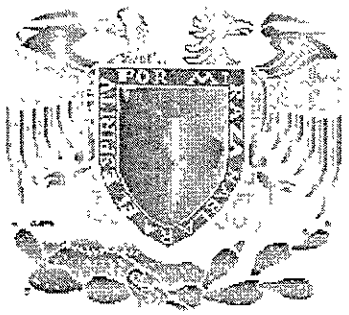
4



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ingeniería

COMUNICACIONES INALÁMBRICAS CON MULTIPLEXACIÓN POR DIVISIÓN ORTOGONAL DE FRECUENCIAS

T E S I S

QUE PARA OBTENER EL TÍTULO DE :

INGENIERO EN TELECOMUNICACIONES

P R E S E N T A N

GIANFRANCO BILIOTTI RODRÍGUEZ

HIRAM EDUARDO GALICIA LUNA

Director: M. en C. Amanda Oralia Gómez González

México, D.F. 2001

Contenido

Agradecimientos

Gianfranco Biliotti

- A mis padres, por impulsarme siempre a ser mejor, por enseñarme a nunca quedarme en el camino y por ayudarme a cumplir poco a poco las metas que me he propuesto.
- A mi hermana, por quererme hasta las últimas consecuencias y ser mi cómplice en las buenas y en las malas.
- A mi abuelo José María, un ejemplo para mí por su fortaleza y energía.
- A tutta la famiglia Italiana, che nonostante la distanza, si trova sempre nei miei pensieri.
- A todos los que me quieren y apoyan, que son importantes para mí ahora y lo serán siempre.

Este trabajo es para ustedes.

Agradecimientos

Hiram Eduardo Galicia Luna

Quiero agradecer:

- A mi mamá Margarita y a mi hermana Lupita por el apoyo que me han dado a lo largo de mis estudios y por el cariño con que siempre me han consentido.
- A mis tíos José, Clemente, Leoncio, Alejandro y Lorenzo por darme un buen consejo todas las veces que lo necesité.
- A mis tías Rosa, Graciela y Magdalena y a mis primos Iván, Gabriela y Roberto, por que me han animado en todas las etapas de mi vida.
- A mi abuelo Clemente por que siempre ha tenido tiempo para conversar conmigo.
- A mis amigos Jaime, Ramiro, Alejandro, Emmanuel y Beatriz por compartir tantos ratos buenos y malos durante toda la carrera.
- A Isaac por su amistad sincera y su valiosa colaboración para la realización de esta tesis.
- A mis maestros y compañeros de la facultad que me ayudaron en mi formación profesional.
- A mis tutores Enrique Arenas, Miguel Moctezuma y Jesús Savage por que siempre estuvieron dispuestos a obsequiarme un poco de su atención.
- A Ismael Magaña por su paciencia y el apoyo que siempre me ha dado durante mi estancia laboral.
- A Ángela Trejo por su conversación amena y todas las consideraciones que me ha otorgado.

A Kb/TEL Telecomunicaciones por todas las facilidades que me ha brindado para concluir este estudio.

- A la Facultad de Ingeniería que ha sido un segundo hogar para mí.

Introducción

El propósito principal de esta tesis es presentar una visión general del esquema de modulación/multiplexación OFDM (*Orthogonal Frequency Division Multiplexing*, Mutiplexación por División Ortogonal de Frecuencias), en qué consiste y cómo opera. Además, se propone una herramienta realizada en lenguaje de programación C para simular el funcionamiento de un modulador y un demodulador basados en las recomendaciones del estándar IEEE 802.11a para transmisiones inalámbricas terrestres en la banda de 5 GHz.

Este estudio representa un primer esfuerzo para mostrar el funcionamiento y desempeño de un sistema OFDM, el cual posteriormente puede servir como base para su realización física. Los resultados reportados representan una aproximación del desempeño real que se puede obtener al implementar el sistema. En las simulaciones se tomaron en cuenta algunos factores físicos que se presentan en un enlace real; sin embargo, existen otros factores que no se simularon, como ruido debido a ciertos componentes (osciladores, antenas, cuantizadores), compresión de potencia debido a los amplificadores, entre otros. Parte del estudio menciona algunos algoritmos para la realización práctica de un sistema.

En el presente estudio se analizan los siguientes aspectos: el primer capítulo muestra un panorama de las tendencias actuales de las telecomunicaciones, junto con la descripción de algunos sistemas estandarizados que utilizan la modulación OFDM, mostrando las características que los definen, sus ventajas y aplicaciones. En el segundo capítulo se explican las bases del funcionamiento de OFDM y los principios teóricos que rigen a esta técnica. El tercer capítulo muestra cómo la modulación y la codificación de las señales de OFDM ayudan a que sea un sistema robusto, disminuyendo los efectos de canales con trayectorias múltiples, permitiendo la corrección de errores o minimizando su efecto por medio de técnicas como el intercalamiento de bits. En el capítulo cuarto se trata el tema de la sincronización, su importancia y técnicas para poder realizarla. En el capítulo quinto se describe el modelo de simulación utilizado así como los procesos de la señal OFDM y los resultados obtenidos de las simulaciones. En el capítulo sexto se recomiendan algunas técnicas para facilitar la realización física del sistema. El último capítulo muestra las conclusiones obtenidas en este estudio.

1 Panorama Actual

1.1 *Comunicaciones Móviles de Banda Ancha*

El crecimiento espectacular de las comunicaciones de vídeo, voz y datos sobre Internet, y la demanda igualmente acelerada de la telefonía móvil, justifican las grandes perspectivas para la denominada *Multimedia Móvil*. Las investigaciones y desarrollos están tomando lugar alrededor del mundo para definir los Sistemas de Comunicaciones Multimedia de Banda ancha Inalámbricos de la siguiente generación que pudieran crear la Villa de la Comunicación Global.

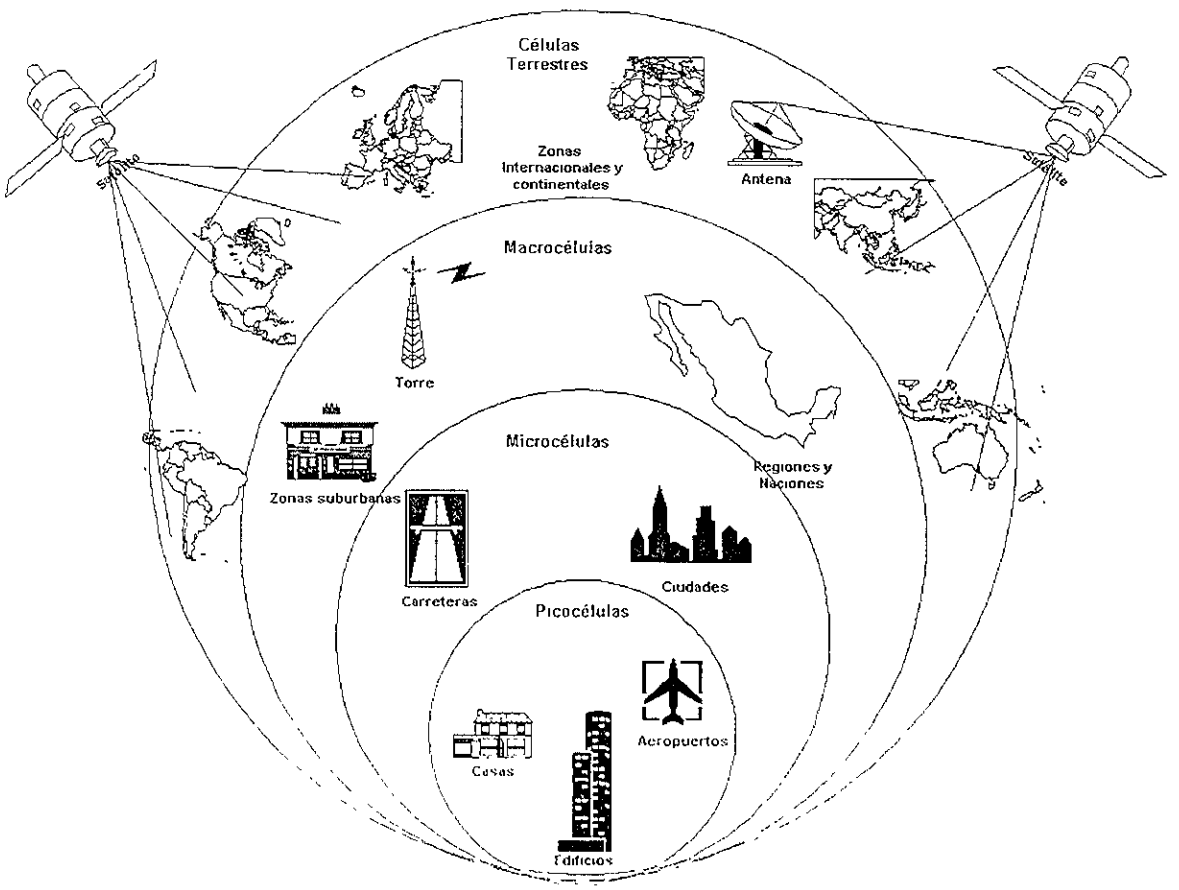


Figura 1.1 Concepto de Villa de la Comunicación Global.

La **figura 1.1** muestra el concepto básico de la Villa de la Comunicación Global, la cual se compone de varios elementos a diferentes escalas, desde la global hasta la picocelular. Como sabemos, la demanda de comunicaciones móviles inalámbricas y comunicaciones Internet/Multimedia está creciendo exponencialmente, por lo tanto, es necesario que ambas soluciones se desarrollen juntas. Así, en un futuro cercano, el Protocolo de Internet (IP) inalámbrico y el ATM inalámbrico jugarán un papel importante en el desarrollo de los Sistemas Inalámbricos de Banda ancha.

Mientras que los sistemas de comunicaciones de hoy se diseñan para una aplicación específica, tal como voz en un teléfono móvil o datos en una Red de Área Local Inalámbrica (WLAN), la siguiente generación de Sistemas Inalámbricos integrarán varias funciones y aplicaciones. Se espera que éstos sistemas provean a los usuarios servicios con tasas de hasta 2 Mbps. El soportar tal velocidad de transferencia de información con suficiente confiabilidad para los impedimentos de los canales de radio requiere una cuidadosa selección de factores técnicos como modulación, codificación, método de acceso, etc. El objetivo de los nuevos sistemas inalámbricos es *proveer a los usuarios un medio de acceso inalámbrico a servicios de banda ancha* soportados en redes privadas u ofrecidos directamente por las redes públicas fijas.

Éstos sistemas están en investigación en Estados Unidos, Europa y Japón en las bandas de onda milimétrica y de microondas y su desarrollo ha llamado mucho la atención debido al incremento de las aplicaciones multimedia y de transferencia de datos.

Existen principalmente 3 áreas de desarrollo:

1. Accesos fijos en redes públicas en el espectro de microondas y ondas milimétricas.
2. Evolución de Redes de Área Local Inalámbrica (WLAN) para sistemas dentro de edificios.
3. Uso de tecnologías LAN en exteriores.

En poco tiempo éstos sistemas proveerán servicios novedosos de multimedia y vídeo móvil, así como otros relacionados con Redes Privadas Inalámbricas (WCPN, *Wireless Customer Premises Network)* y de Lazo Local Inalámbrico (WLL, *Wireless Local Loop)*. Para desarrollar los sistemas de comunicación inalámbricos de banda ancha se deben considerar los siguientes aspectos:

- Selección y atribución de frecuencias.
- Caracterización del canal
- Reconocimiento de la aplicación.

- Desarrollo tecnológico.
- Técnicas de acceso de interfaz aérea.
- Redes y protocolos.
- Desarrollo de sistemas con modulación eficiente, codificación y técnicas de antenas inteligentes.

Se están poniendo en marcha un número significativo de proyectos de investigación y desarrollo para los sistemas inalámbricos de la siguiente generación. Dentro de los programas del ACTS (*European Advanced Communications Technologies and Services)* existen cuatro proyectos de la Unión Europea: Demostrador de Red ATM Inalámbrica (Magic WAND, *Wireless ATM Network Demonstrator*), Sistema de Comunicación de Acceso Inalámbrico ATM (AWACS, *ATM Wireless Access Communication System),* Sistema para Aplicaciones Avanzadas Móviles de Banda ancha (SAMBA, *System for Advanced Mobile Broadband Applications)* y CPN/LAN (*Customer Premises Network*) inalámbricos de banda ancha para aplicaciones residenciales multimedia.

Las comunicaciones multimedia y por computadora están jugando un papel muy importante en la sociedad actual creando nuevos retos en el área de desarrollo de los sistemas de comunicación. Varios sistemas están siendo considerados para diferentes necesidades. Estos se acomodan en tasas de datos de entre 2 y 155 Mbps, las terminales pueden ser móviles o portátiles, la velocidad de movimiento puede ser tan rápida como la de un tren de alta velocidad, los usuarios pueden o no utilizar más de un canal si su aplicación lo requiere, el ancho de banda del sistema se asigna de manera fija o dinámica de acuerdo a las necesidades del usuario, las comunicaciones entre las terminales pueden ser directas o pueden enlazarce hacia una estación base, es posible utilizar tecnología ATM, etc. Se pueden mencionar muchos otros casos a manera de delimitar las perspectivas para un sistema inalámbrico de banda ancha, pero dos principales desarrollos están emergiendo: IEEE 802.11 e HIPERLAN/2, los cuales son estándares de Red de Área Local Inalámbrica dirigidos a la transferencia de datos entre computadoras.

Por experiencia podemos afirmar que cuando se desarrolle un método que permita incrementar la calidad y cantidad de servicios de comunicaciones para el usuario, de tal manera que en un momento dado cubra las necesidades del mismo, surgirán nuevas necesidades y retos que obligarán a continuar investigando y desarrollando nuevos métodos para satisfacer las necesidades que surgen día con día.

El crecimiento en el ancho de banda requerido por los usuarios y la tendencia mundial hacia la movilidad de los sistemas requiere una adaptación de los sistemas de multiplexación y modulación en las redes terrestres y también satelitales. En este último caso las grandes distancias de transmisión y las altas capacidades de información requieren de sistemas robustos, poco susceptibles al ruido, con esquemas de codificación y corrección de errores adecuados, buscando obtener la máxima eficiencia en el uso de la potencia y el espectro y el abatimiento de costos. Debido a esto es necesario buscar nuevas alternativas para su aplicación en los sistemas satelitales. La Multiplexación por División Ortogonal de Frecuencias (OFDM, *Orthogonal Frequency Division Multiplexing*) es una de estas novedosas alternativas.

1.2 Sistemas Estandarizados

El concepto de Red de Área Local Inalámbrica, tal como el del estándar IEEE 802.11, y de un sistema celular móvil de banda ancha, como MBS, es totalmente diferente: cada uno está dirigido a servicios y aplicaciones que difieren en muchos aspectos. Una comparación de varios sistemas, basados en dos características clave (movilidad y tasa de datos), se muestra en la **figura 1.2**, donde es claro que no existe competencia entre los diferentes sistemas.

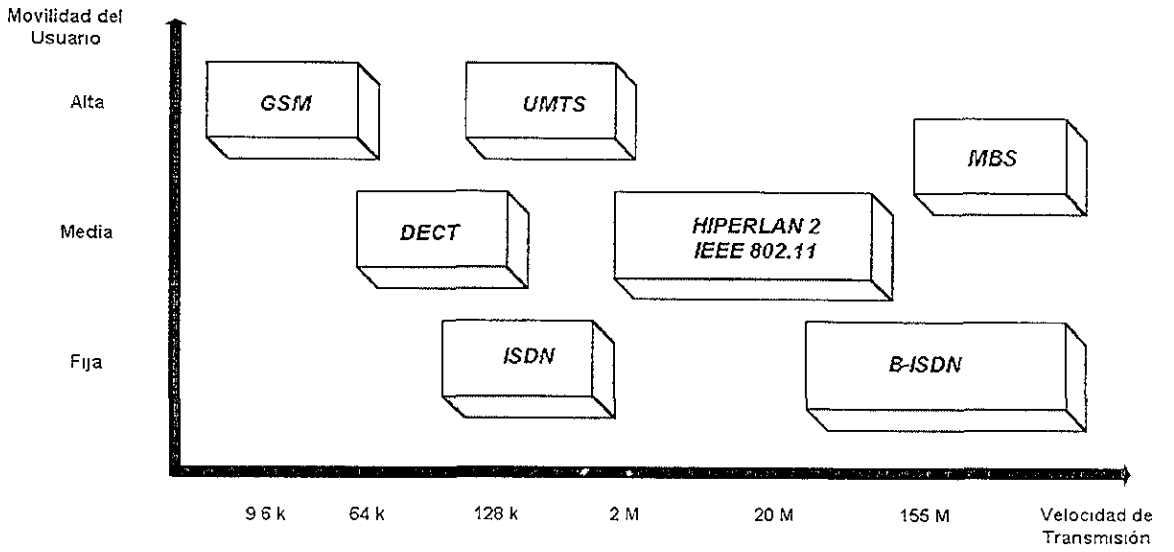


Figura 1.2 Comparación de movilidad y tasas de datos para varios sistemas

Las aplicaciones y servicios de los sistemas ilustrados también son diferentes, IEEE 802.11 se dirige principalmente a la transferencia de datos entre computadoras (siendo así una extensión de las LAN's alámbricas), aunque puede soportar voz en tiempo real y señales de imagen, y los usuarios cuentan con cierta movilidad y pueden tener acceso a redes públicas.

En las siguientes secciones se muestran las características técnicas generales de algunos sistemas estandarizados que se basan en la modulación digital multiportadora OFDM.

1.2.1 DAB (Digital Audio Broadcasting) – DARS (Digital Audio Radio Satellite)

Los estándares DAB-S (*Digital Audio Broadcasting-Satellite*, Radiodifusión Digital de Audio por Satélite) y DARS (*Digital Audio Radio Satellite*, Radiodifusión Digital de Audio Satelital) son sistemas de radiodifusión que realizan la transmisión desde la emisora hasta el receptor de una forma completamente digital. La idea de DAB surge a partir del proyecto Eureka 147, que actualmente se encuentra estandarizado por el Instituto Europeo de Estándares de Telecomunicaciones en el documento EN-300-401. Proporciona los medios necesarios para proveer al usuario un servicio de radio digital de alta calidad.

Las características principales que hacen de este sistema un avance significativo para la radiodifusión son principalmente:

- Señales digitales de audio de alta calidad, comparable con la de un CD.
- Transmisión robusta y confiable hacia receptores fijos, portátiles y móviles sin interferencia.
- Uso eficiente del espectro debido a la modulación OFDM
- Flexibilidad y servicios adicionales temporales o permanentes.
- Capacidad de introducir nuevos servicios como texto, gráficas, etc.

Los principales componentes técnicos de este sistema son:

- La codificación de audio MUSICAM (*Masking pattern Universal Sub-band Integrated Coding And Multiplexing*) que emplea el método de codificación pseudoacústica el cual consiste en eliminar ciertas frecuencias que no son audibles para el ser humano, como se utiliza en el estándar MPEG2, pudiéndose obtener tasas desde 8 hasta 384 kbps. Esto permite mantener una alta calidad en la señal de audio utilizando una menor cantidad de información.
- La codificación y multiplexación de transmisión, cuya función es la de combinar la información de los diferentes servicios de DAB en una sola cadena lista para su transmisión. Comprende tres etapas principalmente: el *canal de sincronización*, que provee información

El valor agregado de este formato de transmisión para las empresas de radiodifusión es que en una misma señal analógica se pueden transmitir diferentes programas (canales de televisión y de audio) con una alta definición y otros servicios, en un ancho de banda reducido debido a la compresión.

DVB no es el único estándar existente para la transmisión digital de vídeo. La compañía General Instruments creó un estándar llamado Digicipher 2 (DCII) que es el sistema de distribución utilizado para su producto de televisión digital 4DTV. Este estándar es único para sus productos, sin embargo está siendo adoptado por el ATSC (*Advanced Television Systems Committee*) como estándar para reemplazar el sistema NTSC en países como Estados Unidos, Canadá, Corea del Sur, Taiwan y México. Este estándar también está basado en MPEG-2, sin embargo existen diferencias que lo hacen incompatible con DVB, sobre todo en la parte de adquisición de datos y en la capa física. Un ejemplo de estas diferencias es que utilizan codificación de audio Dolby AC3 en vez de MUSICAM.

Los problemas que conllevan estas diferencias consisten en que no existe un estándar mundial para la transmisión de vídeo y es necesario fabricar receptores híbridos que sean capaces de soportar ambos sistemas. No existe una ventaja clara del DCII sobre el DVB en las pruebas de campo realizadas hasta la fecha, por lo que a primera vista no es justificable adoptar un sistema alternativo y llevar la contra a la tendencia de globalización de estándares de telecomunicaciones que se ha dado en los últimos años. Esto es necesario tomar en cuenta en países como México, que aún no han adoptado completamente un sistema, por lo que es importante considerar las ventajas y desventajas de adoptar una nueva tecnología.

En la **tabla 1.1** se muestran algunas características de la capa física de DCII y DVB.

Característica	*DCII*	*DVB*
Formato de Transmisión	8-VSB	OFDM-QPSK (Satélites y terrestre) OFDM-QAM (Cable y terrestre)
Codificación Reed-Solomon	DCII polinomial	DVB polinomial
Intercalamiento en tiempo	Convolucional I = 12, J = 19	Convolucional I = 12, J = 17
Codificación	Convolucional de longitud 7 (Entralazado Viterbi de 64 estados)	Convolucional de longitud 7 (Entralazado Viterbi de 64 estados)
Forma de Pulsos	Filtrado por medio de una ventana Butterworth	Filtrado por medio de una ventana de raíz cuadrada de coseno elevado

Tabla 1.1 Características de los estándares DCII y DVB.

Como se mencionó anteriormente, existen diferentes estándares de DVB para satélite, cable y transmisión terrestre. Esto se debe a las diferentes problemáticas que presentan los canales de transmisión para cada caso. En un canal de satélite el principal reto a vencer es la atenuación de potencia debido a las grandes distancias que recorre la señal. Sin embargo las variaciones de la potencia son pequeñas excepto en zonas con mucha atenuación debido a lluvias, las multitrayectorias son despreciables y hay menos limitantes en cuanto a ancho de banda. En cambio, en un canal de cable es necesario transmitir en un ancho de banda de 8 MHz para cumplir con las necesidades comerciales y estándares, pero las pérdidas de potencia son mucho menores y los operadores pueden controlar de cierta forma los problemas de intermodulación debido a las etapas de amplificación y los defectos en el medio de transmisión y las terminaciones.

Estas diferencias influenciaron la decisión de modular las subportadoras OFDM con QPSK en las especificaciones para satélite (eficiencia de 2bits/s/Hz) y con 16-QAM y 64-QAM para transmisión por cable (eficiencia de 4 bits/s/Hz y 6 bits/s/Hz, respectivamente). Los problemas que presenta una transmisión terrestre se encuentran asociados a las multitrayectorias, las diferencias de potencia entre las señales y la calidad de las antenas, así como su instalación. Esto justifica la utilización de la modulación OFDM, que es menos sensible a estos problemas que la modulación con una sola portadora.

El futuro de la transmisión de datos por medio del estándar DVB es prometedor ya que existen ventajas importantes sobre las tecnologías analógicas, además de que ya está siendo adoptado en muchos países. Muestra de ello son los proyectos de satélites digitales que adoptarán este sistema en Europa, Australia, Japón y Tailandia, entre otros. Por ejemplo, en Estados Unidos la compañía ECHOSTAR utiliza los estándares de DVB para transmisión de video con formato MPEG-2. Además DVB es compatible con sistemas como el SMATV (Satellite Master Antenna TeleVision) que actualmente cuenta con más de 37 millones de usuarios en Europa, 2 millones más que los usuarios de televisión por cable [3].

1.2.3 HIPERLAN/2 (High Performance Local Area Network) e IEEE 802.11

El estándar de Red de Área Local de Alto Desempeño (HIPERLAN/2, *High Performance LAN*) es una tecnología prometedora para redes de área local inalámbrica. Ha sido recientemente aprobada por el proyecto BRAN (*Broadband Radio Access Networks*) dentro del Instituto Europeo de Estándares para Telecomunicaciones. Los primeros productos podrían estar disponibles en el año 2002. El HIPERLAN/2 será el primer estándar de LAN inalámbrica con soporte completo de calidad de servicio (QoS) para diferentes tipos de redes, junto con interoperabilidad de dispositivos de diferentes fabricantes.

La demanda de ancho de banda en redes privadas ha crecido de manera importante durante los últimos años y, al mismo tiempo, la conectividad inalámbrica se ha vuelto cada vez más usada. Otro punto importante son las redes de comunicaciones en las que converjan datos y voz. Estos desarrollos se han convertido en requerimientos para redes inalámbricas de banda ancha capaces de soportar aplicaciones multimedia. La red HIPERLAN/2 ha sido diseñada para cumplir con estas expectativas.

Existen varios productos de diferentes fabricantes basados en el estándar 802.11 para redes de área local inalámbricas. Dependiendo del esquema de transmisión, estos productos pueden ofrecer anchos de banda desde 1 Mbps hasta 11 Mbps. Se espera que los precios bajen, convirtiendo a las WLAN en una alternativa seria para redes fijas Ethernet. Las redes inalámbricas del futuro, como la conceptualizada en HIPERLAN/2, incluyen soporte de calidad de servicio (para construir redes multiservicio), seguridad, handover entre áreas locales y amplias, así como conectividad a redes públicas y privadas y capacidades incrementadas para un mejor desempeño en transferencia de datos y aplicaciones de vídeo.

Una red HIPERLAN/2 típicamente tiene una topología como la de la **figura 1.4**. Las Terminales Móviles (MT) se comunican con los Puntos de Acceso (AP) sobre una interfaz definida en el estándar. Existe también un modo directo de comunicación entre dos MT's que aún está desarrollándose y que no tiene un estándar. El usuario de una MT se puede mover libremente en la red HIPERLAN/2, la cual asegurará que la MT tengan la mejor calidad de comunicación posible.

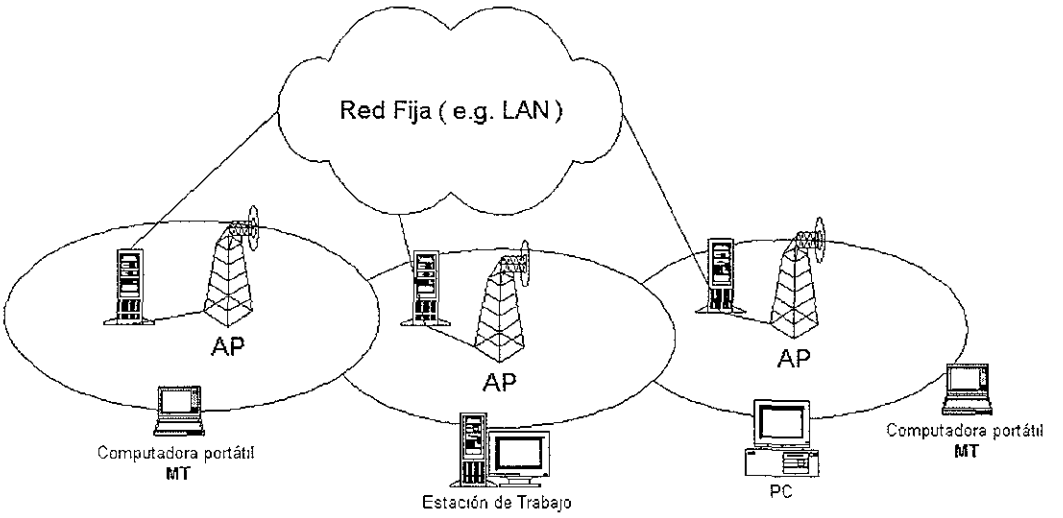


Figura 1.4 Configuración de una red HIPERLAN/2.

Las características generales de la tecnología HIPERLAN/2 se pueden resumir como:

- Transmisión de alta velocidad
- Orientado a conexiones
- Soporte de Calidad de servicio (QoS)
- Locación automática de frecuencias
- Soporte de seguridad
- Soporte de Movilidad
- Independencia entre la red y las aplicaciones

La **tabla 1.2** mostrada abajo resume las características de 802.11, 802.11b, 802.11a, e Hiperlan/2.

Característica	*802.11*	*802.11b*	*802.11ª*	*HiperLAN2*
Espectro	2.4 GHz	2.4 GHz	5 GHz	5 GHz
Vel. Máxima (C. Física)	2 Mb/s	11 Mbit/s	54 Mb/s	54 Mb/s
Vel Máxima, (Capa 3)	1 2 Mb/s	5 Mb/s	32 Mb/s	32 Mb/s
Control de Acceso Medio/ Partición de Medios	Sentido de portadora	CSMA/CA	CSMA/CA	Control Central de Recursos/ TDMA/TDD
Selección de Frecuencias	Saltos de frecuencia/ DSSS	DSSS	Portadora Única	Portadora Única con Selección Dinámica de Frecuencia
Soporte de redes fijas	Ethernet	Ethernet	Ethernet	Ethernet, IP, ATM, UMTS, FireWire, PPP

Tabla 1.2 Comparación entre diferentes tecnologías para WLAN.

A pesar de las grandes ventajas que presenta el estándar Hiperlan/2 con respecto al IEEE 802.11a, decidimos optar por simular un sistema basado en éste último debido a que en el mercado existen productos de comprobada aceptación basados en la misma familia (IEEE 802.11 e IEEE 802.11b). Esto representa para los fabricantes e integradores de equipos un menor costo en la actualización de sus productos, ya que solamente deben adaptar la capa física propuesta en IEEE 802.11a con las especificaciones de Capa de Enlace de Datos y de Control de Acceso al Medio que ya tienen implementadas. Aparte de representar un ahorro en costos, se traduce en una respuesta mas rápida de mercado, ya que para desarrollar productos nuevos basados en el estándar Hiperlan/2 se necesita un periodo de estudio y depuración, tanto de hardware como de software, que puede provocar un retardo considerable en la comercialización de los productos.

1.3 Televisión Digital

La tecnología digital ha permitido incrementar el número de servicios que se ofrecen a los consumidores, ofreciendo ventajas sobre las tecnologías analógicas. En el caso específico de la televisión, la tecnología digital le permite a los usuarios más opciones de programación, mejor calidad en las imágenes, capacidad para utilizar formatos de imagen ampliados (Wide Screen), guías de programación electrónicas y televisión de alta definición. En un futuro no muy lejano se espera contar con servicios de vídeo por demanda, acceso a Internet, compras electrónicas, etc.

Para poder ofrecer toda esta gama de nuevos servicios es necesaria una mayor capacidad de transmisión. Esto se ha logrado gracias a técnicas de compresión y nuevas tecnologías para su transmisión. Es una decisión importante la que tienen que tomar los países al escoger qué tipo de modulación utilizaran para estandarizar estos servicios. En el caso de la televisión digital han sobresalido dos estándares capaces de ofrecer servicios digitales: El estándar de ATSC, basado en la tecnología 8-VSB, y el DVB, basado en OFDM. El primero ha sido adoptado por los E.E.U.U. y Canadá (ambos países están reconsiderando esa decisión) mientras que DVB ha sido la opción de la mayoría de los países, especialmente en Europa, donde fue creada la comisión de DVB. En la **figura 1.5** se puede ver con claridad que el estándar basado en OFDM ha tenido una mayor aceptación a nivel mundial. Esto de se debe a que presenta ciertas ventajas importantes con respecto a la tecnología 8-VSB de portadora única. En la **tabla 1.3** se resumen las ventajas más significativas de cada sistema con respecto a su competidor. Estas ventajas son las que han ayudado a los países y a la industria a decidir entre los dos estándares mencionados.

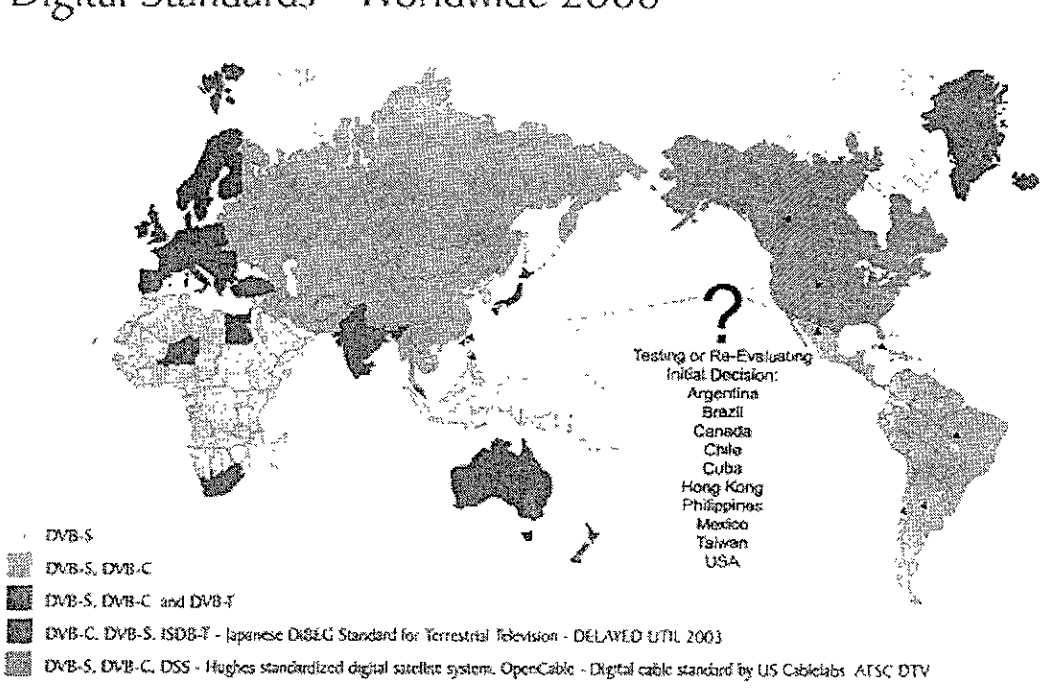


Figura 1.5. Países que han adoptado la tecnología DVB basada en OFDM

Ventaja	*OFDM*	*8-VSB*	*Comentarios*
Movilidad	X		OFDM presenta menos sensibilidad a multitrayectorias (eco)
Ancho de banda		X	8-VSB puede transmitir hasta 19.4 Mb/s en un ancho de banda de 6 MHz
Antena	X		DVB soporta antenas omnidireccionales sencillas, que no necesitan instalación profesional
Sensibilidad	X		OFDM es menos sensible a cambios en el canal de transmisión
Receptores	X		Es más robusto OFDM para receptores en internos (e.g "Set top boxes")
Potencia		X	Menor potencia de transmisión necesaria en 8-VSB
Interferencia		X	8-VSB presenta una menor interferencia con los sistemas analógicos existentes
Recepción en baja potencia	X		OFDM soporta en DVB modos jerárquicos que permiten la recepción de la señal en calidad estándar cuando la potencia recibida es baja
Transmisoras	X		OFDM soporta transmisiones desde múltiples emisoras sin que se observen "fantasmas" en la imagen
Flexibilidad	X		OFDM es más flexible, por lo que es más probable que se acople a necesidades futuras de nuevos servicios

Tabla 1.3 Ventajas competitivas de los sistemas OFDM y 8-VSB para televisión digital

1.4 Ventajas de OFDM sobre sistemas de 3ª generación en movilidad baja

En la actualidad los sistemas de 3ª generación (3G) para comunicación inalámbrica están siendo desarrollados como una evolución de la 2ª generación (2G). Esta evolución consiste en la introducción de servicios de datos y otras mejoras, por ejemplo en las técnicas de codificación de voz. Las tasas de datos que se espera obtener van desde los 384 kbps hasta los 2 Mbps.

Los usuarios de los servicios de telecomunicaciones demandan cada vez más ancho de banda, además de nuevos servicios como voz sobre IP. Las tecnologías fijas para Internet como xDSL ofrecen altas tasas de transmisión (>Mbps) a los usuarios, quienes aceptarían con dificultad regresar a una conexión lenta a la red, ya que la velocidad de respuesta de ésta afecta directamente la productividad de sus usuarios. Los nuevos servicios requieren de una calidad de servicio (QoS) mayor. Esto implica que los requerimientos de los sistemas son más exigentes en cuanto a tasas de transmisión y disponibilidad.

Los sistemas 3G están basados en la tecnología CDMA, que presenta un compromiso entre ganancia de procesamiento e interferencia. Esta propiedad es útil para tasas de transmisión baja como las que se utilizan en las comunicaciones celulares móviles, en términos de reuso de frecuencias. Sin embargo, para aumentar la tasa de transmisión efectiva su capacidad de rechazar interferencias se disminuye considerablemente. En la actualidad los sistemas de 3G operan con una tasa de transmisión relativamente baja pero muy robusta. Los sistemas CDMA tienen entonces dificultad en alcanzar una QoS aceptable cuando se transmiten tasas altas de información, cosa que no sucede en los sistemas OFDM. Otra desventaja que presenta la tecnología CDMA es que no puede operar de manera satisfactoria con varios usuarios de banda ancha al mismo tiempo, lo que limita su capacidad de ofrecer tasas de transmisión lo suficientemente altas en horas pico.

Los principios de diseño de la tecnología CDMA están optimizados para servicios de voz en los que todos los usuarios requieren de una misma tasa de transmisión, cosa que no sucede en la transmisión de datos. OFDM en cambio tiene estándares específicos para redes de banda ancha (e.g. IEEE 802.11a e Hiperlan/2) diseñados para tasas de datos variables.

Otras ventajas importantes que presenta OFDM sobre los sistemas de 3G en caso de movilidad baja son su menor susceptibilidad a multitrayectorias, errores de tiempo e interferencias de banda angosta. Además, los sistemas OFDM son más sencillos de implementar en la práctica. Las desventajas de OFDM comparado con 3G son principalmente la sensibilidad a errores de frecuencia, ruido de fase (explicados posteriormente en este estudio) y las no linealidades que se pueden presentar en las etapas de amplificación debido a la mayor relación potencia pico/potencia promedio.

utilización para tecnologías inalámbricas terrestres (objeto de este estudio), satelitales o alámbricas.

Otra ventaja significativa que presenta esta tecnología es que se pueden utilizar componentes disponibles en el mercado, debido a la gran capacidad de procesamiento con la que se cuenta en la actualidad a un precio razonable. Nuestro objetivo y justificación tecnológica es poder realizar simulaciones que nos muestren la viabilidad de un sistema OFDM que nos ayude a transmitir más información en un menor ancho de banda, aunado a otras ventajas propias del sistema como la capacidad de disminuir el efecto de múltiples trayectorias y la sensibilidad a errores de tiempo, para que en estudios posteriores sirva como base para su realización práctica, teniendo una aproximación teórica de los resultados que se pueden obtener.

2 Características de OFDM

2.1 *Visión General*

El principio básico de OFDM es dividir un flujo de datos de alta velocidad en varios flujos con menor tasa de datos los cuales se transmiten simultáneamente sobre subportadoras a diferentes frecuencias. Como la duración del símbolo se incrementa al tener subportadoras paralelas de más baja velocidad, la cantidad de dispersión relativa en el tiempo causada por el retraso de las multitrayectorias disminuye (**figura 2.1**). La interferencia intersímbolo (ISI, *InterSymbol Interference*) se elimina casi completamente mediante la introducción de un intervalo de guarda antes de cada símbolo OFDM. Durante el intervalo de guarda el símbolo se extiende cíclicamente para evitar la interferencia intersímbolo como se explicará mas adelante.

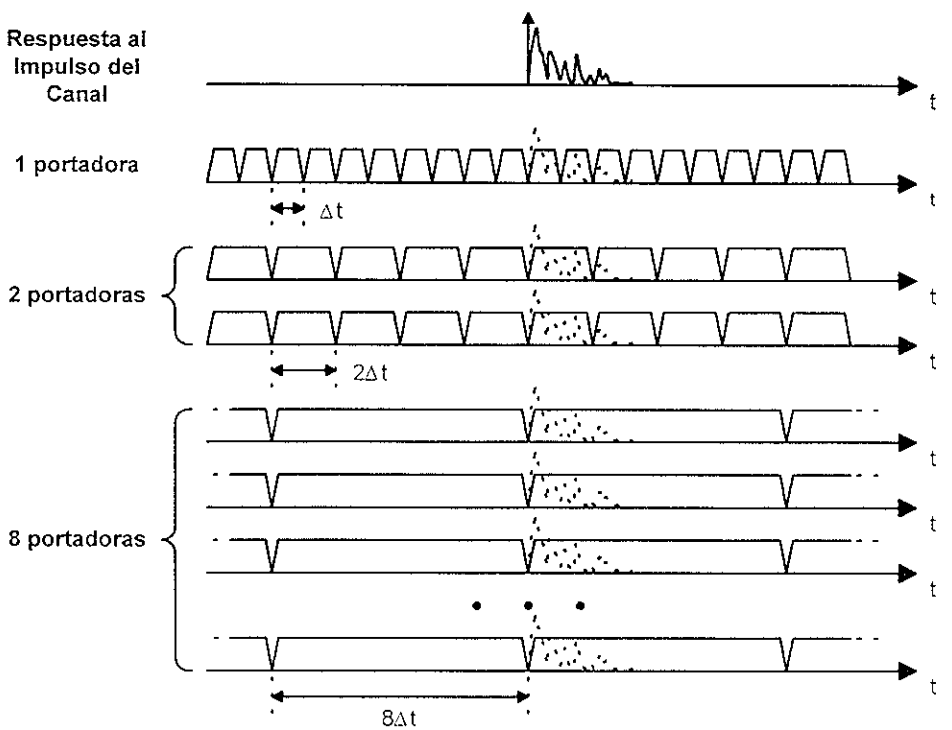


Figura 2.1 El efecto de adoptar un sistema multiportadora. La interferencia intersímbolo afecta sólo a un pequeño porcentaje de cada símbolo cuando se incrementa el número de portadoras

Para diseñar un sistema OFDM se deben tomar en consideración los siguientes parámetros: número de subportadoras, duración del símbolo, duración del intervalo de guarda, espaciamiento entre subportadoras, tipo de modulación por subportadora y tipo de codificación para corrección de errores. La elección de éstos parámetros es influenciada por los requerimientos del sistema tales como tasa de bits, ancho de banda, tolerancia a las multitrayectorias y tolerancia al efecto Doppler. Algunas exigencias son conflictivas, por ejemplo, para obtener una buena tolerancia al efecto de las multitrayectorias es deseable contar con un gran número de subportadoras con un pequeño espaciamiento entre ellas, pero lo opuesto es deseable para una buena tolerancia en contra del efecto Doppler y el ruido de fase.

OFDM es un caso especial de transmisión por *multiportadora*, donde una sola fuente de datos se transmite sobre un número finito de subportadoras de baja velocidad. Vale la pena mencionar aquí que OFDM puede ser visto tanto como una *técnica de modulación* como una *técnica de multiplexación* ya que una sola o varias fuentes de información pueden modular a las subportadoras. Una de las principales razones para utilizar OFDM es su inherente protección contra el desvanecimiento por selectividad de frecuencias o interferencia de banda angosta. En un sistema con una sola portadora, un solo desvanecimiento o interferencia causa el rompimiento completo del enlace, pero en un sistema de multiportadora sólo será afectado un pequeño porcentaje de subportadoras. La codificación para corrección de errores puede ser usada entonces para corregir las portadoras erróneas. El concepto de utilizar transmisión de datos en paralelo y multiplexación por división de frecuencias fue publicado en la década de los 60's [4]. La primera patente norteamericana fue registrada en 1970.

En los 60's, la técnica OFDM se utilizó en varios sistemas militares de alta frecuencia tales como KINEPLEX, ANDEFT y KATHRYN. Por ejemplo, el módem de tasa de datos variable en KATHRYN fue construido para la banda de alta frecuencia. Este usaba hasta 34 canales paralelos modulados con una tasa baja y un espaciamiento de 82 Hz entre subportadoras. En los 80's, OFDM fue estudiado para desarrollar módems de alta velocidad, comunicaciones digitales móviles y grabaciones de alta densidad. En los 90's, OFDM fue explotado para comunicaciones de banda ancha, sobre canales de radio FM, Líneas Digitales de Abonado de tasa alta (HDSL, *High-bit-rate Digital Suscriber Line)* a 1.6 Mbps, ADSL (6 Mbps), VDSL (100 Mbps), Radiodifusión de Audio Digital (DAB, *Digital Audio Broadcasting)* y Radiodifusión de

Video Digital Terrestre (DVB-T, *Digital Video Broadcasting*). El esquema de transmisión OFDM tiene las siguientes ventajas clave:

- OFDM es una manera eficiente de abatir los efectos de las multitrayectorias. Para una distribución de retardos dada, por ejemplo la de la figura 2.2, la complejidad de implementación es significativamente menor comparada con la de un sistema de portadora única con un ecualizador.
- OFDM es un esquema robusto contra la interferencia de banda angosta porque dicha interferencia afecta solo un pequeño porcentaje de las subportadoras.
- OFDM hace posible implementar Redes de Frecuencia Única (SFN, *Single Frequency Networks*) lo cual es especialmente atractivo para aplicaciones de difusión masiva.

Por otro lado, OFDM también tiene algunas desventajas comparado con los esquemas de portadora única:

- OFDM es más sensible a las desviaciones de frecuencia y el ruido de fase.
- OFDM tiene una gran relación de *potencia pico a potencia promedio*, lo que tiende a reducir la eficiencia del amplificador de radiofrecuencia.

Antes de proceder con un estudio más detallado del funcionamiento de este esquema se expondrá el modelo del canal utilizado en las simulaciones de esta tesis.

2.2 *Modelo del canal con mulitrayectorias*

Los resultados de los trabajos realizados en materia de estudio y modelado de canales con multitrayectoria [5], los cuales son corroborados por mediciones reportadas en la literatura [6, 7], concluyen que la potencia promedio recibida en una multitrayectoria decae exponencialmente con respecto a su retardo. Esto ha permitido simplificar en gran medida los modelos que presentan este efecto. Estos modelos asumen un número fijo de trayectorias con retrasos equidistantes. Las amplitudes de las trayectorias son variables independientes de Rayleigh mientras que las fases de las trayectorias tienen una distribución de probabilidad uniforme. La **figura 2.2** muestra un ejemplo de perfil de un canal con multitrayectorias. La potencia media se representa con la línea continua y la potencia instantánea se representa con la línea punteada.

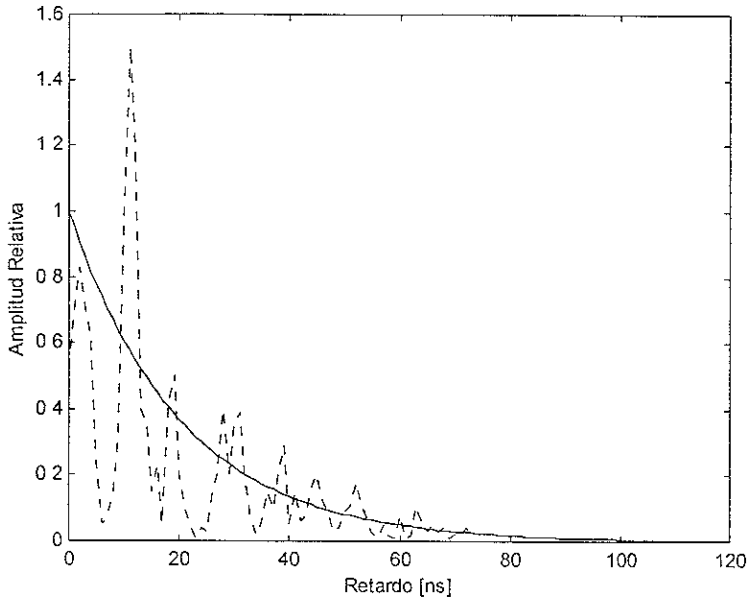


Figura 2.2 Ejemplo de perfil de un canal con multitrayectorias.

2.3 Ortogonalidad

En un sistema clásico de datos en paralelo la banda total de frecuencias se divide en N subcanales los cuales se encuentran sin traslapes. Cada subcanal se modula por separado y luego los N subcanales son multiplexados en frecuencia (FDM). Bajo este esquema es necesario evitar el traslape espectral de los subcanales para prevenir la *interferencia intercanal*; Sin embargo, esto conlleva a un uso ineficiente del espectro disponible debido a la introducción de *bandas de guarda*. Para contrarrestar esta ineficiencia, las ideas propuestas a mediados de la década de los 60's fueron utilizar datos en paralelo y FDM con subcanales traslapados, en donde cada portadora ocupa un ancho de banda B y las portadoras están espaciadas $B/2$ en el dominio de la frecuencia para evitar el uso de ecualización de alta velocidad, combatir el ruido impulsivo y la distorsión por multitrayectoria, así como incrementar la eficiencia en el uso del ancho de banda disponible.

La **figura 2.3** muestra la diferencia entre la técnica convencional de multiportadoras no traslapadas y la técnica de modulación con multiportadoras traslapadas. Utilizando la última es posible ahorrar casi el 50% del ancho de banda. Sin embargo, para implementar esta técnica es necesario anular la interferencia entre las subportadoras, es decir, que las subportadoras sean *ortogonales* entre sí

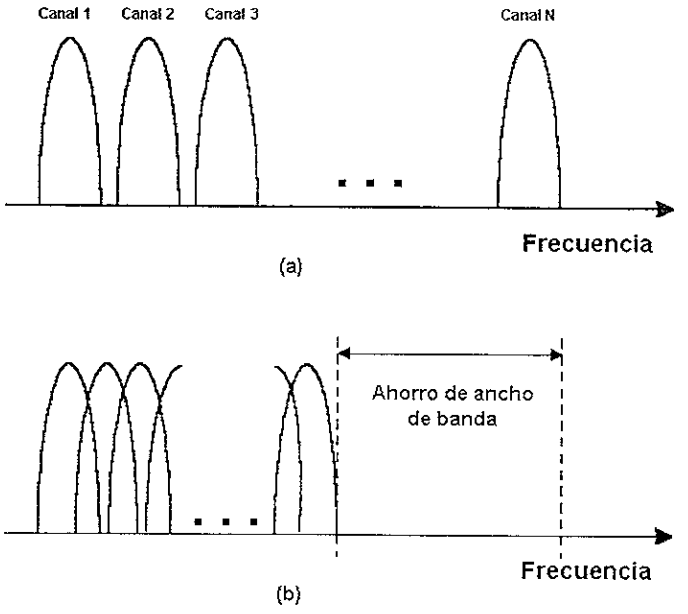


Figura 2.3. Concepto de la señal OFDM: (a) Técnica convencional de multiportadora FDM. (b) Técnica de modulación con multiportadoras ortogonales.

La palabra *ortogonal* indica que existe una relación matemática específica entre las frecuencias de las portadoras en el sistema. En un sistema de multiplexación por división de frecuencias (FDM) normal, las portadoras son espaciadas de tal manera que las señales puedan ser recibidas utilizando filtros y demoduladores convencionales por lo que son introducidas *bandas de guarda* entre portadoras en el dominio de la frecuencia.

Sin embargo, es posible ordenar las subportadoras en un sistema OFDM de tal manera que sus bandas laterales se traslapen y aún así sean recibidas sin interferencia de portadora adyacente. Para llevar a cabo esto las subportadoras deben ser matemáticamente ortogonales. El receptor actúa como un banco de demoduladores, trasladando cada portadora nuevamente a la banda base. Si las frecuencias de las subportadoras fueran seleccionadas de tal manera que en el dominio del tiempo tuvieran un número entero de ciclos en el periodo del símbolo T [seg] (**figura 2.6**), entonces el proceso de demodulación resultaría en una contribución nula por parte de todas las portadoras, excepto de la que se desea demodular. De esta manera se consigue que las portadoras sean ortogonales, siendo el espaciamiento entre subportadoras múltiplo de 1/T [Hz].

Matemáticamente, se tiene un conjunto de señales Ψ donde Ψ_p es el p-ésimo elemento del conjunto. Las señales son ortogonales si el producto interno:

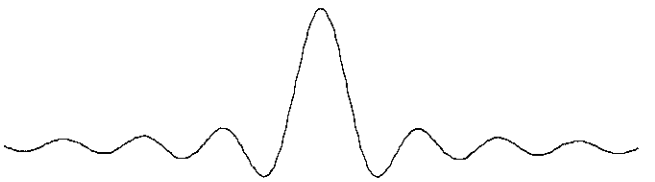
$$\int_0^{\phi} \Psi_p(t)\Psi_q^*(t)dt = K \quad \text{para } p = q \qquad (2.1)$$

$$= 0 \quad \text{para } p \neq q$$

donde '*' indica el complejo conjugado. Mucha de la teoría de las transformaciones utiliza series ortogonales. Las series de Fourier son un ejemplo bien conocido de series ortogonales aunque no son las únicas.

