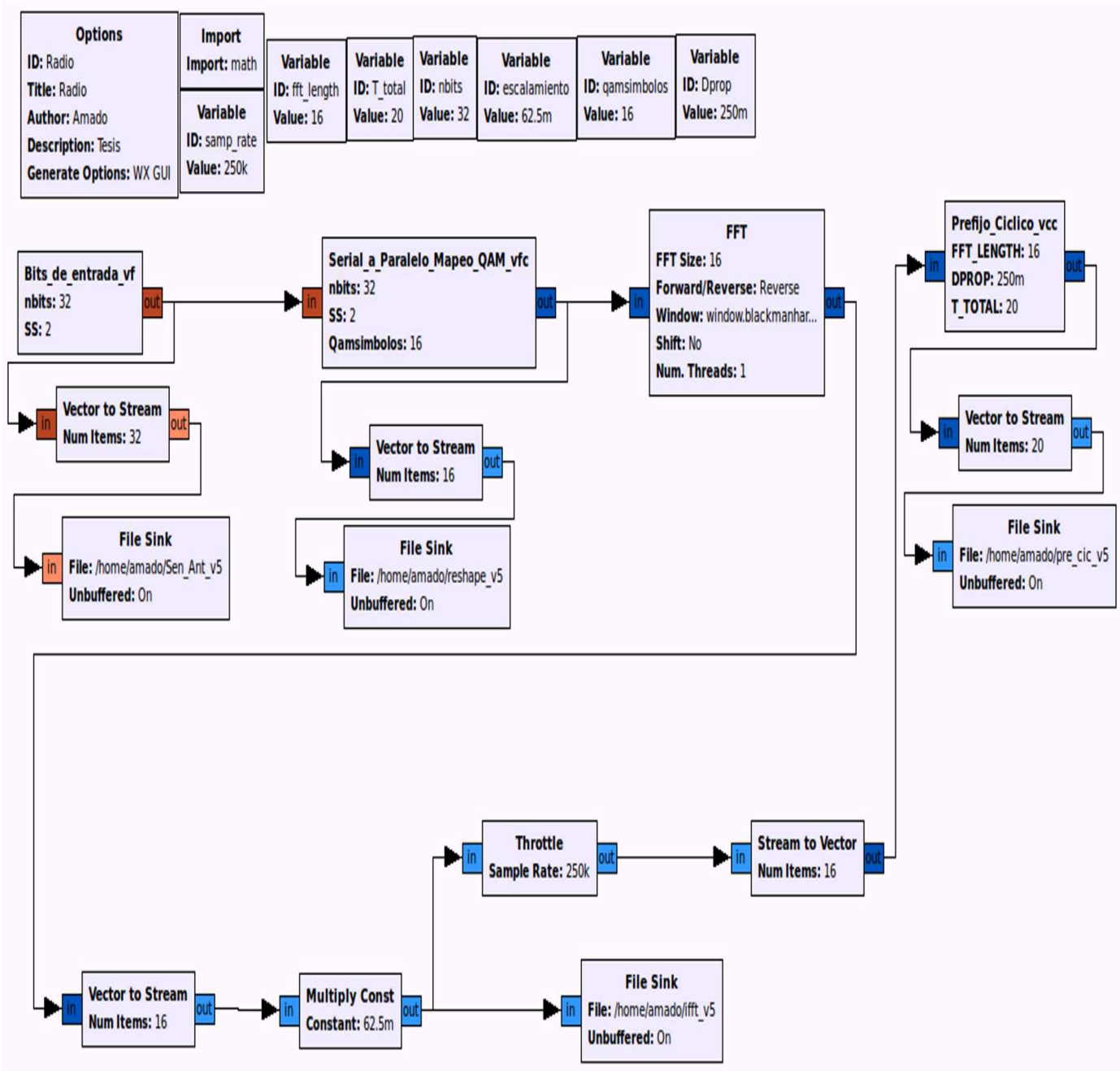


Figura 4.5: Bloque de nivel físico de un radio OFDM en banda base en la plataforma SDR (GNU Radio Companion).

Ahora, presentaremos los resultados de una demostración funcional respecto de la información enviada a través del canal inalámbrico por medio de un USRP1 como transmisor y otro como receptor. En particular, se muestra como analizar la señal transmitida con un simple analizador de espectros en el USRP1 receptor.

En la figura 4.6, se muestran los bloques de nivel físico de un radio OFDM realizado dentro del software SDR GNU Radio Companion, utilizando un esquema de portadora modulada. Con el bloque UHD: USRP Sink se monta la señal de datos en una portadora de 900 Mhz.

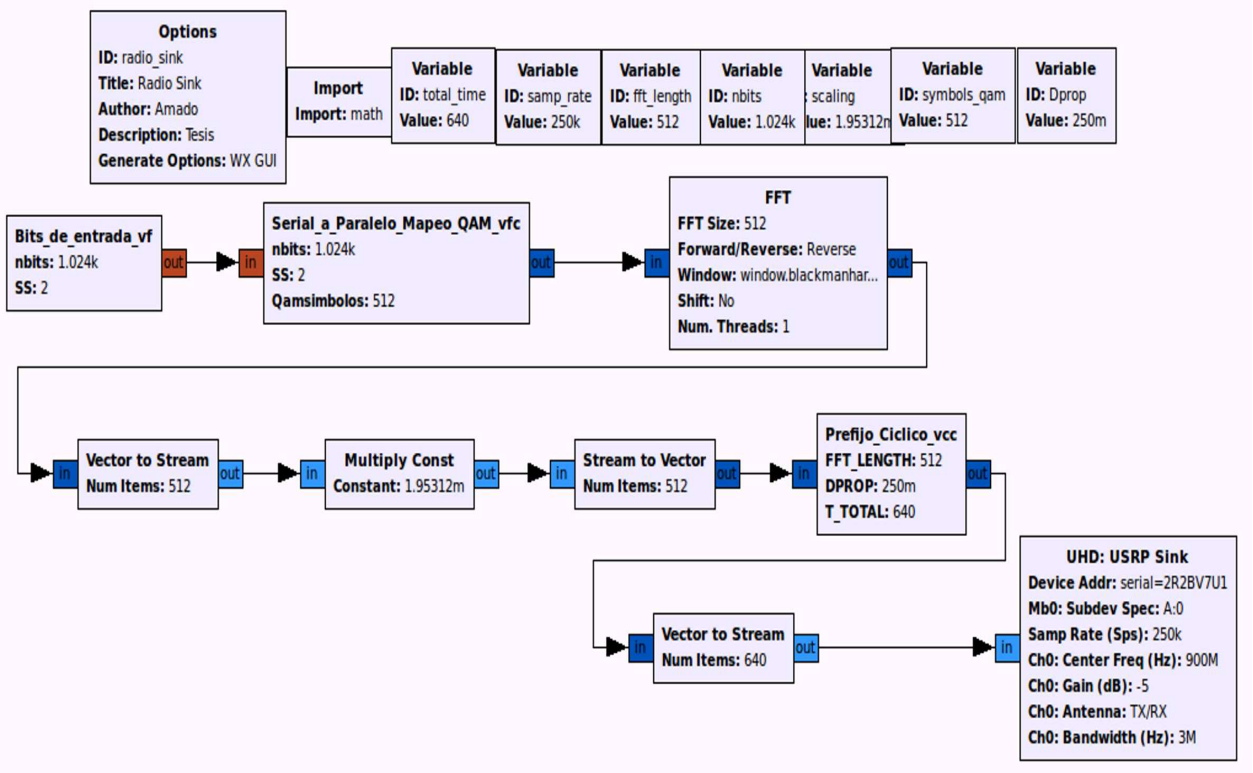
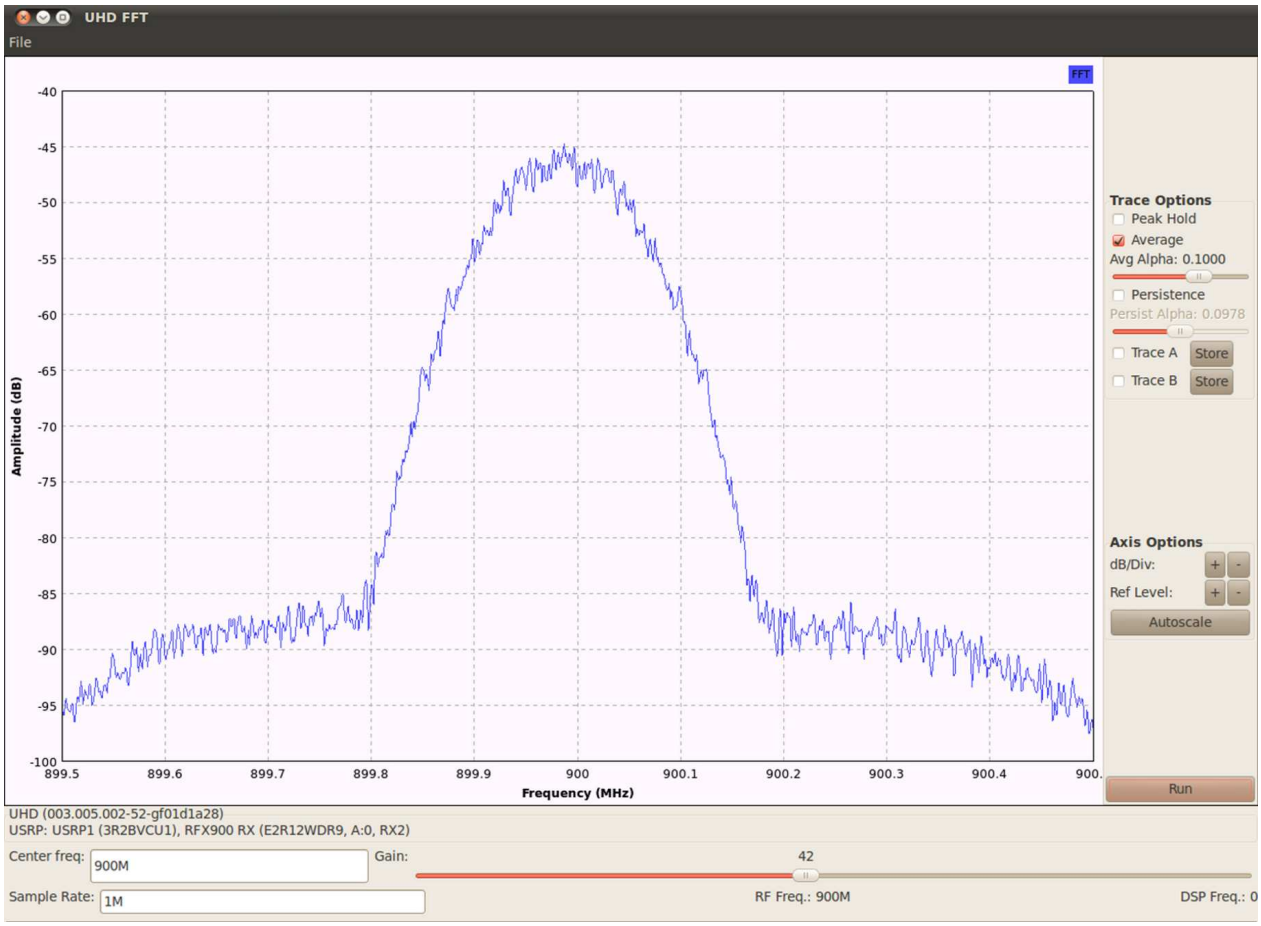