En 1971, Weinstein y Ebert aplicaron la *Transformada Discreta de Fourier* (DFT, *Discrete Fourier Transform*) a los sistemas de transmisión de datos en paralelo como parte del proceso de modulación y demodulación, logrando obtener subportadoras ortogonales [4]. La **figura 2.4** (a) muestra el espectro de una sola subportadora y la figura 2.4 (b) muestra los espectros individuales multiplexados con un espaciamiento en frecuencia igual a la mitad del ancho de banda de cada subportadora. La figura muestra además que en la frecuencia central de cada subportadora no existe interferencia de las demás subportadoras, por lo tanto, si se utiliza la DFT en el receptor y se calculan los valores de correlación en la frecuencia central de cada subportadora se recobrarían los datos transmitidos sin interferencias. Utilizando la técnica de multiportadora basada en la transformada discreta de Fourier, la multiplexación en frecuencia se logra por procesamiento en banda base, en vez de utilizar filtros pasa banda.

a) Espectro de una Subportadora



b) Espectro de Subportadoras Ortogonales

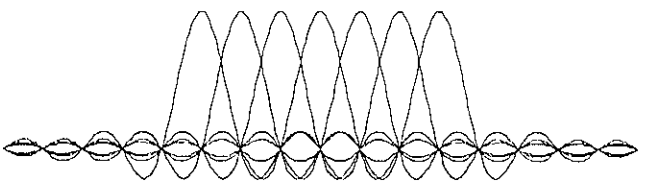


Figura 2.4 Espectro de: (a) una subportadora
(b) varias subportadoras multiplexadas.

La **figura 2.5** describe el principio del modulador OFDM. En la figura se muestran las diferentes fuentes de datos que modularán a las subportadoras. Estas fuentes provienen del flujo principal de datos que se subdivide en varios flujos. Las subportadoras serán generadas por un banco de osciladores cuyas frecuencias se seleccionan para cumplir con la condición de ortogonalidad de la ecuación (2.1). Finalmente las subportadoras moduladas son sumadas para obtener el símbolo OFDM que se transmitirá. Para eliminar los bancos de osciladores de las subportadoras y los demoduladores coherentes requeridos para la recepción se puede construir un sistema completamente digital desarrollando hardware de propósito especial capaz de realizar la ***Transformada Rápida de Fourier*** (FFT, *Fast Fourier Transform*) que es una versión eficiente de la DFT. Los recientes avances en tecnología de muy alta escala de integración (VLSI, *Very Large Scale Integration*) hacen posible la realización de circuitos integrados capaces de realizar grandes FFT's a precio muy económico. Utilizando este método, tanto transmisor como receptor pueden implementarse utilizando la FFT que reduce el número de operaciones de N^2 en DFT hasta NlogN.

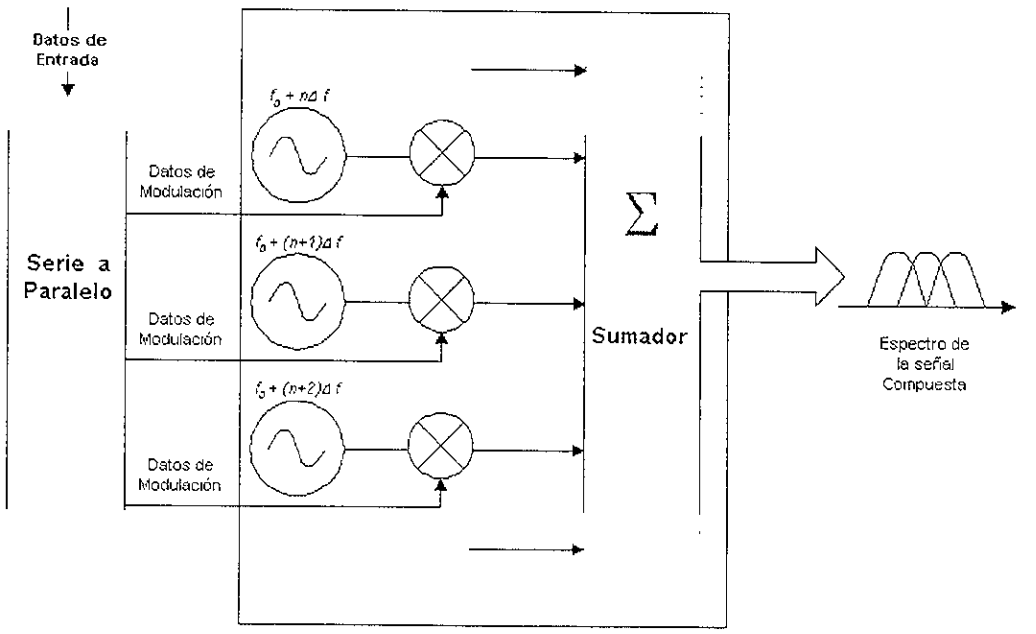


Figura 2.5 Principio del modulador OFDM.

2.4 *Generación de portadoras ortogonales por medio de la Transformada Rápida de Fourier*

Después de la descripción cualitativa del sistema, es necesario discutir la definición matemática de éste esquema de modulación. Para ello se mostrará como es generada la señal y cómo debe operar el receptor. Como ya se mencionó, OFDM transmite un gran número de portadoras de banda angosta espaciadas en el dominio de la frecuencia. Para evitar implementar el gran número de moduladores y filtros en el transmisor, y filtros y demoduladores complementarios en el receptor, sería deseable tener la capacidad de utilizar técnicas modernas de procesamiento digital de señales para generar las señales ortogonales.

Matemáticamente, cada portadora puede ser descrita como una onda compleja:

$$s_c(t) = A_c(t)e^{j[\omega t+\phi(t)]} \tag{2.2}$$

La señal real es la parte real de $s_c(t)$. Tanto $A_c(t)$ como $\phi(t)$ (amplitud y fase de la portadora) pueden variar entre símbolo y símbolo. Para QPSK la amplitud es nominalmente unitaria y la fase toma uno de los cuatro valores en cuadratura del sistema convencional QPSK.

Las portadoras deben cumplir con la condición de ortogonalidad, por lo tanto una consecuencia de ello es que la señal OFDM puede ser generada utilizando procedimientos asociados con la Transformada de Fourier.

2.4.1 La Transformada de Fourier

La *Transformada de Fourier* relaciona funciones en el dominio del tiempo con funciones en el dominio de la frecuencia. Existen varias versiones de la transformada de Fourier y la elección de cual utilizar depende de las circunstancias específicas de trabajo.

La transformada convencional relaciona señales continuas no limitadas ni en frecuencia ni en tiempo. Sin embargo el procesamiento de señales se simplifica si las señales son muestreadas. El muestreo de señales con un espectro infinito causa una variedad de problemas porque el proceso de muestreo ocasiona *aliasing* y el procesamiento de señales que no son limitadas en tiempo crean problemas de almacenamiento al desarrollar el hardware.

Para evitar esto, la mayor parte de los procesadores de señales utilizan una versión de la Transformada Discreta de Fourier. La DFT es una variante de la transformada normal, en donde las señales son muestreadas tanto en el dominio del tiempo como en el de la frecuencia. Por definición, la forma de onda en el tiempo debe repetirse continuamente, lo que ocasiona que el espectro de frecuencias se repita continuamente en el dominio de la frecuencia. La FFT es meramente un método matemático rápido para calcular la DFT. Es precisamente la disponibilidad de esta técnica y la tecnología asociada con ella lo que ha permitido la implementación de circuitos integrados a precio razonable para sistemas basados en OFDM.

2.4.2 El uso de la Transformada Rápida de Fourier (FFT) en OFDM

La principal razón por la cual OFDM había tomado tanto tiempo en tomar popularidad como esquema para transmisión de datos fue la dificultad que existía para generar dicha señal, y peor aún, de recibirla y demodularla. La solución en *hardware* que hace uso de múltiples moduladores y demoduladores en paralelo (figura 2.5) no es práctica pensando en el uso de sistemas domésticos.

Ahora, la posibilidad de definir la señal en el dominio de la frecuencia en *software* y de generar la señal en tiempo utilizando la Transformada Rápida de Fourier Inversa (IFFT), es la clave para su popularidad. El uso del proceso inverso (FFT) en la recepción es esencial si se piensa disponer de receptores baratos y confiables. Aunque las propuestas originales fueron hechas hace décadas, es evidente que ha sido necesario un periodo considerable de tiempo para que la tecnología pueda asimilarlas y aprovecharlas.

En el transmisor, la señal es definida en el dominio de la frecuencia de tal manera que su espectro existe sólo en frecuencias discretas. Cada subportadora OFDM corresponde a un elemento del espectro discreto de Fourier. Las amplitudes y fases de las portadoras dependen de los datos que serán transmitidos y éstos son definidos para cada símbolo. Todas las subportadoras tienen sus transiciones de datos y pueden ser procesadas en paralelo símbolo tras símbolo.

Una señal OFDM consiste en una suma de subportadoras que se modulan por medio de PSK (*Phase Shift Keying*, conmutación por cambio de fase) o QAM (*Quadrature Amplitude Modulation*, Modulación en cuadratura y amplitud). Si suponemos que s_i son símbolos complejos modulados en QAM o PSK, N_s es el número de subportadoras, T es la duración de un símbolo y

f_c la frecuencia central de una subportadora, entonces un símbolo OFDM que empieza en un tiempo dado $t = t_s$, se puede escribir como:

$$s(t) = \mathrm{Re}\left\{ \sum_{i=-\frac{N_s}{2}}^{\frac{N_s}{2}-1} s_{i+N_s/2} \exp\left(j2\pi \left(f_c - \frac{i+0.5}{T} \right)(t-t_s) \right) \right\}, \quad t_s \le t \le t_s + T$$
$$s(t) = 0, \quad t < t_s \quad \wedge \quad t > t_s + T \tag{2.3}$$

También es posible encontrar una notación diferente en la literatura en donde las partes real e imaginaria corresponden a las señales en fase y en cuadratura de la señal OFDM las cuales deben multiplicarse por un seno y un coseno a la frecuencia de la portadora deseada para producir la señal final OFDM. Esta notación se conoce como *Compleja en Banda Base* y está dada por la ecuación:

$$s(t) = \sum_{i=-\frac{N_s}{2}}^{\frac{N_s}{2}-1} s_{i+N_s/2} \exp\left(j2\pi \frac{i}{T}(t-t_s) \right), \quad t_s \le t \le t_s + T$$
$$s(t) = 0, \quad t < t_s \quad \wedge \quad t > t_s + T \tag{2.4}$$

La **figura 2.6** muestra cuatro subportadoras ortogonales de una señal OFDM. En este ejemplo todas las subportadoras tienen la misma fase y amplitud pero en la práctica las fases y amplitudes pueden ser moduladas de forma diferente para cada subportadora dependiendo del dato a transmitir.

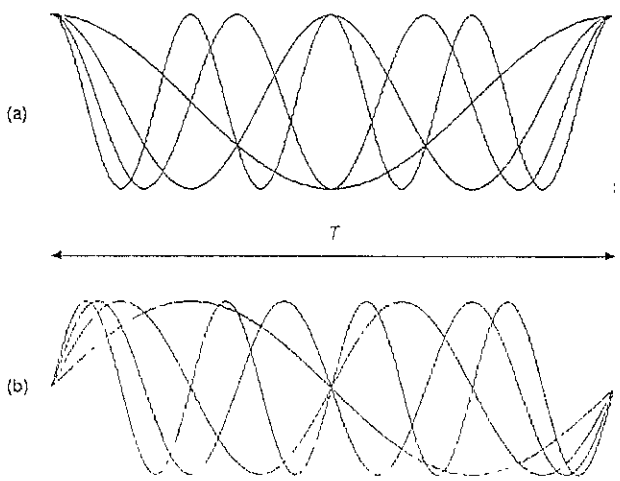


Figura 2.6 Ejemplo con cuatro subportadoras ortogonales en el dominio del tiempo a) Funciones coseno ortogonales; b) Funciones seno ortogonales

En la figura anterior cada subportadora tiene un número entero de ciclos dentro del intervalo T y el número de ciclos entre subportadoras adyacentes difiere únicamente en uno (1, 2, 3 y 4 ciclos). Esta propiedad es necesaria para garantizar la ortogonalidad entre las diferentes subportadoras.

Cuando la n-ésima subportadora obtenida de la ecuación (2.4) se traslada a su banda base por medio de una frecuencia de n/T y después se integra en un intervalo de T segundos, se obtiene una señal expresada de la siguiente forma:

$$\int_{t_s}^{t_s+T} \exp\left(-j2\pi\frac{n}{T}(t-t_s)\right)\sum_{i=-\frac{N_s}{2}}^{\frac{N_s}{2}-1} s_{i+N_s/2}\exp\left(j2\pi\frac{i}{T}(t-t_s)\right)dt = \sum_{i=-\frac{N_s}{2}}^{\frac{N_s}{2}-1} s_{i+N_s/2}\int_{t_s}^{t_s+T}\exp\left(j2\pi\frac{i-n}{T}(t-t_s)\right)dt = s_{n+N_s/2}T \tag{2.5}$$

En la ecuación anterior se puede ver que la portadora compleja se integra en un intervalo de T segundos, que para el caso de la subportadora n demodulada, arroja el valor deseado de la señal $S_{n+Ns/2}$ (multiplicado por un valor constante T), mientras que la integral para cualquier otra subportadora es *cero*, ya que la diferencia de frecuencias $(i-n)/T$ produce un número entero de ciclos dentro del intervalo de integración.

La ortogonalidad de las diferentes subportadoras de OFDM se puede demostrar también de otro modo. De acuerdo con la ecuación (2.3), cada símbolo OFDM contiene subportadoras con frecuencia diferente de cero en un intervalo de T segundos. Por lo tanto, el espectro de un solo símbolo es equivalente a la convolución de un grupo de pulsos de Dirac localizados en las frecuencias de las subportadoras con un espectro rectangular que vale uno durante un periodo de T segundos y cero en cualquier otro caso. El espectro de amplitud del pulso rectangular es igual a una función sinc(πfT), que tiene un valor de cero para todas las frecuencias cuyo valor es un múltiplo entero de $1/T$. Este hecho se observa en la **figura 2.7** que muestra la superposición de los espectros de diferentes subportadoras independientes.

Se puede ver que en el valor máximo del espectro de una subportadora dada, el valor del espectro de todas las demás subportadoras es cero. Ya que un receptor OFDM calcula los valores del espectro en el punto máximo de amplitud para cada subportadora, es decir, en la frecuencia

central de la subportadora, es capaz de demodular cada una de éstas sin interferencia proveniente de las subportadoras adyacentes.

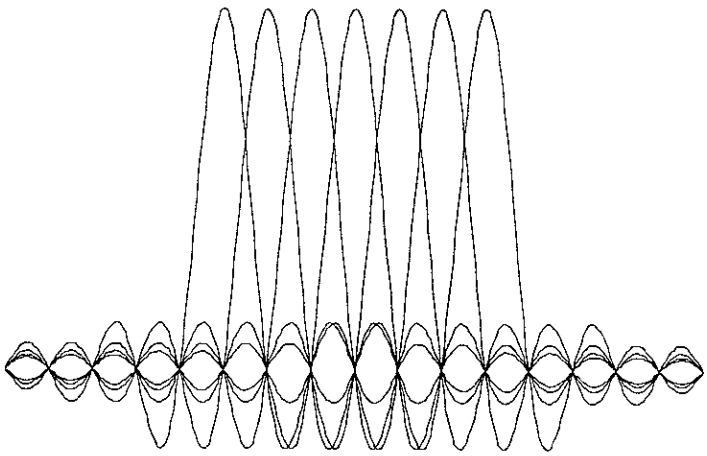


Figura 2.7 Espectros de subportadoras OFDM ortogonales.

La señal OFDM compleja en banda base definida por la ecuación (2.4) es la transformada inversa de Fourier de N_s símbolos QAM de entrada. El equivalente en una señal discreta es la Transformada Discreta Inversa de Fourier (IDFT), en donde el tiempo t se sustituye por el número de muestra n:

$$x[n] = \sum_{k=0}^{N_s-1} X[k] \exp\left(j2\pi \frac{kn}{N} \right) \tag{2.6}$$

donde x[n] se encuentra en el dominio del tiempo y X[k] se encuentra en el dominio de la frecuencia.

En la práctica, se puede implementar esta transformada de una manera muy eficiente por medio de la FFT, modelo que se desarrolla como parte fundamental de este trabajo. Una IDFT de N puntos necesita un total de N^2 multiplicaciones complejas. También se necesitan sumas para obtener la IDFT, pero su realización es mucho más sencilla tanto en niveles de hardware como de software. La IFFT reduce drásticamente la cantidad de cálculos necesarios aprovechando la regularidad de las operaciones de la IDFT. Por ejemplo, si se utiliza el algoritmo de raíz 2 (*radix-2*), una IFFT de N puntos necesita sólo $(N/2)log_2(N)$ multiplicaciones complejas. Si la transformada tiene 16 puntos, la IDFT realiza 256 multiplicaciones complejas, mientras que la IFFT sólo 32. La diferencia se hace más grande cuando existen más subportadoras, ya que la

complejidad de la IDFT crece de forma cuadrática, mientras que la complejidad de la IFFT crece de manera casi lineal.

La cantidad de multiplicaciones de la IFFT se puede reducir aún más utilizando un algoritmo de raíz 4 (*radix-4*). Esta técnica aprovecha el hecho de que en una IFFT de cuatro puntos existen sólo multiplicaciones por 1, -1, j, -j, por lo tanto no es necesario un multiplicador completo, simplemente se utiliza un algoritmo que realice sumas y restas junto con un interruptor que diferencie las partes real e imaginaria. En el algoritmo de raíz 4 se calcula la transformada dividiéndola en un cierto número de transformadas de cuatro puntos, logrando obtener un resultado de una transformada de N puntos por medio de $(3/8)N(\log_2 N\text{-}2)$ multiplicaciones complejas (que pueden verse como *rotaciones de fase*) y $N\log_2 N$ sumas de números complejos, reduciendo drásticamente el número de operaciones necesarias para obtener la transformada, lo que implica una mayor eficiencia en términos de cómputo.

En el caso de esta tesis, para cumplir con el estándar 802.11a de la IEEE, utilizamos una transformada de 64 puntos, para la que son necesarias 96 rotaciones de fase y 384 sumas. Por estas razones, este trabajo presenta como parte fundamental el algoritmo de la IFFT y de la FFT, aplicando estos conceptos para optimizar las simulaciones y hacer viable en un futuro su construcción física.

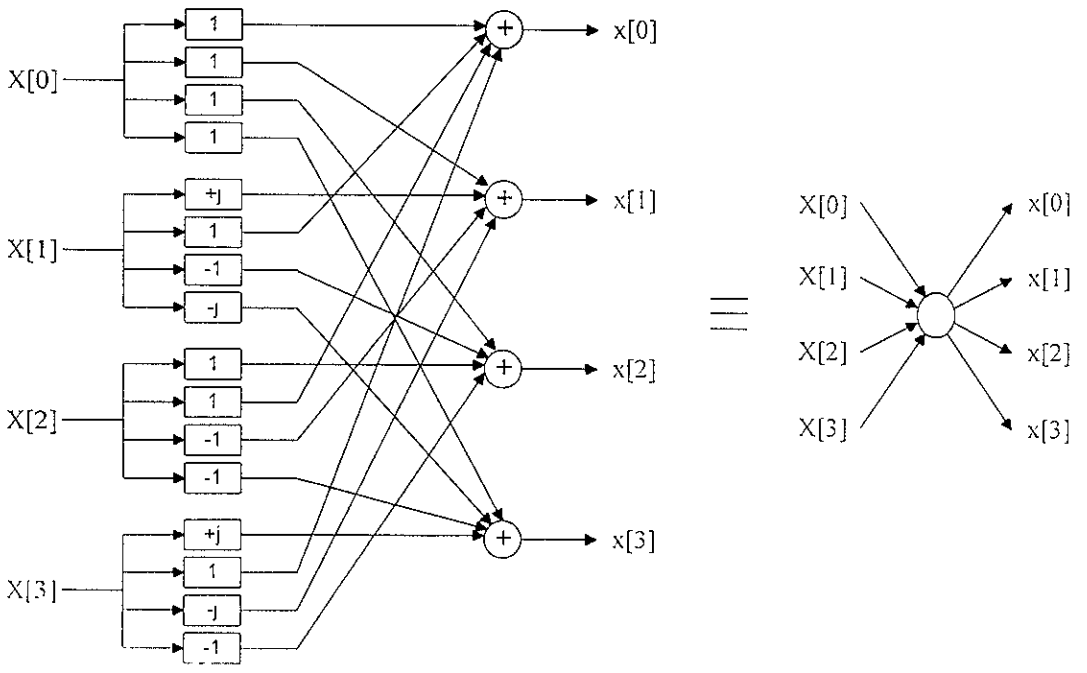


Figura 2.8 Diagrama de bloques de la *mariposa de raíz 4*

En la **figura 2.8** se muestra un diagrama de bloques de la IFFT de cuatro puntos, conocida como mariposa de raíz 4 (*radix-4 butterfly*), la cual constituye la base para formar transformadas más grandes y que se puede obtener desarrollando la ecuación (2.6) con N=4. Cuatro valores de entrada (*X[0]* a *X[3]*) generan cuatro salidas (*x[0]* a *x[3]*) por medio de sumas y rotaciones de fase triviales. Por ejemplo, el valor de *x[1]* está dado por $X[0] + jX[1] - X[2] - jX[3]$, que se puede calcular realizando cuatro sumas junto con rotaciones entre Real/Imaginario e inversiones para obtener los resultados de las multiplicaciones por j y –1 como se explicará en el capitulo 5.

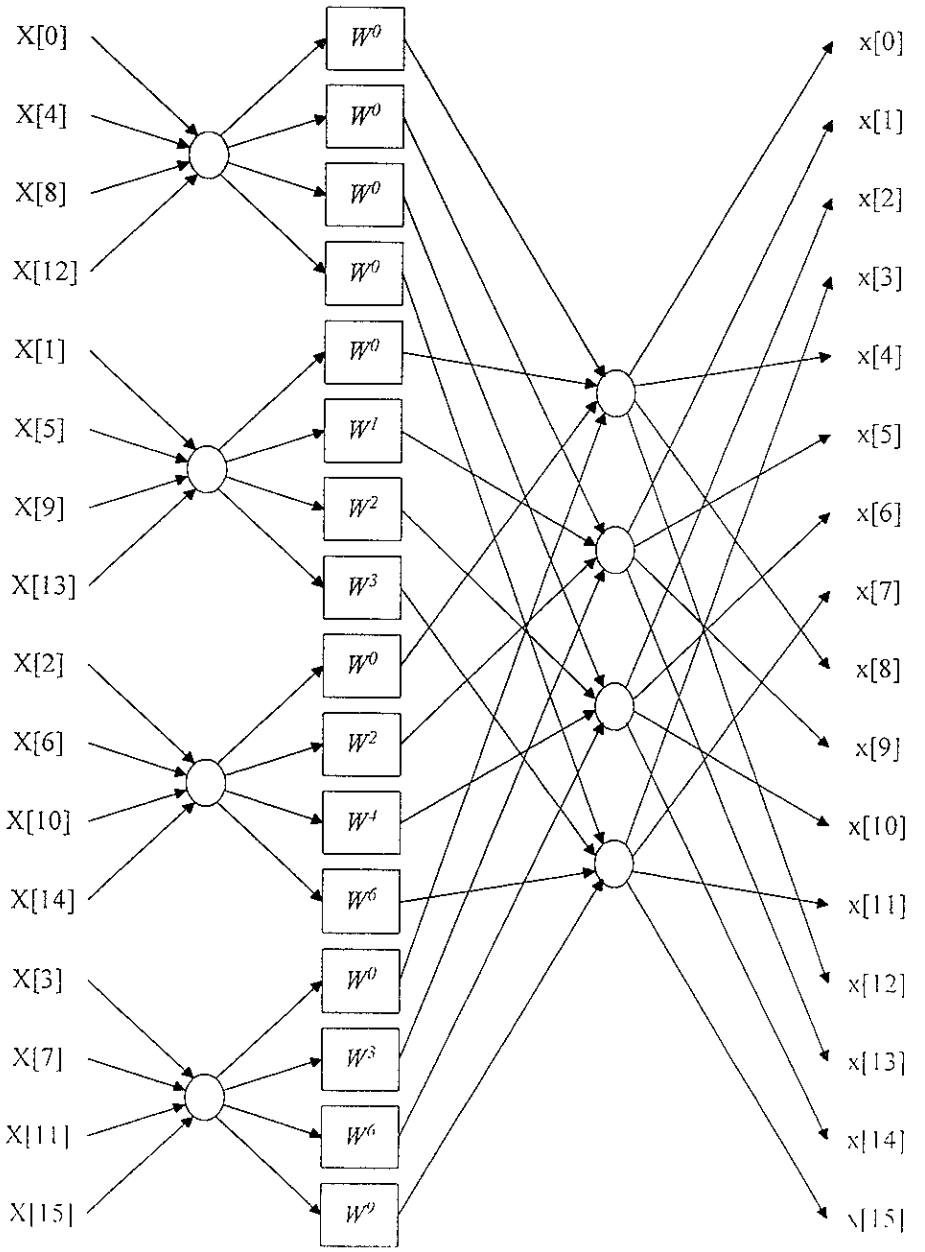


Figura 2.9 IFFT de 16 puntos utilizando el algoritmo de raíz cuarta

Este algoritmo puede utilizarse de manera eficiente para construir una IFFT más grande, utilizando diferentes etapas. En la **figura 2.9** se puede ver una transformada inversa de 16 puntos. Esta contiene dos etapas con mariposas de raíz 4, separadas por una etapa intermedia en donde los 16 resultados parciales sufren una rotación de fase por un factor ω^i, definido por la ecuación $exp(j2\pi i/N)$. Nótese que para N=16, la rotación por el factor ω^i se convierte en una operación trivial para i=0, 4, 8 y 12 donde ω^i=1, j, -1 y -j respectivamente.

Tomando estas consideraciones, la IFFT de 16 puntos tiene sólo 8 rotaciones de fase no triviales, que es un número 32 veces menor que las rotaciones necesarias para realizar una IDFT. Estas rotaciones de fase no triviales determinan en gran medida la complejidad del sistema, ya que es más difícil realizar una rotación de fase o una multiplicación de números complejos que una suma de éstos.

2.5 *Intervalo de Guarda*

Una de las razones más importantes para desarrollar sistemas basados en OFDM es su robusto desempeño en ambientes con multitrayectorias, como se demostrará en las gráficas de la sección 5.6. Dividiendo el flujo de entrada en N flujos, la duración del símbolo se vuelve N veces más larga (figura 2.1) lo cual reduce el retardo de las multitrayectorias con respecto a la duración del símbolo. Para eliminar la interferencia intersímbolo casi completamente se introduce un intervalo de guarda antes de cada símbolo. La duración de este intervalo se selecciona para que sea mas larga que el retardo máximo esperado, de tal forma que las reflexiones de las multitrayectorias de un símbolo no interfieran con el siguiente símbolo.

Cabe señalar que la introducción de un intervalo de guarda conlleva a una pérdida en potencia debido a que se transmite más potencia de la necesaria para demodular los símbolos. Para el caso en que el intervalo de guarda es la cuarta parte de la duración del símbolo se tiene que las pérdidas son de $10\times\log(5/4) = 0.969$ dB ≈ 1 dB. (Gráficas de la sección 5.6)

El intervalo de guarda puede consistir de una señal nula. En este caso, aparece el problema de interferencia intercanal (ICI). Este efecto se ilustra en la **figura 2.10** donde se muestra una *subportadora 1* y una *subportadora 2* retardada. Cuando el receptor OFDM trata de demodular la primera subportadora encontrará interferencia de la segunda subportadora debido a que en el intervalo de integración de la FFT no hay un número entero de ciclos de la subportadora

2 retardada. De la misma forma, la subportadora 1 retardada provocará interferencia a la subportadora 2.

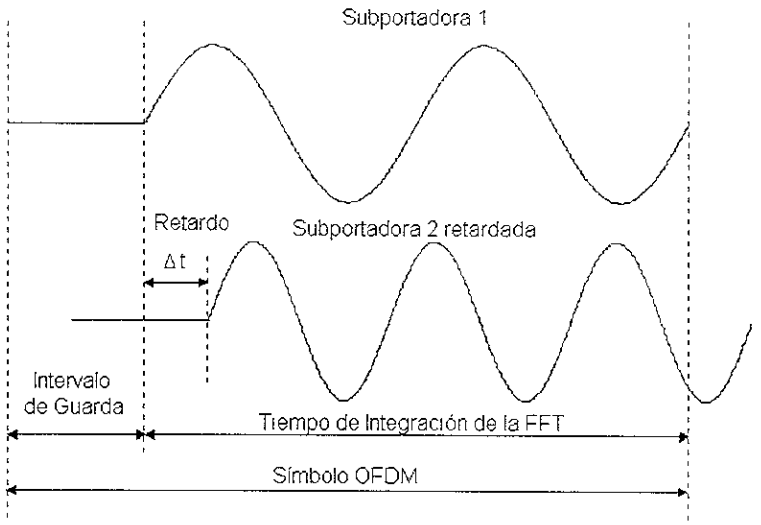


Figura 2.10 Efecto del intervalo de guarda nulo. La subportadora 2 retardada interfiere a la subportadora 1.

Para eliminar la ICI, el símbolo OFDM se extiende cíclicamente durante el intervalo de guarda como se muestra en la **figura 2.11**. Esto asegurará que las reflexiones de los símbolos OFDM siempre tendrán un número entero de ciclos dentro del intervalo de la FFT, mientras el retardo sea menor que el intervalo de guarda.

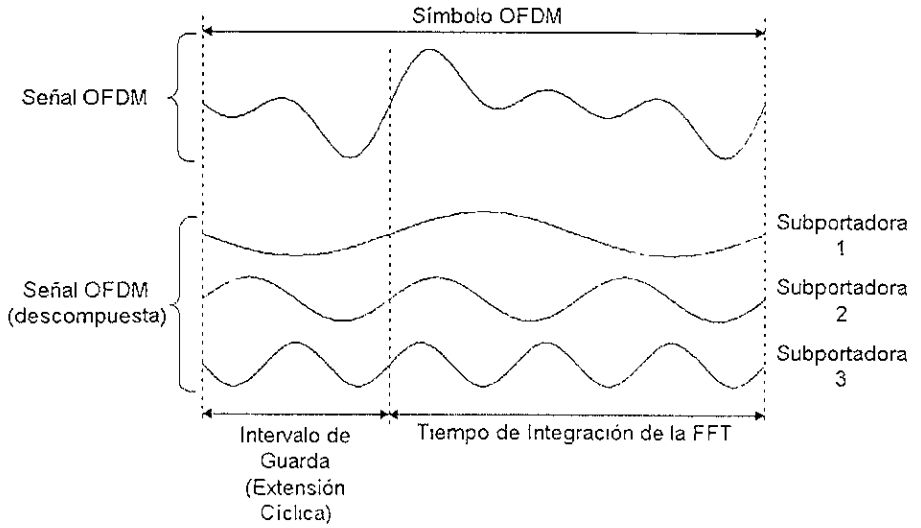


Figura 2.11 Símbolo OFDM con extensión cíclica

La **figura 2.12** (a) muestra un ejemplo de señal OFDM con multitrayectoria y debajo (b) se muestra la misma señal OFDM descompuesta en sus 3 diferentes subportadoras. Las líneas continuas representan las señales originales y las líneas punteadas representan las reflexiones retardadas y atenuadas de las señales originales.

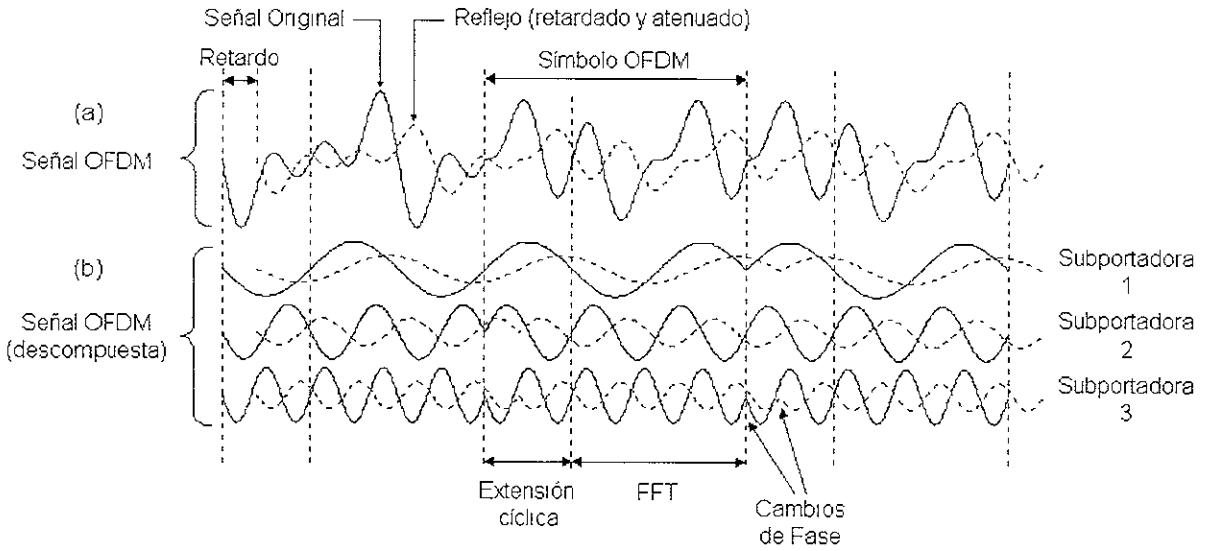


Figura 2.12 Ejemplo de una señal OFDM y una reflexión: (a) señal OFDM transmitida (b) señal OFDM descompuesta en sus 3 subportadoras.

En la figura anterior se puede ver que las subportadoras OFDM están moduladas con BPSK, lo que significa que en las fronteras existen cambios de fase de 180°. Para la línea punteada estos cambios ocurren después de un retardo Δt con respecto de la señal original. En este ejemplo el retardo de la multitrayectoria es menor que el intervalo de guarda lo cual significa que no hay cambios de fase durante el intervalo de integración de la FFT, es decir, no hay *interferencia intersímbolo* (ISI). Por lo tanto, el receptor OFDM percibe la suma de ondas senoidales puras con diferentes variaciones de fase. Esta suma no invalida la condición de ortogonalidad entre las subportadoras, solamente introduce una variación de fase diferente a la transmitida para cada subportadora. La ortogonalidad se pierde si el retardo de la multitrayectoria es más largo que el intervalo de guarda. En ese caso, los cambios de fase de las reflexiones retardadas ocurren dentro del intervalo de la FFT provocando ISI e ICI

Para poder tener una idea del nivel de interferencia que se introduce cuando el retardo de las multitrayectorias excede el intervalo de guarda se muestran tres constelaciones en la **figura**

2.13. Las constelaciones se obtuvieron de una simulación OFDM con 48 subportadoras, cada una modulada con 16 QAM. Es necesario recalcar que las tres constelaciones pertenecen a la misma subportadora en 3 circunstancias diferentes, aunque realmente se tienen 48 constelaciones diferentes correspondientes a las 48 subportadoras. La figura 2.13 (a) muestra la constelación 16 QAM sin distorsión que se obtiene cuando el retardo de las multitrayectorias es menor que el intervalo de guarda. En la figura 2.13 (b) el retardo de las multitrayectorias excede un 10% el intervalo de guarda, por lo tanto las subportadoras dejan de ser ortogonales, aunque la interferencia sigue siendo lo suficientemente pequeña para recibir la constelación adecuadamente. La figura 2.13 (c) muestra la constelación recibida cuando el retardo de la multitrayectoria excede el 20% del intervalo de guarda. En este caso la interferencia afecta seriamente a la constelación provocando la recepción de la señal con una alta tasa de errores.

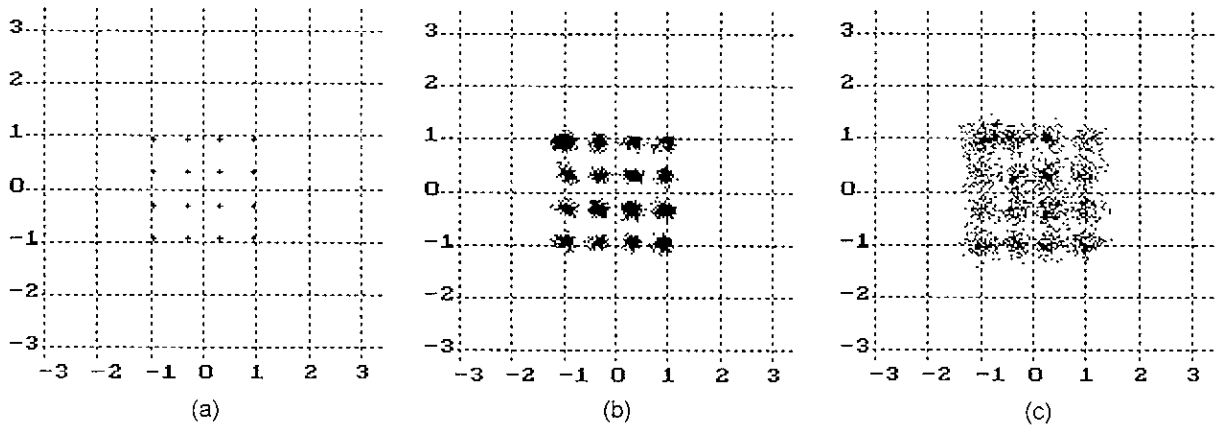


Figura 2.13 Constelación 16 QAM de una subportadora OFDM en 3 circunstancias: (a) retardo de las multitrayectorias menor que el intervalo de guarda, (b) retardo excediendo el 10% del intervalo de guarda, (c) retardo excediendo el 20% del intervalo de guarda.

2.6 Filtrado de los símbolos OFDM

Como se explicó anteriormente, un símbolo OFDM se forma llevando a cabo una IFFT y agregando una extensión cíclica. Tomando como ejemplo la señal de la figura 2.12 se pueden observar cambios de fase pronunciados en los límites de los símbolos. Esencialmente, una señal OFDM como la de la figura consiste de subportadoras QAM sin filtrar. Como resultado, las emisiones en bandas espurias disminuyen de forma relativamente lenta, de acuerdo con la

función sinc. La **figura 2.14** muestra un ejemplo de las Densidades Espectrales de Potencia (*PSD, Power Spectral Density*) obtenidas para una señal con 16, 64 y 256 subportadoras. Para un número mayor de subportadoras, la PSD se atenúa más rápido al principio debido a que los lóbulos laterales se encuentran menos separados. Sin embargo, aún para la PSD con 256 subportadoras el ancho de banda a –40 dB es casi 4 veces el ancho de banda a –3 dB.

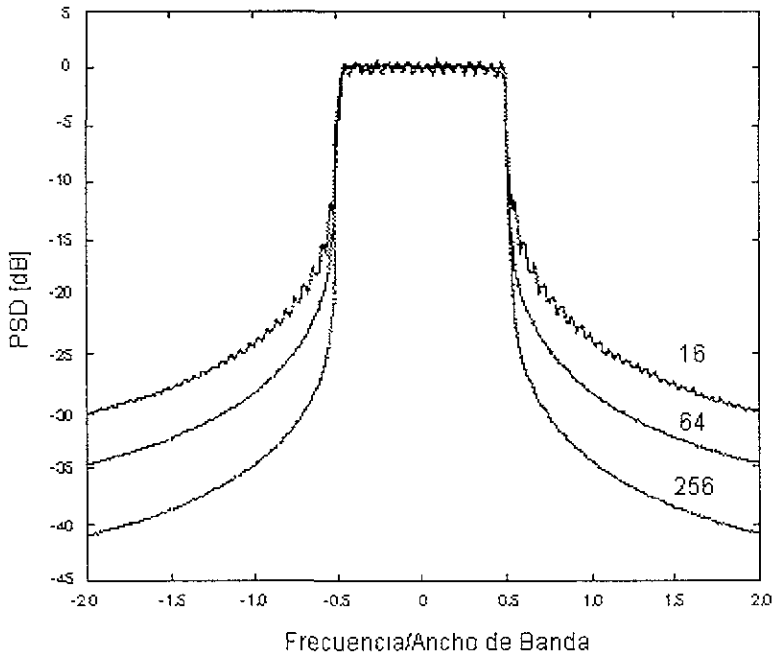


Figura 2.14 Densidad Espectral de Potencia (PSD) para una señal OFDM con 16, 64 y 256 subportadoras.

Para atenuar más rápidamente los lóbulos laterales se puede aplicar una ventana a los símbolos OFDM. Este proceso hace que la amplitud en los limites de los símbolos disminuyan suavemente. Un tipo de ventana comúnmente utilizado es la ventana de coseno elevado la cual se define como:

$$w(t) = \begin{cases} 0.5 + 0.5\cos(\pi + t\pi/(\beta T_s)) , & 0 \le t \le \beta T_s \\ 1.0 , & \beta T_s \le t \le T_s \\ 0.5 + 0.5\cos((t - T_s)\pi/(\beta T_s)), & T_s \le t \le (1+\beta)T_s \end{cases} \quad (2.7)$$

donde T_s es el intervalo del símbolo el cual es menor que la duración total del símbolo debido a que se permite que los símbolos adyacentes se encimen parcialmente en la región de traslape. La estructura de una señal OFDM es la que se muestra en la **figura 2.15**.

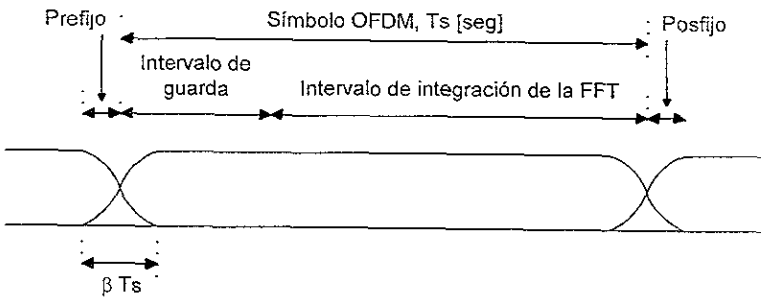


Figura 2.15 Estructura de un símbolo OFDM: intervalo de la FFT, extensión cíclica, región de traslape. β es el factor de traslape.

La **figura 2.16** presenta las PSD de señales OFDM con diferentes valores del factor de traslape β. Se puede ver que con un factor de 0.025 las emisiones espurias disminuyen notablemente. El ancho de banda en –40 dB disminuye a casi la mitad.

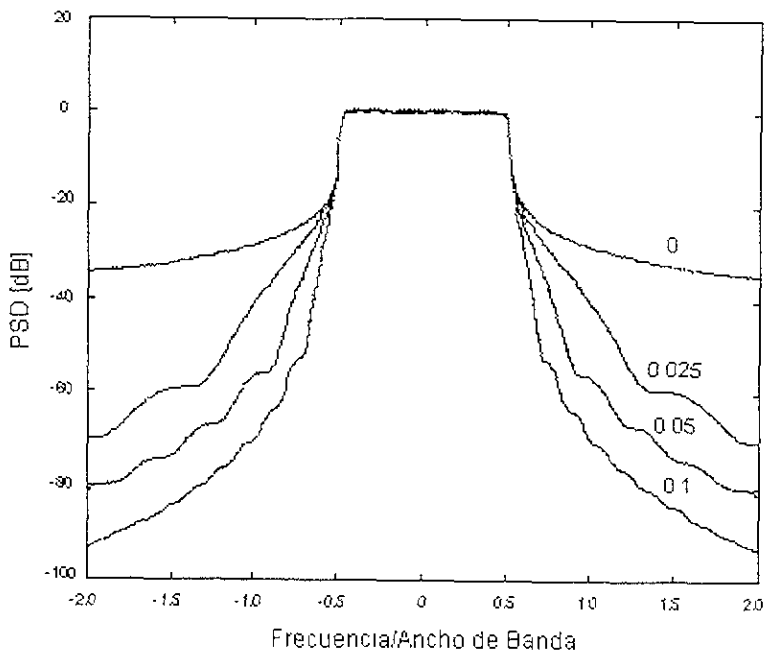


Figura 2.16 PSD de una señal OFDM con factores de traslape: β =0, β =0.025, β =0.05, β =0.1.

Los factores de traslape más grandes mejoran la característica de la PSD, pero disminuyen la tolerancia al retardo de las multitrayectorias. Este último efecto se demuestra en la **figura 2.17**. El receptor demodula las subportadoras llevando a cabo la FFT en el intervalo de integración. A

3 Modulación y Codificación

3.1 Modulación en Cuadratura y Amplitud

La modulación en cuadratura y amplitud (QAM) es la más utilizada en combinación con OFDM para modular cada una de las subportadoras ortogonales. Especialmente las constelaciones rectangulares son fáciles de implementar debido a que se pueden separar en dos componentes independientes de flujos con modulación en amplitud de pulso (PAM) para la señal en fase (*In-phase*) y la señal en cuadratura (*Quadrature*). La **figura 3.1** muestra las constelaciones para BPSK (Binary Phase Shift Keying), QPSK (*Quadrature Phase Shift Keying*) y 16-QAM (*16-Quadrature Ampliude Modulation*).

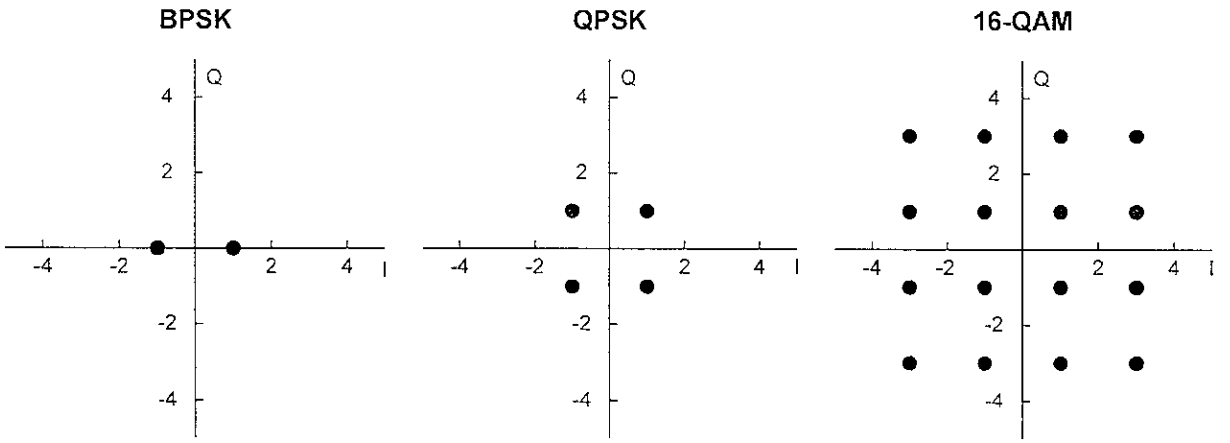


Figura 3.1 Constelaciones BPSK, QPSK y 16-QAM.

Las constelaciones no están normalizadas, para normalizarlas a una potencia promedio de 1, suponiendo que todos los puntos de la constelación tienen la misma probabilidad de ocurrencia, cada constelación debe ser multiplicada por un factor de normalización listado en la **tabla 3.1**.

Modulación	Factor de Normalización	Pérdida de Eb/No relativa a BPSK [dB]
BPSK	1	0
QPSK	$1/\sqrt{2}$	0
16-QAM	$1/\sqrt{10}$	3.98

Tabla 3.1 Factores de Normalización y Pérdida relativa a BPSK.

La tabla también indica la pérdida máxima en E_b/N_0 relativa a BPSK que se necesita para tener la misma tasa de errores en una transmisión QAM sin codificación. Las curvas de BER (*Bit Error Rate*, Tasa de Bits Erróneos) de la **figura 3.2** muestran las pérdidas de la tabla 3.1. Estas pérdidas son más precisas para valores debajo de una tasa de error de 10^{-2}. La diferencia entre QPSK y 16-QAM es de 4 dB aproximadamente. Para constelaciones mas grandes las pérdidas por cada par de bits adicional que transporte el símbolo convergen a 3 dB, por ejemplo, para una constelación 64-QAM (6 bits por símbolo) la pérdida con respecto a la curva de 16-QAM será de 3 dB.

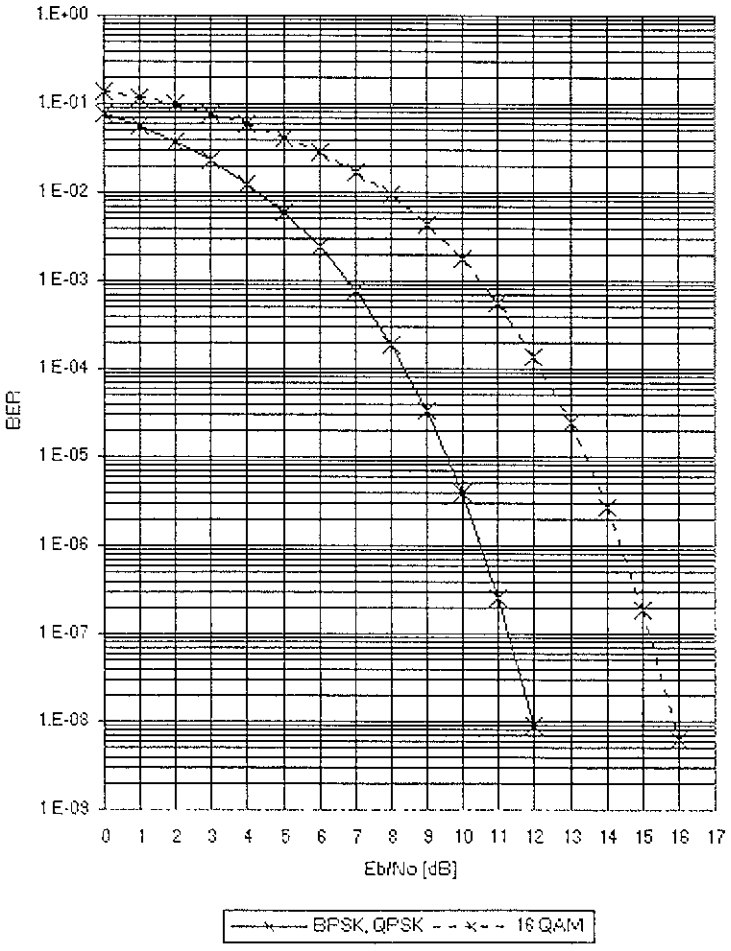


Figura 3.2 Curvas teóricas BER vs. Eb/No para BPSK, QPSK y 16-QAM.

Esta tesis tiene como objetivo que el sistema OFDM simulado alcance un desempeño similar o mejor al de las gráficas anteriores en un ambiente con multitrayectorias y desvanecimiento selectivo de frecuencias. Para llevar a cabo este objetivo se deben analizar los

efectos de las multitrayectorias en el espectro de la señal. Estos efectos se tratarán de anular con el uso de técnicas de codificación y de intercalamiento que se detallarán en las siguientes secciones.

3.2 *Efecto de las multitrayectorias*

A diferencia de las comunicaciones por satélite donde se tiene una sola trayectoria desde la antena transmisora hasta la antena receptora, en el escenario clásico de enlaces terrestres se debe lidiar con los efectos de los canales con multitrayectorias: la señal transmitida llega al receptor por varias trayectorias como resultado de las múltiples reflexiones que sufre en obstáculos fijos como colinas, árboles, edificios y objetos en movimiento como vehículos, aviones e incluso personas (**figura 3.3**). Debido a que las reflexiones de la misma señal interfieren unas con otras (figura 3.4) se vuelve difícil extraer la información de la señal original.

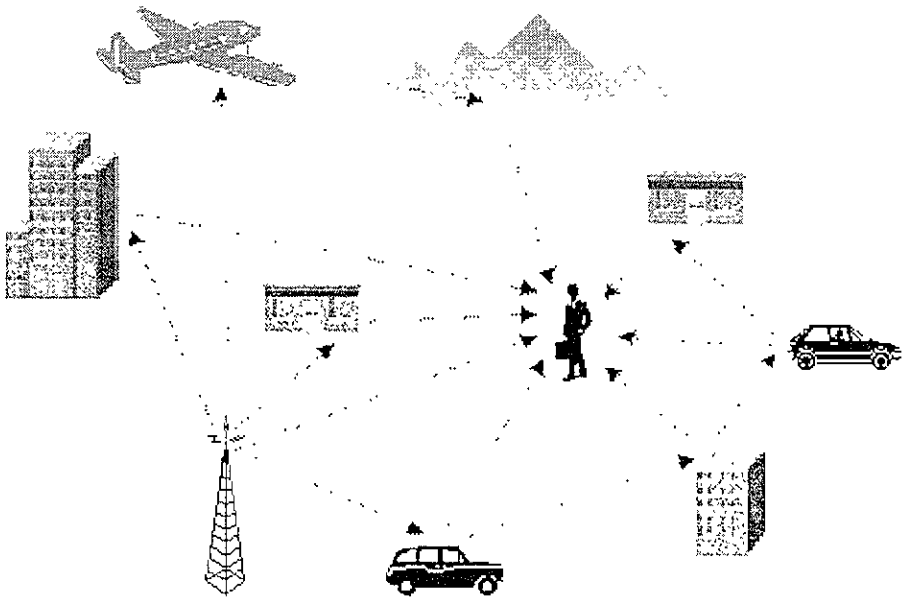


Figura 3.3 Ejemplo de un ambiente con multitrayectorias.

Una característica de los canales con desvanecimiento selectivo de frecuencias es que la densidad espectral de potencia aumenta en algunas frecuencias mientras que en otras la densidad disminuye. Si la banda de la señal transmitida es comparable con respecto a las regiones de desvanecimiento y alguna de estas regiones coincide con la banda de transmisión, la calidad de la

recepción se degradará. Por otra parte, si existe movimiento, ya sea del receptor o de algún objeto que lo rodee, entonces las longitudes relativas y las atenuaciones de las diferentes trayectorias de recepción cambiarán con el tiempo. La calidad de la señal variará al cambiar la respuesta del canal. También existirá una variación notable en la fase y amplitud de la señal que afectará a los sistemas que utilicen modulación de fase. En la **figura 3.4** se muestra esta variación de fase y de amplitud. La subportadora 1 mantiene su amplitud igual pero su fase se retrasa, mientras que la subportadora 2 disminuye su amplitud y adelanta su fase. La subportadora 3 aumenta su amplitud y su fase permanece casi estática. Esta disminución de las amplitudes se refleja en la densidad espectral de potencia, haciendo menos confiables a esas portadoras. Si bien la subportadora 3 queda beneficiada por la multitrayectoria, la subportadora 2 queda afectada. Extrapolando este efecto podemos pensar en un sistema con mas subportadoras de las cuales unas serán mejoradas y otras empeoradas (**figura 3.5**).

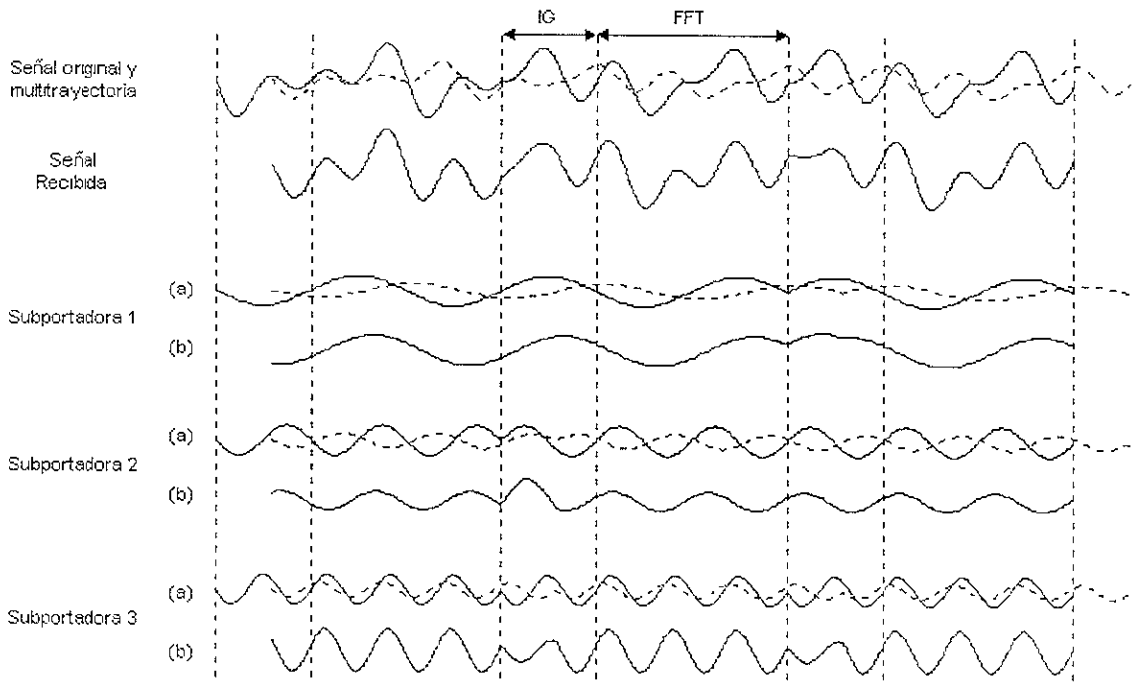


Figura 3.4 Efecto de las multitrayectorias en la señal OFDM.

En la figura anterior se puede ver que para cada subportadora se tiene una señal perfectamente senoidal dentro del intervalo de la FFT. Sin embargo, en los intervalos de guarda se pueden observar irregularidades debido a la interferencia intersímbolo.

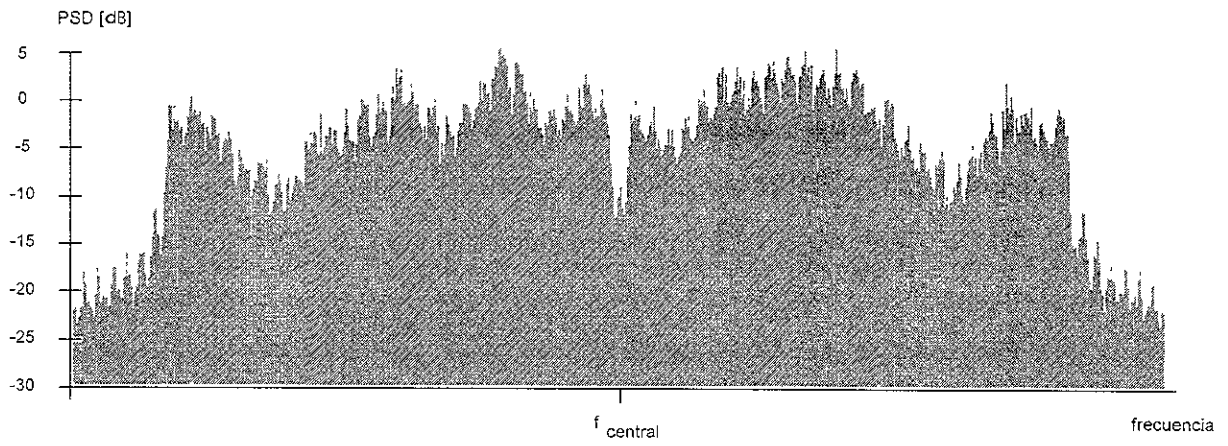


Figura 3.5 Efecto de las multitrayectorias en el espectro.

Para contrarrestar el efecto de las multitrayectorias se necesita codificar la información que transportan las subportadoras. En el transmisor se utilizan algoritmos de codificación que proporcionan una ganancia de potencia a cambio de un aumento en el ancho de banda. Es el caso de los códigos convolucionales y los códigos de bloque. En el receptor se utilizan algoritmos de decodificación para recuperar la información original. En esta tesis se utiliza solamente codificación convolucional ya que se toma el modelo de comunicación del estándar IEEE 802.11a. El algoritmo de decodificación utilizado es el de Viterbi con *decisión dura* (*hard decision*), es decir, de dos niveles de cuantización. Otros algoritmos que se necesitan implementar en el sistema son el intercalamiento y el desintercalamiento que se describirán mas adelante.

3.3 Codificación Convolucional

Una codificación convolucional relaciona cada *k* bits de entrada con *n* bits a la salida. Esta relación se lleva a cabo convolucionando los bits de entrada con una respuesta al impulso binaria. La codificación convolucional puede ser implementada con registros de corrimiento y sumadores de módulo 2. La **figura 3.6** muestra un codificador de tasa 1/2 el cual se utiliza en el estándar IEEE 802.11a y que se programó para la simulación del sistema de comunicación en esta tesis. Este codificador tiene *una* entrada de datos ($k = 1$) y *dos* salidas ($n = 2$) A_i y B_i, las cuales son intercaladas para formar la secuencia de salida $\{A_1B_1A_2B_2 \ldots\}$. Cada par de bits de salida $\{A_iB_i\}$ dependen de 7 bits (el bit de entrada y otros 6 bits que han sido almacenados en un registro de

corrimiento). A este valor de longitud se le llama *longitud limitante*. Los valores con los que se convoluciona el registro de corrimiento corresponden a los polinomios generadores o vectores generadores. En la figura se puede ver que los vectores generadores son {1011011, 1111001} o en notación octal {133,171}.

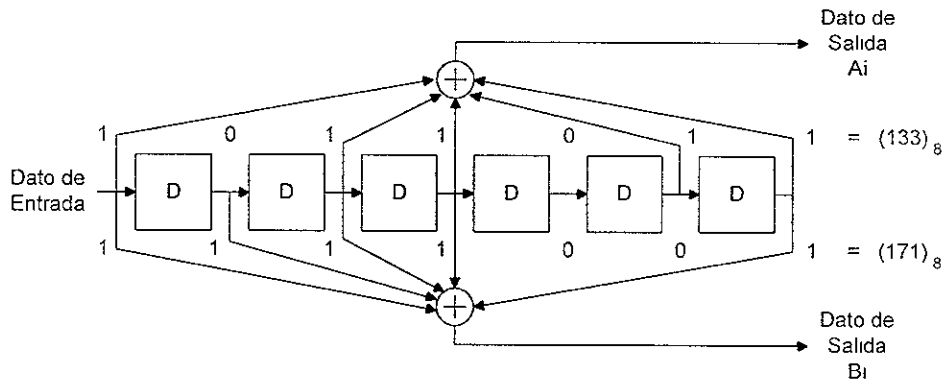


Figura 3.6 Diagrama de un codificador convolucionador con longitud limitante igual a 7.

3.3.1 Codificación

Para poder explicar con mayor claridad el algoritmo de decodificación de Viterbi utilizado en las simulaciones, utilizaremos el codificador de la **figura 3.7**.

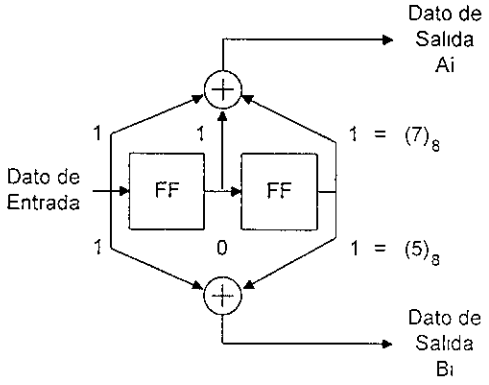


Figura 3.7 Diagrama de un codificador convolucionador con longitud limitante igual a 3.

En este codificador los bits entran a una tasa de k bits por segundo. Los símbolos de salida son entregados a una tasa de $n=2k$ bits por segundo. La longitud limitante es K=3 y los vectores generadores son 7_8 y 5_8 (111_2 y 101_2). Si a la entrada se tiene la siguiente secuencia: 0101110010100001_2, el codificador proporcionará la siguiente salida: 00 11 10 00 01 10 01 11 11 10 00 10 11 00 11. Para cada bit de entrada existe una pareja de bits en la salida, sin embargo,

cada bit entrante afecta a 3 parejas de bits (K=3). Si se desea que los últimos dos bits también tengan efecto en 3 parejas se necesita *limpiar* el registro de corrimiento (*flushing*) introduciendo en el codificador dos bits nulos mas. La salida final del codificador queda descrita por la siguiente secuencia: 00 11 10 00 01 10 01 11 11 10 00 10 11 00 11 10 11_2. Si no se lleva a cabo esta operación los últimos dos bits del mensaje original tienen menor capacidad de corrección que los primeros 13 bits. El codificador debe comenzar y terminar en estados conocidos para que el decodificador sea capaz de reconstruir apropiadamente la secuencia original.

También se puede pensar en el codificador como una máquina de estados. El codificador de la figura 3.7 tiene dos bits de memoria por lo que existen cuatro posibles estados. Podemos considerar un peso binario de 2^1 para el flip-flop de la izquierda y un peso binario de 2^0 para el de la derecha. Gráficamente se muestra en la **figura 3.8** el comportamiento de la máquina de estados. Dentro de cada óvalo se encuentra el estado actual, sobre cada flecha se muestra el bit de entrada y los bits de salida (X/Y_AY_B) y cada flecha apunta al siguiente estado. Inicialmente el codificador se encuentra en el estado 00_2. Si el primer bit de entrada es 0, el codificador permanece en el estado 00_2 y la salida es 00_2, pero si el bit es 1 el codificador pasa al estado 10_2 y la salida es 11_2. De la misma forma se puede describir el comportamiento en los demás estados.

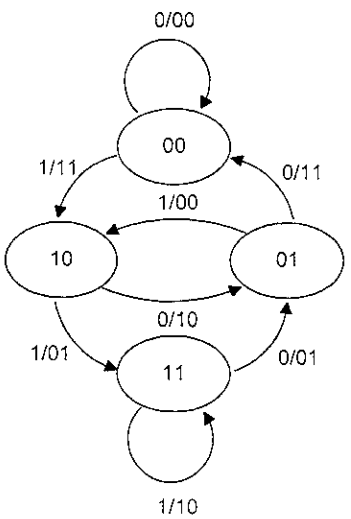


Figura 3.8 Diagrama de la máquina de estados del codificador con K=3.

3.3.2 Decodificación

Probablemente el concepto más importante para comprender el algoritmo de Viterbi es el *diagrama trellis*. La **figura 3.9** muestra el diagrama trellis para un mensaje de 15 bits utilizando el codificador convolucional con K=3.

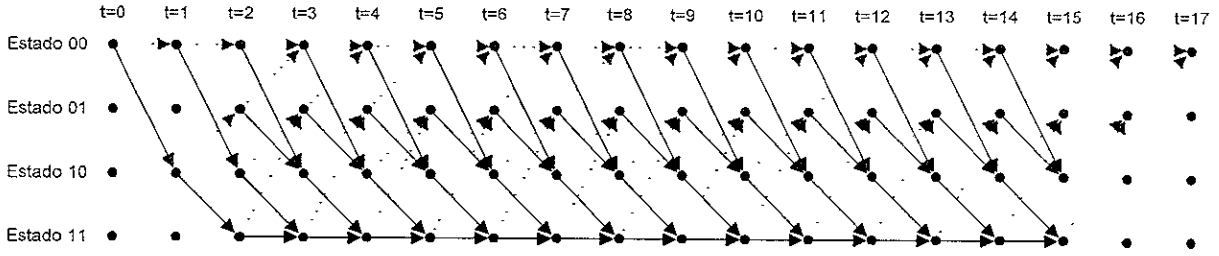


Figura 3.9 Diagrama trellis para un mensaje de 15 bits.

Los cuatro renglones de puntos representan los cuatro estados posibles del codificador. Hay una columna de cuatro puntos para el estado inicial del codificador y una para cada instante durante el mensaje. Para un mensaje de 15 bits con dos bits de *limpieza* (*flushing*) existen 17 instantes además del instante de inicio (t=0) que representa la condición inicial del codificador. Las líneas continuas del diagrama representan transiciones de estados cuando el bit de entrada es 1. Las líneas punteadas representan transiciones cuando el bit de entrada es 0. Se puede verificar la correspondencia de las flechas con el comportamiento de la máquina de estados descrito gráficamente en la figura 3.8. También se puede ver que debido a que la condición inicial del codificador es el estado 00_2 y los bits de *limpieza* son ceros, la trayectoria de la codificación comienza y termina en el estado 00_2.

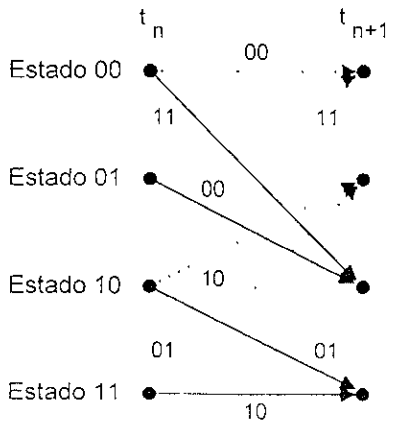


Figura 3.10 Transiciones posibles y sus salidas

En la **figura 3.10** se puede ver la relación de las salidas del codificador con todas las transiciones posibles entre estados. La salida son las parejas asociadas a cada flecha. También se puede verificar la correspondencia con la máquina de estados de la figura 3.8.

El diagrama de la **figura 3.11** muestra los estados por los que pasa la máquina para codificar el ejemplo de 15 bits propuesto.

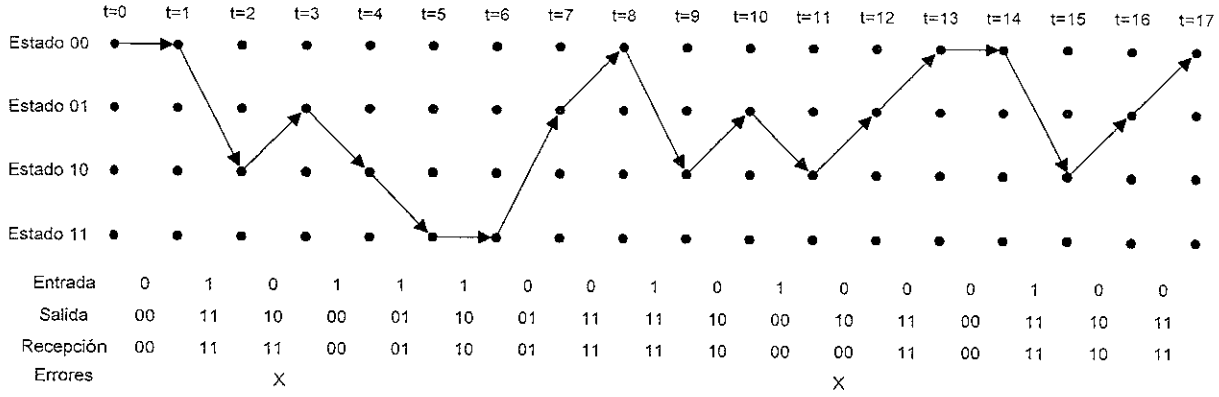


Figura 3.11 Diagrama trellis para la secuencia: 010111001010001.

Debajo del diagrama se muestra la secuencia de entrada al codificador, la secuencia de salida y un ejemplo de secuencia recibida con errores. El algoritmo de decodificación funciona de la siguiente forma: cada que se recibe un par de bits codificados se calcula la distancia de Hamming entre ese par y todos los posibles pares que se pueden recibir. La distancia de Hamming se calcula contando los dígitos diferentes entre dos números binarios. En este caso la distancia puede ser 0, 1 o 2, es decir, que no difieren los pares, que un dígito difiere o que los dos dígitos difieren.

Los valores de la distancia de Hamming que se calculan entre el instante del estado previo y el instante del estado actual se llaman *métricas de rama*. Para el primer instante se guardan estos resultados en unos valores de *error acumulado de la métrica* asociados a cada estado. Para los siguientes instantes estos valores se calcularán sumando el valor anterior del error acumulado con la métrica de rama actual.

Del instante t=0 al instante t=1 solamente se pueden recibir dos pares de bits: 00_2 y 11_2. Esto se debe a que el codificador inició en el estado 00_2 y que solamente se puede recibir un 0 o un 1, por lo que solamente existen dos posibles transiciones. En el ejemplo la pareja recibida es 00_2. La distancia de Hamming entre 00_2 y 00_2 es cero. La distancia entre 00_2 y 11_2 es dos. Por lo

tanto, la métrica de rama asociada a la transición entre el estado 00_2 y el estado 00_2 es cero y entre el estado 00_2 y el estado 10_2 es dos. En la **figura 3.12** se muestran los resultados hasta el instante t=1.

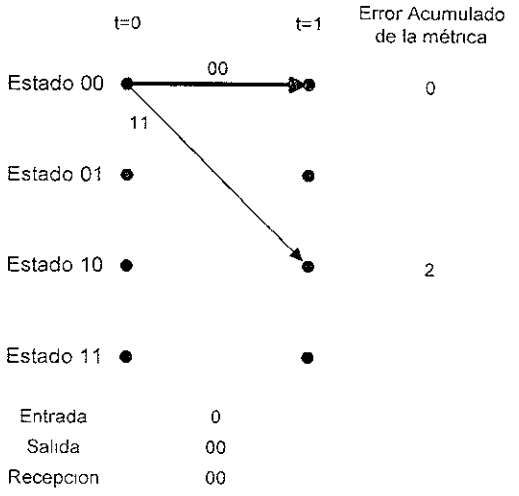


Figura 3.12 Error acumulado de la métrica hasta el instante t=1.

Se debe hacer notar que las líneas sólidas entre los estados del instante t=1 y el estado en t=0 representan la relación *predecesor-sucesor*. Esta información se muestra gráficamente en la figura pero se almacena numéricamente en el implemento real, es decir, para cada instante t se almacenará el número del estado predecesor que lleva a cada uno de los estados en el instante sucesor.

En el instante t=2 se recibe el par de bits 11_2. Los únicos pares que se pueden recibir del instante t=1 a t=2 son: $\mathbf{00_2}$ del estado 00_2 al estado 00_2, $\mathbf{11_2}$ del estado 00_2 al estado 10_2, $\mathbf{10_2}$ del estado 10_2 al estado 01_2 y $\mathbf{01_2}$ del estado 10_2 al estado 11_2. Las métricas de rama son, respectivamente, 2, 0, 1 y 1. Se deben sumar estas métricas al valor del error acumulado de la métrica de cada estado y se deben almacenar los estados predecesores.

En el instante t=1 solamente se puede estar en el estado 00_2 o en el estado 10_2. Los valores del error acumulado de la métrica asociados con esos estados son 0 y 2, respectivamente. La **figura 3.13** muestra el cálculo del error acumulado asociado con cada estado hasta el instante t=2

Las líneas continuas representan las rutas seleccionadas debido a que tienen menor error acumulado. Las líneas punteadas representan las rutas que se descartan. Se puede ver que el tercer par de bits recibidos tiene un bit erróneo. El menor error acumulado es 1 y existen dos resultados de estos. Para el instante t=4 se procede de la misma forma que en t=3. Los resultados se muestran en la **figura 3.15**.

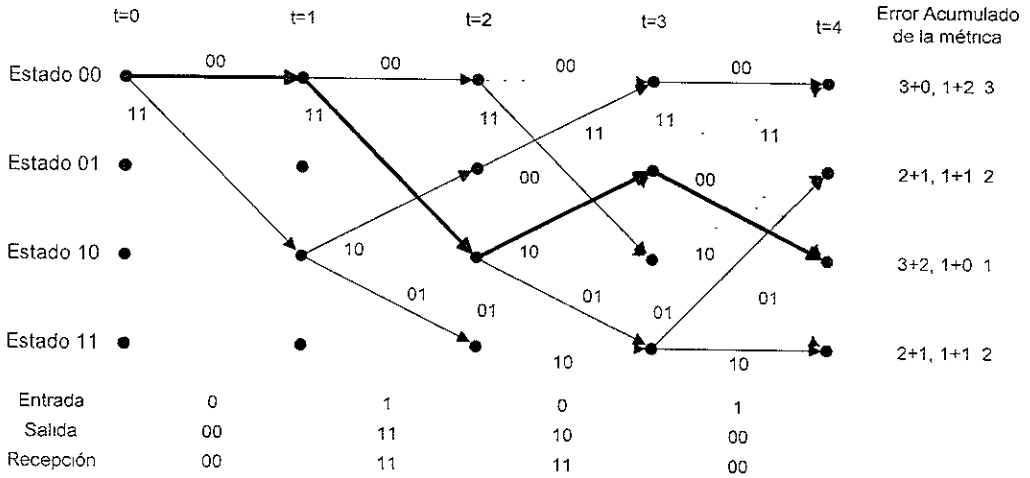


Figura 3.15 Error acumulado de la métrica hasta el instante t=4.

En la figura se observa la ruta (línea gruesa) que se siguió a través del diagrama trellis para codificar el mensaje original y esta ruta está asociada con el menor error acumulado. Los resultados hasta el instante t=5 se muestran en la **figura 3.16**.

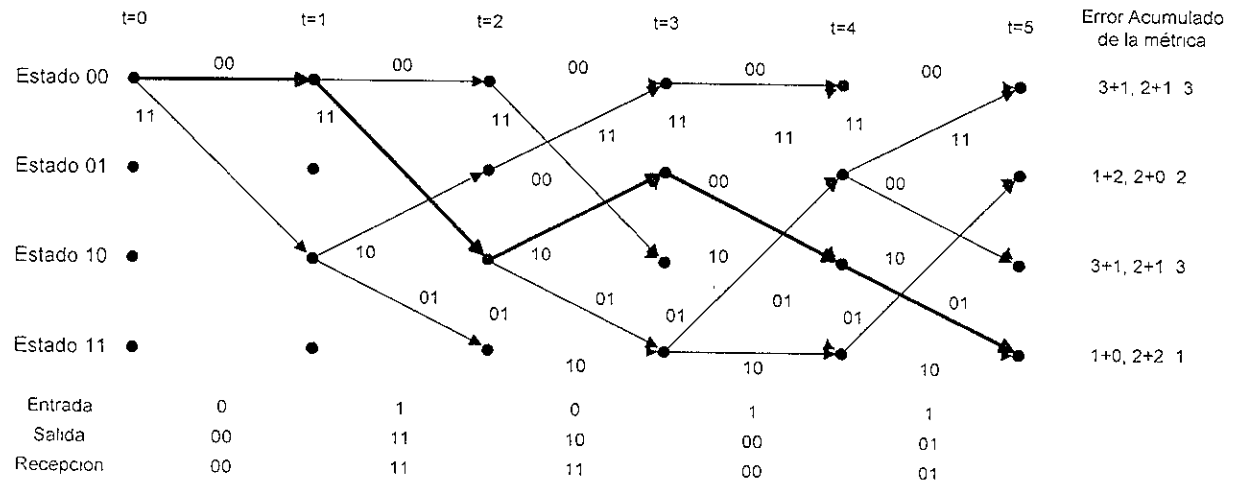


Figura 3.16 Error acumulado de la métrica hasta el instante t=5.

En el instante t=5 la ruta correcta sigue asociada con el menor error acumulado. Esta es la característica que explota el algoritmo de decodificación de Viterbi para recuperar el mensaje original. Para el instante t=17 la ruta que tiene el menor error acumulado se muestra en la **figura 3.17**.

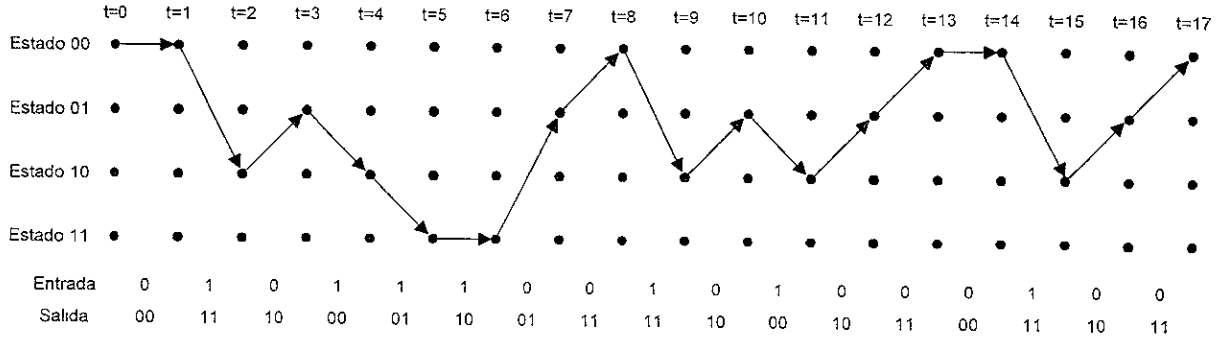


Figura 3.17 Ruta con el menor error acumulado hasta el instante t=17.

El proceso de decodificación comienza almacenando el error acumulado de la métrica para algunos pares de bits recibidos y la historia de los estados que precedieron a cada estado en cada uno de los instantes del mensaje. Una vez que se tiene esta información el decodificador esta listo para recuperar la secuencia del mensaje original. Esto se lleva a cabo mediante los siguientes pasos:

- Primero, se selecciona el estado con el menor error acumulado y se almacena el número de ese estado.
- Se lleva a cabo iterativamente el siguiente paso hasta alcanzar el comienzo del diagrama: retrocediendo por el historial de estados, para el estado seleccionado, se elige el estado marcado como predecesor. Este paso se llama *rastreo*.
- Ahora se tiene una lista de los estados seleccionados en los pasos anteriores. Se buscan los bits de entrada que corresponden con las transiciones de cada estado predecesor al estado sucesor. Esos deben ser los bits que se introdujeron en el codificador convolucional.

La siguiente tabla muestra el error acumulado para el mensaje de 15 bits (con 2 bits de limpieza) para cada instante.

t =	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Estado 00_2		0	2	3	3	3	3	4	1	3	4	3	3	2	2	4	5	2
Estado 01_2			3	1	2	2	3	1	4	4	1	4	2	3	4	4	2	
Estado 10_2		2	0	2	1	3	3	4	3	1	4	1	4	3	3	2		
Estado 11_2			3	1	2	1	1	3	4	4	3	4	2	3	4	4		

Tabla 3.2 Error acumulado de la métrica hasta el instante t=17

Es interesante notar que para este codificador de decisión dura, el menor error acumulado de la métrica en el estado final indica cuantos errores ocurrieron en la recepción del mensaje. La siguiente tabla es el historial de estados y muestra los estados predecesores seleccionados para cada estado en todos los instantes.

t =	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Estado 00_2	00	00	00	01	00	01	01	00	01	00	00	01	00	01	00	00	00	01
Estado 01_2	00	00	10	10	11	11	10	11	11	10	10	11	10	11	10	10	10	00
Estado 10_2	00	00	00	00	01	01	01	00	01	00	00	01	01	00	01	00	00	00
Estado 11_2	00	00	10	10	11	10	11	10	11	10	10	11	10	11	10	10	00	00

Tabla 3.3 Historial de estados predecesores seleccionados hasta el instante t=17.

La siguiente tabla muestra los estados seleccionados durante el rastreo a través del historial de estados de la tabla anterior.

t =	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Estado	00	00	10	01	10	11	11	01	00	10	01	10	01	00	00	10	01	00

Tabla 3.4 Estados predecesores seleccionados durante el rastreo.

Utilizando una tabla que mapea cada transición de estados con la entrada que la causó se puede reconstruir el mensaje original. La tabla para el convolucionador de este ejemplo es la siguiente:

	Siguiente Estado			
Estado Actual	00_2	01_2	10_2	11_2
00_2	0	X	1	X
01_2	0	X	1	X
10_2	X	0	X	1
11_2	X	0	X	1

Tabla 3.5 Tabla para mapear una transición con el bit de entrada que la provocó.

En la tabla anterior, la X denota una transición imposible de un estado al siguiente. El mensaje recuperado es el que se muestra a continuación:

t =	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Secuencia	0	1	0	1	1	1	0	0	1	0	1	0	0	0	1

Tabla 3.6 Secuencia decodificada.

Los dos bits de limpieza se descartan. Este es el funcionamiento del decodificador implementado para las simulaciones reportadas en esta tesis

3.4 *Intercalamiento de bits*

Debido al desvanecimiento selectivo de frecuencias de los canales de radio típicos, las subportadoras de OFDM generalmente tienen diferentes amplitudes (figura 3.4 y 3.5). Los desvanecimientos severos del espectro pueden ocasionar que algunos grupos de subportadoras sean menos confiables que otros, provocando que los errores ocurran en ráfagas en lugar de estar dispersos aleatoriamente. La mayoría de los códigos correctores de errores no están diseñados para corregir ráfagas de errores, por lo tanto se necesitan utilizar técnicas de dispersamiento de errores como el intercalamiento de bits (*bit interleaving*) o intercalamiento de símbolos (*symbol interleaving*). En la transmisión, los bits codificados son cambiados de posición de acuerdo a un algoritmo de permutación que asegurará que los bits adyacentes quedarán separados después del intercalamiento. En la recepción se utiliza el algoritmo inverso de permutación antes de la decodificación. Un esquema de intercalamiento que se utiliza comúnmente es el intercalador de bloque donde los bits de entrada se almacenan en una matriz ordenados columna por columna y los bits de salida son leídos renglón por renglón. La **figura 3.18** muestra un ejemplo de este tipo de intercalador para un bloque de 48 bits donde se ve que el orden de escritura (0, 1, 2, 3, ..., 46, 47) es diferente al orden de lectura (0, 8, 16, 24, ..., 39, 47).

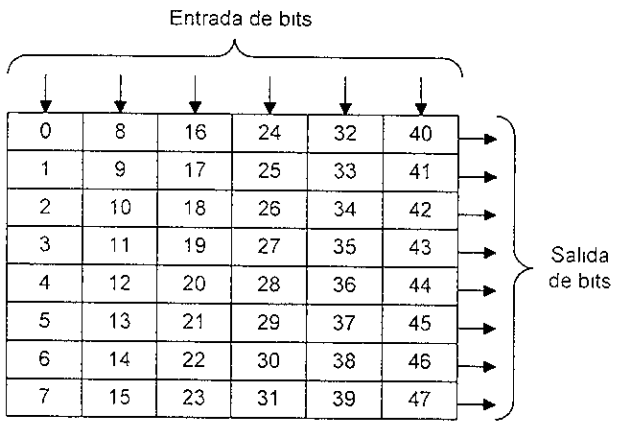


Figura 3.18 Diagrama de un intercalador de bloque para 48 bits.

En lugar de permutar bits, se puede aplicar el mismo algoritmo para permutar símbolos, lo cual es especialmente útil cuando se emplean códigos Reed-Solomon. Otro tipo de algoritmo de permutación es el que se utiliza en el intercalador convolucional. En la **figura 3.19** se muestra un ejemplo de este tipo de intercalador

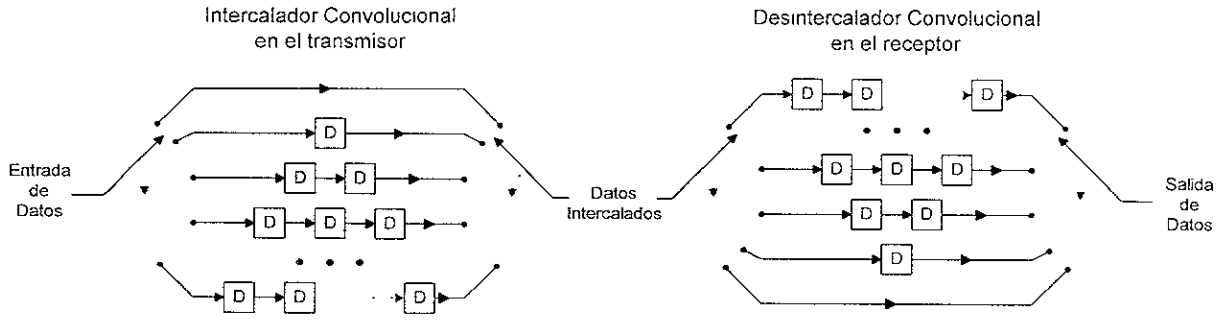


Figura 3.19 Diagrama de un intercalador y un desintercalador convolucional.

El intercalador almacena cíclicamente cada símbolo o bit de entrada en registros de corrimiento que introducen un retardo de 0 a k-1 unidades. Los registros de corrimiento son leídos cíclicamente para producir los símbolos o bits intercalados. En el desintercalador se disponen los registros de corrimiento en orden inverso para obtener la secuencia original a la entrada del intercalador.

Otro tipo de algoritmo para llevar a cabo las permutaciones de los bits consiste en definir ecuaciones de intercambio de posición de bit como las que se recomiendan en el estándar IEEE 802.11a. Estas ecuaciones se presentan en el capítulo 5 y son las que se utilizan para llevar a cabo el intercalamiento y desintercalamiento en las simulaciones reportadas en este estudio.

4 Sincronización

4.1 Importancia

Para que un receptor OFDM pueda demodular correctamente las subportadoras de una señal necesita realizar por lo menos dos tareas de sincronización. Primero tiene que encontrar los límites de un símbolo dado y calcular los instantes óptimos para tomar las muestras con el fin de minimizar los efectos de la interferencia entre portadoras (ICI) y de la interferencia entre símbolos (ISI). La segunda tarea importante en el caso de los receptores coherentes es la de estimar la desviación de fase que sufre la portadora en su transmisión. La desviación de fase provoca que las constelaciones QAM de las subportadoras estén rotadas un cierto ángulo el cual es necesario corregir para evitar errores en la detección de los símbolos (capítulo 5).

Otra de las causas de interferencia ICI es la desviación de frecuencia de la portadora que sufre la transmisión debido a que los osciladores del transmisor y del receptor no son completamente precisos, provocando un cambio de fase de la portadora en el tiempo.

4.2 Ruido de Fase

En un enlace OFDM, las subportadoras son perfectamente ortogonales sólo si el transmisor y el receptor tienen exactamente la misma frecuencia de portadora. Cualquier desviación de frecuencia se traduce en interferencia entre portadoras. El ruido de fase está relacionado con este tipo de fenómeno, ya que un oscilador práctico no genera una portadora con una frecuencia única, sino una señal que se encuentra modulada en fase de forma aleatoria. Como consecuencia, la frecuencia (que es la derivada con respecto al tiempo de la fase) no permanece constante, introduciendo ICI en el receptor. En los sistemas de portadora única, este efecto causa una degradación en la relación señal a ruido (SNR) en vez de causar interferencia; por esta razón la sensibilidad al ruido de fase y las desviaciones de frecuencia se consideran desventajas de OFDM con respecto a los sistemas de portadora única.

El problema relacionado con el ruido de fase se ha estudiado extensivamente. Un ejemplo es el artículo [8], donde el espectro de densidad de potencia de un oscilador con ruido de fase se modela por medio de un espectro Lorentziano, que es igual a la magnitud de la función de transferencia de un filtro pasabajas al cuadrado. El espectro de un lado correspondiente está dado por la ecuación 4.1:

$$S_s(f) = \frac{2/\pi f_l}{1 + f^2 / f_l} \tag{4.1}$$

f_l es el ancho de banda a –3 dB de la señal del oscilador. En la práctica se mide el espectro de ambos lados alrededor de una frecuencia central f_c, que es el reflejo del espectro de un solo lado (ecuación 4.1). Como el ancho de banda se duplica, es necesario dividir el espectro entre dos para que la potencia se mantenga normalizada. La ecuación 4.2 representa el espectro completo normalizado del ruido de fase:

$$S_d(f) = \frac{1/\pi f_l}{1 + |f - f_c|^2 / f_l^2} \tag{4.2}$$

La **figura 4.1** muestra un ejemplo del modelo Lorentziano para el ruido de fase, con espectro de un solo lado y ancho de banda de 1 Hz a –3 dB. Se puede ver en esta figura que la densidad espectral de potencia (PSD) del ruido de fase disminuye 20 dB por década con respecto a la portadora. Esto quiere decir que en una frecuencia desviada 100 kHz con respecto a la frecuencia central de la portadora, la potencia se encuentra 100 dBc/Hz por debajo de la potencia máxima.

El ruido de fase presenta principalmente dos efectos. Uno de estos efectos es una variación aleatoria de fase común a todas las subportadoras. Si se tiene un ancho de banda del oscilador mucho menor que la tasa de símbolos de OFDM (cosa que sucede frecuentemente), el error común de fase está fuertemente correlacionado entre símbolos. Este fenómeno facilita la corrección de estas desviaciones de fase por medio de detección diferencial o de técnicas de rastreo. El segundo efecto que se presenta debido al ruido de fase es la ICI, debido a que las subportadoras en la práctica no se encuentran separadas 1/T en el dominio de la frecuencia.

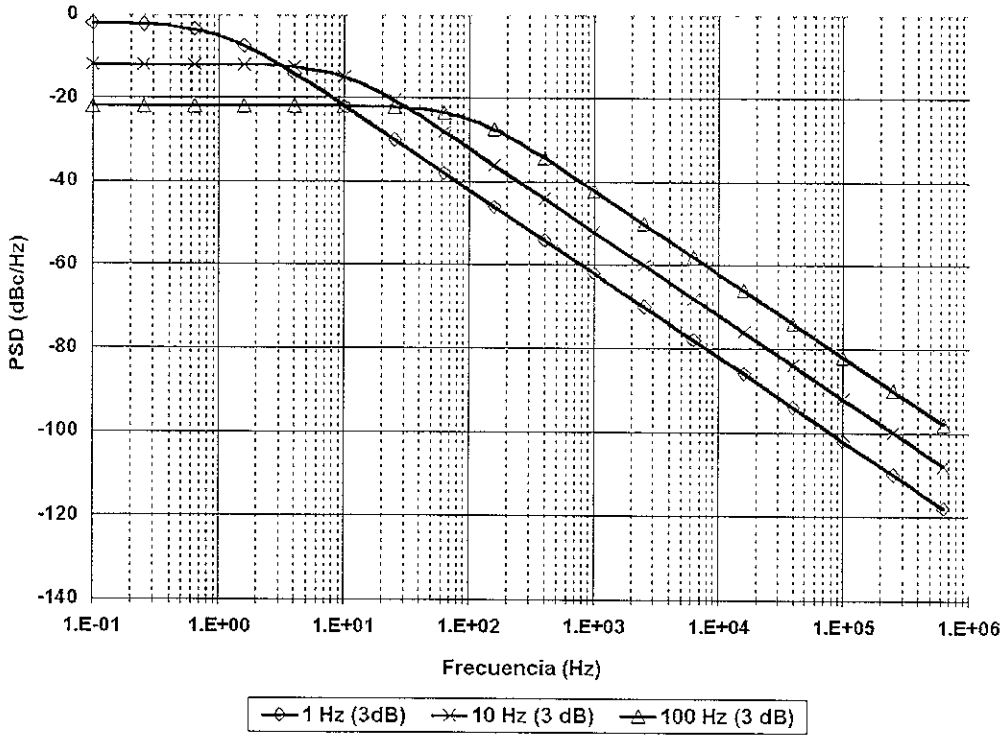


Figura 4.1 Densidad espectral de potencia del ruido de fase para diferentes anchos de banda a –3 dB.

4.3 Desviaciones de Frecuencia

Las desviaciones de frecuencia que ocurren en los osciladores que generan las portadoras, tienen como consecuencia que el número de ciclos en el intervalo donde se realiza la FFT no es entero. Como se explicó anteriormente, una de las condiciones para que todas las subportadoras sean ortogonales es que el intervalo de la FFT tenga un número entero de ciclos. Por lo tanto, las desviaciones de frecuencia causan interferencias entre las portadoras después de la FFT. La salida de la FFT de cada subportadora tendrá entonces términos interferentes que provienen de otras subportadoras, con una potencia inversamente proporcional a la separación en frecuencia entre éstas. La cantidad de ICI en el centro del espectro de una señal OFDM es aproximadamente el doble que las interferencias en los bordes de la banda, ya que las subportadoras que se encuentran cerca de las frecuencias centrales tienen interferencias provenientes de subportadoras de ambos lados, por lo que hay más interferencia proveniente de un mismo ancho de banda.

Una forma de cuantificar esta degradación se muestra en la referencia [8], en donde la degradación de la relación señal a ruido (SNR) causada por una desviación de frecuencia (que debe de ser pequeña con respecto a la separación entre subportadoras) se aproxima por medio de la siguiente ecuación:

$$Deg_{frec} \cong \frac{10}{3\ln 10}(\pi \Delta f T)^2 \frac{E_b}{N_0} \tag{4.3}$$

Al realizar una gráfica de esta ecuación podemos conocer la degradación teórica debido a errores de frecuencia para diferentes valores de desviación con respecto al espacio entre subportadoras, obteniendo una familia de curvas como se muestra en la **figura 4.2**.

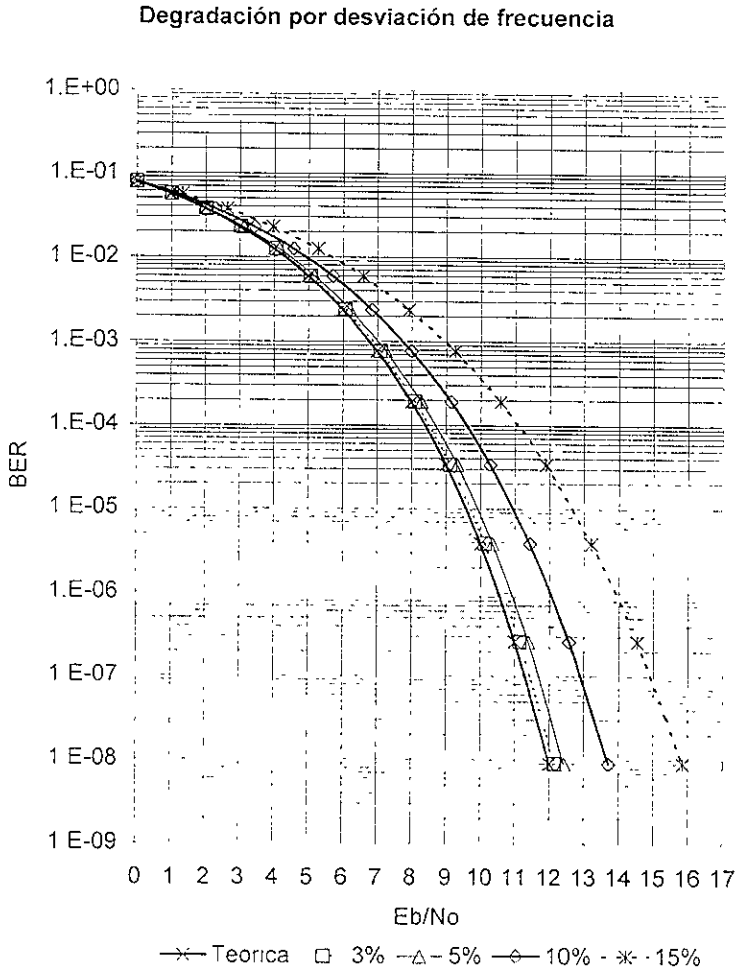


Figura 4.2 Sensibilidad de un sistema OFDM-BPSK a las desviaciones de frecuencia con diferentes porcentajes con respecto al espacio entre subportadoras.

Por ejemplo, para una desviación de frecuencia del 5% con respecto a la frecuencia exacta del oscilador, se tienen pérdidas de hasta 0.5 dB para obtener una tasa de error de bits de 10^{-8}. Esto demuestra su impacto y la necesidad, ya sea de utilizar osciladores con un alto grado de exactitud, o bien de realizar algún tipo de corrección para minimizar los efectos de este fenómeno físico.

4.4 Temporización

En un sistema OFDM, es necesario encontrar el instante de muestreo óptimo, es decir, el momento en el que se debe de realizar la FFT dentro de un símbolo sin que se introduzca interferencia proveniente de los símbolos contiguos. OFDM es relativamente robusto con respecto a estas desviaciones de tiempo, ya que el intervalo de integración puede variar dentro de un intervalo de tiempo igual que el intervalo de guarda sin causar ICI o ISI. Un ejemplo se muestra en la **figura 4.3**.

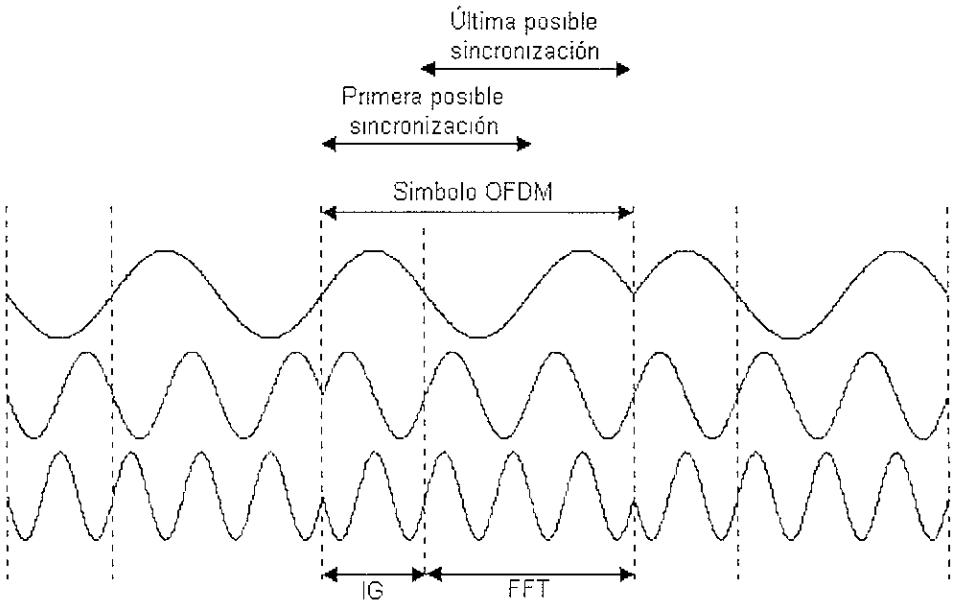


Figura 4.3 Ejemplo de una señal OFDM con tres subportadoras, en donde se puede ver el instante de tiempo máximo y mínimo para tomar las muestras evitando interferencias.

Este tipo de interferencia ocurre sólo cuando el intervalo en el que se realiza la FFT se extiende dentro de los límites de otro símbolo o en su región de traslape. El intervalo de guarda que se introduce en los símbolos de OFDM hace la demodulación poco sensible a las

desviaciones de tiempo. Sin embargo, como se mencionaba anteriormente, existe un instante óptimo que evita que la sensibilidad a los retardos se incremente. El control de este fenómeno es importante sobre todo cuando se trata con señales que presentan múltiples trayectorias. Para minimizar esta pérdida de eficiencia, el sistema se debe diseñar de forma que el error de tiempo es pequeño comparado con la duración del intervalo de guarda.