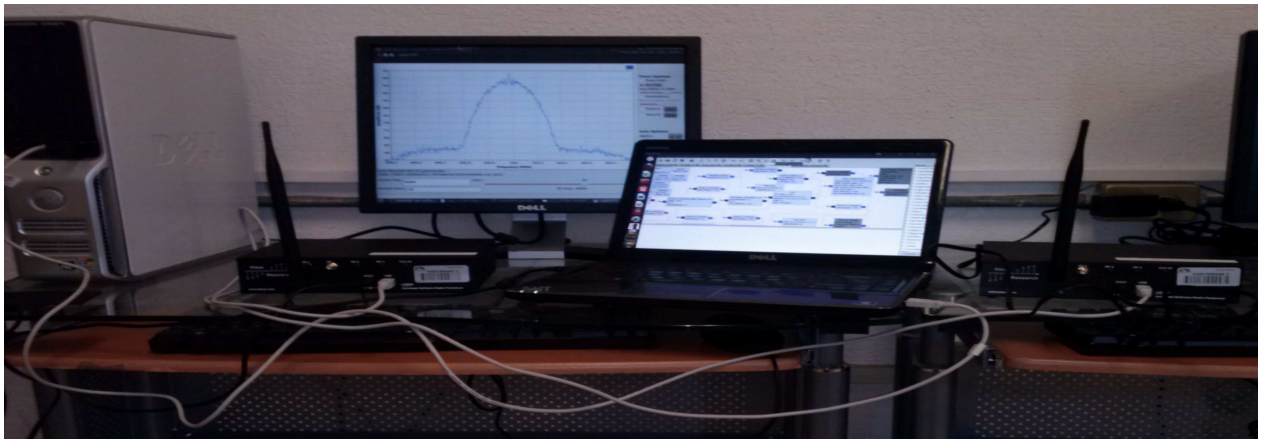
Figura 4.6: Bloques para la realización de nivel físico de un radio OFDM con portadora modulada en GNU Radio.

El espectro de la señal transmitida como se muestra en la figura 4.7 a que corresponde a lo reportado. En esta figura se observa que se tiene un ligero offset de frecuencia, porque la portadora no coincide exactamente con el centro del espectro, pero se cumple con el objetivo de validación de los algoritmos de nivel físico del radio OFDM en un plataforma SDR.

En la figura 4.7 b se muestra el sistema completo implementado con la plataforma SDR software SDR GNU Radio Companion (en la computadora portatil) el hardware SDR USRP1, las tarjetas hijas RFX900 y, las antenas, VERT900.



(a)



(b)

Figura 4.7: a) Espectro OFDM en el analizador de espectro utilizando software y hardware SDR, b) Banco de pruebas para la validación de los bloques de los algoritmos de nivel físico de un radio OFDM para sistema de comunicación digital en un canal de comunicación inalámbrica con una plataforma SDR.

La frecuencia más alta contenida en la señal analógica es $F_{max} = B$, F_{max} la frecuencia máxima y B el ancho de banda. Por lo tanto, se tiene:

$$F_s > 2F_{max} \equiv F_s > 2B \tag{4.1}$$

F_s es la frecuencia de muestreo (*sample rate*). El ancho de banda de la señal analógica se, obtiene despejando B de la ecuación 4.1 (teorema de muestreo). Por lo tanto, se tiene:

$$B < \frac{F_s}{2} = \frac{250\ Khz}{2} = 125\ Khz$$

En la figura 4.7 a) obtenida se, muestra que la frecuencia máxima es 125 Khz (900.125 Mhz). Con esto se comprueba que:

$$F_s(sample\ rate) = 2(125\ Khz) = 250\ Khz.$$

Por lo tanto, el ancho de banda analógico es de 125 Khz.

El ancho de banda digital es 250 Khz, lo que implica, que la separación de las subportadoras Δ_f, es:

$$\Delta_f = \frac{sample\ rate}{subportadoras} = \frac{250\ Khz}{512} = 488\ Hz$$

$$T_b = \frac{1}{\Delta_f} = 2\ ms$$

Para el, tiempo de guarda, es de $\frac{1}{4}$, se tiene:

$$\Delta_{f_{TG}} = \frac{250\ Khz}{128} = 2\ Khz$$

$$T_{b_{TG}} = \frac{1}{\Delta_{f_{TG}}} = 0{,}5\ ms$$

Tiempo de cada subportadora:

$$T_{subportadora} = T_b + T_G = 2\ ms + 0{,}5\ ms = 2{,}5\ ms$$

La frecuencia de muestro de 250 Khz y 32 Mhz, esta dada por la interpolación de 512 y 4 respectivamente.

4.0.16. Sincronización burda (en tiempo)

En la figura 4.8 se muestra la realización del sistema de sincronización burda (en tiempo), propuesto en [2][27], la plataforma GNU Radio.

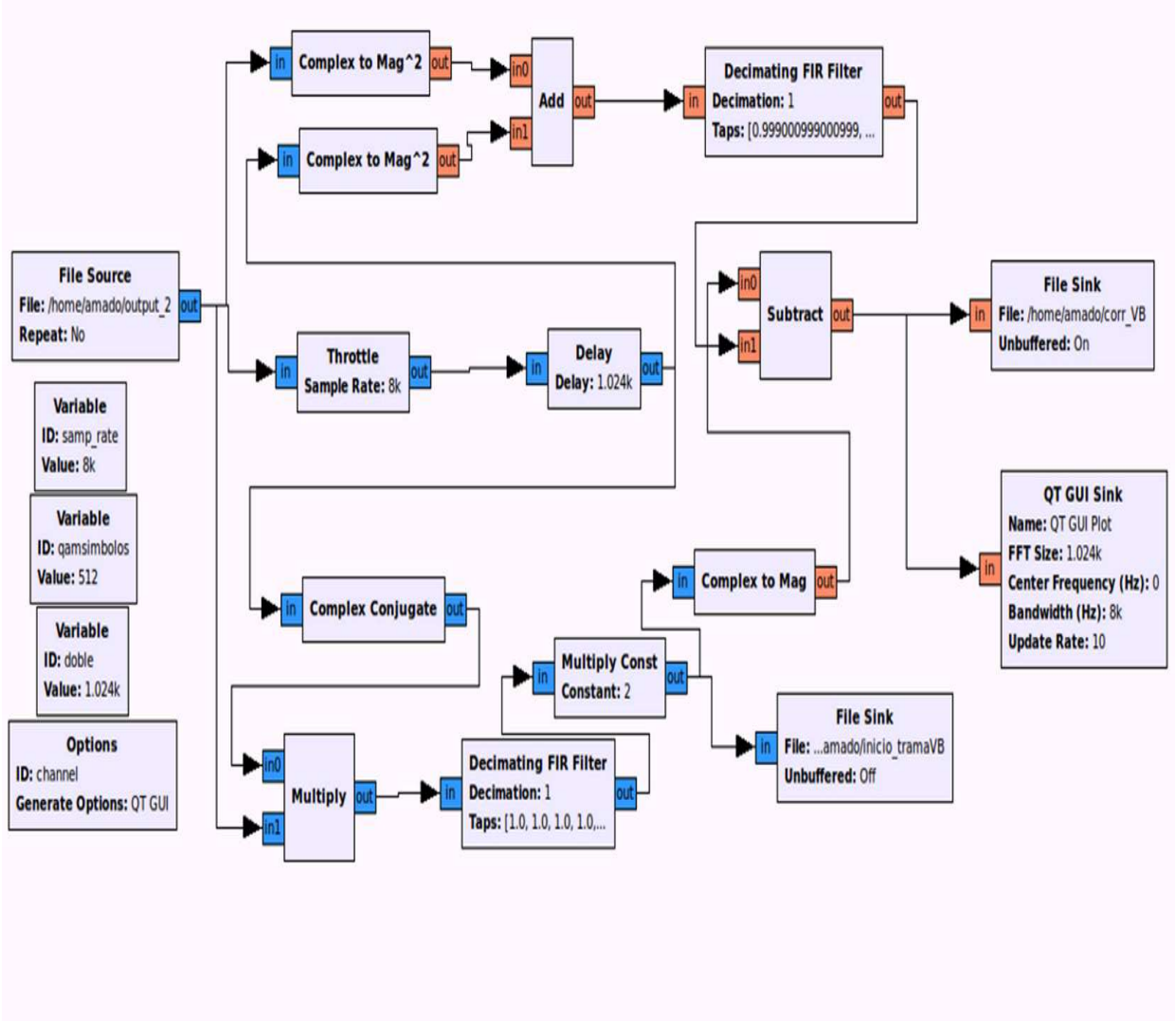


Figura 4.8: Aplicación para la realización de la sincronización burda en tiempo.

En la figura 4.9 se muestra el resultado obtenido al implementar el sistema de sincronización burda (en tiempo), dentro del software GNU Radio Companion. El pico de la correlación indica el inicio de un símbolo OFDM.

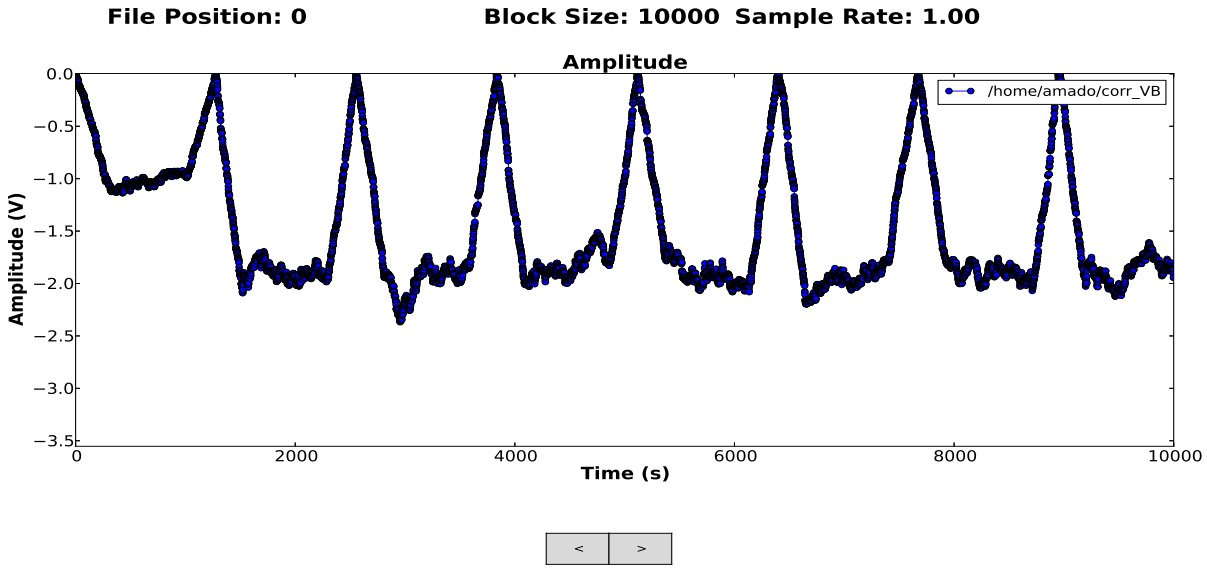


Figura 4.9: Sincronización burda.

Capítulo 5
Conclusiones y trabajo Futuro

5.1. Conclusiones

En el presente trabajo de investigación se presentó la implementación práctica de los algoritmos de nivel físico de un radio OFDM para comunicación digital con una plataforma Software Defined Radio (SDR). En este trabajo se adaptaron los modelos formales propuestos por M. Crussière [27] para su funcionamiento a plataforma SDR con un canal inalámbrico.

Como el objetivo general propuesto era diseñar algoritmos básicos de nivel físico de un radio OFDM utilizando una plataforma SDR, para satisfacer dicho objetivo, se requerió realizar nuestros propios en dicha plataforma.

Nos encontramos con la problemática de la poca información que existe sobre la plataforma SDR aunque existen algunos foros donde se exponen ideas, preguntas, experiencias. El hardware SDR elegido para la realización de este proyecto se compone por el módulo USRP1, las tarjetas hijas RFX900, las antenas VERT900 y el software SDR de GNU Radio.

El **diseño** de los algoritmos de nivel físico consistió en adecuar los algoritmos al software SDR GNU Radio que esta formado por tres capas; la capa mas interna, la programación en lenguaje C++ de la función principal del algoritmo, la programación gráfica de los bloques en XML, importante para visualizar los algoritmos en un interfaz gráfico de GNU Radio y así simular el sistema de comunicación digital en GNU Radio Companion (interfaz gráfica de GNU radio) y enlazar el flujo de datos de los bloques del sistema de comunicación por medio del lenguaje de programación Python.

Una vez diseñados los algoritmos en una plataforma SDR se migraron para su implementación práctica programando los bloques de generación de datos, conversor serie-paralelo, modulación QAM, prefijo cíclico, modulación de portadora, utilizando los lenguajes de programación C++, XML y PYTHON. Cabe mencionar que la propuesta de GNU Radio en la salida de cada bloques es “flujo de datos por determinado tamaño y tiempo”, los cuales

no son adecuados para el sistema implementado, así que se utilizaron "vectores de datos continuos".

La ejecución de los algoritmos de nivel físico en el entorno GNU Radio Companion proporcionaron resultados individuales del sistema de comunicación digital (banda base) muy satisfactorios. El diseño y programación de los algoritmos en GNU Radio Companion permitieron la **verificación** de éstos en una plataforma SDR, logrando satisfactoriamente dicho objetivo.

La utilización del hardware SDR (compatible con GNU Radio) permitió **validar** los algoritmos de nivel físico de un radio OFDM que conforman el sistema de comunicación digital, corroborando el espectro de la señal de datos transmitido por el USRP1 transmisor a través del analizador de espectros USRP1 receptor. Se midió el espectro de forma real, de la señal transmitida, y se comprobó que cumplía lo indicado por la teoría.

La poca información existente sobre aspectos prácticos de la plataforma SDR (GNU Radio y USRP) ocasionó que el proceso fuera lento. Cómo aporte, después de la realización de este trabajo de investigación, reportamos en el Apéndice B, C y D la metodología para migrar algoritmos de nivel físico, de cualquier tecnología digital de comunicaciones, a la plataforma SDR. En general, la metodología propuesta en el presente trabajo es el siguiente:

- **Diseñar los algoritmos en las 3 capas que conforman la arquitectura de GNU Radio.**

- **Verificar el funcionamiento de los algoritmos dentro del software GNU Radio Companion.**

- **Validar los algoritmos en el USRP1 transmisor y receptor.**

Actualmente la presión de los avances tecnológicos obliga a los desarrolladores a reducir el tiempo de ciclo y desarrollo. Por esta razón si no se sigue la metodología adecuada para el proceso de diseño, verificación y validación de los algoritmos de nivel físico de cualquier tecnología de sistema de comunicación digital pueden ser de tiempos muy largos.

En este contexto, el presente trabajo de investigación hace énfasis en la metodología para el diseño y verificación de los algoritmos de comunicación digital utilizando la plataforma SDR para acelerar la verificación y validación de los algoritmos de nivel físico y así presentar un enfoque práctico para el diseño de sistemas de comunicación digital que son utilizados en

aplicaciones reales.

En el módulo de la sincronización burda, se obtuvieron los picos que indican el inicio de los símbolos OFDM recibidos y, es utilizado para la posición correcta de la ventana FFT.

Sincronización fina, aspecto práctico y real. La estimación de CFO depende del hardware, modulador en cuadratura y ADC. Sin embargo, la teoría muchas veces no contempla los efectos reales del canal y las características físicas del hardware, parámetros importantes para obtener un buen desempeño en la sincronización fina. La solución propuesta es utilizar el metodo empírico de prueba y error y, una buena observación del espectro de la señal recibida para estimar el CFO.

5.2. Trabajo futuro

El alcance de este proyecto se ve afectado por el tiempo y el financiamiento, sin embargo, se propone, como continuación de este trabajo de investigación, la programación de la sincronización fina ya que este requiere de un estudio muy cuidadoso en el momento de la implementación práctica. En muchas simulaciones de sincronización fina no se toman en cuenta parámetros importantes como la estimación del offset de frecuencia, la sincronización del equipo transmisor y receptor, que son factores muy importantes que pueden llegar a originar la rotación de las constelaciones en el receptor.

Como resultado de este trabajo, hemos comprobado que el método de estimación de canal más conveniente es el de asistido por símbolos pilotos y lo más complejo en la plataforma SDR es realizar la estimación de CFO (siglas del ingl. *Carrier Frequency Offset*). Algunos métodos propuestos en el estado del arte para la estimación de CFO que sugerimos desarrollar y probar como trabajo futuro, son el método de bisección y el método empírico de prueba y error.

Se espera que este proyecto sirva a nuevas generaciones de investigadores para fomentar el desarrollo de algoritmos de nivel físico con implementación práctica (es decir, ir más allá del proceso de simulación), así como la impartición de cátedras con un enfoque práctico en el área de sistemas de comunicaciones digitales nivel Maestría y Licenciatura.

Referencias

[1] G. Laguna and A. Prieto, *Reporte técnico: Simulador Montecarlo para el esquema básico de comunicación OFDM sobre la red eléctrica*. México, DF.: Universidad Autónoma Metropolitana, ISBN 02.0201.II.15.003.2011, 2011.

[2] J.-J. van de Beek, M. Sandell, and P.O. Borjesson, "ML estimation of time and frequency offset in OFDM systems," *IEEE Transactions on Signal Processing*, vol. 45, no. 7, pp. 1800-1805, 1997.

[3] J. Mitola, "Software Radios: Survey, Critical Evaluation and Future Directions," *IEEE National Telesystems Conference*, 1992.

[4] M. Palkovic, P. Raghavan, M. Li, A. Dejonghe, L. Van der Perre, and F. Catthoor, "Future Software-Defined Radio Platforms and Mapping Flows," *Signal Processing Magazine*, IEEE, vol. 27, no. 2, pp. 22-33, 2010.

[5] A. Marwanto, M. A. Sarijari, N. Fisal, S. K. Yusof and R. A. Rashid, "Experimental Study of OFDM Implementation Utilizing GNU Radio and USRP-SDR," *IEEE 9th Malasya International Conference on Communications*, 2009.

[6] M. Dardaillon, K. Marquet, T. Risset and A. Scherrer, "Software Defined Radio Architecture Survey for Cognitive Testbeds," *in Proceedings of the International Wireless Communications and Mobile Computing Conference*, IEEE, 2012.

[7] "Universal software radio peripheral (usrp)."http://www.ettus.com

[8] "Quicksilver."http://www.philcovington.com/QuickSilver.

[9] K. Tan, H. Liu, J. Zhang, Y. Zhang, J. Fang, and G. M. Voelker, "Sora: high-performance software radio using general-purpose multi-core processors," *Communications of the ACM,* vol. 54, no. 1, pp. 99-107, 2011.

[10] T. Ulversoy, "Software defined radio: Challenges and opportunities," *Communications Surveys and Tutorials*, IEEE, vol. 12, no. 4, pp. 531-550, 2010.

[11] P.-H. Horrein, C. Hennebert, and F. Pétrot, "Adapting a SDR environment to GPU architectures,"in *Wireless Innovation Forum (SDR Forum)*, (Brussels, Belgium), June 2011.

[12] Minden et al., "KUAR: A Flexible Software-Defined Radio Development Platform,"in *2nd IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks*, (Dublin, Ireland), pp. 428-439, IEEE, Apr. 2007.

[13] K. V. Berkel, F. Heinle, P. P. E. Meuwissen, K. Moerman, and M. Weiss, "Vector Processing as an Enabler for Software-Defined Radio in Handheld Devices," *EURASIP Journal on Advances in Signal Processing*, no. 16, pp. 2613-2625, 2005.