Existe una relación entre la temporización y las fases de las subportadoras demoduladas. Si vemos la figura 4.3, podemos notar que cuando el instante de muestreo cambia, las fases de las subportadoras cambian también. Esta relación se puede expresar matemáticamente por medio de la ecuación (4.4), en donde la desviación de tiempo es τ y la fase de la subportadora i es φ_i:

$$\varphi_i = 2\pi f_i \tau \tag{4.4}$$

En esta ecuación, f_i es la frecuencia de la subportadora i antes del proceso de muestreo. Para un sistema OFDM con N subportadoras y un espacio entre portadoras de $1/T$, un retraso en tiempo equivalente a un intervalo de muestra T/N provoca una desviación de fase significativa igual a $2\pi(1-1/N)$ entre la primera subportadora y la última. Un ejemplo del efecto que tiene en la fase las desviaciones de tiempo se observa en la **figura 4.4**, en la que una constelación QPSK sufre rotaciones de fase en N debidas a errores en la temporización:

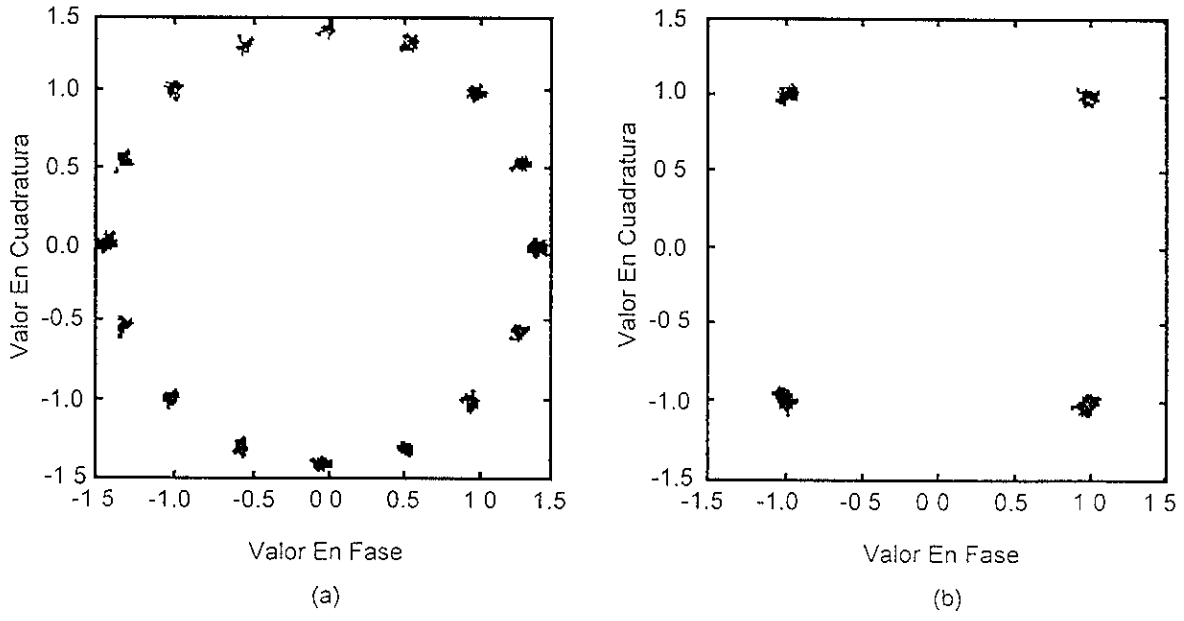


Figura 4.4 Constelación QPSK con un error de tiempo equivalente a $T/16$ *(a)* sin corrección y *(b)* corregida

4.5 Algoritmos de Sincronización

Debido a la existencia de una extensión cíclica dentro del símbolo OFDM, la primera parte de T_G segundos de cada símbolo es idéntica a la última parte. Esta propiedad puede ser explotada para sincronizar fase, frecuencia y temporización utilizando un sistema de sincronización como el de la **figura 4.5**. Básicamente este sistema correlaciona una parte de T_G segundos de la señal con otra parte que está retardada T segundos. La salida del correlacionador puede escribirse como:

$$x(t) = \int_0^{T_G} r(t-\tau) r^*(t-\tau-T) d\tau \tag{4.5}$$

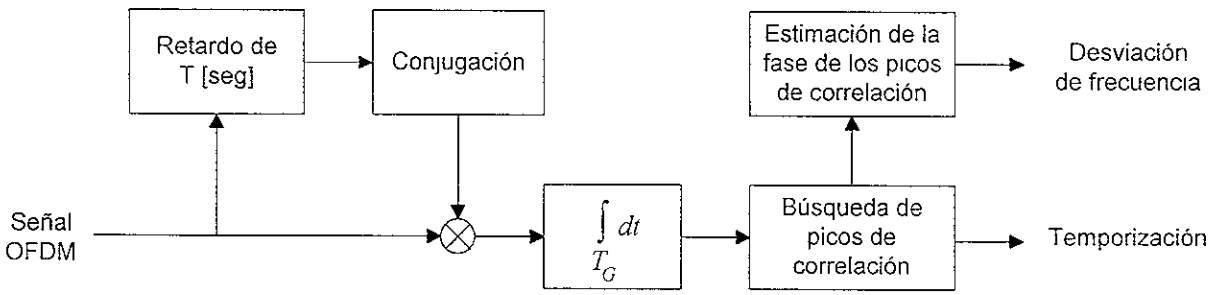


Figura 4.5 Sincronización utilizando la extensión cíclica.

Las **figuras 4.6** y **4.7** muestran dos ejemplos de la salida del correlacionador para ocho símbolos OFDM con 192 y 48 subportadoras, respectivamente. Estas figuras muestran algunas características interesante del método de correlación de la extensión cíclica. En primer lugar, ambas figuras muestran claramente ocho picos para cada uno de los ocho diferentes símbolos, pero las amplitudes de los picos muestran una variación significativa. La razón de esto es que a pesar de que la potencia promedio para un intervalo de T segundos de cada símbolo OFDM es constante, la potencia en el intervalo de guarda puede variar notablemente con respecto al nivel promedio de potencia. Otro efecto es el nivel de los lóbulos laterales indeseados de la correlación entre dos partes de la señal OFDM que pertenecen parcial o totalmente a dos símbolos diferentes. Debido a que diferentes símbolos contienen datos independientes, la salida del correlacionador es una variable aleatoria la cual alcanza su valor máximo en el pico de correlación deseado. La desviación estándar de la magnitud de la correlación se relaciona con el número de muestras independientes sobre las cuales se lleva a cabo la correlación. Mientras sea más grande el número

de muestras independientes, la desviación estándar disminuirá. En el caso extremo donde la correlación se lleva a cabo sobre una muestra solamente, la magnitud de la salida es proporcional a la potencia de la señal. En el otro caso extremo donde la correlación se lleva a cabo sobre un gran número de muestras la relación lóbulos laterales/pico tenderá a cero. Debido a que el número de muestras independientes es proporcional al número de subportadoras, la técnica de correlación de la extensión cíclica es mas eficiente cuando se utiliza un gran número de subportadoras, preferentemente más de 100. Una excepción a esto es el caso donde se utilizan símbolos de entrenamiento especialmente diseñados en lugar de símbolos con datos aleatorios. En este caso, la integración puede llevarse a cabo sobre la duración entera del símbolo en lugar del intervalo de guarda solamente. El nivel de lóbulos laterales indeseados puede ser minimizado con una selección apropiada de los símbolos de entrenamiento.

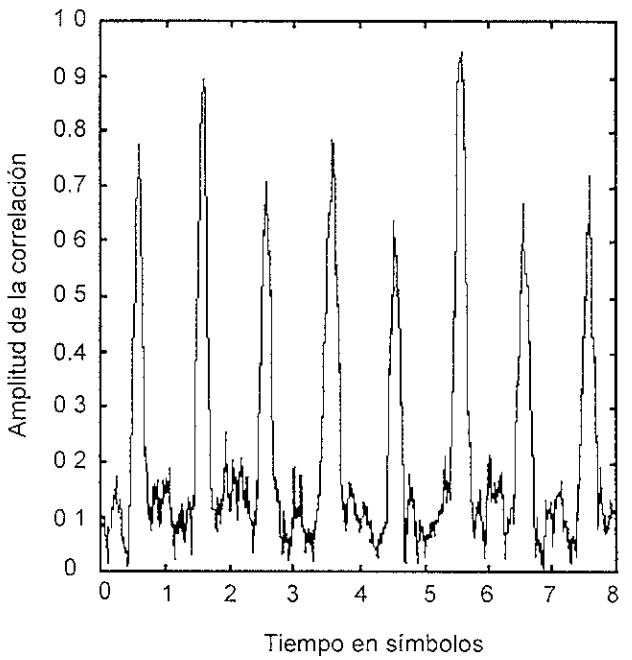


Figura 4.6 Ejemplo de la salida del correlacionador para ocho símbolos con 192 subportadoras y 20 % de intervalo de guarda.

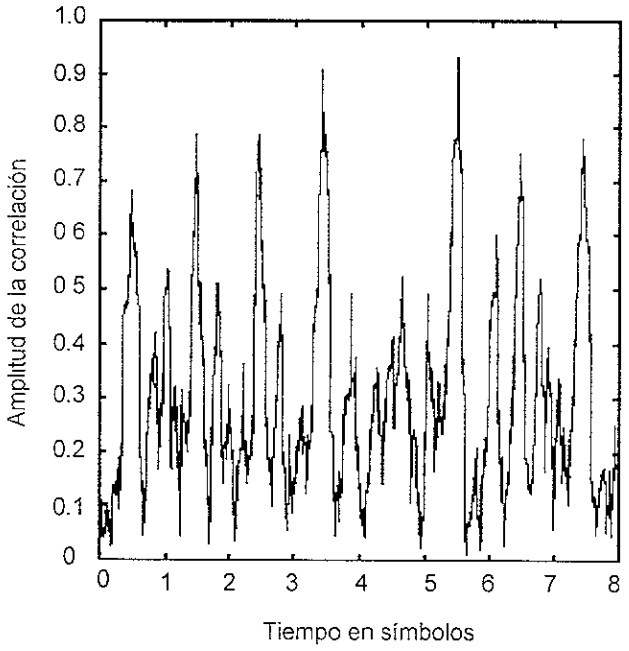


Figura 4.7 Ejemplo de la salida del correlacionador para ocho símbolos con 48 subportadoras y 20 % de intervalo de guarda.

Se puede notar que los lóbulos laterales indeseados solamente representan un problema para la sincronización del intervalo de la FFT. Para la estimación de frecuencia no representa un conflicto. Una vez que se conoce la sincronización del símbolo, la salida de la correlación puede ser utilizada para estimar la desviación de frecuencia. La fase de la salida de la correlación es igual a la desviación de fase entre muestras que están T segundos separadas. Por lo tanto, la desviación de frecuencia puede ser encontrada con la fase de la correlación dividida por $2\pi T$. Este método funciona hasta un máximo absoluto de desviación de frecuencia de la mitad del espaciamiento entre subportadoras.

La técnica de sincronización basada en la extensión cíclica es utilizada particularmente para el rastreo de la desviación de frecuencia en una conexión de circuito conmutado donde no se puede disponer de señales especiales de entrenamiento. Sin embargo, en la transmisión de paquetes no resulta ser una técnica eficiente ya que para obtener una sincronización aceptable se necesita un número grande (>10) de símbolos para tener un pico de correlación de buen nivel. En una transmisión de paquetes con alta tasa de datos se requiere que el tiempo de sincronización sea el menor posible, preferentemente de unos cuantos símbolos. Para llevar a cabo esto, se pueden utilizar símbolos especiales de entrenamiento los cuales contienen datos conocidos por el

5 Simulaciones

Para llevar a cabo las simulaciones se utilizó el modelo de comunicación descrito por el siguiente gráfico:

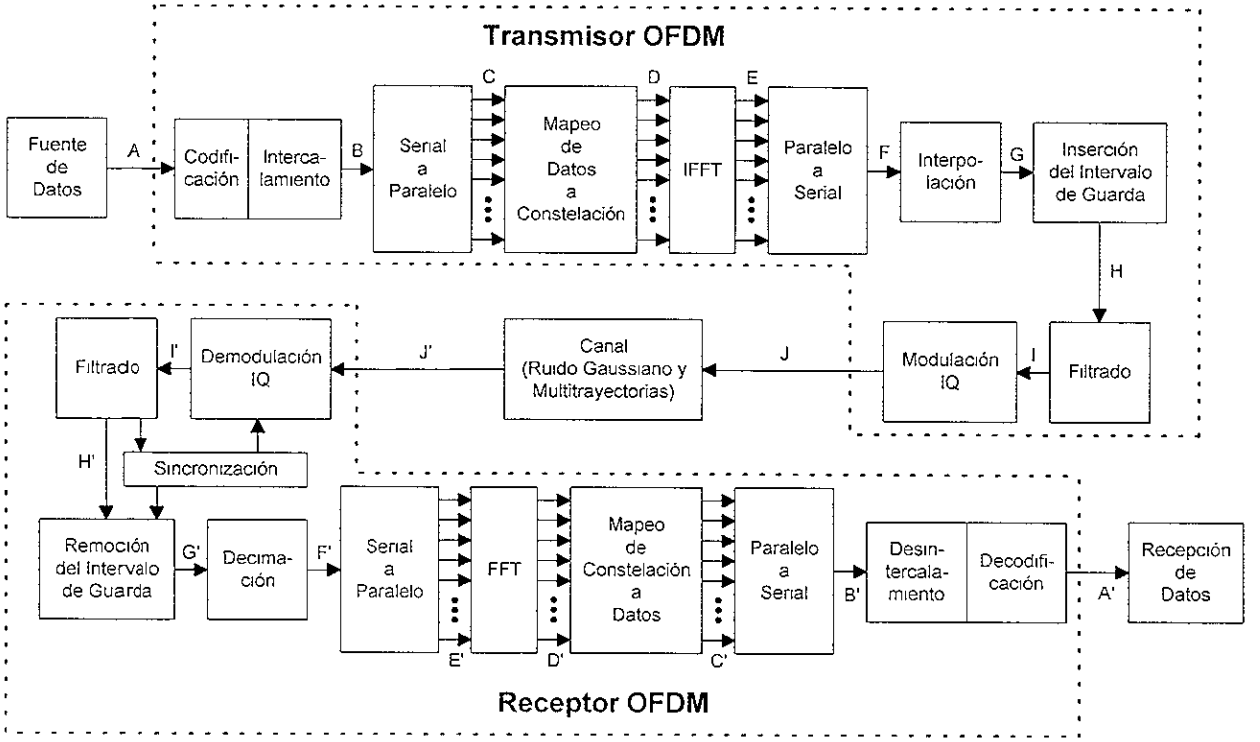


Figura 5.1 Modelo utilizado para realizar las simulaciones.

En él se muestran cada uno de los bloques que se utilizaron para obtener los resultados que se reportan en esta tesis. Los parámetros utilizados para la simulación son los que se recomiendan en la especificación de Capa Física (PHY) del estándar IEEE 802.11a:

- Tamaño de la IFFT y FFT: 64 puntos
- Subportadoras de datos: 48
- Subportadoras piloto: 4
- Subportadoras totales: 52
- Relación IG/FFT: 1/4

Los datos utilizados para generar el preámbulo corto y largo son los que especifica el estándar pero el número óptimo de símbolos utilizados se obtuvo de las simulaciones. Para el preámbulo corto, utilizado para sincronizar la fase de la portadora, se obtuvo que 3 símbolos eran suficientes para tener una degradación despreciable con respecto de la curva teórica para los tres tipos de constelaciones utilizadas: BPSK, QPSK y 16QAM (Sección 5.6). Para el preámbulo largo se obtuvo que con 7 símbolos la degradación era menor que 0.5 dB de relación Eb/No para los tres tipos de constelaciones. Los polinomios generadores del codificador convolucional son los que se definen en el estándar IEEE 802.11a (figura 3.6). Los mapas de las constelaciones son los que se muestran en la **figura 5.2**.

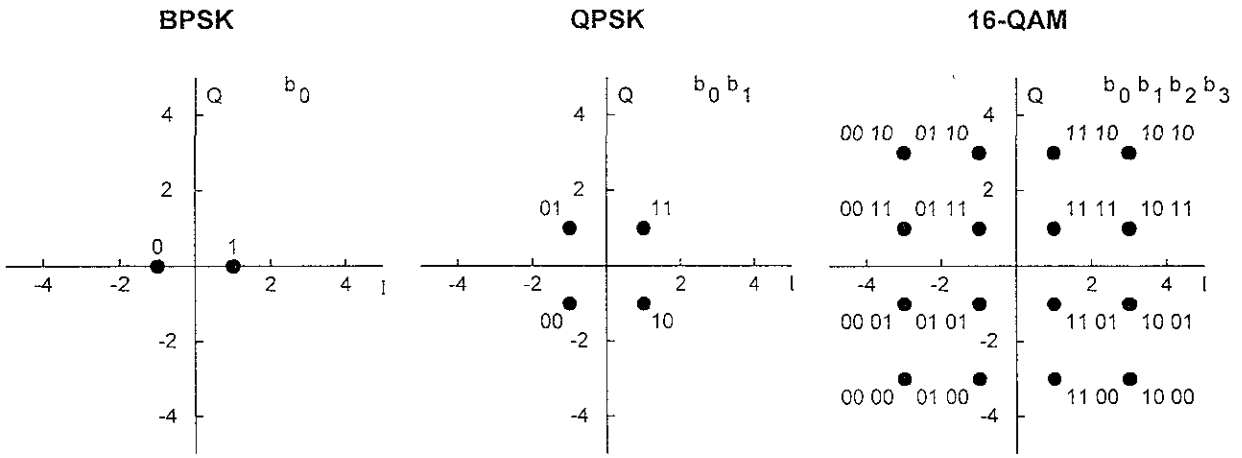


Figura 5.2 Mapas de las constelaciones utilizadas en las simulaciones.

Las tablas de codificación utilizadas para mapear los datos transmitidos con los puntos de las constelaciones son las siguientes:

Bit de Entrada (b0)	Salida I	Salida Q
0	-1	0
1	1	0

Tabla 5.1 Tabla de codificación para BPSK.

Bit de Entrada (b0)	Salida I
0	-1
1	1

Bit de Entrada (b1)	Salida Q
0	-1
1	1

Tabla 5.2 Tabla de codificación para QPSK.

Bits de Entrada (b0 b1)	Salida I
00	-3
01	-1
11	1
10	3

Bits de Entrada (b2 b3)	Salida Q
00	-3
01	-1
11	1
10	3

Tabla 5.3 Tabla de codificación para 16-QAM.

Las funciones de cada uno de los bloques de la **figura 5.1** se describen en la **tabla 5.4**.

Bloque	Función
Fuente de datos	Los datos a transmitir se leen de un archivo cuya información es aleatoria.
Codificador Convolucional	Lleva a cabo la codificación de los datos utilizando la configuración de la figura 3.6.
Intercalador	Realiza el reordenamiento de los bits para dispersar el error en ráfagas que pudiera sufrir el símbolo.
Convertidor serial a paralelo	Los datos codificados e intercalados se agrupan en 48 conjuntos de 1, 2 y 4 bits, dependiendo de la constelación que se desee utilizar (BPSK, QPSK o 16QAM). Cada conjunto será transportado por una subportadora por lo que cada símbolo podrá transmitir 48, 96 o 192 bits.
Mapeo en la constelación	Relaciona el dato que se desea transmitir con el valor real e imaginario de la constelación para cada una de las 48 subportadoras.
IFFT	Realiza la Transformada Rápida de Fourier Inversa de 64 puntos.
Convertidor paralelo a serial	Los valores real e imaginario que salen en paralelo del bloque de IFFT se ordenan secuencialmente para formar el símbolo. Los valores reales serán la señal en fase del símbolo y los valores imaginarios serán la señal en cuadratura.
Interpolación	La señal en cuadratura y en fase se interpolan para pasar de una tasa de muestreo T_s (64 muest/simb) a una tasa de muestreo $2T_s$ (128 muest/simb).
Inserción del Intervalo de Guarda	Agrega la última parte del símbolo al principio para obtener la extensión cíclica del símbolo.
Filtrado del símbolo	Se aplica la ventana de coseno elevado al principio y al final del símbolo.
Modulación IQ	Se multiplica la señal en fase por el coseno de la portadora y de la señal en cuadratura por el seno de la portadora y se suman ambas señales
Multitrayectoria	Se agregan multitrayectorias a la señal transmitida por medio de un filtro FIR.
Ruido Gaussiano	Se agrega ruido de distribución gaussiana con media = 0 y desviación estándar variable de acuerdo con la relación Eb/No que se desee simular.
Demodulación IQ	Multiplicación de la señal transmitida por coseno para obtener la señal en fase y por seno para obtener la señal en cuadratura
Filtrado de la señal en bandabase	Se procesa la señal en fase y en cuadratura con un filtro de Nyquist para obtener las señales en bandabase.
Sincronización de fase y temporización	Se lleva a cabo la correlación del preámbulo corto para sincronizar la fase de la portadora y se realiza la correlación del preámbulo largo para alinear el periodo de FFT con el símbolo.
Remoción del Intervalo de Guarda	Elimina la extensión cíclica del símbolo
Decimador	Las señales en cuadratura y en fase se deciman para pasar de una tasa de muestreo $2T_s$ a una tasa de muestreo T_s.
Convertidor serial a paralelo	Ordena las muestras para alimentar las entradas real e imaginaria de cada uno de los puntos de la FFT
FFT	Realiza la Transformada Rápida de Fourier de 64 puntos

Mapeo de la constelación	Relaciona los valores de amplitud de cada subportadora con los límites de cada punto de la constelación para obtener el dato que se transmitió.
Convertidor paralelo a serial	Pasa los datos en paralelo a una secuencia de bits.
Desintercalador	Reordena los bits para tener la secuencia original antes del intercalamiento en la transmisión.
Decodificador	Decodifica los datos para disminuir el número de errores de la transmisión utilizando el algoritmo de Viterbi.
Receptor de datos	Los datos recibidos se escriben en un archivo para su posterior comparación con el archivo de transmisión La comparación arroja como resultado un valor de BER.

Tabla 5.4 Funciones de los bloques en el sistema.

Los tipos de datos y el número de flujos que se obtienen entre cada bloque son los siguientes:

Punto	Tipo de dato	Número de flujos
A y A'	Binario	1
B y B'	Binario	1
C y C'	Binario	48
D y D'	Complejo	64 Parte real
		64 Parte imag
E y E'	Complejo	64 Parte real
		64 Parte imag
F y F'	Complejo	1 Parte real
		1 Parte imag
G y G'	Complejo	1 Parte real
		1 Parte imag
H y H'	Complejo	1 Parte real
		1 Parte imag
I y I'	Complejo	1 Parte real
		1 Parte imag
J y J'	Real	1

Tabla 5.5 Tipo de datos y número de flujos en el sistema.

5.1 Canal de Transmisión

El canal de transmisión se simuló por medio de dos procesos: agregación de multitrayectorias y adición de ruido con distribución de probabilidad gaussiana.

Para agregar multitrayectorias a la señal transmitida se utiliza un filtro FIR cuyos coeficientes son iguales a las amplitudes relativas de un perfil de canal con multitrayectoria como el de la figura 2.2. Mientras el retardo aumenta la amplitud relativa a la señal original disminuye

de una forma exponencial de acuerdo a lo establecido en la sección 2.2. Para las simulaciones de esta tesis se tiene que el intervalo de guarda consta de 32 muestras por lo que un retardo menor no provocará ISI ni ICI a la señal. Los perfiles utilizados en las simulaciones se muestran en la **figura 5.3**.

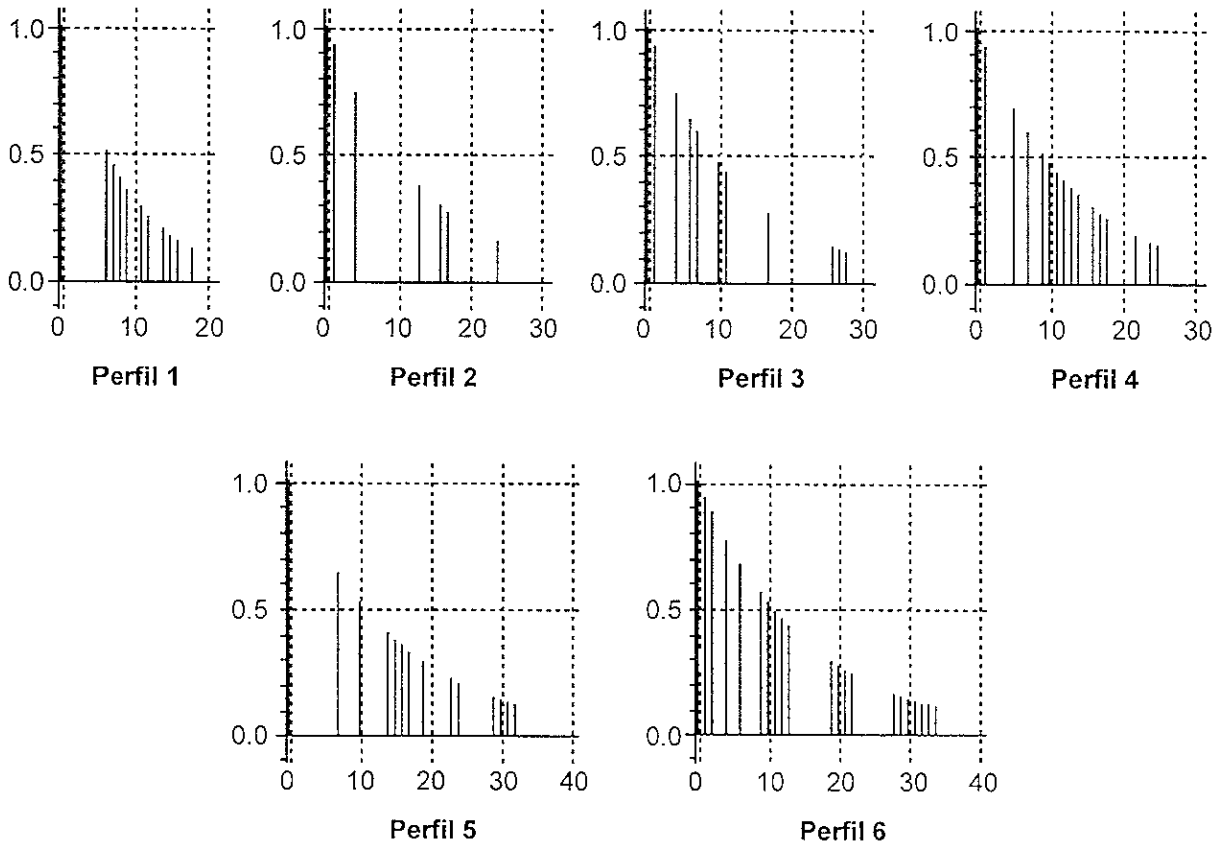


Figura 5.3 Perfiles de canal utilizados en las simulaciones.

Se puede ver que los todos los perfiles tienen un retardo menor a 32 muestras excepto el perfil 6. El desempeño del sistema para ese perfil no es satisfactorio ya que el retardo es mayor al 15 % del intervalo de guarda provocando la existencia de ISI e ICI. El factor más relevante en el desempeño del sistema no es el número de multitrayectorias sino el retardo máximo de éstas.

Las subportadoras son atenuadas o amplificadas de diferente forma para cada uno de los perfiles. En la **figura 5.4** se muestra el cambio de amplitud de las subportadoras para el preámbulo largo contra el número de punto a la salida de la FFT. Esta salida es importante para estimar el factor de corrección de amplitud de cada una de las subportadoras y obtener constelaciones del mismo tamaño, aproximadamente

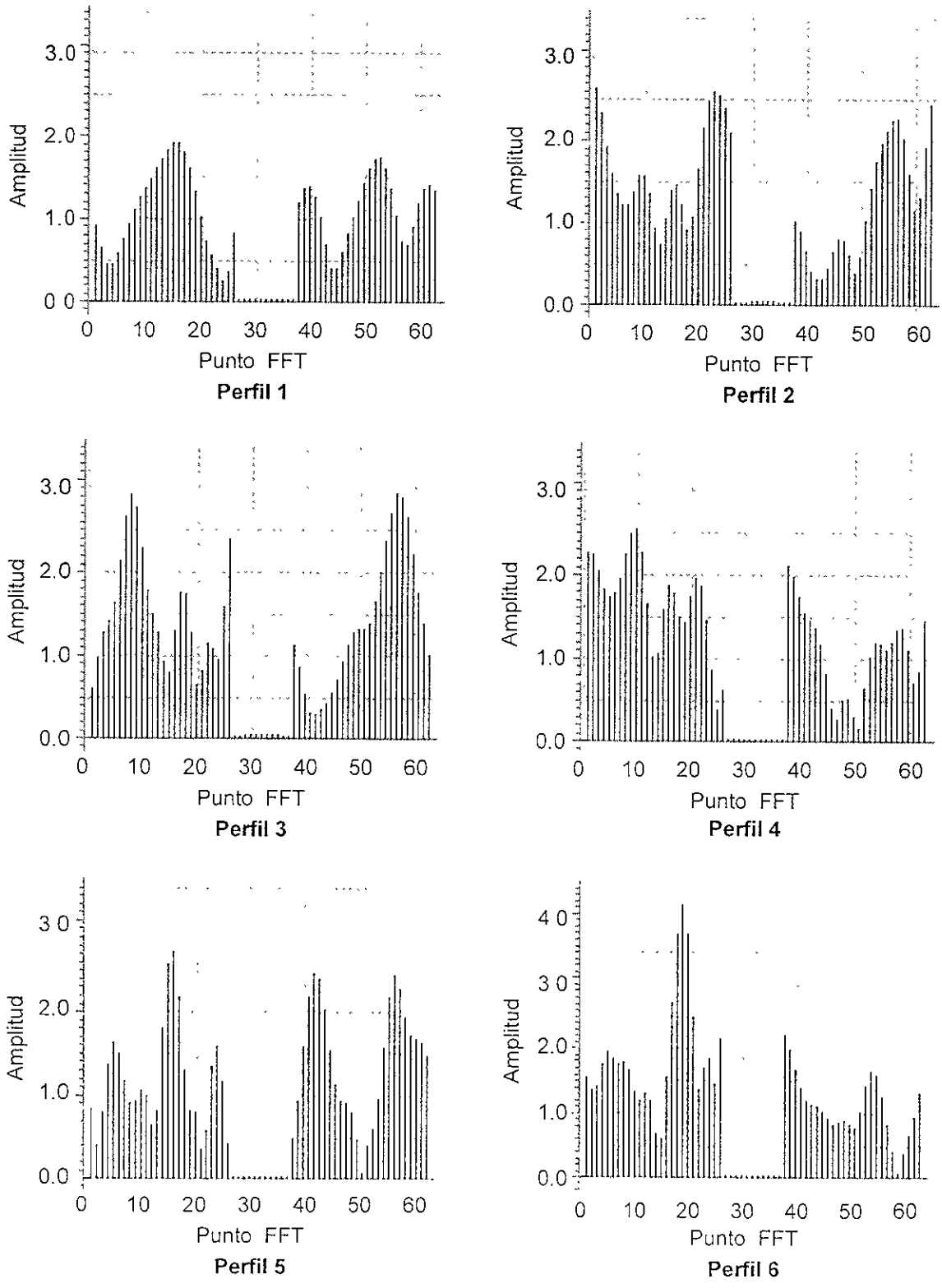


Figura 5.4 Amplificación y atenuación de las subportadoras para los diferentes perfiles de canal

En las gráficas anteriores se puede ver que las subportadoras pueden tener una amplitud del doble o triple con respecto a la amplitud original de 1, así como también pueden tener amplitudes de la mitad o menos. Por lo tanto la relación SNR es diferente para cada subportadora haciendo más confiables a unas subportadoras y menos confiables a otras.

Para agregar ruido con distribución gaussiana se utiliza el generador de números aleatorios del lenguaje C. Por medio de dos variables con distribución uniforme de probabilidad se puede obtener una variable de distribución gaussiana si: u, v son variables con distribución uniforme dentro del intervalo [0,1), σ es la desviación estándar, $N_0/2$ es el nivel de la densidad espectral de potencia del ruido y

$$\sigma = \sqrt{\frac{N_0}{2}} \tag{5.1}$$

$$R = \sigma \sqrt{2\log\left(\frac{1}{1-u}\right)}$$

$$G = R\cos(2\pi v)$$

donde R es una variable con distribución de Rayleigh y G una variable con distribución gaussiana.

Para obtener diferentes relaciones de E_b/N_0 se tiene que variar la desviación estándar de acuerdo a la relación:

$$\sigma = \sqrt{\frac{N_0}{2}} = \sqrt{\frac{E_b}{2\left(\frac{E_b}{N_0}\right)}} \tag{5.2}$$

donde: $\frac{E_b}{N_0} = 10^{\left(\frac{1}{10}\left(\frac{E_b}{N_0}\right)_{dB}\right)}$

Para cada muestra de la señal transmitida se suma un valor G con distribución gaussiana con lo que se simula la presencia de ruido en el canal.

5.2 *Transformada Rápida de Fourier*

Como se mencionó en el capítulo 2, el algoritmo que se utilizó para implementar la Transformada Rápida de Fourier fue el de raíz cuarta. Desarrollando la ecuación (2.6) con N=4 obtenemos las ecuaciones para implementar la mariposa de raíz cuarta:

$$\begin{aligned} x[0] &= X[0] + X[1] - X[2] + X[3] \\ x[1] &= X[0] + j\,X[1] - X[2] - j\,X[3] \\ x[2] &= X[0] - X[1] + X[2] - X[3] \\ x[3] &= X[0] - j\,X[1] - X[2] + j\,X[3] \end{aligned} \tag{5.3}$$

que se ilustran gráficamente en la figura 2.8. Desarrollando para la parte real y la parte imaginaria se tienen las ecuaciones:

$$\begin{aligned} x_R[0] &= X_R[0] + X_R[1] + X_R[2] + X_R[3] \\ x_R[1] &= X_R[0] - X_I[1] - X_R[2] + X_I[3] \\ x_R[2] &= X_R[0] - X_R[1] + X_R[2] - X_R[3] \\ x_R[3] &= X_R[0] + X_I[1] - X_R[2] - X_I[3] \end{aligned} \tag{5.4}$$

$$\begin{aligned} x_I[0] &= X_I[0] + X_I[1] + X_I[2] + X_I[3] \\ x_I[1] &= X_I[0] + X_R[1] - X_I[2] - X_R[3] \\ x_I[2] &= X_I[0] - X_I[1] + X_I[2] - X_I[3] \\ x_I[3] &= X_I[0] - X_R[1] - X_I[2] + X_R[3] \end{aligned}$$

que, como se puede ver, constan de sumas y restas únicamente, además de algunas rotaciones entre parte real y parte imaginaria. Para llevar a cabo las rotaciones de fase que se muestran en la figura 2.9 (multiplicaciones por factores ω^i) se utiliza la transformada de rotación general:

$$\begin{aligned} x' &= x\cos\phi - y\,sen\,\phi \\ y' &= y\cos\phi + x\,sen\,\phi \end{aligned} \tag{5.5}$$

si tomamos en cuenta que $\omega^i = exp(j2\pi i/N)$ representa una rotación por el ángulo $2\pi i/N$ entonces tenemos que:

$$\phi = 2\pi\, i/N \tag{5.6}$$

Aplicando el conjunto de ecuaciones (5.4), (5.5) y (5.6) en el esquema de la figura 2.9 obtenemos la Transformada Rápida de Fourier inversa y directa de 16 puntos. Los factores de giro para el primer nivel son:

n	i 1er
0	0
1	0
2	0
3	0

n	i 1er
4	0
5	1
6	2
7	3

n	i 1er
8	0
9	2
10	4
11	6

n	i 1er
12	0
13	3
14	6
15	9

Tabla 5.6 Factores de giro ω^i para la IFFT de 16 puntos.

Para el primer nivel de la transformada se tiene que alimentar la mariposa de raíz cuarta con las siguientes combinaciones de puntos:

Número de combinación	X[0] x[0]	X[1] x[1]	X[2] x[2]	X[3] x[3]
1	0	4	8	12
2	1	5	9	13
3	2	6	10	14
4	3	7	11	15

Tabla 5.7 Combinaciones para el primer nivel de la IFFT de 16 puntos.

Para el segundo nivel de la transformada las salidas se escriben en una localidad diferente de la de lectura:

Número de combinación	X[0]	X[1]	X[2]	X[3]	x[0]	x[1]	x[2]	x[3]
1	0	1	2	3	0	4	8	12
2	4	5	6	7	1	5	9	13
3	8	9	10	11	2	6	10	14
4	12	13	14	15	3	7	11	15

Tabla 5.8 Combinaciones para el segundo nivel de la IFFT de 16 puntos.

Los factores de giro y las combinaciones, así como las localidades de lectura y escritura, se pueden verificar en la figura 2.9.

Para obtener la transformada de 64 puntos también se necesitan saber los factores de giro ω^i para el primer y segundo nivel de las transformadas. Estos factores se listan en la **tabla 5.9**.

n	i 1er	2do
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
5	0	4
6	0	8
7	0	12
8	0	0
9	0	8
10	0	16
11	0	24
12	0	0
13	0	12
14	0	24
15	0	36

n	i 1er	2do
16	0	0
17	0	1
18	0	2
19	0	3
20	4	0
21	4	5
22	4	10
23	4	15
24	8	0
25	8	9
26	8	18
27	8	27
28	12	0
29	12	13
30	12	26
31	12	39

n	i 1er	2do
32	0	0
33	0	2
34	0	4
35	0	6
36	8	0
37	8	6
38	8	12
39	8	18
40	16	0
41	16	10
42	16	20
43	16	30
44	24	0
45	24	14
46	24	28
47	24	42

n	i 1er	2do
48	0	0
49	0	3
50	0	6
51	0	9
52	12	0
53	12	7
54	12	14
55	12	21
56	24	0
57	24	11
58	24	22
59	24	33
60	36	0
61	36	15
62	36	30
63	36	45

Tabla 5.9 Factores de giro ω^i para la IFFT de 64 puntos.

Para llevar a cabo el primer nivel de la transformada se alimentan las mariposas de raíz cuarta con las siguientes combinaciones de puntos y los resultados se escriben en la misma localidad de lectura, tanto la parte real como la imaginaria:

Número de combinación	X[0] x[0]	X[1] x[1]	X[2] x[2]	X[3] x[3]
1	0	16	32	48
2	1	17	33	49
3	2	18	34	50
4	3	19	35	51
5	4	20	36	52
6	5	21	37	53
7	6	22	38	54
8	7	23	39	55
9	8	24	40	56
10	9	25	41	57
11	10	26	42	58
12	11	27	43	59
13	12	28	44	60
14	13	29	45	61
15	14	30	46	62
16	15	31	47	63

Tabla 5.10 Combinaciones para el primer nivel de la IFFT de 64 puntos.

Para el segundo nivel se tienen las siguientes combinaciones:

Número de combinación	X[0] x[0]	X[1] x[1]	X[2] x[2]	X[3] x[3]
1	0	4	8	12
2	1	5	9	13
3	2	6	10	14
4	3	7	11	15
5	16	20	24	28
6	17	21	25	29
7	18	22	26	30
8	19	23	27	31
9	32	36	40	44
10	33	37	41	45
11	34	38	42	46
12	35	39	43	47
13	48	52	56	60
14	49	53	57	61
15	50	54	58	62
16	51	55	59	63

Tabla 5.11 Combinaciones para el segundo nivel de la IFFT de 64 puntos.

Para el tercer nivel se tiene que las salidas se escriben en una localidad diferente de la de entrada:

Número de combinación	X[0]	X[1]	X[2]	X[3]	x[0]	x[1]	x[2]	x[3]
1	0	1	2	3	0	16	32	48
2	4	5	6	7	4	20	36	52
3	8	9	10	11	8	24	40	56
4	12	13	14	15	12	28	44	60
5	16	17	18	19	1	17	33	49
6	20	21	22	23	5	21	37	53
7	24	25	26	27	9	25	41	57
8	28	29	30	31	13	29	45	61
9	32	33	34	35	2	18	34	50
10	36	37	38	39	6	22	38	54
11	40	41	42	43	10	26	42	58
12	44	45	46	47	14	30	46	62
13	48	49	50	51	3	19	35	51
14	52	53	54	55	7	23	39	55
15	56	57	58	59	11	27	43	59
16	60	61	62	63	15	31	47	63

Tabla 5.12 Combinaciones para el tercer nivel de la IFFT de 64 puntos.

5.3 *Interpolación*

Para poder montar las señales bandabase en una señal portadora se necesita implementar un algoritmo de interpolación con lo que se eleva la tasa de muestreo de T_s (64 muestras por símbolo) a una tasa de $2T_s$ (128 muestras por símbolo). Esto se lleva a cabo procesando las señales, tanto en fase como en cuadratura, con un filtro FIR cuyos coeficientes corresponden a una sinc.

Para interpolar la señal se insertan L-1 valores nulos entre cada muestra de la señal con tasa T_s. Posteriormente la señal resultante, con tasa de muestreo LT_s, será filtrada. Los parámetros utilizados para diseñar el filtro interpolador son los siguientes:

- **L** Factor de elevación de la frecuencia de muestreo
- **2N+1** Número de coeficientes del filtro

Se debe señalar que mientras aumente el factor L se tendrá un comportamiento más exacto de la señal. Además si aumenta el número de muestras nulas (L-1), el número de operaciones innecesarias es mayor (multiplicaciones cuyo resultado es cero y sumas innecesarias) por lo que, implementando el filtro con un algoritmo eficiente su operación es más rápida. Por otro lado, al aumentar el factor de elevación de frecuencia de muestreo se tiene un mayor retardo ya que el número de coeficientes del filtro debe aumentar.

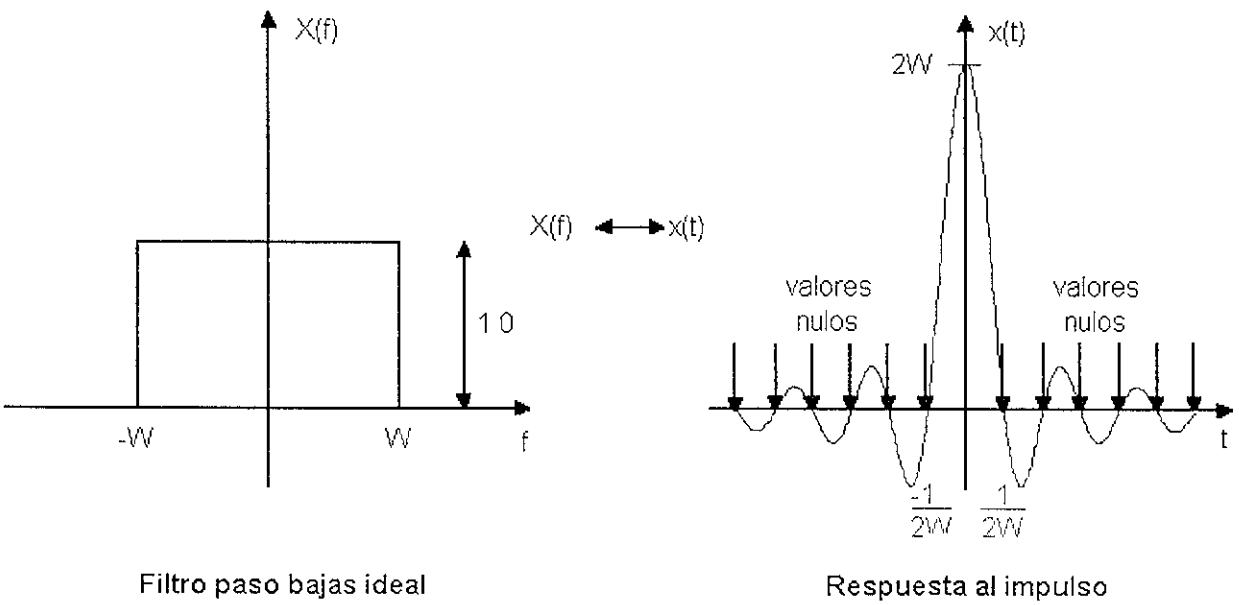


Figura 5.5 Filtro paso bajas ideal y su respuesta al impulso.

Para obtener los coeficientes del filtro interpolador se utiliza la función Sinc que es la respuesta al impulso de un filtro pasabajas ideal (**figura 5.5**). La función utilizada para generar los coeficientes es:

$$x[n] = \frac{\operatorname{sen}\left(\pi \frac{n-N}{L}\right)}{\left(\pi \frac{n-N}{L}\right)}\Bigg|_{0}^{2N} \tag{5.7}$$

Con el fin de obtener una respuesta en frecuencia del filtro con los lóbulos laterales más atenuados, los coeficientes pueden multiplicarse por una ventana de Blackman, Hamming, Hanning o Kaiser con el mismo número de coeficientes. En el caso de las simulaciones reportadas se multiplicaron los coeficientes por una ventana de Blackman y se obtuvo una reducción de 20 dB, aproximadamente, en los lóbulos laterales de la respuesta en frecuencia. Los coeficientes del filtro utilizados en las simulaciones se muestra en la **figura 5.6** y la respuesta en frecuencia se muestra en la **figura 5.7**.

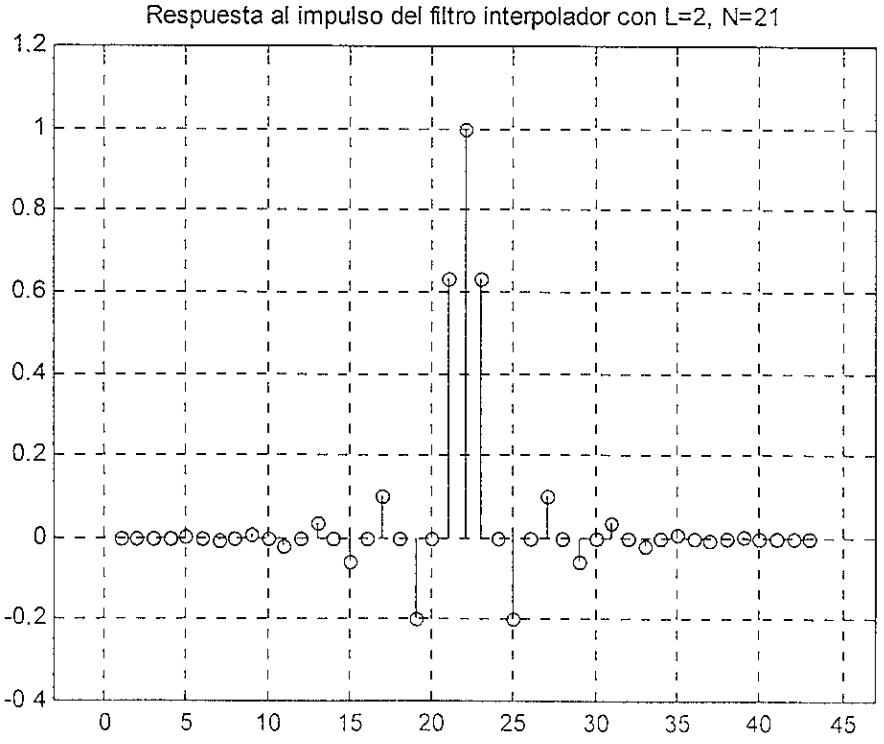


Figura 5.6 Coeficientes del filtro interpolador.

Gianfranco Biliotti Rodriguez
Hiram Eduardo Galicia Luna

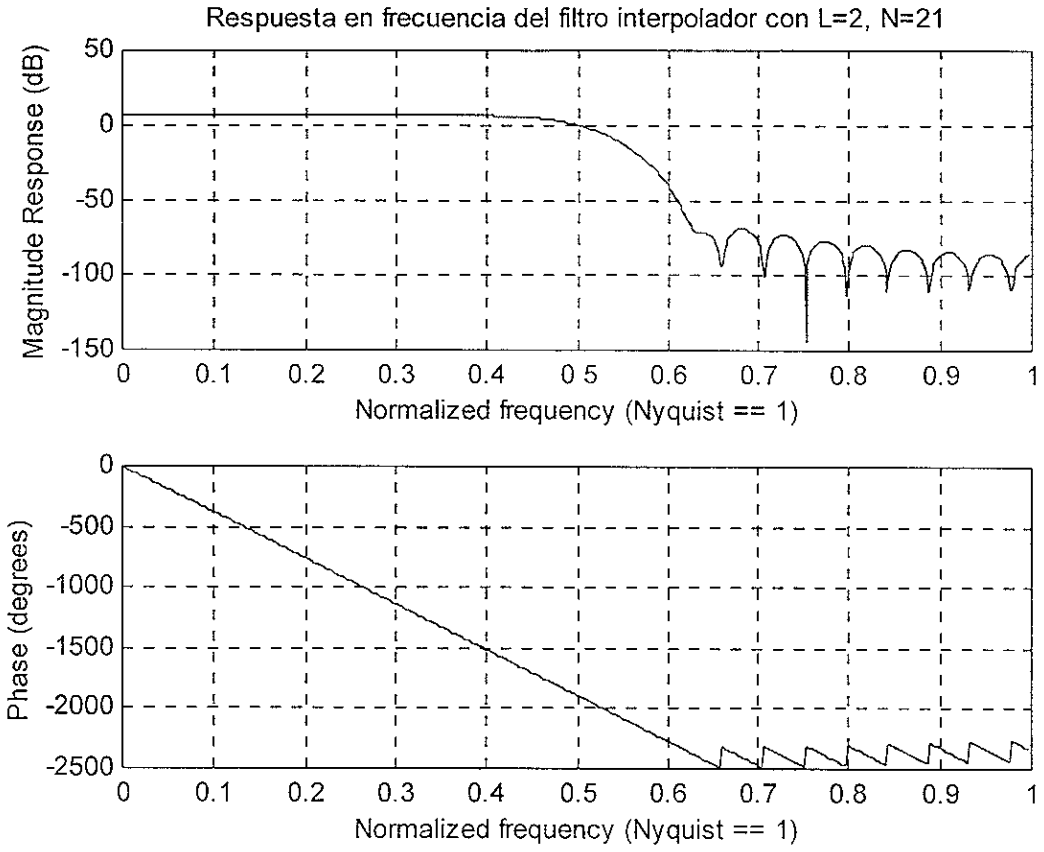
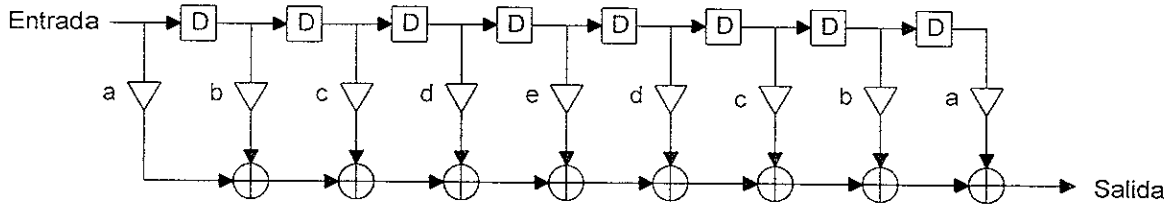


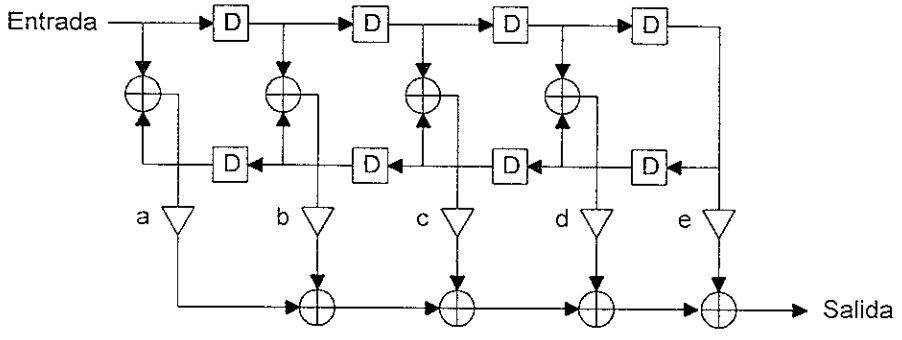
Figura 5.7 Respuesta en frecuencia del filtro interpolador.

Para poder optimizar el algoritmo de interpolación y tener un menor tiempo de cálculo se deben tomar en cuenta los siguientes puntos:

1. Las muestras de valor nulo que se insertan entre cada muestra de la señal original representan multiplicaciones cuyo valor se sabe de antemano (cero).
2. Algunos coeficientes del filtro interpolador son nulos (figura 5.5 y 5.6) por lo que las multiplicaciones que se hagan con éstos serán innecesarias.
3. Para que la respuesta angular del filtro sea lineal se necesita que los coeficientes sean simétricos, por lo tanto, se llevarán a cabo dos multiplicaciones por el mismo coeficiente que se sumaran eventualmente. Para mejorar el algoritmo se preferiría sumar los dos valores que se multiplican por el mismo coeficiente y luego llevar a cabo la multiplicación dado que la multiplicación requiere una mayor complejidad que una suma (**figura 5.8**).



Filtro sin optimización



Filtro optimizado

Figura 5.8 Optimización del filtro interpolador con coeficientes simétricos.

5.4 *Codificación e Intercalamiento de bits*

La codificación utilizada es la que se recomienda en la especificación IEEE 802.11a. Los polinomios generadores son $(133)_8$ y $(171)_8$. El algoritmo de decodificación es el de Viterbi con decisión dura. Cabe señalar que se puede obtener un mejor resultado si se utiliza un algoritmo con decisión suave de 3 bits, pero tanto su implementación como su explicación requieren de un trato más detallado y por lo tanto queda fuera del alcance de esta tesis. Esta mejoría consiste generalmente en una ganancia de 2 o 3 dB con respecto de la curva de decodificación con decisión dura.

El algoritmo de intercalamiento se define por dos permutaciones. La primera permutación asegura que los bits codificados adyacentes se colocan en subportadoras no adyacentes. La segunda permutación asegura que los bits codificados se colocan alternadamente en los bits mas y menos significativos de la constelación y, por lo tanto, se evita tener grandes bloques de confiabilidad baja.

Se denota k como el índice del bit codificado antes de la primera permutación; i como el índice después de la primera permutación y antes de la segunda permutación; j es el índice después de la segunda permutación, justo antes de pasar al mapeo en la constelación.

La primera permutación se define por la regla:

$$i = (N_{CBPS}/16)(k \bmod 16) + \text{Entero}(k/16),\ k = 0, 1,..., N_{CBPS}-1$$

La función Entero(*) denota el entero mas grande que no excede al parámetro * y N_{CBPS} es el número de bits codificados por símbolo. La segunda permutación se define por la regla:

$$j = s \times \text{Entero}(i/s) + (i+N_{CBPS} - \text{Entero}(16 \times i/N_{CBPS})) \bmod s,\ i = 0, 1,..., N_{CBPS}-1$$

El valor de s es determinado por el número de bits codificados por subportadora, N_{BPSC}, de acuerdo a:

$$s = \max(N_{BPSC}/2,1)$$

El desintercalador, que lleva a cabo la operación inversa, también está definido por dos permutaciones.