[14] C. Jalier, D. Lattard, A. Jerraya, G. Sassatelli, P. Benoit, and L. Torres, "Heterogeneous vs homogeneous MPSoC approaches for a mobile LTE modem,ïn *Conference on Design, Automation and Test in Europe*, (Dresden, Germany), pp. 184-189, Mar. 2010.

[15] www.icast.icst.org/2011/06/cooperative-cognitive-radio-network-testbed

[16] S. Olivieri, "Modular FPGA-Based Software Defined Radio for CubeSats", *Thesis of Master*, May. 2011.

[17] http://www.analog.com/en/analog-to-digital-converters/broadband codecs/ad9862/products/product.html

[18] C. Crespo, Radiación y Radiocomunicación", Departamento de Teoría de la Señal y Comunicaciones, 2006.

[19] A. Quintero, "SDR: La alternativa para la evolución inalámbrica a nivel físico", 2006.

[20] http://gnuradio.org/redmine/projects/gnuradio/wiki

[21] N. Manicka, "GNU Radio Tesbed", *Thesis for Master of Science in Computer Science*, University of Delaware, Spring 2007.

[22] N. Weste and D. Skellern, "VLSI for OFDM", *IEEE Communications Magazine*, 1998.

[23] D. Torres, "Programación y evaluación en un DSP del nivel físico de un módem OFDM para comunicación digital por la red eléctrica doméstica", tesis de Maestría , 2012.

[24] W. Shieh and I. Djordjevic,"OFDM for Optical Communications ,"Academic Press, Elsevier, 2010.

[25] G. Zhang, M. Leenheer, A. Morea and B. Mukherjee, "A Survey on OFDM-Based Elastic Core Optical Networking", *IEEE Communications Surveys and Tutorials*, vol. 15, No. 1, First Quarter 2013.

[26] G. Laguna and R. Barrón, *Reporte técnico: Especificación de referencia para sistema de comunicación por la línea eléctrica* (PLC). Instituto Politécnico Nacional, Centro de Investigación en Computación, ISBN 978-607-414-048-4, Enero 2009.

[27] M. Crussière, *Tesis: Étude et Optimisation de Communications à Haut-débit sur lignes d'énergie: exploitation de la combinaison* OFDM/CDMA. Institut National des Sciences Appliquèes de Rennes, Rennes, Francia, 2005.

Apéndice A
Glosario

A

Amplitud. Valor de una señal, se mide en volts o ampers.
Analógico. Entidad que varia de forma continúa.
Atenuación. Pérdida de la energía de una señal debido a la resistencia del medio.

B

Banda ancha. Se refiere a señales que utilizan una portadora como modulación.
Banda base. Se refiere a las señales que no son moduladas en un portadora.
Bit. Digito binario; unidad más pequeña de información; 1 o 0.

C

Canal. El medio donde se va transmitir los datos.
Comunicación de datos. Intercambio de información entre dos o más entidades.
Constelación. Representación gráfica de la fase y la amplitud de combinaciones de bits diferentes.
Corriente continua. Señal con frecuencia cero y amplitud constante.

D

Decibelio(dB). Medida de la energía relativa entre dos puntos de una señal .
Demodulación. Proceso de separación de la señal portadora de la señal de información.
Demodulador. Dispositivo que realiza la demodulación.
Detección de errores. Proceso que determina si algunos bits se ha cambiado durante la transmisión.
Desplazamiento de fase. Cambio de fase de una señal.
Distorsión. Cualquier cambio de una señal como consecuencia del ruido, atenuación u otras influencias.

E

Emisor. Sistema que origina un mensaje.
Emisor. Sistema que origina un mensaje.
Enlace. Camino de comunicación físico que transfiere datos de un dispositivo a otro.
Error. Daño producido en la transmisión de datos.
Espectro. Rango de frecuencias de una señal.

Estándar. Base o modelo en el que todo el mundo se ha puesto de acuerdo.

F

Fase. Posición realtiva en el tiempo de una señal.
Frecuencia. Número de ciclos por segundo de una señal periõdica.

H

Hercio (Hz). Unidad utilizada para medir frecuencia.

I

Instituto de Ingenieros Eléctricos y Electrónicos (IEEE). Grupo formado por ingenieros profesionales que tiene sociedades especializadas cuyos comités preparan estándares en las áreas de la especialidad de los miembros.
I/O. Input/Output; Entrada/Salida.

K

Kbps. Kilobits por segundo.

L

Línea de abonado digital (DSL). Tecnología que utiliza las redes de telecomunicaciones existentes para conseguir la entrega a alta velocidad de datos, voz, video y datos multimedia, todo en banda base.

M

Módem. Dispositivo para modular y demodular. Convierte una señal digital en una analógica(modulación) y viceversa (demodulación).
Modulación. Modificación de una o más características de una onda portadora por una señal de modulación.
Modulación por amplitud en cuadratura(QAM). Método de modulación digital o analógica en el que la fase y la amplitud de una señal portadora varía con la señal que modula.
Multiplexado por división en frecuencia(FDM). Combinación de señales analógicas en una única señal.
Multiplexado por división de frecuencias ortogonales (OFDM). Combinación de señales analógicas en múltiples señales portadoras espaciadas en frecuencia ortogonales.

N

Nivel físico. Primer nivel del modelo OSI, responsables de las especificaciones eléctricas y mecánicas del medio.

P

Protocolo. Reglas para la comunicación.

R

Receptor. Punto destino de un transmisión.
Ruido. Señal indeseables que afectan a un medio de transmisión y da lugar a la degradación o distorsión de los datos.

S

Señal. Ondas electromagnéticas propagadas a través de un medio de transmisión.

Símbolo OFDM. Conjunto de datos modulados con portadoras ortogonales.
Sincronización. Proceso que permite la transmisión y recepción entre dos dipositivos simultáneamente.

T

Tasa de transmisión. El número de bits enviados por segundo.
Transformada de Fourier. Técnica matemática que reduce una señal periódica en una serie de señales cosenoidales.

Apéndice B
Instalación del software de GNU Radio

Todas estas operaciones son realizadas en una terminal de linux.

1. Descargar el archivo build-gnuradio en el siguiente enlace.
   www.gnuradio.org/redmine/projects/gnuradio/wiki/installingGR.

2. Dar permisos para compilar el archivo build-gnuradio, tecleando los siguientes comandos:

   chmod a+x build-gnuradio
   ./build-gnuradio

3. Dar permisos de usuario (opcional)

   Habilitar SUDO
   nano/etc/sudoes
   root ALL = (ALL) ALL
   justo abajo de esta línea
   nombre de usuario ALL=(ALL) ALL

4. Una vez compilado el script build-gnuradio comienza a descargar el software al, terminar la descarga, verificar que se encuentra gnuradio y uhd, con el comando **(ls)** en terminal de linux.

INSTALACIÓN DE UHD.

5. Una vez localizado UHD, cambiar de directorio con los siguientes comandos:

   cd /uhd/host
   cd build
   cmake ../

6. Al terminar el proceso, poner el siguiente comando:

   cmake _DENABLE_USRP_E_UTILS=ON_DENABLE_E100=ON ../

7. Teclear los comandos:

   make
   make test
   sudo make install
   cd

   Nota: esperar a que termine el proceso de cada comando.

8. Teclear el siguiente comando:

   sudo gedit .bashrc

   Al abrir el archivo en blanco agregar lo siguiente:

   #LD_LIBRARY_PATH FOR UHD SET BY ME
   export LD_LIBRARY_PATH=$ LD_LIBRARY_PATH:usr/local/lib
   #LD_LIBRARY_PATH FOR UHD SET BY ME

9. Para finalizar la instalación, teclear el siguiente comando:

sudo ldconfig

INSTALACIÓN DE GNU RADIO.

10. Ir a la carpeta gnuradio y cambiar directorío

    cd /build

11. Una vez posicionado en la ruta /gnuradio/build, ejecutamos los siguientes comandos:

    cmake ../
    make
    make test
    sudo ldconfig
    cd

    Nota: esperar a que termine el proceso de cada comando.

12. Teclear el siguiente comando:

    sudo gedit .bashrc

    Al abrir el archivo en blanco agregar lo siguiente:

    #Python path for gnuradio set by me
    export PYTHONPATH=$PYTHONPATH:/usr/local/lib/python2.7/site-packages
    #Python path for gnuradio set by me

13. Para finalizar la instalación, teclear el siguiente comando.

    sudo ldconfig

14. Por último verificar si la instalación de gnuradio es correcta, tecleando en terminal.

gnuradio_companion

Nota: Debe abrirse la interfaz gráfica de GNU Radio Companion.

Apéndice C

Creación de bloques personalizados compatibles con el software de GNU Radio

La creación de bloques personalizados en GNU Radio una, herramienta muy importante para validar algoritmos de nivel físico en plataforma SDR, se procesa de la siguiente manera:

Todas estas operaciones son realizadas en la terminal de linux.

- Instalar el archivo gr_modtool.py en la carpeta de GNU Radio (donde se encuentran las carpetas gr_).

- Crear el módulo con la siguiente instrucción python gr_modtool.py create nombre_del_modulo.

  Nota: la correcta creación del módulo, permite visualizar el módulo, nombre_del_modulo.

- Cambiar de directorio con el comando **cd** gr-nombre_del_modulo, visualizar las siguientes carpetas: apps, cmake, CMakelists.txt, docs, grc, include, lib, python, swig.

  Nota: la correcta instalación, permite visualizar las carpetas.

- Dentro de la carpeta gr-nombre_del_modulo, copiar gr_modtool.py, vsiualizar las siguiente carpetas:apps, cmake, CMakelists.txt, docs, grc, include, lib, python, swig, gr_modtool.py.

- Dentro de la carpeta gr-nombre_del_modulo, crear los bloques de algoritmos individualmente, con la instrucción python gr_modtool.py add -t tipo_modulo nombre_del_modulo. tipo_modulo puede ser, general, sink y source.

  Nota: En este punto, la primera capa de GNU Radio esta completa. Sin embargo, el usuario, programa la función principal de su algoritmo.