La primera permutación se define por la regla:

$$i = s \times \text{Entero}(j/s) + (j + \text{Entero}(16 \times j/N_{CBPS})) \bmod s, j = 0, 1,..., N_{CBPS}-1$$

La segunda permutación se define por la regla:

$$k = 16 \times i - (N_{CBPS}-1)\ \text{Entero}(16 \times i/N_{CBPS}),\ i = 0, 1, . , N_{CBPS}-1$$

5.5 *Sincronización*

Para sincronizar la fase y la frecuencia de la portadora, así como la temporización de los símbolos se utilizó el algoritmo de correlación con símbolos de entrenamiento conocidos. Se utilizan dos tipos de preámbulo de entrenamiento, uno con símbolos de entrenamiento cortos y otro con símbolos de entrenamiento largos. Con el preámbulo de símbolos cortos se sincroniza fase y frecuencia de la portadora y con el preámbulo de símbolos largos se lleva a cabo la temporización de los símbolos y se estiman los factores de corrección de amplitud y de fase de las constelaciones. Un símbolo de entrenamiento corto consiste de 12 subportadoras moduladas por los siguientes valores

Número de punto	Frec.	Valor
0	0	0
1	1	0
2	2	0
3	3	0
4	4	-1-j
5	5	0
6	6	0
7	7	0
8	8	-1-j
9	9	0
10	10	0
11	11	0
12	12	1+j
13	13	0
14	14	0
15	15	0

Número de punto	Frec.	Valor
16	16	1+j
17	17	0
18	18	0
18	18	0
20	20	1+j
21	21	0
22	22	0
23	23	0
24	24	1+j
25	25	0
26	26	0
27	27	0
28	28	0
29	29	0
30	30	0
31	31	0

Número de punto	Frec.	Valor
32	32	0
33	-31	0
34	-30	0
35	-29	0
36	-28	0
37	-27	0
38	-26	0
39	-25	0
40	-24	1+j
41	-23	0
42	-22	0
43	-21	0
44	-20	-1-j
45	-19	0
46	-18	0
47	-17	0

Número de punto	Frec.	Valor
48	-16	1+j
49	-15	0
50	-14	0
51	-13	0
52	-12	-1-j
53	-11	0
54	-10	0
55	-9	0
56	-8	-1-j
57	-7	0
58	-6	0
59	-5	0
60	-4	1+j
61	-3	0
62	-2	0
63	-1	0

Tabla 5.13 Datos modulados para obtener el símbolo de entrenamiento corto.

La salida de la IFFT debe ser multiplicada por un factor de $(13/6)^{1/2}$ para normalizar la potencia promedio del símbolo resultante el cual utiliza 12 subportadoras en lugar de 52. Para obtener un símbolo de entrenamiento largo se modulan las subportadoras con los siguientes valores:

Número de punto	Frec.	Valor
0	0	0
1	1	1
2	2	-1
3	3	-1
4	4	1
5	5	1
6	6	-1
7	7	1
8	8	-1
9	9	1
10	10	-1
11	11	-1
12	12	-1
13	13	-1
14	14	-1
15	15	1

Número de punto	Frec.	Valor
16	16	1
17	17	-1
18	18	-1
18	18	1
20	20	-1
21	21	1
22	22	-1
23	23	1
24	24	1
25	25	1
26	26	1
27	27	0
28	28	0
29	29	0
30	30	0
31	31	0

Número de punto	Frec.	Valor
32	32	0
33	-31	0
34	-30	0
35	-29	0
36	-28	0
37	-27	0
38	-26	1
39	-25	1
40	-24	-1
41	-23	-1
42	-22	1
43	-21	1
44	-20	-1
45	-19	1
46	-18	-1
47	-17	1

Número de punto	Frec.	Valor
48	-16	1
49	-15	1
50	-14	1
51	-13	1
52	-12	1
53	-11	-1
54	-10	-1
55	-9	1
56	-8	1
57	-7	-1
58	-6	1
59	-5	-1
60	-4	1
61	-3	1
62	-2	1
63	-1	1

Tabla 5.14 Datos modulados para obtener el símbolo de entrenamiento largo.

La correlación del preámbulo con símbolos de entrenamiento cortos entrega como resultado un flujo de datos complejo, es decir, con parte real y parte imaginaria, el cual se utiliza para estimar la fase de la portadora. La fase se puede estimar si se calcula la fase de los picos de correlación por medio de la siguiente ecuación:

$$\Delta\phi = \tan^{-1}(-A_R/-A_I) \tag{5.8}$$

donde A_R es el valor de la amplitud de la parte real del pico y A_I es el valor de la parte imaginaria.

En la **figura 5.9** se muestra un ejemplo de la salida del filtro correlacionador para el preámbulo con símbolos cortos. El hecho de que solamente las componentes espectrales que son múltiplos de 4 tienen amplitud no nula (tabla 5.13) tiene como consecuencia que dentro del símbolo se repita una señal de 16 muestras 4 veces en el dominio del tiempo.

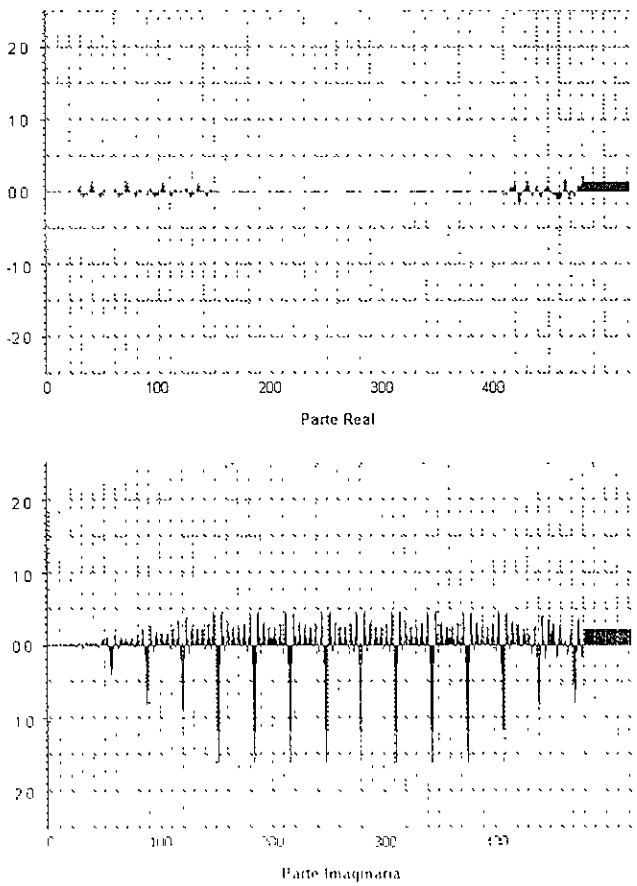


Figura 5.9 Salida del correlacionador para el preámbulo corto sin ruido, sin multitrayectorias.

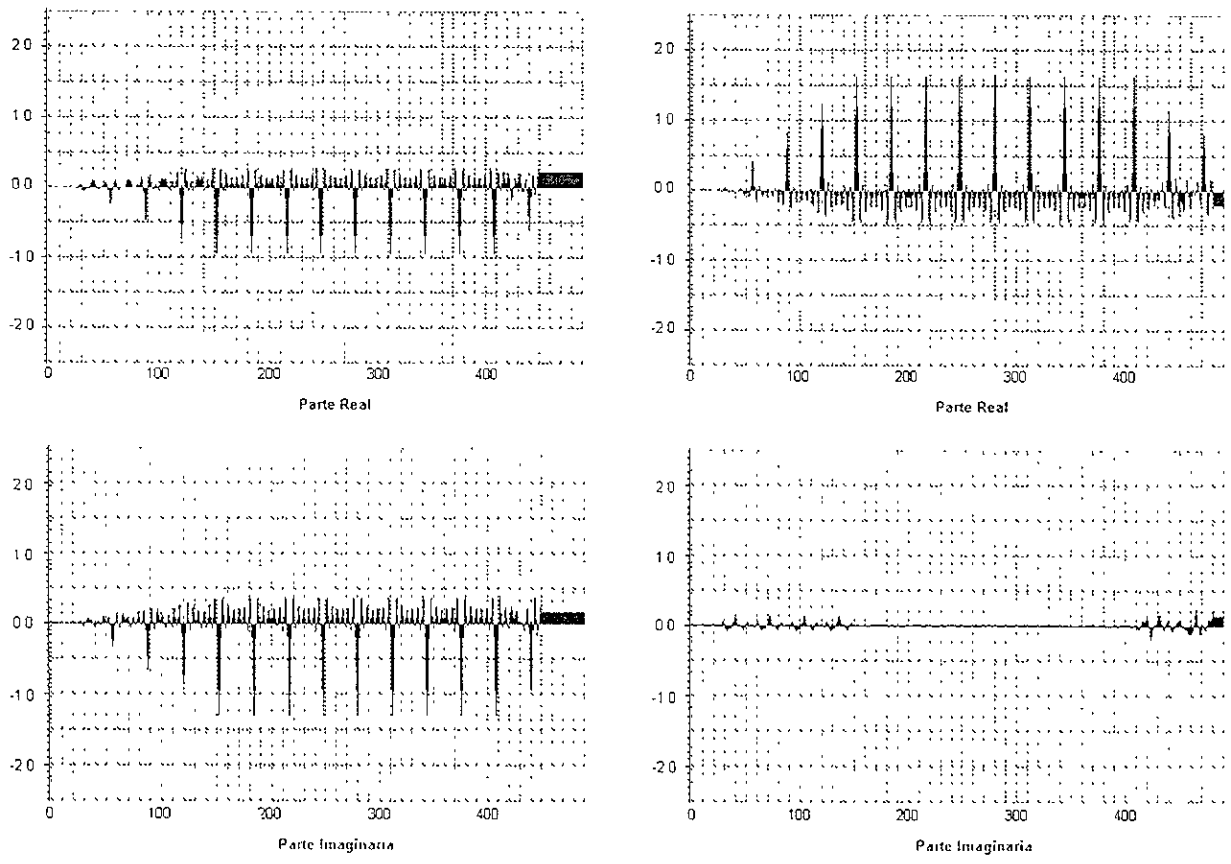
Con las amplitudes de los picos de correlación de la figura anterior se puede calcular la desviación de fase de la portadora, donde A_R tiene un valor de 0 y A_I tiene un valor de -1.6, aproximadamente. Por lo tanto se puede ver que la desviación de fase es $\tan^{-1}(0/1.6) = 0°$.

En la **figura 5.10 (a)** y **(b)** se muestran dos ejemplos de salida del correlacionador cuando la desviación de fase de la portadora es igual a 36° y -90°, respectivamente. Si se calculan estos valores por medio de la ecuación (5.8) se tiene que:

$$\Delta\phi_{(a)} = \tan^{-1}(0.9/1.25) = 35.75° \approx 36°$$

$$\Delta\phi_{(b)} = \tan^{-1}(-1.6/0) = -90°$$

Estos valores corresponden con los que se programaron para simular la desviación de fase de la portadora.

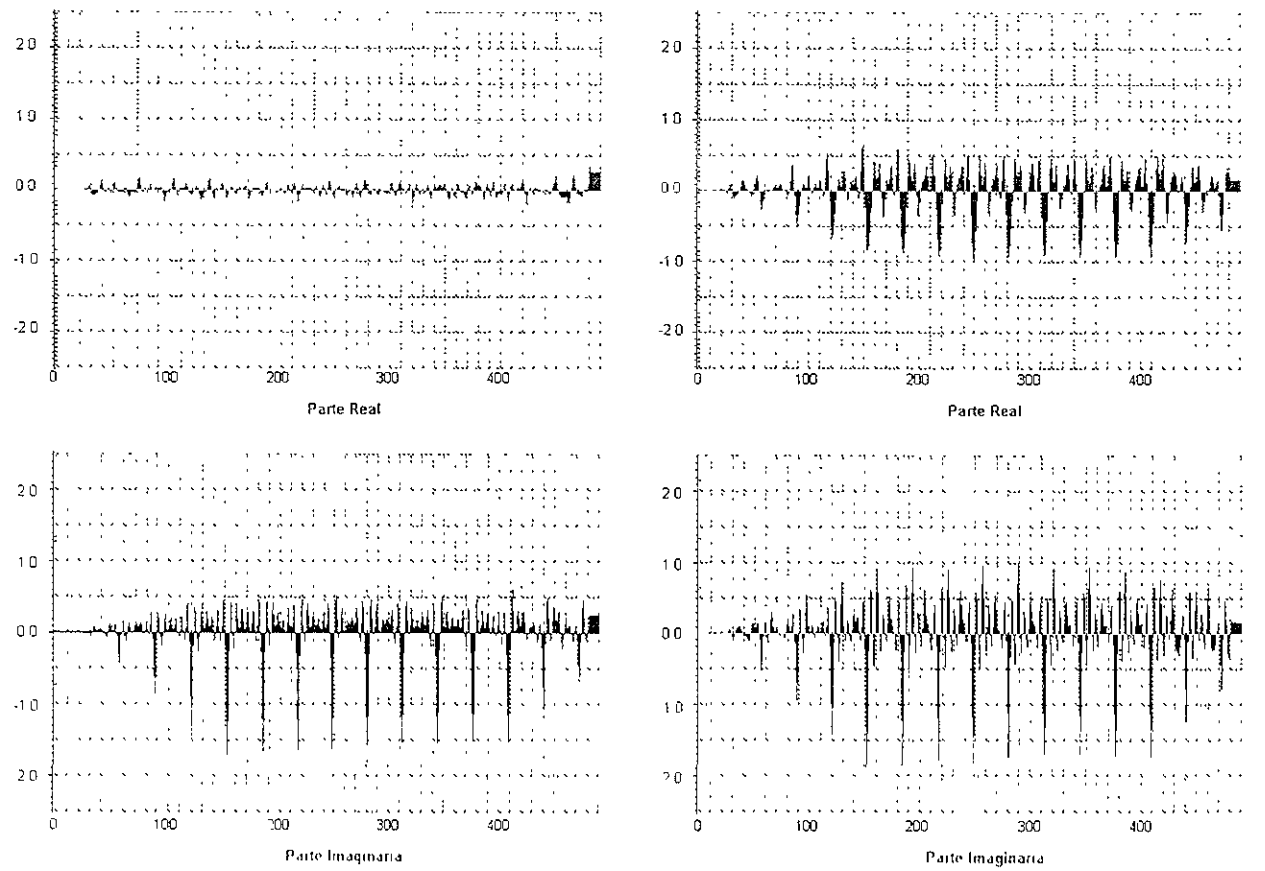


(a) (b)

Figura 5.10 Salida del correlacionador para el preámbulo con símbolos cortos sin ruido, sin multitrayectorias y con desviación de fase de la portadora de: (a) 36°, (b) -90°.

La **figura 5.11 (a)** y **(b)** muestra la respuesta del correlacionador cuando la transmisión se lleva a cabo en un canal sin multitrayectorias y con multitrayectorias, respectivamente. En ambos casos se encuentra en el canal la presencia de ruido con distribución gaussiana cuya relación E_b/N_0 es igual a 3 dB. A pesar de que la relación E_b/N_0 es pequeña, generalmente se opera con tasas de BER = 10^{-6} cuya relación E_b/N_0 es mayor que 11 dB, se puede ver que los picos de correlación tienen una magnitud de más de 6 dB con respecto a los lóbulos laterales.

Para el caso de la salida en el canal con multitrayectorias (figura 5.11 b) se puede ver que la magnitud de los picos aumenta ya que la señal tiene más potencia. Esto se debe a la contribución de potencia de las reflexiones que se suman a la potencia de la señal original. También se puede ver que la magnitud de los picos de correlación no es constante, como en el caso de las figuras 5.9 y 5.10. Esto se debe a la presencia de ruido en el canal.



(a) (b)

Figura 5.11 Salida del correlacionador para el preámbulo de símbolos cortos con relación Eb/No de 3 dB en la transmisión: a) sin multitrayectorias, b) con multitrayectorias

La salida del correlacionador con el preámbulo de símbolos largos en condiciones ideales se muestra en la **figura 5.12**. Se puede ver que los picos de correlación son 6 dB mayores a los lóbulos laterales. También se observa el fenómeno de correlación aperiódica mencionado en la sección 4.5 debido a que se lleva a cabo la correlación entre el preámbulo con símbolos cortos y el conjugado del preámbulo con símbolos largos. La separación entre cada pico de correlación sirve para estimar la temporización del símbolo y poder establecer la ventana en la que se realizará la FFT. Por ejemplo entre el primer pico y el segundo se tienen 128 muestras que es la duración del símbolo.

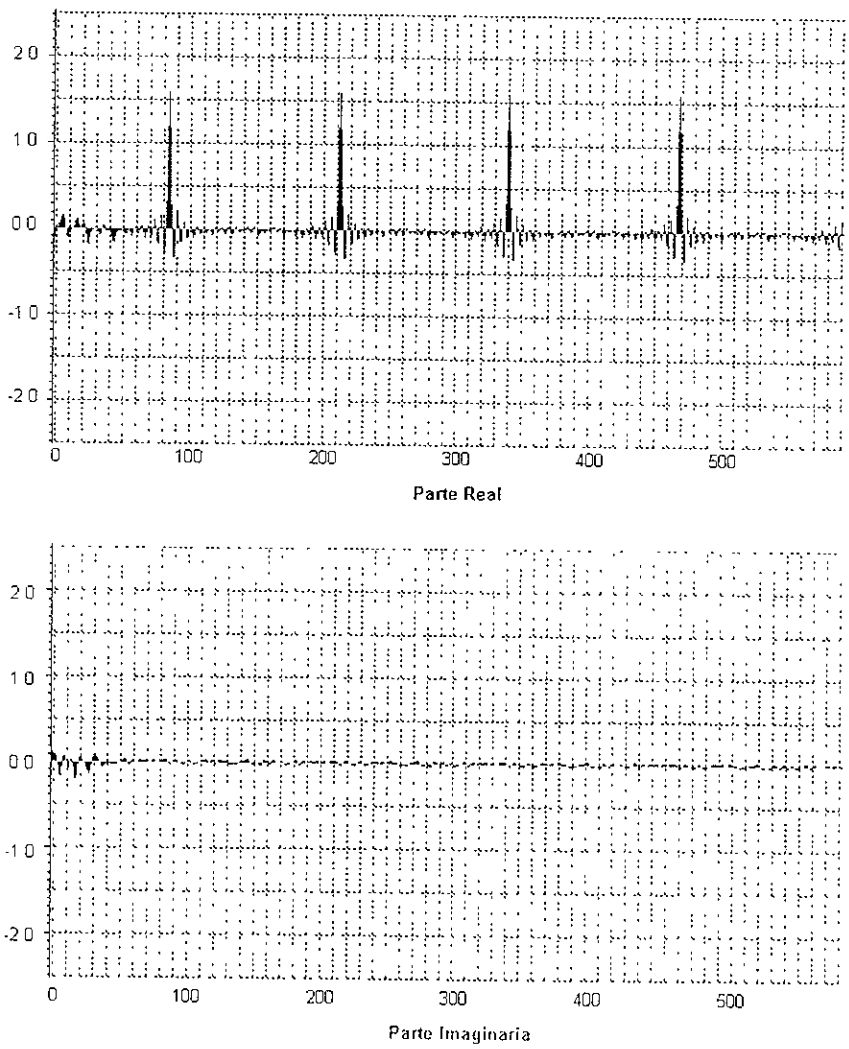


Figura 5.12 Salida del correlacionador para el preámbulo con símbolos largos sin ruido, sin multitrayectorias y sin desviación de fase de la portadora.

Las **figuras 5.13** y **5.14** muestran la salida del correlacionador en un canal sin multitrayectorias y con multitrayectorias, respectivamente, con una relación E_b/N_0 igual a 3 dB. También en estas figuras se puede ver que los picos de correlación tienen una magnitud de más de 6 dB con respecto a los lóbulos laterales, a pesar de tener una relación E_b/N_0 pequeña.

La magnitud de los picos de correlación no es constante debido a la presencia de ruido en el canal.

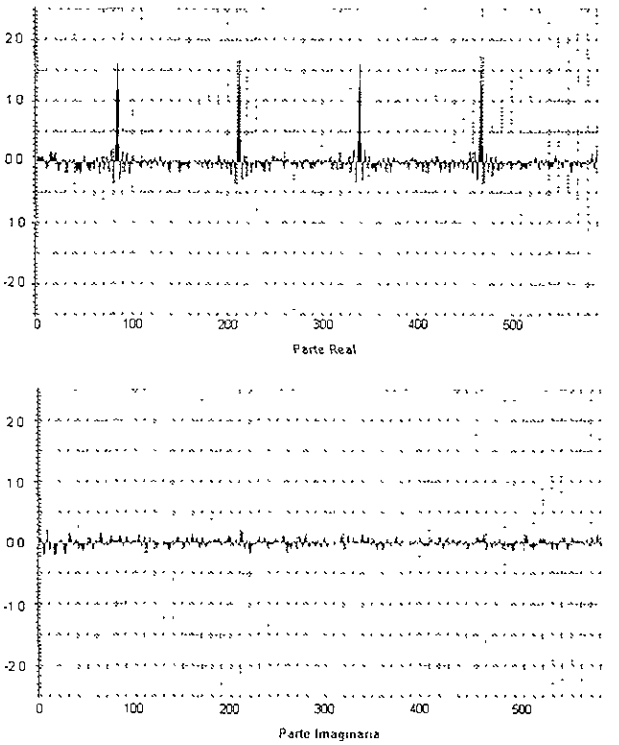


Figura 5.13 Salida del correlacionador para el preámbulo con símbolos largos con relación Eb/No de 3 dB, sin desviación de fase de la portadora y sin multitrayectorias.

La segunda función del preámbulo largo se puede observar en las figuras 5.15, 5.16 y 5.17. En la **figura 5.15** se muestra (de izquierda a derecha y de arriba hacia abajo): la constelación transmitida de una subportadora, la constelación recibida con ruido gaussiano solamente (E_b/N_0 = 10 dB), la constelación corregida con ayuda del preámbulo largo y las magnitudes de las subportadoras del preámbulo largo. Se puede ver que las magnitudes de las subportadoras son similares ya que en este caso no se simulan multitrayectorias en la transmisión.

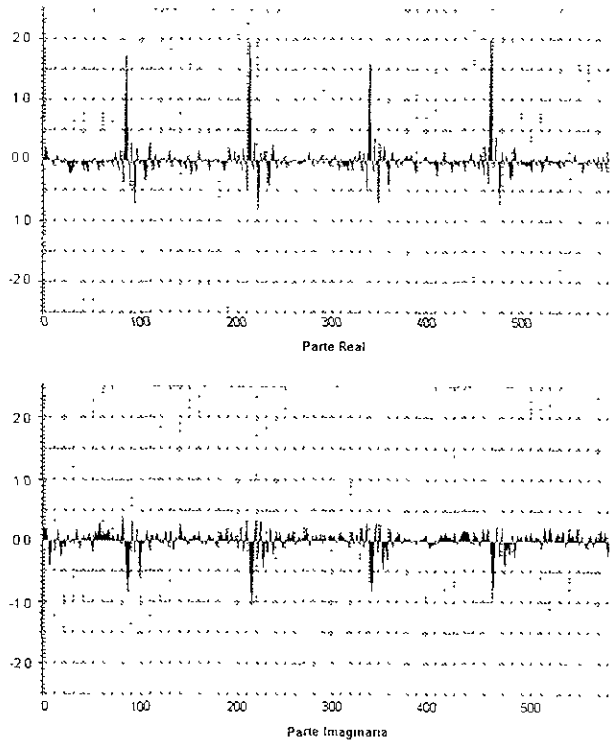


Figura 5.14 Salida del correlacionador para el preámbulo con símbolos largos con relación Eb/No de 3 dB, sin desviación de fase de la portadora y con multitrayectorias.

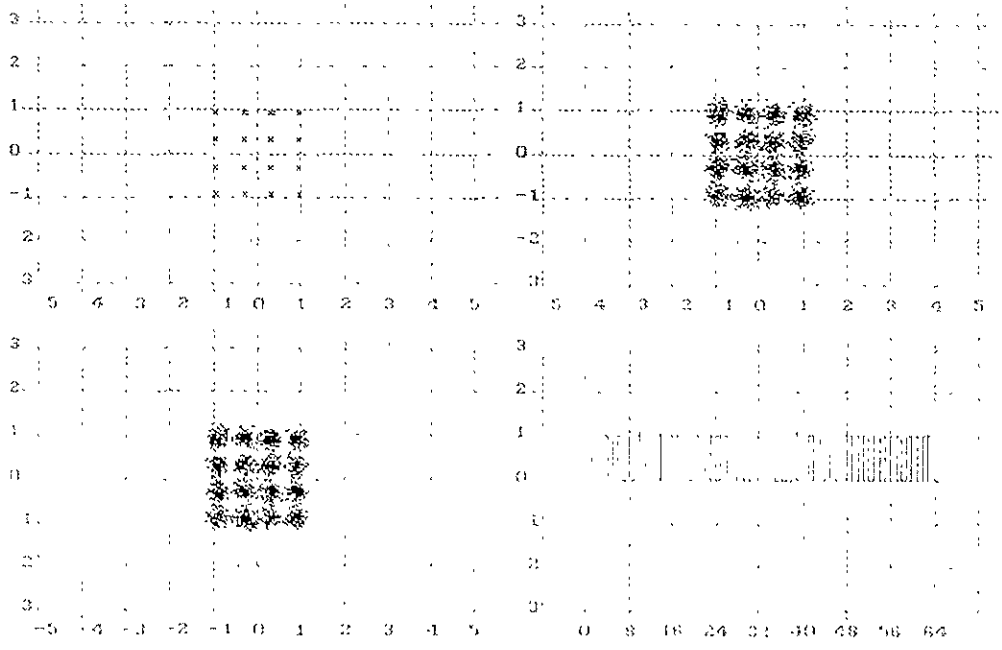


Figura 5.15 Constelaciones16-QAM de una subportadora para una transmisión con relación Eb/No de 10 dB, sin multitrayectorias.

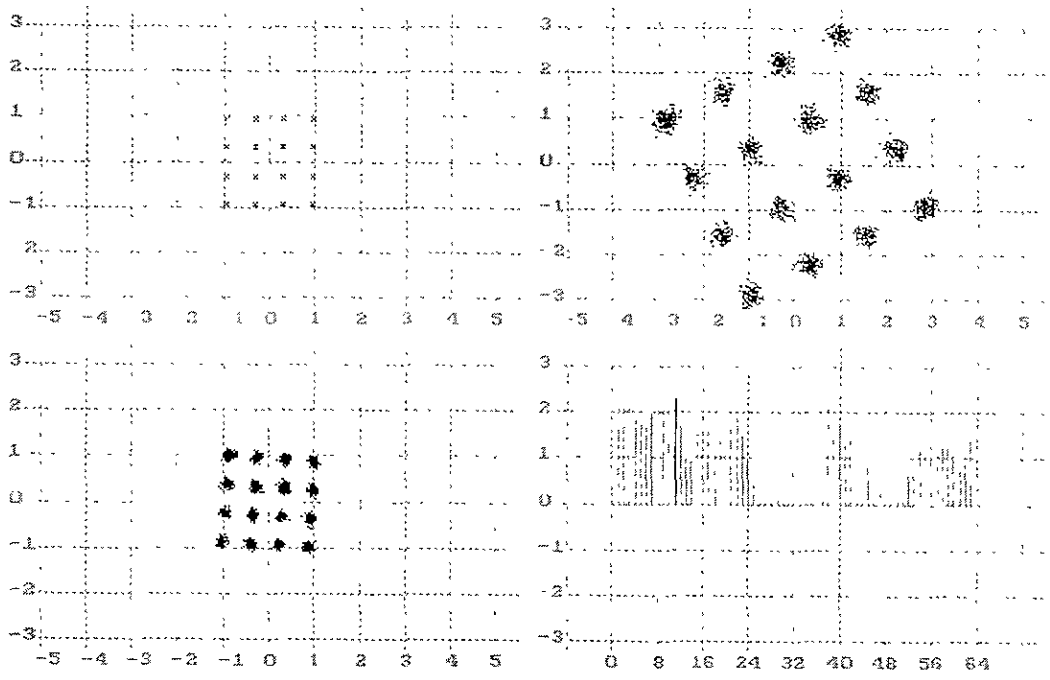


Figura 5.16 Constelaciones 16-QAM de una subportadora beneficiada por la multitrayectoria con relación Eb/No de 10 dB.

En la **figura 5.16** se muestran las constelaciones de una subportadora para una transmisión con multitrayectorias. La constelación recibida tiene una mejor relación E_b/N_0 que la de la figura 5.15, sin embargo está girada y tiene una amplitud diferente a la de la constelación transmitida. La constelación recibida se corrige tanto en amplitud como en fase por un factor complejo calculado con ayuda del preámbulo largo para poder llevar a cabo el mapeo entre el punto recibido y los bits que representa ese punto. Se puede ver que los puntos tienen menos probabilidad de error que en la figura 5.15. En este caso la subportadora ha sido beneficiada por las multitrayectorias. Sin embargo existen otras subportadoras que son atenuadas. Un ejemplo de estas subportadoras se muestra en la **figura 5.17** donde la constelación recibida tiene un error mayor que la constelación recibida de la figura 5.15. Estas figuras ayudan a entender la necesidad de un algoritmo intercalador en la transmisión ya que, como se puede observar, existen regiones donde hay mas atenuación y regiones donde las subportadoras son amplificadas. De esta forma con el algoritmo intercalador se puede distribuir el error de las regiones con baja confiabilidad.

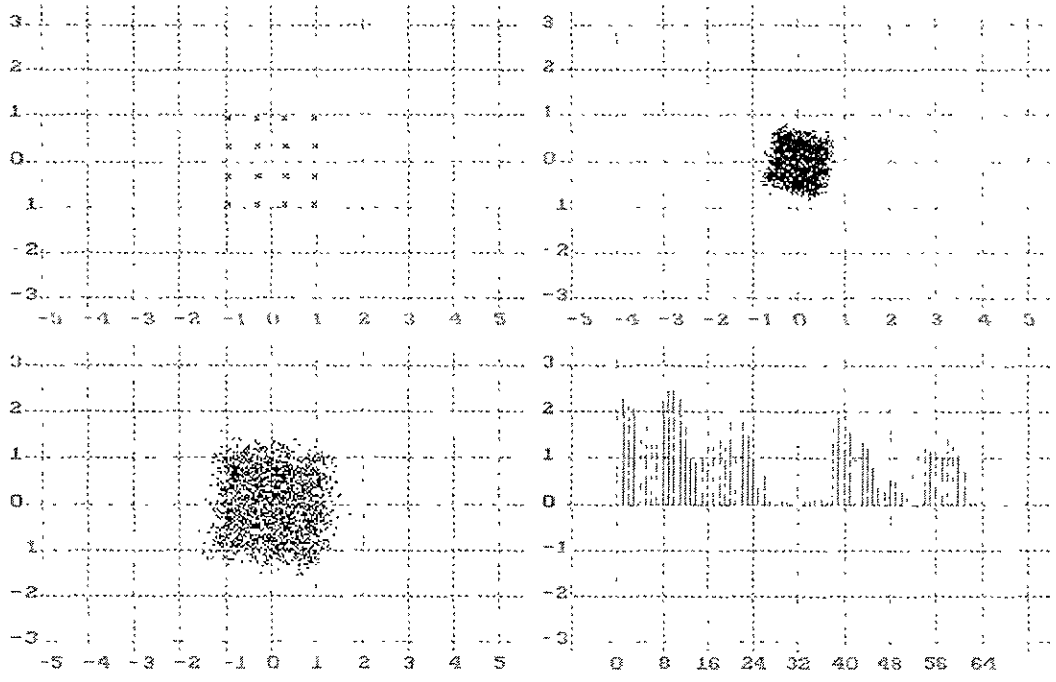


Figura 5.17 Constelaciones 16-QAM de una subportadora atenuada por la multitrayectoria con relación Eb/No de 10 dB.

5.6 *Resultados*

Las siguientes gráficas muestran los resultados obtenidos en las simulaciones. Se tienen 7 gráficas para cada una de las constelaciones utilizadas para modular las subportadoras: BPSK, QPKS y 16-QAM.

El orden de las gráficas es el siguiente (de izquierda a derecha y de arriba hacia abajo):

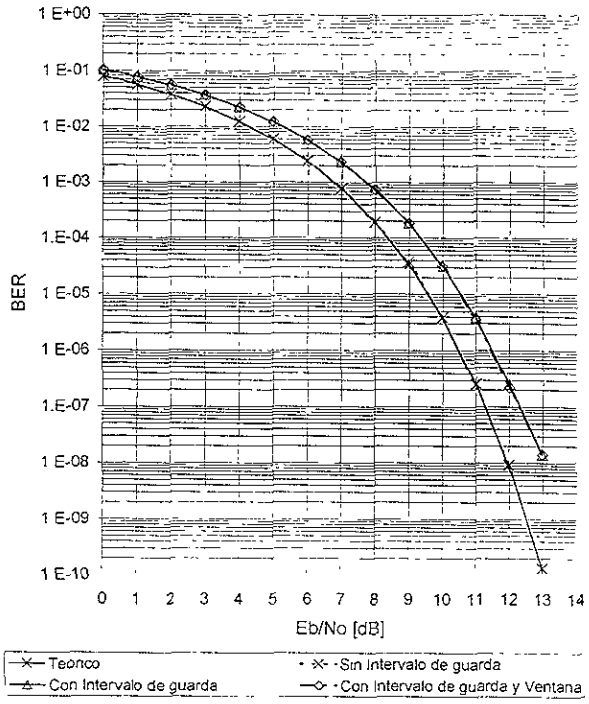
1. *Resultados con sincronización perfecta.* Estos resultados se obtienen cuando en la simulación solo se evalúa el efecto de ruido en el sistema. No se implementa ningún algoritmo de sincronización porque se conoce el intervalo exacto para realizar la FFT en cada uno de los símbolos recibidos. En estas gráficas se puede observar la degradación de 1 dB con respecto a la curva teórica debido a la inserción del intervalo de guarda. También se puede ver que no importa si se utiliza o no una ventana de coseno elevado para reducir la emisión en bandas espurias ya que el desempeño es el mismo con y sin ventana.
2. *Resultados con símbolos cortos variables y 10 símbolos largos.* Estas gráficas demuestran que con 3 símbolos cortos se obtiene una degradación despreciable con respecto a la curva

teórica con intervalo de guarda, por lo que resultan suficientes 3 símbolos cortos en la primer parte del preámbulo de sincronización.

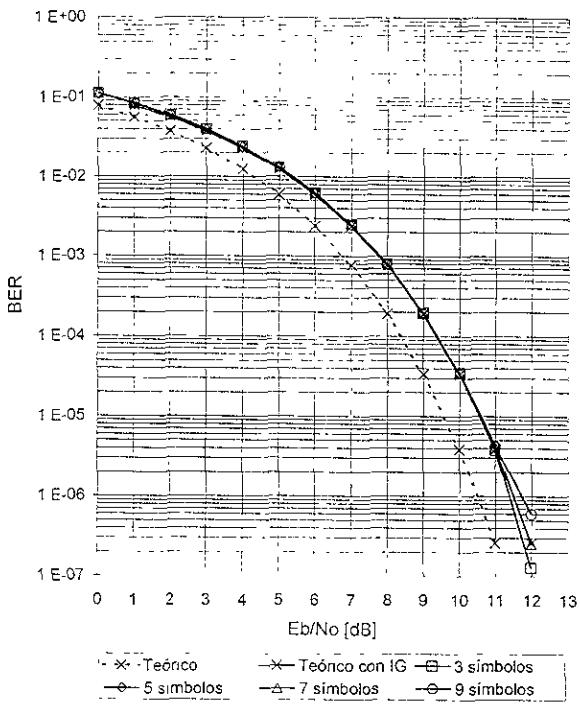
3. *Resultados con 10 símbolos cortos y símbolos largos variables.* Se puede ver que la degradación con 7 símbolos es menor que 0.5 dB y que con menos símbolos de entrenamiento se obtienen degradaciones de hasta 1.5 dB, por lo tanto 7 es el número de símbolos de entrenamiento largos que se utilizan para obtener los resultados de las gráficas subsecuentes.
4. *Resultados con intercalamiento y codificación.* Estos resultados muestran que al incluir intercalamiento en la transmisión de los datos no se sufre ninguna degradación, pero tampoco se tiene alguna ganancia. También se puede ver que la codificación proporciona una ganancia aproximada de 3 dB para una tasa de errores de 10^{-6}. En estos resultados no se simulan multitrayectorias en el canal.
5. *Resultados sin codificación y sin intercalamiento para varios perfiles.* Los resultados demuestran que con la presencia de multitrayectorias en el canal y sin codificación ni intercalamiento en la transmisión el desempeño del sistema es en general malo ya que las menores tasas de error a 17 dB de relación Eb/No están alrededor de 10^{-3} debido a que algunas de las subportadoras son atenuadas excesivamente.
6. *Resultados con codificación y sin intercalamiento para varios perfiles.* Estos resultados demuestran que la codificación ayuda a recuperar algunos decibeles que se pierden debido a la presencia de multitrayectorias en el canal. Sin embargo no se tiene en ningún caso algún resultado semejante al de la curva teórica con intervalo de guarda que es lo que se desearía.
7. *Resultados con codificación y con intercalamiento para varios perfiles.* Estos son los resultados más importantes que reportamos ya que en ellos se considera tanto la codificación como el intercalamiento. Como se puede ver el intercalamiento ayuda a la codificación a tener un mejor desempeño ya que se distribuye en todo el símbolo los errores ocurridos en las regiones de mayor atenuación. Los resultados muestran que se puede tener un desempeño semejante al de la curva teórica, aún cuando existen multitrayectorias en el canal.

Para los tres tipos de constelaciones se puede ver que el desempeño con el perfil 6 no es aceptable. A pesar de que se utiliza codificación e intercalamiento el desempeño para ese perfil nunca mejora. Esto, como se mencionó, se debe a que el retardo del perfil 6 tiene mas del 15% de exceso con respecto al intervalo de guarda lo que ocasiona ISI e ICI en el demodulador

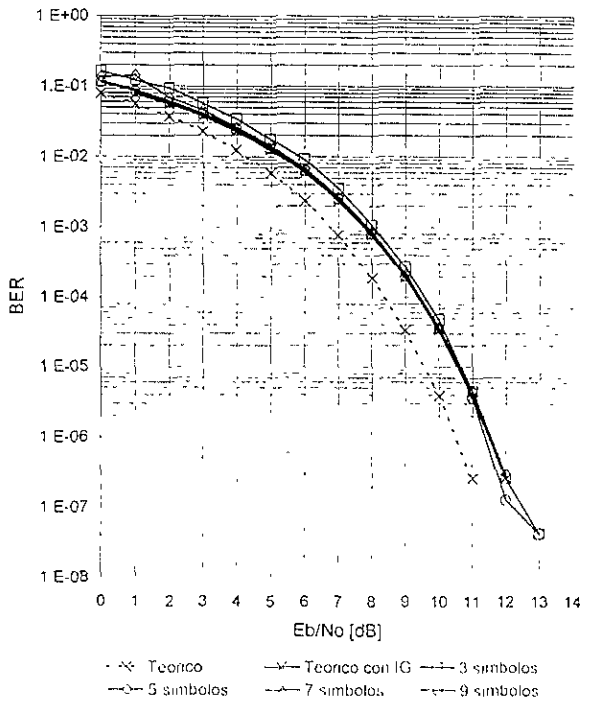
Resultados OFDM-BPSK con Sincronización Perfecta



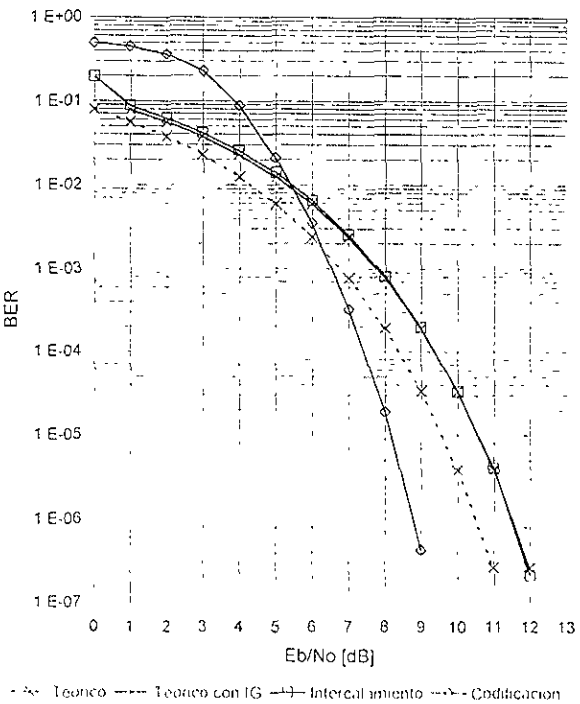
Resultados OFDM-BPSK, símbolos cortos variables, 10 símbolos largos



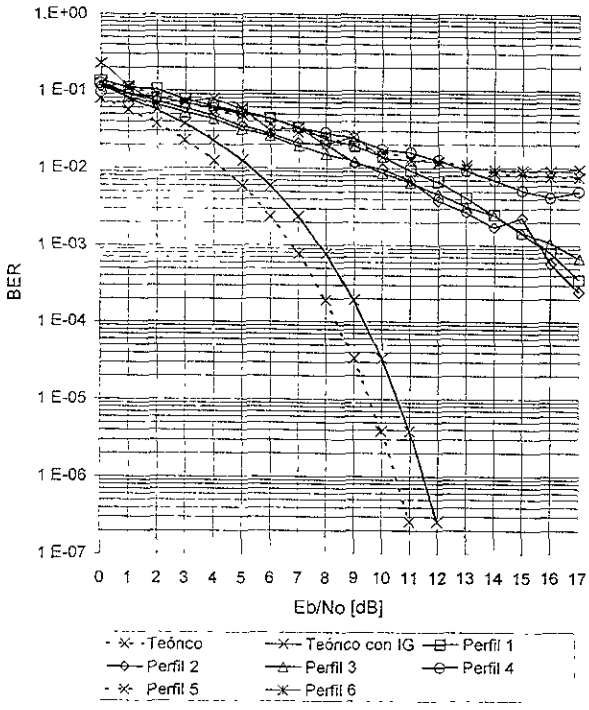
Resultados OFDM-BPSK, 10 símbolos cortos, símbolos largos variables



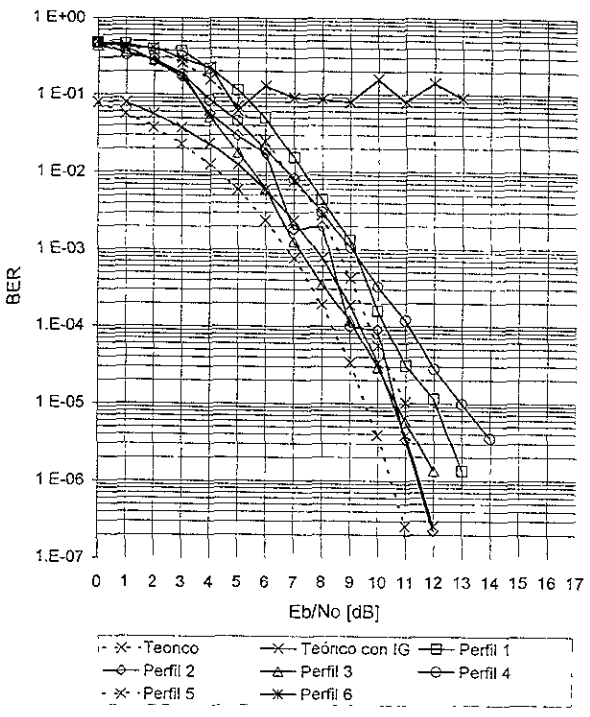
Resultados OFDM-BPSK con Intercalamiento y Codificación



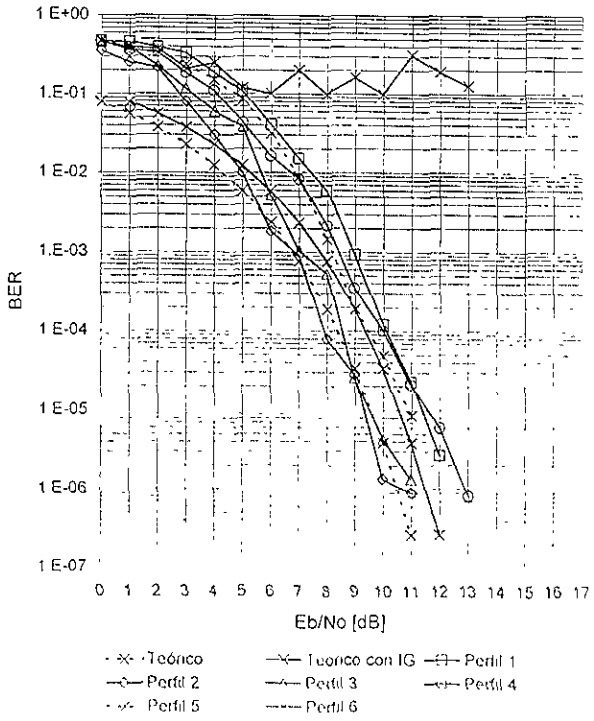
Gráfica OFDM-BPSK, Sin Codificación,
Sin Intercalamiento para varios perfiles



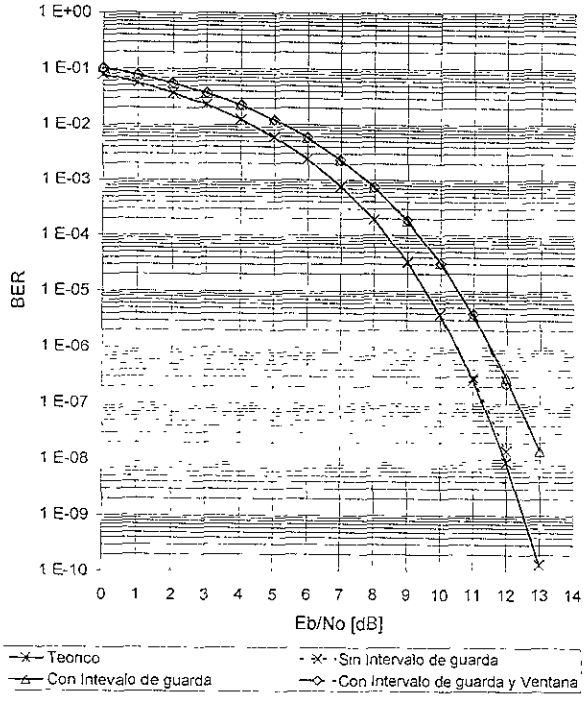
Gráfica OFDM-BPSK, Con Codificación,
Sin Intercalamiento para varios perfiles



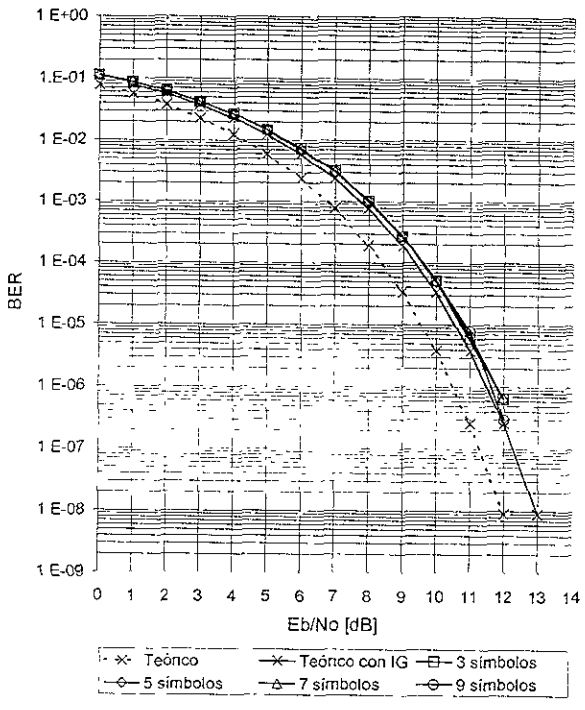
Gráfica OFDM-BPSK, Con Codificación,
Con Intercalamiento para varios perfiles



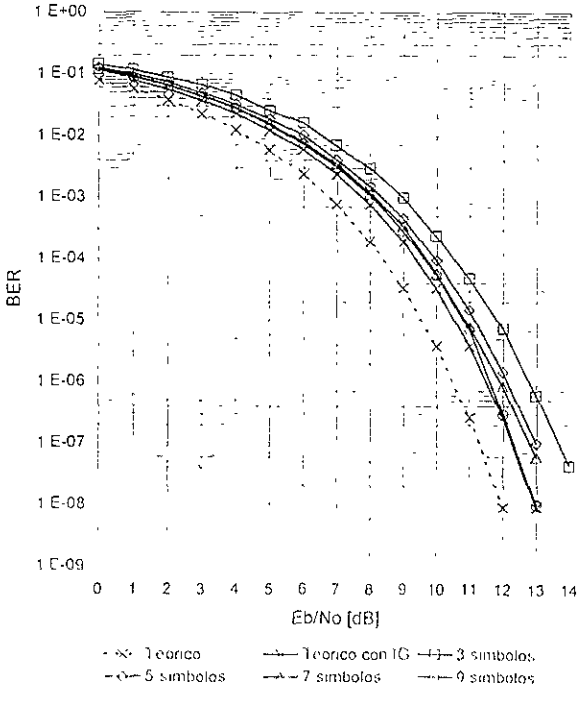
Resultados OFDM-QPSK con Sincronización Perfecta



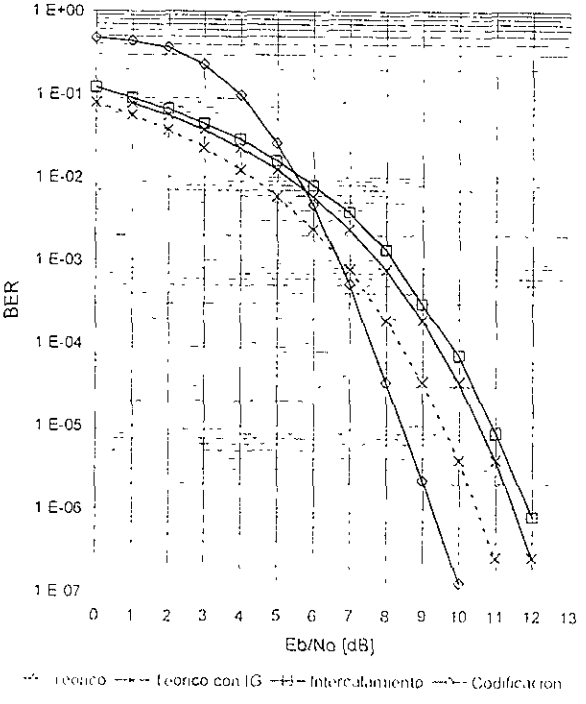
Resultados OFDM-QPSK, símbolos cortos variables, 10 símbolos largos



Resultados OFDM-QPSK, 10 símbolos cortos, símbolos largos variables

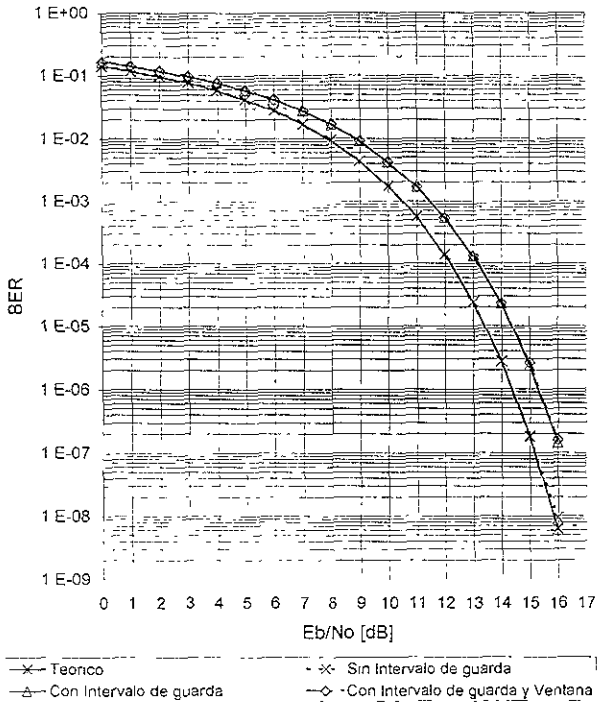


Resultados OFDM-QPSK con Intercalamiento y Codificación

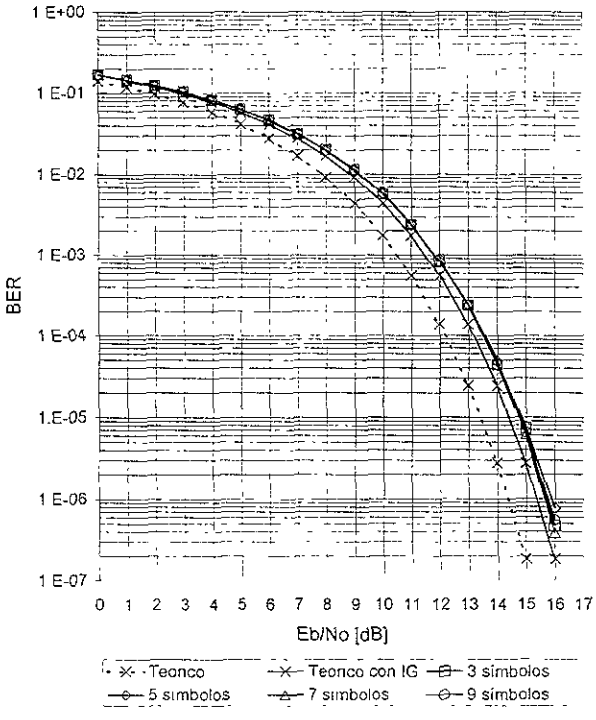


Gianfranco Biliotti Rodriguez
Hiram Eduardo Galicia Luna

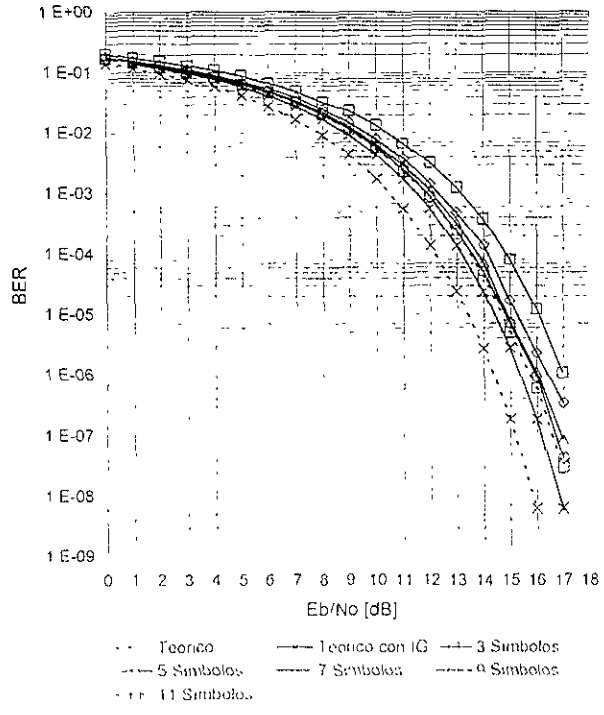
Resultados OFDM-16QAM con Sincronización Perfecta



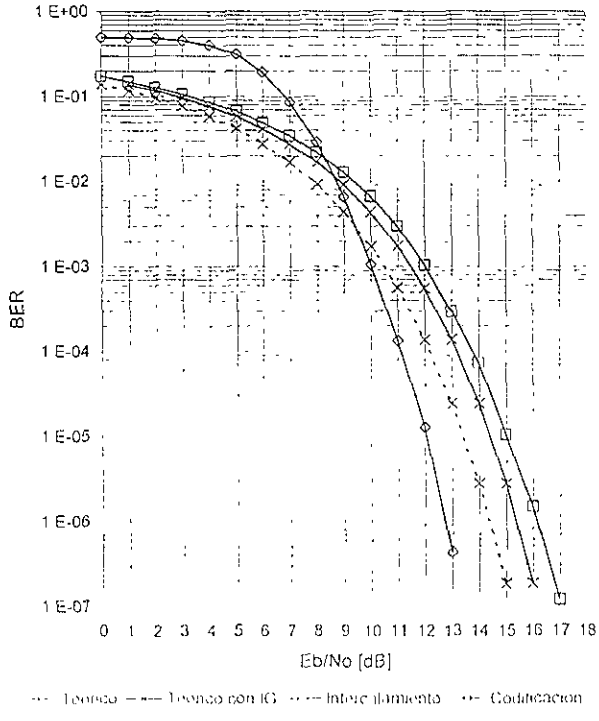
Resultados OFDM-16QAM, símbolos cortos variables, 10 símbolos largos



Resultados OFDM-16QAM, 10 símbolos cortos, símbolos largos variables



Resultados OFDM-16QAM con Intercalamiento y Codificación



6 Algoritmos para implementar el hardware

6.1 *Transformada Rápida de Fourier*

Como se ha explicado en los capítulos anteriores, para poder implementar la Transformada Rápida de Fourier Inversa y Directa se pueden utilizar simplemente sumas, restas, intercambios entre parte real y parte imaginaria y rotaciones de fase. De las operaciones mencionadas anteriormente la que presenta un grado de complejidad mas alto es la rotación de fase de un número complejo. Estas rotaciones se llevaron a cabo en las simulaciones por medio de la transformada de rotación general (ecuaciones 5.5). Sin embargo las rotaciones se pueden implementar por medio de un algoritmo CORDIC, el cual solamente utiliza sumas, restas y corrimientos de bits para realizar la rotación. Por lo tanto se pueden ahorrar todas las multiplicaciones de la transformada.

6.1.1 CORDIC

CORDIC es una familia de algoritmos de tipo iterativo para implementar funciones trigonométricas y algunas otras funciones trascendentales que utiliza solamente registros de corrimiento y sumadores para implementarlas. Las funciones trigonométricas se basan en rotaciones vectoriales, mientras que otras funciones, tales como la raíz cuadrada, se implementan utilizando una expresión incremental de la función deseada. CORDIC es el acrónimo de ***CO**ordinate **R**otation **DI**gital **C**omputer* (Computadora Digital de Rotación de Coordenadas). Los algoritmos CORDIC generalmente producen un bit de precisión adicional por cada iteración.

Los algoritmos trigonométricos fueron desarrollados originalmente como una solución digital para problemas de navegación en tiempo real. El trabajo original se le atribuye a Jack Volder. Las extensiones a la teoría de CORDIC, basadas en el trabajo de John Walther y otros, proveen soluciones a una amplia gama de funciones

La rotación vectorial puede ser utilizada para calcular conversiones polares a rectangulares y rectangulares a polares, para obtener la magnitud de un vector y para construir

El ángulo de una rotación está definido por la secuencia de las rotaciones elementales. Esta secuencia puede ser representada por un vector de decisión. El conjunto de todos los vectores de decisión posibles es un sistema de medidas angulares. Las conversiones entre este sistema angular y cualquier otro puede llevarse a cabo utilizando una tabla de búsqueda. Un mejor método de conversión utiliza un sumador-substractor adicional que acumula los ángulos de rotación elementales en cada iteración. Estos valores angulares pueden proveerse por una tabla de búsqueda o pueden alambrarse físicamente. El acumulador de ángulo requiere de una tercera ecuación para el algoritmo CORDIC:

$$z_{i+1} = z_i - d_i \tan^{-1}(2^{-i}) \tag{6.5}$$

El rotador CORDIC opera normalmente en uno de dos modos. El primero, llamado rotación de Volder, rota el vector de entrada por un ángulo especificado (dado como un argumento). El segundo modo, llamado vectorización, alinea el vector de entrada con el eje x obteniendo el ángulo requerido para hacer la rotación.

En el modo de rotación de Volder, el acumulador de ángulo es inicializado con el ángulo de rotación deseado. La decisión de rotación en cada iteración se hace disminuyendo la magnitud del ángulo residual del acumulador. Por lo tanto, la decisión en cada iteración se basa en el signo del ángulo residual después de cada operación. Para el modo de rotación, las ecuaciones del algoritmo son:

$$\begin{aligned} x_{i+1} &= x_i - y_i d_i 2^{-i} \\ y_{i+1} &= y_i + x_i d_i 2^{-i} \\ z_{i+1} &= z_i - d_i \tan^{-1}(2^{-i}) \end{aligned} \tag{6.6}$$

donde: $d_i = -1$ si $z_i<0$, $+1$ en otro caso, por lo que al final de las iteraciones se obtiene:

$$\begin{aligned} x_n &= A_n[x_0 \cos z_0 - y_0 \sin z_0] \\ y_n &= A_n[y_0 \cos z_0 + x_0 \sin z_0] \\ z_n &= 0 \\ A_n &= \prod_n \sqrt{1 + 2^{-2i}} \end{aligned} \tag{6.7}$$

En el modo de vectorización el algoritmo rota el vector de entrada las veces que sea necesario para alinear el vector resultante con el eje x. El resultado de la operación de vectorización es un ángulo y la magnitud escalada del vector original (la componente x del

Gianfranco Biliotti Rodríguez
Juan Eduardo Galicia Luna

resultado). La función de vectorización trabaja buscando minimizar la componente *y* del vector residual en cada rotación. El signo de la componente residual *y* es utilizado para determinar la siguiente dirección de rotación. Si el ángulo del acumulador es inicializado con cero, contendrá el ángulo de rotación al final de las iteraciones. En el modo de vectorización las ecuaciones resultantes son:

$$\begin{aligned} x_{i+1} &= x_i - y_i d_i 2^{-i} \\ y_{i+1} &= y_i + x_i d_i 2^{-i} \\ z_{i+1} &= z_i - d_i \tan^{-1}(2^{-i}) \end{aligned} \tag{6.8}$$

donde: $d_i = +1$ si $y_i<0$, -1 en otro caso, por lo que el resultado final es:

$$\begin{aligned} x_n &= A_n \sqrt{x_0^2 + y_0^2} \\ y_n &= 0 \\ z_n &= z_0 + \tan^{-1}(y_0/x_0) \\ A_n &= \prod_n \sqrt{1+2^{-2i}} \end{aligned} \tag{6.9}$$

Los algoritmos CORDIC de rotación y de vectorización están limitados a ángulos de rotación entre $-\pi/2$ y $\pi/2$. Esta limitación se debe al uso de 2^0 para la tangente en la primera iteración. Para ángulos de rotación mas grandes que $\pi/2$ se requiere una rotación adicional. Volder describe una rotación inicial de $\pm\pi/2$. Se tiene entonces:

$$\begin{aligned} x' &= -d \cdot y \\ y' &= d \cdot x \\ z' &= z + d \cdot \pi/2 \end{aligned}$$

donde d = +1 si y<0, -1 en otro caso.

En la figura 2.9 se puede ver la Transformada Rápida de Fourier Inversa de 16 puntos construida a partir de dos etapas de mariposa de raíz cuarta, separadas por una etapa intermedia donde los 16 resultados sufren una rotación de fase por un factor ω^i, el cual se encuentra definido por la ecuación $\omega^i = exp(j2\pi i/N)$. Esta rotación puede realizarse fácilmente con el algoritmo de rotación de Volder.

Para entender el algoritmo se analizará un ejemplo donde se utiliza un formato de punto fijo de 8 bits: 1 bit de signo, 2 bits para la parte entera del número y 5 bits para la fracción. En

primer lugar debemos obtener la tabla de ángulos elementales de rotación. La segunda columna de la **tabla 6.1** indica el valor de $\phi = 2^{-i}$. La tercera columna lista los valores de $\tan^{-1}(2^{-i})$ en radianes para convertirlos al formato de punto fijo sugerido. Esta conversión se lleva a cabo dividiendo el valor entre la resolución del formato ($2^{-5} = 0.03125$). Por ejemplo $0.12435/0.03125 = 3.9793 \approx 4$. Los valores de la última columna son los que se utilizarán para llevar a cabo las iteraciones del algoritmo.

i	2^{-i}	$\tan^{-1}(2^{-i})$ [°]	$\tan^{-1}(2^{-i})$ [rad]	Punto fijo	Punto fijo (redondeo)	Binario
0	1 00000	45.00000	0.78539	25.1327	25	00001101
1	0 50000	26.56505	0.46364	14.8367	15	00001110
2	0.25000	14.03624	0.24497	7.8393	8	00001000
3	0.12500	7.12501	0.12435	3.9793	4	00000100
4	0.06250	3.57633	0.06241	1.9974	2	00000010
5	0.03125	1.78991	0.03123	0.9996	1	00000001

Tabla 6.1 Tabla de ángulos elementales de rotación.

Ocupando las ecuaciones (6.6) para un vector $x = 00001000_2$, $y = 00000101_2$, con un ángulo de giro de $z = 00000111_2$, se tiene un resultado al final de las cinco iteraciones de $x' = 00001011_2$, $y' = 00001011_2$. La **tabla 6.2** muestra los resultados parciales que se obtienen entre cada iteración.

I	x_i	y_i	z_i	d_i	$\tan^{-1}(2^{-i})$	x_{i+1}	y_{i+1}	z_{i+1}
0	**8**	**5**	**7**	1	25	3	13	8
1	3	13	-18	-1	15	9	12	-3
2	9	12	-3	-1	8	12	10	5
3	12	10	5	1	4	11	11	1
4	11	11	1	1	2	11	11	-1
5	11	11	-1	-1	1	**11**	**11**	**0**

Tabla 6.2 Resultados parciales para el algoritmo de Volder.

Para comprender los resultados se puede pasar de la base binaria al número en base decimal que representa: $x = (00001000_2)\times(2^{-5}) = 0.25$, $y = (00000101_2)\times(2^{-5}) = 0.15625$, $z = (00000111_2)\times(2^{-5}) = 0.21875$ [rad] $\approx 12.53°$. Si se lleva a cabo la rotación exacta por medio de la transformada de rotación general (ecuaciones 6.1) se tiene que $x' = 0.21014$, $y' = 0.20676$. Con el algoritmo de Volder se obtiene que $x = (00001011_2)\times(2^{-5}) = 0.34375$, $y = 0.34375$. Podemos ver que el algoritmo provee una ganancia al vector de

$$\frac{\sqrt{0.34375^2 + 0.34375^2}}{\sqrt{0.21014^2 + 0.20676^2}} = 1.648$$

que se aproxima a la ganancia cuando el número de iteraciones tiende a infinito (1.647). El error de fase es menor de 0.5° entre ambos resultados. Si el algoritmo utiliza mas bits de precisión en la parte fraccionaria, entonces este error de fase tenderá a cero.

El algoritmo CORDIC de vectorización se puede utilizar para implementar la función trigonométrica tan^{-1} que se requiere para hacer estimaciones en el algoritmo de sincronización de fase. Utilizando los mismos vectores *x*, *y* del ejemplo anterior y la tabla 6.1 para 5 bits de precisión y utilizando las ecuaciones (6.8) se tiene como resultado: x' = 00010001, y' = 11111111, z' = 000010011, es decir, x' = 0.53125, y' = -0.03125, z' = 0.59375 [rad] = 34.02°.

i	x_i	y_i	z_i	d_i	$\tan^{-1}(2^{-i})$	x_{i+1}	y_{i+1}	z_{i+1}
0	**8**	**5**	**0**	-1	25	13	-3	25
1	13	-3	25	1	15	15	3	10
2	15	3	10	-1	8	15	0	8
3	15	0	8	-1	4	15	-1	22
4	15	-1	22	1	2	16	-1	20
5	16	-1	20	1	1	**17**	**-1**	**19**

Tabla 6.3 Resultados parciales para el algoritmo de vectorización.

El resultado exacto de la fase es 32.005°. El error es de 2° pero, como se mencionó anteriormente, este error puede disminuirse si la resolución del formato aumenta.

7 Conclusiones

Tendencias actuales de comunicación inalámbrica

En la actualidad, los usuarios de los sistemas de comunicación exigen una capacidad mayor para transmitir información. Cada vez más la movilidad es un factor determinante en el éxito o el fracaso de una tecnología. Los usuarios prefieren utilizar productos que puedan soportar diferentes servicios, como vídeo, datos, voz, etc. Las redes actuales buscan cumplir estas expectativas, utilizando métodos de comunicación innovadores y robustos. La diversidad de las técnicas utilizadas necesita una alta capacidad de integración en los equipos para poder realizar tareas que en el pasado parecían imposibles de implementar debido a su complejidad. Ahora existen circuitos de alta escala de integración que permiten procesar las señales de una manera más eficiente, con mayores tasas de transmisión y menores potencias, permitiendo la utilización de dispositivos pequeños a costos accesibles.

La estandarización ha permitido que la interconexión de estos equipos sea posible. Sin embargo, no hay un estándar único que tenga ventajas competitivas contundentes sobre todos los demás. Es necesario entonces entender la problemática real que presenta el sistema para poder escoger la tecnología adecuada que satisfaga nuestras necesidades.

El objeto de este estudio se centró en simular un sistema basado en el estándar IEEE 802.11a. La razón principal de utilizar este estándar es la comprobada aceptación de productos de la misma familia (IEEE 802.11 y 802.11b) en el mercado, ya que otros estándares como el Hiperlan/2, el cual presenta algunas ventajas competitivas irrefutables, a penas está en proceso de ser implementado y comercializado. En el mundo que vivimos hoy, es muy importante la capacidad de respuesta en el mercado, por lo que muchos fabricantes e integradores de equipos que ya tienen desarrollada una gama de productos basados en estándares de IEEE, no piensan migrar inmediatamente al estándar Hiperlan/2 a pesar de la mayor capacidad de servicios que ofrece.

Ventajas de OFDM

Las principales ventajas, como se pudo demostrar en este estudio, son: el desempeño robusto en ambientes con multitrayectorias, aspecto fundamental para soportar la movilidad de los sistemas; la utilización eficiente del espectro radioeléctrico que nos permite utilizar técnicas de corrección de error para hacer más robusta la señal y continuar teniendo una relación de señal a ruido cercana a la teórica, cuando se utiliza codificación e intercalamiento a pesar de los perfiles con multitrayectorias.

Pudimos observar en los resultados que algunas subportadoras se benefician por las multitrayectorias, fenómeno que facilita la recuperación de la señal. Otras subportadoras, sin embargo, se atenúan significativamente y evitan que los datos originales sean recuperados. Esto no impide que el desempeño del sistema sea aceptable en todos los canales de transmisión cuyo retardo de las multitrayectorias es menor que el intervalo de guarda.

Al comparar un sistema OFDM con uno de portadora única podemos identificar varias ventajas. La más significativa se refiere a la capacidad que tiene para distribuir los errores en todas las subportadoras, haciendo al sistema menos susceptible a ráfagas de errores que son difíciles de corregir. Al tener el error distribuido en varias señales, algoritmos como el de Viterbi pueden corregir fácilmente errores pequeños en cada subportadora.

Limitaciones de OFDM

Las dos limitaciones principales de OFDM son su sensibilidad al ruido de fase y la relación de potencia pico a potencia promedio. Un sistema OFDM requiere que el ruido de fase de sus componentes sea 10 dB menor que el de un sistema de portadora única con las mismas características. El costo asociado para resolver este problema es bajo, ya que únicamente se deben sustituir los osciladores para tener una frecuencia estable (con una desviación menor al 5% del espacio entre subportadoras) y con un ancho de banda a –3 dB no mayor a 10 Hz para que sus efectos sean despreciables. El segundo efecto mencionado disminuye la cobertura del sistema, ya que los picos de potencia pueden saturar el amplificador de transmisión, por lo tanto se debe disminuir el back-off, obteniendo como resultado una menor amplificación (una desventaja con

respecto a los sistemas de portadora única de 0.5 a 1.5 dB[11]). Este efecto se puede corregir o atenuar utilizando códigos de reducción de relación potencia pico/potencia promedio.

Lugar que ocupa OFDM en la solución de los problemas de la comunicación

En nuestra opinión la tecnología basada en OFDM tiene un futuro promisorio en el desarrollo de las telecomunicaciones, ya que ocupa un nicho de gran interés para los usuarios: la transmisión de voz, datos y vídeo a grandes velocidades, con soporte de baja movilidad en su modalidad inalámbrica. En la actualidad los sistemas móviles de transmisión soportan tasas de datos muy bajas comparados con los sistemas fijos alámbricos. Las aplicaciones actuales requieren de anchos de banda cada vez mayores lo que obliga a los sistemas móviles a actualizarse para disminuir la brecha que existe con sus competidores fijos.

La importancia que tiene esta información requiere de sistemas confiables para su transmisión, cuidando la eficiencia con la que se utiliza el espectro radioeléctrico. OFDM tiene como propósito solucionar parte de estos problemas, llenando un hueco que habían dejado otros sistemas inalámbricos, proporcionando grandes tasas de transmisión, utilizando de manera eficiente el espectro, ofreciendo un alto grado de confiabilidad.

Resultados obtenidos

Los resultados muestran que se puede alcanzar un desempeño de tasa de error de bits similar al teórico para las diferentes modulaciones estudiadas, utilizando un algoritmo de decodificación con decisión dura e intercalamiento, aunque este desempeño se puede mejorar todavía mas implementando un algoritmo de decodificación con decisión suave.

Es posible desarrollar un sistema OFDM con un nivel de complejidad mucho menor comparado con un sistema de portadora única con ecualizador, ya que, como se ha demostrado, las únicas operaciones utilizadas son sumas, restas, intercambios entre parte real y parte imaginaria y corrimientos de bits para implementar la FFT, el codificador y decodificador, el intercalador y desintercalador y los algoritmos de sincronización de fase y temporización. Solo se necesitan utilizar multiplicaciones complejas para realizar la correlación con el preámbulo

conocido en el algoritmo de sincronización. Todas estas operaciones son necesarias para que el desempeño del sistema sea adecuado, ya que contribuyen a disminuir la tasa de error de bits en el receptor y a que el sistema soporte las desviaciones que sufre la señal en canales de transmisión típicos para transmisiones inalámbricas terrestres.

Se demuestra entonces por medio de simulaciones que es posible realizar un sistema de transmisión inalámbrica utilizando Multiplexación por División Ortogonal de Frecuencias basado en el estándar IEEE 802.11a capaz de transmitir información con una tasa de error de bits similar a la teórica a través de un canal con ruido gaussiano y multitrayectorias.

Aportaciones

En el presente estudio se recopila la información necesaria para el desarrollo de un sistema de comunicaciones basado en la Multiplexación por División Ortogonal de Frecuencias. Se incluye desde un panorama de los sistemas estandarizados basados en OFDM dentro de los sistemas de comunicación actuales hasta las técnicas en software para la simulación de un sistema con los elementos necesarios para su funcionamiento.

En este estudio aportamos los algoritmos para la implementación de la Transformada Rápida de Fourier Inversa y Directa de una manera eficiente, que puede servir como base para su realización en dispositivos genéricos como DSP o FPGA. Además, realizamos algoritmos para simular otras etapas fundamentales para un Sistema OFDM, como: filtros optimizados, algoritmos de sincronización, codificación, decodificación e intercalamiento, cuidando siempre que las operaciones fueran sencillas y la menor cantidad posible, para que en un futuro se puedan implementar en circuitos integrados con un costo relativamente bajo y con un desempeño óptimo. Además se presenta la familia de algoritmos CORDIC, que facilitan aún más el cálculo de las operaciones que debe realizar el circuito integrado, aumentando su eficiencia.

Aplicaciones posibles

Este tipo de sistema puede ser utilizado principalmente en aplicaciones de datos (e.g. Internet, vídeo) con movilidad media; es decir, cuando se envía una ráfaga de datos el canal no cambia hasta que termina esta transacción. Una nueva ráfaga de datos puede ser enviada desde otro lugar dentro del área de cobertura de la señal, ya que cada vez que se envía una ráfaga se envían los símbolos de entrenamiento necesarios para caracterizar al canal. Esto es de gran utilidad para redes de área local inalámbricas en edificios, aeropuertos, universidades y cualquier otro ambiente con multitrayectorias en el que se necesite libertad de movimiento y una tasa de transmisión elevada.

La transmisión de voz es posible con estos sistemas. Sin embargo, aunque el estándar IEEE 802.11a soporta un cierto grado de calidad de servicio (QoS), no puede garantizarlo al 100%. Esta es una desventaja con respecto al estándar Hiperlan/2. No obstante, la aceptación del estándar IEEE 802.11a en el mercado y la compatibilidad con productos ya existentes para capas superiores, nos hacen pensar que en un futuro muy cercano se actualizará y evolucionará de tal suerte que soportará aplicaciones de voz.

Siguientes pasos

Este estudio considera los aspectos más importantes para realizar un sistema de transmisión OFDM. Sin embargo, se debe tomar en cuenta que no se considera el efecto de ruido de fase en las simulaciones y tampoco se analiza el problema de la relación potencia pico/potencia promedio, factores que se mencionan como desventajas de los sistemas basados en OFDM. Una extensión de este estudio podría abarcar estos temas, que pueden ser controlados por medio de osciladores de buena calidad y códigos de reducción de relación potencia pico/potencia promedio.

El siguiente paso sería la realización del sistema simulado, utilizando los algoritmos presentados para el desarrollo de un sistema de comunicación barato, eficiente y robusto, pensando en la adopción del estándar IEEE 802.11a.

8 Índices

8.1 Índice de figuras

8.2 Índice de tablas

8.3 Índice de ecuaciones

8.4 Índice de acrónimos

- **Codificación**: Técnica que incluye bits de redundancia que permiten la corrección de errores que se generen en el canal de transmisión.
- **Conmutación de circuitos**: Conexión física o electrónicamente conmutada que conecta al emisor con el receptor directamente
- **Conmutación de paquetes**: Transferencia de datos por medio de paquetes ajustados. El canal se ocupa sólo durante la transmisión del paquete. Existe un circuito virtual entre los sitios de comunicación.
- **Decibel (dB)**: Es una forma de indicar el cociente de dos potencias de manera logarítmica. No es una unidad, es una cantidad adimensional. Se define por medio de la expresión: 10 log (P_1/P_2) [dB]
- **Densidad espectral de potencia (PSD)**: La distribución de la potencia total de una señal presente en un ancho de banda dado. Se expresa generalmente en Watts/Hz
- **Distorsión**: Es una alteración de la señal debido a uno o varios subsistemas no adecuados para la señal que se transmite.
- **Ethernet**: Estándar para redes de área local de la IEEE
- **FDM**: Multiplexación por División de Frecuencias. Técnica que consiste en transmitir varios mensajes al mismo tiempo a través de un canal de banda ancha modulando primero las señales de mensaje en varias subportadoras y formando una señal de banda base compuesta que consiste en la suma de estas subportadoras moduladas.
- **Filtro FIR**: Filtro de Respuesta Finita al Impulso. Filtro que utiliza un número de muestras finito para obtener una respuesta. Los coeficientes del filtro dependen del rango de frecuencias que se quiere rescatar.
- **ICI**: Interferencia entre Portadoras. Fenómeno que ocurre cuando el ancho de banda de una señal es invadido por el ancho de banda de otra señal.
- **Interferencia**: Es la contaminación de la señal deseada por otras señales con características similares. Se puede eliminar conociendo quién o qué la ocasiona.
- **IP**: Conjunto de capas de protocolos que permiten la interconexión de computadoras con diferentes características a una red Internet. Corresponde a las capas 3 y 4 del modelo OSI (Interconexión de Sistema Abierto)
- **ISDN**: Red Digital de Servicios Integrados. Servicio que proporciona un enlace de datos digital en forma directa al usuario sobre una línea de par trenzado

- **ISI**: Dispersión en tiempo de un símbolo que provoca que los límites entre símbolos adyacentes se pierdan.
- **LAN**: Red de Área Local. Conexión de instalaciones de computadora en red.
- **Modulación**: El cambio de las características de una señal (e.g. Frecuencia, amplitud o fase) por medio de otra señal
- **MPEG-2**: Método de compresión para vídeo que permite un ahorro de ancho de banda considerable.
- **Multimedia**: Integración de texto, gráficas y sonido en la presentación de la información
- **Multiplexación**: Combinación de dos o más canales de información en un medio de
- transmisión compartido. Esta combinación puede hacerse en tiempo, frecuencia o por medio de códigos.
- **Multitrayectoria**: Fenómeno que ocurre cuando una señal de radio se refleja en diversos objetos que existen en el medio (e.g. edificios, vehículos, etc) provocando que llegue a la antena receptora por mas de una trayectoria.
- **Ortogonalidad**: Condición que se obtiene cuando un conjunto de funciones satisfacen que el producto interno definido en el espacio es cero.
- **Paquete**: Grupo de bits con bits de identificación, dirección o control agregados.
- **Protocolo**: Reglas para el control de la transmisión de datos sobre una red de comunicación
- **Ruido**: Son señales aleatorias con un comportamiento impredecible que se generan dentro y fuera del sistema de comunicaciones, que se agregan a la señal de información. El ruido es un efecto que no se puede eliminar.

10 Referencias

[1] Losquadro, G.; "Digital Audio Broadcasting: High-grade service quality through onboard processing techniques", http://www.ieee.org/.

[2] Bensberg, G.; "Digital Satellite Transmission Standards: An overview of the DVB Satellite specification and its implementation" , http://www.ieee.org/.

[3] Molina, A., Prieto, H., Seseña, J.; "Advantages of SMATV for interactive digital TV" , http://www.ieee.org/.

[4] Weinstein S, Ebert P.; "Data Transmision by Frequency - Division Multiplexing using the Discrete Fourier Transform"; IEEE Trans. Comm. Tech.; Vol. 19, No. 5.; Oct.,1971.

[5] Saleh, A., Valenzuela, R.; "A Statistical Model for Indoor Multipath Propagation"; IEEE Journal on Selected Areas in Communications; Vol SAC-5; No. 2, pp. 128-137; Feb. 1987.

[6] Devarsivatham, D.; "Time Delay Spread Measurements at 850 MHz and 1.7 GHz inside a Metropolitan Office Building"; Electronic Letters; pp. 194-196; Feb. 1989.

[7] Hawbaker, D., Rappaport, T.; "Indoor Wideband Radiowave Propagation Measurements at 1.3 GHz and 4.0 GHz"; Electronic Letters; Vol 26; No. 21; pp 1800-1802; Oct. 1990.

[8] Pollet, T., M. Van Bladel and M. Moeneclaey, "BER Sensitivity of OFDM Systems to Carrier Frequency Offset and Wiener Phase Noise" IEEE Trans. On Comm., vol. 43, pp. 191-193, Feb-Apr, 1995.

[9] Nee, R., Prasad, R.; "OFDM Wireless Multimedia Communications"; Artech House; U.S.A., 2000.

[10] Prasad, R.; "Universal Wireless Personal Communications"; Artech House; U.S.A., 1998.

[11] Ayanoglu, Ender y Otros; "VOFDM Broadband Wireless Transmissions and its advantages over Single Carrier Modulation"; Broadband Wireless Internet Forum; U.S.A., 2000.

[12] Ayanoglu, Douglas y Otros; "Bringing Broadband Wireless Access Indoors"; Broadband Wireless Internet Forum; U.S.A., 2000.

[13] Massel, Mark; "Digital Television. DVB-T COFDM and ATSC 8-VSB"; Digital TV Books, UK, 2000.

11 Programas de simulación

A continuación se enlistan los programas que se utilizaron para simular el sistema de comunicación OFDM.

11.1 Programa de transmisión

```
/* ofdmtx.c : ************************************************************

  Universidad Nacional Autonoma de Mexico, Facultad de Ingenieria

  OFDM (Orthogonal Frecuency Division Multiplexing) Transmission System

  Parameters:

       in      : name of the in file
       out     : name of the out file
       bps     : bits per subcarrier ( 1=BPSK, 2=QPSK, 4=16QAM )
       gtime   : guard time adding (1/4 of symbol time) ( 1=Yes, 0=No )
       swind   : symbol window samples ( 0 to 32 )
       inits   : number of initial null samples
       shrts   : number of short preamble symbols
       longs   : number of long preamble symbols
       [spect] : real & imag spectrums output flag
       [baseb] : inph & quad baseband signals output flag
       [basei] : inph & quad baseband interpolated signals output flag
       [texto] : text output flag

  Notes:

       * Cyclic Extension adding (Enabler)
       * Symbol windowing (in time domain) (Number of window samples)
       * Preamble (short & long) adding (Number of symbols)
       * Continue transmission mode (No bursts partition)
       * No Viterbi Codification
       * No Bit Interleaving

  bps  = 1 (BPSK), 2 (QPSK), 4 (16QAM) [Bits Per Subcarrier]
  sps  = 64 [Subcarriers Per OFDM Symbol = IFFT size]
  aps  = 48 [Active Subcarriers Per OFDM Symbol]
  BpS  = aps*bps/8 = 6,12,24  [Bytes Per OFDM Symbol,

************************************************************************** */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>

/* constants used for mapping & modulation ************************** */

#define NSPS   64
#define NAPS   48
#define MAMAX  16
#define MIMAP  64
```

```
#define L   2
#define M  21
#define M1 22

/* number of samples per symbol with interpolation */
#define NSPSI 128

/* auxiliar functions ***************************************************

  subcarrindx  : load active & pilot subcarrier index tables
  ang_mapper   : creates the angle table (map) for the (2^bps) entries
  rot_angles   : calculate the rotation angles for the ifft
  ifft64       : performs the ifft (64 points)
  interpol     : load interpolation filter taps
  load_shrt    : load short preamble symbol (in time domain)
  load_long    : load long  preamble symbol (in time domain)

*********************************************************************** */

void subcarrindx( char *active,char *pilots );
void ang_mapper ( int bps,float *mapi, float *mapq );
void rot_angles ( float *r1r,float *r1i,float *r2r,float *r2i );
void ifft64     ( float *Xr ,float *Xi ,float *xr ,float *xi,
                  float *r1r,float *r1i,float *r2r,float *r2i );
void interpol   ( float *itp );
void load_shrt ( float *spr,float *spi,float *r1r,float *r1i,float *r2r,float *r2i,float *itp  );
void load_long ( float *lpr,float *lpi,float *r1r,float *r1i,float *r2r,float *r2i,float *itp  );

/* main function ******************************************************* */

int main( int argc,char *argv[] )
{
   FILE *in,*i1tx,*q1tx;      /* Input & baseband          */
   FILE *tx,*i2tx,*q2tx;      /* Output & interpol         */
   FILE *rstx,*istx;          /* Real & imag spectrms      */

   unsigned int bps = 0;      /* Bits Per Subcarrier       */
   unsigned int BpS = 0;      /* Bytes Per Symbol          */

   int   baseb=0,basei=0;     /* baseband flags            */
   int   spect=0,texto=0;     /* spectrum & text flag      */
   int   gtime=0;             /* guard time flag           */
   int   swind=0;             /* symbol window samples     */
   char  strng[5]="wb\0";

   float mapi[MAXMAP],mapq[MAXMAP];/* I-Q Constel Stack    */
   float Xr[NSPS],Xi[NSPS];   /* Frequency domain          */
   float xr[NSPS],xi[NSPS];   /* Time domain               */
   float r1r[NSPS],r1i[NSPS]; /* ifft rotation angles      */
   float r2r[NSPS],r2i[NSPS]; /* ifft rotation angles      */

   unsigned char datau[MAXDAT];/* data (uncodified)        */
   unsigned char mask=0;      /* data mask                 */
   int  i=0,j=0,k=0,rB=0;     /* counters                  */

   char active[NAPS],pilots[4];/* subcarriers indeces      */

   float itp[M1];             /* interpolation taps        */
   float xri[M1],xii[M1];     /* Time domain interp.       */
   float bi[NSPS*L],bq[NSPS*L];/* interpolation buffers    */
   float ceb_i[NSPS*L/4];     /* cyclic extension buffer (inph) */
   float ceb_q[NSPS*L/4];     /* cyclic extension buffer (quad) */
   float wndw1[NSPS*L/4];     /* fading out window         */
   float wndw2[NSPS*L/4];     /* fading in window          */

   int   cant = 0,            /* initial [illegible] symbols */
   int   sbit = 0,long = 0,   /* pre-cable [illegible]       */
   float [illegible],         /* [illegible]                 */
   float [illegible],         /* [illegible]                 */
   float [illegible],         /* [illegible]                 */
   float [illegible],         /* [illegible]                 */
```

```
  int   m=0;                  /* phase counter              */

  interpol( itp );            /* interpolating filter taps initializ      */
  rot_angles( r1r,r1i,r2r,r2i);/* ifft rotation angles initialization     */
  load_shrt( spr,spi,r1r,r1i,r2r,r2i,itp );/* load short preamble         */

/* get parameters ********************************************************* */

  if( argc < 9 ) {
     printf( "ofdmtx: in out bps gtime swind inits shrts longs " );
     printf( "[spect baseb basei texto]" ); return( 1 );
  } else {
     bps   = atoi( argv[ 3] );      /* Bits Per Subcarrier : 1,2,4       */
     gtime = atoi( argv[ 4] );      /* Guard time flag                   */
     swind = atoi( argv[ 5] );      /* Symbol windowing samples          */
     inits = atoi( argv[ 6] );      /* Initial null samples              */
     shrts = atoi( argv[ 7] );      /* Short preamble symbols            */
     longs = atoi( argv[ 8] );      /* Long  preamble symbols            */
  }
  if( argc > 9 ) {
     spect = atoi( argv[ 9] );
     baseb = atoi( argv[10] );
     basei = atoi( argv[11] );
     texto = atoi( argv[12] );  /* text output */
  }

  if( texto ) strng[1] = 't';

/* open files to read and write ******************************************* */

  if( (in=fopen( argv[1],"rb" )) == NULL ) {
     printf( "Can't open file : %s",argv[1] ); return( 1 ); }
  if( (tx=fopen( argv[2],strng)) == NULL ) {
     printf( "Can't open file : %s",argv[2] ); return( 1 ); }

  if( spect ) { /* real and imaginary spectrums output streams ******* */
     if( (rstx=fopen( "rstx",strng )) == NULL ) {
        printf( "Can't open file : rstx" ); return( 1 ); }
     if( (istx=fopen( "istx",strng )) == NULL ) {
        printf( "Can't open file : istx" ); return( 1 ); }
  }

  if( baseb ) { /* baseband signals output streams ****************** */
     if( (i1tx=fopen( "i1tx",strng )) == NULL ) {
        printf( "Can't open file : i1tx" ); return( 1 ); }
     if( (q1tx=fopen( "q1tx",strng )) == NULL ) {
        printf( "Can't open file : q1tx" ); return( 1 ); }
  }

  if( basei ) { /* interpolated baseband signals output streams ****** */
     if( (i2tx=fopen( "i2tx",strng )) == NULL ) {
        printf( "Can't open file : i2tx" ); return( 1 ); }
     if( (q2tx=fopen( "q2tx",strng )) == NULL ) {
        printf( "Can't open file : q2tx" ); return( 1 ); }
  }

/* tables & constants initialization ************************************** */

  interpol( itp );            /* interpolating filter taps initializ      */
  subcarrindx( active,pilots );/* subcarriers indeces initialization      */
  rot_angles( r1r,r1i,r2r,r2i);/* ifft rotation angles initialization     */
  BpS   NAPS*bps/8,           /* Bytes Per OFDM Symbol                    */
  switch( bps ) {             /* mask according to bps                    */
     case 1 : mask   0x01; break;
     case 2 : mask = 0x03; break;
     case 4 : mask   0x0F; break;
  }
  for( i=0; i<NI;       ++i ) { sr1[i]   0.0, si1[i]   0.0;
  for( i=0;   ;       ++i ) { sr2[i]   0.0; Sc[i]   0.0;
  for( i=0;   ;       ++i )                 ;
```

```
   for( i=0, i<NSPS*L/4; i++ ) { wndw1[i] = 0.0; wndw2[i] = 1.0; }
   for( i=0; i<swind;    i++ ) {
      wndw1[i] = 0.5*(1.0+cos(M_PI*(float)(i+1)/(float)(swind+1)));
      wndw2[i] = 1.0-wndw1[i];
   }
   load_shrt( spr,spi,r1r,r1i,r2r,r2i,itp ); /* load short preamble*/
   load_long( lpr,lpi,r1r,r1i,r2r,r2i,itp ); /* load long  preamble*/

/* data transmission ************************************************** */

   ang_mapper( bps,mapi,mapq ); /* angle mapping table initialization   */

   /* null samples insertion ************************************** */

   for( i=0; i<inits; i++,m++ )
      if( texto ) fprintf( tx,"%f\n",nullsamp );
      else        fwrite( &nullsamp,4,1,tx );

   m %= 4; /* m is the phase counter */

   /* initial short preamble windowing ******************************** */

   for( i=0; i<swind; i++,m++ ) {
      switch( m%4 ) {
        case 0: sample = +spr[NSPSI-swind+i]*wndw2[i]; break;
        case 1: sample = +spi[NSPSI-swind+i]*wndw2[i]; break;
        case 2: sample = -spr[NSPSI-swind+i]*wndw2[i]; break;
        case 3: sample = -spi[NSPSI-swind+i]*wndw2[i]; break;
      }
      if( texto ) fprintf( tx,"%f\n",sample );
      else        fwrite( &sample,4,1,tx );
   }
   m %= 4;

   /* write the short preamble shrts times **************************** */

   for( i=0; i<shrts; i++ ) {
      for( j=0; j<NSPSI; j++,m++ ) {
        switch( m%4 ) {
           case 0: sample = +spr[j]; break;
           case 1: sample = +spi[j]; break;
           case 2: sample = -spr[j]; break;
           case 3: sample = -spi[j]; break;
        }
        if( texto ) fprintf( tx,"%f\n",sample );
        else        fwrite( &sample,4,1,tx );
      }
   }
   m %= 4;

   /* initial long preamble cyclic extension windowing **************** */

   for( i=0; i<swind; i++,m++ ) {
      switch( m%4 ) {
        case 0: sample = +spr[i]*wndw1[i] + lpr[3*NSPSI/4+i]*wndw2[i]; break;
        case 1: sample = +spi[i]*wndw1[i] + lpi[3*NSPSI/4+i]*wndw2[i]; break;
        case 2: sample = -spr[i]*wndw1[i] - lpr[3*NSPSI/4+i]*wndw2[i]; break;
        case 3: sample = -spi[i]*wndw1[i] - lpi[3*NSPSI/4+i]*wndw2[i]; break;
      }
      if( texto ) fprintf( tx,"%f\n",sample );
      else        fwrite( &sample,4,1,tx );
   }
   m %= 4;

   /* continue writing long preamble cyclic extension ***************** */

   for( /* i must have the last value*/; i<NSPSI/4; i++,m++ ) {
      switch( m%4 ) {
        case 0: sample = +lpr[3*NSPSI/4+i]; break;
        case 1: sample = +lpi[3*NSPSI/4+i]; break;
        case 2: sample = -lpr[3*NSPSI/4+i]; break;
        case 3: sample = -lpi[3*NSPSI/4+i]; break;
      }
      if( texto ) fprintf( tx,"%f\n",sample );
      else        fwrite( &sample,4,1,tx );
   }
```

```
      case 3: sample = -lpi[3*NSPSI/4+i]; break;
    }
    if( texto ) fprintf( tx,"%f\n",sample );
    else        fwrite( &sample,4,1,tx );
  }
  m %= 4;

  /* write the long preamble longs times ***************************** */

  for( i=0; i<longs; i++ ) {
    for( j=0; j<NSPSI; j++,m++ ) {
      if( (j<swind)&&(i==longs-1) ) { ceb_i[j]=lpr[j]; ceb_q[j]=lpi[j]; }
      switch( m%4 ) {
        case 0: sample = +lpr[j]; break;
        case 1: sample = +lpi[j]; break;
        case 2: sample = -lpr[j]; break;
        case 3: sample = -lpi[j]; break;
      }
      if( texto ) fprintf( tx,"%f\n",sample );
      else        fwrite( &sample,4,1,tx );
    }
  }
  m %= 4;

  /* transmit the OFDM symbols burst ********************************** */

  rB = fread( datau,1,BpS,in );

  do {

    if( rB<BpS ) for( i=rB; i<BpS; i++ ) datau[i] = 0x00;

    for( i=0; i<NSPS; i++ ) { Xr[i] = 0.0; Xi[i] = 0.0; }

  /* subcarrier amplitude & phase mapping (map inph & map quad) ****** */

    for( i=0,k=0; i<BpS; i++ ) {
      for( j=0; j<(8/bps); j++ ) {
        Xr[ active[k] ] = mapi[ datau[i]&mask ];
        Xi[ active[k] ] = mapq[ datau[i]&mask ];
        datau[i]>>=bps; k++;
      }
    }

    if( spect ) { /* write real & imag spectrums ******************** */
      if( texto ) for( i=0; i<NSPS; i++ ) { fprintf( rstx,"%f\n",Xr[i] ); fprintf(
istx,"%f\n",Xi[i] ); }
      else        { fwrite( Xr,4,NSPS,rstx ); fwrite( Xi,4,NSPS,istx ); }
    }

    ifft64( Xr,Xi,xr,xi,r1r,r1i,r2r,r2i ); /* frec to time transform  */

    if( baseb ) { /* write baseband signals ************************** */
      if( gtime ) {
        if( texto ) for( i=48; i<NSPS; i++ ) { fprintf( i1tx,"%f\n",xr[i] ); fprintf(
q1tx,"%f\n",xi[i] ); }
        else { fwrite( &xr[48],4,NSPS/4,i1tx ); fwrite( &xi[48],4,NSPS/4,q1tx ); }
      }
      if( texto ) for( i=0; i<NSPS; i++ ) { fprintf( i2tx,"%f\n",xr[i] ); fprintf(
q2tx,"%f\n",xi[i] ); }
      else { fwrite( xr,4,NSPS,i1tx ); fwrite( xi,4,NSPS,q2tx ); }
    }

  /* interpolation filtering ************************************** */

    for( i=0 ; i<12; i++ ) { xi1[i]   = xi[11-i]; xi1[i]   = xr[11-i]; }
    for( i=12; i<Mi; i++ ) { xr1[i]   = xr[i-12]; xi1[i]   = xr[i-12]; }

    for( i=0 ; i<Mi; i++ ) { xr2[i]   = xr[i]   ; xi2[i]   = xi[i]   ; }

    for( i=0 ; i<Mi; i++ ) { xi2[i]   = xi[i]   ; xr2[i]   = xr[i]   ; }
```

```
      bi[i*L+1]  = (xri[21]+xri[ 0])*itp[ 0] + (xri[20]+xri[ 1])*itp[ 2];
      bi[i*L+1] += (xri[19]+xri[ 2])*itp[ 4] + (xri[18]+xri[ 3])*itp[ 6];
      bi[i*L+1] += (xri[17]+xri[ 4])*itp[ 8] + (xri[16]+xri[ 5])*itp[10];
      bi[i*L+1] += (xri[15]+xri[ 6])*itp[12] + (xri[14]+xri[ 7])*itp[14];
      bi[i*L+1] += (xri[13]+xri[ 8])*itp[16] + (xri[12]+xri[ 9])*itp[18];
      bi[i*L+1] += (xri[11]+xri[10])*itp[20];

      bq[i*L] = xii[11];

      bq[i*L+1]  = (xii[21]+xii[ 0])*itp[ 0] + (xii[20]+xii[ 1])*itp[ 2];
      bq[i*L+1] += (xii[19]+xii[ 2])*itp[ 4] + (xii[18]+xii[ 3])*itp[ 6];
      bq[i*L+1] += (xii[17]+xii[ 4])*itp[ 8] + (xii[16]+xii[ 5])*itp[10];
      bq[i*L+1] += (xii[15]+xii[ 6])*itp[12] + (xii[14]+xii[ 7])*itp[14];
      bq[i*L+1] += (xii[13]+xii[ 8])*itp[16] + (xii[12]+xii[ 9])*itp[18];
      bq[i*L+1] += (xii[11]+xii[10])*itp[20];

      for( j=M; j>0; j-- ) { xri[j] = xri[j-1]; xii[j] = xii[j-1]; }

      xri[0] = (i+12<NSPS)? xr[i+12]:xr[i-NSPS+12];
      xii[0] = (i+12<NSPS)? xi[i+12]:xi[i-NSPS+12];

    }

 /* end interpolation filtering ************************************* */

    if( basei ) { /* write interpolated baseband signals ********* */
      if( gtime ) {
        if( texto ) for( i=48*L; i<NSPSI; i++ ) { fprintf( i2tx,"%f\n",bi[i] ); fprintf(
q2tx,"%f\n",bq[i] ); }
        else { fwrite( &bi[48*L],4,NSPSI/4,i2tx ); fwrite( &bq[48*L],4,NSPSI/4,q2tx ); }
      }
      if( texto ) for( j=0; j<NSPSI; j++ ) { fprintf( i2tx,"%f\n",bi[j] ); fprintf(
q2tx,"%f\n",bq[j] ); }
      else { fwrite( bi,4,NSPSI,i2tx ); fwrite( bq,4,NSPSI,q2tx ); }
    }

 /* write one OFDM symbol ******************************************* */

    if( gtime ) {

    /* windowing ***************************************************** */

      for( i=3*NSPSI/4,k=0; k<swind; i++,k++,m++ ) {
        switch( m%4 ) {
          case 0: sample = +ceb_i[k]*wndw1[k] + bi[i]*wndw2[k]; break;
          case 1: sample = +ceb_q[k]*wndw1[k] + bq[i]*wndw2[k]; break;
          case 2: sample = -ceb_i[k]*wndw1[k] - bi[i]*wndw2[k]; break;
          case 3: sample = -ceb_q[k]*wndw1[k] - bq[i]*wndw2[k]; break;
        }
        if( texto ) fprintf( tx,"%f\n",sample ),
        else        fwrite( &sample,4,1,tx );
      }
      m %= 4;

    /* end windowing & continue writing the cyclic extension ******** */

      for( /*i -> must have the last value*/; i<NSPSI; i++,m++ ) {
        switch( m%4 ) {
          case 0  sample = +bi[i]; break;
          case 1: sample = +bq[i]; break;
          case 2: sample = -bi[i]; break;
          case 3: sample = -bq[i]; break;
        }
        if( texto ) fprintf( tx,"%f\n",sample ),
        else        fwrite( &sample,4,1,tx );
      }
      m %= 4;
    }

    /* [illegible] ************************************** */
```

```
      for( i=0; i<NSPSI; i++,m++ ) {
        if( i<swind ) { ceb_i[i] = bi[i]; ceb_q[i] = bq[i]; }
        switch( m%4 ) {
           case 0: sample = +bi[i]; break;
           case 1: sample = +bq[i]; break;
           case 2: sample = -bi[i]; break;
           case 3: sample = -bq[i]; break;
        }
        if( texto ) fprintf( tx,"%f\n",sample );
        else        fwrite( &sample,4,1,tx );
      }
      m %= 4;

      rB = fread( datau,1,BpS,in );

   } while( rB );

/* close files ************************************************** */

   if( spect ) { fclose( rstx ); fclose( istx ); }
   if( baseb ) { fclose( iltx ); fclose( qltx ); }
   if( basei ) { fclose( i2tx ); fclose( q2tx ); }
   fclose( in ); fclose( tx );

   return( 0 ); /* No error, finished o.k. */
}

/* auxiliar functions ******************************************** */

void subcarrindx( char *active,char *pilots )
{
   char activ[NAPS] = {
      38,39,40,41,42,
      44,45,46,47,48,49,50,51,52,53,54,55,56,
      58,59,60,61,62,63,
      1 ,2 ,3 ,4 ,5 ,6 ,
      8 ,9 ,10,11,12,13,14,15,16,17,18,19,20,
      22,23,24,25,26
   };
   char pilot[4] = { 43,57,7,21 }; int i;

   for( i=0; i<NAPS; i++ ) active[i] = activ[i];
   for( i=0; i<4; i++ )    pilots[i] = pilot[i];
}

void ang_mapper( int bps,float *mapi, float *mapq )
{
   int   i,j,ind[MAXMAP];
   float nfact = 0.0; /* Normalization Factor */
   float auxmapi[MAXMAP],auxmapq[MAXMAP];

   switch( bps ) {
      case 1 : /* BPSK mapping ********************************** */
        mapi[0] = -1.0; mapq[0] = 0.0;
        mapi[1] =  1.0; mapq[1] = 0.0; break;
      case 2 : /* QPSK mapping ********************************** */
        nfact = sqrt( 2.0 );
        mapi[0] = -1.0/nfact; mapq[0] = -1.0/nfact;
        mapi[1] =  1.0/nfact; mapq[1] = -1.0/nfact;
        mapi[2] = -1.0/nfact; mapq[2] =  1.0/nfact;
        mapi[3] =  1.0/nfact; mapq[3] =  1.0/nfact; break;
      case 4 : /* 16QAM mapping ********************************* */
        nfact = sqrt( 10.0 );
        for( j=0; j<4; j++ ) {
           for( i=0; i<4;  i++ ) {
              auxmapi[j*4+i] = (-3.0+2.0*(float)i)/nfact;
              auxmapq[j*4+i] = (-3.0+2.0*(float)j)/nfact;
           }
```

```
        ind[ 8]=12; ind[ 9]=14; ind[10]=15; ind[11]=13;
        ind[ 4]= 8; ind[ 5]=10; ind[ 6]=11; ind[ 7]= 9;
        ind[ 0]= 0; ind[ 1]= 2; ind[ 2]= 3; ind[ 3]= 1;
        for( i=0; i<16; i++ ) {
           mapi[ind[i]] = auxmapi[i];
           mapq[ind[i]] = auxmapq[i];
        }
        break;
   }


void rot_angles( float *r1r,float *r1i,float *r2r,float *r2i )

   float angle=(2.0*M_PI)/64.0;
   char  l=0,m=0,n=0,o=0;
   char  a[NSPS] = {
      0,0,0,0,0,4,8,12,0,8,16,24,0,12,24,36,0,1,2,3,0,5,10,15,0,9,18,27,
      0,13,26,39,0,2,4,6,0,6,12,18,0,10,20,30,0,14,28,42,0,3,6,9,0,7,14,
      21,0,11,22,33,0,15,30,45
   };

   /* rotation factors calculation */
   for( l=0; l<4; l++ ) {
      for( m=0; m<4; m++ ) {
         for( n=0; n<4; n++ ) {
            r1r[o] = cos( angle*(float)(l*m*4) );
            r1i[o] = sin( angle*(float)(l*m*4) );
            r2r[o] = cos( angle*(float)a[o] );
            r2i[o] = sin( angle*(float)a[o] );
            o++;
         }
      }
   }


void radix4bi( float *X0r,float *X0i,float *X1r,float *X1i,
               float *X2r,float *X2i,float *X3r,float *X3i )

   float x0r,x0i,x1r,x1i,x2r,x2i,x3r,x3i;

   x0r = *X0r+*X1r+*X2r+*X3r, x0i = *X0i+*X1i+*X2i+*X3i,
   x1r = *X0r-*X1i-*X2r+*X3i; x1i = *X0i+*X1r-*X2i-*X3r;
   x2r = *X0r-*X1r+*X2r-*X3r; x2i = *X0i-*X1i+*X2i-*X3i;
   x3r = *X0r+*X1i-*X2r-*X3i; x3i = *X0i-*X1r-*X2i+*X3r;

   *X0r = x0r; *X0i = x0i;
   *X1r = x1r; *X1i = x1i;
   *X2r = x2r; *X2i = x2i;
   *X3r = x3r; *X3i = x3i;


void ifft64( float *Xr ,float *Xi ,float *xr ,float *xi,
             float *r1r,float *r1i,float *r2r,float *r2i )

   float XX;
   char l,m,n,a[16] = { 0,4,8,12,1,5,9,13,2,6,10,14,3,7,11,15 };

   /* radix 4, firts level  */
   for( l=0; l<16; l++ ) {
      radix4bi( Xr+l,Xi+l,Xr+l+16,Xi+l+16,Xr+l+32,Xi+l+32,Xr+l+48,Xi+l+48 );
      for( m=0; m<NSPS, m+=16 ) {
         XX      = Xr[l+m]*r1r[l+m] - Xi[l+m]*r1i[l+m], /* rotation */
         Xi[l+m]   Xr[l+m]*r1i[l+m] + Xi[l+m]*r1r[l+m], /* rotation */
         Xr[l+m]   XX;
      }
   }
   /* radix 4, second level    */
   for( l=0, l<16; l++ ) {
      radix4bi( [illegible] );
      for( [illegible] ) {
         [illegible]
      }
   }
```

```
        XX      = Xr[m+n]*r2r[m+n] - Xi[m+n]*r2i[m+n]; /* rotation */
        Xi[m+n] = Xr[m+n]*r2i[m+n] + Xi[m+n]*r2r[m+n]; /* rotation */
        Xr[m+n] = XX;
      }
   }
   /* radix 4, third level */
   for( l=0; l<NSPS; l+=4 ) {
      radix4bi( Xr+l,Xi+l,Xr+l+1,Xi+l+1,Xr+l+2,Xi+l+2,Xr+l+3,Xi+l+3 );
      for( m=0; m<4; m++ ) {
        xr[a[l/4]+m*16] = Xr[l+m]/64.0;
        xi[a[l/4]+m*16] = Xi[l+m]/64.0;
      }
   }
}

void interpol( float *itp )
{
   float aux[M1] = {
       0.0001,0.0000,-0.0006,0.0000,0.0020,0.0000,-0.0049,0.0000,
       0.0104,0.0000,-0.0197,0.0000,0.0349,0.0000,-0.0598,0.0000,
       0.1030,0.0000,-0.1968,0.0000,0.6313,1.0000
   };
   int i;

   for( i=0; i<M1; i++ ) itp[i] = aux[i];
}

void load_shrt( float *spr,float *spi,float *r1r,float *r1i,float *r2r,float *r2i,float *itp  )
{
   float shrt_frec_r[NSPS] = {
      0.0,
      0.0,0.0,0.0,-1.47196,0.0,0.0,0.0,-1.47196,0.0,0.0,0.0,+1.47196,
      0.0,0.0,0.0,+1.47196,0.0,0.0,0.0,+1.47196,0.0,0.0,0.0,+1.47196,
      0.0,0.0,0.0,     0.0,0.0,0.0,0.0,0.0,     0.0,0.0,0.0,0.0,     0.0,
      0.0,0.0,0.0,+1.47196,0.0,0.0,0.0,0.0,-1.47196,0.0,0.0,0.0,+1.47196,
      0.0,0.0,0.0,-1.47196,0.0,0.0,0.0,-1.47196,0.0,0.0,0.0,+1.47196,
      0.0,0.0,0.0
   };

   float shrt_frec_i[NSPS] = {
      0.0,
      0.0,0.0,0.0,-1.47196,0.0,0.0,0.0,-1.47196,0.0,0.0,0.0,+1.47196,
      0.0,0.0,0.0,+1.47196,0.0,0.0,0.0,+1.47196,0.0,0.0,0.0,+1.47196,
      0.0,0.0,0.0,     0.0,0.0,0.0,0.0,0.0,     0.0,0.0,0.0,0.0,     0.0,
      0.0,0.0,0.0,+1.47196,0.0,0.0,0.0,0.0,-1.47196,0.0,0.0,0.0,+1.47196,
      0.0,0.0,0.0,-1.47196,0.0,0.0,0.0,0.0,-1.47196,0.0,0.0,0.0,+1.47196,
      0.0,0.0,0.0
   };

   float shrt_time_r[NSPS],shrt_time_i[NSPS],xr1[M1],xi1[M1];

   int i=0,j=0;

   ifft64( shrt_frec_r,shrt_frec_i,shrt_time_r,shrt_time_i,r1r,r1i,r2r,r2i );

   /* interpolation filtering ************************************** */

   for( i=0 ; i<12; i++ ) { xr1[i] = shrt_time_r[11-i]; xi1[i] = shrt_time_i[11-i]; }
   for( i=12; i<M1; i++ ) { xr1[i] = shrt_time_r[75-i]; xi1[i] = shrt_time_i[75-i]; }

   for( i=0; i<NSPS; i++ ) {

      spr[i*L] = xr1[11];

      spr[i*L+1] = (xr1[21]+xr1[ 0])*itp[ 0] + (xr1[20]+xr1[ 1])*itp[ 2];
      spr[i*L+2] = (xr1[19]+xr1[ 2])*itp[ 4] + (xr1[18]+xr1[ 3])*itp[ 6];
      spr[i*L+3] = (xr1[17]+xr1[ 4])*itp[ 8] + (xr1[16]+xr1[ 5])*itp[10];
      spr[i*L+4] = (xr1[15]+xr1[ 6])*itp[12] + (xr1[14]+xr1[ 7])*itp[14];
      spr[i*L+5] = (xr1[13]+xr1[ 8])*itp[16] + (xr1[12]+xr1[ 9])*itp[18];
      spr[i*L+6] = xr1[10];

      spi[i*L] = xi1[11];

      spi[i*L+1] = (xi1[21]+xi1[ 0])*itp[ 0] + (xi1[20]+xi1[ 1])*itp[ 2];
      spi[i*L+2] = (xi1[19]+xi1[ 2])*itp[ 4] + (xi1[18]+xi1[ 3])*itp[ 6];
      spi[i*L+3] = (xi1[17]+xi1[ 4])*itp[ 8] + (xi1[16]+xi1[ 5])*itp[10];
      spi[i*L+4] = (xi1[15]+xi1[ 6])*itp[12] + (xi1[14]+xi1[ 7])*itp[14];
      spi[i*L+5] = (xi1[13]+xi1[ 8])*itp[16] + (xi1[12]+xi1[ 9])*itp[18];
      spi[i*L+6] = xi1[10];
```

```
      spi[i*L] = xii[11];

      spi[i*L+1]  = (xii[21]+xii[ 0])*itp[ 0] + (xii[20]+xii[ 1])*itp[ 2];
      spi[i*L+1] += (xii[19]+xii[ 2])*itp[ 4] + (xii[18]+xii[ 3])*itp[ 6];
      spi[i*L+1] += (xii[17]+xii[ 4])*itp[ 8] + (xii[16]+xii[ 5])*itp[10];
      spi[i*L+1] += (xii[15]+xii[ 6])*itp[12] + (xii[14]+xii[ 7])*itp[14];
      spi[i*L+1] += (xii[13]+xii[ 8])*itp[16] + (xii[12]+xii[ 9])*itp[18];
      spi[i*L+1] += (xii[11]+xii[10])*itp[20];

      for( j=M; j>0; j-- ) { xri[j] = xri[j-1]; xii[j] = xii[j-1]; }

      xri[0] = (i+12<NSPS)? shrt_time_r[i+12]:shrt_time_r[i-NSPS+12];
      xii[0] = (i+12<NSPS)? shrt_time_i[i+12]:shrt_time_i[i-NSPS+12];
   }
}

void load_long( float *lpr,float *lpi,float *r1r,float *r1i,float *r2r,float *r2i,float *itp  )
{
   float long_frec_r[NSPS] = {
      0 0,
     +1.0,-1.0,-1.0,+1.0,+1.0,-1.0,+1.0,-1.0,+1.0,-1 0,-1.0,-1.0,
     -1.0,-1.0,+1.0,+1.0,-1.0,-1.0,+1.0,-1.0,+1.0,-1.0,+1.0,+1.0,
     +1.0,+1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
      0.0,+1.0,+1.0,-1.0,-1.0,+1.0,+1.0,-1.0,+1.0,-1 0,+1.0,+1.0,
     +1.0,+1.0,+1.0,+1.0,-1.0,-1.0,+1.0,+1.0,-1.0,+1.0,-1.0,+1.0,
     +1.0,+1.0,+1.0
   };

   float long_frec_i[NSPS] = {
      0.0,
      0.0, 0.0, 0 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
      0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0 0, 0.0, 0.0,
      0 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
      0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
      0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
      0.0, 0.0, 0.0
   };

   float long_time_r[NSPS],long_time_i[NSPS],xri[M1],xii[M1];

   int i=0,j=0;

   ifft64( long_frec_r,long_frec_i,long_time_r,long_time_i,r1r,r1i,r2r,r2i );

   /* interpolation filtering ************************************** */

   for( i=0 ; i<12; i++ ) { xri[i] = long_time_r[11-i]; xii[i] = long_time_i[11-i]; }
   for( i=12; i<M1; i++ ) { xri[i] = long_time_r[75-i]; xii[i] = long_time_i[75-i]; }

   for( i=0; i<NSPS; i++ ) {

      lpr[i*L] = xri[11];

      lpr[i*L+1]  = (xri[21]+xri[ 0])*itp[ 0] + (xri[20]+xri[ 1])*itp[ 2];
      lpr[i*L+1] += (xri[19]+xri[ 2])*itp[ 4] + (xri[18]+xri[ 3])*itp[ 6];
      lpr[i*L+1] += (xri[17]+xri[ 4])*itp[ 8] + (xri[16]+xri[ 5])*itp[10];
      lpr[i*L+1] += (xri[15]+xri[ 6])*itp[12] + (xri[14]+xri[ 7])*itp[14];
      lpr[i*L+1] += (xri[13]+xri[ 8])*itp[16] + (xri[12]+xri[ 9])*itp[18];
      lpr[i*L+1] += (xri[11]+xri[10])*itp[20];

      lpi[i*L]   = xii[11];

      lpi[i*L+1]  = (xii[21]+xii[ 0])*itp[ 0] + (xii[20]+xii[ 1])*itp[ 2];
      lpi[i*L+1] += (xii[19]+xii[ 2])*itp[ 4] + (xii[18]+xii[ 3])*itp[ 6];
      lpi[i*L+1] += (xii[17]+xii[ 4])*itp[ 8] + (xii[16]+xii[ 5])*itp[10];
      lpi[i*L+1] += (xii[15]+xii[ 6])*itp[12] + (xii[14]+xii[ 7])*itp[14];
      lpi[i*L+1] += (xii[13]+xii[ 8])*itp[16] + (xii[12]+xii[ 9])*itp[18];
      lpi[i*L+1] += (xii[11]+xii[10])*itp[20];

      for( j=M; j>0; j-- ) { xri[j] = xri[j-1]; xii[j] = xii[j-1]; }

      xri[0] = (i+12<NSPS)? shrt_time_r[i+12]:shrt_time_r[i-NSPS+12];
      xii[0] = (i+12<NSPS)? shrt_time_i[i+12]:shrt_time_i[i-NSPS+12];
   }
}
```

```
      xri[0] = (i+12<NSPS)? long_time_r[i+12]:long_time_r[i-NSPS+12];
      xii[0] = (i+12<NSPS)? long_time_i[i+12]:long_time_i[i-NSPS+12];
   }
}

/* end of file ********************************************************* */
```