- Para compilar y ejecutar el programa del algoritmo, con las instrucciones.

  sudo cmake ../
  sudo make
  sudo makes install
  sudo ldconfig

Apéndice D
Algoritmos

Algoritmo D.1: Señal antipodal_vf.h

```
1
2  #ifndef INCLUDED_TRANSMISOR_OFDMV4_SENAL_ANTIPODAL_VF_H
3  #define INCLUDED_TRANSMISOR_OFDMV4_SENAL_ANTIPODAL_VF_H
4
5  #include <Transmisor_OFDMv4_api.h>
6  #include <gr_sync_block.h>
7
8  class Transmisor_OFDMv4_Senal_Antipodal_vf;
9
10 typedef boost::shared_ptr<Transmisor_OFDMv4_Senal_Antipodal_vf>
       Transmisor_OFDMv4_Senal_Antipodal_vf_sptr;
11
12 TRANSMISOR_OFDMV4_API Transmisor_OFDMv4_Senal_Antipodal_vf_sptr
       Transmisor_OFDMv4_make_Senal_Antipodal_vf (int nbits=1, int ss=2);
13
14 /*!
15  * \brief <+description+>
16  * \ingroup block
17  *
18  */
19 class TRANSMISOR_OFDMV4_API Transmisor_OFDMv4_Senal_Antipodal_vf :
       public gr_sync_block
20 {
21
22         friend TRANSMISOR_OFDMV4_API
              Transmisor_OFDMv4_Senal_Antipodal_vf_sptr
              Transmisor_OFDMv4_make_Senal_Antipodal_vf (int nbits, int ss
              );
23         int d_nbits;
24         int d_ss;
25
26   Transmisor_OFDMv4_Senal_Antipodal_vf(int nbits, int ss);
```

```
27
28  public:
29   ~Transmisor_OFDMv4_Senal_Antipodal_vf();
30
31          // Where all the action really happens
32          int work (int noutput_items,
33              gr_vector_const_void_star &input_items,
34              gr_vector_void_star &output_items);
35
36  private:
37          int nbits;
38          int a;
39          int n[1][2000];
40          int qamsimbolos;
41          int ss;
42
43
44
45  };
46
47  #endif /* INCLUDED_TRANSMISOR_OFDMV4_SENAL_ANTIPODAL_VF_H */
```

Algoritmo D.2: Señal antipodal_vf.cc

```
1   #ifdef HAVE_CONFIG_H
2   #include "config.h"
3   #endif
4
5   #include <gr_io_signature.h>
6   #include "Transmisor_OFDMv4_Senal_Antipodal_vf.h"
7   #include<iostream>
8   #include<cstdlib>
9   #include<ctime>
10  #include<string.h>
11  #include<fstream>
12  #include <errno.h>
13  #include<cstdio>
14  using std::cout;
15  using std::endl;
16  using std::cin;
17  #include <iomanip>
18  using std::setw;
19
20
21  Transmisor_OFDMv4_Senal_Antipodal_vf_sptr
22  Transmisor_OFDMv4_make_Senal_Antipodal_vf (int nbits, int ss)
23  {
24          return gnuradio::get_initial_sptr (new
```

```
            Transmisor_OFDMv4_Senal_Antipodal_vf(nbits,ss));
}

/*
 * The private constructor
 */
Transmisor_OFDMv4_Senal_Antipodal_vf::
   Transmisor_OFDMv4_Senal_Antipodal_vf (int nbits, int ss)
  : gr_sync_block ("Senal_Antipodal_vf",
                   gr_make_io_signature(0,0,0),
                   gr_make_io_signature(1,1, sizeof (float)*nbits)),
                   d_nbits(nbits),
                  d_ss(ss)

{
        // Put in <+constructor stuff+> here
}
int cont=0;

/*
 * Our virtual destructor.
 */
Transmisor_OFDMv4_Senal_Antipodal_vf::~
   Transmisor_OFDMv4_Senal_Antipodal_vf()
{
        // Put in <+destructor stuff+> here
}


int
Transmisor_OFDMv4_Senal_Antipodal_vf::work(int noutput_items,
                   gr_vector_const_void_star &input_items,
                   gr_vector_void_star &output_items)
{
        //const float *in = (const float *) input_items[0];
        float *out = (float *) output_items[0];

        // Do <+signal processing+>

//nbits=d_qamsimbolos*d_ss;




unsigned int seed;
FILE* urandom=fopen("/dev/urandom","r");
fread(&seed, sizeof(int),1,urandom);
fclose(urandom);
```

```
70 srand(seed);
71
72 for(int i=0;i<d_ss-1;i++){
73                   for(int j=0;j<d_nbits;j++){
74                   a=rand()%2;
75                   if (a==1)
76                           a=1;
77                   else
78                           a=-1;
79           n[i][j]=a;}}
80
81 //impresion de los valores antipodales
82 for(int j=0;j<d_nbits;j++){
83 out[j]=n[0][j];
84 //cout<<""<<setw(6)<<j;
85 //cout<<""<<setw(6)<<out[j];
86 }
87 cont = cont+1;
88 cout<<""<<cont;
89 cout<<endl;
90
91 return 1;
92
93          // Tell runtime system how many output items we produced.
94 //       return noutput_items;
95 }
```

Algoritmo D.3: Señal antipodal_vf.xml

```
1  <?xml version="1.0"?>
2  <block>
3    <name>Bits_de_entrada_vf</name>
4    <key>Transmisor_OFDMv4_Senal_Antipodal_vf</key>
5    <category>Transmisor_OFDMv4</category>
6    <import>import Transmisor_OFDMv4</import>
7    <make>Transmisor_OFDMv4.Senal_Antipodal_vf($nbits,$ss)</make>
8    <callback>set_nbits($nbits)</callback>
9    <callback>set_ss($ss)</callback>
10
11   <param>
12     <name>nbits</name>
13     <key>nbits</key>
14     <value>1</value>
15     <type>int</type>
16   </param>
17
18  <param>
19          <name>SS</name>
```

```
        <key>ss</key>
        <value>2</value>
        <type>int</type>
</param>

  <source>
    <name>out</name>
    <type>float</type>
    <vlen>$nbits</vlen>
  </source>
<doc>
ss=2, por lo tanto  qamsimbolos=nbits/ss
</doc>
</block>
```

Algoritmo D.4: Reshape_vfc.h

```
#ifndef INCLUDED_TRANSMISOR_OFDMV4_RESHAPE_VFC_H
#define INCLUDED_TRANSMISOR_OFDMV4_RESHAPE_VFC_H

#include <Transmisor_OFDMv4_api.h>
#include <gr_block.h>

class Transmisor_OFDMv4_reshape_vfc;

typedef boost::shared_ptr<Transmisor_OFDMv4_reshape_vfc>
   Transmisor_OFDMv4_reshape_vfc_sptr;

TRANSMISOR_OFDMV4_API Transmisor_OFDMv4_reshape_vfc_sptr
   Transmisor_OFDMv4_make_reshape_vfc (int nbits=1, int ss=2, int
   qamsimbolos=1);

/*!
 * \brief <+description+>
 * \ingroup block
 *
 */
class TRANSMISOR_OFDMV4_API Transmisor_OFDMv4_reshape_vfc : public
   gr_block
{

        friend TRANSMISOR_OFDMV4_API Transmisor_OFDMv4_reshape_vfc_sptr
           Transmisor_OFDMv4_make_reshape_vfc (int nbits, int ss, int
           qamsimbolos);
        int d_nbits;
        int d_ss;
        int d_qamsimbolos;

```

```
26
27    Transmisor_OFDMv4_reshape_vfc(int nbits, int ss, int qamsimbolos);
28
29   public:
30    ~Transmisor_OFDMv4_reshape_vfc();
31
32          // Where all the action really happens
33          int general_work (int noutput_items,
34              gr_vector_int &ninput_items,
35              gr_vector_const_void_star &input_items,
36              gr_vector_void_star &output_items);
37  private:
38
39          float m[2000][2];
40          float d[2][2000];
41          int  qamsimbolos;
42          int ss;
43          int nbits;
44  };
45
46  #endif /* INCLUDED_TRANSMISOR_OFDMV4_RESHAPE_VFC_H */
```

Algoritmo D.5: Reshape_vfc.cc

```
1   #ifdef HAVE_CONFIG_H
2   #include "config.h"
3   #endif
4
5   #include <gr_io_signature.h>
6   #include "Transmisor_OFDMv4_reshape_vfc.h"
7   #include<iostream>
8   #include<cstdlib>
9   #include<ctime>
10  #include<string.h>
11  using std::cout;
12  using std::endl;
13  using std::cin;
14  #include <iomanip>
15  using std::setw;
16
17
18  Transmisor_OFDMv4_reshape_vfc_sptr
19  Transmisor_OFDMv4_make_reshape_vfc (int nbits, int ss, int qamsimbolos)
20  {
21          return gnuradio::get_initial_sptr (new
                Transmisor_OFDMv4_reshape_vfc(nbits,ss,qamsimbolos));
22  }
23
```

```
24 /*
25  * The private constructor
26  */
27 Transmisor_OFDMv4_reshape_vfc::Transmisor_OFDMv4_reshape_vfc (int nbits,
       int ss, int qamsimbolos)
28   : gr_block ("reshape_vfc",
29                     gr_make_io_signature(1,1, sizeof (float)*nbits),
30                     gr_make_io_signature(1,1, sizeof (gr_complex)*
                          qamsimbolos)),
31                 d_nbits(nbits),
32                 d_ss(ss),
33                 d_qamsimbolos(qamsimbolos)
34 {
35         // Put in <+constructor stuff+> here
36 }
37
38
39 /*
40  * Our virtual destructor.
41  */
42 Transmisor_OFDMv4_reshape_vfc::~Transmisor_OFDMv4_reshape_vfc()
43 {
44         // Put in <+destructor stuff+> here
45 }
46
47
48 int
49 Transmisor_OFDMv4_reshape_vfc::general_work (int noutput_items,
50                                    gr_vector_int &ninput_items,
51                                    gr_vector_const_void_star &
                                         input_items,
52                                    gr_vector_void_star &output_items)
53 {
54         const float *in = (const float *) input_items[0];
55         gr_complex *out = (gr_complex *) output_items[0];
56
57         // Do <+signal processing+>
58
59 d_nbits=d_qamsimbolos*d_ss;
60
61
62
63 for (int i=0;i<d_qamsimbolos;i++){
64         for(int j=0;j<d_ss;j++){
65         m[i][j]=in[i+d_qamsimbolos*j];}}
66
67
68 if(d_ss==2){
```

```
69
70          for(int i=0;i<d_ss;i++){
71                  for(int j=0;j<d_qamsimbolos;j++){
72                  d[i][j]=m[j][i];}}
73          }
74 for(int j=0;j<d_qamsimbolos;j++)
75 {
76 out[j]=gr_complex(d[0][j],d[1][j]);
77 //cout<<""<<setw(6)<<out[j];
78 }
79          // Tell runtime system how many input items we consumed on
80          // each input stream.
81          consume_each (1);
82
83          return 1;
84
85 }
```

Algoritmo D.6: Reshape_vfc.xml

```
1  <?xml version="1.0"?>
2  <block>
3    <name>Serial_a_Paralelo_Mapeo_QAM_vfc</name>
4    <key>Transmisor_OFDMv4_reshape_vfc</key>
5    <category>Transmisor_OFDMv4</category>
6    <import>import Transmisor_OFDMv4</import>
7    <make>Transmisor_OFDMv4.reshape_vfc($nbits,$ss,$qamsimbolos)</make>
8   <callback>set_nbits($nbits)</callback>
9    <callback>set_ss($ss)</callback>
10   <callback>set_qamsimbolos($qamsimbolos)</callback>
11
12
13   <param>
14     <name>nbits</name>
15     <key>nbits</key>
16     <value>1</value>
17     <type>int</type>
18   </param>
19
20    <param>
21     <name>SS</name>
22     <key>ss</key>
23     <value>2</value>
24     <type>int</type>
25   </param>
26
27  <param>
28     <name>Qamsimbolos</name>
```

```
    <key>qamsimbolos</key>
    <value>2</value>
    <type>int</type>
  </param>



  <sink>
    <name>in</name>
    <type>float</type>
    <vlen>$nbits</vlen>
  </sink>

  <source>
    <name>out</name>
    <type>complex</type>
    <vlen>$qamsimbolos</vlen>
  </source>

</block>
```

Algoritmo D.7: Simetría hermitiana_vcc.h

```
#ifndef INCLUDED_TRANSMISOR_OFDMV4_SIM_HERM_VCC_H
#define INCLUDED_TRANSMISOR_OFDMV4_SIM_HERM_VCC_H

#include <Transmisor_OFDMv4_api.h>
#include <gr_block.h>

class Transmisor_OFDMv4_sim_herm_vcc;

typedef boost::shared_ptr<Transmisor_OFDMv4_sim_herm_vcc>
    Transmisor_OFDMv4_sim_herm_vcc_sptr;

TRANSMISOR_OFDMV4_API Transmisor_OFDMv4_sim_herm_vcc_sptr
    Transmisor_OFDMv4_make_sim_herm_vcc (int qamsimbolos=1, int ss=2,
    int fft_length=1 );

/*!
 * \brief <+description+>
 * \ingroup block
 *
 */
class TRANSMISOR_OFDMV4_API Transmisor_OFDMv4_sim_herm_vcc : public
    gr_block
{

        friend TRANSMISOR_OFDMV4_API Transmisor_OFDMv4_sim_herm_vcc_sptr
```

```
            Transmisor_OFDMv4_make_sim_herm_vcc (int qamsimbolos, int
           ss, int fft_length);
        int d_qamsimbolos;
        int d_ss;
        int d_fft_length;

  Transmisor_OFDMv4_sim_herm_vcc(int qamsimbolos, int ss, int fft_length
     );

 public:
  ~Transmisor_OFDMv4_sim_herm_vcc();

        // Where all the action really happens
        int general_work (int noutput_items,
            gr_vector_int &ninput_items,
            gr_vector_const_void_star &input_items,
            gr_vector_void_star &output_items);
private:
        gr_complex sim_sc[2][2000];
        gr_complex sim_cc[2][2000];


};

#endif /* INCLUDED_TRANSMISOR_OFDMV4_SIM_HERM_VCC_H */
```

Algoritmo D.8: Simetría hermitiana_vcc.cc

```
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <gr_io_signature.h>
#include "Transmisor_OFDMv4_sim_herm_vcc.h"
#include<iostream>
#include<cstdlib>
#include<ctime>
#include<complex>
#include<string.h>
using std::cout;
using std::endl;
using std::cin;
#include <iomanip>
using std::setw;

Transmisor_OFDMv4_sim_herm_vcc_sptr
Transmisor_OFDMv4_make_sim_herm_vcc (int qamsimbolos, int ss, int
   fft_length)
```

```
20 {
21         return gnuradio::get_initial_sptr (new
              Transmisor_OFDMv4_sim_herm_vcc(qamsimbolos,ss,fft_length));
22 }
23
24 /*
25  * The private constructor
26  */
27 Transmisor_OFDMv4_sim_herm_vcc::Transmisor_OFDMv4_sim_herm_vcc (int
       qamsimbolos, int ss, int fft_length)
28   : gr_block ("sim_herm_vcc",
29                     gr_make_io_signature(1, 1, sizeof (gr_complex)*
                          qamsimbolos),
30                     gr_make_io_signature(1, 1, sizeof (gr_complex)*
                          fft_length)),
31                     d_qamsimbolos(qamsimbolos),
32                     d_ss(ss),
33                     d_fft_length(fft_length)
34 {
35         // Put in <+constructor stuff+> here
36 }
37
38
39 /*
40  * Our virtual destructor.
41  */
42 Transmisor_OFDMv4_sim_herm_vcc::~Transmisor_OFDMv4_sim_herm_vcc()
43 {
44         // Put in <+destructor stuff+> here
45 }
46
47
48 int
49 Transmisor_OFDMv4_sim_herm_vcc::general_work (int noutput_items,
50                                     gr_vector_int &ninput_items,
51                                     gr_vector_const_void_star &
                                          input_items,
52                                     gr_vector_void_star &output_items)
53 {
54         gr_complex *in = (gr_complex *) input_items[0];
55         gr_complex *out = (gr_complex *) output_items[0];
56
57         // Do <+signal processing+>
58          for(int j=0;j<d_fft_length;j++){
59         if(j>0 && j<d_qamsimbolos){
60         sim_sc[0][j]=in[j];}
61         sim_cc[0][j]=conj(in[j]);
62         if(j>d_qamsimbolos && j<d_fft_length){
```

```
63          sim_sc[0][j]=sim_cc[0][d_qamsimbolos-(j-d_qamsimbolos)];}}
64
65  for(int j=0;j<d_fft_length;j++)
66  {
67
68  out[j]=sim_sc[0][j];
69  //cout<<""<<setw(6)<<out[j];
70  }
71  //cout<<endl;
72
73          // Tell runtime system how many input items we consumed on
74          // each input stream.
75          consume_each (1);
76
77          // Tell runtime system how many output items we produced.
78          return 1;
79  }
```

Algoritmo D.9: Simetría hermitiana_vcc.xml

```
1   <?xml version="1.0"?>
2   <block>
3     <name>Simetria_Hermitiana_vcc</name>
4     <key>Transmisor_OFDMv4_sim_herm_vcc</key>
5     <category>Transmisor_OFDMv4</category>
6     <import>import Transmisor_OFDMv4</import>
7     <make>Transmisor_OFDMv4.sim_herm_vcc($qamsimbolos,$ss,$fft_length)</
          make>
8   <callback>set_qamsimbolos($qamsimbolos)</callback>
9   <callback>set_ss($ss)</callback>
10  <callback>set_fft_length($fft_length)</callback>
11
12    <param>
13      <name>Qamsimbolos</name>
14      <key>qamsimbolos</key>
15      <value>1</value>
16      <type>int</type>
17    </param>
18
19     <param>
20      <name>SS</name>
21      <key>ss</key>
22      <value>2</value>
23      <type>int</type>
24    </param>
25
26  <param>
27      <name>FFT_LENGTH</name>
```

```
    <key>fft_length</key>
    <value>2</value>
    <type>int</type>
  </param>



  <sink>
    <name>in</name>
    <type>complex</type>
    <vlen>$qamsimbolos</vlen>
  </sink>

  <source>
    <name>out</name>
    <type>complex</type>
    <vlen>$fft_length</vlen>
  </source>

</block>
```

Algoritmo D.10: Prefijo cíclico_vcc.h

```
#ifndef INCLUDED_TRANSMISOR_OFDMV4_PRE_CICLICO_VCC_H
#define INCLUDED_TRANSMISOR_OFDMV4_PRE_CICLICO_VCC_H

#include <Transmisor_OFDMv4_api.h>
#include <gr_block.h>

class Transmisor_OFDMv4_pre_ciclico_vcc;

typedef boost::shared_ptr<Transmisor_OFDMv4_pre_ciclico_vcc>
   Transmisor_OFDMv4_pre_ciclico_vcc_sptr;

TRANSMISOR_OFDMV4_API Transmisor_OFDMv4_pre_ciclico_vcc_sptr
   Transmisor_OFDMv4_make_pre_ciclico_vcc (int fft_length, float Dprop,
    int T_total);

/*!
 * \brief <+description+>
 * \ingroup block
 *
 */
class TRANSMISOR_OFDMV4_API Transmisor_OFDMv4_pre_ciclico_vcc : public
   gr_block
{

        friend TRANSMISOR_OFDMV4_API
```

```
            Transmisor_OFDMv4_pre_ciclico_vcc_sptr
            Transmisor_OFDMv4_make_pre_ciclico_vcc (int fft_length,
            float Dprop, int T_total);
22          int d_fft_length;
23          float d_Dprop;
24          int d_T_total;
25
26    Transmisor_OFDMv4_pre_ciclico_vcc(int fft_length, float Dprop, int
         T_total);
27
28   public:
29    ~Transmisor_OFDMv4_pre_ciclico_vcc();
30
31          // Where all the action really happens
32          int general_work (int noutput_items,
33              gr_vector_int &ninput_items,
34              gr_vector_const_void_star &input_items,
35              gr_vector_void_star &output_items);
36  private:
37          float D;
38          float Dprop;
39          gr_complex prefijo[4][1500];
40         // float FFTsize;
41  };
42
43  #endif /* INCLUDED_TRANSMISOR_OFDMV4_PRE_CICLICO_VCC_H */
```