11.2 Programa de recepción

```
/* ofdmrx.c : ********************************************************

  Universidad Nacional Autonoma de Mexico, Facultad de Ingenieria

  OFDM (Orthogonal Frecuency Division Multiplexing) Reception System

  Parameters:

       in      : name of the in file
       out     : name of the out file
       bps     : bits per subcarrier, [ 1 (BPSK), 2 (QPSK), 4 (16QAM) ]
       gtime   : guard time enabler (1/4 of symbol time)    [ 1,0 ]
       longs   : number of long  preamble symbols           [ 1 ... 4 ]
       rphse   : random phase initializer enabler           [ 1,0 ]
       iphse   : initial carrier phase (% of ô)             [-1.0 ... 1.0 ]
       fdevt   : frequency deviation (in parts per million) [ 0.0 ... 50.0 ]
       fcsiz   : filtered correlations buffers size         [ 1 ... 5 ]
       [iqsep] : inph & quad separation signals output flag
       [bbitp] : inph & quad baseband interpolated signals output flag
       [baseb] : inph & quad baseband decimated signals output flag
       [corrl] : real & imag correlations output flag
       [fcorr] : real & imag filtered correlations output flag
       [fcpol] : magnitude & phase filtered correlations output flag
       [spect] : real & imag spectrums output flag
       [texto] : text output flag
       [spcmg] : long preamble spectrum magnitude (uncorrected & corrected)

  Notes:

       * Cyclic Extension stripping
       * Short & long preamble processing:
         Synchronization: Phase sync, Frec sync, Timing estimation
       * No Viterbi Decodification
       * No Bit Deinterleaving

  bps  = 1 (BPSK), 2 (QPSK), 4 (16QAM) [Bits Per Subcarrier]
  sps  = 64 [Subcarriers Per OFDM Symbol = IFFT size]
  aps  = 48 [Active Subcarriers Per OFDM Symbol]
  BpS  = aps*bps/8 = 6,12,24  [Bytes Per OFDM Symbol]

********************************************************************** */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

/* constants used for demapping & demodulation ********************** */

#define NSPS   64
#define NAPS   48
#define MAXMAP 16
#define MAXDAT 3

/* lowpass filter (used in synchronization) constants */
#define L    4
#define M    (2*L+1)
```

```
#define M1 22
#define M2 43

/* number of samples per symbol (interpolated) */
#define NSPSI NSPS*L
#define NSPS2 NSPSI+4

/* maximum size of the correlation filter */
#define MAX_COR_FIL 5

/* preambles detection thresholds */
#define SHRT_THRS 2.5
#define LONG_THRS 2.5

#define M_EST_D    40
#define PEAK_LTNC  25

/* auxiliar functions ***************************************************

  subcarrindx : load active & pilot subcarrier index tables
  ang_mapper  : creates the angle table (map) for the (2^bps) entries
  rot_angles  : calculate the rotation angles for the fft
  fft64       : performs the fft (64 points)
  interpol    : load interpolation (lowpass) filter taps
  load_shrt   : load short preamble symbol (in time domain)
  load_long   : load long  preamble symbol (in time domain)

************************************************************************ */

void subcarrindx( char *active,char *pilots );
void ang_mapper ( int bps,float *mapi, float *mapq );
void rot_angles ( float *r1r,float *r1i,float *r2r,float *r2i );
void fft64      ( float *xr ,float *xi ,float *Xr ,float *Xi,
                  float *r1r,float *r1i,float *r2r,float *r2i );
void interpol   ( float *itp );
void load_shrt ( float *spr,float *spi,float *itp );
void load_long ( float *lpr,float *lpi,float *itp );
void load_long_freq( float *lfr );

/* main function ******************************************************* */

int main( int argc,char *argv[] )
{
   FILE *in,*rx;
   FILE *i1rx,*q1rx,*i2rx,*q2rx,*i3rx,*q3rx,*i4rx,*q4rx;
   FILE *i5rx,*q5rx,*i6rx,*q6rx,*i7rx,*q7rx,*rsrx,*isrx;
   FILE *mags,*magl,*fcms,*fcml,*fcps,*fcpl,*inds,*indl;
   FILE *lprs,*lpis,*rsrc,*isrc,*smag;

   int  iqsep=0,basei=0,baseb=0;/* Output streams flags    */
   int  spect=0,texto=0,gtime=0;/* Output flags            */
   int  rphse=0;              /* Random phase enabler      */
   int  corrl=0;              /* Corr with preambles*/
   int  fcorr=0;              /* Filtered correlation      */
   int  fcpol=0;              /* Filt corr polar mode      */
   int  fcsiz=0;              /* Filt corr buff size*/
   char strng[5]="wb\0";      /* Auxiliar string    */

   unsigned int bps = 0;      /* Bits Per Subcarrier*/
   unsigned int BpS = 0;      /* Bytes Per Symbol   */
   unsigned int spS = 0;      /* Samples Per Symbol */

   float mapi[MAXMAP],mapq[MAXMAP],/* I-Q Constel Stack    */
   float Xr[NSPS],Xi[NSPS];   /* Frequency domain    */
   float xr[NSPS],xi[NSPS];   /* Time domain         */
   float r1r[NSPS],r1i[NSPS]; /* fft rotation angles      */
   float r2r[NSPS],r2i[NSPS]; /* fft rotation angles      */

   unsigned char data[MAXDAT],      /* data (uncoded) or   */
   int i=0,j=0,k=0,l=0,     /* counters            */
```

```
  char active[NAPS],pilots[4];      /* subcarrier indeces */

  float sumi = 0.0,sumq = 0.0;      /* filters outputs    */
  float sumiS= 0.0,sumqS= 0.0;      /* filters outputs bak*/
  float sumiL= 0.0,sumqL= 0.0;      /* filters outputs bak*/
  float rxx[NSPSI];               /* received symbol  */
  float gtm[NSPSI/4];           /* guard time buffer  */
  float bi1[NSPSI],bq1[NSPSI];      /* separation buffers */
  float bi2[   M2],bq2[   M2]; /* interpolation buffers*/
  float bi3[NSPS2],bq3[NSPS2];      /* filtered basebands */
  float itp[M1];                /* interpolation taps */

  int   longs=0;                /* long preamble symbols*/
  float spr[NSPSI],spi[NSPSI];      /* short pream buffer */
  float lpr[NSPSI],lpi[NSPSI];      /* long  pream buffer */
  float lfr[NSPS];              /* long  pream freq   */

  float sample=0.0;             /* auxiliar variable  */
  float fdevt =0.0;             /* frequency deviation*/
  float phase =0.0;             /* carrier phase      */
  float iphse =0.0;             /* initial carrier phase*/
  float phinc =0.0;             /* phase increment    */
  float fcmag =0.0;             /* filt corr magnitude*/
  float fcmagS=0.0;             /* filt corr magn bak */
  float fcmagL=0.0;             /* filt corr magn bak */
  float fcphs =0.0;             /* filt corr phase    */
  float phsac =0.0;             /* phase accumulator  */
  int   signS =0;               /* sign counter (phase)       */
  int   mestd =0;               /* maximum estimt delay       */
  int   mestd_=0;               /* maximum estimt delay       */
  int   nestm =0;               /* number of estimations*/
  int   brstd =0;               /* detected burst flag*/

  float fcsr[MAX_COR_FIL];  /* Filter Corr buffer for Short (Real)      */
  float fcsi[MAX_COR_FIL];  /* Filter Corr buffer for Short (Imag)      */
  float fclr[MAX_COR_FIL];  /* Filter Corr buffer for Long  (Real)      */
  float fcli[MAX_COR_FIL];  /* Filter Corr buffer for Long  (Imag)      */

  int   pkdts =0;               /* peak detected      */
  int   pkdtl =0;               /* peak detected      */
  int   pklts =0;               /* peak search latency*/
  int   pkltl =0;               /* peak search latency*/
  int   ind   =0;               /* sample index               */
  int   indS  =0;               /* sample auxiliar index*/
  int   indL  =0;               /* sample auxiliar index*/

  int   lpsrc =0;               /* long preamble search       */
  int   lpind =0;               /* long preamble index*/
  int   lpsym =0;               /* long preamble symbol       */
  int   nlest =0;               /* number of long estim       */

  int   vlddt =0;               /* valid data flag    */
  int   waits =0;               /* wait samp after pream*/

  float XR[NSPS],XI[NSPS];  /* Long Pream Freq Buff       */
  float Cr[NSPS],Ci[NSPS];  /* Correction Factors */
  float Cm = 0.0,Cp = 0.0;  /* Correction Factors */
  float Ar = 0.0,Ai = 0.0;  /* Auxiliar variables */

  int   spcmg =0;               /* spectrum magnitudes*/

/* get parameters ************************************************** */

  if( argc < 10 ) {
     printf( "ofdmrx: in out bps gtime longs rphse iphse fdevt fcsr:\n" );
     printf( "[iqsep base] baseb corrl fcorr fcvol spect [exto] [spcmg]" ); return( 1 );
  } else {
     bps     = atoi( argv[ 3] );       /* Bits Per Subcarrier : 1,2,4       */
     gtime   = atoi( argv[ 4] );       /* Guard time samples                */
     longs   = atoi( argv[ 5] );       /* Long preamble symbols             */
     rphse   = atoi( argv[ 6] );       /* Receiver phase noise              */
```

```
      iphse = atof( argv[ 7] );      /* Initial carrier phase (rad)       */
      fdevt = atof( argv[ 8] );      /* Frequency deviation (in ppm)      */
      fcsiz = atoi( argv[ 9] );      /* Filtered correlations buffers*/
   }
   if( argc > 10 ) {
      iqsep = atoi( argv[10] );
      basei = atoi( argv[11] );
      baseb = atoi( argv[12] );
      corrl = atoi( argv[13] );
      fcorr = atoi( argv[14] );
      fcpol = atoi( argv[15] );
      spect = atoi( argv[16] );
      texto = atoi( argv[17] );      /* text output */
   }
   if( argc > 18 ) {
      spcmg = 1;
   }

   if( texto ) strng[1] = 't';

/* open files to read and write ************************************* */

   if( (in=fopen( argv[1],"rb" )) == NULL ) {
      printf( "Can't open file : %s",argv[1] ); return( 1 ); }
   if( (rx=fopen( argv[2],"wb" )) == NULL ) {
      printf( "Can't open file : %s",argv[2] ); return( 1 ); }
   if( iqsep ) {
      if( (i1rx=fopen( "i1rx",strng )) == NULL ) {
        printf( "Can't open file : i1rx" ); return( 1 ); }
      if( (q1rx=fopen( "q1rx",strng )) == NULL ) {
        printf( "Can't open file : q1rx" ); return( 1 ); }
   }
   if( basei ) {
      if( (i2rx=fopen( "i2rx",strng )) == NULL ) {
        printf( "Can't open file : i2rx" ); return( 1 ); }
      if( (q2rx=fopen( "q2rx",strng )) == NULL ) {
        printf( "Can't open file : q2rx" ); return( 1 ); }
   }
   if( baseb ) {
      if( (i3rx=fopen( "i3rx",strng )) == NULL ) {
        printf( "Can't open file : i3rx" ); return( 1 ); }
      if( (q3rx=fopen( "q3rx",strng )) == NULL ) {
        printf( "Can't open file : q3rx" ); return( 1 ); }
   }
   if( corrl ) {
      if( (i4rx=fopen( "i4rx",strng )) == NULL ) { /* for short real      */
        printf( "Can't open file : i4rx" ); return( 1 ); }
      if( (q4rx=fopen( "q4rx",strng )) == NULL ) { /* for short imag      */
        printf( "Can't open file : q4rx" ); return( 1 ); }
      if( (i5rx=fopen( "i5rx",strng )) == NULL ) { /* for long real       */
        printf( "Can't open file : i5rx" ); return( 1 ); }
      if( (q5rx=fopen( "q5rx",strng )) == NULL ) { /* for long imag       */
        printf( "Can't open file : q5rx" ); return( 1 ); }
   }
   if( fcorr ) {
      if( (i6rx=fopen( "i6rx",strng )) == NULL ) { /* for short real      */
        printf( "Can't open file : i6rx" ), return( 1 ); }
      if( (q6rx=fopen( "q6rx",strng )) == NULL ) { /* for short imag      */
        printf( "Can't open file : q6rx" ); return( 1 ); }
      if( (i7rx=fopen( "i7rx",strng )) == NULL ) { /* for long real       */
        printf( "Can't open file : i7rx" ); return( 1 ); }
      if( (q7rx fopen( "q7rx",strng )) == NULL ) { /* for long imag       */
        printf( "Can't open file : q7rx" ); return( 1 ), }
   }
   if( fcpol ) {
      if( (mag= fopen( "mag=",strng ))    NULL ) { /* for short magnitude */
        printf( "Can't open file : mag=" ); return( 1 ), }
      if( (mag' fopen( "mag!",strng ))    NULL ) { /*  or long  magnitude */
        printf( "Can't open file : mag!" ); return( 1 ); }
      if( (fcm, fopen( "fcm,","wt" ))     NULL ) { /* for short magnitude */
        printf( "Can't open file : fcm," ); return( 1 );
```

```
      if( (fcml=fopen( "fcml","wt"  )) == NULL ) { /* for long  magnitude */
        printf( "Can't open file : fcml" ); return( 1 ); }
      if( (fcps=fopen( "fcps","wt"  )) == NULL ) { /* for short phase     */
        printf( "Can't open file : fcps" ); return( 1 ); }
      if( (fcpl=fopen( "fcpl","wt"  )) == NULL ) { /* for long  phase     */
        printf( "Can't open file : fcpl" ); return( 1 ); }
      if( (inds=fopen( "inds","wt"  )) == NULL ) { /* index short output*/
        printf( "Can't open file : inds" ); return( 1 ); }
      if( (indl=fopen( "indl","wt"  )) == NULL ) { /* index long  output*/
        printf( "Can't open file : indl" ); return( 1 ); }
   }
   if( spect ) {
      if( (rsrx=fopen( "rsrx",strng )) == NULL ) {
        printf( "Can't open file : rsrx" ); return( 1 ); }
      if( (isrx=fopen( "isrx",strng )) == NULL ) {
        printf( "Can't open file : isrx" ); return( 1 ); }
      if( (lprs=fopen( "lprs",strng )) == NULL ) {
        printf( "Can't open file : lprs" ); return( 1 ); }
      if( (lpis=fopen( "lpis",strng )) == NULL ) {
        printf( "Can't open file : lpis" ); return( 1 ); }
      if( (rsrc=fopen( "rsrc",strng )) == NULL ) {
        printf( "Can't open file : rsrc" ); return( 1 ); }
      if( (isrc=fopen( "isrc",strng )) == NULL ) {
        printf( "Can't open file : isrc" ); return( 1 ); }
   }
   if( spcmg ) {
      if( (smag=fopen( argv[18],strng )) == NULL ) {
        printf( "Can't open file : %s",argv[18] ); return( 1 ); }
   }

/* tables & constants initialization ********************************** */

   interpol( itp );           /* interpolating filter taps initializ      */
   subcarrindx( active,pilots );/* subcarriers indeces initialization     */
   rot_angles( r1r,r1i,r2r,r2i);/* ifft rotation angles initialization  */
   BpS = NAPS*bps/8;          /* Bytes Per OFDM Symbol                    */
   spS = NSPSI;                     /* Samples Per OFDM Symbol            */
   phinc = M_PI/2.0+fdevt/4.0e6;/* Phase increment calculation            */
   mestd = M_EST_D;           /* Maximum estimation delay                 */
   mestd_= mestd;             /* Maximum estimation delay                 */

   load_long_freq( lfr );     /* load long preamble in freq domain */

   for( i=0; i<NSPSI; i++ ) { bi1[i]=0.0; bq1[i]=0.0; bi3[i]=0.0; bq3[i]=0.0; }
   for( i=0; i<NSPS;  i++ ) { xr [i]=0.0; xi [i]=0.0; Xr [i]=0.0; Xi [i]=0.0; XR [i]=0.0; XI
[i]=0.0; Cr [i]=0.0; Ci [i]=0.0; }
   for( i=0; i<M2;    i++ ) { bi2[i]=0.0; bq2[i]=0.0; }

   load_shrt( spr,spi,itp );  /* load short preamble symbol */
   load_long( lpr,lpi,itp );  /* load long  preamble symbol */

   for( i=0; i<MAX_COR_FIL; i++ ) { fcsr[i]=0.0;fcsi[i]=0.0;fclr[i]=0.0;fcli[i]=0.0; }

/* burst preamble detection ******************************************* */

   randomize();

   phase = ( rphse )? 2.0*M_PI*((float)rand()/(float)RAND_MAX)-M_PI:iphse*M_PI;

   fread( &sample,4,1,in );

   while( !vlddt ) { /* bursts preamble detector ********************** */

   /* shift for lowpass filtering ************************************ */

      for( i=M2-1; i>0; i-- ) { bi2[i]   bi2[i-1]; bq2[i]   bq2[i-1], }
      bi2[0]   cos( phase )*sample; bq2[0]   sin( phase )*sample;

      if( iqcpl ) { /* with quadrature component ******************** */
        fprintf( fcml,"%d\n",mi2[0], fprintf( fcps,"%d\n",bi2[0] );
        else        { fprintf( fcml,"%d\n",bq2[0], fprintf( fcps,"%d\n",bq2[0] );
```

```
    }

  /* lowpass filtering ********************************************** */

    for( i=0; i<M0; i++ ) {  sumi += (bi2[i]+bi2[M2-i-1])*itp[i]; sumq += (bq2[i]+bq2[M2-i-
1])*itp[i]; }
    sumi += (bi2[i]*itp[i]); sumq += (bq2[i]*itp[i]);

    if( basei ) { /* write interpolated baseband signals ************ */
      if( texto ) { fprintf( i2rx,"%f\n",sumi ); fprintf( q2rx,"%f\n",sumq ); }
      else        { fwrite( &sumi,4,1,i2rx );fwrite( &sumq,4,1,q2rx ); }
    }

  /* shift for correlations processing ****************************** */

    for( i=NSPS2-1; i>0; i-- ) { bi3[i] = bi3[i-1]; bq3[i] = bq3[i-1]; }
    bi3[0] = sumi; bq3[0] = sumq; sumi = 0.0; sumq = 0.0;

    if( !lpsrc ) { /* SHORT PREAMBLE PROCESSING ******************** */

    /* complex correlation with short preamble ********************** */

      for( i=0; i<NSPSI; i++ ) { sumi += (bi3[i]*spr[NSPSI-i]+bq3[i]*spi[NSPSI-i]); sumq +=
(bi3[i]*spi[NSPSI-i]-bq3[i]*spr[NSPSI-i]); }

      if( corrl ) { /* write correlation processing outputs ******** */
        if( texto ) { fprintf( i4rx,"%f\n",sumi ); fprintf( q4rx,"%f\n",sumq ); }
        else        { fwrite( &sumi,4,1,i4rx );fwrite( &sumq,4,1,q4rx ); }
      }

    /* shift for filter the correlations outputs (short preamble) *** */

      for( i=fcsiz; i>0; i-- ) { fcsr[i] = fcsr[i-1]; fcsi[i] = fcsi[i-1]; }
      fcsr[0] = sumi; fcsi[0] = sumq; sumi = 0.0; sumq = 0.0;

    /* correlation outputs filtering (for short preamble) *********** */

      for( i=0; i<fcsiz; i++ ) { sumi +=
(fcsr[i]*fcsr[i]*fcsr[i]+fcsr[i+1]*fcsr[i+1]*fcsr[i+1]); sumq +=
(fcsi[i]*fcsi[i]*fcsi[i]+fcsi[i+1]*fcsi[i+1]*fcsi[i+1]); }

      if( fcorr ) { /* write filtered correlation outputs ********** */
        if( texto ) { fprintf( i6rx,"%f\n",sumi ); fprintf( q6rx,"%f\n",sumq ); }
        else        { fwrite( &sumi,4,1,i6rx );fwrite( &sumq,4,1,q6rx ); }
      }

      fcmag = sqrt( sumi*sumi + sumq*sumq );

      if( fcpol ) { /* write filtered correlation magnitude ******** */
        if( texto ) fprintf( mags,"%f\n",fcmag );
        else        fwrite( &fcmag,4,1,mags );
      }

      if( brstd ) mestd--;

      if( fcmag > SHRT_THRS ) { /* peak processing ***************** */
        brstd = 1;
        if( !pklts ) {
          if( fcmag > fcmagS ) {

            fcmagS = fcmag; sumiS = fcsr[1]; sumqS = fcsi[1]; indS = ind-1, pkdts = 1;

          } else { /* peak found ********************************** */

            if( sumiS    0.0 ) sumiS = 0.00001;
            fcphs   atan2( -sumiS,-sumqS ),
            if( fcpol ) { /* filtered correlation output : polar1 */
              fprintf( fcmag," \n",fcmagS ); fprintf( fcphs," \n",fcphs ); fprintf(
"%f"," \n", pkdts );
            }
            ...                    ...;  ...
```

```
                phsac += fabs(fcphs); if( fcphs>0.0 ) signS++; else signS--;
                nestm++; mestd = mestd_;
              }
           } else {
              pklts--;
           }
        } else {
           if( pklts ) pklts--;
           if( pkdts ) { /* peak found ****************************** */

              if( sumqS == 0.0 ) sumqS = 0.00001;
              fcphs = atan2( -sumiS,-sumqS );
              if( fcpol ) { /* filtered correlation outputs (polar) ** */
                fprintf( fcms,"%f\n",fcmagS ); fprintf( fcps,"%f\n",fcphs ); fprintf(
inds,"%d\n",indS );
              }
              pklts = PEAK_LTNC; pkdts = 0; fcmagS = 0.0;
              phsac += fabs(fcphs); if( fcphs>0.0 ) signS++; else signS--;
              nestm++; mestd = mestd_;
           }
        }

        if( !mestd && brstd ) { /* short preamble is finished ******** */
           if( signS<0 ) phsac=(-phsac);
           phase -= (phsac/(float)nestm);
           if( phase > 2.0*M_PI ) phase -= (2.0*M_PI);
           phsac = 0.0; nestm = 0; mestd = mestd_; brstd = 0;
           lpsrc = 1; lpind = NSPSI;
        }

     } else { /* LONG PREAMBLE PROCESSING ***************************** */

     /* complex correlation with long preamble ********************** */

        for( i=0; i<NSPSI; i++ ) { sumi += (bi3[i]*lpr[NSPSI-i]+bq3[i]*lpi[NSPSI-i]); sumq +=
(bi3[i]*lpi[NSPSI-i]-bq3[i]*lpr[NSPSI-i]); }

        if( corrl ) { /* write correlation processing outputs ******** */
           if( texto ) { fprintf( i5rx,"%f\n",sumi ); fprintf( q5rx,"%f\n",sumq ); }
           else        { fwrite( &sumi,4,1,i5rx );fwrite( &sumq,4,1,q5rx ); }
        }

     /* shift for filter the correlations outputs (long preamble) **** */

        for( i=fcsiz; i>0; i-- ) { fclr[i] = fclr[i-1]; fcli[i] = fcli[i-1]; }
        fclr[0] = sumi; fcli[0] = sumq; sumi = 0.0; sumq = 0.0;

     /* correlation outputs filtering (for long preamble) ************ */

        for( i=0; i<fcsiz; i++ ) { sumi +=
(fclr[i]*fclr[i]*fclr[i]+fclr[i+1]*fclr[i+1]*fclr[i+1]); sumq +=
(fcli[i]*fcli[i]*fcli[i]+fcli[i+1]*fcli[i+1]*fcli[i+1]); }

        if( fcorr ) { /* write filtered correlation outputs ********** */
           if( texto ) { fprintf( i7rx,"%f\n",sumi ); fprintf( q7rx,"%f\n",sumq ); }
           else        { fwrite( &sumi,4,1,i7rx );fwrite( &sumq,4,1,q7rx ); }
        }

        fcmag = sqrt( sumi*sumi + sumq*sumq );

        if( fcpol ) { /* write filtered correlation outputs (polar) ** */
           if( texto ) fprintf( magl,"%f\n",fcmag );
           else        fwrite( &fcmag,4,1,magl );
        }

        if( fcmag > LONG_THRS ) { /* peak processing ***************** */
           if( !pklt ) {
              if( fcmag > fcmagL ) {

                 "%f\n",fcmag; sumi[ ] = 0.0; sumq[ ] = fcli[ ], sumi[ ] = fclr[ ], sumi[ ] = 0.0; sumq[ ] = 0.0;
```

```
            } else { /* peak found ******************************** */

               if( sumiL == 0.0 ) sumiL = 0.00001;
               fcphs = atan2( -sumqL,sumiL );
               if( fcpol ) { /* filtered correlation outputs (polar) */
                  fprintf( fcml,"%f\n",fcmagL ); fprintf( fcpl,"%f\n",fcphs ); fprintf(
indl,"%d\n",indL );
               }
               pkltl = PEAK_LTNC; pkdtl = 0; fcmagL = 0.0;
               if( lpsym ) {
                  for( j=0; j<NSPS; j++ ) { xr[j] = bi3[NSPSI+2-j*2]; xi[j] = bq3[NSPSI+2-j*2];
}
                  fft64( xr,xi,Xr,Xi,rlr,rli,r2r,r2i );
                  for( j=0; j<NSPS; j++ ) { XR[j] += Xr[j]; XI[j] += Xi[j]; }
                  nlest++;
               }
               lpsym++; lpind = NSPSI + 5; waits = 7;
            }
         } else {
            pkltl--;
         }

      } else {

         if( pkltl ) pkltl--;
         if( pkdtl ) { /* peak found ******************************** */

            if( sumiL == 0.0 ) sumiL = 0.00001;
            fcphs = atan2( -sumqL,sumiL );
            if( fcpol ) { /* filtered correlation outputs (polar) ** */
               fprintf( fcml,"%f\n",fcmagL ); fprintf( fcpl,"%f\n",fcphs ); fprintf(
indl,"%d\n",indL );
            }
            pkltl = PEAK_LTNC; pkdtl = 0; fcmagL = 0.0;
            if( lpsym ) {
               for( j=0; j<NSPS; j++ ) { xr[j] = bi3[NSPSI+2-i*2]; xi[j] = bq3[NSPSI+2-i*2]; }
               fft64( xr,xi,Xr,Xi,rlr,rli,r2r,r2i );
               for( j=0; j<NSPS; j++ ) { XR[j] += Xr[j]; XI[j] += Xi[j]; }
               nlest++;
            }
            lpsym++; lpind = NSPSI + 5; waits = 7;
         }
      }

      if( lpind ) { lpind--; } else { lpsym++; lpind = NSPSI; waits = 2; }

      if( lpsym == longs ) { /* long preamble finished ************* */

         if( nlest ) {

            for( j=0; j<NSPS; j++ ) { XR[j] /= (float)nlest; XI[j] /= (float)nlest; }

            if( spect ) { // write average preamble spectrum **********
               if( texto ) for( i=0; i<NSPS; i++ ) { fprintf( lprs,"%f\n",XR[i] ); fprintf(
lpis,"%f\n",XI[i] ); }
               else        { fwrite( XR,4,NSPS,lprs ); fwrite( XI,4,NSPS,lpis ); }
            }

            if( spcmg ) { // write spectrum magnitude ****************
               for( j=0; j<NSPS; j++ ) {
                  Cm = sqrt( XR[j]*XR[j] + XI[j]*XI[j] );
                  if( texto ) fprintf( smag,"%f\n",Cm );
                  else        fwrite( &Cm,4,1,smag );
               }
               for( ; 0; j NSPS, ) + ) {
                  Cm   [0]([1][1]/sqrt( XR[j]*XR[j] + XI[j]*XI[j] );
                  if( texto ) fprintf( smag," f\n",Cm );
                  else        fwrite( &Cm, 4, 1, smag );
               }
            }
```

```
            vlddt = 1; /* activate the valid data flag */

            for( j=0; j<NSPS; j++ ) {

               Cm = lfr[j]/sqrt( XR[j]*XR[j] + XI[j]*XI[j] );
               if( XR[j] == 0.0 ) XR[j] = 0.00001;
               Cp = -atan2( XI[j],XR[j] );
               Cr[j] = Cm*cos(Cp); Ci[j] = Cm*sin(Cp);

               if( spect || spcmg ) {
                  Ar = XR[j]; Ai = XI[j];
                  XR[j] = Ar*Cr[j] - Ai*Ci[j];
                  XI[j] = Ar*Ci[j] + Ai*Cr[j];/**/
               }
            }

            if( spect ) { // write real & imag spectrums **************
               if( texto ) for( i=0; i<NSPS; i++ ) { fprintf( lprs,"%f\n",XR[i] ); fprintf(
lpis,"%f\n",XI[i] ); }
               else        { fwrite( XR,4,NSPS,lprs ); fwrite( XI,4,NSPS,lpis ); }
            }

            if( spcmg ) { // write spectrum magnitude *****************
               for( j=0; j<NSPS; j++ ) {
                  Cm = sqrt( XR[j]*XR[j] + XI[j]*XI[j] );
                  if( texto ) fprintf( smag,"%f\n",Cm );
                  else        fwrite( &Cm,4,1,smag );
               }
               fclose( in ); fclose( rx ); fclose( smag );
               return( 0 );
            }

         } else {
            sprintf( datau,"ERROR0ERROR1ERROR2ERROR3" );
            fwrite( datau,1,BpS,rx ); return( 1 );
         }
      }
   }

/* variables reinitialization ************************************ */

   sumi = 0.0; sumq = 0.0;

   phase += phinc;
   if( phase > 2.0*M_PI ) phase -= (2.0*M_PI);

   fread( &sample,4,1,in ); ind++;
}

/* end of burst preamble detection ******************************** */

/* data demodulation ********************************************** */

phase += (phinc*(float)waits);
while( phase > 2.0*M_PI ) phase -= (2.0*M_PI);

if( gtime ) fread( gtm,4,waits,in ); /* strip the cyclic extension     */
/*  if( gtime ) fread( gtm,4,spS/4,in ); /* strip the cyclic extension   */
rs = fread( rxx,4,spS,in );              /* read one ofdm symbol              */

do {

   for( i=0; i<NSPSI; i++ ) { /* i & q separation */

      phase += phinc;
      if( phase > 2.0*M_PI ) phase -= (2.0*M_PI);

      xr[i] = rxx[i]*cos(phase), bg[i] = rxx[i]*sin(phase);

   }
```

```
      if( iqsep ) { /* write i & q separated signals ****************** */
        if( texto ) for( i=0; i<NSPSI; i++ ) { fprintf( i1rx,"%f\n",bi1[i] ); fprintf(
q1rx,"%f\n",bq1[i] ); }
        else        { fwrite( bi1,4,NSPSI,i1rx ); fwrite( bq1,4,NSPSI,q1rx ); }
      }

   /* low pass filtering ********************************************** */

      for( i= 0; i<M1; i++ ) { bi2[i] = bi1[M0 -i]; bq2[i] = bq1[M0-i];  }
      for( i=M1; i<M2; i++ ) { bi2[i] = bi1[149-i]; bq2[i] = bq1[149-i]; }

      for( i=0; i<NSPSI; i++ ) {

        for( j=0; j<M0; j++ ) {  sumi += (bi2[j]+bi2[M2-j-1])*itp[j]; sumq += (bq2[j]+bq2[M2-j-
1])*itp[j]; }
        sumi += (bi2[j]*itp[j]); sumq += (bq2[j]*itp[j]);

        bi3[i] = sumi; bq3[i] = sumq;
        sumi   =  0.0; sumq   =  0.0;

        for( j=M2-1; j>0; j-- ) { bi2[j] = bi2[j-1]; bq2[j] = bq2[j-1]; }

        bi2[0] = (i+M1<NSPSI)? bi1[i+M1]:bi1[i-NSPSI+M1];
        bq2[0] = (i+M1<NSPSI)? bq1[i+M1]:bq1[i-NSPSI+M1];
      }

   /* end low pass filtering ****************************************** */

      if( basei ) { /* write interpolated baseband signals ************ */
        if( texto ) for( i=0; i<NSPSI; i++ ) { fprintf( i2rx,"%f\n",bi3[i] ); fprintf(
q2rx,"%f\n",bq3[i] ); }
        else        { fwrite( bi3,4,NSPSI,i2rx ); fwrite( bq3,4,NSPSI,q2rx ); }
      }

   /* baseband decimation ********************************************* */

      for( i=0; i<NSPS; i++ ) { xr[i] = bi3[i*L]; xi[i] = bq3[i*L]; }

      if( baseb ) { /* write decimated baseband signals *************** */
        if( texto ) for( i=0; i<NSPS; i++ ) { fprintf( i3rx,"%f\n",xr[i] ); fprintf(
q3rx,"%f\n",xi[i] ); }
        else        { fwrite( xr,4,NSPS,i3rx ); fwrite( xi,4,NSPS,q3rx ); }
      }

   /* transform from time to frequency ******************************** */

      fft64( xr,xi,Xr,Xi,r1r,r1i,r2r,r2i );

      if( spect ) { /* write real & imag spectrums ******************** */
        if( texto ) for( i=0; i<NSPS; i++ ) { fprintf( rsrx,"%f\n",Xr[i] ); fprintf(
isrx,"%f\n",Xi[i] ); }
        else        { fwrite( Xr,4,NSPS,rsrx ); fwrite( Xi,4,NSPS,isrx ); }
      }

      for( j=0; j<NAPS; j++ ) { // Constellation rotation
        Ar = Xr[active[j]]; Ai = Xi[active[j]];
        Xr[active[j]] = Ar*Cr[active[j]] - Ai*Ci[active[j]];
        Xi[active[j]] = Ar*Ci[active[j]] + Ai*Cr[active[j]];
      }

      if( spect ) { // write real & imag spectrums ***************
        if( texto ) for( i=0; i<NSPS; i++ ) { fprintf( rsrc,"%f\n",Xr[i] ); fprintf(
isrc,"%f\n",Xi[i] ); }
        else        { fwrite( Xr,4,NSPS,rsrc ); fwrite( Xi,4,NSPS,isrc ); }
      }

   /* subcarriers demapping ******************************************* */

      for( i=0; i<NSPS; i++ ) {
        if(i==0) {
        }
```

```
            datau[i]>>=bps;
            switch( bps ) { /* demapping ***************************** */
               case 1: /* BPSK demapping */
                  if( Xr[active[k]]>0.0 ) datau[i] |= 0x80; break;
               case 2: /* QPSK demapping */
                  if( Xr[active[k]]>0.0 ) datau[i] |= 0x40;
                  if( Xi[active[k]]>0.0 ) datau[i] |= 0x80; break;
               case 4: /* 16-QAM demapping */
                  if( Xr[active[k]] > 0.0 )  { datau[i] |= 0x10;
                    if(Xr[active[k]] < 0.6325 )datau[i] |= 0x20;
                  } else
                    if( Xr[active[k]]>-0.6325 )datau[i] |= 0x20;

                  if( Xi[active[k]] > 0.0 )  { datau[i] |= 0x40;
                    if( Xi[active[k]]< 0.6325 )datau[i] |= 0x80;
                  } else
                    if( Xi[active[k]]>-0.6325 )datau[i] |= 0x80;
               break;
            }
            k++;
         }
      }

      fwrite( datau,1,BpS,rx );      /* write one decoded symbol   */

      if( gtime ) fread( gtm,4,spS/4,in ); /* strip the cyclic extension*/
      rs = fread( rxx,4,spS,in );       /* read one ofdm symbol    */

   } while( rs );

/* close files ************************************************** */

   fclose( in ); fclose( rx );

   if( iqsep ) { fclose( i1rx ); fclose( q1rx ); }
   if( base1 ) { fclose( i2rx ); fclose( q2rx ); }
   if( baseb ) { fclose( i3rx ); fclose( q3rx ); }
   if( corr1 ) { fclose( i4rx ); fclose( q4rx ); fclose( i5rx ); fclose( q5rx ); }
   if( fcorr ) { fclose( i6rx ); fclose( q6rx ); fclose( i7rx ); fclose( q7rx ); }
   if( fcpol ) { fclose( mags ); fclose( magl ); fclose( fcms ); fclose( fcml );
                 fclose( fcps ); fclose( fcpl ); fclose( inds ); fclose( indl ); }
   if( spect ) { fclose( rsrx ); fclose( isrx ); fclose( lprs ); fclose( lpis );
                 fclose( rsrc ); fclose( isrc ); }

   return( 0 ); /* No error, finished o.k. */
}

/* auxiliar functions ******************************************** */

void subcarrindx( char *active,char *pilots )
{
   char activ[NAPS] = {
      38,39,40,41,42,
      44,45,46,47,48,49,50,51,52,53,54,55,56,
      58,59,60,61,62,63,
      1 ,2 ,3 ,4 ,5 ,6 ,
      8 ,9 ,10,11,12,13,14,15,16,17,18,19,20,
      22,23,24,25,26
   };
   char pilot[4] = { 43,57,7,21 };

   int i;

   for( i=0; i<NAPS; i++ ) active[i]   activ[i];
   for( i 0; i<4; i++ )    pilots[i]   pilot[i];


void rot_angles( float *ir,float *ii,float *qr,float *qi )

   float angle  ( ***ii/a  );
      ,i,
```

```
   char  a[NSPS] = {
      0,0,0,0,0,4,8,12,0,8,16,24,0,12,24,36,0,1,2,3,0,5,10,15,0,9,18,27,
      0,13,26,39,0,2,4,6,0,6,12,18,0,10,20,30,0,14,28,42,0,3,6,9,0,7,14,
      21,0,11,22,33,0,15,30,45
   };

   /* rotation factors calculation */
   for( l=0; l<4; l++ ) {
      for( m=0; m<4; m++ ) {
        for( n=0; n<4; n++ ) {
           r1r[o] = cos( angle*(float)(l*m*4) );
           r1i[o] = sin( angle*(float)(l*m*4) );
           r2r[o] = cos( angle*(float)a[o] );
           r2i[o] = sin( angle*(float)a[o] );
           o++;
        }
      }
   }
}

void radix4b( float *x0r,float *x0i,float *x1r,float *x1i,
              float *x2r,float *x2i,float *x3r,float *x3i )
{
   float X0r,X0i,X1r,X1i,X2r,X2i,X3r,X3i;

   X0r = *x0r+*x1r+*x2r+*x3r; X0i = *x0i+*x1i+*x2i+*x3i;
   X1r = *x0r+*x1i-*x2r-*x3i; X1i = *x0i-*x1r-*x2i+*x3r;
   X2r = *x0r-*x1r+*x2r-*x3r; X2i = *x0i-*x1i+*x2i-*x3i;
   X3r = *x0r-*x1i-*x2r+*x3i; X3i = *x0i+*x1r-*x2i-*x3r;

   *x0r = X0r; *x0i = X0i;
   *x1r = X1r; *x1i = X1i;
   *x2r = X2r; *x2i = X2i;
   *x3r = X3r; *x3i = X3i;
}

void fft64( float *xr ,float *xi ,float *Xr ,float *Xi,
            float *r1r,float *r1i,float *r2r,float *r2i )
{
   float xx;
   char l,m,n,a[16] = { 0,4,8,12,1,5,9,13,2,6,10,14,3,7,11,15 };

   /* radix 4, firts level  */
   for( l=0; l<16; l++ ) {
      radix4b( xr+l,xi+l,xr+l+16,xi+l+16,xr+l+32,xi+l+32,xr+l+48,xi+l+48 ),
      for( m=0; m<NSPS; m+=16 ) {
        xx      = xr[l+m]*r1r[l+m] - xi[l+m]*r1i[l+m]; /* rotation */
        xi[l+m] = xr[l+m]*r1i[l+m] + xi[l+m]*r1r[l+m]; /* rotation */
        xr[l+m] = xx;
      }
   }
   /* radix 4, second level */
   for( l=0; l<16; l++ ) {
      m=16*(l/4)+(l%4);
      radix4b( xr+m,xi+m,xr+m+4,xi+m+4,xr+m+8,xi+m+8,xr+m+12,xi+m+12 );
      for( n=0; n<16; n+=4 ) {
        xx      = xr[m+n]*r2r[m+n] - xi[m+n]*r2i[m+n]; /* rotation */
        xi[m+n] = xr[m+n]*r2i[m+n] + xi[m+n]*r2r[m+n]; /* rotation */
        xr[m+n] = xx;
      }
   }
   /* radix 4, third level */
   for( l=0; l<NSPS; l+=4 ) {
      radix4b( xr+l,xi+l,xr+l+1,xi+l+1,xr+l+2,xi+l+2,xr+l+3,xi+l+3 ),
      for( m=0; m<4; m++ ) {
        xr[a[l/4]+m*16]   = xr[l+m];
        xi[a[l/4]+m*16]   = xi[l+m];
      }
```

```
void interpol( float *itp )
{
   float aux[M1] = {
       0.0001,0.0000,-0.0006,0.0000,0.0020,0.0000,-0.0049,0.0000,
       0.0104,0.0000,-0.0197,0.0000,0.0349,0.0000,-0.0598,0.0000,
       0.1030,0.0000,-0.1968,0.0000,0.6313,1.0000
   };
   int i;

   for( i=0; i<M1; i++ ) itp[i] = aux[i];
}

/* functions to generate the short & long preambles ****************** */

void rot_angles_i( float *r1r,float *r1i,float *r2r,float *r2i )
{
   float angle=(2.0*M_PI)/64.0;
   char  l=0,m=0,n=0,o=0;
   char  a[NSPS] = {
      0,0,0,0,0,4,8,12,0,8,16,24,0,12,24,36,0,1,2,3,0,5,10,15,0,9,18,27,
      0,13,26,39,0,2,4,6,0,6,12,18,0,10,20,30,0,14,28,42,0,3,6,9,0,7,14,
      21,0,11,22,33,0,15,30,45
   };

   /* rotation factors calculation */
   for( l=0; l<4; l++ ) {
      for( m=0; m<4; m++ ) {
        for( n=0; n<4; n++ ) {
           r1r[o] = cos( angle*(float)(l*m*4) );
           r1i[o] = sin( angle*(float)(l*m*4) );
           r2r[o] = cos( angle*(float)a[o] );
           r2i[o] = sin( angle*(float)a[o] );
           o++;
        }
      }
   }
}

void radix4bi( float *X0r,float *X0i,float *X1r,float *X1i,
               float *X2r,float *X2i,float *X3r,float *X3i )
{
   float x0r,x0i,x1r,x1i,x2r,x2i,x3r,x3i;

   x0r = *X0r+*X1r+*X2r+*X3r; x0i = *X0i+*X1i+*X2i+*X3i;
   x1r = *X0r-*X1i-*X2r+*X3i; x1i = *X0i+*X1r-*X2i-*X3r;
   x2r = *X0r-*X1r+*X2r-*X3r; x2i = *X0i-*X1i+*X2i-*X3i;
   x3r = *X0r+*X1i-*X2r-*X3i; x3i = *X0i-*X1r-*X2i+*X3r;

   *X0r = x0r; *X0i = x0i;
   *X1r = x1r; *X1i = x1i;
   *X2r = x2r; *X2i = x2i;
   *X3r = x3r; *X3i = x3i;
}

void ifft64( float *Xr ,float *Xi ,float *xr ,float *xi,
             float *r1r,float *r1i,float *r2r,float *r2i )
{
   float XX;
   char l,m,n,a[16] = { 0,4,8,12,1,5,9,13,2,6,10,14,3,7,11,15 };

   /* radix 4, firts level  */
   for( l=0; l<16; l++ ) {
      radix4bi( Xr+l,Xi+l,Xr+l+16,Xi+l+16,Xr+l+32,Xi+l+32,Xr+l+48,Xi+l+48 );
      for( m=0; m<NSPS; m+=16 ) {
        XX      = Xr[l+m]*r1r[l+m] - Xi[l+m]*r1i[l+m]; /* rotation */
        Xi[l+m] = Xr[l+m]*r1i[l+m] + Xi[l+m]*r1r[l+m]; /* rotation */
        Xr[l+m] = XX;
      }
   }
   /* ... */
```

```
      m=16*(l/4)+(l%4);
      radix4bi( Xr+m,Xi+m,Xr+m+4,Xi+m+4,Xr+m+8,Xi+m+8,Xr+m+12,Xi+m+12 );
      for( n=0; n<16; n+=4 ) {
        XX      = Xr[m+n]*r2r[m+n] - Xi[m+n]*r2i[m+n]; /* rotation */
        Xi[m+n] = Xr[m+n]*r2i[m+n] + Xi[m+n]*r2r[m+n]; /* rotation */
        Xr[m+n] = XX;
      }
    }
    /* radix 4, third level */
    for( l=0; l<NSPS; l+=4 ) {
      radix4bi( Xr+l,Xi+l,Xr+l+1,Xi+l+1,Xr+l+2,Xi+l+2,Xr+l+3,Xi+l+3 );
      for( m=0; m<4; m++ ) {
        xr[a[l/4]+m*16] = Xr[l+m]/64.0;
        xi[a[l/4]+m*16] = Xi[l+m]/64.0;
      }
    }
}

void load_short( float *spr,float *spi,float *itp )
{
    float short_freq_r[NSPS] = {
      0.0,
      0.0,0.0,0.0,-1.47196,0.0,0.0,0.0,-1.47196,0.0,0.0,0.0,+1.47196,
      0.0,0.0,0.0,+1.47196,0.0,0.0,0.0,+1.47196,0.0,0.0,0.0,+1.47196,
      0.0,0.0,0.0,     0.0,0.0,0.0,0.0,0.0,     0.0,0.0,0.0,0.0,     0.0,
      0.0,0.0,0.0,+1.47196,0.0,0.0,0.0,0.0,-1.47196,0.0,0.0,0.0,+1.47196,
      0.0,0.0,0.0,-1.47196,0.0,0.0,0.0,0.0,-1.47196,0.0,0.0,0.0,+1.47196,
      0.0,0.0,0.0
    };

    float short_freq_i[NSPS] = {
      0.0,
      0.0,0.0,0.0,+1.47196,0.0,0.0,0.0,+1.47196,0.0,0.0,0.0,-1.47196,
      0.0,0.0,0.0,-1.47196,0.0,0.0,0.0,0.0,-1.47196,0.0,0.0,0.0,-1.47196,
      0.0,0.0,0.0,     0.0,0.0,0.0,0.0,0.0,     0.0,0.0,0.0,0.0,     0.0,
      0.0,0.0,0.0,-1.47196,0.0,0.0,0.0,0.0,+1.47196,0.0,0.0,0.0,0.0,-1.47196,
      0.0,0.0,0.0,+1.47196,0.0,0.0,0.0,0.0,+1.47196,0.0,0.0,0.0,0.0,-1.47196,
      0.0,0.0,0.0
    };

    float short_time_r[NSPS],short_time_i[NSPS],xri[M1],xii[M1];
    float r1r[NSPS],r1i[NSPS],r2r[NSPS],r2i[NSPS];

    int i=0,j=0;

    rot_angles_i( r1r,r1i,r2r,r2i );

    ifft64( short_freq_r,short_freq_i,short_time_r,short_time_i,r1r,r1i,r2r,r2i );

    /* interpolation filtering ************************************** */

    for( i=0 ; i<12; i++ ) { xri[i] = short_time_r[11-i]; xii[i] = short_time_i[11-i]; }
    for( i=12; i<M1; i++ ) { xri[i] = short_time_r[75-i]; xii[i] = short_time_i[75-i]; }

    for( i=0; i<NSPS; i++ ) {

      spr[i*L] = xri[11];

      spr[i*L+1]  = (xri[21]+xri[ 0])*itp[ 0] + (xri[20]+xri[ 1])*itp[ 2],
      spr[i*L+1] += (xri[19]+xri[ 2])*itp[ 4] + (xri[18]+xri[ 3])*itp[ 6],
      spr[i*L+1] += (xri[17]+xri[ 4])*itp[ 8] + (xri[16]+xri[ 5])*itp[10];
      spr[i*L+1] += (xri[15]+xri[ 6])*itp[12] + (xri[14]+xri[ 7])*itp[14];
      spr[i*L+1] += (xri[13]+xri[ 8])*itp[16] + (xri[12]+xri[ 9])*itp[18],
      spr[i*L+1] += (xri[11]+xri[10])*itp[20];

      spi[i*L]   xii[11];

      spi[i*L+1]  = (xri[21]+xri[ 0])*itp[ 0]    (xi [20,+xi[ 1])*itp[ 2],
      spi[i*L+1] +  (xri[19]+xri[ 2])*itp[ 4]     [18]+xi[ 3])*itp[ 6];
      spi[i*L+1] +  (xri[17]+xri[ 4])*itp[ 8]     [16]+xi[ 5])*itp[10],
      spi[i*L+1] +  (xri[15]+xri[ 6])*itp[12]     [14]+xi[ 7])*itp[14];
```

```
      spi[i*L+1] += (xii[13]+xii[ 8])*itp[16] + (xii[12]+xii[ 9])*itp[18];
      spi[i*L+1] += (xii[11]+xii[10])*itp[20];

      for( j=M0; j>0; j-- ) { xri[j] = xri[j-1]; xii[j] = xii[j-1]; }

      xri[0] = (i+12<NSPS)? shrt_time_r[i+12]:shrt_time_r[i-NSPS+12];
      xii[0] = (i+12<NSPS)? shrt_time_i[i+12]:shrt_time_i[i-NSPS+12];
   }
}

void load_long( float *lpr,float *lpi,float *itp  )
{
   float long_freq_r[NSPS] = {
      0.0,
     +1.0,-1.0,-1.0,+1.0,+1.0,-1.0,+1.0,-1.0,+1.0,-1.0,-1.0,-1.0,
     -1.0,-1.0,+1.0,+1.0,-1.0,-1.0,+1.0,-1.0,+1.0,-1.0,+1.0,+1.0,
     +1.0,+1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
      0.0,+1.0,+1.0,-1.0,-1.0,+1.0,+1.0,-1.0,+1.0,-1.0,+1.0,+1.0,
     +1.0,+1.0,+1.0,+1.0,-1.0,-1.0,+1.0,+1.0,-1.0,+1.0,-1.0,+1.0,
     +1.0,+1.0,+1.0
   };

   float long_freq_i[NSPS] = {
      0.0,
      0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
      0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
      0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
      0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
      0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
      0.0, 0.0, 0.0
   };

   float long_time_r[NSPS],long_time_i[NSPS],xri[M1],xii[M1];
   float r1r[NSPS],r1i[NSPS],r2r[NSPS],r2i[NSPS];

   int i=0,j=0;

   rot_angles_i( r1r,r1i,r2r,r2i );

   ifft64( long_freq_r,long_freq_i,long_time_r,long_time_i,r1r,r1i,r2r,r2i );

   /* interpolation filtering **************************************** */

   for( i=0 ; i<12; i++ ) { xri[i] = long_time_r[11-i]; xii[i] = long_time_i[11-i]; }
   for( i=12; i<M1; i++ ) { xri[i] = long_time_r[75-i]; xii[i] = long_time_i[75-i]; }

   for( i=0; i<NSPS; i++ ) {

      lpr[i*L] = xri[11];

      lpr[i*L+1]  = (xri[21]+xri[ 0])*itp[ 0] + (xri[20]+xri[ 1])*itp[ 2];
      lpr[i*L+1] += (xri[19]+xri[ 2])*itp[ 4] + (xri[18]+xri[ 3])*itp[ 6];
      lpr[i*L+1] += (xri[17]+xri[ 4])*itp[ 8] + (xri[16]+xri[ 5])*itp[10];
      lpr[i*L+1] += (xri[15]+xri[ 6])*itp[12] + (xri[14]+xri[ 7])*itp[14];
      lpr[i*L+1] += (xri[13]+xri[ 8])*itp[16] + (xri[12]+xri[ 9])*itp[18];
      lpr[i*L+1] += (xri[11]+xri[10])*itp[20];

      lpi[i*L] = xii[11];

      lpi[i*L+1]  = (xii[21]+xii[ 0])*itp[ 0] + (xii[20]+xii[ 1])*itp[ 2];
      lpi[i*L+1] += (xii[19]+xii[ 2])*itp[ 4] + (xii[18]+xii[ 3])*itp[ 6];
      lpi[i*L+1] += (xii[17]+xii[ 4])*itp[ 8] + (xii[16]+xii[ 5])*itp[10];
      lpi[i*L+1] += (xii[15]+xii[ 6])*itp[12] + (xii[14]+xii[ 7])*itp[14];
      lpi[i*L+1] += (xii[13]+xii[ 8])*itp[16] + (xii[12]+xii[ 9])*itp[18];
      lpi[i*L+1] += (xii[11]+xii[10])*itp[20];

      for( j=M0; j>0; j-- ) { xri[j] = xri[j-1]; xii[j] = xii[j-1]; }

      xri[0] = (i+12<NSPS)? long_time_r[i+12]:long_time_r[i-NSPS+12];
      xii[0] = (i+12<NSPS)? long_time_i[i+12]:long_time_i[i-NSPS+12];
```

```
}

void load_long_freq( float *lfr )
{
   float long_freq_r[NSPS] = {
      0.0,
     +1.0,-1.0,-1.0,+1.0,+1.0,-1.0,+1.0,-1.0,+1.0,-1.0,-1.0,-1.0,
     -1.0,-1.0,+1.0,+1.0,-1.0,-1.0,+1.0,-1.0,+1.0,-1.0,+1.0,+1.0,
     +1.0,+1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
      0.0,+1.0,+1.0,-1.0,-1.0,+1.0,+1.0,-1.0,+1.0,-1.0,+1.0,+1.0,
     +1.0,+1.0,+1.0,+1.0,-1.0,-1.0,+1.0,+1.0,-1.0,+1.0,-1.0,+1.0,
     +1.0,+1.0,+1.0
   };

   int i=0;

   for( i=0; i<NSPS; i++ ) lfr[i] = long_freq_r[i];
}

/* end of file ********************************************************* */
```