Algoritmo D.11: Prefijo cíclico_vcc.cc

```
1   #ifdef HAVE_CONFIG_H
2   #include "config.h"
3   #endif
4
5   #include <gr_io_signature.h>
6   #include "Transmisor_OFDMv4_pre_ciclico_vcc.h"
7   #include<iostream>
8   #include<cstdlib>
9   #include<ctime>
10  #include<complex>
11  using std::cout;
12  using std::endl;
13  using std::cin;
14  #include <iomanip>
15  using std::setw;
16
17
18  Transmisor_OFDMv4_pre_ciclico_vcc_sptr
19  Transmisor_OFDMv4_make_pre_ciclico_vcc (int fft_length, float Dprop, int
```

```
    T_total)
{
        return gnuradio::get_initial_sptr (new
            Transmisor_OFDMv4_pre_ciclico_vcc(fft_length,Dprop,T_total))
            ;
}

/*
 * The private constructor
 */
Transmisor_OFDMv4_pre_ciclico_vcc::Transmisor_OFDMv4_pre_ciclico_vcc (
    int fft_length, float Dprop, int T_total)
  : gr_block ("pre_ciclico_vcc",
                    gr_make_io_signature(1,1, sizeof (gr_complex)*
                        fft_length),
                    gr_make_io_signature(1,1, sizeof (gr_complex)*T_total
                        )),
                    d_fft_length(fft_length),
                    d_Dprop(Dprop),
                    d_T_total(T_total)
{
        // Put in <+constructor stuff+> here
}


/*
 * Our virtual destructor.
 */
Transmisor_OFDMv4_pre_ciclico_vcc::~Transmisor_OFDMv4_pre_ciclico_vcc()
{
        // Put in <+destructor stuff+> here
}


int
Transmisor_OFDMv4_pre_ciclico_vcc::general_work (int noutput_items,
                                    gr_vector_int &ninput_items,
                                    gr_vector_const_void_star &
                                        input_items,
                                    gr_vector_void_star &output_items)
{
        gr_complex *in = (gr_complex *) input_items[0];
        gr_complex *out = (gr_complex *) output_items[0];

        // Do <+signal processing+>


D=floor(d_Dprop*d_fft_length);
```

```
61 //T_total=D+d_fft_length;
62
63 int n;
64 int m;
65 int k;
66 n=d_fft_length-D;
67 for(int i=d_fft_length-D;i<d_fft_length;i++)
68 {
69 prefijo[0][i]=in[i];
70 }
71
72 for(int j=0;j<d_T_total;j++)
73 {
74          if(j<D){
75          m=j+(d_fft_length-D);
76          out[j]=prefijo[0][m];}
77          if(j>=D){
78          k=j-D;
79          out[j]=in[k];}
80 //cout<<""<<setw(12)<<j;
81 //cout<<""<<setw(12)<<out[j];
82
83 }
84          // Tell runtime system how many input items we consumed on
85          // each input stream.
86          consume_each (1);
87
88          // Tell runtime system how many output items we produced.
89          return 1;
90 }
```

Algoritmo D.12: Prefijo cíclico_vcc.xml

```
1 <?xml version="1.0"?>
2 <block>
3   <name>Prefijo_Ciclico_vcc</name>
4   <key>Transmisor_OFDMv4_pre_ciclico_vcc</key>
5   <category>Transmisor_OFDMv4</category>
6   <import>import Transmisor_OFDMv4</import>
7   <make>Transmisor_OFDMv4.pre_ciclico_vcc($fft_length, $Dprop, $T_total)
       </make>
8  <callback>set_fft_length($fft_length)</callback>
9  <callback>set_Dprop($Dprop)</callback>
10 <callback>set_T_total($T_total)</callback>
11
12   <param>
13     <name>FFT_LENGTH</name>
14     <key>fft_length</key>
```

```
    <value>1</value>
    <type>int</type>
  </param>

<param>
    <name>DPROP</name>
    <key>Dprop</key>
    <value>0.25</value>
    <type>real</type>
  </param>

<param>
    <name>T_TOTAL</name>
    <key>T_total</key>
    <value>1</value>
    <type>int</type>
  </param>



  <sink>
    <name>in</name>
    <type>complex</type>
    <vlen>$fft_length</vlen>
 </sink>


  <source>
    <name>out</name>
    <type>complex</type>
    <vlen>$T_total</vlen>
</source>
<doc>
Dprop = El tiempo de guarda, longitud del prefijo ciclico puede ser
   0.25(1/4), 0.5(1/2), 0.75(3/4)
</doc>

</block>
```

Algoritmo D.13: OFDM_banda_base.py

```
#!/usr/bin/env python
##################################################
# Gnuradio Python Flow Graph
# Title: OFDM
# Author: Amado
# Description: tesis
# Generated: Tue Dec 10 08:21:28 2013
```

```
8  ##################################################
9
10 from gnuradio import eng_notation
11 from gnuradio import fft
12 from gnuradio import gr
13 from gnuradio import window
14 from gnuradio.eng_option import eng_option
15 from gnuradio.gr import firdes
16 from grc_gnuradio import wxgui as grc_wxgui
17 from optparse import OptionParser
18 import Transmisor_OFDMv4
19 import math
20 import wx
21
22 class OFDM(grc_wxgui.top_block_gui):
23
24         def __init__(self):
25                 grc_wxgui.top_block_gui.__init__(self, title="OFDM")
26                 _icon_path = "/usr/share/icons/hicolor/32x32/apps/
                       gnuradio-grc.png"
27                 self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))
28
29                 ##################################################
30                 # Variables
31                 ##################################################
32                 self.nbits = nbits = 32
33                 self.qamsimbolos = qamsimbolos = nbits/2
34                 self.fft_length = fft_length = 2*qamsimbolos
35                 self.Dprop = Dprop = (1.0/4.0)
36                 self.samp_rate = samp_rate = 250e3
37                 self.escalamiento = escalamiento = 1.0/fft_length
38                 self.T_total = T_total = int(math.floor(Dprop*fft_length
                       )+fft_length)
39
40                 ##################################################
41                 # Blocks
42                 ##################################################
43                 self.gr_vector_to_stream_4 = gr.vector_to_stream(gr.
                       sizeof_gr_complex*1, T_total)
44                 self.gr_vector_to_stream_3 = gr.vector_to_stream(gr.
                       sizeof_gr_complex*1, fft_length)
45                 self.gr_vector_to_stream_2 = gr.vector_to_stream(gr.
                       sizeof_gr_complex*1, fft_length)
46                 self.gr_vector_to_stream_1 = gr.vector_to_stream(gr.
                       sizeof_gr_complex*1, qamsimbolos)
47                 self.gr_vector_to_stream_0 = gr.vector_to_stream(gr.
                       sizeof_float*1, nbits)
48                 self.gr_throttle_0 = gr.throttle(gr.sizeof_gr_complex*1,
```

```
                samp_rate)
49              self.gr_stream_to_vector_0 = gr.stream_to_vector(gr.
                    sizeof_gr_complex*1, nbits)
50              self.gr_multiply_const_vxx_0 = gr.multiply_const_vcc((
                    escalamiento, ))
51              self.gr_file_sink_4 = gr.file_sink(gr.sizeof_gr_complex
                    *1, "/home/amado/pre_cic_v4")
52              self.gr_file_sink_4.set_unbuffered(True)
53              self.gr_file_sink_3 = gr.file_sink(gr.sizeof_gr_complex
                    *1, "/home/amado/sim_herm_v4")
54              self.gr_file_sink_3.set_unbuffered(True)
55              self.gr_file_sink_2 = gr.file_sink(gr.sizeof_gr_complex
                    *1, "/home/amado/ifft_v4")
56              self.gr_file_sink_2.set_unbuffered(True)
57              self.gr_file_sink_1 = gr.file_sink(gr.sizeof_gr_complex
                    *1, "/home/amado/reshape_v4")
58              self.gr_file_sink_1.set_unbuffered(True)
59              self.gr_file_sink_0 = gr.file_sink(gr.sizeof_float*1, "/
                    home/amado/Sen_Ant_v4")
60              self.gr_file_sink_0.set_unbuffered(True)
61              self.fft_vxx_0 = fft.fft_vcc(fft_length, False, (window.
                    blackmanharris(1024)), False, 1)
62              self.Transmisor_OFDMv4_sim_herm_vcc_0 =
                    Transmisor_OFDMv4.sim_herm_vcc(qamsimbolos,2,
                    fft_length)
63              self.Transmisor_OFDMv4_reshape_vfc_0 = Transmisor_OFDMv4
                    .reshape_vfc(nbits,2,qamsimbolos)
64              self.Transmisor_OFDMv4_pre_ciclico_vcc_0 =
                    Transmisor_OFDMv4.pre_ciclico_vcc(fft_length, Dprop,
                     T_total)
65              self.Transmisor_OFDMv4_Senal_Antipodal_vf_0 =
                    Transmisor_OFDMv4.Senal_Antipodal_vf(nbits,2)
66
67              ##################################################
68              # Connections
69              ##################################################
70              self.connect((self.Transmisor_OFDMv4_reshape_vfc_0, 0),
                    (self.gr_vector_to_stream_1, 0))
71              self.connect((self.Transmisor_OFDMv4_reshape_vfc_0, 0),
                    (self.Transmisor_OFDMv4_sim_herm_vcc_0, 0))
72              self.connect((self.Transmisor_OFDMv4_sim_herm_vcc_0, 0),
                     (self.fft_vxx_0, 0))
73              self.connect((self.gr_vector_to_stream_3, 0), (self.
                    gr_file_sink_3, 0))
74              self.connect((self.gr_vector_to_stream_0, 0), (self.
                    gr_file_sink_0, 0))
75              self.connect((self.Transmisor_OFDMv4_sim_herm_vcc_0, 0),
                     (self.gr_vector_to_stream_3, 0))
```

```
76                  self.connect((self.fft_vxx_0, 0), (self.
                        gr_vector_to_stream_2, 0))
77                  self.connect((self.gr_vector_to_stream_1, 0), (self.
                        gr_file_sink_1, 0))
78                  self.connect((self.gr_vector_to_stream_4, 0), (self.
                        gr_file_sink_4, 0))
79                  self.connect((self.gr_stream_to_vector_0, 0), (self.
                        Transmisor_OFDMv4_pre_ciclico_vcc_0, 0))
80                  self.connect((self.Transmisor_OFDMv4_pre_ciclico_vcc_0,
                        0), (self.gr_vector_to_stream_4, 0))
81                  self.connect((self.
                        Transmisor_OFDMv4_Senal_Antipodal_vf_0, 0), (self.
                        Transmisor_OFDMv4_reshape_vfc_0, 0))
82                  self.connect((self.
                        Transmisor_OFDMv4_Senal_Antipodal_vf_0, 0), (self.
                        gr_vector_to_stream_0, 0))
83                  self.connect((self.gr_vector_to_stream_2, 0), (self.
                        gr_multiply_const_vxx_0, 0))
84                  self.connect((self.gr_multiply_const_vxx_0, 0), (self.
                        gr_file_sink_2, 0))
85                  self.connect((self.gr_multiply_const_vxx_0, 0), (self.
                        gr_throttle_0, 0))
86                  self.connect((self.gr_throttle_0, 0), (self.
                        gr_stream_to_vector_0, 0))
87
88
89          def get_nbits(self):
90                  return self.nbits
91
92          def set_nbits(self, nbits):
93                  self.nbits = nbits
94                  self.Transmisor_OFDMv4_Senal_Antipodal_vf_0.set_nbits(
                        self.nbits)
95                  self.Transmisor_OFDMv4_reshape_vfc_0.set_nbits(self.
                        nbits)
96                  self.set_qamsimbolos(self.nbits/2)
97
98          def get_qamsimbolos(self):
99                  return self.qamsimbolos
100
101         def set_qamsimbolos(self, qamsimbolos):
102                 self.qamsimbolos = qamsimbolos
103                 self.Transmisor_OFDMv4_reshape_vfc_0.set_qamsimbolos(
                        self.qamsimbolos)
104                 self.Transmisor_OFDMv4_sim_herm_vcc_0.set_qamsimbolos(
                        self.qamsimbolos)
105                 self.set_fft_length(2*self.qamsimbolos)
106
```

```
107         def get_fft_length(self):
108                 return self.fft_length
109 
110         def set_fft_length(self, fft_length):
111                 self.fft_length = fft_length
112                 self.Transmisor_OFDMv4_sim_herm_vcc_0.set_fft_length(
                        self.fft_length)
113                 self.Transmisor_OFDMv4_pre_ciclico_vcc_0.set_fft_length(
                        self.fft_length)
114                 self.set_T_total(int(math.floor(self.Dprop*self.
                        fft_length)+self.fft_length))
115                 self.set_escalamiento(1.0/self.fft_length)
116 
117         def get_Dprop(self):
118                 return self.Dprop
119 
120         def set_Dprop(self, Dprop):
121                 self.Dprop = Dprop
122                 self.Transmisor_OFDMv4_pre_ciclico_vcc_0.set_Dprop(self.
                        Dprop)
123                 self.set_T_total(int(math.floor(self.Dprop*self.
                        fft_length)+self.fft_length))
124 
125         def get_samp_rate(self):
126                 return self.samp_rate
127 
128         def set_samp_rate(self, samp_rate):
129                 self.samp_rate = samp_rate
130                 self.gr_throttle_0.set_sample_rate(self.samp_rate)
131 
132         def get_escalamiento(self):
133                 return self.escalamiento
134 
135         def set_escalamiento(self, escalamiento):
136                 self.escalamiento = escalamiento
137                 self.gr_multiply_const_vxx_0.set_k((self.escalamiento, )
                        )
138 
139         def get_T_total(self):
140                 return self.T_total
141 
142         def set_T_total(self, T_total):
143                 self.T_total = T_total
144                 self.Transmisor_OFDMv4_pre_ciclico_vcc_0.set_T_total(
                        self.T_total)
145 
146 if __name__ == '__main__':
147         parser = OptionParser(option_class=eng_option, usage="%prog: [
```

```
            options]")
148         (options, args) = parser.parse_args()
149         tb = OFDM()
150         tb.Run(True)
```

Algoritmo D.14: OFDM_portadora.py

```
1   #!/usr/bin/env python
2   ##################################################
3   # Gnuradio Python Flow Graph
4   # Title: OFDM
5   # Author: Amado Gutierrez
6   # Generated: Thu Aug  1 19:13:53 2013
7   ##################################################
8
9   from gnuradio import eng_notation
10  from gnuradio import fft
11  from gnuradio import gr
12  from gnuradio import window
13  from gnuradio.eng_option import eng_option
14  from gnuradio.gr import firdes
15  from grc_gnuradio import wxgui as grc_wxgui
16  from optparse import OptionParser
17  import Transmisor_OFDMv4
18  import math
19  import wx
20
21  class ofdm_transmitter(grc_wxgui.top_block_gui):
22
23          def __init__(self):
24                  grc_wxgui.top_block_gui.__init__(self, title="OFDM")
25                  _icon_path = "/usr/share/icons/hicolor/32x32/apps/
                        gnuradio-grc.png"
26                  self.SetIcon(wx.Icon(_icon_path, wx.BITMAP_TYPE_ANY))
27
28                  ##################################################
29                  # Variables
30                  ##################################################
31                  self.nbits = nbits = 32
32                  self.qamsimbolos = qamsimbolos = nbits/2
33                  self.fft_length = fft_length = 2*qamsimbolos
34                  self.Dprop = Dprop = (1.0/4.0)
35                  self.samp_rate = samp_rate = 1e6
36                  self.escalamiento = escalamiento = 1.0/fft_length
37                  self.T_total = T_total = int(math.floor(Dprop*fft_length
                        )+fft_length)
38
39                  ##################################################
```

```
40                  # Blocks
41                  ##################################################
42                  self.gr_vector_to_stream_4 = gr.vector_to_stream(gr.
                        sizeof_gr_complex*1, T_total)
43                  self.gr_vector_to_stream_2 = gr.vector_to_stream(gr.
                        sizeof_gr_complex*1, fft_length)
44                  self.gr_stream_to_vector_0 = gr.stream_to_vector(gr.
                        sizeof_gr_complex*1, nbits)
45                  self.gr_multiply_const_vxx_0 = gr.multiply_const_vcc((
                        escalamiento, ))
46                  self.gr_file_sink_4 = gr.file_sink(gr.sizeof_gr_complex
                        *1, "/home/amado/pre_cic_v4")
47                  self.gr_file_sink_4.set_unbuffered(True)
48                  self.fft_vxx_0 = fft.fft_vcc(fft_length, False, (window.
                        blackmanharris(1024)), False, 1)
49                  self.Transmisor_OFDMv4_sim_herm_vcc_0 =
                        Transmisor_OFDMv4.sim_herm_vcc(qamsimbolos,2,
                        fft_length)
50                  self.Transmisor_OFDMv4_reshape_vfc_0 = Transmisor_OFDMv4
                        .reshape_vfc(nbits,2,qamsimbolos)
51                  self.Transmisor_OFDMv4_pre_ciclico_vcc_0 =
                        Transmisor_OFDMv4.pre_ciclico_vcc(fft_length, Dprop,
                         T_total)
52                  self.Transmisor_OFDMv4_Senal_Antipodal_vf_0 =
                        Transmisor_OFDMv4.Senal_Antipodal_vf(nbits,2)
53
54                  ##################################################
55                  # Connections
56                  ##################################################
57                  self.connect((self.Transmisor_OFDMv4_reshape_vfc_0, 0),
                        (self.Transmisor_OFDMv4_sim_herm_vcc_0, 0))
58                  self.connect((self.Transmisor_OFDMv4_sim_herm_vcc_0, 0),
                         (self.fft_vxx_0, 0))
59                  self.connect((self.gr_vector_to_stream_2, 0), (self.
                        gr_multiply_const_vxx_0, 0))
60                  self.connect((self.fft_vxx_0, 0), (self.
                        gr_vector_to_stream_2, 0))
61                  self.connect((self.gr_vector_to_stream_4, 0), (self.
                        gr_file_sink_4, 0))
62                  self.connect((self.gr_stream_to_vector_0, 0), (self.
                        Transmisor_OFDMv4_pre_ciclico_vcc_0, 0))
63                  self.connect((self.Transmisor_OFDMv4_pre_ciclico_vcc_0,
                        0), (self.gr_vector_to_stream_4, 0))
64                  self.connect((self.
                        Transmisor_OFDMv4_Senal_Antipodal_vf_0, 0), (self.
                        Transmisor_OFDMv4_reshape_vfc_0, 0))
65                  self.connect((self.gr_multiply_const_vxx_0, 0), (self.
                        gr_stream_to_vector_0, 0))
```

```
66
67
68          def get_nbits(self):
69                  return self.nbits
70
71          def set_nbits(self, nbits):
72                  self.nbits = nbits
73                  self.Transmisor_OFDMv4_Senal_Antipodal_vf_0.set_nbits(
                        self.nbits)
74                  self.Transmisor_OFDMv4_reshape_vfc_0.set_nbits(self.
                        nbits)
75                  self.set_qamsimbolos(self.nbits/2)
76
77          def get_qamsimbolos(self):
78                  return self.qamsimbolos
79
80          def set_qamsimbolos(self, qamsimbolos):
81                  self.qamsimbolos = qamsimbolos
82                  self.Transmisor_OFDMv4_reshape_vfc_0.set_qamsimbolos(
                        self.qamsimbolos)
83                  self.Transmisor_OFDMv4_sim_herm_vcc_0.set_qamsimbolos(
                        self.qamsimbolos)
84                  self.set_fft_length(2*self.qamsimbolos)
85
86          def get_fft_length(self):
87                  return self.fft_length
88
89          def set_fft_length(self, fft_length):
90                  self.fft_length = fft_length
91                  self.Transmisor_OFDMv4_sim_herm_vcc_0.set_fft_length(
                        self.fft_length)
92                  self.Transmisor_OFDMv4_pre_ciclico_vcc_0.set_fft_length(
                        self.fft_length)
93                  self.set_escalamiento(1.0/self.fft_length)
94                  self.set_T_total(int(math.floor(self.Dprop*self.
                        fft_length)+self.fft_length))
95
96          def get_Dprop(self):
97                  return self.Dprop
98
99          def set_Dprop(self, Dprop):
100                 self.Dprop = Dprop
101                 self.Transmisor_OFDMv4_pre_ciclico_vcc_0.set_Dprop(self.
                        Dprop)
102                 self.set_T_total(int(math.floor(self.Dprop*self.
                        fft_length)+self.fft_length))
103
104         def get_samp_rate(self):
```

```
105                 return self.samp_rate
106
107         def set_samp_rate(self, samp_rate):
108                 self.samp_rate = samp_rate
109
110         def get_escalamiento(self):
111                 return self.escalamiento
112
113         def set_escalamiento(self, escalamiento):
114                 self.escalamiento = escalamiento
115                 self.gr_multiply_const_vxx_0.set_k((self.escalamiento, )
                        )
116
117         def get_T_total(self):
118                 return self.T_total
119
120         def set_T_total(self, T_total):
121                 self.T_total = T_total
122                 self.Transmisor_OFDMv4_pre_ciclico_vcc_0.set_T_total(
                        self.T_total)
123
124 if __name__ == '__main__':
125         parser = OptionParser(option_class=eng_option, usage="%prog: [
                options]")
126         (options, args) = parser.parse_args()
127         tb = ofdm_transmitter()
128         tb.Run(True)
```