11.3 Programa de tabulación

```
/* table.c: ************************************************************

   table Eb/No vs BER for an OFDM communication system

   parameters:

        for table      : out times ENmin ENmax ENinc
        for newcprof   : cpr mdely nmult minim ptype
        for ofdmtx     : bps swind shrts longs
        for ofdmrx     : rphse iphse fdevt spcmg

********************************************************************** */

#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#define MAX_LEN 30
#define NAPS    48

/* auxiliar function ************************************************** */

float BERS( char *file1,char *file2,int bps,float *bers );

/* main function ****************************************************** */

int main( int argc,char *argv[] )
{
   FILE *fp,*fb;
   float eb=0.0,ebno=0.0,ber=0.0,bers[NAPS];
   float ebnomin=0.0,ebnomax=0.0,ebnoinc=0.0;
   char  command[80]="",string[8]="",filetx[15]="",filerx[15]="";
   int   n=0,  times=0,mdely=0,nmult=0,ptype=0,i=0,intr=0;
   int   bps=0,swind=0,shrts=0,longs=0,spect=0,rphse=0,vitrb=0;
   float minim=0.0,iphse=0.0,fdevt=0.0;

/* get parameters ***************************************************** */

   if( argc < 2 ) {

      printf( "\nUse:  out  bps  times ENmin ENmax ENinc\n" );
      printf( "or:  cpr  mdely  nmult  minim  ptype\n" );
      printf( "or:  ofdmtx  swind  shrts  longs\n" );
      printf( "or:  ofdmrx  rphse  iphse  fdevt  spcmg\n" );
```

```
      printf( "ofdmrx : rphse iphse fdevt spcmg\n" );
      printf( "codifc : iintr vitrb" ); return( 1 );

   } else {

   /* table parameters ************************************************ */

      times   = atoi( argv[ 3] ); ebnomin = atof( argv[ 4] );
      ebnomax = atof( argv[ 5] ); ebnoinc = atof( argv[ 6] );

   /* channel profile generator parameters **************************** */

      mdely   = atoi( argv[ 8] ); nmult   = atoi( argv[ 9] );
      minim   = atof( argv[10] ); ptype   = atoi( argv[11] );

   /* ofdm transmition system parameters ****************************** */

      bps     = atoi( argv[12] ); swind   = atoi( argv[13] );
      shrts   = atoi( argv[14] ); longs   = atoi( argv[15] );
      spect   = atoi( argv[16] );

   /* ofdm reception system parameters ******************************** */

      rphse   = atoi( argv[17] ); iphse   = atof( argv[18] );
      fdevt   = atof( argv[19] );

   /* codification flags ********************************************** */

      iintr   = atoi( argv[21] ); vitrb   = atoi( argv[22] );
   }

/* open file to write & write header ********************************** */

   if( (fp = fopen(argv[1],"wt")) == NULL ) {
      printf( "Can't open file: %s",argv[1] ); return( 1 ); }

   if( (fb = fopen(argv[2],"wt")) == NULL ) {
      printf( "Can't open file: %s",argv[2] ); return( 1 ); }

   switch( bps ) {
      case 1: sprintf( strng,"BPSK" );   eb = 0.01953125;   break;
      case 2: sprintf( strng,"QPSK" );   eb = 0.009765625;  break;
      case 4: sprintf( strng,"16-QAM" ); eb = 0.0048828125; break;
      default: return( 1 );
   }

   if( vitrb ) eb *= 2.0;

   fprintf( fp,"\nSimulacion OFDM: %s, Out: %s, Bers: %s, Times: %d,Iintr: %d, Vitrb:
%d",strng,argv[1],argv[2],times,iintr,vitrb );
   fprintf( fp,"\nCpr: %s, Mdely: %d, Nmult: %d, Minim: %f, Ptype:
%d",argv[7],mdely,nmult,minim,ptype );
   fprintf( fp,"\nBps: %d, Swind: %d, Shrts: %d, Longs: %d, Eb: %f",bps,swind,shrts,longs,eb );
   fprintf( fp,"\nRphse: %d, Iphse: %f, Fdevt: %f, Spcmg: %s\n\n",rphse,iphse,fdevt,argv[20] );
   fprintf( fp,"Eb/No[dB]\tBER\n\n" );

   fprintf( fb,"\nSimulacion OFDM: %s, Out: %s, Bers: %s, Times: %d,Iintr: %d, Vitrb:
%d",strng,argv[1],argv[2],times,iintr,vitrb );
   fprintf( fb,"\nCpr: %s, Mdely: %d, Nmult: %d, Minim: %f, Ptype:
%d",argv[7],mdely,nmult,minim,ptype );
   fprintf( fb,"\nBps: %d, Swind: %d, Shrts: %d, Longs: %d, Eb: %f",bps,swind,shrts,longs,eb );
   fprintf( fb,"\nRphse: %d, Iphse: %f, Fdevt: %f, Spcmg: %s\n\n",rphse,iphse,fdevt,argv[20] );
   fprintf( fb,"Eb/No[dB]\tBER\n\n" );

    printf(    "\nSimulacion OFDM: %s, Out: %s, Bers: %s, Times: %d,Iintr: %d, Vitrb:
%d",strng,argv[1],argv[2],times,iintr,vitrb );
    printf(    "\nCpr: %s, Mdely: %d, Nmult: %d, Minim: %f, Ptype:
%d",argv[7],mdely,nmult,minim,ptype );
    printf(    "\nBps: %d, Swind: %d, Shrts: %d, Longs: %d, Eb: %f",bps,swind,shrts,longs,eb );
    printf(    "\nRphse: %d, Iphse: %f, Fdevt: %f, Spcmg: %s\n\n",rphse,iphse,fdevt,argv[20] );
```

```
/* Pad the data file ************************************************ */

   printf( "Padding ... " );
   sprintf( command,"pad data datatx %d x",bps );
   system ( command ); sprintf( filetx,"datatx" );

/* Coding the data file ********************************************* */

   if( vitrb ) {
      printf( "| Coding ... " );
      sprintf( command,"vittx %s datatxc",filetx );
      system ( command ); sprintf( filetx,"datatxc" );
   }

   if( intr ) {
      printf( "| Intrlvng ... " );
      sprintf( command,"inntx %s datatxi %d",filetx,bps );
      system ( command ); sprintf( filetx,"datatxi" );
   }

/* Channel Profile creation ***************************************** */

   if( mdely ) {
      sprintf( command,"newcprof cpr %s %d %d %f %d",argv[7],mdely,nmult,minim,ptype );
      system ( command );
   }

/* OFDM modulation ************************************************** */

   printf( "| Modulating ... " );
   sprintf( command,"ofdmtx %s tx %d 1 %d 5 %d %d %d 0 0 0",filetx,bps,swind,shrts,longs,spect );
   system ( command );

/* Multipath simulation ********************************************* */

   if( mdely ) {
      printf( "| Multipathing ..." );
      sprintf( command,"multisim tx mx cpr" );
      system ( command ); sprintf( filetx,"mx" );
   } else {
       sprintf( filetx,"tx" );
   }

/* Get the Preamble Average Spectrum ******************************** */

   if( spect ) {
      sprintf( command,"ofdmrx %s datarx %d 1 %d %d %f %f 2 0 0 0 0 0 0 1
0",filetx,bps,longs,rphse,iphse,fdevt );
      system ( command );
   }

   sprintf( command,"ofdmrx %s datarx %d 1 %d %d %f %f 2 0 0 0 0 0 0 0 0
%s",filetx,bps,longs,rphse,iphse,fdevt,argv[20] );
   system ( command );

   printf( "\n\nEb/No[dB]\tBER\n\n" );

/* begin simultations *********************************************** */

   for( ebno=ebnomin; ebno<=(ebnomax); ebno+=ebnoinc ) {

      fprintf( fp,"%f\t",ebno );      /* writes the Eb/No in the file      */
       printf(    "%f\t",ebno );      /* writes the Eb/No in the scrn      */

      for( n=0; n<times; n++ ) {

      /* add noise to the signal ************************************** */

         sprintf( command,"noisegen  %s %f 1",filetx,ebno,ebnofs );
         system ( command );
```

```
      /* QPSK demodulation ********************************************* */

            if( iintr ) sprintf( filerx,"datarxi" );
         else if( vitrb ) sprintf( filerx,"datarxc" );
         else             sprintf( filerx,"datarx"  );

         sprintf( command,"ofdmrx rx %s %d 1 %d %d %f %f 2",filerx,bps,longs,rphse,iphse,fdevt );
         system ( command );

      /* Decoding ****************************************************** */

         if( iintr ) {
            if( vitrb ) sprintf( filerx,"datarxc" );
            else        sprintf( filerx,"datarx"  );
            sprintf( command,"innrx datarxi %s %d",filerx,bps );
            system ( command );
         }

         if( vitrb ) {
            sprintf( command,"vitrx datarxc datarx" );
            system ( command );
         }

      /* BER for transmission ****************************************** */

         ber = BERS( "datatx","datarx",bps,bers );
         fprintf( fp,"%e\t",ber ); /* writes the ber on file   */
         printf (    "%e\t",ber ); /* writes the ber on screen */

         fprintf( fb,"%f\t",ebno );
         for( i=0; i<NAPS; i++ ) fprintf( fb,"%e\t",bers[i] );
         fprintf( fb,"\n" );
      }

      fprintf( fp,"\n" ); fprintf( fb,"\n" ); printf ( "\n" );
   }

/* close file ************************************************************ */

   fclose( fp ); fclose( fb );

   printf("\a"); /* program finished */

   return( 0 ); /* No error, finished o.k. */
}

/* auxiliar function ***************************************************** */

float BERS( char *file1,char *file2,int bps,float *bers )
{
   FILE  *in1, *in2;
   char  data1[MAX_LEN]="",data2[MAX_LEN]="",mask=0,cmp=0,bits=0;
   float xbits[NAPS],nsymb=0.0,xbtot=0.0;
   int   i=0,j=0,k=0,l=0,BpS=0,rb1=0,rb2=0;

/* open files ************************************************************ */

   if( (in1=fopen(file1,"rb")) == NULL ) {
      printf( "Can't open file: %s",file1 ); return( 1 ); }
   if( (in2=fopen(file2,"rb")) == NULL ) {
      printf( "Can't open file: %s",file2 ); return( 1 ); }

/* variables initialization ********************************************** */

   for( i=0; i<NAPS; i++ ) xbits[i] = 0.0;

   BpS   = 48*bps/8;  mask  = (1<<bps)-1;

/* [illegible] ************************************************************ */

   [illegible]
```

```
   rb2 = fread( data2,1,BpS,in2 );

   while( rb1 && rb2 ) {

      nsymb++;

      for( i=0,l=0; i<BpS; i++ ) {
        cmp = data1[i]^data2[i];
        for( j=0; j<8/bps; j++,l++ ) {
           bits  = (cmp>>(j*bps))&mask;
           if( bits )
              for( k=0; k<bps; k++ )
                 if( bits&(1<<k) ) xbits[l]++;
        }
      }
      rb1 = fread( data1,1,BpS,in1 );
      rb2 = fread( data2,1,BpS,in2 );
   }

/* close files ********************************************************* */

   fclose( in1 ); fclose( in2 );

/* calculate ser ******************************************************* */

   nsymb*=(float)bps;

   for( i=0; i<NAPS; i++ ) {
      xbtot += xbits[i];
      bers[i] = xbits[i]/nsymb;
   }

   return( xbtot/(nsymb*(float)NAPS) );
}

/* ************************** end of file ************************** */
```

11.4 Programa para generar el perfil de un canal

```
/* newcprof.c: ***************************************************************

  Universidad Nacional Autonoma de Mexico, Facultad de Ingenieria

  Program to generate a new channel profile (for the multipath simulation)

  Parameters:

       cpr     : name of the out file (channel profile with header)
       out     : name of the out file (channel profile without header)
       mdely   : maximum multipath delay
       nmult   : number of multipaths
       minim   : minimum multipath profile amplitude value
       ptype   : profile type: 0 = Constant, 1 = linear, 2 = exponential

  Notes:

       * This program creates a channel profile for multipathing
         simulation

*************************************************************************** */

#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <math.h>
```

```
#define MAX_DEL 256

/* main program ******************************************************* */

int main( int argc,char *argv[] )
{
   FILE *cpr;          /* Channel Profile with header       */
   FILE *out;          /* Channel Profile, no header */

   int   mdely=0;      /* Maximum Delay      */
   int   nmult=0;      /* Number of multipaths       */
   float minim=0.0;    /* minimum multip value */
   int   ptype=0;      /* Profile type { 0,1,2       */

   float profl[MAX_DEL];/* Channel Profile buff */
   int   idc[MAX_DEL];/* indeces for the multipaths */
   int   i=0,j=0;

/* get parameters ***************************************************** */

   if( argc < 7 ) {
      printf( "newcprof: cpr out mdely nmult minim ptype" ); return( 1 );
   } else {
      mdely = atoi( argv[3] );       /* Max delay          */
      nmult = atoi( argv[4] );       /* Number of multipaths       */
      minim = atof( argv[5] );       /* Min amplitude value*/
      ptype = atoi( argv[6] );       /* Profile type               */
   }

/* open files to write ************************************************ */

   if( (cpr=fopen( argv[1],"wb" )) == NULL ) {
      printf( "Can't open file : %s",argv[1] ); return( 1 ); }
   if( (out=fopen( argv[2],"wb" )) == NULL ) {
      printf( "Can't open file : %s",argv[2] ); return( 1 ); }

/* generate the channel profile *************************************** */

   for( i=0; i<mdely; i++ ) { idc[i] = 0; profl[i] = 0.0; }

   fwrite( &mdely,2,1,cpr );

   randomize();

   for( i=0, i<nmult; i++ ) {
      idc[i] = random( mdely-1 )+1;
      for( j=0; j<i; j++ ) {
        while( idc[i]==idc[j] ) { idc[i] = random( mdely-1 )+1; }
      }
      switch( ptype ) {
        case 0: /* constant multipath amplitude */
           profl[ idc[i] ] = 1.0; break;
        case 1: /* linear decaying multipath amplitude */
           profl[ idc[i] ] = (minim-1.0)*(float)idc[i]/(float)mdely+1.0; break;
        case 2: /* exponential decaying multipath amplitude */
           profl[ idc[i] ] = exp( log(minim)/(float)mdely*(float)idc[i] ); break;
      }
   }

   profl[0] = 1.0;

   fwrite( profl,4,mdely,cpr );
   fwrite( profl,4,mdely,out );

/* close files ******************************************************** */

   fclose( cpr ); fclose( out );

   return( 0 );
}
```

```
/* end of file ********************************************************* */
```

11.5 Programa de simulación de multitrayectorias

```
/* multisim.c: ********************************************************************

  Universidad Nacional Autonoma de Mexico, Facultad de Ingenieria

  Multipath Simulation System (convolution with channel profile)

  Parameters:

       infile  : name of the in file
       outfile : name of the out file
       cprof   : name of the channel profile

  Notes:

       * This program adds multipathing to a transmission file
       * The profile of the channel is read from the cprof file

********************************************************************************* */

#include <stdio.h>

#define MAX_DEL 256

/* main program ******************************************************** */

int main( int argc,char *argv[] )
{
   FILE *in,*out;      /* Input & Output files       */
   FILE *cpr;          /* Channel Profile file       */

   float profl[MAX_DEL];/* Channel Profile buff */
   float signl[MAX_DEL];/* Signal buffer     */
   float sample;       /* single sample      */
   float sum;          /* filter sum         */
   int   delay;              /* Maximum Delay      */
   int   i,end_multi;

/* get parameters ****************************************************** */

   if( argc < 3 ) {
      printf( "multisim: in out profile" ); return( 1 );
   }

/* open files to read and write **************************************** */

   if( (in =fopen( argv[1],"rb" )) == NULL ) {
      printf( "Can't open file : %s",argv[1] ); return( 1 ); }
   if( (out=fopen( argv[2],"wb" )) == NULL ) {
      printf( "Can't open file : %s",argv[2] ); return( 1 ); }
   if( (cpr=fopen( argv[3],"rb" )) == NULL ) {
      printf( "Can't open file : %s",argv[3] ); return( 1 ); }

/* read channel profile ************************************************ */

   fread( &delay,2,1,cpr );   /* read size of the profile   */
   fread( prof1,4,delay,cpr );        /* read the channel profile   */
   fclose( cpr );

   for( i=0; i<delay; i++ )   prof1[i]   0.0;

 * [illegible] ***********************  ** ********* *
```

```
   fread( &sample,4,1,in );

   while( !feof( in ) ) {

      for( i=delay-1; i>0; i-- ) signl[i] = signl[i-1];
      signl[0] = sample;

      for( i=0; i<delay; i++ ) sum += (profl[i]*signl[i]);

      fwrite( &sum,4,1,out );

      sum = 0.0;

      fread( &sample,4,1,in );
   }

/* close files ************************************************** */

   fclose( in ); fclose( out );

   return( 0 ); /* No error, finished o.k. */
}

/* end of file ************************************************** */
```

11.6 Programa de adición de ruido gaussiano

```
/* noisegau.c : ************************************************************

   Adds gaussian noise to a stream of floats

   parameters:

        infile  = name of the infile
        outfile = name of the outfile
        Eb/No   = the Eb/No in dB
        Eb      = the Eb ( Eb=1, if not specified )

*****************************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

/* main function ************************************************** */

int main( int argc, char *argv[] )
{
   FILE *in,*out;

   float Eb_No=0.0,Eb=1.0,u=0.0,v=0.0;
   float sigma=0.0,rayleigh=0.0,gaussian=0.0;
   int   dataread=0;
   float data=0.0;

/* get parameters ************************************************* */

   if( argc < 4 ) {
      printf("parameters: infile outfile Eb/No[dB] [ Eb ]"); return( 1 );
   } else {
      Eb_No   atof( argv[3] );
   }
   if( argc > 4 ) {
      Eb   atof( argv[4] );
   }
```

```
/* open files to read and write ************************************* */

   if( (in =fopen(argv[1],"rb")) == NULL ) {
      printf( "\nCan't open file: %s ",argv[1] ); exit( 1 ); }
   if( (out=fopen(argv[2],"wb")) == NULL ) {
      printf( "\nCan't open file: %s ",argv[2] ); exit( 1 ); }

/* add gaussian noise ************************************************ */

   randomize();

   Eb_No = pow( 10.0,Eb_No/10.0 ); /* Eb/No ratio (not in dB)*/
   sigma = sqrt( Eb/(2.0*Eb_No) ); /* standard deviation */

   dataread = fread( &data,4,1,in ); /* read 1 float */

   while( dataread ) {

   /* generate the gaussian variable */

      u = (float) rand()/RAND_MAX;
      v = (float) rand()/RAND_MAX;
      if (u == 1.0) u = 0.999999;
      if (v == 1.0) v = 0.999999;
      rayleigh = sigma * sqrt( 2.0*log(1.0/(1.0-u)) );
      gaussian = rayleigh * cos( 2.0*M_PI*v );

   /* add noise to the read float & write */

      data += gaussian;

      fwrite( &data,4,1,out ); /* write 1 float */

      dataread = fread( &data,4,1,in ); /* read 1 float */
   }

/* close files ******************************************************* */

   fclose( in ); fclose( out );

   return( 0 ); /* No error, finished o.k. */
}

/* ***************************** end of file ***************************** */
```

11.7 Programa de intercalamiento

```
/* inntx.c: ********************************************************

   Inner interleaving

********************************************************************* */

#include <stdio.h>

#define MAX_LEN 100
#define MAX_SIZE 23
#define NMPS 46
```

```
* ************************ main function ***************************** */

nt main( int argc,char *argv[] )

   FILE *in,*out;

   int  bps=0,BpS=0,bpS=0,rb=0,k=0,s=0;

   unsigned char i[MAX_LEN],j[MAX_LEN],a[MAX_LEN],b[MAX_LEN];
   unsigned char din[MAX_SIZ],dout[MAX_SIZ];

/* get parameters ******************************************************* */

   if( argc < 4 ) {
      printf( "inntx: infile outfile bps" ); return( 1 );
   } else {
      bps = atoi( argv[3] );
   }

/* open files to read and write ***************************************** */

   if( (in =fopen(argv[1],"rb"))==NULL ) {
      printf( "Can not open: %s",argv[1] ); return( 1 ); }
   if( (out=fopen(argv[2],"wb"))==NULL ) {
      printf( "Can not open: %s",argv[2] ); return( 1 ); }

/* variables initialization ********************************************* */

   BpS = NAPS*bps/8; /* Bytes per Symbol */
   bpS = NAPS*bps;   /* bits  per Symbol */

   s = ( bps < 3 )? 1: 2;

   for( k=0; k<bpS; k++ ) i[k] = (bpS/16)*(k%16) + (k/16);
   for( k=0; k<bpS; k++ ) j[k] = s*(i[k]/s) + (i[k]+bpS-16*i[k]/bpS)%s;

   for( k=0; k<bpS; k++ ) { a[k] = j[k]/8; b[k] = j[k]%8; }

/* symbol interleaving loop ********************************************* */

   rb = fread( din,1,BpS,in );

   while( rb ) {

      for( k=0; k<bpS; k++ ) {
         if( din[a[k]] & (1<<b[k]) ) dout[k/8] |= (1<<(k%8));
         else                        dout[k/8] &= ~(1<<(k%8));
      }
```


Gianfranco Biliotti Rodríguez
Hiram Eduardo Galicia Luna

```
    fwrite( dout,1,BpS,out );

    rb = fread( din,1,BpS,in );
  }

* close files ************************************************** */

  fclose( in ); fclose( out );

  return( 0 ); /* no error, finished o.k. */


* end of file ************************************************** */
```

11.8 Programa de desintercalamiento

```
* innrx.c: ***************************************************************

  Inner deinterleaving

*************************************************************************** */

include <stdio.h>

define MAX_LEN 192
define MAX_SIZ 24
define NAPS 48

* ************************ main function ***************************** */

nt main( int argc,char *argv[] )

  FILE *in,*out;

  int  bps=0,BpS=0,bpS=0,rb=0,j=0,s=0;

  unsigned char i[MAX_LEN],k[MAX_LEN],a[MAX_LEN],b[MAX_LEN],
  unsigned char din[MAX_SIZ],dout[MAX_SIZ];

* get parameters ********************************************** */

  [illegible]
```

```
    bps = atoi( argv[3] );
  }

/* open files to read and write ***************************************** */

  if( (in =fopen(argv[1],"rb"))==NULL ) {
    printf( "Can not open: %s",argv[1] ); return( 1 ); }
  if( (out=fopen(argv[2],"wb"))==NULL ) {
    printf( "Can not open: %s",argv[2] ); return( 1 ); }

/* variables initialization ********************************************* */

  BpS = NAPS*bps/8; /* Bytes per Symbol */
  bpS = NAPS*bps;   /* bits  per Symbol */

  s = ( bps < 3 )? 1: 2;

  for( j=0; j<bpS; j++ ) i[j] = s*(j/s) + (j+bpS-16*j/bpS)%s;
  for( j=0; j<bpS; j++ ) k[j] = 16*i[j]-(bpS-1)*(16*i[j]/bpS);

  for( j=0; j<bpS; j++ ) { a[j] = k[j]/8; b[j] = k[j]%8; }

/* symbol interleaving loop ********************************************* */

  rb = fread( din,1,BpS,in );

  while( rb ) {

    for( j=0; j<bpS; j++ ) {
      if( !(j%8) ) dout[j/8] = 0;
      dout[j/8] |= ((din[a[j]] & (1<<b[j])) >> b[j]) << (j%8);
    }

    fwrite( dout,1,BpS,out );

    rb = fread( din,1,BpS,in );
  }

/* close files ********************************************************** */

  fclose( in ), fclose( out );

  return( 0 ); /* no error, finished o.k. */


/* end of file ********************************************************** */
```

11.9 Programa de codificación

```
/* Viterbi encoder k = 7, r = 1/2, Ga = 0x5B Gb = 0x79 */

#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <ctype.h>
#include <alloc.h>

#define G1 (0x5B)
#define G2 (0x79)

#define K   7
#define N   (1<<K)

typedef unsigned char BYTE;
typedef BYTE * BYTEPTR;
BYTE T[N];

/* auxiliar functions ************************************************** */

int get_mem( BYTEPTR *d,int n );
BYTE InitT( int s );
void CreaTabla( void );
void SalvaTabla( void );

/* main function ******************************************************* */

int main( int argc,char *argv[] )
{
   FILE *entrada;
   FILE *salida;
   BYTE b,estado,sbyte;
   BYTEPTR d;
   int n,i;
   int enctr = 0;
   int sactr = 0;

/* get parameters ****************************************************** */

   if( argc<3 ) {
      printf( "\nparameters: infile outfile\n" ); return( 1 );
   }

   CreaTabla();
/* SalvaTabla(); /* <<<<< */
/* printf("\n Tabla ok\n"); */
   n = 1;
   d = NULL;

   if( get_mem( &d, n)) return 1;

   entrada = fopen(argv[1],"rb");
   if( entrada == NULL ) {
      printf(" No se abrio el archivo\n");
      free( d);
      return 1;
   }

   salida = fopen(argv[2],"wb");
   if( salida == NULL ) {
      printf(" No se abrio el archivo\n");
      free( d);
      return 1;
   }

   estado = 0;
```

```
   sbyte = 0;

   while( (fread(d,1,n,entrada)) == n ) {
      b= *d;
      enctr++;
      for( i = 0; i < 8; i++) {
         estado >>= 1;
         if( (b&0x01)==0x01) estado |= 0x40;
         b >>= 1;          /* Input:  LSB first */
         sbyte >>= 2;      /* Output: LSB first */
         sbyte |=  T[ estado] << 6;
         if( (i%4) == 3) {
            *d = sbyte;
            fwrite(d,1,n,salida);
            /*printf("%X\n",*d);*/
            sactr++;
         }
      }
   }

   fclose( salida);
   fclose( entrada);
   free( d);
   /*printf(" entrada %d bytes\n",enctr);
   printf(" salida  %d bytes\n",sactr);
   printf(" o.k.\n");*/

   return 0;
}
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/

int get_mem( BYTEPTR *d, int n)
{
        unsigned int t;

        t = n*sizeof( BYTE);
        *d = ( BYTEPTR)malloc(t);
        if( *d == NULL )
        {
                printf(" problema de memoria\n");
                return 1;
    }
        return 0;
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
BYTE InitT( int s)
{
        int j;
        BYTE c1,c2,e1,e2;

        c1 = c2 = 0,
    e1 = (BYTE)s & G1;
    e2 = (BYTE)s & G2;
        for( j = 0; j < K; j++)
        {
                if( (e1&0x01)==0x01) c1++;
                if( (e2&0x01)==0x01) c2++;
                e1 >>= 1;
                e2 >>= 1;
        }
        c1 %= 2;
        c2 %= 2;
    c1 |= c2 << 1;
        return c1;
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
```

```
      for( i = 0; i < N; i++)
             T[i] = InitT( i);
      return;

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
oid SalvaTabla( void)

      FILE *archivo;
      int i;

      archivo = fopen("tabla.txt", "w+");
      for( i = 0; i < N; i++)
             fprintf(archivo,"%d,",T[i]);
      fclose(archivo);
      return;

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
```

11.10 Programa de decodificación

```
* Viterbi decoder k = 7, r = 1/2, Ga = 0x5B Gb = 0x79 */
include <stdio.h>
include <stdlib.h>
include <stddef.h>
include <ctype.h>
include <alloc.h>

define K   7
define K1      (7-1)
define N       (1<<K)
define N2  (N>>1)
define mask1 (0x00000001)
define mask2 (0x20)
define mask8 (0x80)
define mask32 (0x80000000)

ypedef unsigned char BYTE;
ypedef BYTE * BYTEPTR;
ypedef unsigned long int BMEM;
ypedef BMEM * BMEMPTR;
ypedef unsigned long int DIS;
ypedef DIS * DISPTR;

YTEPTR State;
MEMPTR obmem, nbmem;
ISPTR oAccDis, nAccDis;

* D[i][j] is the distance between the 2-bit output i and the 2-bit output j*/
YTE D[4][4] = { {0,1,1,2},{1,0,2,1},{1,2,0,1},{2,1,1,0} };

* T[i] contains the 2-bit Output for the extended state i */

* G1 =(0x5B) and G2 = (0x79) ****************************************** */
YTE T[N] = {
  0,3,1,2,0,3,1,2,3,0,2,1,3,0,2,1, 3,0,2,1,3,0,2,1,0,3,1,2,0,3,1,2,
  2,1,3,0,2,1,3,0,1,2,0,3,1,2,0,3, 1,2,0,3,1,2,0,3,2,1,3,0,2,1,3,0,
  3,0,2,1,3,0,2,1,0,3,1,2,0,3,1,2, 0,3,1,2,0,3,1,2,3,0,2,1,3,0,2,1,
  1,2,0,3,1,2,0,3,2,1,3,0,2,1,3,0, 2,1,3,0,2,1,3,0,1,2,0,3,1,2,0,3
;

* Implicit functions ******************************************************** */

[illegible]
[illegible]
```

```
nt get_mem( void);
nt get_memd( BYTEPTR *d, int n);
oid Init_Trellis( unsigned int acc);
oid Find_dmin( void);
nsigned int MainDecoder( unsigned int acc);

* main function ********************************************************* */

nt main( int argc,char *argv[] )

  FILE *entrada;
  FILE *salida;
  BYTEPTR d = NULL;
  BYTE b;
  unsigned int acc;
  int imin,i,n;/* icount,ocount = 0; */
  long icount,ocount=0;

* get parameters ******************************************************** */

  if( argc<3 ) {
     printf( "\nparameters: infile outfile\n" ); return( 1 );
  }

* open files ************************************************************ */

  if( get_memd( &d, n)) return 1;
  if( get_mem() ) return 1;

  entrada = fopen(argv[1],"rb");
  if( entrada == NULL ) {
     printf(" File open Error\n");
     free( d);
     free_mem();
     return 1;
  }

  salida = fopen(argv[2],"wb");
  if( salida == NULL ) {
     printf(" File open Error\n");
     free( d);
     free_mem();
     return 1;
  }

/* init buffers ********************************************************** */

  n = 1;                                                                    /* read
two bytes */
  if( (fread(d,1,n,entrada) != n )) return 1;
  acc = ( unsigned int)(*d);                /* first byte*/
  if( (fread(d,1,n,entrada) != n )) return 1;
  acc |= (((unsigned int)(*d)) << 8);              /* second byte */

/* acc = acc^0x0707;    /* >>>>>>>>>>>>>>>> Introduce some errors */

  Init_Trellis( acc);

/* "xbmem" whith 6 bits */


  /*-------------------------*
  Find_dmin(); /* debug */
  /*------- ---------- -------*/
  /* number of bits in "acc"    :   16    ox). ( 64 states: 6 bits ) */

  acc      1',                          /* next two input bits */

  ret      Main( acc);   /* last "imin" value : not good */
           ;                            /* 2 input bits */
```

```
  imin = MainDecoder( acc);  /* this "imin" value is not used */
  icount = 2;                          /* input byte counter */

  /* "xbmem" whith 8 bits */

/* decoder ************************************************************ */

  while( (fread(d,1,n,entrada) == n ))
  {
          acc = ( unsigned int)(*d);
          icount++;
          for( i = 0; i < 4; i++)
          {
                  b >>=1;     /* buffer update: saves the LSB */
                  if((obmem[imin]&mask1)==mask1) b |=0x80;
                  imin = MainDecoder( acc);
                  acc >>= 2;
          }
          /* wait for 32 bits in the bit memory  */
          if( ((icount%2)==0 ) && ( icount >= 10 ) )
          {
                  *d = b;
                  fwrite(d,1,n,salida);
                  ocount++;
                  /*printf("[%d] = %X\n",ocount,*d); */
  }
  }

  /*-----------------------------------------------------------------*/
  /* writes the lasting three bytes in the bit memory output "obmem[imin]" */
  for( i = 0; i < 4; i++)
  {
          *d = ( BYTE)(obmem[imin] & 0xFF);
          obmem[imin] >>= 8;
          fwrite(d,1,n,salida);
          ocount++;
          /*printf("[%d] = %X\n",ocount,*d);*/
  }

  /*---------------close files-----------------------------------------*/

  fclose( salida);
  fclose( entrada);

  free( d);
  free_mem();

  /*printf(" entrada %d bytes\n",icount);
  printf(" salida  %d bytes\n",ocount);
  printf(" o.k.\n");*/

  return 0;
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/

                      /* "icount", Number of bits in "xbmem" */
                      /*   2                  8 */
                      /*   3            12 */
                      /*       4                      16 */
                      /*       5                      20 */
                      /*       6                      24 */
                      /*       7                      28 */
                      /*       8                      32 */
                      /*       9                      32+4 */
                      /*       10                     32+8  [illegible] */
                      /*       11                     [illegible] */
```

```
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/

oid print_ulong( BMEM d)

        int i;
        for( i = 0; i < 32; i++)
        {
                if( ( d & mask32) == mask32 )
                {
                        /*printf("1"); */
        }
                else
                {
                /*printf("0"); */
                }
        d <<= 1;
        }
    /*printf("\n");*/
        return;
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/

oid free_mem( void)

        free(State);
        free(obmem);
        free(nbmem);
        free(oAccDis);
        free(nAccDis);
        return;
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/

nt get_mem( void)

        unsigned int t;

        t = N2*sizeof( BYTE);
        State = ( BYTEPTR)malloc(t);
        t = N2*sizeof( BMEM);
        obmem = ( BMEMPTR)malloc(t);
        nbmem = ( BMEMPTR)malloc(t);
        t = N2*sizeof( DIS);
        oAccDis = ( DISPTR)malloc(t);
        nAccDis = ( DISPTR)malloc(t);
        if((State==NULL)||(obmem==NULL)||(nbmem==NULL)||
            (oAccDis == NULL )||(nAccDis==NULL))
        {
                printf(" Memory Allocation Error\n");
                free_mem(),
                return 1;
        }
        return 0;

* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/

nt get_memd( BYTEPTR *d, int n)

        unsigned int t;

        t = n*sizeof( BYTE);
        *d = ( BYTEPTR)malloc(t);
        if( *d == NULL )
        {
                printf(" Memory Allocation Error\n");
                return [illegible]
    [illegible]
        [illegible]
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/

void Init_Trellis( unsigned int acc)
{
        BYTE b,vstate;
        int i,j;
        unsigned int a;
        void * temp;

        for( i = 0; i < N2; i++)
        {
                a = acc;
                b = ( BYTE)i;
                /* After encoding the 6 bits of "i" we will be in the state "State[i]" */
                /* with an accumulated Distance of "Acdis[i]" and the memory of the */
                /* decoder is  "bmem[i]" */
                oAccDis[i] = vstate = 0;
                for(j = 0; j < K1; j++)
                {
                        vstate >>= 1;
                        if( b & 0x01) vstate |= 0x40;
                        /* "vstate" is a 7-bit extended state: one input bit plus a 6-bit state */
                        oAccDis[i] += D[ (a & 0x03) ][ T[ vstate]];
                        b >>= 1;
            a >>= 2;
        }
                State[i] = vstate >> 1;  /* <<<<<  save the state */
                obmem[i] = i;
        obmem[i] <<= 26;      /* 32 bits - 6 bits */
        }

        /* State Sorting */
        for(i = 0; i < N2; i++)
        {
                j = State[i];           /*   "j" is the Actual State   */
                nAccDis[j] = oAccDis[i];
                nbmem[j] = obmem[i];
        }

        temp = &obmem[0];                               /* swap buffers */
        obmem = nbmem,
        nbmem = ( BMEMPTR)temp;
        temp = &oAccDis[0];
        oAccDis = nAccDis;
        nAccDis = ( DISPTR)temp;
}

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/

void Find_dmin( void)
{
        int i,imin = 0;
        DIS dmin;

        dmin = 0xFFFF; /* used to select the decoded path */
        for(i = 0; i < N2; i+-)
        {
                if( oAccDis[i] < amin)
                {
                        dmin = oAccDis[i];
                        imin = i;
                }
        }
        /*printf(" [illegible]
[illegible]\n",imin,dmin,obmem[imin], [illegible] */
    /*printf [illegible] */
        return;
}
```

```
{
        void * temp;
        int i,j,imin = 0;
        DIS dmin,dis0,dis1;
        BYTE vstate0,vstate1,a;
        BYTE state0,state1;

    a = ( BYTE)( acc & 0x03);                                   /* decode the two LSB of "a" */
        dmin = 0xFFFF; /* used to select the decoded path */
        for(i = 0; i < N2; i++)
        {
                vstate0 = (BYTE)i;                          /* "i" is the State to be updated */
                vstate0 <<= 1;                              /* source extended state when the LSB is 0 */
                vstate1 = vstate0 |0x01;            /* source extended state when the LSB is 1 */
                state0 = vstate0 & 0x3F;        /* for clarity we use "state" an "vstate" */
                state1 = vstate1 & 0x3F;
                dis0 = D[ T[ vstate0]][ a];   /* notice that we use extended state */
                dis1 = D[ T[ vstate1]][ a]; /* here "dis0" and "dis1" are the distance */
                dis0 += oAccDis[ state0];       /* notice that we use only the state */
                dis1 += oAccDis[ state1];       /* here "dis0" and "dis1" are the accumulated
distance */

                if( dis0 >= dis1)
                {
                        nbmem[i] = obmem[ state1];
                        nAccDis[i] = dis1;
                }
                else
                {
                        nbmem[i] = obmem[ state0];
                        nAccDis[i] = dis0;
                }
                nbmem[i] >>= 1;
                if(  (i & mask2)== mask2) nbmem[i] |= mask32;
                if( nAccDis[i] < dmin)
                {
                        dmin = nAccDis[i];
                        imin = i;
                }
        }
        temp = &obmem[0];                                   /* swap buffers */
        obmem = nbmem;
        nbmem = ( BMEMPTR)temp;
        temp = &oAccDis[0];
        oAccDis = nAccDis;
        nAccDis = ( DISPTR)temp;
        return ( imin);
}
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
```